UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Clusteranalyseverfahren für hochdimensionale binäre Daten: Algorithmen und Performanz für Big Data Anwendungen

Von der Fakultät für Gesellschaftswissenschaften der Universität Duisburg-Essen

zur Erlangung des akademischen Grades

Dr. phil.

genehmigte Dissertation

von

Philip Höcker

aus

Mülheim an der Ruhr

1. Gutachter: Prof. Dr. Rainer Schnell
2. Gutachterin: Prof. Dr. Petra Stein

Tag der Disputation: 29.11.2023

Danksagung

In erster Linie bin ich meinem Betreuer Prof. Dr. Rainer Schnell zu großem Dank verpflichtet. Er hat mir die Tür zur Arbeit in der Wissenschaft geöffnet, mir die Möglichkeit gegeben diese Dissertation anzufertigen und mich stets unterstützt. In der Zeit die ich für ihn gearbeitet habe, habe ich viel gelernt und viele interessante wissenschaftliche Problemstellungen und Themenbereiche kennengelernt. Nicht zuletzt stammt die Idee und das Design für diese Arbeit und die Idee Blockingmethoden aus dem Record Linkage zum Clustern von binären Daten zu verwenden von ihm. Ich danke Prof. Dr. Petra Stein für die Erstellung des Zweitgutachten und Prof. Dr. Marcel Erlinghagen und Prof. Dr. Helen Baykara-Krumme für die Teilnahme an der Prüfungskommission und der Abnahme der Disputation.

Des Weiteren bin ich sehr dankbar für die Personen mit denen ich während meiner Zeit am Lehrstuhl von Prof. Dr. Rainer Schnell zusammengearbeitet habe. Das sind Nora Halstenberg, Sarah Redlich, Isabell Rohling, Sven Brocker, Christian Borgs, Monika Obersneider und Jonas Klingwort. Die vielen Gespräche und Anregungen über unsere jeweiligen Dissertationsprojekte waren sehr hilfreich und haben stets dazu beigetragen am Ball zu bleiben. Ein ganz besonderer Dank gilt dem stets hilfsbereiten Julian Reinhold für die unzähligen Gespräche und Anregungen, der technischen Hilfe beim Setzen des Textes und des Korrekturlesens der Abgabeversion.

Zusammenfassung

Die vorliegende Dissertation stellt eine Simulationsstudie dar, welche die Performanz verschiedener Clusteralgorithmen für hochdimensionale binäre Datensätze evaluiert. Bei binären Daten handelt es sich um Daten, die nur 2 Ausprägungen annehmen können. In der Regel sind dies 0 und 1. Dabei wird der Einfluss der Dimensionalität der Daten, der Anzahl der Cluster, der Prävalenz der Einsen und der Korrelationsstruktur analysiert. Die Art und Weise, wie das Datenmaterial simuliert wird und die damit einhergehende Clusterstruktur ist ebenfalls Ziel der Analyse. Aufgrund der Tatsache, dass es sich um simulierte Daten handelt, sind die tatsächlichen Clusterzugehörigkeiten der einzelnen Beobachtungen bekannt. Auf dieser Grundlage kann daher die Güte der Clusterlösungen bewertet werden. Da im Kontext von Big Data Anwendungen die Laufzeit der Algorithmen und der Speicherbedarf ebenfalls wichtige Faktoren sind, die über die Praktikabilität von Algorithmen entscheiden können, werden auch Laufzeit und Speicherbedarf betrachtet. Die qualitativ besten Clusterlösungen bringt in der Simulation der Algorithmus *SCALE* hervor. Sind jedoch Laufzeit und Speicherbedarf von Relevanz, schneiden die Algorithmen *Proximus* und *BUBBLE* am besten ab. Ist die Clusterstruktur jedoch nicht klar getrennt und es befindet sich eine größere Menge an Rauschen in den Daten, findet sich kein Algorithmus, der konstant gute Ergebnisse liefert.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Clusteranalyse ist eine häufig verwendete Analysemethode in der quantitativen Forschung und ist Forschungsgegenstand dieser Arbeit. Zunächst muss geklärt werden, was unter Clusteranalyse verstanden wird. Ein sehr simple Definition stammt von Kaufman und Rousseeuw (2005, S. 1): „Cluster analysis is the art of finding groups in data“. Daten in sinnvolle Gruppen einzuteilen ist Jain und Dubes (1988, S. 1) zufolge eine der wichtigsten Aufgaben der Datenanalyse. Cluster sind dabei so definiert, dass sich Beobachtungen innerhalb eines Cluster möglichst ähnlich sind, während sich Beobachtungen in unterschiedlichen Clustern möglichst unähnlich sein sollten (Gan et al., 2007, S. 3; Bacher et al., 2010, S. 16).

Es darf dabei nicht der Fehler begangen werden Clusteranalyse mit Klassifizierung zu verwechseln. Bei der Klassifikation wird auf Grundlage eines Datensatzes mit bekannter Klassenzugehörigkeit (Labels) der einzelnen Beobachtungen zunächst ein Modell erstellt. Mithilfe dieses Modells werden dann neue Daten ohne bekannte Klassenzugehörigkeit einzelnen Klassen zugewiesen. Bei der Clusteranalyse hingegen wird mit Datensätzen ohne bekannte Klassen- bzw. Clusterzugehörigkeit direkt gearbeitet und die einzelnen Beobachtungen Gruppen zugewiesen, ohne vorher ein Modell zu erstellen (Gan, 2011, S. 4–5). Ersteres wird auch zum *Supervised Learning* gezählt (Everitt et al., 2011, S. 7). Entsprechend gehört daher Clusteranalyse zum *Unsupervised Learning*.

Es kann grob von vier Ansätzen innerhalb der Clusteranalyse gesprochen werden: *Partitioning Methods*, *Hierarchical Methods*, *Density-Based Methods* und *Grid-Based Methods* (Han et al., 2012, S. 448–450). Beispiele für die einzelnen Ansätze wären *K-Means* (Partitioning Method), *ROCK* (Hierarchical Method), *DBSCAN* (Density-Based Method) und *COBWEB* (Grid-Based Method). Nicht alle Methoden lassen sich jedoch in dieser Klassifikation verordnen. Es gibt bspw. Methoden, die auf der Erstellung von Graphen basieren (Andreopoulos et al., 2009), wie z.B. *Chromatic Clustering* (Bonchi et al., 2015).

Es existieren einige Übersichtsarbeiten, welche verschiedenste Clusteralgorithmen vergleichend gegenüberstellen (siehe Abschnitt 2). Eine umfangreiche Simulationsstudie die die Performance von Algorithmen geeignet für das Clustern großer hochdimensionaler binärer Datensätze evaluiert, ist zu diesem Zeitpunkt jedoch nicht vorhanden. Unter binären Daten werden Daten verstanden, die lediglich 2 Ausprägungen annehmen können. In der Regel sind diese mit 0 und 1 codiert (Kaufman & Rousseeuw, 2005, S. 22). Das kann zum Beispiel bedeuten, dass ein Merkmal vorhanden oder nicht vorhanden ist. Im Zeitalter immer größer werdender Datensätze ist eine Simulationsstudie für diese Art von Algorithmen dringend vonnöten. Diese Arbeit[1] versucht diese Lücke zu schließen.

Ziel der Arbeit ist es geeignete Algorithmen für das Clustern hochdimensionaler binärer Daten anhand einer Simulation zu evaluieren. Sowohl die Güte der Clusterlösungen ist dabei Ziel der Analyse, als auch die Laufzeit und der Speicherbedarf, da dies im Big-Data-Kontext wichtige Kriterien sind, welche über die Angemessenheit der Verwendung eines Algorithmus für eine bestimmte Aufgabe eine zentrale Rolle spielen. Die Algorithmen selbst werden auf simulierte Datensätze unterschiedlichster Konfiguration angewendet. So wird die Anzahl der Beobachtungen, Variablen und Cluster variiert, sowie die Prävalenz der vorhandenen Einsen in den Daten, als auch die Korrelationsstruktur. Zudem wird die Art und Weise wie die Daten simuliert werden in der Analyse als weiterer Punkt evaluiert. Die unterschiedlichen Arten und Weisen, wie die Daten generiert werden, sind in dieser Arbeit mit *Szenario* benannt. Siehe dazu Paragraph *Korrelation* in Abschnitt 4.

Die Arbeit teilt sich in mehrere Abschnitte auf. Zunächst wird eine Übersicht über den Stand der Forschung gegeben. Dabei wird auf Studien eingegangen, die verschiedene Clusteralgorithmen anhand von simulierten und empirischen Daten vergleichen und evaluieren. Folgend werden einige relevante Anwendungsgebiete aufgeführt. Schließlich muss geklärt werden, wozu die Clusteranalyse von binären Daten überhaupt in und außerhalb der Sozialwissenschaften verwendet werden kann. Dabei ist vor allem das Einteilen von Social-Media-Beiträgen zu inhaltlich zusammengehörenden Gruppen zu nennen. Der Simulationsaufbau wird im anschließenden Abschnitt dargestellt. Dabei wird auf die unterschiedlichen Konfigurationen der Simulationsparameter eingegangen, sowie die Erstellung der simulierten Clusterstruktur vorgestellt. Zudem werden die technischen Details der Simulationsbedingungen besprochen. Darauf folgt in Abschnitt 5 die Beschreibung einiger Simulationsmethoden für korrelierte binäre Daten und die

[1] Die Idee und die einzelnen Faktoren des Simulationsaufbaus stammen von Prof. Dr. Rainer Schnell.

Darlegung der Simulationsstrategie der simulierten Daten. Dazu wird erörtert wie die nötigen Korrelationsmatrizen erzeugt werden und es wird eine Analyse der simulierten Daten durchgeführt. Zusätzlich wird eine effiziente Variante der technischen Umsetzung der Datensimulation diskutiert. Zur angemessenen Evaluation der Ergebnisse werden geeigente Kriterien benötigt. Abschnitt 6 erläutert daher einige Beurteilungskriterien zur Evaluation der Ergebnisse von Clusterlösungen. Die nächsten beiden Abschnitte beschreiben die verwendeten Clusteralgorithmen, sowie Clusteralgorithmen, die zunächst als potenziell geeignet betrachtet wurden, jedoch aus verschiedensten Gründen nicht verwendet werden. Die Ergebnisse der Simulation werden in Abschnitt 9 vorgestellt. Abschließend folgt das Fazit der Arbeit mit den wichtigsten Ergebnissen, Limitationen und Hinweise für weiterführende Forschung, die an diese Arbeit anknüpfen könnte. Die Simulation wurde komplett in der statistischen Programmiersprache `R` (R Core Team, 2021) umgesetzt. Im Anhang befindet sich der verwendete R-Code und einige weitere Abbildungen, auf die an entsprechender Stelle verwiesen wird.

2. Stand der Forschung

Der Bereich der Clusteranalyse umfasst eine riesige Menge an Literatur, mit unzähligen unterschiedlichen Algorithmen, Bewertungskriterien und Methoden zur Auswahl an verwendeten Variablen oder zur Erkennung der Anzahl der vorliegenden Cluster. Entsprechend findet sich eine Vielzahl an Übersichtsarbeiten, die einen Überblick über populäre Algorithmen, deren Eigenschaften und Anwendungsgebiete geben (siehe Bspw. Jain et al., 1999; Berkhin, 2002; Jiang et al., 2004; Andreopoulos et al., 2009; Arora & Chana, 2014; Xu & Tian, 2015; Dave & Gianey, 2016; Oyelade et al., 2016; Brusco et al., 2017; Saxena et al., 2017; Benabdellah et al., 2019; Dafir et al., 2020; Mahdi et al., 2021; Ezugwu et al., 2022). Studien die die verschiedenen Algorithmen tatsächlich anhand von simulierten oder empirischen Daten vergleichend evaluieren, sind demgegenüber jedoch vergleichsweise rar (Van Mechelen & Vach, 2019). Die folgenden Abschnitte dieses Kapitels geben eine Übersicht über Studien, die anhand von simulierten oder empirischen Daten unterschiedliche Algorithmen vergleichen und die Performance evaluieren. Die Abschnitte sind unterteilt in Studien, die sich mit metrischen, kategorialen, Daten mit gemischtem Skalenniveau und binären Daten beschäftigen.

2.1. Metrische Daten

Eine Studie, die 5 verschiedene Algorithmen anhand von empirischen Daten vergleicht, stammt von Dash und Misra (2018). Die verwendeten Algorithmen sind Hybrid Swarm Based Clustering (HSC) (Kennedy & Eberhart, 1997), K-Means (MacQueen, 1967), Partitioning Around Medoids (PAM) (Kaufman & Rousseeuw, 2005), Vector Quantization (VQ) und Agglomerative Nesting (AGNES). Bei den verwendeten Datensätzen handelt es sich um empirische Datensätze aus dem Bereich der Krebsforschung und beinhalten Gensequenzen. Es werden 5 verschiedene Datensätze verwendet. Die Dimensionalität der Datensätze ist dabei jedoch relativ gering. So besitzen die Datensätze lediglich ungefähr 100 Beobachtungen, jedoch 1.000 bis 10.000 Variablen. Diese werden jedoch nicht alle

verwendet und es werden nur 5 bis 9 wichtige Variablen für das Clustering herangezogen. Ziel der Studie ist es den geeignetsten Algorithmus für das Clustern von Datensätzen zu finden, die eben genau die Eigenschaften aufweisen, die im Feld der Gensequenzen zu finden sind. Dash und Misra (2018, S. 157) kommen zu dem Ergebnis, dass HSC die konsistentesten Ergebnisse liefert, obgleich dieser Algorithmus nicht bei jedem Datensatz die besten Ergebnisse vorweisen kann.

Eine weitere Studie auf der Basis von Daten aus der Biologie in Form von Gensequenzen wurde von Thalamuthu et al. (2006) durchgeführt. Verglichen werden die Algorithmen Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005), K-Means (MacQueen, 1967), Partitioning Around Medoids (PAM, Kaufman & Rousseeuw, 2005), Self Organizing Maps (SOM, Kohonen, 2001), Mixture Model-Based Clustering (Yeung et al., 2001) und Tight Clustering (Tseng & Wong, 2005). Es werden 4 verschiedene Datensätze verwendet. Die Dimensionen der Datensätze sind dabei 77 Beobachtungen mit 1.663 Variablen, 45 Beobachtungen mit 1.744 Variablen, 114 Beobachtungen mit 2.570 Variablen und 203 Beobachtungen mit 1.920 Variablen. Die besten Ergebnisse lieferten Mixture Model-Based Clustering und Tight Clustering. Die schlechtesten Ergebnisse konnten bei SOM und Agglomerative Hierarchical Clustering beobachtet werden.

Budayan et al. (2009) vergleichen in ihrer Studie Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005), K-Means (MacQueen, 1967), SOM (Kohonen, 2001) und Fuzzy C-Means (FCM, Bezdek, 1981). Getestet werden die Algorithmen an einem empirischen Survey Datensatz deren Beobachtungen aus Firmen bestehen und die Variablen aus unterschiedlichen Attributen dieser Firmen. Die Anzahl der Beobachtungen ist mit 84 Firmen angegeben und 13 Variablen. Es handelt sich somit wiederum um einen sehr kleinen Datensatz. Aufgrund der Ergebnisse empfehlen Budayan et al. (2009) die Verwendung von SOM und FCM gegenüber den restlichen Algorithmen.

Eine explizit im Big Data Kontext durchgeführte Vergleichsstudie auf der Grundlage empirischer Daten stammt von Fahad et al. (2014). Neben einer Übersicht über verschiedene Clusteralgorithmen allgemein befasst sich die Studie mit folgenden Algorithmen: FCM (Bezdek, 1981), BIRCH (Zhang et al., 1996), DENCLUE (Hinneburg & Keim, 1998), OptiGird (Hinneburg & Keim, 1999) und dem Expectation Maximization Algorithmus (EM, Dempster et al., 1977). Es werden 10 verschiedene Datensätze verwendet, die aus unterschiedlichsten Bereichen kommen. Die Anzahl der Beobachtungen reicht dabei

von 400 bis eine Million. Die Anzahl der Variablen liegt bei den meisten verwendeten Datensätzen im niedrigen zweistelligen Bereich. Die höchste Anzahl an Variablen ist mit 149 angegeben. Die Anzahl der Cluster beläuft sich meist auf 2. Ein Datensatz weist 3 Cluster auf, ein weiterer Datensatz weist 5 Cluster auf und die meisten Cluster sind bei einem Datensatz mit 12 Clustern zu finden. Die Ergebnisse legen nahe, dass zwar kein Algorithmus universell gute Ergebnisse geliefert hat, jedoch FCM und der EM Algorithmus insgesamt die besten Ergebnisse erzielten. Diese waren jedoch auch die ineffizientesten und für große Datensätze eher ungeeignet. Diesbezüglich schnitten die anderen Algorithmen besser ab.

Eine Vergleichsstudie die ebenfalls das Ziel hat geeignete Clusteralgorithmen für große Datensätze zu finden stammt von Kumar et al. (2016). So werden verschiedene K-Means basierte Algorithmen, CURE (Guha et al., 2001) und der eigens entwickelte Algorithmus clusiVAT verglichen. Es werden sowohl simulierte Daten verwendet, als auch drei große empirischer Datensätze. Bei den simulierten Datensätzen handelt es sich um Daten mit 100.000, 200.000, 500.000 und 1.000.000 Beobachtungen und 2, 100, 200 und 500 Variablen. Die Anzahl der Cluster ist mit 3, 4 und 5 angegeben. Bei den empirischen Daten wurde der Datensatz *Forest Cover Data* mit 581.012 Beobachtungen und 54 kontinuierlichen und binären Variablen verwendet. Zudem der Datensatz *KDD Cup 99* mit 4.292.637 Beobachtungen und 41 Variablen. Die besten Ergebnisse zeigen sich bei clusiVAT. Das gilt sowohl für die simulierten als auch für die empirischen Datensätze. Zudem wird hervorgehoben, dass clusiVAT wesentlich effizienter als die anderen Algorithmen ist und die K-Means basierten Algorithmen Probleme durch die initiale Phase der Algorithmen haben.

Eine Vergleichsstudie mit 42 verschiedenen und damit sehr vielen empirischen Datensätzen findet sich bei Hennig (2022). Verglichen werden die Algorithmen K-Means (MacQueen, 1967), CLARA (Kaufman & Rousseeuw, 2005), Gaussian Mixture Model (mclust, Fraley & Raftery, 2002), Mixture of skew t-distributions (emskewt, Lee & McLachlan, 2013), Mixture of t-distributions (teigen, McLachlan & Peel, 2000), Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005) in den Varianten Single Linkage, Average Linkage und Complete Linkage und schließlich Spectral Clustering (Ng et al., 2001). Die verwendeten Datensätze können alle als klein beschrieben werden und stammen vom UCI Machine Learning Archive aus unterschiedlichen inhaltlichen Gebieten. Die kleinsten Datensätze haben weniger als 100 Beobachtungen, während der größte

Datensatz lediglich 4.601 Beobachtungen enthält. Die Anzahl der Variablen ist ebenfalls in den meisten Datensätzen relativ klein. Lediglich 3 Datensätze weisen mehr als 50 Variablen auf. Die meisten Datensätze beinhalten jedoch deutlich weniger Variablen. Die Anzahl der Cluster rangiert bei den meisten Datensätzen im niedrigen einstelligen Bereich. Im Ergebnis gibt es keinen Algorithmus, der universell über alle Datensätze die besten Ergebnisse erzielt. Je nach Beschaffenheit der Datensätze, sind unterschiedliche Algorithmen besser darin die korrekte Clusterstruktur zu identifizieren als andere Algorithmen.

Murugesan et al. (2021) vergleichen die Algorithmen Spectral Clustering (Shi & Malik, 2000), DBSCAN (Ester et al., 1996) und K-Means (MacQueen, 1967) anhand von simulierten und empirischen Datensätzen. Die simulierten Datensätze sind wie bei den meisten Vergleichsstudien von geringem Umfang mit wenigen hundert Beobachtungen, 2 bzw. 3 Variablen und einer Clusteranzahl im einstelligen Bereich. Die Daten sind dabei so simuliert, dass in den verschiedenen simulierten Szenarien unterschiedliche Formen der Cluster erschaffen werden, wie z.B. spiralförmige Cluster, kreisförmige Cluster und getrennte bzw. überlappende Cluster. Bei den empirischen Datensätzen handelt es sich zum einen um einen Datensatz über Fahrräder, bestehend aus 97 Beobachtungen und 35 Variablen, zum anderen ein Datensatz über Waldbrände mit 517 Beobachtungen und 4 Variablen und letztlich ein Datensatz über Immobilien mit 5.854 Beobachtungen und 8 Variablen. Murugesan et al. (2021) kommen zu dem Ergebnis, dass keiner der Algorithmen universal zu empfehlen ist und über alle Datensätze die besten Ergebnisse liefert. In Szenarien mit gut getrennten Clustern, zeigt DBSCAN die besten Ergebnisse, wobei bei überlappenden Clustern Spectral Clustering und K-Means bessere Ergebnisse liefern. Spectral Clustering liefert nochmals etwas bessere Ergebnisse als K-Means, jedoch wird angemerkt, dass dies auf Kosten einer erhöhten Laufzeit geschieht.

Ebenfalls im biomedizinischen Bereich angesiedelt, vergleichen Wiwie et al. (2015) 13 verschiedene Clusteralgorithmen an empirischen Datensätzen. Bei den verwendeten Algorithmen handelt es sich um Affinity Propagation (Frey & Dueck, 2007), Clusterdp (Rodriguez & Laio, 2014), ClusterONE (Nepusz et al., 2012), DBSCAN (Ester et al., 1996), Fanny (Kaufman & Rousseeuw, 2005), Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005), K-Means (MacQueen, 1967), PAM (Kaufman & Rousseeuw, 2005), Markov Clustering (van Dongen, 2000), MCODE (Bader & Hogue, 2003), SOM (Kohonen, 2001), Spectral Clustering (Shi & Malik, 2000) und Transitivity Clustering (Wittkop et al., 2010). Es werden keine Daten simuliert, sondern die Datenbasis besteht komplett

aus 24 unterschiedlichen empirischen Datensätzen aus der Biomedizin. Die Anzahl der Beobachtungen reicht dabei von 34 bis 5.000. Die Anzahl der Variablen ist meist mit lediglich 2 angegeben. Bei manchen Datensätzen fehlt diese Information komplett. Das Maximum liegt bei 999 Variablen. Wie bei den meisten Vergleichsstudien, kann auch hier kein Algorithmus identifiziert werden, der bei allen Datensätzen die besten Ergebnisse liefert. Zusätzlich wird bei den verwendeten Datensätzen noch *Rauschen*[1] hinzugefügt und die Algorithmen erneut angewendet. Entsprechend werden etwas schlechtere Ergebnisse beobachtet (F-Score[2] fällt um bis zu 20 %). Die meisten Algorithmen weisen insgesamt sowohl sehr schlechte als auch sehr gute Ergebnisse auf. Insgesamt werden die besten Ergebnisse jedoch bei Transitivity Clustering, Agglomerative Hierarchical Clustering und PAM beobachtet.

2.2. Kategoriale Daten

Die meisten Vergleichsstudien von Clusteralgorithmen befassen sich mit metrischen Daten. In der Studie von Anderlucci und Hennig (2014) hingegen wird eine kategoriale Datengrundlage genutzt. Jedoch werden lediglich zwei Algorithmen in Form von Latent Class Clustering (LCC, Vermunt & Magidson, 2002) und PAM (Kaufman & Rousseeuw, 2005) miteinander verglichen. Bei der Datengrundlage handelt es sich um simulierte Daten, was die Möglichkeit eröffnet einzelne Parameter der Daten zu kontrollieren. So unterscheiden sich die Konfigurationen der Datensätze in der Anzahl der Cluster (2, 3, 5), Anzahl der Variablen (4, 12), Anzahl der Kategorien (2, 4, 8), Clustergrößen (unterschiedlich, ähnlich), Trennbarkeit der Cluster (klar getrennt, nicht klar getrennt) und Anzahl der Beobachtungen (wenig, viel). Generell handelt es sich jedoch um kleine Datensätze. Die Konfigurationen mit wenigen Beobachtungen beinhalten lediglich wenige hundert Beobachtungen und die Konfigurationen mit vielen Beobachtungen beinhalten 1,000 Beobachtungen. Anderlucci und Hennig (2014) kommen zu dem Ergebnis, dass LCC insgesamt die etwas besseren Ergebnisse liefert, jedoch in manchen Konfigurationen PAM die bessere Wahl darstellt. Dies ist vor allem der Fall, wenn die Clustergrößen ähnlich sind.

Ein weiterer Vergleich von Algorithmen explizit für kategoriale Daten ist bei Hautamäki et al. (2014) zu finden. Dabei werden verschiedene Varianten von K-Means (MacQueen,

[1] Zur Definition von Rauschen siehe Abschnitt 4.
[2] Für eine Definition des F-Score siehe Abschnitt 6.2.1.

1967) mit ROCK (Guha et al., 2000) und Agglomerative Categorical Clustering with Entropy Criterion (ACE, Chen & Liu, 2005a) verglichen. Die verschiedenen Varianten von K-Means basieren auf unterschiedlichen Optimierungskriterien im *Estimation Step*. Namentlich heißen die Varianten K-Distributions, K-Histograms, K-Modes, K-Medoids und K-Entropies. Die Datenbasis stellen 6 unterschiedliche kategoriale Datensätze vom UCI Machine Learning Archive dar. Der kleinste Datensatz hat dabei 47 Beobachtungen und der größte Datensatz weist 2.458.285 Beobachtungen auf. Die Anzahl der Variablen ist bei allen Datensätzen im niedrigen bis mittleren zweistelligen Bereich. ACE zeigte die besten Ergebnisse, ist jedoch nicht für große Datensätze geeignet. In diesem Fall wird von Hautamäki et al. (2014) K-Entropies empfohlen.

2.3. Gemischtes Skalenniveau

Clusteralgorithmen, die geeignet für das Clustern von Daten mit sowohl kontinuierlichen als auch kategorialen Variablen sind, werden bei Jimeno et al. (2021) verglichen. Die verwendeten Algorithmen sind KAMILA (Foss et al., 2016) und K-Prototypes (Huang, 1998). Zudem werden nach einer vorgelagerten multiplen Korrespondenzanalyse die Algorithmen K-Means (MacQueen, 1967), Fuzzy C-Means (FCM, Bezdek, 1981), Probabilistic Distance (Iyigun & Ben-Israel, 2008) und Student-t Mixture Models (Andrews et al., 2011) verwendet. Der Studie liegt ein Simulationsdesign mit 27 verschiedenen Konfigurationen zugrunde. Variiert werden die Anzahl der Cluster (2, 5, 7), das Level der Überlappung der Cluster (30 %, 60 %, 80 %) und das Ratio von kontinuierlichen zu kategorialen Variablen (1:3, 1:1, 3:1). Die Anzahl der Beobachtungen ist bei jeder Konfiguration auf 1.920 Beobachtungen gesetzt und die Anzahl der Variablen auf 128. Insgesamt zeigt KAMILA die besten Ergebnisse, wobei auch K-Prototypes konsistent gute Ergebnisse liefert.

Eine weitere Simulationsstudie, die sich mit dem Vergleich von Clusteralgorithmen explizit bei der Verwendung von Datensätzen mit gemischtem Skalenniveau befasst, stammt von Costa et al. (2022). Genauer gesagt, handelt es sich bei den verwendeten Variablen sowohl um kontinuierliche als auch um kategoriale Variablen. Die verwendeten Algorithmen sind PAM (Kaufman & Rousseeuw, 2005), K-Prototypes (Huang, 1998) und KAMILA (Foss et al., 2016). Dabei werden unterschiedliche Varianten von PAM und K-Prototypes verwendet, indem unterschiedliche Distanzmaße zugrunde gelegt werden oder

eine Dimensionsreduktion vorgelagert wird. Der Simulationsaufbau besteht aus folgenden Faktoren: Anzahl der Cluster (3, 5, 8), Anzahl der Beobachtungen (100, 600, 1.000), Anzahl der Variablen (8, 12, 16), Überlagerung der Cluster (sehr wenig, wenig, mittel, hoch, sehr hoch), Anteil der kategorialen Variablen (20 %, 50 %, 80 %), Größe der Cluster (gleichgroß, 10 % in einem Cluster und der Rest gleichmäßig auf alle anderen verteilt) und zuletzt zwei verschiedenen Ausprägungen der Kovarianzmatrix. Zusammenfassend kommen Costa et al. (2022) zu dem Schluss, dass KAMILA die besten Ergebnisse liefert und bestätigen damit die gute Performance von KAMILA, die auch schon bei Jimeno et al. (2021) beobachtet werden konnte.

Preud'homme et al. (2021) nehmen sich ebenfalls der Problemstellung an, die geeignetsten Algorithmen für das Clustern von Datensätzen mit gemischtem Skalenniveau anhand einer Simulationsstudie zu finden. Die verglichenen Algorithmen sind KAMILA (Foss et al., 2016), Latent Class Analysis (Vermunt & Magidson, 2002), Latent Class Model (Marbac & Sedki, 2018), Clustering by Mixture Modeling (Everitt, 1988), Agglomerative Hierarchical Clustering, PAM (Kaufman & Rousseeuw, 2005) und K-Prototypes (Huang, 1998). Die simulierten Daten werden anhand folgender Parameter variiert: Anzahl der Beobachtungen (300, 600, 1200), Anzahl der Cluster (2, 6, 10), Ratio zwischen kontinuierlichen und kategorialen Variablen, Anteil relevanter vs. irrelevanter Variablen[3] (20 %, 50 %, 90 %) und die Separierung der Cluster (niedrig, mittel, hoch). Wiederum lassen sich die besten Ergebnisse bei KAMILA finden und auch LCM liefert gute Ergebnisse. Preud'homme et al. (2021) resümieren, dass bei gemischen Skalenniveaus modellbasierte Algorithmen mit KAMILA und LCM scheinbar generell die besseren Ergebnisse liefern als Distanzbasierte Algorithmen wie PAM und K-Prototypes.

2.4. Binäre Daten

In einer Studie von Sevcik et al. (2011) werden drei Clusteralgorithmen für das Clustern binärer Datensätze verglichen. Dabei handelt es sich um Agglomerative Hierarchical Clustering, Divisive Clustering (Kaufman & Rousseeuw, 2005) und eine Variante von K-Means names Direct Clustering (Karypis, 2002). Bei den verwendeten Daten handelt es sich um empirische Datensätze bestehend aus Dokumenten von thematisch verschiedenen Nachrichtengruppen[4]. Aus jedem Themenbereich wird eine Stichprobe von Dokumenten

[3] Diese werden von Preud'homme et al. (2021) als *Noise* bezeichnet.
[4] Vom englischen Begriff *Newsgroup*.

gezogen, die damit die einzelnen Cluster bilden. Der so entstandene Datensatz wird in eine Term-Document Matrix (Manning et al., 2008, S. 3–4) umgewandelt und nimmt daher eine binäre Datenstruktur an. Die Anzahl der Beobachtungen wird mit 2.016 angegeben und die Anzahl der Variablen mit 34.438. Dieser Datensatz wird einmal aus 6 verschiedenen Nachrichtengruppen erstellt um 6 Cluster zu erhalten und ein weiteres mal aus 20 Nachrichtengruppen um 20 Cluster zu erhalten. Die Ergebnisse weisen darauf hin, dass Agglomerative Hierarchical Clustering nicht geeignet ist diese Form von Daten adäquat zu Clustern, zumal die Laufzeit als sehr hoch angegeben wurde. Die besten Ergebnisse werden bei Direct Clustering beobachtet.

Eine weitere Vergleichsstudie für Algorithmen zum Clustern binärer Datensätze stammt von Trejos-Zelaya et al. (2021). Hier werden die Optimierungsalgorithmen Simulated Annealing (Kirkpatrick et al., 1983), Threshold Accepting (Dueck & Scheuer, 1990) und Tabu Search (Glover, 1989) verwendet, um ein Unähnlichkeitskriterium innerhalb von Clustern zu optimieren und so zu einer Clusterlösung zu kommen. Zusätzlich werden die Algorithmen Agglomerative Hierarchical Clustering und PAM (Kaufman & Rousseeuw, 2005) verwendet. Die Datenbasis stellen simulierte Daten dar. Diese werden anhand folgender Parameter variiert: Anzahl der Beobachtungen (120, 1200), Anzahl der Cluster (3, 5), Verteilung der Clustergrößen (gleiche Größe aller Cluster, ein großer Cluster mit 50 % der Beobachtungen und der Rest verteilt auf die restlichen Cluster) und Trennung der Cluster (eindeutig getrennt, nicht eindeutig getrennt). Die Anzahl der Variablen ist mit 120 angegeben. Die besten Ergebnisse werden mit PAM und Agglomerative Hierarchical Clustering erreicht. Simulated Annealing liefert ebenfalls gute Ergebnisse. Die restlichen Algorithmen weisen keine guten Ergebnisse auf. Als zusätzliche Hürde wird bezüglich Simulated Annealing jedoch das Finden der optimalen Parameter genannt.

Tamasauskas et al. (2012) vergleichen 10 unterschiedliche Linkage Methoden für Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005) anhand binärer Datensätze. Bei den Linkage Methoden handelt es sich um Average Linkage, Centroid Linkage, Complete Linkage, Density Linkage, Flexible-Beta, McQuitty's, Median, Single Linkage, Two-Stage Density Linkage und Ward's. Als Distanzmetrik wird die Metrik Dmatch verwendet. Die verwendeten Daten sind simuliert, bestehen aus 120 Beobachtungen und werden anhand der Anzahl der Variablen (5, 10, 15), Anzahl der Cluster (2, 3, 4) und der Trennung der Cluster (eindeutig getrennte Cluster, nicht deutlich getrennte Cluster, überhaupt nicht deutlich getrennte Cluster) variiert. Keine Methode liefert

Tamasauskas et al. (2012) zufolge universell gute Ergebnisse. Die besten Ergebnisse sind jedoch bei Complete Linkage, Flexible-Beta und Ward's zu finden. Eindeutig getrennte Cluster können besser erkannt werden als nicht deutlich getrennte Cluster und mit steigender Anzahl an Variablen lassen sich ebenfalls bessere Ergebnisse erzielen.

Ein weiterer Vergleich zwischen den möglichen Varianten von Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005), die sich neben den verwendeten Linkage Methoden auch in den zugrundeliegenden Distanzmetriken unterscheiden, findet sich bei Pratsinakis et al. (2020). Thematisch ist die Studie im Bereich der Biologie angesiedelt. Die Datenbasis besteht entsprechend aus binären Daten in Form von molekularen Markern. Zur Dimensionalität der verwendeten Datenbasis wird keine Angabe gemacht. Aufgrund der abgebildeten Dendrogramme, kann jedoch von Daten moderater Größe ausgegangen werden. Pratsinakis et al. (2020) kommen zu dem Ergebnis, dass die weit verbreitetste Variante bestehend aus Average Linkage und euklidischer Distanz zwar zum Clustern des gegebenen Datenmaterials geeignet ist, jedoch eine Kombination aus 37 anderen Linkage Methoden und Distanzmetriken zu ähnlich guten Ergebnisse kommt.

In der Studie von d'Angella und Hennig (2022) werden neben Agglomerative Hierarchical Clustering (Kaufman & Rousseeuw, 2005) mit Single Linkage, Average Linkage und Complete Linkage, K-Modes (Huang, 1998), PAM (Kaufman & Rousseeuw, 2005) und Latent Class Analysis (Vermunt & Magidson, 2002) auch Algorithmen verwendet, die für das Clustern kontinuierlicher Daten gedacht sind. Diesen wird jedoch eine Multidimensionale Skalierung (Borg & Groenen, 2005) vorgelagert und der Output wird dann zum Clustern verwendet. Bei der Multidimensionalen Skalierung wird zum einen die Variante Classical Scaling (Torgerson, 1958) und Ratio MDS (Borg & Groenen, 2005) verwendet. Die genutzten Algorithmen sind K-Means (MacQueen, 1967), Gaussian Mixture Model (Fraley & Raftery, 2002) und pdfCluster (Azzalini & Menardi, 2016). Die Datenbasis ist simuliert und soll biogeographische Daten repräsentieren, die das gleichzeitige Aufkommen von unterschiedlichen Spezies in einem geographischen Areal indizieren. Variiert werden die Anzahl der Beobachtungen (300, 400), Überlappung der Cluster (wenig, mittel, hoch), Anzahl der Cluster (3, 4) und Größe der Cluster (gleiche Größe der Cluster, ungleiche Größe der Cluster). Die Anzahl der Variablen ist mit 60 angegeben. Generell lässt sich sagen, dass alle Methoden bei schlecht getrennten Clustern keine guten Ergebnisse abliefern. Agglomerative Hierarchical Clustering erreicht insgesamt die schlechtesten Ergebnisse. K-Means und Gaussian Mixture Model liefert

die besten Ergebnisse. Alle Algorithmen zeigen jedoch eine relativ hohe Varianz in den Ergebnissen mit sowohl guten als auch schlechten Ergebnissen.

2.5. Fazit

Zusammenfassend lassen sich aufgrund des Forschungsstandes in der Literatur keine Algorithmen finden, die sich klar universal empfehlen lassen. Lediglich KAMILA scheint bei gemischem Skalenniveau der Algorithmus zu sein, der die besten Ergebnisse liefert. In den anderen Fällen hängt es stark von den Eigenschaften der Daten ab, welcher Algorithmus als geeignet angesehen werden kann. Zudem fällt auf, dass gerade im Big Data Kontext nur wenige Studien existieren. Zwar gibt es durchaus eine Vielzahl an Literatur, die sich generell mit Clusteranalyse im Big Data Kontext befasst, wie bspw. Saldhi et al. (2014), Shirkhorshidi et al. (2014), Masulli et al. (2015), Bin (2018), Nasraoui und N'Cir (2019), Zgurovsky und Zaychenko (2019), Dafir et al. (2020) und Mahdi et al. (2021) und auch existieren Lehrbücher für Clusteranalyse und Big Data wie bspw. Kogan (2007) und Zgurovsky und Zaychenko (2020), aber der Großteil an Studien, die verschiedene Clusteralgorithmen an simulierten oder empirischen Datensätzen anwenden und die Ergebnisse vergleichen, befasst sich mit kleinen Datensätzen. Ebenfalls sind die Anzahl der Cluster und vor allem auch die Anzahl der Variablen zumeist sehr gering.

3. Anwendungsgebiete

Innerhalb der letzten 20 Jahre ist generell, aber vor allem auch im Bereich der Sozialwissenschaften, ein immer größerer Bedarf an großen Datenmengen beobachtet worden. Gerade in den Sozialwissenschaften lassen sich gesellschaftliche Fragestellungen mithilfe großer Datenmengen analysieren, wie sie beispielsweise durch die sozialen Medien oder groß angelegte Bevölkerungsbefragungen bzw. Bevölkungszählungen wie dem Zensus produziert werden. Sowohl der *Social Science Citation Index* als auch *Scopus* wurden nach dem Suchwort „Big Data“ durchsucht. Scopus wurde dabei auf den Fachbereich Social Sciences beschränkt. In Abbildung 3.1 sind die Anzahl der Publikationen im Laufe der Zeit abgebildet, welche unter diesem Suchbegriff gefunden wurden, woran sich der beschriebene Trend hin zu großen Datenmengen in den Sozialwissenschaften gut erkennen lässt und die Relevanz der Evaluation geeigneter Methoden für die Sozialwissenschaften unterstreicht. Auch außerhalb der Sozialwissenschaften spielt das Clustern großer binärer Datenmengen eine wichtige Rolle. So gibt Wang et al. (1999) beispielsweise die Verwendung in der Medizin, beim Webbrowsing und Image Retrieval als mögliche Bereiche an. Folgend werden einige Anwendungsgebiete aufgeführt, die einen direkten oder indirekten Bezug zu den Sozialwissenschaften haben.

3.1. Social Media

Ein großes Anwendungsgebiet stellt das Einteilen von Beiträgen zu Gruppen in Social-Media-Portalen dar. Um dies zu tun, müssen zunächst diese Einträge in Textform so aufbereitet werden, dass es möglich ist, entsprechende Algorithmen anzuwenden. Durch die Möglichkeit einen Textkorpus in eine *Term-Document-Matrix* (Manning et al., 2008, S. 3–4) zu transformieren, lassen sich Textkörper in binäre Datensätze überführen und entsprechend analysieren. In dieser ij-Matrix stellt jede Zeile ein Dokument i dar. Die Spalten ergeben sich als die Vereinigungsmenge aller verwendeten Terme (Wörter) der Textkörper aller verwendeten Dokumente. Jede Spalte steht dabei für einen Term j. Enthält der Textkörper von Dokument i nun Term j, nimmt der Eintrag der Matrix in

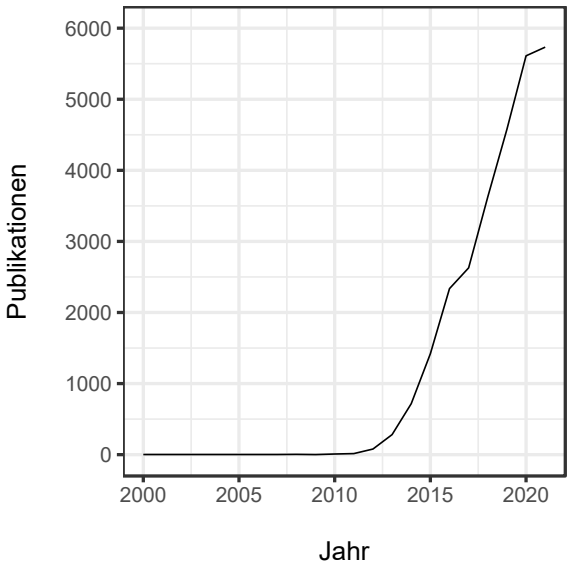


Abbildung 3.1.: Anzahl Ergebnisse des Suchbegriffs Big Data innerhalb der Sozialwissenschaften der Datenbank Scopus und im Social Science Citation Index

Zelle ij den Wert 1 an. Andernfalls nimmt die Zelle den Wert 0 an. Auf diese Weise lassen sich zum Beispiel Beiträge in unterschiedlichen Social Media Plattformen in binäre Daten überführen und mithilfe von Clusteralgorithmen für binäre Daten zu Gruppen zusammenfassen. Dies wurde so zum Beispiel von Bibi et al. (2020, S. 68584) zum Clustern von Twitterbeiträgen vorgeschlagen. Auch Yue et al. (2018) identifizieren Clusteranalyse als soziologisch relevante Methode zur Analyse von Social Media Daten.

Darüber hinaus wird die Analyse von Social Media Daten als sozialwissenschaftlich relevant verdeutlicht, indem sie beispielsweise verwendet wurde, um die soziale Perzeption von Wahlen zu erforschen (Anstead & O'Loughlin, 2014). Ein anderes Beispiel ist der Zusammenhang zwischen Nutzung sozialer Medien und politischer Partizipation (Boulianne, 2015) oder das Identifizieren antagonistischer Gruppen in Bezug auf soziale Sachverhalte (Zhang et al., 2013).

Ein weiteres Anwendungsgebiet ist das Erkennen des Phänomens *Astroturfing* in sozialen Medien und bspw. Kommentarspalten in Onlineartikeln oder Nachrichten. Astroturfing wird dabei von Kovic et al. (2018, S. 71) wie folgt definiert: „Digital astroturfing is a form of manufactured, deceptive and strategic top-down activity on the Internet initiated by political actors that mimics bottom-up activity by autonomous individuals". Es wird dabei also von einigen wenigen, in der Regel politisch motivierten, Personen

unter der Verwendung vieler unterschiedlicher Pseudonyme vorgegeben, dass eine Graswurzelbewegung existiert, welche Stimmung in eine bestimmte Richtung macht, um den öffentlichen Diskurs eines Sachverhalts zu manipulieren. Iqbal et al. (2010) überführen den Textkörper in eine Matrix ähnlich der Term-Document-Matrix mit dem Unterschied, dass hier die Spalten nicht aus Termen, sondern aus *Stylistic Features* (bspw. das Verhältnis von Anzahl der Buchstaben zur Anzahl der Zeichen insgesamt) besteht. Das Resultat ist wiederum eine binäre Datenmatrix, welche dann mittels Clusteranalyse nach Autorenschaft eingeteilt wird. Peng et al. (2016) wandeln den Textkorpus in n-Gramme und verwenden verschiedene Data-Mining Methoden, um den analysierten Inhalt individuellen Entitäten zuzuordnen.

3.2. Warenkörbe

In der Marktforschung ist ein zentrales Ziel die Analyse des Kaufverhaltens von Personen. Entsprechend ist es hilfreich, Personen die unterschiedliche Interessen haben und demnach verschiedene Arten von Artikeln erwerben, in Gruppen zusammenzufassen. Die durch den Kauf von Artikeln entstehende Warenkörbe bilden hierbei den zentralen Datenkörper. Um diese nun zu analysieren und die zugehörigen Personen zu Clustern zusammenzufassen, lässt sich das Konzept der Term-Document-Matrix auf Warenkörbe übertragen. Analog zur Term-Document-Matrix stellen die Zeilen die Warenkörbe von Personen dar und die Spalten die unterschiedlichen Artikel, welche potenziell erworben werden können. Liegt ein Artikel im betrachteten Warenkorb vor, nimmt der korrespondierende Eintrag den Wert 1 an. Liegt er nicht vor, nimmt er entsprechend eine 0 an (Tummala et al., 2018, S. 9). Nun lassen sich Clusteralgorithmen auf die Daten anwenden und Warenkörbe zu Clustern zusammenfassen, die auf ein ähnliches Kaufverhalten hinweisen.

3.3. Binary Programs

Besonders in der Informatik ist das Identifizieren und Clustern von ähnlich funktionierenden Programmen in Form von Binärdateien von großem Interesse. Auf diese Weise lassen sich zum Beispiel ähnlich geartete Schadprogramme zu Gruppen zusammenfassen und von Programmen unterscheiden, welche keine Schadsoftware enthalten. Oprisa et al. (2013) nennen in diesem Atemzug auch das Identifizieren von plagiierten Computerprogrammen von Studierenden, welche lediglich Variablennamen und Ähnliches abändern, um nicht bei der Suche nach Duplikaten aufzufallen, wobei der Inhalt des Codes aber

essenziell derselbe ist. Das Clustern wird nach Oprisa (2015) möglich, indem die *Assembly Instructions* der Programme und deren zugehörigen *OpCodes* extrahiert werden, die die Anweisungen an den Prozessor enthalten, welche Operationen ausgeführt werden. Diese Anweisungen können zu Strings zusammengefasst und in n-Gramme aufgeteilt werden. Unter einem n-Gramm wird in diesem Kontext n aufeinanderfolgende Zeichen einer Zeichenkette verstanden. Diese stellen dann jeweils ein *Feature* dar. Nun kann wieder eine Term-Document-Matrix erstellt werden, wobei die Features in den Spalten zu finden sind und die Programme in den Zeilen. Entsprechend kann nun die resultierende Matrix mit binären Einträgen mit den entsprechenden Algorithmen geclustert werden.

3.4. Blocking von Bloom Filtern beim Record Linkage

Ein zusätzliches potenzielles Anwendungsgebiet liegt im *Blocking* oder *Indexing* als Vorbereitungsschritt beim Record Linkage (Christen, 2012, S. 69). Dabei wird der Datensatz in möglichst ähnliche Blöcke eingeteilt, sodass nur noch innerhalb dieser Blöcke Record Paare zum Linken miteinander verglichen werden müssen und somit die Komplexität verringert wird. Liegen die Records nun in Form von Bloom-Filtern vor (siehe dazu Schnell et al., 2009), handelt es sich ebenfalls um einen binären Datensatz, auf welchen die in dieser Arbeit verwendeten Algorithmen angewandt werden können.

4. Simulationsaufbau

Es wird eine Vielzahl unterschiedlicher Konfigurationen des Forschungsdesigns[1] erstellt, um die verwendeten Algorithmen ausgiebig zu testen. Wie der Simulationsaufbau im Detail aussieht wird im folgenden Abschnitt beschrieben.

4.1. Faktoren des Designs und Faktorstufen

In Tabelle 4.1 findet sich eine Aufstellung der einzelnen Faktoren des Simulationsaufbau und der verwendeten Faktorstufen.[2] Die genutzten Clusteralgorithmen werden an jedem der resultierenden Datensätze angewendet und anhand der in Abschnitt 6 beschriebenen Beurteilungskriterien evaluiert. Die Daten in der jeweiligen Konfiguration werden dabei genau einmal simuliert und jeweils alle Algorithmen an diesem Datensatz angewandt. Es ergeben sich insgesamt 120 verschiedene Konfigurationen der einzelnen Faktoren.

Anzahl Cluster Es werden die Anzahl der in den Daten vorhandenen Cluster variiert. So werden Datensätze simuliert, die 2, 4 und 6 Cluster enthalten.

Anzahl Variablen Der nächste Faktor ist die Anzahl der Variablen. Es werden Datensätze gebildet, die 100 oder 1.000 Variablen enthalten. Die gebildeten Variablenblöcke bestehen demnach aus 25 oder 250 Variablen bzw. 50 oder 500 in Szenario *Einfach* (siehe dazu Paragraph *Korrelation*).

[1] Im Rahmen der Simulation wird jede Konfiguration des Simulationsaufbau lediglich einmal iteriert. Grund dafür liegt in der hohen Laufzeit der Simulation. So liegt die Laufzeit der kompletten Simulation mit dem gewählten Simulationssetup (siehe Abschnitt 4.3) bei mehreren Wochen. Zudem sind keine grundsätzlich unterschiedlichen Ergebnisse bei wiederholten Simulationsdurchläufen zu erwarten. Dennoch handelt es sich aufgrund der Tatsache, dass nur ein Simulationsdurchlauf durchgeführt wurde, um eine explorative Studie.

[2] Aus pragmatischen Gründen wurden bei den Faktoren *Prävalenz* und *Korrelationsstruktur* nur 2 Faktorstufen gewählt, um die Anzahl der Iterationen und somit die Laufzeit der Simulation insgesamt auf einem durchführbaren Niveau zu halten. Aus technischen Gründen wurde auf eine Konfiguration mit 10.000 Variablen verzichtet, da dies die Laufzeit und vor allem den Speicherbedarf der meisten Algorithmen auf ein nicht mehr praktikables Niveau gehoben hätte.

Tabelle 4.1.: Designfaktoren und Faktorstufen

Faktor	Faktorstufen
Anzahl Cluster	2, 4, 6
Anzahl Variablen	100, 1.000
Prävalenz	Niedrig: 0,005 – 0,15 Mittel: 0,001 – 0,49 (0,1 – 0,8 in Szenario „Einfach“)
Anzahl Beobachtungen	1 Mio 10 Mio 50 Mio
Korrelationsstruktur	Niedrig: 0,05 – 0,1 Mittel: 0,3 – 0,4 Ein Block hoch 0,9, Rest niedrig 0,1 (Szenario „Spezial“) Undefiniert (Szenario „Einfach“)

Anzahl Beobachtungen Natürlich ist auch von entscheidender Bedeutung, vor allem was die Dauer der Berechnungen angeht, wie gut die verschiedenen Clusteralgorithmen bei unterschiedlicher Anzahl der Beobachtungen abschneiden. Dabei wird eine niedrige, eine mittlere und eine hohe Konfiguration generiert. Aufgrund technischer Hürden (sehr hohe Laufzeit und Speicherbedarf) wird auf die Konfiguration mit 50 Millionen Beobachtungen und 1.000 Variablen verzichtet.

Korrelationsstruktur Ein weiterer Punkt des Simulationsaufbau besteht aus der Korrelationsstruktur der Daten. Diesbezüglich werden der Simulation der Daten Korrelationsmatrizen zugrunde gelegt, welche zum einen eine niedrige Korrelation der simulierten Variablen untereinander, eine mittlere und eine hohe Korrelation abbilden. Diese Korrelationen beziehen sich allerdings nicht auf alle Variablen zu jeweils allen anderen, sondern es handelt sich dabei um Variablenblöcke, innerhalb welcher sich die angestrebten Korrelationen beobachten lassen. Daten solcher Struktur lassen sich oftmals in der Marktforschung beobachten, wenn viele bzw. sehr viele Kunden analysiert und klassifiziert werden sollen (Korzeniewski, 2016, S. 296), weshalb eine solche Konfiguration bezogen auf die inhaltlichen Aspekte der Arbeit angemessen erscheinen. Zwischen Variablen unterschiedlicher Blöcke wird in allen Konfigurationen eine niedrige Korrelationsstruktur zugrunde gelegt.

Die hohe Konfiguration stellt dabei einen Spezialfall dar, indem nur ein Variablenblock bestehend aus 20 % der Variablen sehr hoch korreliert ist ($r = 0,9$) und der Rest

relativ niedrig ($r = 0,1$). Dies wird fortan als Szenario *Spezial* betitelt. Ebenfalls einen Spezialfall stellt das Szenario mit der undefinierten Korrelationsstruktur dar. Bei diesem Szenario wird die Korrelation nicht berücksichtigt und die Daten werden nur über die Kontrolle der Mittelwerte simuliert. Dies resultiert in eindeutig getrennten Clustern, da der Simulationsalgorithmus die Korrelationsstruktur nicht als Randbedingung berücksichtigen muss.[3] Dieses Szenario bietet sich damit auch an, den einfachsten aller Fälle zu simulieren, mit lediglich 2 eindeutig getrennten Clustern und 2 Variablenblöcken. Fortan wird dieses Szenario als Szenario *Einfach* betitelt. Das letzte Szenario besteht aus 4 Variablenblöcken (Szenario *4 Block*), die durch ihre Prävalenzstruktur gebildet werden. Mehr dazu im nächsten Punkt.

Prävalenz Die Variablen wiederum werden nicht nur in ihrer Anzahl variiert, sondern auch in ihrer Prävalenz der Ausprägung „1“. Dies wird über die Variation der Mittelwerte der Variablen kontrolliert. So werden Konfigurationen mit niedriger (0,005 bis 0,15) und mittlerer (0,001 bis 0,49) Prävalenz geschaffen. In Szenario *Einfach* rangiert das mittlere Design von 0,1 bis 0,8 um noch eindeutigere Cluster zu erhalten.[4] Diese Prävalenzen wiederum, werden zwischen den vorherig genannten untereinander korrelierten Variablenblöcken variiert. Die Manipulation dieser Prävalenzen zwischen den einzelnen Variablenblöcken lässt es zu, Gruppen von Fällen zu schaffen, welche sich somit in den Ausprägungen innerhalb und zwischen den Variablenblöcken ähneln bzw. unterscheiden, wodurch die Clusterstrukturen geschaffen werden können (siehe Abschnitt 4.2). Es gibt demnach innerhalb den beiden Konfigurationen mit niedriger und mittlerer Prävalenzstruktur wiederum je drei verschiedene Ausprägungen zwischen denen die Variablenblöcke variieren und sich dadurch unterscheiden. In Tabelle 4.2 sind die einzelnen Konfigurationen bezüglich der Korrelations- und Prävalenzstruktur der drei Szenarien *Einfach*, *Spezial* und *4 Block* abgebildet.

Das Niveau bei der Variante „Low“ zwischen den Konfigurationen „Niedrig“ und „Mittel“ unterscheidet sich kaum, da diese Variante in beiden Konfigurationen sehr niedrig sein sollte und in Testläufen die besten Ergebnisse erzielt hat.

[3] Die Daten in diesem Szenario werden daher auch nicht mit dem Algorithmus nach Emrich und Piedmonte (1991) (siehe Abschnitt 5.3.1) erstellt, sondern wie bereits angesprochen über die Kontrolle der Mittelwerte. Wie sich das auf die resultierenden Daten auswirkt siehe Abschnitt 5.6.

[4] Zur Begründung warum keine Werte über 0,5 außer bei Szenario *Einfach* gewählt wurden siehe Abschnitt 5.6.

Tabelle 4.2.: Korrelations- und Prävalenzstrukturen der einzelnen Szenarien

Szenario	Korrelationsstruktur	Prävalenzstruktur	
4 Block	Niedrig = 0,05 - 0,1 Mittel = 0,3 - 0,4	Niedrig	Low = 0,005 - 0,049 Mid = 0,05 - 0,099 High = 0,1 - 0,15
		Mittel	Low = 0,001 - 0,04 Mid = 0,1 - 0,2 High = 0,45 - 0,49
Spezial	Ein Block = 0,9 Rest = 0,1	Niedrig	Low = 0,005 - 0,049 Mid = 0,05 - 0,099 High = 0,1 - 0,15
		Mittel	Low = 0,001 - 0,04 Mid = 0,1 - 0,2 High = 0,45 - 0,49
Einfach	Nicht kontrolliert	Niedrig	Low = 0,005 - 0,049 Mid = 0,05 - 0,099 High = 0,1 - 0,15
		Mittel	Low = 0,1 - 0,2 Mid = 0,3 - 0,5 High = 0,7 - 0,8

Szenarien Auch wenn die verwendeten Szenarien keine Faktorstufen im eigentlichen Sinne darstellen, werden sie aufgrund der unterschiedlichen entstehenden Eigenschaften der Daten in Abschnitt 9.3 ebenfalls getrennt evaluiert. Zur Verdeutlichung worum es sich dabei handelt werden sie an dieser Stelle nochmals beschrieben. Szenario *Einfach* stellt das einfachste Szenario dar und bringt die deutlichsten Clusterstrukturen mit der geringsten Menge an Rauschen[5] hervor. Es wird daher erwartet, dass in diesem Szenario die besten Ergebnisse erzielt werden. Das Szenario wird gebildet, indem nicht der Algorithmus von Emrich und Piedmonte (1991) verwendet wird (siehe Abschnitt 5), sondern die Korrelationsstruktur ignoriert und lediglich die Mittelwerte berücksichtigt werden. Dies wird erreicht, indem die Ausprägungen der einzelnen Elemente der Vektoren des jeweiligen Variablenblocks mit einer Wahrscheinlichkeit gemäß des gewünschten Mittelwert gezogen werden. Die Daten in Szenario *Spezial* besitzen lediglich 2 Variablenblöcke. Einen kleinen mit 20 % der Variablen mit einer hohen Korrelation und der Rest mit einer niedrigen Korrelation. Szenario *4 Block* enthält 4 Variablenblöcke und wird gemäß der verwendeten

[5]Als Rauschen werden hier Beobachtungen definiert, die bei visueller Betrachtung der Daten (siehe Abschnitt 5.6) nicht eindeutig einem Cluster zugeordnet werden können.

Faktoren und deren Faktorstufen variiert. Es sei an dieser Stelle nochmals auf Tabelle 4.2 für eine Übersicht der einzelnen Szenarien verwiesen.

4.2. Clusterstruktur

Wie bereits im vorangegangenen Abschnitt beschrieben, werden die Prävalenzen zwischen den einzelnen Variablenblöcken variiert. Die gewünschte Clusterstruktur wird demnach dadurch geschaffen, dass die Prävalenzstrukturen der Variablenblöcke zwischen den einzelnen Clustern variiert werden. Dieses Vorgehen wurde bspw. in einer Simulationsstudie zur Determinierung der Clusteranzahl in binären Datensätzen von Dimitriadou et al. (2002) umgesetzt und in einer Studie zum Vergleich zur Variablenauswahl innerhalb der Clusteranalyse von Korzeniewski (2016) adaptiert. Eine beispielhafte Darstellung der Clusterstruktur findet sich in Tabelle 4.3. Wird dort exemplarisch zum Beispiel Cluster 1 und Cluster 3 betrachtet, haben die ersten 4 Variablen in Block 1 bei Cluster 1 eine hohe Prävalenz und in Cluster 3 eine niedrige. In Block 3 ist es entsprechend umgekehrt. So können die Cluster relativ deutlich voneinander unterschieden werden.

Tabelle 4.3.: Beispiel für Clusterstruktur

	Block 1				Block 2				Block 3			
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
Cluster 1	H	H	H	H	M	M	M	M	L	L	L	L
Cluster 2	M	M	M	M	H	H	H	H	L	L	L	L
Cluster 3	L	L	L	L	M	M	M	M	H	H	H	H
Cluster 4	M	M	M	M	L	L	L	L	M	M	M	M

H: Hohe Prävalenz, M: Mittlere Prävalenz, L: Niedrige Prävalenz
Tabelle angelehnt an: Dimitriadou et al. (2002, S. 138)

Um nun Datensätze mit mehr als 2 Clustern simulieren zu können, müssen mehr als 2 Variablenblöcke vorhanden sein, um die Prävalenzen entsprechend variieren zu können. Aus diesem Grund liegen den Daten immer 4 Blöcke zugrunde. Ausnahme bildet hier bei 2 Clustern das Szenario *Einfach*, in dem ebenfalls nur 2 Variablenblöcke verwendet werden, um eine möglichst einfache Clusterstruktur zu erzielen. Des Weiteren werden die Blöcke in den Faktorstufen mit bis zu 4 Clustern nur zwischen hohen und niedrigen Prävalenzen variiert, um die Blöcke eindeutiger voneinander abzugrenzen. Erst ab 6 Clustern werden auch Blöcke mit mittlerer Prävalenzstruktur erzeugt. Wie die Datensätze

dann im Detail genau aussehen wird anhand der Abbildungen 5.13 und 5.14 in Abschnitt 5.6 verdeutlicht.

4.3. Technische Details der Durchführung der Berechnungen

Sämtliche der in Abschnitt 7 vorgestellten verwendeten Clusteralgorithmen wurden im Rahmen dieser Arbeit in der statistischen Programmiersprache `R` Version 4.1.2 (R Core Team, 2021) implementiert. Der zugehörige R-Code zu den einzelnen Clusteralgorithmen findet sich in Anhang C.4. Ausnahme bildet der Algorithmus *Proximus*, dessen Kernfunktion mittels der Funktion `proximus` aus dem Paket `cba` (Buchta & Hahsler, 2019) verwendet wird und dort intern in `C` implementiert ist. Eine Aufzählung der verwendeten R-Pakete mit Versionsnummer findet sich in Tabelle 4.4.

Tabelle 4.4.: Verwendete R-Pakete und Versionsnummern

Paket	Version
aricode (Chiquet et al., 2020)	0.1.2
bigtabulate (Kane & Emerson, 2016)	1.1.5
cba (Buchta & Hahsler, 2019)	0.2-22
cluster (Maechler et al., 2021)	2.1.2
ClusterR (Mouselimis, 2021)	1.2.5
data.table (Dowle & Srinivasan, 2019)	1.14.2
doMC (Danie et al., 2022)	1.3.7
doRNG (Gaujoux, 2020)	1.8.2
dplyr (Wickham et al., 2021)	1.0.7
e1071 (Meyer et al., 2021)	1.7-9
fastdigest (Becker & Jenkins, 2015)	0.6-3
foreach (Microsoft & Weston, 2020)	1.5.1
itertools (Weston & Wickham, 2014)	0.1-3
Matrix (Bates & Maechler, 2021)	1.4-0
matrixcalc (Novomestky, 2021)	1.0-5
matrixStats (Bengtsson, 2021)	0.61.0
mvnfast (Fasiolo, 2016)	0.2.7

parallelDist (Eckert, 2018)	0.2.4
psych (Revelle, 2021)	2.1.9
RcppAlgos (Wood, 2021)	2.4.3
rdist (Blaser, 2020)	0.0.5
reshape2 (Wickham, 2007)	1.4.4
RhpcBLASctl (Nakano & Nakama, 2021)	0.21-247
rlist (Ren, 2021)	0.4.6.2
stringr (Wickham, 2019)	1.4.0

Bezüglich der Parallelisierung sind 2 verschiedene Herangehensweisen möglich. Zum einen via *Socket*, zum anderen via *Fork*. Wird die Parallelisierung via Socket durchgeführt, erstellt der Hauptprozess ein Netzwerk an Subprozessen auf den einzelnen Prozessorkernen, die über den Hauptprozess miteinander kommunizieren müssen und vom Hauptprozess mit sämtlichen Daten und Objekten versorgt werden. Wegen diesen Kommunikationskosten hat diese Variante einen Performance- und Speichernachteil, da jedes in den Subprozessen verwendetes Objekt kopiert werden muss (Lim & Tjhi, 2015, S. 108). Mittels Fork wiederum, wird die komplette R-Session so oft dupliziert, wie Kerne verwendet werden sollen. Der große Vorteil dabei ist, dass via shared memory jedes Objekt im Speicher jeder Session zur Verfügung steht ohne kopiert werden zu müssen[6] (Matloff, 2016, S. 58). Dieser Umstand macht die Parallelisierung via Fork wesentlich effizienter als via Socket, weshalb in dieser Arbeit die Parallelisierung via Fork verwendet wird. Um Seeds zu generieren, welche auch für die Verwendung bei einer Parallelisierung angemessen sind, wird die Methode von L'Ecuyer (1999) verwendet, die im Paket `doRNG` (Gaujoux, 2020) implementiert ist.

Trotz der Nutzung von Fork kommt es dennoch durch die Parallelisierung zu einem hohen Speicherbedarf. Sobald die Daten von einer Session beschrieben werden, muss eine Kopie angefertigt werden. Zudem kann `R` nicht auf Subsets einer Datenmatrix zugreifen, ohne diese physisch im Speicher zu bilden, was bei großen Datensätzen und vielen Kernen auch unter der Verwendung von Fork ggf. zu einem großen Speicherbedarf führen kann[7].

[6] Dazu sollte erwähnt werden, dass Parallelisierung via Fork nur auf Unix-basierten Systemen zur Verfügung steht. Außerdem funktioniert es nur bei Multicore-Prozessoren innerhalb eines einzelnen Computers. Ein HPC-Cluster mit vielen einzelnen Prozessoren oder ein Netzwerk von Computern kann nicht auf diese Weise betrieben werden, da sie sich nicht den gleichen physischen Arbeitsspeicher teilen.

[7] Es wurde versucht den generellen Speicherbedarf durch die Verwendung der Pakete `bit` (Oehlschlägel & Ripley, 2020), `bigmemory` (Kane et al., 2013) und `ff` (Adler et al., 2021) zu reduzieren. Jedoch

Aus diesem Grund werden in dieser Arbeit nur 10 Prozessorkerne parallel verwendet, da sonst der verfügbare Speicher überschritten würde. Dies ist jedoch erst bei einer Datenmenge ab 10 Millionen Beobachtungen mit 1.000 Variablen zu beobachten. Dennoch werden auch bei kleineren Datensätzen nur 10 Kerne verwendet, um die Ergebnisse vergleichbar zu halten. Für die Berechnungen, die die Verwendung von linearer Algebra benötigen, werden in der Regel optimierte BLAS (Basic Linear Algebra Subprograms) und LAPACK (Linear Algebra PACKage) Implementationen verwendet. In dieser Arbeit wird die Implementation `OpenBLAS` verwendet.

Die Simulation wird auf einem Server mit einem AMD Epyc 7702P 64-Core CPU und 1 TB Arbeitsspeicher durchgeführt. Das Betriebssystem ist Ubuntu 21.10.

unterstützt `bit` keine Matrizen und konnte daher nicht verwendet werden. `bigmemory` unterstützt nicht das Subsetting von `big.matrix`-Objekten, wenn die Zeilen und/oder Spalten nicht aufeinanderfolgen. Um dies zu tun, müssen wieder reguläre `R` Matrizen gebildet werden, was die Nutzung von `bigmemory` ausschließt. Das Paket `ff` bringt ähnliche Restriktionen mit sich.

5. Simulationsmethoden für korrelierte binäre Daten

Um die gewählten Clusteralgorithmen angemessen überprüfen zu können, bedarf es einer Datenbasis, die eine Struktur aufweist, welche im Vorhinein bekannt ist. Ebenfalls muss diese Datenbasis gezielt manipuliert werden können, um die in Abschnitt 4 beschriebenen Szenarien umsetzen zu können. Diese Anforderungen sind von empirischen Daten schwer zu erfüllen, weshalb auf das Mittel der Simulation zurückgegriffen wird.

Bei der Art und Weise, wie diese Datenbasis simuliert werden kann, sind verschiedene Vorgehensweisen denkbar. Lalonde (2017, S. 87) nennt vier unterschiedliche Methoden, um binäre Daten zu erzeugen, die sowohl eine vorher spezifizierte Prävalenz- als auch Korrelationsstruktur berücksichtigen:

1. Korrelierte binäre Daten, welche direkt mittels einer voll spezifizierten multivariaten Verteilung produziert werden,

2. Mittels Mischverteilungen von diskreten oder kontinuierlichen Variablen,

3. Durch die Dichotomisierung von kontinuierlichen- oder Zählvariablen,

4. Via einer konditionalen Wahrscheinlichkeitsverteilung.

5.1. Generierung mittels einer voll spezifizierten multivariaten Verteilung

Die Generierung korrelierter binärer Daten mittels einer voll spezifizierten multivariaten Verteilung stellt den ältesten Ansatz dar und geht auf Devroye (1986) zurück.

5.1.1. Ziehen aus einer voll spezifizierten multivariaten Verteilung

Devroye (1986) argumentiert, dass gegeben einer voll spezifizierten Wahrscheinlichkeitsdichtefunktion (kurz pdf[1]) für den binären Zufallsvektor $\mathbf{Y}^T = (Y_1, \ldots, Y_N)$, die marginalen Wahrscheinlichkeiten für jede mögliche Kombination von binären Realisationen berechnet werden kann. Die Wahrscheinlichkeiten aller möglichen Realisationen werden zunächst geordnet und kumuliert. Anschließend wird eine Zufallszahl zwischen 0 und 1 gezogen. Je nachdem wo diese Zufallszahl im Intervall der geordneten kumulierten Wahrscheinlichkeiten verordnet werden kann, wird die entsprechende Realisation des binären Vektors extrahiert und bildet eine Beobachtung.

Eine Möglichkeit eine voll spezifizierte pdf für N korrelierte binäre Variablen zu identifizieren, um den vorangegangenen Algorithmus zu nutzen, besteht in der Methode von Bahadur (1961), welcher diese für autoregressive Zeitreihen entwickelt hat, prinzipiell aber auch für Beobachtungen mit N binären Variablen genutzt werden kann (Lalonde, 2017, S. 89). Seien y_i Realisationen binärer Realisationen und $\pi = P(Y_i = 1)$ der konstante Erwartungswert der Variablen mit der Autokorrelation ρ_{ij} (wie bereits genannt, wurde die Methode für Zeitreihen entwickelt) zwischen Variablen Y_i und Y_j, ergibt sich die gemeinsame Wahrscheinlichkeitsdichtefunktion als:

$$f(y_1, \ldots, y_N) = 1 + \frac{\sum\limits_{i \neq j} \rho_{ij} (-1)^{(y_i + y_j)} \pi^{(2 - y_i - y_j)} (1 - \pi)^{(y_i + y_j)}}{\pi(1 - \pi)}. \tag{5.1}$$

5.1.2. Erstellung von Wahrscheinlichkeiten ohne voll spezifizierte multivariate Verteilung

Einige Methoden nutzen den Algorithmus von Devroye (1986) aus Abschnitt 5.1.1 ohne jedoch auf eine analytisch hergeleitete pdf zurückzugreifen. Die benötigten Wahrscheinlichkeitsverteilungen werden dabei zum einen unter Nutzung der archimedischen Copula (Lee, 1993) generiert, zum anderen durch das Berechnen eines log-linearen Modell auf eine 2^N Kreuztabelle unter Nutzung des Iterative Proportional Fitting Algorithmus (Gange, 1995).

Die Methode von Lee (1993) beruht darauf, mithilfe der archimedischen Copula (Genest & MacKay, 1986) und gegebener, innerhalb einer Beobachtung konstanter Korrelation ρ

[1] Vom englischen *probability density function.*

zwischen den binären Variablen $Y_1, \ldots, Y_N$, die Wahrscheinlichkeitsverteilung P_0, P_{2^N-1} aller möglichen Kombinationen von binären Ausprägungen zu ermitteln. Dafür müssen neben der gewünschten (konstanten) (Auto-)Korrelation ρ die marginalen Wahrscheinlichkeiten $\pi_1, \ldots, \pi_N$ gegeben sein. Um nun an die benötigten Wahrscheinlichkeiten P_0, P_{2^N-1} zu kommen, wird zur Nutzung der archimedischen Copula der Parameter Ψ benötigt, welcher Lalonde (2017, S. 90) zufolge durch die Werte von π und ρ berechnet werden kann und Werte zwischen 0 und 1 annimmt, wobei ein Wert nahe 1 Unabhängigkeit bedeutet. Nachdem nun mithilfe der archimedischen Copula die Wahrscheinlichkeiten P_0, P_{2^N-1} aller möglichen Kombinationen von binären Ausprägungen ermittelt wurden, können wie von Devroye (1986) vorgeschlagen, diese Wahrscheinlichkeiten sortiert und mit einer Zufallszahl verglichen werden. Je nachdem, wo im Intervall der Wahrscheinlichkeiten diese Zufallszahl verortet ist, wird die entsprechende Sequenz von binären Ausprägungen gewählt, um eine Beobachtung zu erzeugen.

Die Methode von Gange (1995) setzt neben der Spezifizierung der marginalen Wahrscheinlichkeiten π_i auch die Spezifizierung der gemeinsamen Wahrscheinlichkeiten $\pi_{i,j} = P(Y_i = y_i, Y_j = y_j)$ und gemeinsamen Wahrscheinlichkeiten höherer Ordnung $\pi_{i_{1,\ldots,i_k}} = P(Y_{i_1} = y_{i_1}, \ldots, Y_{i_k} = y_{i_k})$ von Rang k. Die benötigte joint probability density function wird nun durch das Gleichsetzten mit dem Fitten eines log-linearen Modells auf eine 2^N Kreuztabelle inklusive Interaktionsterme von Rang k erzeugt. Dieses Berechnen des log-linearen Modells wird mithilfe des Iterative Proportional Fitting Algorithmus (Deming & Stephan, 1940) vollzogen. Die so gewonnenen Wahrscheinlichkeiten können wiederum wie von Devroye (1986) vorgeschlagen genutzt werden, um die binären Ausprägungen einer Beobachtung zu erstellen. Die vorgeschlagene Methode von Gange (1995) ist allerdings sehr rechenintensiv (Spezifizierung von gemeinsamen Wahrscheinlichkeiten und gemeinsamen Wahrscheinlichkeiten höherer Ordnung; IPF) und aufwendig, weshalb sie für die Erzeugung hochdimensionaler Daten eher ungeeignet erscheint.

5.2. Mittels Mischverteilungen von diskreten oder kontinuierlichen Variablen

Folgend werden Methoden aufgeführt, welche auf Mischverteilungen von diskreten oder kontinuierlichen Variablen beruhen.

5.2.1. Mischverteilungen von diskreten Variablen

Kanter (1975) schlägt einen Ansatz vor, welcher eine autoregressive Modellierungsstrategie nutzt, um die binären Variablen Y_i mit konstantem Erwartungswert π zu simulieren. Somit werden die binären Werte Y_i analog zu Zeitreihen-Analyse verstanden. Um die Y_i nun zu erzeugen, wird folgendes Modell verwendet:

$$Y_i = U_i \left[Y_{i-1} \oplus W_i \right] + (1 - U_i) W_i \tag{5.2}$$

$\oplus$: modulo-2 Addition.
U_i: Binär mit Erwartungswert π_U.
W_i: Binär mit Erwartungswert $\pi_W = (\pi (1 - \pi_U)) / (1 - 2\pi\pi_U)$.

Es wird davon ausgegangen, dass Y_{i-1}, U_i und W_i unabhängig sind. Dies führt dazu, dass alle Y_i den Erwartungswert π erfüllen und die (Auto-)Korrelation zwischen den simulierten Variablen folgende Form hat:

$$\text{Corr}\left(Y_i, Y_j\right) = \left(\frac{\pi_U (1 - 2\pi)}{1 - 2\pi\pi_U} \right)^{|i-j|}, \tag{5.3}$$

wobei $\pi_U \in (0, \min((1 - \pi)/\pi, 1))$ gegeben sein muss (Farrell & Sutradhar, 2006, S. 355).

Lunn und Davies (1998) schlagen ebenfalls eine Methode vor, welche auf einer Mischverteilungen von binären Variablen beruht. Vorteil dieser Methode ist, dass nicht jede Beobachtung nacheinander simuliert werden muss, sondern die Daten in einem Schritt erzeugt werden können (Lalonde, 2017, S. 92). Die Erstellung der binären Werte Y_{ij} hat ebenfalls, wie auch die vorangegangene Methode, die Bedingung, dass ein konstanter Erwartungswert für eine Beobachtung π_i angenommen werden muss. Ebenfalls wird die (Auto-)Korrelation ρ_i zwischen den Werten innerhalb einer Beobachtung als konstant angenommen. Die Formel zum Erstellen der Werte hat die Form:

$$Y_{ij} = (1 - U_{ij}) W_{ij} + U_{ij} Z_i \,, \tag{5.4}$$

wobei U_{ij} binär ist mit Erwartungswert $\pi_U = \sqrt{\rho_i}$ und W_{ij} binär ist mit Erwartungswert π_i und Z_i binär ist mit Erwartungswert π_i.

Daraus folgt, dass $\mathrm{E}[Y_{ij}] = \pi_i$ und $\mathrm{Corr}(Y_{ij}, Y_{ik}) = \rho_i$ (Lunn & Davies, 1998, S. 488). Durch den Austausch lediglich eines Terms innerhalb von Formel 5.4 ist die Methode nicht mehr durch konstante (Auto-)Korrelationen innerhalb einer Beobachtung begrenzt, sondern erlaubt auch die Spezifizierung unterschiedlicher Korrelationswerte innerhalb einer Beobachtung (Lunn & Davies, 1998, S. 488):

$$Y_{ij} = (1 - U_{ij})\, W_{ij} + U_{ij} Y_{i,j-1}\,. \tag{5.5}$$

Um auch „M-dependent correlation structures“ (Lunn & Davies, 1998, S. 489) zuzulassen, muss in Formel 5.4 wiederum nur ein Term getauscht werden:

$$Y_{ij} = (1 - U_{ij})\, W_{ij} + U_{ij} W_{i,j-1}\,. \tag{5.6}$$

5.3. Dichotomisierung

In diesem Abschnitt werden Methoden vorgestellt, welche darauf basieren, die erwünschten binären Variablen nicht direkt zu simulieren, sondern über den Umweg der Dichotomisierung von Zufallsvariablen.

5.3.1. Dichotomisierung von normalverteilten Variablen

Die in dieser Arbeit verwendete Methode wird zu Punkt 3 der vorangegangenen Aufzählung gezählt. Sie wurde von Emrich und Piedmonte (1991) entwickelt und stellt eine der verbreitetsten Formen der Simulation von korrelierten binären Daten dar (Lalonde, 2017, S. 96) und wird von Haynes et al. (2015, S. 511) als „gold-standard approach“ beschrieben, weshalb sie zur Nutzung in dieser Arbeit gewählt wurde.

Vorgegangen wird dabei wie folgt: Sei es das Ziel binäre Variablen Y_i zu erstellen, welche den erwünschten Mittelwert π_i und eine gegebene paarweise Korrelation $Corr(Y_i, Y_j) = \rho_{ij}$ aufweisen, wird zunächst die folgende Gleichung nach der paarweisen Korrelation δ_{ij}

von normalverteilten Variablen aufgelöst, wobei die bivariate kumulierte (Normal-)-Verteilungsfunktion Φ genutzt wird:

$$\Phi\left(z\left(\pi_i\right), z\left(\pi_j\right), \delta_{ij}\right) = \rho_{ij}\sqrt{\pi_i\left(1-\pi_i\right)\pi_j\left(1-\pi_j\right)} + \pi_i\pi_j\,, \tag{5.7}$$

wobei $z(\pi)$ das π-te Perzentil der Standardnormalverteilung darstellt. Anschließend werden die Werte $r_{ij} \forall i \neq j$ in die Korrelationsmatrix R geschrieben, um einen $k \times 1$ Vektor $z = (z_p, \ldots, z_k)$ mit Mittelwert 0 und Korrelation R zu erstellen, welcher danach dichotomisiert wird. Dies geschieht in dem $Y_i = 1$ gesetzt wird, sollte $z_i \geq z(\pi_i)$ sein. Sollte dem nicht der Fall sein, erhält Y_i entsprechend den Wert 0. Um nun auf die gewünschte Fallzahl zu kommen, werden n solcher Vektoren simuliert und entsprechend dichotomisiert (Wang & Sabo, 2015, S. 1).

Es gilt allerdings zu beachten, dass die Korrelationsmatrix, welche in den Algorithmus eingegeben wird und deren Werte von den simulierten Daten repliziert werden soll, gewissen Restriktionen unterliegt. Abhängig von den Mittelwerten π_i und π_j gibt es untere und obere Schwellen für ρ_{ij}, welche nicht unter- bzw. überschritten werden dürfen (Prentice, 1988, S. 1046). Die obere Schwelle hat die Form:

$$\max\left\{\sqrt{-\left(\pi_i\pi_j/\left(1-\pi_i\right)\left(1-\pi_j\right)\right)}\,,\sqrt{-\left(\left(1-\pi_i\right)\left(1-\pi_j\right)/\pi_i\pi_j\right)}\right\}\,. \tag{5.8}$$

Analog dazu nimmt die untere Schwelle folgende Form an:

$$\min\left\{\sqrt{\left(\pi_i\left(1-\pi_j\right)/\pi_j\left(1-\pi_i\right)\right)}, \sqrt{\left(\pi_j\left(1-\pi_i\right)/\pi_i\left(1-\pi_j\right)\right)}\right\}\,. \tag{5.9}$$

Das Einhalten dieser Schwellen von ρ_{ij} ist vonnöten, um zu garantieren, „that the joint mass function for the outcomes be nonnegative for all outcomes“ (Emrich & Piedmonte, 1991, S. 303).

Dieser ganze Vorgang kann nun so oft wiederholt werden, wie Cluster gewünscht sind (Wang & Sabo, 2015, S. 1). Vorteil der Methode ist es, dass die Mittelwerte und Korrelationsstruktur im Vorhinein so gewählt werden können, wie gewünscht. Durch die Manipulation der Mittelwerte können den unterschiedlichen Prävalenzstrukturen der

Variablen in den unterschiedlichen Clustern Rechnung getragen werden und auch die Korrelationsstruktur kann dem geplanten Szenario angepasst werden. Um die Größe der verschiedenen Cluster zu variieren, kann die Anzahl der simulierten Vektoren z im Simulationstemplate auf den gewünschten Wert gesetzt werden.

5.3.2. Effizenter R-Code für die Umsetzung der Methode von Emrich und Piedmonte (1991)

Das R-Paket `MultiOrd` von Amatya et al. (2019) stellt die Funktion `generate.binary` zur Verfügung, welche die Methode von Emrich und Piedmonte (1991) umsetzt. Leider ist diese Funktion nicht sehr effizient geschrieben, weshalb für die Erstellung großer Datensätze viel Zeit benötigt wird und daher ungeeignet ist.

In diesem Abschnitt wird der Originalcode der Funktion `generate.binary` aus dem Paket `MultiOrd` einer im Rahmen dieser Arbeit erstellten effizienteren Variante des Codes gegenübergestellt.

5.3.3. Originalcode der Funktion `generate.binary`

Die Funktion `generate.binary` hat folgende Form:

```
function (nObs, prop.vec.bin, corr.mat)
{
  validation.CorrMat(prop.vec.bin, corr.mat)
  sigma_star = compute.sigma.star(prop.vec.bin, corr.mat)
  d = ncol(sigma_star)
  data = rmvnorm(nObs, mean = rep(0, d), sigma = sigma_star)
  p = prop.vec.bin
  q = 1 - p
  for (i in 1:nObs) {
    for (j in 1:d) {
      if (data[i, j] <= qnorm(1 - p[j]))
        data[i, j] = 0
      else data[i, j] = 1
    }
  }
  return(data)
```

```
}
```

Listing 5.1: Funktion `generate.binary` aus dem Paket `MultiOrd`

Zunächst wird mit der Funktion `validation.CorrMat` die eingegangene Korrelationsmatrix darauf überprüft, ob sie positiv definit und symmetrisch ist (was Korrelationsmatrizen ohnehin sein müssen) und ob der angegebene Mittelwertvektor die gleiche Anzahl an Elementen enthält wie die Korrelationsmatrix Spalten bzw. Zeilen aufweist. Die Hauptfunktion besteht allerdings darin, die Korrelationsmatrix auf Überschreitungen der durch die Mittelwerte festgelegten Schwellen zu prüfen (siehe dazu Emrich & Piedmonte, 1991, S. 303). Bei Verstoß bricht der Algorithmus ab.

```
function (prop.vec.bin, CorrMat)
{
  if (is.positive.definite(CorrMat) == FALSE) {
    stop("Specified correlation matrix is not positive definite! \n")
  }
  if (isSymmetric(CorrMat) == FALSE) {
    stop("Specified correlation matrix is not symmetric! \n")
  }
  if (length(prop.vec.bin) != ncol(CorrMat)) {
    stop("Dimensions of binary probability vector and correlation
     matrix
          do not match")
  }
  d = length(prop.vec.bin)
  sigma = CorrMat
  p = prop.vec.bin
  q = 1 - p
  no.bin = d
  L_sigma = diag(d)
  U_sigma = diag(d)
  for (i in 1:no.bin) {
    for (j in 1:no.bin) {
      if (i != j)
        L_sigma[i, j] = L_sigma[j, i] = max(-sqrt((p[i] *
            p[j])/(q[i] * q[j])), -sqrt((q[i] * q[j])/(p[i] * p[j])))
      if (i != j)
        U_sigma[i, j] = U_sigma[j, i] = min(sqrt((p[i] *
            q[j])/(q[i] * p[j])), sqrt((q[i] * p[j])/(p[i] * q[j])))
    }
  }
  valid.state = TRUE
```

```
  for (i in 1:d) {
    for (j in 1:d) {
      if (j >= i) {
        if (sigma[i, j] < L_sigma[i, j] | sigma[i, j] >
            U_sigma[i, j]) {
          stop("Range violation occurred in the binary data
               generation routine! \n \n  Corr[",
               i, ",", j, "] must be between ",
               round(L_sigma[i, j], 3), " and ", round(U_sigma[i,
                j], 3), "\n This algorithm cannot generate ordinal
               data with specified correlations")
          valid.state = FALSE
        }
      }
    }
  }
  if (valid.state == FALSE)
    stop("All correlations must be in feasible range!")
}
```

Listing 5.2: Funktion `validation.CorrMat`

Durch die Funktion `compute.sigma.star` werden die Eingangskorrelationen mittels der Funktion `phi2tetra` aus dem Paket `psych` (Revelle, 2021) in tetrachorische Korrelationen konvertiert. Sollte die resultierende Matrix nicht positiv definit sein, wird mit `nearPD` die nächste positiv definite Matrix gesucht.

```
function (prop.vec.bin, corr.mat) {
  no.bin = length(prop.vec.bin)
  sigma = corr.mat
  p = prop.vec.bin
  q = 1 - p
  sigmaBB = diag(no.bin)
  for (i in 1:no.bin) {
    for (j in 1:no.bin) {
      if (i != j) {
        sigmaBB[i, j] = phi2tetra(sigma[i, j], c(p[i],
                                                 p[j]))
      }
    }
  }
  if (!is.positive.definite(sigmaBB)) {
    warning("Tetrachoric correlation matrix is not positive definite)")
```

```
17      sigmaBB = as.matrix(nearPD(sigmaBB, corr = TRUE, keepDiag = TRUE)$
        mat)
18    }
19    sigmaBB = (sigmaBB + t(sigmaBB))/2
20    return(sigmaBB)
21  }
```

Listing 5.3: Funktion `compute.sigma.star`

Anschließend werden mit der Funktion `rmvnorm` aus dem Paket `mvtnorm` (Genz et al., 2020) und der aus dem vorangegangenen Schritt resultierenden Korrelationsmatrix so viele Observationen erzeugt, wie gewünscht. Abschließend werden die Daten dichotomisiert und ausgegeben.

Nachteile dieses Codes ist vor allem die Verwendung von `for`-loops innerhalb der Funktion `compute.sigma.star` und bei der Dichotomisierung der Daten am Ende der Funktion `generate.binary`. Viele Iterationen innerhalb von `for`-loops sind in `R` generell ineffizient. Darüber hinaus ist die Funktion `rmvnorm`, mit welchem aus der multivariaten Normalverteilung gezogen wird, für große Datenmengen sehr langsam.

5.3.4. Effizientere Version des Codes

Die effizientere im Rahmen dieser Arbeit erstellte Version der Funktion folgt im wesentlichen der Vorlage von Amatya et al. (2019) und trägt den Namen `corrbinary`. Zunächst werden die Einträge der eingegebenen Korrelationsmatrix geprüft, ob eine Verletzung des zulässigen Wertebereichs vorliegt. Das Argument `sigma_correction` an die Funktion `corrbinary` gibt, falls gewünscht, die Möglichkeit eine Korrektur der Einträge vorzunehmen, welche die Schwellen überschreiten.

Wird `sigma_correction = TRUE` an `corrbinary` übergeben, werden die Werte der Korrelationsmatrix, welche die zulässige Schwelle nach oben überschreiten, auf den maximal zulässigen Wert gesetzt. Bei der Unterschreitung der unteren zulässigen Schwelle wird analog vorgegangen. Dies kann dazu führen, dass die vorgegebenen Korrelationen nicht mehr eingehalten werden. Sollten die Überschreitungen der Schwellenwerte jedoch nicht zu stark sein, kann die angestrebte Korrelationsstruktur dennoch approximiert werden.

Sollte dieses Verhalten nicht gewünscht sein, kann `sigma_correction = FALSE` übergeben werden. In diesem Fall bricht der Algorithmus mit einer Fehlermeldung ab. Zusätzlich wird ein `data.frame` ausgegeben, welcher drei Spalten enthält: `row` die Zeilenposition des Wertes welcher zu hoch bzw. zu niedrig ist, `column` die Spaltenposition und `lower.bound` bzw. `upper.bound`, welche die zulässige Schwelle beinhalten. Auf die Überprüfung der in die Funktion eingegangene Korrelationsmatrix auf Symmetrie, positive Definitheit und Dimensionalität wird verzichtet und als gegeben betrachtet.

Das Argument `cores` legt fest, wie viele Prozessorkerne in den parallelisierten Teilen der Funktion verwendet werden sollen.

```
corrbinary <- function(n, means, sigma, sigma_correction = TRUE, cores)
    {

if (sigma_correction) {

  lows <- matrix(nrow = nvars, ncol = nvars)
  highs <- matrix(nrow = nvars, ncol = nvars)
  p <- ms
  q <- 1 - p
  for (i in 1:length(p)) {
    for (j in 1:length(p)) {
      if (i == j) {
        lows[i, j] <- 1
        highs[i, j] <- 1
      } else {
        lows[i, j] <- max(-sqrt((p[i] * p[j])/(q[i] * q[j])),
                          -sqrt((q[i] * q[j])/(p[i] * p[j])))
        highs[i, j] <- min(sqrt((p[i] * q[j])/(q[i] * p[j])),
                           sqrt((q[i] * p[j])/(p[i] * q[j])))
        sigma[i, j] <- ifelse(sigma[i, j] < lows[i, j], lows[i, j],
                              ifelse(sigma[i, j] > highs[i, j],
                                     highs[i, j], sigma[i, j]))
      }
    }
  }
} else {
  toolowstop <- 0
  toohighstop <- 0
  toolow <- data.table()
  toohigh <- data.table()
```

```
30
31  lows <- matrix(nrow = nvars, ncol = nvars)
32  highs <- matrix(nrow = nvars, ncol = nvars)
33  p <- ms
34  q <- 1 - p
35  for (i in 1:length(p)) {
36    for (j in 1:length(p)) {
37      if (i == j) {
38        lows[i, j] <- 1
39        highs[i, j] <- 1
40      } else {
41        lows[i, j] <- max(-sqrt((p[i] * p[j])/(q[i] * q[j])),
42                          -sqrt((q[i] * q[j])/(p[i] * p[j])))
43        highs[i, j] <- min(sqrt((p[i] * q[j])/(q[i] * p[j])),
44                           sqrt((q[i] * p[j])/(p[i] * q[j])))
45        if (sigma[i, j] < lows[i, j]) {
46          toolow <- rbindlist(list(toolow,
47                                   data.table(row = i,
48                                              column = j,
49                                              lower.bound =
50                                                lows[i, j])))
51          toolowstop <- 1
52        }
53        if (sigma[i, j] > highs[i, j])  {
54          toohigh <- rbindlist(list(toohigh,
55                                    data.table(row = i,
56                                               column = j,
57                                               upper.bound =
58                                                 highs[i, j])))
59          toohighstop <- 1
60        }
61      }
62    }
63  }
64  if (toolowstop == 1 & toohighstop == 0) {
65    setDF(toolow)
66    print(toolow)
67    stop("There were correlations outside the acceptable
68         range. Check output for positions and bounds that
69         were exceeded.")
70  }
71  if (toohighstop == 1 & toolowstop == 0) {
```

```
72      setDF(toohigh)
73      print(toohigh)
74      stop("There were correlations outside the acceptable
75           range. Check output for positions and bounds that
76           were exceeded.")
77    }
78    if (toohighstop == 1 & toolowstop == 1) {
79      setDF(toohigh)
80      setDF(toolow)
81      print(toohigh)
82      print(toolow)
83      stop("There were correlations outside the acceptable range.
84           Check output for positions and bounds that were exceeded.")
85    }
86  }}
```

Listing 5.4: Funktion `corrbinary`

Sollte durch das Setzen der nicht zulässigen Werte auf die jeweiligen Schwellenwerte die Matrix nicht mehr positiv semi-definit[2] sein, wird die nächste positiv semi-definite Matrix gesucht. Erreicht wird dies durch die Funktion `nearPSD`, was wesentlich effizienter ist, als die nächste positiv definite Matrix mit der Funktion `nearPD` aus dem Paket `Matrix` (Bates & Maechler, 2021) zu suchen. Die Methode, auf der die Funktion `nearPSD` basiert, ist bei Rebonato und Jäckel (1999, S. 7–9) beschrieben und wurde DomPazz (2011) entnommen.

```
1
2  nearPSD <- function(c) {
3
4    n <- dim(c)[1]
5    e <- eigen(c, sym = TRUE)
6    val <- e$values * (e$values > 0)
7    vec <- e$vectors
8    T <- sqrt(diag(as.vector(1/(vec^2 %*% val)), n, n))
9    B <- T %*% vec %*% diag(as.vector(sqrt(val)), n, n)
10   out <- B %*% t(B)
11
12   return(out)
13 }
14
15
```

[2]Eine Korrelationsmatrix muss nicht zwingend positiv definit sein. Es reicht wenn sie positiv semi-definit ist.

```
16 if (!is.positive.semi.definite(sigma)) {
17     sigma <- nearPSD(sigma)
18 }
```

Listing 5.5: Funktion `nearPSD`

Analog zum Vorgehen in der Funktion `generate.binary` werden als nächstes tetrachorische Korrelationen gebildet. Dies wird in der Funktion `compute.sigma.star` innerhalb der Funktion `generate.binary` sehr ineffizient durch `for`-loops erreicht. In der effizienteren Variante `sigma.star_fun` bleibt das Vorgehen im Grunde gleich, jedoch wurde der verschachtelte `for`-loop mit einem verschachtelten `foreach`-loop ersetzt. Der äußere Loop wird parallelisiert ausgeführt.

```
1  sigma.star_fun <- function(ms, sigma, cores) {
2    no.bin <- length(ms)
3    p <- ms
4    result <- foreach(i = 1:no.bin,
5                      .packages = c("psych", "foreach"),
6                      .combine = rbind) %dopar%  {
7      foreach(j = 1:no.bin, .combine = c) %do% {
8        phi2tetra(sigma[i, j], c(p[i], p[j]))
9      }
10   }
11
12   if (!all(diag(result) == 1)) {
13     diag(result) <- 1
14   }
15   if (!is.symmetric.matrix(result)) {
16     result <- as.matrix(forceSymmetric(result, uplo = "L"))
17   }
18   if (!is.positive.semi.definite(result)) {
19     result <- nearPSD(result)
20   }
21
22   result <- (result + t(result)) / 2
23   return(result)
24 }
25 sigma_star <- sigma.star_fun(ms, sigma, cores)
```

Listing 5.6: Funktion `sigma.star_fun`

Nachdem sichergestellt wurde, dass die resultierende Matrix positiv definit ist[3], werden die Beobachtungen aus der multivariaten Normalverteilung gezogen. Dies geschieht mittels der Funktion `rmvn` aus dem Paket `mvnfast` (Fasiolo, 2016), welches in `C` geschrieben, via `OpenMP` parallelisiert und somit weitaus effizienter ist als die in `generate.binary` verwendete Funktion `rmvnorm` aus dem Paket `mvtnorm` (Genz et al., 2020). Das Resultat wird in einen `data.table` (Dowle & Srinivasan, 2019) geschrieben.

```
if (!is.positive.semi.definite(sigma_star)) {
  sigma_star <- nearPSD(sigma_star)
}

if (!is.positive.definite(sigma_star)) {
  sigma_star <- as.matrix(nearPD(sigma_star,
                                 corr = TRUE, keepDiag = TRUE)$mat)
}

data <- as.data.table(rmvn(n, mu = rep(0, nvars),
                           sigma = sigma_star, ncores = cores))
```

Listing 5.7: Sicherstellen, dass `sigma_star` positiv definit ist und anschließendes Ziehen aus multivariater Normalverteilung

Via `data.table`-Syntax werden die Daten dichotomisiert. Zum einen wird an dieser Stelle von Vektorisierung Gebrauch gemacht, zum anderen die Vorteile von `data.table` genutzt. Beides führt dazu, dass dieser Abschnitt wesentlich schneller abgeschlossen wird, als bei den verschachtelten `for`-loops in `generate.binary`.

Abschließend wird der Datentyp noch zu `integer` geändert, um den Speicherbedarf des auszugebenen `data.frame` zu verringern.

```
rm(sigma_star)
rm(sigma)
gc()

for (i in 1:nvars) {
  data[get(paste0("V", i)) < qnorm(1 - p[i]), c(paste0("V", i)) := 0L]
  data[get(paste0("V", i)) > qnorm(1 - p[i]), c(paste0("V", i)) := 1L]
}

gc()
```

[3]Die Funktion `rmvn` prüft intern auf positive Definitheit, weshalb positive semi-Definitheit in diesem Fall leider nicht ausreicht.

```
11
12 for (j in 1:nvars) {
13   set(data, j = j, value = as.integer(data[[j]]))
14 }
15
16 setDF(data)
17 gc()
18 return(data)
19
20 }
```

Listing 5.8: Dichotomisieren der Daten und Ändern des Datentyps zu `integer`

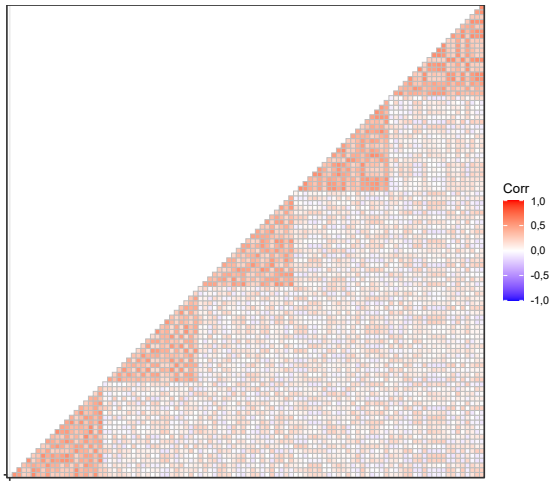


Abbildung 5.1.: Eingangskorrelationen

5.3.5. Benchmark der Funktionen `corrbinary` und `generate.binary`

Um zu demonstrieren, dass `corrbinary` wesentlich effizienter als `generate.binary` ist, folgt ein Benchmark beider Funktionen.

In Abbildung 5.1 ist die Korrelationsmatrix `sigma` abgebildet, welche als Funktionsargument eingegeben wird und deren Korrelationen vom simulierten Datensatz repliziert werden sollen.

Es sollen 100 Variablen generiert werden. Die Mittelwerte befinden sich zwischen 0.15 und 0.35.

```
1 ms <- runif(100, min = 0.15, max = 0.35)
```

Listing 5.9: Ziehen der Mittelwerte

Es werden 1000000 Beobachtungen generiert.

```
1 system.time(binarydata <- corrbinary(n = 1e6,
2                                      means = ms,
3                                      sigma = sigma,
4                                      sigma_correction = TRUE,
5                                      cores = 12))
```

Listing 5.10: Benchmark von `corrbinary`

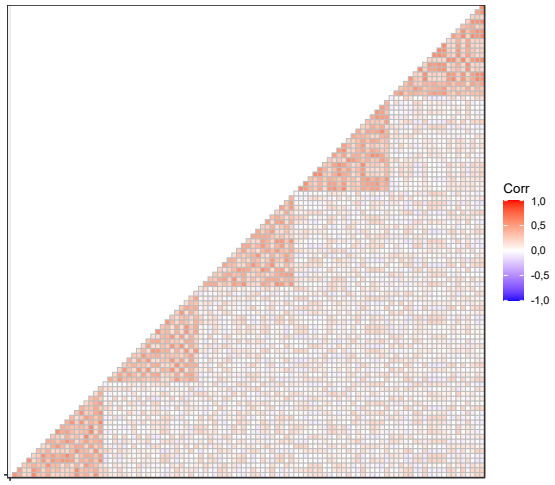


Abbildung 5.2.: Korrelationsmatrix der simulierten Daten mit corrbinary

`corrbinary` benötigte 6,28 Sekunden.

In Abbildung 5.2 findet sich die Korrelationsmatrix der simulierten Daten mit der Funktion `corrbinary`.

```
1 mean(colMeans(binarydata) - ms)
```

Listing 5.11: Abweichung der Mittelwerte

Die vorgegebenen Mittelwerte `ms` und die Mittelwerte der Spalten der simulierten Daten weichen im Mittel um 0 Einheiten voneinander ab.

```
1 system.time(binarydata2 <- generate.binary(nObs = 1e6,
2                                            prop.vec.bin = ms,
3                                            corr.mat = sigma))
```

Listing 5.12: Benchmark von `generate.binary`

`generate.binary` benötigte 93,55 Sekunden.

In Abbildung 5.3 findet sich die Korrelationsmatrix der simulierten Daten mit der Funktion `generate.binary`.

```
1 mean(colMeans(binarydata2) - ms)
```

Listing 5.13: Abweichung der Mittelwerte

Die vorgegebenen Mittelwerte `ms` und die Mittelwerte der Spalten der simulierten Daten weichen im Mittel um 0 Einheiten voneinander ab.

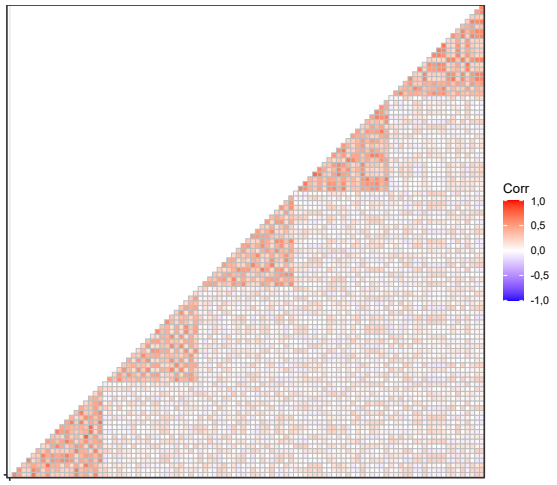


Abbildung 5.3.: Korrelationsmatrix der simulierten Daten mit generate.binary

5.3.6. Dichotomisieren von Zufallsvariablen

Auf Headrick (2002) geht eine Methode zurück, welche auf Grundlage von gleichverteilten Zufallszahlen zwischen 0 und 1 und im Vorhinein festgelegten Schwellenwerten die gezogenen Zufallszahlen dichotomisiert. Dabei wird Beobachtung für Beobachtung vorgegangen, wobei das Outcome der vorherigen Beobachtung als Ausgangspunkt für die nächste Beobachtung dient. Daher handelt es sich um ein iteratives Vorgehen. Im ersten Schritt wird die erste Beobachtung simuliert und die Schwellenwerte bestehen aus den angestrebten Mittelwerten für die Variablen. Diese Schwellenwerte verändern sich bei den darauffolgenden Beobachtungen und müssen durch die Lösung eines nicht linearen Gleichungssystems, in welches auch die gewünschte Korrelationsstruktur eingeht, berechnet werden (Headrick, 2002, S. 196–197). Mit steigender Dimensionalität können diese Gleichungssysteme sehr komplex werden (Lalonde, 2017, S. 98), weshalb sich die Methode nicht für die Simulation hochdimensionaler Datensätze eignet. Eine ähnlich funktionierende Methode, die im Kern auf der Dichotomisierung von Variablen aus einer Poissonverteilung basiert, findet sich bei Park et al. (1996).

5.4. Generierung mittels einer konditionalen Wahrscheinlichkeitsverteilung

Qaqish (2003) entwickelte eine Methode[4], die die binären Einträge der Beobachtungsvektoren nach und nach einzeln auf Grundlage der vorangegangenen Einträge erzeugt. Dabei werden sowohl die vorab angegebenen Wahrscheinlichkeiten π_i für die Ausprägung 1 und die Kovarianzstruktur $\mathbf{s}_i = \text{Cov}\left([Y_1, \ldots, Y_{i-1}]^T, Y_i\right)$ der jeweils zu bildenden Einträge mit den vorangegangenen Ausprägungen berücksichtigt. Der Erwartungswert für die konditionale Verteilung von Ausprägung Y_i lässt sich laut Lalonde (2017, S. 100) auf der Grundlage der vorangegangenen Ausprägungen wie folgt berechnen:

$$\text{E}\left[Y_i \mid [Y_1, \ldots, Y_{i-1}]^T\right] = \pi_i + \kappa_i^T \left([Y_1, \ldots, Y_{i-1}]^T - [\pi_1, \ldots, \pi_{i-1}]^T\right) \tag{5.10}$$

$$= \pi_i + \sum_{j=1}^{i-1} \kappa_{ij} \left(Y_j - \pi_j\right) , \tag{5.11}$$

κ_i ist dabei definiert als $\left[\text{Cov}\left([Y_1, \ldots, Y_{i-1}]^T\right)\right]^{-1} \mathbf{s}_i$.

Lalonde (2017, S. 100) zufolge wird die erste Ausprägung Y_1 des Beobachtungsvektors als einfaches Bernoulli-Experiment mit Wahrscheinlichkeit π_1 erstellt. Alle folgenden Ausprägungen sind das Ergebnis eines Bernoulli-Experiment mit den jeweils berechneten konditionalen Wahrscheinlichkeiten $\text{E}\left[Y_i \mid [Y_1, \ldots, Y_{i-1}]^T\right]$.

5.5. Simulation von Korrelationsmatrizen

Um eine gewünschte Korrelationstruktur innerhalb der Daten zu reproduzieren, ist es zunächst vonnöten realistische Korrelationsmatrizen zu simulieren, die sowohl die gewünschte Korrelationsstruktur wiedergeben als auch die formalen Bedingungen einer Korrelationsmatrix einhalten und dabei möglichst Eigenschaften aufweisen, die auch bei empirischen Korrelationsmatrizen auftreten. Die gewünschte Korrelationsstruktur wird dabei so definiert, dass sowohl innerhalb als auch zwischen festgelegten Variablenblöcken eine gewisse Basiskorrelation besteht, um die die einzelnen Einträge der Matrix streuen. Die formalen Bedingungen an eine Korrelationsmatrix sind erfüllt, wenn diese sowohl

[4] Eine Erweiterung der Methode mittels eines nicht-linearen Modells und externer Prädiktoren wird von Farrell und Sutradhar (2006) vorgeschlagen.

symmetrisch als auch positiv semi-definit ist und die Hauptdiagonale aus Einsen besteht (Hadd & Rodgers, 2021, S. 19). Eine wünschenswerte Eigenschaft einer realistischen Korrelationsmatrix ist darüber hinaus ein gewisser Grad an *Rauschen*[5] (Hardin et al., 2013, S. 1733–1734).

In dieser Arbeit wird die Methode von Hardin et al. (2013) verwendet. Die Grundidee der Methode liegt darin, zunächst eine Ausgangsmatrix mit der gewünschten Basiskorrelation innerhalb und zwischen den jeweiligen Variablenblöcken zu erstellen und dann dieser Matrix Rauschen hinzuzufügen, was die Korrelationsmatrix vergleichbarer mit empirischen Korrelationsmatrizen macht. Wichtig ist hierbei, dass die Residuen[6] zwischen der resultierenden Korrelationsmatrix und der Ausgangsmatrix innerhalb eines vorher spezifizierten Intervalls bleiben. Außerdem ist es wichtig, dass die im vorangegangenen Absatz beschriebenen formalen Bedingungen an eine Korrelationsmatrix eingehalten werden.

Das Vorgehen wird von Hardin et al. (2013, S. 1739) wie folgt angegeben[7]: Zunächst wird eine $N \times N$ Ausgangsmatrix Σ erstellt, deren diagonale Blöcke aus den Submatrizen $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ bestehen und sonst Nullen aufweist. Eine Submatrix Σ_k ist eine $g_k \times g_k$ Korrelationsmatrix, welche folgende Form annehmen kann:

$$\Sigma_k = \begin{pmatrix} 1 & \rho_k & \cdots & \rho_k \\ \rho_k & 1 & \cdots & \rho_k \\ \vdots & \vdots & \ddots & \vdots \\ \rho_k & \rho_k & \cdots & 1 \end{pmatrix}, \tag{5.12}$$

wobei K die Anzahl der Variablenblöcke $k = 1, 2, \ldots, K$ darstellt und g_k die Anzahl der Variablen innerhalb des Block k. Die Basiskorrelation innerhalb Block k ist mit ρ_k angegeben. Es werden N Einheitsvektoren aus einer Gleichverteilung erstellt, die folgende Form annehmen (Hardin et al., 2013, S. 1742): x_i ist eine unabhängig und identisch verteilte Zufallsvariable mit $N(0, 1)$ und $i = 1, 2, \ldots, M$. Das ergibt den Vektor $\mathbf{x} = (x_1, x_2, \ldots, x_M)$, welcher mittels $\mathbf{v} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$ normiert wird. Der Wert M ist dabei die

[5] Unter Rauschen wird in diesem Zusammenhang eine geringe zufällige Abweichung in beide Richtungen der Korrelation einzelner Variablen verstanden.

[6] Ein Residuum ist in diesem Kontext die Abweichung der Einträge der beiden Matrizen voneinander.

[7] Der Beweis, dass die resultierende Korrelationsmatrix die formellen Vorgaben an Korrelationsmatrizen erfüllt kann bei Hardin et al. (2013, S. 1754–1755) nachgelesen werden.

Dimensionalität des Vektorraumes.

Das Skalarprodukt der auf diese Weise zufällig erstellten Einheitsvektoren $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$ bildet die Grundlage des hinzugefügten Rauschens. Dies resultiert dann nach folgender Regel in der $N \times N$ Korrelationsmatrix S:

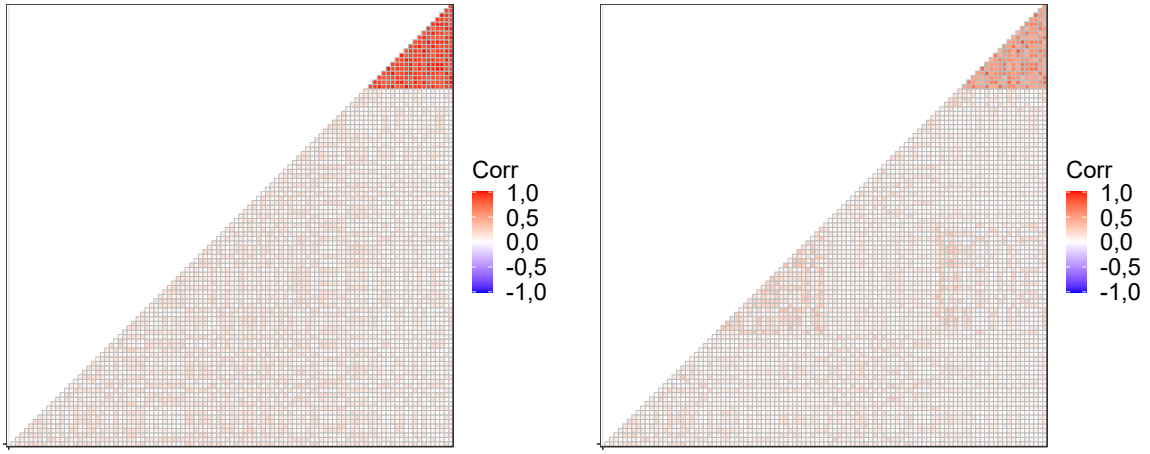
$$S_{ij} = \begin{cases} 1, & \text{falls } i = j, \\ \rho_k + \varepsilon \mathbf{u}_i^T \mathbf{u}_j, & \text{falls } i, j \text{ in Block } k \text{ sind und } i \neq j, \\ \delta + \varepsilon \mathbf{u}_i^T \mathbf{u}_j, & \text{falls } i, j \text{ in unterschiedlichen Blöcken sind .} \end{cases} \tag{5.13}$$

Dabei ist δ das Basisrauschen zwischen den Blöcken und wird mit $0 \leq \delta < \rho_{\min}$ gewählt. $\rho_{\min}$ ist die minimale Korrelation innerhalb aller Blöcke. Der Maximalwert des Rauschens ist mit ε angegeben und wird gewählt mit $0 \leq \varepsilon < 1 - \rho_{\max}$. Entsprechend $\rho_{\min}$ ist $\rho_{\max}$ die maximale Korrelation innerhalb aller Blöcke.

5.6. Analyse der Simulationsdaten

In diesem Abschnitt wird zunächst beispielhaft die Struktur eines simulierten Datensatzes nach der Methode von Emrich und Piedmonte (1991) (siehe Abschnitt 5.3.1) analysiert. Die Kriterien sind dabei vor allem, ob die gewünschte Korrelationsstruktur erreicht wird, ebenso wie die gewünschten Mittelwerte der Variablen. Darüber hinaus werden die Daten daraufhin überprüft, ob etwaige Auffälligkeiten wie etwa Ausreißer auszumachen sind. Außerdem werden Beispieldaten visualisiert um mögliche Auffälligkeiten zu identifizieren und die Clusterstruktur sichtbar zu machen.

Um die Funktionsfähigkeit der Methode zu überprüfen wurden die 3 verschiedenen Szenarien *Spezial*, *4 Block* und *Einfach* analysiert. Es wurden jeweils 2 Cluster mit jeweils 50.000 Beobachtungen gebildet. Die Mittelwerte der Variablen innerhalb der verschiedenen Blöcke entsprechen der Konfiguration „Mittel“ und die Höhe der Basiskorrelation ebenfalls der Konfiguration „Mittel“. Diese wurden gewählt, da sie Datensätze mit den ausgeprägtesten Strukturen bilden. Es wurden ebenfalls Mittelwerte über dem Wert von 0,5 verwendet. Dabei fiel jedoch auf, dass dies dazu führt, dass die Standardabweichung der resultierenden Variablen bei 0 lag. Anders ausgedrückt bestanden die entsprechenden Spalten der Datenmatrix entweder ausschließlich aus 0 oder ausschließlich aus 1. Daher können nur Mittelwerte bis 0,49 verwendet werden.



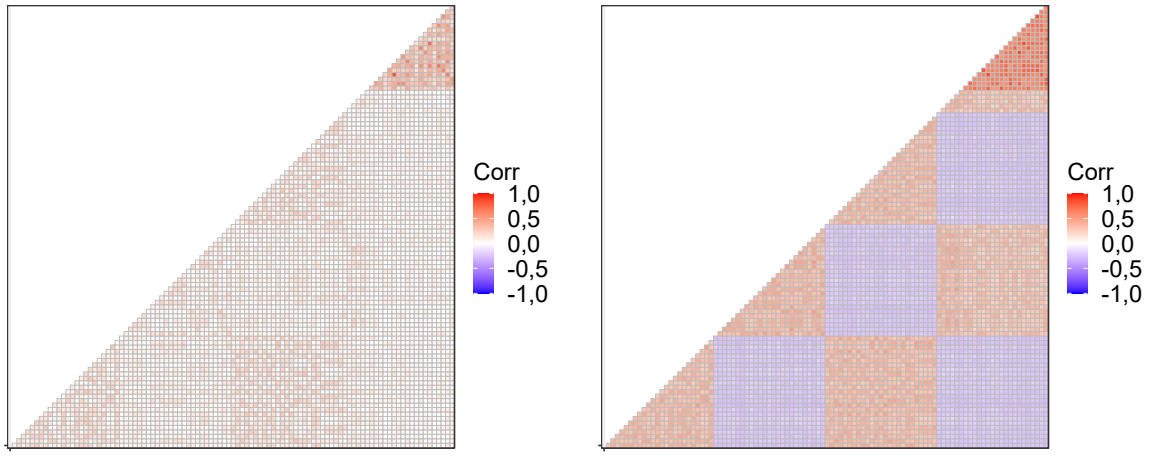
(a) Gewünschte Korrelation (b) Korrelation Cluster 1

Abbildung 5.4.: Vergleich gewünschte Korrelationsmatrix vs. Korrelationsmatrix Cluster 1

Es wurden 100 Variablen simuliert. Im Falle der Konfiguration mit 2 Blöcken bestand der kleine Block aus 20 Variablen und der große Block aus 80 Variablen. Im Falle von 4 Blöcken waren die Blöcke gleicher Größe.

Konfiguration 1: Szenario *Spezial* Blöcke in Korrelationsmatrix: 2
Basiskorrelation (pro Block): 0,1, 0,9
n: 100.000 (50.000 pro Cluster)
Prävalenz: Konfiguration „Mittel“
Variablenblöcke in Daten bezüglich unterschiedlicher Prävalenz: 4
Mittelwerte Cluster 1: low (Block 1), high (Block 2), low (Block 3), high (Block 4)
Mittelwerte Cluster 2: high (Block 1), low (Block 2), high (Block 3), low (Block 4)

Wie sich in Abbildung 5.4 und 5.5 sehen lässt, sind die Korrelationen in den beiden Clustern etwas niedriger, treffen die gewünschte Korrelationsstruktur aber dennoch relativ gut. Selbiges lässt sich für den gesamten Datensatz (Cluster 1 und Cluster 2) sagen. Die mittlere Abweichung der gewünschten Korrelationsmatrix von der Korrelationsmatrix von Cluster 1 beträgt 0,02, von Cluster 2 0,03 und vom gesamten Datensatz 0,06.



(a) Cluster 2 (b) Gesamter Datensatz

Abbildung 5.5.: Korrelationsmatrix Cluster 2 vs. Korrelationsmatrix Gesamt

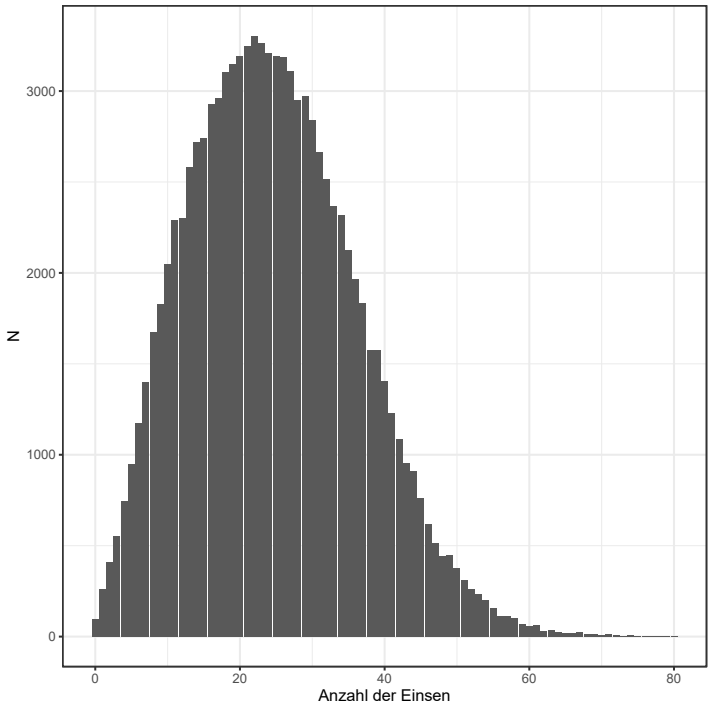
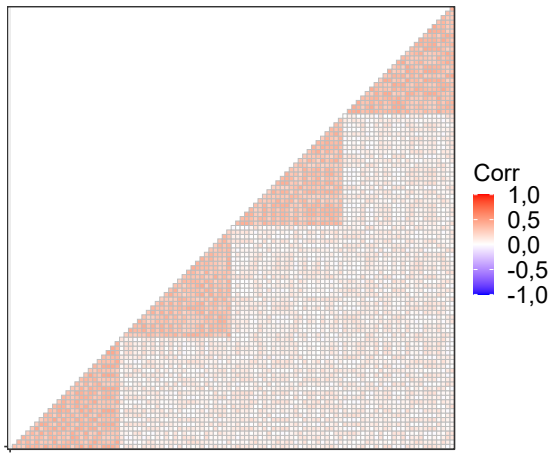


Abbildung 5.6.: Anzahl Beobachtungen mit Anzahl an Einsen in Konfiguration 1

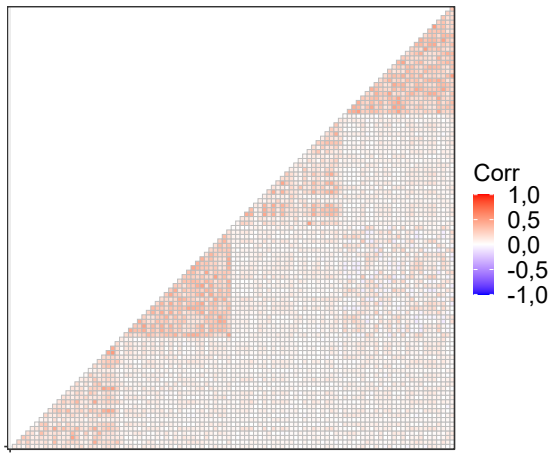
Das arithmetische Mittel der angestrebten Mittelwerte minus der Mittelwerte der simulierten Daten beträgt in Cluster 1: 0 und in Cluster 2: 0. Um einen Eindruck über die Gestalt der einzelnen Beobachtungen bzw. die Verteilung der Ausprägungen „1“ über die einzelnen Beobachtungen zu bekommen wurde ein Barplot (Abbildung 5.6) mit der Anzahl der Beobachtungen mit entsprechender Anzahl der Ausprägung „1“ erstellt. In Anbetracht der gewählten Mittelwertstruktur zeigt sich ein erwartbares Bild: Der Großteil der Beobachtungen weist weniger als 50 Einsen auf mit einem Modus bei 22. Allerdings gibt es auch 82 Beobachtungen in Cluster 1 und 14 Beobachtungen in Cluster 2, die lediglich aus Nullen bestehen.

Dieser Umstand deutet eine gewisse Problematik an. Die Cluster replizieren die gewünschte Mittelwertstruktur sehr gut, was sie dann bei unterschiedlich gewählter Mittelwertstruktur auch strukturell voneinander abgrenzt, ohne die gewünschte Korrelationsstruktur zu verletzen. Während somit die aggregierten Werte die Cluster voneinander trennen, sind die einzelnen Beobachtungen nicht zwingend trennscharf einem bestimmten Cluster zuzuordnen, was sich an dem Beispiel der Beobachtungen, die nur aus Nullen bestehen und in beiden Clustern vorkommen, gut demonstrieren lässt. Daraus lässt sich folgern, dass die gewählte Methode zwar strukturell unterschiedliche Cluster erzeugt, diese aber eine gewisse Variabilität in den Beobachtungen mit sich bringen und somit nicht perfekt trennscharf sind und keine perfekten Cluster darstellen. Dies muss bei der Evaluation der getesteten Algorithmen bedacht werden. Eine perfekte Abgrenzung der einzelnen Cluster wäre natürlich für die Evaluation der Clusteralgorithmen wünschenswert (Szenario „Einfach“ approximiert dies), wobei bei empirischen Daten tatsächlich eher damit gerechnet werden muss, dass diese ebenfalls keine Clusterstrukturen aufweisen, welche so eindeutig getrennt werden können.

Konfiguration 2: Szenario „4 Block“ Blöcke in Korrelationsmatrix: 4
Basiskorrelation (pro Block): Zwischen 0,3 und 0,4 (Konfiguration „Mittel“)
n: 100.000 (50.000 pro Cluster)
Prävalenz: Konfiguration „Mittel“
Variablenblöcke in Daten bezüglich unterschiedlicher Prävalenz: 4
Mittelwerte Cluster 1: low (Block 1), high (Block 2), low (Block 3), high (Block 4)
Mittelwerte Cluster 2: high (Block 1), low (Block 2), high (Block 3), low (Block 4)

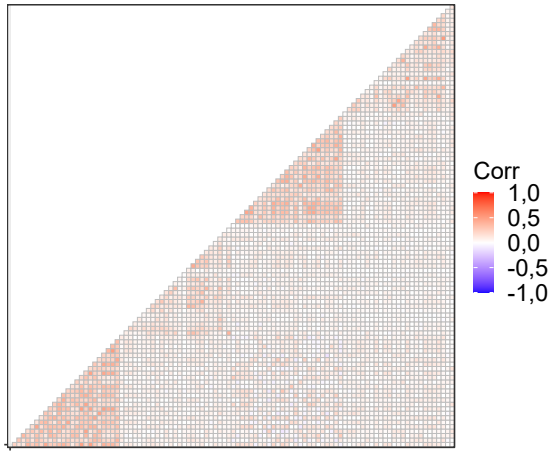


(a) Gewünschte Korrelation

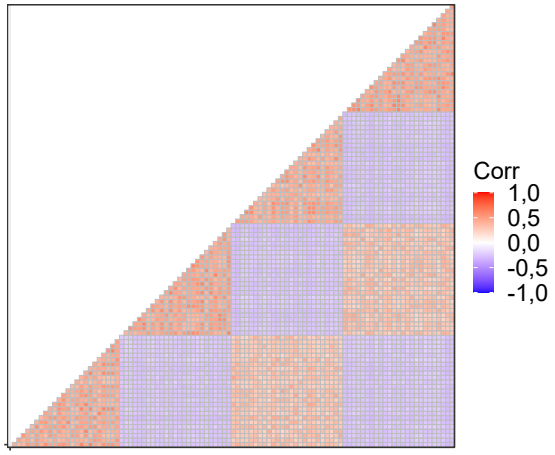


(b) Korrelation Cluster 1

Abbildung 5.7.: Vergleich Gewünschte Korrelationsmatrix vs. Korrelationsmatrix Cluster 1



(a) Cluster 2



(b) Gesamter Datensatz

Abbildung 5.8.: Korrelationsmatrix Cluster 2 vs. Korrelationsmatrix Gesamt

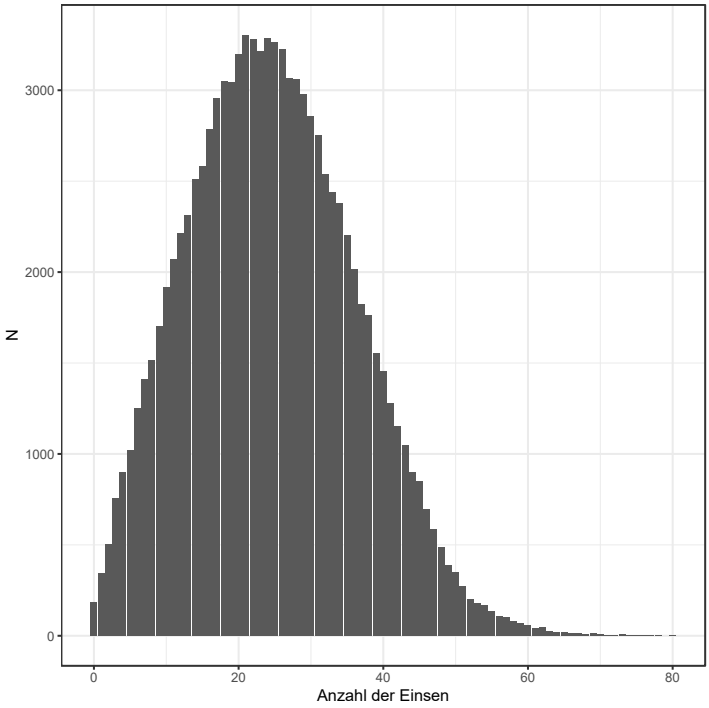
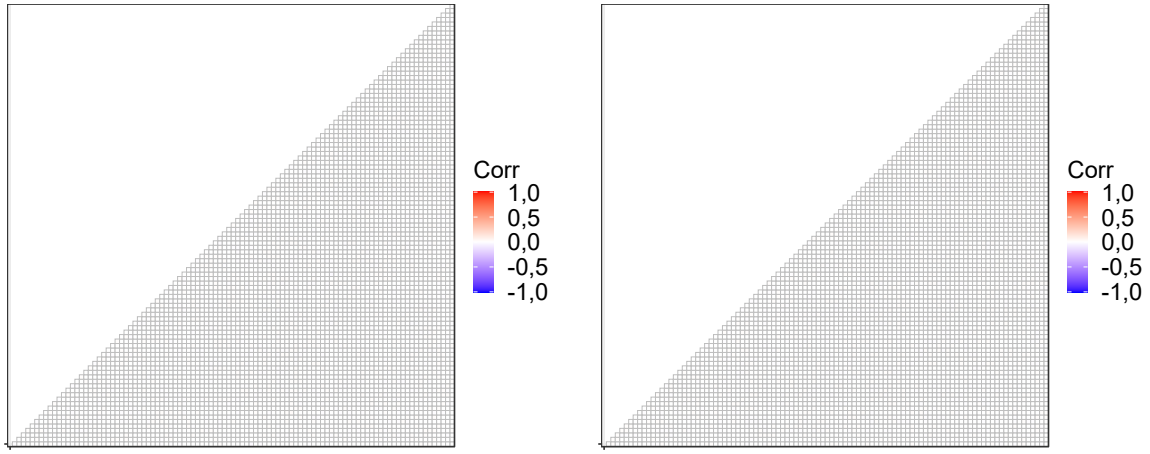


Abbildung 5.9.: Anzahl Beobachtungen mit Anzahl an Einsen in Konfiguration 2

Auch hier lässt sich sehen, dass die gewünschte Korrelationsstruktur approximativ erreicht wurde. Innerhalb der einzelnen Cluster kann zwar festgestellt werden, dass in den Blöcken mit niedriger Prävalenz die Korrelationen zwischen einzelnen Variablen erkennbar niedriger sind. Beim Zusammenfügen der beiden Cluster zu einem Datensatz gleicht sich dies allerdings wieder etwas aus und die Korrelationsstruktur entspricht in etwa der gewünschten. Wird die Korrelationsmatrix des simulierten Datensatzes von der gewünschten Korrelationsmatrix subtrahiert, entspricht das arithmetische Mittel der Differenzen der einzelnen Einträge der Matrizen 0,07. Die mittlere Differenz zwischen dem gewünschten Mittelwertvektor und dem simulierten innerhalb des ersten Clusters beträgt 0 und innerhalb des zweiten 0.

Die Problematik der nicht eindeutigen Abgrenzbarkeit zwischen den Clustern tritt allerdings auch hier wieder auf. So gibt es 1 Beobachtungspattern welches in beiden Clustern zu finden ist. Jedoch besteht dieses wiederum nur aus Nullen. Insgesamt gibt es 410 dieser Duplikate im gesamten Datensatz, welche sich auf beide Cluster aufteilen. Um dem Problem der identischen Beobachtungen in unterschiedlichen Clustern zu entgehen, wird in dieser Arbeit bei der Simulation eines Clusters ein Hashwert jeder erzeugten Beobachtung innerhalb des Clusters erstellt. Bei der Simulation des jeweils nächsten



(a) Korrelation Cluster 1 (b) Korrelation Cluster 2

Abbildung 5.10.: Vergleich Korrelationsmatrix Cluster 1 vs. Korrelationsmatrix Cluster 2

Clusters werden die Hashwerte der Beobachtungen dann mit denen der vorangegangenen Cluster verglichen. Sollte auf diese Weise ein Duplikat innerhalb eines anderen Clusters entdeckt werden, wird die Beobachtung erneut simuliert, bis keine Duplikate in unterschiedlichen Clustern mehr vorhanden sind.

Konfiguration 3: Szenario „Einfach“ Blöcke: 2
n: 100.000 (50.000 pro Cluster)
Prävalenz: Szenario „Mittel“
Variablenblöcke in Daten bezüglich unterschiedlicher Prävalenz: 4
Mittelwerte Cluster 1: low (Block 1), high (Block 2)
Mittelwerte Cluster 2: high (Block 1), low (Block 2)

Die Verteilung der Anzahl an Beobachtungen mit entsprechender Anzahl von Einsen innerhalb der Beobachtungen in Abbildung 5.12 zeigt ebenfalls keinerlei Auffälligkeiten.

