Eine vergleichende Untersuchung von exaktem, durch Schranken beschleunigtem k-means-Clustering

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science

vorgelegt von:
Tim Wolfgang Düsterhus

Thema gestellt von:
Prof. Dr. Christian Beecks

Zweitprüfer:
Prof. Dr. Xiaoyi Jiang

Münster, 9. Januar 2020

Abstract

Diese Arbeit analysiert die Leistung von sechs durch Schranken beschleunigten Algorithmen zum exakten k-means-Clustering. Im ersten Teil wird durch eine systematische Zerlegung der Algorithmen untersucht, welche Einzelkomponenten genutzt werden, um Rechenaufwand einzusparen. Es lässt sich feststellen, dass sich die Einzelkomponenten konzeptionell in mehreren Algorithmen wiederfinden. Insgesamt ergeben sich bei der Zerlegung der sechs Algorithmen drei unterschiedliche Arten von unteren Schranken zwischen Datenpunkten und Clusterzentren. Es werden zwei zusätzliche Informationen über die Positionen der Clusterzentren erfasst. Zuletzt lassen sich zwei weitere Techniken zur effizienteren Datenhaltung finden, die auf jeden untersuchten Algorithmus anwendbar sind. Anschließend wird diskutiert, wie sich die Algorithmen aus den vorgestellten Einzelkomponenten zusammensetzen.

Im zweiten Teil dieser Arbeit wurden die Algorithmen gemäß der Beschreibung in ihren Originalveröffentlichungen in C++ implementiert. Soweit dies sinnvoll möglich war, wurden Varianten der Algorithmen erstellt, bei denen zusätzliche Komponenten integriert oder deaktiviert wurden. Auch im Falle mehrerer plausibler Interpretationen der Beschreibung wurden unterschiedliche Varianten des Algorithmus implementiert. Insgesamt ergeben sich 39 Varianten, deren Leistung bei jeweils identischer Startsituation empirisch mit unterschiedlichen Datensätzen überprüft wurde. Es stellt sich heraus, dass geringe Modifikationen einen großen Einfluss auf die Leistung haben. Auch bei identischen Einzelkomponenten können abhängig von der konkreten Zusammensetzung des Algorithmus große Leistungsunterschiede bestehen. Insbesondere kann es mitunter sinnvoll sein, Einzelkomponenten zu deaktivieren, da der Overhead in der konkreten Kombination höher als die Zeitersparnis ist.

Inhaltsverzeichnis

1

Einleitung

Menschen beschäftigen sich schon seit Jahrhunderten mit der Analyse von Daten [Ges15]. Zu Beginn geschah dies manuell, mit Stift und Papier. Mit der Erfindung der Rechenmaschinen und Computer konnte diese Arbeit erleichtert werden. Heutzutage leben wir im Zeitalter von „Big Data“. Sekündlich werden fast 30 TB Daten neu geschaffen [Mar18] und vollautomatisch verarbeitet, gefiltert, sortiert, analysiert und klassifiziert. Alleine der weltgrößte Internet-Exchange DE-CIX in Frankfurt am Main leitet im Jahresmittel 4,1 Terabit Daten pro Sekunde durch [DEC].

Unternehmen haben ein Interesse daran, dass diese Datenverarbeitung möglichst schnell und effizient abläuft. Schnell ist hier sowohl in Bezug auf die Realzeit, als auch in Bezug auf die CPU-Zeit zu verstehen. Die realzeitliche Komponente ist leicht zu erkennen: Menschen warten nicht gerne. Der Computer soll das gewünschte Ergebnis im Idealfall „sofort“ liefern. Ein anderer Anwendungsfall ist Hochfrequenzhandel. Dort entscheiden Millisekunden über die Höhe der Gewinne beziehungsweise der Verluste [Sim17]. Aber auch die CPU-Zeit darf nicht außer Acht gelassen werden. Eine hochparallele Implementierung kann sehr kurze Reaktionszeiten liefern. Um diese Leistung in der Praxis jedoch zu erreichen, würde potentiell Hardware im Wert von hunderttausenden Euro benötigt. Neben den Anschaffungskosten für die Hardware wird für jede Berechnung selbstverständlich auch elektrischer Strom und eine angemessene Kühlung der einzelnen Komponenten benötigt. Im Rahmen der aktuellen Diskussionen zur „Klimakrise“ besteht für Unternehmen ein Anreiz, sich durch effizientere Software und dem damit verbundenen geringeren Ressourcenverbrauch als besonders ökologisch handelnd auf dem Markt zu positionieren. Betreiber von Rechenzentren, Webhostingpaketen und Mietservern werben beispielsweise damit, dass sie ausschließlich Ökostrom beziehen.