In den Abbildungen 5.13 und 5.14 sind Visualisierungen der simulierten Daten zu sehen. Ein schwarzer Punkt stellt eine 1 dar, während ein weißer Punkt entsprechend eine 0 darstellt. Um die Daten abbilden zu können wurden hier Datensätze mit lediglich 1.000 Beobachtungen simuliert, was ausreicht um die Struktur der Daten erkennbar

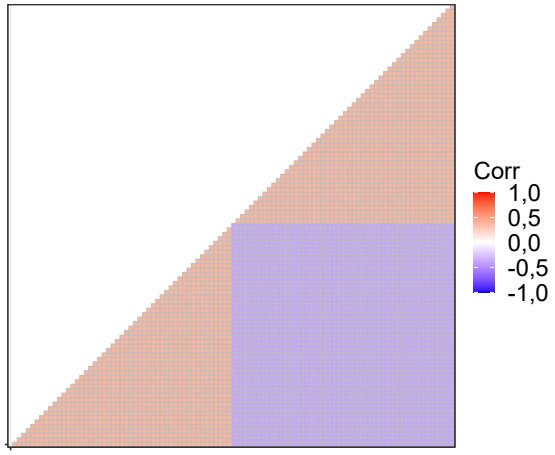


Abbildung 5.11.: Korrelationsmatrix Gesamt

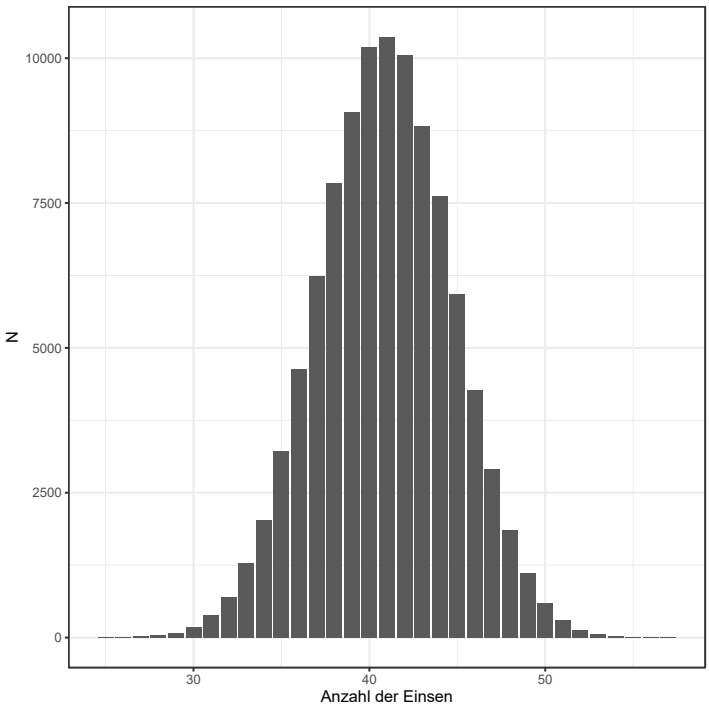
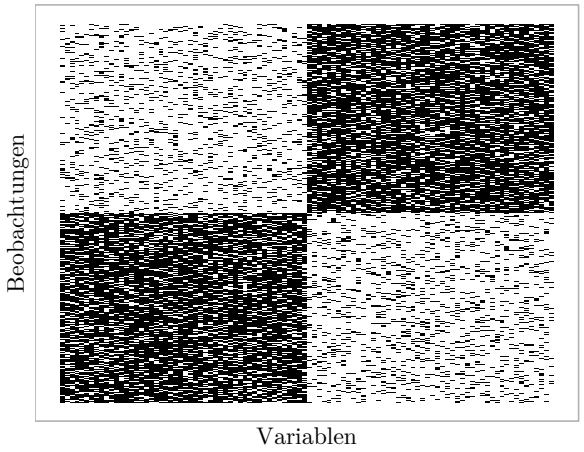


Abbildung 5.12.: Anzahl Beobachtungen mit Anzahl an Einsen in Konfiguration 3

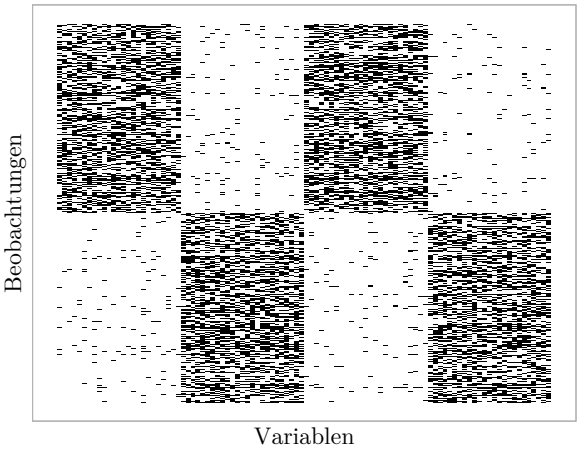
zu machen. Zusätzlich werden an dieser Stelle lediglich die Datensätze mit mittlerer Prävalenzstruktur abgebildet. Die entsprechenden Abbildungen der Datensätze mit niedriger Prävalenz finden sich in Anhang A. Außerdem wurden hier die Beobachtungen nicht zufällig angeordnet, was in der Simulation der Fall ist. Auf diese Weise lassen sich die Cluster in den Abbildungen besser erkennen. Ansonsten stimmen die verwendeten Parameter mit denen aus der Simulation überein.

In allen Datensätzen ist direkt auffallend, dass die Generierung der Cluster sehr gut funktioniert hat und die Clusterstruktur mit bloßem Auge sofort ausgemacht werden kann. Die einzelnen Variablenblöcke mit unterschiedlicher Prävalenzstruktur setzen sich sehr deutlich voneinander ab. Erkennbar ist auch die generell höhere Prävalenz in den Daten des Szenario *Einfach*. Bei genauerer Betrachtung fällt auf, dass bei höherer Korrelation aufeinanderfolgende Ausprägungen innerhalb einer Beobachtung und innerhalb eines Variablenblocks dazu tendieren die gleiche Ausprägung aufzuweisen. Vor allem fällt dies beim Szenario *Spezial* in den 20 Variablen am äußeren rechten Rand auf, wo die Korrelation der Variablen untereinander bei $r = 0{,}9$ liegt. Dies führt dazu, dass es Beobachtungen mit vielen aufeinanderfolgenden Einsen innerhalb eines Variablenblocks gibt und ebenfalls Beobachtungen mit aufeinanderfolgenden Nullen. Dies ist zwangsläufig der Fall, damit der Algorithmus die hohe Korrelationsstruktur repliziert, als auch die Mittelwerte einhält. Entsprechend lässt sich dieses Phänomen nicht in den Daten des Szenario *Einfach* beobachten, indem die Korrelation keine Rolle spielt und lediglich die gewünschten Mittelwerte als Zielwert erfüllt werden müssen.

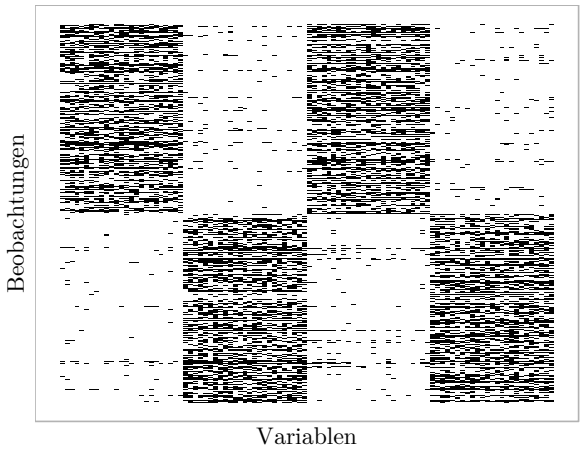
n = 1000
Prävalenz = mid
n Cluster = 2
Szenario = einfach
Korrelationsstruktur = N/A



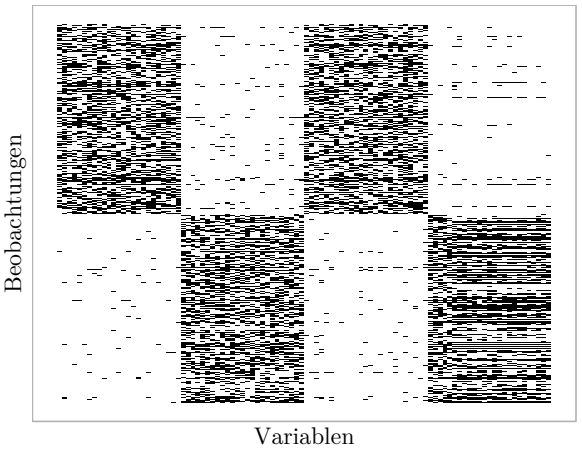
n = 1000
Prävalenz = mid
n Cluster = 2
Szenario = 4block
Korrelationsstruktur = low



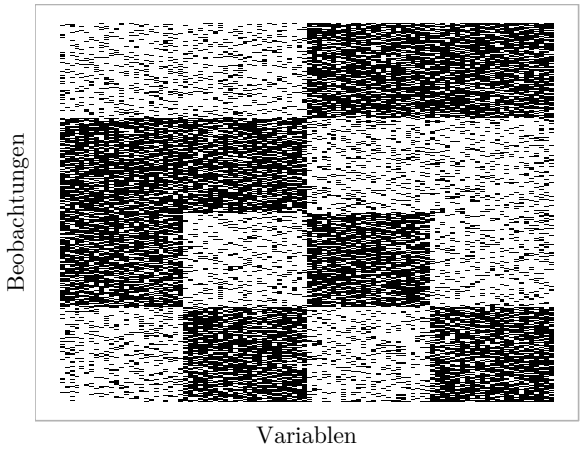
n = 1000
Prävalenz = mid
n Cluster = 2
Szenario = 4block
Korrelationsstruktur = mid



n = 1000
Prävalenz = mid
n Cluster = 2
Szenario = special
Korrelationsstruktur = N/A



n = 1000
Prävalenz = mid
n Cluster = 4
Szenario = einfach
Korrelationsstruktur = N/A



n = 1000
Prävalenz = mid
n Cluster = 4
Szenario = 4block
Korrelationsstruktur = low

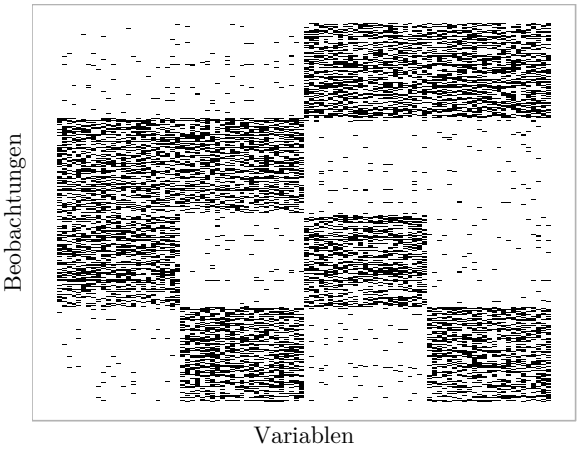
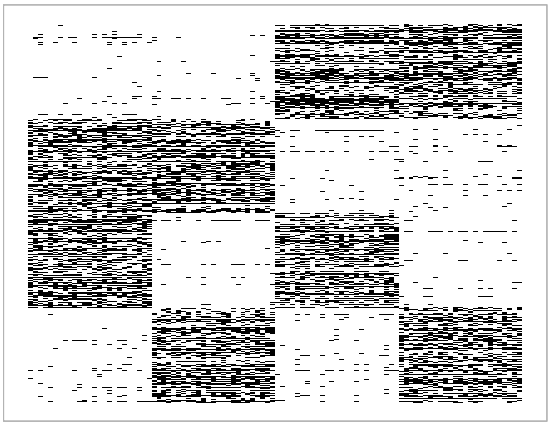
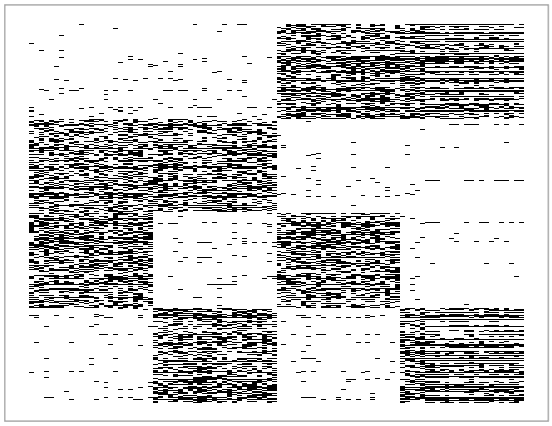


Abbildung 5.13.: Visualisierung der simulierten Daten

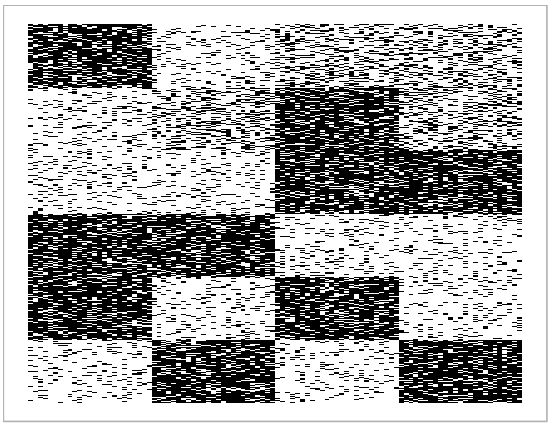
n = 1000
Prävalenz = mid
n Cluster = 4
Szenario = 4block
Korrelationsstruktur = mid



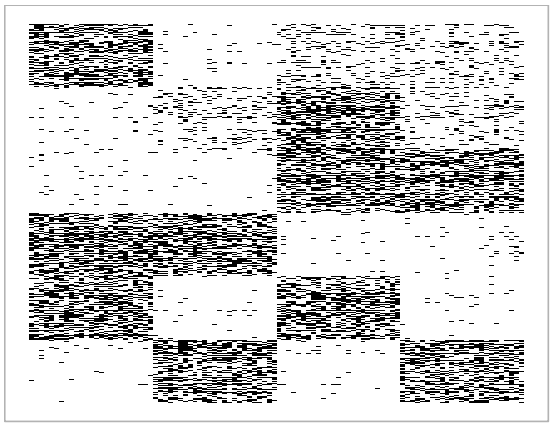
n = 1000
Prävalenz = mid
n Cluster = 4
Szenario = special
Korrelationsstruktur = N/A



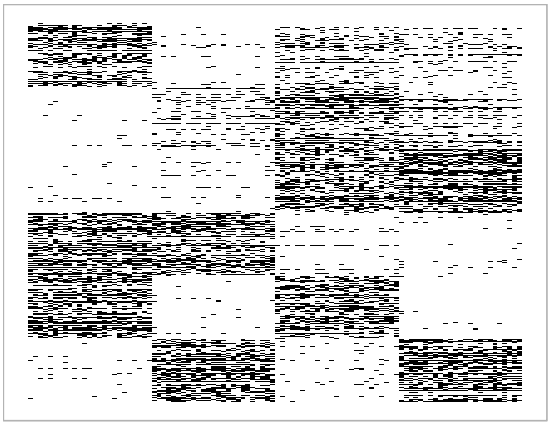
n = 1000
Prävalenz = mid
n Cluster = 6
Szenario = einfach
Korrelationsstruktur = N/A



n = 1000
Prävalenz = mid
n Cluster = 6
Szenario = 4block
Korrelationsstruktur = low



n = 1000
Prävalenz = mid
n Cluster = 6
Szenario = 4block
Korrelationsstruktur = mid



n = 1000
Prävalenz = mid
n Cluster = 6
Szenario = special
Korrelationsstruktur = N/A

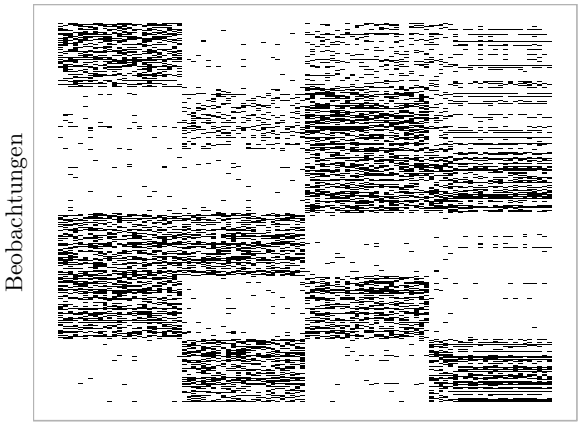


Abbildung 5.14.: Visualisierung der simulierten Daten

6. Beurteilungskriterien

Um die Performance der getesteten Clusteralgorithmen angemessen bewerten zu können, werden verschiedene Kennwerte herangezogen, welche sich in der Literatur zur Beurteilung von Klassifikationsergebnissen und Clusteranalysen bewährt haben. Dabei muss zwischen externen und internen Beurteilungskriterien unterschieden werden. Externe Beurteilungskriterien evaluieren die resultierende Clusterlösung anhand eines Sets von vorher bekannten Clusterlabels der Beobachtungen im Datensatz. Ein Clusterlabel beschreibt die tatsächliche Zugehörigkeit einer Beobachtung zu einem Cluster. In der Regel ist das Label im Vorhinein nicht bekannt. Das Anwendungsgebiet dieser Kriterien ist primär die Evaluation der Ergebnisse eines neuen Clusteralgorithmus oder die Performance eines bereits bekannten Algorithmus unter verschiedenen Szenarien. Daher wird diese Form der Evaluierungskriterien in dieser Arbeit verwendet. Interne Beurteilungskriterien hingegen evaluieren das resultierende Clustering rein anhand inhärenter Eigenschaften des jeweiligen Clusterings, ohne jedwede Informationen darüber im Vorhinein zu benötigen. Diese Form der Beurteilungskriterien werden in der Regel in der Datenanalyse verwendet, wenn das Erkenntnisinteresse tatsächlich am Clustering der Daten und dem damit einhergehenden Informationsgewinn über den inhaltlichen Gegenstand von Interesse ist. Im Folgenden werden sowohl einige verwendete und nicht verwendete externe Beurteilungskriterien wie auch interne Beurteilungskriterien vorgestellt.

Aufgrund der Tatsache, dass die Performance der Clusteralgorithmen in großen Datensätzen evaluiert werden soll, stellt die Berechnungsdauer der betrachteten Algorithmen ein weiteres Evaluationskriterium dar. Außerdem lässt sich im Bedarf an verfügbaren Arbeitsspeicher ein weiteres zentrales Kriterium sehen, das bei dem Arbeiten mit großen Datensätzen berücksichtigt werden muss. Demnach wird die Höhe des benötigten Speichers für die Erstellung der jeweiligen Clusterlösungen ebenfalls als Kriterium herangezogen. Für die Beschreibung des Ausmaß der Konzentration der Beobachtungen auf die Cluster wird der Gini-Koeffizient verwendet.

6.1. Externe Beurteilungskriterien

Wie bereits im vorangegangenen Abschnitt erläutert, werden in diesem Abschnitt, mögliche externe Evaluierungskriterien beschrieben, die das bereits bekannte Set der Clusterlabel nutzen, um die Güte der Clusterlösung zu bewerten. Aufgrund der Tatsache, dass in dieser Arbeit mit simulierten Daten gearbeitet wird, sind diese, in der Regel sonst unbekannten Clusterlabels, bekannt und können zur Evaluation verwendet werden.

6.2. Verwendete externe Beurteilungskriterien

Die wichtigste Eigenschaft der Clusteralgorithmen, welche es zu überprüfen gilt, liegt darin, den einzelnen Datenpunkten die korrekte Clusterzugehörigkeit zuzuweisen oder zumindest Beobachtungen mit gleichen Clusterlabels auch den gleichen Clustern zuzuweisen.

6.2.1. Recall, Precision, F-Score

Recall und *Precision* sind zwei Kriterien, welche vor allem im Information Retrieval verwendet werden. Dabei wird bewertet, wie gut ein Algorithmus aus einem Set von Dokumenten[1] die relevanten Dokumente abrufen kann, die ein bestimmtes Kriterium erfüllen (Witten et al., 1999, S. 188–189). In diesem Anwendungsfall berechnet sich Recall als der Anteil der erkannten relevanten Dokumente an den relevanten Dokumenten insgesamt. Precision wiederum ergibt sich als der Anteil der relevanten Dokumente an den vom Algorithmus abgerufenen Dokumenten.

Generell lässt sich Recall und Precision mit Klassifikationsergebnissen berechnen, die als *True Positive (TP)*, *False Positive (FP)* und *False Negative (FN)* ausgedrückt werden können (Christen, 2012, S. 167):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}\,, \tag{6.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}\,. \tag{6.2}$$

[1] Dabei handelt es sich in der Regel um Textdokumente.

Aufgrund der Tatsache, dass Recall und Precision unterschiedliche Aspekte des Klassifikationsergebnisses bewerten, kann durch das Ändern eines Parameters des angewendeten Algorithmus, der eine Wert ansteigen, während der andere fällt. In Fällen jedoch, wo ein hoher Wert für beide Kriterien wünschenswert ist, bietet sich die Kombination beider Werte an (Baeza-Yates, 1999, S. 81). Der sogenannte *F-Score* ist eine Möglichkeit genau das zu erreichen und wird durch das harmonische Mittel von Recall und Precision berechnet:

$$\text{F-Score } = 2 \times \left(\frac{\text{Precision } \times \text{ Recall}}{\text{Precision } + \text{ Recall}} \right) . \tag{6.3}$$

Jedoch gibt es auch Kritik an der Verwendung des F-Score. Hand und Christen (2017) zeigen, dass der F-Score auch als gewichtete Summe von Recall und Precision berechnet werden kann. Formell bedeutet das:

$$\begin{aligned} \text{F-Score } &= \frac{c+d}{c+b+2d} \times \frac{d}{c+d} + \frac{b+d}{c+b+2d} \times \frac{d}{b+d} \\ &= p \times \text{Recall} + (1-p)\text{Precision} , \end{aligned} \tag{6.4}$$

mit

$$p = \frac{c+d}{c+b+2d} . \tag{6.5}$$

Dabei gibt b einen *False Match*[2] an, c einen *False Non-Match* und d einen *True Match*. Das Verhältnis des Gewichts p zu $1-p$ ist durch die Anzahl der True Matches dividiert durch die Anzahl der geschätzten Matches bestimmt. Hand und Christen (2017, S. 542) argumentieren, dass die Anzahl der geschätzten Matches eine Funktion des verwendeten Algorithmus ist, wodurch je nach verwendetem Algorithmus bei Berechnung des F-Scores Recall und Precision unterschiedlich gewichtet werden. Das bedeutet, dass für einen korrekten Vergleich der Performance zweier Algorithmen sichergestellt werden müsste, dass beide Algorithmen zur gleichen Anzahl an geschätzten Matches kommen. Sie schlagen vor, dies zu erreichen, indem der Schwellenwert, ab wann ein Beobachtungspaar als

[2]Das zitierte Paper stammt aus dem Bereich des Record Linkage, wo ein Match das Zusammenfügen eines Records aus der einen Datenbank mit einem Record aus einer anderen Datenbank beschreibt.

hinreichend ähnlich betrachtet wird, entsprechend gewählt wird.

Für den Vergleich der Ergebnisse unterschiedlicher Clusteralgorithmen ist dies jedoch nicht so ohne weiteres möglich, da diese oftmals auf vollkommen unterschiedlichen Heuristiken beruhen. Dadurch ist es nahezu unmöglich die erforderlichen Parameter so zu wählen, dass die gleiche Anzahl an True und False Positives sichergestellt werden kann (zur Definition von True und False Positives im Rahmen der Clusteranalyse siehe weiter unten). Die beschriebene Kritik am F-Score sollte jedoch bei der Evaluation im Hinterkopf behalten werden.

Übertragen auf die Bewertung von Ergebnissen einer Clusteranalyse sind zwei verschiedene Vorgehensweisen zur Berechnung von Recall, Precision und des F-Score denkbar. Die eine basiert darauf, ob die Beobachtungen den richtigen Clustern zugewiesen wurden. Die andere basiert darauf, ob Beobachtungspaare korrekt entsprechend ihrer Zusammengehörigkeit geclustert wurden oder nicht. Ersterer Ansatz betrachtet die Cluster als Ganzes und bewertet, wie gut die Originalclusterstruktur reproduziert werden konnte. Die zweite Vorgehensweise setzt eine Ebene tiefer an und bewertet, wie gut der Algorithmus die Ähnlichkeit einzelner Beobachtungspaare bewertet. Aufgrund der Tatsache, dass mit den anderen in diesem Abschnitt vorgestellten Kriterien angemessene Methoden zur Verfügung stehen, welche das Clustering als Ganzes bewerten, wird in dieser Arbeit der Ansatz gewählt, Recall, Precision und in der Folge auch den F-Score auf Grundlage von Beobachtungspaaren zu berechnen.

Demnach ergibt sich ein True Positive als ein Beobachtungspaar, welches korrekterweise vom Clusteralgorithmus im selben Cluster verortet wurde. Ein False Positive ist dementsprechend ein Paar, welches im selben Cluster verortet wurde, aber tatsächlich unterschiedliche Clusterlabels trägt. Ein False Negative schließlich ist ein Beobachtungspaar, welches fälschlicherweise in unterschiedliche Cluster verortet wurde, aber tatsächlich aus dem selben Cluster stammt.

Diese Werte lassen sich einfach aus der Kreuztabelle des Vektors der tatsächlichen Clusterlabels und dem Vektor des Clusterings des Clusteralgorithmus berechnen. Die Summe der True Positives bildet sich als:

$$\mathrm{TP} = \sum_{i}^{I} \sum_{j}^{J} \binom{x_{ij}}{2}, \tag{6.6}$$

wobei i und j die Zeilen- bzw. Spaltenindezes der Kreuztabelle X darstellen und x_{ij} die jeweiligen Einträge. False Positives ergeben sich als die Differenz aller Positiven zu den True Positives:

$$\mathrm{FP} = \mathrm{TPFP} - \mathrm{TP}\,, \tag{6.7}$$

wobei sich alle Positiven TPFP berechnen als:

$$\mathrm{TPFP} = \sum_{j}^{J} \binom{\sum_{i}^{I} x_{ij}}{2}. \tag{6.8}$$

False Negatives berechnen sich wie folgt:

$$\mathrm{FN} = \sum_{i}^{I} \sum_{j}^{J-1} x_{ij} \times \sum_{j+1}^{J} x_{ij}; \forall x_{ij} > 0\,. \tag{6.9}$$

6.2.2. Rand Index

Ebenfalls bietet es sich an den sogenannten *Rand Index* (Rand, 1971) zur Evaluation der Clusterlösung zu nutzen. Dabei existieren bei zwei gegebenen Clusterlösungen für jedes Paar von Datenpunkten zwei verschiedene Szenarien:

1. Die beiden Punkte werden von beiden Clusterlösungen in jeweils denselben Cluster klassifiziert oder die beiden Punkte werden von beiden Clusterlösungen in jeweils unterschiedliche Cluster klassifiziert.

2. Die beiden Punkte werden von einer Clusterlösung zusammen klassifiziert und von der anderen getrennt.

Gan et al. (2007, S. 311) definieren auf dieser Grundlage den Rand Index folgendermaßen: Sei Datensatz D gegeben mit n Datenpunkten $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ und $\mathcal{P} = \{c_1, c_2, \ldots, c_{k_1}\}$ und $\mathcal{P}' = \left\{c'_1, c'_2, \ldots, c'_{k_2}\right\}$ stellen die beiden Clusterlösungen dar. Der Rand Index ergibt sich dann als:

$$c\left(\mathcal{P}, \mathcal{P}'\right) = \frac{1}{\left(\begin{array}{c} n \\ 2 \end{array}\right)} \sum_{1 \leq i < j \leq n} \gamma_{ij}\,, \tag{6.10}$$

mit

$$\gamma_{ij} = \begin{cases} 1 & \text{wenn } \exists l, l' \text{ s.t. } \mathbf{x}_i \in c_l \cap c'_{l'} \text{ und } \mathbf{x}_j \in c_l \cap c'_{l'} \\ 1 & \text{wenn } \exists l, l' \text{ s.t. } \mathbf{x}_i \in c_l \cap c'_{l'} \text{ und } \mathbf{x}_j \notin c_l \cup c'_{l'} \\ 0 & \text{anderenfalls.} \end{cases} \tag{6.11}$$

Der Rand Index nimmt Werte von 0 bis 1 an, wobei die beiden Clusterlösungen komplett verschieden sind, wenn der Rand Index 0 beträgt und, sollten die beiden Clusterlösungen komplett übereinstimmen, nimmt der Rand Index den Wert 1 an. Für die Evaluation in dieser Arbeit werden nicht zwei verschiedene Clusteralgorithmen verglichen, sondern die Clusterlösung $\mathcal{P}'$ stellt in diesem Falle das vorher bekannte simulierte Set der Clusterlabels[3] dar. Somit würde ein Rand Index von 1 bedeuten, dass der Clusteralgorithmus die vorher festgelegte Clusterstruktur perfekt reproduzieren konnte.

6.2.3. Entropie

Der Begriff der Entropie wurde von Shannon (1948) eingeführt und beschreibt die Unsicherheit der Realisation eines Outcomes einer Zufallsvariable. Der Begriff stammt aus der Informationstheorie und kann in anderen Worten auch wie folgt ausgedrückt werden: „The entropy of a random variable is a measure of the uncertainty of the random variable; it is a measure of the amount of information required on the average to describe the random variable“ (Cover & Thomas, 2006, S. 19). Formell ist die Entropie H wie folgt definiert:

$$H(P) = -\sum_{x \in X} P(x) \log_2 P(x)\,. \tag{6.12}$$

[3]Die Bezeichnung $\mathcal{P}'$ für das bekannte Set der Clusterlabels, wird für den Rest der Arbeit beibehalten.

Ein maximaler Wert für H wird erreicht, wenn die Wahrscheinlichkeit $P(x)$ für jedes Outcome den gleichen Wert annimmt. Minimiert wird sie entsprechend, wenn die Wahrscheinlichkeit für ein Ereignis 1 beträgt und für die restlichen 0 (Manning et al., 2008, S. 92). Übertragen auf die Bewertung von Clusterlösungen beschreibt Entropie den Informationsgehalt der Clusterlösung. Die Wahrscheinlichkeiten für die jeweiligen Ereignisse nehmen in diesem Fall die Form an, dass eine Beobachtung mit Clusterlabel c' in Cluster c verortet ist. Die Entropie eines einzelnen Clusters ist somit formell definiert mit:

$$H(c) = -\sum_{c' \in P'} \frac{|\omega_{c'}|}{n_c} \log_2 \frac{|\omega_{c'}|}{n_c}\,, \tag{6.13}$$

wobei $|\omega_{c'}|$ die Anzahl der Beobachtungen mit Clusterlabel c' in Cluster c darstellen und n_c die Anzahl der Beobachtungen in Cluster c. Die Entropie der gesamten Clusterlösung ergibt sich dann als mit der Größe der Cluster gewichtete Summe der Werte für die Entropie der einzelnen Cluster:

$$H(P) = \sum_{c \in \mathcal{P}'} H(c) \frac{N_c}{N}\,. \tag{6.14}$$

Der Wertebereich der Entropie liegt zwischen 0 und $\log_2 K$, wobei K die Anzahl der Cluster darstellt. Ein Wert von 0 würde entsprechend bedeuten, dass keinerlei Unsicherheit über die Clusterlösung besteht und ein perfektes Clustering vorliegt. Ein Problem der Entropie ist, dass der Wert mit der Anzahl der Cluster steigt.

6.2.4. Normalized Mutual Information

Mutual Information geht in seiner ursprünglichen Form ebenfalls auf Shannon (1948) zurück und ist mit Entropie verwandt. Um Mutual Information darzustellen, muss zunächst der Begriff der *relativen Entropie* erklärt werden. Die relative Entropie kann als Distanz zwischen zwei Verteilungen p und q begriffen werden (Cover & Thomas, 2006, S. 19). Formell ist die relative Entropie wie folgt definiert:

$$D(p\|q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}\,. \tag{6.15}$$

Ähnlich wie Entropie beschreibt Mutual Information den Informationsgehalt einer Zufallsvariable. Es wird der Informationsgehalt der Zufallsvariable x bei Kenntnis der Zufallsvariable y beschrieben (Cover & Thomas, 2006, S. 19). Berechnet wird Mutual Information als die relative Entropie der gemeinsamen Verteilung $p(x, y)$ der beiden Zufallsvariablen x und y und dem Produkt der beiden Randverteilungen $p(x)$ und $p(y)$:

$$I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \,. \tag{6.16}$$

Um diesen Wert nun als Bewertungskriterium für Clusterlösungen verwenden zu können, muss die Formel entsprechend angepasst werden (Manning et al., 2008, S. 329):

$$I(\mathcal{P};\mathcal{P}') = \sum_{k} \sum_{j} P\left(c_k \cap c'_j\right) \log \frac{P\left(c_k \cap c'_j\right)}{P\left(c_k\right) P\left(c'_j\right)} \,. \tag{6.17}$$

Ähnlich wie bei den vorangegangenen Kriterien wird auch bei Mutual Information die Anzahl der Cluster in der Clusterlösung nicht adäquat berücksichtigt. So resultiert für ein Clustering der gleiche Wert für Mutual Information, wenn die Clusterlösung in Subcluster zerlegt wird (Manning et al., 2008, S. 330). Dadurch würden zwar immer noch ähnliche Objekte zusammen in Clustern gruppiert, jedoch stellt ein einzelner Cluster, aufgespaltet in mehrere Subcluster, einen Informationsverlust dar und sollte idealerweise von einem Beurteilungskriterium entsprechend berücksichtigt werden.

Der Vorteil von Mutual Information gegenüber den vorangegangenen Beurteilungskriterien ist, dass es durch die halbierte Summe der Entropie der Clusterlösung $H(\mathcal{P})$ und des Set der Clusterlabel $H(\mathcal{P}')$ dividiert und somit auf einen Wertebereich zwischen 0 und 1 normiert werden kann. Dies erlaubt auch auf adäquate Art und Weise Clusterlösungen mit unterschiedlicher Anzahl von Clustern gegenüber dem Set der bekannten Clusterlabel zu bewerten (Manning et al., 2008, S. 330). Somit ergibt sich *Normalized Mutual Information* (NMI) als

$$\mathrm{NMI}(\mathcal{P},\mathcal{P}') = \frac{I(\mathcal{P};\mathcal{P}')}{[H(\mathcal{P}) + H(\mathcal{P}')]/2} \,. \tag{6.18}$$

6.3. Nicht verwendete externe Beurteilungskriterien

In diesem Abschnitt werden Beurteilungskriterien dargestellt, welche theoretisch verwendet werden könnten, aber von denen aus verschiedenen Gründen Abstand genommen wird.

6.3.1. Kophenetischer Korrelationskoeffizient

Der kophenetische Korrelationskoeffizient ist geeignet unterschiedliche Clusterlösungen miteinander zu vergleichen (Handl & Kuhlenkasper, 2017, S. 450). Gebildet wird dabei die Pearson-Korrelation zwischen den Nebendiagonaleinträgen der ursprünglichen Distanzmatrix D und der kophenetischen Matrix D^*. Die Einträge dieser kophenetischen Matrix D^* bestehen aus den Distanzen, bei denen die Cluster zu denen die entsprechenden Elemente gehören zusammengefasst wurden und aus dem Dendrogramm der Clusteranalyse ablesbar sind (Handl & Kuhlenkasper, 2017, S. 419). Entsprechend ist eine Distanzmatrix vonnöten um dieses Gütekriterium zu berechnen. Da dies zu aufwendig für die Größe der verwendeten Datensätze in dieser Arbeit ist, wird der kophenetische Korrelationskoeffizient nicht verwendet.

6.3.2. Accuracy, Specificity, False Positive Rate

Weit verbreitet sind die Beurteilungskriterien *Accuracy*, *Specificity* und die *False Positive Rate*. Sie berechnen sich wie folgt:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{6.19}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{6.20}$$

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{TN} + \text{FP}}\,. \tag{6.21}$$

Accuracy stellt dabei den Anteil korrekt klassifizierter Beobachtungspaare dar. Specificity ist der Anteil korrekt klassifizierter Beobachtungspaare, die nicht zusammengehören und die False Positive Rate entsprechend der Anteil korrekt klassifizierter Beobachtungspaare die zusammengehören. Alle drei Maßzahlen sind jedoch nur sinnvoll anzuwenden, wenn der Anteil von True Positives und True Negatives etwa ausgeglichen ist (Christen,

2012, S. 166–169). Jedoch dominiert in vielen Anwendungen, so auch hier, der Anteil von True Negatives, weshalb diese Beurteilungskriterien hier nicht verwendet werden.

6.4. Interne Beurteilungskriterien

Folgend werden interne Beurteilungskriterien aufgezählt, welche die Güte einer Clusterlösung bewerten können, ohne die wahren Labels des Datensatzes zu kennen. Da in dieser Arbeit jedoch explizit der Vergleich der resultierenden Clusterlösungen mit der tatsächlichen in diesem Falle bekannten tatsächlichen Clusterlösung getätigt wird, werden die hier aufgezählten internen Beurteilungskriterien nicht verwendet.

6.4.1. Interclusterheterogenität und Intraclusterhomogenität

Saxena et al. (2017) nennen einige Beurteilungskriterien, die die Interclusterheterogenität und Intraclusterhomogenität messen. Sie zählen die Maßzahlen *Sum of Squared Error*, *Scatter Criteria*, *Condorcet's Criterion* und *C-criterion* auf[4], die sich wie folgt berechnen lassen:

$$\text{Sum of Squared Error} = \sum_{k=1}^{K} \sum_{\forall x_i \in C_k} \|x_i - \mu_k\|^2 \,, \tag{6.22}$$

dabei stellt C_k die Anzahl der Beobachtungen in Cluster k dar und μ_k den Vektormittelwert in Cluster k.

$$\text{Scatter Criteria}_k = \sum_{x \in C_k} (x - \mu_k)(x - \mu_k)^T \,. \tag{6.23}$$

$$\text{Condorcet's Criterion} = \sum_{C_i \in C} \sum_{\substack{x_j, x_k \in C_i \\ x_j \neq x_k}} s(x_j, x_k) + \sum_{C_i \in C} \sum_{x_j \in C_i; x_k \notin C_i} d(x_j, x_k) \,. \tag{6.24}$$

wobei $s(x_j, x_k)$ und $d(x_j, x_k)$ die Ähnlichkeit bzw. Distanz zwischen den Vektoren x_j und x_k darstellen.

[4]Sie nennen auch noch *Category Utility Metric* und *Edge Cut Metrics*, die jedoch im Kontext dieser Arbeit nicht relevant sind.

$$\text{C-criterion} = \sum_{C_i \in C} \sum_{\substack{x_j, x_k \in C_i \\ x_j \neq x_k}} \left(s\left(x_j, x_k\right) - \gamma\right) + \sum_{C_i \in C} \sum_{x_j \in C_i; x_k \notin C_i} \left(\gamma - s\left(x_j, x_k\right)\right) , \qquad (6.25)$$

mit y als ein zu definierender Schwellenwert.

6.4.2. Silhouetten

Um die Performance der Clusteralgorithmen auch grafisch zu bewerten, eignen sich neben den üblichen Dendrogrammen vor allem Silhouettenplots (Rousseeuw, 1987). Dabei wird für jede Beobachtung der Wert s_i berechnet, welche dann abfallend Cluster für Cluster jeweils in Form einer Linie, dessen Länge s_i entspricht, untereinander abgetragen werden. Um s_i zu berechnen, wird zunächst der Wert a_i berechnet, der die mittlere Distanz von Beobachtung i zu allen anderen Beobachtungen des Clusters A berechnet, zu denen auch i gehört. Anschließend werden auch die mittleren Distanzen $d_{i,C}$ zu jeweils allen anderen Clustern $C \neq A$ berechnet, womit dann mit b_i das Minimum dieser Distanzen $d_{i,C}$ beschrieben wird. s_i ergibt sich dann als

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}} . \qquad (6.26)$$

s_i kann Werte zwischen -1 und 1 annehmen. Wobei ein Wert nahe 1 eine gute Klassifizierung der Beobachtung darstellt. Ein Wert nahe 0 bedeutet, dass es nicht klar ist, ob die Beobachtung zum zugewiesenen Cluster gehört oder zum nächsten Nachbarcluster. Ein Wert nahe -1 kann so interpretiert werden, dass die entsprechende Beobachtung vermutlich fehlklassifiziert ist (Kaufman & Rousseeuw, 2005, S. 85–86). Zusätzlich können auch noch Kennwerte aus den einzelnen s_i gebildet werden: Zum einen das arithmetische Mittel von s_i innerhalb eines Clusters, als auch das arithmetische Mittel von s_i für den gesamten Datensatz, was somit einen Eindruck von der Güte der Clusterlösung sowohl für einzelne Cluster, als auch für den gesamten Datensatz verschaffen kann.

6.5. Gini-Koeffizient

Neben den Beurteilungskriterien die die Güte der Clusterlösungen direkt evaluieren, wird auch ein Kriterium benötigt, das die Konzentration der Beobachtungen auf die verschie-

denen Cluster beschreibt. Damit kann evaluiert werden, ob der jeweilige Algorithmus die Beobachtungen gleichmäßig auf die Cluster verteilt oder dazu neigt schiefe Verteilungen der Clustergrößen zu produzieren, indem viele Beobachtungen in einen einzelnen Cluster allokiert werden und nur wenige in die restlichen Cluster. Ein solches Maß stellt der Gini-Koeffizient dar.

Um den Gini-Koeffizient zu beschreiben muss zunächst die Lorenzkurve erläutert werden, auf welcher der Gini-Koeffizient aufbaut. Dabei wird eine Linie entlang der Punkte $(0,0),(u_1,v_1),\ldots,(u_n,v_n)=(1,1)$ abgetragen. Wobei $u_j = j/n$ gleich dem Anteil der Merkmalsträger entspricht und $v_j = \sum_{i=1}^{j} x_i / \sum_{i=1}^{n} x_i$ der kumulierten relativen Merkmalssumme (Fahrmeir et al., 2016, S. 73). Übertragen auf die Problemstellung dieser Arbeit entspricht der u_j dem Anteil der Cluster und v_j der kumulierten relativen Summe der Beobachtungen. Vorher müssen die Cluster der Größe nach geordnet werden. Bei einer Clusterlösung in der alle Cluster die gleiche Anzahl an Beobachtungen aufweist, liegt diese Kurve dann komplett auf der Winkelhalbierenden. Je weiter sich die Kurve von der Winkelhalbierenden entfernt, desto mehr Beobachtungen sind in nur wenigen Clustern konzentriert.

Um diese grafische Darstellungsweise der Lorenzkurve formell auszudrücken, kann der Gini-Koeffizient verwendet werden (Fahrmeir et al., 2016, S. 76–78). Dieser wird gebildet, indem die Fläche zwischen Diagonale und Lorenzkurve durch die Fläche zwischen Diagonale und u-Achse geteilt wird, was gleichbedeutend zur doppelten Fläche zwischen Diagonale und Lorenzkurve ist:

$$G = \frac{2\sum_{i=1}^{n} i x_i}{n\sum_{i=1}^{n} x_i} - \frac{n+1}{n}\,. \tag{6.27}$$

Da die maximale Ausprägung des Gini-Koeffizienten nun mit $\frac{n-1}{n}$ gegeben ist, eine Spannweite zwischen 0 und 1 aber besser interpretiert werden kann, wird der normierte Gini-Koeffizient verwendet:

$$G^* = \frac{G}{G_{\max}} = \frac{n}{n-1} G\,. \tag{6.28}$$

Der Koeffizient nimmt den Wert 0 an, wenn alle Cluster die gleiche Anzahl an Beobachtungen aufweisen und er nimmt den Wert 1 an, wenn sämtliche Beobachtungen in einem Cluster zu finden sind.

7. Clusteralgorithmen

In diesem Abschnitt werden die gängigen Algorithmen dargestellt, welche sich für das Clustering hochdimensionaler binärer Daten eignen. Dabei wird auf Algorithmen eingegangen, welche sich bereits für das Clustering binärer Daten bewährt haben und explizit dafür entworfen wurden. Es werden aber auch Algorithmen herangezogen, welche ursprünglich für andere Anwendungen entwickelt wurden, wie zum Beispiel das Blocking beim Privacy Preserving Record Linkage mittels Bloom Filter (Schnell et al., 2009). Aufgrund der Tatsache, dass es sich dabei ebenfalls um das Einteilen ähnlicher, in diesem Falle Records handelt, deren einzelne Werte ebenfalls aus 0 und 1 bestehen, soll im Rahmen dieser Arbeit geprüft werde, inwiefern sich diese Algorithmen auch für das Clustering binärer Daten eignen.

7.1. Parallelisierung der Algorithmen

Aufgrund der Tatsache, dass es sich bei der verwendeten simulierten Datenbasis dieser Arbeit um Daten mit sehr vielen Beobachtungen handelt, ist eine Ausführung der Clusteralgorithmen in paralleler Art und Weise unverzichtbar. Die wenigsten Schritte in der Clusteranalyse können als *embarrassingly parallel*[1] beschrieben werden. Daher ist ein möglicher Ansatz dennoch Algorithmen parallel auszuführen der sogenannte *Divide and Conquer*-Ansatz (Xu & Wunsch, 2009, S. 227–228). Dabei werden die Beobachtungen des Datensatz im ersten Schritt in Teilmengen aufgeteilt. In dieser Arbeit werden dabei so viele Teilmengen gebildet, wie Prozessorkerne verwendet werden. Jede dieser Teilmengen wird dann von einem Prozessorkern bearbeitet, indem der eigentliche Clusteralgorithmus ausgeführt wird und so eine lokale Clusterlösung der jeweiligen Teilmenge des Datensatzes gebildet wird. Xu und Wunsch (2009, S. 228) nach wird dann im nächsten Schritt jeweils ein Repräsentant der jeweiligen Cluster der einzelnen lokalen Clusterlösungen gewählt und diese wiederum einer finalen Clusteranalyse unterzogen. In dieser Arbeit wird dafür

[1] Als embarrassingly parallel werden Algorithmen beschrieben, die ein Problem in einzelne Teile aufteilen und parallel bearbeiten können, ohne das die einzelne Prozesse miteinander kommunizieren müssen und ein geringer Overhead besteht (Matloff, 2016, S. 33–34).

der jeweilige Medoid[2] verwendet und mittels K-Means geclustert. Schließlich werden im letzten Schritt die einzelnen Beobachtungen der ursprünglichen lokalen Clusterlösungen den finalen Clustern zugewiesen, die die Medoids der ursprünglichen Cluster enthalten.

Wenn nicht anders beschrieben, wird dieses Vorgehen für alle Algorithmen verwendet, für die keine Parallelisierungen veröffentlicht sind oder die keine Besonderheiten aufweisen, welche für die Parallelisierung zunutze gemacht werden können, wie es z.B. bei *Proximus* der Fall ist (siehe Abschnitt 7.12). Durch die Bildung lokaler Clusterlösungen, kann jedoch keine optimale Clusterlösung garantiert werden, da für die einzelnen lokalen Clusterlösungen nicht das volle Datenmaterial zur Verfügung steht. Dass das Vorgehen dennoch angemessen ist und der Geschwindigkeitsvorteil der Methode keinen schwerwiegenden Verlust der Clustergüte mit sich bringt, wird empirisch in Abschnitt 9.2 gezeigt.

7.2. Distanzen für binäre Daten

Bevor die Clusteralgorithmen vorgestellt werden, muss zunächst evaluiert werden, welche verschiedenen Metriken für die Distanzberechnung zwischen binären Vektoren verfügbar sind. So kann es zu unterschiedlichen Ergebnissen kommen, wenn die zugrundeliegende Distanzmetrik geändert wird.

Sämtliche Distanzmetriken für binäre Daten basieren auf einer Vierfelder-Kontingenztabelle mit den Indizes a, b, c, d, die wie folgt gefüllt wird:

$$a = \sum_{i=1}^{d} x_i y_i \tag{7.1}$$

$$b = \sum_{i=1}^{d} (1 - x_i)\, y_i \tag{7.2}$$

$$c = \sum_{i=1}^{d} x_i\, (1 - y_i) \tag{7.3}$$

$$d = \sum_{i=1}^{d} (1 - x_i)\, (1 - y_i) \tag{7.4}$$

[2]Zur Definition siehe Abschnitt 7.4.

mit d gleich der Anzahl der Variablen und x und y gleich zweier binärer Vektoren mit Ausprägungen 0 und 1, zwischen denen die Distanz berechnet werden soll. Die Vierfelder-Kontingenztabelle hat demnach folgende Form:

Tabelle 7.1.: A ist die Anzahl der Variablen, in denen sowohl x als auch y eine 1 haben. B ist die Anzahl der Variablen, in denen x eine 1 hat und y eine 0. C ist der umgekehrte Fall. D ist die Anzahl der Variablen, in denen sowohl x als auch y eine 0 haben.

		y: 1	y: 0
x	1	a	b
	0	c	d

Für binäre Daten muss zwischen 2 Familien von Distanzmetriken unterschieden werden. Zum einen gibt es symmetrische Metriken und zum anderen asymmetrische Metriken (Kaufman & Rousseeuw, 2005, S. 23). Diese Unterscheidung basiert auf der Beschaffenheit der Variablen selbst. Von Symmetrie wird in diesem Kontext gesprochen, wenn die beiden Ausprägungen innerhalb der Variable die gleiche Wertigkeit besitzen und lediglich 2 unterschiedliche Ausprägungen eines Merkmals darstellen, wie zum Beispiel „rechts“ und „links“. Asymmetrie bedeutet meist, dass ein Merkmal prävalent ist oder nicht, wie zum Beispiel „rote Haare“ oder „keine roten Haare“. Symmetrische Distanzmetriken berechnen sich daher unter Berücksichtigung aller Felder aus genannter Kontingenztabelle, während asymmetrische Metriken das Feld d nicht verwenden. Dieser Umstand kommt daher, dass sich die Verteilungen symmetrischer und asymmetrischer Merkmale in der Regel unterscheiden. So zeichnen sich asymmetrische Merkmale wie zum Beispiel Warenkörbe (Produkt gekauft/Produkt nicht gekauft) dadurch aus, viele Nullen und nur wenige Einsen aufzuweisen, was zu einer hohen Zellbesetzung von d führen würde und damit zu einer hohen Ähnlichkeit zweier Beobachtungen, deren Feld a so gut wie leer oder gar komplett leer ist.

Kaufman und Rousseeuw (2005, S. 26) listen folgende Distanzmetriken für symmetrische Merkmale. Den *Simple-Matching-Koeffizienten* (Zubin, 1938), welcher die gängigste Metrik für symmetrische Merkmale darstellt:

$$S_{SM}(x_i, x_j) = \frac{a+d}{a+b+c+d}. \tag{7.5}$$

Er berechnet sich einfach durch den Anteil der Positionen der beiden binären Vektoren x_i und x_j, die übereinstimmen. Darüber hinaus wird noch die *Rogers-Tanimoto-* (Rogers & Tanimoto, 1960) und die *Sokal-Sneath-Metrik* (Sokal & Sneath, 1963) genannt:

$$S_{RT}\left(x_i, x_j\right) = \frac{a+d}{(a+b)+2\left(c+d\right)}\,, \tag{7.6}$$

$$S_{SS_{sym}}\left(x_i, x_j\right) = \frac{2\left(a+d\right)}{2\left(a+d\right)+\left(c+d\right)}\,. \tag{7.7}$$

Diese beiden Metriken unterscheiden sich lediglich durch die unterschiedliche Gewichtung der Übereinstimmung der beiden Vektoren bzw. nicht Übereinstimmung. Ein weiterer weit verbreiteter Vertreter dieser Familie von Metriken ist die *Hamming-Distanz*, die sich durch den Anteil der nicht Übereinstimmungen der beiden zu vergleichenden Vektoren ergibt (Aggarwal, 2014, S. 282). Entsprechend stellt die Hamming-Distanz den Gegenpart zum Simple-Matching-Koeffizienten dar und lässt sich auch mit $1-S_{SM}\left(x_i, x_j\right)$ berechnen (Xu & Wunsch, 2009, S. 27).

Die wichtigsten Metriken[3] für die Verwendung bei asymmetrischen Merkmalen sind vor allem der *Jaccard-Koeffizient* (Jaccard, 1908), der *Dice-Koeffizient* (Dice, 1945) und die Metrik von *Sokal und Sneath* (Sokal & Sneath, 1963):

$$S_J\left(x_i, x_j\right) = \frac{a}{a+b+c}\,, \tag{7.8}$$

$$S_D\left(x_i, x_j\right) = \frac{2a}{2a+b+c}\,, \tag{7.9}$$

$$S_{SS_{asym}}\left(x_i, x_j\right) = \frac{a}{a+2\left(b+c\right)}\,. \tag{7.10}$$

[3]Eine Reihe weiterer auch weniger verbreiteter Metriken auf die hier nicht weiter eingegangen werden soll, findet sich bspw. bei Gan et al. (2007, S. 78–79).

Sämtliche der hier vorgestellten Metriken lassen sich sowohl als Ähnlichkeitsmaß, als auch als Distanzmaß verwenden, wenn der Wert der verwendeten Metrik entsprechend von 1 subtrahiert wird. Grundsätzlich sollte die Wahl des verwendeten Distanzmaß in Abhängigkeit des Untersuchungsgegenstandes und der Beschaffenheit der vorliegenden Daten individuell getroffen werden. Da im Rahmen dieser Arbeit sowohl Daten mit hoher als auch niedriger Prävalenz an Einsen verwendet werden, die Daten simuliert und keinem Untersuchungsgegenstand direkt zugeordnet werden können und es daher nicht klar ist, ob es sich um symmetrische oder asymmetrische Merkmale handelt, wird in dieser Arbeit, wenn nicht anders angegeben, die Hamming-Distanz verwendet.

7.3. Locality Sensitive Hashing

Eine der Hauptverwendungszwecke und Motivation des *Locality Sensitive Hashing* (Indyk & Motwani, 1998) stellt die Suche nach ähnlichen Objekten innerhalb eines (sehr) großen Datensatzes dar. In der Literatur wird als Beispiel häufig das Finden ähnlicher Dokumente in einer Vielzahl von Dokumenten (etwa Webseiten) oder das Identifizieren ähnlicher Kunden von Onlinehändlern angegeben (Leskovec et al., 2011).

Die Kerngedanken von Locality Sensitive Hashing bestehen aus 2 Konzepten, welche zum einen den Speicherbedarf gering halten und zum anderen die Zeitkomponente beim Identifizieren ähnlicher Einträge in der zugrunde liegenden Datenmatrix optimieren. Der Speicherbedarf wird dadurch reduziert, dass statt der kompletten Datenmatrix die sogenannte *Signature-Matrix* der originalen Datenmatrix verwendet wird, welche die originale Datenmatrix in verdichteter Form abbildet. Die Ähnlichkeit dieser verdichteten Einträge der Signature-Matrix approximieren dabei die Ähnlichkeit der original Datenmatrix. Diese Ähnlichkeit wird dabei üblicherweise durch die Jaccard-Similarity ausgedrückt (siehe Abschnitt 7.2).

Die Zeitkomponente wiederum wird optimiert, indem Teilstücke der Einträge der Signature-Matrix via effizienter Hashing-Funktionen in verschiedene Körbe verteilt werden, wobei ähnliche Teilstücke idealerweise im selben Korb landen.

Grundsätzlich kann eine Familie von Funktionen als *Locality Sensitive* definiert werden, wenn die Wahrscheinlichkeit, dass ein Beobachtungspaar im selben Korb landet, mindestens p_1 ist, sollte die Ähnlichkeit zwischen den Beobachtungen höchstens r_1 sein.

Gleichzeitig nimmt die Wahrscheinlichkeit im selben Korb zu landen höchstens den Wert p_2 an, sollte die Ähnlichkeit zwischen den beiden Beobachtungen größer als r_2 sein (Indyk & Motwani, 1998, S. 609). Diese Familie von Funktionen wird dann auch als (r_1, r_2, p_1, p_2)-sensitiv bezeichnet. Als Familie von Funktionen F wird beispielsweise die Menge der verschiedenen Permutationen des Minhashing (siehe folgenden Abschnitt) bezeichnet, wo jede Hashfunktion $f \in F$ Teil dieser Familie von Funktionen ist (Leskovec et al., 2011, S. 100–101). Sollte eine Familie von Funktionen locality sensitiv sein, führt das dazu, dass bei entsprechend gewählten Parametern, Beobachtungspaare welche nahe beieinander liegen mit hoher Wahrscheinlichkeit im selben Korb landen, während Beobachtungspaare, welche weit auseinander liegen, eine niedrige Wahrscheinlichkeit haben, im selben Korb zu landen (Das et al., 2007, S. 274).

7.3.1. MinHashing

Bevor die Signature-Matrix erstellt werden kann, erfolgt üblicherweise das Transformieren der Daten in eine binäre Struktur. Bei Dokumenten beispielsweise werden sogenannte *shingles* oder auch *q-grams* gebildet. Je nach Wahl von q sind dies aufeinanderfolgende Zeichenketten mit Länge q innerhalb des Dokuments. Die Dokumente können so als binäre Vektoren dargestellt werden, welche eine 1 erhalten, sollte ein entsprechendes q-gram im Dokument enthalten sein und eine 0, wenn dem nicht so ist. Die Länge der Vektoren wird dann durch die Anzahl aller in den Dokumenten, welche es zu vergleichen gilt, vorkommenden einzigartigen q-grams bestimmt. Bei dem Vergleich von Kunden in Supermärkten oder Onlinehändlern wäre dies entsprechend das Kaufen eines bestimmten Artikels aus der Anzahl aller möglichen Artikel bzw. das Nicht-Kaufen. Daten die bereits in binärer Form vorliegen, können direkt verwendet werden.

Um nun die Signature-Matrix zu erstellen, wird das Prinzip des *Min-wise independent permutations* oder auch *MinHash* (Broder, 1997) verwendet. Gegeben die Ausgangsmatrix ist so angeordnet, dass jede Spalte eine Beobachtung darstellt und die Zeilen die Merkmale, werden die Zeilen der Matrix zunächst indiziert und dann permutiert. Anschließend wird geprüft in welcher der nun permutierten Zeile die erste Ausprägung mit einer 1 beobachtet werden kann. Der entsprechende, permutierte, Zeilenindex stellt dann das erste Element der jeweiligen betrachteten Beobachtung in der Signature-Matrix dar. Dieser Vorgang kann sooft wiederholt werden, wie die Signatures der Beobachtungen lang sein sollen. Der Grund für das Minhashing ist der, dass die Wahrscheinlichkeit, dass 2 unterschiedliche Beobachtungen den gleichen Minhashing-Wert zugewiesen bekommen

gleich der Jaccard-Similarity zwischen diesen beiden Beobachtungen ist (Leskovec et al., 2011, S. 82). Somit kann bei Reduktion des Speicherbedarfs die Ähnlichkeit der beiden Beobachtungen präserviert werden.

Das Permutieren von Matrizen ist jedoch nicht effizient. Der gleiche Effekt kann jedoch erzielt werden, indem die Zeilenindizes mittels einer arbiträren zufälligen Hashfunktion einem Korb, in Form einer Zahl innerhalb des Intervalls 0 bis zur Anzahl an Zeilen der Ausgangsmatrix, zugewiesen werden. Wenn die Signature einer Beobachtung die Länge n haben soll, müssen demnach n verschiedene zufällige Hashfunktionen verwendet werden, um die Zeilenindizes einem Korb zuzuweisen, welcher dann einen Eintrag in der Signature der Beobachtung darstellt (Leskovec et al., 2011, S. 83). In dieser Arbeit wird dafür *Universal Hashing* (Carter & Wegman, 1979) verwendet. Diese Hashfunktionen haben die formale Form:

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m\,. \tag{7.11}$$

Dabei stellen a und b zufällig ausgewählte Parameter aus dem Intervall $\{0, 1, \ldots, p-1\}$ respektive $\{1, 2, \ldots, p-1\}$ dar, mittels welcher eine lineare Transformation des zu hashenden Wertes k (in diesem Fall die Zeilenindezes) vorgenommen wird. Dabei p ist eine Primzahl größer als die Anzahl der zu hashenden Werte (in diesem Fall dem Wert des höchsten Zeilenindezes) und m begrenzt den Wertebereich der resultierenden Hashwerte (Cormen et al., 2009, S. 265–267). Für Minhashing nimmt m wiederum das Maximum der Zeilenindizes an. In der Praxis werden oftmals auch andere Hashing-Techniken verwendet, wie etwa MD5 mit einem zufälligen Seed. Da das Erstellen der Signature-Matrix ein Problem ist, welches als embarrassingly parallel beschrieben werden kann, wird dieser Prozess auf parallele Art und Weise ausgeführt.

Aufgrund der Tatsache, dass MinHash Beobachtungen zusammen gruppiert, welche bezüglich ihrer Ähnlichkeit einen gewissen Schwellenwert überschreiten und Beobachtungspaare eliminiert, welche diesen unterschreiten, kann MinHashing auch als Preclustering Algorithmus verwendet werden, was die nötigen Distanzberechnungen für einen elaborierteren Algorithmus verringert (Oprisa, 2015). Ähnlich wie auch Canopy Clustering (siehe Abschnitt 7.13) kann MinHash selbst prinzipiell schon zum Clustern von Daten verwendet werden. In diesem Fall werden dann Beobachtungen als ein Cluster betrachtet, welche die

selben Hashwerte in der Signature-Matrix aufweisen und damit als ähnlich gelten. Daher wird in dieser Arbeit Minhashing selbst ebenfalls als eigenständiger Clusteralgorithmus evaluiert.

7.3.2. Banding and Hashing

Um nun ähnliche Beobachtungen zu finden wird *Banding und Hashing* (Leskovec et al., 2011, S. 88) verwendet. Dazu wird zunächst die Signature-Matrix in mehrere vertikale Blöcke (sogenannte *Bands*) geteilt. Jedes dieser Bands enthält dieselbe Anzahl an Zeilen. Teilen nun die einzelnen Beobachtungen innerhalb eines Bands die identischen Ausprägungen der Signatures, werden sie von einer Hashingfunktion demselben Korb zugewiesen. Unterschiedliche Signatures innerhalb eines Bands werden in unterschiedliche Körbe verteilt. Jedes Band hat dabei eine andere Hashingfunktion, sodass identische Teile der Signature in unterschiedlichen Bands nicht denselben Korb zugewiesen bekommen. Der Gedanke dahinter ist, dass ähnliche Beobachtungen in mindestens einem Band identische Signatures haben und unähnliche Beobachtungen nicht. Üblicherweise werden 2 Beobachtungen als ähnlich betrachtet, wenn sie wenigstens in einem Band in demselben Korb lokalisiert sind. Auch das Banding und Hashing ist ein Problem, welches als embarrassingly parallel beschrieben werden kann und daher auf parallele Art und Weise ausgeführt wird.

7.3.3. Evaluierung von LSH

Es ist leider nicht möglich mit Hilfe von Locality Sensitive Hashing die Anzahl möglicher Cluster zu identifizieren. Ebenfalls ist es nicht möglich die Anzahl der Cluster als Argument an die Methode zu übergeben und entsprechende Clusterzugehörigkeiten festzustellen. Aufgrund dessen ist es nicht möglich den Rand Index, die Entropie oder NMI zu berechnen. Jedoch ist es möglich True Positives, False Positive, True Negatives und False Negatives zu berechnen, was die Verwendung von Recall und Precision und den F-Score erlaubt.

Als True Positive werden dabei Beobachtungen klassifiziert, wenn sie aus demselben Cluster stammen und vom Algorithmus als zusammengehörig identifiziert wurden. Ein False Positive stellen Paare dar, welche als ähnlich klassifiziert wurden, aber aus unterschiedlichen Clustern stammen. True Negatives sind definiert als ein Beobachtungspaar,

welches nicht als ähnlich identifiziert wurde und korrekterweise auch aus unterschiedlichen Clustern stammt. Diese lassen sich aus der Differenz der Anzahl aller möglichen Paare aus unterschiedlichen Clustern und den False Positives errechnen, ohne wirklich jedes Paar aus unterschiedlichen Clustern mit dem LSH Ergebnis vergleichen zu müssen, was extrem rechenaufwändig und somit nicht umsetzbar wäre. Analog dazu verhält es sich mit False Negatives, was Beobachtungen aus denselben Cluster wären, welche nicht als ähnlich klassifiziert wurden.

Eine weitere Problematik besteht in der Komplexität der Berechnung der Klassifizierungsergebnisse. Ein True Positive zum Beispiel ist gegeben, wenn 2 Beobachtungen innerhalb mindestens eines Bandes in denselben Korb gehasht werden. Um nun auszuschließen, dass dasselbe Paar nicht auch in einem anderen Band in einen Korb gehasht wurde und bei einer einfachen Auszählung aller Paare mehrfach gezählt wird, müsste jeder Korb mit jeweils jedem anderen Korb der restlichen Bands verglichen werden. Bei sehr großen Datensätzen mit sehr großer Anzahl an Körben und Wertepaaren ist dies nicht mehr effizient umsetzbar. Daher wird in dieser Arbeit zur Bewertung eine Stichprobe der Beobachtungen gezogen und nur die Ergebnisse für diese bewertet. Um zu überprüfen, dass dies eine vertretbare Herangehensweise ist, wurde eine Simulation durchgeführt. Es wurden sämtliche Datensätze simuliert, die auch bei der kompletten Simulation in dieser Arbeit genutzt werden. Diese wurden jedoch auf 10.000 Beobachtungen[4] begrenzt, um einmal die kompletten Ergebnisse zu evaluieren und anschließend mit den Ergebnissen der Stichprobe vergleichen zu können.

Eine ähnliches Hindernis kann bei der Evaluation der Ergebnisse des MinHash-Algorithmus auftreten. Es ist hier möglich, dass es zu viele Beobachtungen gibt, die sich nicht ähnlich genug sind, um denselben Hashwert zugewiesen zu bekommen und es daher zu zu vielen Clustern kommen kann, um diese mit akzeptablen Zeitaufwand evaluieren zu können. Aus diesem Grund wird bei diesem Algorithmus ebenfalls eine Stichprobe der Clusterlösung gezogen, sollte die Anzahl der Cluster den Wert 10.000 überschreiten. Um zu überprüfen, ob auch in diesem Fall dieses Vorgehen adäquat ist und sich die Ergebnisse der Evaluation der kompletten Clusterlösung mit der einer Stichprobe

[4]Bei Datensätzen der Größe $n > 10.000$ kam es bereits zu Abbrüchen des Evaluationsprogramms, da die intern verwendete Funktion `pairsfun`, welche verwendet wird um alle möglichen Paare einer Menge von Objekten zu bilden, einen Vektor größer 2^{32} Einträge zurückgeben würde, was die maximal zulässige Vektorlänge von `R` darstellt. Prinzipiell kann `R` auch mit längeren Vektoren in Form von `Long Vectors` umgehen, die Problematik der zu hohen Komplexität bleibt jedoch bestehen.

vergleichen lässt, wurde die gleiche Simulation durchgeführt, die im vorangegangenen Absatz beschrieben wurde. Aufgrund der Tatsache, dass bei MinHash die Evaluation erst bei größeren Datensätzen problematisch werden kann, konnten hier größere Datensätze mit Umfang $n = 100.000$ gewählt werden. Bei den Stichproben handelt es sich sowohl beim LSH als auch bei MinHash um Stichproben der Größe $n = 2.000$, die geschichtet nach der bekannten Clusterzugehörigkeit gezogen wurden.

Tabelle 7.2.: Ergebnisse des Vergleichs der Evaluation mit LSH und MinHash

Methode	Statistik	Recall	Precision	F-Score
LSH	Mittelwert	0	0	0
	Standardabweichung	0,01	0	0
MinHash	Mittelwert	0,01	0	0,01
	Standardabweichung	0,01	0,01	0,01

In Tabelle 7.2 finden sich die Ergebnisse der Simulation. Es sind die Mittelwerte und die Standardabweichung der Differenzen zwischen der Evaluation des kompletten Datenmaterials und der Evaluation basierend auf der Stichprobe bezüglich Recall, Precision und des F-Scores über alle simulierten Datensätze dargestellt. Wie sich sehen lässt, sind die Unterschiede zwischen der Evaluation des gesamten Ergebnismaterials und der Stichprobe der Ergebnisse bei beiden Algorithmen marginal. Das Bewerten des Algorithmus auf Grundlage einer Stichprobe scheint daher in Anbetracht der anderweitig enormen Komplexität angebracht.

7.4. k-Medoids

In diesem Abschnitt werden die drei Algorithmen *PAM* (Partitioning Around Medoids, Kaufman & Rousseeuw, 2005), *CLARA* (Clustering LARge Applications, Kaufman & Rousseeuw, 2005) und *CLARANS* (Clustering Large Applications based on RANdomized Search, Ng & Han, 2002) beschrieben, welchen allen dasselbe Prinzip zugrunde liegt. Dieses Prinzip lässt sich so beschreiben, dass k Cluster jeweils durch ein zentrales Objekt (Medoid) innerhalb des jeweiligen Cluster repräsentiert werden und die restlichen Objekte aufgrund ihrer Distanz zu den Medoids, dem entsprechenden Cluster zugeordnet werden. Die beiden Algorithmen CLARA und CLARANS basieren dabei auf der Idee von PAM, sind aber für die Analyse großer Datenmengen entwickelt worden und sind

daher effizienter, dafür aber auch ungenauer. Der Algorithmus PAM selbst ist aufgrund der hohen Rechenzeit nicht für das Clustering großer Datensätze geeignet, wird aber dennoch in diesem Abschnitt beschrieben, da er, wie bereits erwähnt, die Grundlage für die beiden verwendeten Algorithmen CLARA und CLARANS bildet. Ebenfalls ist allen Algorithmen gemein, dass die Anzahl der erwarteten Cluster k als Parameter im Vorhinein an den Algorithmus übergeben werden müssen.

7.4.1. PAM - Partitioning Around Medoids

Wie bereits in Abschnitt 7.4 angedeutet, besteht die Aufgabe von PAM darin, k Beobachtungen (Medoids) zu finden, welche jeweils zentral im Cluster k liegen und damit als Repräsentant dieses Clusters beschrieben werden können. Der Cluster selbst bildet sich dann dadurch, dass jede Beobachtung dem Medoid zugewiesen wird, der die niedrigste Distanz zur jeweiligen Beobachtung aufweist. Grundsätzlich funktioniert der Algorithmus mit jedem gängigen Distanzmaß, welches den Daten angemessen ist. Im Falle binärer Daten eignet sich die Jaccard-Distanz, welche in Abschnitt 7.3 beschrieben ist. Das Ziel des Algorithmus ist demnach k Medoids zu finden, welche die zentralste Position innerhalb des Clusters einnehmen.

Um die idealen k Medoids zu finden, verwendet der Algorithmus 2 Phasen: die *BUILD*-Phase und die *SWAP*-Phase (Kaufman & Rousseeuw, 2005, S. 102–104). In der *BUILD*-Phase werden zunächst k initiale Medoids gesucht. Zunächst wird ein einzelner Medoid gewählt, welcher die Zielfunktion minimiert. Die Zielfunktion ist definiert als die Summe der Distanzen aller Objekte im Datensatz zum jeweils nächsten Medoid. Anschließend werden sukzessiv weitere Medoids gesucht, bis k Medoids gefunden wurden und eine erste Clusterlösung erstellt wurde. Kaufman und Rousseeuw (2005, S. 102–103) beschreiben das Vorgehen des Algorithmus wie folgt:

1. Es wird zunächst eine Beobachtung i gewählt.

2. Es wird die Distanz D_j zwischen Beobachtung j und der vorherigen gewählten Beobachtung berechnet. Zusätzlich wird die Distanz $d(j, i)$ zwischen j und i berechnet.

3. Es wird der Gewinn C_{ji} für Beobachtung j berechnet, sollte i als Medoid gewählt werden:

$$C_{ji} = \max\left(D_j - d\left(j,i\right), 0\right) . \tag{7.12}$$

4. Schritt 3 wird für alle j ausgeführt und der totale Gewinn berechnet:

$$\sum_j C_{ji} . \tag{7.13}$$

5. Die Schritte 1 bis 4 werden für alle i wiederholt und es wird die Beobachtung i als Medoid gewählt, welche den höchsten Wert für $\sum_j C_{ji}$ aufweist.

Die Schritte 1 bis 5 werden solange wiederholt bis k Medoids gefunden wurden. Nach dieser *BUILD*-Phase folgt die *SWAP*-Phase, in welcher versucht wird, die Clusterlösung repräsentiert durch die in der *BUILD*-Phase initial gewählten k Medoids zu verbessern indem ein Medoid i durch eine andere Beobachtung h ersetzt wird (Kaufman & Rousseeuw, 2005, S. 103–104):

1. Es wird eine Beobachtung j gewählt und die Kosten C_{jih} berechnet, sollte Medoid i durch Beobachtung h ersetzt werden:
   a) Sollte die Distanz von j sowohl zu i (die Distanz von j zu seinem ursprünglich nächsten Medoid wird mit D_j gekennzeichnet) als auch zu h größer sein als zu einem anderen Medoid, folgt: $C_{jih} = 0$.
   b) Sollte j zu Medoid i gehören, muss zwischen 2 Fällen unterschieden werden:
      i. Sollte die Distanz $d\left(j,h\right)$ kleiner sein, als die Distanz zum zweitnächsten Medoid E_j, gestalten sich die Kosten C_{jih} als:

$$C_{jih} = d\left(j,h\right) - d\left(j,i\right) . \tag{7.14}$$

      ii. Sollte die Distanz $d\left(j,h\right)$ größer sein, als die Distanz zum zweitnächsten Medoid E_j, berechnen sich die Kosten C_{jih} als:

$$C_{jih} = E_j - D_j . \tag{7.15}$$

c) Sollte j näher an einem anderen Medoid liegen, als an i, aber noch näher an h, belaufen sich die Kosten für j bei einem Tausch von i gegen h als Medoid, zu:

$$C_{jih} = d(j,h) - D_j \,. \tag{7.16}$$

2. Die Gesamtkosten für den Tausch von i gegen h als Medoid berechnen sich dann aus der Summe von C_{jih}:

$$T_{ih} = \sum_j C_{jih} \,. \tag{7.17}$$

3. Der Prozess wird für jedes Paar (i,h) wiederholt und es wird das Paar gewählt, welches T_{ih} minimiert.

4. Sollte T_{ih} negativ sein, wird i gegen h getauscht. Sollte T_{ih} positiv oder 0 sein, ist der Algorithmus abgeschlossen.

Aufgrund der vielen Vergleiche, vor allem in der *SWAP*-Phase, wachsen die Anzahl der Berechnungen mit steigender Fallzahl stark an, weshalb der Algorithmus nur für kleinere Datensätze geeignet ist. Jedoch kann der Algorithmus angepasst werden, wobei die Grundidee k Medoids zu finden, welche die Cluster definieren, erhalten bleibt. Wie bereits angesprochen, handelt es sich bei diesen Algorithmen um CLARA und CLARANS, welche in den nächsten beiden Unterabschnitten behandelt werden.

7.4.2. CLARA - Clustering LARge Applications

CLARA (Kaufman & Rousseeuw, 2005) ist eine relativ simple Adaption von PAM, welche auf der Ziehung von Stichproben beruht. Kurz gesagt wird anstatt den gesamten Datensatz zu betrachten und den Algorithmus PAM durchzuführen, eine Stichprobe gezogen, PAM angewandt und sämtliche Beobachtungen des kompletten Datensatzes entsprechend der in der Stichprobe ermittelten k Medoids geclustert.

Durch die vorhandene Stichprobenvarianz entspricht diese Clusterlösung allerdings höchstwahrscheinlich nicht dem optimalen Clustering, weshalb dieser Prozess mehrfach durchgeführt wird. Kaufman und Rousseeuw (2005, S. 144) schlagen fünf Stichproben

vor. Nach jeder Iteration wird dabei die durchschnittliche Distanz aller Beobachtungen zu den jeweiligen Medoids in den entsprechenden Clustern berechnet. Die Clusterlösung bei der diese durchschnittliche Distanz den niedrigsten Wert annimmt, wird dann als die beste deklariert. Werden mehr Stichproben gezogen, verbessern sich auch die Chancen eine bessere Clusterlösung zu erhalten. Mit der gestiegenen Rechenleistung seit der Algorithmus entwickelt wurde ist es natürlich ratsam mehr als fünf Stichproben zu ziehen. Dies ist im Endeffekt eine Abwägung zwischen Rechenzeit und der Qualität der Ergebnisse.

Die Stichprobengröße geben Kaufman und Rousseeuw (2005, S. 145) mit $40 + 2k$ an und machen diese somit abhängig von der Anzahl der Cluster. Es liegt auf der Hand, dass mit einer höheren Stichprobengröße auch bessere Ergebnisse erzielt werden können. Jedoch wird aus Effizienzgründen nach wie vor an dem Richtwert $40 + 2k$ festgehalten (Schubert & Rousseeuw, 2019, S. 173).

7.4.3. CLARANS - Clustering Large Applications based on RANdomized Search

Der Algorithmus CLARANS (Clustering Large Applications based on RANdomized Search, Ng & Han, 2002) zeigt ebenfalls Parallelen zu PAM und CLARA auf, geht allerdings etwas anders vor. Ng und Han (2002, S. 1007) beschreiben den Algorithmus auf grafische Weise als Suche eines Knotenpunktes in einem Graphen. Jeder Knotenpunkt S stellt dabei eine Repräsentation von k Medoids und damit eine eindeutige Clusterlösung dar. Ein Nachbarknoten S_2 von Knoten S_1 ist dadurch definiert, dass die Kardinalität der Schnittmenge der repräsentierten Medoids mit $k - 1$ gegeben ist. Die beiden jeweiligen Sets der k Medoids unterscheiden sich somit lediglich in einem Element. Aufgrund der Tatsache, dass jeder Knotenpunkt einer Clusterlösung entspricht, ist das Ziel der Suche auf dem Graphen den Knotenpunkt zu finden, dessen Kostenfunktion minimiert ist. Im Falle von PAM wird dabei jeder Knotenpunkt betrachtet, die Kostendifferenz (siehe Formel 7.17) zum bisherigen besten Knoten berechnet und solange fortgefahren, bis das globale Minimum gefunden ist. Es werden also alle $k\,(n - k)$ Nachbarknoten betrachtet (was sehr kostspielig ist), der mit der höchsten Kostendifferenz herangezogen und dann wiederum von diesem alle Nachbarn betrachtet usw. CLARA geht ähnlich vor, jedoch wird durch das Ziehen der Stichproben, lediglich ein Subgraph betrachtet. Sollte das globale Minimum nicht Teil dieses Subgraphen sein, ist es nicht möglich dieses zu finden.

CLARANS wiederum geht so vor wie PAM, jedoch werden nicht alle Nachbarknoten betrachtet, sondern lediglich eine Stichprobe. Sollte sich innerhalb dieser Stichprobe an Nachbarknoten ein Knoten finden, dessen Kostendifferenz positiv ist, wird dieser Knoten als aktueller Knoten gewählt und es wird wiederum eine Stichprobe von Nachbarknoten dieses Knotens betrachtet. Dieses Prozedere wird solange fortgeführt bis sich innerhalb der Stichproben kein Knotenpunkt mit positiver Kostendifferenz mehr findet. Diese Stichprobe von Nachbarknoten wird jedoch nicht auf einmal gezogen und evaluiert, sondern es handelt sich um einen iterativen sequenziellen Prozess, welcher solange läuft bis entweder die Anzahl an Iterationen *maxneighbor* erreicht ist oder ein Nachbarknoten mit positiver Kostendifferenz gefunden ist. Bei jeder Iteration wird dabei der Laufindex j um 1 erhöht, bis der an den Algorithmus übergebene Wert maxneighbor erreicht ist, sollte nicht wie bereits erwähnt ein Nachbarknoten mit positiver Kostendifferenz gefunden worden sein, in welchem Falle Laufindex j wieder auf 1 gesetzt wird. Um die Wahrscheinlichkeit zu erhöhen das globale Minimum zu finden, wird das komplette Vorgehen *numlocal* mal durchgeführt und somit entsprechend viele lokale Minima gespeichert. Das lokale Minimum mit den niedrigsten Kosten stellt dann die endgültige Clusterlösung dar.

Die einzelnen Schritte des Algorithmus sind demnach die Folgenden:

1. Die beiden Parameter numlocal und maxneighbor werden an den Algorithmus übergeben und die zugehörigen Laufindizes i und j werden auf 1 gesetzt.

2. Es wird ein zufälliger Knotenpunkt *current* gewählt.

3. Es wird ein zufälliger Nachbar S von current gewählt und die Kostendifferenz berechnet.

4. Sollte diese positiv sein, wird S zu current, j wird auf 1 gesetzt und der Algorithmus springt wieder zu Schritt 3.

5. Sollte die Kostendifferenz nicht positiv sein, folgt $j = j + 1$ und der Algorithmus springt zu Schritt 3.

6. Im Falle von $j = maxneighbor$ wird current als eines von numlocal lokalen Minima gespeichert, es folgt $i = i + 1$ und der Algorithmus springt zu Schritt 2.

7. Im Falle von $i = numlocal$ wird das lokale Minimum mit den niedrigsten Kosten als Clusterlösung akzeptiert und ausgegeben.

Bezüglich der Parallelisierung ist der Algorithmus leider nicht so aufgebaut, dass er als embarrassingly parallel beschrieben werden kann und es finden sich in der Literatur keine Hinweise auf eine mögliche Parallelisierung. Eine Möglichkeit mehrere Prozessorkerne zu Verwenden, lässt sich allerdings in der zufälligen Auswahl der Nachbarn S des aktuell betrachteten Knotenpunktes finden. Demnach werden so viele zufällig ausgewählte Nachbarknoten betrachtet, wie Prozessorkerne verwendet werden sollen und die Kostendifferenz parallel berechnet. Der Knotenpunkt mit der besten Kostendifferenz wird dann zum aktuell betrachteten Knotenpunkt und der Algorithmus fährt wie oben beschrieben fort.

7.5. BUBBLE

Der Algorithmus *BUBBLE* (Ganti, Ramakrishnan et al., 1999) baut auf dem weit verbreiteten Algorithmus *BIRCH* (Zhang et al., 1996) auf, indem er die Baumstruktur (CF-Tree) übernimmt, sich jedoch in einigen Attributen unterscheidet. BIRCH basiert auf dem Prinzip einzelne Beobachtungen nacheinander entlang innerer Knotenpunkte (fortan *Interior Nodes*) zu einem finalen äußeren Knotenpunkt (fortan *Leaf Node*) zu führen. In diesem werden die Cluster gebildet, jedoch nur einzelne zentrale Clusterfeatures (CFs) des Clusters gespeichert, anstatt die jeweiligen Beobachtungen. Während BIRCH entwickelt wurde um metrische Daten clustern zu können und bspw. euklidische Distanzen als Grundlage zur Erstellung des CF-Tree verwendet, verfolgt BUBBLE das Ziel Daten clustern zu können, deren Distanzmaß lediglich die Dreiecksungleichung erfüllt (Zhang et al., 1996, S. 1). Somit lassen sich auch binäre Daten clustern, deren Distanz durch die Jaccard- oder Hammingdistanz bestimmt wird. Das Grundprinzip von BIRCH und BUBBLE werden im Folgenden erläutert.

7.5.1. BIRCH - Balanced Iterative Reducing and Clustering using Hierarchies

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) hat den Vorteil, dass jede Beobachtung nur ein einziges Mal betrachtet werden muss, um ein Clusterergebnis zu erzielen und somit linear mit der Größe des Datensatzes skaliert. Außerdem sind die Anforderungen an den verfügbaren Speicher gering, da jede Beobachtung nur ein einziges Mal betrachtet werden und danach nicht mehr im Speicher behalten werden muss. Die Berechnungen innerhalb des Algorithmus basieren auf den Attributen *Centroid* $\vec{X}0$

(zentraler Punkt im Cluster), *Radius* R (mittlere Distanz zum Centroid) und *Diameter* D (mittlere paarweise Distanz innerhalb eines Clusters) eines Clusters (Zhang et al., 1996, S. 105):

$$\vec{X0} = \frac{\sum_{i=1}^{N} \vec{X}_i}{N}\,, \tag{7.18}$$

$$R = \left(\frac{\sum_{i=1}^{N} \left(\vec{X}_i - \vec{X0} \right)^2}{N} \right)^{\frac{1}{2}}\,, \tag{7.19}$$

$$D = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \left(\vec{X}_i - \vec{X}_j \right)^2}{N\,(N-1)} \right)^{\frac{1}{2}}\,. \tag{7.20}$$

mit N d-dimensionalen Beobachtungen in einem Cluster: $\{\vec{X}_i\}$ und $i = 1, 2, \ldots, N$.

Zusätzlich werden von Zhang et al. (1996, S. 105) die fünf verschiedenen Distanzmaße $D0$ (euklidische Distanz), $D1$ (Manhatten-Distanz), $D2$ (mittlere Inter-Cluster-Distanz), $D3$ (mittlere Intra-Cluster-Distanz) und $D4$ (Varianzanstieg-Distanz) vorgeschlagen, um die Distanz zwischen 2 Clustern zu berechnen:

$$D0 = \sqrt{\left(\vec{X0}_1 - \vec{X0}_2 \right)^2}\,, \tag{7.21}$$

$$D1 = \left| \vec{X0}_1 - \vec{X0}_2 \right| = \sum_{i=1}^{d} \left| \vec{X0}_1^{(i)} - \vec{X0}_2^{(i)} \right|\,, \tag{7.22}$$

mit N_1 d-dimensionalen Beobachtungen in einem Cluster: $\{\vec{X}_i\}$ und $i = 1, 2, \ldots, N_1$ und N_2 Beobachtungen in einem anderen Cluster: $\{\vec{X}_j\}$ und $j = N_1 + 1, N_1 + 2, \ldots, N_1 + N_2$.

$$D2 = \left(\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_i - \vec{X}_j \right)^2}{N_1 N_2} \right)^{\frac{1}{2}} , \tag{7.23}$$

$$D3 = \left(\frac{\sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} \left(\vec{X}_i - \vec{X}_j \right)^2}{(N_1 + N_2)(N_1 + N_2 - 1)} \right)^{\frac{1}{2}} , \tag{7.24}$$

$$\begin{aligned} D4 = \sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \vec{X}_l}{N_1 + N_2} \right)^2 - \sum_{i=1}^{N_1} \left(\vec{X}_i - \frac{\sum_{l=1}^{N_1} \vec{X}_l}{N_1} \right)^2 \\ - \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_2} \right)^2 . \end{aligned} \tag{7.25}$$

Wie bereits angesprochen wird in BIRCH ein CF-Tree gebildet, deren Nodes aus Clustering Features (CFs) bestehen. Ein CF wiederum wird durch einen Vektor $\mathbf{CF} = \left(N, \vec{LS}, SS \right)$ abgebildet, wobei N die Anzahl der Beobachtungen im Cluster darstellt, $\vec{LS}$ die lineare Summe der Beobachtungsvektoren und SS die quadrierte Summe (Zhang et al., 1996, S. 105). Der große Vorteil der CFs besteht in der Eigenschaft, dass alle vorangegangenen Berechnungsgrundlagen ($\vec{X}0$, R, D, $D0$, $D1$, $D2$, $D3$, $D4$) mithilfe der CFs berechnet werden können. Wird eine Beobachtung zu einem Cluster hinzugefügt, müssen lediglich die CFs aktualisiert werden. Durch das Additivitätstheorem $\mathbf{CF_1} + \mathbf{CF_2} = \left(N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, SS_1 + SS_2 \right)$ ist dies einfach möglich. Anschließend wird die Beobachtung nicht mehr im Speicher benötigt.

Das eigentliche Clustering erfolgt durch die Bildung des CF-Tree, welcher nach und nach auf dynamische Art und Weise durch das Einfügen von Beobachtungen erstellt wird. Zhang et al. (1996, S. 106) beschreiben den Vorgang wie folgt: Es existieren eine Root Node, welche den Startpunkt darstellt, Interior Nodes, die den Pfad durch den CF-Tree zur angemessenen Leaf Node weisen, welche die eigentlichen Cluster beinhalten. Jede Node beinhaltet dabei ein CF der Nodes, die darunter liegen und entsprechende Pointer zu den jeweiligen Child Nodes. Ausgenommen davon sind Leaf Nodes, die die Cluster Features der eigentlichen Cluster enthalten.

Darüber hinaus besitzt ein CF-Tree die 2 Parameter B (Branching Factor) und T (Threshold). Parameter B gibt dabei die Anzahl von Einträgen an, die eine Node halten kann. Parameter T wiederum gibt den Schwellenwert an, welchen R bzw. D beim Hinzufügen einer Beobachtung zu einem Cluster nicht überschreiten darf (Zhang et al., 1996, S. 106). Somit stellen T und B die beiden Parameter dar, welche direkt die Größe des CF-Tree bestimmen.

Wird eine Beobachtung in den CF-Tree eingegeben, wird an der Root Node gestartet. Es wird eine der Distanzmetriken $D0, D1, D2, D3$ oder $D4$ genutzt, um die nächstgelegene Child Node zu finden. Entlang dieser Interior Nodes wird die Beobachtung solange entlang geführt bis eine Leaf Node erreicht wird. Ist eine Leaf Node erreicht, wird der nächstgelegene Centroid der an der Leaf Node angesiedelten Cluster gesucht und die Beobachtung zu diesem Cluster hinzugefügt. Dies geschieht indem die CFs des Clusters durch das Hinzufügen des Falles aktualisiert werden. Ebenfalls müssen dann die Interior Nodes, welche auf dem Pfad zur Leaf Node liegen, entsprechend aktualisiert werden indem der betrachtete CF Vektor addiert wird. Wird allerdings durch das Hinzufügen der Beobachtung der Threshold T überschritten, wird die Beobachtung nicht zu dem Cluster hinzugefügt, sondern bildet einen neuen Cluster, welcher ebenfalls an der aktuellen Leaf Node angesiedelt wird. Wird dadurch allerdings die durch den Parameter L festgelegte Anzahl an Clustern pro Leaf Node (analog zu B) überschritten, muss die Node gesplittet werden.

Das Splitting einer Leaf Node geschieht indem die beiden Cluster, deren Centroiden am weitesten voneinander entfernt sind, gewählt werden und mit den ihrerseits jeweils nächstgelegenen Clustern 2 neue Leaf Nodes eröffnen. Durch das Splitting muss die Parent Node entsprechend aktualisiert werden und es wird ein neuer Eintrag eingefügt. Wird dadurch die durch den Branching Faktor B festgelegte Anzahl an Child Nodes überschritten, muss auch diese Node gesplittet werden. Dies kann sich dann bis zur Root Node fortsetzen, wodurch sich die Höhe des CF-Tree um einen Schritt erhöht. Je höher also T, L und B gesetzt werden, desto kleiner bleibt der CF-Tree und desto weniger Cluster werden gebildet.

Sollte durch niedrig gewählte Werte für T, L und B der CF-Tree größere Dimensionen annehmen als gewünscht, kann ein zusätzlicher Schritt getätigt werden, welcher bestimmte Nodes fusioniert und somit den Tree wieder verkleinert und Einträge an

den Nodes freimacht. Dies kann geschehen, wenn ein Split bis zu einer bestimmten Interior Node reicht. Dort werden dann die beiden Child Nodes zusammengefügt, sollte es sich nicht um die beiden Nodes handelt, welche dem ursprünglichen Split unterlagen.

Nachdem der CF-Tree gebildet wurde, folgt das globale Clustering der Leaf Entries, wodurch dann die finale Clusterlösung generiert wird. Für dieses Clustering kann ein beliebiger Clusteralgorithmus gewählt werden. Beispielsweise agglomerative Verfahren (zum Beispiel *Single Linkage, Average Linkage, Ward...*) wie in Kaufman und Rousseeuw (2005) beschrieben. Vor dem globalen Clustering kann optional der CF-Tree verdichtet werden, indem die Cluster, vertreten durch ihre Centroide, selbst in den CF-Tree eingegeben werden, sollte die Größe des CF-Tree zu groß für den gewählten Algorithmus des globalen Clustering sein (Zhang et al., 1996, S. 107).

Ebenfalls optional kann ein finaler Schritt nach dem globalen Clustering ausgeführt werden. Dieser nutzt die durch das globale Clustering entstandenen Centroide und fügt dann die einzelnen Beobachtungen dem jeweiligen räumlich nächsten Centroiden zu (Zhang et al., 1996, S. 107). Dieser Schritt ist allerdings sehr rechenaufwändig, da jede Beobachtung erneut betrachtet und die Distanz zu den Centroiden berechnet werden muss.

7.5.2. Adaption für nicht-metrische Daten

Nachdem das Grundgerüst von BIRCH beschrieben wurde, folgt nun die Funktionsweise des Algorithmus BUBBLE, der auf BIRCH aufbaut. Es wird ebenfalls auf die gleiche Art und Weise ein CF-Tree gebildet mit anschließendem globalen Clustering der Leaf Entries. Die Berechnungen, die Cluster Features und das Aktualisieren dieser, unterscheiden sich jedoch.

Einer der grundlegenden Unterschiede ist die Erstellung des Clusterzentrums. Während in BIRCH ein Centroid in der geometrischen Mitte des Clusters gebildet wird, ist dies bei nicht-metrischen Daten nicht ohne weiteres möglich. Daher nutzen Ganti, Ramakrishnan et al. (1999) den Begriff des *Clustroid* $\hat{\mathcal{O}}$, welcher die zentrale Beobachtung innerhalb des Clusters darstellt. Des weiteren basieren die Berechnungen auf dem Konzept des *Image Space*, in welchem jedem Objekt O (die Beobachtungen) ein *Image Vector* in einem euklidischen Koordinatensystem zugewiesen wird und die Distanzen zwischen den jeweiligen Objekten beibehalten wird. Das Konzept ist gleich dem der multidimensionalen

Skalierung, bildet aber nur die theoretische Grundlage ohne tatsächlich umgesetzt zu werden.[5] Der Gedanke dahinter ist, dass die Beobachtung, dessen Image Vector die Distanz zum Centroid im Image Space minimiert, diesen am besten generalisiert.

Zur Bestimmung des Clustroid $\hat{\mathcal{O}}$ wird für ein Set von Objekten $\mathcal{O} = \{O_1, \ldots, O_n\}$ jeweils die *Rowsum* berechnet. Diese ist wie folgt definiert:

$$Rowsum = \sum_{j=1}^{n} d^2 \left(O, O_j\right) . \tag{7.26}$$

Der Clustroid $\hat{\mathcal{O}}$ ergibt sich dann als die Beobachtung, dessen Rowsum das Minimum innerhalb des betrachteten Set $\mathcal{O}$ annimmt.

Der Radius $r\left(\mathcal{O}\right)$ kann dann mithilfe des Clustroid $\hat{\mathcal{O}}$ berechnet werden:

$$r\left(\mathcal{O}\right) = \sqrt{\frac{\sum_{i=1}^{n} d^2 \left(O_i, \hat{\mathcal{O}}\right)}{n}} . \tag{7.27}$$

Bei jeder Beobachtung O_{new}, welche in den CF-Tree eingegeben wird, müssen die Cluster Features des Clusters in der entsprechenden Leaf Node aktualisiert werden. Prinzipiell kann der Clustroid $\hat{\mathcal{O}}$ wie in Formel 7.26 bei jedem Durchlauf neu berechnet werden. Dies wäre allerdings sehr rechenaufwändig und würde zusätzlich verlangen, dass sämtliche Beobachtungen im Speicher behalten werden. Daher wird die Rowsum(O_{new}) der eingegebenen Beobachtung zu der Rowsum($\hat{\mathcal{O}}$) des Clustroid addiert. Dazu muss jedoch zunächst Rowsum(O_{new}) berechnet werden, was die gleiche Problematik mit sich bringt.

Um dies wiederum zu vermeiden, schlagen Ganti, Ramakrishnan et al. (1999) vor, Rowsum(O_{new}) nicht exakt zu berechnen, sondern mithilfe des Radius, der Distanz zwischen O_{new} und $\hat{\mathcal{O}}$ und der Anzahl an Objekten im Cluster zu approximieren:

[5] Eine multidimensionale Skalierung hat den Nachteil sämtliche Daten im Speicher zu haben (BUBBLE hat diese Restriktion nicht) und eine komplette Distanzmatrix aller Beobachtungen berechnen zu müssen, was eine quadratische Zeitkomplexität hat und daher sehr rechenaufwändig ist.

$$Rowsum\left(O_{new}\right) = nr^2\left(\mathcal{O}\right) + nd^2\left(\hat{\mathcal{O}}, O\right) . \tag{7.28}$$

Sollte durch das Hinzufügen der neuen Beobachtung zum Cluster der alte Clustroid nicht mehr die zentralste Beobachtung im Cluster sein, muss der neue Clustroid identifiziert werden. Um nun nicht wieder den Clustroid mit allen Beobachtungen im Cluster neu berechnen zu müssen, werden lediglich die p nächstgelegenen Beobachtungen O_{pnear} im Speicher behalten und als Cluster Feature gespeichert. Die Idee dahinter ist folgende: sollte sich durch das Hinzufügen der neuen Beobachtung der Clustroid ändern, ist davon auszugehen, dass sich der neue Clustroid unter O_{pnear} oder O_{new} befindet. Identifiziert wird der neue Clustroid dann durch den Vergleich der Rowsums $\hat{\mathcal{O}}$, O_{pnear} und O_{new}. Die Beobachtung mit der niedrigsten Rowsum stellt den Clustroiden dar.

Ist es, wie in Abschnitt 7.5.1 angesprochen, gewollt, dass der CF-Tree keine zu großen Dimensionen annimmt und daher das Merging zweier Cluster Teil des Algorithmus ist, werden zusätzlich auch noch die p Beobachtungen O_{pfar} im Speicher behalten, die die größten Rowsum Werte aufweisen. Im Falle eines Zusammenfügens zweier dicht aneinander liegender Cluster, ist dann Ganti, Ramakrishnan et al. (1999) zufolge der neue Clustroid in diesen Beobachtungen zu suchen.

Die Cluster Features in BUBBLE bestehen daher gegensätzlich zu BIRCH aus der Anzahl der Beobachtungen n des betrachteten Cluster, des Clustroid $\hat{\mathcal{O}}$, des Radius $r\left(\mathcal{O}\right)$, den p nächstgelegenen Beobachtungen O_{pnear} und gegebenenfalls den p am weitest entfernten Beobachtungen O_{pfar}. Bei Interior Nodes werden diese Cluster Features jedoch nicht benötigt. Schließlich ist die Aufgabe der Interior Nodes lediglich O_{new} den Weg zur adäquaten Leaf Node zu weisen und den nächstgelegenen Cluster auszuwählen. Die CFs einer Interior Node müssen daher ihre jeweiligen Child Nodes zusammenfassen. Ganti, Ramakrishnan et al. (1999) schlagen vor, dafür eine Stichprobe der Clustroids des Subtrees unterhalb der betrachteten Interior Node zu ziehen, um die nächstgelegenen Nodes zu identifizieren. Kritisiert werden kann daran, dass dies Stichprobenfehler mit sich bringt und daher eine Fehlerquelle einführt, welche umgangen werden könnte. Wenn ausgehend von den Leaf Nodes die Parent Nodes ein Level höher den Clustroid der Clustroids (die CFs einer Leaf Node enthalten so viele Clustroids wie Cluster) als Cluster Feature speichert und dies entsprechend wieder für die Parent Nodes dieser Nodes geschieht, lässt

sich der Pfad durch den CF-Tree zum nächstgelegenen Cluster von O_{new} identifizieren, ohne auf Stichproben zurückgreifen zu müssen.

Um den Algorithmus in parallelisierter Form ausführen zu können, ist Garg et al. (2006) zufolge eine Möglichkeit, die Daten gleichmäßig auf die verfügbaren Prozessorkerne zu verteilen und jeden Kern einen eigenen CF-Tree bilden zu lassen. Die Leaf Nodes können anschließend gesammelt werden und dann analog zum sequenziellen Vorgehen, das globale Clustering der Leaf Node Einträge durchgeführt werden.

7.6. MONA - Monothetic Analysis

MONA (Monothetic Analysis, Kaufman & Rousseeuw, 2005) gehört zu der Familie hierarchischer Algorithmen. Im Gegensatz zu agglomerativen Verfahren wie Single Linkage, Ward oder auch K-Means, beginnt der Algorithmus jedoch nicht mit den einzelnen Beobachtungen und fügt diese nach und nach auf Grundlage bestimmter Kriterien zusammen bis am Ende alle Beobachtungen agglomeriert wurden, sondern dreht dieses Vorgehen um. Zu Beginn bildet der gesamte Datensatz einen großen Cluster und wird dann nach und nach in einzelne Cluster aufgeteilt. Dies kann theoretisch solange durchgeführt werden, bis nur noch einzelne Beobachtungen übrig sind bzw. die Cluster sich nicht weiter trennen lassen (bspw. wenn alle Beobachtungen innerhalb eines Clusters identisch sind). Aus diesem Grund handelt es sich bei MONA um ein divisives Verfahren.

Das grundlegende Vorgehen bei MONA besteht darin die zentralst gelegene Variable zu finden und dann den Datensatz anhand der Ausprägungen zu splitten (Kaufman & Rousseeuw, 2005, S. 280). Die Beobachtungen, welche auf der ausgewählten Variable eine 0 aufweisen, bilden ein neues Subset des Datensatzes. Selbiges passiert mit den Beobachtungen, welche eine 1 aufweisen. Aus diesem Grund ist der Algorithmus explizit für binäre Datensätze entwickelt worden. Dass lediglich eine Variable für das Splitting genutzt wird, verleiht dem Algorithmus seinen Namen (*Monothetic*). Würden mehrere Variablen genutzt werden, würde es sich um ein *polythetisches* Verfahren handeln. Der Vorteil gegenüber vielen anderen hierarchischen Verfahren besteht in der Tatsache, dass keine Distanzmatrix der Beobachtungen benötigt wird. Das macht den Algorithmus geeignet für große Datensätze mit vielen Beobachtungen, solange die Anzahl der Variablen nicht zu groß ist. Der Grund dafür liegt in der Art und Weise, wie die zentralste Variable gewählt wird, an-

hand welcher der jeweilige Datensatz gesplittet wird. Dies wird im Folgenden beschrieben.

Es wird ein Assoziationmaß benötigt, das geeignet ist, die zentralste Variable zu identifizieren. Kaufman und Rousseeuw (2005, S. 281) wählen dafür folgenden Ansatz: Es werden zunächst 2 Variablen miteinander verglichen. Dies geschieht indem die Anzahl an Beobachtungen gezählt werden, welche auf beiden Variablen den Wert 0 aufweisen. Diese Summe wird mit der Anzahl an Beobachtungen multipliziert, welche auf beiden Variablen eine 1 aufweisen. Dieses Vorgehen wird nun wiederum ebenfalls mit den Kombinationen 0 und 1 bzw. 1 und 0 durchgeführt. Anschließend wird die Differenz zwischen den beiden Produkten gebildet und notiert. Dies kann auf einfache Weise erreicht werden, indem eine Kreuztabelle der beiden Variablen gebildet wird und die Differenz zwischen dem Produkt der Diagonalen und dem Produkt der Gegendiagonalen berechnet wird. Everitt et al. (2011, S. 85) zählen neben dem gerade genannten noch folgende mögliche Assoziationmaße auf:

$$(ad - bc)^2 \,, \tag{7.29}$$

$$\frac{(ad - bc)^2 \, n}{(a + b)\,(a + c)\,(b + d)\,(c + d)} \,, \tag{7.30}$$

$$\sqrt{\frac{(ad - bc)^2 \, n}{(a + b)\,(a + c)\,(b + d)\,(c + d)}} \,, \tag{7.31}$$

$$\frac{(ad - bc)^2}{(a + b)\,(a + c)\,(b + d)\,(c + d)} \,. \tag{7.32}$$

Dabei stellt a die Summe der Beobachtungen dar, welche auf beiden betrachteten Variablen eine 1 aufweisen, b entspricht der Summe mit den Ausprägungen 1 und 0, c ist entsprechend 0 und 1 und d wenn beide Variablen den Wert 0 annehmen (siehe Tabelle 7.3).

Tabelle 7.3.: Kreuztabelle zweier binärer Variablen als Grundlage zur Berechnung der Assoziationsmaße.

Ausprägung	1	0
1	a	b
0	c	d

Auf eine dieser Art und Weisen werden sämtliche Variablen innerhalb des Clusters verglichen. Die resultierenden Werte werden für jede Variable aufsummiert. Die Variable mit dem höchsten Wert stellt die zentralste Variable dar und wird für das Splitting ausgewählt. Die Komplexität des Algorithmus hängt damit entscheidend von der Anzahl der Variablen ab und nicht so sehr von der Anzahl der Beobachtungen.

Alternativ dazu kann auch der von Lance und Williams (1968) entwickelte *Information Content* I genutzt werden, welcher wie folgt definiert ist:

$$I = sn \log n - \sum_{j=1}^{s} \left(a_j \log a_j + (n - a_j) \log \left(n - a_j\right)\right) , \tag{7.33}$$

wobei s die Anzahl der Variablen darstellt und a_j die Summe der Beobachtungen, welche auf Variable j eine 1 aufweisen. Diese Statistik I wird zunächst für den betrachteten Datensatz I_i berechnet und danach für die beiden Subsets g und h des Datensatzes, welche sich ergeben würden, wenn der Datensatz anhand von Variable j gesplittet wird. Anschließend wird die Abweichung, die von Lance und Williams (1968) als „information-fall“ bezeichnet wird, zwischen den resultierenden Statistiken berechnet:

$$\Delta I_{gh,i} = I_i - I_g - I_h \, . \tag{7.34}$$

Dieser Vorgang wird für jede Variable des betrachteten Datensatzes wiederholt und $\Delta I_{gh,i}$ gebildet. Die Variable für die $\Delta I_{gh,i}$ das Maximum annimmt, wird für das Splitting ausgewählt.

Wurde der Datensatz bereits einmal gesplittet, beginnt der Algorithmus innerhalb der beiden resultierenden Cluster von vorn. Das heißt es wird für jeden Cluster eine neue Variable zum Splitten gewählt. Wurde eine Variable bereits einmal gewählt, kann diese logischerweise nicht erneut gewählt werden, weil sämtliche Beobachtungen innerhalb des

betrachteten Clusters bereits den gleichen Wert auf dieser Variable aufweisen und anhand dieser nicht erneut gesplittet werden könnten (Kaufman & Rousseeuw, 2005, S. 301).

Prinzipiell kann dieser Algorithmus solange fortgeführt werden bis ein Cluster nur noch aus einer einzelnen Beobachtung besteht oder sämtliche Beobachtungen innerhalb eines Cluster identische Ausprägungen aufweisen. Ein Kriterium zum Stoppen des Algorithmus wird von Kaufman und Rousseeuw (2005) nicht angegeben. Denkbar wären jedoch zum Beispiel Schwellenwerte für die Kompaktheit eines Clusters. Wird dieser erreicht, wird der Cluster nicht weiter gesplittet. Auf der anderen Seite kann der Algorithmus natürlich auch auf manuelle Weise begrenzt werden, indem die Anzahl der gewünschten Splits an den Algorithmus als Argument übergeben wird. Daraus würde sich dann auch die Anzahl der gewünschten Cluster ergeben wie beispielsweise bei K-Means. Dies ist auch das Vorgehen, welches bei der Analyse in dieser Arbeit gewählt wurde.

Der rechenaufwändigste Teil des Algorithmus liegt in der Identifikation der zentralsten Variable, was für jeden Split wiederholt werden muss. Daher bietet es sich an, das Parallelisierungskonzept des Algorithmus an diesem Teil der Berechnungen auszurichten.

Wie angesprochen, existieren mehrere Ansätze, die Variable zu identifizieren, die zum Splitten des Datensatzes verwendet wird. Zum einen gibt es den Vergleich jeder Variable mit jeder anderen und der Berechnung eines Assoziationmaßes. Zum anderen auf Grundlage des Information Content, welcher lediglich eine einmalige Betrachtung jeder Variable benötigt und in seiner Zeitkomplexität daher linear mit der Anzahl der Variablen skaliert. Dieser Ansatz sollte daher weniger Rechenaufwand benötigen. Um dies zu überprüfen und die geeignetere Variante für diese Arbeit zu bestimmen, wurde ein Benchmark beider Varianten (beim Assoziationmaß wurde das von Kaufman und Rousseeuw (2005, S. 281) vorgeschlagene Maß verwendet) durchgeführt. Als Grundlage diente ein Datensatz mit 100.000 Beobachtungen und 100 Variablen. Es wurden 5 Splits durchgeführt, sodass 6 Cluster gebildet wurden. Die Variante auf Grundlage des Assoziationmaß benötigte 64,11 Sekunden, wohingegen die Variante auf der Grundlage des Information Content nach lediglich 35,49 Sekunden konvergierte. Daher wird in dieser Arbeit auch diese Variante verwendet.

7.7. Multibit Trees

Multibit Trees stellen einen Algorithmus dar, der von der Funktionsweise sehr der vorangegangenen Methode MONA ähnelt. Ursprünglich wurde der Algorithmus von Kristensen et al. (2010) entwickelt, um eine Datenbank molekularer Fingerabdrücke nach einem bestimmten Eintrag abzufragen. Diese Fingerabdrücke haben einen binären Datentyp und die Datenbanken sind in der Regel sehr groß. Das Vorgehen wird zum Teil auch beim Blocking im Record Linkage verwendet, wenn die Records in Bloom-Filtern encodiert sind, die ebenfalls einen binären Datentyp aufweisen (Bachteler et al., 2013). Der Grundgedanke ist dabei, die Daten in Blocks zusammenzufassen, welche sich bzw. dem Record ähnlich sind und Blöcke ignoriert werden können, die einen bestimmten Ähnlichkeitsschwellenwert unterschreiten. Das Einteilen des betrachteten Datensatzes in Datenblöcke mit einer bestimmten Ähnlichkeit ist dabei dem Clustering von Datensätzen sehr ähnlich, weshalb es nahe liegt, diese Vorgehensweise auch zum Clustering zu nutzen.

Der offensichtliche Unterschied beim Clustering zu den vorher genannten Anwendungsgebieten, besteht darin, dass keine einzelne Datenzeile gefunden werden muss, welche einer gesuchten Datenzeile entspricht bzw. diesem hinreichend ähnlich ist. Stattdessen müssen ganze Gruppen ähnlicher Beobachtungen gefunden werden. Wie dies mithilfe von Multibit Trees geschehen kann, wird im Folgenden erläutert.

Den ersten Schritt stellt das Einteilen des Datensatzes in Subsets dar, die durch die Summe der Variablen mit der Ausprägung 1 definiert sind, was auch als die Größe einer Beobachtung bezeichnet wird (Kristensen et al., 2010, S. 2). Das bedeutet, dass alle Beobachtungen mit beispielsweise 4 Variablen mit Ausprägung 1 ein Subset darstellen, das Gleiche gilt für alle Beobachtungen mit 5 Einsen und so weiter. Beim Durchsuchen des Datensatzes nach einer bestimmten Beobachtung hat dies den Vorteil, dass alle Subsets, welche Beobachtungen mit einer anderen Größe beinhalten, ignoriert werden können. Für das Clustering spielt dieser Schritt keine Rolle und wird daher ignoriert.

Der Kern des Algorithmus besteht im eigentlichen Erstellen des Multibit Trees, was dem Vorgehen bei MONA sehr ähnelt. Es wird bei der Root Node gestartet und es werden 2 Child Nodes gebildet. Genau wie bei MONA wird eine Variable gewählt und alle Beobachtungen, welche die Ausprägung 0 auf dieser Variable aufweisen, konstituieren die eine Child Node, während alle Beobachtungen mit einer 1 die andere Child Node darstellen.

Bei der Auswahl dieser Splittingvariable unterscheiden sich die beiden Algorithmen jedoch. Während bei MONA ein Kriterium gewählt wird, welches die Beziehung einer Variable zu den anderen Variablen abbildet und auf Grundlage dessen die Splittingvariable gewählt wird, wird beim Erstellen von Multibit Trees bei jedem Split die Variable gewählt, welche zu einem möglichst balancierten Subtree führt (Kristensen et al., 2010, S. 4). Dieser Vorgang dauert solange an, bis entweder die Größe der entstehenden Partitionen ein bestimmtes Schwellenwertargument t überschreitet (Bachteler et al., 2013, S. 5) oder alle Beobachtungen in einer Node Duplikate darstellen und die Node demnach nicht weiter gesplittet werden kann (Kristensen et al., 2010, S. 4).

Das Vorgehen beim Wählen der Splittingvariable ist im Vergleich zu MONA eher simpel aber dennoch sinnvoll. Durch das Speichern von Listen an jeder Node, welche die Anzahl an Variablen speichern, die einen konstanten Wert 0 oder 1 in allen Beobachtungen unterhalb der betrachteten Node beinhalten, kann eine untere Grenze des Jaccard Koeffizienten zwischen gesuchter Datenzeile und sämtlichen Beobachtungen im Subtree berechnet werden. Sollte dieser Wert unter der zulässigen Grenze für den Jaccard Koeffizient liegen, kann der komplette Subtree bei der Suche ignoriert werden. Für diese Eigenschaft des Multibit Tree ist die beschriebene Methode bei der Wahl der Splittingvariable ausreichend.

Die Idee, Multibit Trees zum Clustern von Beobachtungen zu verwenden, macht sich die vorangegangene beschriebene Eigenschaft zunutze, dass sich die Beobachtungen immer ähnlicher werden, je öfter die Nodes gesplittet werden und der Tree immer mehr Level erhält. Allerdings führt dies auch dazu, dass je nach gewähltem Schwellenwertargument t bei Konvergieren des Algorithmus, sehr viele unterschiedliche Nodes resultieren, welche die Anzahl an vorliegenden Clustern um ein Vielfaches überschreiten kann. In dieser Arbeit wird ein Wert von 1 % von n für t gewählt. Wenn die Beobachtungen innerhalb der resultierenden Leafnodes jedoch als sehr ähnlich betrachtet werden, liegt es nahe, Repräsentanten der jeweiligen Leafnodes zu bestimmten und diese mittels eines arbiträren Clusteralgorithmus (bspw. PAM) zu clustern und sämtliche Beobachtungen einer Leafnode dem Cluster zuzufügen, der den jeweiligen Repräsentanten enthält.

Das Bestimmen des jeweiligen Repräsentanten ist zum einen durch das Ziehen eines zufälligen Falles aus den Leafnodes denkbar, was jedoch anfällig für Stichprobenfehler wäre. Geeigneter, wenn auch rechenaufwändiger, wäre das Bestimmen des Repräsentanten mittels Identifikation der zentralsten Beobachtung durch das Erstellen einer Distanzma-

Tabelle 7.4.: Mittelwerte der Differenzen nach jedem Simulationsschritt, wenn die Werte der Evaluationskriterien von PAM von K-Means abgezogen werden.

Recall	Precision	F-Score	Rand Index	Entropy	NMI
0,02	0,14	0,11	0,15	-0,32	0,21

trix innerhalb der Leafnodes, solange Schwellenwertargument t klein genug ist, dass das Berechnen vollständiger Distanzmatrizen innerhalb der Leafnodes aus Effizienzgründen vertretbar ist. Durch Parallelisierung erscheint ein Wert von etwa 1.000 für t vertretbar, wenn genug Prozessorkerne vorhanden sind, um die gegebenenfalls sehr vielen Distanzmatrizen, je nach Dimensionalität des Datensatz, gleichzeitig zu berechnen.

Für das angesprochene Clustern der Repräsentanten der jeweiligen Leafnodes sollte ursprünglich PAM verwendet werden. Jedoch stellten sich die verwendeten Funktionen `Cluster_Medoids` und `pam` aus den Paketen `ClusterR` (Mouselimis, 2021) respektive `cluster` (Maechler et al., 2021) als unzuverlässig heraus und brachen nicht nachvollziehbar in manchen Fällen ab. Es sollen keine Algorithmen verwendet werden, die in dieser Arbeit selbst betrachtet werden, um das Ergebnis des Algorithmus nicht von dem eines anderen Algorithmus abhängig zu machen, dessen Evaluation aussteht. Aus diesem Grund wurde K-Means verwendet. Da K-Means eigentlich nicht für binäre Daten geeignet ist, wurde der Algorithmus PAM gegenübergestellt und die Ergebnisse der beiden Algorithmen anhand einer Simulation verglichen. Diese bestand aus den gleichen Faktorstufen, wie die eigentliche Simulation, welche die Grundlage dieser Arbeit darstellt. Jedoch wurde die Anzahl der Beobachtungen auf 1.000 und die Anzahl der Variablen auf 100 begrenzt. Die Ergebnisse von Recall, Precision, F-Score, Rand Index, Entropy und NMI für PAM wurden nach jeder Faktorstufe von den jeweiligen Ergebnissen für K-Means subtrahiert und der Mittelwert dieser Differenzen über alle Faktorstufen gebildet. Diese sind in Tabelle 7.4 abgebildet. Positive Ergebnisse bei Recall, Precision, F-Score, Rand Index und NMI und negative Ergebnisse bei Entropy weisen demnach auf bessere Clusterlösungen bei K-Means hin. Wie sich sehen lässt schneidet K-Means insgesamt besser ab als PAM.

7.8. CLOPE - Clustering with sLOPE

Dem Algorithmus *CLOPE*[6] (Yang et al., 2002), welcher ursprünglich für das Clustering von Transaktionen erdacht wurde, liegt ein intuitives Optimierungskriterium zugrunde. Dieses Optimierungskriterium besteht daraus, bei der Allokation der Beobachtungen auf die verschiedenen Cluster, das Histogramm der Beobachtungen eines Clusters so kompakt wie möglich zu gestalten. Die Höhe der Balken des Histogramms ergibt sich durch die Spaltensummen der Datenmatrix des betrachteten Clusters, wobei die Zeilen in dieser Matrix die Beobachtungen darstellen und die Spalten die Variablen. Sollte eine Spaltensumme 0 ergeben, wird diese Variable entsprechend nicht im Histogramm berücksichtigt. Die Kompaktheit eines Histogramms wird dann dadurch optimiert, indem das Verhältnis Höhe zu Breite des Histogramms maximiert wird. Dies wird erreicht, indem Beobachtungen, welche sich in den Ausprägungen der Variablen ähneln, zum selben Cluster allokiert werden (Yang et al., 2002, S. 682–683).

Je höher also der Wert für Ausdruck $H(C)/W(C)$, desto kompakter das Histogramm. Dabei stellt $H(C))$ die Höhe und $W(C)$ die Breite des Histogramms von Cluster C dar. Die Höhe $H(C)$ ergibt sich als das Verhältnis von Größe $S(C)$ zu Breite $W(C)$. Die Größe $S(C)$ wiederum ist durch die Summe aller Ausprägungen innerhalb des Clusters gegeben:

$$S(C)) = \sum_{j \in D(C)} Occ(j, C) \ , \tag{7.35}$$

wobei $Occ(j, C)$ das Vorkommen von Variable j in Cluster C beschreibt und $D(C)$ die Datenmatrix von C. Die Breite $W(C)$ ergibt sich entsprechend einfach als die Anzahl der Spaltensummen, welche ungleich 0 sind.

Yang et al. (2002, S. 683) argumentieren jedoch, dass die Verwendung von $H(C))$ nicht angemessen sei und illustrieren dies mit einem Beispiel: Im Falle eines Datensatzes mit lediglich 2 Beobachtungen, welche keinerlei identische Ausprägungen auf den Variablen aufweisen, hätten die beiden möglichen Clusterings (beide Beobachtungen bilden jeweils einen Cluster oder bilden zusammen einen Cluster) die gleichen Ausprägungen für $H(C))$, wobei ein möglicher Beispieldatensatz Tabelle 7.5 entnommen werden kann. Wird

[6] Varianten für das Clustering von Datastreams finden sich unter den Namen SCLOPE und σ-SCLOPE bei Ong et al. (2004) und Yap und Ong (2005).

stattdessen der Gradient $G(C) = S(C)/W(C)^2$ verwendet, stellt sich heraus, dass das Clustering, in dem beide Beobachtungen jeweils einen Cluster bilden, höher bewertet wird.

Tabelle 7.5.: Beispieldatensatz

	V1	V2	V3	V4	V5	V6
Fall 1	1	1	1	0	0	0
Fall 2	0	0	0	1	1	1

Um nun die bestmögliche Verteilung der Beobachtungen auf die jeweiligen Cluster zu erreichen, welche die kompaktesten Histogramme der Cluster garantiert, wird ein globales Optimierungskriterium benötigt. Dieses wird von Yang et al. (2002, S. 683) als *Profit* tituliert und berechnet sich wie folgt:

$$\text{Profit}_r(\mathbf{C}) = \frac{\sum_{i=1}^{k} \frac{S(C_i)}{W(C_i)^r} * |C_i|}{\sum_{i=1}^{k} |C_i|}, \tag{7.36}$$

wobei k die Anzahl der Cluster beschreibt. Außerdem wurde der Exponent im Nenner des Gradienten $S(C)/W(C)^2$ durch r ersetzt, sodass sich $S(C)/W(C)^r$ ergibt. Dieser Exponent r stellt die sogenannte *Repulsion* dar. Diese Repulsion ist ein Parameter, der dazu genutzt werden kann die Homogenität innerhalb der Cluster zu steuern. Je höher der Wert für r gewählt wird, desto homogener müssen die Cluster sein, um Profit zu maximieren (Yang et al., 2002, S. 684).

Der Ablauf des Algorithmus gestaltet sich in 2 Phasen. In der ersten Phase (*Initialization*) wird der Datensatz einmalig durchlaufen und eine initiale Clusterlösung erstellt. Dies geschieht indem jede Beobachtung einmal zu jedem Cluster hinzugefügt und Profit berechnet wird. Dabei wird auch die Lösung betrachtet, in der die Beobachtung selbst einen neuen Cluster darstellt. Der Algorithmus beginnt also mit einem einzigen Cluster, welcher aus der ersten Beobachtung im Datensatz besteht. In der zweiten Phase (*Iteration*) wird der Datensatz iterativ durchlaufen und die Beobachtungen nacheinander zwischen den Clustern hin und hergeschoben, um Profit zu maximieren. Es wird solange iteriert, bis entweder Profit nicht mehr weiter erhöht werden kann oder die vorher definierte maximale Anzahl an Iterationen erreicht worden ist.

Aufgrund der Tatsache, dass es relativ aufwändig ist, bei dem genannten Vorgehen jedes mal die Histogramme der einzelnen Cluster zu bilden bzw. jedes Mal erneut Profit zu berechnen (setzt die Berechnung von $S(C_i)$ und $W(C_i)$ voraus), sobald eine Beobachtung zum betrachteten Cluster hinzugefügt wurde, schlagen Yang et al. (2002, S. 684) die Funktion *DeltaAdd* vor. DeltaAdd ergibt sich dabei wie folgt:

$$\Delta_{add} = \frac{S_{new}(C) * N(C) + 1}{W_{new}(C)^r} - \frac{S(C) * N(C)}{W(C)^r}. \tag{7.37}$$

Wobei die Terme mit dem Subscript *new* die Werte des betrachteten Clusters inklusive der betrachteten Beobachtung sind. Die Lösung, die Δ_{add} maximiert, ist dabei auch die, welche Profit maximiert[7]. Es müssen also lediglich die Werte für $S(C)$, $N(C)$ und $W(C)$ für jeden Cluster gespeichert und entsprechend aktualisiert werden sobald eine Beobachtung zu einem Cluster hinzugefügt wurde. In der Iterationsphase muss demnach die jeweilige betrachtete Beobachtung zunächst aus dem entsprechenden Cluster entfernt werden, um dann Δ_{add} für alle Cluster berechnen zu können.

Wie aus dem Ablauf ersichtlich, wird das Ergebnis des Prozesses von der Sortierung des Datensatz beeinflusst. Yang et al. (2002, S. 686) argumentieren auf der Grundlage von Experimenten mit einem Datensatz in unterschiedlichen randomisierten Reihenfolgen, dass die Ergebnisse vergleichbar sind und daher die Reihenfolge keine große Rolle spielt und zu vernachlässigen ist. Allerdings wurde dies nur anhand eines einzelnen Datensatzes[8] getestet, welcher mit lediglich 8124 Beobachtungen nicht sehr groß ist. Eine Möglichkeit diesen Kritikpunkt zu umgehen, wird von Li, Le und Wang (2015) in einer Abwandlung des Algorithmus vorgeschlagen. Dabei wird nicht linear jede Beobachtung der Datenbank einmalig betrachtet und jede Beobachtung nach und nach einem Cluster zugewiesen, sondern ähnlich den agglomerativen hierarchischen Clusteralgorithmen wie zum Beispiel Single Linkage, werden sämtliche Beobachtungen anfangs als einzelne Cluster betrachtet, welche jeweils miteinander verglichen werden. Der zentrale Unterschied zu den anderen agglomerativen hierarchischen Clusteralgorithmen besteht dann darin, dass Profit als Distanzfunktion agiert und die beiden Cluster fusioniert werden, die Profit maximieren. Aufgrund der dadurch stark erhöhten Komplexität, es wird jede Beobachtung mit jeweils jeder anderen Beobachtung verglichen, wird diese Variante in dieser Arbeit nicht ver-

[7] Der Beweis ist bei Yang et al. (2002, S. 684) zu finden.

[8] Es handelt sich um den *Mushroom* Datensatz, welcher unter https://archive.ics.uci.edu/ml/datasets/mushroom heruntergeladen werden kann.

wendet, da sie nicht mit großen Datensätzen skaliert. Ein weiterer Kritikpunkt liegt im Parameter Repulsion, welcher arbiträr gewählt werden muss.

Um den Algorithmus angemessen parallelisieren zu können, wird das Vorgehen von Li, Le, Wang et al. (2015) verwendet[9]. Dabei wird ein klassischer *Divide and Conquer*-Ansatz (Cormen et al., 2009, S. 65) verwendet. Die Beobachtungen des Datensatzes werden in mehrere Datensätze aufgeteilt und können dann von mehreren Prozessorkernen gleichzeitig bearbeitet werden. Dabei wird der CLOPE Algorithmus jeweils auf jeden dieser Datensätze angewandt. Anschließend wird geprüft, ob diese Clusterlösung durch das Zusammenfügen der resultierenden Cluster verbessert werden kann. Dies geschieht, indem eine leicht abgewandelte Form der Iteration Phase angewandt wird. Jedoch werden diesmal die Cluster selbst als Input gewählt. Es wird also wiederum Δ_{add} für jeden Cluster berechnet, mit dem einzigen Unterschied, dass die Cluster selbst hinzugefügt werden und nicht die einzelnen Beobachtungen. Die dadurch resultierende Clusterlösung stellt das finale Clustering dar.

Der optimale Wert für r zur Anwendung in dieser Arbeit wurde experimentell ermittelt. Dabei wurden Datensätze der Größe $n = 10.000$ mit 100 Variablen für sämtliche Faktorstufen des Simulationsdesigns erstellt und CLOPE jeweils mit den Werten 1, 2, 3 und 4 für r darauf angewandt. Dabei wurde eine mittlere Differenz über alle Faktorstufen von 0,33 für den Wert des F-Score zwischen dem schlechtesten Wert ($r = 1$) und dem besten Wert ($r = 2$) beobachtet. Zwar zeigte sich bei der Laufzeit für einen Wert von $r = 2$, dass diese um das 1,59-fache höher gegenüber einem Wert von $r = 1$ ist, jedoch überwiegt in diesem Fall die höhere Qualität der Clusterlösung gegenüber der moderaten Zeitersparnis. Daher wird in dieser Arbeit ein Wert von $r = 2$ gewählt.

7.9. Large Item

Dem Algorithmus CLOPE sehr ähnlich ist der Algorithmus *Large Item* (Wang et al., 1999), welcher ebenfalls für das Clustering von Transaktionen entwickelt wurde. Genau wie CLOPE liegt Large Item ein globales Optimierungskriterium zugrunde, das sich jedoch etwas anders gestaltet. Die Intuition dieses Optimierungskriterium basiert darauf, dass viele Beobachtungen in den einzelnen Clustern jeweils eine 1 auf einigen für den

[9]Das gleiche Vorgehen findet sich bei Yuping et al. (2016).

Cluster charakteristischen Variablen (Large Items) haben und möglichst wenige Variablen, welche nur wenige Einsen aufweisen (small items). Eine weitere Gemeinsamkeit liegt darin, dass beide Algorithmen in einem ersten Scan des Datensatzes ein initiales Clustering vornehmen und dann in einer zweiten Phase dieses Clustering iterativ verbessern, indem jede Beobachtung erneut betrachtet und dem Cluster zugewiesen wird, welcher das Optimierungskriterium verbessert. Damit besteht ebenfalls eine gewisse Ähnlichkeit zum EM-Algorithmus und der K-Means Familie.

Die angesprochenen Large Items, welche charakteristisch für den jeweiligen Cluster sind, werden durch den Parameter *minimum support* bestimmt. *Support* beschreibt in diesem Kontext die Anzahl der Einsen einer Variable innerhalb eines Clusters. Minimum support gibt demnach an, wie hoch der Anteil der Einsen einer Variable sein muss, um als large item zu gelten und wird im Vorhinein festgelegt. Aufgrund dieser Eigenschaft des Algorithmus Cluster anhand charakteristischer Variablen zu diskriminieren, welche einen hohen Anteil an Einsen haben, besteht ebenfalls eine gewisse Ähnlichkeit zu MONA und Multibit Trees, wobei es sich im Unterschied zu den beiden genannten Algorithmen um einen polythetischen[10] Algorithmus handelt, solange minimum support kleiner als 100 % gewählt wird (Wang et al., 1999, S. 484). Eine Empfehlung, wie minimum support gewählt werden sollte, wird von Wang et al. (1999) nicht getätigt. Ein sehr niedriger Wert führt dabei zu wenigen Clustern. Im Extremfall von $minsup = 1/n$ werden alle Beobachtungen in einem einzigen Cluster zusammengefasst (Wang et al., 1999, S. 486).

Das Optimierungskriterium $Cost\,(C)$, errechnet sich aus den Intraclusterkosten $Intra\,(C)$ und Interclusterkosten $Inter\,(C)$. Die Intraclusterkosten ergeben sich als die Vereinigungsmenge der small items aller Cluster:

$$Intra\,(C) = |\cup_{i=1}^{k} Small_i|\,. \tag{7.38}$$

Die Interclusterkosten hingegen bilden sich indem die Vereinigungsmenge der Large Items aller Cluster von der Summe aller Large Items aller Cluster subtrahiert werden:

$$Inter\,(C) = \sum_{i=1}^{k} |Large_i| - |\cup_{i=1}^{k} Large_i|\,. \tag{7.39}$$

[10] Zur Definition siehe Abschnitt 7.6.

Daraus folgt, dass $Intra\,(C)$ die Unähnlichkeit innerhalb der Cluster misst und verhindert, dass sich Cluster bilden, welche eine hohe Anzahl von small items haben. Analog misst $Inter\,(C)$ die Ähnlichkeit zwischen den Clustern und sorgt dafür, dass die Cluster eigene charakteristische Variablen haben, welche in den anderen Clustern selten vorkommen (Wang et al., 1999, S. 485). Das eigentliche Optimierungskriterium $Cost\,(C)$ berechnet sich dann als Summe von $Intra\,(C)$ und $Inter\,(C)$, wobei $Intra\,(C)$ noch mit dem Faktor w gewichtet wird. Wie genau w bestimmt wird oder welche Rolle der Parameter spielt, wird von Wang et al. (1999) nicht angegeben. Es wird lediglich angegeben, dass standardmäßig $w = 1$ gewählt wird und ein Wert kleiner 1 den Einfluss von $Intra\,(C)$ verringert, während ein Wert größer 1 den Einfluss erhöht.

Die beiden Phasen des Algorithmus sind dabei nahezu identisch mit denen von CLOPE in Abschnitt 7.8. In der ersten Phase wird eine initiale Clusterlösung erstellt. Jede Beobachtung wird nacheinander betrachtet. Die erste Beobachtung stellt automatisch den ersten Cluster dar. Die zweite Beobachtung wird dann zum einen mit dem ersten Cluster zusammengefügt und $Cost\,(C)$ berechnet und zum anderen bildet sie wiederum einen eigenen Cluster und $Cost\,(C)$ wird berechnet. Die Lösung mit dem niedrigeren Wert für $Cost\,(C)$ wird akzeptiert bis alle Beobachtungen einem Cluster zugefügt wurden. In der anschließenden *Refinement*-Phase wird wiederum jede Beobachtung einzeln betrachtet, aus ihrem ursprünglichen Cluster extrahiert und wiederum einmal allen Clustern zugefügt. Es wird die Lösung akzeptiert, welche $Cost\,(C)$ minimiert. Diese Phase iteriert solange, bis entweder keine Beobachtungen mehr zwischen den Clustern verschoben werden oder die vorher definierte maximale Anzahl an Iterationen erreicht ist. Wang et al. (1999, S. 484) geben an, dass in der Regel nicht mehr als 3 Iterationen nötig sind.

Dieses Vorgehen ist sehr speichereffizient: es müssen lediglich die Werte für support und welche Variablen jeweils ein Large Items oder small item in den einzelnen Clustern darstellen, sowie die Anzahl der Beobachtungen in den Clustern N, im Speicher behalten werden. Das Aktualisieren von support gestaltet sich als einfache Vektoraddition zwischen der betrachteten Beobachtung und support. Als large bzw. small item ergeben sich dann einfach die Positionen im Vektor von support, welche gleich $minsupport * N$ sind bzw. diesen Wert über- oder unterschreiten.

Bezüglich des Parallelisierungskonzeptes liegen in der Literatur keine Ansätze für Large Item vor. Aufgrund der Ähnlichkeit zum Algorithmus CLOPE, was den Ablauf des

Algorithmus angeht, bietet es sich an, die gleiche Parallelisierungsstrategie zu wählen, die auch in Abschnitt 7.8 beschrieben ist, welche für Large Item angepasst ist. Das bedeutet, der Datensatz wird aufgeteilt und jeder Prozessorkern führt den Algorithmus für dieses Subset aus. Nur das im Unterschied zum Vorgehen bei CLOPE logischerweise bei dem Zusammenfügen der einzelnen Cluster, die als Produkt des parallelisierten Prozesses entstehen, $Cost\,(C)$ minimiert wird (im Gegensatz zur Maximierung von Profit bei CLOPE).

7.10. SCALE

Ein weiterer Algorithmus, welcher ebenfalls wie CLOPE und Large Item für das Clustern von Transaktionen entwickelt wurde, ist *SCALE* (Yan et al., 2009). Die Struktur des Algorithmus folgt im Wesentlichen dem Vorgehen von CLOPE und Large Item. Der Hauptteil besteht aus 2 Phasen. In der ersten Phase wird jede Beobachtung einmalig betrachtet und es resultiert ein erstes Clustering. Diese Phase wird von Yan et al. (2009) *weighted coverage density measure based initial cluster assignment phase* genannt. Die zweite Phase ist wiederum iterativ und weist einzelne Beobachtungen vom ursprünglichen Cluster jeweils alle anderen Clustern zu und behält die Lösung bei, welche ein globales Optimierungskriterium maximiert. Diese Phase wird von Yan et al. (2009) *weighted coverage density measure based iterative clustering refinement phase* genannt. Jedoch lagert SCALE noch eine weitere entscheidende Phase vor die beiden erwähnten. Bevor die Betrachtung der einzelnen Beobachtungen des Datensatzes erfolgt und jede Beobachtung einem Cluster zugewiesen wird, wird zunächst die grobe Clusterstruktur ermittelt. Dies geschieht, indem eine Stichprobe gezogen und dann eine bereits etablierte Clustermethode darauf angewandt wird. Yan et al. (2009, S. 11) schlagen dafür die *BKPlot*-Methode (Chen & Liu, 2005b) vor. Dies gilt vor allem, wenn die Anzahl der Cluster nicht im Vorhinein bekannt ist. Prinzipiell kann jedoch ein beliebiger Algorithmus verwendet werden. Im Rahmen dieser Arbeit wird der etablierte Algorithmus PAM (siehe Abschnitt 7.4.1) verwendet. Über die Größe der Stichprobe wird leider keine Angabe gemacht und muss daher arbiträr festgelegt werden. Für die Simulationen in dieser Arbeit wird die Größe auf 1.000 Beobachtungen festgelegt, da dies trotz der hohen Komplexität von PAM sehr einfach bewältigt werden kann. Ein weiterer Unterschied zu den genannten Algorithmen besteht darin, dass die Beobachtungen des Datensatzes nicht in der Reihenfolge betrachtet werden, wie sie vorliegen, sondern zufällig erfolgt. Somit kann es auch zu unterschiedlichen Er-

gebnissen bei der wiederholten Anwendung des Algorithmus auf dieselben Daten kommen.

Nach diesem ersten Schritt, von Yan et al. (2009, S. 10–11) „*weighted coverage density clustering preparation step*“ genannt, existieren dann bereits Cluster, denen die Beobachtungen zugefügt werden können, ohne dass die erste Beobachtung zunächst einen einzelnen Cluster bildet, welchem dann sukzessive Beobachtungen zugeordnet werden. Aufgrund dieser Tatsache ist es auch nicht möglich für einzelne Beobachtungen bei der jeweiligen Evaluation einen eigenen Cluster zu bilden, was theoretisch die Anzahl der Cluster und damit auch die Komplexität in die Höhe treiben kann. Stattdessen werden die Beobachtungen nach und nach den aus dem *weighted coverage density clustering preparation step* bekannten Clustern zugewiesen.

Neben diesem zusätzlichen Schritt besteht der andere Unterschied zu CLOPE und Large Item im Optimierungskriterium. Grundlage dieses Kriteriums bildet die Abstraktion des Datensatzes als zweidimensionales Grid. Die X-Achse dieses Grids stellen die Variablen dar und die Y-Achse die Beobachtungen. Die Zellen dieses Grids gelten als gefüllt, wenn eine Beobachtung eine 1 auf der entsprechenden Variable aufweist. Enthält sie eine 0 ist die Zelle leer. Ziel des Algorithmus ist es Rechtecke in diesem Grid zu finden, welche eine hohe Dichte an gefüllten Zellen beinhalten, mit möglichst wenig leeren Zellen. Die Beobachtungen, welche dann einem solchen Rechteck angehören, bilden einen Cluster. Um diese möglichst kompakten Rechtecke zu finden, wird von Yan et al. (2009, S. 7) die *Coverage Density (CD)* eingeführt, welche den Anteil der gefüllten Zellen innerhalb eines Rechtecks beschreibt und die es für einen Cluster zu maximieren gilt. Berechnet wird die Coverage Density eines Cluster k als:

$$CD\left(C_k\right) = \frac{S_k}{N_k * M_k} = \frac{\sum_{j=1}^{M_k} occur\left(I_{kj}\right)}{N_k * M_k}\,, \tag{7.40}$$

mit S_k = Summe aller Einsen innerhalb des Clusters, N_k = Anzahl der Beobachtungen im Cluster, M_k = Anzahl der Variablen im Cluster mit mindestens einer 1, $occur\left(I_{kj}\right)$ = Ausprägung von Variable I_j im Cluster k mit 1.

Ein großer Nachteil dieser Metrik ist allerdings, dass jede Variable als gleichbedeutend für die Bildung des Clusters angesehen wird, obwohl manche Variablen (wie bei MONA oder Large Item) als besonders charakteristisch für den Cluster gelten. Wird demnach

eine Zelle (i, j) betrachtet, steuern sowohl die Beobachtung i als auch Variable j den gleichen Beitrag zur Wichtigkeit der Zelle bei. Wenn der Beitrag der Beobachtung als $T_i = T = \frac{1}{N_k}$ und der Beitrag der Variable als $W_j = W = \frac{1}{M_k}$ beschrieben werden kann, ergibt sich die Coverage Density alternativ als

$$CD = T_i * \sum_{j=1}^{M_k} occur\left(I_{kj}\right) * W_j = T * W * \sum_{j=1}^{M_k} occur\left(I_{kj}\right) . \tag{7.41}$$

Demnach hat jede Zelle den gleichen Einfluss auf die Kompaktheit des Clusters. Um diesem Problem entgegen zu wirken, schlagen Yan et al. (2009, S. 8) die sogenannte *Weighted Coverage Density (WCD)* vor, welche bestimmten Variablen einen stärkeren Beitrag zuschreibt als anderen. Bei dieser Metrik ergibt sich der Beitrag einer Variable als Anteil der Einsen der Variable an der Summe aller Einsen aller Variablen im Cluster:

$$W_j = \frac{occur\left(I_{kj}\right)}{S_k} , \tag{7.42}$$

mit $\sum_{j=1}^{M_k} W_j = 1$. Somit berechnet sich die WCD als:

$$\begin{aligned} WCD\left(C_k\right) &= T * \sum_{j=1}^{M_k} occur\left(I_{kj}\right) * W_j && (7.43)\\ &= \frac{1}{N_k} * \sum_{j=1}^{M_k} \left(occur\left(I_{kj}\right) * \frac{occur\left(I_{kj}\right)}{S_k} \right) && (7.44)\\ &= \frac{\sum_{j=1}^{M_k} occur\left(I_{kj}\right)^2}{S_k * N_k} . && (7.45) \end{aligned}$$

Durch diese Metrik werden nun den charakteristischen Variablen eines Clusters höhere Gewichte zugewiesen. Auf Grundlage der WCD lässt sich nun das globale Optimierungskriterium bilden, welches die Güte einer jeweiligen Clusterlösung darstellt. Dieses Kriterium nennt sich *Expected Weighted Coverage Density (EWCD)* und wird als die Summe der mit der Anzahl der zugehörigen Beobachtungen der jeweiligen Cluster gewichteten WCD berechnet:

$$EWCD\left(C^K\right) = \sum_{k=1}^{K} \frac{N_k}{N} * WCD\left(C_k\right) \tag{7.46}$$

$$= \frac{1}{N} \sum_{k=1}^{K} \frac{\sum_{j=1}^{M_k} occur\left(I_{kj}\right)^2}{S_k} . \tag{7.47}$$

Yan et al. (2009, S. 10) weisen darauf hin, dass EWCD in jedem Falle das Maximum annimmt, wenn jede Beobachtung einen eigenen Cluster darstellt. Um dies zu verhindern, muss die maximale Anzahl an Cluster begrenzt werden. Dies sollte jedoch durch den *weighted coverage density clustering preparation step* automatisch gegeben sein. Das gilt vor allem wenn für diesen Schritt ein Algorithmus wie PAM gewählt wird, in welchem die Anzahl der Cluster einen Parameter darstellt, der im Vorhinein festgelegt wird.

Um die EWCD berechnen zu können, müssen die WCDs der einzelnen Cluster bei jedem Hinzufügen einer Beobachtung aktualisiert werden. Dazu müssen lediglich einige wenige Informationen gespeichert werden, wodurch der Algorithmus, genau wie CLOPE und Large Item, einen sehr niedrigen Speicherbedarf hat. Es reicht, wenn bei jedem Hinzufügen einer Beobachtung der Vektor der Beobachtungen zu der Summe der vorherigen Vektoren hinzuaddiert wird. Daraus lassen sich dann M_k, S_k und $occur\left(I_{kj}\right)$ ableiten. Zusätzlich muss noch die Anzahl der Beobachtungen im Cluster gespeichert werden.

Bezüglich eines Parallelisierungskonzeptes finden sich keine Hinweise in der Literatur. Wie auch CLOPE und Large Item lässt sich leider weder der Algorithmus selbst, noch einzelne Schritte innerhalb des Algorithmus als embarrassingly parallel beschreiben. Aufgrund des nahezu identischen strukturellen Ablaufs von SCALE zu CLOPE und Large Item, könnte allerdings prinzipiell das gleiche Parallelisierungskonzept verwendet werden. Jedoch ist auch eine andere Variante denkbar, welche bei diesem Algorithmus genutzt werden soll. Diese funktioniert wie folgt: Nach dem *weighted coverage density clustering preparation step* wird der Datensatz in den darauffolgenden Schritten nicht sequenziell Beobachtung für Beobachtung prozessiert, sondern in Chunks. Ein (Main-)Chunk wird wiederum in Subchunks geteilt, welche dann von den einzelnen Prozessorkernen Beobachtung für Beobachtung prozessiert werden. Durch diese parallele Zuweisung der Beobachtungen zu Clustern folgt allerdings, dass die Clustersummaries (M_k, S_k und $occur\left(I_{kj}\right)$) nicht gleichzeitig global aktualisiert werden können. Die Aktualisierung erfolgt zunächst temporär auf lokaler Ebene innerhalb der einzelnen Tasks und wird nach

dem Fertigstellen des Tasks wieder verworfen. Durch den *weighted coverage density clustering preparation step* besitzen die Cluster bereits eine relativ stabile Struktur, sodass, wenn auch nicht optimal, die Beobachtungen dennoch dem korrekten Cluster zugewiesen werden sollten. Die tatsächliche Aktualisierung der Clustersummaries findet dann statt, nachdem alle Subchunks prozessiert wurden. Bei der Wahl der Größe der Chunks muss zwischen Performance und Güte der Clusterlösung abgewogen werden. Werden zu kleine Chunks gewählt, lohnen sich die Kommunikationskosten zwischen den Prozessorkernen nicht und der Performancegewinn fällt niedrig aus. Werden zu große Chunks gewählt, leidet gegebenenfalls die Güte der Clusterlösung dadurch, dass die Clustersummaries nicht adäquat auf parallele Weise aktualisiert werden können.

Um die Performance der Parallelisierung zu überprüfen, wurde in einem kleinen Experiment die sequenzielle und parallele Variante des Algorithmus gegenübergestellt. Dafür wurde ein Datensatz mit 1.000.000 Beobachtungen, 2 Clustern und einer mittleren Korrelation ($0,35 \leq \rho \geq 0,45$) in 5 Variablenblöcken simuliert. Die Mittelwerte der Variablen in den verschiedenen Blöcken wurden aus folgenden Kategorien gezogen: Cluster 1: *low, mid, high, low, mid*. Cluster 2: *high, low, mid, high, low*, wobei *low* Mittelwerten zwischen 0,05 und 0,2, *mid* Mittelwerten zwischen 0,2 und 0,35 und *high* Mittelwerten zwischen 0,3 und 0,49 entsprechen. Die Stichprobengröße für den *weighted coverage density clustering preparation step* wurde auf 1.000 Beobachtungen gesetzt und mittels PAM durchgeführt. Die Anzahl der Iterationen in der *weighted coverage density measure based iterative clustering refinement phase* wurde auf 2 begrenzt, da danach nur noch mit marginalen Verbesserungen gerechnet werden kann, welche die Kosten die dadurch entstehen nicht rechtfertigen. In der parallelisierten Variante wurde die Größe der (Main-)Chunks auf 50.000 gesetzt. Beide Varianten lieferten nahezu identische Ergebnisse. Die sequenzielle Variante ergab einen F-Score Wert von 0,834108 und die parallele einen F-Score Wert von 0,833684. Jedoch war die parallele Variante 8,66 mal schneller als die sequenzielle. Daher wird in dieser Arbeit die parallele Variante genutzt.

7.11. Sorted Neighborhood

Der Algorithmus *Sorted Neighborhood* (Hernández & Stolfo, 1998) wurde ursprünglich entwickelt, um (approximative) Duplikate in großen Datensätzen zu finden und wird

daher auch bevorzugt beim Record Linkage verwendet.[11] Die behandelten Datensätze beinhalten in der Regel personenbezogene Daten, wie zum Beispiel Namen, Adressen und bestimmte Attribute. Mit ein paar Adaptionen, sollte die Methode jedoch auch für das Clustern von binären Datensätzen verwendet werden können, was in dieser Arbeit geprüft werden soll. Der Algorithmus selbst besteht in der ursprünglichen Form im wesentlichen aus drei Schritten (Hernández & Stolfo, 1998, S. 12–13):

1. Create Keys: Es wird ein *Key* auf Basis einer Variable erstellt, welcher sicherstellt, dass auch approximative Duplikate sich im wesentlichen auf diesem Key ähneln. Dieser Key ist stark domainabhängig, erfordert daher im Regelfall domainspezifisches Wissen und muss manuell festgelegt werden. Als Beispiel nennen Hernández und Stolfo (1998, S. 14) eine Zeichenkette bestehend aus den ersten drei Konsonanten des Nachnamens, den ersten drei Buchstaben des Vornamens, der Hausnummer und allen Konsonanten des Straßennamens.

2. Sort Data: Der Datensatz wird nach dem gebildeten Key aus Schritt 1 sortiert.

3. Merge: Es wird ein *Gleitfenster* verwendet, um nur die Beobachtungen zu vergleichen, welche sich innerhalb dieses Fensters befinden und so die Komplexität des Algorithmus zu reduzieren. Durch das Sortieren des Datensatzes mit dem Key aus Schritt 1 sollten sich ähnliche Beobachtungen im sortierten Datensatz in unmittelbarer Nähe befinden, weshalb Beobachtungen am Anfang des Datensatzes nicht mit Beobachtungen am Ende verglichen werden müssen. Dazu wird ein Fenster der Größe w gebildet und jede neue Beobachtung, die sich neu innerhalb der Grenzen des Fensters befindet, wird mit jeweils allen anderen vorherigen $w - 1$ Beobachtungen verglichen, während die erste Beobachtung wieder aus dem Fenster entfernt wird, sodass die Größe des Fensters während einer linearen Betrachtung des Datensatzes konstant bleibt.

Die Wahl des Parameters w ist dabei von entscheidender Bedeutung und stellt einen Kompromiss zwischen Komplexität und Genauigkeit dar. Die niedrigste Komplexität wird erreicht, wenn $w = 2$ gewählt und immer nur Paare von aufeinanderfolgenden Beobachtungen verglichen werden, was zur niedrigsten Genauigkeit führt. Der Gegenpol

[11] Beim Record Linkage besteht eine der Herausforderungen darin, Paare von Beobachtungen in verschiedenen Datensätzen zu finden, welche zwar der selben Entität zuzuordnen sind, aber aufgrund von Datenfehlern (zum Beispiel Rechtschreibfehler) nicht komplett, sondern nur approximativ identisch sind (Christen, 2012, S. 42–43).

dazu liegt im Fall $w = N$, bei dem jeder Fall mit jedem anderen Fall verglichen wird.

Probleme können entstehen, wenn der Key, nach dem sortiert wurde, fehleranfällig ist. Das bedeutet, wenn durch zum Beispiel Zahlendreher ein Duplikat am Ende des Datensatzes positioniert wird statt am Anfang. Hernández und Stolfo (1998, S. 17) geben hier das Beispiel 193456782 und 913456782 an, in welchen die ersten beiden Ziffern vertauscht sind. Um auch solche Duplikate entdecken zu können, wird vorgeschlagen, den sogenannten *Multi-Pass*-Ansatz zu nutzen. Das bedeutet, dass mehrere Durchläufe des Algorithmus genutzt werden, wobei bei jedem Durchlauf ein anderer Key genutzt wird. Zusätzlich wird abschließend eine *transitive Hülle* gebildet. Das bedeutet, dass sollten Beobachtungen a und b als Duplikat identifiziert werden und Beobachtung b und c ebenfalls, folgt daraus, dass auch a und c Duplikate sind. Diese transitive Hülle kann sowohl für einzelne Durchläufe genutzt werden, sollten sich a und b, und b und c jeweils in einem Fenster befinden, nicht jedoch a und c, als auch beim Multi-Pass-Ansatz mit verschiedenen Durchläufen. Hernández und Stolfo (1998) schlagen vor den Multi-Pass-Ansatz zu verwenden, da dieser die Genauigkeit erhöht und dabei zwecks Effizienz ein kleines w zu wählen. Sie begründen das damit, dass sich in ihren Experimenten gezeigt hat, dass beim Multi-Pass-Ansatz ein großes w lediglich die Komplexität erhöht, aber keinen nennenswerten Vorteil bei der Genauigkeit mit sich bringt. Entsprechend wurde in dieser Arbeit $w = 100$ gewählt.

Um den Algorithmus für das Clustern binärer Daten zu nutzen, müssen einige Anpassungen getätigt werden. Vor allem Schritt 1 und 2 können nicht so ohne weiteres übernommen werden. Eine Möglichkeit den Datensatz dennoch in eine sinnvolle Reihenfolge zu bringen, besteht in der Verwendung des *Hammingweights*. Dadurch wird die Möglichkeit erhöht Beobachtungen aus denselben Clustern im Datensatz zueinander zu bringen. Das bedeutet, dass die Beobachtungen entsprechend ihrer Zeilensumme sortiert werden. Darüber hinaus muss ein Schwellenwert festgelegt werden, bei welcher Distanz zueinander entschieden wird, ob die Beobachtungen aus demselben Cluster stammen oder nicht.

Zudem zeigt sich das Problem, dass bei jedem Multi-Pass-Durchlauf eine neue Sortierung gewählt werden muss. Das Sortieren mittels des Hammingweights kann jedoch nur einmal verwendet werden. Eine sinnvolle Alternative besteht jedoch in der zufälligen Anordnung der Beobachtungen. Das führt zwar zunächst dazu, dass gegenläufig zum

eigentlichen Ziel möglichst ähnliche Beobachtungen nahe beieinander zu haben, eben nicht mehr gegeben ist, allerdings bringt es durch die Verwendung der transitiven Hülle einen entscheidenden Vorteil mit sich. Durch die heterogene Verteilung der Angehörigen eines Clusters über den Datensatz bei einer zufälligen Sortierung ist die Wahrscheinlichkeit geringer, dass es innerhalb des Moving Window dazu kommt, dass ein Cluster nicht mehr innerhalb des Fensters repräsentiert ist und ein Cluster, welcher eigentlich zusammengehört, mehrere Cluster bildet. Voraussetzung dafür ist, dass das Fenster eine gewisse Größe besitzt und auch die Cluster ausreichend groß sind. Durch das Verwenden mehrerer Durchläufe und der transitiven Hülle sollte jedoch auch dieses Problem abgemildert werden.

Um den Algorithmus auf parallelisierte Art und Weise anwenden zu können, schlagen Hernández und Stolfo (1995) einen Ansatz vor, welchen sie *Clustering Method* nennen. Dabei wird, wie oftmals üblich, der Datensatz in Blöcke aufgeteilt. Auf jeden Block wird von einem einzelnen Prozessorkern der Algorithmus angewendet und das Ergebnis wird zusammengefügt. Um den Datensatz angemessen aufzuteilen, schlagen sie vor, ähnlich der Bildung des Keys in Schritt 1 des regulären Algorithmus, ein domainspezifisches Mapping auf Grundlage der Variablen vorzunehmen, so dass Blöcke mit bereits möglichst ähnlichen Beobachtungen entstehen, welche parallel bearbeitet werden können. Um die Effizienz zu optimieren, müsse dabei jedoch angestrebt werden, dass die entstehenden Blöcke eine ähnliche Größe aufweisen. Dies sollte approximativ gegeben sein, wenn ein Mapping gewählt wird, das auf Attributen basiert, die in den Daten uniform verteilt sind. Die Breite der Bins des Histogramms der gewählten Attribute, wird dann so gebildet, dass sich so viele Bins ergeben, wie Blöcke gewünscht sind. In diesem Falle wäre das die Anzahl der verfügbaren Prozessorkerne.

Durch das Verwenden des Multi-Pass-Ansatzes und das Prozessieren der Blöcke auf parallele Art und Weise entsteht so eine Anzahl an Partitionen, welche wiederum durch das Verwenden der transitiven Hülle zusammengeführt werden können. Bei jedem Durchlauf des Multi-Pass-Ansatz entstehen so viele dieser Partitionen in den einzelnen Blöcken wie Cluster gefunden werden. Bei einem Multi-Pass-Ansatz mit 2 Durchläufen mit 10 Blöcken, welche jeweils von einem Prozessorkern bearbeitet werden, und 2 Clustern, welche in jedem Block identifiziert werden, wäre das Resultat 40 Partitionen. Beim Bilden der transitiven Hülle wird nun geprüft, ob sich eine Beobachtung in mehr als einer Partition befindet. Sollte dem so sein, werden die betroffenen Partitionen zusammengelegt

und dedupliziert. Befinden sich Beobachtungen der betroffenen Partitionen wiederum ebenfalls in anderen Partitionen, werden auch diese zum selben Cluster hinzugefügt. So ist am Ende jede Beobachtung genau einem Cluster zugeordnet.

Technisch kann die Umsetzung der transitive Hülle auch als *All-Pairs Shortest Path Problem* (Cormen et al., 2009, S. 684) beschrieben und mittels des *Floyd-Warshall-Algorithmus* gelöst werden. Im konkreten Anwendungsfall geschieht dies über die Erstellung einer symmetrischen Matrix, deren Dimensionalität durch die Anzahl der resultierenden Partitionen bestimmt wird, welche im vorherigen Absatz beschrieben wurden. Es werden alle Partitionen paarweise miteinander verglichen und geprüft, ob sich eine Beobachtung in beiden Partitionen befindet. Sollte dem so sein, erhält die eben angesprochene symmetrische Matrix an entsprechender Stelle eine 1. Die Matrix beschreibt somit einen Graphen, welcher die direkte Verbindung zwischen den einzelnen Partitionen darstellt. Anschließend wird diese Matrix dem Floyd-Warshall-Algorithmus übergeben, welcher als Output eine Matrix gleicher Dimensionalität liefert und die Anzahl der Schritte von einem beliebigen Punkt in besagtem Graph der Partitionen zu einem beliebigen anderen Punkt angibt, sollte denn eine Verbindung existieren. Die Partitionen, deren Indizes in der Outputmatrix des Floyd-Warshall-Algorithmus einen Wert größer 1 aufweisen, werden zusammengelegt. Ein einfaches Beispiel sei folgendes: Beobachtung a befindet sich in Partition 1 und 2. Beobachtung b befindet sich in Partition 2 und 3. Die Schrittlänge zwischen Partition 1 und 2, als auch zwischen Partition 2 und 3 beträgt 1, da eine direkte Verbindung besteht. Partition 1 und 3 haben keinen direkten Kontakt, sind aber über Partition 2 verbunden. Die Schrittlänge zwischen Partition 1 und 3 beträgt somit 2. Sollte Partition 4 keinerlei Beobachtungen besitzen, welche sich in den Partitionen 1 bis 3 befinden, besteht kein Pfad zwischen Partition 4 und den Partitionen 1 bis 3. Das Resultat dieses Beispiels wäre, dass alle Beobachtungen der Partitionen 1 bis 3 zu einem Cluster zusammengefügt werden und Partition 4 einen einzelnen Cluster darstellt.

Ähnlich zu Single Linkage besteht eine offensichtliche Schwäche des Algorithmus in der Tendenz einzelne große Cluster zu bilden, sollten die Cluster nicht klar voneinander getrennt sein. Sobald auch nur ein kleiner Overlap zwischen sonst eindeutig getrennten Clustern vorhanden ist, erkennt der Algorithmus diese als einen. Bei 2 Clustern reicht schon eine einzelne Beobachtung, welche sich unterhalb des Schwellenwertes in Distanz zu 2 Clustern befindet, dass der Algorithmus durch die transitive Hülle beide Cluster zu einem zusammenfügt. Eine weitere Schwäche ist die hohe Komplexität der transitiven

Hülle, welche mit Ansteigen der resultierenden Partitionen, auf welche die transitive Hülle angewendet werden muss, zu einem Performanceproblem werden kann. Die Wahl eines geeigneten Thresholds ist daher von großer Wichtigkeit. Ein großer Wert führt dazu, dass der Algorithmus einzelne große Cluster bildet. Im schlimmsten Fall entsteht so ein einziger großer Cluster mit allen Beobachtungen. Ein kleiner Wert begünstigt einzelne Ausreißer, (zu) viele resultierende Cluster und den Anstieg der Anzahl der Partitionen und damit der Komplexität. Somit ist der Algorithmus stark von den vorliegenden Daten abhängig, ob er effizient genutzt werden kann und die Clusterstruktur identifiziert oder nicht.

Um einen geeigneten Schwellenwert zu finden bei welcher Distanz zueinander entschieden wird, ob die Beobachtungen aus demselben Cluster stammen oder nicht, wäre eine Möglichkeit experimentell vorzugehen. Es wurden für alle Faktorstufen des Simulationsdesigns, wobei die Anzahl der Beobachtungen auf 10.000 und die Anzahl der Variablen auf 100 begrenzt wurden, Werte von 0,3 bis 0,6 als Schwellenwert evaluiert. Dabei zeigte sich lediglich ein mittlerer Unterschied von 0,06 für den F-Score über alle Faktorstufen hinweg zwischen dem besten (0,3) und dem schlechtesten Wert (0,6). Jedoch lag die Laufzeit im Median für einen Schwellenwert von 0,3 um das 1,47-fache höher als für einen Schwellenwert von 0,6. Bei einzelnen Faktorstufen zeigten sich Ausreißer, die noch wesentlich höher lagen (weshalb für die vorangegangene Rechnung der Median verwendet wurde). Dies konnte vor allem bei einzelnen Versuchen mit größeren Datensätzen beobachtet werden. In dieser Arbeit wird daher ein Schwellenwert von 0,5 verwendet.

7.12. Proximus

Der Algorithmus *Proximus* (Koyuturk et al., 2005) wurde explizit für binäre Daten entwickelt und basiert nicht wie viele andere Algorithmen auf der Optimierung eines Kriteriums abhängig von der Zuweisung einer Beobachtung zu einem bestimmten Cluster, sondern auf der (rekursiven) Dekomposition der kompletten Datenmatrix. Dadurch weist der Algorithmus Parallelen zu gängigen Methoden der Dimensionsreduktion wie Singular Value Decomposition (SVD, Gentle, 2007, S. 127–128) und vor allem Semi Discrete Decomposition (SDD, O'Leary & Peleg, 1983) auf. Alle drei Methoden basieren darauf, die Eingangsmatrix in eine Approximationsmatrix zu zerlegen, welche den Rang 1 aufweist. Die Art und Weise unterscheidet sich jedoch. Bei der SVD stellt die beste Approximation mit Rang 1 das äußere Produkt der ersten Singulärvektoren skaliert mit

dem zugehörigen ersten Singulärwert dar und beschreiben die erste Komponente der Matrix. Folgende Rang-1-Approximationen mit niedrigeren Singulärwerten beschreiben Komponenten, welche orthogonal zur den restlichen Komponenten stehen und damit jeweils die Residualmatrix der vorangegangenen Approximation approximieren. SDD wiederum extrahiert bei jeder Approximation jeweils sowohl Zeilen als auch Spalten, wodurch es sich zum Biclustering[12] eignet. Dabei weist die Residualmatrix möglichst homogene Werte auf. Das Ziel von Proximus ist jedoch die Approximation der Datenmatrix via zweier Matrizen $\mathbf{A} \approx \mathbf{X}\mathbf{Y}^{\mathbf{T}}$, welche ultimativ sowohl zum Clustern des Datensatzes genutzt werden können, als auch darüber hinaus die dominanten Pattern innerhalb des Datenmaterials offenbaren, welche für die jeweiligen Cluster verantwortlich sind und somit auch als eine Art Centroid der Cluster interpretiert werden können. Dabei ist Proximus weder darauf angewiesen, dass dominante Pattern orthogonal zueinander sind, noch werden Zeilen *und* Spalten geclustert, wodurch es zur klassischen Partitionierung der Zeilen in der Clusteranalyse verwendet werden kann. Die Matrix $\mathbf{X}$ stellt dabei die *Presence Matrix* dar, deren Einträge die Indikatoren für die Clusterzugehörigkeit der Beobachtungen sind, während Matrix $\mathbf{Y}^{\mathbf{T}}$ als *Pattern Matrix* bezeichnet wird und die dominanten Pattern der zugehörigen Cluster enthält. Wie diese beiden Matrizen gebildet werden können, wird im Folgenden beschrieben.

Koyuturk et al. (2005, S. 449) beschreiben die Problemstellung so, dass gegeben m binärer Vektoren $\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}$ in einem n dimensionalen Raum versucht wird k n-dimensionale binäre Vektoren $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_m}$ zu finden, so dass die Differenz der Vektoren $\mathbf{a_i}$ und $\mathbf{y_j}$ unterhalb einer gewissen Fehlertoleranz liegt:

$$\forall\, 1 \leq i \leq m, \exists j \text{ s.t. } \|\mathbf{a_i}] - \mathbf{y_j}\|_2^2 \leq \epsilon\,. \tag{7.48}$$

Wird nun die Ausgangsmatrix mit $\mathbf{A} = [\mathbf{a_1}\mathbf{a_2}\ldots\mathbf{a_m}]^T$ binären Vektoren und die Approximationsmatrix mit $\mathbf{Y} = [\mathbf{y_1}\mathbf{y_2}\ldots\mathbf{y_m}]^T$ binären Vektoren beschrieben, kann das Problem als $\left\|\mathbf{A} - \mathbf{X}\mathbf{Y}^{\mathbf{T}}\right\|_\infty \leq \epsilon$ beschrieben werden, wobei die Presence Matrix $\mathbf{X}$ eine $m \times k$ Matrix darstellt, deren Einträge mit $x_i = 1$ gefüllt sind, sollte der zugehörige Pattern Vektor $\mathbf{y_j}$ aus der Pattern Matrix $\mathbf{Y}^{\mathbf{T}}$ innerhalb der benötigten Hamming Distanz von Vektor $\mathbf{a_i}$ lokalisiert sein. Diese benötigte Hamming Distanz wird durch ϵ definiert

[12] Beim Biclustering werden nicht nur die Beobachtungen eines Datensatzes geclustert, sonder auch die Variablen. Siehe dazu z.B. Reddy et al. (2014, S. 288).

und muss im Vorhinein als Parameter festgelegt werden.

Das im vorherigen Absatz beschriebene Problem wird gelöst, indem rekursiv Approximationen mit Rang 1 berechnet werden. Eine Approximation mit Rang 1 ist in diesem Falle das äußere Produkt zweier Vektoren $\mathbf{xy^T}$, welches die Anzahl der Nullen in der Matrix $\mathbf{A} - \mathbf{xy^T}$ maximiert. Als einfaches Beispiel geben Koyuturk et al. (2005, S. 449) eine 3×3 Matrix mit deren zugehöriger Rang-1-Approximation:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} = \mathbf{xy^T}$$

Der Presence Vektor $\mathbf{x}$ ist in diesem Fall nur mit Einsen gefüllt, was bedeutet, dass das dominante Pattern aus dem Pattern Vektor $\mathbf{y^T}$ in allen Zeilen der Ausgangsmatrix innerhalb der angegebenen Fehlertoleranz liegt.[13] Würde der Pattern Vektor außerhalb der Fehlertoleranz einer Zeile der Ausgangsmatrix liegen, würde der Presence Vektor an dieser Stelle eine 0 erhalten.

Um diese Rang-1-Approximationen zu finden, schlagen Koyuturk et al. (2005, S. 451) eine alternierende iterative Heuristik vor und geben eine Errorfunktion an:

$$\left\| \mathbf{A} - \mathbf{xy^T} \right\|_F^2 = \|\mathbf{A}\|_F^2 - 2\mathbf{x^T Ay} + \|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2 \,. \tag{7.49}$$

Den Fehler zu minimieren ist äquivalent zur Maximierung von:

$$C_d(\mathbf{x}, \mathbf{y}) = 2\mathbf{x^T Ay} - \|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2 \,, \tag{7.50}$$

dabei kann $\mathbf{x}$ nun gefunden werden, indem $\mathbf{y}$ fix gehalten und $s = \mathbf{Ay}$ gesetzt wird. Mit Formel 7.51 werden die Werte entschieden, welche die Einträge von $\mathbf{x}$ annehmen:

$$x(i) = \begin{cases} 1, & \text{falls } 2s(i) \geq \|\mathbf{y}\|_2^2 \\ 0, & \text{andernfalls } . \end{cases} \tag{7.51}$$

[13] In diesem einfachen Beispiel stimmt er gar komplett mit den Zeilen der Matrix überein.

Dies folgt daraus, dass $\mathbf{x}$ zur Maximierung von $C_d(\mathbf{x}, \mathbf{y})$ beiträgt, wenn mindestens die Hälfte der Positionen in $\mathbf{y}$, welche mit einer 1 besetzt sind, den Positionen in der entsprechenden Zeile in der Ausgangsdatenmatrix entsprechen. Mit diesen aktualisierten Werten für $\mathbf{x}$ kann nun $\mathbf{y}$ gefunden werden, indem $\mathbf{x}$ fix gehalten wird. In diesem alternierenden Verfahren wird solange iteriert, bis sich keine Verbesserungen mehr feststellen lassen und der Algorithmus somit konvergiert.

Das genannte iterative Verfahren setzt eine adäquate Initialisierung des Vektors $\mathbf{y}$ voraus. Dahingehend nennen Koyuturk et al. (2005, S. 452) drei Möglichkeiten zur Initialisierung:

1. Partition: Es wird eine Spalte des Datensatzes gewählt und der Datensatz entlang dieser Dimension geteilt. In der Partition, die Einsen in dieser Spalte enthält, wird ein Centroid gebildet, welcher als initialer Patternvektor genutzt wird.

2. Greedy Graph Growing: Es wird zufällig eine Zeile des Datensatzes gezogen. Nach und nach werden weitere Zeilen hinzugefügt, welche Einsen auf einer Position mit der gezogenen Beobachtung teilen. Dies wird solange iteriert, bis eine balancierte Partition erreicht ist. Es wird ebenfalls wieder der Centroid gebildet, welcher als initialer Patternvektor genutzt wird.

3. Random Row: Die simpelste Art der Initialisierung besteht einfach in der Ziehung einer zufälligen Beobachtung aus dem Ausgangsdatensatz und der Nutzung dieser als initialer Patternvektor.

In ihrer zeitlich vorangegangenen Beschreibung von Proximus geben Koyutürk und Grama (2003, S. 152) statt *Random Row* die Variante *Neighbor* an. Diese ist gleich der Variante Random Row mit dem Unterschied, dass die benachbarten Beobachtungen zusätzlich herangezogen werden und wiederum der Centroid gebildet wird. Aufgrund der Tatsache, dass diese Variante in der Veröffentlichung von 2005 nicht mehr enthalten ist und nicht erwähnt wird, muss davon ausgegangen werden, dass sie verworfen und durch Random Row ersetzt wurde.

Die angesprochene Rekursivität des Algorithmus liegt in der wiederholten Partitionierung der Ausgangsmatrix $\mathbf{A}$ entlang des Presence Vektors $\mathbf{x}$. Alle Zeilen von $\mathbf{A}$, welche eine 1 in $\mathbf{x}$ aufweisen, bilden Matrix $\mathbf{A_1}$. Die restlichen bilden Matrix $\mathbf{A_0}$. Auf Matrix $\mathbf{A_0}$ wird wiederum die beste Rang-1-Approximation gesucht und die Matrix entsprechend in

die Matrizen $\mathbf{A_{01}}$ und $\mathbf{A_{00}}$ zerlegt. In Matrix $\mathbf{A_1}$ wird nun geprüft, ob sich alle Zeilen innerhalb eines Hamming-Radius von ϵ befinden. Sollte dem so sein, stoppt die Rekursion an dieser Stelle für diese Submatrix und die jeweiligen Presence- und Pattern-Vektoren werden gespeichert und sind Teil der finalen Approximation. Der Presence-Vektor definiert einen finalen Cluster und der Pattern-Vektor das zugehörige dominante Pattern des Clusters. Sollten nicht alle Zeilen aus $\mathbf{A_1}$ innerhalb des Radius ϵ sein, wird auch Matrix $\mathbf{A_1}$ erneut in Matrizen $\mathbf{A_{11}}$ und $\mathbf{A_{10}}$ zerlegt. Die Rekursion läuft solange bis entweder alle Zeilen innerhalb des Radius ihres zugehörigen Pattern-Vektors liegen oder die Matrix nicht mehr weiter zerlegt werden kann.

Ein Nachteil des Algorithmus liegt darin, dass sich keiner der einzelnen Vorgänge während des Ablaufs des Algorithmus einfach parallelisieren lässt. Eine Möglichkeit, die jedoch gegebenenfalls zulasten der Güte des Clusterings gehen könnte, stellt das klassische Divide-and-Conquer Vorgehen (Cormen et al., 2009, S. 65) dar. Der Datensatz wird zunächst zufällig angeordnet und es werden so viele Partitionen gebildet, wie Prozessorkerne genutzt werden sollen. Auf jeder Partition wird Proximus parallel angewandt, was zu so vielen Clusterings wie Partitionen führt, welche sinnvoll zusammengeführt werden müssen. An dieser Stelle lässt sich jedoch der Umstand zunutze machen, dass Proximus eine Matrix mit den dominanten Pattern des Datensatzes zurückgibt. Es werden die einzelnen dominanten Pattern aus den jeweiligen Pattern Matrizen herangezogen und mittels K-Means[14] geclustert. Die zugehörigen Beobachtungen werden dann anschließend aufgrund des Ergebnis von K-Means zusammengefügt. Das kann dazu führen, dass vor allem Beobachtungen, die weiter weg von ihrem dominanten Pattern entfernt sind, nicht dem idealen Cluster zugewiesen werden, sollten sie zufälligerweise in einer Partition verortet sein, in der dieser Cluster nicht entdeckt wird. Dies könnte vor allem bei Daten mit einer nicht sehr klar getrennten Clusterstruktur und viel Rauschen problematisch sein, wohingegen bei einer klar gegebenen Clusterstruktur keine größeren Probleme zu erwarten sind.

Ähnlich wie bei Sorted Neighborhood gibt es das Problem, dass im Vorhinein ein Wert für den zulässigen Radius ϵ festgelegt werden muss. Aus einem kleinen Wert folgen viele kleine Cluster, während ein großer Wert zu wenigen großen Clustern führt. Koyuturk et al. (2005) geben für den Radius keinen Richtwert an. Daher wird wiederum der Wert experimentell ermittelt, indem für sämtliche Faktorstufen des Studiendesigns der

[14]Für den Grund der Verwendung von K-Means siehe Abschnitt 7.7.

Algorithmus angewandt und die Werte für ϵ auf jeweils 20, 50, 100 und 200 gesetzt wurden. Die Anzahl der Beobachtungen wurde auf 10.000 begrenzt und die Anzahl der Variablen auf 100. Es zeigten sich kaum Unterschiede bezüglich der Qualität des Clusterings. Die Differenz des besten Wertes für ϵ (20) und dem schlechtesten (200) betrug im Durchschnitt über alle Faktorstufen lediglich 0,03 für den F-Score. Bezüglich der Laufzeit schnitt ein ϵ von 20 jedoch wesentlich schlechter ab und benötigte im Durchschnitt das 6,37-fache der Zeit. Die anderen Parameter waren sich sowohl in Qualität als auch in Laufzeit nahezu identisch, weshalb der Wert für ϵ in der Simulation gleich dem Wert für die Anzahl der Variablen gesetzt wird.

7.13. Canopy Clustering

Canopy Clustering (McCallum et al., 2000) ist ein simpler Algorithmus, welcher jedoch aufgrund seiner geringen Komplexität oftmals als Preclustering Schritt für komplexere Clusteralgorithmen verwendet wird oder auch als Blocking-Algorithmus im Record Linkage. Ziel ist es, die Beobachtungen auf effiziente Weise in Gruppen einzuteilen (sogenannte *Canopies*), welche die Grundlage für eine weitere Prozessierung der Daten bilden. Wird Canopy Clustering für sich genommen, entsprechen die resultierenden Canopies den jeweiligen Clustern. Sollte beispielsweise ein komplexer Clusteralgorithmus folgen, welcher auf der Erstellung einer Distanzmatrix zwischen den Beobachtungen basiert, können anstatt der sehr komplexen Berechnung der kompletten Distanzmatrix zwischen jeweils allen Beobachtungen, diese Berechnungen nur auf die Beobachtungen beschränkt werden, welche innerhalb eines Canopys liegen. Analog würde beim Record Linkage ein Canopy ein Block darstellen, innerhalb dessen nach den Records gesucht wird, die zusammengeführt werden sollen. Aufgrund der Tatsache, dass diese Canopies auf eine Art und Weise gebildet werden, dass Beobachtungen auch in mehreren Canopies landen können, wird Canopy Clustering auch zur Familie der Fuzzy Clustering Algorithmen gezählt.

Der Ablauf des Algorithmus ist wie folgt:

1. Es wird eine Liste initialisiert, welche die Beobachtungen des Datensatzes in zufälliger Reihenfolge aufnimmt. Zusätzlich werden 2 Thresholds T_1 und T_2 gewählt. Dabei ist $T_1 \geq T_2$ zu wählen.

2. Der Liste der Beobachtungen wird eine Beobachtung entnommen. Diese Beobachtung gründet ein neues Canopy und stellt gleichzeitig das Center dieses Canopy dar. Anschließend wird die Distanz dieser Beobachtung zu jeweils allen anderen Beobachtungen der Liste berechnet.

3. Alle Beobachtungen, die innerhalb des Thresholds T_1 liegen, werden dem Canopy zugefügt.

4. Alle Beobachtungen, die innerhalb des Thresholds T_2 liegen, werden zusätzlich von der Liste gelöscht.

5. Sollte die Liste leer sein, endet der Algorithmus an dieser Stelle. Andernfalls werden die Schritte 2 bis 5 wiederholt.

Demnach bilden Beobachtungen, welche innerhalb von T_1 liegen, aber nicht innerhalb von T_2, selbst wiederum Canopy Center und können somit in mehreren Canopies liegen. Dieser Umstand lässt sich umgehen, indem die beiden Thresholds gleichgesetzt werden. Jede Beobachtung liegt dann in nur einem Canopy und damit auch nur einem Cluster (Christen, 2012, S. 91). Daher werden nachfolgend die Thresholds T_1 und T_2 einfach als Threshold T bezeichnet.

Wird der Algorithmus als Preclustering Schritt für einen anderen Clusteralgorithmus verwendet, besteht zusätzlich die Möglichkeit, nicht direkt das Distanzmaß des komplexeren Clusteralgorithmus zu verwenden, sondern ein einfacheres, welches schneller berechnet werden kann, auch wenn es gegebenenfalls ungenauer ist. McCallum et al. (2000, S. 171) geben als Beispiel das Clustern von Dokumenten an. Dabei kann dann für das Canopy Clustering zum Beispiel nur das Vorhandensein bestimmter gemeinsamer Wörter verwendet werden, während dann im nachfolgenden Clusteralgorithmus ein genaueres Maß verwendet wird, welches aufwändiger zu berechnen ist. Da jedoch in dieser Arbeit die Performance des Canopy Clustering selbst als eigenständiger Clusteralgorithmus evaluiert werden soll, wird direkt das genauere Distanzmaß verwendet (siehe Abschnitt 7.2).

Als Parallelisierungsstrategie wird wiederum ein Divide-and-Conquer Ansatz gewählt. Der Datensatz wird randomisiert in so viele Partitionen unterteilt, wie Prozessorkerne verwendet werden sollen und dann jeweils prozessiert. Jeder Prozessorkern gibt dann ein eigenes Clustering der entsprechenden Partition zurück. Anschließend werden die Canopy

Center der resultierenden Clusterings selbst nochmals mittels K-Means[15] geclustert und jeweils alle angehörigen Beobachtungen der Canopies, welche zusammengefügt wurden, in den zugehörigen Cluster verschoben.

7.14. A General Model for Clustering Binary Data

Li (2005) beschreibt eine Methode, die schlicht *A General Model for Clustering Binary Data* genannt wird. Das vorgestellte Konzept kann je nach Annahme sowohl als Biclusteringmethode, als auch als traditionelle Clusteringmethode verwendet werden. Liegt das Interesse im Biclustering der Daten, kann eine Variante der Methode verwendet werden, um sowohl die Beobachtungen (Datenzeilen) zu clustern, als auch die vorhandenen Variablen (Datenspalten). Diese Variante wird als *Two Side Clustering* beschrieben (Li, 2005, S. 189). Sollte die Annahme getroffen werden, dass jedem Cluster, bestehend aus Beobachtungen, genau ein Cluster, bestehend aus Variablen, angehört, kann eine Variante genutzt werden, die mit *Block Diagonal Variant* bezeichnet wird.

7.14.1. Two Side Clustering

Die Methode basiert dabei auf einer Approximation der originalen Datenmatrix, wodurch eine Ähnlichkeit zu Proximus (siehe Abschnitt 7.12) offensichtlich wird. Diese Approximation der Datenmatrix wiederum wird ebenfalls durch ein alternierendes iteratives Verfahren erreicht. Li (2005, S. 189) gibt das Modell selbst mit folgender Form an:

$$\mathbf{W} = \mathbf{AXB^T} + \mathbf{E}\,, \tag{7.52}$$

wobei $\mathbf{W}$ die originale Datenmatrix darstellt und $\mathbf{A}$ ist eine $n \times K$ binäre Matrix, welche die Clusterzugehörigkeit von Beobachtung i zu Cluster k anzeigt. Daraus folgt: $\sum_{k=1}^{K} a_{ik} = 1$. $\mathbf{B}$ wiederum ist eine $m \times C$ binäre Matrix, welche analog die Clusterzugehörigkeit von Variable m zu Cluster c anzeigt. Daraus folgt wiederum analog: $\sum_{c=1}^{C} b_{jc} = 1$. K beschreibt die Anzahl der Cluster der Beobachtungen und C die Anzahl der Cluster der Variablen. Die Matrix $\mathbf{X}$ ist eine $K \times C$ Matrix, welche die Assoziation der Beobachtungen zu den Variablen innerhalb der jeweiligen Cluster beschreibt. $\mathbf{E}$ ist eine Matrix der

[15] Für den Grund der Verwendung von K-Means siehe wiederum Abschnitt 7.7.

Fehlerkomponente. Eine möglichst gute Approximation der Datenmatrix geht mit der Minimierung der Frobeniusnorm der Datenmatrix abzüglich der Approximation einher:

$$O(\mathbf{A}, \mathbf{X}, \mathbf{B}) = \|\mathbf{W} - \hat{\mathbf{W}}\|_F^2 = \text{Trace}\left[(\mathbf{W} - \hat{\mathbf{W}})(\mathbf{W} - \hat{\mathbf{W}})^{\mathbf{T}}\right] \tag{7.53}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m}(w_{ij} - \hat{w}_{ij})^2 \tag{7.54}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m}\left(w_{ij} - \sum_{k=1}^{K}\sum_{c=1}^{C} a_{ik}b_{jc}x_{kc}\right)^2 . \tag{7.55}$$

Dies stellt das Kriterium dar, dessen Minimierung den später folgenden iterativen Algorithmus zur Bestimmung der optimalen Approximation konvergieren lässt.

Aus Formel 7.53 folgt:

$$O(\mathbf{A}, \mathbf{X}, \mathbf{B}) = \|\mathbf{W} - \hat{\mathbf{W}}\|_F^2 \tag{7.56}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m}\left(w_{ij} - \sum_{k=1}^{K}\sum_{c=1}^{C} a_{ik}b_{jc}x_{kc}\right)^2 \tag{7.57}$$

$$= \sum_{k=1}^{K}\sum_{c=1}^{C}\sum_{i\in P_k}\sum_{j\in Q_c}(w_{ij} - x_{kc})^2 . \tag{7.58}$$

Wobei $P = \{P_1, P_2, \cdots, P_K\}$ die Partition der Beobachtungen in K Cluster darstellt und entsprechend $i \in P_k$ die Beobachtung i in Cluster P_k beschreibt. Analog stellt $Q = \{Q_1, Q_2, \cdots, Q_C\}$ die Partition der Variablen in C Cluster dar und damit entsprechend $j \in Q_c$ die Variable j in Cluster Q_c. Um nun die optimalen Werte für $\mathbf{A}$, $\mathbf{X}$ und $\mathbf{B}$ zu erhalten, kann wie folgt vorgegangen werden: Für gegebenes P_k und Q_c ergibt sich das Optimum für $\mathbf{X}$ als:

$$x_{kc} = \frac{1}{p_k q_c}\sum_{i\in P_k}\sum_{j\in Q_c} w_{ij} . \tag{7.59}$$

Wobei mit p_k die Größe des k-ten Cluster angegeben ist und analog mit q_c die Größe des c-ten Clusters.

Gegeben $\mathbf{X}$ und $\mathbf{B}$, lässt sich die optimale Matrix $\mathbf{A}$ berechnen mit:

$$\hat{a}_{ik} = \begin{cases} 1 & \text{falls } \sum_{c=1}^{C} \sum_{j \in Q_c} (w_{ij} - x_{kj})^2 < \sum_{c=1}^{C} \sum_{j \in Q_c} (w_{ij} - x_{lj})^2 \\ & \text{für } l = 1, \cdots, K, l \neq k \\ 0 & \text{andernfalls .} \end{cases} \tag{7.60}$$

Analog dazu lässt sich, gegeben $\mathbf{X}$ und $\mathbf{A}$, die optimale Matrix $\mathbf{B}$ berechnen mit:

$$\hat{b}_{jc} = \begin{cases} 1 & \text{falls } \sum_{k=1}^{K} \sum_{i \in P_k} (w_{ij} - x_{ic})^2 < \sum_{k=1}^{K} \sum_{i \in P_k} (w_{ij} - x_{il})^2 \\ & \text{für } l = 1, \cdots, C, l \neq c \\ 0 & \text{andernfalls .} \end{cases} \tag{7.61}$$

Der Ablauf des Algorithmus sieht vor, dass zunächst die Matrizen $\mathbf{A}$ und $\mathbf{B}$ initialisiert werden müssen, um $\mathbf{X}$ berechnen zu können (Li, 2005, S. 189). Leider wird dafür keine Vorgehensweise vorgeschlagen. Eine Möglichkeit wäre jedoch, die Matrizen mit zufälligen Werten aus $\{0, 1\}$ zu füllen[16], wobei darauf zu achten ist, dass die Zeilensummen jeweils 1 ergeben, damit gewährleistet ist, dass jede Beobachtung bzw. jede Variable jeweils nur einem Cluster angehört.

Der Algorithmus selbst hat folgende Schritte:

1. Initialisiere $\mathbf{A}$ und $\mathbf{B}$.
2. Berechne $\mathbf{X}$.
3. Berechne Optimierungskriterium $O(\mathbf{A}, \mathbf{X}, \mathbf{B})_0$.
4. Update $\mathbf{A}$ und $\mathbf{B}$.
5. Berechne wiederum $\mathbf{X}$.
6. Berechne Optimierungskriterium $O(\mathbf{A}, \mathbf{X}, \mathbf{B})_1$.
7. Sollte $O(\mathbf{A}, \mathbf{X}, \mathbf{B})_1 < O(\mathbf{A}, \mathbf{X}, \mathbf{B})_0$ setze $O(\mathbf{A}, \mathbf{X}, \mathbf{B})_1 = O(\mathbf{A}, \mathbf{X}, \mathbf{B})_0$ und kehre zu Schritt 4 zurück. Ansonsten konvergiert der Algorithmus an dieser Stelle.

[16] Dieses Vorgehen ähnelt dem Vorgehen bei K-Means eine Startkonfiguration zu wählen, in der die Clusterzentren zufällig gewählt sind und dann während des algorithmischen Prozesses aktualisiert und optimiert werden.

7.14.2. Block Diagonal Variant

Li (2005, S. 188) argumentieren, dass die Assoziation zwischen den Beobachtungen und den Variablen oftmals symmetrischer Natur ist und sich diese Eigenschaft von einem Clusteralgorithmus zunutze gemacht werden kann. Eine symmetrische Assoziation zwischen Beobachtungen und Variablen impliziert, dass jedem Zeilencluster genau ein Spaltencluster zugeschrieben ist und ergo jede Variable zu genau einem Zeilencluster gehört. Aus dieser Annahme folgt, dass die Matrix $\mathbf{X}$ die Form einer Identitätsmatrix annimmt und $C = K$ ist (Li, 2005, S. 190). Weiterhin wird angeführt, dass die Approximationsmatrix, welche sich bei dieser Variante zu $\mathbf{AB^T}$ vereinfacht, die Form einer block-diagonalen Matrix annimmt, sollte sie entsprechend permutiert werden.

Das Minimierungskriterium stellt nach wie vor die Frobeniusnorm der Differenz zwischen der originalen Datenmatrix und der Approximationsmatrix. Aufgrund des Wegfallens der Matrix $\mathbf{X}$ lässt sich dies auch darstellen als:

$$\begin{aligned} O(A,B) &= \left\| \mathbf{W} - \mathbf{AB^T} \right\|_F^2 \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \left(w_{ij} - \sum_{k=1}^{K} a_{ik} b_{kj} \right)^2 \\ &= \sum_{i=1}^{n} \sum_{k=1}^{K} a_{ik} \sum_{j=1}^{m} \left(w_{ij} - b_{kj} \right)^2 \\ &= \sum_{i=1}^{n} \sum_{k=1}^{K} a_{ik} \sum_{j=1}^{m} \left(w_{ij} - y_{kj} \right)^2 \\ &\quad + \sum_{k=1}^{K} n_k \sum_{j=1}^{m} \left(y_{kj} - b_{kj} \right)^2 , \end{aligned} \tag{7.62}$$

mit $y_{kj} = \frac{1}{n_k} \sum_{i=1}^{n} a_{ik} w_{ij}$ und $n_k = \sum_{i=1}^{n} a_{ik}$. Die Wahrscheinlichkeit stellt dabei y_{kj} dar, dass Variable j Teil des Clusters k ist (Li, 2005, S. 190) (jede Variable ist lediglich bedeutend für einen Cluster). Auch in dieser Variante setzt der Algorithmus auf das iterative alternierende Optimieren der beiden Matrizen, indem eine Matrix fix gehalten wird und über ein Kleinst-Quadrate-Verfahren optimiert wird. Matrix $\mathbf{A}$ wird dabei wie folgt optimiert:

$$\hat{a}_{ik} = \begin{cases} 1 & \text{falls } \sum_{j=1}^{m} (w_{ij} - b_{kj})^2 < \sum_{j=1}^{m} (w_{ij} - b_{lj})^2 \\ & \text{für } l = 1, \cdots, K, l \neq k \\ 0 & \text{andernfalls .} \end{cases} \tag{7.63}$$

Die Matrix **B** hingegen kann optimiert werden, indem folgende Gleichung minimiert wird:

$$O'(B) = \sum_{k=1}^{K} n_k \sum_{j=1}^{m} (y_{kj} - b_{kj})^2 . \tag{7.64}$$

Dies geschieht, indem $\hat{b}_{kj}$ den Wert 1 annimmt, sollte die Wahrscheinlichkeit, dass j Teil von Cluster k ist größer als 0,5 sein:

$$\hat{b}_{kj} = \begin{cases} 1 & \text{falls } y_{kj} > 1/2 \\ 0 & \text{andernfalls .} \end{cases} \tag{7.65}$$

Der Ablauf des Algorithmus ist nahezu identisch mit dem aus Abschnitt 7.14.1, mit dem Unterschied, dass das Optimieren der Matrix **X** wegfällt.

7.15. Zeitkomplexität der verwendeten Algorithmen

In Tabelle 7.6 findet sich eine Auflistung der berichteten Zeitkomplexität der jeweiligen Algorithmen. Für die Algorithmen MONA, Multibit Trees, Large Item und General Model for Clustering Binary Data sind keine Komplexitätsanalysen angegeben und können daher an dieser Stelle auch nicht berichtet werden.

Dabei ist p gleich der Anzahl der MinHash-Funktionen, b der Anzahl der Bands, a der Anzahl der Variablen, δ der Anzahl der Iterationen, w der Windowsize, r dem verwendeten Radius, f der mittleren Anzahl an Canopies in die ein Punkt fällt und c der Anzahl der Canopies.

Tabelle 7.6.: Komplexitätsanalyse

Algorithmus	Zeitkomplexität
LSH	$O(np + nb)$
MinHash	$O(np)$
CLARA	$O(k(40 + k)^2 + k(n - k))$
CLARANS	$O(kn^2)$
BUBBLE	$O(n)$
MONA	Nicht berichtet
MBT	Nicht berichtet
CLOPE	$O(nka)$
Large Item	Nicht berichtet
SCALE	$O(\delta nka)$
Sorted Neighborhood	$O(n \log n)$ wenn $w < [\log n]$ sonst $O(wn)$
Proximus	$O(rn)$
Canopy Clustering	$O(f^2 n^2/c)$
General Model	Nicht berichtet

7.16. Überprüfung der Funktionsfähigkeit der Algorithmen

Wie bereits in Kapitel 4.3 beschrieben, wurden sämtliche Algorithmen im Rahmen der Arbeit vom Verfasser implementiert. Die einzige Ausnahme bildet Proximus, dessen Kernfunktion aus dem Paket `cba` (Buchta & Hahsler, 2019) stammt. Die Parallelisierung wurde jedoch ebenfalls vom Verfasser implementiert. Laut McConnell (2004, S. 521–522) können im Quellcode kommerzieller Programme zwischen 1 und 25 Programmierfehler pro 1.000 Zeilen Code erwartet werden. Entsprechend ist es denkbar, dass auch bei der Programmierung im Rahmen dieser Arbeit Programmierfehler unterlaufen sind. Um nun zu prüfen, ob die Programme korrekt implementiert sind und Programmierfehler größtenteils ausgeschlossen werden können, wurden die Programme an einem einfachen Datensatz mit klarer Clusterstruktur und geringem Umfang (10.000 Beobachtungen und 100 Variablen) getestet. Eine Visualisierung der Daten findet sich in Abbildung 7.1.

In Tabelle 7.7 finden sich die Ergebnisse der Algorithmen mit zugehörigem Rand-Index. Wie sich sehen lässt sind alle Algorithmen in der Lage eine perfekte oder fast perfekte Clusterlösung zu produzieren. Die Implementierung der Algorithmen wird daher als funktionsfähig angesehen.

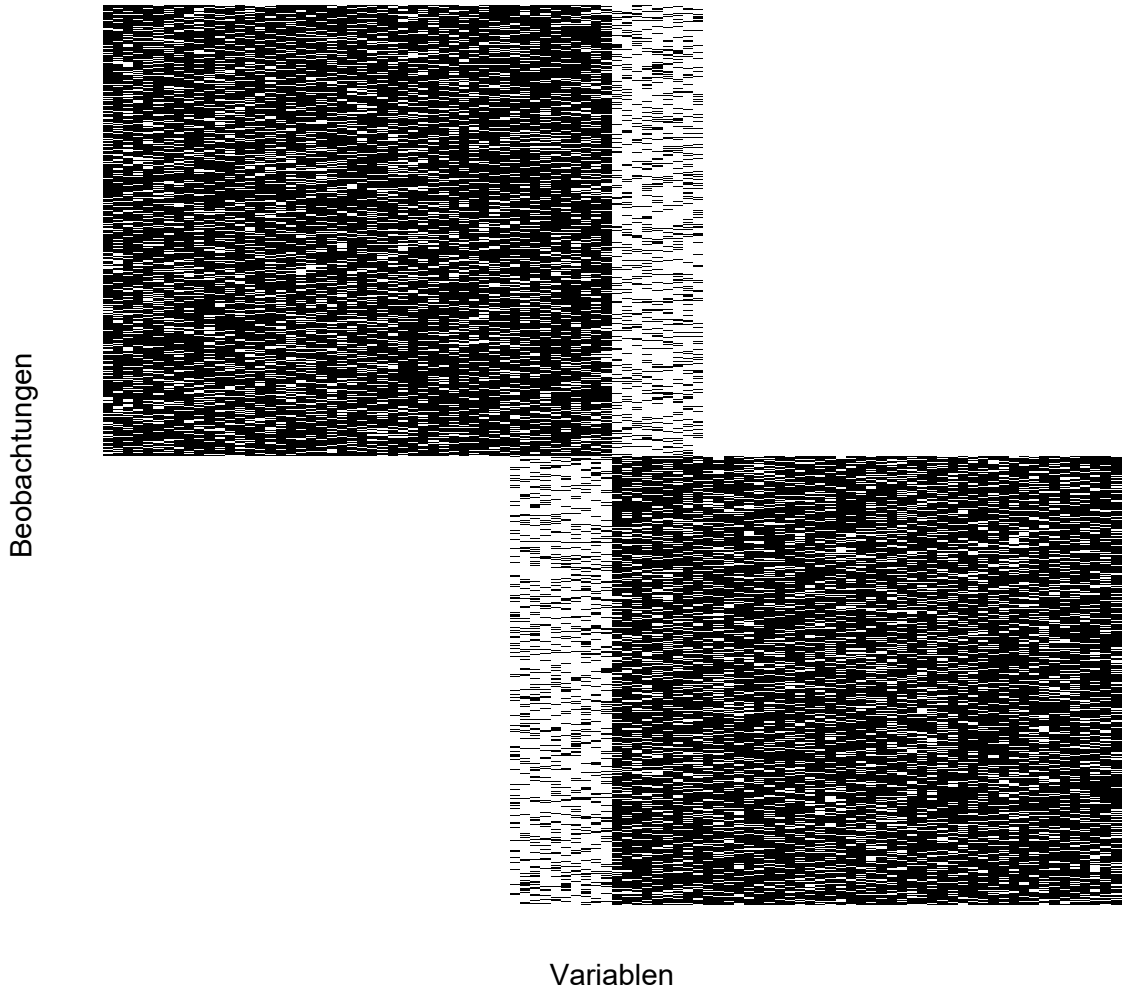


Abbildung 7.1.: Visualisierung der Daten zur Überprüfung der Algorithmen.

Tabelle 7.7.: Ergebnisse der Überprüfung der Algorithmen an einem einfachen Datensatz mit eindeutiger Clusterstruktur

Algorithmus	Rand-Index
bubble	1
clara	1
clarans	1
clope	1
large_item	1
scale	1
proximus	1
sorted_neighborhood	1
mbt	0,99
mona	0,83
lsh	0,86
minhash	0,81
canopy	1
gm2	0,99
gm1	0,99

8. Nicht verwendete Algorithmen

Zusätzlich zu den verwendeten Algorithmen werden auch einige Algorithmen betrachtet, welche aus unterschiedlichen Gründen nicht in dieser Arbeit berücksichtigt werden. Diese werden im Folgenden aufgeführt und kurz beschrieben. Ebenfalls wird jeweils eine Begründung genannt, warum sie nicht für diese Arbeit geeignet sind.

8.1. CACTUS

Ganti, Gehrke und Ramakrishnan (1999) stellen den Algorithmus *CACTUS* (Clustering Categorical Data Using Summaries) vor, welcher speziell für das Clustering von Daten mit nominalem Skalenniveau entwickelt wurde. Der Kern des Algorithmus besteht darin Clusterprojektionen der einzelnen Variablen zu finden, welche aus einem bestimmten Set an Ausprägungen auf der jeweiligen Variable bestehen. Analog zu dem Clustering numerischer Werte, in dem die einzelnen Intervallregionen der einzelnen Dimensionen einen Raum aufspannen, in dem sich besonders viele Beobachtungen befinden und damit einen Cluster bilden, entspricht die Clusterprojektion eines Clusters auf einer einzelnen Variable, bestehend aus einem Set an möglichen Ausprägungen auf dieser Variable, diesem Intervall (Ganti, Gehrke & Ramakrishnan, 1999, S. 74). Aufgrund der Tatsache, dass der Algorithmus verlangt, dass das Set, aus welchem eine Clusterprojektion besteht größer 1 sein muss, ist CACTUS nicht für das Clustering binärer Datensätze geeignet. Es würde dazu führen, dass jede Clusterprojektion jeder Variable aus der kompletten Domain des Attributes bestehen würde.

Um diese Intervalle zu finden, werden sowohl *Inter-* als auch *Intraattributesummaries* verwendet. Diese bestehen aus der Anzahl von *Attribute/Werte Paaren*, deren Häufigkeit (auch *Support* genannt) über einem gegebenen Schwellenwert liegt (Ganti, Gehrke & Ramakrishnan, 1999, S. 75). Der Hauptteil des Algorithmus besteht aus drei Phasen: Der *Summarization Phase*, der *Clustering Phase* und der *Validation Phase*. In der Summarization Phase werden die Inter- und Intraattributesummaries gebildet. Die

Clustering Phase ermittelt die Clusterprojektionen der einzelnen Variablen und bildet damit *Candidate Cluster* und in der Validation Phase wird dann auf Grundlage des Supports der einzelnen Cluster entschieden, welche Candidate Cluster akzeptiert werden und damit das finale Clustering bilden.

8.2. COOLCAT

Ein weiterer Algorithmus, welcher speziell für das Clustering von Daten mit kategorialen Variablen entwickelt wurde, ist *COOLCAT* (Barbará et al., 2002). Die Grundidee der Methode ist die kombinierte Entropie[1] der Clusterlösung zu minimieren. Das *CAT* steht dabei für *Categorical* und das *COOL* spielt auf das Minimieren der Entropie an, was Barbará et al. (2002, S. 582) als „Cooling“ bezeichnen.

Das Vorgehen des Algorithmus besteht darin, die Entropie innerhalb der Cluster zu minimieren. Die einzelnen Werte der Entropie je Cluster werden dann gewichtet mit dem Anteil der Elemente im Cluster am Gesamtdatensatz zusammenaddiert und stellt somit das Optimierungskriterium dar, welches es zu minimieren gilt. Die Art und Weise der Minimierung erinnert dabei stark an das Vorgehen bei SCALE, CLOPE und Large Item. Jede Beobachtung wird einzeln betrachtet und einmal jedem vorhandenen Cluster zugewiesen. Dabei wird jedes Mal der Wert für das Optimierungskriterium berechnet und die Beobachtung letztendlich dem Cluster zugewiesen, welcher das Optimierungskriterium minimiert. Ähnlich wie bei SCALE wird zunächst eine Stichprobe gezogen, innerhalb welcher ein initiales Clustering erstellt wird. Die Anzahl der Cluster k muss dabei vorgegeben werden. Innerhalb dieser Stichproben werden die k Beobachtungen gesucht, welche die höchste paarweise Entropie aufweisen und damit als initiale Cluster fungieren (Barbará et al., 2002, S. 585). Die Stichprobengröße wird dabei durch die sogenannten *Chernoff bounds* (Chernoff, 1952) festgelegt. Auch Barbará et al. (2002) erkennen den Einfluss der Reihenfolge der Beobachtungen und schlagen vor, die Beobachtungen, welche einen schlechten Fit haben, aus der Clusterlösung zu entnehmen und erneut zu clustern.

Aufgrund der starken Ähnlichkeit zu den genannten Methoden SCALE, CLOPE und Large Item, welche bereits in dieser Arbeit genutzt werden, findet COOLCAT keine Verwendung.

[1] Für eine Definition von Entropie siehe Abschnitt 6.2.

8.3. ROCK

RObust Clustering using linKs (ROCK, Guha et al., 2000) ist eine hierarchische und agglomerative Cluster-Methode, welche ebenfalls speziell für nominale Merkmale entwickelt wurde und ebenfalls prinzipiell für Datensätze mit binären Variablen geeignet ist. Die Grundidee ist dabei nicht nur die Distanz zwischen den Datenpunkten als Grundlage für das Clustering zu verwenden, sondern die Anzahl gemeinsamer Nachbarpunkte eines Beobachtungspaares, welche als *Links* bezeichnet werden.

Dazu muss zunächst geklärt werden, wie ein Link definiert ist. Eine Beobachtung gilt als Nachbar einer anderen Beobachtung, wenn die Ähnlichkeit einen vorher festgelegten Threshold θ überschreitet. Die Ähnlichkeit zwischen den beiden Beobachtungen kann dabei mittels einer arbiträren Ähnlichkeitsfunktion berechnet werden[2]. Hat ein Beobachtungspaar eine gemeinsame Nachbarbeobachtung, wird diese als Link bezeichnet. Ziel ist es nun Cluster zu bilden, deren Beobachtungen möglichst viele gemeinsame Links haben und gleichzeitig möglichst wenige Links zwischen den Beobachtungen verschiedener Cluster vorhanden sind. Wie bei anderen hierarchischen und agglomerativen Clustermethoden startet jede Beobachtung als einzelner Cluster und diese werden iterativ zusammengefasst bis entweder die vorher festgelegte Anzahl an Cluster k erreicht ist oder keine Links mehr zwischen den Clustern existieren. Damit die korrekten Cluster zusammengefasst werden, steht ein Optimierungskriterium zur Verfügung, das auf der Summe der Links zwischen den Clustern basiert und durch die erwartete Anzahl an Links zwischen den Clustern normiert wird (Guha et al., 2000, S. 517).

Um die Komplexität zu verringern, wird zunächst eine Stichprobe gezogen, auf welcher der Algorithmus ausgeführt wird. Aus den resultierenden Clustern wird wiederum nur eine Teilmenge verwendet, anhand derer die restlichen Beobachtungen des Datensatzes geclustert werden. Auch wenn durch dieses Vorgehen die Komplexität des Algorithmus verringert wird, ist der Algorithmus dennoch als sehr komplex anzusehen. Schließlich muss für jedes Beobachtungspaar die Anzahl an Links berechnet werden, welche selbst wiederum eine komplette Distanzmatrix aller Beobachtungen voraussetzen. Angegeben wird der Algorithmus mit einer Komplexität von $O\left(n^2 + nm_m m_a + n^2 \log n\right)$. Dabei stellt m_m die maximale Anzahl an Nachbarbeobachtungen dar und m_a die durchschnittliche

[2]Aufgrund dieser Tatsache kann der Algorithmus prinzipiell auch für Daten jeglichem Skalenniveaus verwendet werden, die es zulassen eine Ähnlichkeitsfunktion zwischen zwei Beobachtungen anzuwenden.

Anzahl an Nachbarbeobachtungen. Für die Verwendung in dieser Arbeit wird dies als zu hoch angesehen.

8.4. LIMBO

Ähnlich wie BUBBLE basiert auch der Algorithmus *LIMBO* (scaLable InforMation BOttleneck, Andritsos et al., 2004) auf der Idee von BIRCH eine Baumstruktur zu bilden, dessen Leaf-Nodes die Cluster darstellen und alle Nodes zusammengefasste Statistiken über den Cluster enthalten. Im Falle von Non-Leaf-Nodes enthalten diese zusammengefasste Statistiken über die Nodes, die unterhalb der entsprechenden Node angesiedelt sind. Eine weitere Gemeinsamkeit ist, dass jede Beobachtung einzeln in den Baum gegeben wird und entlang der Nodes zu einer Leaf-Node geleitet wird. In Phase 2 werden die entstandenen Cluster mittels eines arbiträren hierarchischen agglomerativen Clusteralgorithmus zusammengeführt und in der dritten Phase die Beobachtungen entsprechend ihrer Clusterzugehörigkeit gelabelt.

Der Hauptunterschied zu BUBBLE besteht in den zusammengefassten Statistiken und der verwendeten Heuristik bei der Berechnung der Distanzen. Dabei bedienen sich Andritsos et al. (2004) beim *Information Bottleneck*-Framework (Tishby et al., 2000) und dem darauf basierenden *AIB*-Algorithmus (Slonim & Tishby, 1999). Dabei besteht das Ziel darin, durch die Zugehörigkeit zu Cluster C möglichst gute Vorhersagen über die Ausprägungen der (kategorialen) Variablen A innerhalb des Clusters machen zu können. Entsprechend soll die Mutual Information[3] $I(A;C)$ maximiert werden. Die Distanzmetrik, die verwendet wird, besteht aus dem Verlust an Informationsgehalt in $I(A;C)$ sollten zwei Cluster[4] zusammengefasst werden.

Entsprechend bestehen die zusammengefassten Statistiken, hier *Distributional Cluster Feature (DCF)* genannt, welche in den Nodes gespeichert und aktualisiert werden, aus den Wahrscheinlichkeiten $p(c)$ für Cluster c und der konditionalen Wahrscheinlichkeitsverteilung $p(A|c)$ der Ausprägungen gegeben Cluster c. Mithilfe dieser Werte lassen sich der Verlust des Informationsgehaltes beim Zusammenfassen zweier Cluster berechnen.

[3] Für Erläuterungen zu Mutual Information siehe Abschnitt 6.2.
[4] Diese können auch aus einzelnen Beobachtungen bestehen.

Aufgrund der starken konzeptionellen Ähnlichkeit zu BUBBLE wird der Algorithmus in dieser Arbeit nicht verwendet.

8.5. CLICKS

Eine weitere graphbasierte Clustermethode nennt sich *CLICKS* (Zaki et al., 2007) und steht für *CLusterIng of Categorical data via maximal K-partite cliques*. Das Grundprinzip der Methode besteht daraus, Teilmengen von Merkmalskombinationen zu finden, welche häufiger auftauchen als der Erwartungswert vermuten lassen würde. Genaugenommen werden Merkmalskombinationen gesucht, deren Support $\sigma(S)$ (siehe Abschnitt 7.9) einen gewissen Schwellenwert überschreiten. Ist dies der Fall, wird von einem „dense subspace“ (Zaki et al., 2007, S. 52) gesprochen. Formell tritt dies ein, wenn $\sigma(S) \geqslant \alpha \cdot E[\sigma(S)]$ ist, wobei $E[\sigma(S)]$ den Erwartungswert von $\sigma(S)$ bei angenommener stochastischer Unabhängigkeit der Merkmalsausprägungen darstellt und α ein vorher festgelegter Skalierungsfaktor für $E[\sigma(S)]$ ist. $E[\sigma(S)]$ selbst ist definiert durch die Anzahl der Beobachtungen multipliziert mit den jeweiligen Realisationswahrscheinlichkeiten der Merkmalsausprägungen, welche als gleichverteilt angenommen werden.

Angenommen jede Merkmalsausprägung stellt einen Knotenpunkt dar und Verbindungen zwischen Knotenpunkten bestehen nur, wenn es sich bei diesen Merkmalskombinationen um ein dense subspace handelt, stellt ein *Clique* einen Merkmalscluster dar. Entsprechend würden alle Beobachtungen zu einem Cluster zusammengefasst, welche diese Merkmalskombinationen aufweisen. Da CLICKS von Zaki et al. (2007, S. 61) mit einer exponentiellen Zeitkomplexität angegeben wird, findet der Algorithmus in dieser Arbeit keine Berücksichtigung.

8.6. Agglomerative Hierarchical Clustering

Die populärsten Methoden der Clusteranalyse bestehen in den verschiedenen Varianten der agglomerativen hierarchischen Clusterverfahren (Kaufman & Rousseeuw, 2005). Jede Beobachtung stellt am Anfang des Algorithmus einen einzelnen Cluster dar und es werden iterativ in jedem Schritt die beiden Cluster zusammengefasst, welche die geringste Distanz zueinander aufweisen. Dies wird solange wiederholt bis nur noch ein Cluster übrig ist bzw. die gewünschte Anzahl an Clustern erreicht ist. Die verschiedenen verfügbaren

Algorithmen Single-Linkage, Complete-Linkage und Average-Linkage[5] unterscheiden sich dabei wie die paarweise Distanz zwischen Clustern definiert ist. Bei Single-Linkage sind das die beiden Punkte des jeweils anderen Clusters, deren Distanz zueinander minimal ist. Bei Complete-Linkage wird entsprechend gegenteilig die Distanz der jeweils am weitest entferntesten Punkte minimiert. In der Variante Average-Linkage wird die mittlere Distanz aller Punkte der unterschiedlichen Cluster zueinander als Kriterium genutzt.

Die genannten Algorithmen können mit jeder beliebigen Distanzmetrik verwendet werden. Daher sind sie prinzipiell für jeden Datentyp geeignet, der es erlaubt, eine angemessene Distanzberechnung zwischen zwei Datenpunkten zuzulassen. Entsprechend können die Methoden auch für das Clustering binärer Daten verwendet werden. Eine Ausnahme stellt die ebenfalls weit verbreitete Variante der Ward-Methode dar (Ward, 1963), welche auf der Agglomeration von Clustern durch Minimierung der internen Varianz der Datenpunkte selbst beruht. Dies ist nur für Daten metrischen Skalenniveaus sinnvoll. Selbiges gilt ebenfalls für das *Zentroid*- und das *Median*-Verfahren, in denen entsprechend der Zentroid bzw. der Median der Cluster Ausgangspunkt für die Distanzberechnung zwischen den Clustern ist.

Darüber hinaus ist es den Methoden gemein, dass sie alle auf kompletten Distanzmatrizen des Datenmaterials beruhen. Aufgrund dieser Tatsache ist die Komplexität dieser Algorithmen sehr hoch und sie werden deshalb nicht in dieser Arbeit berücksichtigt.

Auch der Algorithmus *CHAMELEON* (Karypis et al., 1999) verfolgt eine agglomerative hierarchische Vorgehensweise. Hier wird jedoch zunächst ein k-Nearest Neighbor Graph erstellt, der in feine Subcluster geteilt und dann auf hierarchische Weise zusammengefasst wird. Da auch dieser Algorithmus das Erstellen einer Distanz- bzw. einer Ähnlichkeitsmatrix voraussetzt (Gan et al., 2007, S. 203), wird auch dieser Algorithmus nicht berücksichtigt.

8.7. Squeezer

Nur kurz angerissen wird der Algorithmus *Squeezer* (He et al., 2002). Dieser verfolgt im Wesentlichen die gleiche Strategie wie Large Item, CLOPE und SCALE. Jede Be-

[5] Bei Kaufman und Rousseeuw (2005, S. 47) auch *unweighted pair-group average method (UPGMA)* genannt.

obachtung wird einzeln betrachtet und aufgrund eines Optimierungskriteriums einem Cluster zugeordnet. Im Falle von Squeezer ist dies die Distanz zu den jeweiligen Clustern. Die erste Beobachtung bildet dabei den ersten Cluster. Überschreitet die Distanz zum nächstliegenden Cluster einen bestimmten Schwellenwert, bildet die aktuell betrachtete Beobachtung einen neuen eigenen Cluster. Aufgrund der starken Ähnlichkeit zu den bereits genannten Algorithmen, findet Squeezer in dieser Arbeit keine Berücksichtigung.

8.8. POPC (Powered Outer Probabilistic Clustering)

Der Algorithmus *POPC* (Powered Outer Probabilistic Clustering, Taraba, 2017) geht implizit von einer Clusterstruktur aus, welche durch die Zugehörigkeit der einzelnen Variablen zu einzelnen Clustern definiert ist. Beobachtungen innerhalb eines Clusters haben demnach vermehrt die Ausprägung 1 auf den jeweiligen dem Cluster zugehörigen Variablen, während Beobachtungen außerhalb des Clusters primär die Ausprägung 0 haben. Der Algorithmus benötigt ein Clustering als Startkonfiguration. Taraba (2017) schlägt dabei vor K-Means zu nutzen, um eine Clusterlösung zu erhalten, die als Startkonfiguration dienen kann. Grundsätzlich sollten aber auch andere Algorithmen verwendet werden können. Der Grundgedanke des Algorithmus liegt darin durch die Anteile der Einsen einer Variable innerhalb der initialen Clusterlösung, die Clusterzugehörigkeit der Variablen zu extrahieren, indem die Beobachtungen zwischen den Cluster hin- und hergeschoben werden und versucht wird, eine Zielfunktion zu maximieren, welche durch Anteile der Einsen innerhalb eines Clusters maximiert wird.

Die Wahrscheinlichkeit einer Variable f_i zu einem bestimmten Cluster k zu gehören, kann wie folgt berechnet werden:

$$p\left(cl\left(f_i\right)=k\right)=\frac{c\left(s_j\left(f_i\right)=1, cl\left(s_j\right)=k\right) C_m+1}{c\left(f_i=1\right) C_m+N}, \tag{8.1}$$

mit

c: Zählfunktion
cl: Clusterzugehörigkeit
N: Anzahl der Cluster
$k \in \{1, \cdots, N\}$: Clusternummer

s_j: Beobachtung j
$s_j\,(f_i) \in \{0, 1\}$: Ausprägung von Beobachtung s_j auf Variable j_i
C_m: Konstante zum multiplizieren. Taraba (2017) schlägt einen Wert von 1000 vor.

Daraus folgt:

$$\sum_{k=1}^{N} p\,(cl\,(f_i) = k) = 1\,, \tag{8.2}$$

und

$$\sum_{i=1}^{F} \sum_{k=1}^{N} p\,(cl\,(f_i) = k) = F\,, \tag{8.3}$$

wobei F die Anzahl der Variablen darstellt. Durch das Potenzieren der Wahrscheinlichkeit p für $cl\,(f_i) = k$ mit $P > 1$ kann die Zielfunktion J gebildet werden:

$$J = \sum_{i=1}^{F} J\,(f_i) = \sum_{i=1}^{F} \sum_{k=1}^{N} p^P\,(cl\,(f_i) = k) \leq F\,. \tag{8.4}$$

Ohne das Potenzieren von p resultiert sonst durch das Summieren der Wahrscheinlichkeiten über Variablen und Cluster in jedem Fall die Konstante F. Ein hoher Wert für J impliziert eine hohe Wahrscheinlichkeit für die einzelnen Variablen den korrekten Clustern zugeordnet zu sein[6]. Das Maximieren dieser Funktion durch das Verschieben der Beobachtungen zwischen den Clustern führt somit zu der besten Clusterlösung. Der Wert J steigt, je eindeutiger die Clusterstruktur ist, indem die Ausprägungen der Variablen vor allem innerhalb der zugehörigen Cluster den Wert 1 annehmen. Der Algorithmus ist als Implementation in Python verfügbar (siehe Taraba, 2022). Er wurde an einem Beispieldatensatz mit Szenario *Einfach* (siehe Abschnitt 4) getestet, wobei die Laufzeit als zu lang bewertet wurde. Daher ist der Algorithmus in dieser Arbeit nicht verwendet worden.

[6]Es sei daran erinnert, dass der Algorithmus davon ausgeht, dass einzelne Variablen einzelnen Clustern angehören.

9. Ergebnisse

Bevor in diesem Abschnitt die Ergebnisse der verwendeten Algorithmen evaluiert werden, wird zunächst dargelegt, auf welche Weise die Parameter für die jeweiligen Algorithmen festgelegt wurden. Danach folgt ein Vergleich der parallelen vs. der sequenziellen Ausführung der Clusteralgorithmen.

9.1. Wahl der Parameter

Viele der hier vorgestellten verwendeten Algorithmen benötigen einige Parameter als Funktionsargumente. Dabei wird bei Parametern, welche einen Einfluss auf die Laufzeit des Algorithmus haben, ein der Simulation vorgelagertes Experiment durchgeführt, um den sinnvollsten Wert zu ermitteln. Die Ergebnisse dieser Experimente sind in den Beschreibungen des jeweiligen Algorithmus in Abschnitt 7 dargelegt. Die restlichen Parameter werden, wenn in Abschnitt 7 nicht anders angegeben, in einer der Anwendung des Algorithmus vorgelagerten Evaluation festgelegt. Dazu wird bei jeder Iteration des Simulationsdurchlaufs für den jeweiligen Algorithmus eine Stichprobe des verwendeten Datensatzes mit Größe $n = 2.000$ gezogen. Auf diese Stichprobe wird dann der jeweilige Algorithmus angewendet und eine Reihe verschiedener sinnvoller Werte für die jeweiligen Parameter ausprobiert und das Ergebnis evaluiert. Die Spannweiten der evaluierten Werte finden sich in Tabelle 9.1. Die Werte der Parameter, die die qualitativ besten Ergebnisse ergeben, werden dann für die eigentliche Anwendung auf dem vollen Datenmaterial genutzt. Nahezu alle Algorithmen benötigen ebenfalls die zu erwartende Anzahl der Cluster als Parameter. Dazu wird jedes mal die korrekte Anzahl an Clustern übergeben.

9.2. Parallele vs. sequenzielle Ausführung

Um zu prüfen, inwiefern sich die parallelen Varianten der in Abschnitt 7 vorgestellten Algorithmen im Gegensatz zu den sequenziellen Versionen im Hinblick auf Qualität des Outputs und der benötigten Zeit unterscheiden, wird experimentell in einer Simulation

Tabelle 9.1.: Spannweite der Parameter der vorgelagerten Experimente

Algorithmus	Spannweite der Parameter
BUBBLE	L = {10, 20, 50, 100} Threshold = {0,5, 0,6, 0,7, 0,8}
LSH	Bands = {2, 3,. . . , 14, 15} Anzahl Hash = {2, 4, 6,. . . , 48, 50}
MinHash	Anzahl Hash = {2, 4, 6,. . . , 48, 50}
Canopy	Threshold = {0,3, 0,35, . . . , 0,55, 0,6}

ermittelt. Es werden sämtliche Faktorstufen des Studiendesigns simuliert. Ausnahmen bilden die Anzahl der Beobachtungen und Variablen, welche respektive auf 100.000 und 100 festgelegt werden. Die Anzahl der verwendeten Prozessorkerne beträgt 12. Es werden die Algorithmen evaluiert, deren parallele Ausführung einen Einfluss auf die Ergebnisse nehmen könnte, da die Parallelisierung der Algorithmen nicht embarrassingly parallel ist.

In Tabelle 9.2 sind die Ergebnisse dargestellt. Als Evaluationskriterium wird der Rand Index und die Zeit in Sekunden gewählt. Dabei wird für beide Varianten (sequenziell und parallel) der Mittelwert[1] des Rand Index über alle Faktorstufen berechnet und die Differenz gebildet (sequenziell − parallel). Da die Zeit in Sekunden nicht wie der Rand Index auf einen Wertebereich genormt ist, wird hier zur besseren Interpretierbarkeit das Verhältnis sequenziell zu parallel gebildet ($\frac{\text{sequenziell}}{\text{parallel}}$).

Wie sich erkennen lässt, gibt es bezüglich der Qualität des Clusterings keine großen Unterschiede zwischen der sequenziellen und der parallelen Variante der Algorithmen. Lediglich bei den Algorithmen Bubble und General Model 2 scheint die parallele Variante sogar bessere Ergebnisse zu liefern. Der Zeitgewinn der parallelen Varianten ist teils sehr groß und rechtfertigt die Verwendung der parallelen Variante, wenn dies ins Verhältnis zur geringen Abweichung des Rand Index gesetzt wird. Lediglich bei *Multibit Trees* scheint der Overhead der parallelen Variante zu groß zu sein, um Geschwindigkeitsvorteile zu bringen. Daher wird dieser Algorithmus sequenziell verwendet.

[1] Die Ergebnisse sind auf zwei Nachkommastellen gerundet.

Tabelle 9.2.: Differenz Sequenziell Parallel

Algorithmus	Rand(seq) – Rand(par)	$\frac{\text{Zeit(seq)}}{\text{Zeit(par)}}$
Bubble	-0,17	7,86
Multibit Trees	0,09	0,46
CLOPE	0,09	6,85
Large Item	0,07	6,13
SCALE	0,01	2,99
Sorted Neighborhood	0,02	3,77
Proximus	0,11	1,62
Canopy Clustering	0	1,01
General Model 1	-0,04	0,89
General Model 2	-0,13	0,92

9.3. Evaluation der Algorithmen

In diesem Abschnitt werden die Ergebnisse der Simulation dargestellt und besprochen. Es folgt zunächst eine Evaluation der jeweiligen Algorithmen getrennt nach den einzelnen Stufen des Simulationsaufbau, sowie für alle Stufen zusammen. Dabei werden die in Abschnitt 6 vorgestellten Evaluationskriterien anhand von Boxplots verglichen. Es wird jeweils ein Faktor des Designs betrachtet und einzelne Boxplots für jede Ausprägung des Faktors berechnet (z.B. Faktor *Prävalenz* mit den Ausprägungen *Niedrig* und *Mittel*). Die Anzahl der Beobachtungen und die Anzahl der Variablen werden zusammen in einer Analyse betrachtet, da sie beide für die Dimensionalität der Daten verantwortlich sind. Zudem konnte aufgrund technischer Hürden die Variante mit 50 Millionen Beobachtungen und 1.000 Variablen nicht durchgeführt werden. Wenn nun in den anderen beiden Stufen des Faktors *Anzahl Beobachtungen* nicht zwischen 100 und 1.000 Variablen unterschieden würde, wäre die Interpretation unnötig erschwert. Nach dieser grafischen Analyse folgt eine multivariate Analyse mithilfe von Betaregressionen. Betaregressionen erlauben es ein Regressionsmodell zu schätzen, in denen die abhängige Variable Ausprägungen im Intervall zwischen 0 und 1 annimmt (Ferrari & Cribari-Neto, 2004). Es wird für jeden Algorithmus eine deskriptive Regressionsanalyse durchgeführt, wobei der F-Score die abhängige Variable darstellt und die einzelnen Faktoren des Simulationsaufbau die unabhängigen Variablen bilden. Abschließend findet eine vergleichende Betrachtung der Algorithmen statt.

9.3.1. BUBBLE

Folgend werden die Ergebnisse des Algorithmus BUBBLE analysiert.

Gesamt Werden die Ergebnisse des Algorithmus BUBBLE über alle Stufen des Simulationsaufbau zusammen betrachtet (siehe Abbildung 9.1), zeigt sich ein gemischtes Bild.

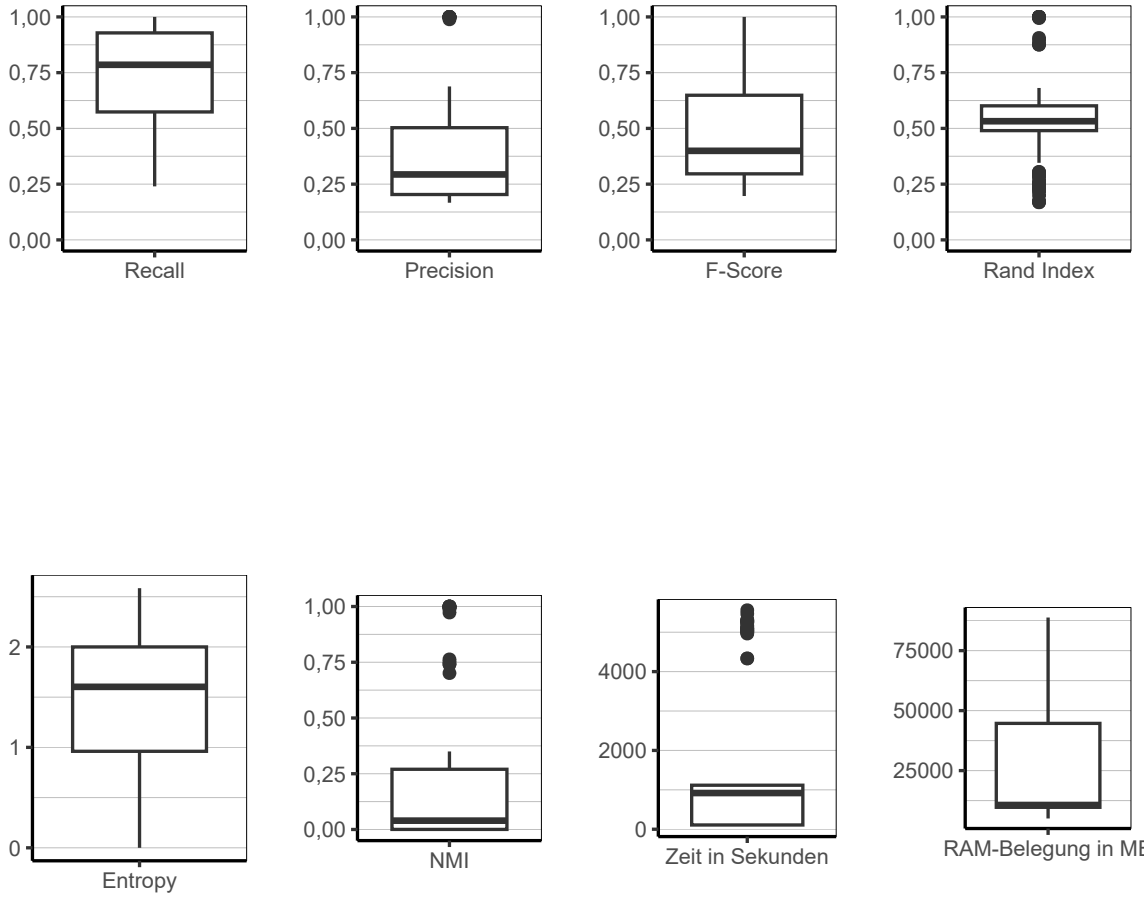


Abbildung 9.1.: BUBBLE: Evaluation aller Iterationen der Simulation insgesamt.

Die Werte für den F-Score und den Rand Index zeigen an, dass bei den meisten Daten die entsprechende Clusterlösung nur eine mittelmäßige Qualität aufweist. Auch

die Werte für Entropy und Normalized Mutual Information zeigen eine relativ hohe Unsicherheit an bzw. einen relativ niedrigen Informationsgehalt der Clusterlösung im Bezug auf die wahren Werte. Jedoch reichen die Whisker der Boxen teilweise bis zum maximal möglichen Wert. Dies bedeutet, dass es durchaus Datensätze gibt, welche sehr gut bzw. sogar perfekt geclustert werden können.

Insgesamt kann eine mittlere Streuung der Qualitätsindikatoren beobachtet werden, was auf eine zumindest mittelhohe Sensitivität gegenüber den verwendeten Faktorstufen schließen lässt. An den hohen Werten für Recall und den eher niedrigen Werten für Precision lässt sich eine Tendenz dazu erkennen, viele Beobachtungen in einen dominanten Cluster zu clustern, während die anderen nur schwach bzw. gar nicht besetzt sind. Bei Betrachtung der Gini-Koeffizienten in Abbildung 9.2 bestätigt sich diese Vermutung. Dort sind auf der Y-Achse die einzelnen Gini-Koeffizienten der Clustergrößen abgetragen und auf der X-Achse der Index der jeweiligen Iteration des Simulationsablaufs. Die Mittelwerte der Gini-Koeffizienten innerhalb der 3 Szenarien sind mit einem eingefärbten X markiert. Lediglich im Szenario *Einfach* lassen sich niedrige Gini-Koeffizienten beobachten. Der Mittelwert des Gini-Koeffizienten in diesem Szenario liegt bei 0,33 gegenüber 0,76 und 0,77 bei den Szenarien *Spezial* und *4 Block*. Dies lässt darauf schließen, dass vor allem in den beiden letztgenannten Szenarien der Algorithmus das besagte Verhalten zeigt.

Die Laufzeit des Algorithmus lässt sich insgesamt als niedrig beschreiben. Das 75 % Quantil liegt gerade mal bei 1117 Sekunden was auf einen zügigen Ablauf in den meisten Konfigurationen hindeutet. Ein Maximum von 5558,81 Sekunden bedeutet, dass auch bei sehr großen Datenmengen zeitliche Bedenken nur eine untergeordnete Rolle spielen. Selbiges lässt sich beim Speicherbedarf feststellen. Die obere Grenze der Box liegt bei gerade 44678 MB. Das bedeutet, dass selbst gut ausgestattete Desktop-PCs in der Lage sind, den Algorithmus bei mittlerer bis hoher Datenmenge auszuführen.

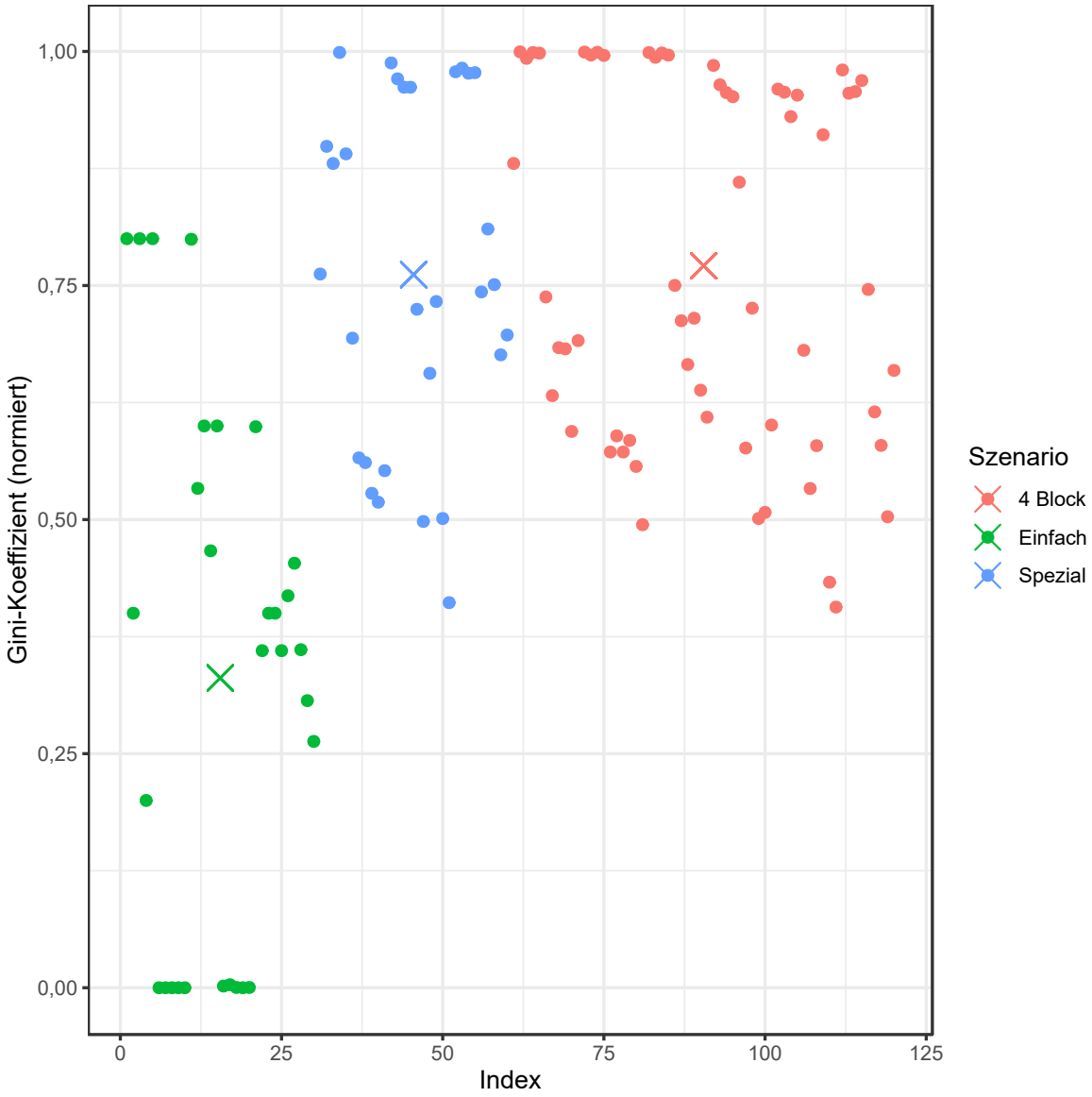


Abbildung 9.2.: BUBBLE: Gini-Koeffizient aller Iterationen der Simulation.

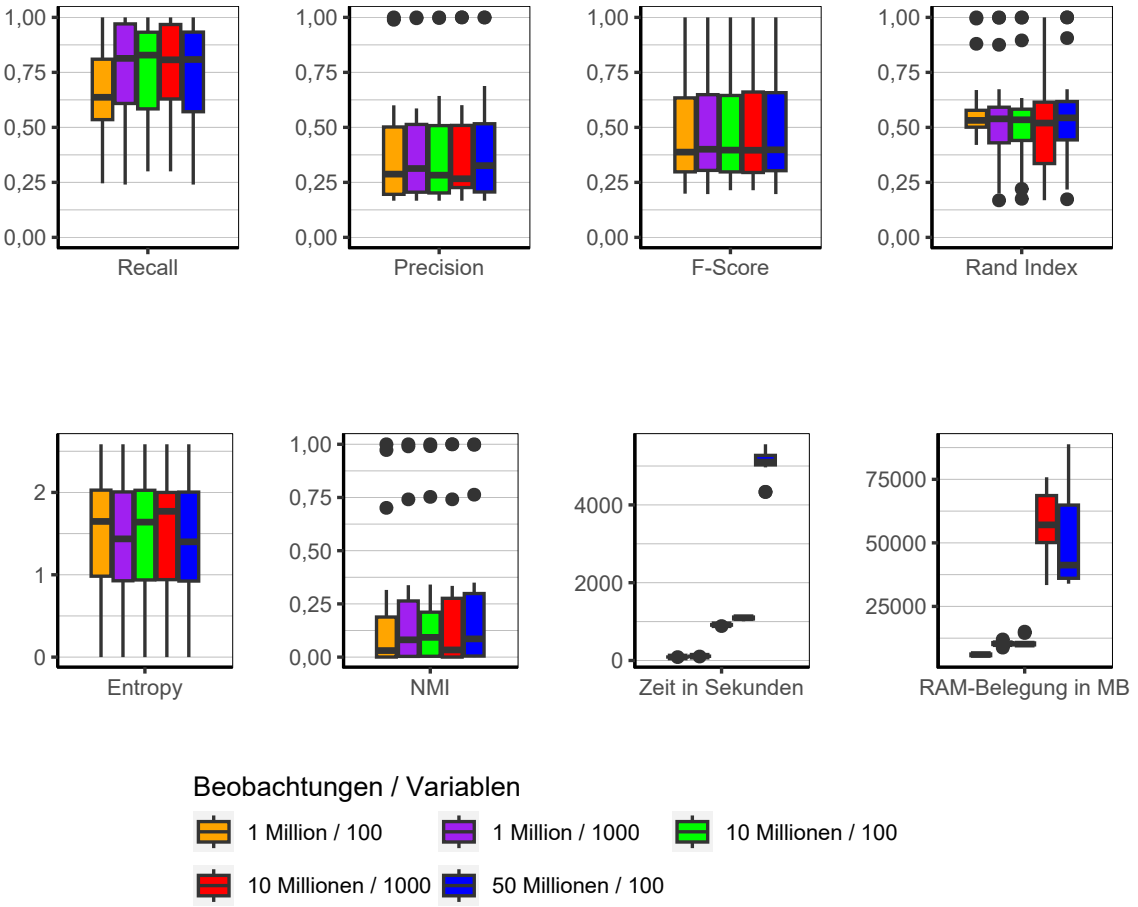


Abbildung 9.3.: BUBBLE: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Werden die Ergebnisse getrennt nach der Anzahl der Beobachtungen und Variablen betrachtet (siehe Abbildung 9.3), zeigt sich, dass bezüglich der Qualitätsindikatoren keine substanziellen Unterschiede ausgemacht werden können. Lediglich die Streuung des Rand Index bei 10 Millionen Beobachtungen und 1.000 Variablen ist etwas höher als bei den anderen Stufen. Ansonsten sind sowohl die Größen der Boxen, das Niveau und Level der Mediane größtenteils vergleichbar. Der Median der Entropie ist bei 50 Millionen Beobachtungen / 100 Variablen und bei 1 Millionen Beobachtungen / 1.000

Variablen etwas niedriger und beim NMI entsprechend etwas höher. Das weist auf eine etwas weniger schiefe Verteilung der resultierenden Clustergrößen hin. Die leicht höhere Precision unterstreicht diese Vermutung. Aufgrund der geringen Unterschiede, kann dies jedoch auch dem Zufall geschuldet sein.

Wie zu erwarten, sind große Unterschiede nur in der Laufzeit und im Speicherverbrauch zu beobachten. Eine Millionen Beobachtungen gehen über alle sonstigen Faktoren sehr schnell und benötigen wenig Speicher. 10 Millionen Beobachtungen sind immer noch schnell berechnet und selbst das Level für 50 Millionen Beobachtungen liegt zwar im Verhältnis zu den anderen Stufen wesentlich höher, ist aber in Anbetracht der großen Datenmenge absolut betrachtet immer noch als schnell zu beurteilen. Der Grund für den geringen Unterschied zwischen der Anzahl der Variablen liegt hier in der geringen Anzahl an Distanzberechnungen für jede einzelne Beobachtung, bei welcher dann die Anzahl der Variablen keinen großen Einfluss hat. Beim Speicherbedarf zeigt sich ein ähnliches Bild mit dem Unterschied, dass hier die Anzahl der Variablen einen nennenswerten Unterschied bei 10 Millionen Beobachtungen ausmacht. Der Speicherbedarf ist hier über große Teile sogar höher als bei 50 Millionen Beobachtungen / 100 Variablen.

Prävalenz Bezüglich der Prävalenz zeigt sich ein erwartbares Bild (siehe Abbildung 9.4). Durch alle Qualitätsindikatoren hinweg sind die Werte für Clusterings der Daten mit der höheren Prävalenz besser. Dies ergibt Sinn, da dadurch die Cluster deutlicher hervortreten und auch vom Algorithmus besser erkannt werden können. Einzige Ausnahme bilden die Werte für Recall, welche bei höherer Prävalenz etwas niedriger sind. Erklärung hierfür könnte sein, dass bei niedrigerer Prävalenz der Algorithmus eher dazu tendiert Cluster mit schiefer Verteilung zu generieren bzw. im Extremfall alle Beobachtungen in einen Cluster zu allokieren. Dies führt zwangsläufig zu einem hohen Wert für Recall. Sowohl die Laufzeit als auch der Speicherverbrauch sind in beiden Varianten nahezu identisch.

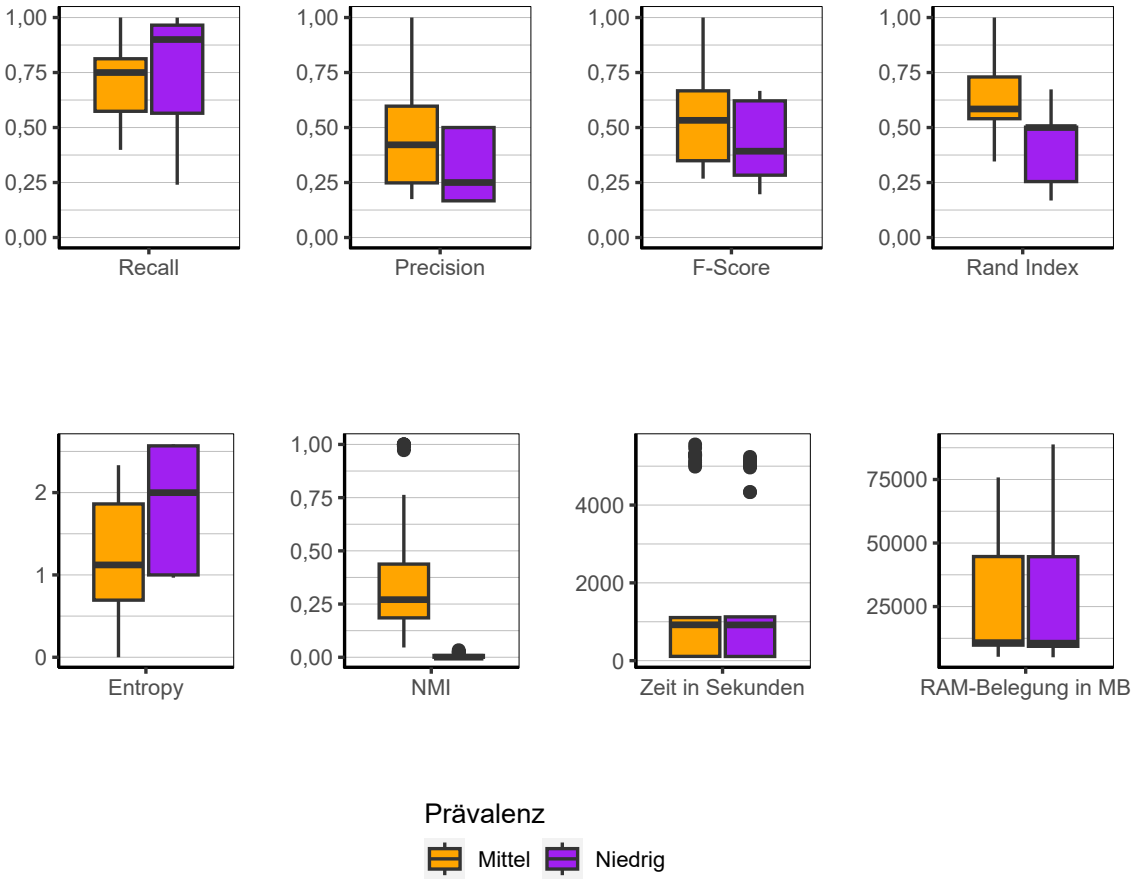


Abbildung 9.4.: BUBBLE: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

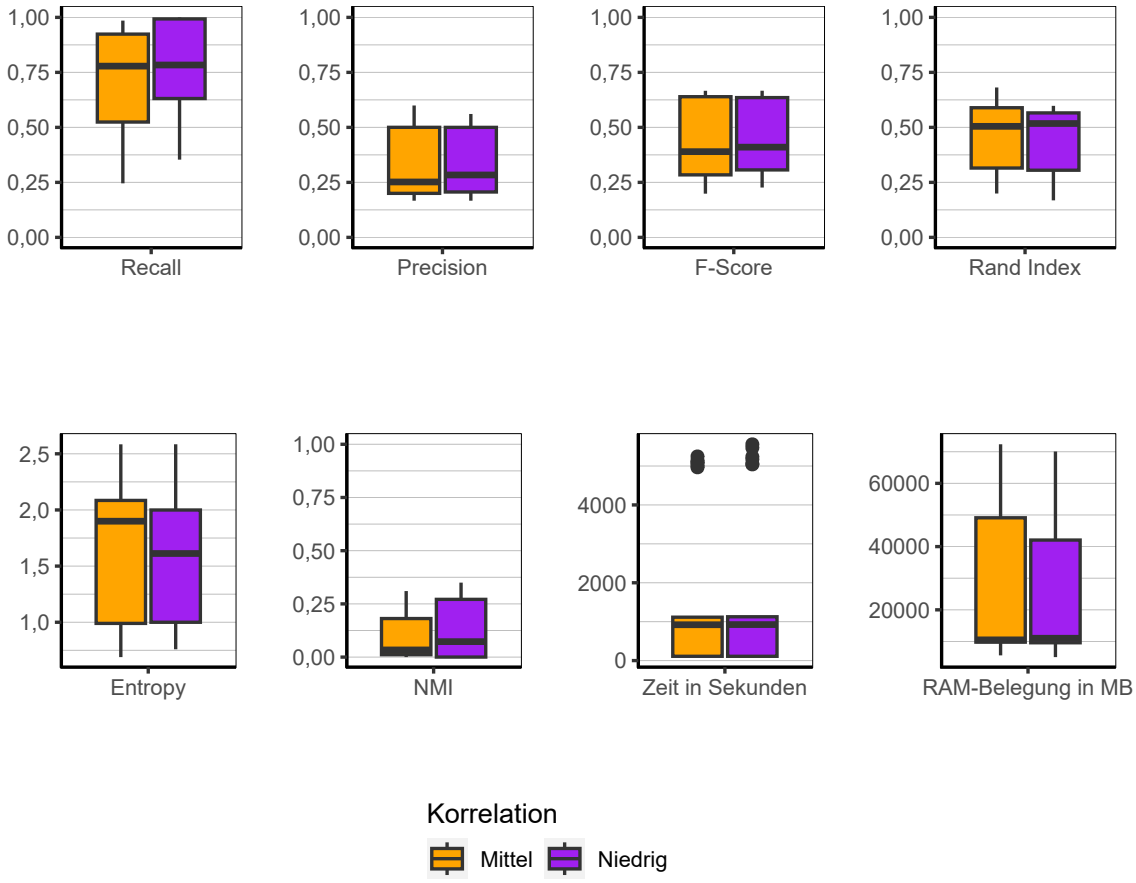


Abbildung 9.5.: BUBBLE: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Die Höhe der Korrelation innerhalb der Daten scheint nahezu keinen Unterschied zu machen (siehe Abbildung 9.5). Sowohl die Qualitätsindikatoren des Clusterings als auch Laufzeit und Speicherbedarf lassen keinen gewichtigen Unterschied zwischen den beiden Stufen des Faktors Korrelation erkennen.

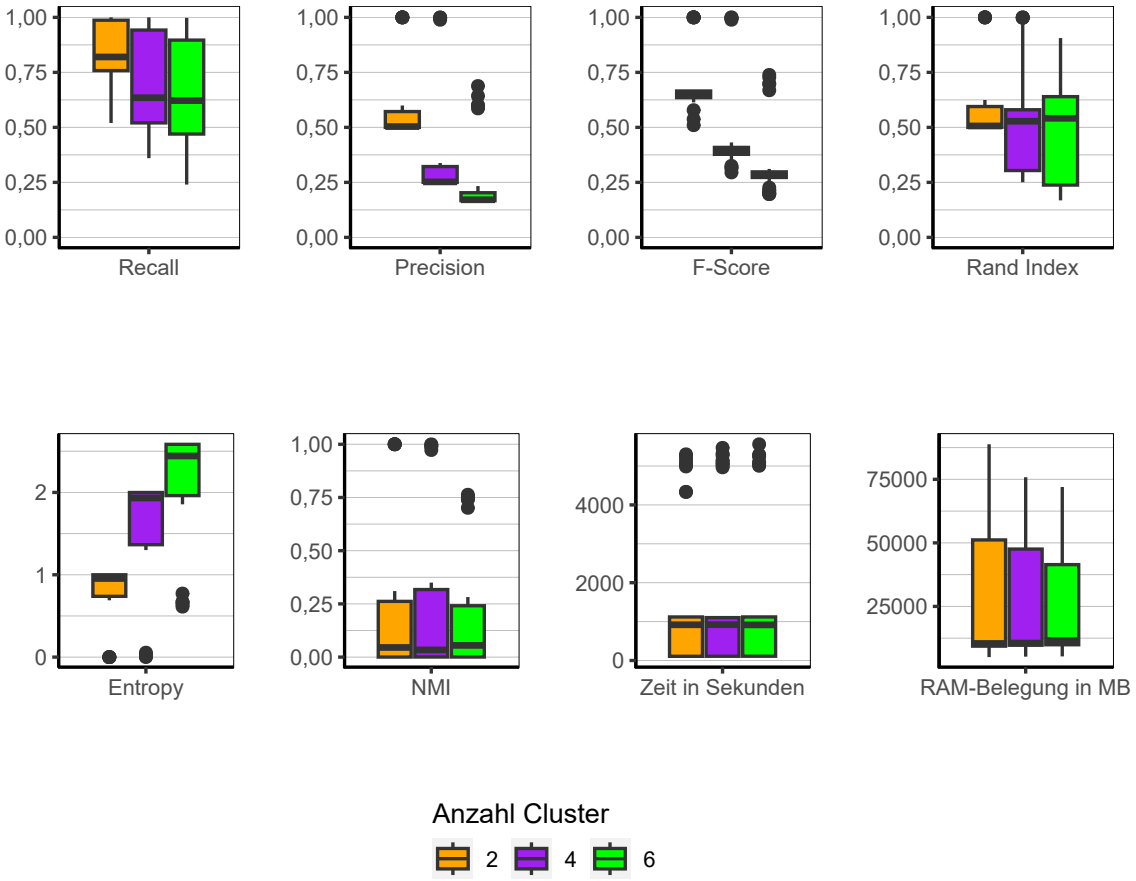


Abbildung 9.6.: BUBBLE: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Die Anzahl der Cluster wirkt sich merklich auf die Qualität des resultierenden Clusterings aus (siehe Abbildung 9.6). Mit Ausnahme des NMI deuten alle Qualitätsindikatoren darauf hin, dass mit steigender Anzahl an Clustern die Güte der Clusterlösung abnimmt. Auffallend ist in diesem Kontext auch, dass sowohl bei 2 als auch bei 4 Clustern perfekte Clusterings beobachtet werden können. Bei 6 Clustern gibt es zwar auch einige Ausreißer deren Qualität noch als gut betrachtet werden kann, aber doch deutlich hinter den Werten für 2 und 4 Clustern zurück liegt. Insgesamt sind die

Clusterlösungen für Daten mit lediglich 2 Clustern am besten und deutlich besser als bei 4 oder 6 Clustern.

Die Laufzeit ist zwischen allen Stufen nahezu identisch. Der Speicherbedarf lässt ebenfalls auf keine großen Unterschiede schließen, obgleich doch ein etwas abfallender Verbrauch mit Anzahl der Cluster beobachtet werden kann.

Szenario In Abbildung 9.7 kann eindeutig erkannt werden, dass BUBBLE beim Szenario *Einfach* weitaus besser in der Lage ist hochwertige Clusterings zu produzieren, als bei den anderen beiden Varianten. Bei jedem Qualitätsindikator reicht die Box bis zum perfekten Wert, während das gegenüberliegende Ende der Box meist etwa auf dem Niveau des Medians der anderen beiden Szenarien liegt, welche nicht perfekt geclustert werden konnten. Es gibt demnach auch Datensätze innerhalb der anderen beiden Szenarien, welche besser oder ähnlich gut geclustert werden können, wie im Szenario *Einfach*. Insgesamt sind die Ergebnisse im Szenario *Einfach* jedoch weitaus besser. Von allen Faktoren hat der Faktor *Szenario* demnach den mit Abstand größten Einfluss. Das ist nicht verwunderlich, muss doch wie in Abschnitt 5.6 beschrieben, bedacht werden, dass die Daten in den Szenarien *4 Block* und *Spezial* mehr Rauschen enthalten und die Clusterstrukturen weniger eindeutig sind als im Szenario *Einfach*. Die Ergebnisse zu Laufzeit und Speicherverbrauch sind nahezu identisch.

Fazit Werden die Ergebnisse insgesamt betrachtet, zeigt sich BUBBLE als ein schneller und effizienter Algorithmus, welcher auch für sehr große Datensätze geeignet ist. Die Dimensionalität der Daten hat keinen großen Einfluss auf die Qualität der Clusterlösungen. Das gleiche kann für die Höhe der Korrelation gesagt werden. Wenige Cluster können besser erkannt werden als viele und eine höhere Prävalenz in den Daten führt tendenziell zu besseren Ergebnissen. Die Analyse der unterschiedlichen Szenarien offenbaren, dass der Algorithmus gut für eindeutig getrennte Cluster verwendet werden kann, während er bei Daten mit mehr Rauschen und somit weniger deutlich getrennter Clusterstruktur auch entsprechend schlechtere Ergebnisse hervorbringt. Dies endet in einer Tendenz eine schiefe Verteilung von Clustergrößen zu generieren, in der sich viele Beobachtungen in einem Cluster befinden und nur wenige bzw. keine in den anderen Clustern.

9.3.2. CLARA

Im folgenden Abschnitt werden die Ergebnisse des Algorithmus CLARA analysiert.[2]

Gesamt Werden die Ergebnisse über alle Faktorstufen zusammen betrachtet (siehe Abbildung 9.8), ergibt sich ein sehr ähnliches Bild wie schon zuvor beim Algorithmus BUBBLE, weshalb die beiden Algorithmen an dieser Stelle direkt verglichen werden sollen. Die Werte für Recall sind hoch, während die Werte für Precision niedrig sind, was zu mittleren Werten für den F-Score führt. Dies deutet ebenfalls wieder auf die Tendenz hin, viele Beobachtungen einem einzigen Cluster zuzuweisen und nur wenige den restlichen Clustern. Dies zeigt sich auch wiederum in der Abbildung der Gini-Koeffizienten (siehe Abbildung 9.9). Die Mittelwerte der Gini-Koeffizienten sind entsprechend 0,44, 0,71 und 0,71 für die Szenarien *Einfach*, *Spezial* und *4 Block*.

Die Streuung des Rand Index ist etwas geringer und Entropy und NMI weisen auf einen etwas höheren Informationsgehalt hin. Der Unterschied ist jedoch marginal. Es gibt ein einige Iterationen innerhalb der Simulation, welche ein sehr gutes bzw. perfektes Ergebnis hervorbringen, was sich daran erkennen lässt, dass die Whisker der Boxen jeweils bis zum optimalen Wert des Evaluationskriteriums reichen (Ausnahme bei NMI). Bei BUBBLE sind dies beim Rand Index und Precision lediglich Ausreißer. CLARA scheint also insgesamt betrachtet marginal bessere Ergebnisse zu liefern.

Die gute zeitliche Performance von BUBBLE kann nochmal übertroffen und als sehr schnell betrachtet werden. Das 75 % Quantil liegt bei gerade mal 273,77 Sekunden mit einem Maximum von 511,05 Sekunden. Der Speicherbedarf ist vergleichbar mit dem des Algorithmus BUBBLE, wobei das obere Ende der Box bei 44884 MB liegt.

[2] Aufgrund eines Segfault Errors bei der Konfiguration mit 50 Millionen Beobachtungen, 100 Variablen, niedriger Prävalenz, Szenario *Spezial* und zwei Clustern konnten die Ergebnisse für diese Konfiguration nicht berichtet werden. Ein Segfault Error tritt auf, wenn das Programm auf einen Speicherbereich zugreift, der für das Programm nicht zur Verwendung freigegeben ist.

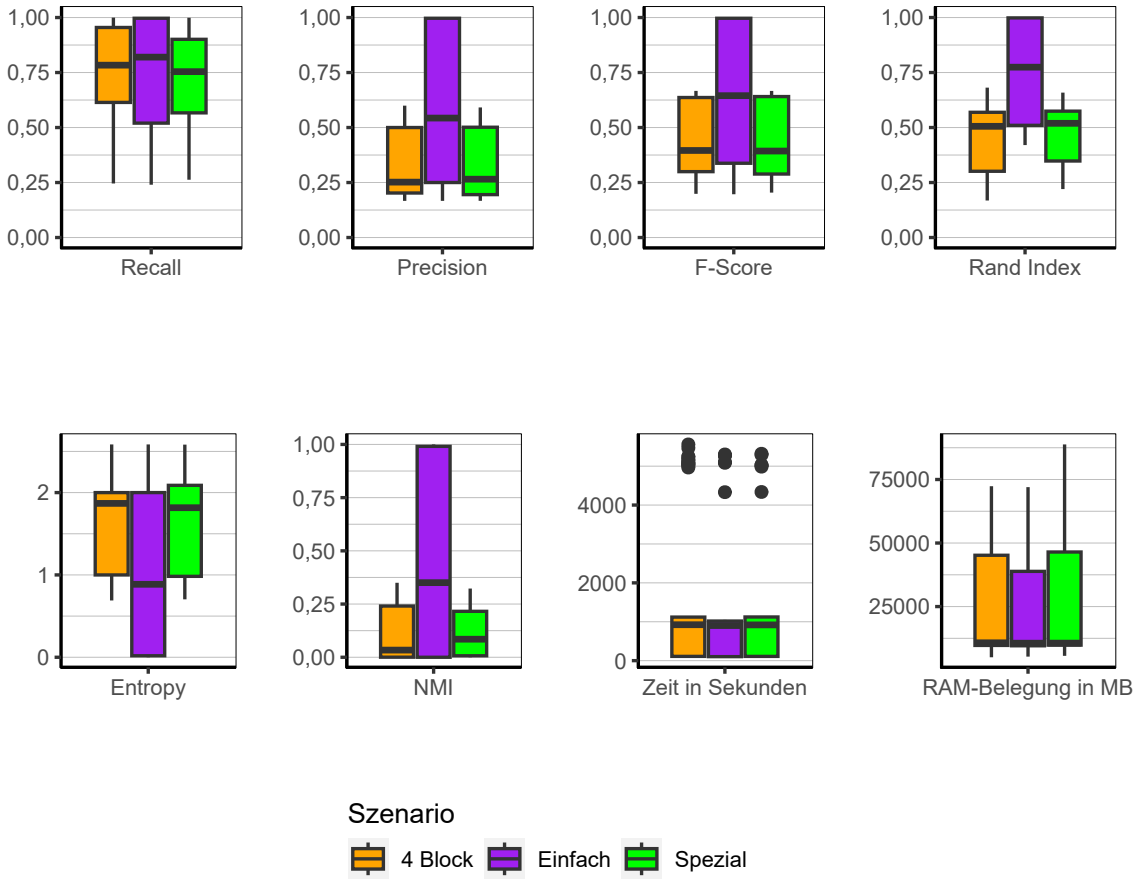


Abbildung 9.7.: BUBBLE: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

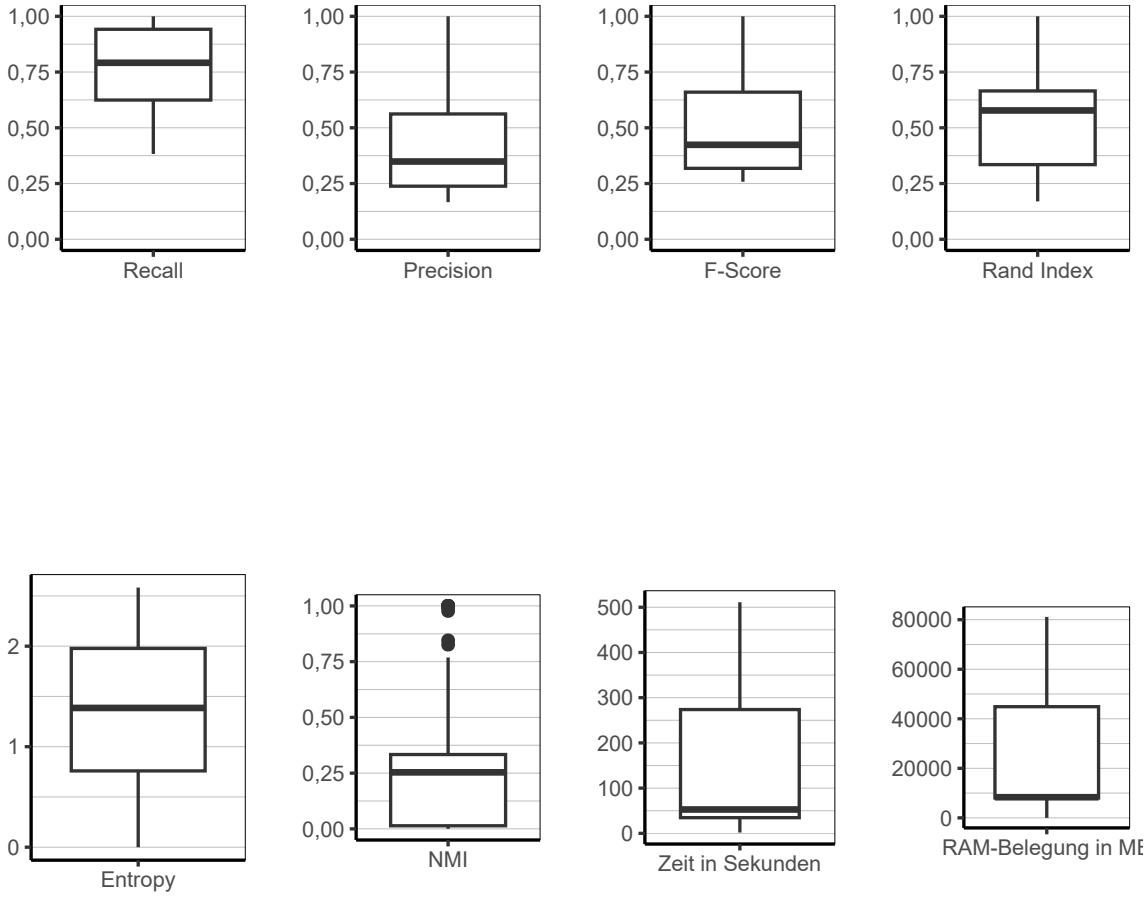


Abbildung 9.8.: CLARA: Evaluation aller Iterationen der Simulation insgesamt.

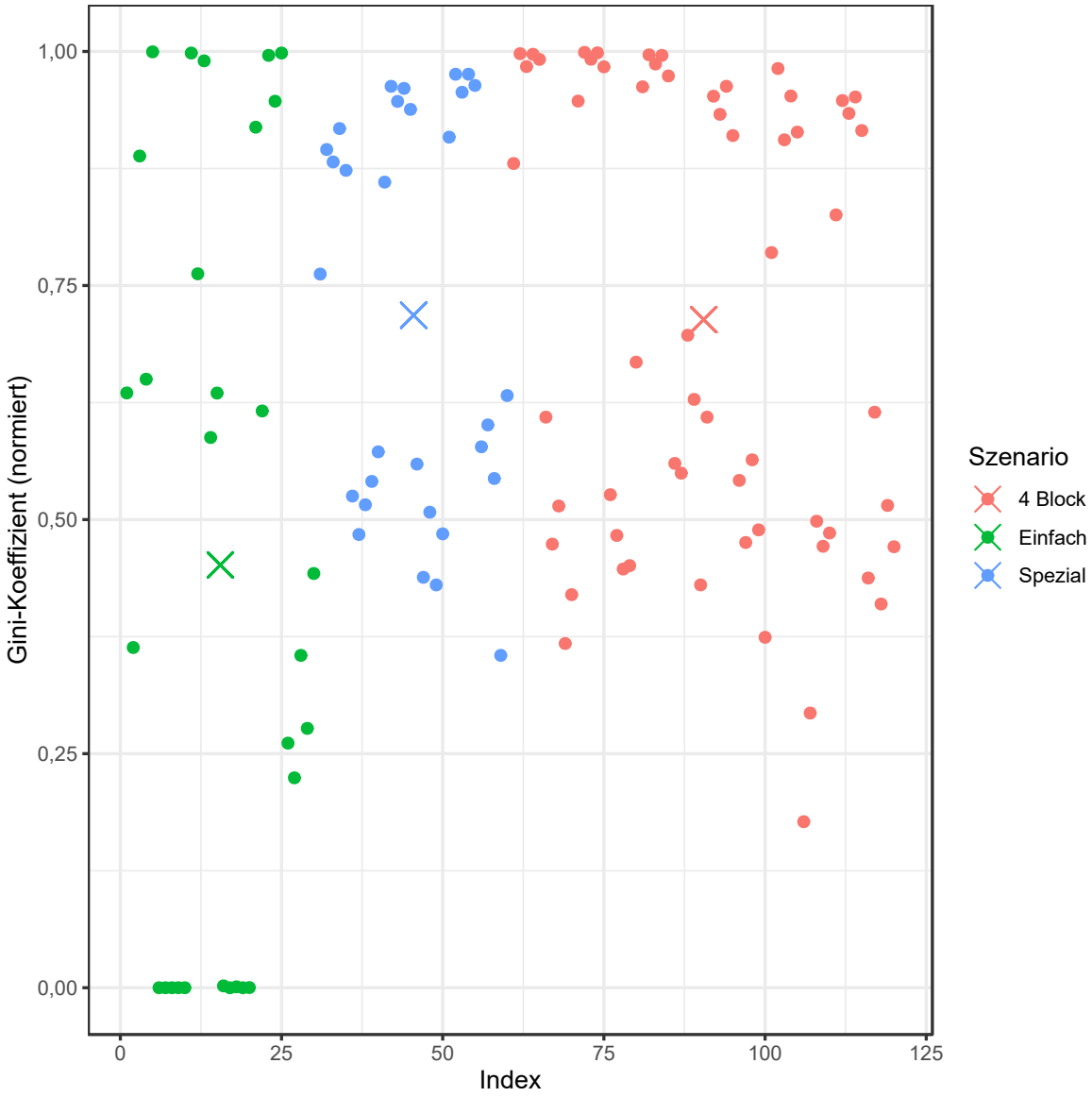


Abbildung 9.9.: CLARA: Gini-Koeffizient aller Iterationen der Simulation.

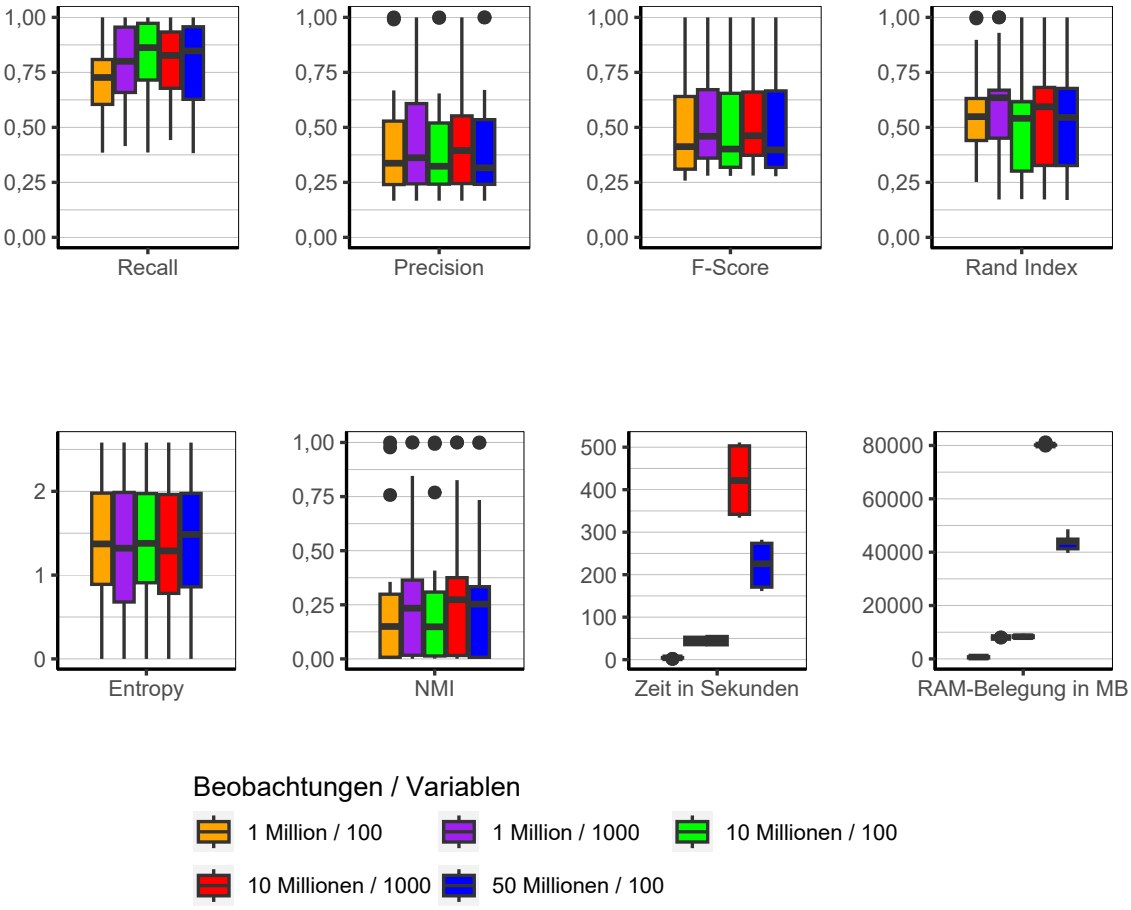


Abbildung 9.10.: CLARA: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Bezüglich der Dimensionalität (siehe Abbildung 9.10) lassen sich wiederum bei den Qualitätsindikatoren kaum substanzielle Unterschiede erkennen. Die Ergebnisse sind analog zu den Ergebnissen beim Algorithmus BUBBLE.