Es besteht somit ein Bedarf an der Entwicklung effizienter Algorithmen, die auch für große und weiter steigende Datenmengen schnell und ressourcenschonend arbeiten und die gewünschten Ergebnisse liefern.

In dieser Arbeit soll es um die Untersuchung von Methoden zum Clustering von Daten, konkret um Algorithmen der k-means-Familie gehen. Beim Clustering wird als Eingabe ein Datenbestand ohne nähere Informationen zum Inhalt übergeben. Der Algorithmus versucht anschließend selbstständig („unsupervised“) Muster in diesen Daten zu finden und diese in Gruppen („Cluster“) aufzuteilen. Daten innerhalb eines resultierenden Clusters sind sich „ähnlich“, Daten unterschiedlicher Cluster „unähnlich“. Die konkrete Ausgestaltung der „Ähnlichkeit“ ist abhängig vom eingesetzten Clustering-Algorithmus und der Struktur des übergebenen Datenbestands.

Zunächst soll in Kapitel 2 das k-means-Problem im Detail vorgestellt werden. Es wird diskutiert, wie Clustering mit einem k-means-Algorithmus funktioniert, wie die resultierenden Cluster beschaffen sind und welche grundsätzlichen Vor- und Nachteile beziehungsweise welche Einschränkungen die Wahl eines k-means-Algorithmus mit sich bringt. Kapitel 3 zerlegt die in dieser Arbeit untersuchten Algorithmen zum beschleunigten, exakten, k-means-Clustering zunächst systematisch in ihre Einzelbestandteile und untersucht anschließend, wie diese zum fertigen Algorithmus zusammengefügt werden. Nachdem die Funktionsweise aus der theoretischen Perspektive hinreichend aufgeschlüsselt ist, soll es in Kapitel 4 um die Testdatensätze gehen, die genutzt werden, um die theoretischen Leistungsmerkmale auf ihre Praxistauglichkeit zu untersuchen. Dazu wird in Kapitel 5 zunächst vorgestellt, welche Designentscheidungen in der konkreten Implementierung getroffen wurden und anschließend werden die resultierenden Messdaten des Clusterings der Datensätze bei verschiedenen Algorithmen und Algorithmenkonfigurationen in Kapitel 6 vergleichend untersucht.

2

Das k-means-Problem

Die wohl am häufigsten verwendete und in dieser Arbeit betrachtete Familie von Algorithmen zum Clustern von Daten ist die k-means-Familie [Wu+08]. Gegeben

1. einer gewünschten Anzahl von Clustern k und

2. einer Menge von Datenpunkten (P)

ist es das Ziel des k-means-Problems k Clusterzentren (C) zu finden, sodass die Summe (J) der quadratischen euklidischen Distanzen zwischen Clusterzentren und ihren jeweils zugeordneten Datenpunkten ($A(c)$) minimiert wird:

$$J = \sum_{c \in C} \sum_{p \in A(c)} d(p, c)^2 \tag{2.1}$$

$$d(p, c) = \|p - c\|_2$$

Aus diesen Anforderungen ergibt sich, dass die Datenpunkte ihrem jeweils nächstgelegenen Clusterzentrum zugeordnet werden und die Position der Clusterzentren dem geometrischen Schwerpunkt aller Datenpunkte eines Clusters, also dem namensgebenden komponentenweisen arithmetischen Mittel (englisch „mean“), entspricht.

Als Resultat entsteht eine Voronoi-Partitionierung des Raumes. Die entstehenden Cluster sind konvexe Polygone, die durch ihr Clusterzentrum repräsentiert werden.

Das Finden der exakten Lösung des k-means-Problems, also das Erreichen des globalen Minimums, ist NP-schwer [MNV12]. Das Finden eines lokalen Minimums (in seltenen Fällen eines Sattelpunktes) ist allerdings effizient möglich.