Einzig bei den beiden Performanceindikatoren lässt sich ein anderes Pattern verzeichnen. Wo die Laufzeit von BUBBLE noch klar von der Anzahl der Beobachtungen beeinflusst ist, scheint die Laufzeit von CLARA auch maßgeblich von der Anzahl der Variablen

abhängig zu sein. So benötigen 10 Millionen Beobachtungen mit 1.000 Variablen mehr Zeit als 50 Millionen Beobachtungen mit 100 Variablen. Jedoch trennen die Mediane der beiden angesprochenen Setups lediglich 195,85 Sekunden. Dies ist vor dem Hintergrund der großen Datenmenge kein substanzieller Unterschied. Der Speicherbedarf verhält sich analog.

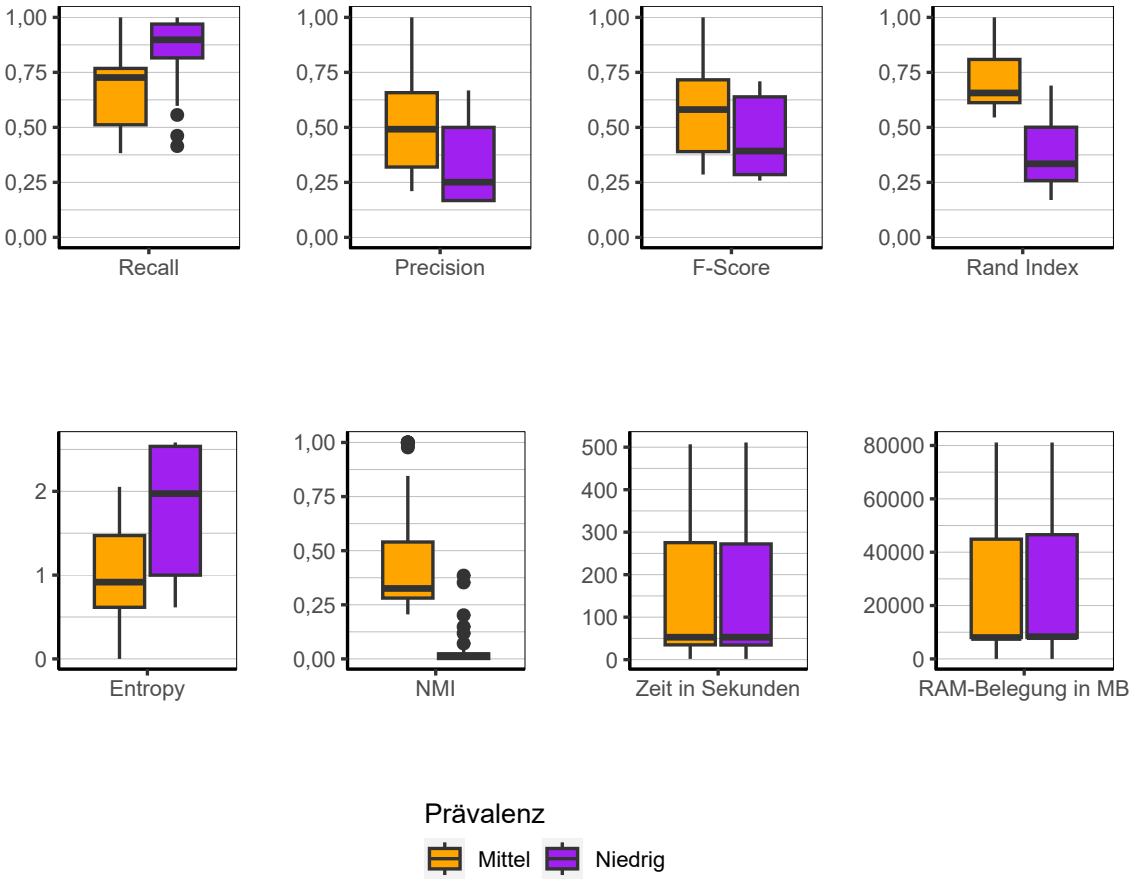


Abbildung 9.11.: CLARA: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Wie zu erwarten sind die Ergebnisse auch bezüglich der Prävalenz (siehe Abbildung 9.11) sehr ähnlich zu denen des Algorithmus BUBBLE. Die Ergebnisse sind besser bei höherer Prävalenz. Aufgrund der Tendenz bei niedriger Prävalenz wenige bzw. einen großen Cluster zu bilden, ist der Wert für Recall bei der niedrigen Prävalenz höher. Die Laufzeit und der Speicherverbrauch unterscheidet sich nicht merklich zwischen den beiden Stufen.

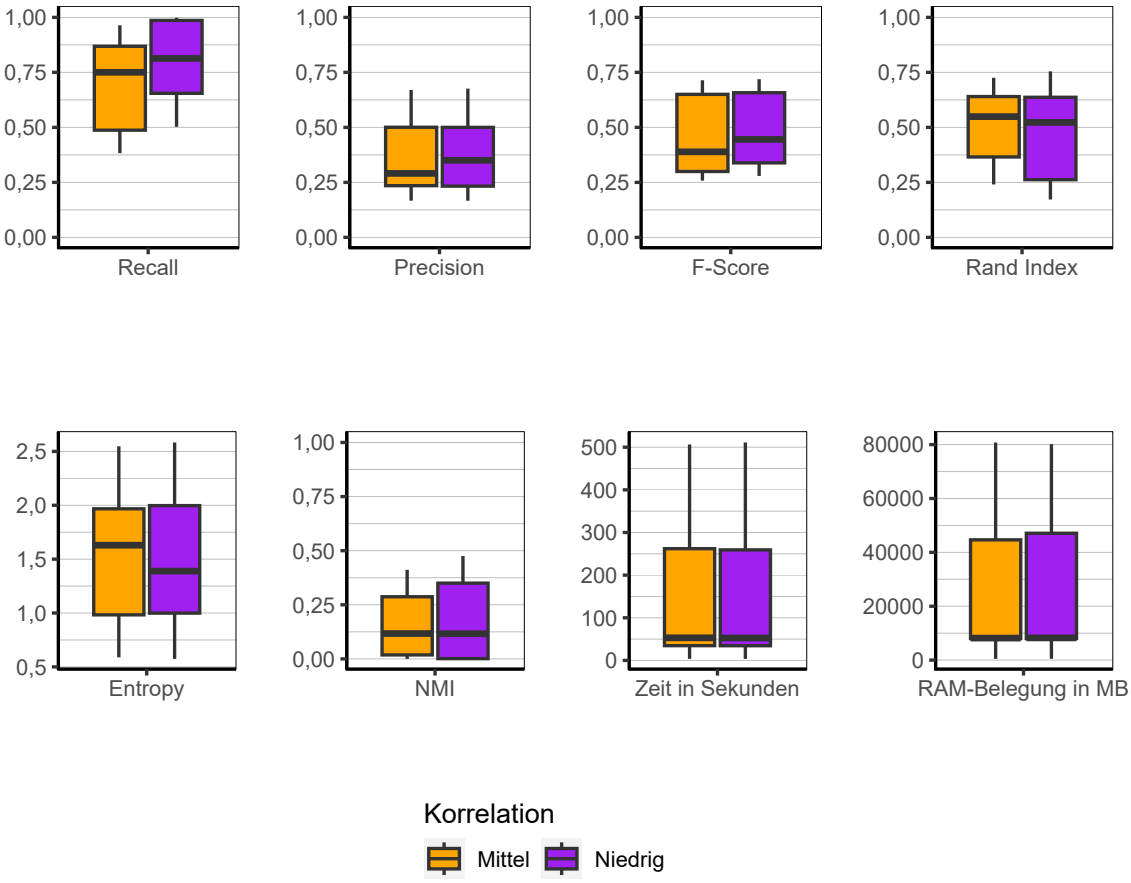


Abbildung 9.12.: CLARA: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Bei Betrachtung der unterschiedlichen Ausprägung der Korrelation (siehe Abbildung 9.12) gibt es nahezu keine Unterschiede. Sowohl bei den Qualitätsindikatoren als auch bei den Performanceindikatoren verhält sich CLARA in beiden Konfigurationen identisch.

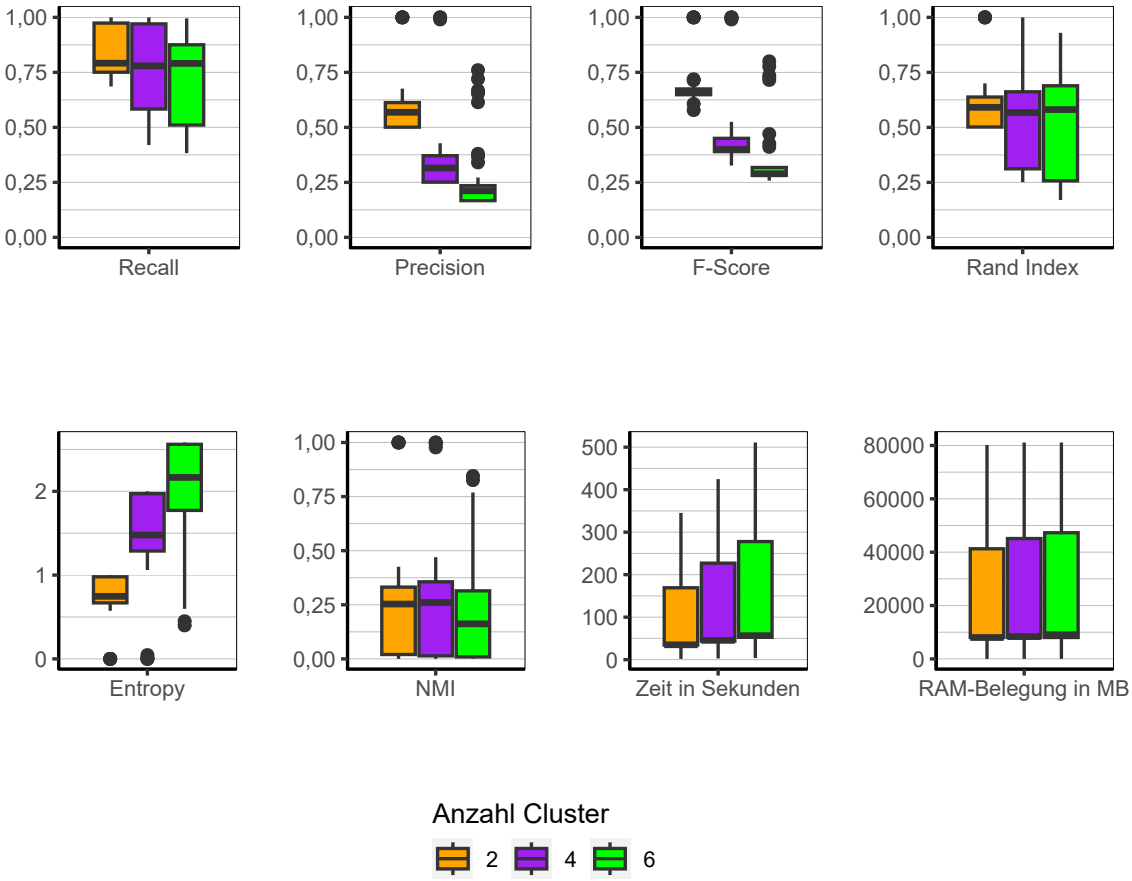


Abbildung 9.13.: CLARA: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Bei der Anzahl der Cluster können die gleichen Schlussfolgerungen (siehe Abbildung 9.13) gezogen werden, wie beim Algorithmus BUBBLE. Einziger Unterschied kann bei den Performanceindikatoren festgestellt werden. Hier steigt sowohl die Laufzeit als auch der Speicherbedarf mit der Anzahl der Cluster an. Dieser Anstieg ist allerdings sehr gering und auch bei 6 Clustern befinden sich beide Indikatoren auf einem geringen Level.

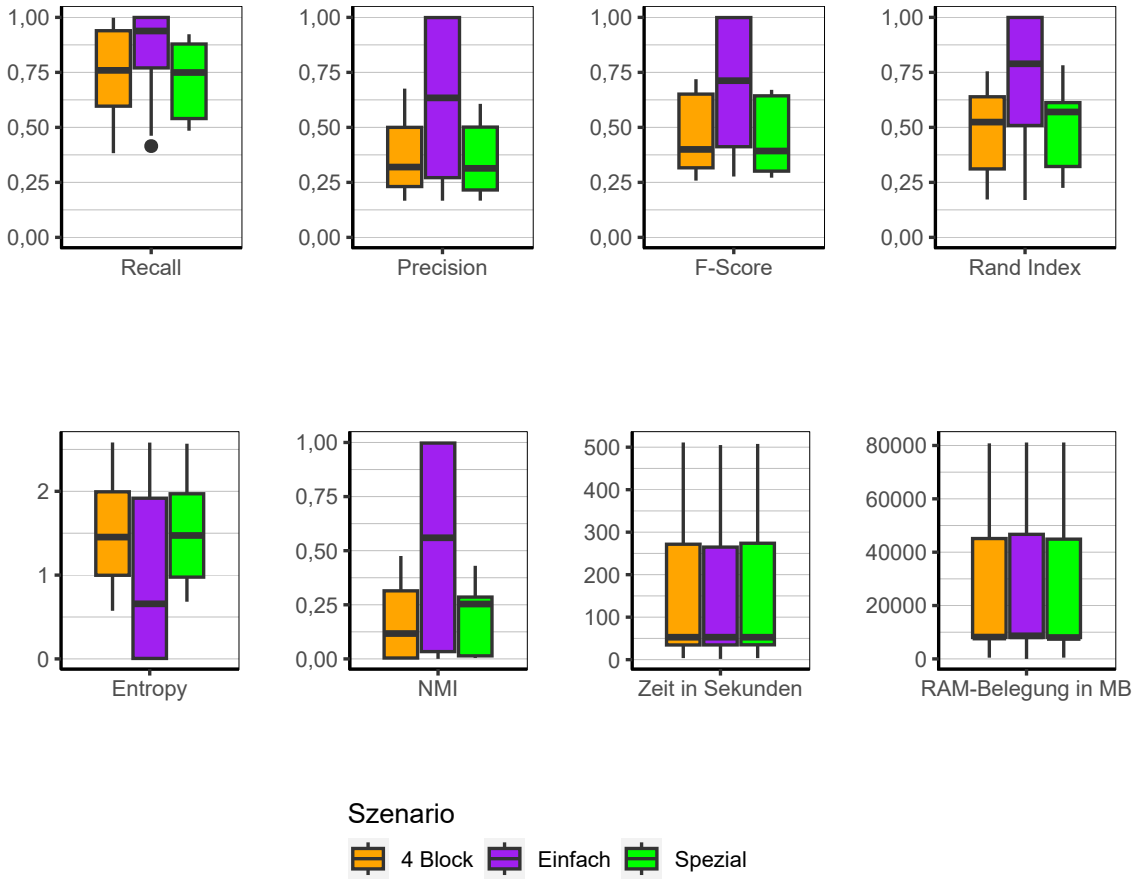


Abbildung 9.14.: CLARA: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Wie schon bei BUBBLE zeigt sich auch bei CLARA, dass der Algorithmus wesentlich bessere Ergebnisse liefert, wenn die Cluster eindeutig getrennt sind und kein Rauschen im Datensatz vorhanden ist. Die Ergebnisse beim Szenario *Einfach* sind weitaus besser als die der anderen beiden und die Box erstreckt sich bis zum optimalen Wert, während die anderen beiden weit dahinter zurück bleiben. Bei den Performanceindikatoren finden sich keine nennenswerten Unterschiede. Auch hier sind die Ergebnisse sehr ähnlich zu den Ergebnissen des Algorithmus BUBBLE.

Fazit Verglichen mit BUBBLE lassen sich nahezu identische Ergebnisse betrachten, was bedeutet, dass alle Schlussfolgerungen aus dem Fazit zu BUBBLE (siehe Abschnitt 9.3.1) auch bei CLARA zutreffen. Das bedeutet, dass die Dimensionalität der Daten und die Korrelationsstruktur keinen großen Einfluss ausübt. Wenige Cluster werden besser erkannt als viele und eindeutig getrennte Cluster werden wesentlich besser erkannt als wenn eine größere Menge an Rauschen in den Daten vorhanden ist. Auch hier finden sich wieder schiefe Verteilungen in den Clustergrößen. Das bedeutet, dass sich oftmals viele Beobachtungen in wenigen Clustern befinden. Einziger Unterschied zu BUBBLE ist, dass CLARA nochmal schneller ist und daher bevorzugt werden sollte.

9.3.3. CLARANS

In diesem Abschnitt werden die Ergebnisse des Algorithmus CLARANS dargestellt. Die Ergebnisse sind wiederum nahezu identisch mit den bereits analysierten von BUBBLE und CLARA und unterscheiden sich lediglich geringfügig. Einzig nennenswerter Unterschied liegt bei der Betrachtung der drei Szenarien.

Gesamt Die Ergebnisse in Abbildung 9.16 sind analog zu denen von BUBBLE und CLARA. Einzig die Laufzeit ist höher und der Speicherbedarf fällt um ein Vielfaches größer aus. Tatsächlich ist der Speicherbedarf von CLARA der höchste von allen betrachteten Algorithmen.

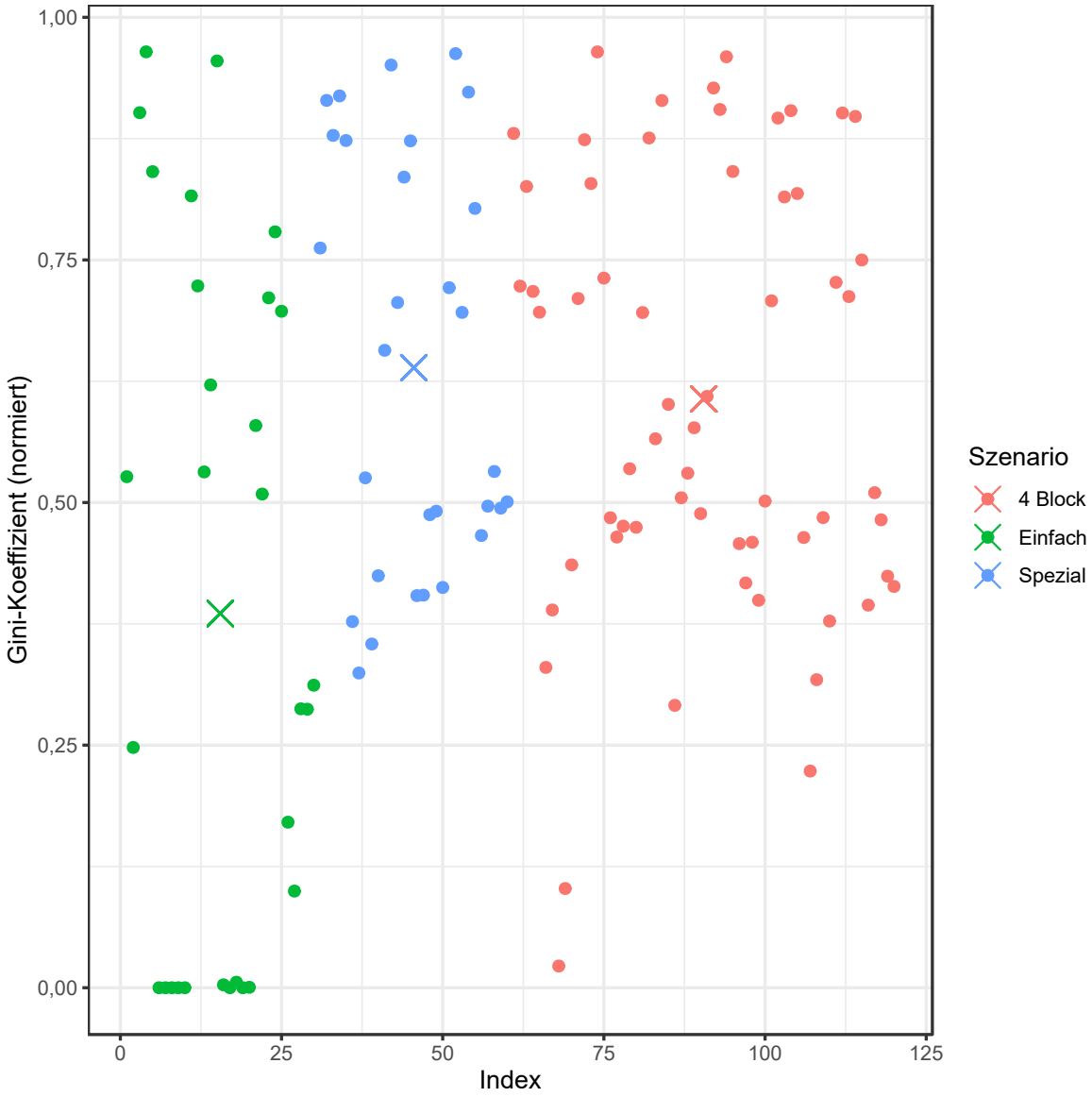


Abbildung 9.15.: CLARANS: Gini-Koeffizient aller Iterationen der Simulation.

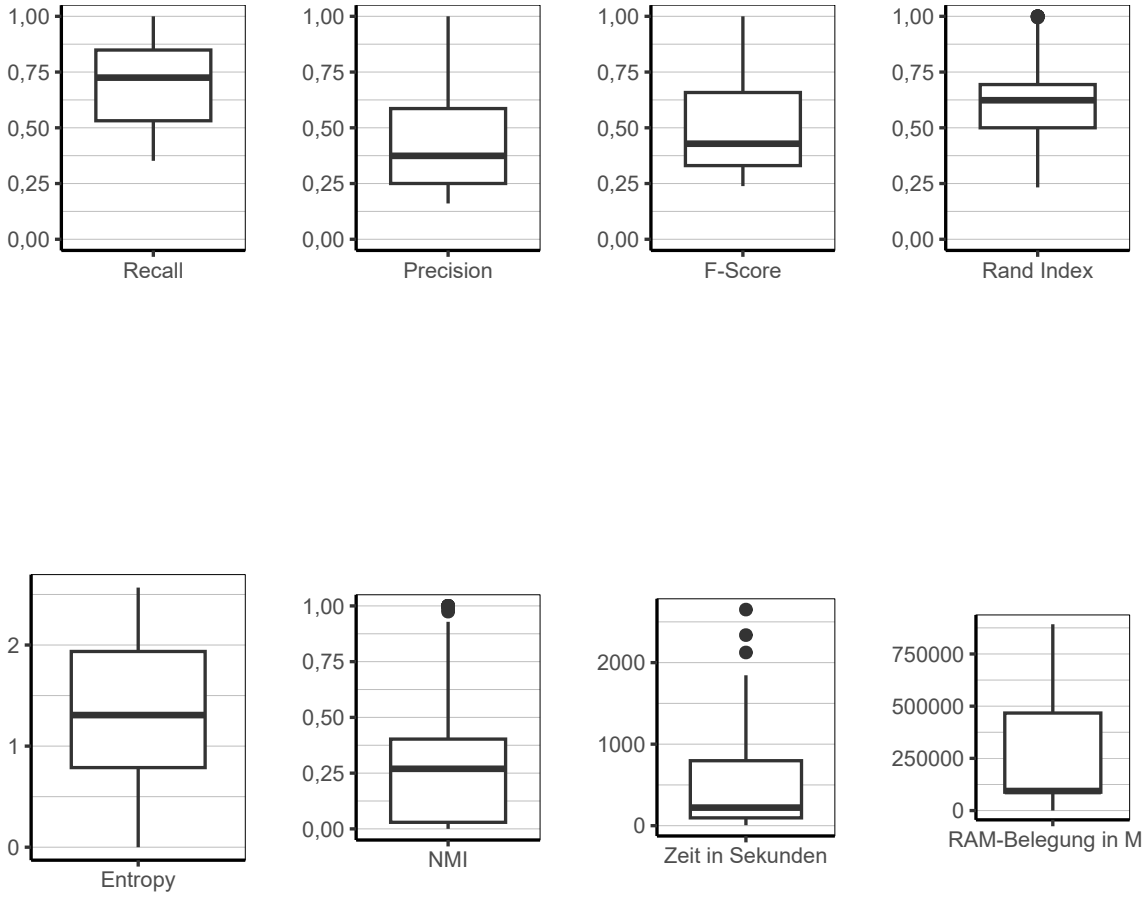


Abbildung 9.16.: CLARANS: Evaluation aller Iterationen der Simulation insgesamt.

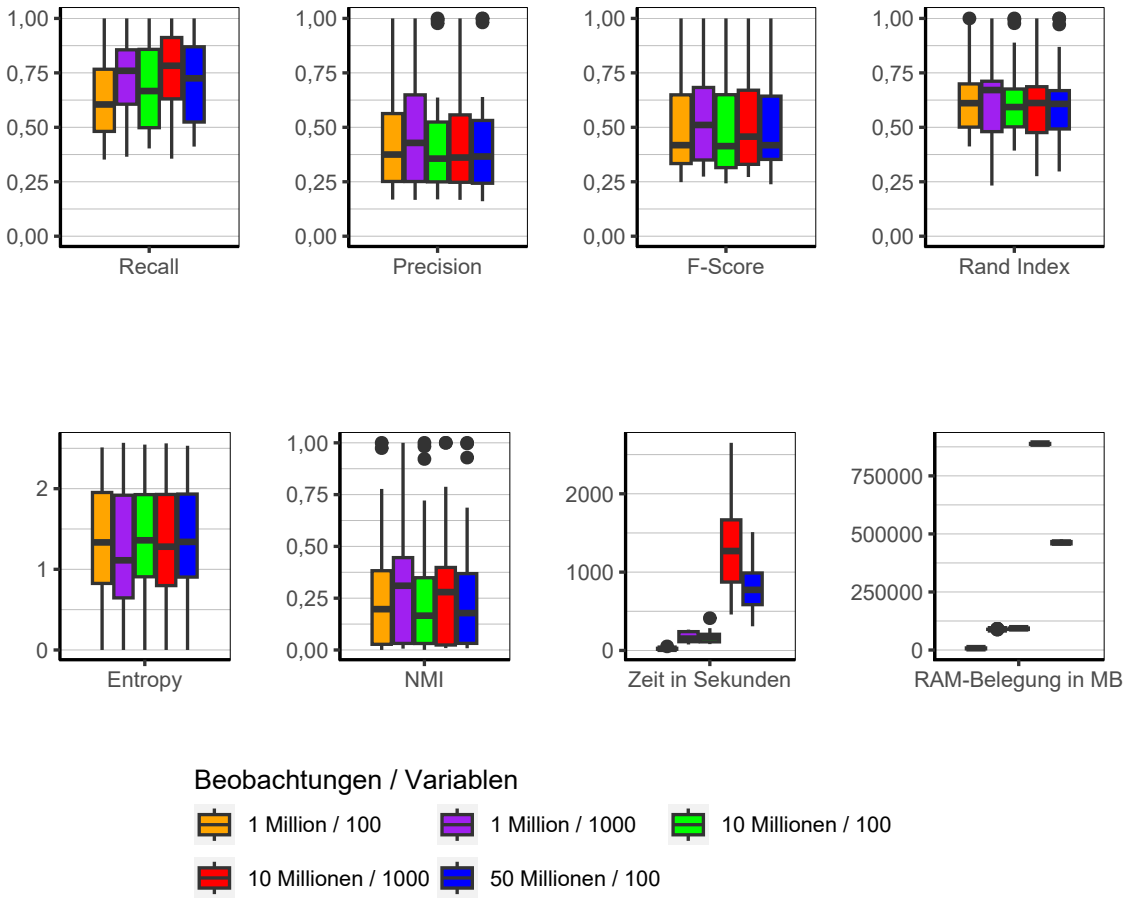


Abbildung 9.17.: CLARANS: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Die Ergebnisse in Abbildung 9.17 zeigen ein ähnliches Bild wie bei BUBBLE und CLARA. Sowohl die Anzahl der Beobachtungen als auch die Anzahl der Variablen bringt keine großen Unterschiede in der Güte der resultierenden Clusterlösungen mit sich. Lediglich bei der Laufzeit und dem Speicherbedarf zeigt sich, dass 10 Millionen Beobachtungen und 1.000 Variablen wesentlich mehr Zeit und Speicher benötigen als die anderen Konfigurationen.

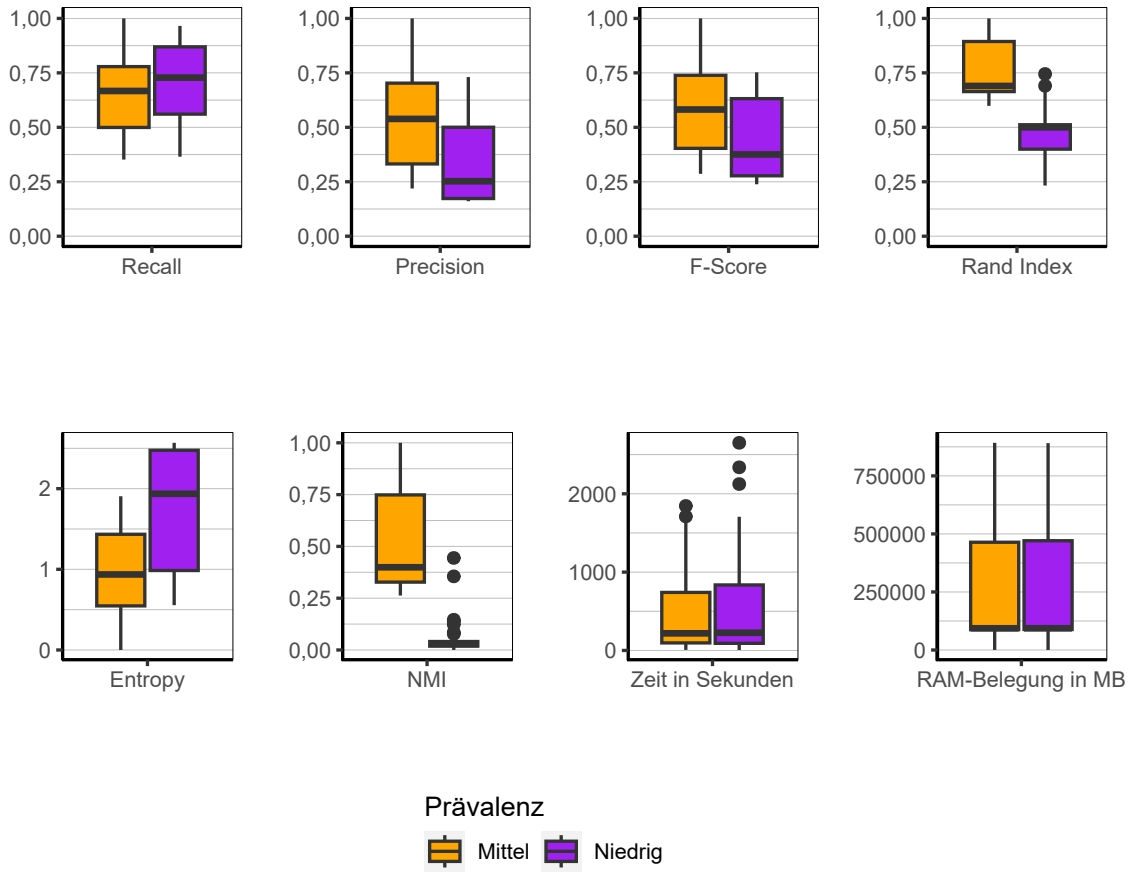


Abbildung 9.18.: CLARANS: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Abbildung 9.18 ist zu entnehmen, dass bei höherer Prävalenz der Algorithmus bessere Ergebnisse erzielt. Bezüglich Laufzeit und Speicherbedarf ist jedoch kein Unterschied festzustellen.

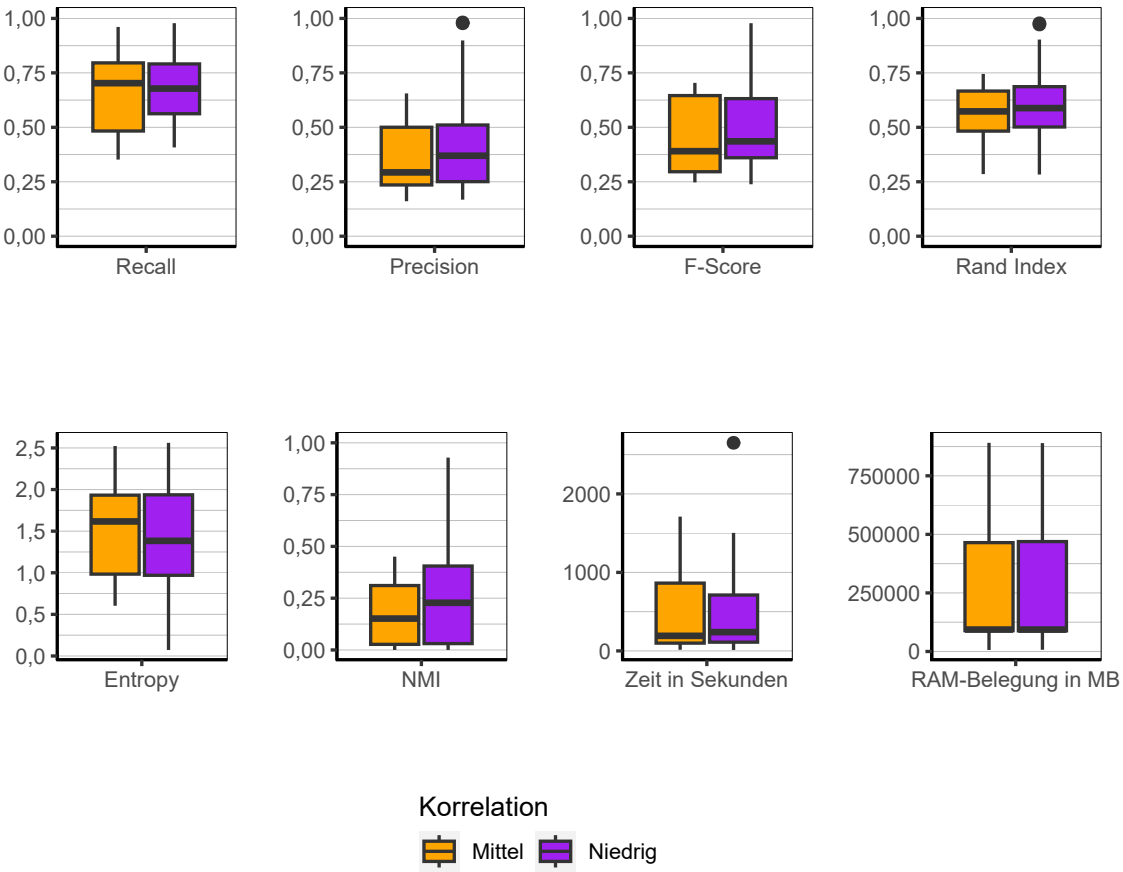


Abbildung 9.19.: CLARANS: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Bezüglich der Korrelationsstruktur (Abbildung 9.19) ist kein nennenswerter Unterschied zwischen den Konfigurationen zu erkennen. Der Algorithmus scheint daher nicht sensitiv gegenüber der Korrelationsstruktur zu sein.

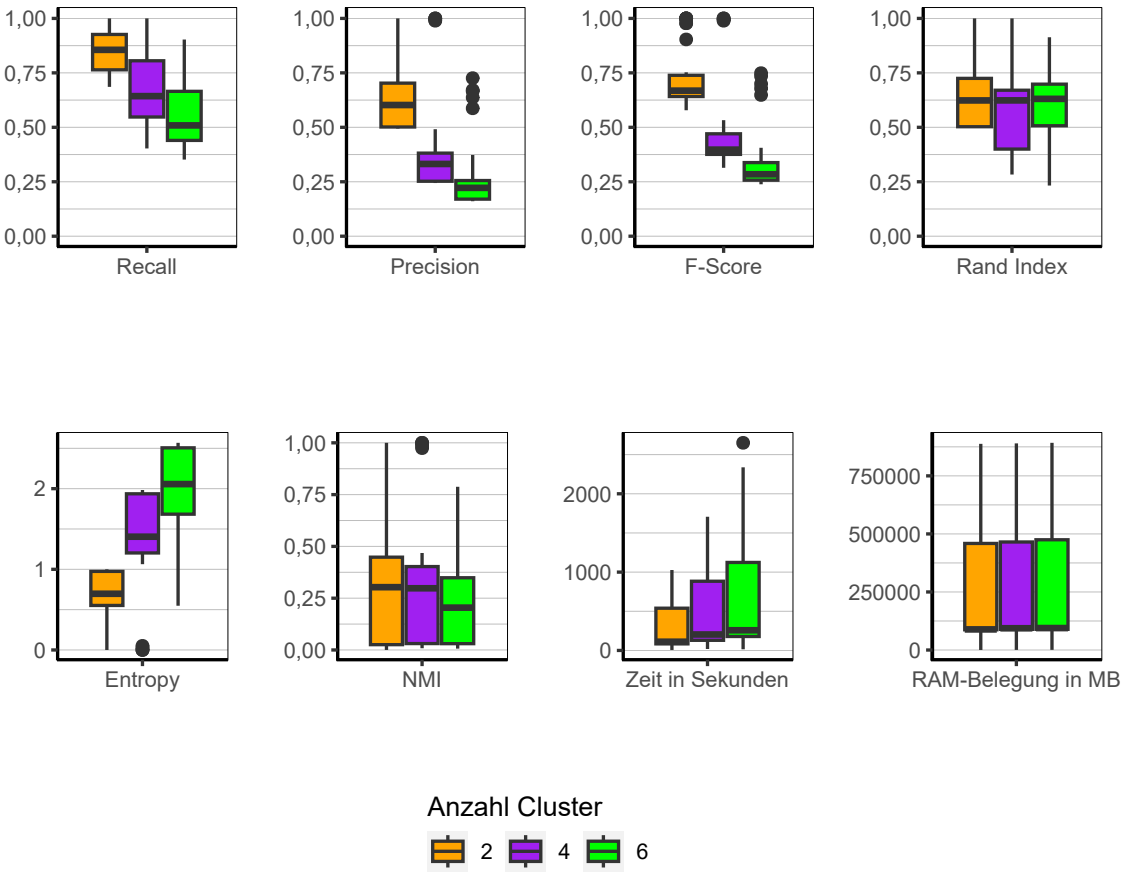


Abbildung 9.20.: CLARANS: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster In Abbildung 9.20 lässt sich mit steigender Anzahl von Clustern ein Sinken von Recall, Precision und F-Score beobachten. Dabei muss bedacht werden, dass bei Clusterlösungen mit schiefer Verteilung der Clustergrößen, der Wert für Recall, Precision und F-Score sinkt, je mehr Cluster vorhanden sind. Beim Rand Index jedoch zeigt sich, dass der tatsächliche Unterschied in den Clusterlösungen bei unterschiedlicher Anzahl von Clustern marginal ist.

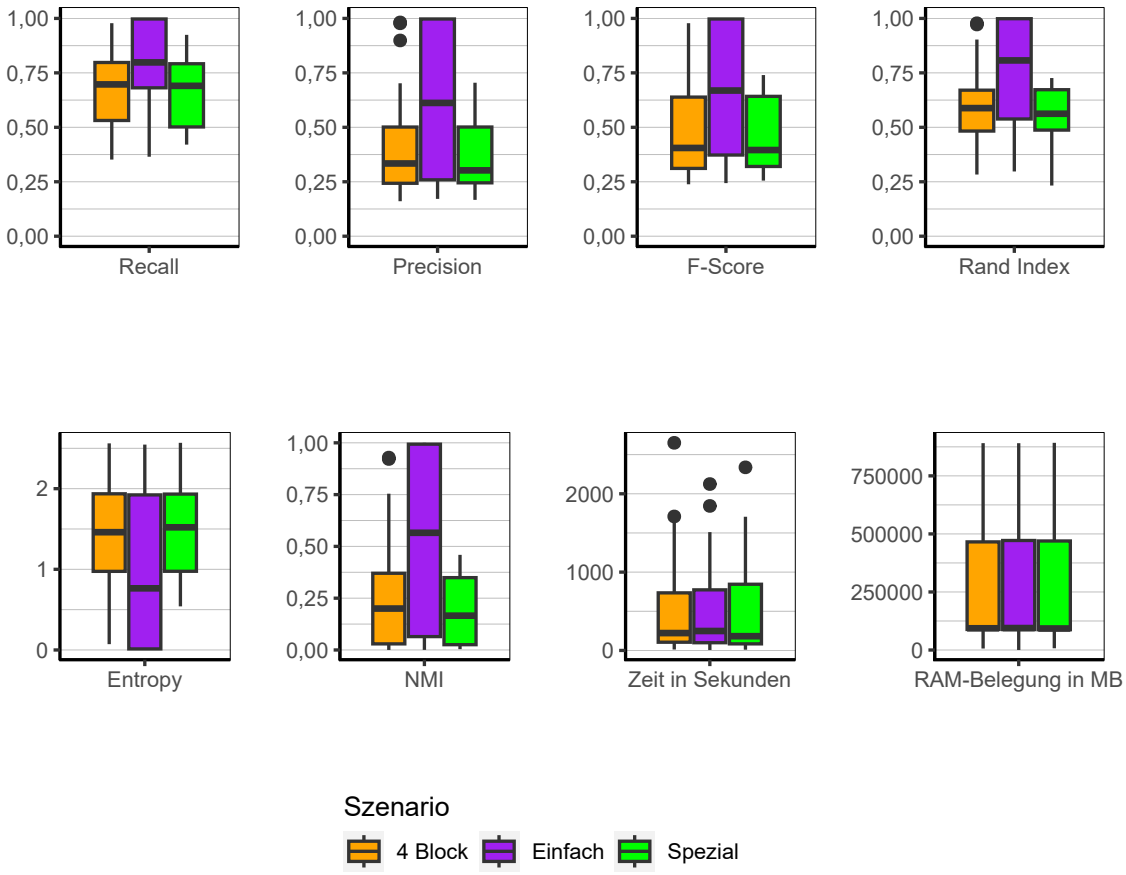


Abbildung 9.21.: CLARANS: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Grundsätzlich werden auch an dieser Stelle die Ergebnisse von BUBBLE und CLARA reproduziert. Ein Unterschied ist jedoch auffallend. Im Szenario *4 Block*, welches genau wie *Spezial* im Unterschied zu *Einfach* nicht so gut abgetrennte Clusterstrukturen aufweist und ebenfalls einige Beobachtungen enthält, welche als Rauschen charakterisiert werden könnten, gelingt es CLARANS zwei gute und zwei nahezu perfekte Ergebnisse zu erzielen. Diese Ergebnisse konnten bei mittlerer Prävalenz und zwei Clustern erreicht werden.

Fazit Wie bereits erwähnt lässt sich kaum ein Unterschied zu den Algorithmen BUBBLE und CLARA festellen. Der Algorithmus liefert vergleichbare Ergebnisse. Die Ergebnisse beim Szenario *4 Block* mit mittlerer Prävalenz und zwei Clustern ist jedoch besser als bei den anderen beiden Algorithmen. Dies kann als möglicher Hinweis interpretiert werden, dass CLARANS etwas besser mit Rauschen umgehen kann als BUBBLE und CLARA. Bei 4 oder 6 Clustern bzw. niedriger Prävalenz konnten diese Ergebnisse jedoch nicht reproduziert werden, weshalb diese Beobachtung nicht generalisiert werden kann. Daher sind BUBBLE und CLARA in den meisten Situationen vorzuziehen. Sie sind wesentlich schneller durchzuführen und benötigen sehr viel weniger Speicher.

9.3.4. CLOPE

In diesem Abschnitt werden die Ergebnisse des Algorithmus CLOPE analysiert.

Gesamt Die Ergebnisse über alle Faktorstufen in Abbildung 9.22 zeigen sehr eindeutig eine sehr schlechte Qualität der Clusterlösungen. Der Algorithmus tendiert noch stärker als die bereits beschriebenen Algorithmen dazu, sehr schiefe Verteilungen bezüglich der Größe der resultierenden Cluster zu bilden. Tatsächlich bestehen 78,33 % der Clusterlösungen aus lediglich einem Cluster mit allen Beobachtungen. Die restlichen Clusterings sind größtenteils ähnlich schief. Entsprechend hoch bzw. niedrig fallen die Werte für Recall und Precision aus. Die Abbildung der Gini-Koeffizienten (Abbildung 9.23) verdeutlicht diese Beobachtung. Nahezu alle Datenpunkte liegen auf oder nahe der 1. Die restlichen Qualitätsindikatoren spiegeln die Verteilung der Werte wieder, die sich als Konsequenz des beobachteten Verhaltens ergeben.

Die Performance wiederum kann als gut bezeichnet werden. Die Laufzeit ist kurz und der Speicherbedarf ist moderat.

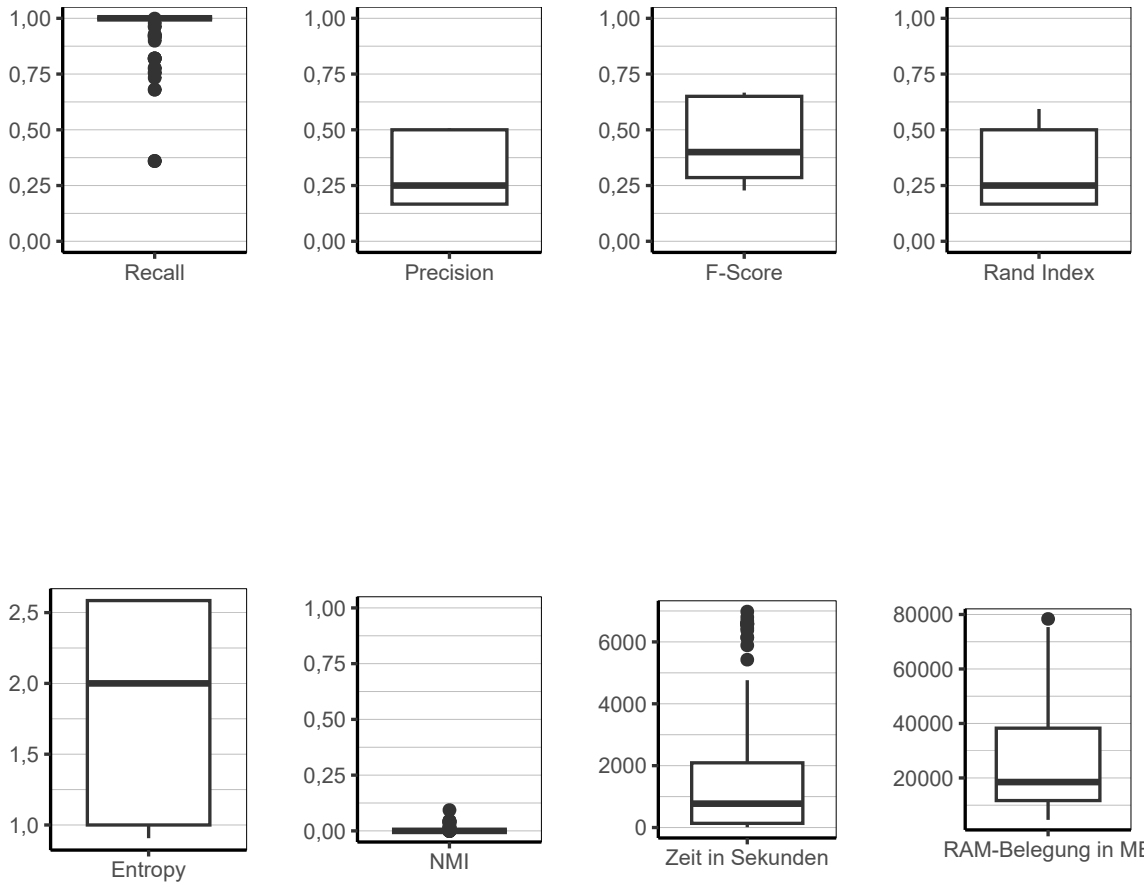


Abbildung 9.22.: CLOPE: Evaluation aller Iterationen der Simulation insgesamt.

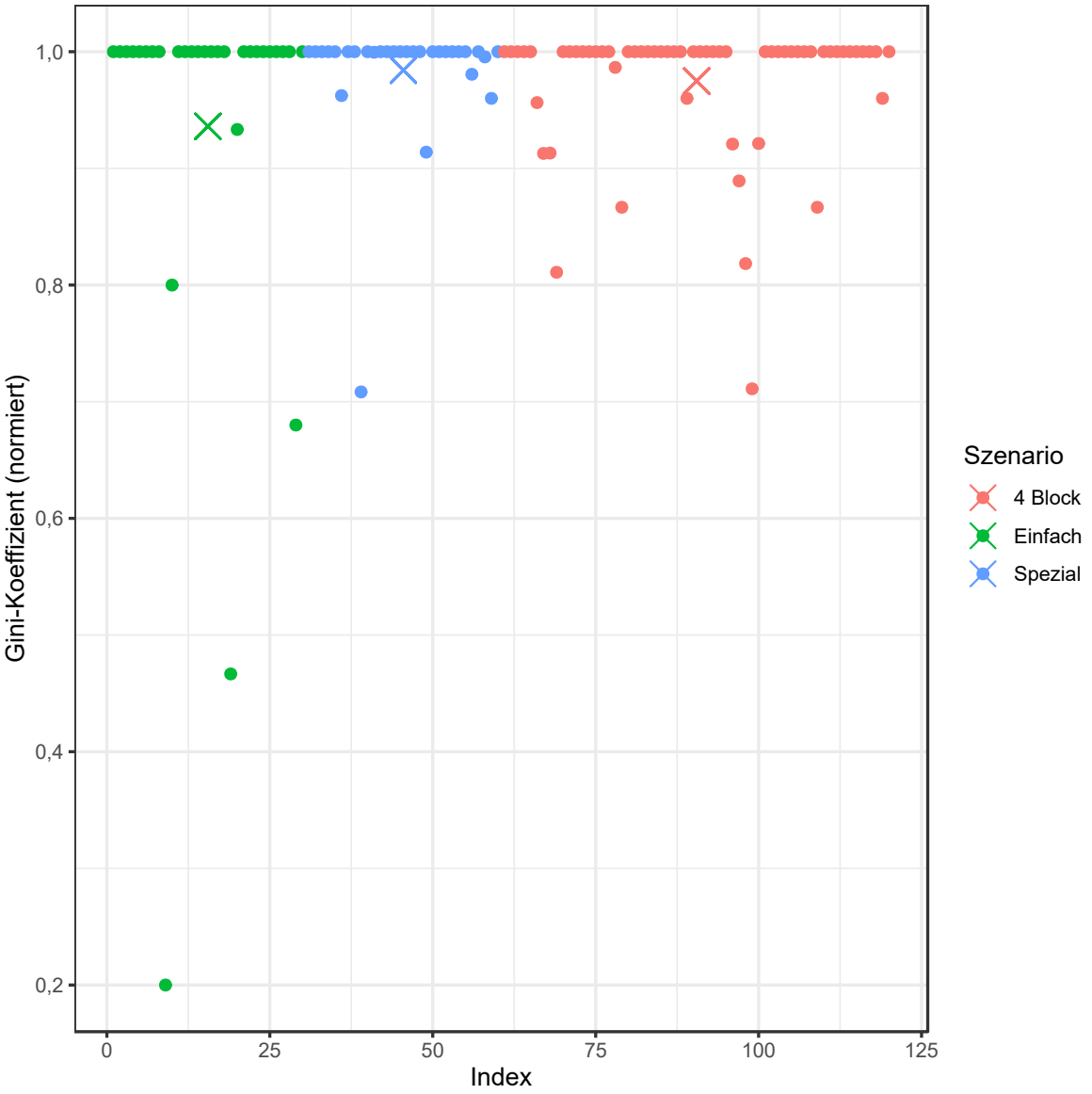


Abbildung 9.23.: CLOPE: Gini-Koeffizient aller Iterationen der Simulation.

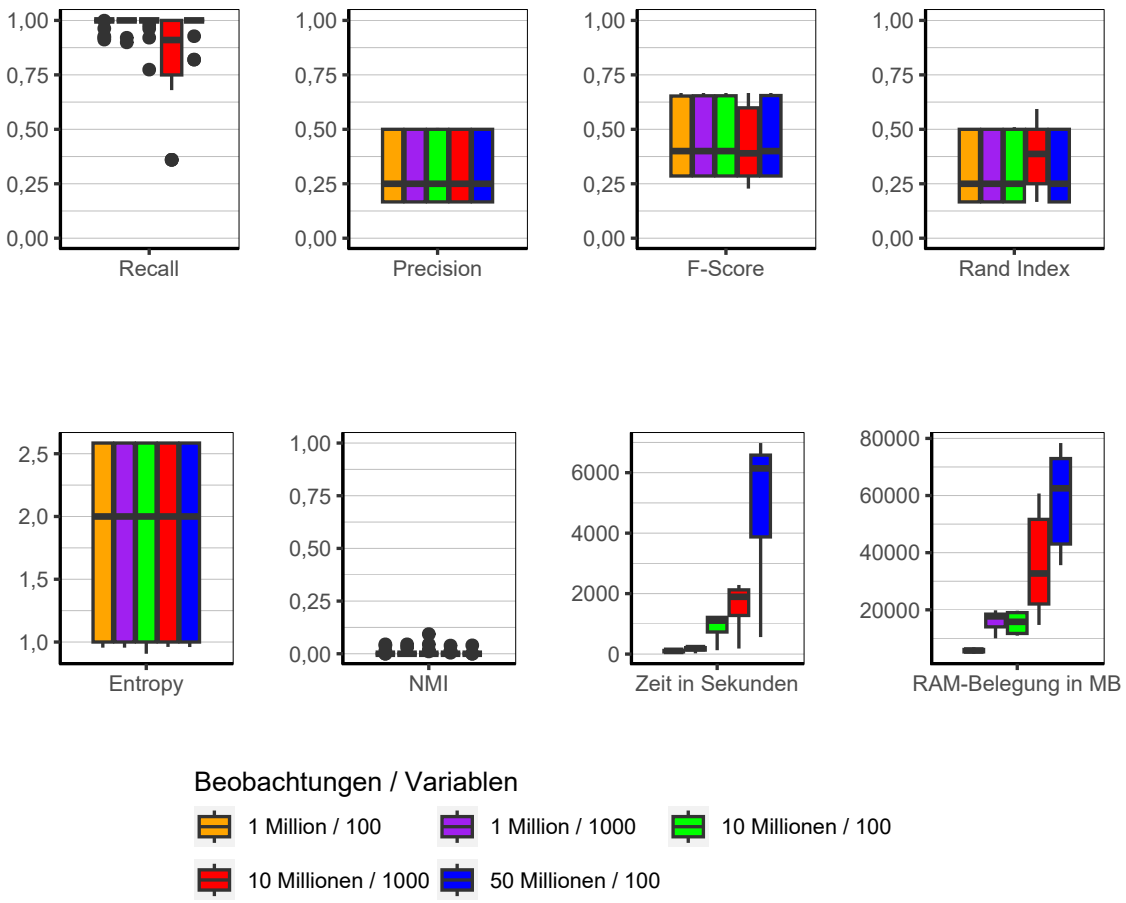


Abbildung 9.24.: CLOPE: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Bei Betrachtung von Abbildung 9.24 zeigt sich eindeutig, dass die Anzahl der Beobachtungen und die Anzahl der Variablen keinerlei Einfluss auf die Qualität der Clusterlösungen ausübt. Auffällig ist, dass sowohl die Laufzeit und der Speicherbedarf vor allem mit zunehmender Anzahl von Beobachtungen ansteigt. Die Anzahl der Variablen hat zwar ebenfalls einen Einfluss bei 10 Millionen Beobachtungen, dieser ist jedoch nicht so stark wie die Anzahl der Beobachtungen.

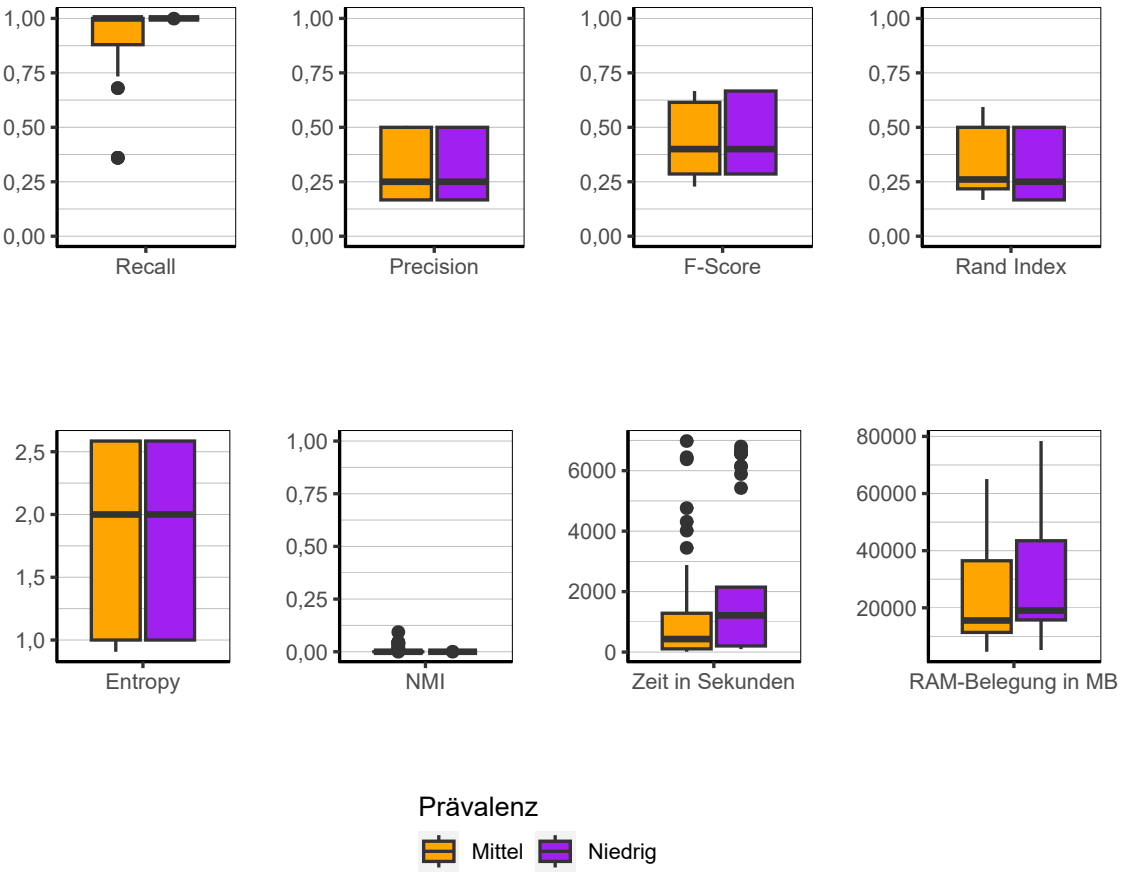


Abbildung 9.25.: CLOPE: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Die Prävalenz (Abbildung 9.25) übt keinerlei Einfluss auf die Güte der Clusterlösungen aus. Auch bei Laufzeit und Speicherbedarf, lassen sich keine großen Unterschiede erkennen. Zwar scheint der Algorithmus bei höherer Prävalenz etwas schneller zu laufen und etwas weniger Speicher zu benötigen, die Unterschiede sind jedoch gering.

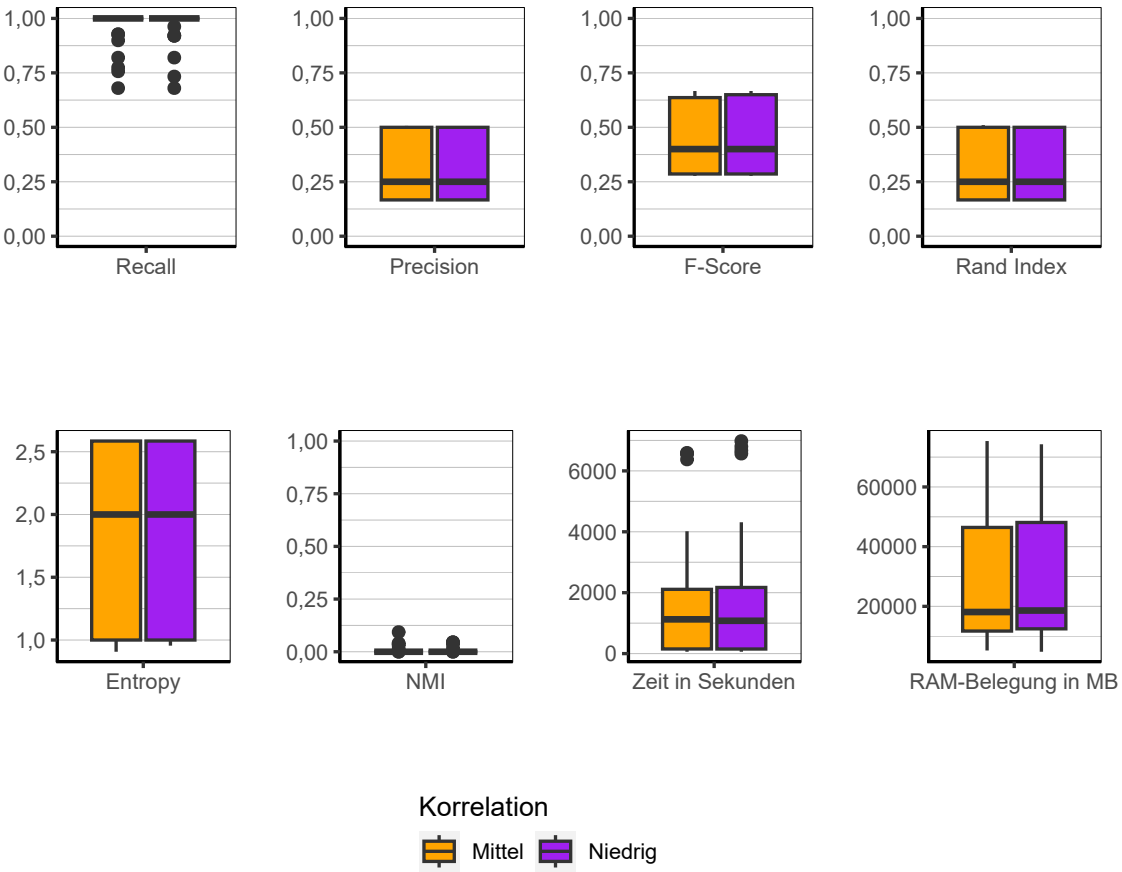


Abbildung 9.26.: CLOPE: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Bei Analyse von Abbildung 9.26 zeigt sich, dass die Korrelationsstruktur weder bei der Qualität der Clusterlösungen als auch bei der Performance des Algorithmus keine Rolle spielt.

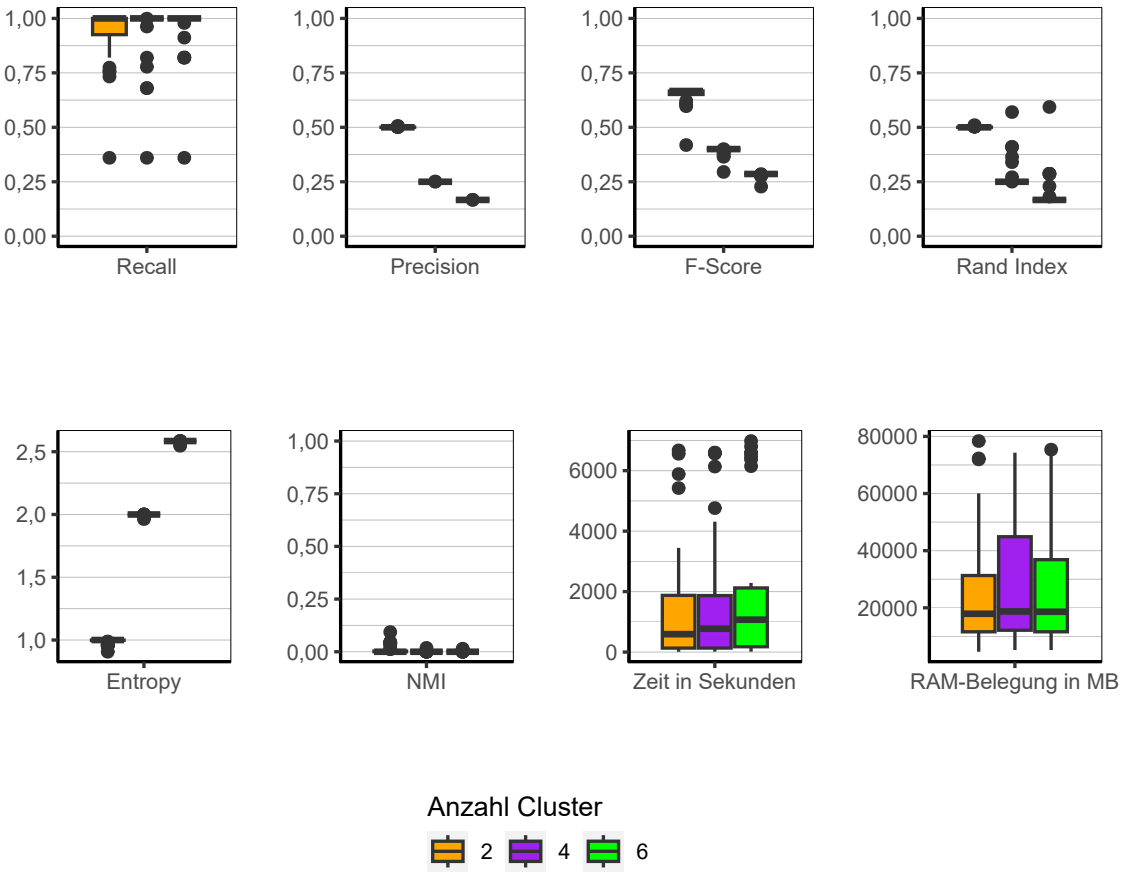


Abbildung 9.27.: CLOPE: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster bei der Anzahl der Cluster sind auf den ersten Blick vermeintliche Unterschiede zu erkennen (Abbildung 9.27). Jedoch handelt es sich dabei nur um die unterschiedlichen Werte, die sich bei der Berechnung der jeweiligen Qualitätsindikatoren zwangsläufig ergeben, wenn alle Beobachtungen in einem Cluster sind, sich jedoch die Anzahl der Cluster in den bekannten „original“ Clusterlabels ändert, welche bei der Evaluierung zugrunde liegt.

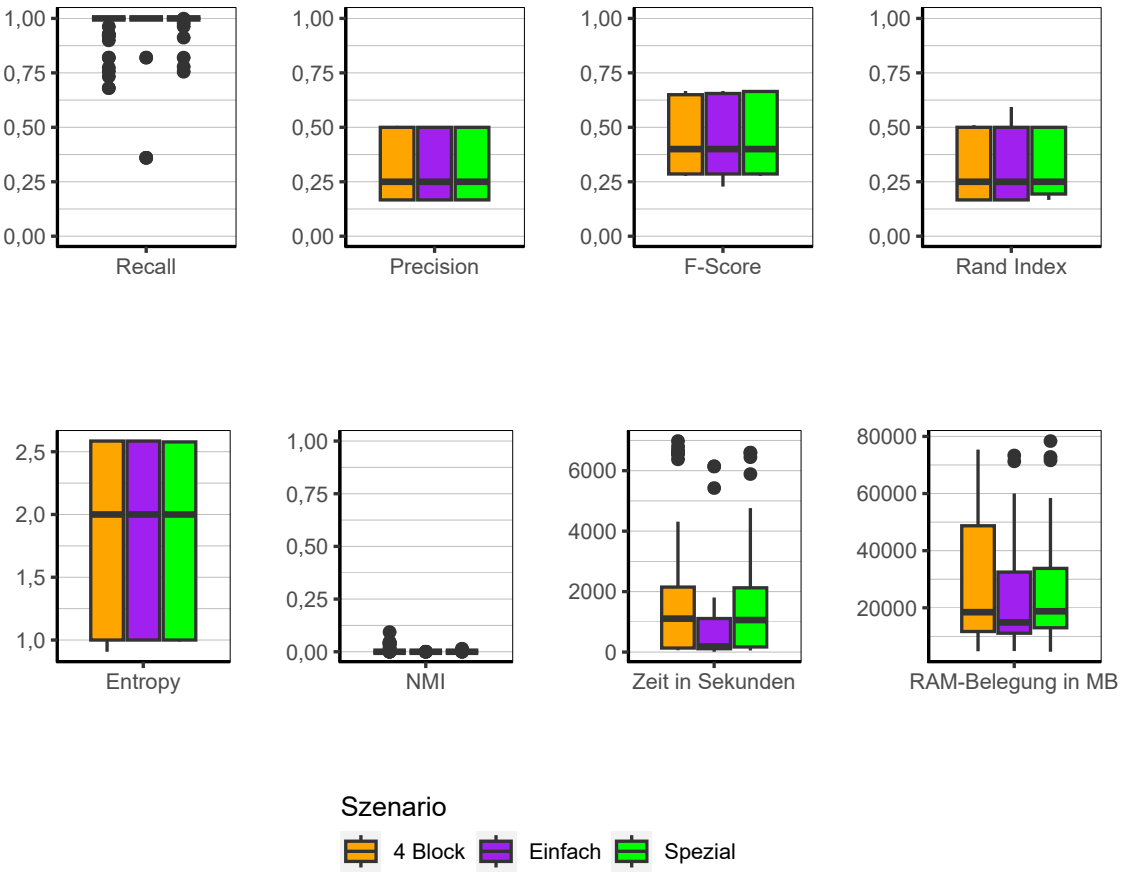


Abbildung 9.28.: CLOPE: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Auch in den unterschiedlichen Szenarien zeigen sich keine Unterschiede in den Ergebnissen. Das bedeutet, dass selbst bei eindeutig getrennten Clustern, keine Unterschiede in den Clusterlösungen zu erkennen sind.

Fazit Der Algorithmus ist in keiner Weise geeignet, die simulierten Daten zu clustern. Es werden durchweg schlechte Clusterlösungen erzeugt. Keiner der Faktoren im Simulationsaufbau scheint darauf einen Einfluss zu haben.

9.3.5. Large Item

In diesem Abschnitt werden die Ergebnisse des Algorithmus Large Item besprochen.

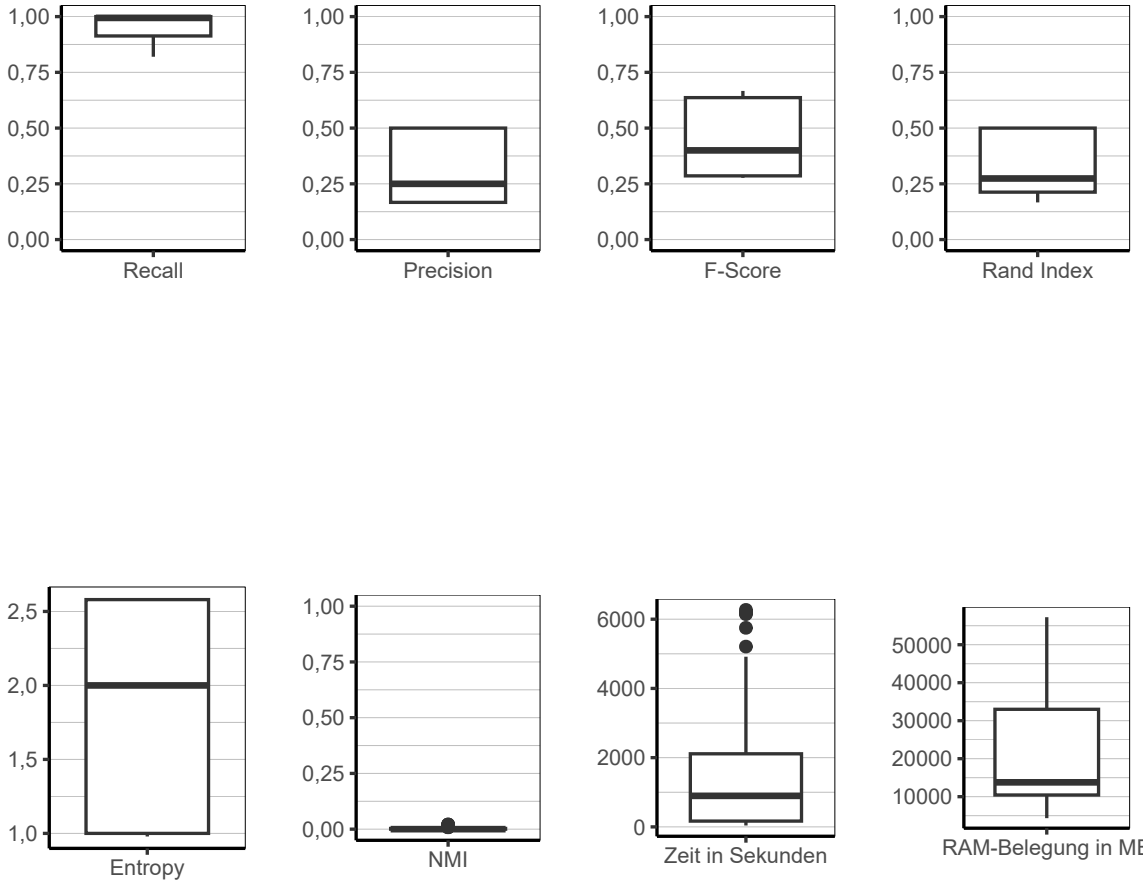


Abbildung 9.29.: Large Item: Evaluation aller Iterationen der Simulation insgesamt.

Gesamt Wie sich in Abbildung 9.29 sehen lässt, zeigt der Algorithmus Large Item ein sehr ähnliches Verhalten wie CLOPE. Dies ist nicht verwunderlich, aufgrund der Tatsache, dass sich die beiden Algorithmen sehr ähneln. Auch hier kann wieder festgestellt werden, dass alle Clusterlösungen alle bzw. fast alle Beobachtungen demselben Cluster zuweisen. In diesem Falle bestehen 50 % der Clusterings aus lediglich einem Cluster mit allen Beobachtungen. Siehe dazu auch Abbildung 9.30 mit den Gini-Koeffizienten. Damit zeigt sich, dass dieses Verhalten etwas weniger stark ausgeprägt ist, als bei CLOPE, aber dennoch dazu führt, dass der Algorithmus zumindest für die Datenformen, welche in dieser Arbeit simuliert werden, nicht geeignet ist. Zumal die restlichen Clusterings, welche nicht nur aus einem Cluster bestehen, wie bereits angesprochen immernoch sehr ungleich verteilt sind und auch nie mehr als zwei Cluster aufweisen. Die Laufzeit ist ebenfalls nahezu identisch zu CLOPE. Nur der Speicherbedarf scheint etwas höher zu sein.

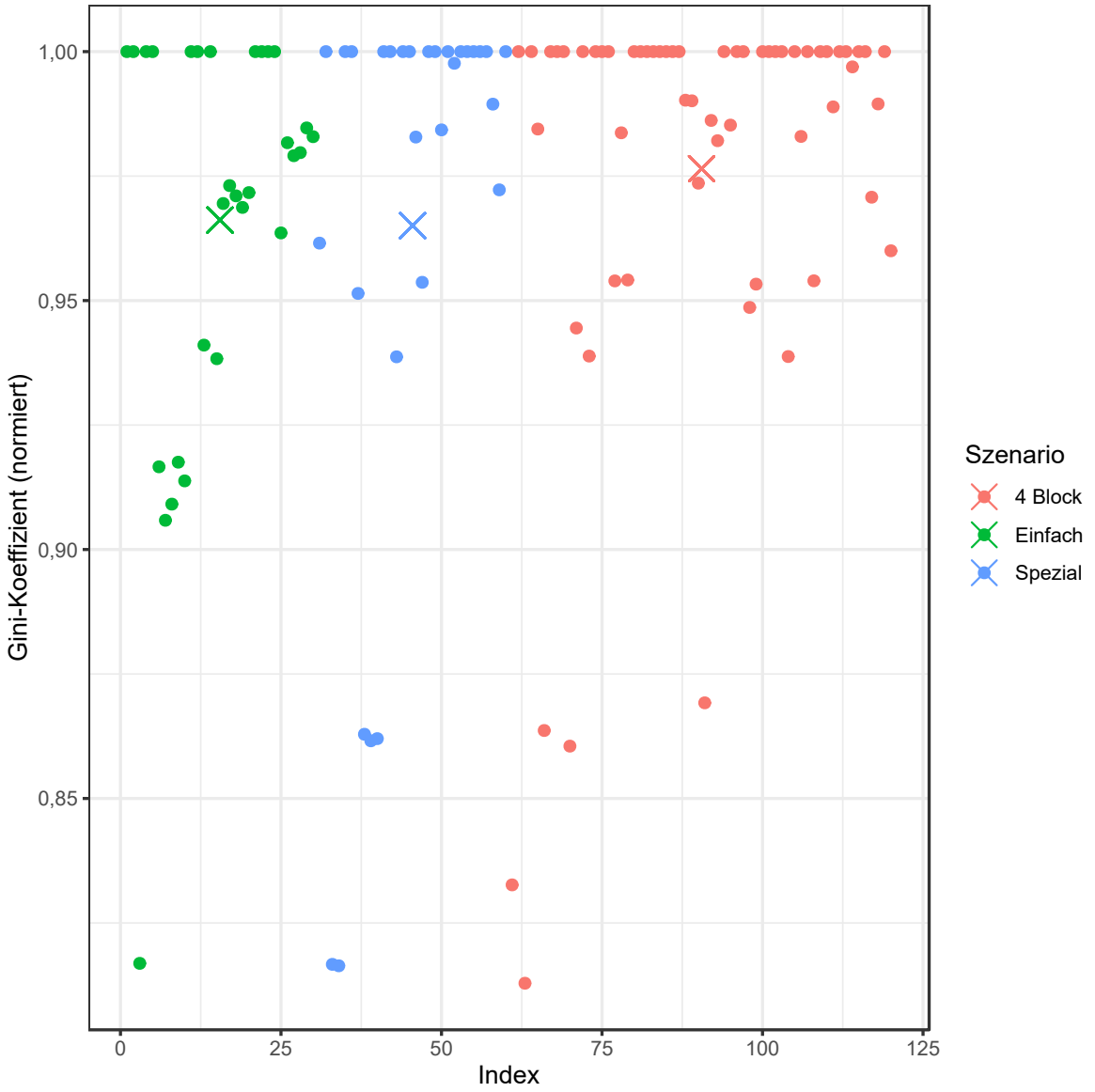


Abbildung 9.30.: Large Item: Gini-Koeffizient aller Iterationen der Simulation.

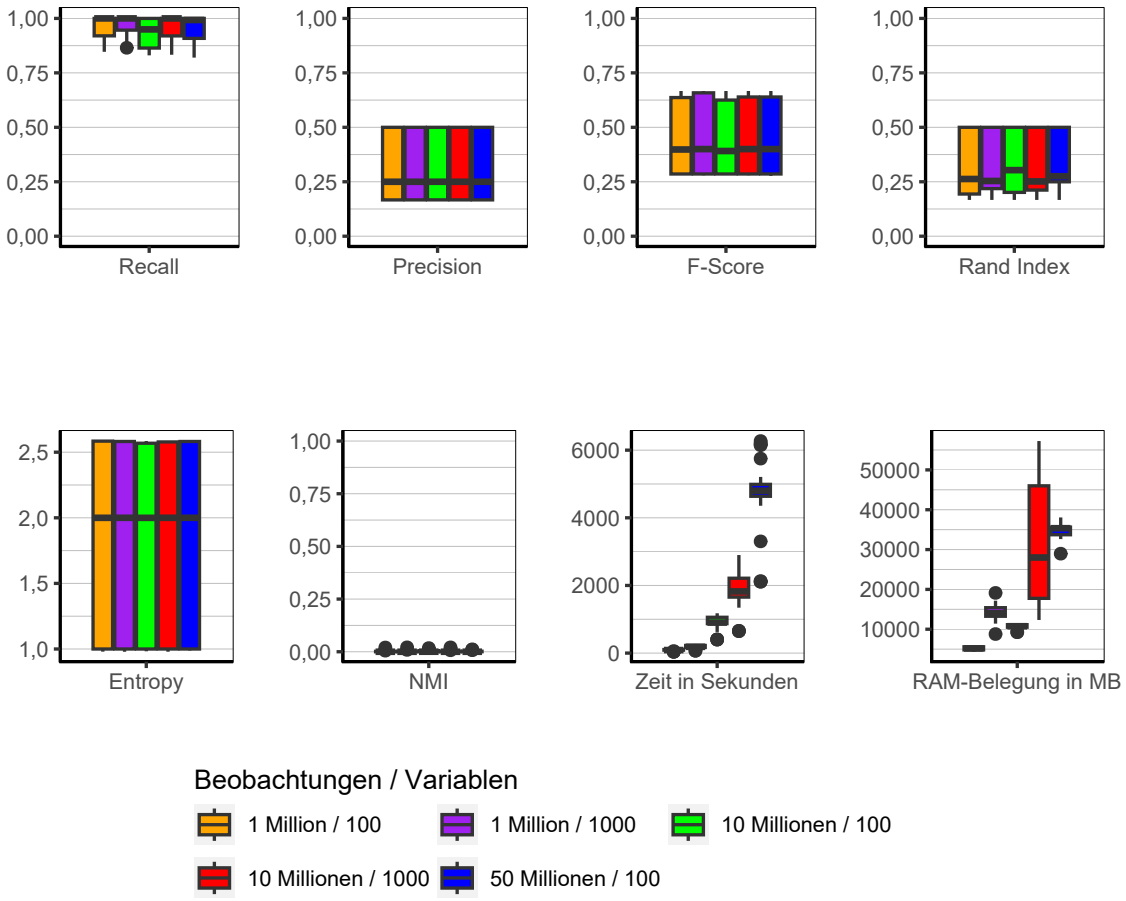


Abbildung 9.31.: Large Item: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Aufgrund der Tatsache, dass der Algorithmus fast immer fast alle oder alle Beobachtungen in einen Cluster allokiert, zeigt sich bezüglich der Anzahl der Beobachtungen und Variablen (Abbildung 9.31) kein Unterschied in den unterschiedlichen Konfigurationen.

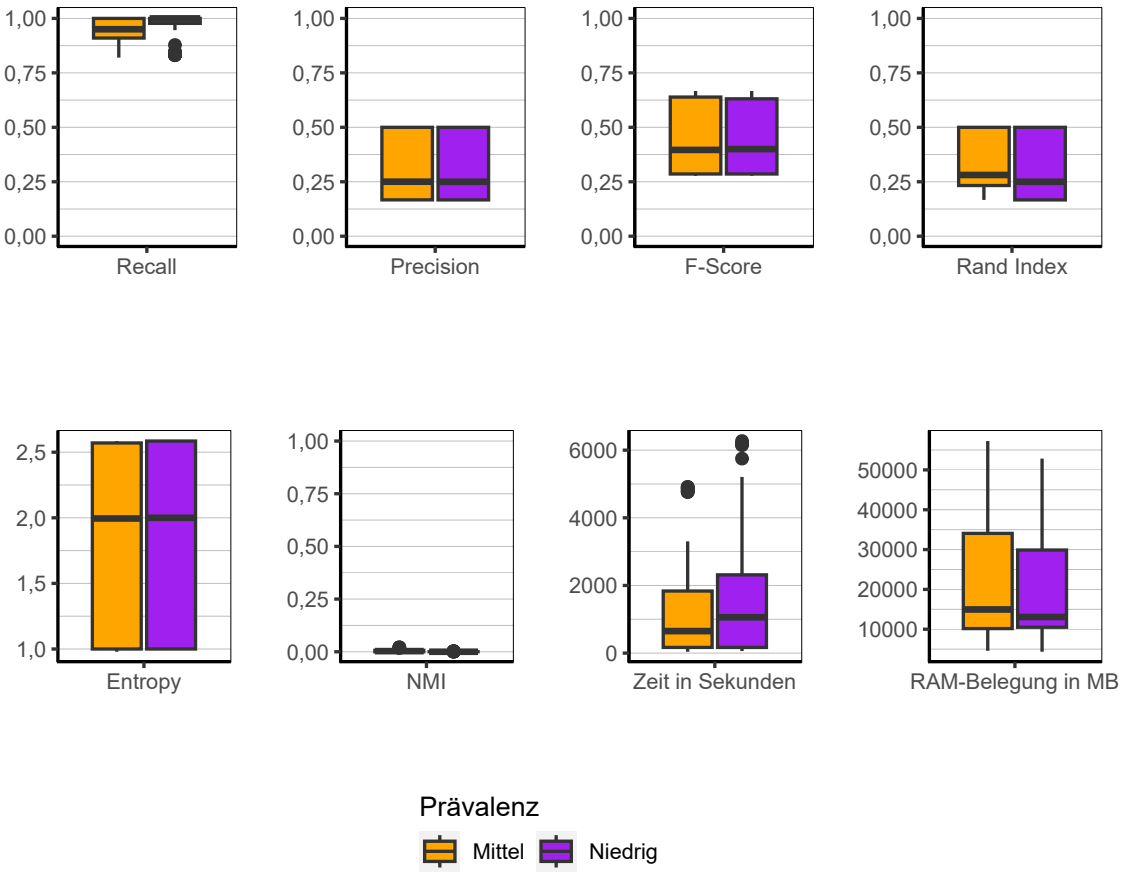


Abbildung 9.32.: Large Item: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Es können keine Unterschiede in den verschiedenen Konfigurationen der Prävalenz ausgemacht werden (siehe Abbildung 9.32). Dies gilt sowohl für die Güte der Clusterlösungen als auch für Laufzeit und Speicherbedarf.

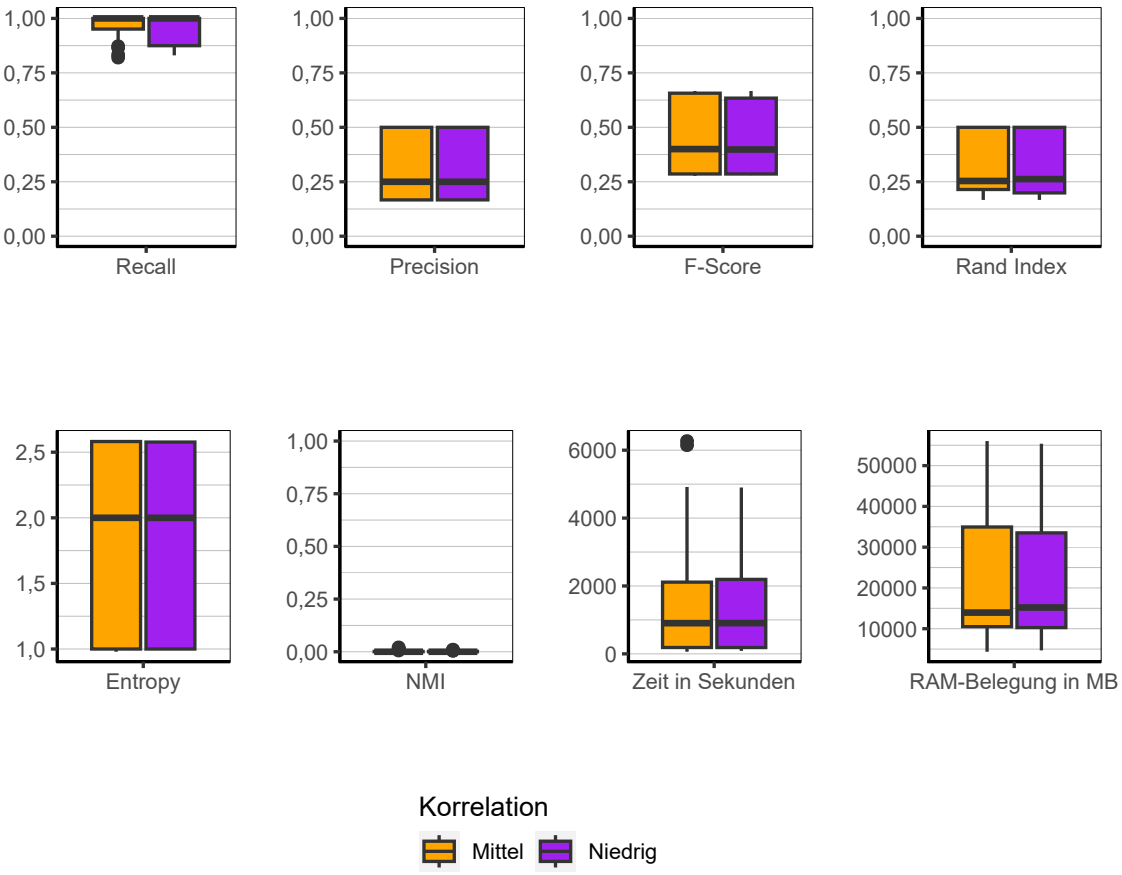


Abbildung 9.33.: Large Item: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Wie auch schon die Prävalenz, hat die Korrelationsstruktur weder Einfluss auf die Qualität der Clusterlösungen, noch auf die Laufzeit oder den Speicherbedarf (siehe Abbildung 9.33).

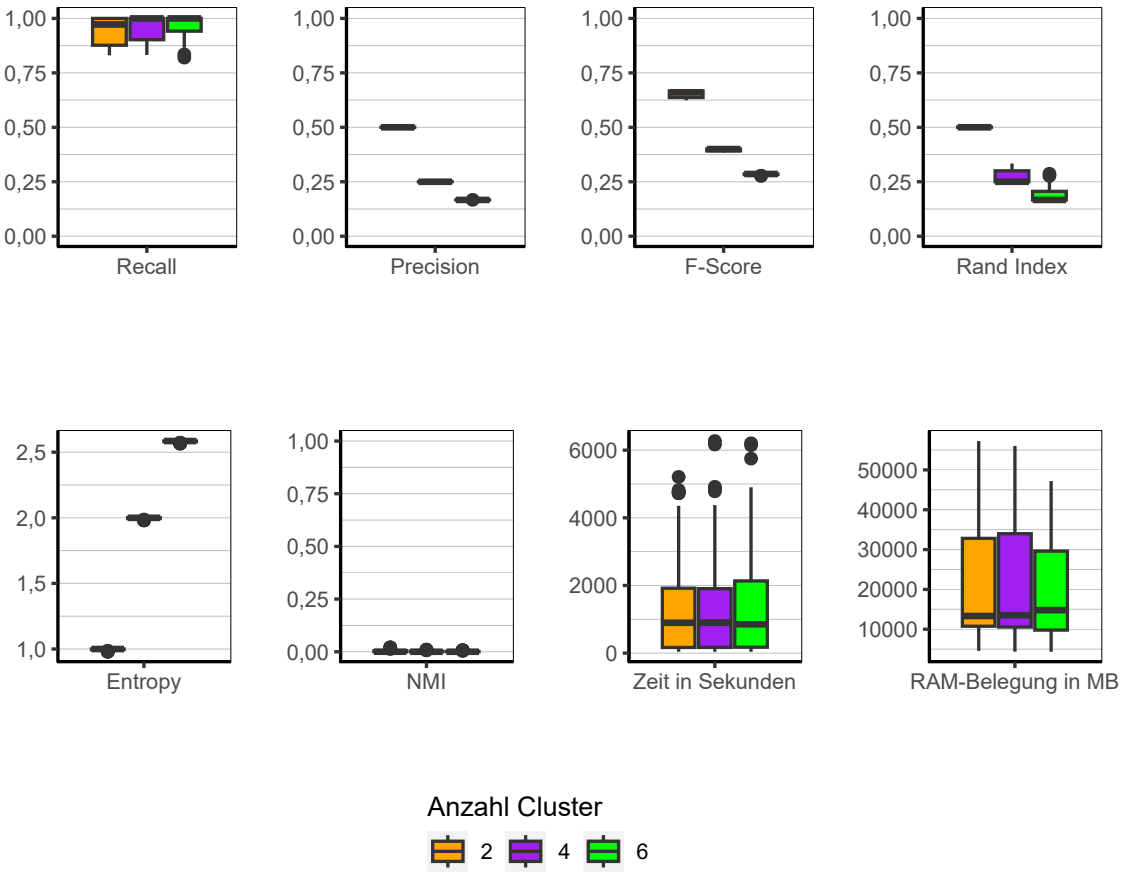


Abbildung 9.34.: Large Item: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Bezüglich der Anzahl der Cluster kann in Abbildung 9.34 beobachtet werden, dass mit Anstieg der Anzahl der Cluster die Qualität der Clusterlösungen abnimmt.

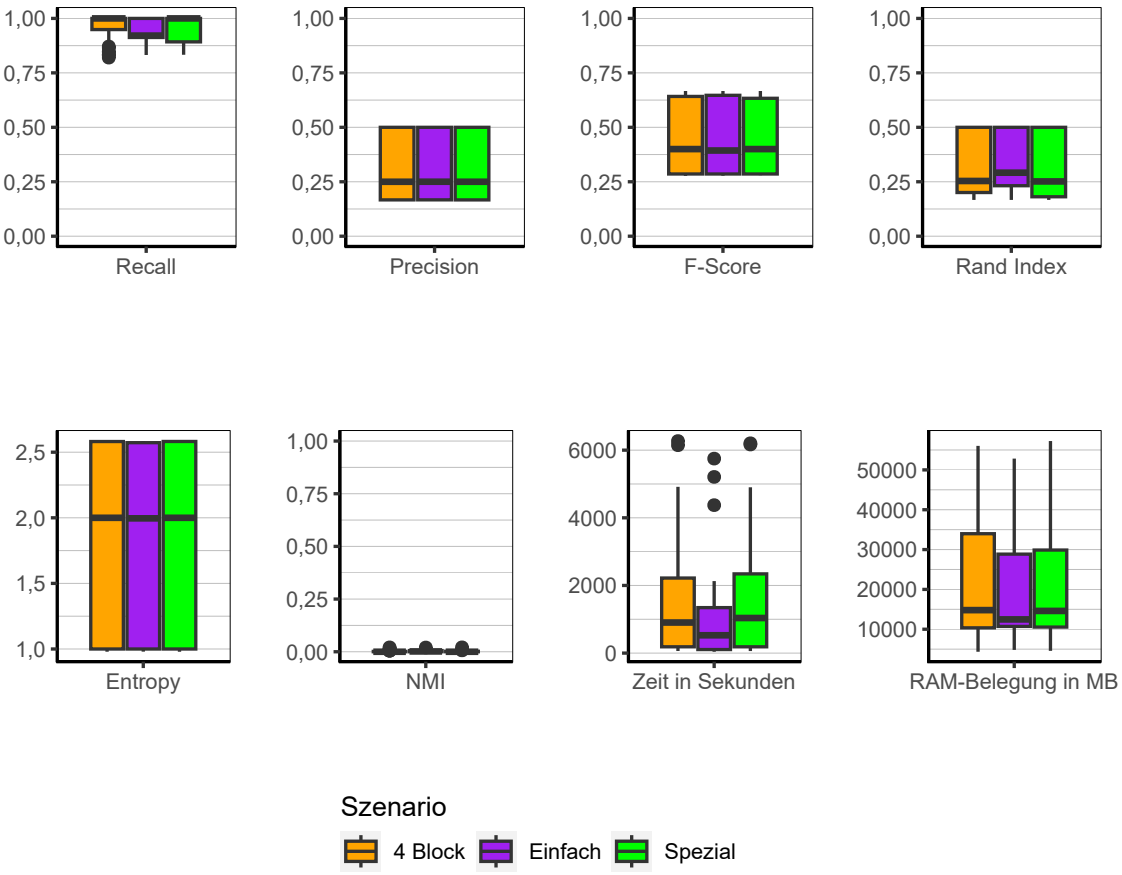


Abbildung 9.35.: Large Item: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Anzahl Cluster Auch das Szenario scheint keinerlei Einfluss auf die Ergebnisse des Algorithmus auszuüben (siehe Abbildung 9.35).

Fazit Wie auch CLOPE ist Large Item nicht in der Lage akzeptable Clusterlösungen hervorzubringen. Keine der verschiedenen Faktorstufen des Studiendesigns hat darauf einen Einfluss.

9.3.6. SCALE

Im Folgenden werden die Ergebnisse des Algorithmus SCALE analysiert.[3]

Gesamt Die Ergebnisse für SCALE sind die interessantesten in der gesamten Simulation. Aufgrund der Ähnlichkeit im Ablauf zu den beiden vorangegangenen Algorithmen CLOPE und Large Item, wäre zu vermuten, dass sich die Ergebnisse auch hier stark ähneln. Das Gegenteil ist jedoch der Fall. Die resultierenden Clusterlösungen sind mit Abstand die qualitativ hochwertigsten in der gesamten Simulation (siehe Abbildung 9.36). Das Phänomen, die meisten Beobachtungen in einem Cluster zu bündeln, was den vorangegangenen Algorithmen noch gemein war, lässt sich hier überhaupt nicht beobachten. Die Clustergrößen sind sehr ausgeglichen, was sich gut in Abbildung 9.37 erkennen lässt. Die meisten Werte der berechneten Gini-Koeffizienten befindet sich auf oder nahe der 0. Die Mittelwerte nach Szenario liegen bei 0,12, 0,27 und 0,05 für die Szenarien *Einfach, Spezial* und *4 Block*. Die Werte für Recall, Precision und den F-Score bestätigen diese Beobachtung. Alle drei haben einen Median von ungefähr 0.75 und das 75 %-Quantil befindet sich bei 1. Fast die komplette Box des Rand Index befindet sich zwischen 0.75 und 1. Entsprechend ist die Entropie niedrig und NMI hoch. All dies deutet auf sehr gute Clusterings der allermeisten simulierten Datensätze hin. Allerdings werden die guten Ergebnisse durch eine sehr lange Laufzeit relativiert. Diese zählt zu den längsten von allen betrachteten Algorithmen. Der Speicherbedarf wiederum kann als moderat eingestuft werden.

[3] Aufgrund eines Segfault Errors bei der Konfiguration mit 10 Millionen Beobachtungen, 1.000 Variablen, mittlerer Prävalenz, Szenario *Einfach* und zwei Clustern konnten die Ergebnisse für diese Konfiguration nicht berichtet werden.

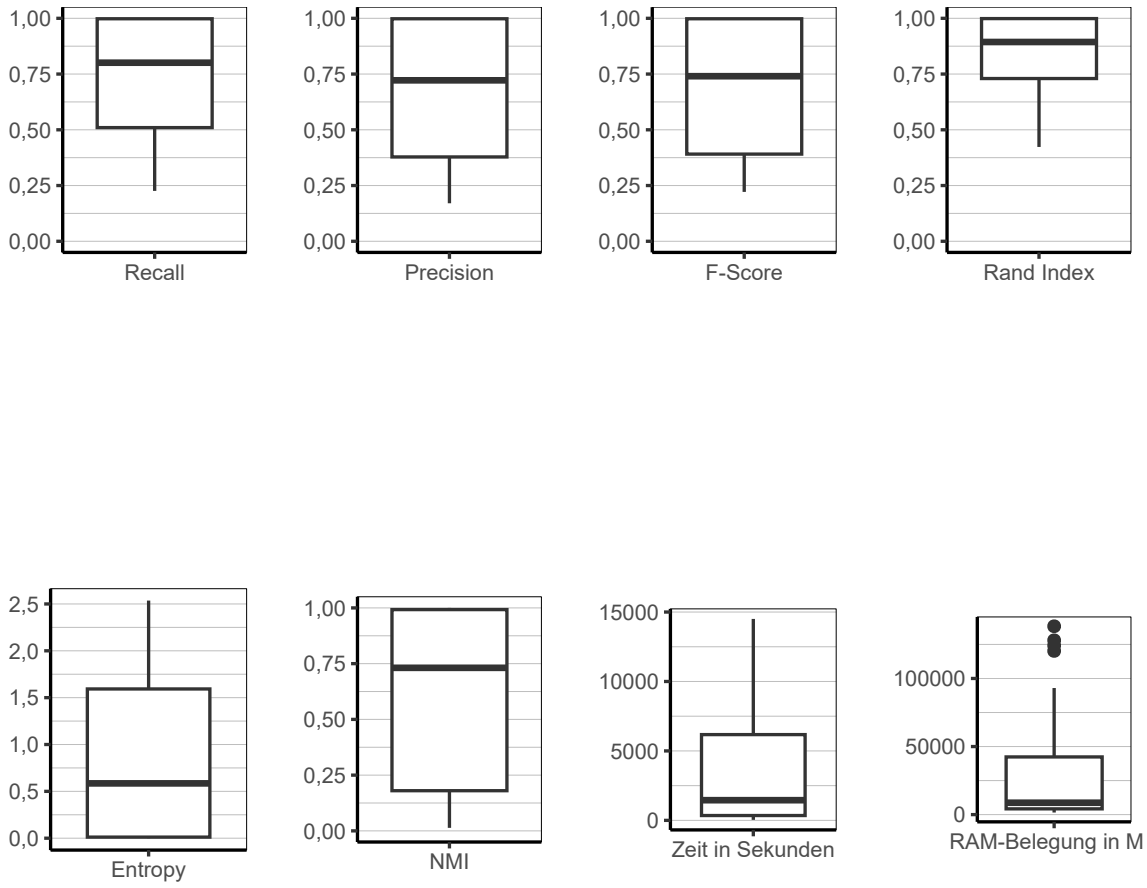


Abbildung 9.36.: SCALE: Evaluation aller Iterationen der Simulation insgesamt.

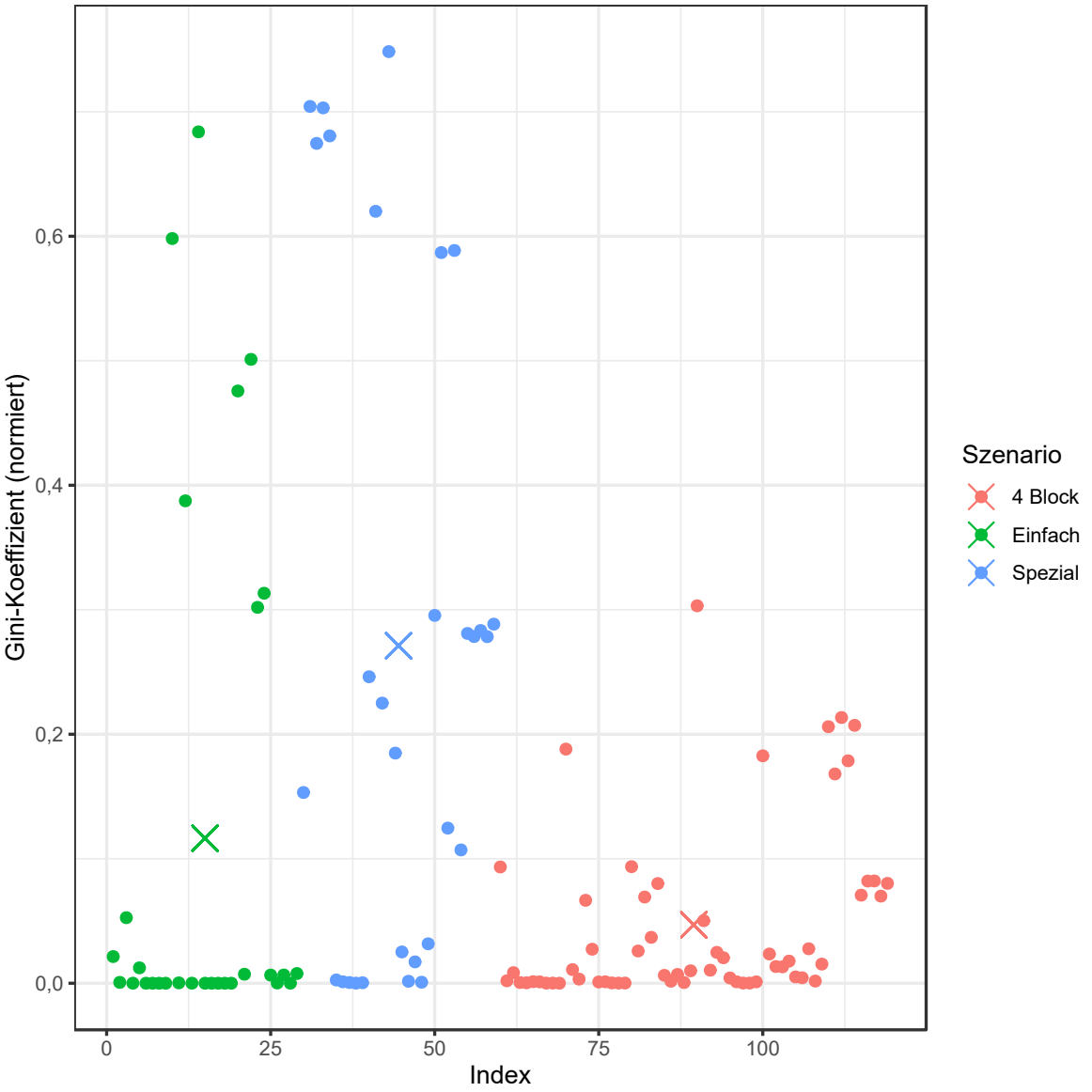


Abbildung 9.37.: SCALE: Gini-Koeffizient aller Iterationen der Simulation.

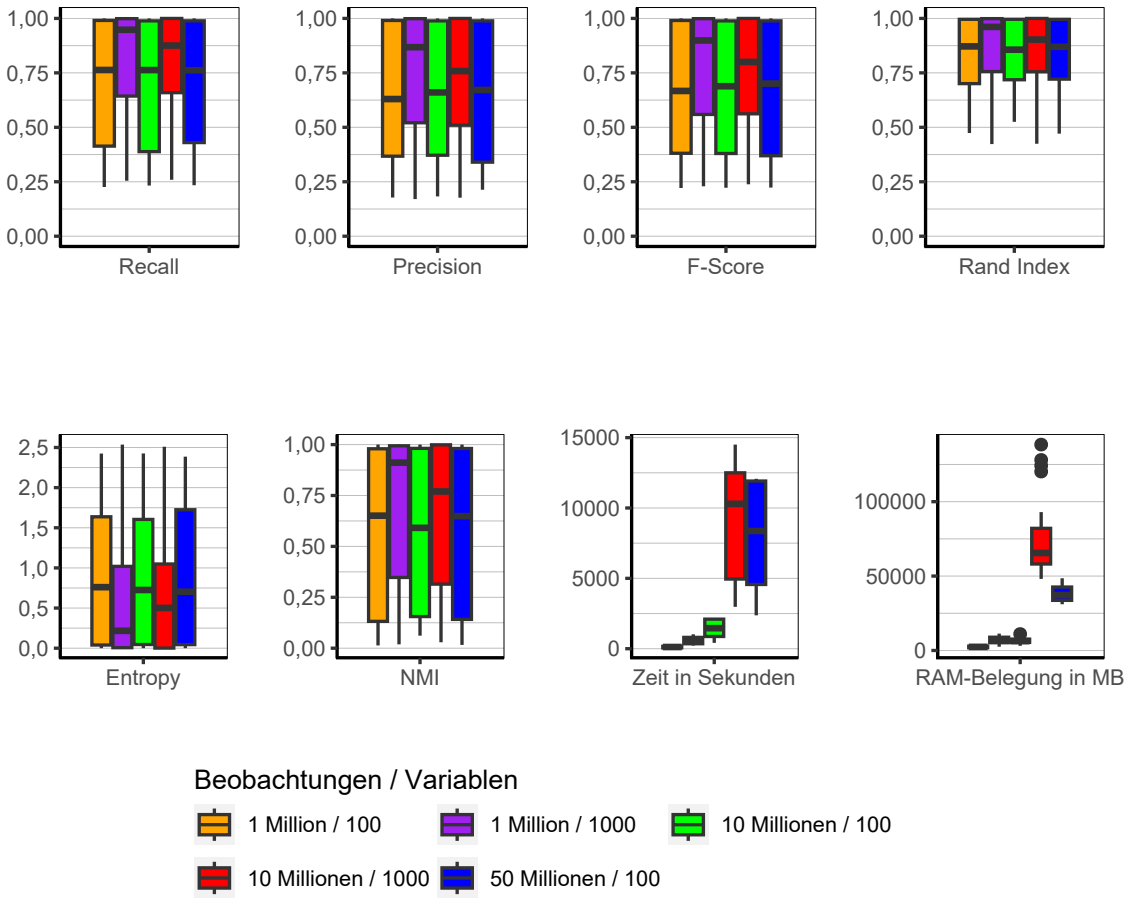


Abbildung 9.38.: SCALE: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Bei der Dimensionalität (siehe Abbildung 9.38) fällt auf, dass die Ergebnisse etwas besser ausfallen, wenn 1.000 Variablen vorhanden sind. Zwar sind die oberen Grenzen der Boxen auf einem sehr ähnlichen Level, jedoch sowohl die Mediane als auch die unteren Grenzen befinden sich höher. Im Falle der Entropie muss der Plot entsprechend aus der anderen Richtung gelesen werden. Das bedeutet, dass sich bei mehr Variablen bessere Ergebnisse erzielen lassen, obgleich auch die Ergebnisse mit 100 Variablen in vielen Fällen gut bzw. sehr gut sind. Ein Grund hierfür könnte

darin liegen, dass es für den Algorithmus einfacher ist, Bereiche mit hoher Dichte zu identifizieren, wenn mehr Variablen vorhanden sind. Somit entsteht ein differenzierteres Optimierungskriterium und der Algorithmus ist besser in der Lage die Beobachtungen den korrekten Clustern zuzuweisen. Bezüglich Laufzeit und Speicherbedarf zeigt sich eine scheinbar höhere Sensitivität gegenüber der Anzahl der Variablen als gegenüber der Anzahl der Beobachtungen. Dies wird vor allem deutlich, wenn die Ergebnisse von 10 Millionen Beobachtungen und 1.000 Variablen mit den Ergebnissen von 50 Millionen Beobachtungen und 100 Variablen verglichen werden.

Interessant ist in diesem Fall der Vergleich mit den ähnlichen Algorithmen CLOPE und Large Item. Diese weisen einen Anstieg von Laufzeit und Speicherbedarf auf, welcher linear mit der Anzahl der Beobachtungen skaliert. Der Unterschied ist hier entweder in der Berechnung der unterschiedlichen Optimierungskriterien zu suchen oder in der anderen Art und Weise der Parallelisierung.

Prävalenz In Abbildung 9.39 zeigt sich, dass bei SCALE die Prävalenz einen starken Einfluss auf die Güte der Clusterlösungen zu haben scheint. In der höheren der beiden Varianten sind die Ergebnisse durchweg sehr gut. Bei niedriger Prävalenz wiederum rangieren die Ergebnisse aller Qualitätsindikatoren über ein größeres Intervall. Es sind somit sowohl schlechte als auch gute Ergebnisse zu sehen, wobei in einigen Fälle sogar perfekte Clusterings beobachtet werden können. Zudem ist die Laufzeit bei höherer Prävalenz geringer als bei niedriger.

Korrelation Ein weiteres wirklich interessantes Ergebnis kann bei der Betrachtung der Auswirkung der unterschiedlichen Ausprägung der Korrelation beobachtet werden. Während bislang bei allen betrachteten Algorithmen die Korrelation keinen merklichen Einfluss auf die Ergebnisse genommen hat, ist dies bei SCALE augenscheinlich der Fall. So erscheinen die Ergebnisse bei niedriger Korrelation besser (Median F-Score: 0,94 vs 0,59), weisen jedoch auch eine höhere Varianz auf (Interquartilsabstand F-Score: 0,56 vs 0,38).

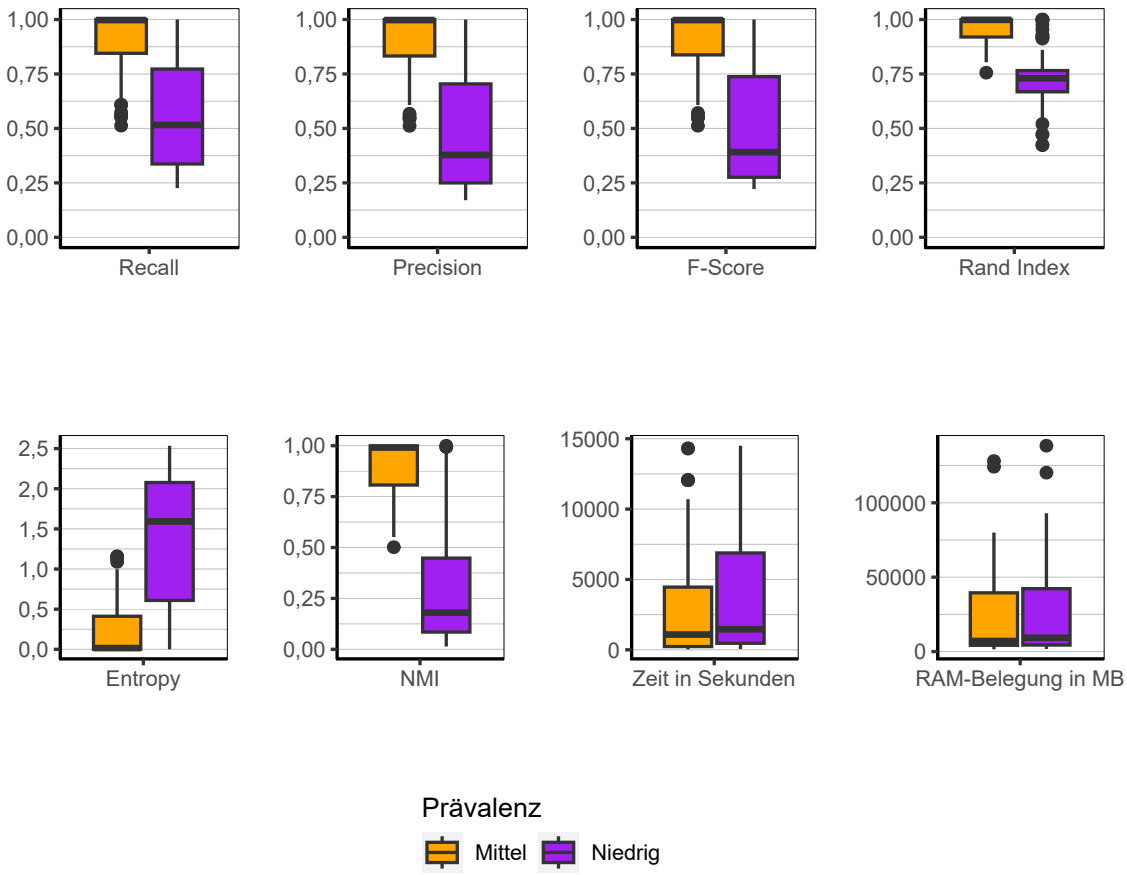


Abbildung 9.39.: SCALE: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

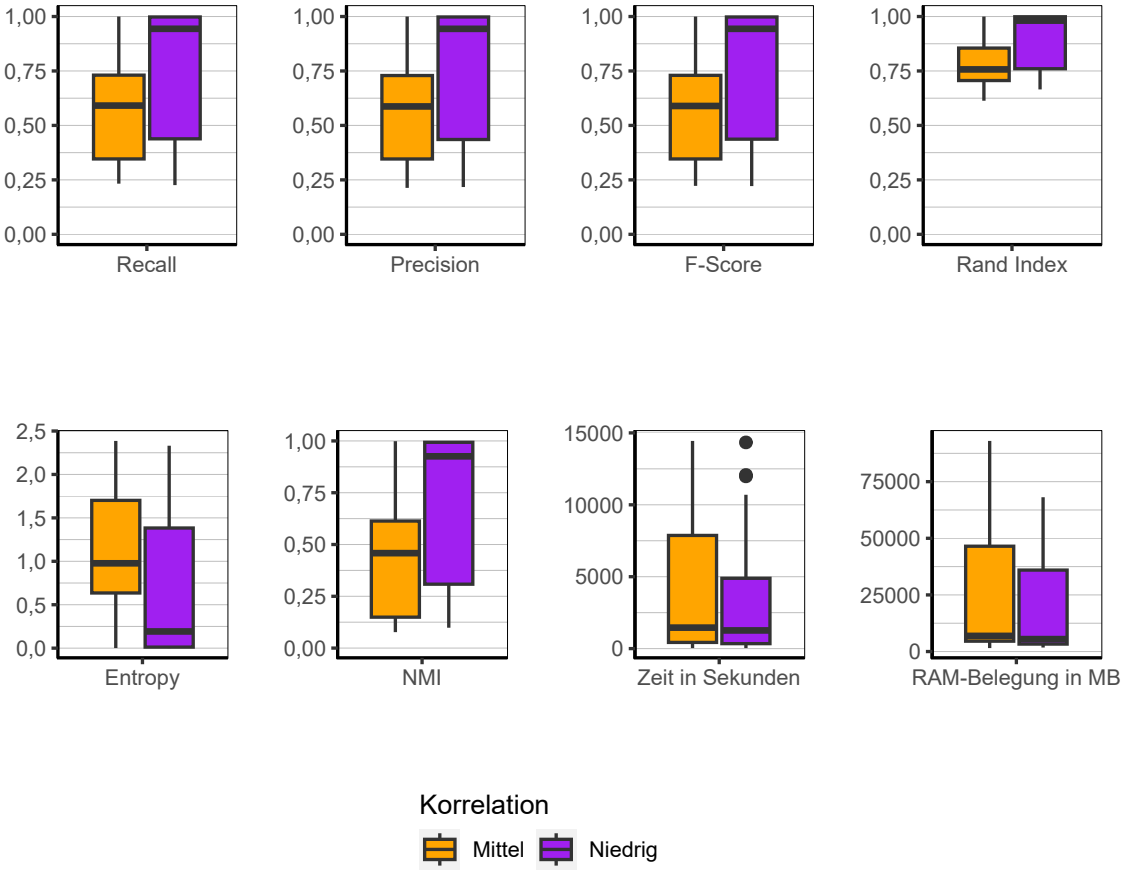


Abbildung 9.40.: SCALE: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

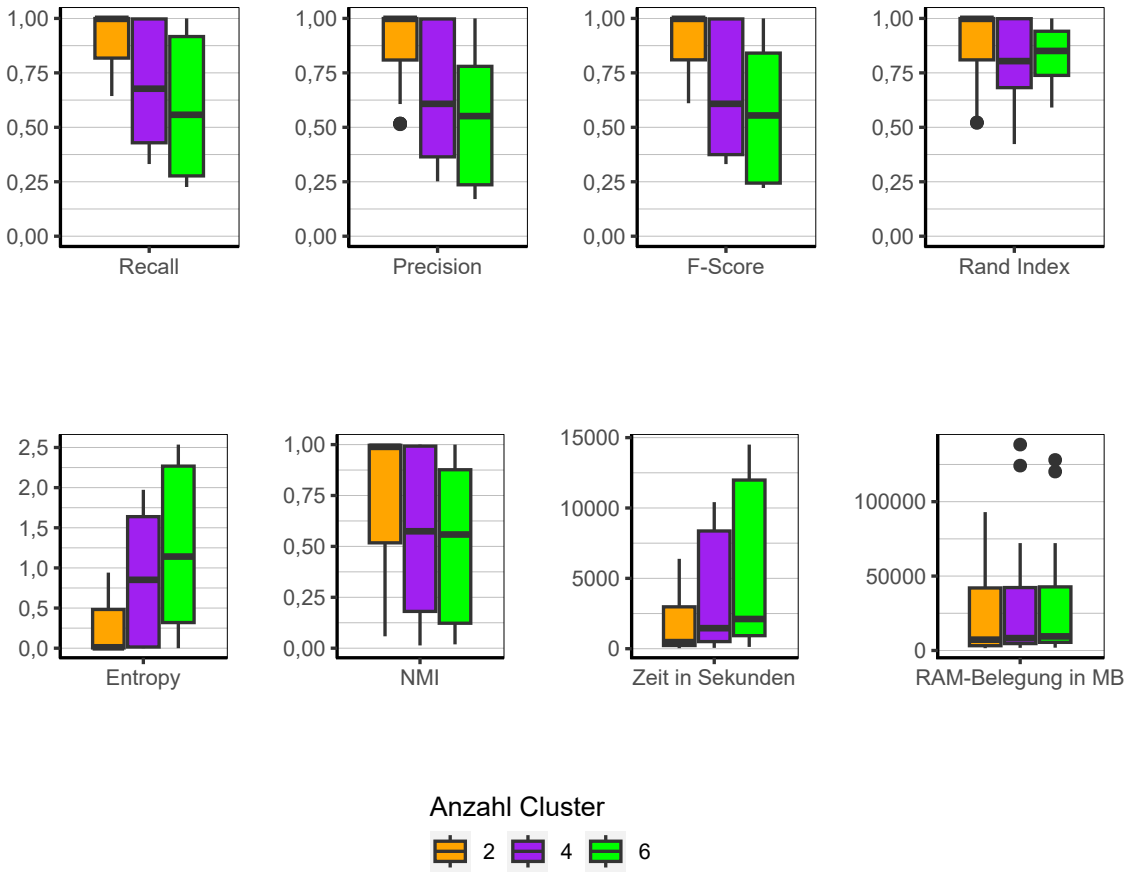


Abbildung 9.41.: SCALE: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Die Anzahl der Cluster (siehe Abbildung 9.41) wirkt sich mit Zunahme negativ auf die Qualität der Clusterlösungen aus. Bei zwei Clustern sind die Ergebnisse sehr gut. Der Median des F-Score liegt bei 1 und der Interquartilsabstand bei 0,19, was eine geringe Streuung der Ergebnisse signalisiert. Mit Anstieg der Anzahl der Cluster verschiebt sich dies substanziell nach unten. Bei vier Clustern liegt der Median des F-Score bei 0,61, der Interquartilsabstand bei 0,62). Auch wenn der Median stark gefallen ist und die Streuung gestiegen, liegt das 75 % Quartil immer noch bei 1. Es gibt demnach trotz

der im Median gefallenen Qualität der Clusterlösungen viele sehr gute Ergebnisse auch bei vier Clustern. Bei sechs Clustern liegt der Median mit 0,55 auf einem ähnlichen Niveau wie bei vier Clustern. Das 75 % Quartil jedoch nur bei 0,84. Bezüglich der benötigten Zeit steigt diese mit Anstieg der Cluster an. Beim Speicherbedarf können keine nennenswerten Unterschiede ausgemacht werden.

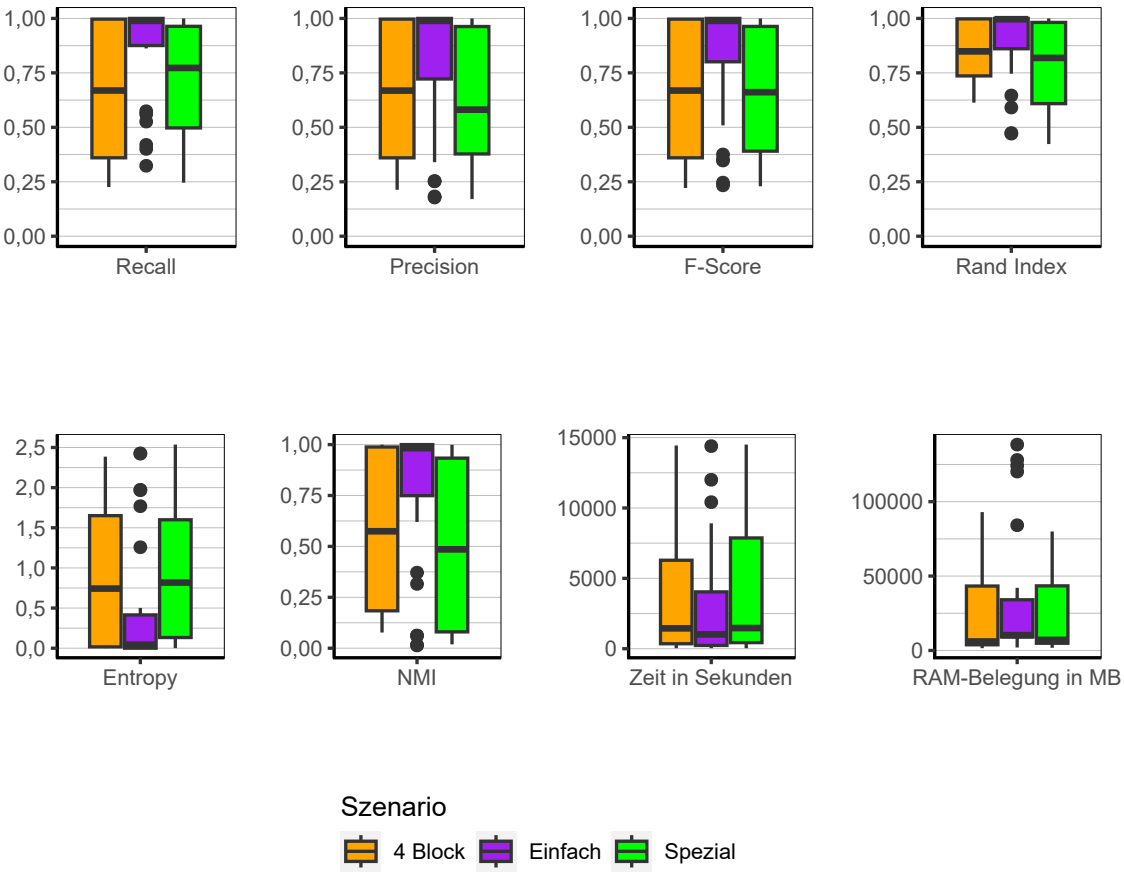


Abbildung 9.42.: SCALE: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Auch SCALE gelingt es mit Abstand am besten das Szenario *Einfach* zu clustern. Der Median des F-Score liegt dort bei 0,99 gegenüber 0,67 und 0,66 in den Szenarien *4 Block* respektive *Spezial*. Auch ist die Streuung wesentlich geringer mit einem Interquartilsabstand des F-Score von 0,2 gegenüber 0,64 und 0,57. Jedoch auch in den beiden Szenarien *4 Block* und *Spezial* sind die Ergebnisse zu großen Teilen sehr gut. Die 75 % Quantile liegen bei 1 bzw. 0,96. Bezüglich der Laufzeit und dem Speicherbedarf,

schneidet das Szenario *Einfach* ebenfalls etwas besser ab. Der Unterschied ist jedoch gering.

Fazit Wie bereits angesprochen ist der Algorithmus SCALE von allen betrachteten Algorithmen qualitativ der beste. Dies muss allerdings mit einer hohen Laufzeit und einem hohen bzw. in Teilen sehr hohen Speicherbedarf verrechnet werden. Es können nennenswerte Unterschiede in allen Faktorstufen des Designs festgestellt werden. So scheint der Algorithmus bessere Ergebnisse zu liefern, wenn mehr Variablen vorliegen. Genauso wirkt sich eine hohe Prävalenz positiv auf die Ergebnisse aus. Selbiges gilt für eine niedrigere Korrelation und einer niedrigen Anzahl von Clustern. Jedoch ist der Algorithmus auch bei den jeweils schlechteren Stufen der einzelnen Faktoren durchaus in der Lage gute Ergebnisse zu erzielen, wenn auch nicht immer. Demnach sollte diesem Algorithmus gegenüber den anderen betrachteten Algorithmen klar der Vorzug gegeben werden, solange Laufzeit und Speicherbedarf von untergeordneter Priorität sind.

9.3.7. Proximus

Im Folgenden werden die Ergebnisse für den Algorithmus Proximus analysiert.[4]

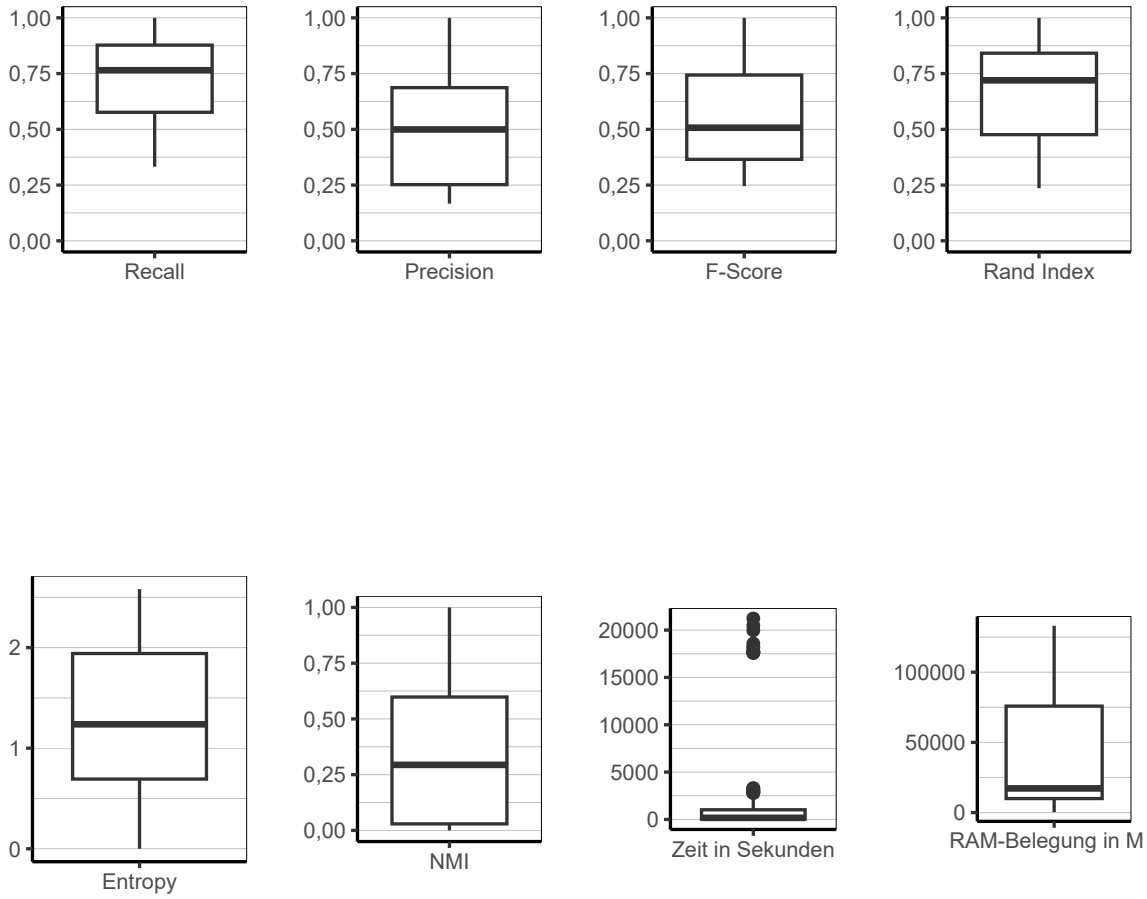


Abbildung 9.43.: Proximus: Evaluation aller Iterationen der Simulation insgesamt.

Gesamt Werden die Ergebnisse für Proximus über alle Faktorstufen insgesamt betrachtet (Abbildung 9.43), zeigt sich, dass Proximus akzeptable Ergebnisse hervorbringt, die etwas

[4]Aufgrund eines Segfault Errors bei den Konfigurationen mit 10 Millionen Beobachtungen, 1.000 Variablen, niedriger Prävalenz, Szenario *Einfach* und zwei, vier oder sechs Clustern konnten die Ergebnisse für diese Konfigurationen nicht berichtet werden.

über dem Niveau von BUBBLE, CLARA und CLARANS zu finden sind. Es lässt sich sehen, dass alle Qualitätsindikatoren etwas über denen der genannten Algorithmen liegen. Laufzeit und Speicherverbrauch sind größtenteils moderat, wobei es vor allem bei der Laufzeit einzelne starke Ausreißer nach oben gibt. Die Abbildung der Gini-Koeffizienten (Abbildung 9.44) zeigt, dass sich etwa die Hälfte der Ergebnisse relativ schief verteilen und in der oberen Hälfte der Abbildung zu finden sind. Die restlichen Ergebnisse befinden sich entsprechend in der unteren Hälfte mit einigen Beobachtungen auf oder Nahe der Null. Dabei kann ebenfalls beobachtet werden, dass sich die Gini-Koeffizienten nicht sehr stark nach Szenario unterscheiden. Die Mittelwerte sind entsprechend bei 0,41, 0,51 und 0,47 zu finden.

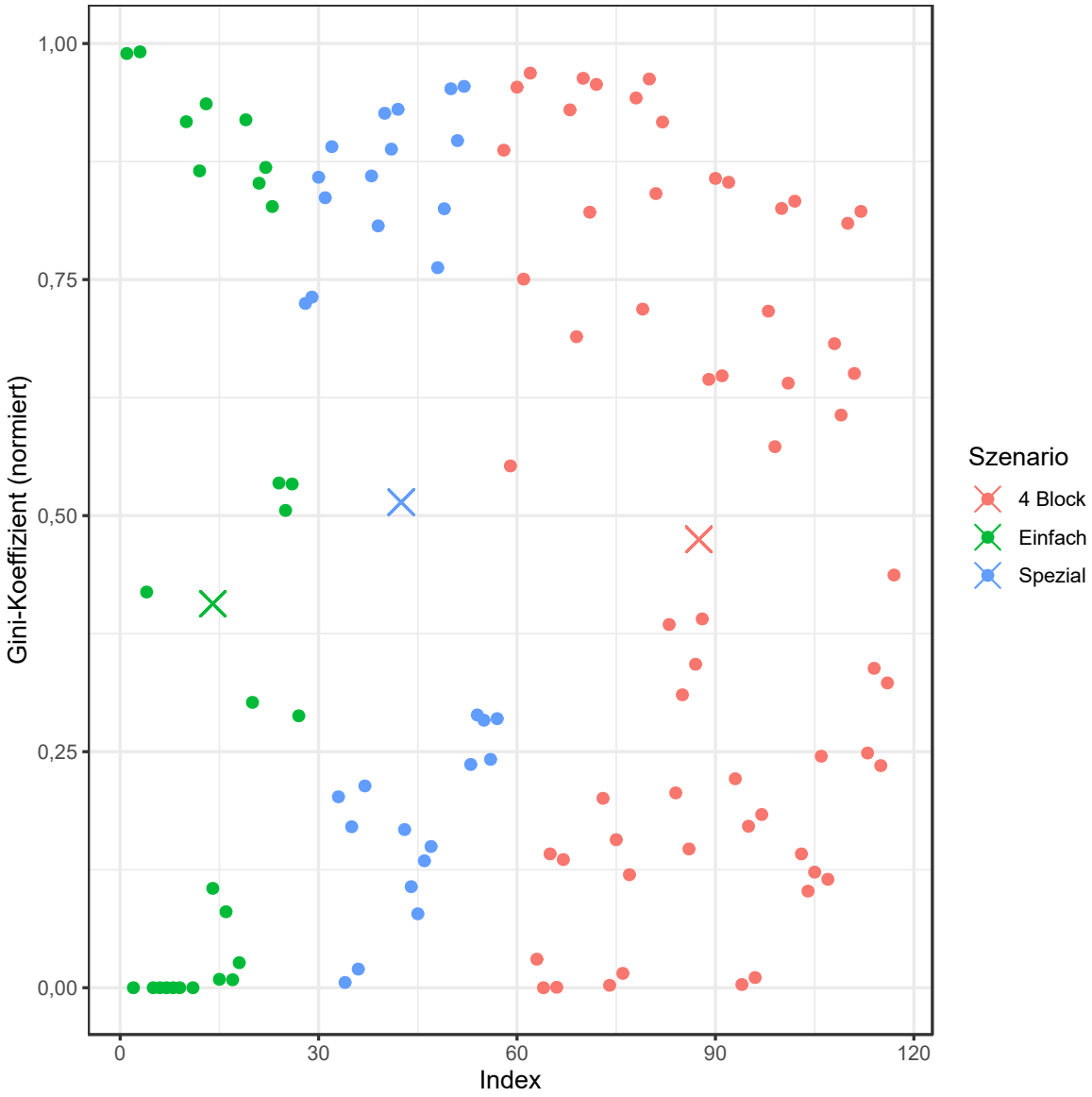


Abbildung 9.44.: Proximus: Gini-Koeffizient aller Iterationen der Simulation.

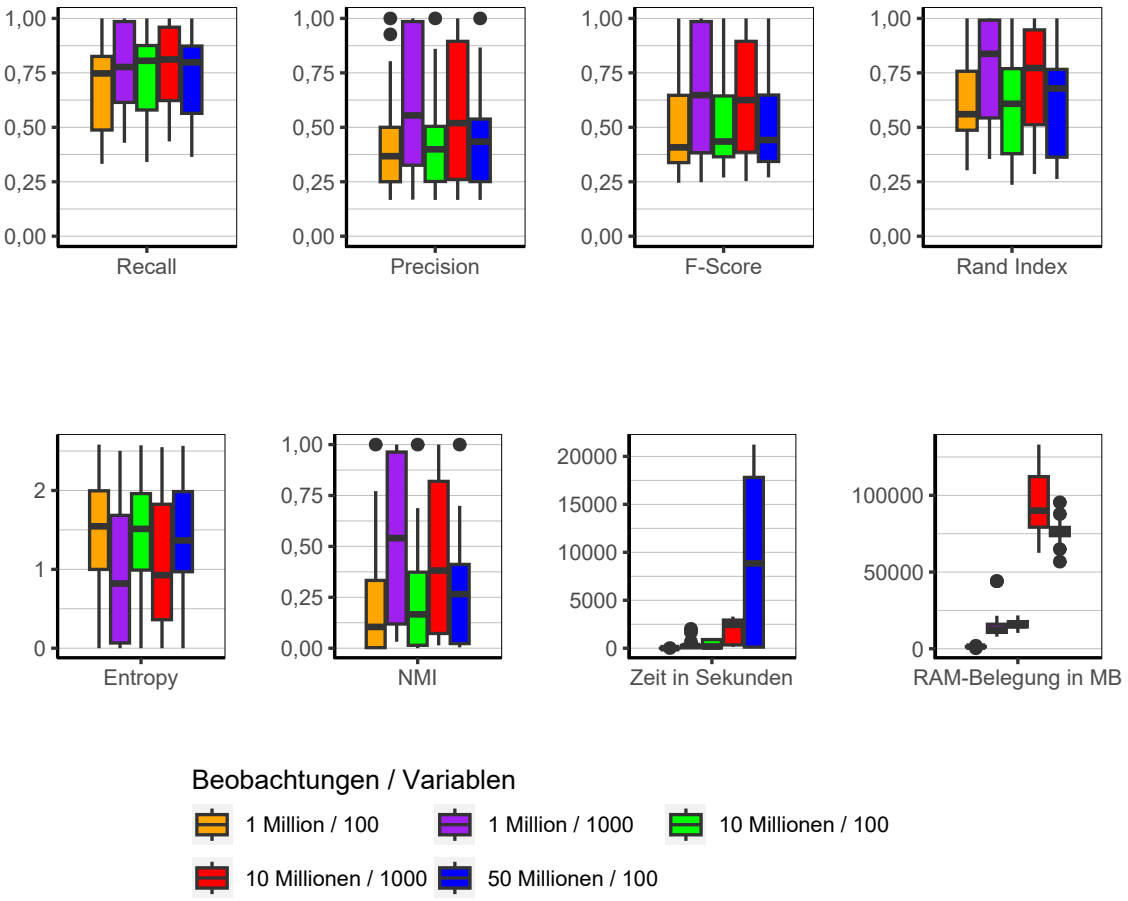


Abbildung 9.45.: Proximus: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Bezüglich der Dimensionalität (Abbildung 9.45) lässt sich eindeutig erkennen, dass der Algorithmus von einer höheren Anzahl an Variablen profitiert. Die Konfigurationen mit 1.000 Variablen schneiden weitaus besser ab als die mit lediglich 100 Variablen. Die Anzahl der Beobachtungen hingegen zeigt keinen Einfluss auf die Ergebnisse. Die Laufzeit allerdings steigt ab einem gewissen Punkt stark an. Die Konfiguration mit 50 Millionen Beobachtungen benötigt weitaus länger als die anderen Konfigurationen. Der

Speicherverbrauch wiederum liegt bei 10 Millionen Beobachtungen und 1.000 Variablen am höchsten.

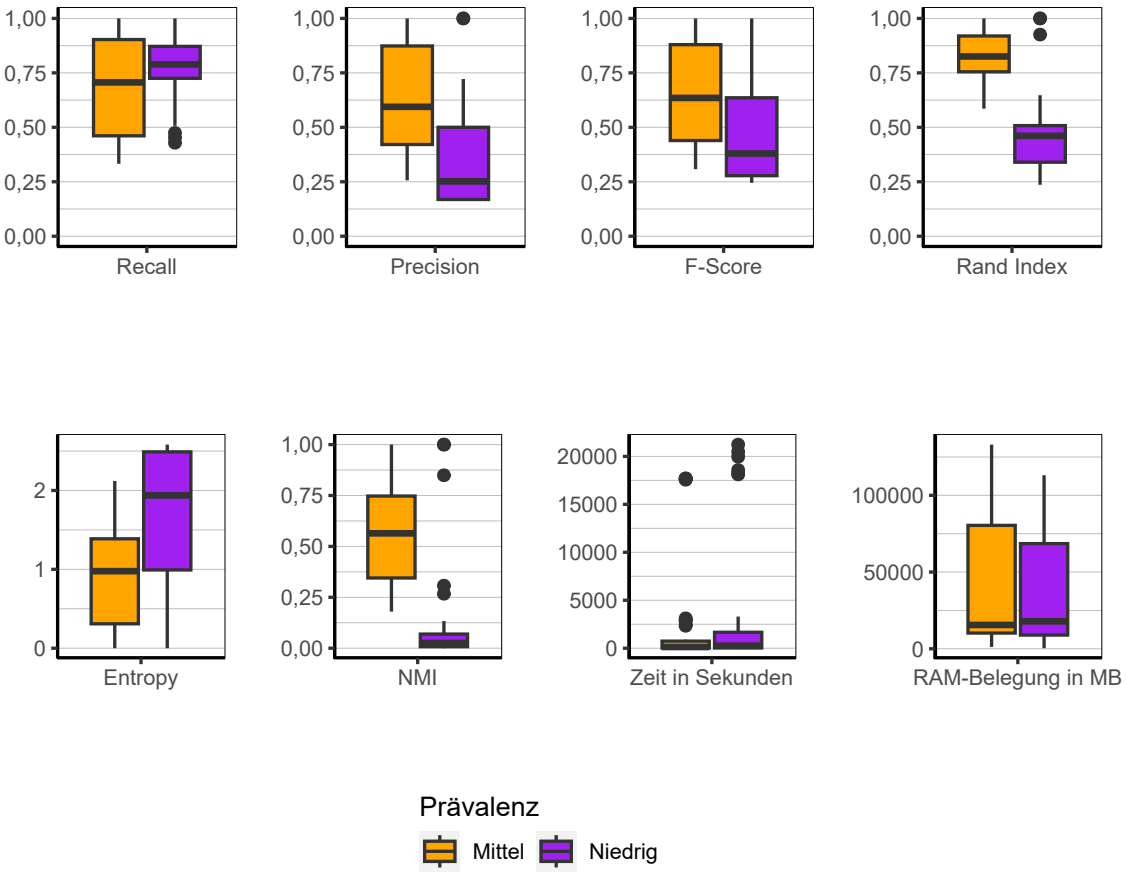


Abbildung 9.46.: Proximus: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Mit höherer Prävalenz verbessern sich die Ergebnisse von Proximus merklich (siehe Abbildung 9.46). Bei niedriger Prävalenz sind die Ergebnisse größtenteils als sehr schlecht zu bewerten. Lediglich einzelne Ausreißer weisen auf perfekte bzw. nahezu perfekte Ergebnisse hin. Abseits dieser Ergebnisse jedoch, benötigt der Algorithms eine höhere Prävalenz um bessere Clusterlösungen produzieren zu können. Die Laufzeit wird nicht von der Prävalenz beeinflusst. Der Speicherbedarf ist für höhere Prävalenz etwas höher, jedoch fällt dies nicht weiter ins Gewicht.

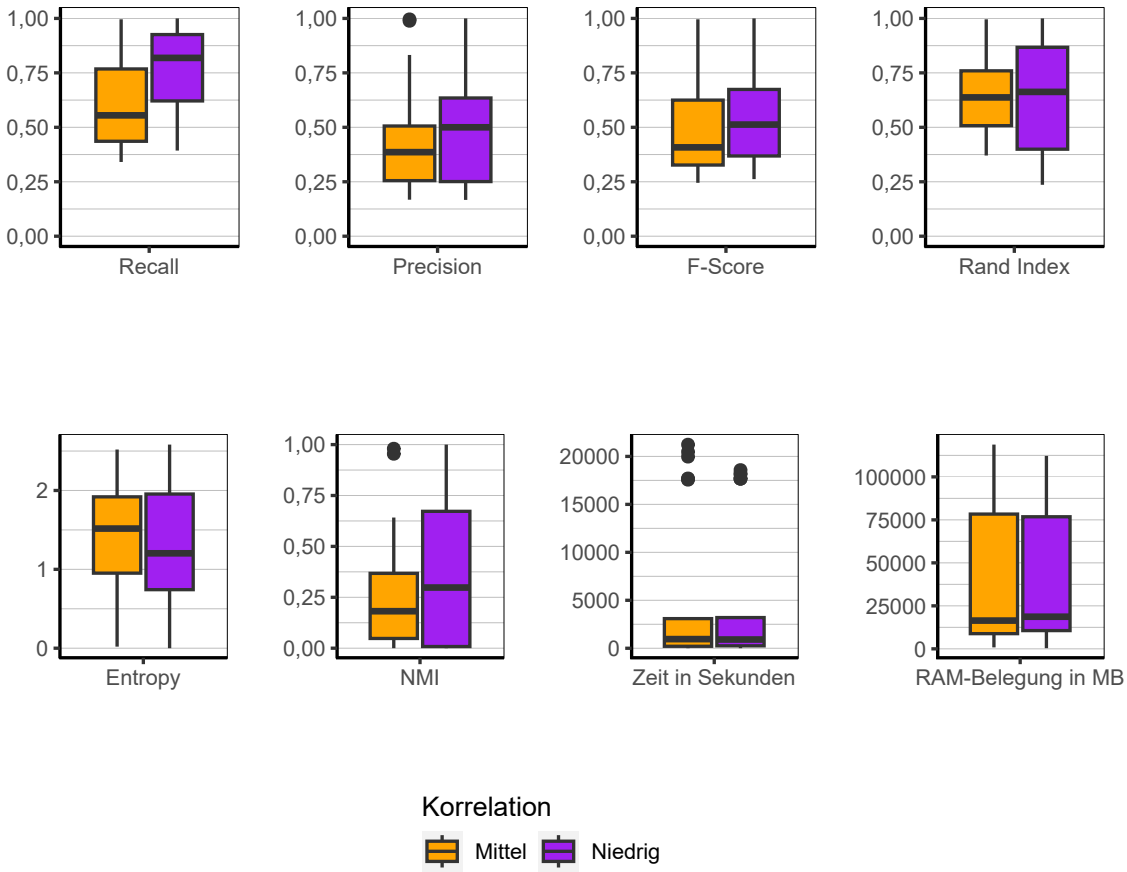


Abbildung 9.47.: Proximus: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Bei der Korrelation (Abbildung 9.47) lässt sich feststellen, dass mit niedrigerer Korrelation etwas bessere Ergebnisse erzielt werden. Der Einfluss ist jedoch nicht sehr groß. Der Median des F-Score beträgt bei niedriger Korrelation 0,51 und bei mittlerer Korrelation 0,41. Laufzeit und Speicherbedarf werden nicht von der Korrelation beeinflusst.

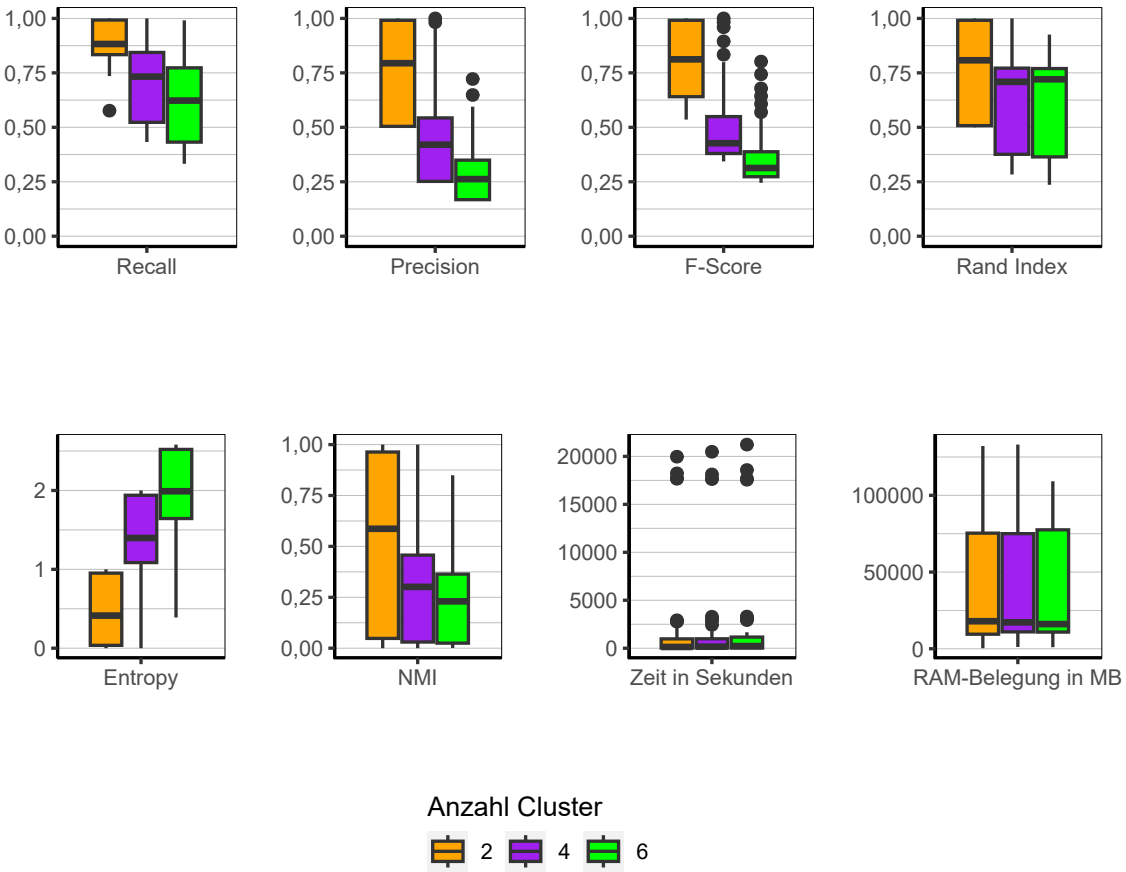


Abbildung 9.48.: Proximus: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Wird die Anzahl der Cluster betrachtet, ergibt sich ein bekanntes Bild. Je höher die Anzahl der Cluster desto schlechter werden die Ergebnisse. Dabei muss jedoch bedacht werden, dass bei Verteilungen in denen viele Beobachtungen einem Cluster zugewiesen wurden, die Ergebnisse für Recall, Precision und F-Score entsprechend niedriger werden. Es sei nochmals auf Abbildung 9.44 mit den Gini-Koeffizienten verwiesen, wonach auch bei Proximus dieses Phänomen zu beobachten ist. Wird jedoch der Rand Index betrachtet, fällt auf, dass zwischen 4 und 6 Clustern kein nennenswerter Unterschied

besteht. Die Box für 2 Cluster jedoch fast hoch bis zur 1 reicht, wohingegen die anderen beiden bei 0,75 enden. Das heißt, der tatsächliche Qualitätsunterschied zwischen 4 und 6 Clustern ist kaum gegeben.

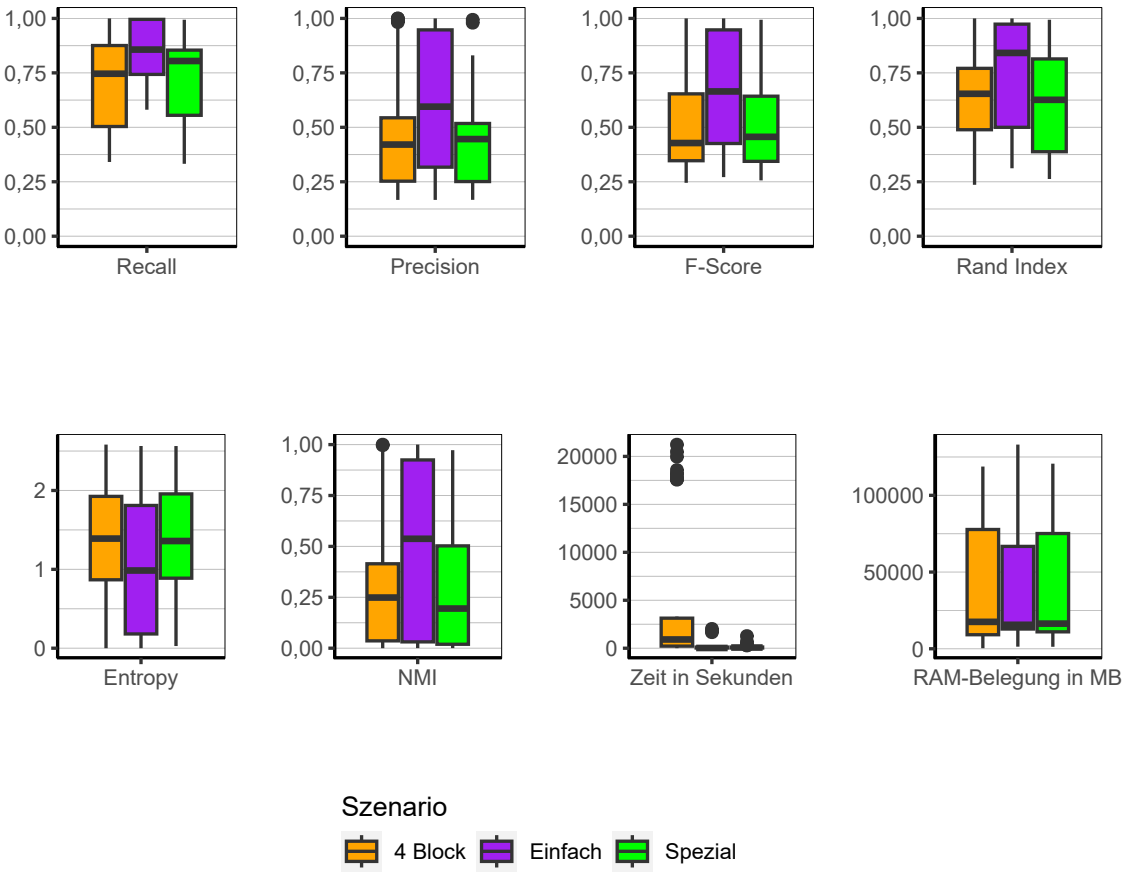


Abbildung 9.49.: Proximus: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Bezüglich der Szenarien zeigt sich wiederum ein bekanntes Muster (siehe Abbildung 9.49). Das Szenario *Einfach* schneidet am besten ab, wohingegen die anderen beiden ungefähr auf dem gleichen Niveau liegen. Es finden sich jedoch bei allen drei Szenarien Konfigurationen, bei denen perfekte Clusterings erreicht werden können. Auffällig ist zusätzlich, dass bei es bei der Laufzeit einige Ausreißer gibt, welche allesamt im Szenario *4 Block* verortet werden können. Der Speicherbedarf ist zwischen allen drei Szenarien in etwa vergleichbar.

Fazit Bei Proximus handelt es sich um einen Algorithmus, welchem vor allem bei hoher Prävalenz und hoher Anzahl von Variablen der Vorzug gegenüber den Algorithmen BUBBLE, CLARA und CLARANS gegeben werden sollte. Dies wird zwar in Teilen mit einer etwas geringeren Effizienz bezüglich Laufzeit und Speicherbedarf erkauft, dieser Preis ist jedoch in Anbetracht der höheren Qualität der Clusterings vertretbar. Ebenso produziert Proximus gleichmäßiger verteilte Clusterlösungen. Auch bei der Anzahl der Cluster zeigt sich zwar das gleiche Muster, jedoch auf einem etwas besseren Niveau.

9.3.8. Sorted Neighborhood

Die Ergebnisse des Algorithmus Sorted Neighborhood deuten auf zwei verschiedene Systematiken hin. Bei Betrachtung der Gini Koeffizienten in Abbildung 9.51 wird deutlich, dass der Algorithmus alle Beobachtungen in einen Cluster allokiert. Dies ist durchweg der Fall mit Ausnahme von 5 perfekten Clusterlösungen. Diese sind allesamt im einfachsten Fall in Szenario *Einfach* mit zwei Clustern und der höheren Prävalenz zu finden.

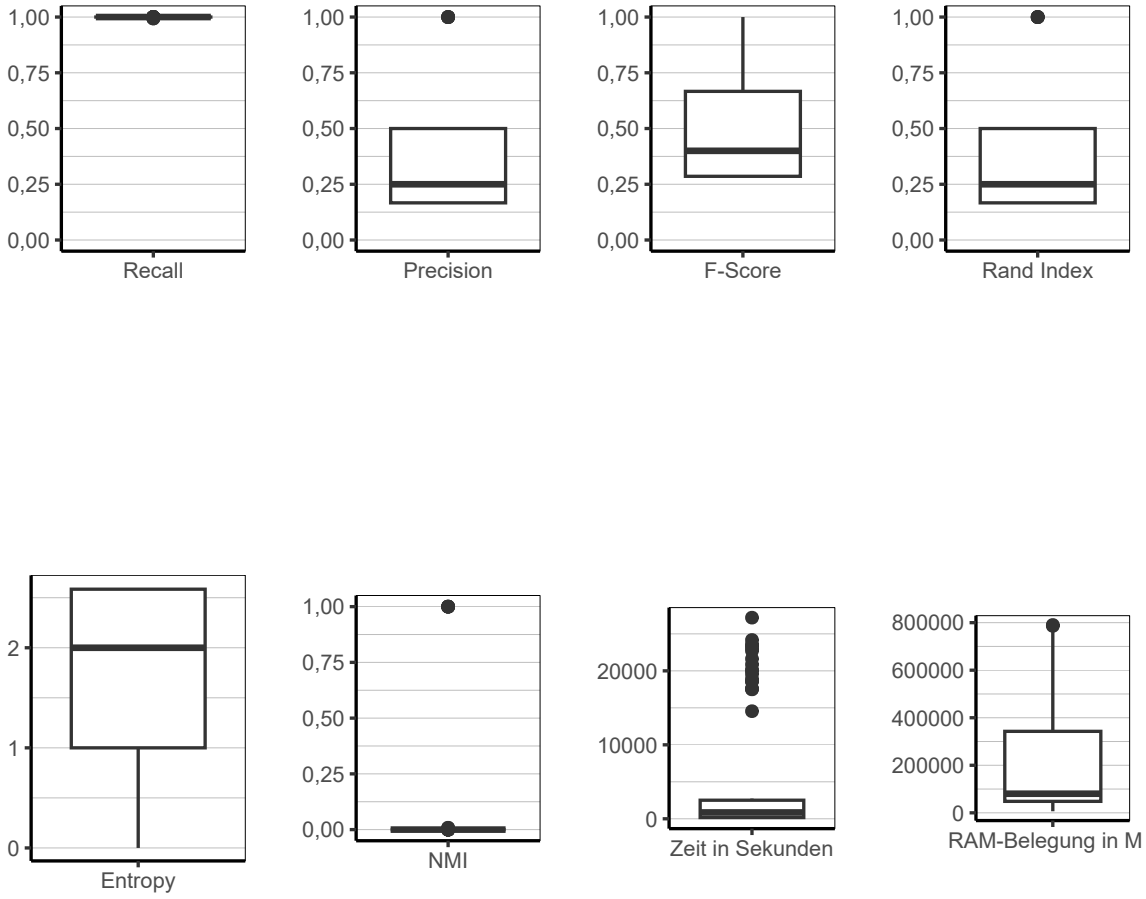


Abbildung 9.50.: Sorted Neighborhood: Evaluation aller Iterationen der Simulation insgesamt.

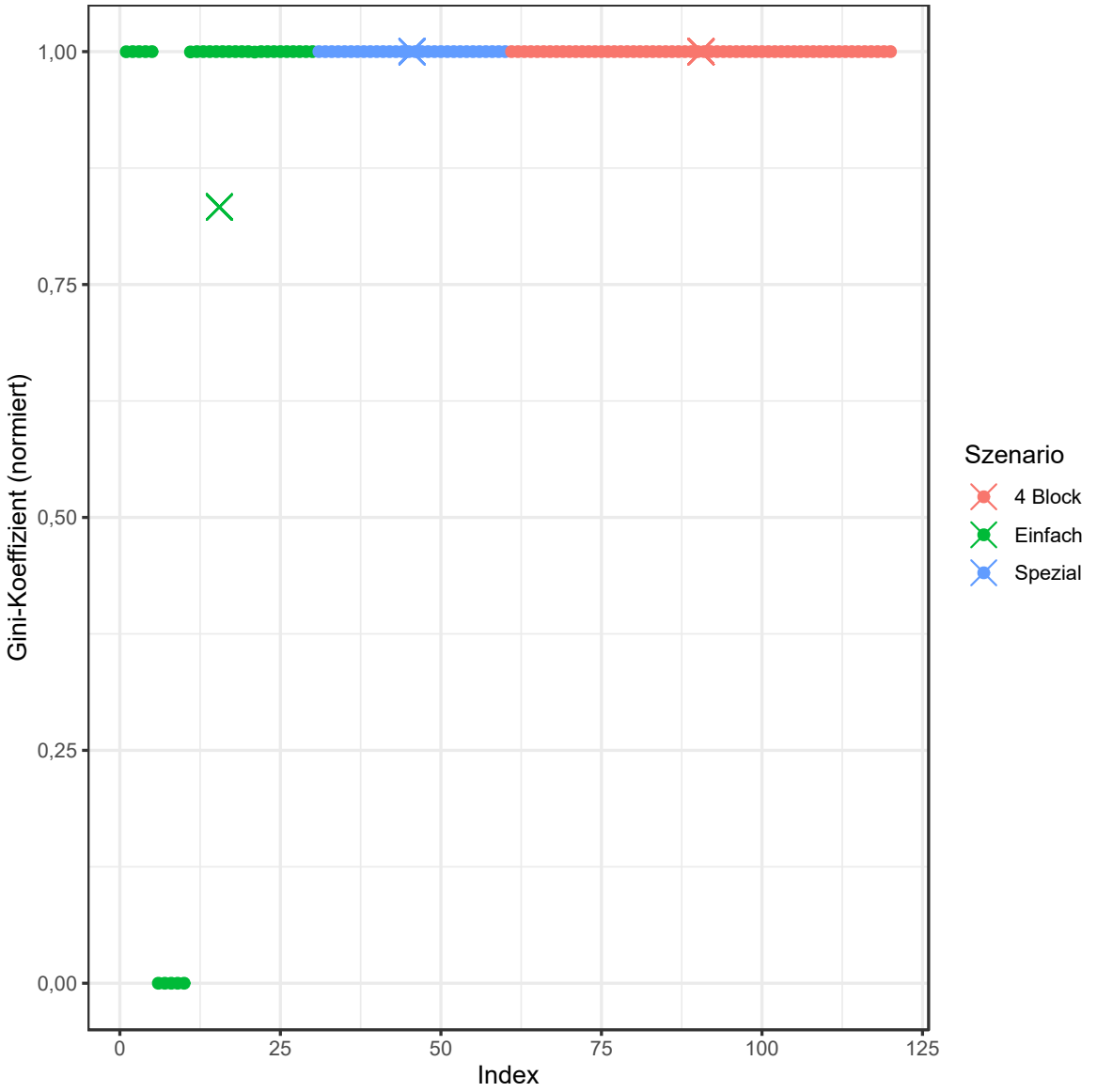


Abbildung 9.51.: Sorted Neighborhood: Gini-Koeffizient aller Iterationen der Simulation.

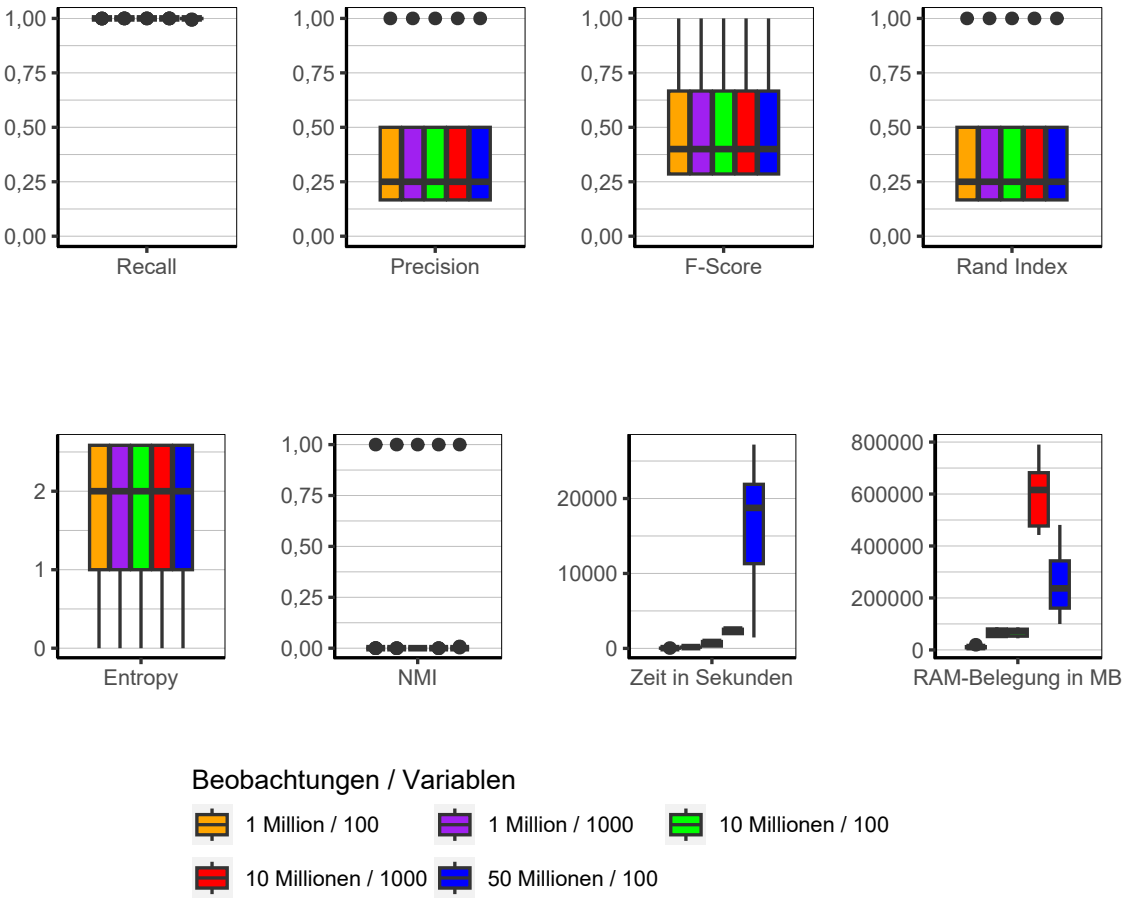


Abbildung 9.52.: Sorted Neighborhood: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität In Abbildung 9.52 zeigt sich, dass die Dimensionalität der Daten keinerlei Einfluss auf die Güte der Clusterlösungen hat. Bezüglich der Laufzeit lässt sich ein starker Anstieg bei 50 Millionen Beobachtungen feststellen. Der Speicherbedarf ist bei 10 Millionen Beobachtungen und 1.000 Variablen am höchsten.

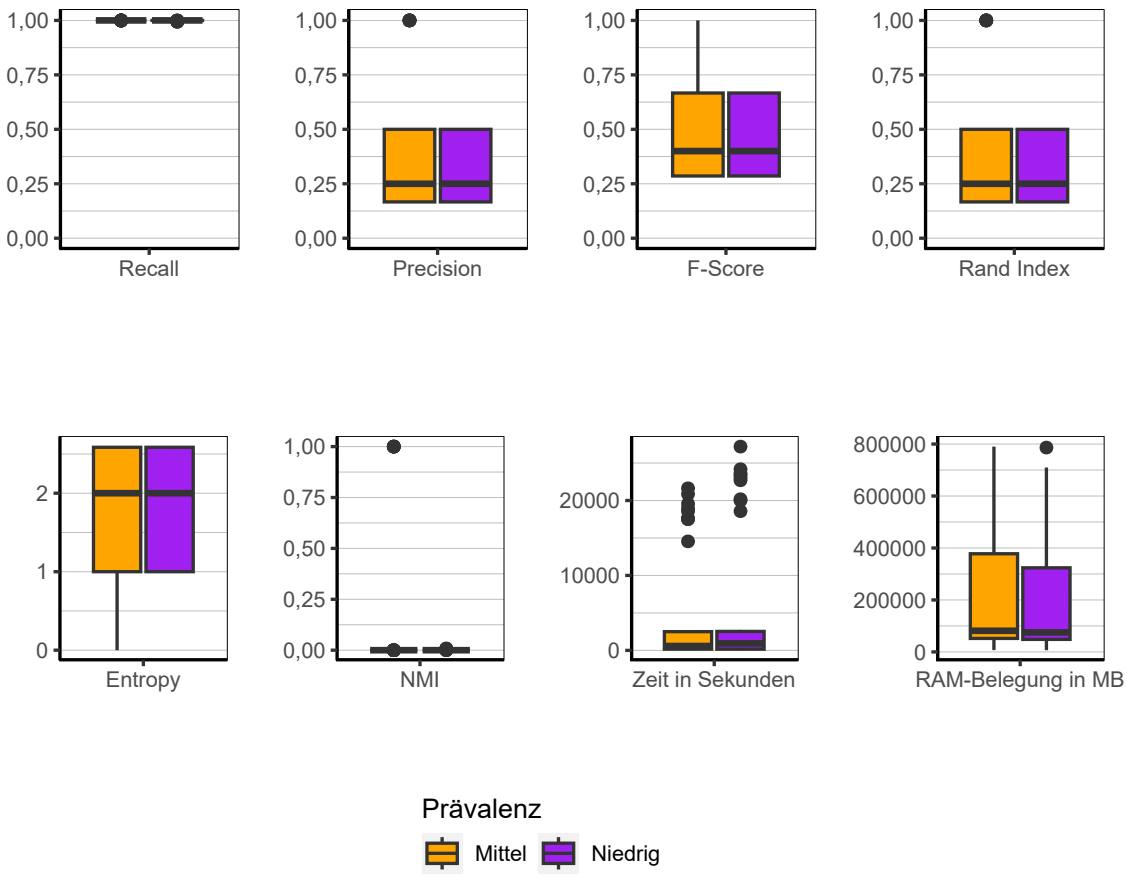


Abbildung 9.53.: Sorted Neighborhood: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Die Prävalenz hat insgesamt betrachtet nahezu keinen Einfluss (siehe Abbildung 9.53). Die einzige Ausnahme stellen die Ausreißer dar, in denen eine perfekte Clusterlösung gebildet werden konnte. Diese finden sich allesamt in den Konfigurationen mit der höheren Prävalenz.

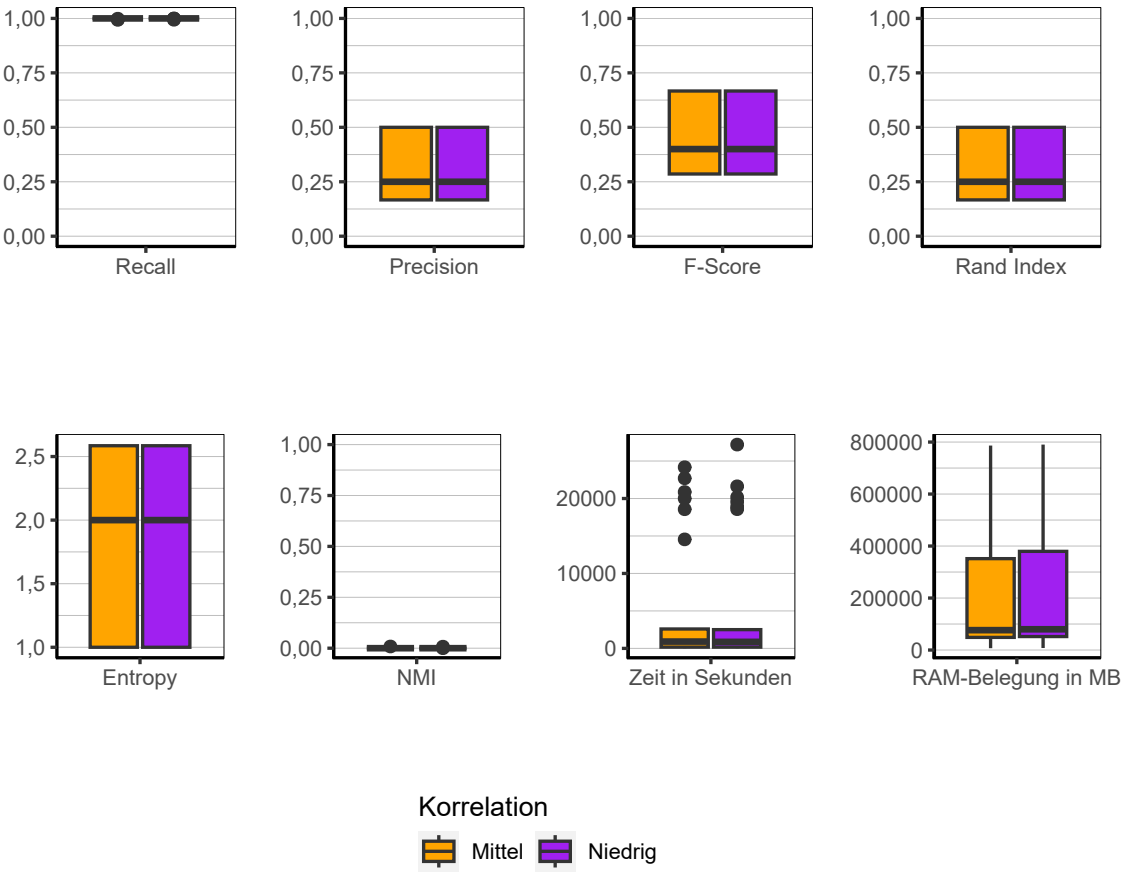


Abbildung 9.54.: Sorted Neighborhood: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Bei Analyse von Abbildung 9.54 wird deutlich, dass die Korrelationsstruktur überhaupt keinen Einfluss auf die Ergebnisse nimmt. Die Ergebnisse beider Konfigurationen sind identisch.

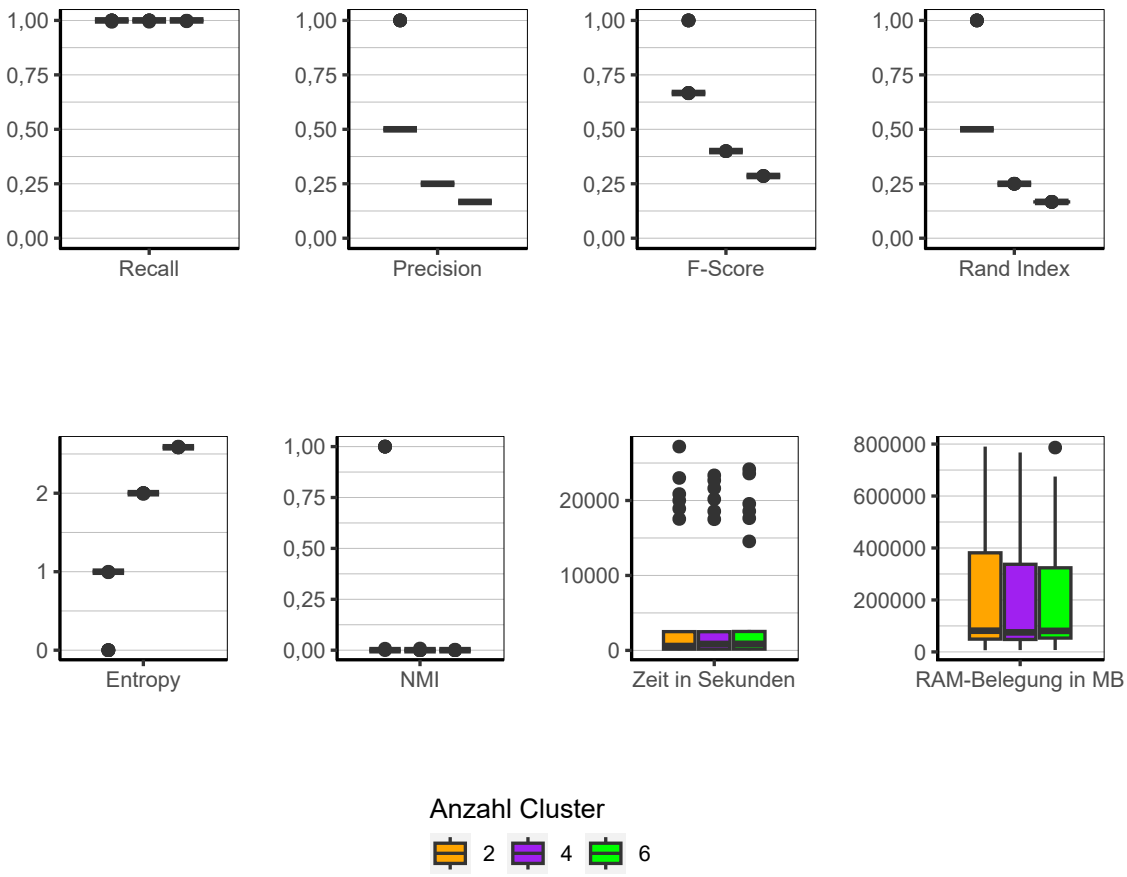


Abbildung 9.55.: Sorted Neighborhood: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Auch hier zeigt sich wieder das erwartbare Ergebnis, wenn der Algorithmus alle Beobachtungen in einen einzelnen Cluster allokiert. Ausnahme bilden wiederum die perfekten Clusterlösungen, die alle in den Konfigurationen mit zwei Clustern zu finden sind.

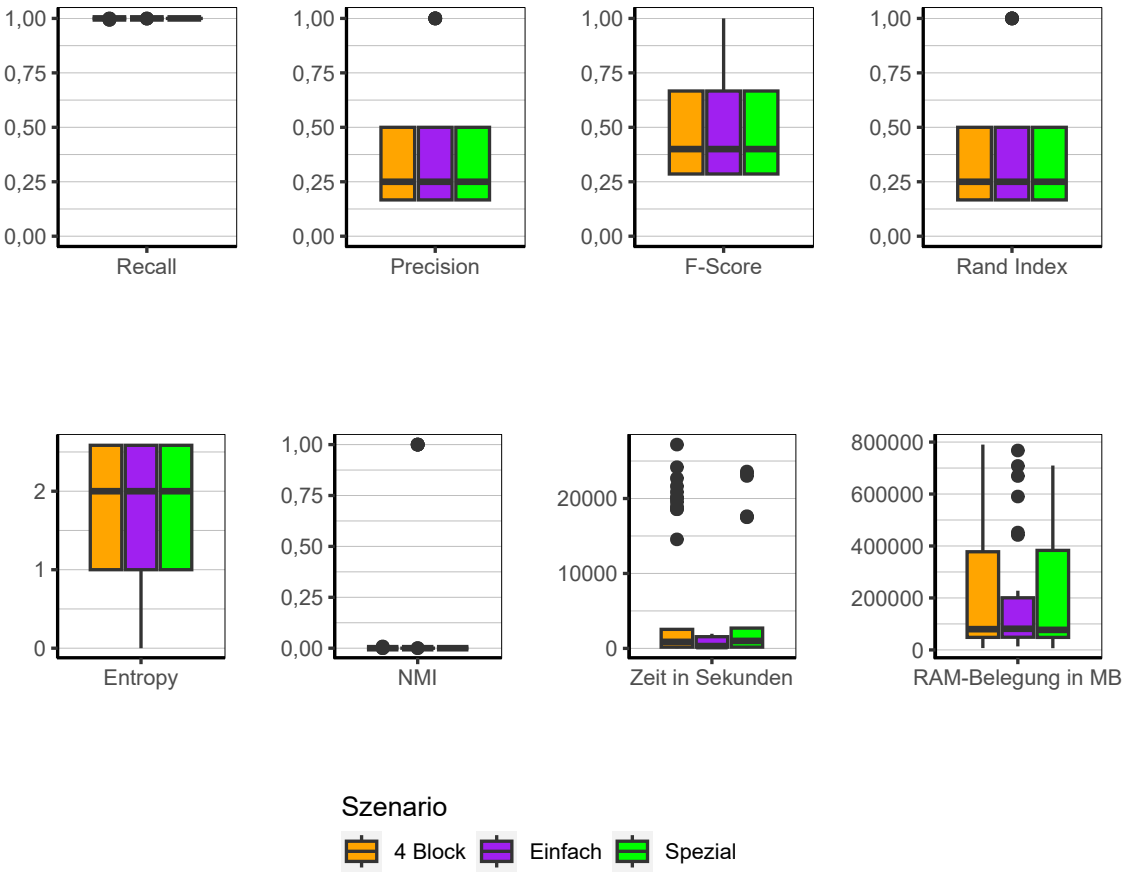


Abbildung 9.56.: Sorted Neighborhood: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Wie bereits eingangs angesprochen, sind sämtliche perfekte Clusterlösungen im Szenario *Einfach* zu finden (siehe Abbildung 9.56). Abseits dessen sind keine Unterschiede zwischen den einzelnen Szenarien zu erkennen.

Fazit Der Algorithmus weist sämtliche Beobachtungen einem einzelnen Cluster zu. Einzige Ausnahme stellen einzelne Konfigurationen in Szenario *Einfach* mit höherer Prävalenz und zwei Clustern dar. Damit wird deutlich, dass sich der Algorithmus nur im einfachsten Fall für das Clustern eignet und ansonsten davon Abstand genommen werden sollte.

9.3.9. Multibit Trees

Der Algorithmus Multibit Trees zeigt die bislang schlechtesten Ergebnisse der Simulation. Die Ergebnisse sind vergleichbar mit einer Clusterlösung, welche die Beobachtung zufällig auf die Cluster aufteilen würde. Es lassen sich keinerlei Unterschiede zwischen den einzelnen Faktorstufen erkennen. Die Abbildung der Gini-Koeffizienten (Abbildung 9.58) unterstreicht diesen Eindruck. Es ergibt sich größtenteils eine recht gleichverteilte Verteilung der Beobachtungen auf die einzelnen Cluster. Ähnlich wie es ein Zufallsgenerator tun würde, läge diesem eine Gleichverteilung zugrunde.

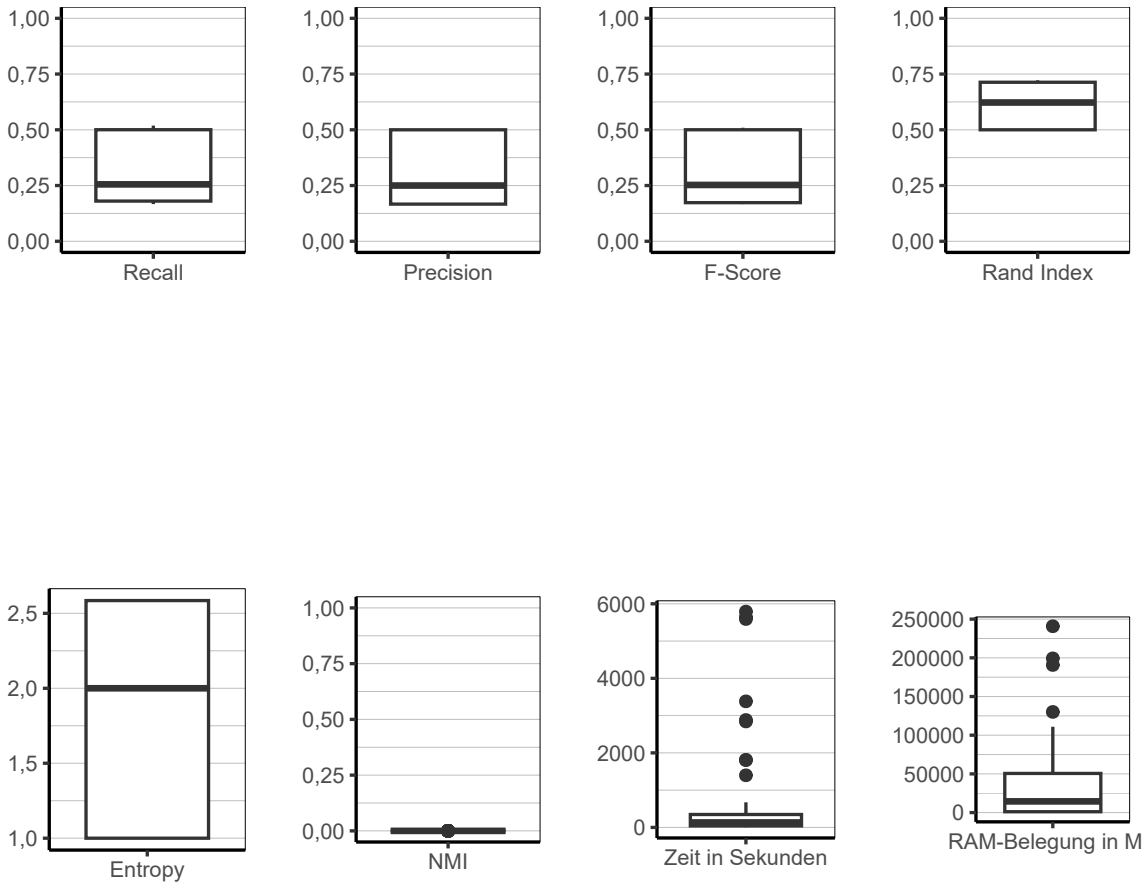


Abbildung 9.57.: Multibit Trees: Evaluation aller Iterationen der Simulation insgesamt.

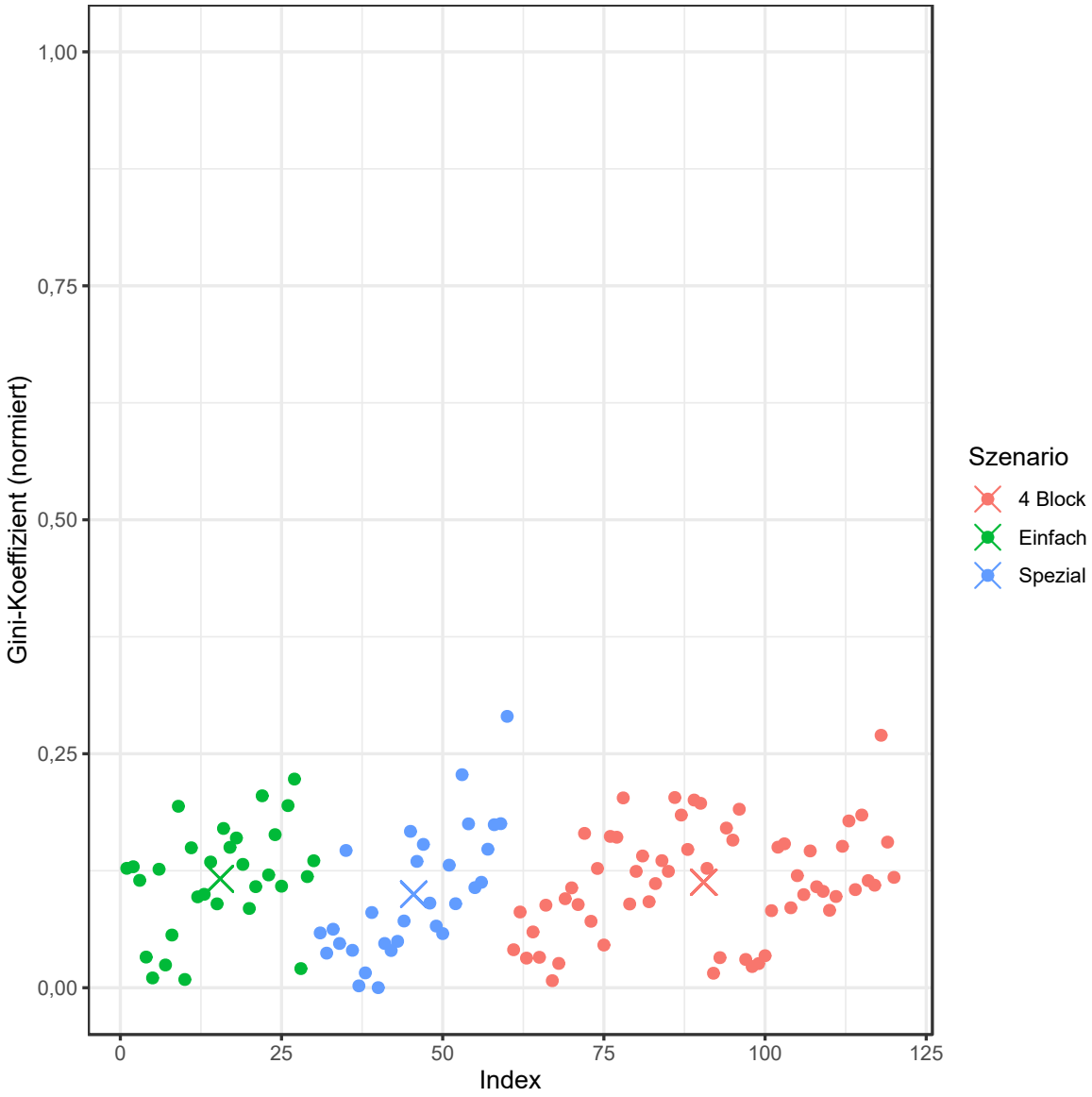


Abbildung 9.58.: Multibit Trees: Gini-Koeffizient aller Iterationen der Simulation.

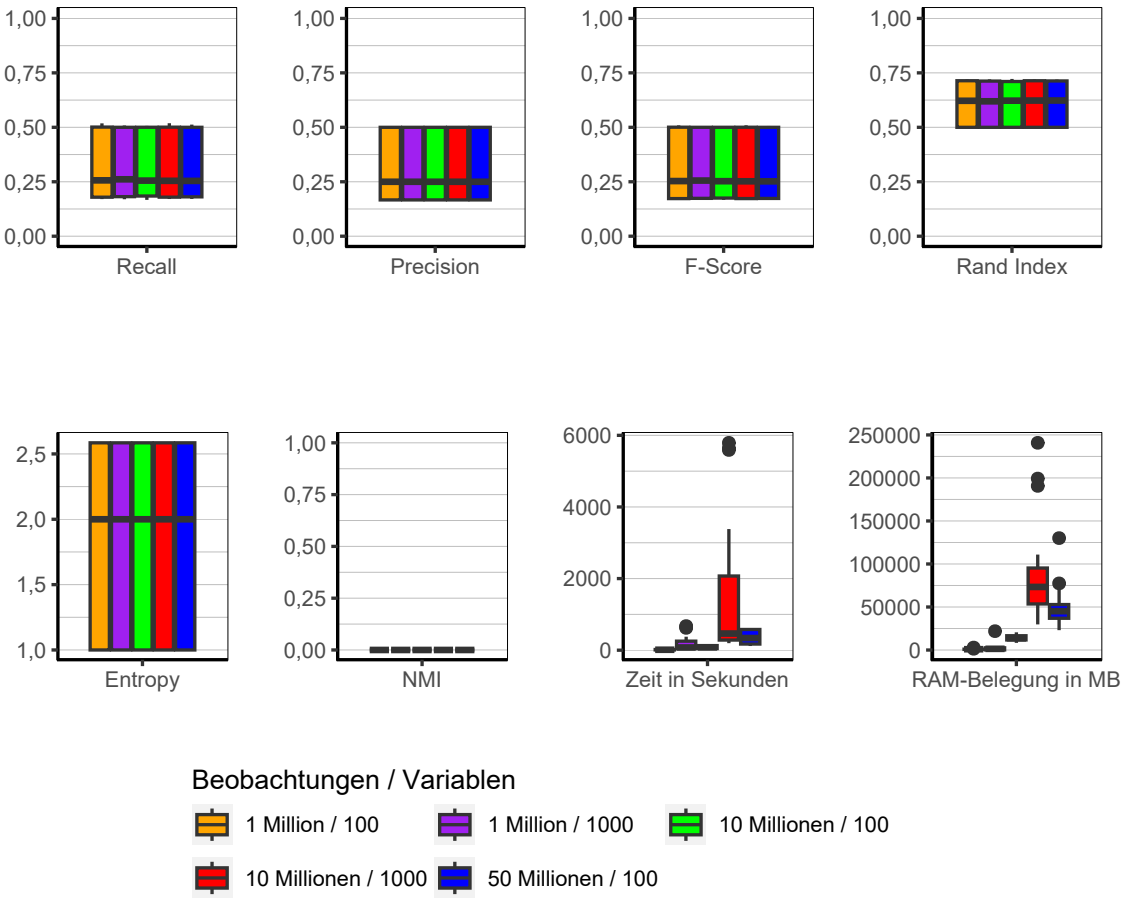


Abbildung 9.59.: Multibit Trees: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Bezüglich der Qualitätsindikatoren lässt sich in Abbildung 9.59 keinerlei Unterschied zwischen den einzelnen Konfigurationen erkennen. Die Dimensionalität der Daten hat keinen Einfluss auf die Qualität der Ergebnisse. Bezüglich der Laufzeit fällt auf, dass 10 Millionen Beobachtungen mit 1.000 Variablen im Vergleich wesentlich länger benötigen als 50 Millionen Beobachtungen mit 100 Variablen. Es scheinen daher vor allem die Anzahl der Variablen einen Einfluss auf die Laufzeit des Algorithmus zu haben. Ähnliches lässt sich beim Speicherbedarf feststellen.

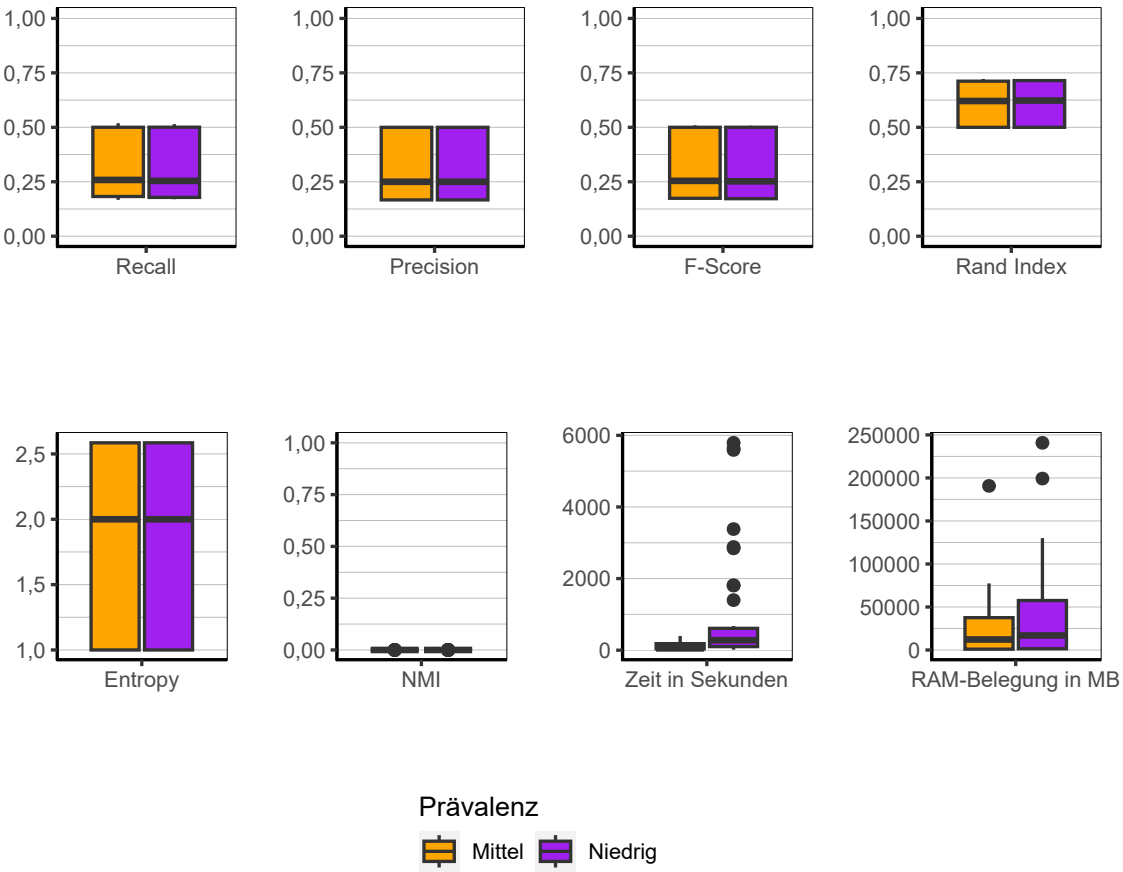


Abbildung 9.60.: Multibit Trees: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz In Abbildung 9.60 ist erkennbar, dass es keinen Unterschied zwischen den Konfigurationen bezüglich der Prävalenz gibt. Unabhängig davon welche Konfiguration gewählt wird, sind die Ergebnisse auf dem gleichen Niveau.

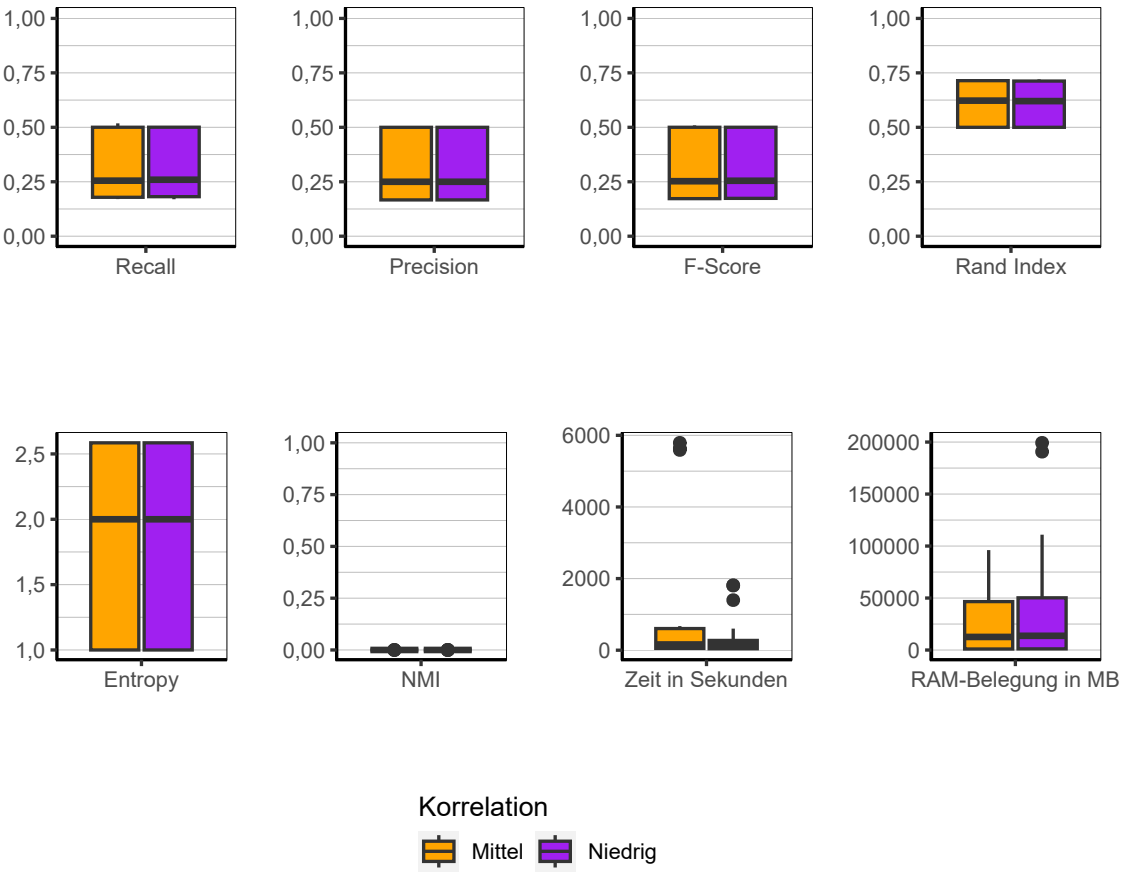


Abbildung 9.61.: Multibit Trees: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Auch bei der Korrelationsstruktur ist kein Einfluss zu erkennen (siehe Abbildung 9.61). Die Ergebnisse bei beiden Konfigurationen sind nahezu identisch.

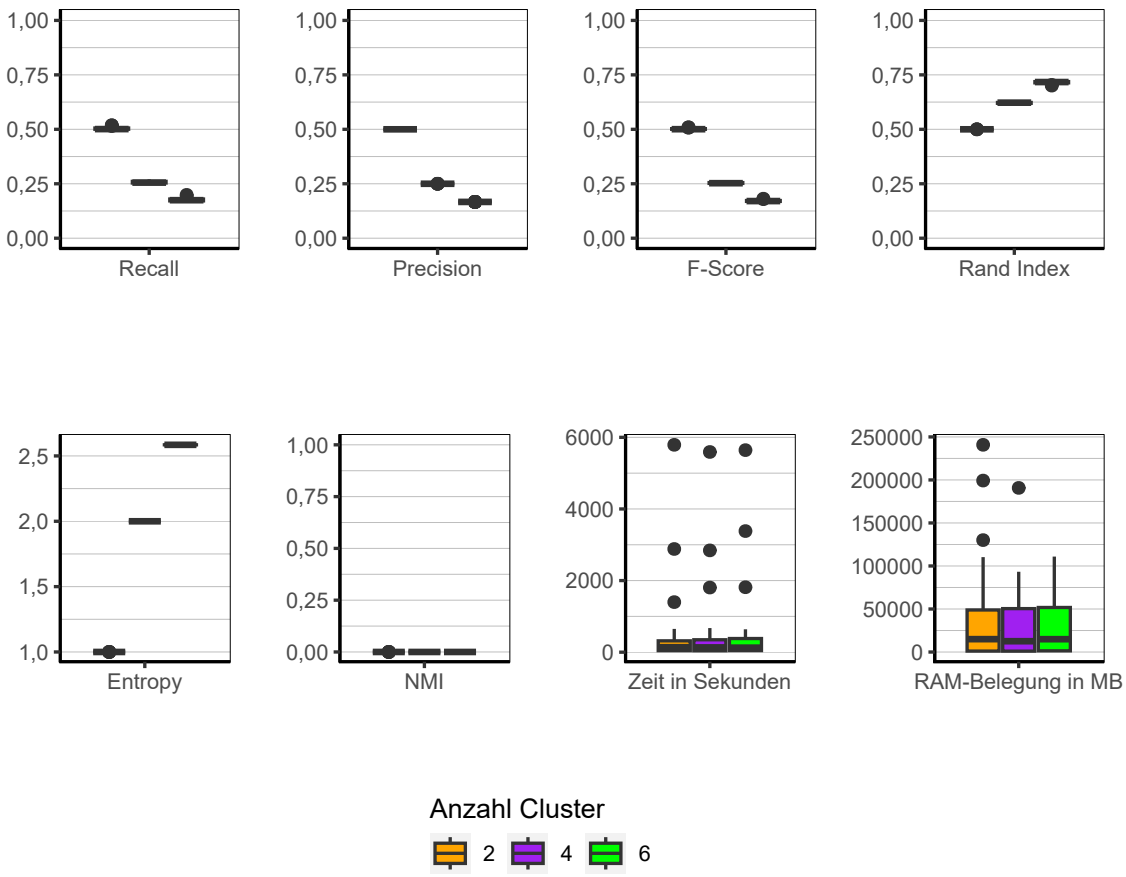


Abbildung 9.62.: Multibit Trees: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Auch hier (siehe Abbildung 9.62) ist wieder das bekannte Bild zu sehen, dass die Anzahl der Cluster keinen Einfluss haben.

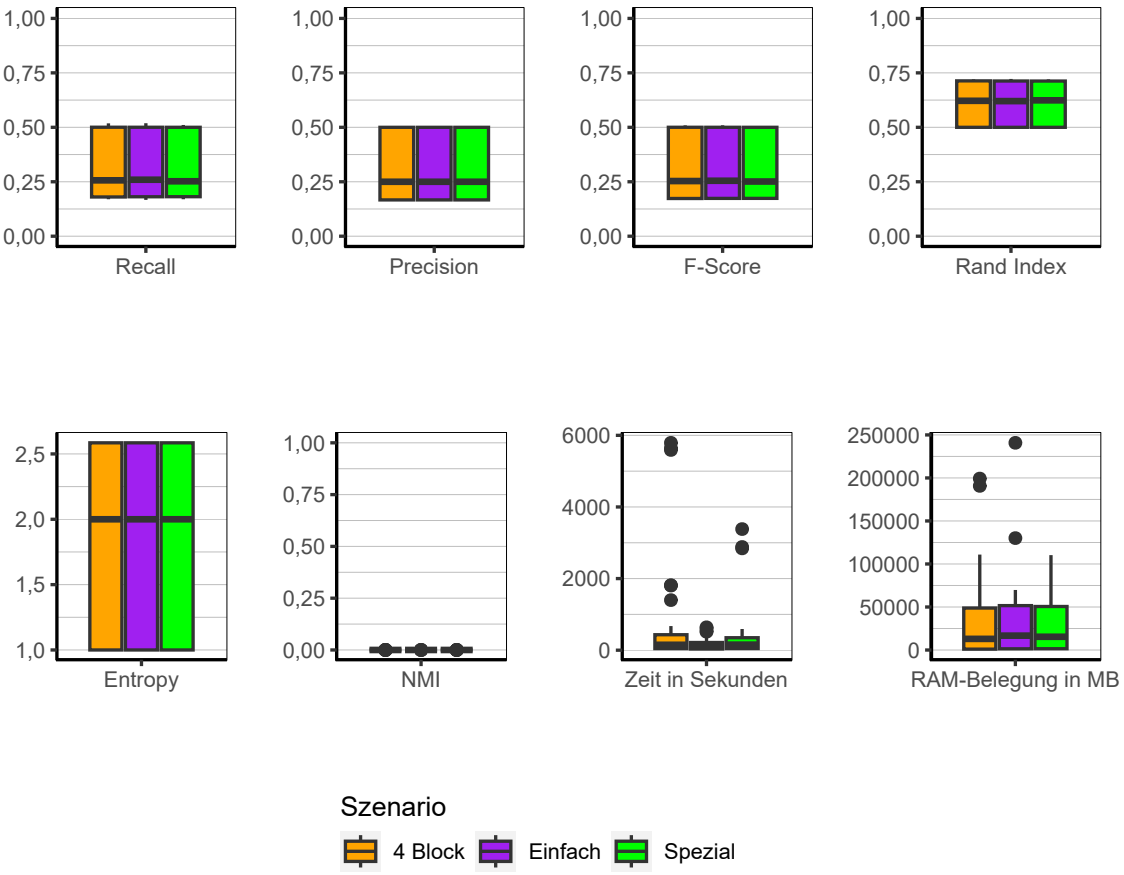


Abbildung 9.63.: Multibit Trees: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Gleiche Ergebnisse zwischen den einzelnen Szenarien sind auch in Abbildung 9.63 zu sehen. Das bedeutet, dass auch bei relativ eindeutig getrennten Clustern der Algorithmus nicht in der Lage war gute Clusterlösungen zu produzieren. Die Laufzeit und der Speicherbedarf werden ebenfalls nicht vom Szenario beeinflusst.

Fazit Wie eingangs erwähnt, ist der Algorithmus nicht in der Lage gute Clusterlösungen hervorzubringen. Die Ergebnisse sind vergleichbar mit denen eines Zufallsgenerators. Die Laufzeit des Algorithmus ist zwar niedrig, der Speicherbedarf jedoch relativ hoch.

9.3.10. MONA

Folgend werden die Ergebnisse des Algorithmus MONA dargestellt. Der Algorithmus wurde nicht auf die Konfigurationen mit 50 Millionen Beobachtungen angewendet, da sich die Laufzeit und der Speicherbedarf als zu exzessiv herausstellten und nicht mehr verhältnismäßig sind.

Gesamt Werden die Ergebnisse über alle Faktoren des Simulationsaufbaus zusammen betrachtet (Abbildung 9.64), stellt sich ein ernüchterndes Bild ein. Zwar fallen die Ergebnisse besser aus als bei den beiden vorangegangenen Algorithmen, dennoch sind die Ergebnisse nicht zufriedenstellend. Die Qualitätsindikatoren sind durchweg niedrig. Die Gini-Koeffizienten (Abbildung 9.65) zeigen eine Tendenz des Algorithmus schiefe Verteilungen der Beobachtungen auf die Cluster zu produzieren. Lediglich im Szenario *Einfach* finden sich einige Koeffizienten unterhalb der 0,25 Marke. Entsprechend sind die Mittelwerte zwischen den Szenarien 0,52 (*Einfach*), 0,75 (*Spezial*) und 0,67 (*4 Block*). Dies kann als erster Indikator dafür gesehen werden, dass auch bei diesem Algorithmus lediglich bei sehr klarer Clusterstruktur zumindest akzeptable Ergebnisse zu erwarten sind.

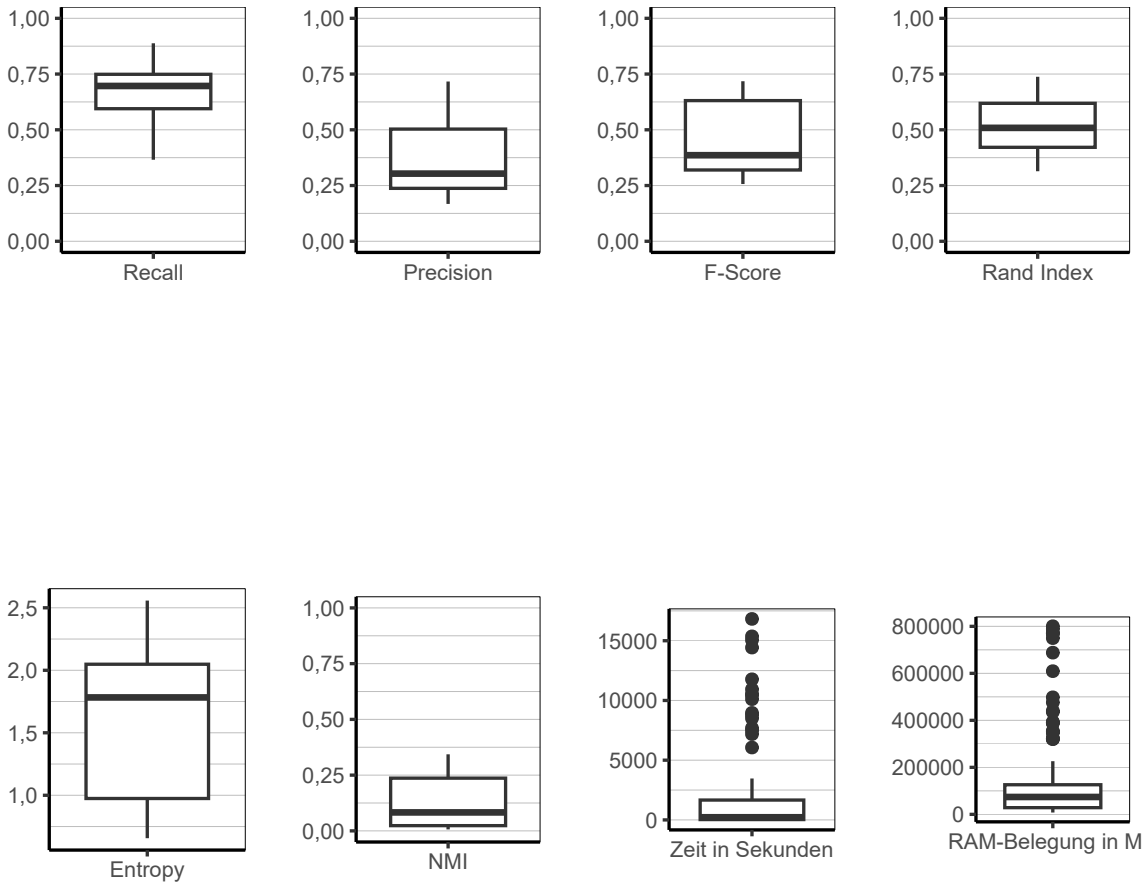


Abbildung 9.64.: MONA: Evaluation aller Iterationen der Simulation insgesamt.

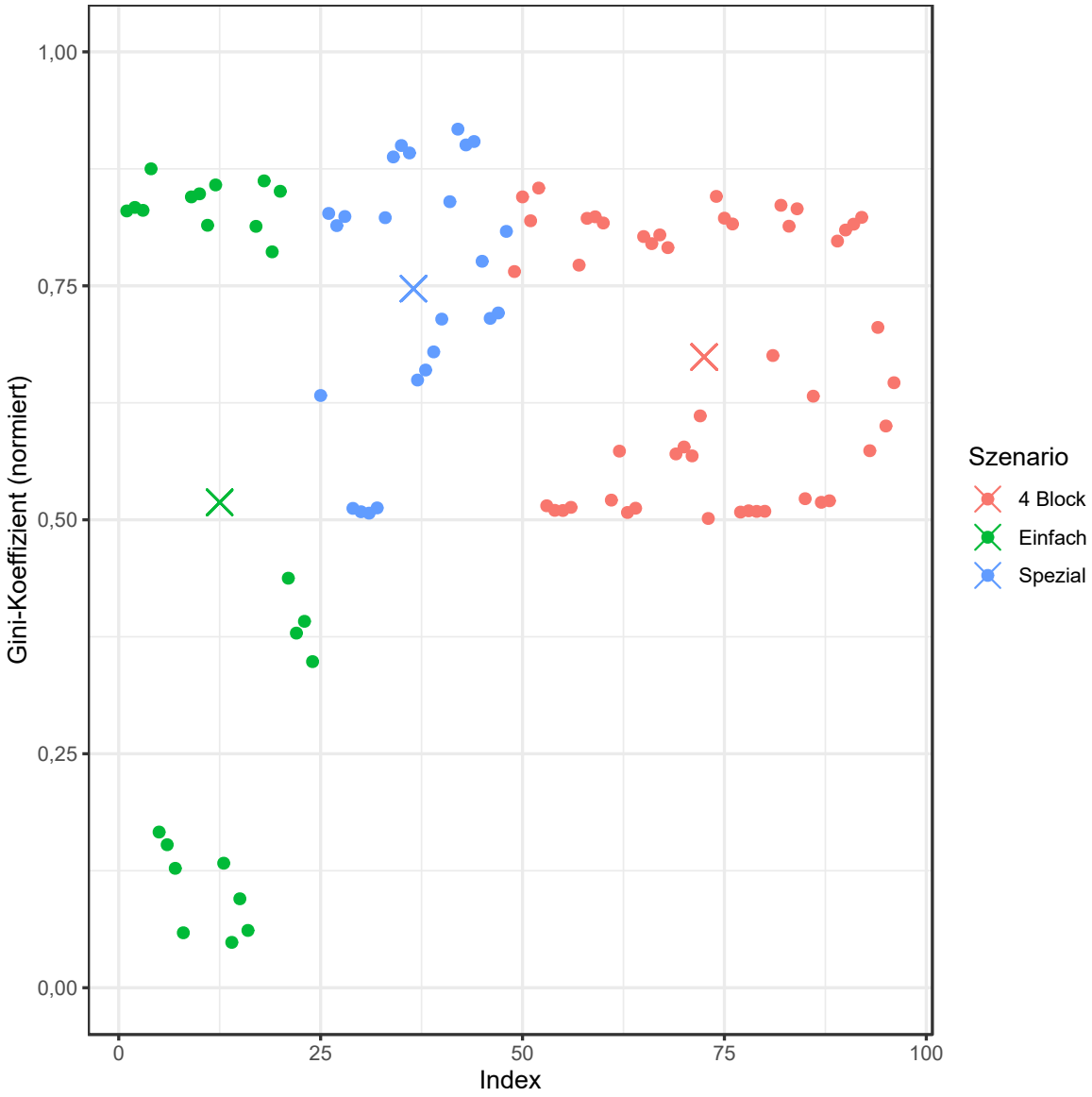


Abbildung 9.65.: MONA: Gini-Koeffizient aller Iterationen der Simulation.

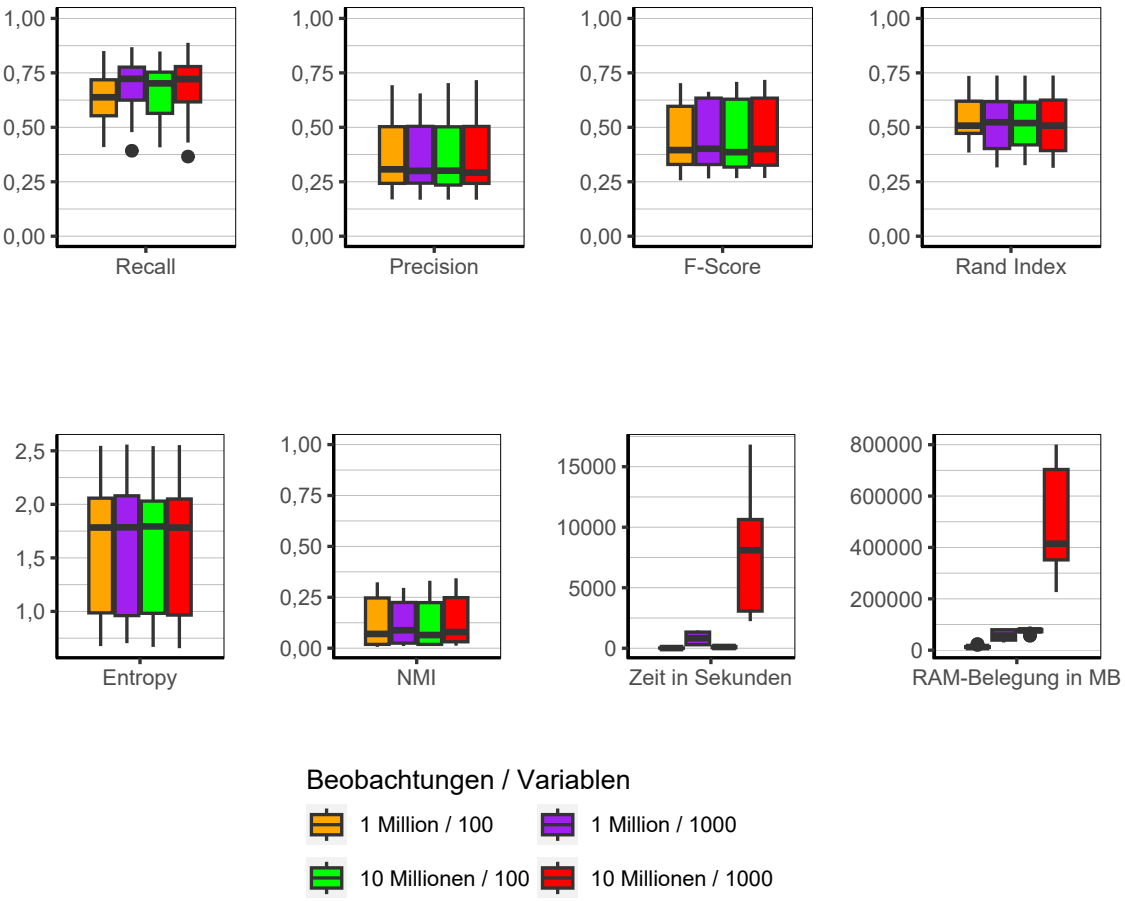


Abbildung 9.66.: MONA: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität In Abbildung 9.66 lässt sich eindeutig erkennen, dass die Dimensionaliät keinen Einfluss auf die Qualität der Clusterlösungen hat. Außerdem wird deutlich, dass mit steigender Anzahl von Variablen sowohl die Laufzeit als auch der Speicherbedarf stark ansteigt, weshalb der Algorithmus nicht für große Datensätze mit vielen Variablen geeignet ist.

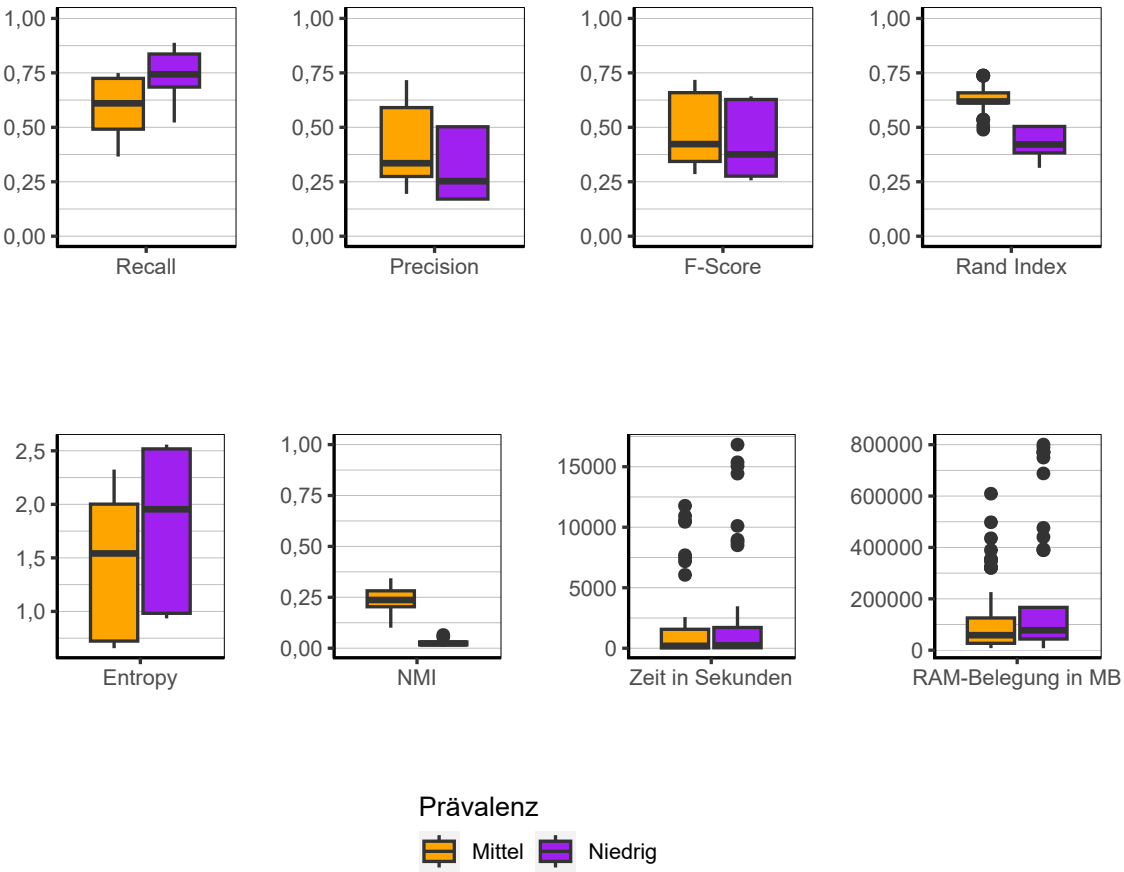


Abbildung 9.67.: MONA: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Wie schon bei vorangegangenen Algorithmen scheint auch MONA bei höherer Prävalenz bessere Ergebnisse zu liefern als bei niedrigerer Prävalenz (siehe Abbildung 9.67). Interessant ist hier jedoch, dass die Ergebnisse für Recall bei niedrigerer Prävalenz höher sind, was sich aber nicht auf die Ergebnisse insgesamt auswirkt. So ist die Precision bei höherer Prävalenz höher und führt dann dazu, dass die Ergebnisse zusammengenommen, bei höherer Prävalenz besser ausfallen.

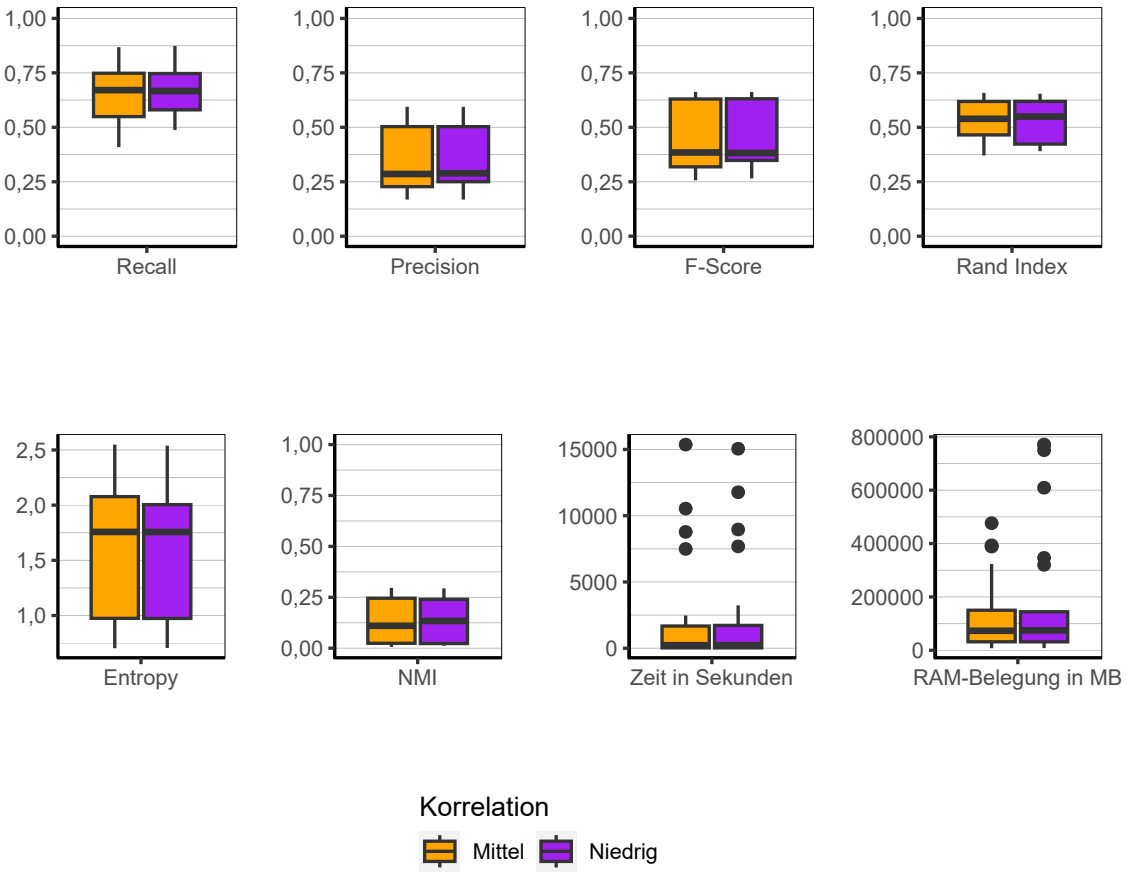


Abbildung 9.68.: MONA: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Bei der Korrelation lassen sich keinerlei nennenswerte Unterschiede zwischen den beiden Stufen erkennen (siehe Abbildung 9.68).

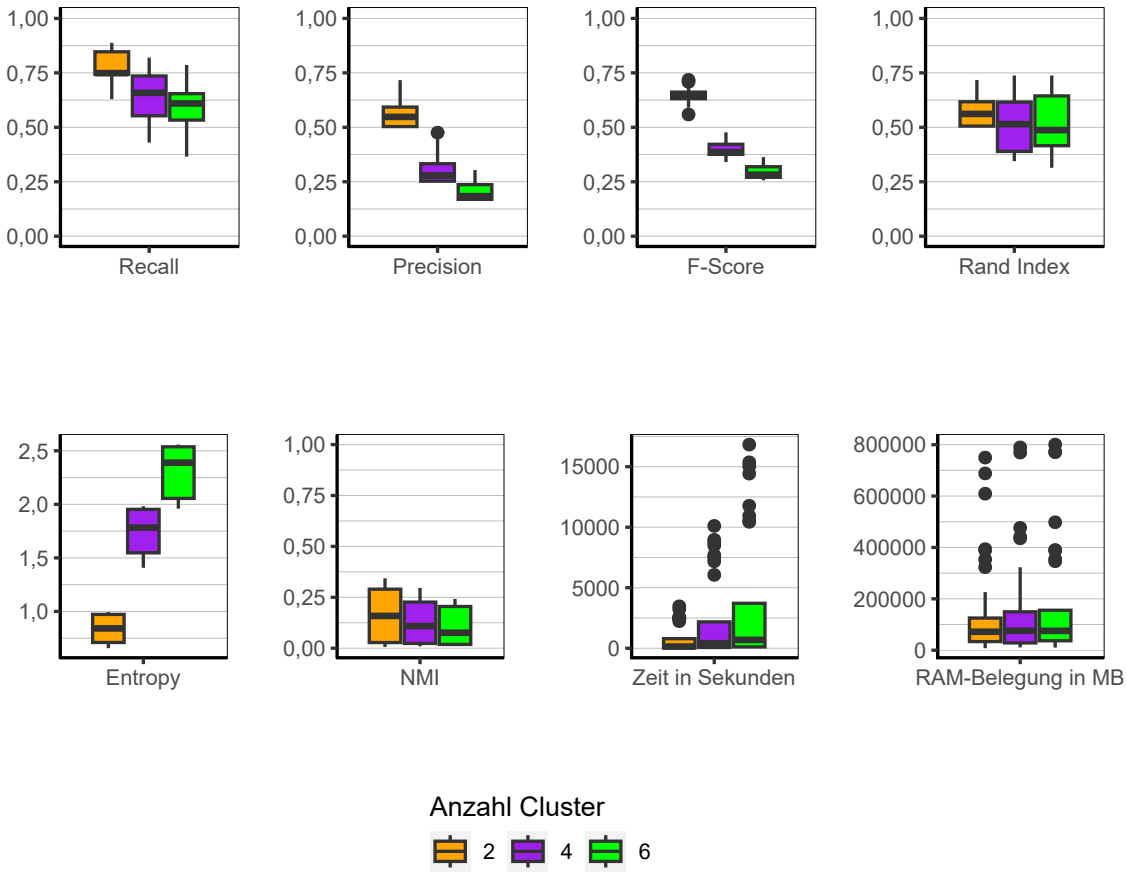


Abbildung 9.69.: MONA: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Bei der Anzahl der Cluster zeigt sich wiederum ein bekanntes Muster. So ist die Qualität der Clusterlösungen geringer, sobald die Anzahl der Cluster steigt. Interessant ist zu beobachten, dass mit dem Anstieg der Anzahl der Cluster auch die Laufzeit des Algorithmus steigt. Zumindest gilt dies für einzelne Ausreißer. Die Mediane der unterschiedlichen Stufen liegen wiederum nah beieinander. Der Median für 2 Cluster liegt demnach bei 141,89 Sekunden, der für 4 Cluster bei 414,3 Sekunden und letztlich der Median für 6 Cluster bei 694,87 Sekunden.

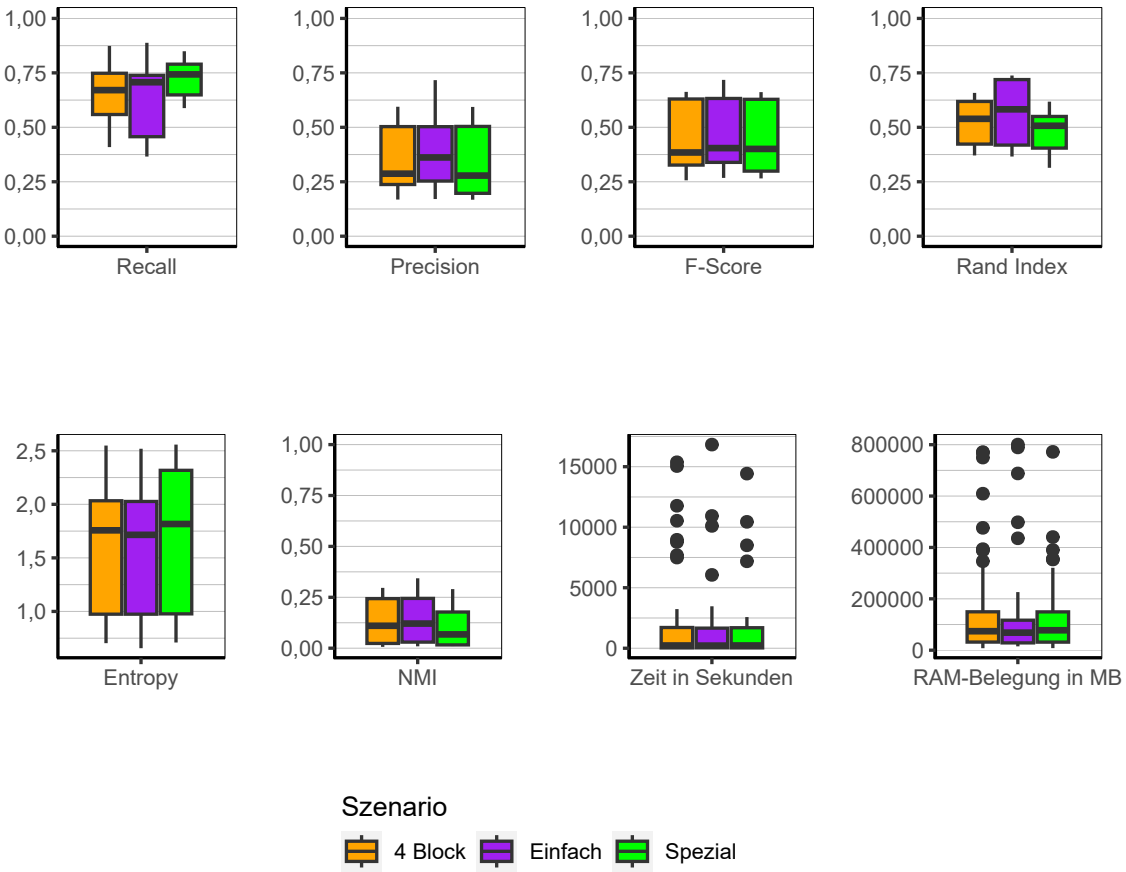


Abbildung 9.70.: MONA: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Bezüglich der Szenarien (Abbildung 9.70) lassen sich entgegen der Vermutung bei Betrachtung der Gini-Koffefizienten (Abbildung 9.65) keine nennenswerten Unterschiede feststellen.

Fazit Der Algorithmus MONA eignet sich nicht für die Verwendung als Clusteralgorithmus für Datensätze, welche der hier verwendeten Struktur folgen. Vor allem für Daten mit vielen Variablen nimmt die Laufzeit und der Speicherbedarf zu große Ausmaße an. Abgesehen davon ist die Qualität der resultierenden Clusterings auch generell nicht zufriedenstellend und ist selbst für einfache Datensätze ungeeignet.

9.3.11. Locality Sensitive Hashing

Nachfolgend werden die Ergebnisse des Algorithmus Locality Sensitive Hashing beschrieben.

Gesamt Wie sich in Abbildung 9.71 sehen lässt, liegt der F-Score für alle Ergebnisse[5] zusammen sehr niedrig. Lediglich einzelne Ergebnisse liegen über der 0,75 Marke. Dies bedeutet, dass Locality Sensitive Hashing in den meisten Fällen nicht gut abschneidet, jedoch in Ausnahmefällen in der Lage ist akzeptable Ergebnisse zu produzieren. Darüber zeigt sich LSH als schneller Algorithmus, welcher jedoch einen hohen Speicherbedarf mit sich bringt.

[5] Ein Gini-Koeffizient kann für die Ergebnisse von LSH nicht sinnvoll herangezogen werden. Dies liegt an der sehr großen Anzahl an Clustern, die der Algorithmus erzeugt. Zudem kann aufgrund der *Banding*-Technik eine Beobachtung in mehreren Clustern landen, was die Interpretation des Gini-Koeffizienten schwierig bzw. unmöglich macht.

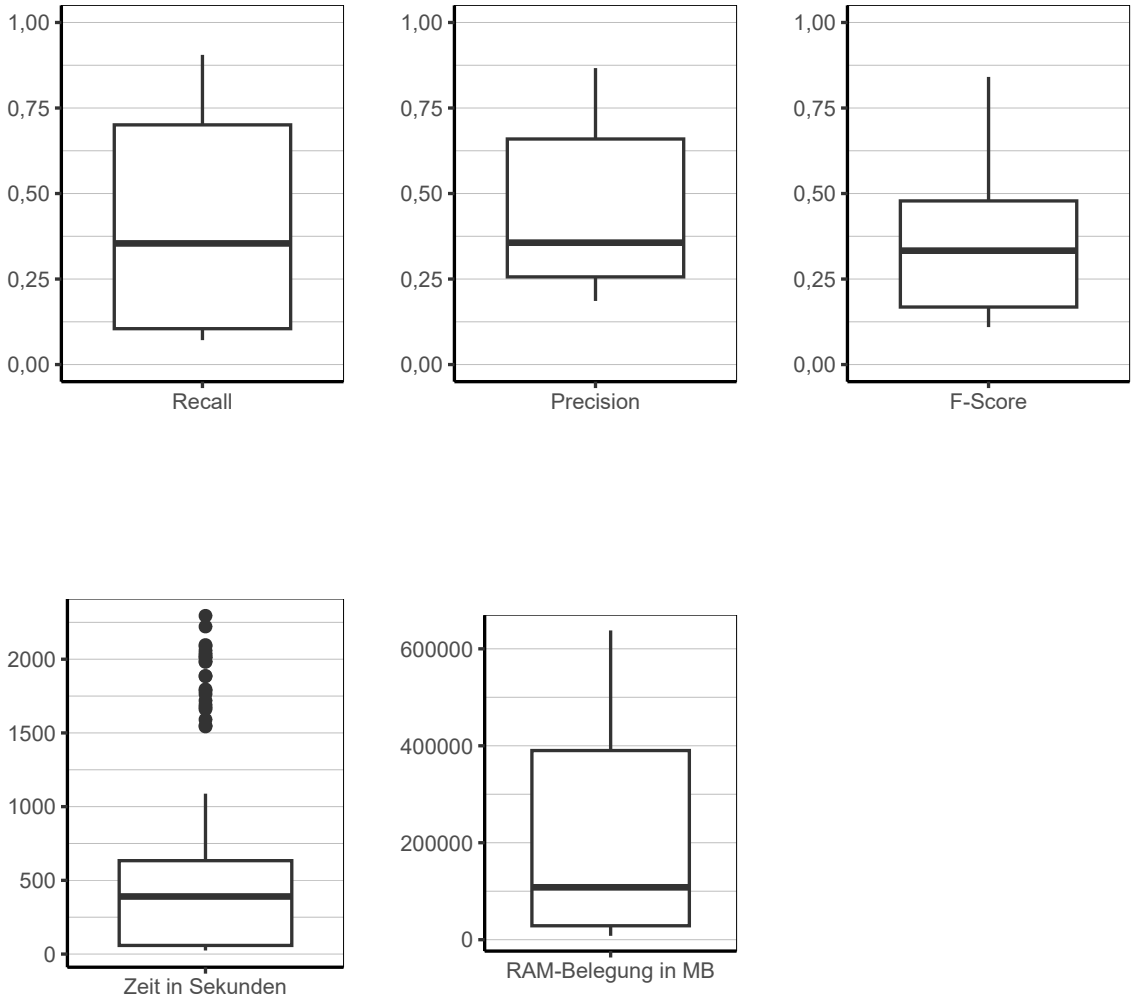


Abbildung 9.71.: Locality Sensitive Hashing: Evaluation aller Iterationen der Simulation insgesamt.

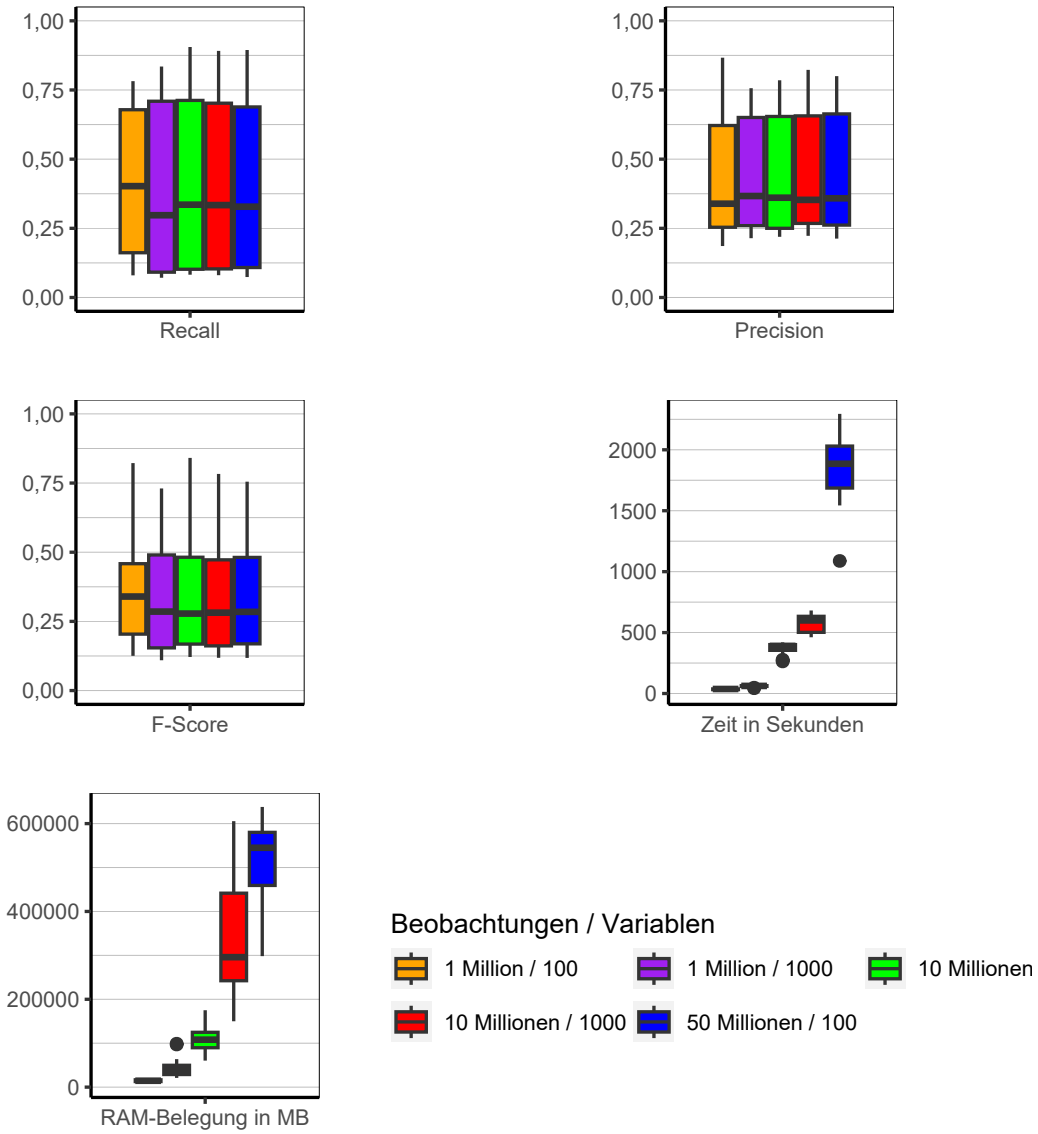


Abbildung 9.72.: LSH: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität In Abbildung 9.72 zeigt sich, dass die Dimensionalität keinen merklichen Einfluss auf die Qualität der Clusterlösungen zu haben scheint. Lediglich der Zeit- und Speicherbedarf steigt mit Anstieg der Beobachtungen an. Die Anzahl der Variablen wiederum wirkt sich kaum auf die Laufzeit aus. Bezüglich des Speicherbedarfs jedoch steigt dieser in der Konfiguration mit 1.000 Variablen gegenüber der mit 100 Variablen stark an. Dies ist vor allem bei 10 Millionen Beobachtungen eindeutig zu sehen.

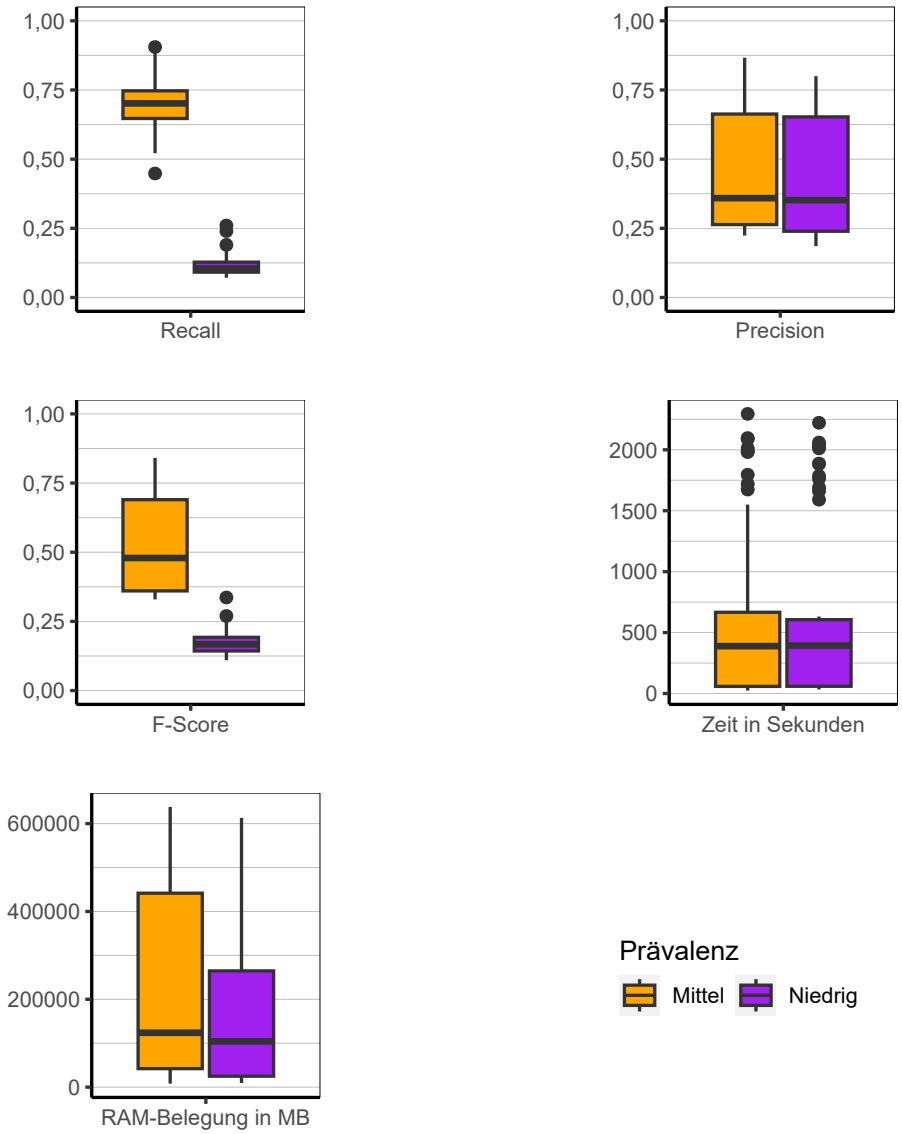


Abbildung 9.73.: LSH: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Ein interessantes Bild zeigt sich in Abbildung 9.73. So sind die Werte für Recall viel höher, wenn die Prävalenz hoch ist. Der Algorithmus ist also wesentlich besser darin, Beobachtungen, die zusammen gehören zu erkennen, wenn die Prävalenz hoch ist. Wie sich jedoch an dem Boxplot für Precision sehen lässt, geht das damit einher, dass auch viele Beobachtungen zusammengeführt werden, welche nicht zusammen in einem Cluster sein sollten. Das kulminiert dann in einem höheren F-Score für die Konfigurationen mit höherer Prävalenz. Bezüglich Laufzeit und Speicherbedarf sind keine

großen Unterschiede zu sehen. Zwar ist die Box des Speicherverbrauchs bei der höheren Prävalenz höher, jedoch liegen die Mediane auf einem ähnlichen Niveau.

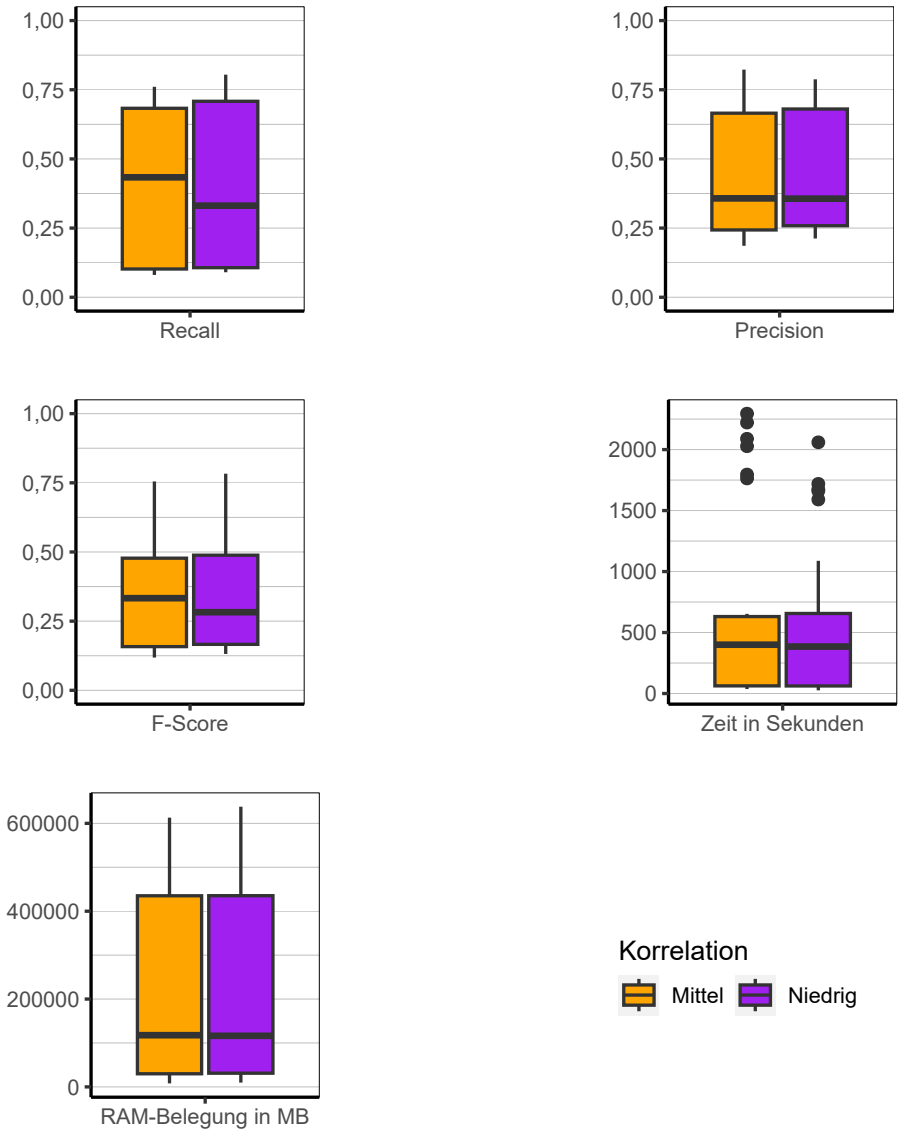


Abbildung 9.74.: LSH: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Die Höhe der Korrelation der Variablen untereinander scheint keinen Einfluss auf die Performance des Algorithmus zu haben (siehe Abbildung 9.74). Weder bei den Qualitätsindikatoren, noch bei Laufzeit und Speicherbedarf sind große Unterschiede auszumachen.

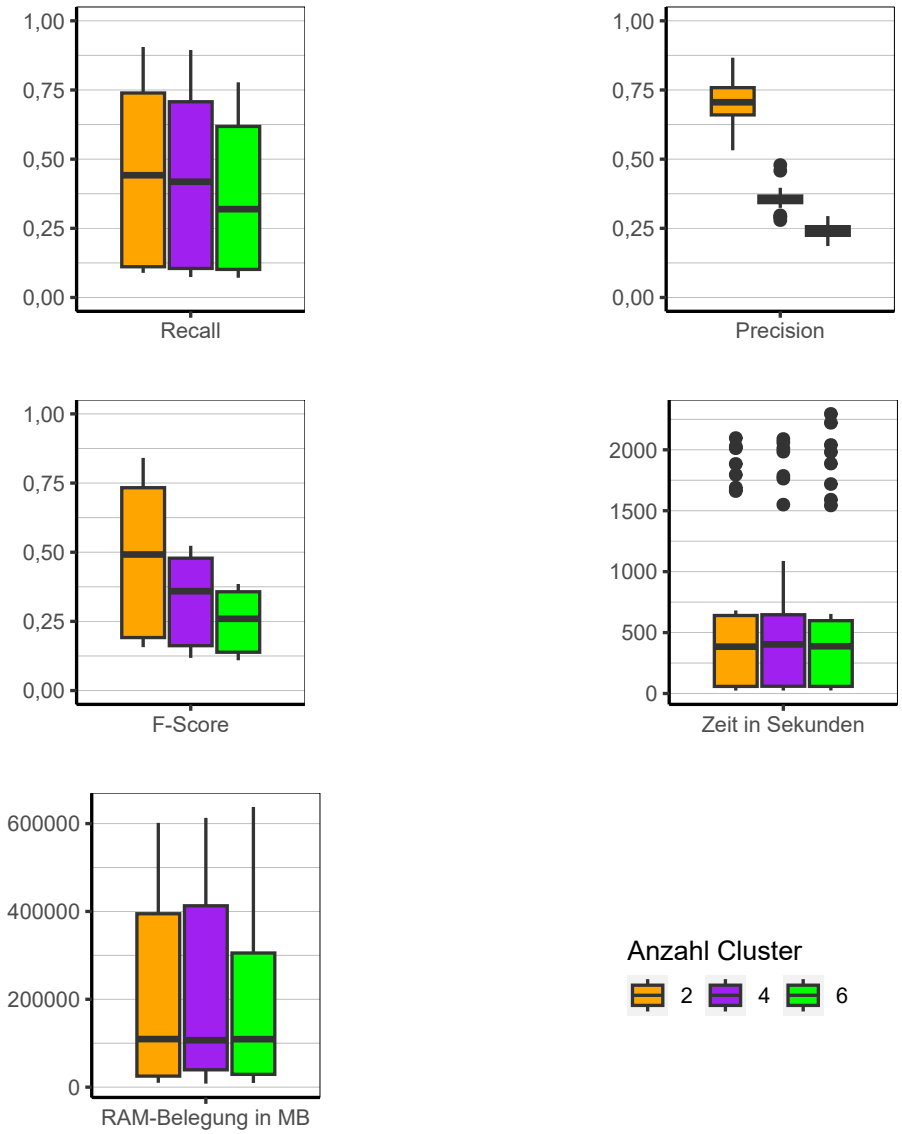


Abbildung 9.75.: LSH: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Wird die Anzahl der Cluster betrachtet (siehe Abbildung 9.75), zeigt sich bei Precision ein bekanntes Bild, was sich auch bei anderen Algorithmen beobachten ließ. Jedoch liegt der Wert bei zwei Clustern mit einem Median von 0,77 relativ hoch und stürzt dann bei mehr Clustern ab. Bezüglich Recall lassen sich keine großen Unterschiede ausmachen und die Werte liegen auf einem relativ niedrigen Niveau. Das bedeutet, bei lediglich zwei Clustern ist der Algorithmus in der Lage Beobachtungen zusammenzuführen, die auch zusammen gehören, findet davon jedoch nicht viele.

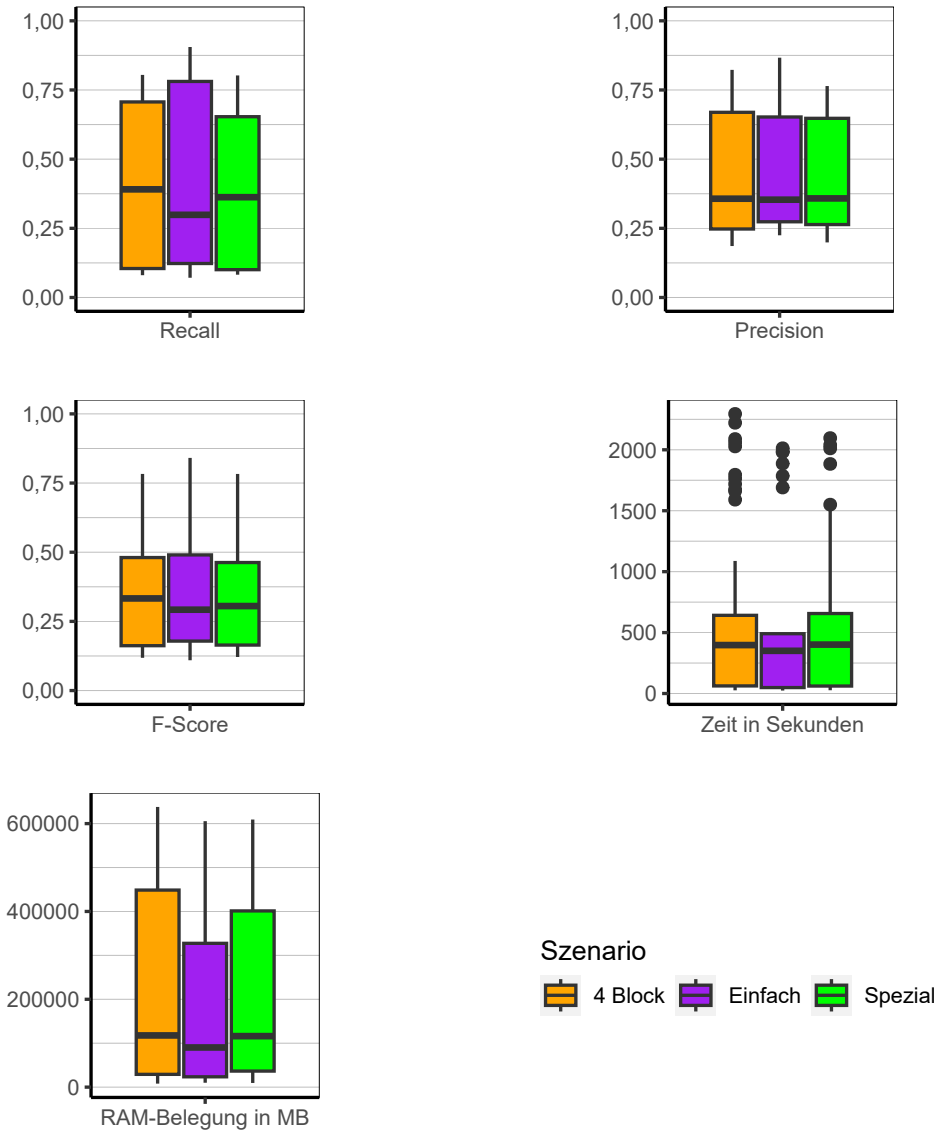


Abbildung 9.76.: LSH: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Waren die meisten bislang betrachteten Algorithmen in der Lage vor allem im Szenario *Einfach* bessere Ergebnisse zu liefern, hat die Art des Szenarios keinen erheblichen Einfluss auf die Qualität der Clusterings (siehe Abbildung 9.76). Sowohl die Qualitätsindikatoren, als auch die Laufzeit und Speicherbedarf zeigen keine wesentlichen Unterschiede je nach Art des Szenarios.

Fazit Wie bereits angesprochen, ist der Algorithmus für das generelle Clustering der verwendeten Daten nicht gut geeignet. In Ausnahmefällen liefert er ein akzeptables Ergebnis was das Finden von Beobachtungen aus dem selben Cluster angeht, bildet allerdings selbst viele kleine Cluster und neigt zu einer niedrigen Precision. Er funktioniert besser bei höherer Prävalenz und ist dabei mit einer relativ geringen Laufzeit versehen.

9.3.12. MinHash

Im Folgenden werden die Ergebnisse des Algorithmus MinHash beschrieben.

Gesamt Über alle Faktorstufen des Designs zusammen (siehe Abbildung 9.77) lässt sich sagen, dass der Algorithmus keine gute Clusterlösungen produziert. Vor allem die Werte für Recall sind sehr niedrig. Das bedeutet, dass der Algorithmus nicht gut darin ist Beobachtungen als zusammengehörig zu identifizieren, welche aus demselben Cluster stammen. Auffallend ist ebenfalls, dass der Algorithmus eine hohe Anzahl an Clustern produziert. Diese liegen im Mittel über alle Iterationen bei 2078,18 Clustern. Aufgrund dieser Tatsache werden wie schon beim LSH die Werte für den Rand Index, Entropy und NMI nicht diskutiert. Die Laufzeit des Algorithmus kann als sehr niedrig beschrieben werden, der Speicherbedarf wiederum als relativ hoch. In Abbildung 9.78 lässt sich sehen, dass der Algorithmus zu sehr schiefen Verteilungen bei der Allokation von Beobachtungen zu Clustern tendiert.

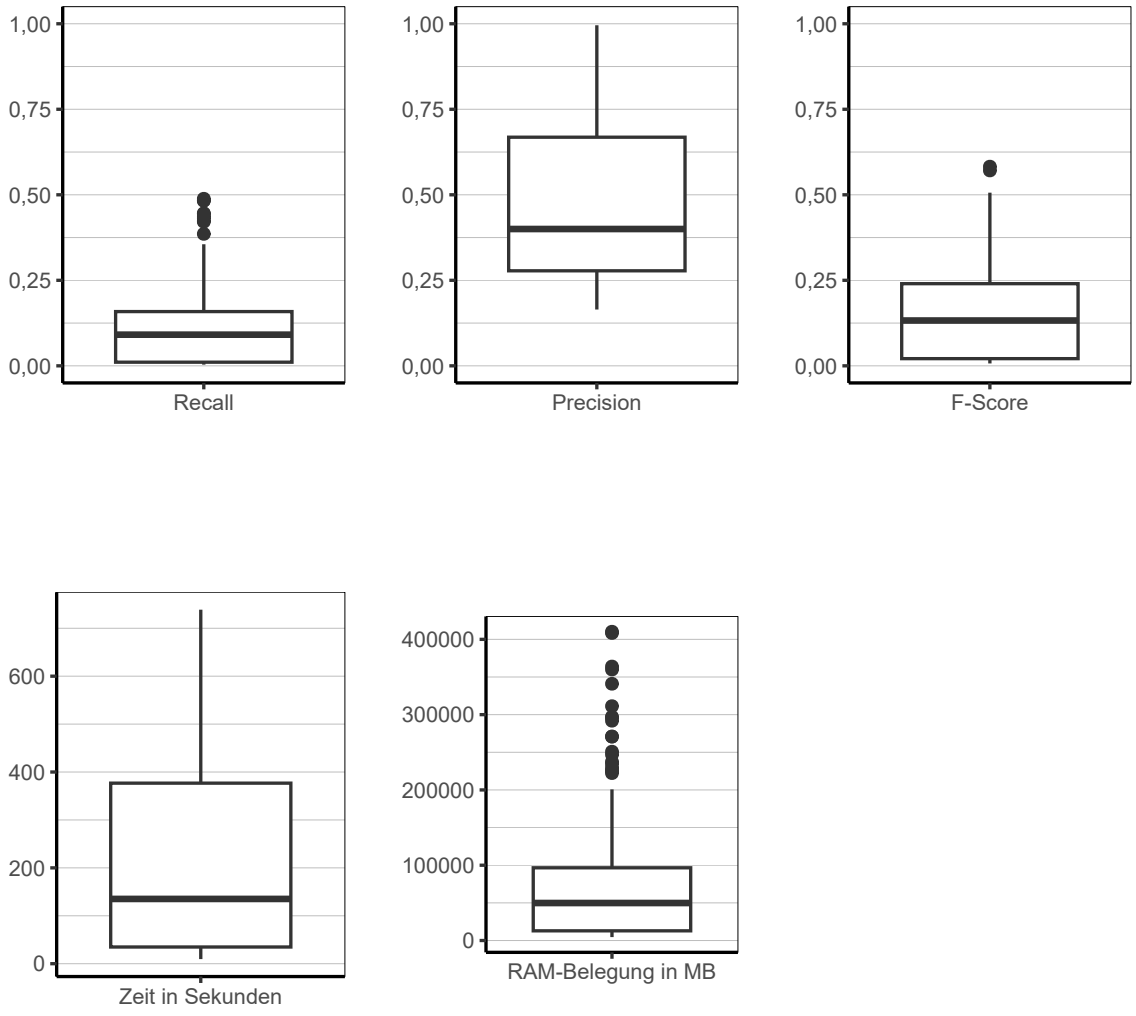


Abbildung 9.77.: MinHash: Evaluation aller Iterationen der Simulation insgesamt.

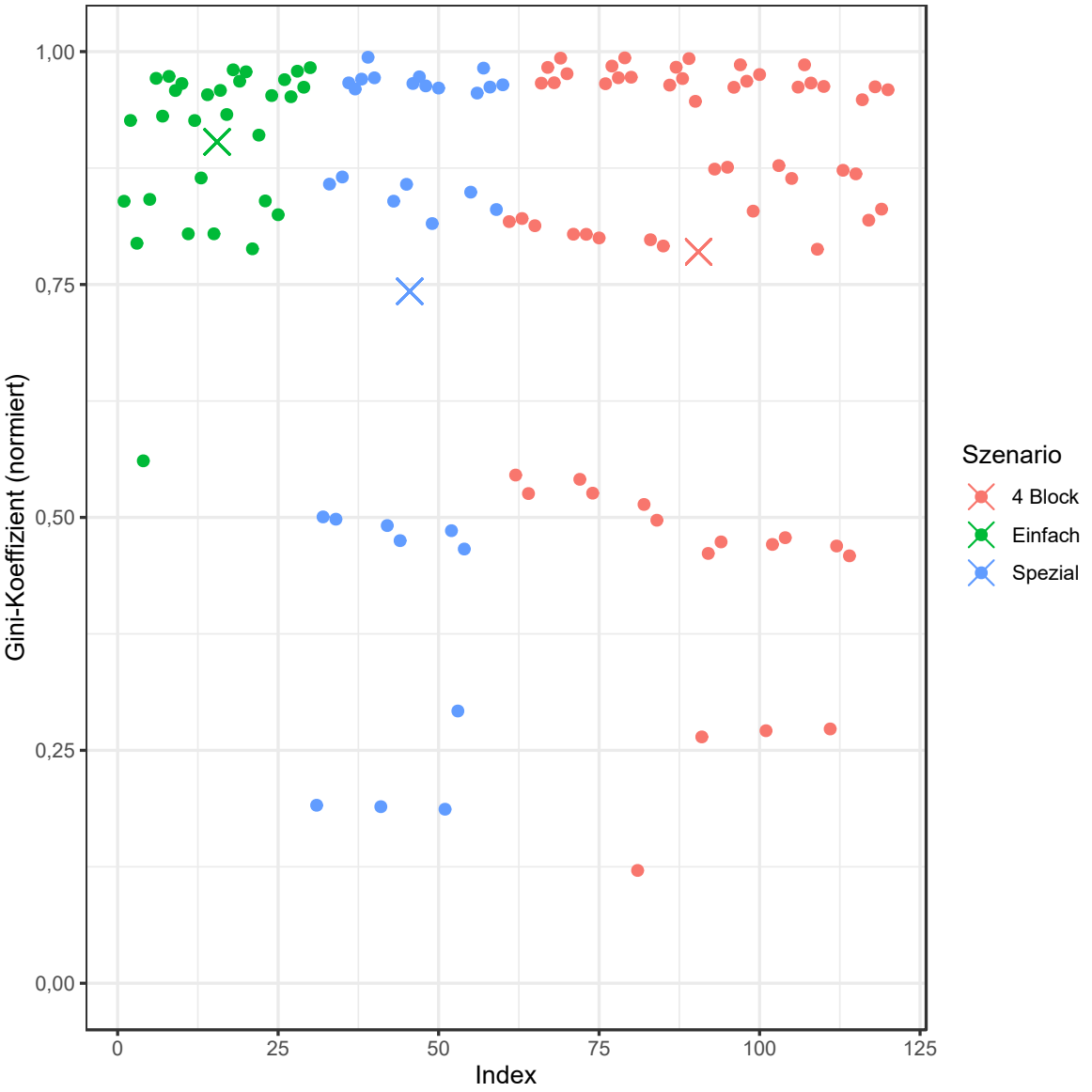


Abbildung 9.78.: MinHash: Gini-Koeffizient aller Iterationen der Simulation.

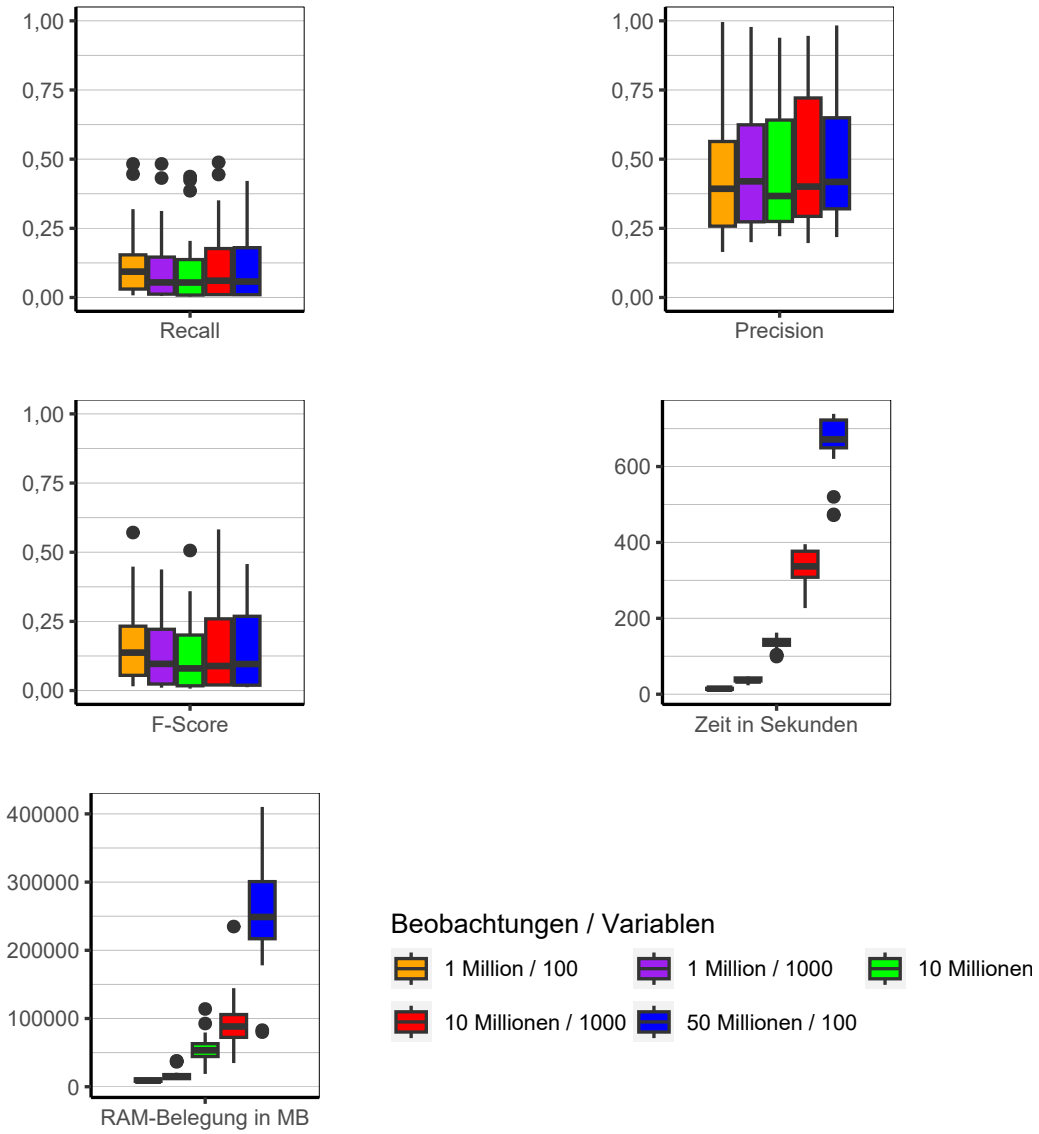


Abbildung 9.79.: MinHash: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Die Dimensionalität hat keinen Einfluss auf die Qualität der resultierenden Clusterings (siehe Abbildung 9.79). Sowohl Laufzeit als auch Speicherbedarf steigen jedoch relativ zueinander stark an, wenn sich auch die Dimensionalität erhöht. Absolut betrachtet bleibt jedoch vor allem die Laufzeit auf einem niedrigen Niveau. Der Speicherbedarf ist vor allem bei 50 Millionen Beobachtungen hoch.

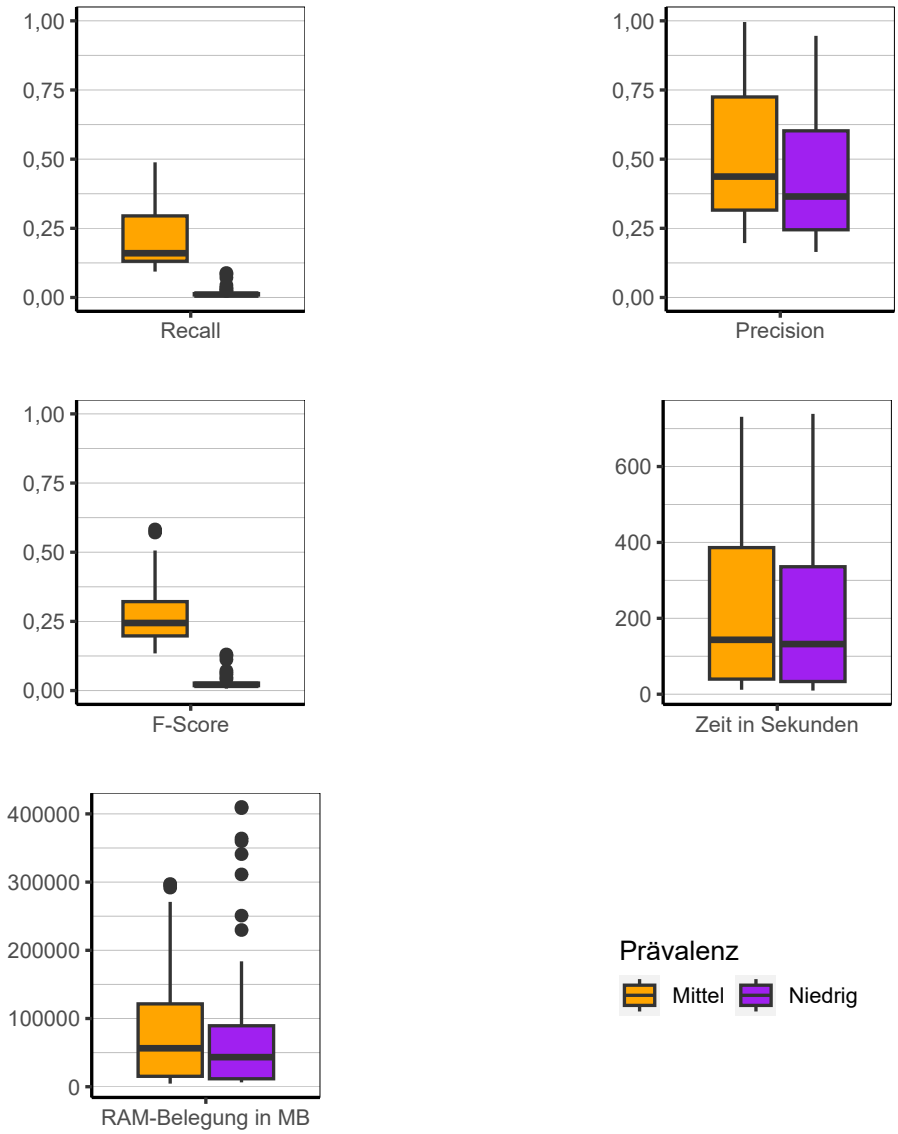


Abbildung 9.80.: MinHash: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Bezüglich der Prävalenz kann bei MinHash ein ähnliches Muster beobachtet werden, wie bei LSH, wenn auch in nicht so starker Ausprägung (siehe Abbildung 9.80). Der Algorithmus ist besser in der Lage Beobachtungen aus demselben Cluster zusammenzuführen, wenn die Prävalenz höher ist. Jedoch befinden sich die Werte für Recall dann immernoch auf einem sehr niedrigen Niveau, weshalb auch bei den höheren der beiden Konfigurationen nicht von guten Ergebnissen gesprochen werden kann. Sowohl Laufzeit als auch Speicherbedarf bleiben von der Prävalenz nahezu unberührt.

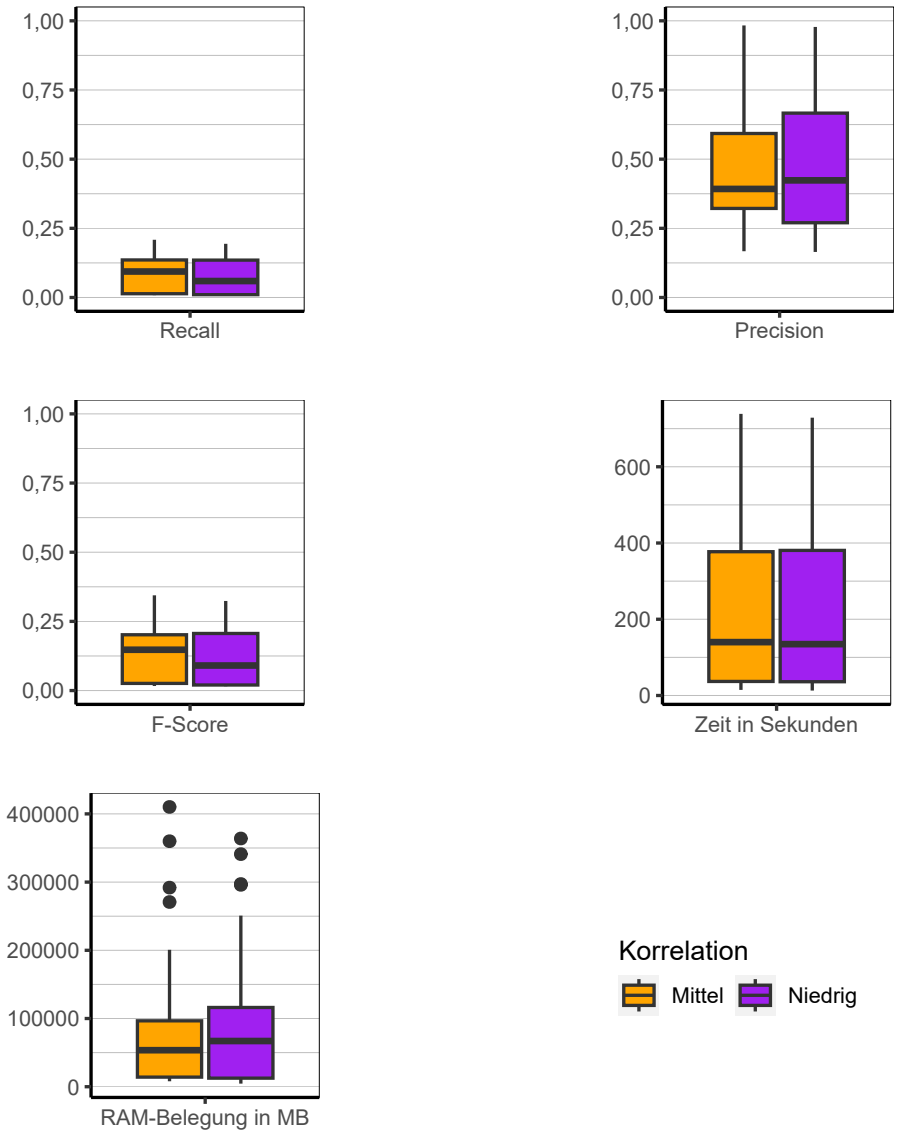


Abbildung 9.81.: MinHash: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Es lassen sich keine Einflüsse der Korrelation auf die Ergebnisse beobachten (siehe Abbildung 9.81).

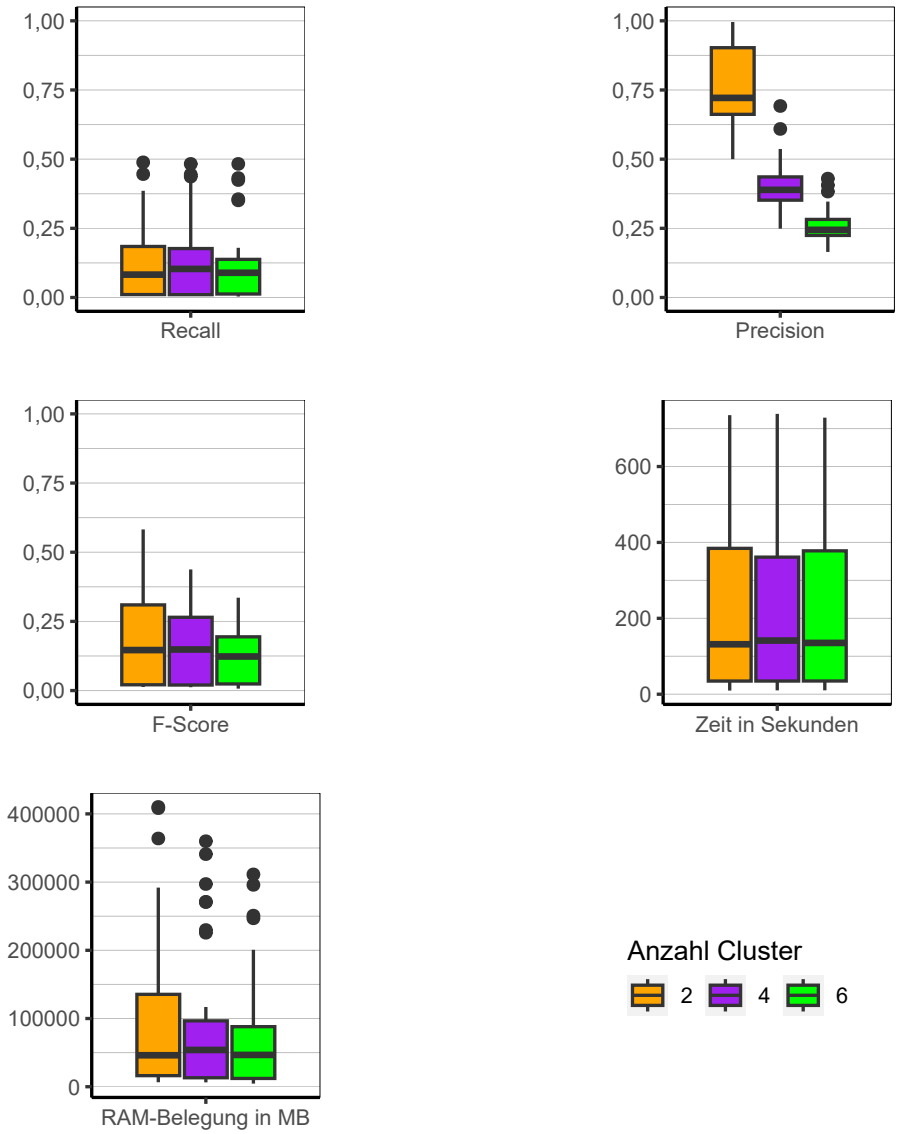


Abbildung 9.82.: MinHash: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Ebenso wie bei der Prävalenz, lassen sich auch bezüglich der Anzahl der Cluster die gleichen Muster erkennen wie beim Algorithmus LSH, jedoch auf einem wesentlich niedrigeren Level bei Recall (siehe Abbildung 9.82). Dies wiederum führt dann auch zu einem niedrigeren F-Score. Die Precision fällt stufenweise mit der Anzahl der Cluster ab. Laufzeit und Speicherbedarf werden nicht von der Anzahl der Cluster beeinflusst.

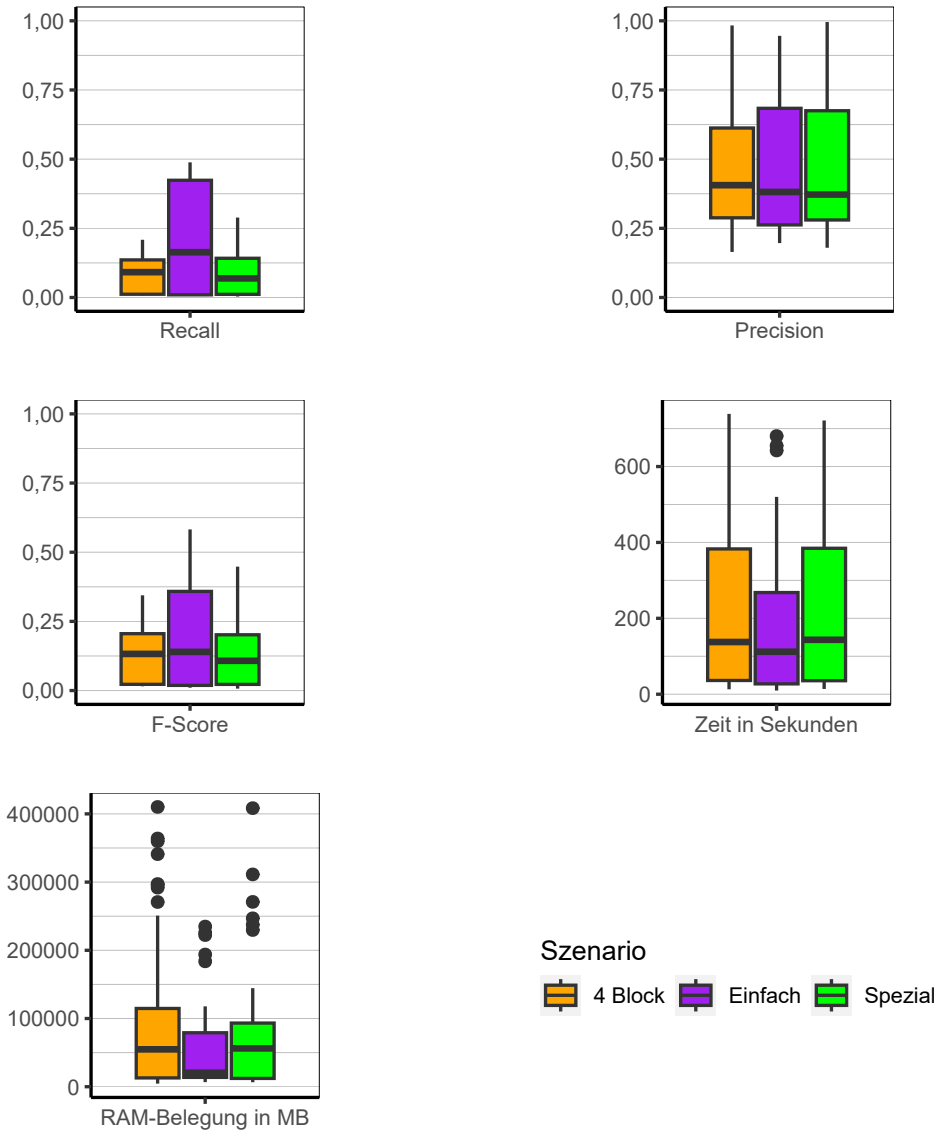


Abbildung 9.83.: MinHash: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Auch bei MinHash werden die besten Ergebnisse im Szenario *Einfach* erzielt (siehe Abbildung 9.83). Die Werte für Recall sind beim Szenario *Einfach* etwas höher als bei *4 Block* und *Spezial*. Jedoch auch hier sind die Werte sehr niedrig. Laufzeit und Speicherbedarf werden nicht merklich von der Art des Szenarios beeinflusst.

Fazit Da es sich bei MinHash um einen ähnlich beschaffenen Algorithmus wie LSH handelt, werden auch ähnliche Muster beobachtet. Jedoch muss geurteilt werden, dass der Algorithmus noch einmal schlechter abschneidet. Er läuft zwar schneller und benötigt weniger Arbeitsspeicher, ist aber dennoch ungeeignet für das klassische Clustering von Daten genutzt zu werden, welche eine Struktur aufweisen, die in dieser Simulation verwendet werden.

9.3.13. Canopy Clustering

Folgend werden die Ergebnisse des Algorithmus Canopy Clustering dargelegt.[6,7]

Gesamt Über alle Faktorstufen insgesamt zeigt sich auch in Abbildung 9.84 bei Canopy Clustering ein ernüchterndes Bild. Die Qualitätsindikatoren sind durchweg auf einem niedrigen Level. Lediglich bei Recall gibt es einen Whisker der bis zur 1 geht. Die Laufzeit des Algorithmus kann jedoch als schnell bezeichnet werden, auch wenn dies mit einem relativ hohen Speicherbedarf einher geht. Bei Betrachtung von Abbildung 9.85 kann eine relativ hohe Varianz des Gini-Koeffizienten festgestellt werden, welcher sich nicht nach Szenario unterscheidet. Der Algorithmus produziert demnach Verteilungen von Beobachtungen auf Cluster, welche von sehr schief bis gleichverteilt gehen.

[6] Aufgrund der schlechten Ergebnisse des Algorithmus, wurde die Simulation für diesen Algorithmus nochmals wiederholt und Canopy Clustering wurde sequenziell ausgeführt, um die Parallelisierung als mögliche Fehlerquelle auszuschließen. Wie sich zeigte, sind die Ergebnisse vergleichbar. Jedoch gibt es einige positive Ausreißer in der sequenziellen Variante, welche sehr gute Ergebnisse zeigen. Die Ergebnisplots finden sich in Anhang B.

[7] Aufgrund eines Segfault Errors bei der Konfiguration mit 10 Millionen Beobachtungen, 1.000 Variablen, mittlerer Prävalenz, Szenario *Einfach* und sechs Clustern konnten die Ergebnisse für diese Konfiguration nicht berichtet werden.

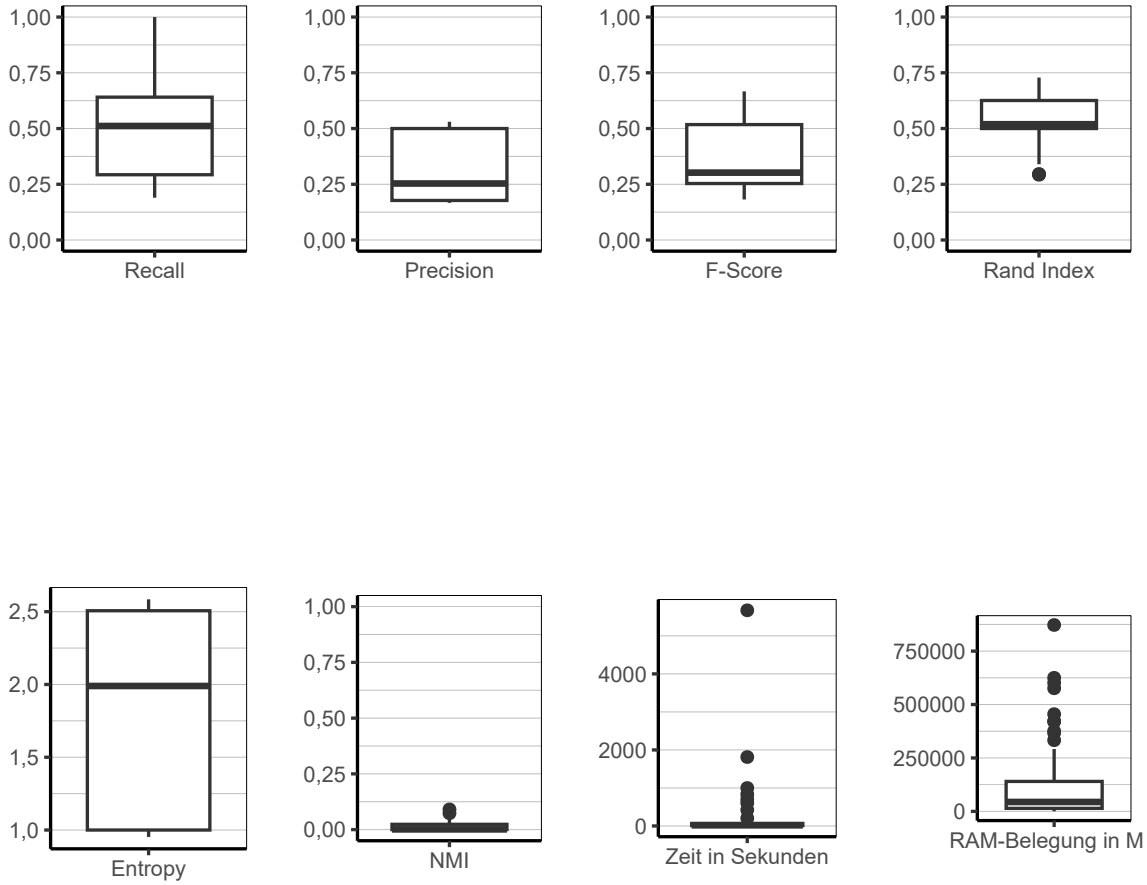


Abbildung 9.84.: Canopy Clustering: Evaluation aller Iterationen der Simulation insgesamt.

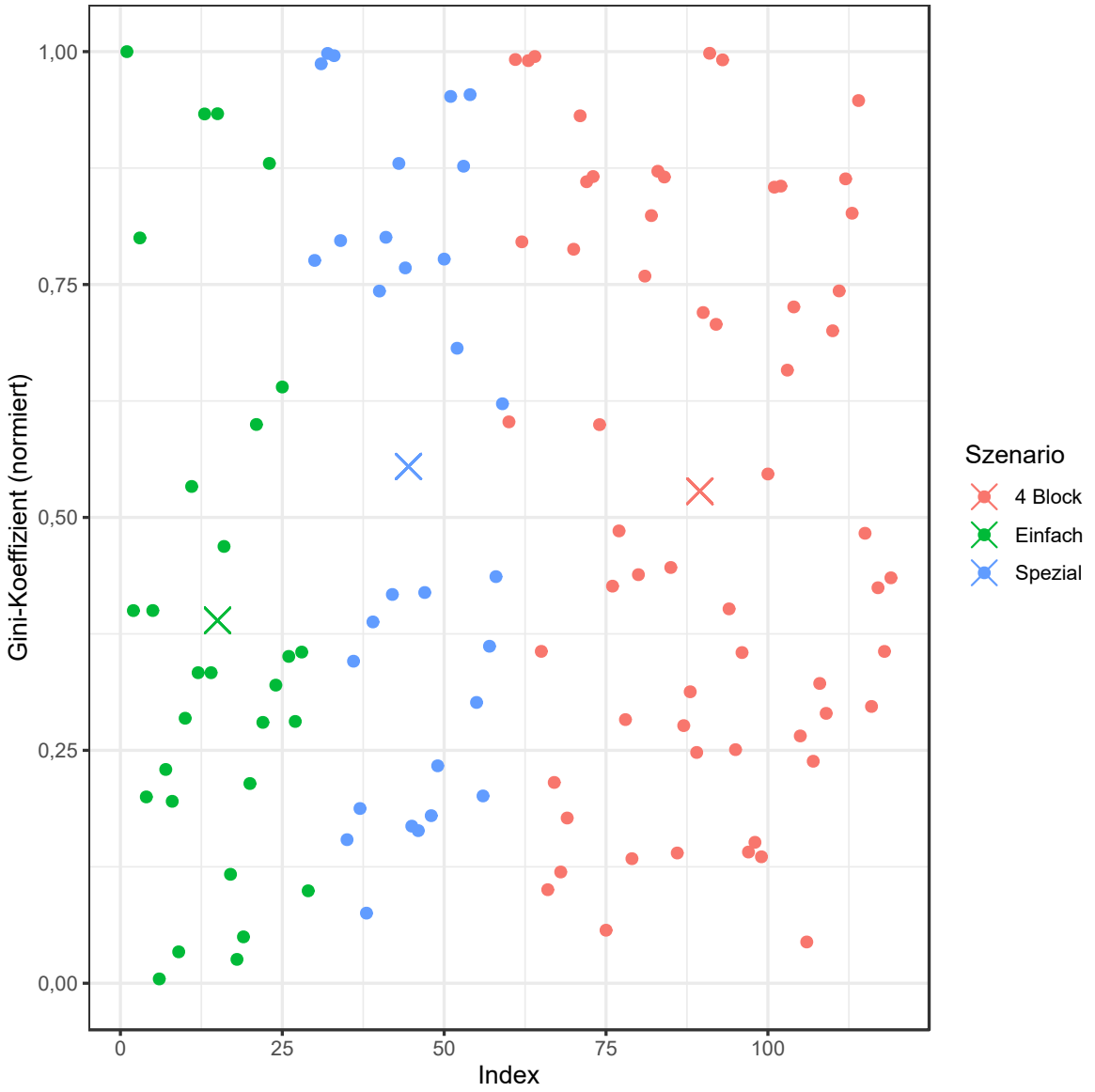


Abbildung 9.85.: Canopy Clustering: Gini-Koeffizient aller Iterationen der Simulation.

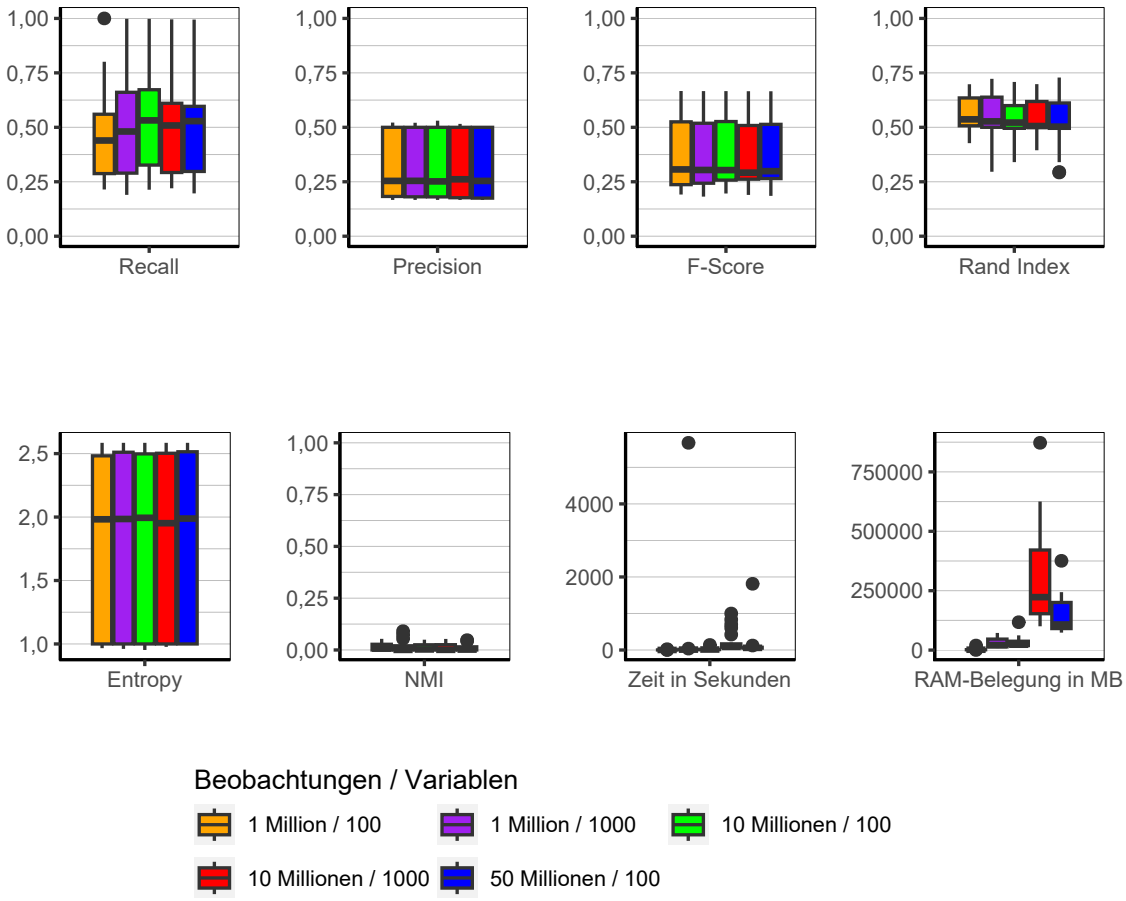


Abbildung 9.86.: Canopy Clustering: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Wie sich in Abbildung 9.86 sehen lässt, hat die Dimensionalität keinen Einfluss auf die Qualität der Clusterlösungen. Das einzig interessante Ergebnis ist der erhöhte Speicherbedarf bei 10 Millionen Beobachtungen und 1.000 Variablen. Dieser liegt noch weit über dem von 50 Millionen Beobachtungen mit 100 Variablen.

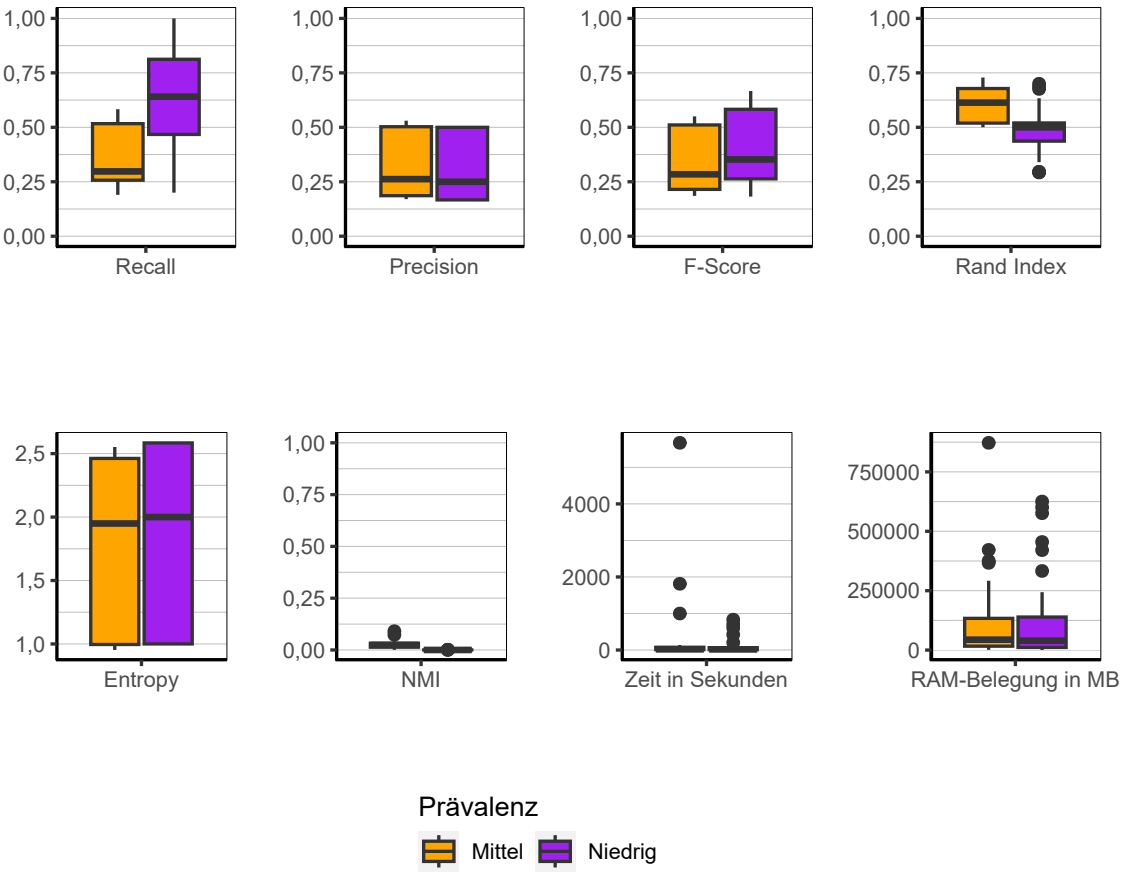


Abbildung 9.87.: Canopy Clustering: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Auch bezüglich der Prävalenz lassen sich kaum Unterschiede ausmachen (siehe Abbildung 9.87). Lediglich die Werte für Recall sind entgegen der Beobachtung bei anderen Algorithmen bei niedrigerer Prävalenz höher.

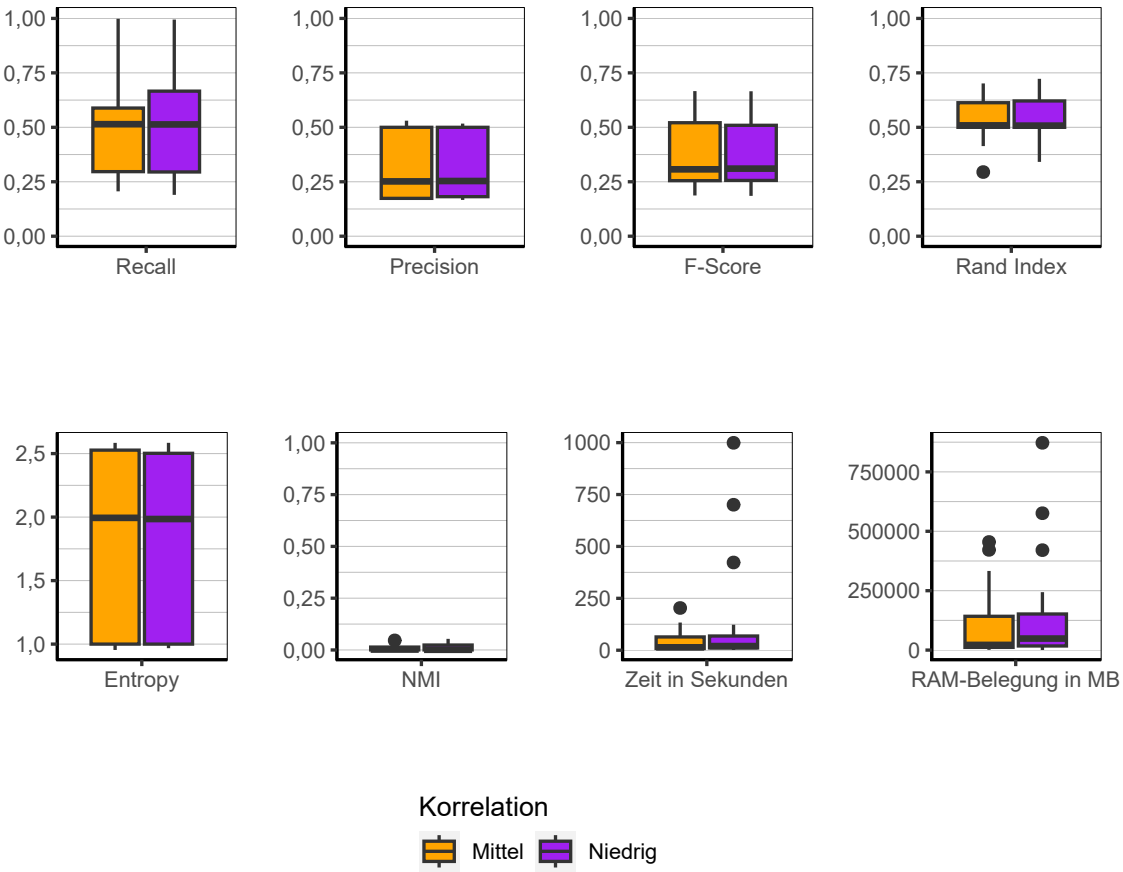


Abbildung 9.88.: Canopy Clustering: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Es lassen sich keine relevanten Unterschiede zwischen den Ergebnissen der beiden Konfigurationen bezüglich der Korrelation beobachten (siehe Abbildung 9.88).

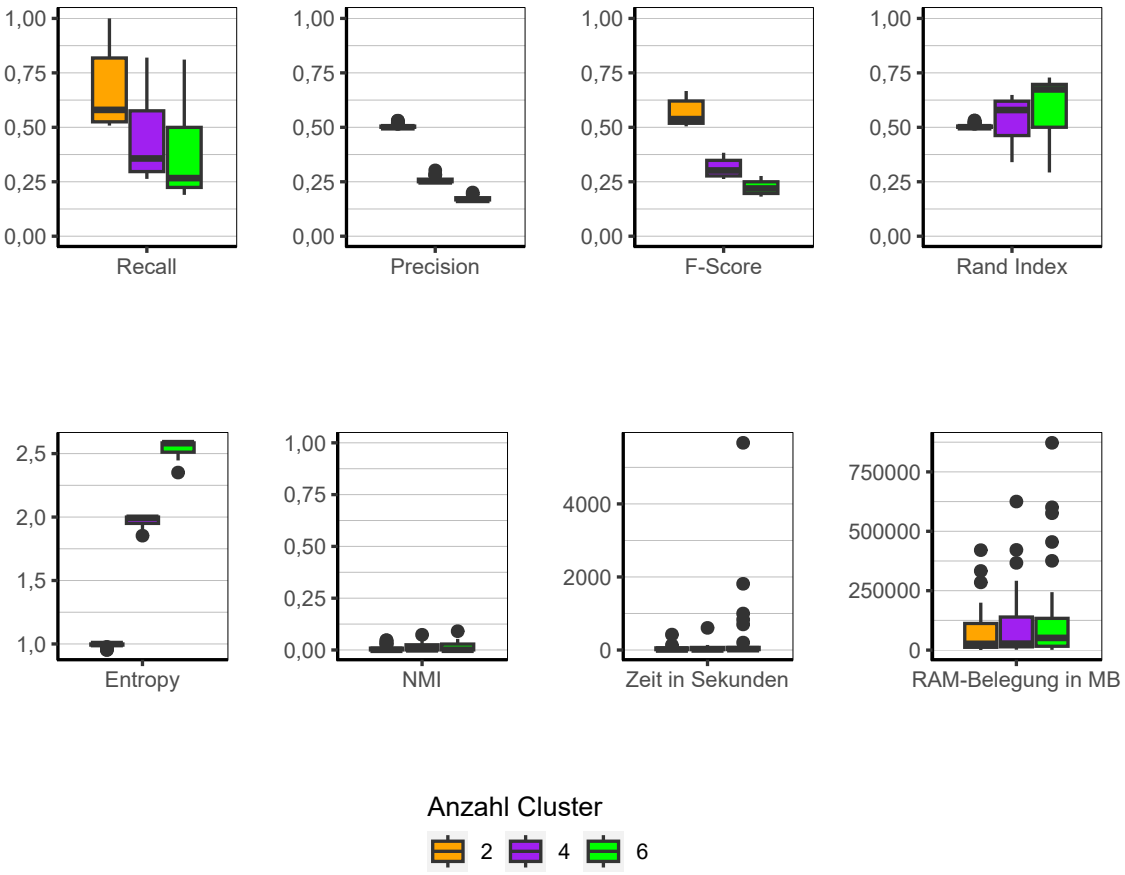


Abbildung 9.89.: Canopy Clustering: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Auch bei Canopy Clustering lässt sich das bekannte Muster sehen, dass mit Anstieg der Anzahl der Cluster die Qualität der Clusterlösungen verschlechtert (siehe Abbildung 9.89). Laufzeit und Speicherbedarf sind davon nicht betroffen.

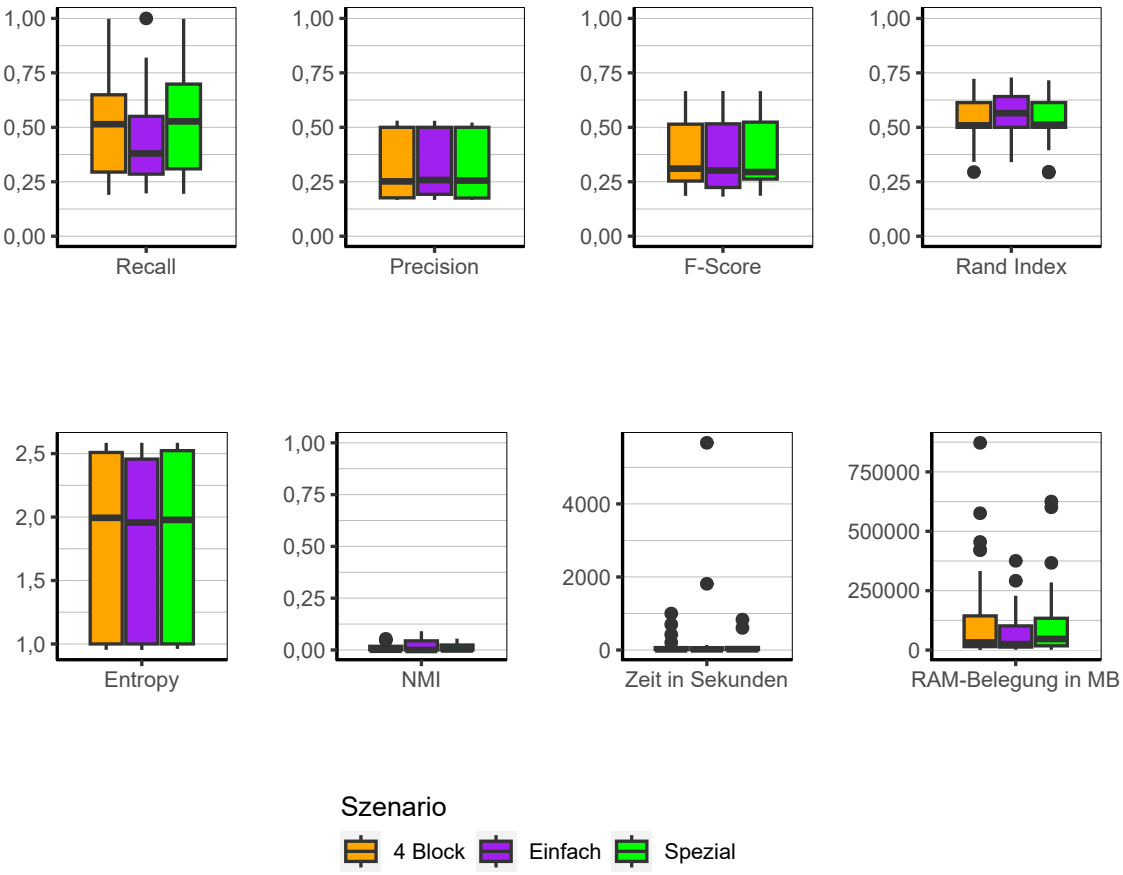


Abbildung 9.90.: Canopy Clustering: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Es sind keine nennenswerten Unterschiede zwischen den Szenarien zu erkennen (Abbildung 9.90). Einzige Ausnahme bilden die Werte für Recall beim Szenario *Einfach*, welche etwas niedriger sind als in den anderen beiden Szenarien.

Fazit Canopy Clustering liefert in der durchgeführten Simulation keine guten Ergebnisse. Keiner der Faktoren des Designs haben darauf einen Einfluss. Die Laufzeit des Algorithmus ist zwar schnell, der Speicherbedarf jedoch hoch.

9.3.14. A General Model for Clustering Binary Data: Two Side Clustering

Folgend werden die Ergebnisse des Algorithmus A General Model for Clustering Binary Data in der Variante Two Side Clustering beschrieben.

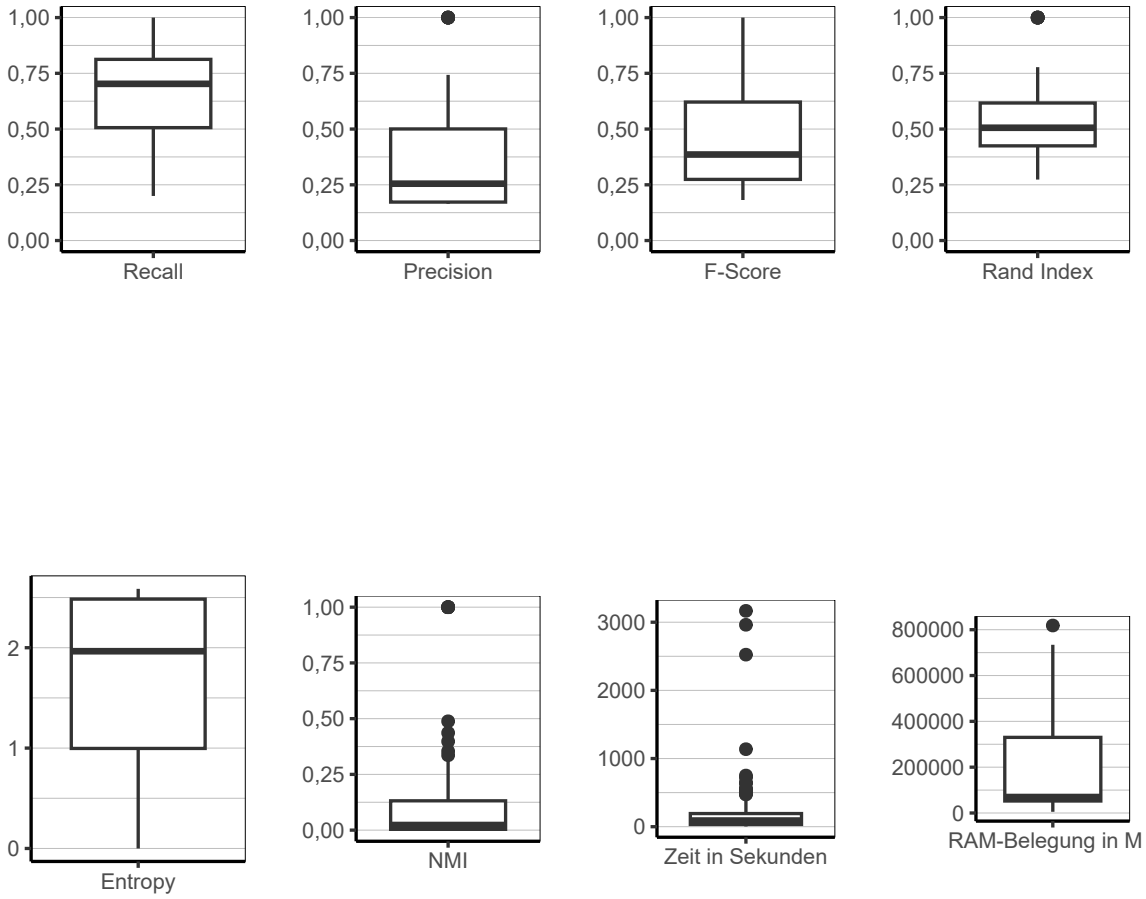


Abbildung 9.91.: General Model (Two Side Clustering): Evaluation aller Iterationen der Simulation insgesamt.

Gesamt Über alle Faktorstufen betrachtet (siehe Abbildung 9.91) zeigt sich, dass auch dieser Algorithmus nicht in der Lage ist konstant gute Ergebnisse zu produzieren.

Wie schon bei anderen Algorithmen lässt sich jedoch sehen, dass vereinzelte Ausreißer exisitieren, in welchen der Algorithmus perfekte Ergebnisse hervor bringt. Auffallend ist, dass die Werte für Precision wesentlich niedriger sind als für Recall. Der Algorithmus ist schnell und der Speicherbedarf größtenteils moderat, in Teilen jedoch hoch. Abbildung 9.92 zeigt eine Tendenz des Algorithmus, schiefe Cluster zu produzieren. Wobei in Szenario *Einfach* mit wesentlich ausgeglicheneren Clustern gerechnet werden kann. Hier sind auch die perfekten Clusterlösungen zu finden.

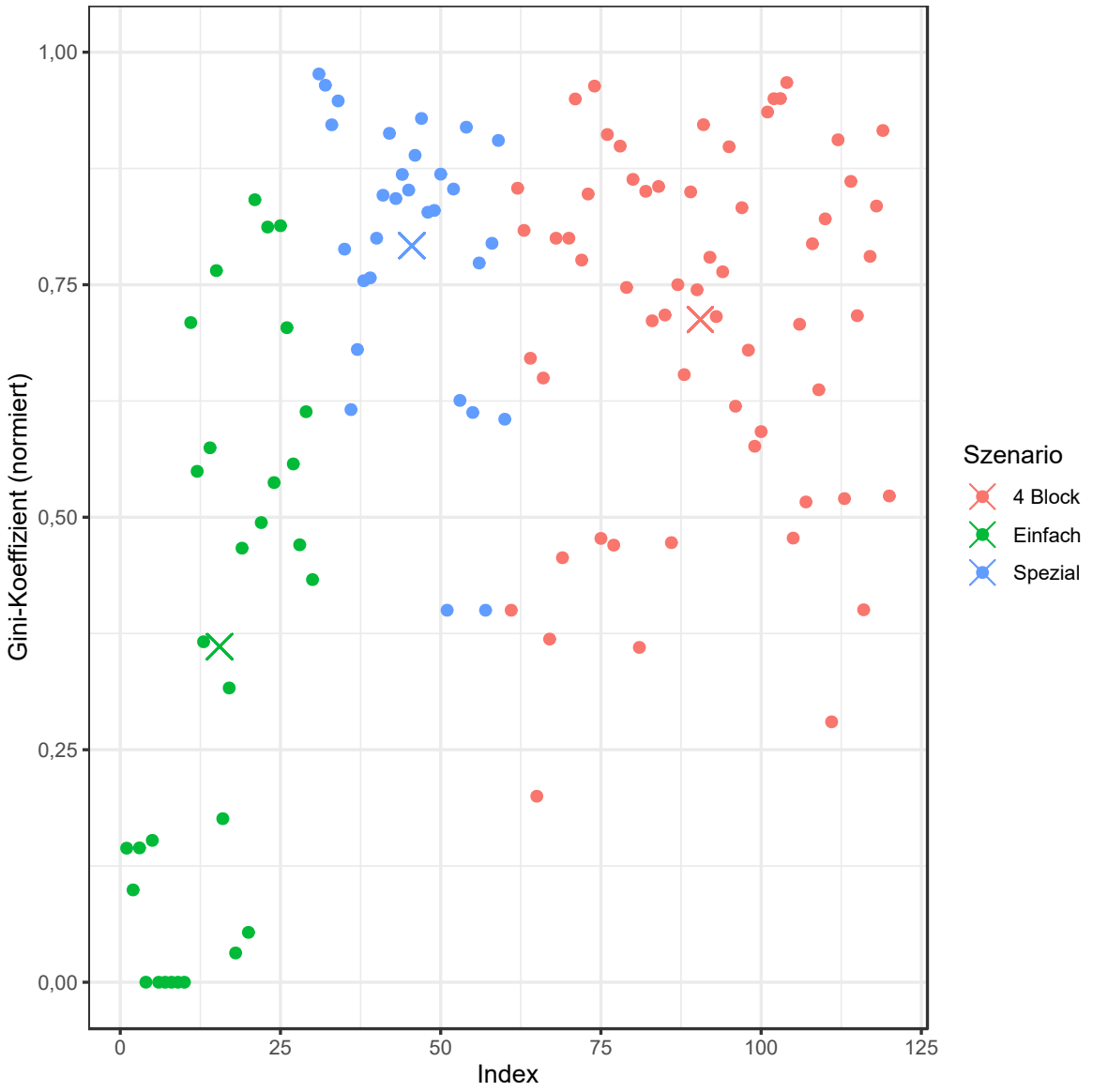


Abbildung 9.92.: General Model (Two Side Clustering): Gini-Koeffizient aller Iterationen der Simulation.

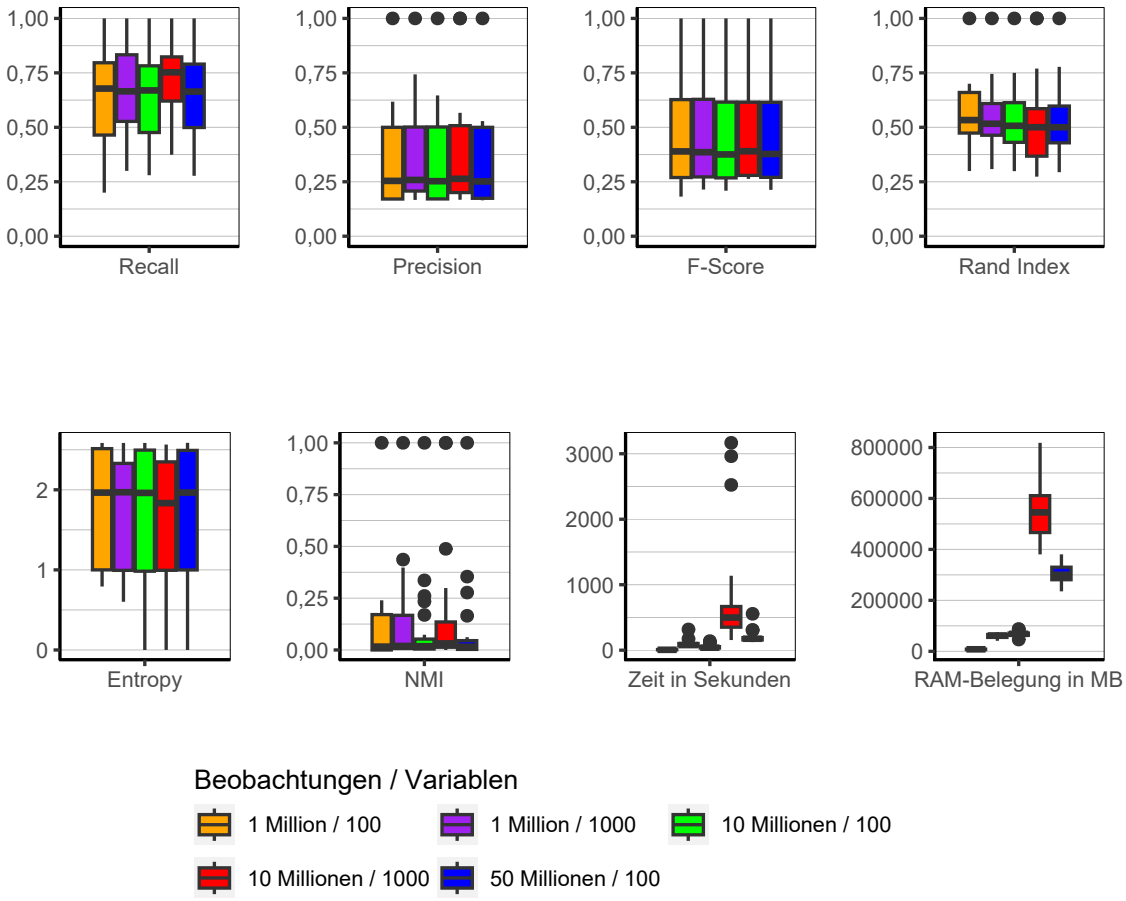


Abbildung 9.93.: General Model (Two Side Clustering): Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität Wie sich in Abbildung 9.93 sehen lässt, ist kein merklicher Effekt zwischen den einzelnen Konfigurationen bezüglich der Dimensionalität der Daten zu erkennen. Der einzig erwähnenswerte Einfluss besteht im bereits mehrfach beobachteten Phänomen, dass 10 Millionen Beobachtungen und 1.000 Variablen mehr Speicher und eine längere Laufzeit benötigt als 50 Millionen Beobachtungen und 100 Variablen. Das bedeutet, dass die Anzahl der Variablen einen größeren Einfluss hat als die Anzahl der Beobachtungen.

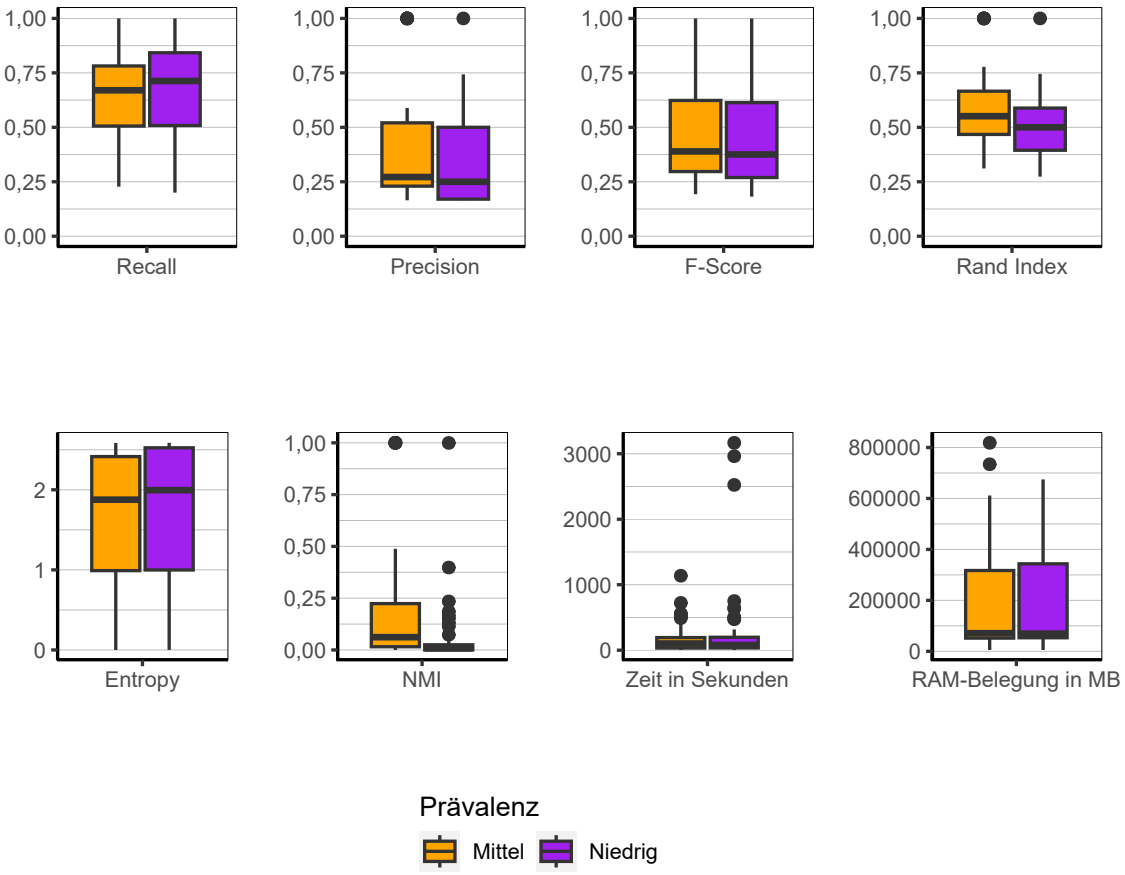


Abbildung 9.94.: General Model (Two Side Clustering): Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Die Ergebnisse deuten darauf hin, dass die Prävalenz der Einsen in den Daten keinen entscheidenden Unterschied in den Ergebnissen hervorruft (siehe Abbildung 9.94).

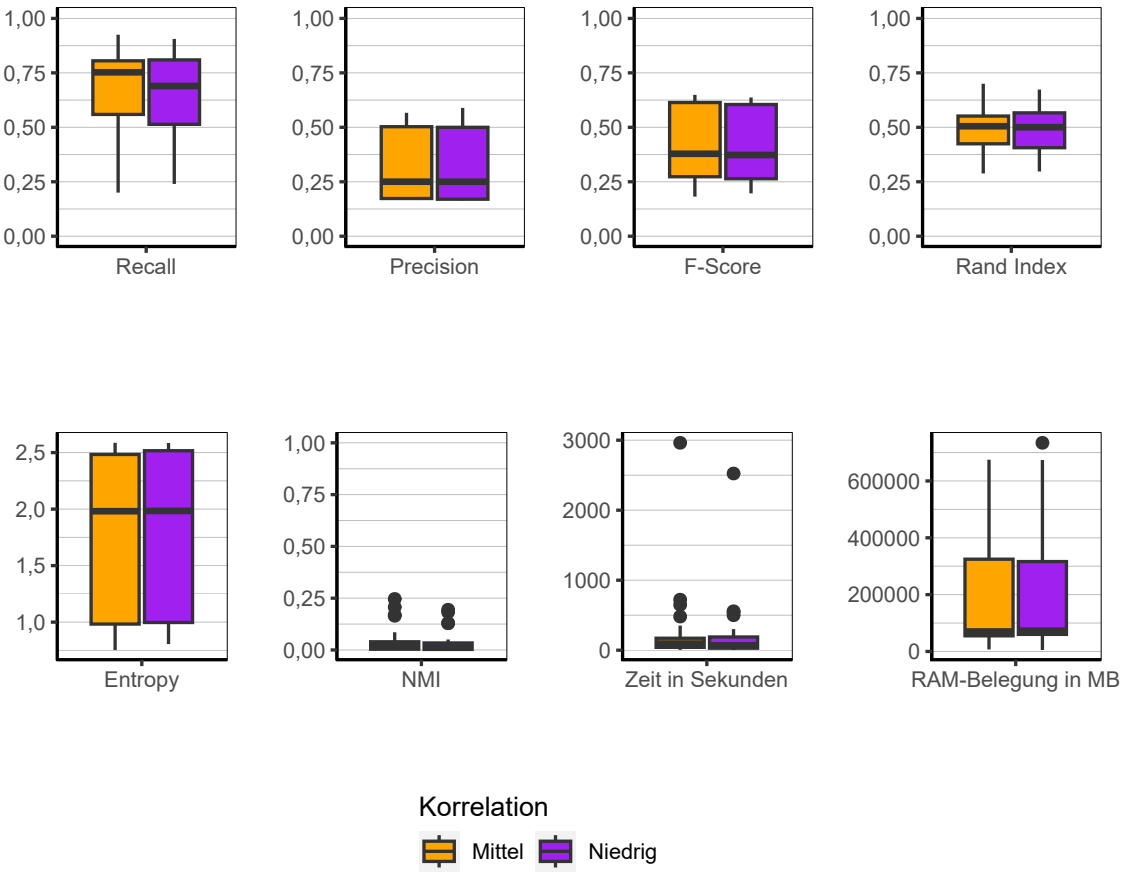


Abbildung 9.95.: General Model (Two Side Clustering): Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Auch die Höhe der Korrelation der Variablen untereinander führt zu keinen Unterschieden in der Qualität der Clusterlösungen, der Laufzeit oder des Speicherbedarfs (siehe Abbildung 9.95).

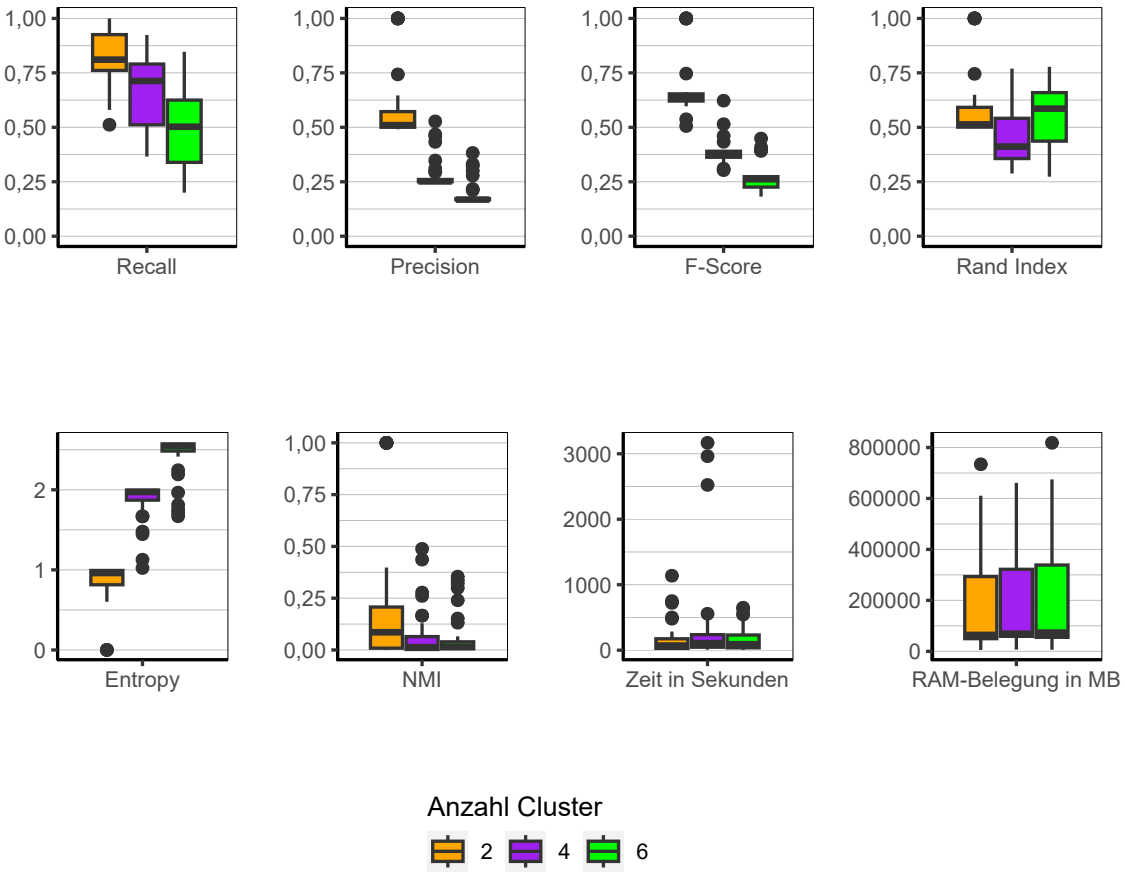


Abbildung 9.96.: General Model (Two Side Clustering): Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Auch bei der Anzahl der Cluster lässt sich wiederum das gleiche Bild wie bei anderen Algorithmen mit schlechten Ergebnissen beobachten (siehe Abbildung 9.96). Die Qualität nimmt mit Anstieg der Anzahl der Cluster ab.

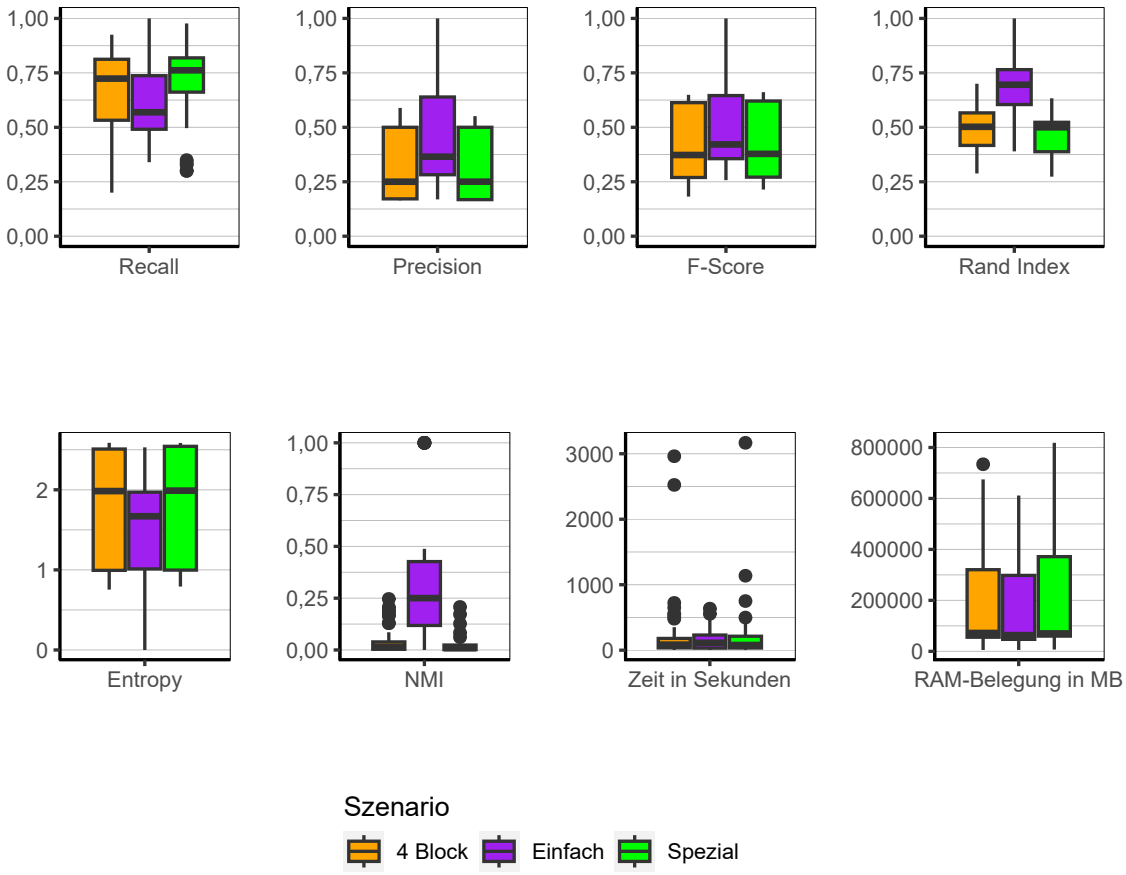


Abbildung 9.97.: General Model (Two Side Clustering): Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Eine interessante Beobachtung lässt sich in Abbildung 9.97 bezüglich des Einflusses des gewählten Szenarios machen. Während die Recall-Werte für Szenario *Einfach* niedriger liegen als bei den anderen beiden, sind die Werte für Precision höher, was zu einem höheren F-Score führt. Die wenigen perfekten Clusterlösungen sind ebenfalls in Szenario *Einfach* zu finden.

Fazit Auch dieser Algorithmus ist zu großen Teilen nicht für das Clustern der hier verwendeten Daten geeignet. Zwar konnte er unter bestimmten, sehr einfachen Umständen perfekte Clusterlösungen erzielen, aber sobald die Cluster nicht mehr eindeutig getrennt sind und vermehrt Rauschen in den Daten zu finden ist, verliert der Algorithmus die Fähigkeit akzeptable Clusterlösungen zu produzieren. Jedoch kann auch diesem Algorithmus eine relativ geringe Laufzeit attestiert werden.

9.3.15. A General Model for Clustering Binary Data: Block Diagonal Variant

Folgend werden die Ergebnisse des Algorithmus A General Model for Clustering Binary Data in der Variante Block Diagonal Variant beschrieben.

Gesamt Bei Betrachtung der Ergebnisse über alle Faktorstufen zusammen (Abbildung 9.98) kann festgestellt werden, dass die Block Diagonal-Variante des Algorithmus noch schlechter abschneidet als Two Side Clustering. Es wird direkt klar, dass der Algorithmus für die verwendeten Daten nahezu unbrauchbar ist. Abbildung 9.99 zeigt ein sehr interessantes Muster. Zwar kann beobachtet werden, dass die Gini-Koeffizienten eine relativ hohe Varianz aufweisen, aber das eigentlich Interessante sind die vielen identischen Werte auf unterschiedlichen Niveaus. So gibt es viele Gini-Koeffizienten mit den Werten 0,2, 0,4, 0,6 und 0,8. Die Häufigkeitstabellen der Clusterlabels innerhalb der einzelnen Clusterlösungen zeigen dabei Verteilungen, in denen die Häufigkeit eines Clusterlabels das Ein- oder Vielfache der Anzahl der tatsächlichen Clusterlabels darstellen. Dies kommt daher, dass die einzelnen parallel arbeitenden Tasks dazu neigen vorläufige Clusterlösungen des jeweiligen Chunks zu erstellen, in denen alle Beobachtungen einem Cluster zugewiesen werden. Beim späteren Zusammentragen der Ergebnisse der einzelnen Tasks werden diese dann wiederum in einzelne Cluster gesteckt bzw. zusammen gruppiert. Das führt dann dazu, dass in den verschiedenen Iterationen der Simulation identische Verteilungen der Clusterlables zu beobachten sind.

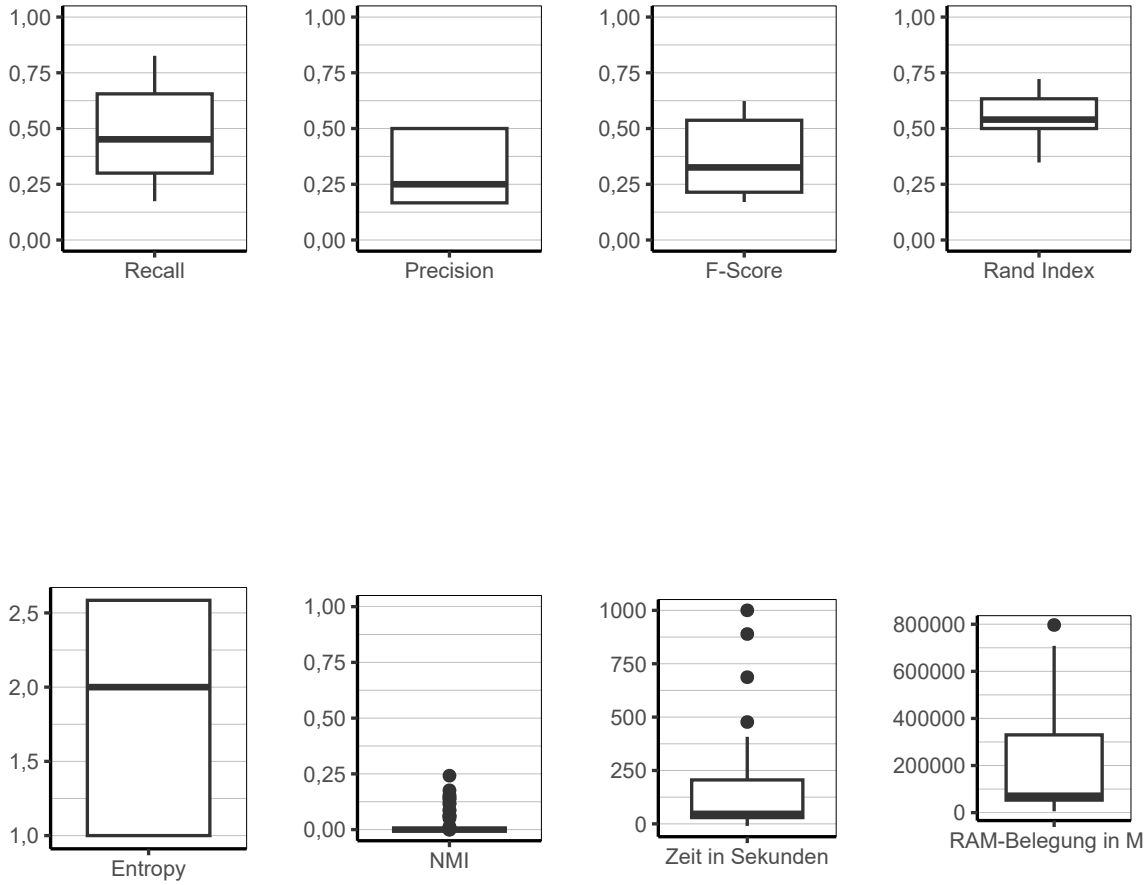


Abbildung 9.98.: General Model (Block Diagonal Variant): Evaluation aller Iterationen der Simulation insgesamt.

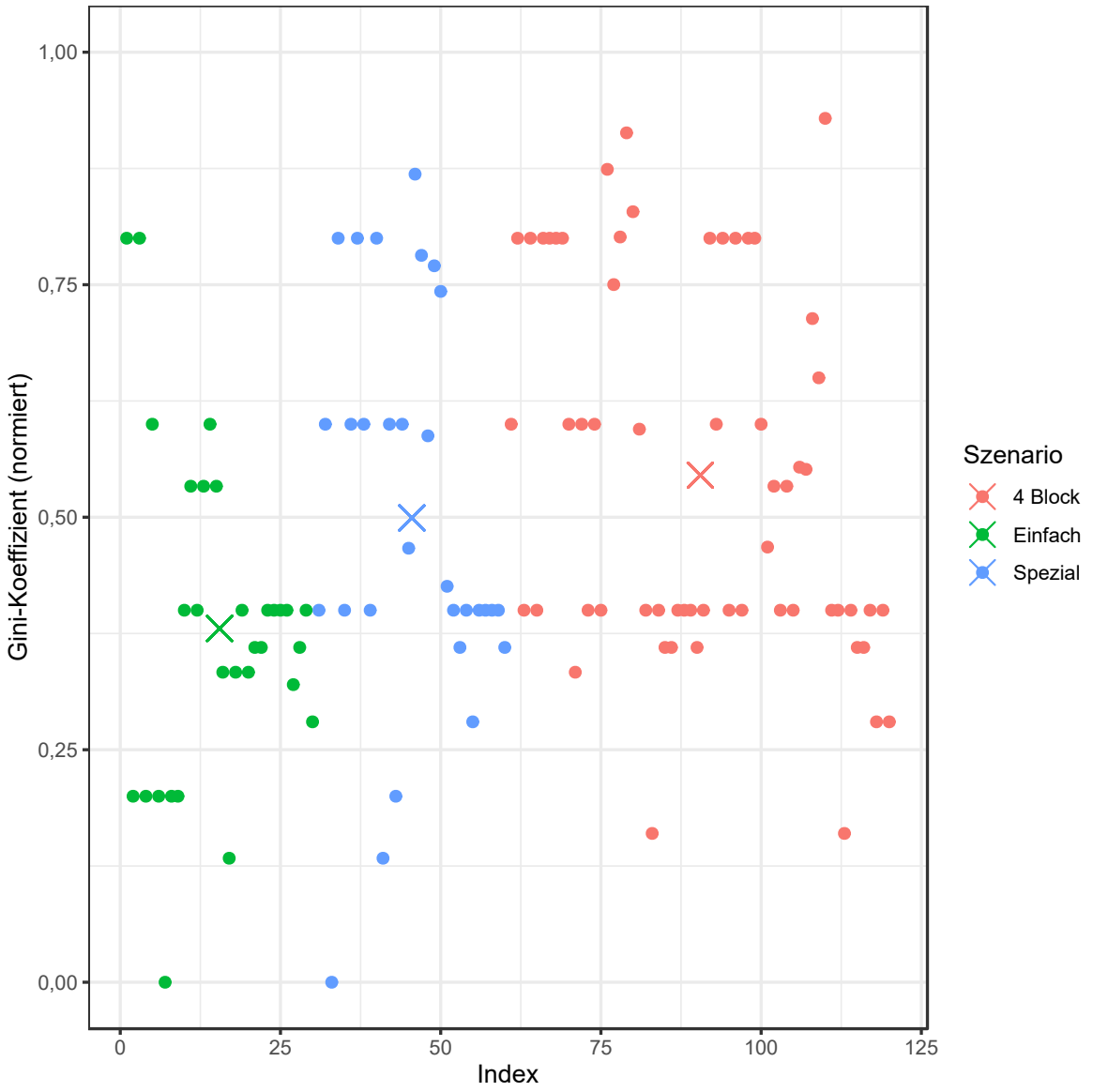


Abbildung 9.99.: General Model (Block Diagonal Variant): Gini-Koeffizient aller Iterationen der Simulation.

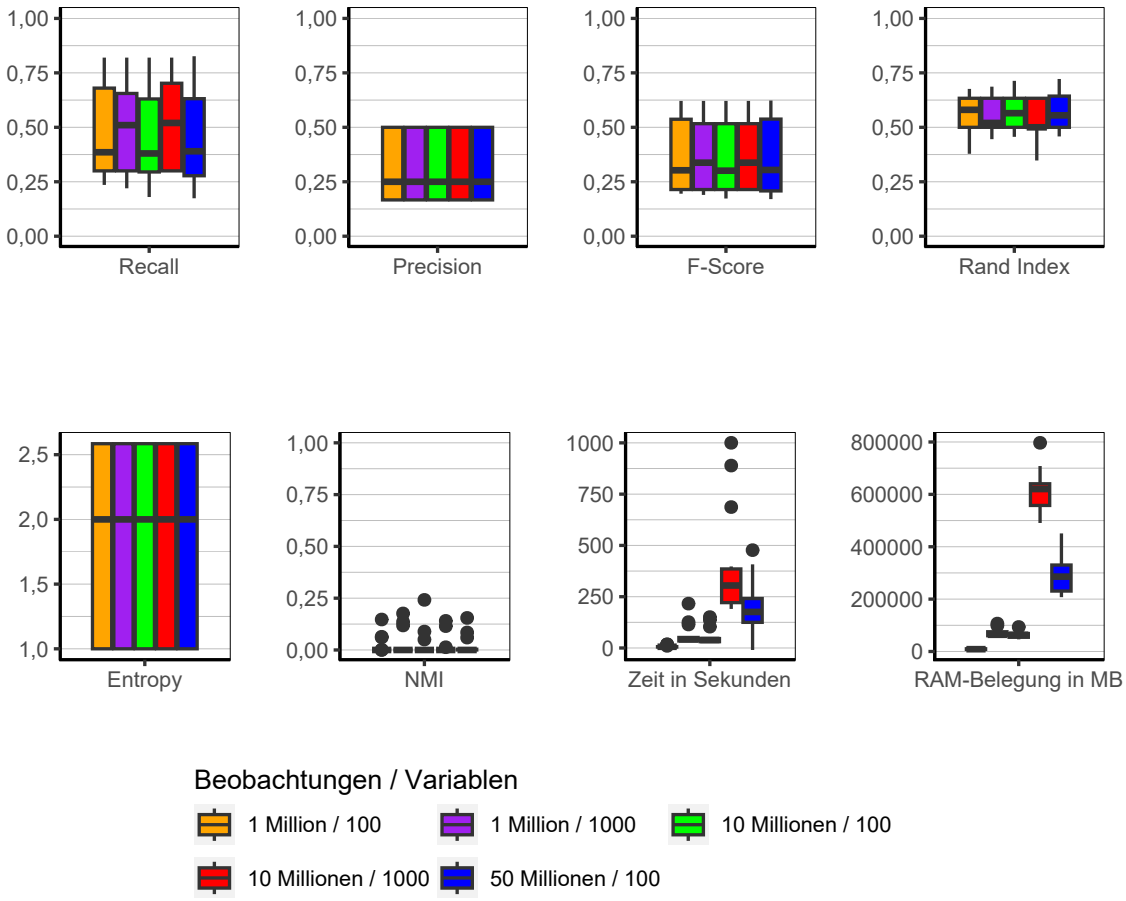


Abbildung 9.100.: General Model (Block Diagonal Variant): Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

Dimensionalität In Abbildung 9.100 ist erkennbar, dass die Werte für Recall bei den Konfigurationen mit 1.000 Variablen im Median etwas höher sind als bei denen mit 100 Variablen. Es gibt allerdings keinen Unterschied zwischen den Konfigurationen bezüglich der Werte für Precision. Insgesamt sind demnach keine großen Unterschiede in der Qualität der Clusterlösungen abhängig von der Dimensionalität der Daten zu sehen.

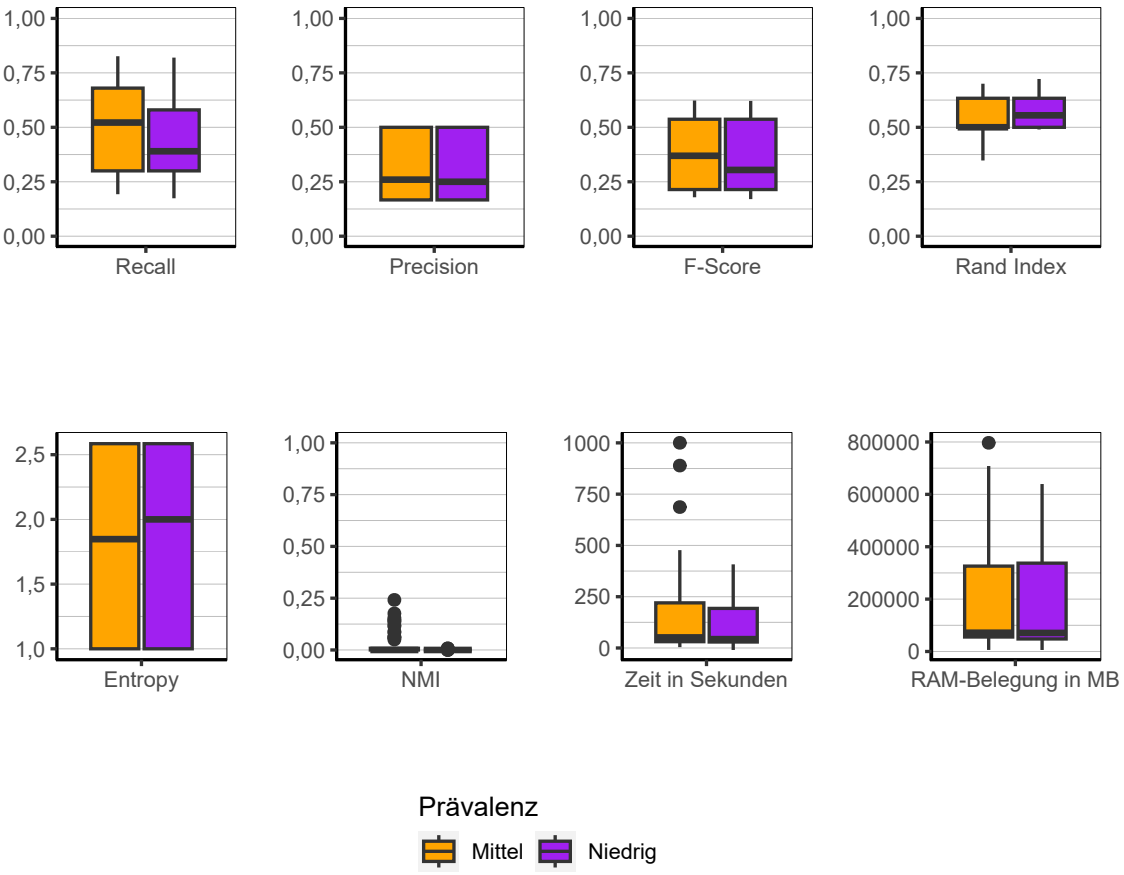


Abbildung 9.101.: General Model (Block Diagonal Variant): Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

Prävalenz Die Konfiguration mit der höheren Prävalenz (siehe Abbildung 9.101) hat etwas höhere Werte für Recall, jedoch ist wiederum kein Unterschied in der Precision zu sehen, sodass festgestellt werden muss, dass die Prävalenz keinen merklichen Einfluss hat.

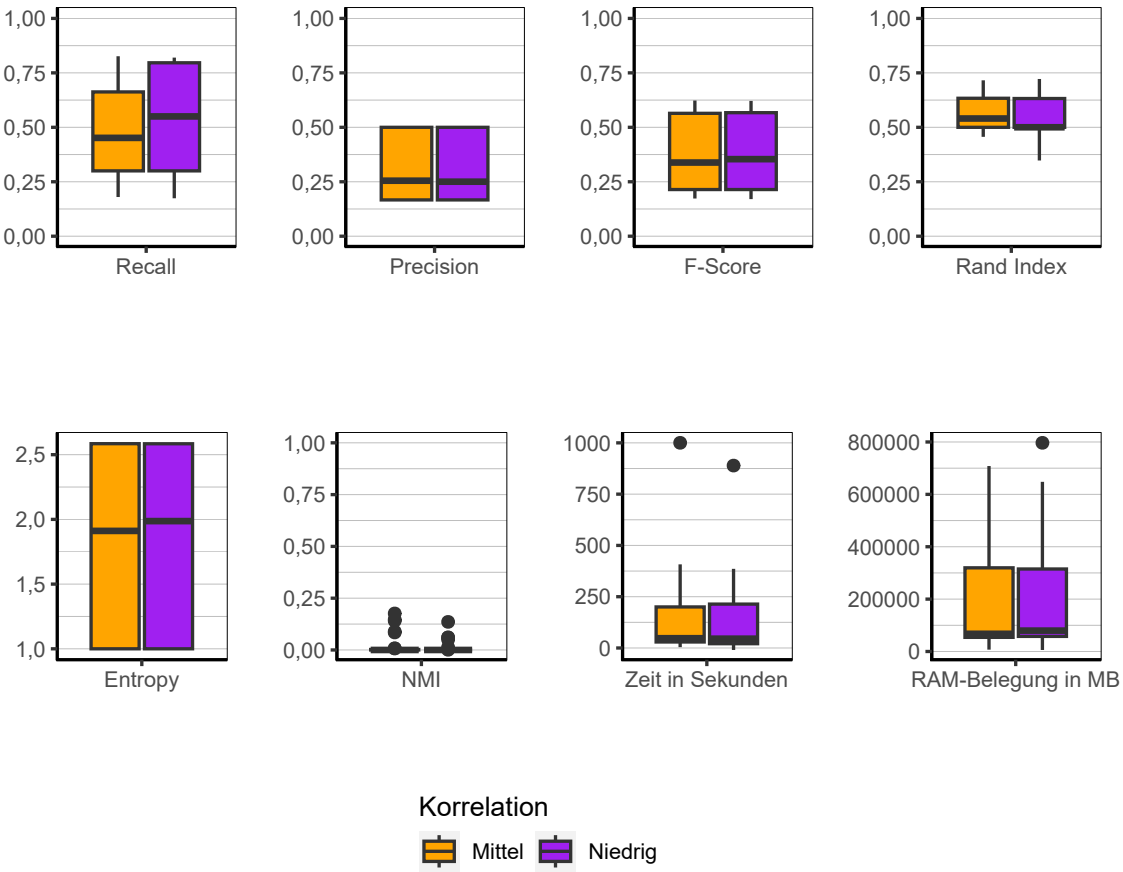


Abbildung 9.102.: General Model (Block Diagonal Variant): Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

Korrelation Dieselbe Beobachtung wie im vorherigen Absatz lässt sich wiederum bezüglich der Korrelation machen (siehe Abbildung 9.102).

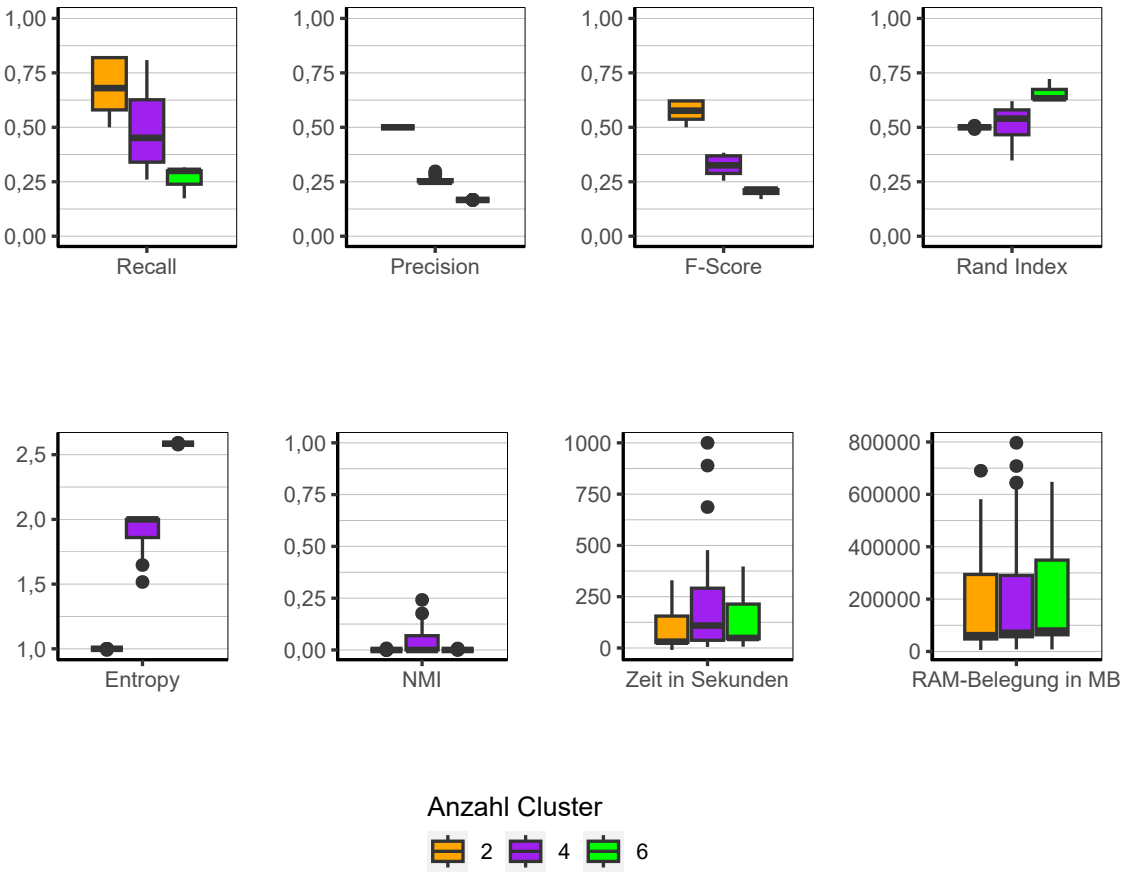


Abbildung 9.103.: General Model (Block Diagonal Variant): Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

Anzahl Cluster Bei der Anzahl der Cluster (siehe Abbildung 9.103) lässt sich das gleiche Muster erkennen, wie schon bei der Variante Two Side Clustering nur auf einem niedrigeren Niveau.

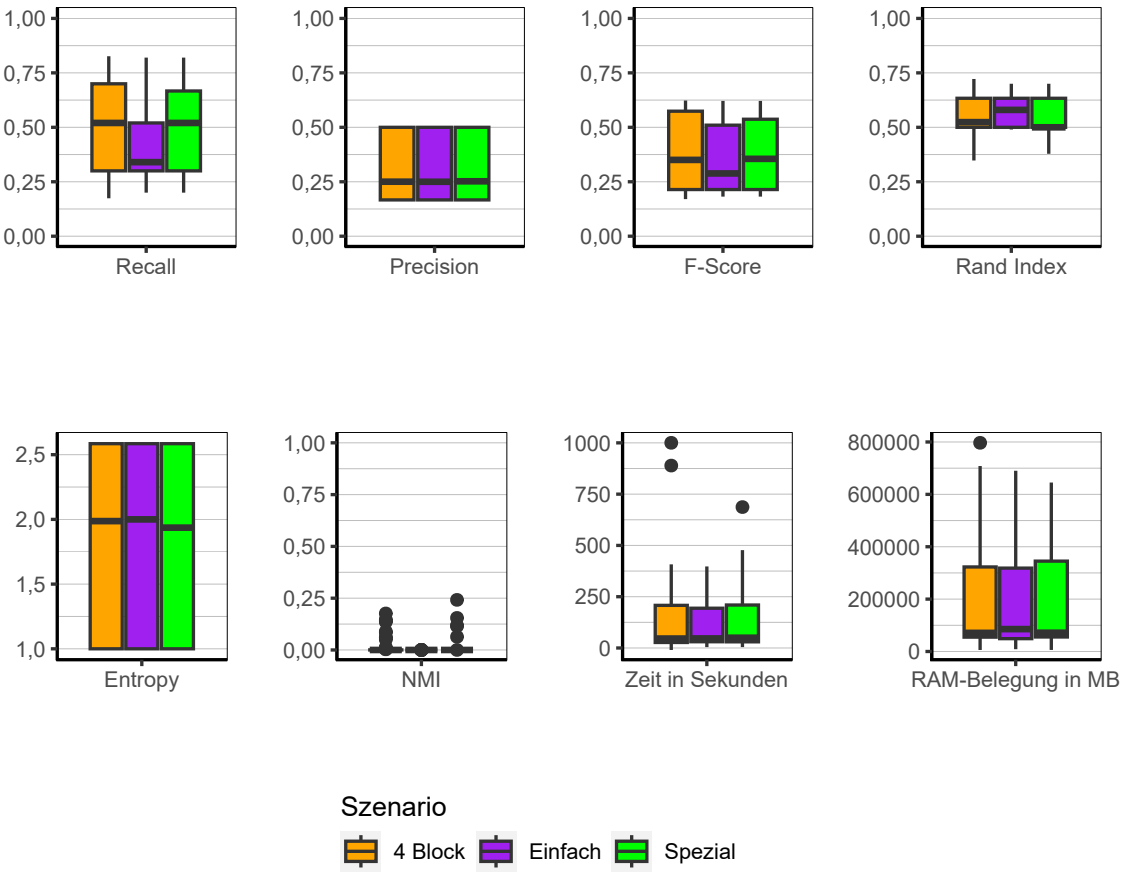


Abbildung 9.104.: General Model (Block Diagonal Variant): Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

Szenario Bezüglich des Einflusses des Szenarios lässt sich, wie in der Variante Two Side Clustering ein niedrigerer Recall-Wert bei Szenario *Einfach* beobachten. Der Precision-Wert jedoch befindet sich auf exakt demselben Niveau wie bei den anderen beiden Szenarios. Alles in allem jedoch sind kaum Unterschiede zwischen den Szenarios auszumachen (siehe Abbildung 9.104).

Fazit Dem Algorithmus kann im Wesentlichen die gleiche Performance bescheinigt werden wie der Variante Two Side Clustering. Während Two Side Clustering jedoch in einigen wenigen Fällen in der Lage war perfekte Ergebnisse zu erzielen, kann dies für die Variante Block Diagonal Variant nicht beobachtet werden. Somit handelt es sich, bezogen auf die verwendeten Daten um eine schlechtere Variante des Algorithmus.

9.3.16. Regressionsanalyse

Zusätzlich zur vorangegangenen grafischen Analyse der Ergebnisse folgt in diesem Abschnitt eine deskriptive multivariate Analyse mittels Betaregressionen. Die abhängige Variable stellt der F-Score aller Simulationsiterationen eines Algorithmus dar. Die unabhängigen Variablen sind die verschiedenen Faktoren des Simulationsaufbau. Lediglich der Faktor *Korrelation* wird nicht in die Regressionen aufgenommen. Das hat den Grund, dass diese Variable eine Multikollinearität mit der Variable *Szenario* hervorruft. Zudem stellte sich in der vorangegangenen Analyse heraus, dass die Korrelation, unabhängig des verwendeten Clusteralgorithmus, nahezu keinen Einfluss auf die Güte der Clusterlösungen ausübt. Daher findet durch das Weglassen der Variable Korrelation kein Erkenntnisverlust statt. Die Ergebnisse finden sich in den Tabellen 9.3 und 9.4. Berichtet werden die *Average Marginal Effects (AME)* (Mood, 2010, S. 75). AMEs werden so interpretiert, dass die abhängige Variable sich im Durchschnitt um den Wert der AMEs ändert, wenn die unabhängige Variable die entsprechende Ausprägung annimmt. Die Werte in den Klammern stellen die zugehörigen Standardfehler dar.

BUBBLE Wie schon in der grafischen Analyse, fällt auch hier auf, dass die Dimensionalität keinen Einfluss auf die Güte der Clusterlösungen hat. Mit höherer Prävalenz steigt der F-Score. Ein noch größeren Effekt hat das verwendete Szenario. In Szenario *Einfach* steigt der F-Score im Durchschnitt um 0,343 Punkte gegenüber Szenario *4 Block*. Mit Anstieg der Anzahl der Cluster, fällt wiederum der F-Score.

CLARA Auch hier lässt sich analog zum Algorithmus *BUBBLE* feststellen, dass die Dimensionalität keine Rolle für die Qualität der Clusterings spielt. Ebenso stellt sich die Prävalenz als wichtiger Einflussfaktor raus. Den stärksten Einfluss hat aber wiederum das Szenario *Einfach*. Mit steigender Anzahl der Cluster fällt die Qualität der Clusterings. Auch in der multivariaten Analyse zeigen sich somit dieselben Ergebnisse wie in der grafischen Betrachtung.

Tabelle 9.3.: Ergebnisse der Betaregressionen des F-Score auf die Faktoren des Designs für jeden Algorithmus

	LSH	MinHash	CLARA	CLARA	BUBBLE	MONA	Multibit Trees	CLOPE
Anzahl Beobachtungen Referenz: 1 Million								
10 Millionen	−0,003 (0,012)	−0,013 (0,009)	0,006 (0,033)	0,008 (0,032)	0,015 (0,033)	0,003 (0,003)	−0,00005 (0,001)	−0,015 (0,005)
50 Millionen	−0,013 (0,016)	−0,005 (0,011)	0,010 (0,044)	0,019 (0,042)	0,006 (0,044)		−0,0002 (0,001)	−0,008 (0,006)
Anzahl Variablen Referenz: 100								
1.000	−0,009 (0,012)	0,0004 (0,009)	0,060 (0,033)	0,060 (0,032)	0,014 (0,033)	0,009 (0,003)	−0,00002 (0,001)	−0,013 (0,005)
Prävalenz Referenz: Niedrig								
Mittel	0,352 (0,011)	0,241 (0,008)	0,193 (0,030)	0,233 (0,029)	0,193 (0,030)	0,049 (0,003)	0,001 (0,001)	−0,016 (0,004)
Szenario Referenz: 4 Block								
Einfach	0,014 (0,013)	0,075 (0,010)	0,396 (0,030)	0,353 (0,030)	0,343 (0,033)	0,017 (0,004)	0,0002 (0,001)	−0,009 (0,005)
Spezial	−0,008 (0,013)	0,004 (0,009)	−0,011 (0,040)	−0,017 (0,038)	−0,004 (0,038)	−0,002 (0,004)	−0,0004 (0,001)	0,003 (0,005)
Anzahl Cluster Referenz: Zwei								
Vier	−0,138 (0,013)	−0,027 (0,010)	−0,176 (0,036)	−0,202 (0,035)	−0,251 (0,036)	−0,248 (0,004)	−0,248 (0,001)	−0,255 (0,005)
Sechs	−0,213 (0,013)	−0,057 (0,009)	−0,359 (0,035)	−0,401 (0,034)	−0,409 (0,035)	−0,351 (0,004)	−0,330 (0,001)	−0,366 (0,005)
n	120	120	119	120	120	96	120	120
Adjusted R^2	0,926	0,894	0,453	0,550	0,498	0,9887	1,000	0,976

Tabelle 9.4.: Ergebnisse der Betaregressionen des F-Score auf die Faktoren des Designs für jeden Algorithmus

	Large Item	SCALE	Sorted Neighborhood	Proximus	Canopy	GM 1	GM 2
Anzahl Beobachtungen Referenz: 1 Million							
10 Millionen	−0,004 (0,002)	0,004 (0,024)	0,00000 (0,033)	−0,048 (0,031)	−0,045 (0,021)	0,004 (0,031)	0,002 (0,006)
50 Millionen	0,001 (0,002)	0,014 (0,030)	−0,0001 (0,043)	−0,019 (0,040)	−0,026 (0,028)	−0,014 (0,041)	−0,0002 (0,009)
Anzahl Variablen Referenz: 100							
1.000	0,004 (0,002)	0,103 (0,023)	0,00000 (0,033)	0,182 (0,030)	−0,047 (0,021)	0,037 (0,031)	0,013 (0,007)
Prävalenz Referenz: Niedrig							
Mittel	−0,001 (0,002)	0,375 (0,022)	0,120 (0,029)	0,173 (0,028)	−0,100 (0,019)	0,112 (0,028)	0,019 (0,006)
Szenario Referenz: 4 Block							
Einfach	−0,002 (0,002)	0,164 (0,022)	0,236 (0,035)	0,296 (0,030)	−0,092 (0,022)	0,276 (0,034)	−0,033 (0,007)
Spezial	−0,003 (0,002)	−0,054 (0,028)	0,00003 (0,036)	−0,018 (0,036)	0,004 (0,023)	0,009 (0,035)	−0,007 (0,007)
Anzahl Cluster Referenz: Zwei							
Vier	−0,258 (0,002)	−0,170 (0,023)	−0,408 (0,035)	−0,249 (0,033)	−0,247 (0,025)	−0,394 (0,034)	−0,251 (0,007)
Sechs	−0,369 (0,002)	−0,311 (0,025)	−0,507 (0,033)	−0,438 (0,032)	−0,388 (0,022)	−0,494 (0,033)	−0,373 (0,007)
n Adjusted R^2	120 0,996	119 0,798	120 0,469	117 0,658	120 0,838	120 0,572	120 0,958

CLARANS Die Regressionsanalyse bestätigt das Bild der grafischen Analyse, dass die drei Algorithmen *BUBBLE*, *CLARA* und *CLARANS* sehr ähnliche Ergebnisse hervorbringen. Auch hier zeigt sich nahezu kein Einfluss der Dimensionalität, mit höherer Prävalenz gelingen hochwertigere Clusterlösungen und in Szenario *Einfach* mit klarer getrennten Clusterstrukturen entstehen ebenfalls bessere Clusterlösungen als in den anderen beiden Szenarien.

CLOPE Die starke Tendenz des Algorithmus *CLOPE* alle Beobachtungen in einen Cluster zu allokieren, schlägt sich in der Regressionsanalyse darin nieder, dass die AMEs Ausprägungen nahe der 0 annehmen und damit keinen Einfluss ausüben. Lediglich mit Anstieg der Anzahl der Cluster fällt der F-Score.

Large Item Wie schon in der grafischen Betrachtung festgestellt, zeigt der Algorithmus *Large Item* nicht zuletzt wegen der funktionalen Ähnlichkeit zu *CLOPE* nahezu identische Ergebnisse in der Regressionsanalyse. Lediglich mit Anstieg der Cluster fällt der F-Score. Die restlichen AMEs unterscheiden sich lediglich auf mehreren Nachkommastellen von der 0.

SCALE Verglichen mit der grafischen Betrachtung der Ergebnisse, wird in der Regressionsanalyse nicht deutlich, dass die Ergebnisse von *SCALE* auf einem sehr viel höheren Niveau liegen, als die der anderen Clusteralgorithmen. Bezüglich der Einflussfaktoren, lassen sich jedoch die gleichen Schlussfolgerungen ziehen. Wenn 1.000 Variablen vorliegen, steigt der F-Score im Mittel um 0,103 Punkte gegenüber Konfigurationen mit 100 Variablen. Die Prävalenz übt den stärksten Einfluss aus und es können AMEs von 0,375 berichtet werden. Szenario *Einfach* führt zu qualitativ hochwertigeren Ergebnissen als die anderen beiden Szenarien. Der Einfluss ist hier jedoch nicht so stark ausgeprägt, wie z.B. bei *BUBBLE*, *CLARA* und *CLARANS*. Mit Anstieg der Anzahl der Cluster, fällt wiederum der F-Score.

Proximus Während die AMEs der Anzahl der Beobachtungen sehr niedrig sind und damit diese kaum einen Einfluss auf die Qualität der Clusterlösungen nimmt, liegen die AMEs der Anzahl der Variablen mit 0,182 verhältnismäßig hoch. Mit 1.000 Variablen werden demnach etwas bessere Ergebnisse erzielt als mit lediglich 100 Variablen. Mit AMEs von 0,173 liegt der Einfluss der Prävalenz auf einem ähnlichen Niveau. Wiederum kann bestätigt werden, dass mit einer klarer getrennten Clusterstruktur in Szenario *Einfach* auch bessere Ergebnisse erzielt werden als in den anderen Szenarien.

Sorted Neighborhood Die Analyse des Algorithmus *Sorted Neighborhood* zeigt wiederum keinen Einfluss der Dimensionalität. Eine höhere Prävalenz führt mit AMEs von 0,12 zu etwas besseren Ergebnissen. Eine klarere Clusterstruktur in Szenario *Einfach* bringt einen etwas höheren F-Score hervor. Mit Anstieg der Anzahl der Cluster, fällt der F-Score wiederum. Auch hier spiegeln die Ergebnisse somit die der vorangegangenen grafischen Analyse.

Multibit Trees Aus der grafischen Analyse des Algorithmus *Multibit Trees* ging hervor, dass der Algorithmus Clusterlösungen erstellt, die denen eines gleichverteilten Zufallsgenerators entsprechen. Entsprechend lassen sich mit Ausnahme der Anzahl der Cluster keine Einflüsse der unabhängigen Variablen auf den F-Score finden.

MONA Auch beim Algorithmus *MONA* gibt es neben der Anzahl der Cluster nahezu keinen Einfluss der unabhängigen Variablen auf den F-Score. Lediglich die Prävalenz zeigt mit AMEs von 0,049 marginal bessere Ergebnisse.

Locality Sensitive Hashing Beim *LSH* zeigt sich wiederum die Prävalenz mit AMEs von 0,352 als stärkster Einflussfaktor. Die AMEs der anderen Variablen sind nahe 0, außer denen der Anzahl der Cluster, welche das bekannte Bild zeigen, dass mit Anstieg der Anzahl der Cluster der F-Score fällt.

MinHash Lediglich die Prävalenz zeigt mit AMEs von 0,241 einen etwas höheren Einfluss auf den F-Score der Ergebnisse des Algorithmus *MinHash*. Ansonsten ist ein leichter positiver Einfluss des Szenario *Einfach* und ein leicht negativer Einfluss der Anzahl der Cluster zu beobachten.

Canopy Clustering Wie schon bei der grafischen Analyse des Algorithmus *Canopy Clustering*, lässt sich auch bei der Regressionsanalyse kaum ein Einfluss der unterschiedlichen unabhängigen Variablen ausmachen. Lediglich leichte negative Einflüsse der Dimensionalität und der Prävalenz können beobachtet werden. Die Anzahl der Cluster führt wiederum zu einem niedrigeren F-Score.

General Model (Two Side Clustering) Es zeigt sich, wie schon bei der grafischen Betrachtung der Ergebnisse, dass in Szenario *Einfach* bessere Ergebnisse erzielt werden können als in den anderen beiden Szenarien. Die Dimensionalität hat nahezu keinen

Einfluss auf die Höhe des F-Score und wiederum fällt der F-Score mit Anstieg der Anzahl der Cluster. Bei mittlerer Prävalenz steigt der F-Score im Schnitt um 0,112 Punkte.

General Model (Block Diagonal Variant) Beim letzten Algorithmus in der Analyse lassen sich wiederum kaum Einflüsse der unabhängigen Variablen feststellen. Dies ist analog zur grafischen Betrachtung. Mit Ausnahme der Anzahl der Cluster, wo mit Anstieg der Anzahl der Cluster, der F-Score fällt, befinden sich alle AMEs nahe der 0.

Fazit Durch die Betrachtung der Ergebnisse der einzelnen Betaregressionen wurden die Ergebnisse der grafischen Analyse bestätigt. Auch bei multivariater Betrachtung der einzelnen Einflussfaktoren können keine unterschiedlichen Schlussfolgerungen gegenüber der grafischen Analyse gezogen werden. Demnach geht der größte Einfluss vor allem vom Szenario aus. Die Algorithmen, die zum Teil gute bzw. akzeptable Ergebnisse hervorbringen, funktionieren am besten bei Szenario *Einfach*. Das gleiche gilt in abgeschwächter Form für die Prävalenz. Bei der höheren der beiden Konfiguration konnten meist bessere Ergebnisse erzielt werden als bei der niedrigeren Konfiguration. Die Dimensionalität spielt keine Rolle bei der Güte der Clusterlösungen. Einzige Ausnahme bildet die Anzahl der Variablen bei den Algorithmen *SCALE* und *Proximus*, die bei 1.000 Variablen etwas höhere F-Scores aufweisen können als bei 100 Variablen. Mit höherer Anzahl an Clustern fällt der F-Score bei allen Algorithmen. Die beobachteten Fitmaße (Adjusted R^2) liegen auf einem durchgehend hohen Niveau. Vor allem bei Algorithmen, die dazu tendieren viele bzw. alle Beobachtungen in einen Cluster zu allokieren, ist das Fitmaß entsprechend sehr hoch und nimmt z.B. bei CLOPE oder MONA Werte über 0,9 an.

9.3.17. Vergleich aller Algorithmen

In Abbildungen 9.105 und 9.106 finden sich Boxplots aller Algorithmen über alle Konfigurationen der Simulation zusammen. Wie sich sehen lässt, produziert der Algorithmus SCALE die mit Abstand besten Ergebnisse. An zweiter Stelle folgt mit großem Abstand der Algorithmus Proximus. An dritter Stelle folgt CLARANS. Außerdem waren unter bestimmten sehr günstigen Bedingungen die Algorithmen BUBBLE, CLARA, CLARANS, General Model: Two Side Clustering, Proximus, SCALE und Sorted Neighborhood alle in der Lage perfekte Clusterlösungen zu produzieren. Alles in allem kann jedoch mit Ausnahme von SCALE keinem Algorithmus die Fähigkeit zugesprochen werden, konstant akzeptable Clusterlösungen hervorzubringen.

Die gute Performance von SCALE wird jedoch mit dem Preis einer hohen Laufzeit erkauft, welche über alle Konfigurationen im Median und 75 % Quantil am höchsten ist. Die höchste Laufzeit generell ist bei einem Ausreißer von Sorted Neighborhood zu finden. Auch bei Proximus und MONA lassen sich Ausreißer beobachten, welche eine sehr hohe Laufzeit haben. Die schnellsten Algorithmen sind Canopy Clustering, CLARA, General Model: Two Side Clustering, General Model: Block Diagonal Variant, Multibit Trees und MinHash.

Der Speicherbedarf ist vor allem bei CLARANS, General Model: Two Side Clustering, General Model: Block Diagonal Variant, Locality Sensitive Hashing, Multibit Trees und Sorted Neighborhood hoch. Der höchste ist bei CLARANS zu beobachten. Hingegen benötigen BUBBLE, CLARA, CLOPE, Large Item und SCALE nur wenig Speicher.

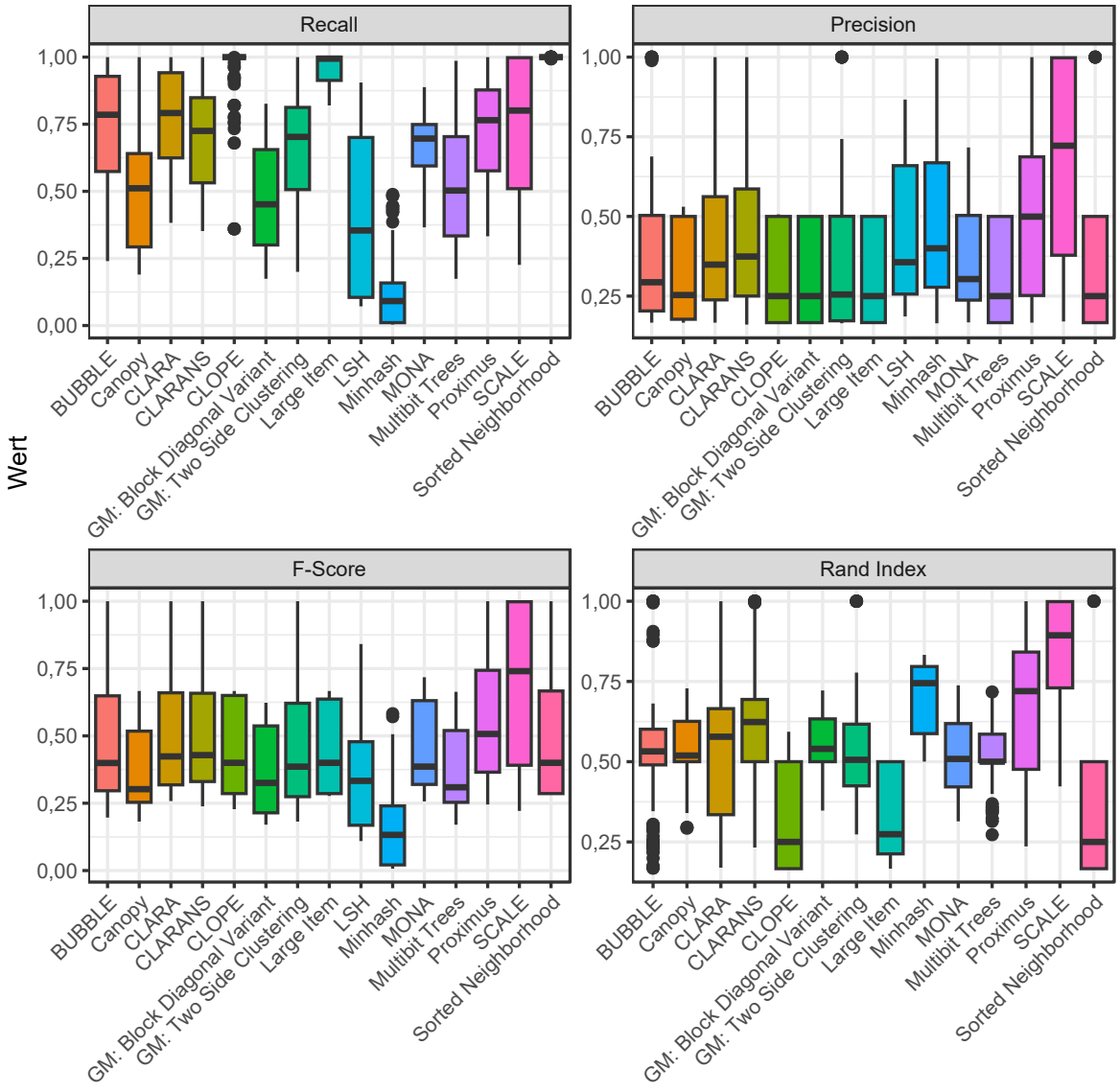


Abbildung 9.105.: Vergleich aller Algorithmen 1

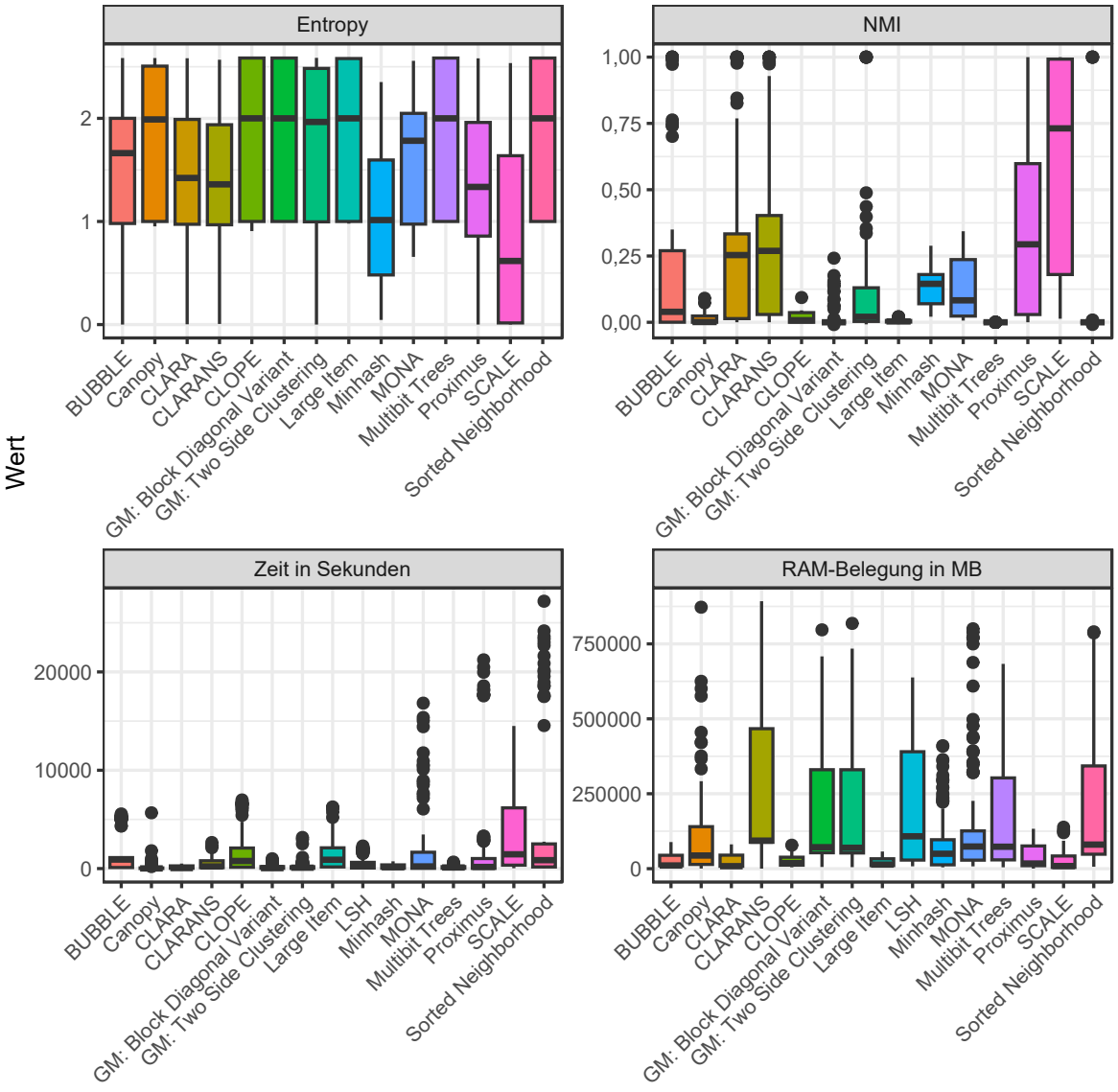


Abbildung 9.106.: Vergleich aller Algorithmen 2

10. Fazit

Nachfolgend wird das Fazit dieser Arbeit gezogen. Dabei wird auf die wichtigsten Ergebnisse eingegangen, sowie Limitationen dieser Simulationsstudie und mögliche Kritikpunkte. Abschließend wird kurz aufgeführt, wo zukünftige Forschungsarbeiten ansetzen könnten, um offene Fragen zu klären.

10.1. Wichtigste Ergebnisse

Die Ergebnisse dieser Simulationsstudie müssen insgesamt als ernüchternd zusammengefasst werden. Lediglich ein Algorithmus (SCALE) war in der Lage über weite Strecken akzeptable Clusterlösungen zu produzieren. Zwar waren die meisten Algorithmen in der Lage sehr gute bzw. perfekte Clusterlösungen bei sehr klar getrennter Clusterstruktur mit wenig Rauschen zu produzieren, jedoch sind die meisten Clusterlösungen vor allem für Daten mit nicht mehr so eindeutig getrennter Clusterstruktur als schlecht anzusehen. Selbst der Algorithmus SCALE, welcher die mit Abstand besten Ergebnisse lieferte, kann aufgrund seiner hohen Laufzeit nicht uneingeschränkt empfohlen werden. Ist die Laufzeit von Relevanz, bietet sich der Algorithmus Proximus an, der zwar in den meisten Fällen schneller läuft, jedoch auch qualitativ schlechtere Ergebnisse liefert. Wird zusätzlich noch ein geringerer Speicherbedarf benötigt, stellt BUBBLE eine Alternative dar, welche zwar wiederum etwas schlechtere Ergebnisse hervorbringt als Proximus, dafür aber relativ schnell läuft und wenig Speicher benötigt.

Wie aus vorherigem Absatz bereits hervorgeht, hatte das Szenario den stärksten Einfluss von allen Faktoren des Simulationsaufbaus. Das heißt, in Szenario *Einfach* mit klarer Clusterstruktur konnten die besten Ergebnisse erzielt werden. Je weniger Cluster vorhanden waren, desto besser waren die Ergebnisse. Die besten Clusterlösungen wurden von allen Algorithmen hervorgebracht, wenn lediglich 2 Cluster vorhanden waren. Die Korrelation spielte überwiegend überhaupt keine Rolle. Eine hohe Prävalenz bot einigen Algorithmen deutliche ausgeprägtere Cluster, weshalb hier zumeist ebenfalls qualitativ

hochwertigere Clusterlösungen produziert werden konnten. Die Anzahl der Beobachtungen wirkte sich im Großen und Ganzen nur auf die Laufzeit und den Speicherbedarf aus. Mit steigender Anzahl der Variablen konnten sowohl bessere als auch schlechtere Ergebnisse beobachtet werden. Oftmals hatte die Anzahl der Variablen neben der ebenfalls erhöhten Laufzeit und des Speicherbedarfs keine direkte Auswirkung.

10.2. Limitationen

Es muss jedoch nochmals darauf hingewiesen werden, dass die simulierten Daten, vor allem in den Szenarien *4 Block* und *Spezial*, eine beträchtliche Menge an Rauschen enthalten. Somit können viele Beobachtungen aufgrund ihrer Ausprägungen nicht eindeutig den jeweiligen Clustern zugeordnet werden. Obgleich bei der visuellen Inspizierung der Daten (siehe Abschnitt 5.6) die Cluster eindeutig erkennbar sind, fällt auch die Menge an Rauschen in den Daten auf und die entsprechenden Beobachtungen können nur anhand ihrer Anordnung im Datensatz ihrem jeweiligen Cluster zugeordnet werden.[1]

Umso erstaunlicher sind die zum Teil guten Ergebnisse des Algorithmus SCALE zu bewerten. Wobei auch dieser die konstant besten Ergebnisse im Szenario *Einfach* liefert, in welchem die Cluster am deutlichsten getrennt sind und am wenigsten Rauschen vorhanden ist. Trotz seiner strukturellen Ähnlichkeit zu den Algorithmen CLOPE und Large Item schneidet SCALE qualitativ wesentlich besser ab. Ein möglicher Grund dafür könnte in der Anwendung des *WCD clustering preparation steps* sein (siehe Abschnitt 7.10), welcher bei CLOPE und Large Item nicht durchgeführt wird.

Was die Evaluation der Laufzeit und des Speicherbedarfs angeht, muss berücksichtigt werden, dass die Algorithmen in der Programmiersprache `R` implementiert wurden. Bei `R` handelt es sich um eine interpretierte Sprache, die nicht auf Effizienz ausgelegt ist. Daher könnte die Laufzeit und der Speicherbedarf sicherlich mit einer Sprache wie `C` besser evaluiert werden.

Zusätzlich ist es natürlich, wie in Abschnitt 7.16 beschrieben, denkbar, dass bei der Programmierung der Programme und vor allem bei der Implementierung der Algorithmen selbst Programmierfehler unterlaufen sind. Zwar wurde die Funktionsfähigkeit der

[1] Natürlich wurde die Anordnung der Beobachtungen vor der Anwendung der Clustermethoden permutiert (siehe Abschnitt 5.6).

Algorithmen weitestgehend überprüft, jedoch sind Fehler nie komplett auszuschließen, was die Ergebnisse kompromittieren kann.

Ebenfalls wurden die Algorithmen nach Möglichkeit parallelisiert, auch wenn sich die Problemstellung nicht als embarrassingly parallel herausstellte. Falls in der Literatur algorithmusspezifische Parallelisierungen publiziert sind, wurden diese implementiert. In Fällen, in denen keine Parallelisierungen in der Literatur gefunden wurden, wurden Parallelisierungsstrategien verwendet, die nach Möglichkeit algorithmusspezifische Eigenheiten verwendeten, um eine sinnvolle Parallelisierung zu gewährleisten. Dies stellt eine der zentralen Neuerungen dieser Arbeit dar. Als Beispiel sei hier die Verwendung der dominanten Patterns bei Proximus genannt, welche als Teil der Lösung der prozessorkernspezifischen Clusterings aus der parallelen Bearbeitung des Datensatzes ergeben. Diese wurden dann als Grundlage für die Zusammenführung der Ergebnisse der einzelnen Tasks verwendet. Die Tatsache, dass sich die Algorithmen nicht als embarrassingly parallel beschreiben lassen, führt zu einem Kompromiss zwischen Geschwindigkeit und Güte der Clusterlösung. Demnach wäre es denkbar, dass die sequenziellen Varianten qualitativ bessere Clusterlösungen hervorbringen könnten, welche sich allerdings durch eine gegebenenfalls massiv erhöhte Laufzeit erkauft werden müsste. Jedoch konnte dieser Qualitätsvorteil der sequenziellen Varianten in einem vorgelagerten Experiment nicht beobachten werden (siehe Abschnitt 9.2).

Letztlich muss die automatisierte Parameterwahl für die einzelnen Algorithmen als Kritikpunkt erwähnt werden. Im Einzelfall sollten diese idealerweise anwendungsbezogen manuell gewählt werden. Dabei sollten mehrere Optionen evaluiert und die für das konkrete Analysevorhaben sinnvollsten Parameter gewählt werden. Dies ist in einer Simulationsstudie wie dieser hier nicht möglich, da zu viele Faktorstufen im Simulationsaufbau existieren. Der nötige Aufwand um die perfekten Parameter für jede einzelne Iteration der Simulation für jeden Algorithmus manuell zu wählen ist immens und nicht verhältnismäßig. Darüber hinaus ist in der Beschreibung der Algorithmen oftmals keine sinnvolle Vorgabe für die Wahl der Parameter angegeben. Demnach musste die Spannweite der möglichen Parameter, aus der in jeder Iteration automatisiert gewählt wurde (siehe Abschnitt 9.1), auf der Grundlage der Ergebnisse einiger Probeanalysen mit kleineren Beispieldatensätzen erstellt werden.

10.3. Weiterführende Forschung

Aufgrund der Tatsache, dass die Beobachtungen, welche als Rauschen betrachtet werden können, in dieser Studie nicht eindeutig identifiziert werden können, wäre es interessant zu evaluieren, wie sich die einzelnen Algorithmen verhalten, wenn die als Rauschen betrachteten Beobachtungen nicht bei der Evaluation der Clustergüte berücksichtigt werden würden.[2] Zwar könnte dann unter diesen Umständen die Korrelation als Faktor im Simulationsaufbau nicht mehr berücksichtigt werden, jedoch zeigten die Ergebnisse dieser Simulationsstudie, dass die Korrelation nur eine untergeordnete bzw. gar keine Rolle bei der Qualität der Clusterlösungen spielt. Entsprechend müssten keine Constraints bezüglich der Korrelation berücksichtigt werden, was auch die Simulation der Daten vereinfacht.

Bezüglich Laufzeit und Speicherbedarf wäre es sicherlich interessant diese Studie, gegebenenfalls mit noch höherer Dimensionalität und mehr Faktorstufen generell, in einer Sprache wie `C/C++` oder `Fortran` auf einem leistungsstärkerem HPC-System zu replizieren. Darüber hinaus wurde in dieser Arbeit lediglich eine geringe Anzahl an Cluster evaluiert. Dies Entscheidung wurde gewählt, da in den Sozialwissenschaften meist nur wenige Cluster zu erwarten und von Relevanz sind. Demnach könnte eine Simulationsstudie mit vielen bzw. sehr vielen Clustern diese Lücke schließen.

Des Weiteren fällt bei der Betrachtung der Optimierungskriterien von CLOPE und Large Item auf, dass beide auf die Existenz von Spaltensummen angewiesen sind die 0 ergeben. Im Falle von CLOPE kann das Histogramm eines Clusters nicht mehr an Breite zunehmen, wenn bereits alle Spaltensummen größer 0 sind und damit das Histogramm die maximale Breite angenommen hat. Das Optimierungskriterium ist dann nicht mehr sensitiv gegenüber einzelnen Beobachtungen und der Algorithmus tendiert dazu alle folgenden Beobachtungen in denselben Cluster zu allokieren. Das gleiche Phänomen tritt im Falle von Large Item auf, wenn bereits alle Items eines Clusters als Large Item oder Small Item definiert sind, was der Fall ist wenn alle Spaltensummen eines Clusters einen Wert größer 0 annehmen. Eine einzelne Beobachtung hat dann keinen Einfluss mehr auf die Anzahl von Large Items oder Small Items. Dies bedeutet, dass beide Algorithmen nur für Datensätze geeignet sind, die eine Vielzahl an Nullen aufweisen und damit erwartet

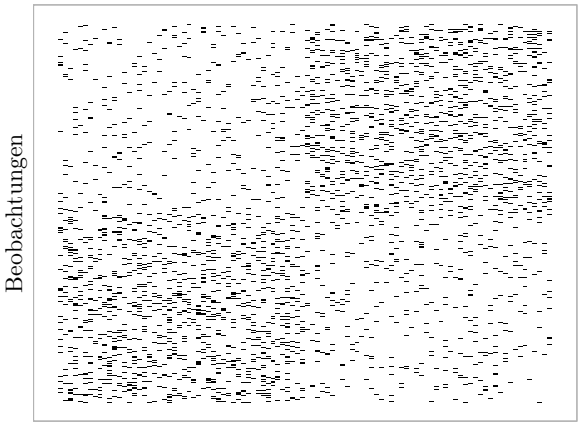
[2] Die als Rauschen betrachteten Beobachtungen werden von der Simulationsmethode erstellt, um die nötigen Constraints bezüglich Korrelation und Mittelwerte einzuhalten, sind somit aber Teil des entstehenden Clusters, auch wenn sie nicht innerhalb diesem lokalisiert sind (siehe auch Abschnitt 5.6). Dies ist vergleichbar mit einer Person, welche zum Cluster einer Familie gehört, jedoch grundverschiedene Eigenschaften mit sich bringt.

werden kann, dass innerhalb der Cluster genug Spaltensummen gleich 0 existieren. Zukünftige Forschung, die eine Datenbasis verwendet, die diese Eigenschaft sicherstellt und erneut CLOPE und Large Item in einer groß angelegten Simulationsstudie verwendet, wäre daher wünschenswert.

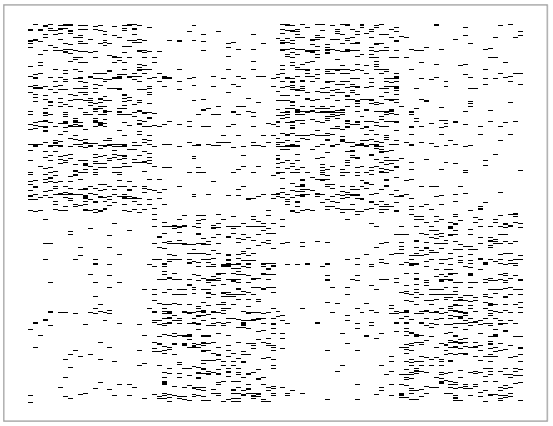
Der vorangegangene Punkt macht deutlich, dass die Wahl eines geeigneten Clusteralgorithmus stark von den Eigenschaften der zugrundeliegenden Datensätze abhängt. Diese können sich je nach inhaltlichem Themenbereich unterscheiden. Entsprechend wäre es interessant die Performance der Algorithmen anhand von Daten zu evaluieren, die so erstellt worden sind, dass sie den üblicherweise empirisch anzutreffenden Daten in unterschiedlichen thematischen Bereichen ähneln.

A. Visualisierung der simulierten Daten mit niedriger Prävalenz

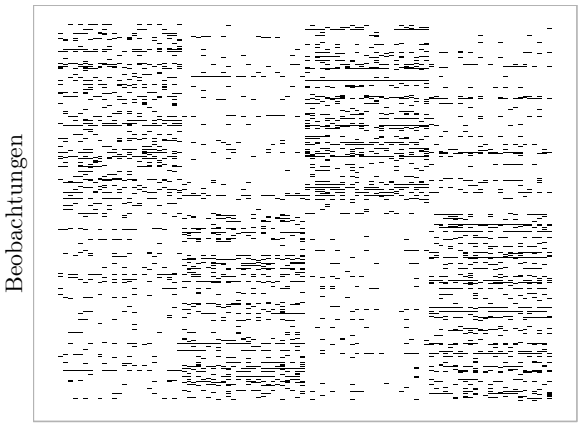
n = 1000
Prävalenz = low
n Cluster = 2
Szenario = einfach
Korrelationsstruktur = N/A



n = 1000
Prävalenz = low
n Cluster = 2
Szenario = 4block
Korrelationsstruktur = low

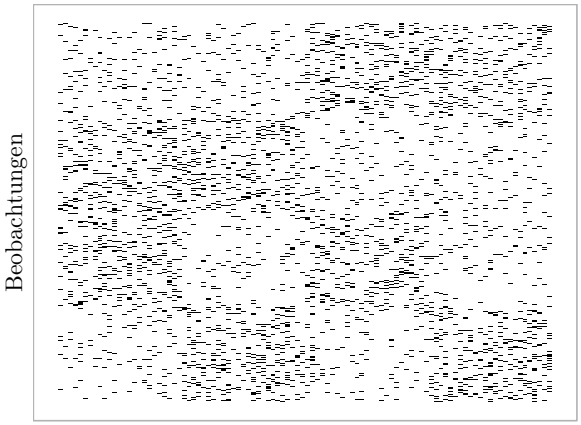


n = 1000
Prävalenz = low
n Cluster = 2
Szenario = 4block
Korrelationsstruktur = mid



n = 1000
Prävalenz = low
n Cluster = 2
Szenario = special
Korrelationsstruktur = N/A

n = 1000
Prävalenz = low
n Cluster = 4
Szenario = einfach
Korrelationsstruktur = N/A



n = 1000
Prävalenz = low
n Cluster = 4
Szenario = 4block
Korrelationsstruktur = low

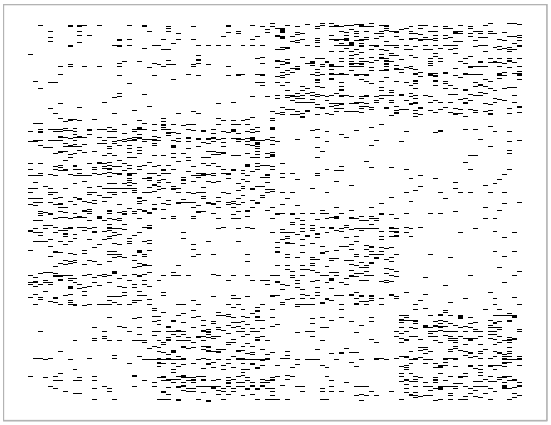
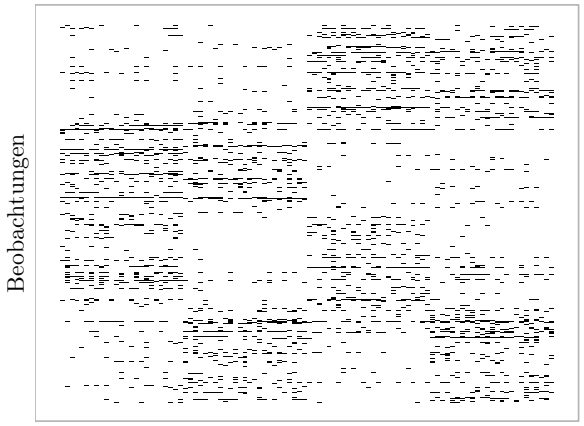
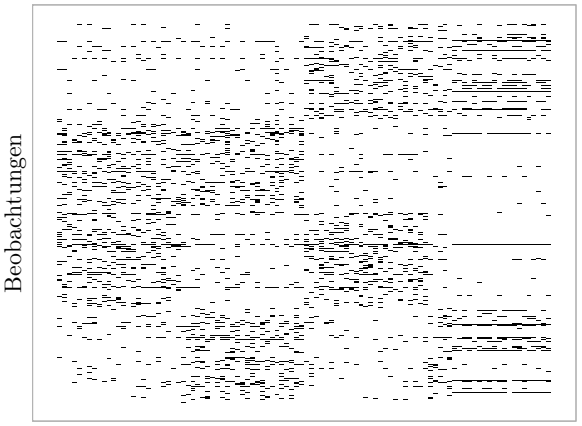


 Abbildung A.1.: Visualisierung der simulierten Daten mit niedriger Prävalenz 1

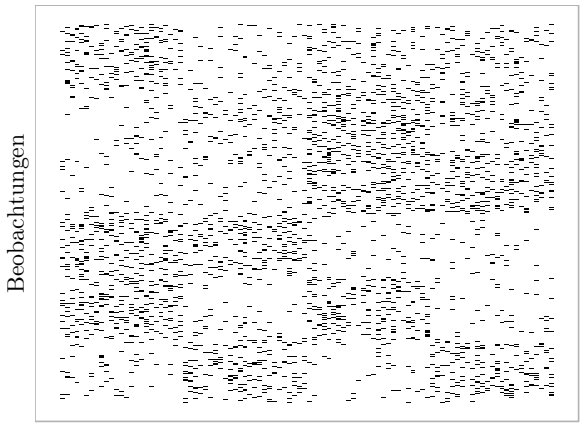
n = 1000
Prävalenz = low
n Cluster = 4
Szenario = 4block
Korrelationsstruktur = mid



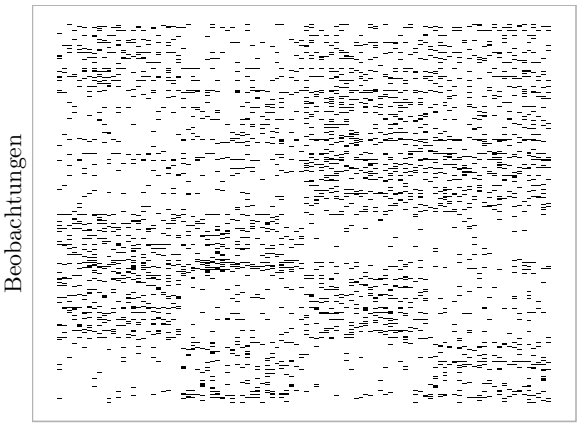
n = 1000
Prävalenz = low
n Cluster = 4
Szenario = special
Korrelationsstruktur = N/A



n = 1000
Prävalenz = low
n Cluster = 6
Szenario = einfach
Korrelationsstruktur = N/A



n = 1000
Prävalenz = low
n Cluster = 6
Szenario = 4block
Korrelationsstruktur = low



n = 1000
Prävalenz = low
n Cluster = 6
Szenario = 4block
Korrelationsstruktur = mid

n = 1000
Prävalenz = low
n Cluster = 6
Szenario = special
Korrelationsstruktur = N/A

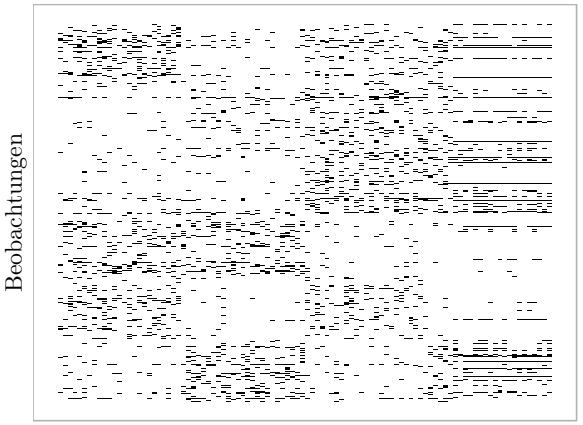


Abbildung A.2.: Visualisierung der simulierten Daten mit niedriger Prävalenz 2

B. Ergebnisplots Canopy Clustering Sequenziell

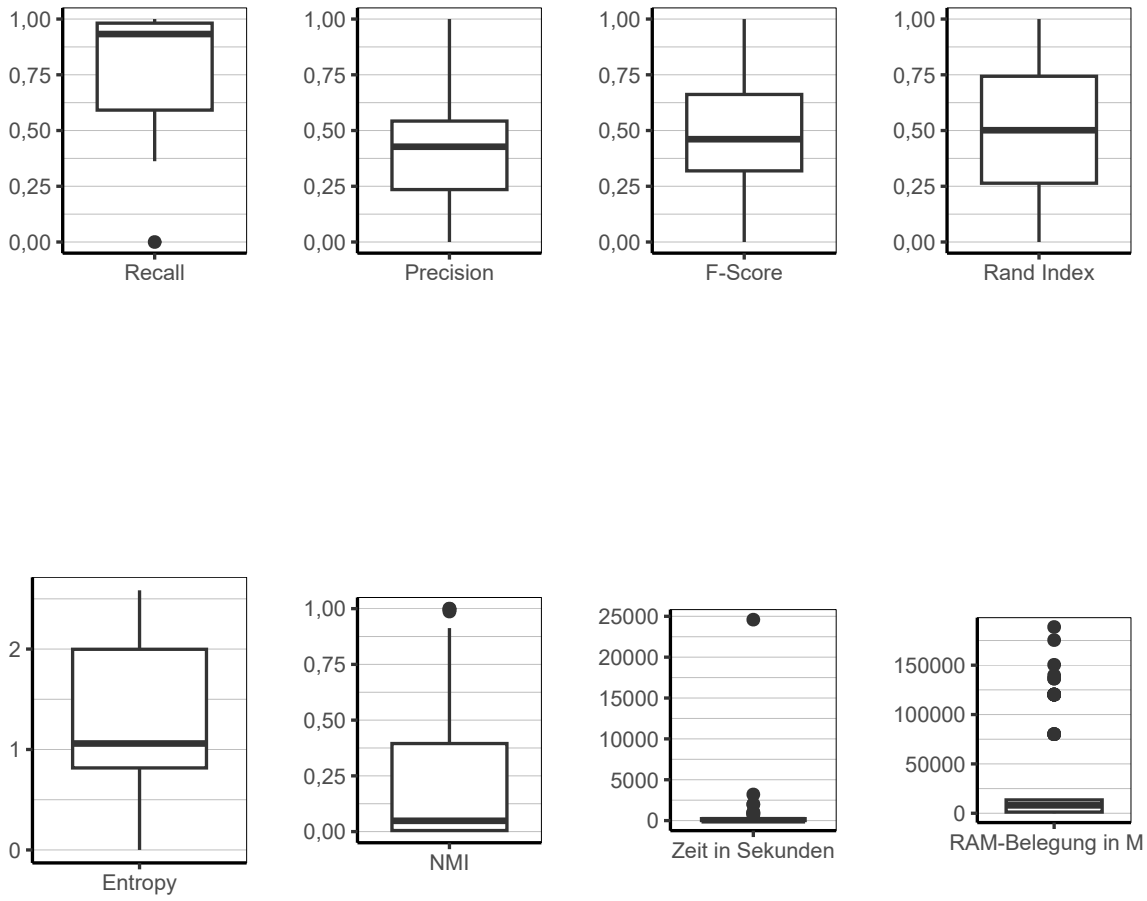


Abbildung B.1.: Canopy Clustering Sequenziell: Evaluation aller Iterationen der Simulation insgesamt.

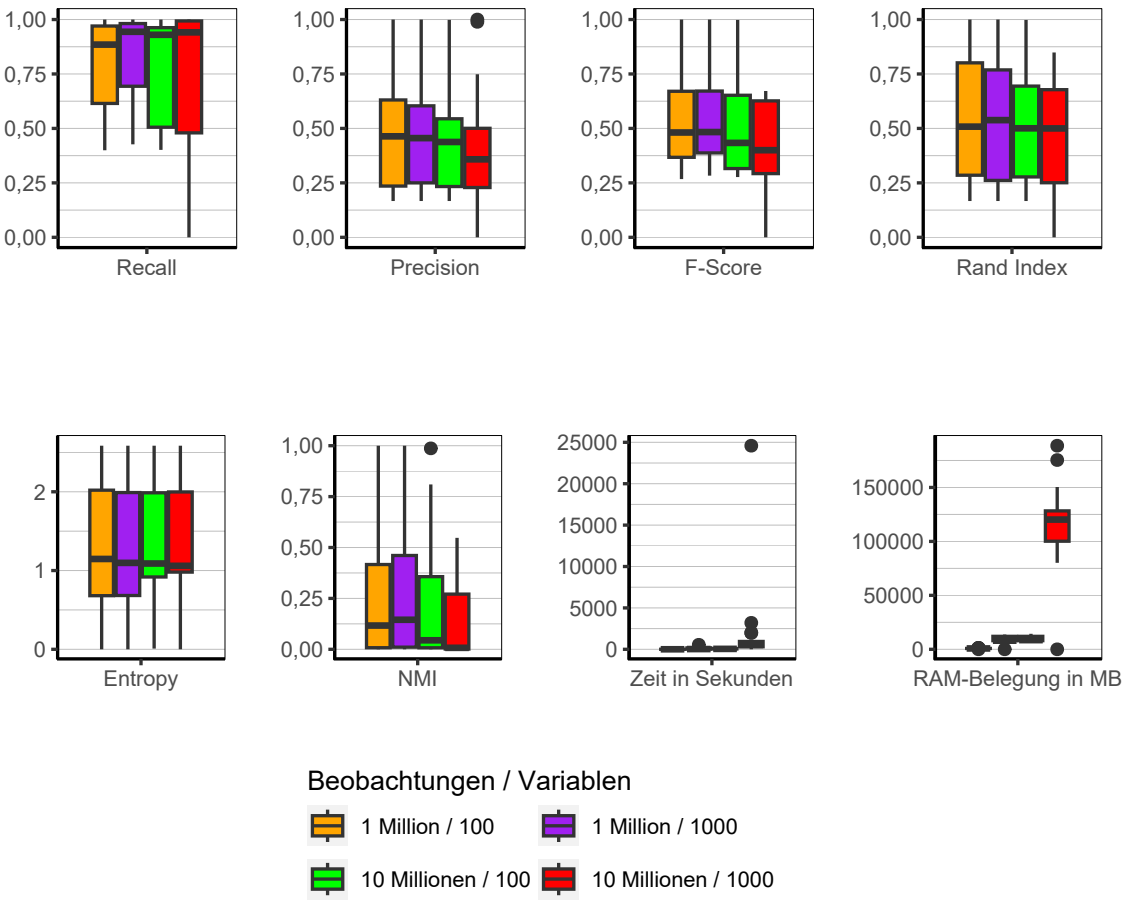


Abbildung B.2.: Canopy Clustering Sequenziell: Evaluation des Einflusses der Dimensionalität über alle Iterationen der Simulation.

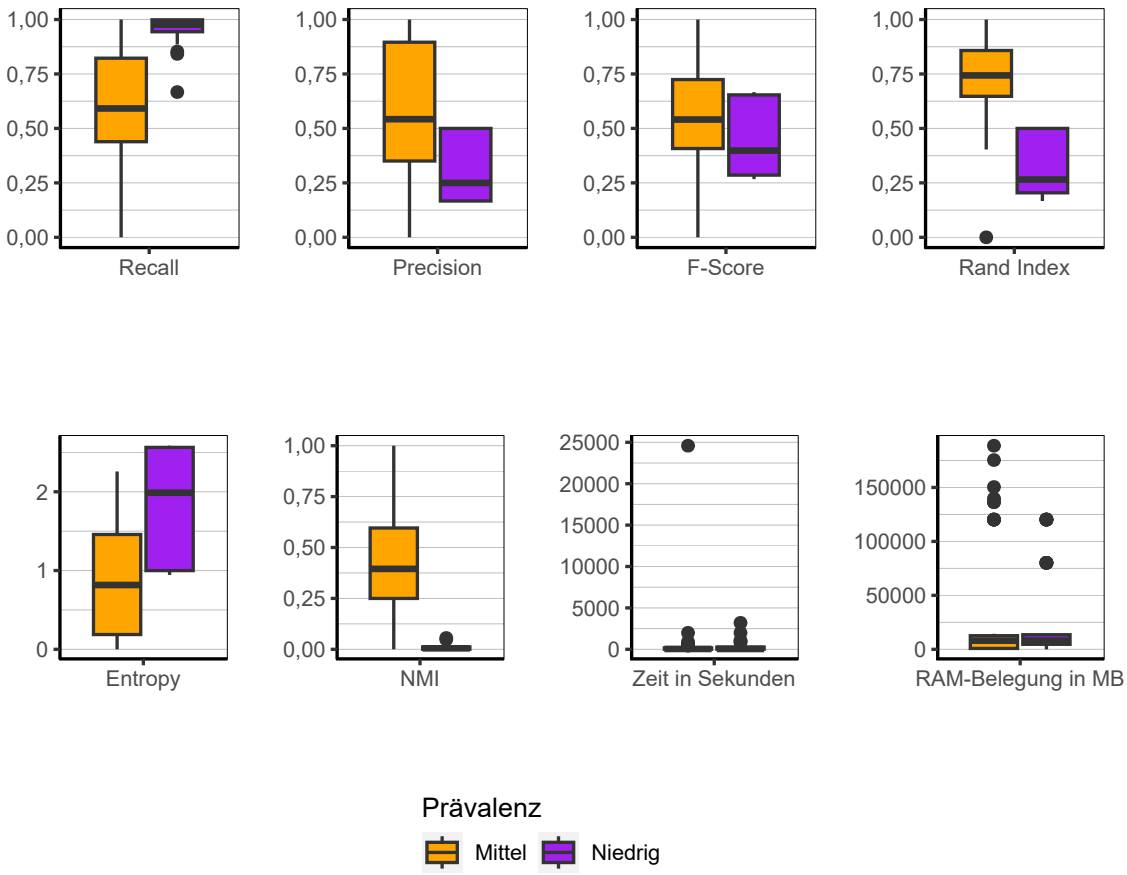


Abbildung B.3.: Canopy Clustering Sequenziell: Evaluation des Einflusses der Prävalenz über alle Iterationen der Simulation.

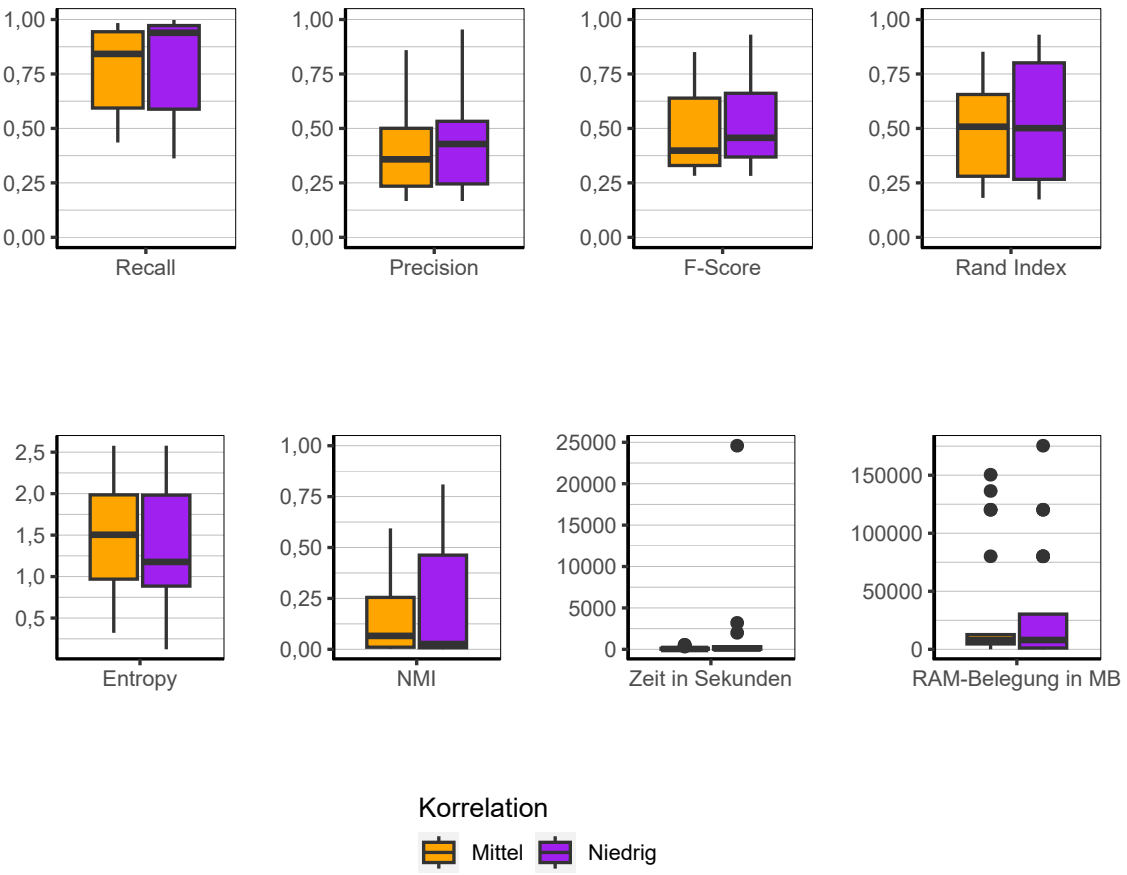


Abbildung B.4.: Canopy Clustering Sequenziell: Evaluation des Einflusses der Korrelation über alle Iterationen der Simulation.

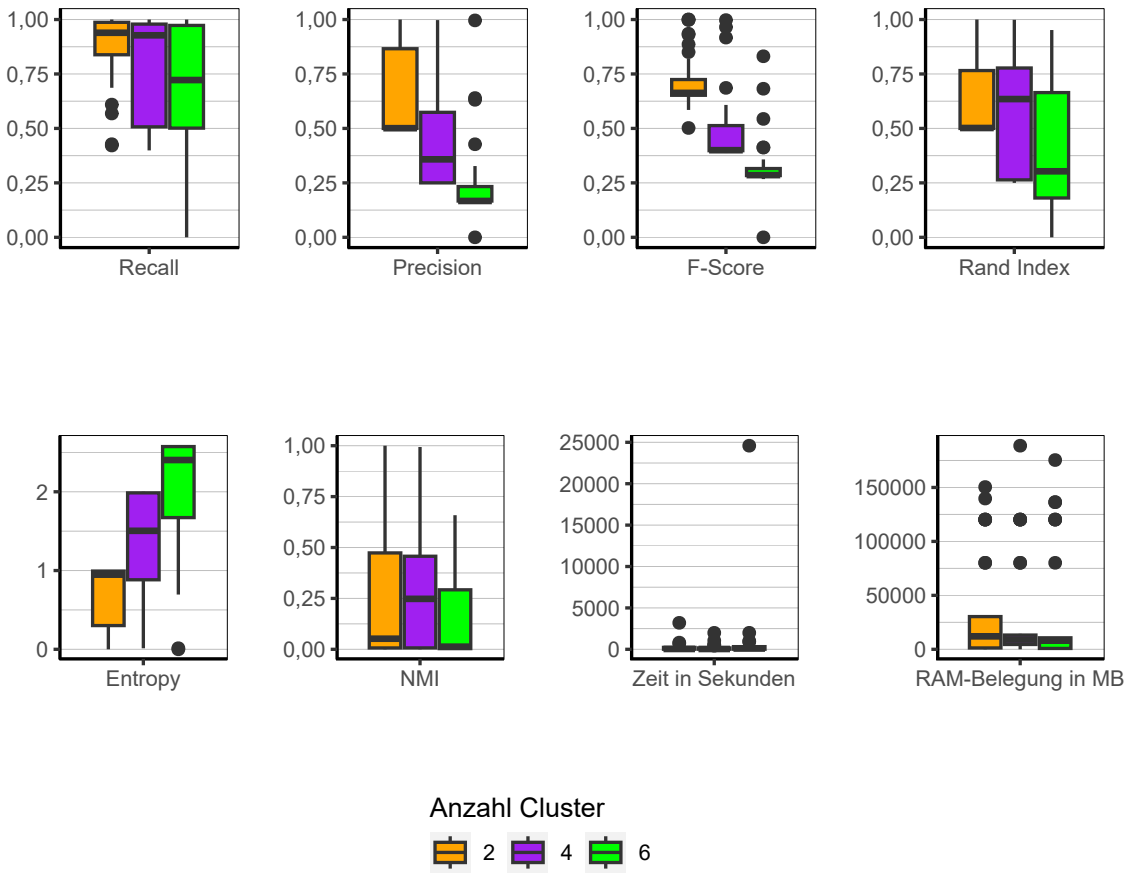


Abbildung B.5.: Canopy Clustering Sequenziell: Evaluation des Einflusses der Anzahl der Cluster über alle Iterationen der Simulation.

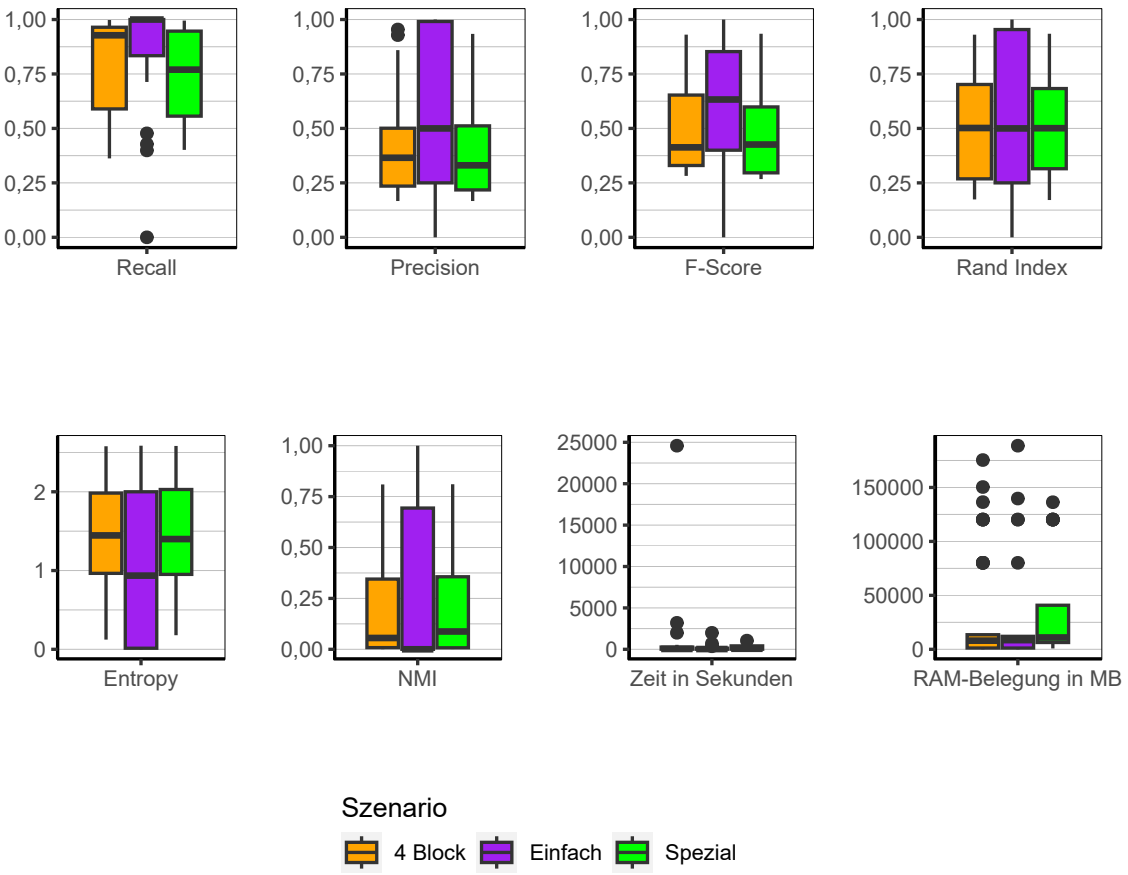


Abbildung B.6.: Canopy Clustering Sequenziell: Evaluation des Einflusses des Szenario über alle Iterationen der Simulation.

C. R-Code

C.1. Simulation der Daten

```
library(psych)
library(Matrix)
library(matrixcalc)
library(data.table)
library(foreach)
library(mvnfast)
if (!require("doMC")) {
  library(doFuture)
}
library(reshape2)
library(ClusterR)
library(cluster)
library(rdist)
library(matrixStats)
library(bigtabulate)
library(aricode)
library(rlist)
library(stringr)
library(itertools)
library(parallel)
library(dplyr)
library(parallelDist)
library(e1071)
library(doRNG)
library(fastdigest)
library(RcppAlgos)
library(primes)
library(RhpcBLASctl)

corrbinary <- function(n, means, sigma, sigma_correction = TRUE,
 hashs = NULL, cores) {
```

```
ms <- means
nvars <- length(means)

nearPSD <- function(c) {

  n <- dim(c)[1]
  e <- eigen(c, sym = TRUE)
  val <- e$values * (e$values > 0)
  vec <- e$vectors
  T <- sqrt(diag(as.vector(1/(vec^2 %*% val)), n, n))
  B <- T %*% vec %*% diag(as.vector(sqrt(val)), n, n)
  out <- B %*% t(B)

  return(out)
}


sigma.star_fun <- function(ms, sigma, cores) {
  no.bin <- length(ms)
  p <- ms

  result <- foreach(i = 1:no.bin, .packages = c("psych", "foreach"),
   .combine = rbind) %dopar%  {
    foreach(j = 1:no.bin, .combine = c) %do% {
      suppressMessages(phi2tetra(sigma[i, j], c(p[i], p[j])))
    }
  }

  if (!all(diag(result) == 1)) {
    diag(result) <- 1
  }

  if (!is.symmetric.matrix(result)) {
    result <- as.matrix(forceSymmetric(result, uplo = "L"))
  }


  if (!is.positive.semi.definite(result)) {
    result <- nearPSD(result)
  }
```

```
  result <- (result + t(result)) / 2
  return(result)
}



if (sigma_correction) {

  # Korrelationen die die Grenzen ueber- bzw. unterschreiten,
  # werden auf den maximal bzw. minimal zulaessigen Wert gesetzt.

  # setup zulaessige Korrelationen
  lows <- matrix(nrow = nvars, ncol = nvars)
  highs <- matrix(nrow = nvars, ncol = nvars)
  p <- ms
  q <- 1 - p
  for (i in 1:length(p)) {
    for (j in 1:length(p)) {
      if (i == j) {
        lows[i, j] <- 1L
        highs[i, j] <- 1L
      } else {
        lows[i, j] <- max(-sqrt((p[i] * p[j])/(q[i] * q[j])),
         -sqrt((q[i] * q[j])/(p[i] * p[j])))
        highs[i, j] <- min(sqrt((p[i] * q[j])/(q[i] * p[j])),
         sqrt((q[i] * p[j])/(p[i] * q[j])))
        sigma[i, j] <- ifelse(sigma[i, j] < lows[i, j], lows[i, j],
         ifelse(sigma[i, j] > highs[i, j], highs[i, j], sigma[i, j]))
      }
    }
  }
} else {
  toolowstop <- 0
  toohighstop <- 0
  toolow <- data.table()
  toohigh <- data.table()

  # setup zulaessige Korrelationen
  lows <- matrix(nrow = nvars, ncol = nvars)
  highs <- matrix(nrow = nvars, ncol = nvars)
  p <- ms
  q <- 1 - p
```

```
for (i in 1:length(p)) {
  for (j in 1:length(p)) {
    if (i == j) {
      lows[i, j] <- 1L
      highs[i, j] <- 1L
    } else {
      lows[i, j] <- max(-sqrt((p[i] * p[j])/(q[i] * q[j])),
       -sqrt((q[i] * q[j])/(p[i] * p[j])))
      highs[i, j] <- min(sqrt((p[i] * q[j])/(q[i] * p[j])),
       sqrt((q[i] * p[j])/(p[i] * q[j])))
      if (sigma[i, j] < lows[i, j]) {
        toolow <- rbindlist(list(toolow, data.table(row = i,
         column = j, lower.bound = lows[i, j])))
        toolowstop <- 1L
      }
      if (sigma[i, j] > highs[i, j])  {
        toohigh <- rbindlist(list(toohigh, data.table(row = i,
         column = j, upper.bound = highs[i, j])))
        toohighstop <- 1L
      }
    }
  }
}
if (toolowstop == 1 & toohighstop == 0) {
  setDF(toolow)
  print(toolow)
  stop("There were correlations outside the acceptable range.
   Check output for positions and bounds that were exceeded.")
}
if (toohighstop == 1 & toolowstop == 0) {
  setDF(toohigh)
  print(toohigh)
  stop("There were correlations outside the acceptable range.
   Check output for positions and bounds that were exceeded.")
}
if (toohighstop == 1 & toolowstop == 1) {
  setDF(toohigh)
  setDF(toolow)
  print(toohigh)
  print(toolow)
  stop("There were correlations outside the acceptable range.
  Check output for positions and bounds that were exceeded.")
```

```
  }
}

if (!is.positive.semi.definite(sigma)) {
  sigma <- nearPSD(sigma)
}

sigma_star <- sigma.star_fun(ms, sigma, cores)

if (!is.positive.semi.definite(sigma_star)) {
  sigma_star <- nearPSD(sigma_star)
}

if (!is.positive.definite(sigma_star)) {
  sigma_star <- as.matrix(nearPD(sigma_star, corr = TRUE,
   keepDiag = TRUE)$mat)
}

data <- as.data.table(rmvn(n, mu = rep(0, nvars),
 sigma = sigma_star, ncores = detectCores() - 10))

for (i in 1:nvars) {
  data[get(paste0("V", i)) < qnorm(1 - p[i]), c(paste0("V", i)) := 0L]
  data[get(paste0("V", i)) > qnorm(1 - p[i]), c(paste0("V", i)) := 1L]
}

for (j in 1:nvars) {
  set(data, j = j, value = as.integer(data[[j]]))
}

if (!is.null(hashs)) {
  hashs2 <- data[, fastdigest(.SD), by = seq_len(n)]$V1
  substitute <- which(hashs2 %in% hashs)
  while (length(substitute > 0)) {
    data_substitute <- as.data.table(rmvn(length(substitute),
     mu = rep(0, nvars),
      sigma = sigma_star, ncores = detectCores() - 10))

    for (i in 1:nvars) {
      data_substitute[get(paste0("V", i)) < qnorm(1 - p[i]),
       c(paste0("V", i)) := 0L]
      data_substitute[get(paste0("V", i)) > qnorm(1 - p[i]),
```

```
          c(paste0("V", i)) := 1L]
      }

      for (j in 1:nvars) {
        set(data_substitute, j = j,
        value = as.integer(data_substitute[[j]]))
      }

      data[substitute, names(data) := data_substitute]
      hashs2[substitute] <- data[substitute, fastdigest(.SD),
       by = seq_len(data[substitute, .N])]$V1
      substitute <- which(hashs2 %in% hashs)
    }
  }

  return(data)

}
```

C.2. Simulation der Korrelationsmatrix

```
simcor <- function(k = 6, size = c(10,5,8,2,15,50),
 rho = c(0.7,0.7,0.5,0.9,0.85,0.4),
 delta = 0.39, epsilon = 0.99 - max(rho), eidim = 2) {
  ndim <- sum(size)
  bigcor <- matrix(rep(delta, ndim * ndim), ncol = ndim)
  for (i in 1:k) {
    cor <- matrix(rep(rho[i], size[i] * size[i]), ncol = size[i])
    if (i == 1) {bigcor[1:size[1], 1:size[1]] <- cor}
    if (i != 1) {bigcor[(sum(size[1:(i - 1)]) + 1):sum(size[1:i]),
                        (sum(size[1:(i - 1)]) + 1):sum(size[1:i])
                        ] <- cor}
  }
  diag(bigcor) <- 1 - epsilon
  eivect <- c()
  for (i in 1:ndim) {
    ei <- runif(eidim, -1, 1)
    eivect <- cbind(eivect,
     sqrt(epsilon) * ei/sqrt(sum(ei^2)))
  }
```

```
  bigE <- t(eivect) %*% eivect
  cor.nz <- bigcor + bigE
  cor.nz
}
```

C.3. Evaluationsfunktionen

```
rec_pre_f1 <- function(orig, result, n) {

  n2 <- (n * (n - 1)) / 2
  m <- cbind(orig, result)
  rt <- bigtabulate(m, c(1,2))
  tp <- sum(apply(rt, 1, choose, 2))
  fptp <- sum(sapply(colSums(rt), choose, 2))
  fp <- fptp - tp
  fn <- sum(apply(rt, 1, function(x) {
    xx <- x
    sumx <- numeric(sum(xx > 1))
    i <- 0
    while(max(xx) > 1) {
      i <- i + 1
      wm <- which.max(xx)
      sumx[i] <- as.numeric(xx[wm]) * as.numeric(sum(xx[-wm]))
      xx <- xx[-wm]
    }
    return(sum(sumx))

  }))

  recall <- tp / (tp + fn)
  precision <- tp / (tp + fp)
  f1 <- 2 * ((precision * recall) / (precision + recall))

  return(data.frame(recall = recall,
   precision = precision,
   f1 = f1))
}


Entropy <- function(orig, result, n) {
```

```
  result <- frank(result, ties.method = "dense")
  maxresult <- max(result)
  maxorig <- max(orig)

  hw <- numeric(maxresult)
  for (i in 1:maxresult) {
    hw_temp <- numeric(maxorig)
    for (j in 1:maxorig) {
      which_i <- which(result == i)
      wc_nw <- length(which(orig[which_i] == j)) / length(which_i)
      if (wc_nw != 0) {
        hw_temp[j] <- wc_nw * log2(wc_nw)
      }
    }

    hw[i] <- -sum(hw_temp)
  }

  h_omega <- numeric(maxresult)
  for (i in 1:maxresult) {
    h_omega[i] <- hw[i] * (length(result[which(result == i)]) / n)
  }
  sum(h_omega)
}

rand <- function(pred, orig, n) {
  m <- cbind(pred, orig)
  tab <- bigtabulate(m, c(1,2))
  ra <- 1 + (sum(tab^2) - 0.5 *
   (sum(rowSums(tab)^2) + sum(colSums(tab)^2))) / choose(n, 2)
  return(ra)
}

rec_pre_f1_lsh <- function(result, ncluster, n, orig) {

  tp_lsh <- function(result, ncluster, n, orig) {
    tp <- integer(ncluster)
    for (i in 1:ncluster) {
      whichlabel <- which(orig == i)
      tp[i] <- result[(id1 %in% whichlabel & id2 %in% whichlabel),
        .N]
    }
```

```
    sum(tp)
  }

  tp <- tp_lsh(result, ncluster, n, orig)

  fp_lsh <- function(result, ncluster, n, orig) {
    fp <- integer(ncluster)
    for (i in 1:ncluster) {
      whichlabel <- which(orig == i)
      fp[i] <-  result[(id1 %in% whichlabel & !id2 %in% whichlabel),
       .N]
    }
    sum(fp)
  }

  fp <- fp_lsh(result, ncluster, n, orig)
  tn <- ((n / ncluster)^ncluster) - fp
  fn <- (choose(n / ncluster, 2) * ncluster) - tp
  precision <- tp / (tp + fp)
  recall <- tp / (tp + fn)
  f1 <- 2 * ((precision * recall) / (precision + recall))
  rand <- (tp + tn) / choose(n, 2)

  return(data.frame(recall = recall,
   precision = precision,
   f1 = f1, rand = rand))
}

lsh_eval_fun <- function(buckets, n_cluster, n_data_sample, orig) {

  clusterseq <- round(seq(0, buckets[[1]][, .N],
  length.out = n_cluster + 1))
  clusterseq[length(clusterseq)] <- buckets[[1]][, .N]

  data_sample <- NULL
  for (i in 1:n_cluster) {
    data_sample <- c(data_sample,
    sample((clusterseq[i] + 1):clusterseq[i + 1],
    n_data_sample / n_cluster, replace = FALSE))
  }

  # find sampled observations in the buckets
```

```
  buckets2 <- lapply(buckets,
   function(x, data_sample) x[id %in% data_sample],
    data_sample)

  # form candidate pairs in the buckets of the sampled observations
  buckets2 <- lapply(buckets2, function(x) {
    whichdup <- which(duplicated(x$h) | duplicated(x$h, fromLast = TRUE))
    if (length(whichdup) > 0) {
      x[whichdup, id, by = h][, pairsfun(id), by = h][, .(id1, id2)]
    } else {
      data.table()
    }
  })

  buckets2 <- rbindlist(buckets2)
  # deduplicate in case a pair is in more than one bucket
  buckets2 <- unique(buckets2)
  rec_pre_f1_lsh(buckets2, n_cluster, n_data_sample, orig)
}
```

C.4. Clusteralgorithmen

C.4.1. LSH

```
pairsfun <- function(d) {
  a <- comboGeneral(d, 2)
  setDT(list(id1 = a[, 1], id2 = a[, 2]))
}

lsh_fun <- function(dataset, n_hash, b, ncores) {

  # minhash

  n <- nrow(dataset)
  ncols <- ncol(dataset)
  n_hash <- n_hash

  pri <- generate_primes(min = ncols + 1, ncols + 1000)
  p <- pri[2]
  hashmat <- foreach(i = 1:n_hash, .combine = cbind) %dorng% {
```

```
    a <- sample(seq(1, p - 1, 2), 1)
    b1 <- sample(0:(p - 1), 1)
    m <- ncols + 1
    ((a*(1:ncols)+b1) %% p) %% m
  }

  # positions of 1s in every observation
  whichhashmat <- apply(dataset, 1, function(x) which(x == 1))

  # the minimum of the hashvalues in the hashmatcolumns
  # form the ultimate hashvalue for hashfunction r -> sigmat
  ii <- isplitVector(whichhashmat, chunks = ncores)

  sigmat <- foreach(i = ii, .combine = cbind,
   .packages = "matrixStats" ) %dopar% {
     sapply(i, function(x) colMins(hashmat[x,, drop = FALSE]))
  }

  ii <- isplitRows(sigmat, chunks = b)

  buckets <- foreach(dat = ii) %dopar% {
    dat2 <- data.table(h = apply(dat, 2, fastdigest),
     id = 1:ncol(dat))
    dat2
  }

  return(buckets)
}

cross_lsh <- function(dataset, ncluster, orig, ncores) {

  n <- nrow(dataset)
  datasample <- matrix(nrow = 2000, ncol = ncol(dataset))
  boundsample <- round(2000 / ncluster)
  boundssample <- round(seq(0, 2000, length.out = ncluster + 1))
  for (i in 1:ncluster) {
    datasample[(boundssample[i] + 1):boundssample[i + 1],] <-
     dataset[which(orig == i),][sample(1:length(which(orig == i)),
     length((boundssample[i] + 1):boundssample[i + 1])),]
  }

  shuf_index <- sample(1:nrow(datasample), nrow(datasample))
```

```
datasample <- datasample[shuf_index,]

orig_sample <- integer()
for (i in 1:ncluster) {
  orig_sample <- c(orig_sample,
   rep(i, round(nrow(datasample) / ncluster)))
}

orig_sample <- orig_sample[shuf_index]
bandslist <- 2:15
n_hash_list <- seq(4, 50, 2)
bestlist <- NULL
h <- 0
for (i in bandslist) {
  for (j in n_hash_list) {
    if ((j / i) %% 2 == 0) {
      h <- h + 1
      lsh_result <- lsh_fun(datasample, n_hash = j,
       b = i, ncores = ncores)

      lsh_result <- lapply(lsh_result, function(x) {
        whichdup <- which(duplicated(x$h) |
         duplicated(x$h, fromLast=TRUE))
        if (length(whichdup) > 0) {
          x[whichdup, id, by = h][, pairsfun(id),
           by = h][, .(id1, id2)]
        } else {
          data.table()
        }
      })

      lsh_result <- rbindlist(lsh_result)

      # deduplicate in case a pair is in more than one bucket
      lsh_result <- unique(lsh_result)

      if (lsh_result[,.N] > 0) {
        bestlist[h] <- rec_pre_f1_lsh(lsh_result,
         ncluster, nrow(datasample), orig_sample)$f1
        names(bestlist)[h] <- paste0(i, " ", j)
      }
    }
```

```
    }
  }

  whichbest <- as.integer(strsplit(
    names(bestlist[which.max(bestlist)]), " ")[[1]])
  bands_number <- whichbest[1]
  n_hash <- whichbest[2]
  return(data.frame(bands_number = bands_number,
   n_hash = n_hash))
}

# minhash

minhash <- function(dataset, n_hash, ncores) {

  n <- nrow(dataset)
  ncols <- ncol(dataset)
  pri <- generate_primes(min = ncols + 1, ncols + 1000)
  p <- pri[2]
  hashmat <- foreach(i = 1:n_hash, .combine = cbind) %dorng% {
    a <- sample(seq(1, p - 1, 2), 1)
    b1 <- sample(0:(p - 1), 1)
    m <- ncols + 1
    ((a*(1:ncols)+b1) %% p) %% m
  }

  # positions of 1s in every observation
  whichhashmat <- apply(dataset, 1, function(x) which(x == 1))

  ii <- isplitVector(whichhashmat, chunks = ncores)

  sigmat <- foreach(i = ii, .combine = cbind,
   .packages = "matrixStats" ) %dopar% {
    sapply(i, function(x) colMins(hashmat[x,, drop = FALSE]))
  }

  colnames(sigmat) <- 1:ncol(sigmat)
  ii <- isplitCols(sigmat, chunks = ncores)

  buckets <- foreach(dat = ii) %dopar% {
    data.table(h = apply(dat, 2, fastdigest),
     id = as.integer(colnames(dat)))
```

```
  }

  buckets <- rbindlist(buckets)
  buckets[, clusters := .GRP, by = h]
  return(buckets$clusters)
}

cross_minhash <- function(dataset, ncluster, orig, ncores) {

  n <- nrow(dataset)
  datasample <- matrix(nrow = 2000, ncol = ncol(dataset))
  boundsample <- round(2000 / ncluster)
  boundssample <- round(seq(0, 2000,
   length.out = ncluster + 1))
  for (i in 1:ncluster) {
    datasample[(boundssample[i] + 1):boundssample[i + 1],] <-
    dataset[which(orig == i),][sample(1:length(which(orig == i)),
     length((boundssample[i] + 1):boundssample[i + 1])),]
  }

  shuf_index <- sample(1:nrow(datasample), nrow(datasample))
  datasample <- datasample[shuf_index,]

  orig_sample <- integer()
  for (i in 1:ncluster) {
    orig_sample <- c(orig_sample, rep(i,
     round(nrow(datasample) / ncluster)))
  }

  orig_sample <- orig_sample[shuf_index]

  n_hash_list <- seq(2, 50, 2)
  bestlist <- NULL
  h <- 0
  for (i in n_hash_list) {
    h <- h + 1
    minhash_result <- minhash(datasample, i, ncores)
    bestlist[h] <- rec_pre_f1(orig_sample,
     minhash_result, nrow(datasample))$f1
  }

  whichbest <- n_hash_list[which.max(bestlist)]
```

```
  return(whichbest)
}
```

C.4.2. CLARA

```
clara_fun <- function(dataset, n, nsamples,
 samplesize, ncores, ncluster) {
  cost_medoids_mat_sum <- NULL
  medoids <- list()

  for (i in 1:nsamples) {
    medoid_indices <- "Cluster_Medoids_error"
    while (is.character(medoid_indices)) {
      binarydata_sample <- dataset[sample(1:n, samplesize,
       replace = FALSE), ]
      medoid_indices <- tryCatch(Cluster_Medoids(binarydata_sample,
       ncluster, distance_metric = "hamming", threads = ncores,
        seed = i)$medoid_indices,
         error = function(x) {
          message("trycatch hat gegriffen clara")
          medoid_indices <- "Cluster_Medoids_error"
          return(medoid_indices)
      })
    }

    medoids[[i]] <- binarydata_sample[medoid_indices,]
    cost_medoids_mat <- rdist:::hamming_cdist(dataset, medoids[[i]])
    cost_medoids_mat_sum[i] <- sum(rowMins(cost_medoids_mat))
    if (i == 1) {
      cluster_clara <- max.col(-cost_medoids_mat, ties="first")
    } else {
      if (cost_medoids_mat_sum[i] < cost_medoids_mat_sum[i - 1]) {
        cluster_clara <- max.col(-cost_medoids_mat, ties="first")
      }
    }
    rm(cost_medoids_mat)
  }

  return(cluster_clara)
}
```

C.4.3. CLARANS

```
clarans_fun <- function(dataset, numlocal, ncluster,
 maxneighbor, ncores) {
  numlocal_list <- list()

  for (i in 1:numlocal) {
    current <- dataset[sample(1:n, ncluster, replace = FALSE),, drop = FALSE]
    cost_current_mat <- rdist:::hamming_cdist(dataset, current)
    cost_current <- sum(rowMins(cost_current_mat))
    rm(cost_current_mat)
    cost_differential <- 1
    j <- 1
    while (j < maxneighbor + ncores) {
      while (cost_differential > 0) {
        cost_list <- foreach(h = 1:ncores, .combine = c,
         .packages = c("rdist", "matrixStats")) %dorng% {
          neighbor <- current
          neighbor[sample(1:ncluster, 1),] <- dataset[sample(1:n, 1),]
          cost_neighbor_mat <- rdist:::hamming_cdist(dataset, neighbor)
          cost_neighbor <- sum(rowMins(cost_neighbor_mat))
          rm(cost_neighbor_mat)
          list(neighbor, cost_neighbor)
        }
        new_best_medoid <- cost_list[[which.min(unlist(cost_list[seq(2,
         length(cost_list), 2)], use.names = FALSE)) * 2 - 1]]
        new_best_cost <- cost_list[[which.min(unlist(cost_list[seq(2,
         length(cost_list), 2)], use.names = FALSE)) * 2]]
        cost_differential <- cost_current - new_best_cost
        if (cost_differential > 0) {
          current <- new_best_medoid
          cost_current <- new_best_cost
          j <- 1
        }
      }
      if (cost_differential <= 0) {
        j <- j + ncores
      }
    }
    numlocal_list <- c(numlocal_list, list(cost_current, current))
  } # i
```

```
  medoids <- numlocal_list[[which.min(unlist(numlocal_list[seq(1,
   length(numlocal_list), 2)], use.names = FALSE)) * 2]]
  cost_medoids_mat <- rdist:::hamming_cdist(dataset, medoids)
  cluster_clarans <- max.col(-cost_medoids_mat, ties="first")

  cost_result <- numlocal_list[[which.min(unlist(numlocal_list[seq(1,
   length(numlocal_list), 2)], use.names = FALSE)) * 2 - 1]]
  return(list(medoids, cluster_clarans, cost_result))
}
```

C.4.4. BUBBLE

```
levelupdate <- function(x, rootid = "1") {
  if (x$id != rootid) {
    x$level <- x$level + 1
  }
  return(x)
}

bubble <- function(dataset, ncores, threshold, L) {

nvars <- ncol(dataset)
cftree <- list()
nearest_history <- NULL
clustroid_change <- FALSE
split_node <- FALSE
h <- 0
id <- 0
R <- 100
p <- 10
plist <- vector(mode = "list", length = L)

main_cftree <- foreach(binarydata = isplitRows(dataset, chunks = ncores),
.packages = c("rlist", "rdist", "stringr", "data.table"),
.export = c("cftree", "clusterlabels", "clustroid_change",
 "split_node", "threshold", "h", "id", "R", "p", "L",
 "plist")) %dorng% {

clusterlabels <- data.table(label = integer(nrow(binarydata)))
for (i in 1:nrow(binarydata)) {
```

```
  # if a node was split, the observation
  # must be inserted into the tree again
run_case_again <- TRUE
while (run_case_again) {
run_case_again <- FALSE
if (length(cftree) == 0) {
  h <- h + 1
  rowsum_clustroid <- 0
  id <- id + 1
  idparent <- "1"
  cftree[[h]] <- list(level = 2, leaf = TRUE,
  id = paste0(2, idparent, 1), L = 1,
  cfs = list(n = 1, clustroid = matrix(binarydata[i,], ncol = nvars),
   rowsum_clustroid = rowsum_clustroid,
   p_near = plist, rowsums_pnear = plist,
   radius = NULL))

  h <- h + 1
  id <- id + 1

  cftree[[h]] <- list(level = 1, leaf = FALSE, id = idparent, L = 1,
   cfs = list(clustroid = t(binarydata[i,]),
    rowsum_clustroid = rowsum_clustroid),
                      child = paste0(2, idparent, 1))
  idlist <- c(cftree[[2]]$id, cftree[[1]]$id)
  clusterlabels[i] <- as.integer(paste0(cftree[[1]]$id, 1))
} else {
  leaf_reached <- FALSE
  root <- TRUE
  while (!leaf_reached) {
    if (root) {
      clustroids_compare <-
      cftree[[list.which(cftree, id == "1")]]$cfs$clustroid
      nextid <- cftree[[list.which(cftree, id == "1")]]$id
      root <- FALSE
      nearest_history <- NULL
      parentid <- NULL
    } else {
      clustroids_compare <-
       cftree[[list.which(cftree, id == nextid)]]$cfs$clustroid
    }
```

```
    leaf <- cftree[[list.which(cftree, id == nextid)]]$leaf

    dist_mat_clustroid <-
     rdist:::hamming_cdist(binarydata[i,, drop = FALSE],
      clustroids_compare)

    nearest <- which.min(dist_mat_clustroid)

    # save nearest in non-leasf for updating clustroids in parent-nodes
    nearest_history <- c(nearest_history, nearest)

dist_to_clust <- min(dist_mat_clustroid)

if (!leaf) {
    parentid <- c(parentid, cftree[[list.which(cftree, id == nextid)]]$id)
    nextid <- cftree[[list.which(cftree, id == nextid)]]$child[nearest]
  }

  # maintain leafnote
  if (leaf) {
    leaf_reached <- TRUE
    current_node <- cftree[[list.which(cftree, id == nextid)]]


    if (dist_to_clust <= threshold) {

      # give clusterlabels
      clusterlabels[i, label := as.integer(paste0(current_node$id,
      nearest))]

      # calculate rowsum_o
      if (current_node$cfs$n[nearest] <= p) {
        R_cases <- rbind(current_node$cfs$p_near[[nearest]],
         current_node$cfs$clustroid[nearest,])

        rowsum_o <-
         sum(rdist:::hamming_cdist(binarydata[i,, drop = FALSE],
         R_cases))

        current_node$cfs$radius[nearest] <-
         sum(rdist:::hamming_cdist(R_cases,
```

```
          current_node$cfs$clustroid[nearest,, drop = FALSE]))

      } else {
        # radius
        clustroid <- current_node$cfs$clustroid[nearest,, drop = FALSE]

        # approximation rowsum_o
        n_cl <- current_node$cfs$n[nearest]
        dist_obs_clustroid <-
         rdist:::hamming_cdist(binarydata[i,, drop = FALSE], clustroid)

        radius <- current_node$cfs$radius[nearest] / n_cl
        rowsum_o <- n_cl * radius + n_cl * dist_obs_clustroid

        current_node$cfs$radius[nearest] <-
         current_node$cfs$radius[nearest] + dist_obs_clustroid
      }

      # update clustroid_rowsum and rowsum_pnear
      # rowsum clustroid needs to be updated, as does rowsums_pnear
if (is.null(current_node$cfs$p_near[[nearest]])) {

 # update rowsum_clustroid
 current_node$cfs$rowsum_clustroid[nearest] <-
  rdist:::hamming_cdist(current_node$cfs$clustroid[nearest,, drop = FALSE],
   binarydata[i,, drop = FALSE])

 } else { # p_near is not NULL
   # update clustroid_rowsum and rowsums_pnear

   # update clustroid_rowsum by adding
   # distance from clustroid to binarydata[i,]
   # to clustroid_rowsum. Same with pfar and pnear.
   current_node$cfs$rowsum_clustroid[nearest] <-
    current_node$cfs$rowsum_clustroid[nearest] +
     rdist:::hamming_cdist(current_node$cfs$clustroid[
      nearest,, drop = FALSE], binarydata[i,, drop = FALSE
       # update rowsums_pnear
       current_node$cfs$rowsums_pnear[[nearest]] <-
        rdist:::hamming_cdist(binarydata[i,, drop = FALSE],
         current_node$cfs$p_near[[nearest]]) +
         current_node$cfs$rowsums_pnear[[nearest]]
```

```
    }

   clustroid <- current_node$cfs$clustroid[nearest,]
   clustroid_rowsum <- current_node$cfs$rowsum_clustroid[nearest]

   # check if any of pnear or O have
   # lower rowsum than clustroid
   # it can happen, that the cluster consists
   # of only the clustroid ->
   # nearest > length(current_node$cfs$rowsums_pnear)
if (ifelse(nearest > length(current_node$cfs$rowsums_pnear), FALSE,
           ifelse(any(append(
             current_node$cfs$rowsums_pnear[[nearest]], rowsum_o) <
             clustroid_rowsum), TRUE, FALSE))) {
  # indicator for clustroid change for updating clustroids of parent-nodes
  clustroid_change <- TRUE
  # old clustroid to pnearest
  if (nrow(current_node$cfs$p_near[[nearest]]) < p) {
    # position in pnearest to fill with old clustroid
    current_node$cfs$p_near[[nearest]] <-
     rbind(current_node$cfs$p_near[[nearest]], clustroid)
    current_node$cfs$rowsums_pnear[[nearest]] <-
     append(current_node$cfs$rowsums_pnear[[nearest]],
      clustroid_rowsum)
  } else {
    # if pnearest is already saturated
    rowsums_pnear <- current_node$cfs$rowsums_pnear[[nearest]]
    current_node$cfs$p_near[[nearest]][which.max(rowsums_pnear),] <-
     clustroid
    current_node$cfs$rowsums_pnear[[nearest]][which.max(rowsums_pnear)] <-
     clustroid_rowsum
  }

   # check which of pnear and O has lowest rowsum
   which_new_clustroid <- which.min(append(
     rowsum_o, current_node$cfs$rowsums_pnear[[nearest]]))

   if (which_new_clustroid == 1) {
     current_node$cfs$clustroid[nearest,] <- binarydata[i,]
     current_node$cfs$rowsum_clustroid[nearest] <- rowsum_o
   } else {
     # -1 because the first one is rowsum_o
```

```
    current_node$cfs$clustroid[nearest,] <-
     current_node$cfs$p_near[[nearest]][which_new_clustroid - 1,]
    current_node$cfs$rowsum_clustroid[nearest] <-
     current_node$cfs$rowsums_pnear[[nearest]][which_new_clustroid - 1]
  }
    # UPDATE PARENTNODE

  } else { # rowsum_o > clustroid_rowsum

    # pnear
    if (is.null(current_node$cfs$p_near[[nearest]])) {

    # it can happen, that the cluster
    # consists of only the clustroid ->
    # nearest > length(current_node$cfs$rowsums_pnear)
    if (nearest > length(current_node$cfs$rowsums_pnear)) {
      current_node$cfs$rowsums_pnear <- append(
        current_node$cfs$rowsums_pnear, rowsum_o)
    } else {
      current_node$cfs$rowsums_pnear[[nearest]] <-
       rowsum_o
        }
        current_node$cfs$p_near[[nearest]] <-
         binarydata[i,, drop = FALSE]

      } else {
        # if pnear not saturated. add binarydata[i,] to pnear
        if (nrow(current_node$cfs$p_near[[nearest]]) < p) {
          current_node$cfs$p_near[[nearest]] <-
           rbind(current_node$cfs$p_near[[nearest]], binarydata[i,])
          current_node$cfs$rowsums_pnear[[nearest]] <-
           c(current_node$cfs$rowsums_pnear[[nearest]], rowsum_o)
          }

          # if pnear already saturated,
          # substitute pnearest case with highest
          # rowsum with binarydata[i,]
          if (rowsum_o < max(
             current_node$cfs$rowsums_pnear[[nearest]])) {
            which_to_fill <- which.max(
               current_node$cfs$rowsums_pnear[[nearest]])
            current_node$cfs$p_near[[nearest]][which_to_fill,] <-
```

```
          binarydata[i,]
         current_node$cfs$rowsums_pnear[[nearest]][which_to_fill] <-
          rowsum_o
       }
     }
   }
   # n + 1
   current_node$cfs$n[nearest] <- current_node$cfs$n[nearest] + 1
 } else { # dist_to_clust > threshold


 if (current_node$L < L) {
   current_L <- current_node$L + 1
   current_node$cfs$n[current_L] <- 1
   current_node$cfs$clustroid <-
    rbind(current_node$cfs$clustroid, binarydata[i,])
   current_node$cfs$rowsum_clustroid[current_L] <- 0
   current_node$L <- current_L
   # add clustroid to clustroids parentnode
   clustroid_change <- TRUE

   # give clusterlabels
   clusterlabels[i, label := as.integer(
    paste0(current_node$id, current_L))]

 } else {
   split_node <- TRUE
   # indicator to run binarydata[i,] again through the tree,
   # because of splitting.
   run_case_again <- TRUE
 }

 # threshold

# update the actual node with current_node
cftree[[list.which(cftree, id == nextid)]] <- current_node
# get current id for splitting later
current_id <- cftree[[list.which(cftree, id == nextid)]]$id
# root to true to enter root node on next iteration
root <- TRUE

# update parentnode
```

```
if (clustroid_change) {
 clustroid_change <- FALSE
 # which clustroid of current_node is clustroid of parentnode
 which_clustroid <- which.min(
  rowSums(rdist:::hamming_pdist(
    current_node$cfs$clustroid)))
 for (pi in length(parentid):1) {
   if (pi == length(nearest_history) - 1) {
     # if leaf_node (length(nearest_history) - 1)
     # change clustroid one node higher with updated clustroid
     cftree[[list.which(cftree,
      id == parentid[pi])]]$cfs$clustroid[
        nearest_history[pi],] <-
         current_node$cfs$clustroid[which_clustroid,]
   } else {
     # if interior_node update clustroid of
     # childnode with the clustroid of the
     # clustroids of the child node
     which_clustroid <- which.min(rowSums(
      rdist:::hamming_pdist(cftree[[list.which(
        cftree, id == parentid[pi + 1])]]$cfs$clustroid)))
     cftree[[list.which(
      cftree, id == parentid[pi])]]$cfs$clustroid[
        nearest_history[pi],] <-
         cftree[[list.which(cftree, id == parentid[
          pi + 1])]]$cfs$clustroid[which_clustroid,]
   }
 }

# split nodes
if (split_node) {
split_node <- FALSE
ids <- c(parentid, current_id)
for (spl in rev(ids)) {
  if (cftree[[list.which(cftree, id == spl)]]$L == L) {
    # find clustroids that are furthes apart
    index_furthes <- which(
      rdist:::hamming_pdist(cftree[[
        list.which(cftree, id == spl)]]$cfs$clustroid) == max(
      rdist:::hamming_pdist(cftree[[
        list.which(cftree, id == spl)]]$cfs$clustroid)),
         arr.ind = T)[1]
```

```
      # find L/2 closest clustroid to clustroid that is furthest out
      dist_clus_to_clus <-
       rdist:::hamming_cdist(cftree[[
        list.which(cftree, id == spl)]]$cfs$clustroid[
          index_furthes,, drop = FALSE],
          cftree[[list.which(cftree, id == spl)]]$cfs$clustroid)
      # positions of L/2 closest
      closest <- order(dist_clus_to_clus)[1:(L/2)]
      # positions of L/2 farthest
      farthest <- order(dist_clus_to_clus)[
        (L/2 + 1):length(dist_clus_to_clus)]

      # new node
      # if leaf_node cfs need to be allocated
if (cftree[[list.which(cftree, id == spl)]]$leaf) {

cfs_new_node <- list(
  n = cftree[[list.which(cftree, id == spl)]]$cfs$n[closest],
  clustroid = cftree[[
    list.which(cftree, id == spl)]]$cfs$clustroid[closest,],
  rowsum_clustroid = cftree[[
    list.which(
      cftree, id == spl)]]$cfs$rowsum_clustroid[closest],
  p_near = plist,
  rowsums_pnear = cftree[[
    list.which(cftree, id == spl)]]$cfs$rowsums_pnear[closest],
  radius = cftree[[
    list.which(cftree, id == spl)]]$cfs$radius[closest])

cfs_new_node$p_near[1:length(closest)] <-
 cftree[[list.which(cftree, id == spl)]]$cfs$p_near[closest]


cfs_old_node <- list(n = cftree[[list.which(
  cftree, id == spl)]]$cfs$n[farthest],
   clustroid = cftree[[
    list.which(cftree, id == spl)]]$cfs$clustroid[farthest,],
   rowsum_clustroid = cftree[[
    list.which(cftree, id == spl)]]$cfs$rowsum_clustroid[farthest],
   p_near = plist,
   rowsums_pnear = cftree[[
    list.which(cftree, id == spl)]]$cfs$rowsums_pnear[farthest],
```

```
    radius = cftree[[
     list.which(cftree, id == spl)]]$cfs$radius[farthest])

cfs_old_node$p_near[1:length(farthest)] <-
 cftree[[list.which(cftree, id == spl)]]$cfs$p_near[farthest]


new_node <- list(level = cftree[[list.which(cftree, id == spl)]]$level,
  leaf = cftree[[list.which(cftree, id == spl)]]$leaf,
  id = as.character(as.numeric(idlist[length(idlist)]) + 1),
  L = L / 2,
  cfs = cfs_new_node)

old_node <- list(level = cftree[[list.which(cftree, id == spl)]]$level,
  leaf = cftree[[list.which(cftree, id == spl)]]$leaf,
  id = cftree[[list.which(cftree, id == spl)]]$id,
  L = L / 2,
  cfs = cfs_old_node)

  # children in parent_node need
  # to be updated. gets last id + 1
  cftree[[list.which(
    cftree, id == parentid[
      length(parentid)])]]$child <-
       append(cftree[[list.which(
        cftree, id == parentid[length(parentid)])]]$child,
        as.character(as.numeric(idlist[length(idlist)]) + 1),
        after = which(cftree[[list.which(
          cftree, id == parentid[
            length(parentid)])]]$child == old_node$id))

  # add new id to idlist
  idlist <- append(idlist,
   as.character(as.numeric(idlist[length(idlist)]) + 1))

  ## update clustroid in parentnode
  # old node first
  # which to update
  which_clustroid_to_update <-
   which(cftree[[list.which(cftree,
    id == parentid[length(parentid)])]]$child == cftree[[
      list.which(cftree, id == spl)]]$id)
```

```
replacement_clustroid <-
 which.min(rowSums(rdist:::hamming_pdist(
  old_node$cfs$clustroid)))
cftree[[list.which(cftree,
 id == parentid[length(parentid)])]]$cfs$clustroid[
  which_clustroid_to_update,] <-
   old_node$cfs$clustroid[replacement_clustroid,]

# increase L in parentnode
cftree[[list.which(cftree,
 id == parentid[length(parentid)])]]$L <-
  cftree[[list.which(cftree, id == parentid[
    length(parentid)])]]$L + 1

# new node
# clustroid of new node gets added
replacement_clustroid <-
 which.min(rowSums(rdist:::hamming_pdist(
  new_node$cfs$clustroid)))
cftree[[list.which(cftree,
id == parentid[length(parentid)])]]$cfs$clustroid <-
 rbind(cftree[[list.which(cftree,
  id == parentid[length(parentid)])]]$cfs$clustroid,
  new_node$cfs$clustroid[replacement_clustroid,])

# old node gets replaced. new node gets appended
cftree[[list.which(cftree, id == spl)]] <- old_node
cftree <- list.append(cftree, new_node)

# update clusterlabels of observations in new node
which_to_change <- which(
  clusterlabels$label %in% as.integer(
    paste0(old_node$id, closest)))
new_labels <- as.character(
  clusterlabels$label[which_to_change])
# change id part of clusterlabel to new nodeid
new_labels <- str_replace(
  new_labels, paste0("\\d{", nchar(
    new_node$id), "}"), new_node$id)
# change last digit to clusterposition in the new node
labelchange <- 1:length(new_labels)
for (ld in as.character(closest)) {
```

```
      labelchange2 <- which(str_detect(
        new_labels[labelchange], paste0(
          new_node$id, ld, "(?!.)")))
      new_labels[labelchange] <-
       str_replace(new_labels[labelchange],
        paste0(new_node$id, ld, "(?!.)"),
        paste0(new_node$id,
         as.character(which(as.character(closest) == ld))))
      labelchange <- labelchange[-labelchange2]
    }
  clusterlabels[which_to_change, label := as.integer(
      new_labels)]
  # update clusterlabels of observations in old node
    which_to_change <- which(
      clusterlabels$label %in% as.integer(
        paste0(old_node$id, farthest)))
    new_labels <- as.character(
      clusterlabels$label[which_to_change])
    # change last digit to clusterposition in the old node
    labelchange <- 1:length(new_labels)
    for (ld in as.character(farthest)) {
      labelchange2 <- which(str_detect(
   new_labels[labelchange], paste0(old_node$id, ld, "(?!.)")))
       new_labels[labelchange] <- str_replace(
         new_labels[labelchange], paste0(old_node$id, ld, "(?!.)"),
         paste0(old_node$id, as.character(
           which(as.character(farthest) == ld))))
       labelchange <- labelchange[-labelchange2]
     }

       clusterlabels[which_to_change,
        label := as.integer(new_labels)]
       } else { # non leaf

       # rootnode needs to be split
       if (cftree[[list.which(cftree, id == spl)]]$id == parentid[1]) {
         if (cftree[[list.which(cftree, id == spl)]]$L == L) {

           # only the clustroids are needed in the cfs
           cfs_new_node <- list(
            clustroid = cftree[[list.which(cftree,
             id == spl)]]$cfs$clustroid[closest,])
```

```
cfs_old_node <- list(
 clustroid = cftree[[list.which(cftree,
  id == spl)]]$cfs$clustroid[farthest,])

# contrary to leaf nodes, the children need to be split
new_node <- list(
 level = cftree[[list.which(cftree, id == spl)]]$level,
 leaf = cftree[[list.which(cftree, id == spl)]]$leaf,
 id = as.character(as.numeric(idlist[length(idlist)]) + 1),
 L = L / 2,
 cfs = cfs_new_node,
 child = cftree[[
   list.which(cftree, id == spl)]]$child[closest])

old_node <- list(
 level = cftree[[list.which(cftree, id == spl)]]$level,
 leaf = cftree[[list.which(cftree, id == spl)]]$leaf,
 id = as.character(as.numeric(idlist[length(idlist)]) + 2),
 L = L / 2,
 cfs = cfs_old_node,
 child = cftree[[list.which(cftree, id == spl)]]$child[farthest])

clustroid_old <- which.min(
 rowSums(rdist:::hamming_pdist(old_node$cfs$clustroid)))
clustroid_new <- which.min(
 rowSums(rdist:::hamming_pdist(new_node$cfs$clustroid)))
clustroids_new_root <- list(
 clustroid = rbind(new_node$cfs$clustroid[clustroid_new,],
  old_node$cfs$clustroid[clustroid_old,]))

# initialize new rootnode
new_root <- list(level = 1,
 leaf = FALSE,
 id = "1",
 L = 2,
 cfs = clustroids_new_root,
 child = c(as.character(
   as.numeric(idlist[length(idlist)]) + 1),
    as.character(as.numeric(
     idlist[length(idlist)]) + 2)))

# add new ids to idlist
```

```
            idlist <- append(idlist, c(new_node$id, old_node$id))

            # old node gets replaced. new node gets appended
            cftree[[list.which(cftree, id == spl)]] <- old_node
            cftree <- list.append(cftree, new_node)

      # add new root
      cftree <- list.append(cftree, new_root)

      # levels need to be increased by 1, as well as the first
      # character in id
      cftree <- lapply(cftree, levelupdate)
    }
  } else { # not root or leaf

    # only the clustroids are needed in the cfs
    cfs_new_node <- list(clustroid = cftree[[
      list.which(cftree, id == spl)]]$cfs$clustroid[closest,])
    cfs_old_node <- list(clustroid = cftree[[
      list.which(cftree, id == spl)]]$cfs$clustroid[farthest,])

    # contrary to leaf nodes, the children need to be split
    new_node <- list(level = cftree[[list.which(cftree, id == spl)]]$level,
                     leaf = cftree[[list.which(cftree, id == spl)]]$leaf,
                     id = as.character(as.numeric(idlist[length(idlist)]) + 1),
                     L = L / 2,
                     cfs = cfs_new_node,
                     child = cftree[[list.which(cftree, id == spl)]]$child[closest])

    old_node <- list(level = cftree[[list.which(cftree, id == spl)]]$level,
                     leaf = cftree[[list.which(cftree, id == spl)]]$leaf,
                     id = cftree[[list.which(cftree, id == spl)]]$id,
                     L = L / 2,
                     cfs = cfs_old_node,
                     child = cftree[[list.which(cftree, id == spl)]]$child[farthest])

    # children in parent_node need to be updated. gets last id + 1
    cftree[[list.which(cftree, id == rev(ids)[
      which(rev(ids) == spl) + 1])]]$child <-
       append(cftree[[list.which(cftree, id == rev(ids)[
        which(rev(ids) == spl) + 1])]]$child,
       as.character(as.numeric(idlist[length(idlist)]) + 1),
```

```
        after = which(cftree[[list.which(cftree, id == rev(ids)[
         which(rev(ids) == spl) + 1])]]$child == old_node$id))

    # add new id to idlist
    idlist <- append(idlist,
    as.character(as.numeric(idlist[length(idlist)]) + 1))

    # increase L in parentnode
    cftree[[list.which(cftree, id == rev(ids)[which(rev(ids) == spl) + 1])]]$L <-
     cftree[[list.which(cftree, id == rev(ids)[which(rev(ids) == spl) + 1])]]$L + 1

    ## update clustroid in parentnode
    # old node first
    # which to update
    which_clustroid_to_update <-
     which(cftree[[list.which(cftree,
     id == parentid[which(parentid == spl) - 1])]]$child == cftree[[
      list.which(cftree, id == spl)]]$id)
    replacement_clustroid <-
     which.min(rowSums(rdist:::hamming_pdist(old_node$cfs$clustroid)))
    cftree[[list.which(cftree,
     id == parentid[which(parentid == spl) - 1])]]$cfs$clustroid[
      which_clustroid_to_update,] <-
      old_node$cfs$clustroid[replacement_clustroid,]

    # new node
    # clustroid of new node gets added
    new_clustroid <- which.min(
      rowSums(rdist:::hamming_pdist(new_node$cfs$clustroid)))
    cftree[[list.which(cftree,
     id == parentid[which(parentid == spl) - 1])]]$cfs$clustroid <-
      rbind(cftree[[list.which(cftree,
      id == parentid[which(parentid == spl) - 1])]]$cfs$clustroid,
      new_node$cfs$clustroid[new_clustroid,])

    # old node gets replaced. new node gets appended
    cftree[[list.which(cftree, id == spl)]] <- old_node
    cftree <- list.append(cftree, new_node)

  }
}
```

```
        } # node$L == L

              } # spl
            } # split_node

          } # leaf
        } # while (!leaf_reached)
      } # else beginning
    } # while (run_case_again)
  } # i
  clusterlabels[, label := as.numeric(paste0(Sys.getpid(), label))]
                                      list(cftree, clusterlabels,
                                       Sys.getpid())
                                    } # foreach

# extract and combine clusterlabels
clusterlabels_combined <-
 rbindlist(lapply(main_cftree, function(x) x[[2]]))


 # extract and combine clustroids of leafnodes
 extractclustroids <- function(x) {
   xlist <- x[[1]][list.which(x[[1]], leaf == TRUE)]
   lapply(xlist, function(y) y$cfs$clustroid)
 }

 clustroids_combined <- lapply(main_cftree, extractclustroids)

 # give rownames to the clustroids for identification for final clustering

 extractids <- function(x) {
   xlist <- x[[1]][list.which(x[[1]], leaf == TRUE)]
   sapply(xlist, function(y) y$id)
 }

 ids <- lapply(main_cftree, extractids)

 # extract sessionids according to ncores for naming
 extractsessionids <- function(x) {
   x[[3]]
 }
```

```
  sessionids <- sapply(main_cftree, extractsessionids)

  # name listelements (cftrees) according to ncores
  names(clustroids_combined) <- sessionids

  nameclustroids <- function(x, id) {
    xlist <- x[1]
    for (i in 1:length(xlist[[1]])) {
      rownames(xlist[[1]][[i]]) <- paste0(names(xlist), id[i],
       seq(1, nrow(xlist[[1]][[i]])))
    }
    return(xlist)
  }

  for (i in 1:length(clustroids_combined)) {
    clustroids_combined[[i]] <-
     nameclustroids(clustroids_combined[i], ids[[i]])
  }

  clustroids_final <- NULL
  for (i in 1:length(clustroids_combined)) {
    for (j in 1:length(clustroids_combined[[i]][[1]])) {
      clustroids_final <- rbind(clustroids_final,
       clustroids_combined[[i]][[1]][[j]])
    }
  }

  return(list(clustroids = clustroids_final,
   clusterlabels = clusterlabels_combined))
}

cross_bubble <- function(dataset, ncluster, orig) {

  n <- nrow(dataset)

  datasample <- matrix(nrow = 2000, ncol = ncol(dataset))
  boundsample <- round(2000 / ncluster)
  boundssample <- round(seq(0, 2000, length.out = ncluster + 1))

  for (i in 1:ncluster) {
    which_sample <- sample(1:floor(n / ncluster),
     boundssample[i + 1] - boundssample[i])
```

```
  datasample[(boundssample[i] + 1):boundssample[i + 1],] <-
   dataset[which(orig == i),][which_sample,]
}

shuf_index <- sample(1:nrow(datasample), nrow(datasample))
datasample <- datasample[shuf_index,]

orig_sample <- integer()
for (i in 1:ncluster) {
  orig_sample <- c(orig_sample, rep(i,
  round(nrow(datasample) / ncluster)))
}

orig_sample <- orig_sample[shuf_index]

L_list <- c(10, 20, 50, 100)
thresholdlist <- seq(0.5, 0.8, 0.1)
flist <- NULL
h <- 0
for (thresh in 1:length(thresholdlist)) {
  for (L in 1:length(L_list)) {
    h <- h + 1
    bubble_result <- bubble(datasample, 1,
    thresholdlist[thresh], L_list[L])

    if (nrow(bubble_result$clustroids) > ncluster) {
      clustroids_dist <- rdist(bubble_result$clustroids, metric = "hamming")
      result_pam <- pam(clustroids_dist, k = ncluster, cluster.only = TRUE)
      for (i in 1:ncluster) {
        set(bubble_result$clusterlabels,
        which(bubble_result$clusterlabels$label %in% rownames(
          bubble_result$clustroids[which(result_pam == i),])),
           j = 1L, value = i)
      }
    } else {
      for (i in 1:ncluster) {
        bubble_result$clusterlabels[
          label == as.integer(rownames(bubble_result$clustroids))[i],
           label := i]
      }
    }
```

```
      flist[h] <- rec_pre_f1(orig_sample,
      bubble_result$clusterlabels$label, n)$f1
      names(flist)[h] <- paste(thresholdlist[thresh], L_list[L])
    }

  }

  threshold <- as.numeric(unlist(str_split(
    names(flist[which.max(flist)]), " "))[1])
  L <- as.numeric(unlist(str_split(
    names(flist[which.max(flist)]), " "))[2])
  return(data.table(threshold = threshold, L = L))
}
```

C.4.5. MONA

```
find_central_variable3 <- function(dataset_fcv, ohne_split = NULL,
 ncores, ncols) {

  vars <- names(dataset_fcv)

  dataset_fcv <- as.matrix(dataset_fcv)
  colnames(dataset_fcv) <- NULL

  ohne <- ohne_split
  if (!is.null(ohne)) {
    ohne <- as.numeric(gsub("V", "", ohne))
    ohne_split <- as.numeric(gsub("V", "", ohne_split))
  }


  cluster_n <- nrow(dataset_fcv)
  if (is.null(ohne)) {
    nvars <- ncols
    a <- colSums2(dataset_fcv)
  } else {
    nvars <- length((1:ncols)[-ohne])
    a <- colSums2(dataset_fcv[, (1:ncols)[-ohne]])
  }
```

```
I_left <- nvars * cluster_n * log(cluster_n)
I_right <- sum(a * log(a) + (cluster_n - a) * log(cluster_n - a))
I <- I_left - I_right

if (is.null(ohne)) {
  ii <- iter(1:ncols)
} else {
  ii <- iter((1:ncols)[-ohne])
}


Is <- foreach(i = ii, .combine = c,
 .packages = "data.table") %dopar% {
  ohne_loop <- i
  ohne <- c(ohne_split, ohne_loop)


  which_n_ix <- which(dataset_fcv[, i] == 0)
  cluster_n_Ix <- length(which_n_ix)

  nvars <- ncol(dataset_fcv[, -ohne])
  I_left_Ix <- nvars * cluster_n_Ix * log(cluster_n_Ix)

  which_cols <- (1:ncols)[-ohne]

  a_Ix <- colSums2(dataset_fcv[which_n_ix, which_cols])

  if (any(a_Ix == 0)) {
    a_Ix[which(a_Ix == 0)] <- 1
  }
  I_right_Ix <- sum(a_Ix * log(a_Ix) +
   (cluster_n_Ix - a_Ix) * log(cluster_n_Ix - a_Ix))
  I_Ix <- I_left_Ix - I_right_Ix




  which_n_iy <- which(dataset_fcv[, i] == 1)
  cluster_n_Iy <- length(which_n_iy)

  I_left_Iy <- nvars * cluster_n_Iy * log(cluster_n_Iy)
```

```
    a_Iy <- colSums2(dataset_fcv[which_n_iy, which_cols])

    if (any(a_Iy == 0)) {
      a_Iy[which(a_Iy == 0)] <- 1
    }
    I_right_Iy <- sum(a_Iy * log(a_Iy) + (cluster_n_Iy - a_Iy) *
     log(cluster_n_Iy - a_Iy))
    I_Iy <- I_left_Iy - I_right_Iy

    I - I_Ix - I_Iy
  }

  if (is.null(ohne)) {
    names(Is) <- (1:ncols)
  } else {
    names(Is) <- (1:ncols)[-ohne]
  }

  return(vars[as.numeric(names(Is)[which.max(Is)])])
}

mona <- function(dataset, n_splits, nvars, ncores, ncols) {
  ohne_split <- NULL
  central_variable <- NULL
  for (i in 1:n_splits) {
    splitvars <- append(ohne_split, central_variable)
    which_splitvars <- NULL
    x <- i
    while (floor(x / 2) >= 1) {
      which_splitvars <- append(which_splitvars, floor(x / 2))
      x <- floor(x / 2)
    }

    if (is.null(splitvars)) {
      ohne_split <- NULL
    } else {
      ohne_split <- splitvars[which_splitvars]
    }

    dataset_fcv <- select(dataset, starts_with("V"))
    vars <- names(dataset_fcv)
```

```
ohne <- ohne_split
if (!is.null(ohne)) {
  ohne <- as.numeric(gsub("V", "", ohne))
  ohne_split <- as.numeric(gsub("V", "", ohne_split))
}

cluster_n <- dataset_fcv[, .N]
if (is.null(ohne)) {
  nvars <- ncols
  a <- dataset_fcv[, lapply(.SD, sum)]
} else {
  nvars <- length((1:ncols)[-ohne])
  a <- dataset_fcv[, lapply(.SD, sum),
   .SDcols = names(dataset_fcv)[-ohne]]
}

I_left <- nvars * as.numeric(cluster_n) * log(cluster_n)
I_right <- sum(a * log(a) + as.numeric((cluster_n - a)) *
 log(cluster_n - a))
I <- I_left - I_right

Is <- foreach(i = if (is.null(ohne)) iter(
  1:ncols) else iter((1:ncols)[-ohne]),
 .combine = c) %dopar% {
  ohne_loop <- i
  ohne1 <- c(ohne_split, ohne_loop)

  which_n_ix <- dataset_fcv[, which(get(paste0("V", i)) == 0)]
  cluster_n_Ix <- length(which_n_ix)

  nvars1 <- dataset_fcv[, ncol(.SD),
   .SDcols = names(dataset_fcv)[-ohne1]]
  I_left_Ix <- as.numeric(nvars1) * as.numeric(cluster_n_Ix) *
   as.numeric(log(cluster_n_Ix))
  which_cols <- (1:ncols)[-ohne1]

  a_Ix <- dataset_fcv[which_n_ix, lapply(.SD, sum),
   .SDcols = names(dataset_fcv)[which_cols]]

  if (any(a_Ix == 0)) {
    a_Ix[which(a_Ix == 0)] <- 1
  }
```

```
  I_right_Ix <- sum(a_Ix * log(a_Ix) +
  as.numeric((cluster_n_Ix - a_Ix)) *
   log(cluster_n_Ix - a_Ix))
  I_Ix <- I_left_Ix - I_right_Ix

  which_n_iy <- dataset_fcv[, which(get(paste0("V", i)) == 1)]
  cluster_n_Iy <- length(which_n_iy)

  I_left_Iy <- nvars1 * as.numeric(cluster_n_Iy) * log(cluster_n_Iy)

  a_all <- dataset_fcv[, lapply(.SD, sum),
   .SDcols = names(dataset_fcv)[which_cols]]
  a_Iy <- a_all - a_Ix

  if (any(a_Iy == 0)) {
    a_Iy[which(a_Iy == 0)] <- 1
  }
  I_right_Iy <- sum(a_Iy * as.numeric(log(a_Iy)) +
  as.numeric((cluster_n_Iy - a_Iy)) * log(cluster_n_Iy - a_Iy))
  I_Iy <- I_left_Iy - I_right_Iy

  I - I_Ix - I_Iy
}

if (is.null(ohne)) {
  names(Is) <- (1:ncols)
} else {
  names(Is) <- (1:ncols)[-ohne]
}

central_variable <- vars[as.numeric(names(Is)[which.max(Is)])]

if (i == 1) {
  dataset[, paste0("split", i) := fcase(get(central_variable) == 0, 0,
                                        get(central_variable) == 1, 1)]
} else {
  dataset[get(paste0("split",
  floor(i / 2))) == ifelse(i %% 2 == 1, 1, 0),
   paste0("split", i) := fcase(get(central_variable) == 0, 0,
   get(central_variable) == 1, 1)]
}
```

```
  }

  splits_for_clusters <- (n_splits + 1) / 2
  splits_for_clusters <-
   names(dataset)[(length(names(dataset)) -
   splits_for_clusters + 1):length(names(dataset))]
  dataset[, clusters := .GRP, by = c(splits_for_clusters)]
  return(dataset$clusters)
}
```

C.4.6. Multibit Trees

```
find_balanced_variable <- function(dataset) {
  paste0("V", as.character(which.min(abs(dataset[,
   lapply(.SD, sum)] - dataset[, .N] / 2))))
}

mbt <- function(dataset, threshold, nclusters) {
  size <- dataset[, .N]
  partitions <- data.table()
  nrow_partition <- threshold + 1

  central_variable <- find_balanced_variable(select(dataset,
   starts_with("V")))
  dataset[, partition := fcase(get(central_variable) == 0, 0,
                               get(central_variable) == 1, 1)]

  stopit <- FALSE
  nrow_partition2 <- 1
  while (any(nrow_partition >= threshold)) {
    n_prior <- dataset[, .N]
    dataset <- split(dataset, by = "partition")
    ii <- iter(dataset)
    dataset <- foreach(dataset = ii) %do% {
      central_variable <- find_balanced_variable(
        select(dataset, starts_with("V")))
      dataset[, partition := fcase(get(
        central_variable) == 0, sample(1:1e10, 1),
                                   get(
        central_variable) == 1, sample(1:1e10, 1))]
```

```
  }

  dataset <- rbindlist(dataset)
  nrow_partition <- dataset[, .N, by = partition]

  # if central variable is constant, it will be an infinite loop,
  # because it doesnt get split nrow_partition[, .N] will be 1
  # this will stop it
  if (nrow_partition[, .N] == 1) {
    stopit <- TRUE
  }

  if (any(nrow_partition$N <= threshold)) {
    partitions <- rbindlist(list(partitions,
     dataset[partition %in% nrow_partition[N <= threshold,
      partition]]))
    dataset <- dataset[partition %in% nrow_partition
    [N > threshold, partition]]
  }
  nrow_partition <- dataset[, .N, by = partition]$N

  # if central variable is constant, it will be an infinite loop,
  # because it doesnt get split.
  # if more than one partition does not get split anymore
  # this will stop it. the remaining partitions will then be clusters
  if (identical(nrow_partition, nrow_partition2)) {
    stopit <- TRUE
  }

  # needed to check if it does not split
  nrow_partition2 <- nrow_partition

  if (stopit) {
    nrow_partition <- threshold - 1
  }

}

# in case of central variable is constant,
# the remaining data will not get split and regarded as one cluster
# needs to be rbinded to partitions
if (partitions[, .N] < size) {
```

```
    dataset[, partition := sample(1:1e10, 1), by = partition]
    partitions <- rbindlist(list(partitions, dataset))
  }

  medoids <- partitions[, .SD[sample(1:.N, 1)], by = partition]
  result <- kmeans(medoids, nclusters)$cluster

  clustertable <- data.table(medoids = medoids$partition,
   clusters = result)

  partitions <- merge(partitions, clustertable,
   by.x = "partition", by.y = "medoids")

  return(partitions$clusters)
}
```

C.4.7. CLOPE

```
deltaadd <- function(cf, x, r) {
  Snew <- cf$S + sum(x)
  Worignew <- cf$Worig + x
  Wnew <- length(which(Worignew != 0))
  Nnew <- cf$N + 1
  delta <- (Snew * Nnew / Wnew^r) -
  (cf$S * cf$N / cf$W^r)
  return(list(Snew = Snew, Wnew = Wnew,
   Worignew = Worignew, Nnew = Nnew,
    delta = delta))
}

deltaaddcluster <- function(cf, x) {
  Snew <- cf$S + x$S
  Worignew <- cf$Worig + x$Worig
  Wnew <- length(which(Worignew != 0))
  Nnew <- cf$N + x$N
  delta <- (Snew * Nnew / Wnew^r) -
   (cf$S * cf$N / cf$W^r)
  return(list(Snew = Snew, Wnew = Wnew,
  Worignew = Worignew, Nnew = Nnew,
  delta = delta))
```

```
}

clope_par <- function(dataset, r, ncores,
 maxiteration, maxmoved, clusterstop) {
  r <- 2

  results_clope <- foreach(dataset_iter = isplitRows(
    dataset, chunks = ncores),
   .combine = append) %dopar% {
    # clusterids
    clusters <- NULL
    clusters[1] <- 1
    maxcluster <- 1

    # first observation constitutes the first cluster
    clusterfeatures <- list()
    clusterfeatures[[1]] <- list(S = sum(dataset_iter[1,]),
                                 W = sum(dataset_iter[1,]),
                                 Worig = dataset_iter[1,],
                                 N = 1)

    for (i in 2:nrow(dataset_iter)) {
      # results for deltaadd
      deltaadd_results <- lapply(clusterfeatures,
       deltaadd, dataset_iter[i,], r)
      deltas <- sapply(deltaadd_results,
      function(x) x$delta)
      # results for deltaadd in case of new cluster
      if (length(deltas) < clusterstop) {
        deltanewcluster <- (sum(dataset_iter[i,]) /
         (sum(dataset_iter[i,]))^r)
        # to which cluster will the observation be added
        clusterwhich <- which.max(c(deltas, deltanewcluster))
      } else {
        # to which cluster will the observation be added
        clusterwhich <- which.max(deltanewcluster)
      }

      # in case of new cluster, add new cluster to
      # clusterfeatures and clusters
      if (clusterwhich > maxcluster) {
        clusterfeatures[[clusterwhich]] <- list(
```

```
        S = sum(dataset_iter[i,]),
        W = sum(dataset_iter[i,]),
        Worig = dataset_iter[i,],
        N = 1)
      clusters[i] <- clusterwhich
      maxcluster <- clusterwhich
    } else {
      # allocate observation to cluster
      clusters[i] <- clusterwhich
      # update clusterfeatures
      clusterfeatures[[clusterwhich]] <- list(
        S = deltaadd_results[[clusterwhich]]$Snew,
        W = deltaadd_results[[clusterwhich]]$Wnew,
        Worig = deltaadd_results[[clusterwhich]]$Worignew,
        N = deltaadd_results[[clusterwhich]]$Nnew)
    }
  }

  maxiter <- 0
  moved <- Inf
  while (maxiter <= maxiteration & moved > maxmoved) {
    maxiter <- maxiter + 1
    moved <- 0

    for (i in 1:nrow(dataset_iter)) {

      # clus_temp for checking if moved
      clus_temp <- clusters[i]

      # subtract case i from clusterfeature
      clusterfeatures[[clusters[i]]]$S <-
      clusterfeatures[[clusters[i]]]$S -
      sum(dataset_iter[i,])
      clusterfeatures[[clusters[i]]]$Worig <-
      clusterfeatures[[clusters[i]]]$Worig -
      dataset_iter[i,]
      clusterfeatures[[clusters[i]]]$W <-
      length(which(clusterfeatures[[
        clusters[i]]]$Worig != 0))
      clusterfeatures[[clusters[i]]]$N <-
      clusterfeatures[[clusters[i]]]$N - 1
```

```
# if cf is empty now -> delete
if (clusterfeatures[[clusters[i]]]$N == 0) {
  clusterfeatures[[clusters[i]]] <- NULL
  # subtract 1 from clusters higher than
  # clusters[i] because the cluster was deleted
  clusters[clusters > clusters[i]] <-
  clusters[clusters > clusters[i]] - 1
}

# results for deltaadd
deltaadd_results <- lapply(clusterfeatures,
 deltaadd, dataset_iter[i,], r)
deltas <- sapply(deltaadd_results, function(x) x$delta)
# results for deltaadd in case of new cluster
if (length(deltas) < clusterstop) {
  deltanewcluster <- (sum(dataset_iter[i,]) /
   (sum(dataset_iter[i,]))^r)
  # to which cluster will the observation be added
  clusterwhich <- which.max(c(deltas, deltanewcluster))
} else {
  # to which cluster will the observation be added
  clusterwhich <- which.max(deltanewcluster)
}

# check if moved
if (clusterwhich != clus_temp) {
  moved <- moved + 1
}

# in case of new cluster, add new cluster
# to clusterfeatures and clusters
if (clusterwhich > length(clusterfeatures)) {
  clusterfeatures[[clusterwhich]] <- list(
    S = sum(dataset_iter[i,]),
    W = sum(dataset_iter[i,]),
    Worig = dataset_iter[i,],
    N = 1)
  clusters[i] <- clusterwhich
  maxcluster <- clusterwhich
} else {
  # allocate observation to cluster
  clusters[i] <- clusterwhich
```

```
        # update clusterfeatures
        clusterfeatures[[clusterwhich]] <- list(
          S = deltaadd_results[[clusterwhich]]$Snew,
          W = deltaadd_results[[clusterwhich]]$Wnew,
          Worig = deltaadd_results[[clusterwhich]]$Worignew,
          N = deltaadd_results[[clusterwhich]]$Nnew)
      }
    }
  }
  list(clusterfeatures = clusterfeatures,
   clusters = clusters)
}

clusterfeatures <- results_clope[seq(1, (2 * ncores) - 1, 2)]

clusters <- list()
# make clusterlabels unique
h <- 0
for (i in 1:length(results_clope[seq(2, 2 * ncores, 2)])) {
  results_clope[seq(2, 2 * ncores, 2)][[i]] <-
   results_clope[seq(2, 2 * ncores, 2)][[i]] + h
  h <- max(results_clope[seq(2, 2 * ncores, 2)][[i]])

}

clusters <- as.integer(unlist(results_clope[
  seq(2, 2 * ncores, 2)]))

clusterf <- list()
h <- 0
for (i in 1:length(clusterfeatures)) {

  clusterf[(h + 1):(h + length(clusterfeatures[[i]]))] <-
   clusterfeatures[[i]][1:(length(clusterfeatures[[i]]))]
  h <- h + length(clusterfeatures[[i]])

}

clusterfeatures2 <- list()
clusterfeatures2[[1]] <- clusterf[[1]]
maxcluster <- 1
```

```
  for (i in 2:length(clusterf)) {

    deltaadd_results <- lapply(clusterfeatures2,
     deltaaddcluster, clusterf[[i]])
    deltas <- sapply(deltaadd_results, function(x) x$delta)

    if (length(deltas) < clusterstop) {
      deltanewcluster <- (clusterf[[i]]$S *
       clusterf[[i]]$N) / ((clusterf[[i]]$W)^r)
      # to which cluster will the observation be added
      clusterwhich <- which.max(c(deltas, deltanewcluster))
    } else {
      # to which cluster will the observation be added
      clusterwhich <- which.max(deltanewcluster)
    }

    # in case of new cluster, add new cluster to clusterfeatures2
    if (clusterwhich > maxcluster) {
      clusterfeatures2[[clusterwhich]] <- clusterf[[i]]

      # clusters[i] <- clusterwhich
      maxcluster <- clusterwhich
    } else {
      # allocate observation to cluster
      clusters[clusters == i] <- unique(clusters)[clusterwhich]
      # update clusterfeatures
      clusterfeatures2[[clusterwhich]] <- list(
        S = deltaadd_results[[clusterwhich]]$Snew,
        W = deltaadd_results[[clusterwhich]]$Wnew,
        Worig = deltaadd_results[[clusterwhich]]$Worignew,
        N = deltaadd_results[[clusterwhich]]$Nnew)
    }

  }

  return(clusters)
}
```

C.4.8. Large Item

```
cost <- function(cf_i, x, minsupport, cf, w, cf_length) {
  support_new <- cf[[cf_i]]$support + x
  N_new <- cf[[cf_i]]$N + 1
  minsup_new <- N_new * minsupport
  large_new <- which(support_new >= minsup_new)
  small_new <- which(support_new < minsup_new & support_new > 0)

  # if there is only one cluster
  if (cf_length == 1) {

    intra <- length(small_new)
    # since there is only one
    # cluster sum(large) - union(large) is always 0
    inter <- 0
    cost_return <- w * intra + inter

  } else {

    small_cfs <- unique(unlist(lapply(cf[-cf_i],
     function(y) y$small)))
    intra <- length(union(small_new, small_cfs))
    large_cfs <- unique(unlist(lapply(cf[-cf_i],
     function(y) y$large)))
    inter <- sum(sapply(cf[-cf_i], function(x) length(x$large)) +
    length(large_new)) - length(union(large_new, large_cfs))
    cost_return <- w * intra + inter
  }

  return(list(support = support_new, N = N_new,
  minsup = minsup_new, large = large_new,
  small = small_new, cost_return = cost_return))
}

cost_single <- function(x, minsupport, cf, w, cf_length) {
  support <- x
  N <- 1
  minsup <- minsupport * 1
  large <- which(x == 1)
  small <- integer(0)

  if (cf_length == 1) {
    intra <- length(union(cf[[1]]$small, small))
```

```
    inter <- (length(cf[[1]]$large) + length(large)) -
     length(union(cf[[1]]$large, large))
    cost_return <- w * intra + inter

  } else {
    small_cfs <- unique(unlist(lapply(cf,
     function(y) y$small)))
    intra <- length(union(small, small_cfs))
    large_cfs <- unique(unlist(lapply(cf,
    function(y) y$large)))
    inter <- sum(sapply(cf, function(x) length(x$large)) +
    length(large)) - length(union(large, large_cfs))
    cost_return <- w * intra + inter
  }

  return(list(support = support, N = N, minsup = minsup,
   large = large, small = small, cost_return = cost_return))
}

cost_cluster <- function(cf_i, x, minsupport, cf, w, cf_length) {
  support_new <- cf[[cf_i]]$support + x$support
  N_new <- cf[[cf_i]]$N + x$N
  minsup_new <- N_new * minsupport
  large_new <- which(support_new >= minsup_new)
  small_new <- which(support_new < minsup_new & support_new > 0)

  # if there is only one cluster
  if (cf_length == 1) {

    intra <- length(small_new)
    # since there is only one cluster
    # sum(large) - union(large) is always 0
    inter <- 0
    cost_return <- w * intra + inter

  } else {

    small_cfs <- unique(unlist(lapply(cf[-cf_i],
     function(y) y$small)))
    intra <- length(union(small_new, small_cfs))
    large_cfs <- unique(unlist(lapply(cf[-cf_i],
     function(y) y$large)))
```

```
    inter <- sum(sapply(cf[-cf_i],
     function(x) length(x$large)) + length(large_new)) -
     length(union(large_new, large_cfs))
    cost_return <- w * intra + inter
  }

  return(list(support = support_new, N = N_new,
   minsup = minsup_new, large = large_new, small = small_new,
    cost_return = cost_return))
}

cost_single_cluster <- function(x, minsupport, cf, w, cf_length) {
  support <- x$support
  N <- x$N
  minsup <- minsupport * x$N
  large <- which(support >= minsup)
  small <- which(support < minsup & support > 0)

  if (cf_length == 1) {
    intra <- length(union(cf[[1]]$small, small))
    inter <- (length(cf[[1]]$large) + length(large)) -
     length(union(cf[[1]]$large, large))
    cost_return <- w * intra + inter

  } else {
    small_cfs <- unique(unlist(lapply(cf,
    function(y) y$small)))
    intra <- length(union(small, small_cfs))
    large_cfs <- unique(unlist(lapply(cf,
    function(y) y$large)))
    inter <- sum(sapply(cf, function(x) length(x$large)) +
     length(large)) - length(union(large, large_cfs))
    cost_return <- w * intra + inter
  }

  return(list(support = support, N = N, minsup = minsup,
   large = large, small = small, cost_return = cost_return))
}

largeitem_par <- function(dataset, w, minsupport, maxiteration,
 maxmoved, ncores) {
```

```
results_largeitem <- foreach(dataset_iter = isplitRows(dataset,
 chunks = ncores), .combine = append) %dopar% {
  # clusterids
  clusters <- NULL
  clusters[1] <- 1
  maxcluster <- 1

  # first observation constitutes the first cluster
  clusterfeatures <- list()
  clusterfeatures[[1]] <- list(support = dataset_iter[1,],
                               N = 1,
                               minsup = minsupport * 1,
                               large = which(dataset_iter[1,] == 1),
                               small = integer(0))

  for (i in 2:nrow(dataset_iter)) {

    # calculate cost for every cluster if observation is added
    cost_result <- lapply(1:length(clusterfeatures), cost,
     dataset_iter[i,], minsupport, clusterfeatures,
      w, cf_length = length(clusterfeatures))
    costs <- sapply(cost_result, function(x) x$cost_return)
    # cost if observations forms new cluster
    cost_single_result <- cost_single(dataset_iter[i,],
     minsupport, clusterfeatures, w, length(clusterfeatures))
    costs <- append(costs, cost_single_result$cost_return)


    clusterwhich <- which.min(costs)
    clusters[i] <- clusterwhich

    # if observation gets added to existing cluster,
    # update the clusterfeatures
    if (clusterwhich <= maxcluster) {
      clusterfeatures[[clusterwhich]] <- list(
        support = cost_result[[clusterwhich]]$support,
        N = cost_result[[clusterwhich]]$N,
        minsup = cost_result[[clusterwhich]]$minsup,
        large = cost_result[[clusterwhich]]$large,
        small = cost_result[[clusterwhich]]$small)

      # if observation forms new cluster add
```

```
    # new entry to clusterfeatures
  } else {
    clusterfeatures[[clusterwhich]] <- list(
      support = cost_single_result$support,
      N = cost_single_result$N,
      minsup = cost_single_result$minsup,
      large = cost_single_result$large,
      small = cost_single_result$small)

    maxcluster <- clusterwhich

  }
}

maxiter <- 0
moved <- Inf
while (maxiter <= maxiteration & moved > maxmoved) {
  maxiter <- maxiter + 1
  moved <- 0

  for (i in 1:nrow(dataset_iter)) {

    # clus_temp for checking if moved
    clus_temp <- clusters[i]

    # subract observation from cluster
    clusterfeatures[[clusters[i]]] <- list(
      support = clusterfeatures[[clusters[i]]]$support -
       dataset_iter[i,],
      N = clusterfeatures[[clusters[i]]]$N - 1,
      minsup = (clusterfeatures[[clusters[i]]]$N - 1) *
      minsupport,
      large = which(clusterfeatures[[clusters[i]]]$support -
       dataset_iter[i,] >= (clusterfeatures[[clusters[i]]]$N - 1) *
        minsupport),
      small = which(clusterfeatures[[clusters[i]]]$support -
       dataset_iter[i,] < (clusterfeatures[[clusters[i]]]$N - 1) *
        minsupport &
       clusterfeatures[[clusters[i]]]$support - dataset_iter[i,] > 0))

    # calculate cost_result for every
    # clustering if observation is added to any cluster
```

```
      cost_result <- lapply(1:length(clusterfeatures),
      cost, dataset_iter[i,], minsupport, clusterfeatures, w,
       cf_length = length(clusterfeatures))
      costs <- sapply(cost_result, function(x) x$cost_return)

      clusterwhich <- which.min(costs)
      clusters[i] <- clusterwhich

      clusterfeatures[[clusterwhich]] <- list(
        support = cost_result[[clusterwhich]]$support,
        N = cost_result[[clusterwhich]]$N,
        minsup = cost_result[[clusterwhich]]$minsup,
        large = cost_result[[clusterwhich]]$large,
        small = cost_result[[clusterwhich]]$small)

      if (clus_temp != clusterwhich) {
        moved <- moved + 1
      }

    }
  }
  list(clusterfeatures = clusterfeatures, clusters = clusters)
}

clusterfeatures <- results_largeitem[seq(1, (2 * ncores) - 1, 2)]

clusters <- list()
# make clusterlabels unique
h <- 0
for (i in 1:length(results_largeitem[seq(2, 2 * ncores, 2)])) {
  results_largeitem[seq(2, 2 * ncores, 2)][[i]] <-
   results_largeitem[seq(2, 2 * ncores, 2)][[i]] + h
  h <- max(results_largeitem[seq(2, 2 * ncores, 2)][[i]])

}

clusters <- as.integer(unlist(results_largeitem[seq(2, 2 *
 ncores, 2)]))

clusterf <- list()
h <- 0
for (i in 1:length(clusterfeatures)) {
```

```
    clusterf[(h + 1):(h + length(clusterfeatures[[i]]))] <-
     clusterfeatures[[i]][1:(length(clusterfeatures[[i]]))]
    h <- h + length(clusterfeatures[[i]])

  }

  clusterfeatures2 <- list()
  clusterfeatures2[[1]] <- clusterf[[1]]
  maxcluster <- 1


  for (i in 2:length(clusterf)) {
    # calculate cost for every cluster if cluster is added
    cost_result <- lapply(1:length(clusterfeatures2),
    cost_cluster, clusterf[[i]], minsupport, clusterfeatures2,
     w, cf_length = length(clusterfeatures2))
    costs <- sapply(cost_result, function(x) x$cost_return)
    # cost if cluster forms new cluster
    cost_single_result <- cost_single_cluster(clusterf[[i]],
     minsupport, clusterfeatures2, w, length(clusterfeatures2))
    costs <- append(costs, cost_single_result$cost_return)

    clusterwhich <- which.min(costs)

    # if cluster gets added to existing cluster,
    # update the clusterfeatures
    if (clusterwhich <= maxcluster) {
      clusterfeatures2[[clusterwhich]] <- list(
        support = cost_result[[clusterwhich]]$support,
        N = cost_result[[clusterwhich]]$N,
        minsup = cost_result[[clusterwhich]]$minsup,
        large = cost_result[[clusterwhich]]$large,
        small = cost_result[[clusterwhich]]$small)

      clusters[clusters == i] <- unique(clusters)[clusterwhich]

      # if cluster forms new cluster
      # add new entry to clusterfeatures
    } else {
      clusterfeatures2[[clusterwhich]] <- list(
        support = cost_single_result$support,
```

```
        N = cost_single_result$N,
        minsup = cost_single_result$minsup,
        large = cost_single_result$large,
        small = cost_single_result$small)

      maxcluster <- clusterwhich

    }
  }

  return(clusters)
}
```

C.4.9. SCALE

```
delta_add_scale <- function(cf_i, x, cf, N_all) {
  occurence_new <- cf[[cf_i]]$occurence + x
  S_new <- cf[[cf_i]]$S + sum(x)
  N_new <- cf[[cf_i]]$N + 1
  WCD_new <- sum(occurence_new^2) / (S_new * N_new)
  EWCD <- sum(sapply(cf[-cf_i],
   function(y, N_all) {y$N / N_all * y$WCD}, N_all)) +
    ((N_new / N_all) * WCD_new)

  return(list(occurence = occurence_new, S = S_new, N = N_new,
   WCD = WCD_new, EWCD = EWCD))
}


scale_cluster <- function(dataset, nsample, ncluster,
 ncores, seed, maxiteration, maxmoved, chunk_n) {

  n_data <- nrow(dataset)

  # draw sample for initial clustering via pam
  dataset_sample_index <- sample(1:n_data, nsample)
  dataset_sample <- dataset[dataset_sample_index,]

  dataset_sample_dist <- rdist(dataset_sample, metric = "hamming")
  sample_cluster <- pam(dataset_sample, k = ncluster,
```

```
 cluster.only = TRUE)

# initial clusters
clusters <- NULL
clusterfeatures <- list()
for (i in 1:ncluster) {
  clusterfeatures[[i]] <- list(
    occurence = colSums(dataset_sample[sample_cluster == i,,
     drop = FALSE]),
     S = sum(dataset_sample[sample_cluster == i,, drop = FALSE]),
     N = sum(sample_cluster == i))

  clusterfeatures[[i]]$WCD <- sum(clusterfeatures[[i]]$occurence^2) /
   (as.numeric(clusterfeatures[[i]]$S) *
   as.numeric(clusterfeatures[[i]]$N))
}

# generate random sequence for processing observations
rnd_orig <- sample(1:nrow(dataset), nrow(dataset))
# minus the observations in sample_cluster
# (they are already in a cluster)
rnd <- setdiff(rnd_orig, dataset_sample_index)

# dataset <- dataset[-dataset_sample_index,]

# create chunks for processing in parallel.
rndseq <- seq(1, length(rnd), ncores * chunk_n)

# phase 1
for (i in 1:length(rndseq)) {

  if (i != length(rndseq)) {
    dataiter <- isplitRows(dataset[rnd[rndseq[i]:(rndseq[i + 1]- 1)],],
     chunks = ncores)
  } else {
    dataiter <- isplitRows(dataset[rnd[rndseq[i]:length(rnd)],],
     chunks = ncores)
  }

  # chunk_n observations get processed at a time
  cluster_result <- foreach(bdata = dataiter, .combine = c,
   .export = c("clusterfeatures")) %dopar% {
```

```
  clusterwhich <- NULL
  for (j in 1:nrow(bdata)) {
    EWCD_result <- lapply(1:length(clusterfeatures),
    delta_add_scale, bdata[j,], clusterfeatures, n_data)
    EWCD <- sapply(EWCD_result, function(x) x$EWCD)
    # to which cluster will observation be added
    clusterwhich[j] <- which.max(EWCD)

    # update clusterfeatures
    clusterfeatures[[clusterwhich[j]]] <- list(
      occurence = EWCD_result[[clusterwhich[j]]]$occurence,
      S = EWCD_result[[clusterwhich[j]]]$S,
      N = EWCD_result[[clusterwhich[j]]]$N,
      WCD = EWCD_result[[clusterwhich[j]]]$WCD)
  }
  clusterwhich
}

if (i != length(rndseq)) {
  valid_ids <- rnd[rndseq[i]:(rndseq[i + 1] - 1)]
} else {
  valid_ids <- rnd[rndseq[i]:(length(rnd))]
}


clusters[valid_ids] <- cluster_result

# get results for resulting clusterfeatures for
# clusters, where the observation has been added

# update clusterfeatures with processed cases
for (j in unique(cluster_result)) {
  clusterfeatures[[j]] <- list(
    occurence = clusterfeatures[[j]]$occurence +
     colSums(dataset[valid_ids,][which(cluster_result == j),]),
    S = clusterfeatures[[j]]$S +
     sum(dataset[valid_ids,][which(cluster_result == j),]),
    N = clusterfeatures[[j]]$N +
     nrow(dataset[valid_ids,][which(cluster_result == j),]))
  clusterfeatures[[j]]$WCD <- sum(clusterfeatures[[j]]$occurence^2) /
   (as.numeric(clusterfeatures[[j]]$S) *
    as.numeric(clusterfeatures[[j]]$N))
```

```
    }

  }

  clusters[dataset_sample_index] <- sample_cluster

  # phase 2

  maxiter <- 0
  moved <- Inf
  while (maxiter <= maxiteration & moved > maxmoved) {
    maxiter <- maxiter + 1
    moved <- 0

    for (i in 1:length(rndseq)) {

      if (i != length(rndseq)) {
        dataiter <- isplitRows(dataset[rnd[rndseq[i]:(rndseq[i + 1]- 1)],],
         chunks = ncores)
      } else {
        dataiter <- isplitRows(dataset[rnd[rndseq[i]:length(rnd)],],
         chunks = ncores)
      }


      if (i != length(rndseq)) {
        cluster_temp <- clusters[rnd[rndseq[i]:(rndseq[i + 1]- 1)]]
      } else {
        cluster_temp <- clusters[rnd[rndseq[i]:length(rnd)]]
      }

      # subtract observation from cluster
      for (j in unique(cluster_temp)) {
        if (i != length(rndseq)) {
          clusterfeatures[[j]] <- list(
            occurence = clusterfeatures[[j]]$occurence -
            colSums(dataset[rnd[rndseq[i]:(rndseq[i + 1]- 1)][which(
              clusters[rnd[rndseq[i]:(rndseq[i + 1]- 1)]] == j)],]),
            S = clusterfeatures[[j]]$S -
             sum(dataset[rnd[rndseq[i]:(rndseq[i + 1]- 1)][which(
              clusters[rnd[rndseq[i]:(rndseq[i + 1]- 1)]] == j)],]),
            N = clusterfeatures[[j]]$N -
```

```
      length(which(clusters[rnd[rndseq[i]:(rndseq[i + 1]- 1)]] == j)))
    clusterfeatures[[j]]$WCD <-
     sum(clusterfeatures[[j]]$occurence^2) /
     (as.numeric(clusterfeatures[[j]]$S) *
      as.numeric(clusterfeatures[[j]]$N))
  } else {
    clusterfeatures[[j]] <- list(
      occurence = clusterfeatures[[j]]$occurence -
       colSums(dataset[rnd[rndseq[i]:length(rnd)][which(
        clusters[rnd[rndseq[i]:length(rnd)]] == j)],]),
      S = clusterfeatures[[j]]$S -
       sum(dataset[rnd[rndseq[i]:length(rnd)][which(
        clusters[rnd[rndseq[i]:length(rnd)]] == j)],]),
      N = clusterfeatures[[j]]$N -
       length(which(clusters[rnd[rndseq[i]:length(rnd)]] == j)))
    clusterfeatures[[j]]$WCD <-
     sum(clusterfeatures[[j]]$occurence^2) /
     (as.numeric(clusterfeatures[[j]]$S) *
      as.numeric(clusterfeatures[[j]]$N))
  }
}

# chunk_n observations get processed at a time
cluster_result <- foreach(bdata = dataiter,
 .combine = c, .export = c("clusterfeatures")) %dopar% {
  clusterwhich <- NULL
  for (j in 1:nrow(bdata)) {

    EWCD_result <- lapply(1:length(clusterfeatures),
     delta_add_scale, bdata[j,], clusterfeatures, n_data)
    EWCD <- sapply(EWCD_result, function(x) x$EWCD)
    # to which cluster will observation be added
    clusterwhich[j] <- which.max(EWCD)

    # update clusterfeatures
    clusterfeatures[[clusterwhich[j]]] <- list(
      occurence = EWCD_result[[clusterwhich[j]]]$occurence,
      S = EWCD_result[[clusterwhich[j]]]$S,
      N = EWCD_result[[clusterwhich[j]]]$N,
      WCD = EWCD_result[[clusterwhich[j]]]$WCD)
  }
  clusterwhich
```

```
      }

      if (any(cluster_temp != cluster_result)) {
        moved <- moved + sum(cluster_temp != cluster_result)
      }

      if (i != length(rndseq)) {
        valid_ids <- rnd[rndseq[i]:(rndseq[i + 1] - 1)]
      } else {
        valid_ids <- rnd[rndseq[i]:(length(rnd))]
      }

      clusters[valid_ids] <- cluster_result

      # update clusterfeatures with processed cases
      for (j in unique(cluster_result)) {
        clusterfeatures[[j]] <- list(
          occurence = clusterfeatures[[j]]$occurence +
           colSums(dataset[valid_ids,][which(cluster_result == j),]),
          S = clusterfeatures[[j]]$S +
           sum(dataset[valid_ids,][which(cluster_result == j),]),
          N = clusterfeatures[[j]]$N +
           nrow(dataset[valid_ids,][which(cluster_result == j),]))
        clusterfeatures[[j]]$WCD <- sum(clusterfeatures[[j]]$occurence^2) /
         (as.numeric(clusterfeatures[[j]]$S) *
          as.numeric(clusterfeatures[[j]]$N))
      }
    }
  }
  return(clusters)
}
```

C.4.10. Sorted Neighborhood

```
sorted_neighborhood <- function(dataset, windowsize,
threshold = mean(pdist(dataset[sample(1:nrow(dataset), 100),],
metric = "hamming")),
runs, allrandom = TRUE, ncores) {

  rownames(dataset) <- 1:nrow(dataset)
```

```
clusids <- list()

run <- 1
while (run <= runs) {
  if (run == 1 && !allrandom) {
    dataset <- dataset[order(rowSums(dataset)),]
  } else {
    dataset <- dataset[sample(1:nrow(dataset), nrow(dataset)),]
  }

  clusids_par <- foreach(dataset_iter = isplitRows(dataset,
   chunks = ncores),
   .combine = append) %dopar% {
    clusters <- 1
    dist_i <- rdist:::hamming_cdist(dataset_iter[2,, drop = FALSE],
     dataset_iter[1,, drop = FALSE])
    which_cases <- which(dist_i < threshold)
    clusters[2] <- ifelse(!length(which_cases), 2,
     clusters[which_cases[1]])

    for (i in 3:windowsize) {
      dist_i <- rdist:::hamming_cdist(dataset_iter[i,, drop = FALSE],
       dataset_iter[1:(i - 1),])
      which_case_min <- which.min(dist_i)
      tresh <- dist_i[which_case_min] < threshold
      clusters[i] <- ifelse(tresh, clusters[which_case_min], i)
    }

    for (i in (windowsize + 1):nrow(dataset_iter)) {
      dist_i <- rdist:::hamming_cdist(dataset_iter[i,, drop = FALSE],
       dataset_iter[(i - windowsize):(i - 1),])
      which_case_min <- which.min(dist_i)
      tresh <- dist_i[which_case_min] < threshold
      clusters[i] <- ifelse(tresh,
       clusters[which_case_min + (i - windowsize - 1)], i)
    }

    clusterids <- list()
    h <- 0
    for (i in unique(clusters)) {
      h <- h + 1
      clusterids[[h]] <-
```

```
        rownames(dataset_iter[clusters == i,, drop = FALSE])
      }
      clusterids
    }

    clusids <- append(clusids, clusids_par)
    run <- run + 1
  }

  # transitive hull ---------------------------------------------------

  # remove singletons (not needed for transitive hull)
  clusids_without_singletons <- clusids[sapply(clusids, length) > 1]

  transmat <- matrix(nrow = length(clusids_without_singletons),
   ncol = length(clusids_without_singletons))

  transmat <- foreach (i = 1:length(clusids_without_singletons),
   .combine = rbind, .export = "transmat") %dopar% {
    for (j in i:length(clusids_without_singletons)) {
      if (i != j && any(clusids_without_singletons[[i]] %in%
       clusids_without_singletons[[j]])) {
        transmat[i, j] <- 1
      }
    }
    transmat[i,]
  }

  transmat <- as.matrix(forceSymmetric(transmat))
  transhull <- allShortestPaths(transmat)$length

  # union of the ids that belong together according to transitive hull
  unionids <- list()
  donelist <- NULL
  h <- 0
  for (i in 1:ncol(transhull)) {
    if (!i %in% donelist) {
      h <- h + 1
      unionids[[h]] <-
       unique(unlist(clusids_without_singletons[
        which(transhull[, i] > 0)]))
      donelist <- append(donelist, which(transhull[, i] > 0))
```

```
    }
  }

  # add the singletons
  unionids <- append(unionids, clusids[sapply(clusids, length) == 1])

  for (i in 1:length(unionids)) {
    for (j in i:length(unionids)) {
      if (i != j && length(unionids[[j]]) == 1) {
        if (unionids[[j]] %in% unionids[[i]]) {
          unionids[[j]] <- "0"
        }
      }
    }
  }

  unionids[which(sapply(unionids, function(x) x) == "0")] <- NULL

  unionids <- lapply(unionids, as.integer)
  clusters <- NULL
  for (i in 1:length(unionids)) {
    clusters[unionids[[i]]] <- i
  }

  return(clusters)
}

# cross

cross_sorted <- function(dataset, ncluster, orig, ncores) {

  n <- nrow(dataset)
  datasample <- matrix(nrow = 2000, ncol = ncol(dataset))
  boundsample <- round(2000 / ncluster)
  boundssample <- round(seq(0, 2000, length.out = ncluster + 1))

  for (i in 1:ncluster) {
    which_sample <- sample(1:floor(n / ncluster),
     boundssample[i + 1] - boundssample[i])
    datasample[(boundssample[i] + 1):boundssample[i + 1],] <-
     dataset[which(orig == i),][which_sample,]
  }
```

```
  shuf_index <- sample(1:nrow(datasample), nrow(datasample))
  datasample <- datasample[shuf_index,]

  orig_sample <- integer()
  for (i in 1:ncluster) {
    orig_sample <- c(orig_sample,
     rep(i, round(nrow(datasample) / ncluster)))
  }

  orig_sample <- orig_sample[shuf_index]

  thresholdlist <- seq(0.3, 0.8, 0.1)

  randlist <- NULL

  for (i in 1:length(thresholdlist)) {
    sorted_neighborhood_result <-
     sorted_neighborhood(datasample, windowsize = 30,
      thresholdlist[i], runs = 3, allrandom = TRUE, ncores)
    randlist[i] <- rand(sorted_neighborhood_result,
     orig_sample, nrow(datasample))
  }

  threshold <- thresholdlist[which.max(randlist)]
  return(threshold)
}
```

C.4.11. Proximus

```
prox <- function(dataset, maxradius, ncores, ncluster) {

  dataset <- as.data.table(dataset)
  for (i in 1:ncol(dataset)) {
    set(dataset, j = i, value = as.logical(dataset[[i]]))
  }
  dataset <- as.matrix(dataset)

  par_result <-
   foreach(dat = isplitRows(dataset, chunks = ncores)) %dorng% {
    pr <- cba::proximus(dat, max.radius = maxradius)
```

```
    fits <- as.integer(fitted(pr)$pl)
    patterns <- lapply(pr$a, function(x) x$y)
    list(pr = fits, patterns = patterns)
  }

  patterns <- lapply(par_result, function(x) x$patterns)
  patterns <- unlist(patterns, recursive = FALSE)
  patternmat <- matrix(0, nrow = length(patterns), ncol = ncol(dataset))

  for (i in 1:length(patterns)) {
    patternmat[i, patterns[[i]]] <- 1
  }

  par_result <- lapply(par_result, function(x) x$pr)

  # make clusterids unique
  maxclusid <- max(unique(par_result[[1]]))
  for (i in 2:length(par_result)) {
    par_result[[i]] <- par_result[[i]] + maxclusid
    maxclusid <- max(unique(par_result[[i]]))
  }

  clusters <- unlist(par_result)
  result_cluster <- "more cluster centers than distinct data points"
  clustercenters <- ncluster
  print(paste("prox while", Sys.time()))
  while (is.character(result_cluster)) {
    result_cluster <- tryCatch(
      kmeans(patternmat, clustercenters)$cluster,
      error = function(x) return(
        "more cluster centers than distinct data points"))
    clustercenters <- clustercenters - 1
  }
  rm(clustercenters)

  clusters <- as.data.table(clusters)
  clusters[, clusters := result_cluster[clusters[1]], by = clusters]

  return(clusters)
}

cross_prox <- function(dataset, ncluster, orig) {
```

```
n <- nrow(dataset)

datasample <- matrix(nrow = 2000, ncol = ncol(dataset))
boundsample <- round(2000 / ncluster)
boundssample <- round(seq(0, 2000, length.out = ncluster + 1))
for (i in 1:ncluster) {
  which_sample <- sample(1:floor(n / ncluster),
   boundssample[i + 1] - boundssample[i])
  datasample[(boundssample[i] + 1):boundssample[i + 1],] <-
   dataset[which(orig == i),][which_sample,]
}

shuf_index <- sample(1:nrow(datasample), nrow(datasample))
datasample <- datasample[shuf_index,]

orig_sample <- integer()
for (i in 1:ncluster) {
  orig_sample <- c(orig_sample,
   rep(i, round(nrow(datasample) / ncluster)))
}

orig_sample <- orig_sample[shuf_index]

datasample <- as.data.table(datasample)
for (i in 1:ncol(datasample)) {
  set(datasample, j = i, value = as.logical(datasample[[i]]))
}
datasample <- as.matrix(datasample)

maxradiusmean <- round(mean(rowSums(datasample)))

maxradiusmean10 <- maxradiusmean / 10

maxradiuslist <- c(seq(maxradiusmean -
 5 * maxradiusmean10, maxradiusmean, maxradiusmean10),
                   seq(maxradiusmean +
                   maxradiusmean10, maxradiusmean +
                   5 * maxradiusmean10, maxradiusmean10))

randlist <- NULL
```

```
  for (i in 1:length(maxradiuslist)) {
    prox_result <- cba::proximus(datasample,
    max.radius = maxradiuslist[i])
    prox_result <- as.integer(fitted(prox_result)$pl)
    randlist[i] <- rand(prox_result, orig_sample,
     nrow(datasample))
  }

  maxradius <- maxradiuslist[which.max(randlist)]
  return(maxradius)
}
```

C.4.12. Canopy Clustering

```
canopy_cluster <- function(dataset, t1, t2) {

  int2 <- sample(1:nrow(dataset), nrow(dataset))
  canopys <- list()
  centers <- NULL
  h <- 1

  # if legnth(int2) == 0 every observation is in canopy
  while (length(int2) > 0) {
    # i is random observation
    i <- int2[1]
    # dist from center to all eligible observations
    dmat <- rdist:::hamming_cdist(dataset[i,, drop = FALSE],
     dataset[int2,, drop = FALSE])
    colnames(dmat) <- int2
    # canopys is list with canopys as list elements,
    # which conist of the indezes of the observation inside
    canopys[[h]] <- as.integer(colnames(dmat)[which(dmat < t1)])
    h <- h + 1
    # which observation must be removed
    int2 <- int2[-which(int2 %in%
     as.integer(colnames(dmat)[which(dmat < t2)]))]
  }

  clusters <- NULL
  for (i in 1:length(canopys)) {
```

```
    clusters[canopys[[i]]] <- i
  }
  return(clusters)
}


canopy_cluster_par <- function(dataset, t1, t2,
 ncluster, ncores) {

  cano <- foreach(dataset_iter = isplitRows(dataset,
   chunks = ncores)) %dorng% {

    int2 <- sample(1:nrow(dataset_iter), nrow(dataset_iter))
    canopys <- list()
    centers <- NULL
    h <- 1
    while (length(int2) > 0) {
      # i is random observation
      i <- int2[1]
      # dist from center to all eligible observations
      if (h == 1) {
        dmat <- rdist:::hamming_cdist(dataset_iter[i,, drop = FALSE],
         dataset_iter)
        colnames(dmat) <- 1:length(dmat)
      } else {
        dmat <- rdist:::hamming_cdist(dataset_iter[i,, drop = FALSE],
         dataset_iter[int2,, drop = FALSE])
        colnames(dmat) <- int2
      }
      # canopys is list with canopys as list elements,
      # which conist of the indezes of the observation inside
      canopys[[h]] <- as.integer(colnames(dmat)[which(dmat < t1)])
      names(canopys)[h] <- as.character(i)
      h <- h + 1
      int2 <- int2[-which(int2 %in%
       as.integer(colnames(dmat)[which(dmat < t2)]))]
    }
    canopys
  }

  chunksizes <- sapply(cano, function(x) {
    y <- NULL
```

```
    for (i in 1:length(x)) {
      y[i] <- length(x[[i]])
    }
    return(sum(y))
  })

  chunksizes_cum <- cumsum(chunksizes)
  chunksizes_cum <- chunksizes_cum[-length(chunksizes_cum)]

  for (i in 2:length(cano)) {
    cano[[i]] <- rapply(cano[[i]], function(x, y) x + y,
     how = "replace", y = chunksizes_cum[i - 1])
  }

  centers <- as.integer(unlist(lapply(cano, names)))

  clustered_canopies <- kmeans(dataset[centers,], ncluster)$cluster

  clusters <- integer(nrow(dataset))
  h <- 1
  for (i in 1:length(cano)) {
    if (length(cano[[i]]) == 1) {
      clusters[unlist(cano[[i]])] <- clustered_canopies[h]
      h <- h + 1
    } else {
      for (j in 1:length(cano[[i]])) {
        clusters[cano[[i]][[j]]] <- clustered_canopies[h]
        h <- h + 1
      }
    }

  }
  return(clusters)
}

# cross

cross_canopy <- function(dataset, ncluster, orig) {
  n <- nrow(dataset)

  datasample <- matrix(nrow = 2000, ncol = ncol(dataset))
  boundsample <- round(2000 / ncluster)
```

```
  boundssample <- round(seq(0, 2000,
   length.out = ncluster + 1))

  for (i in 1:ncluster) {
    which_sample <- sample(1:floor(n / ncluster),
    boundssample[i + 1] - boundssample[i])
    datasample[(boundssample[i] + 1):boundssample[i + 1],] <-
     dataset[which(orig == i),][which_sample,]
  }

  shuf_index <- sample(1:nrow(datasample), nrow(datasample))
  datasample <- datasample[shuf_index,]

  orig_sample <- integer()
  for (i in 1:ncluster) {
    orig_sample <- c(orig_sample, rep(i,
     round(nrow(datasample) / ncluster)))
  }

  orig_sample <- orig_sample[shuf_index]

  thresholdlist <- seq(0.3, 0.6, 0.05)

  randlist <- NULL

  for (i in 1:length(thresholdlist)) {
    canopy_result <- canopy_cluster(datasample,
     thresholdlist[i], thresholdlist[i])
    randlist[i] <- rand(canopy_result, orig_sample,
     nrow(datasample))
  }

  threshold <- thresholdlist[which.max(randlist)]
  return(threshold)
}
```

C.4.13. A General Model for Clustering Binary Data: Two Side Clustering

```r
g_mod_bin_data1 <- function(dataset, ncluster, nclusterf) {

  if (!is.matrix(dataset)) {
    dataset <- as.matrix(dataset)
  }

  # initialize A
  A <- matrix(0L, nrow = nrow(dataset), ncol = ncluster)
  clusterseq <- floor(seq(0, nrow(dataset),
   length.out = ncluster + 1))
  for (i in 1:ncluster) {
    A[(clusterseq[i] + 1):clusterseq[i + 1], i] <- 1L
  }

  # initialize B
  B <- matrix(0L, nrow = nclusterf, ncol = ncol(dataset))
  clusterseq <- floor(seq(0, ncol(dataset),
   length.out = nclusterf + 1))
  for (i in 1:nclusterf) {
    B[i, (clusterseq[i] + 1):clusterseq[i + 1]] <- 1L
  }

  # compute X using eq 5

  pk <- colSums2(A)
  qc <- rowSums2(B)

  # if there are zeros, they will be replaced
  # with a low number (cant devide by 0 in next step)
  if (any(pk == 0)) {
    pk[pk == 0] <- 0.00000001
  }

  if (any(qc == 0)) {
    qc[qc == 0] <- 0.00000001
  }

  X <- matrix(nrow = ncluster, ncol = nclusterf)
  for (i in seq_len(ncluster)) {
    for (j in seq_len(nclusterf)) {
      X[i, j] <- (1 / (pk[i] * qc[j])) *
       sum(dataset[which(A[, i] == 1), which(B[j,] == 1)])
```

```
  }
}

What <- A %*% X %*% B
Oaxb1 <- norm(dataset - What, type = "F")
Oaxb0 <- Inf

while (Oaxb1 < Oaxb0) {

  Oaxb0 <- Oaxb1

  # update A using eq 6
  Aik <- matrix(nrow = nrow(dataset), ncol = ncluster)
  for (i in 1:ncluster) {
    for (j in 1:nclusterf) {
      Aik[, i] <- rowSums2((dataset[, which(B[j, ] == 1),
       drop = FALSE] - X[i, j])^2)
    }
  }

  whichcluster <- max.col(-Aik, ties = "first")
  A <- matrix(0L, nrow = nrow(dataset), ncol = ncluster)
  for (i in 1:ncluster) {
    A[which(whichcluster == i), i] <- 1L
  }

  # update B using eq 6
  Bjc <- matrix(nrow = nclusterf, ncol = ncol(dataset))
  for (i in 1:ncluster) {
    for (j in 1:nclusterf) {
      Bjc[j,] <- colSums2((dataset[which(A[, i] == 1), ,
       drop = FALSE] - X[i, j])^2)
    }
  }

  whichcluster <- max.col(t(-Bjc), ties = "first")
  B <- matrix(0L, nrow = nclusterf, ncol = ncol(dataset))
  for (j in 1:nclusterf) {
    B[j, which(whichcluster == j)] <- 1L
  }

  # compute X using eq 5
```

```
    pk <- colSums2(A)
    qc <- rowSums2(B)

    # if there are zeros, they will be replaced with
    # a low number (cant devide by 0 in next step)
    if (any(pk == 0)) {
      pk[pk == 0] <- 0.00000001
    }

    if (any(qc == 0)) {
      qc[qc == 0] <- 0.00000001
    }

    X <- matrix(nrow = ncluster, ncol = nclusterf)
    for (i in seq_len(ncluster)) {
      for (j in seq_len(nclusterf)) {
        X[i, j] <- (1 / (pk[i] * qc[j])) *
         sum(dataset[which(A[, i] == 1), which(B[j,] == 1)])
      }
    }

    What <- A %*% X %*% B
    Oaxb1 <- norm(dataset - What, type = "F")

  }

  clusters <- max.col(A, ties = "first")
  clusters
}

g_mod_bin_data1_par <- function(dataset, ncluster,
 nclusterf, ncores) {
  if (!is.matrix(dataset)) {
    dataset <- as.matrix(dataset)
  }

  results <- foreach(dat = isplitRows(dataset,
   chunks = ncores)) %dopar% {
    g_mod_bin_data1(dat, ncluster, nclusterf)
  }

  # make clusterids unique
```

```
  maxclusid <- max(unique(results[[1]]))
  for (i in 2:length(results)) {
    results[[i]] <- results[[i]] + maxclusid
    maxclusid <- max(unique(results[[i]]))
  }

  results <- unlist(results)

  medoids <- matrix(nrow = max(results), ncol = ncol(dataset))
  for (i in unique(results)) {
    clustersample <- which(results == i)
    clustersample <- dataset[clustersample,, drop = FALSE][
      sample(1:length(clustersample), ifelse(length(clustersample) <
       1000, length(clustersample), 1000)),, drop = FALSE]
    cs_dist <- pdist(clustersample, metric = "hamming")
    medoids[i,] <- clustersample[which.min(rowSums2(cs_dist)),]
  }

  medoids_diss <- pdist(medoids, metric = "hamming")
  result_medoids <- cluster::pam(medoids_diss, ncluster,
   cluster.only = TRUE)

  for (i in seq_len(ncluster)) {
    results[which(results %in% which(result_medoids == i))] <- i
  }
  results
}
```

C.4.14. A General Model for Clustering Binary Data: Block Diagonal Variant

```
g_mod_bin_data2 <- function(dataset, ncluster) {

  if (!is.matrix(dataset)) {
    dataset <- as.matrix(dataset)
  }

  A <- sparseMatrix(1:nrow(dataset),
   sample(1:ncluster, nrow(dataset), replace = TRUE))
```

```
nk <- colSums(A)

Y <- matrix(nrow = ncluster, ncol = ncol(dataset))
for (i in seq_len(ncluster)) {
  dd <- matrix(A[, i], nrow = nrow(dataset),
   ncol = ncol(dataset))
  Y[i,] <- (1/nk[i]) * colSums(dd * dataset)
}


B <- matrix(nrow = ncluster, ncol = ncol(dataset))
B[Y > 1 / ncluster] <- 1L
B[Y <= 1 / ncluster] <- 0L

# objective criterion

Oab1 <- sum(sqrt((dataset - A %*% B)^2))

Oab0 <- Inf

while (Oab1 < Oab0) {

  Oab0 <- Oab1


  Abkj <- matrix(nrow = nrow(dataset), ncol = ncluster)
  for (j in 1:ncluster) {
    BJ <- matrix(rep(B[j,], each = nrow(dataset)),
     nrow = nrow(dataset))
    Abkj[, j] <- rowSums((dataset - BJ)^2)
  }

  whichcluster <- apply(Abkj, 1, which.min)
  A <- matrix(0L, nrow = nrow(dataset), ncol = ncluster)
  for (i in 1:ncluster) {
    A[which(whichcluster == i), i] <- 1L
  }

  # compute B
  nk <- colSums(A)
  if (any(nk == 0)) {
    nk[which(nk == 0)] <- 1L
```

```
    }

    Y <- matrix(nrow = ncluster, ncol = ncol(dataset))
    for (i in seq_len(ncluster)) {
      dd <- matrix(A[, i], nrow = length(A[, i]),
       ncol = ncol(dataset))
      Y[i,] <- (1/nk[i]) * colSums(dd * dataset)
    }

    B <- matrix(nrow = ncluster, ncol = ncol(dataset))
    B[Y > 1 / ncluster] <- 1L
    B[Y <= 1 / ncluster] <- 0L

    # objective criterion. not too sure. rolling with it for now

    Oab1 <- sum(sqrt((dataset - A %*% B)^2))

  }

  clusters <- apply(A, 1, which.max)
  clusters
}


g_mod_bin_data2_par <- function(dataset, ncluster, ncores) {
  if (!is.matrix(dataset)) {
    dataset <- as.matrix(dataset)
  }

  results <- foreach(dat = isplitRows(dataset,
   chunks = ncores)) %dopar% {
    g_mod_bin_data2(dat, ncluster)
  }

  # make clusterids unique
  maxclusid <- max(unique(results[[1]]))
  for (i in 2:length(results)) {
    results[[i]] <- results[[i]] + maxclusid
    maxclusid <- max(unique(results[[i]]))
  }

  results <- unlist(results)
```

```
  medoids <- matrix(nrow = max(results), ncol = ncol(dataset))
  for (i in unique(results)) {
    clustersample <- which(results == i)
    clustersample <- dataset[clustersample,, drop = FALSE][
      sample(1:length(clustersample), ifelse(length(clustersample) <
       1000, length(clustersample), 1000)),, drop = FALSE]
    cs_dist <- pdist(clustersample, metric = "hamming")
    medoids[i,] <- clustersample[which.min(rowSums2(cs_dist)),]
  }

  medoids_diss <- pdist(medoids, metric = "hamming")
  result_medoids <- cluster::pam(medoids_diss, ncluster,
   cluster.only = TRUE)

  for (i in seq_len(ncluster)) {
    results[which(results %in% which(result_medoids == i))] <- i
  }
  results
}
```

C.5. Simulationsskript

```
# Run all functions that are needed for the simulation
# i.e. algorithms, evaluationfunctions, datasimulation...
source("simulation_setup.R")

ncores <- 10L
blas_set_num_threads(1)
registerDoMC(ncores)

anfang_alles <- Sys.time()
resultlist_list <- list()
tablelist_list <- list()
timings_list <- list()

lowp <- c(0.001, 0.04)
midp <- c(0.1, 0.2)
highp <- c(0.45, 0.49)
```

```
lowp_low <- c(0.005, 0.049)
midp_low <- c(0.05, 0.099)
highp_low <- c(0.1, 0.15)

lowp_high <- c(0.1, 0.2)
midp_high <- c(0.3, 0.5)
highp_high <- c(0.7, 0.8)

ms_list <- list()
ms_list[["low"]] <- data.table(praev = c("low", "mid", "high"),
 min = c(0.005, 0.05, 0.1), max = c(0.049, 0.099, 0.15))
ms_list[["mid"]] <- data.table(praev = c("low", "mid", "high"),
 min = c(0.001, 0.1, 0.45), max = c(0.04, 0.2, 0.49))

basecor_list <- data.table(corstruct = c("low", "mid"),
 min = c(0.05, 0.3), max = c(0.1, 0.4))

seed <- 0

for (n in c(1e6, 1e7, 5e7)) {
  for (n_cols in ifelse(n %in% c(1e6, 1e7), c(100, 1000), 100)) {
    ncols <- n_cols
    for (praevalenz in c("low", "mid")) {
      for (n_cluster in c(2, 4, 6)) {
        for (setup in c("einfach", "4block", "special")) {
          if (setup == "4block") {
            for (corstructure in c("low", "mid")) {

              print(paste(corstructure, setup, n_cluster,
               praevalenz, n, n_cols, sep = "_"))



              if (setup == "4block") {



                seed <- seed + 1
                set.seed(seed)
                print(seed)

                if (seed > 0) {
```

```
anfang_data <- Sys.time()
registerDoMC(ncores)

basecor <- runif(4, basecor_list[
  corstruct == corstructure, min],
  basecor_list[corstruct == corstructure, max])

n_vars <- ncols / 4

cormat <- simcor(k = 4, size = rep(n_vars, 4),
 rho = basecor, delta = 0.07, epsilon = 0.1,
  eidim = 3)

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' > ',
  'ram/data' ,
  corstructure, "_", setup, "_",
  n_cluster, "_", praevalenz, "_",
  n, "_", n_cols, '.txt'), wait = FALSE)

if (n_cluster == 2) {

  ms1 <-
   c(runif(n_vars, ms_list[[praevalenz]][
    praev == "low", min], ms_list[[
      praevalenz]][praev == "low", max]),
           runif(n_vars, ms_list[[
            praevalenz]][praev == "high", min],
             ms_list[[praevalenz]][praev == "high",
              max]),
           runif(n_vars, ms_list[[
            praevalenz]][praev == "low", min],
             ms_list[[praevalenz]][praev == "low", max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
            ms_list[[praevalenz]][
              praev == "high", max]))

  binarydata1 <- corrbinary(n = n / 2,
   means = ms1, sigma = cormat, cores = ncores)
```

```
  ms2 <- c(runif(n_vars, ms_list[[praevalenz]][
    praev == "high", min], ms_list[[praevalenz]][
      praev == "high", max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "low", min],
            ms_list[[praevalenz]][praev == "low",
             max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
            ms_list[[praevalenz]][praev == "high",
            max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "low", min],
            ms_list[[praevalenz]][praev == "low",
            max]))

  binarydata2 <- corrbinary(n = n / 2, means = ms2,
   sigma = cormat, cores = ncores)


  dataset <- rbindlist(list(binarydata1,
   binarydata2))
  rm(binarydata1, binarydata2)
}

if (n_cluster == 4) {

  ms1 <- c(runif(n_vars, ms_list[[praevalenz]][
    praev == "low", min], ms_list[[praevalenz]][
      praev == "low", max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
            ms_list[[praevalenz]][praev == "high",
             max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "low", min],
            ms_list[[praevalenz]][praev == "low",
             max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
             ms_list[[praevalenz]][praev == "high",
```

```
            max]))

binarydata1 <- corrbinary(n = n / 4, means = ms1,
 sigma = cormat, cores = ncores)


ms2 <- c(runif(n_vars, ms_list[[praevalenz]][
  praev == "high", min], ms_list[[praevalenz]][
    praev == "high", max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
           max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
           ms_list[[praevalenz]][praev == "high",
            max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
           ms_list[[praevalenz]][praev == "low",
            max]))

binarydata2 <- corrbinary(n = n / 4, means = ms2,
 sigma = cormat, cores = ncores)

ms3 <- c(runif(n_vars, ms_list[[praevalenz]][
  praev == "high", min],
  ms_list[[praevalenz]][praev == "high",
  max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
          ms_list[[praevalenz]][praev == "high",
           max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
           max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
          max]))
```

```
  binarydata3 <- corrbinary(n = n / 4, means = ms3,
   sigma = cormat, cores = ncores)


  ms4 <- c(runif(n_vars, ms_list[[praevalenz]][p
  raev == "low", min],
  ms_list[[praevalenz]][praev == "low", max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "low", min],
            ms_list[[praevalenz]][praev == "low",
            max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
            ms_list[[praevalenz]][praev == "high",
             max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
            ms_list[[praevalenz]][praev == "high",
             max]))

  binarydata4 <- corrbinary(n = n / 4, means = ms4,
   sigma = cormat, cores = ncores)


  dataset <- rbindlist(list(binarydata1,
   binarydata2, binarydata3, binarydata4))
  rm(binarydata1, binarydata2, binarydata3,
   binarydata4)
}


if (n_cluster == 6) {

  ms1 <- c(runif(n_vars, ms_list[[praevalenz]][
    praev == "low", min],
    ms_list[[praevalenz]][praev == "low",
    max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "high", min],
             ms_list[[praevalenz]][praev == "high",
              max]),
           runif(n_vars, ms_list[[praevalenz]][
```

```
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
          max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
          ms_list[[praevalenz]][praev == "high",
          max]))

binarydata1 <- corrbinary(n = floor(n / 6) + 1,
 means = ms1, sigma = cormat, cores = ncores)


ms2 <- c(runif(n_vars, ms_list[[praevalenz]][
  praev == "high", min],
  ms_list[[praevalenz]][praev == "high",
  max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
          max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
           ms_list[[praevalenz]][praev == "high",
            max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
          max]))

binarydata2 <- corrbinary(n = floor(n / 6) + 1,
 means = ms2, sigma = cormat, cores = ncores)

ms3 <- c(runif(n_vars, ms_list[[praevalenz]][
  praev == "high", min],
   ms_list[[praevalenz]][praev == "high",
   max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
          ms_list[[praevalenz]][praev == "high",
          max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
```

```
          ms_list[[praevalenz]][praev == "low",
          max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
          ms_list[[praevalenz]][praev == "low",
           max]))

binarydata3 <- corrbinary(n = floor(n / 6) + 1,
 means = ms3, sigma = cormat, cores = ncores)


ms4 <- c(runif(n_vars, ms_list[[praevalenz]][
  praev == "low", min],
  ms_list[[praevalenz]][praev == "low",
  max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "low", min],
           ms_list[[praevalenz]][praev == "low",
            max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
          ms_list[[praevalenz]][praev == "high",
          max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
          ms_list[[praevalenz]][praev == "high",
           max]))

binarydata4 <- corrbinary(n = floor(n / 6) + 1,
 means = ms4, sigma = cormat, cores = ncores)


ms5 <- c(runif(n_vars, ms_list[[praevalenz]][
  praev == "low", min],
  ms_list[[praevalenz]][praev == "low",
  max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "mid", min],
           ms_list[[praevalenz]][praev == "mid",
           max]),
         runif(n_vars, ms_list[[praevalenz]][
          praev == "high", min],
```

```
            ms_list[[praevalenz]][praev == "high",
            max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "mid", min],
            ms_list[[praevalenz]][praev == "mid",
            max]))

  binarydata5 <- corrbinary(n = floor(n / 6),
   means = ms5, sigma = cormat, cores = ncores)


  ms6 <- c(runif(n_vars, ms_list[[praevalenz]][
    praev == "high", min],
     ms_list[[praevalenz]][praev == "high",
     max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "low", min],
            ms_list[[praevalenz]][praev == "low",
             max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "mid", min],
             ms_list[[praevalenz]][praev == "mid",
             max]),
           runif(n_vars, ms_list[[praevalenz]][
            praev == "mid", min],
             ms_list[[praevalenz]][praev == "mid",
             max]))

  binarydata6 <- corrbinary(n = floor(n / 6),
   means = ms6, sigma = cormat, cores = ncores)


  dataset <- rbindlist(list(binarydata1, binarydata2,
   binarydata3, binarydata4, binarydata5, binarydata6))
  rm(binarydata1, binarydata2, binarydata3,
   binarydata4, binarydata5, binarydata6)
}

system('kill $(pgrep -f free)')

ende_data <- Sys.time()
zeit1 <- as.numeric(difftime(ende_data,
```

```
      anfang_data, units = "secs"))
    print(paste(zeit1, "data"))
  } # if seed

} # 4block


if (seed > 0) {
  print(paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"))

  # shuffle

  n2 <- n / n_cluster

  if (n_cluster == 2) {
    orig <- c(rep(1, n2), rep(2, n2))
  }

  if (n_cluster == 4) {
    orig <- c(rep(1, n2), rep(2, n2),
     rep(3, n2), rep(4, n2))
  }

  if (n_cluster == 6) {
    orig <- c(rep(1, n2+1), rep(2, n2+1),
     rep(3, n2+1), rep(4, n2+1), rep(5, n2),
      rep(6, n2))
  }

  shufindex <- sample(1:n, n)

  orig <- orig[shufindex]
  if (setup == "4block") {
    dataset <- as.matrix(dataset)
  }
  dataset <- dataset[shufindex,]
  rm(shufindex)

  # in case of only 0s in an observation,
  # add a random 1 (some algos crash otherwise)
  rsums <- rowSums2(dataset)
```

```
if (any(rsums == 0)) {
  which_to_replace <- which(rsums == 0)

  for (i in which_to_replace) {
    dataset[i, sample(1:ncol(dataset), 1)] <- 1L
  }
}

ncluster <- n_cluster

# clustering

resultlist <- data.table(algo = c("lsh", "minhash",
 "clara", "clarans", "bubble", "mona", "mbt",
  "clope", "large_item", "scale",
  "sorted_neighborhood", "proximus", "canopy",
   "general_model_algo_1", "general_model_algo_2"),
  recall = numeric(15), precision = numeric(15),
   f1 = numeric(15), rand = numeric(15),
  purity = numeric(15), entropy = numeric(15),
  nmi = numeric(15), zeit = numeric(15),
   ram = numeric(15))

tablelist <- list()

registerDoSEQ()
gc()

# lsh ----------------------------------------

registerDoMC(ncores)

lsh_cross <- cross_lsh(dataset, ncluster,
 orig, ncores = ncores)
gc()
ram_pre <-
as.numeric(system('free --mega |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 |
```

```
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
  'ram/lsh_' ,
              corstructure, "_", setup,
               "_", n_cluster, "_", praevalenz,
                "_", n, "_", n_cols, '.txt'),
                 wait = FALSE)

print(paste("lsh", Sys.time()))
anfang <- Sys.time()
lsh_result <- lsh_fun(dataset,
n_hash = lsh_cross$n_hash, b = lsh_cross$bands_number,
 ncores = ncores)
ende <- Sys.time()
saveRDS(lsh_result,
paste0("result_lsh_",
paste(corstructure, setup, n_cluster,
praevalenz, n, n_cols, sep = "_"), ".RDS"))
print(paste("lsh done", Sys.time()))

system("kill $(pgrep -f free)")

print(paste("lsh_eval", Sys.time()))
lsh_eval <- lsh_eval_fun(lsh_result,
 n_cluster = ncluster, n_data_sample = 2000, orig)
print(paste("lsh_eval done", Sys.time()))

ram_lsh <- fread(paste0("ram/lsh_", corstructure,
 "_", setup, "_", n_cluster, "_", praevalenz, "_",
  n, "_", n_cols, ".txt"))
ram_lsh <- max(ram_lsh$V1) - ram_pre

resultlist[algo == "lsh", c("recall", "precision",
 "f1", "rand") := lsh_eval]
resultlist[algo == "lsh", ram := ram_lsh]

zeit1 <- as.numeric(difftime(ende, anfang,
 units = "secs"))
print(paste(zeit1, "lsh"))


resultlist[algo == "lsh", zeit := zeit1]
```

```
rm(lsh_cross, lsh_result, lsh_eval, ram_lsh)


registerDoSEQ()
gc()

# minhash

registerDoMC(ncores)

minhash_cross <- cross_minhash(dataset,
ncluster, orig, ncores)

print(paste("minhash", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 |
grep --line-buffered Mem |
awk \'{print $3; system("")}\' > ',
 'ram/minhash_' ,
              corstructure, "_",
              setup, "_", n_cluster,
               "_", praevalenz, "_",
                n, "_", n_cols,
                 '.txt'),
                  wait = FALSE)

anfang <- Sys.time()
minhash_result <- minhash(dataset,
 minhash_cross, ncores)
ende <- Sys.time()
saveRDS(minhash_result, paste0("result_minhash_",
   paste(corstructure, setup, n_cluster,
    praevalenz, n, n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")
```

```
print(paste("minhash done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "minhash"))

origmh <- orig
nmh <- n
if (length(unique(minhash_result)) > 10000) {
  clusterseq <- round(seq(0, n,
   length.out = n_cluster + 1))
  clusterseq[length(clusterseq)] <- n

  sample_size_cl <- round(seq(0,
   2000, length.out = n_cluster + 1))

  minhash_sample <- NULL
  for (i in 1:n_cluster) {
    minhash_sample <- c(minhash_sample,
     sample((clusterseq[i] + 1):clusterseq[
      i + 1], sample_size_cl[i + 1] -
      sample_size_cl[i], replace = FALSE))
  }
  minhash_result <-
   minhash_result[minhash_sample]
  origmh <- orig[minhash_sample]
  nmh <- 2000
}

resultlist[algo == "minhash",
 c("recall", "precision", "f1",
  "rand", "purity", "entropy",
  "nmi", "zeit") :=
  data.table(rec_pre_f1(origmh,
   minhash_result, nmh),
              rand = rand(minhash_result,
               origmh, nmh),
              purity = purity(origmh,
               minhash_result, nmh, ncluster),
              entropy = Entropy(origmh,
               minhash_result, nmh),
              nmi = NMI(minhash_result,
```

```
              origmh),
             zeit = zeit1)]

ram_minhash <- fread(paste0("ram/minhash_",
 corstructure, "_", setup, "_", n_cluster,
  "_", praevalenz, "_", n, "_", n_cols,
  ".txt"))
ram_minhash <- max(ram_minhash$V1) - ram_pre

resultlist[algo == "minhash",
ram := ram_minhash]

tablelist <- append(tablelist,
list(minhash = bigtable(
  as.matrix(minhash_result), 1)))

rm(minhash_result, minhash_cross,
 origmh, nmh, ram_minhash)

registerDoSEQ()

gc()

# clara ---------------------------------

registerDoMC(ncores)

print("clara")

nsamples <- 5
samplesize <- 40 + 2 * n_cluster


print(paste("clara", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
```

```
 awk \'{print $3; system("")}\' > ',
 'ram/clara_' ,
              corstructure, "_",
               setup, "_", n_cluster,
                "_", praevalenz, "_",
                 n, "_", n_cols,
                  '.txt'),
                   wait = FALSE)

anfang <- Sys.time()
clara_result <- clara_fun(dataset, n,
 nsamples, samplesize, ncores, ncluster)
ende <- Sys.time()
saveRDS(clara_result, paste0("result_clara_",
   paste(corstructure, setup, n_cluster,
    praevalenz, n, n_cols, sep = "_"),
     ".RDS"))

system("kill $(pgrep -f free)")

print(paste("clara done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang,
 units = "secs"))
print(paste(zeit1, "clara"))

print(paste("clara eval", Sys.time()))
resultlist[algo == "clara", c("recall",
"precision", "f1", "rand", "purity",
"entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig,
  clara_result, n),
            rand = rand(clara_result,
             orig, n),
            purity = purity(orig,
             clara_result, n, ncluster),
            entropy = Entropy(orig,
             clara_result, n),
            nmi = NMI(clara_result,
             orig),
            zeit = zeit1)]
```

```
print(paste("clara eval done", Sys.time()))

ram_clara <- fread(paste0("ram/clara_",
 corstructure, "_", setup, "_", n_cluster,
  "_", praevalenz, "_", n, "_", n_cols,
   ".txt"))
ram_clara <- max(ram_clara$V1) - ram_pre

resultlist[algo == "clara", ram := ram_clara]


tablelist <- append(tablelist,
 list(clara = bigtable(as.matrix(clara_result),
 1)))

rm(clara_result, nsamples, samplesize)
registerDoSEQ()

gc()

# clarans ----------------------------------------

registerDoMC(ncores)

print("clarans")

numlocal <- 2
maxneighbor <- 1000

print(paste("clarans", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
  'ram/clarans_' ,
   corstructure, "_", setup, "_",
    n_cluster, "_", praevalenz, "_",
```

```
      n, "_", n_cols, '.txt'),
      wait = FALSE)

anfang <- Sys.time()
clarans_result <- clarans_fun(dataset,
 numlocal, ncluster, maxneighbor, ncores)
ende <- Sys.time()
saveRDS(clarans_result, paste0("result_clarans_",
  paste(corstructure, setup,
   n_cluster, praevalenz, n, n_cols,
   sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("clarans done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang,
 units = "secs"))
print(paste(zeit1, "clarans"))


print(paste("clarans eval", Sys.time()))
resultlist[algo == "clarans", c("recall",
 "precision", "f1", "rand", "purity",
  "entropy", "nmi", "zeit") :=
   data.table(rec_pre_f1(orig,
   clarans_result[[2]], n),
              rand = rand(clarans_result[[2]],
               orig, n),
              purity = purity(orig,
              clarans_result[[2]], n, ncluster),
              entropy = Entropy(orig,
              clarans_result[[2]], n),
              nmi = NMI(clarans_result[[2]],
              orig),
              zeit = zeit1)]

print(paste("clarans eval done", Sys.time()))

ram_clarans <- fread(paste0("ram/clarans_",
corstructure, "_", setup, "_", n_cluster, "_",
praevalenz, "_", n, "_", n_cols, ".txt"))
```

```
ram_clarans <- max(ram_clarans$V1) - ram_pre

resultlist[algo == "clarans", ram := ram_clarans]

tablelist <- append(tablelist,
list(clarans = bigtable(as.matrix(clarans_result[[2]]),
 1)))

rm(clarans_result, numlocal, maxneighbor)
registerDoSEQ()

gc()

# bubble ----------------------------

print("bubble")

registerDoMC(ncores)

threshold_L <- cross_bubble(dataset,
ncluster, orig)
threshold <- threshold_L$threshold
L <- threshold_L$L


print(paste("bubble", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
  'ram/bubble_' ,
  corstructure, "_", setup, "_",
   n_cluster, "_", praevalenz, "_", n,
    "_", n_cols, '.txt'),
     wait = FALSE)

anfang <- Sys.time()
```

```
bubble_result <- bubble(dataset,
 ncores, threshold, L)
ende <- Sys.time()

system("kill $(pgrep -f free)")

print(paste("bubble done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang,
 units = "secs"))
print(paste(zeit1, "bubble"))

result_pam <- cluster::pam(
  rdist(bubble_result$clustroids,
 metric = "hamming"), k = ncluster)


for (i in 1:ncluster) {
  set(bubble_result$clusterlabels,
  which(bubble_result$clusterlabels$label %in%
  rownames(bubble_result$clustroids[
    which(result_pam$clustering == i),])),
     j = 1L, value = i)
}

saveRDS(bubble_result$clusterlabels$label,
paste0("result_bubble_",
        paste(corstructure, setup,
         n_cluster, praevalenz, n,
          n_cols, sep = "_"), ".RDS"))

print(paste("bubble eval", Sys.time()))
resultlist[algo == "bubble", c("recall",
 "precision", "f1", "rand", "purity",
 "entropy", "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
   bubble_result$clusterlabels$label, n),
              rand = rand(
               bubble_result$clusterlabels$label,
               orig, n),
              purity = purity(
               orig, bubble_result$clusterlabels$label,
```

```
                n, ncluster),
              entropy = Entropy(
               orig, bubble_result$clusterlabels$label,
                n),
              nmi = NMI(
               bubble_result$clusterlabels$label,
                orig),
              zeit = zeit1)]

print(paste("bubble eval done", Sys.time()))

ram_bubble <- fread(paste0("ram/bubble_", corstructure,
 "_", setup, "_", n_cluster, "_", praevalenz, "_", n,
  "_", n_cols, ".txt"))
ram_bubble <- max(ram_bubble$V1) - ram_pre

resultlist[algo == "bubble", ram := ram_bubble]

tablelist <- append(tablelist,
list(bubble = bigtable(as.matrix(
  bubble_result$clusterlabels$label),
 1)))
rm(bubble_result, result_pam, threshold,
 L, ram_bubble)
registerDoSEQ()

gc()

# mona ------------------------------

registerDoMC(ncores)

print("mona")

nvars <- ncol(dataset)
n_splits <- ncluster - 1
datasetdt <- as.data.table(dataset)

print(paste("mona", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem |
```

```
 awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 |
grep --line-buffered Mem |
awk \'{print $3; system("")}\' > ',
 'ram/mona_' ,
              corstructure, "_", setup,
               "_", n_cluster, "_",
               praevalenz, "_", n, "_",
               n_cols, '.txt'),
               wait = FALSE)

anfang <- Sys.time()
mona_result <- mona(datasetdt,
 n_splits, nvars, ncores, ncols)
ende <- Sys.time()

saveRDS(mona_result, paste0("result_mona_",
     paste(corstructure, setup, n_cluster,
     praevalenz, n, n_cols, sep = "_"),
      ".RDS"))

system("kill $(pgrep -f free)")

print(paste("mona done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang,
 units = "secs"))
print(paste(zeit1, "mona"))

print(paste("mona eval", Sys.time()))
resultlist[algo == "mona", c("recall",
 "precision", "f1", "rand", "purity",
  "entropy", "nmi", "zeit") :=
   data.table(rec_pre_f1(orig,
    mona_result, n),
              rand = rand(mona_result,
               orig, n),
              purity = purity(orig,
               mona_result, n, ncluster),
              entropy = Entropy(orig,
```

```
                mona_result, n),
               nmi = NMI(mona_result,
                orig),
               zeit = zeit1)]
print(paste("mona eval done", Sys.time()))

ram_mona <- fread(paste0("ram/mona_",
 corstructure, "_", setup, "_",
 n_cluster, "_", praevalenz, "_",
 n, "_", n_cols, ".txt"))
ram_mona <- max(ram_mona$V1) - ram_pre

resultlist[algo == "mona",
 ram := ram_mona]

tablelist <- append(tablelist,
 list(mona = bigtable(as.matrix(mona_result),
  1)))
rm(mona_result, datasetdt, nvars, n_splits)
registerDoSEQ()

gc()


# mbt ---------------------------------------------------
registerDoMC(ncores)

print("mbt")

datasetdt <- as.data.table(dataset)

threshold <- n / 100

print(paste("mbt", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
```

```
  awk \'{print $3; system("")}\' > ',
   'ram/mbt_' ,
              corstructure, "_",
               setup, "_",
                n_cluster, "_",
                 praevalenz, "_",
                  n, "_", n_cols,
                   '.txt'),
                    wait = FALSE)

anfang <- Sys.time()
mbt_result <- mbt_par(datasetdt,
 threshold, ncluster)
ende <- Sys.time()

saveRDS(mbt_result, paste0("result_mbt_",
  paste(corstructure, setup,
   n_cluster, praevalenz, n,
    n_cols, sep = "_"),
    ".RDS"))

system("kill $(pgrep -f free)")

print(paste("mbt done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "mbt"))

print(paste("mbt eval", Sys.time()))
resultlist[algo == "mbt",
c("recall", "precision", "f1",
 "rand", "purity", "entropy",
  "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
   mbt_result, n),
              rand = rand(mbt_result,
               orig, n),
              purity = purity(orig,
               mbt_result, n,
               ncluster),
              entropy = Entropy(orig,
```

```
                mbt_result, n),
               nmi = NMI(mbt_result,
                orig),
               zeit = zeit1)]
print(paste("mbt eval done",
 Sys.time()))

ram_mbt <- fread(paste0("ram/mbt_",
 corstructure, "_", setup, "_",
  n_cluster, "_", praevalenz,
   "_", n, "_", n_cols,
    ".txt"))
ram_mbt <- max(ram_mbt$V1) - ram_pre

resultlist[algo == "mbt", ram := ram_mbt]

tablelist <- append(tablelist,
 list(mbt = bigtable(as.matrix(mbt_result),
  1)))

rm(threshold, mbt_result, datasetdt)
registerDoSEQ()

gc()

# clope ------------------------------------------------

print("clope")
registerDoMC(ncores)

r <- 2

print(paste("clope", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
```

```
  'ram/clope_' ,
  corstructure, "_", setup, "_",
  n_cluster, "_", praevalenz, "_",
   n, "_", n_cols, '.txt'),
    wait = FALSE)

anfang <- Sys.time()
clope_result <- clope_par(dataset,
 r, ncores, maxiteration = 10,
  maxmoved = 50, clusterstop = 100)
ende <- Sys.time()

saveRDS(clope_result, paste0("result_clope_",
    paste(corstructure, setup,
     n_cluster, praevalenz, n,
      n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("clope done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "clope"))

print(paste("clope eval", Sys.time()))
resultlist[algo == "clope", c("recall",
 "precision", "f1", "rand", "purity",
  "entropy", "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
  clope_result, n),
             rand = rand(clope_result,
              orig, n),
             purity = purity(orig,
              clope_result, n,
              ncluster),
             entropy = Entropy(orig,
              clope_result, n),
             nmi = NMI(clope_result,
              orig),
             zeit = zeit1)]
print(paste("clope eval done", Sys.time()))
```

```
ram_clope <- fread(paste0("ram/clope_",
 corstructure, "_", setup, "_",
 n_cluster, "_", praevalenz, "_",
 n, "_", n_cols, ".txt"))
ram_clope <- max(ram_clope$V1) - ram_pre

resultlist[algo == "clope",
ram := ram_clope]

tablelist <- append(tablelist,
 list(clope = bigtable(as.matrix(clope_result),
 1)))

rm(r, clope_result)
registerDoSEQ()

gc()

# large_item ----------------------------------------

print("large_item")

registerDoMC(ncores)

print(paste("large_item", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
  'ram/large_item_' ,
  corstructure, "_", setup, "_",
   n_cluster, "_", praevalenz,
    "_", n, "_", n_cols, '.txt'),
     wait = FALSE)

anfang <- Sys.time()
```

```
large_item_result <- largeitem_par(dataset,
 w = 1, minsupport = 0.8,
 maxiteration = 10, maxmoved = 10,
  ncores = ncores)
ende <- Sys.time()

saveRDS(large_item_result,
paste0("result_large_item_",
   paste(corstructure, setup, n_cluster,
    praevalenz, n, n_cols, sep = "_"),
     ".RDS"))

system("kill $(pgrep -f free)")

print(paste("large_item done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
anfang, units = "secs"))
print(paste(zeit1, "large_item"))

print(paste("large_item eval", Sys.time()))
resultlist[algo == "large_item",
 c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi",
  "zeit") :=
  data.table(rec_pre_f1(orig,
  large_item_result, n),
             rand = rand(large_item_result,
              orig, n),
             purity = purity(orig,
              large_item_result,
               n, ncluster),
             entropy = Entropy(orig,
              large_item_result, n),
             nmi = NMI(large_item_result,
              orig),
             zeit = zeit1)]
print(paste("large_item eval",
 Sys.time()))

tablelist <- append(tablelist,
list(large_item = bigtable(
```

```
  as.matrix(large_item_result),
 1)))

ram_large_item <- fread(paste0("ram/large_item_",
 corstructure, "_", setup, "_",
  n_cluster, "_", praevalenz, "_",
   n, "_", n_cols, ".txt"))
ram_large_item <- max(ram_large_item$V1) - ram_pre

resultlist[algo == "large_item",
 ram := ram_large_item]

rm(large_item_result)
registerDoSEQ()

gc()

# scale ------------------------------

print("scale")

registerDoMC(ncores)

nsample <- 2000
maxiteration <- 10
maxmoved <- 100
chunk_n <- n / 100 * 2

print(paste("scale", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' > ',
   'ram/scale_' ,
  corstructure, "_",
  setup, "_", n_cluster, "_",
  praevalenz, "_", n, "_", n_cols,
```

```
   '.txt'), wait = FALSE)

anfang <- Sys.time()
scale_results <- scale_cluster(dataset,
 nsample, ncluster, ncores, seed,
  maxiteration, maxmoved, chunk_n)
ende <- Sys.time()

saveRDS(scale_results, paste0("result_scale_",
 paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"),
  ".RDS"))

system("kill $(pgrep -f free)")

print(paste("scale done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "scale"))

print(paste("scale eval", Sys.time()))
resultlist[algo == "scale", c("recall",
"precision", "f1", "rand", "purity",
 "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig,
 scale_results, n),
            rand = rand(scale_results,
             orig, n),
            purity = purity(orig,
             scale_results, n, ncluster),
            entropy = Entropy(orig,
             scale_results, n),
            nmi = NMI(scale_results,
             orig),
            zeit = zeit1)]
print(paste("scale eval done",
 Sys.time()))

ram_scale <- fread(paste0("ram/scale_",
 corstructure, "_", setup, "_",
 n_cluster, "_", praevalenz, "_", n,
```

```
 "_", n_cols, ".txt"))
ram_scale <- max(ram_scale$V1) - ram_pre

resultlist[algo == "scale",
 ram := ram_scale]

tablelist <- append(tablelist,
 list(scale = bigtable(
  as.matrix(scale_results),
  1)))

rm(scale_results, nsample,
 maxiteration, maxmoved,
 chunk_n, ram_scale)
registerDoSEQ()

gc()



# sorted_neighborhood -----------------------------------

print("sorted_neighborhood")

registerDoMC(ncores)
threshold <- 0.5

windowsize <- 100

print(paste("sorted_neighborhood", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' > ',
   'ram/sorted_neighborhood_' ,
 corstructure, "_",
 setup, "_", n_cluster, "_",
```

```
  praevalenz, "_", n, "_", n_cols,
  '.txt'), wait = FALSE)

anfang <- Sys.time()
sorted_neighborhood_result <-
 sorted_neighborhood(dataset,
 windowsize, threshold = threshold,
  runs = 3, ncores = ncores)
ende <- Sys.time()

saveRDS(sorted_neighborhood_result,
 paste0("result_sn_",
 paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"),
   ".RDS"))

system("kill $(pgrep -f free)")

print(paste("sorted_neighborhood done",
Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "sorted_neighborhood"))

print(paste("sorted_neighborhood eval",
 Sys.time()))
resultlist[algo == "sorted_neighborhood",
 c("recall", "precision", "f1", "rand",
  "purity", "entropy", "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
  sorted_neighborhood_result, n),
             rand = rand(sorted_neighborhood_result,
              orig, n),
             purity = purity(orig,
              sorted_neighborhood_result,
               n, ncluster),
             entropy = Entropy(orig,
              sorted_neighborhood_result,
               n),
             nmi = NMI(sorted_neighborhood_result,
              orig),
```

```
             zeit = zeit1)]
print(paste("sorted_neighborhood eval done",
 Sys.time()))

ram_sorted_neighborhood <-
 fread(paste0("ram/sorted_neighborhood_",
 corstructure, "_", setup, "_", n_cluster,
  "_", praevalenz, "_", n, "_", n_cols,
   ".txt"))
ram_sorted_neighborhood <-
 max(ram_sorted_neighborhood$V1) - ram_pre

resultlist[algo == "sorted_neighborhood",
 ram := ram_sorted_neighborhood]

tablelist <- append(tablelist,
list(sorted_neighborhood = bigtable(
  as.matrix(sorted_neighborhood_result),
  1)))

rm(threshold, windowsize,
 sorted_neighborhood_result,
 ram_sorted_neighborhood)
registerDoSEQ()

gc()

# proximus ---------------------------------------------

# if (seed > 81) {
print("proximus")

registerDoMC(ncores)

maxradius <- ncols

print(paste("proximus", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem |
 awk \'{print $3; system("")}\' ',
 intern = TRUE))
```

```
system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
  'ram/proximus_' ,
 corstructure, "_", setup, "_",
 n_cluster, "_", praevalenz, "_", n,
 "_", n_cols, '.txt'), wait = FALSE)

ncores <- 30
registerDoMC(ncores)


proximus_result <- prox(dataset,
 maxradius, ncores, ncluster)

ende <- Sys.time()

saveRDS(proximus_result$clusters,
paste0("result_proximus_",
       paste(corstructure, setup,
        n_cluster,
        praevalenz, n, n_cols,
        sep = "_"),
         ".RDS"))

ncores <- 10
registerDoMC(ncores)

system("kill $(pgrep -f free)")

print(paste("proximus done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "proximus"))

print(paste("proximus eval", Sys.time()))
resultlist[algo == "proximus",
 c("recall", "precision", "f1", "rand",
  "purity", "entropy", "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
```

```
   proximus_result$clusters, n),
              rand = rand(
               proximus_result$clusters,
                orig, n),
              purity = purity(
               orig, proximus_result$clusters,
                n, ncluster),
              entropy = Entropy(
               orig, proximus_result$clusters,
                n),
              nmi = NMI(
               proximus_result$clusters,
                orig),
              zeit = zeit1)]
print(paste("proximus eval done", Sys.time()))

ram_proximus <- fread(paste0(
  "ram/proximus_", corstructure, "_",
  setup, "_", n_cluster, "_", praevalenz,
   "_", n, "_", n_cols, ".txt"))
ram_proximus <- max(ram_proximus$V1) - ram_pre

resultlist[algo == "proximus",
 ram := ram_proximus]

tablelist <- append(tablelist,
list(proximus = bigtable(
  as.matrix(proximus_result$clusters),
  1)))

rm(maxradius, proximus_result,
 ram_proximus)
registerDoSEQ()

gc()
#}

# canopy ------------------------------------------

if (seed > 0) {
  print("canopy")
  registerDoMC(ncores)
```

```
threshold <- cross_canopy(dataset,
 ncluster, orig)

print(paste("canopy", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' > ',
  'ram/canopy_' ,
  corstructure, "_", setup, "_",
   n_cluster, "_", praevalenz, "_",
    n, "_", n_cols, '.txt'),
     wait = FALSE)

anfang <- Sys.time()
canopy_result <- canopy_cluster_par(dataset,
 threshold, threshold, ncluster, ncores)
ende <- Sys.time()

saveRDS(canopy_result, paste0("result_canopy_",
  paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"),
   ".RDS"))

system("kill $(pgrep -f free)")

print(paste("canopy done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
anfang, units = "secs"))
print(paste(zeit1, "canopy"))

print(paste("canopy eval", Sys.time()))
resultlist[algo == "canopy", c("recall",
 "precision", "f1", "rand", "purity",
  "entropy", "nmi", "zeit") :=
```

```
    data.table(rec_pre_f1(orig,
    canopy_result, n),
                rand = rand(canopy_result,
                 orig, n),
                purity = purity(orig,
                 canopy_result, n,
                  ncluster),
                entropy = Entropy(orig,
                 canopy_result, n),
                nmi = NMI(canopy_result,
                 orig),
                zeit = zeit1)]
  print(paste("canopy eval done", Sys.time()))

  ram_canopy <- fread(paste0("ram/canopy_",
   corstructure, "_", setup, "_",
    n_cluster, "_", praevalenz, "_",
    n, "_", n_cols, ".txt"))
  ram_canopy <- max(ram_canopy$V1) - ram_pre

  resultlist[algo == "canopy",
  ram := ram_canopy]

  tablelist <- append(tablelist,
   list(canopy = bigtable(
    as.matrix(canopy_result),
   1)))

  rm(threshold, canopy_result,
  ram_canopy)
  registerDoSEQ()

  gc()
}

# general model ----------------------------------

print("general_model_algo_2")

if (setup != "einfach") {
  nclusterf <- 4
} else {
```

```
  nclusterf <- 2
}

registerDoMC(ncores)

print(paste("general_model_algo_2", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' > ',
   'ram/general_model_algo_2_' ,
   corstructure, "_", setup, "_",
    n_cluster, "_", praevalenz,
    "_", n, "_", n_cols, '.txt'),
     wait = FALSE)

anfang <- Sys.time()
g_model_result2 <- g_mod_bin_data2_par(
  dataset, ncluster, ncores)
ende <- Sys.time()

saveRDS(g_model_result2,
 paste0("result_general_model_algo_2_",
  paste(corstructure, setup, n_cluster,
   praevalenz, n, n_cols, sep = "_"),
   ".RDS"))

system("kill $(pgrep -f free)")

print(paste("general_model_algo_2 done", Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "general model2"))

print(paste("general_model_algo_2 eval", Sys.time()))
resultlist[algo == "general_model_algo_2",
```

```
 c("recall", "precision",
  "f1", "rand", "purity",
  "entropy", "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
   g_model_result2, n),
              rand = rand(g_model_result2,
               orig, n),
              purity = purity(orig,
               g_model_result2, n,
                ncluster),
              entropy = Entropy(orig,
               g_model_result2, n),
              nmi = NMI(g_model_result2,
              orig),
              zeit = zeit1)]
print(paste("general_model_algo_2 eval done",
 Sys.time()))

ram_general_model_algo_2 <-
 fread(paste0("ram/general_model_algo_2_",
  corstructure, "_", setup, "_", n_cluster,
   "_", praevalenz, "_", n, "_", n_cols, ".txt"))
ram_general_model_algo_2 <-
 max(ram_general_model_algo_2$V1) - ram_pre

resultlist[algo == "general_model_algo_2",
 ram := ram_general_model_algo_2]

tablelist <- append(tablelist,
list(general_model_algo_2 = bigtable(
  as.matrix(g_model_result2),
   1)))

rm(g_model_result2, ram_general_model_algo_2)

print("general model1")

print(paste("general_model_algo_1", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ',
```

```
   intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ',
  'ram/general_model_algo_1_' ,
 corstructure, "_", setup, "_",
 n_cluster, "_", praevalenz, "_",
  n, "_", n_cols, '.txt'),
   wait = FALSE)

anfang <- Sys.time()
g_model_result1 <- g_mod_bin_data1_par(
  dataset, ncluster, nclusterf, ncores)
ende <- Sys.time()

saveRDS(g_model_result1,
 paste0("result_general_model_algo_1_",
 paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"),
  ".RDS"))

system("kill $(pgrep -f free)")

print(paste("general_model_algo_1 done",
Sys.time()))

zeit1 <- as.numeric(difftime(ende,
 anfang, units = "secs"))
print(paste(zeit1, "general model1"))

print(paste("general_model_algo_1 eval",
 Sys.time()))
resultlist[algo == "general_model_algo_1",
 c("recall", "precision", "f1", "rand",
  "purity", "entropy", "nmi", "zeit") :=
  data.table(rec_pre_f1(orig,
  g_model_result1, n),
             rand = rand(g_model_result1,
              orig, n),
             purity = purity(orig,
              g_model_result1, n,
```

```
                ncluster),
              entropy = Entropy(orig,
               g_model_result1, n),
              nmi = NMI(g_model_result1,
               orig),
              zeit = zeit1)]
print(paste("general_model_algo_1 eval done",
 Sys.time()))

ram_general_model_algo_1 <- fread(
  paste0("ram/general_model_algo_1_",
   corstructure, "_", setup, "_",
    n_cluster, "_", praevalenz, "_",
     n, "_", n_cols, ".txt"))
ram_general_model_algo_1 <- max(
  ram_general_model_algo_1$V1) - ram_pre

resultlist[algo == "general_model_algo_1",
 ram := ram_general_model_algo_1]

tablelist <- append(tablelist,
 list(general_model_algo_1 = bigtable(
  as.matrix(g_model_result1), 1)))

rm(g_model_result1,
 ram_general_model_algo_1)

registerDoSEQ()

gc()

saveRDS(resultlist, paste0("resultlist_",
 paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"),
   ".RDS"))
saveRDS(tablelist, paste0("tablelist_",
 paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"),
   ".RDS"))

resultlist_list[[paste(corstructure,
 setup, n_cluster, praevalenz, n,
```

```
                  n_cols, sep = "_")]] <- resultlist
                tablelist_list[[paste(corstructure,
                 setup, n_cluster, praevalenz, n,
                 n_cols, sep = "_")]] <- tablelist

              }


            } # corstructure

} else { # special einfach

  corstructure <- "einfach_special"
  print(paste(corstructure, setup, n_cluster,
  praevalenz, n, n_cols, sep = "_"))

  if (setup == "einfach") {

    seed <- seed + 1
    set.seed(seed)
    print(seed)

    if (seed > 0) {
      anfang_data <- Sys.time()

      system(paste0('free --mega -s 1 |
       grep --line-buffered Mem |
       awk \'{print $3; system("")}\' > ', 'ram/data' ,
                    corstructure, "_", setup,
                    "_", n_cluster, "_", praevalenz,
                     "_", n, "_", n_cols, '.txt'),
                      wait = FALSE)



      dataset2 <- matrix(runif(n * ncols),
       nrow = n, ncol = ncols)
      dataset <- matrix(integer(n * ncols),
       nrow = n, ncol = ncols)

      chunks <- round(seq.int(0, n,
      length.out = n_cluster + 1))
```

```
ncols_chunks <- round(seq.int(0,
ncols, length.out = 5))

if (praevalenz == "low") {
  mslow <- runif(1, lowp_low[1], lowp_low[2])
  msmid <- runif(1, midp_low[1], midp_low[2])
  mshigh <- runif(1, highp_low[1], highp_low[2])
} else {
  mslow <- runif(1, lowp_high[1], lowp_high[2])
  msmid <- runif(1, midp_high[1], midp_high[2])
  mshigh <- runif(1, highp_high[1], highp_high[2])
}

if (n_cluster == 2) {

  for (i in 1:(length(chunks) - 1)) {
    dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
     1:(ncols / 2)][
      which(dataset2[seq.int(chunks[i] + 1,
       chunks[i + 1]), 1:(ncols / 2)] < ifelse(
        i %% 2 == 0, mslow, mshigh))] <- 1L

    dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols / 2 + 1):ncols][
      which(dataset2[seq.int(chunks[i] + 1,
       chunks[i + 1]), (ncols / 2 + 1):ncols] < ifelse(
        i %% 2 == 0, mshigh, mslow))] <- 1L
  }


}

if (n_cluster == 4) {
  for (i in 1:(length(chunks) - 1)) {
    if (i == 1) {
      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       1:ncols_chunks[2]][
        which(dataset2[seq.int(chunks[i] + 1,
        chunks[i + 1]), 1:ncols_chunks[2]] < mslow)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
```

```
    (ncols_chunks[2] + 1):ncols_chunks[3]][
     which(dataset2[seq.int(chunks[i] + 1,
      chunks[i + 1]),
      (ncols_chunks[2] + 1):ncols_chunks[3]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1,
    chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[4] + 1):ncols_chunks[5]] < mshigh)] <- 1L
}

if (i == 2) {
  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   1:ncols_chunks[2]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]), 1:ncols_chunks[2]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[2] + 1):ncols_chunks[3]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[2] + 1):ncols_chunks[3]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[4] + 1):ncols_chunks[5]] < mslow)] <- 1L
}

if (i == 3) {
  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   1:ncols_chunks[2]][
```

```
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     1:ncols_chunks[2]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[2] + 1):ncols_chunks[3]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[2] + 1):ncols_chunks[3]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
    (ncols_chunks[3] + 1):ncols_chunks[4]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[4] + 1):ncols_chunks[5]] < mslow)] <- 1L
}

if (i == 4) {
  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   1:ncols_chunks[2]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
    1:ncols_chunks[2]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
  (ncols_chunks[2] + 1):ncols_chunks[3]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[2] + 1):ncols_chunks[3]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
  (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
      (ncols_chunks[4] + 1):ncols_chunks[5]] < mshigh)] <- 1L
```

```
    }

  }

}


if (n_cluster == 6) {
  for (i in 1:(length(chunks) - 1)) {
    if (i == 1) {
      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       1:ncols_chunks[2]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]), 1:ncols_chunks[2]] < mslow)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[2] + 1):ncols_chunks[3]][
        which(dataset2[seq.int(chunks[i] + 1,
        chunks[i + 1]),
         (ncols_chunks[2] + 1):ncols_chunks[3]] < mshigh)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[3] + 1):ncols_chunks[4]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]),
          (ncols_chunks[3] + 1):ncols_chunks[4]] < mslow)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[4] + 1):ncols_chunks[5]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]),
         (ncols_chunks[4] + 1):ncols_chunks[5]] < mshigh)] <- 1L
    }

    if (i == 2) {
      dataset[seq.int(chunks[i] + 1,
       chunks[i + 1]), 1:ncols_chunks[2]][
        which(dataset2[seq.int(chunks[i] + 1,
        chunks[i + 1]), 1:ncols_chunks[2]] < mshigh)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[2] + 1):ncols_chunks[3]][
```

```
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[2] + 1):ncols_chunks[3]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
     (ncols_chunks[4] + 1):ncols_chunks[5]] < mslow)] <- 1L
}

if (i == 3) {
  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   1:ncols_chunks[2]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]), 1:ncols_chunks[2]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[2] + 1):ncols_chunks[3]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
      (ncols_chunks[2] + 1):ncols_chunks[3]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
  (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1,
    chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1,
    chunks[i + 1]),
     (ncols_chunks[4] + 1):ncols_chunks[5]] < mslow)] <- 1L
}

if (i == 4) {
```

```
  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   1:ncols_chunks[2]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
      1:ncols_chunks[2]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[2] + 1):ncols_chunks[3]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
      (ncols_chunks[2] + 1):ncols_chunks[3]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mshigh)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[4] + 1):ncols_chunks[5]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[4] + 1):ncols_chunks[5]] < mshigh)] <- 1L
}

if (i == 5) {
  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
  1:ncols_chunks[2]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]), 1:ncols_chunks[2]] < mslow)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[2] + 1):ncols_chunks[3]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
      (ncols_chunks[2] + 1):ncols_chunks[3]] < msmid)] <- 1L

  dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
   (ncols_chunks[3] + 1):ncols_chunks[4]][
    which(dataset2[seq.int(chunks[i] + 1,
     chunks[i + 1]),
     (ncols_chunks[3] + 1):ncols_chunks[4]] < mshigh)] <- 1L
```

```
      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[4] + 1):ncols_chunks[5]][
        which(dataset2[seq.int(chunks[i] + 1, chunks[i + 1]),
         (ncols_chunks[4] + 1):ncols_chunks[5]] < msmid)] <- 1L
    }

    if (i == 6) {
      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       1:ncols_chunks[2]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]), 1:ncols_chunks[2]] < mshigh)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[2] + 1):ncols_chunks[3]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]),
         (ncols_chunks[2] + 1):ncols_chunks[3]] < mslow)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
      (ncols_chunks[3] + 1):ncols_chunks[4]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]),
          (ncols_chunks[3] + 1):ncols_chunks[4]] < msmid)] <- 1L

      dataset[seq.int(chunks[i] + 1, chunks[i + 1]),
       (ncols_chunks[4] + 1):ncols_chunks[5]][
        which(dataset2[seq.int(chunks[i] + 1,
         chunks[i + 1]),
          (ncols_chunks[4] + 1):ncols_chunks[5]] < msmid)] <- 1L
    }

  }


}
rm(dataset2)

system('kill $(pgrep -f free)')

ende_data <- Sys.time()
zeit1 <- as.numeric(difftime(ende_data, anfang_data,
```

```
      units = "secs"))
    print(paste(zeit1, "data"))
  } # if seed



} # if setup == einfach

if (setup == "special") {

  seed <- seed + 1
  set.seed(seed)
  print(seed)

  if (seed > 0) {
    anfang_data <- Sys.time()

    system(paste0('free --mega -s 1 |
    grep --line-buffered Mem |
     awk \'{print $3; system("")}\' > ', 'ram/data' ,
                  corstructure, "_",
                   setup, "_", n_cluster,
                    "_", praevalenz, "_", n, "_",
                     n_cols, '.txt'), wait = FALSE)

    registerDoMC(ncores)

    print(paste(corstructure, setup, n_cluster, praevalenz,
     n, n_cols, sep = "_"))

    basecor <- c(0.1, 0.9)

    n_vars <- ncols / 4

    cormat <- simcor(k = 2, size = c((ncols / 100 * 80),
     (ncols / 100 * 20)), rho = basecor, delta = 0.07,
      epsilon = 0.99 - max(basecor), eidim = 3)

    if (n_cluster == 2) {

      ms1 <- c(
        runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
```

```
      ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
      ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
      ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
      ms_list[[praevalenz]][praev == "high", max]))

  binarydata1 <- corrbinary(n = n / 2, means = ms1,
   sigma = cormat, cores = ncores)


  ms2 <- c(
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
      ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
      ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
      ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
      ms_list[[praevalenz]][praev == "low", max]))

  binarydata2 <- corrbinary(n = n / 2, means = ms2,
   sigma = cormat, cores = ncores)


  dataset <- rbindlist(list(binarydata1, binarydata2))
  rm(binarydata1, binarydata2)
}

if (n_cluster == 4) {

  ms1 <- c(
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
      ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
      ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
      ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
      ms_list[[praevalenz]][praev == "high", max]))
```

```
binarydata1 <- corrbinary(n = n / 4, means = ms1,
 sigma = cormat, cores = ncores)


ms2 <- c(
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
   ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
   ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
   ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
   ms_list[[praevalenz]][praev == "low", max]))

binarydata2 <- corrbinary(n = n / 4, means = ms2,
 sigma = cormat, cores = ncores)

ms3 <- c(
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
   ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
   ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
   ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
   ms_list[[praevalenz]][praev == "low", max]))

binarydata3 <- corrbinary(n = n / 4, means = ms3,
 sigma = cormat, cores = ncores)


ms4 <- c(
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
   ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
   ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
   ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
   ms_list[[praevalenz]][praev == "high", max]))

binarydata4 <- corrbinary(n = n / 4, means = ms4,
```

```
   sigma = cormat, cores = ncores)


  dataset <- rbindlist(list(binarydata1, binarydata2,
   binarydata3, binarydata4))
  rm(binarydata1, binarydata2, binarydata3, binarydata4)
}


if (n_cluster == 6) {

  ms1 <- c(
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
     ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
     ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
     ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
     ms_list[[praevalenz]][praev == "high", max]))

  binarydata1 <- corrbinary(n = floor(n / 6) + 1, means = ms1,
   sigma = cormat, cores = ncores)


  ms2 <- c(
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
     ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
     ms_list[[praevalenz]][praev == "low", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
     ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
     ms_list[[praevalenz]][praev == "low", max]))

  binarydata2 <- corrbinary(n = floor(n / 6) + 1, means = ms2,
   sigma = cormat, cores = ncores)

  ms3 <- c(
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
     ms_list[[praevalenz]][praev == "high", max]),
    runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
```

```
    ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
    ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
    ms_list[[praevalenz]][praev == "low", max]))

binarydata3 <- corrbinary(n = floor(n / 6) + 1, means = ms3,
 sigma = cormat, cores = ncores)


ms4 <- c(
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
    ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
    ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
    ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
    ms_list[[praevalenz]][praev == "high", max]))

binarydata4 <- corrbinary(n = floor(n / 6) + 1, means = ms4,
 sigma = cormat, cores = ncores)


ms5 <- c(
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
    ms_list[[praevalenz]][praev == "low", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "mid", min],
    ms_list[[praevalenz]][praev == "mid", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
    ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "mid", min],
    ms_list[[praevalenz]][praev == "mid", max]))

binarydata5 <- corrbinary(n = floor(n / 6), means = ms5,
 sigma = cormat, cores = ncores)


ms6 <- c(
  runif(n_vars, ms_list[[praevalenz]][praev == "high", min],
    ms_list[[praevalenz]][praev == "high", max]),
  runif(n_vars, ms_list[[praevalenz]][praev == "low", min],
```

```
          ms_list[[praevalenz]][praev == "low", max]),
         runif(n_vars, ms_list[[praevalenz]][praev == "mid", min],
          ms_list[[praevalenz]][praev == "mid", max]),
         runif(n_vars, ms_list[[praevalenz]][praev == "mid", min],
          ms_list[[praevalenz]][praev == "mid", max]))

      binarydata6 <- corrbinary(n = floor(n / 6), means = ms6,
       sigma = cormat, cores = ncores)


      dataset <- rbindlist(list(binarydata1, binarydata2,
       binarydata3, binarydata4, binarydata5, binarydata6))
      rm(binarydata1, binarydata2, binarydata3,
       binarydata4, binarydata5, binarydata6)
    }
    system('kill $(pgrep -f free)')

    ende_data <- Sys.time()
    zeit1 <- as.numeric(difftime(ende_data, anfang_data,
     units = "secs"))
    print(paste(zeit1, "data"))
  }



} # special

if (seed > 0) {
  # shuffle

  n2 <- n / n_cluster

  if (n_cluster == 2) {
    orig <- c(rep(1, n2), rep(2, n2))
  }

  if (n_cluster == 4) {
    orig <- c(rep(1, n2), rep(2, n2), rep(3, n2), rep(4, n2))
  }

  if (n_cluster == 6) {
    orig <- c(rep(1, n2+1), rep(2, n2+1), rep(3, n2+1),
```

```
  rep(4, n2+1), rep(5, n2), rep(6, n2))
}

shufindex <- sample(1:n, n)

orig <- orig[shufindex]
dataset <- as.matrix(dataset)
dataset <- dataset[shufindex,]
rm(shufindex)

# in case of only 0s in an observation,
# add a random 1 (some algos crash otherwise)
rsums <- rowSums2(dataset)
if (any(rsums == 0)) {
  which_to_replace <- which(rsums == 0)

  for (i in which_to_replace) {
    dataset[i, sample(1:ncol(dataset), 1)] <- 1L
  }
}


ncluster <- n_cluster

print(paste(corstructure, setup, n_cluster, praevalenz,
 n, n_cols, sep = "_"))

# clustering

resultlist <- data.table(algo = c("lsh", "minhash",
 "clara", "clarans", "bubble", "mona", "mbt", "clope",
 "large_item", "scale", "sorted_neighborhood", "proximus",
 "canopy", "general_model_algo_1", "general_model_algo_2"),
  recall = numeric(15), precision = numeric(15),
   f1 = numeric(15), rand = numeric(15),
  purity = numeric(15), entropy = numeric(15),
   nmi = numeric(15), zeit = numeric(15), ram = numeric(15))

tablelist <- list()

registerDoSEQ()
gc()
```

```
# lsh ------------------------------------------------

registerDoMC(ncores)

lsh_cross <- cross_lsh(dataset, ncluster, orig,
 ncores = ncores)
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem |
  awk \'{print $3; system("")}\' ', intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/lsh_' ,
 corstructure, "_", setup, "_", n_cluster, "_",
  praevalenz, "_", n, "_", n_cols, '.txt'), wait = FALSE)

print(paste("lsh", Sys.time()))
anfang <- Sys.time()
lsh_result <- lsh_fun(dataset, n_hash = lsh_cross$n_hash,
 b = lsh_cross$bands_number, ncores = ncores)
ende <- Sys.time()

saveRDS(lsh_result, paste0("/result_lsh_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))
print(paste("lsh done", Sys.time()))

system("kill $(pgrep -f free)")

print(paste("lsh_eval", Sys.time()))
lsh_eval <- lsh_eval_fun(lsh_result, n_cluster = ncluster,
 n_data_sample = 2000, orig)
print(paste("lsh_eval done", Sys.time()))

ram_lsh <- fread(paste0("ram/lsh_", corstructure, "_", setup,
 "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols, ".txt"))
ram_lsh <- max(ram_lsh$V1) - ram_pre

resultlist[algo == "lsh", c("recall", "precision",
 "f1", "rand") := lsh_eval]
resultlist[algo == "lsh", ram := ram_lsh]
```

```
zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "lsh"))

resultlist[algo == "lsh", zeit := zeit1]

rm(lsh_cross, lsh_result, lsh_eval, ram_lsh)

registerDoSEQ()
gc()

# minhash

registerDoMC(ncores)

minhash_cross <- cross_minhash(dataset, ncluster, orig, ncores)

print(paste("minhash", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/minhash_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz,
 "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
minhash_result <- minhash(dataset, minhash_cross, ncores)
ende <- Sys.time()

saveRDS(minhash_result, paste0("/result_minhash_",
 paste(corstructure, setup, n_cluster, praevalenz, n,
 n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("minhash done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "minhash"))
```

```
origmh <- orig
nmh <- n
if (length(unique(minhash_result)) > 10000) {
  clusterseq <- round(seq(0, n, length.out = n_cluster + 1))
  clusterseq[length(clusterseq)] <- n

  sample_size_cl <- round(seq(0, 2000, length.out = n_cluster + 1))

  minhash_sample <- NULL
  for (i in 1:n_cluster) {
    minhash_sample <- c(minhash_sample,
    sample((clusterseq[i] + 1):clusterseq[i + 1],
     sample_size_cl[i + 1] - sample_size_cl[i], replace = FALSE))
  }
  minhash_result <- minhash_result[minhash_sample]
  origmh <- orig[minhash_sample]
  nmh <- 2000
}

resultlist[algo == "minhash", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(origmh, minhash_result, nmh),
            rand = rand(minhash_result, origmh, nmh),
            purity = purity(origmh, minhash_result,
             nmh, ncluster),
            entropy = Entropy(origmh, minhash_result, nmh),
            nmi = NMI(minhash_result, origmh),
            zeit = zeit1)]

ram_minhash <- fread(paste0("ram/minhash_", corstructure, "_", setup,
 "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols, ".txt"))
ram_minhash <- max(ram_minhash$V1) - ram_pre

resultlist[algo == "minhash", ram := ram_minhash]

tablelist <- append(tablelist,
 list(minhash = bigtable(as.matrix(minhash_result), 1)))

rm(minhash_result, minhash_cross, origmh, nmh, ram_minhash)
```

```
registerDoSEQ()

gc()

# clara ------------------------------------------------------------

registerDoMC(ncores)

print("clara")

nsamples <- 5
samplesize <- 40 + 2 * n_cluster


print(paste("clara", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/clara_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_",
 n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
clara_result <- clara_fun(dataset, n, nsamples,
 samplesize, ncores, ncluster)
ende <- Sys.time()

saveRDS(clara_result, paste0("/result_clara_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))


system("kill $(pgrep -f free)")

print(paste("clara done", Sys.time()))


zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "clara"))
```

```
print(paste("clara eval", Sys.time()))
resultlist[algo == "clara", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
             data.table(rec_pre_f1(orig, clara_result, n),
                        rand = rand(clara_result, orig, n),
                        purity = purity(orig, clara_result,
                         n, ncluster),
                        entropy = Entropy(orig, clara_result, n),
                        nmi = NMI(clara_result, orig),
                        zeit = zeit1)]

print(paste("clara eval done", Sys.time()))

ram_clara <- fread(paste0("ram/clara_", corstructure, "_",
 setup, "_", n_cluster, "_", praevalenz, "_", n, "_",
  n_cols, ".txt"))
ram_clara <- max(ram_clara$V1) - ram_pre

resultlist[algo == "clara", ram := ram_clara]


tablelist <- append(tablelist,
 list(clara = bigtable(as.matrix(clara_result), 1)))

rm(clara_result, nsamples, samplesize)
registerDoSEQ()

gc()

# clarans ------------------------------------------------------------

registerDoMC(ncores)

print("clarans")

numlocal <- 2
maxneighbor <- 1000

print(paste("clarans", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
```

```
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/clarans_' ,
 corstructure, "_", setup, "_", n_cluster, "_",
  praevalenz, "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
clarans_result <- clarans_fun(dataset, numlocal, ncluster,
maxneighbor, ncores)
ende <- Sys.time()

saveRDS(clarans_result, paste0("/result_clarans_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("clarans done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "clarans"))


print(paste("clarans eval", Sys.time()))
resultlist[algo == "clarans", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, clarans_result[[2]], n),
            rand = rand(clarans_result[[2]], orig, n),
            purity = purity(orig, clarans_result[[2]],
             n, ncluster),
            entropy = Entropy(orig, clarans_result[[2]], n),
            nmi = NMI(clarans_result[[2]], orig),
            zeit = zeit1)]

print(paste("clarans eval done", Sys.time()))

ram_clarans <- fread(paste0("ram/clarans_", corstructure,
 "_", setup, "_", n_cluster, "_", praevalenz, "_", n, "_",
  n_cols, ".txt"))
ram_clarans <- max(ram_clarans$V1) - ram_pre
```

```
resultlist[algo == "clarans", ram := ram_clarans]

tablelist <- append(tablelist,
 list(clarans = bigtable(as.matrix(clarans_result[[2]]), 1)))

rm(clarans_result, numlocal, maxneighbor)
registerDoSEQ()

gc()


# bubble ------------------------------------------------

print("bubble")

registerDoMC(ncores)

threshold_L <- cross_bubble(dataset, ncluster, orig)
threshold <- threshold_L$threshold
L <- threshold_L$L


print(paste("bubble", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
awk \'{print $3; system("")}\' > ', 'ram/bubble_' ,
 corstructure, "_", setup, "_", n_cluster, "_",
 praevalenz, "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
bubble_result <- bubble(dataset, ncores, threshold, L)
ende <- Sys.time()


system("kill $(pgrep -f free)")

print(paste("bubble done", Sys.time()))
```

```
zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "bubble"))

result_pam <- cluster::pam(rdist(bubble_result$clustroids,
 metric = "hamming"), k = ncluster)


for (i in 1:ncluster) {
  set(bubble_result$clusterlabels,
   which(bubble_result$clusterlabels$label %in% rownames(
    bubble_result$clustroids[which(result_pam$clustering == i),])),
     j = 1L, value = i)
}

saveRDS(bubble_result$clusterlabels$label, paste0("/result_bubble_",
  paste(corstructure, setup, n_cluster, praevalenz,
   n, n_cols, sep = "_"), ".RDS"))

print(paste("bubble eval", Sys.time()))
resultlist[algo == "bubble", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, bubble_result$clusterlabels$label, n),
            rand = rand(bubble_result$clusterlabels$label, orig, n),
            purity = purity(orig, bubble_result$clusterlabels$label,
             n, ncluster),
            entropy = Entropy(orig, bubble_result$clusterlabels$label,
             n),
            nmi = NMI(bubble_result$clusterlabels$label, orig),
            zeit = zeit1)]

print(paste("bubble eval done", Sys.time()))

ram_bubble <- fread(paste0("ram/bubble_", corstructure, "_",
 setup, "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols,
  ".txt"))
ram_bubble <- max(ram_bubble$V1) - ram_pre

resultlist[algo == "bubble", ram := ram_bubble]

tablelist <- append(tablelist,
 list(bubble = bigtable(as.matrix(bubble_result$clusterlabels$label),
  1)))
```

```
rm(bubble_result, result_pam, threshold, L, ram_bubble)
registerDoSEQ()

gc()


# mona ------------------------------------------------------------------

registerDoMC(ncores)

print("mona")

nvars <- ncol(dataset)
n_splits <- ncluster - 1
datasetdt <- as.data.table(dataset)

print(paste("mona", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/mona_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_",
  n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
mona_result <- mona(datasetdt, n_splits, nvars, ncores, ncols)
ende <- Sys.time()

saveRDS(mona_result, paste0("/result_mona_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("mona done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "mona"))
```

```
print(paste("mona eval", Sys.time()))
resultlist[algo == "mona", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, mona_result, n),
            rand = rand(mona_result, orig, n),
            purity = purity(orig, mona_result,
             n, ncluster),
            entropy = Entropy(orig, mona_result, n),
            nmi = NMI(mona_result, orig),
            zeit = zeit1)]
print(paste("mona eval done", Sys.time()))

ram_mona <- fread(paste0("ram/mona_", corstructure, "_",
setup, "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols,
 ".txt"))
ram_mona <- max(ram_mona$V1) - ram_pre

resultlist[algo == "mona", ram := ram_mona]

tablelist <- append(tablelist,
 list(mona = bigtable(as.matrix(mona_result), 1)))
rm(mona_result, datasetdt, nvars, n_splits)
registerDoSEQ()

gc()


# mbt ------------------------------------------------------------

registerDoMC(ncores)

print("mbt")

datasetdt <- as.data.table(dataset)

# crossvalidation for threshold

threshold <- n / 100

print(paste("mbt", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
```

```
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 |
 grep --line-buffered Mem | awk \'{print $3; system("")}\' > ',
 'ram/mbt_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz,
  "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
mbt_result <- mbt_par(datasetdt, threshold, ncluster)
ende <- Sys.time()

saveRDS(mbt_result, paste0("/result_mbt_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("mbt done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "mbt"))

print(paste("mbt eval", Sys.time()))
resultlist[algo == "mbt", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, mbt_result, n),
            rand = rand(mbt_result, orig, n),
            purity = purity(orig, mbt_result,
             n, ncluster),
            entropy = Entropy(orig, mbt_result, n),
            nmi = NMI(mbt_result, orig),
            zeit = zeit1)]
print(paste("mbt eval done", Sys.time()))

ram_mbt <- fread(paste0("ram/mbt_", corstructure, "_",
setup, "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols,
".txt"))
ram_mbt <- max(ram_mbt$V1) - ram_pre

resultlist[algo == "mbt", ram := ram_mbt]
```

```
tablelist <- append(tablelist,
 list(mbt = bigtable(as.matrix(mbt_result), 1)))

rm(threshold, mbt_result, datasetdt)
registerDoSEQ()

gc()

# clope ------------------------------------------------------

print("clope")

registerDoMC(ncores)

r <- 2

print(paste("clope", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/clope_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz,
  "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
clope_result <- clope_par(dataset, r, ncores, maxiteration = 10,
 maxmoved = 50, clusterstop = 100)
ende <- Sys.time()

saveRDS(clope_result, paste0("/result_clope_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("clope done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
```

```
print(paste(zeit1, "clope"))

print(paste("clope eval", Sys.time()))
resultlist[algo == "clope", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, clope_result, n),
            rand = rand(clope_result, orig, n),
            purity = purity(orig, clope_result,
             n, ncluster),
            entropy = Entropy(orig, clope_result, n),
            nmi = NMI(clope_result, orig),
            zeit = zeit1)]
print(paste("clope eval done", Sys.time()))

ram_clope <- fread(paste0("ram/clope_", corstructure, "_",
 setup, "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols,
  ".txt"))
ram_clope <- max(ram_clope$V1) - ram_pre

resultlist[algo == "clope", ram := ram_clope]

tablelist <- append(tablelist,
 list(clope = bigtable(as.matrix(clope_result), 1)))

rm(r, clope_result)
registerDoSEQ()

gc()

# large_item ----------------------------------------------------------

print("large_item")

registerDoMC(ncores)

print(paste("large_item", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
```

```
 awk \'{print $3; system("")}\' > ', 'ram/large_item_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz,
  "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
large_item_result <- largeitem_par(dataset, w = 1,
 minsupport = 0.8, maxiteration = 10, maxmoved = 10,
  ncores = ncores)
ende <- Sys.time()

saveRDS(large_item_result, paste0("/result_large_item_",
 paste(corstructure, setup, n_cluster, praevalenz, n,
  n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("large_item done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "large_item"))

print(paste("large_item eval", Sys.time()))
resultlist[algo == "large_item", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, large_item_result, n),
            rand = rand(large_item_result, orig, n),
            purity = purity(orig, large_item_result,
             n, ncluster),
            entropy = Entropy(orig, large_item_result, n),
            nmi = NMI(large_item_result, orig),
            zeit = zeit1)]
print(paste("large_item eval", Sys.time()))

tablelist <- append(tablelist,
 list(large_item = bigtable(as.matrix(large_item_result), 1)))

ram_large_item <- fread(paste0("ram/large_item_", corstructure,
 "_", setup, "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols,
  ".txt"))
ram_large_item <- max(ram_large_item$V1) - ram_pre

resultlist[algo == "large_item", ram := ram_large_item]
```

```
rm(large_item_result)
registerDoSEQ()

gc()

# scale ------------------------------------------------------------

if (seed > 0) {

print("scale")
registerDoMC(ncores)

nsample <- 2000
maxiteration <- 10
maxmoved <- 100
chunk_n <- n / 100 * 2

print(paste("scale", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
 grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/scale_' ,
  corstructure, "_", setup, "_", n_cluster, "_", praevalenz,
   "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
scale_results <- scale_cluster(dataset, nsample, ncluster,
 ncores, seed, maxiteration, maxmoved, chunk_n)
ende <- Sys.time()

saveRDS(scale_results, paste0("/result_scale_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("scale done", Sys.time()))
```

```
zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "scale"))

print(paste("scale eval", Sys.time()))
resultlist[algo == "scale", c("recall", "precision", "f1",
 "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, scale_results, n),
            rand = rand(scale_results, orig, n),
            purity = purity(orig, scale_results,
             n, ncluster),
            entropy = Entropy(orig, scale_results, n),
            nmi = NMI(scale_results, orig),
            zeit = zeit1)]
print(paste("scale eval done", Sys.time()))

ram_scale <- fread(paste0("ram/scale_", corstructure, "_",
 setup, "_", n_cluster, "_", praevalenz, "_", n, "_",
  n_cols, ".txt"))
ram_scale <- max(ram_scale$V1) - ram_pre

resultlist[algo == "scale", ram := ram_scale]

tablelist <- append(tablelist,
 list(scale = bigtable(as.matrix(scale_results), 1)))

rm(scale_results, nsample, maxiteration, maxmoved,
 chunk_n, ram_scale)
registerDoSEQ()

gc()

} # if (seed > 85)


# sorted_neighborhood ----------------------------------------------

print("sorted_neighborhood")

registerDoMC(ncores)

threshold <- 0.5

windowsize <- 100
```

```
print(paste("sorted_neighborhood", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/sorted_neighborhood_' ,
corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_",
 n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
sorted_neighborhood_result <- sorted_neighborhood(dataset,
 windowsize, threshold = threshold, runs = 3, ncores = ncores)
ende <- Sys.time()

saveRDS(sorted_neighborhood_result, paste0("/result_sn_",
  paste(corstructure, setup, n_cluster, praevalenz,
   n, n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("sorted_neighborhood done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "sorted_neighborhood"))

print(paste("sorted_neighborhood eval", Sys.time()))
resultlist[algo == "sorted_neighborhood",
 c("recall", "precision", "f1", "rand", "purity",
  "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, sorted_neighborhood_result, n),
            rand = rand(sorted_neighborhood_result, orig, n),
            purity = purity(orig, sorted_neighborhood_result,
             n, ncluster),
            entropy = Entropy(orig, sorted_neighborhood_result, n),
            nmi = NMI(sorted_neighborhood_result, orig),
            zeit = zeit1)]
print(paste("sorted_neighborhood eval done", Sys.time()))

ram_sorted_neighborhood <- fread(paste0("ram/sorted_neighborhood_",
```

```
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_", n,
  "_", n_cols, ".txt"))
ram_sorted_neighborhood <- max(ram_sorted_neighborhood$V1) - ram_pre

resultlist[algo == "sorted_neighborhood",
 ram := ram_sorted_neighborhood]

tablelist <- append(tablelist,
 list(sorted_neighborhood = bigtable(
  as.matrix(sorted_neighborhood_result),
  1)))

rm(threshold, windowsize, sorted_neighborhood_result,
 ram_sorted_neighborhood)
registerDoSEQ()

gc()


# proximus ----------------------------------------------------

if (seed > 0) {
print("proximus")

registerDoMC(ncores)

maxradius <- ncols

print(paste("proximus", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/proximus_' ,
 corstructure, "_", setup, "_", n_cluster, "_",
 praevalenz, "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()

ncores <- 30
```

```
registerDoMC(ncores)


proximus_result <- prox(dataset, maxradius, ncores, ncluster)

ende <- Sys.time()

saveRDS(proximus_result$clusters, paste0("/result_proximus_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
  n_cols, sep = "_"), ".RDS"))

ncores <- 10
registerDoMC(ncores)

system("kill $(pgrep -f free)")

print(paste("proximus done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "proximus"))

print(paste("proximus eval", Sys.time()))
resultlist[algo == "proximus", c("recall", "precision",
 "f1", "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, proximus_result$clusters, n),
            rand = rand(proximus_result$clusters, orig, n),
            purity = purity(orig, proximus_result$clusters,
             n, ncluster),
            entropy = Entropy(orig, proximus_result$clusters,
             n),
            nmi = NMI(proximus_result$clusters, orig),
            zeit = zeit1)]
print(paste("proximus eval done", Sys.time()))

ram_proximus <- fread(paste0("ram/proximus_", corstructure, "_",
 setup, "_", n_cluster, "_", praevalenz, "_", n, "_", n_cols,
  ".txt"))
ram_proximus <- max(ram_proximus$V1) - ram_pre

resultlist[algo == "proximus", ram := ram_proximus]

tablelist <- append(tablelist,
```

```
 list(proximus = bigtable(as.matrix(proximus_result$clusters), 1)))

rm(maxradius, proximus_result, ram_proximus)
registerDoSEQ()

gc()

}
# canopy ------------------------------------------------------
if (seed > 0) {
  print("canopy")

  # cl <- makeCluster(ncores)
  registerDoMC(ncores)

  threshold <- cross_canopy(dataset, ncluster, orig)

  print(paste("canopy", Sys.time()))
  gc()
  ram_pre <- as.numeric(system('free --mega |
  grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
  intern = TRUE))

  system(paste0('free --mega -s 1 | grep --line-buffered Mem |
   awk \'{print $3; system("")}\' > ', 'ram/canopy_' ,
   corstructure, "_", setup, "_", n_cluster, "_",
   praevalenz, "_", n, "_", n_cols, '.txt'), wait = FALSE)

  anfang <- Sys.time()
  canopy_result <- canopy_cluster_par(dataset, threshold,
   threshold, ncluster, ncores)
  ende <- Sys.time()

  saveRDS(canopy_result, paste0("/result_canopy_",
   paste(corstructure, setup, n_cluster, praevalenz, n,
    n_cols, sep = "_"), ".RDS"))

  system("kill $(pgrep -f free)")

  print(paste("canopy done", Sys.time()))
```

```
  zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
  print(paste(zeit1, "canopy"))

  print(paste("canopy eval", Sys.time()))
  resultlist[algo == "canopy", c("recall", "precision", "f1",
   "rand", "purity", "entropy", "nmi", "zeit") :=
   data.table(rec_pre_f1(orig, canopy_result, n),
              rand = rand(canopy_result, orig, n),
              purity = purity(orig, canopy_result,
               n, ncluster),
              entropy = Entropy(orig, canopy_result, n),
              nmi = NMI(canopy_result, orig),
              zeit = zeit1)]
  print(paste("canopy eval done", Sys.time()))

  ram_canopy <- fread(paste0("ram/canopy_", corstructure, "_",
   setup, "_", n_cluster, "_", praevalenz, "_", n, "_",
    n_cols, ".txt"))
  ram_canopy <- max(ram_canopy$V1) - ram_pre

  resultlist[algo == "canopy", ram := ram_canopy]

  tablelist <- append(tablelist,
   list(canopy = bigtable(as.matrix(canopy_result), 1)))

  rm(threshold, canopy_result, ram_canopy)
  registerDoSEQ()

  gc()
}


# general model ------------------------------------------------

print("general_model_algo_2")

if (setup != "einfach") {
  nclusterf <- 4
} else {
  nclusterf <- 2
}
```

```
registerDoMC(ncores)

print(paste("general_model_algo_2", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
 intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/general_model_algo_2_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz,
  "_", n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
g_model_result2 <- g_mod_bin_data2_par(dataset, ncluster, ncores)
ende <- Sys.time()

saveRDS(g_model_result2, paste0("/result_general_model_algo_2_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("general_model_algo_2 done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "general model2"))

print(paste("general_model_algo_2 eval", Sys.time()))
resultlist[algo == "general_model_algo_2", c("recall", "precision",
 "f1", "rand", "purity", "entropy", "nmi", "zeit") :=
 data.table(rec_pre_f1(orig, g_model_result2, n),
            rand = rand(g_model_result2, orig, n),
            purity = purity(orig, g_model_result2,
             n, ncluster),
            entropy = Entropy(orig, g_model_result2, n),
            nmi = NMI(g_model_result2, orig),
            zeit = zeit1)]
print(paste("general_model_algo_2 eval done", Sys.time()))

ram_general_model_algo_2 <- fread(paste0("ram/general_model_algo_2_",
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_", n,
```

```
  "_", n_cols, ".txt"))
ram_general_model_algo_2 <- max(ram_general_model_algo_2$V1) - ram_pre

resultlist[algo == "general_model_algo_2",
 ram := ram_general_model_algo_2]

tablelist <- append(tablelist,
 list(general_model_algo_2 = bigtable(as.matrix(g_model_result2), 1)))

rm(g_model_result2, ram_general_model_algo_2)

print("general model1")

print(paste("general_model_algo_1", Sys.time()))
gc()
ram_pre <- as.numeric(system('free --mega |
grep --line-buffered Mem | awk \'{print $3; system("")}\' ',
intern = TRUE))

system(paste0('free --mega -s 1 | grep --line-buffered Mem |
 awk \'{print $3; system("")}\' > ', 'ram/general_model_algo_1_' ,
 corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_",
  n, "_", n_cols, '.txt'), wait = FALSE)

anfang <- Sys.time()
g_model_result1 <- g_mod_bin_data1_par(dataset, ncluster,
 nclusterf, ncores)
ende <- Sys.time()

saveRDS(g_model_result1, paste0("/result_general_model_algo_1_",
  paste(corstructure, setup, n_cluster, praevalenz, n,
   n_cols, sep = "_"), ".RDS"))

system("kill $(pgrep -f free)")

print(paste("general_model_algo_1 done", Sys.time()))

zeit1 <- as.numeric(difftime(ende, anfang, units = "secs"))
print(paste(zeit1, "general model1"))

print(paste("general_model_algo_1 eval", Sys.time()))
resultlist[algo == "general_model_algo_1", c("recall", "precision",
```

```
    "f1", "rand", "purity", "entropy", "nmi", "zeit") :=
    data.table(rec_pre_f1(orig, g_model_result1, n),
               rand = rand(g_model_result1, orig, n),
               purity = purity(orig, g_model_result1,
                n, ncluster),
               entropy = Entropy(orig, g_model_result1, n),
               nmi = NMI(g_model_result1, orig),
               zeit = zeit1)]
  print(paste("general_model_algo_1 eval done", Sys.time()))

  ram_general_model_algo_1 <- fread(paste0("ram/general_model_algo_1_",
   corstructure, "_", setup, "_", n_cluster, "_", praevalenz, "_", n,
    "_",
    n_cols, ".txt"))
  ram_general_model_algo_1 <- max(ram_general_model_algo_1$V1) - ram_pre

  resultlist[algo == "general_model_algo_1",
  ram := ram_general_model_algo_1]

  tablelist <- append(tablelist,
   list(general_model_algo_1 = bigtable(as.matrix(g_model_result1), 1)))

  rm(g_model_result1, ram_general_model_algo_1)

  registerDoSEQ()

  gc()


  saveRDS(resultlist, paste0("/resultlist_", paste(corstructure,
   setup, n_cluster, praevalenz, n, n_cols, sep = "_"), ".RDS"))
  saveRDS(tablelist, paste0("/tablelist_", paste(corstructure,
   setup, n_cluster, praevalenz, n, n_cols, sep = "_"), ".RDS"))

  resultlist_list[[paste(corstructure, setup, n_cluster,
   praevalenz, n, n_cols, sep = "_")]] <- resultlist
  tablelist_list[[paste(corstructure, setup, n_cluster,
   praevalenz, n, n_cols, sep = "_")]] <- tablelist
} # if seed

      } # special einfach
    } # setup
```

```
      } # n_cluster
    } # praevalenz
  }

} # n


ende_alles <- Sys.time()
ende_alles - anfang_alles
```

Literatur

Adler, D., Gläser, C., Nenadic, O., Oehlschlägel, J., Schuemie, M., & Zucchini, W. (2021). *ff: Memory-Efficient Storage of Large Data on Disk and Fast Access Functions* [R package version 4.0.5]. https://CRAN.R-project.org/package=ff

Aggarwal, C. (2014). *Data Clustering : Algorithms and Applications*. CRC Press.

Amatya, A., Demirtas, H., & Gao, R. (2019). *MultiOrd: Generation of Multivariate Ordinal Variates* [R package version 2.4.1]. https://CRAN.R-project.org/package=MultiOrd

Anderlucci, L., & Hennig, C. (2014). The Clustering of Categorical Data: A Comparison of a Model-based and a Distance-based Approach. *Communications in Statistics - Theory and Methods*, *43*(4), 704–721. https://doi.org/10.1080/03610926.2013.806665

Andreopoulos, B., An, A., Wang, X., & Schroeder, M. (2009). A Roadmap of Clustering Algorithms: Finding a Match for a Biomedical Application. *Briefings in Bioinformatics*, *10*(3), 297–314. https://doi.org/10.1093/bib/bbn058

Andrews, J. L., McNicholas, P. D., & Subedi, S. (2011). Model-Based Classification Via Mixtures of Multivariate t-Distributions. *Computational Statistics & Data Analysis*, *55*(1), 520–529. https://doi.org/10.1016/j.csda.2010.05.019

Andritsos, P., Tsaparas, P., Miller, R. J., & Sevcik, K. C. (2004). LIMBO: Scalable Clustering of Categorical Data. In E. Bertino, W. Gao, B. Steffen & M. Yung (Hrsg.), *Advances in Database Technology - EDBT 2004* (S. 123–146, Bd. 2992). Springer. https://doi.org/10.1007/978-3-540-24741-8_9

Anstead, N., & O'Loughlin, B. (2014). Social Media Analysis and Public Opinion: The 2010 UK General Election. *Journal of Computer-Mediated Communication*, *20*(2), 204–220. https://doi.org/10.1111/jcc4.12102

Arora, S., & Chana, I. (2014). A Survey of Clustering Techniques for Big Data Analysis. In B. Shukla, A. Bansal, N. Hasteer & A. Singhal (Hrsg.), *5th International Conference - Confluence The Next Generation Information Technology Summit*. IEEE. https://doi.org/10.1109/confluence.2014.6949256

Azzalini, A., & Menardi, G. (2016). Density-Based Clustering with Non-Continuous Data. *Computational Statistics*, *31*(2), 771–798. https://doi.org/10.1007/s00180-016-0644-8

Bacher, J., Pöge, A., & Wenzig, K. (2010). *Clusteranalyse*. Oldenbourg Verlag.

Bachteler, T., Reiher, J., & Schnell, R. (2013). *Similarity Filtering with Multibit Trees for Record Linkage* (Working Paper Nr. WP-GRLC-2013-01). German Record Linkage Center.

Bader, G. D., & Hogue, C. W. V. (2003). An Automated Method for Finding Molecular Complexes in Large Protein Interaction Networks. *BMC Bioinformatics*, *4*(2). https://doi.org/10.1186/1471-2105-4-2

Baeza-Yates, R. (1999). *Modern Information Retrieval*. ACM Press Addison-Wesley.

Bahadur, R. R. (1961). A Representation of the Joint Distribution of Responses to n Dichotomous Items. *Stanford Mathematical Studies in the Social Sciences*, *6*, 158–168.

Barbará, D., Li, Y., & Couto, J. (2002). COOLCAT: An Entropy-Based Algorithm for Categorical Clustering. In K. Kalpakis & D. Grossmann (Hrsg.), *Proceedings of the Eleventh International Conference on Information and Knowledge Management - CIKM '02* (S. 582–589). ACM Press. https://doi.org/10.1145/584792.584888

Bates, D., & Maechler, M. (2021). *Matrix: Sparse and Dense Matrix Classes and Methods* [R package version 1.3-4]. https://CRAN.R-project.org/package=Matrix

Becker, G., & Jenkins, B. (2015). *fastdigest: Fast, Low Memory-Footprint Digests of R Objects* [R package version 0.6-3]. https://CRAN.R-project.org/package=fastdigest

Benabdellah, A. C., Benghabrit, A., & Bouhaddou, I. (2019). A Survey of Clustering Algorithms for an Industrial Context. *Procedia Computer Science*, *148*, 291–302. https://doi.org/10.1016/j.procs.2019.01.022

Bengtsson, H. (2021). *matrixStats: Functions that Apply to Rows and Columns of Matrices (and to Vectors)* [R package version 0.61.0]. https://CRAN.R-project.org/package=matrixStats

Berkhin, P. (2002). A Survey of Clustering Data Mining Techniques. In J. Kogan, C. Nicholas & M. Teboulle (Hrsg.), *Grouping Multidimensional Data* (S. 25–71). Springer. https://doi.org/10.1007/3-540-28349-8_2

Bezdek, J. C. (1981). Pattern Recognition with Fuzzy Objective Function Algorithms.

Bibi, M., Aziz, W., Almaraashi, M., Khan, I. H., Nadeem, M. S. A., & Habib, N. (2020). A Cooperative Binary-Clustering Framework Based on Majority Voting for Twitter

Sentiment Analysis. *IEEE Access*, *8*, 68580–68592. https://doi.org/10.1109/access.2020.2983859

Bin, N. (2018). Research on Methods and Techniques for IoT Big Data Cluster Analysis. In J. K. Zhang & X. P. Zheng (Hrsg.), *2018 International Conference on Information Systems and Computer Aided Education*. IEEE. https://doi.org/10.1109/iciscae.2018.8666889

Blaser, N. (2020). *rdist: Calculate Pairwise Distances* [R package version 0.0.5]. https://CRAN.R-project.org/package=rdist

Bonchi, F., Gionis, A., Gullo, F., Tsourakakis, C. E., & Ukkonen, A. (2015). Chromatic Correlation Clustering. *ACM Transactions on Knowledge Discovery from Data*, *9*(4), 1–24. https://doi.org/10.1145/2728170

Borg, I., & Groenen, P. J. (2005). *Modern Multidimensional Scaling: Theory and Applications* (2. Aufl.). Springer.

Boulianne, S. (2015). Social Media Use and Participation: A Meta-Analysis of Current Research. *Information, Communication & Society*, *18*(5), 524–538. https://doi.org/10.1080/1369118x.2015.1008542

Broder, A. Z. (1997). On the Resemblance and Containment of Documents. In B. Carpentieri, A. D. Santis, U. Vaccaro & J. A. Storer (Hrsg.), *Proceedings. Compression and Complexity of SEQUENCES 1997* (S. 21–29). IEEE. https://doi.org/10.1109/sequen.1997.666900

Brusco, M. J., Singh, R., Cradit, J. D., & Steinley, D. (2017). Cluster Analysis in Empirical Om Research: Survey and Recommendations. *International Journal of Operations & Production Management*, *37*(3), 300–320. https://doi.org/10.1108/ijopm-08-2015-0493

Buchta, C., & Hahsler, M. (2019). *cba: Clustering for Business Analytics* [R package version 0.2-22]. https://CRAN.R-project.org/package=cba

Budayan, C., Dikmen, I., & Birgonul, M. T. (2009). Comparing the Performance of Traditional Cluster Analysis, Self-Organizing Maps and Fuzzy C-Means Method for Strategic Grouping. *Expert Systems with Applications*, *36*(9), 11772–11781. https://doi.org/10.1016/j.eswa.2009.04.022

Carter, J. L., & Wegman, M. N. (1979). Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, *18*(2), 143–154. https://doi.org/10.1016/0022-0000(79)90044-8

Chen, K., & Liu, L. (2005a). The "Best K" for Entropy-Based Categorical Data Clustering. In J. Frew (Hrsg.), *Proceedings of the 17th International Conference on Scientific and StatisticalDatabase Management (SSDBM 2005)* (S. 253–262).

Chen, K., & Liu, L. (2005b). The „Best K" for Entropy-based Categorical Data Clustering [Paper presented at the Scientific and Statistical Database Management Conference (SSDBM05)].

Chernoff, H. (1952). A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *The Annals of Mathematical Statistics*, *23*(4), 493–507. https://doi.org/10.1214/aoms/1177729330

Chiquet, J., Rigaill, G., & Sundqvist, M. (2020). *aricode: Efficient Computations of Standard Clustering Comparison Measures* [R package version 0.1.2]. https://CRAN.R-project.org/package=aricode

Christen, P. (2012). *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

Cormen, T. H., Leiserson, C. F., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3. Aufl.). The MIT Press.

Costa, E., Papatsouma, I., & Markos, A. (2022). Benchmarking Distance-Based Partitioning Methods for Mixed-Type Data. *Advances in Data Analysis and Classification*, *17*(3), 701–724. https://doi.org/10.1007/s11634-022-00521-7

Cover, T. M., & Thomas, J. A. (2006). *Elements of Information Theory* (2. Aufl.). Wiley.

Dafir, Z., Lamari, Y., & Slaoui, S. C. (2020). A Survey on Parallel Clustering Algorithms for Big Data. *Artificial Intelligence Review*, *54*(4), 2411–2443. https://doi.org/10.1007/s10462-020-09918-2

d'Angella, G., & Hennig, C. (2022). A Comparison of Different Clustering Approaches for High-Dimensional Presence-Absence Data. In A. Bekker, J. T. Ferreira, M. Arashi & D.-G. Chen (Hrsg.), *Innovations in Multivariate Statistical Modeling* (S. 299–318). Springer. https://doi.org/10.1007/978-3-031-13971-0_13

Danie, F., Revolution Analytics & Weston, S. (2022). *doMC: Foreach Parallel Adaptor for „parallel"* [R package version 1.3.7].

Das, A., Datar, M., Garg, A., & Rajaram, S. (2007). Google News Personalization: Scalable Online Collaborative Filtering. *Proceedings of the 16th International Conference on World Wide Web - WWW '07*, 271–280. https://doi.org/10.1145/1242572.1242610

Dash, R., & Misra, B. B. (2018). Performance Analysis of Clustering Techniques Over Microarray Data: A Case Study. *Physica A: Statistical Mechanics and its Applications*, *493*, 162–176. https://doi.org/10.1016/j.physa.2017.10.032

Dave, M., & Gianey, H. (2016). Different Clustering Algorithms for Big Data Analytics: A Review. In R. K. Dwivedi, A. B. A. Hamid, D. Ather, A. K. Saxena & A. K. Agarwal (Hrsg.), *2016 International Conference System Modeling & Advancement in Research Trends* (S. 328–333). IEEE. https://doi.org/10.1109/sysmart.2016.7894544

Deming, W. E., & Stephan, F. F. (1940). On a Least Squares Adjustment of a Sampled Frequency Table when the Expected Marginal Totals Are Known. *The Annals of Mathematical Statistics*, *11*(4), 427–444.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, *39*(1), 1–22. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer.

Dice, L. R. (1945). Measures of the Amount of Ecologic Association Between Species. *Ecology*, *26*(3), 297–302.

Dimitriadou, E., Dolničar, S., & Weingessel, A. (2002). An Examination of Indexes for Determining the Number of Clusters in Binary Data Sets. *Psychometrika*, *67*(3), 137–160. https://doi.org/10.1007/bf02294713

DomPazz. (2011). Non-PD Matrices in R, Cont. [Blogeintrag auf R-bloggers]. Verfügbar 6. März 2020 unter https://www.r-bloggers.com/2011/12/non-pd-matrices-in-r-cont/

Dowle, M., & Srinivasan, A. (2019). *data.table: Extension of 'data.frame'* [R package version 1.12.8]. https://CRAN.R-project.org/package=data.table

Dueck, G., & Scheuer, T. (1990). Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics*, *90*(1), 161–175. https://doi.org/10.1016/0021-9991(90)90201-b

Eckert, A. (2018). *parallelDist: Parallel Distance Matrix Computation using Multiple Threads* [R package version 0.2.4]. https://CRAN.R-project.org/package=parallelDist

Emrich, L. J., & Piedmonte, M. R. (1991). A Method for Generating High-Dimensional Multivariate Binary Variates. *The American Statistician*, *45*(4), 302–304.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In E. Simoudis, J. Han & U. Fayyad (Hrsg.), *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (S. 226–231). AAAI Press.

Everitt, B. S. (1988). A Finite Mixture Model for the Clustering of Mixed-Mode Data. *Statistics & Probability Letters*, *6*(5), 305–309. https://doi.org/10.1016/0167-7152(88)90004-1

Everitt, B. S., Landau, S., Leese, M., & Stahl, D. (2011). *Cluster Analysis* (5. Aufl.). Wiley.

Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O., Abualigah, L., Agushaka, J. O., Eke, C. I., & Akinyelu, A. A. (2022). A Comprehensive Survey of Clustering Algorithms: State-of-the-art Machine Learning Applications, Taxonomy, Challenges, and Future Research Prospects. *Engineering Applications of Artificial Intelligence*, *110*, 104743. https://doi.org/10.1016/j.engappai.2022.104743

Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., Foufou, S., & Bouras, A. (2014). A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis. *IEEE Transactions on Emerging Topics in Computing*, *2*(3), 267–279. https://doi.org/10.1109/tetc.2014.2330519

Fahrmeir, L., Heumann, C., Künstler, R., Pigeot, I., & Tutz, G. (2016). *Statistik: Der Weg zur Datenanalyse* (8. Aufl.). Springer.

Farrell, P. J., & Sutradhar, B. C. (2006). A Non-linear Conditional Probability Model for Generating Correlated Binary Data. *Statistics & Probability Letters*, *76*(4), 353–361. https://doi.org/10.1016/j.spl.2005.08.012

Fasiolo, M. (2016). *An Introduction to Mvnfast* [R Package Version 0.2.7]. https://CRAN.R-project.org/package=mvnfast

Ferrari, S., & Cribari-Neto, F. (2004). Beta Regression for Modelling Rates and Proportions. *Journal of Applied Statistics*, *31*(7), 799–815. https://doi.org/10.1080/0266476042000214501

Foss, A., Markatou, M., Ray, B., & Heching, A. (2016). A Semiparametric Method for Clustering Mixed Data. *Machine Learning*, *105*(3), 419–458. https://doi.org/10.1007/s10994-016-5575-7

Fraley, C., & Raftery, A. E. (2002). Model-Based Clustering, Discriminant Analysis, and Density Estimation. *Journal of the American Statistical Association*, *97*(458), 611–631. https://doi.org/10.1198/016214502760047131

Frey, B. J., & Dueck, D. (2007). Clustering by Passing Messages Between Data Points. *Science*, *315*(5814), 972–976. https://doi.org/10.1126/science.1136800

Gan, G. (2011). *Data Clustering in C++. An Object-Oriented Approach*. Chapman & Hall.

Gan, G., Ma, C., & Wu, J. (2007). *Data Clustering: Theory, Algorithms, and Applications*. Siam.

Gange, S. J. (1995). Generating Multivariate Categorical Variates Using the Iterative Proportional Fitting Algorithm. *The American Statistician*, *49*(2), 134–138. https://doi.org/10.1080/00031305.1995.10476130

Ganti, V., Gehrke, J., & Ramakrishnan, R. (1999). CACTUS – Clustering Categorical Data Using Summaries. In U. Fayyad, S. Chaudhuri & D. Madigan (Hrsg.), *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '99* (S. 73–83). ACM Press. https://doi.org/10.1145/312129.312201

Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A., & French, J. (1999). Clustering Large Datasets in Arbitrary Metric Spaces. In M. Kitsuregawa, L. Maciaszek, M. Papazoglou & C. Pu (Hrsg.), *Proceedings 15th International Conference on Data Engineering*. Institute of Electrical; Electronics Engineers (IEEE). https://doi.org/10.1109/icde.1999.754966

Garg, A., Mangla, A., Gupta, N., & Bhatnagar, V. (2006). PBIRCH: A Scalable Parallel Clustering Algorithm for Incremental Data. In B. C. Desai & S. K. Gupta (Hrsg.), *10th International Database Engineering and Applications Symposium (IDEAS'06)*. Institute of Electrical und Electronics Engineers (IEEE). https://doi.org/10.1109/ideas.2006.36

Gaujoux, R. (2020). *doRNG: Generic Reproducible Parallel Backend for „foreach" Loops* [R package version 1.8.2]. https://CRAN.R-project.org/package=doRNG

Genest, C., & MacKay, J. (1986). The Joy of Copulas: Bivariate Distributions with Uniform Marginals. *The American Statistician*, *40*(4), 280–283. https://doi.org/10.2307/2684602

Gentle, J. E. (2007). *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer Nature.

Genz, A., Bretz, F., Miwa, T., Mi, X., Leisch, F., Scheipl, F., & Hothorn, T. (2020). *mvtnorm: Multivariate Normal and t Distributions* [R package version 1.0-12]. http://CRAN.R-project.org/package=mvtnorm

Glover, F. (1989). Tabu Search–Part I. *ORSA Journal on Computing*, *1*(3), 190–206. https://doi.org/10.1287/ijoc.1.3.190

Guha, S., Rastogi, R., & Shim, K. (2000). ROCK: a Robust Clustering Algorithm for Categorical Attributes. *Information Systems*, *25*(5), 345–366.

Guha, S., Rastogi, R., & Shim, K. (2001). Cure: An Efficient Clustering Algorithm for Large Databases. *Information Systems*, *26*(1), 35–58. https://doi.org/10.1016/s0306-4379(01)00008-4

Hadd, A. R., & Rodgers, J. L. (2021). *Understanding Correlation Matrices*. SAGE.

Han, J., Kamber, M., & Pei, J. (2012). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Hand, D., & Christen, P. (2017). A Note on Using the F-Measure for Evaluating Record Linkage Algorithms. *Statistics and Computing*, *28*(3), 539–547. https://doi.org/10.1007/s11222-017-9746-6

Handl, A., & Kuhlenkasper, T. (2017). *Multivariate Analysemethoden: Theorie und Praxis mit R* (3. Aufl.). Springer.

Hardin, J., Garcia, S. R., & Golan, D. (2013). A Method for Generating Realistic Correlation Matrices. *The Annals of Applied Statistics*, *7*(3), 1733–1762. https://doi.org/10.1214/13-aoas638

Hautamäki, V., Pöllänen, A., Kinnunen, T., Lee, K. A., Li, H., & Fränti, P. (2014). A Comparison of Categorical Attribute Data Clustering Methods. In P. Fränti, G. Brown, M. Loog, F. Escolano & M. Pelillo (Hrsg.), *Structural, Syntactic, and Statistical Pattern Recognition* (S. 53–62). Springer.

Haynes, M. E., Sabo, R. T., & Chaganty, N. R. (2015). Simulating Dependent Binary Variables through Multinomial Sampling. *Journal of Statistical Computation and Simulation*, *86*(3), 510–523. https://doi.org/10.1080/00949655.2015.1020313

He, Z., Xu, X., & Deng, S. (2002). Squeezer: An Efficient Algorithm for Clustering Categorical Data. *Journal of Computer Science and Technology*, *17*(5), 611–624. https://doi.org/10.1007/bf02948829

Headrick, T. C. (2002). JMASM3: A Method for Simulating Systems of Correlated Binary Data. *Journal of Modern Applied Statistical Methods*, *1*(1), 195–201. https://doi.org/10.22237/jmasm/1020256080

Hennig, C. (2022). An Empirical Comparison and Characterisation of Nine Popular Clustering Methods. *Advances in Data Analysis and Classification*, *16*(1), 201–229. https://doi.org/10.1007/s11634-021-00478-z

Hernández, M. A., & Stolfo, S. J. (1995). The Merge/Purge Problem for Large Databases. *ACM SIGMOD Record*, *24*(2), 127–138. https://doi.org/10.1145/568271.223807

Hernández, M. A., & Stolfo, S. J. (1998). Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, *2*(1), 9–37. https://doi.org/10.1023/a:1009761603038

Hinneburg, A., & Keim, D. A. (1998). An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In R. Agrawal & P. Stolorz (Hrsg.), *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (S. 58–65). AAAI Press.

Hinneburg, A., & Keim, D. A. (1999). Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In M. P. Atkinson & M. E. Orlowska (Hrsg.), *Proceedings of the 25th International Conference on Very Large Data Bases* (S. 506–517). Morgan Kaufmann Publishers Inc.

Huang, Z. (1998). Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Mining and Knowledge Discovery*, *2*, 283–304. https://doi.org/10.1023/A:1009769707641

Indyk, P., & Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing - STOC '98*, 604–613. https://doi.org/10.1145/276698.276876

Iqbal, F., Binsalleeh, H., Fung, B. C. M., & Debbabi, M. (2010). Mining Writeprints from Anonymous E-Mails for Forensic Investigation. *Digital Investigation*, *7*(1-2), 56–64. https://doi.org/10.1016/j.diin.2010.03.003

Iyigun, C., & Ben-Israel, A. (2008). Probabilistic Distance Clustering Adjusted for Cluster Size. *Probability in the Engineering and Informational Sciences*, *22*(4), 603–621. https://doi.org/10.1017/s0269964808000351

Jaccard, P. (1908). Nouvelles Recherches Sur La Distribution Florale. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, *44*(163), 223–270. https://doi.org/10.5169/SEALS-268384

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data Clustering: A Review. *ACM Computing Surveys*, *31*(3), 264–323. https://doi.org/10.1145/331499.331504

Jiang, D., Tang, C., & Zhang, A. (2004). Cluster Analysis for Gene Expression Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, *16*(11), 1370–1386. https://doi.org/10.1109/tkde.2004.68

Jimeno, J., Roy, M., & Tortora, C. (2021). Clustering Mixed-Type Data: A Benchmark Study on KAMILA and K-Prototypes. In T. Chadjipadelis, B. Lausen, A. Markos, T. R. Lee, A. Montanari & R. Nugent (Hrsg.), *Studies in Classification, Data Analysis, and Knowledge Organization* (S. 83–91). Springer. https://doi.org/10.1007/978-3-030-60104-1_10

Kane, M. J., Emerson, J., & Weston, S. (2013). Scalable Strategies for Computing with Massive Data. *Journal of Statistical Software*, *55*(14), 1–19.

Kane, M. J., & Emerson, J. W. (2016). *bigtabulate: Table, Apply, and Split Functionality for Matrix and „big.matrix“ Objects* [R package version 1.1.5]. https://CRAN.R-project.org/package=bigtabulate

Kanter, M. (1975). Autoregression for Discrete Processes Mod 2. *Journal of Applied Probability*, *12*(2), 371–375. https://doi.org/10.2307/3212453

Karypis, G. (2002). *CLUTO - A Clustering Toolkit* (Technischer Bericht Nr. 02-017). University of Minnesota, Department of Computer Science.

Karypis, G., Han, E.-H., & Kumar, V. (1999). Chameleon: Hierarchical Clustering using Dynamic Modeling. *Computer*, *32*(8), 68–75. https://doi.org/10.1109/2.781637

Kaufman, L., & Rousseeuw, P. J. (2005). *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley.

Kennedy, J., & Eberhart, R. C. (1997). A Discrete Binary Version of the Particle Swarm Algorithm. In IEEE (Hrsg.), *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* (S. 4104–4108). https://doi.org/10.1109/icsmc.1997.637339

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Kogan, J. (2007). *Introduction to Clustering Large and High-Dimensional Data*. Cambridge University Press.

Kohonen, T. (2001). *Self-Organizing Maps* (3. Aufl.). Springer.

Korzeniewski, J. (2016). New Method of Variable Selection for Binary Data Cluster Analysis. *Statistics in Transition. New Series*, *17*(2), 295–304. https://doi.org/10.21307/stattrans-2016-020

Kovic, M., Rauchfleisch, A., Sele, M., & Caspar, C. (2018). Digital Astroturfing in Politics: Definition, Typology, and Countermeasures. *Studies in Communication Sciences*, *18*(1), 69–85. https://doi.org/10.24434/j.scoms.2018.01.005

Koyuturk, M., Grama, A., & Ramakrishnan, N. (2005). Compression, Clustering, and Pattern Discovery in Very High-dimensional Discrete-attribute Data Sets. *IEEE Transactions on Knowledge and Data Engineering*, *17*(4), 447–461. https://doi.org/10.1109/tkde.2005.55

Koyutürk, M., & Grama, A. (2003). PROXIMUS: A Framework for Analyzing Very High Dimensional Discrete Attributed Datasets. *Proceedings of the Ninth ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '03, 147–156. https://doi.org/10.1145/956750.956770

Kristensen, T. G., Nielsen, J., & Pedersen, C. N. S. (2010). A Tree-based Method for the Rapid Screening of Chemical Fingerprints. *Algorithms for Molecular Biology*, *5*(9). https://doi.org/10.1186/1748-7188-5-9

Kumar, D., Bezdek, J. C., Palaniswami, M., Rajasegarar, S., Leckie, C., & Havens, T. C. (2016). A Hybrid Approach to Clustering in Big Data. *IEEE Transactions on Cybernetics*, *46*(10), 2372–2385. https://doi.org/10.1109/tcyb.2015.2477416

Lalonde, T. L. (2017). Monte-Carlo Simulation of Correlated Binary Responses. In D.-G. Chen & J. D. Chen (Hrsg.), *Monte-Carlo Simulation-Based Statistical Modeling*. Springer.

Lance, G. N., & Williams, W. T. (1968). Note on a New Information-Statistic Classificatory Program. *The Computer Journal*, *11*(2), 195–195. https://doi.org/10.1093/comjnl/11.2.195

L'Ecuyer, P. (1999). Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators. *Operations Research*, *47*(1), 159–164. https://doi.org/10.1287/opre.47.1.159

Lee, A. J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association. *The American Statistician*, *47*(3), 209–215. https://doi.org/10.1080/00031305.1993.10475980

Lee, S. X., & McLachlan, G. J. (2013). On Mixtures of Skew Normal and Skew t-Distributions. *Advances in Data Analysis and Classification*, *7*(3), 241–266. https://doi.org/10.1007/s11634-013-0132-8

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press.

Li, T. (2005). A General Model for Clustering Binary Data. *Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining - KDD '05*, 188–197. https://doi.org/10.1145/1081870.1081894

Li, Y., Le, J., & Wang, M. (2015). Improving CLOPE's Profit Value and Stability with an Optimized Agglomerative Approach. *Algorithms*, *8*(3), 380–394. https://doi.org/10.3390/a8030380

Li, Y., Le, J., Wang, M., Zhang, B., & Liu, L. (2015). MR-CLOPE: A MapReduce Based Transactional Clustering Algorithm for DNS Query Log Analysis. *Journal of Central South University*, *22*(9), 3485–3494. https://doi.org/10.1007/s11771-015-2888-9

Lim, A., & Tjhi, W. (2015). *R High Performance Programming*. Packt Publishing.

Lunn, A. D., & Davies, S. J. (1998). A Note on Generating Correlated Binary Variables. *Biometrika*, *85*(2), 487–490.

MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations. In L. M. Le Cam & J. Neyman (Hrsg.), *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* (S. 281–297). University of California Press.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., & Hornik, K. (2021). *cluster: Cluster Analysis Basics and Extensions* [R package version 2.1.2]. https://CRAN.R-project.org/package=cluster

Mahdi, M. A., Hosny, K. M., & Elhenawy, I. (2021). Scalable Clustering Algorithms for Big Data: A Review. *IEEE Access*, *9*, 80015–80027. https://doi.org/10.1109/access.2021.3084057

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Marbac, M., & Sedki, M. (2018). Varsellcm: An R/c++ Package for Variable Selection in Model-Based Clustering of Mixed-Data with Missing Values. *Bioinformatics*, *35*(7), 1255–1257. https://doi.org/10.1093/bioinformatics/bty786

Masulli, F., Petrosino, A., & Rovetta, S. (Hrsg.). (2015). *Clustering High–Dimensional Data*. Springer. https://doi.org/10.1007/978-3-662-48577-4

Matloff, N. (2016). *Parallel Computing for Data Science: with Examples in R, C++ and CUDA*. CRC Press.

McCallum, A., Nigam, K., & Ungar, L. H. (2000). Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '00*. https://doi.org/10.1145/347090.347123

McConnell, S. (2004). *Code Complete* (2. Aufl.). Microsoft Press.

McLachlan, G., & Peel, D. (2000). *Finite Mixture Models*. Wiley. https://doi.org/10.1002/0471721182

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., & Leisch, F. (2021). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien* [R package version 1.7-9]. https://CRAN.R-project.org/package=e1071

Microsoft & Weston, S. (2020). *foreach: Provides Foreach Looping Construct* [R package version 1.5.1]. https://CRAN.R-project.org/package=foreach

Mood, C. (2010). Logistic Regression: Why We Cannot Do What We Think We Can Do, and What We Can Do About It. *European Sociological Review*, *26*(1), 67–82. https://doi.org/10.1093/esr/jcp006

Mouselimis, L. (2021). *ClusterR: Gaussian Mixture Models, K-Means, Mini-Batch-Kmeans, K-Medoids and Affinity Propagation Clustering* [R package version 1.2.5]. https://CRAN.R-project.org/package=ClusterR

Murugesan, N., Cho, I., & Tortora, C. (2021). Benchmarking in Cluster Analysis: A Study on Spectral Clustering, DBSCAN, and K-Means. In *Studies in Classification, Data Analysis, and Knowledge Organization* (S. 175–185). Springer. https://doi.org/10.1007/978-3-030-60104-1_20

Nakano, J., & Nakama, E.-j. (2021). *RhpcBLASctl: Control the Number of Threads on „BLAS“* [R package version 0.21-247.1]. https://CRAN.R-project.org/package=RhpcBLASctl

Nasraoui, O., & N'Cir, C.-E. B. (Hrsg.). (2019). *Clustering Methods for Big Data Analytics: Techniques, Toolboxes and Applications*. Springer. https://doi.org/10.1007/978-3-319-97864-2

Nepusz, T., Yu, H., & Paccanaro, A. (2012). Detecting Overlapping Protein Complexes in Protein-Protein Interaction Networks. *Nature Methods*, *9*(5), 471–472. https://doi.org/10.1038/nmeth.1938

Ng, A. Y., Jordan, M. I., & Weiss, Y. (2001). On Spectral Clustering: Analysis and an Algorithm. In T. G. Dietterich, S. Becker & Z. Ghahramani (Hrsg.), *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*.

Ng, R. T., & Han, J. (2002). CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, *14*(5), 1003–1016. https://doi.org/10.1109/tkde.2002.1033770

Novomestky, F. (2021). *matrixcalc: Collection of Functions for Matrix Calculations* [R package version 1.0-5]. https://CRAN.R-project.org/package=matrixcalc

Oehlschlägel, J., & Ripley, B. (2020). *bit: Classes and Methods for Fast Memory-Efficient Boolean Selections* [R package version 4.0.4]. https://CRAN.R-project.org/package=bit

O'Leary, D., & Peleg, S. (1983). Digital Image Compression by Outer Product Expansion. *IEEE Transactions on Communications*, *31*(3), 441–444. https://doi.org/10.1109/tcom.1983.1095823

Ong, K.-L., Li, W., Ng, W.-K., & Lim, E.-P. (2004). SCLOPE: An Algorithm for Clustering Data Streams of Categorical Attributes. In Y. Kambayashi & M. M. W. Wöß (Hrsg.), *Data Warehousing and Knowledge Discovery* (S. 209–218). Springer. https://doi.org/10.1007/978-3-540-30076-2_21

Oprisa, C. (2015). A MinHash Approach for Clustering Large Collections of Binary Programs. *20th International Conference on Control Systems and Computer Science (CSCS)*. https://doi.org/10.1109/cscs.2015.27

Oprisa, C., Cabau, G., & Colesa, A. (2013). From Plagiarism to Malware Detection. *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. https://doi.org/10.1109/synasc.2013.37

Oyelade, J., Isewon, I., Oladipupo, F., Aromolaran, O., Uwoghiren, E., Ameh, F., Achas, M., & Adebiyi, E. (2016). Clustering Algorithms: Their Application to Gene Expression Data. *Bioinformatics and Biology Insights*, *10*, BBI.S38316. https://doi.org/10.4137/bbi.s38316

Park, C. G., Park, T., & Shin, D. W. (1996). A Simple Method for Generating Correlated Binary Variates. *The American Statistician*, *50*(4), 306–310. https://doi.org/10.2307/2684925

Peng, J., Choo, R. K.-K., & Ashman, H. (2016). Astroturfing Detection in Social Media: Using Binary n-Gram Analysis for Authorship Attribution. *2016 IEEE Trustcom/BigDataSE/ISPA*, 121–128. https://doi.org/10.1109/trustcom.2016.0054

Pratsinakis, E. D., Ntoanidou, S., Polidoros, A., Dordas, C., Madesis, P., Eleftherohorinos, I., & Menexes, G. (2020). Comparison of Hierarchical Clustering Methods for Binary Data from Molecular Markers. *International Journal of Data Analysis Techniques and Strategies*, *12*(3), 190–212. https://doi.org/10.1504/ijdats.2020.108036

Prentice, R. L. (1988). Correlated Binary Regression with Covariates Specific to Each Binary Observation. *Biometrics*, *44*(4), 1033–1048. https://doi.org/10.2307/2531733

Preud'homme, G., Duarte, K., Dalleau, K., Lacomblez, C., Bresso, E., Smaïl-Tabbone, M., Couceiro, M., Devignes, M.-D., Kobayashi, M., Huttin, O., Ferreira, J. P., Zannad, F., Rossignol, P., & Girerd, N. (2021). Head-To-Head Comparison of Clustering Methods for Heterogeneous Data: A Simulation-Driven Benchmark. *Scientific Reports*, *11*(4202). https://doi.org/10.1038/s41598-021-83340-8

Qaqish, B. F. (2003). A Family of Multivariate Binary Distributions for Simulating Correlated Binary Variables with Specified Marginal Means and Correlations. *Biometrika*, *90*(2), 455–463. https://doi.org/10.1093/biomet/90.2.455

R Core Team. (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. https://www.R-project.org/

Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, *66*(336), 846–850. https://doi.org/10.1080/01621459.1971.10482356

Rebonato, R., & Jäckel, P. (1999). The Most General Methodology to Create a Valid Correlation Matrix for Risk Management and Option Pricing Purposes. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1969689

Reddy, C. K., Hasan, M. A., & Zaki, M. J. (2014). Clustering Biological Data. In C. C. Aggarwal & C. K. Reddy (Hrsg.). CRC Press.

Ren, K. (2021). *rlist: A Toolbox for Non-Tabular Data Manipulation* [R package version 0.4.6.2]. https://CRAN.R-project.org/package=rlist

Revelle, W. (2021). *psych: Procedures for Psychological, Psychometric, and Personality Research* [R package version 2.1.9]. Northwestern University. Evanston, Illinois. https://CRAN.R-project.org/package=psych

Rodriguez, A., & Laio, A. (2014). Clustering by Fast Search and Find of Density Peaks. *Science*, *344*(6191), 1492–1496. https://doi.org/10.1126/science.1242072

Rogers, D. J., & Tanimoto, T. T. (1960). A Computer Program for Classifying Plants. *Science*, *132*(3434), 1115–1118. https://doi.org/10.1126/science.132.3434.1115

Rousseeuw, P. J. (1987). Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

Saldhi, A., Yadav, D., Saksena, D., Goel, A., Saldhi, A., & Indu, S. (2014). Big Data Analysis Using Hadoop Cluster. In N. Krishnan & M. Karthikeyan (Hrsg.), *2014 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE. https://doi.org/10.1109/iccic.2014.7238418

Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., Er, M. J., Ding, W., & Lin, C.-T. (2017). A Review of Clustering Techniques and Developments. *Neurocomputing*, *267*(6), 664–681. https://doi.org/10.1016/j.neucom.2017.06.053

Schnell, R., Bachteler, T., & Reiher, J. (2009). Privacy-Preserving Record Linkage Using Bloom Filters. *BMC Medical Informatics and Decision Making*, *9*(41). https://doi.org/10.1186/1472-6947-9-41

Schubert, E., & Rousseeuw, P. J. (2019). Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. In G. Amato, C. Gennaro, V. Oria & M. Radovanovic (Hrsg.), *Similarity Search and Applications* (S. 171–187). Springer International Publishing. https://doi.org/10.1007/978-3-030-32047-8_16

Sevcik, R., Rezankova, H., & Husek, D. (2011). Comparison of Selected Methods forDocument Clustering. In E. Mugellini, M. C. Pettenati, P. S. Szczepaniak & M. Sokhn (Hrsg.), *Proceedings of the 7th Atlantic Web Intelligence Conference* (S. 101–110). Springer.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, *27*(3), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Shi, J., & Malik, J. (2000). Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905. https://doi.org/10.1109/34.868688

Shirkhorshidi, A. S., Aghabozorgi, S., Wah, T. Y., & Herawan, T. (2014). Big Data Clustering: A Review. In B. Murgante, S. Misra, A. M. A. C. Rocha, C. Torre, J. G. Rocha, M. I. Falcão, D. Taniar, B. O. Apduhan & O. Gervasi (Hrsg.), *Lecture Notes in Computer Science* (S. 707–720). Springer. https://doi.org/10.1007/978-3-319-09156-3_49

Slonim, N., & Tishby, N. (1999). Agglomerative Information Bottleneck. *NIPS 99: Proceedings of the 12th International Conference on Neural Information Processing Systems*, 617–623.

Sokal, R. R., & Sneath, P. H. A. (1963). *Principles of Numerical Taxonomy*. Freeman.

Tamasauskas, D., Sakalauskas, V., & Kriksciuniene, D. (2012). Evaluation Framework of Hierarchical Clustering Methods for Binary Data. In A. Abraham, A. Zomaya, V. Wadhai, R. Yager, A. K. Muda & M. Koeppen (Hrsg.), *12th International Conference on Hybrid Intelligent Systems* (S. 421–426). IEEE. https://doi.org/10.1109/his.2012.6421371

Taraba, P. (2017). Powered Outer Probabilistic Clustering. *Proceedings of the World Congress on Engineering and Computer Science*, 394–398.

Taraba, P. (2022). POPC [Github Repository]. Verfügbar 20. Dezember 2022 unter https://github.com/ptsk78/POPC

Thalamuthu, A., Mukhopadhyay, I., Zheng, X., & Tseng, G. C. (2006). Evaluation and Comparison of Gene Clustering Methods in Microarray Analysis. *Bioinformatics*, *22*(19), 2405–2412. https://doi.org/10.1093/bioinformatics/btl406

Tishby, N., Pereira, F. C., & Bialek, W. (2000). *The Information Bottleneck Method*. arXiv: physics/0004057 [physics.data-an].

Torgerson, W. S. (1958). *Theory and Methods of Scaling*. Wiley.

Trejos-Zelaya, J., Amaya-Briceño, L. E., Jiménez-Romero, A., Murillo-Fernández, A., Piza-Volio, E., & Villalobos-Arias, M. (2021). Clustering Binary Data by Application of Combinatorial Optimization Heuristics. In T. Chadjipadelis, B. Lausen, A. Markos, T. R. Lee, A. Montanari & R. Nugent (Hrsg.), *Studies in Classification, Data Analysis, and Knowledge Organization* (S. 301–309). Springer. https://doi.org/10.1007/978-3-030-60104-1_33

Tseng, G. C., & Wong, W. H. (2005). Tight Clustering: A Resampling-Based Approach for Identifying Stable and Tight Patterns in Data. *Biometrics*, *61*(1), 10–16. https://doi.org/10.1111/j.0006-341x.2005.031032.x

Tummala, K., Oswald, C., & Sivaselvan, B. (2018). A Frequent and Rare Itemset Mining Approach to Transaction Clustering. In A. Mobasheri, J. Li, J. J. Nakahara & A. Mobasheri (Hrsg.), *Communications in Computer and Information Science* (S. 8–18). Springer. https://doi.org/10.1007/978-981-10-8603-8_2

Van Mechelen, I., & Vach, W. (2019). Cluster Analyses of a Target Data Set from the IFCS Cluster Benchmark Data Repository: Introduction to the Special Issue. *Archives of Data Science*, *1*(1), 1–9. https://doi.org/10.5445/KSP/1000085952/01

van Dongen, S. (2000). *A Cluster Algorithm for Graphs* (Technischer Bericht Nr. R0010). Centrum voor Wiskunde en Informatica.

Vermunt, J. K., & Magidson, J. (2002). Latent Class Cluster Analysis. In *Applied Latent Class Analysis* (S. 89–106). Cambridge University Press. https://doi.org/10.1017/cbo9780511499531.004

Wang, A., & Sabo, R. T. (2015). Simulating Clustered and Dependent Binary Variables. *Austin Biometrics and Biostatistics*, *2*(2), 1020.

Wang, K., Xu, C., & Liu, B. (1999). Clustering Transactions Using Large Items. *Proceedings of the Eighth International Conference on Information and Knowledge Management - CIKM '99*, 483–490. https://doi.org/10.1145/319950.320054

Ward, J. H. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, *58*(301), 236–244.

Weston, S., & Wickham, H. (2014). *itertools: Iterator Tools* [R package version 0.1-3]. https://CRAN.R-project.org/package=itertools

Wickham, H. (2007). Reshaping Data with the reshape Package. *Journal of Statistical Software*, *21*(12), 1–20. http://www.jstatsoft.org/v21/i12/

Wickham, H. (2019). *stringr: Simple, Consistent Wrappers for Common String Operations* [R package version 1.4.0]. https://CRAN.R-project.org/package=stringr

Wickham, H., François, R., Henry, L., & Müller, K. (2021). *dplyr: A Grammar of Data Manipulation* [R package version 1.0.7]. https://CRAN.R-project.org/package=dplyr

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann.

Wittkop, T., Emig, D., Lange, S., Rahmann, S., Albrecht, M., Morris, J. H., Böcker, S., Stoye, J., & Baumbach, J. (2010). Partitioning Biological Data with Transitivity Clustering. *Nature Methods*, *7*(6), 419–420. https://doi.org/10.1038/nmeth0610-419

Wiwie, C., Baumbach, J., & Röttger, R. (2015). Comparing the Performance of Biomedical Clustering Methods. *Nature Methods*, *12*(11), 1033–1038. https://doi.org/10.1038/nmeth.3583

Wood, J. (2021). *RcppAlgos: High Performance Tools for Combinatorics and Computational Mathematics* [R package version 2.4.3]. https://CRAN.R-project.org/package=RcppAlgos

Xu, D., & Tian, Y. (2015). A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, *2*(2), 165–193. https://doi.org/10.1007/s40745-015-0040-1

Xu, R., & Wunsch, D. (2009). *Clustering*. John Wiley & Sons.

Yan, H., Chen, K., Liu, L., & Yi, Z. (2009). SCALE: A Scalable Framework for Efficiently Clustering Transactional Data. *Data Mining and Knowledge Discovery*, *20*(1), 1–27. https://doi.org/10.1007/s10618-009-0134-5

Yang, Y., Guan, X., & You, J. (2002). CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '02*, 682–687. https://doi.org/10.1145/775047.775149

Yap, P. H., & Ong, K.-L. (2005). σ-SCLOPE: Clustering Categorical Streams Using Attribute Selection. In R. Khosla, R. J. Howlett & L. C. Jain (Hrsg.), *Knowledge-Based Intelligent Information and Engineering Systems* (S. 929–935). Springer. https://doi.org/10.1007/11552451_128

Yeung, K. Y., Fraley, C., Murua, A., Raftery, A. E., & Ruzzo, W. L. (2001). Model-Based Clustering and Data Transformations for Gene Expression Data. *Bioinformatics*, *17*(10), 977–987. https://doi.org/10.1093/bioinformatics/17.10.977

Yue, L., Chen, W., Li, X., Zuo, W., & Yin, M. (2018). A Survey of Sentiment Analysis in Social Media. *Knowledge and Information Systems*, *60*(2), 617–663. https://doi.org/10.1007/s10115-018-1236-4

Yuping, W., Huiyun, W., Liang, J., Jiaguo, O., & Huang, Y. (2016). A Clustering with Slope Algorithm based on MapReduce. *Journal of Digital Information Management*, *14*(3), 168–176.

Zaki, M. J., Peters, M., Assent, I., & Seidl, T. (2007). Clicks: An Effective Algorithm for Mining Subspace Clusters in Categorical Datasets. *Data & Knowledge Engineering*, *60*(1), 51–70. https://doi.org/10.1016/j.datak.2006.01.005

Zgurovsky, M. Z., & Zaychenko, Y. P. (2019). The Cluster Analysis in Big Data Mining. In *Big Data: Conceptual Analysis and Applications* (S. 1–42). Springer. https://doi.org/10.1007/978-3-030-14298-8_1

Zgurovsky, M. Z., & Zaychenko, Y. P. (2020). *Big Data: Conceptual Analysis and Applications*. Springer. https://doi.org/10.1007/978-3-030-14298-8

Zhang, K., Lo, D., Lim, E.-P., & Prasetyo, P. K. (2013). Mining Indirect Antagonistic Communities from Social Interactions. *Knowledge and Information Systems*, *35*(3), 553–583. https://doi.org/10.1007/s10115-012-0519-4

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Record*, *25*(2), 103–114. https://doi.org/10.1145/235968.233324

Zubin, J. (1938). A Technique for Measuring Like-Mindedness. *The Journal of Abnormal and Social Psychology*, *33*(4), 508–516. https://doi.org/10.1037/h0055441

DOI: 10.17185/duepublico/81778
URN: urn:nbn:de:hbz:465-20240328-135052-6