Variablen

Zunächst möchten wir die in dieser Arbeit verwendeten Variablen und Bezeichner definieren:

P	= Die Menge aller Datenpunkte.
N	= $\|P\|$, die Anzahl der Datenpunkte.
p	= Ein Datenpunkt.
C	= Die Menge aller Clusterzentren.
k	= $\|C\|$, die Anzahl der gewünschten Cluster.
c	= Ein Clusterzentrum.
J	= Der Wert der Zielfunktion.
d	= Eine Distanzfunktion (die euklidische Distanz).
$A(c)$	= Die Menge der dem Clusterzentrum c zugeordneten Datenpunkte.
$a(p)$	= Das einem Datenpunkt p zugewiesene Clusterzentrum.
$u(\dots)$	= Eine obere Schranke. Parameter und Bedeutung sind abhängig vom Algorithmus.
$l(\dots)$	= Eine untere Schranke. Parameter und Bedeutung sind abhängig vom Algorithmus.
$cc(c_1, c_2)$	= Der Abstand zwischen den Clusterzentren c_1 und c_2.
$cc_g(c)$	= Der minimale Abstand $cc(c, c')$ für $c' \neq c$.

2.1. Der Lloyd-Algorithmus

Eine der ersten und die wohl populärste Implementierung von k-means ist der Algorithmus von Lloyd [Elk03; Ham10; Dra12]. Dieser wird daher typischerweise als *der* k-means-Algorithmus bezeichnet. Nach der Initialisierung, bei der k Punkte als initiale Clusterzentren ausgewählt werden, werden zwei Phasen, die jeweils die Zielfunktion verringern, abwechselnd wiederholt, bis der Algorithmus konvergiert. Die Auswahl dieser initialen Clusterzentren wird in den Abschnitten 2.1.5 und 3.1 genauer betrachtet.

Insgesamt ergibt sich die in Algorithmus 2.1 dargestellte Struktur für den Lloyd-Algorithmus.

Algorithmus 2.1 Hauptschleife des Lloyd-Algorithmus.

```
1: selectInitialCenters()
2: J_new ← ∞
3: repeat
4:     J ← J_new
5:     assignPointsToCluster()
6:     moveCenters()
7:     J_new ← getBadness()
8: until J_new ≥ J
```

Algorithmus 2.2 Berechnung der Zielfunktion des Lloyd-Algorithmus.

```
1: function getBadness
2:     J ← 0
3:     for all c ∈ C do
4:         for all p ∈ A(c) do
5:             J ← J + d(p, c)^2
6:         end for
7:     end for
8:     return J
9: end function
```

Algorithmus 2.3 Zuordnung der Datenpunkte beim Lloyd-Algorithmus.

```
1: procedure assignPointsToCluster
2:     for all p ∈ P do
3:         nearest ← ⊥
4:         nearest_dist ← ∞
5:         for all c ∈ C do
6:             dist ← d(p, c)
7:             if dist < nearest_dist then
8:                 nearest ← c
9:                 nearest_dist ← dist
10:            end if
11:        end for
12:        assignPointToCluster(p, nearest)
13:    end for
14: end procedure
```

2.1.1. Zuordnung der Datenpunkte

In diesem Schritt werden für alle Datenpunkte die jeweils nächstgelegenen Clusterzentren ermittelt. Dazu ist es notwendig, für jeden Datenpunkt die Distanz zu jedem Clusterzentrum zu berechnen. Anschließend wird der Datenpunkt dem Zentrum mit der geringsten Distanz zugeordnet.

Dadurch, dass über alle Datenpunkte und Clusterzentren iteriert wird (Algorithmus 2.3), entsteht eine Laufzeitkomplexität von $\Theta(k \cdot N)$ mit genau so vielen Distanzberechnungen.

2.1.2. Verschiebung der Clusterzentren

Nachdem die Neuzuordnung der Datenpunkte abgeschlossen ist, wird die Position der Clusterzentren korrigiert. Um die Summe der quadratischen euklidischen Distanzen innerhalb eines Clusters und somit auch die globale Fehlersumme zu minimieren, muss die Position des Clusterzentrums in den geometrischen Schwerpunkt (englisch „Centroid“) des Clusters verschoben werden [Gz].

Falls der Algorithmus noch nicht konvergiert ist, führt diese Verschiebung dazu, dass Datenpunkte sich nun näher an einem anderen als ihrem bisher zugeordneten Clusterzentrum befinden. Die erste Phase, die Zuordnung der Datenpunkte, muss daher erneut durchgeführt werden.

2.1.3. Konvergenz

Da die Anzahl der möglichen Zustände durch die Anzahl der Cluster (k) und die Anzahl der Datenpunkte (N) auf die Stirling-Zahl zweiter Art $S_{N,k}$[1] beschränkt ist, konvergiert der Lloyd-Algorithmus in endlicher Zeit.

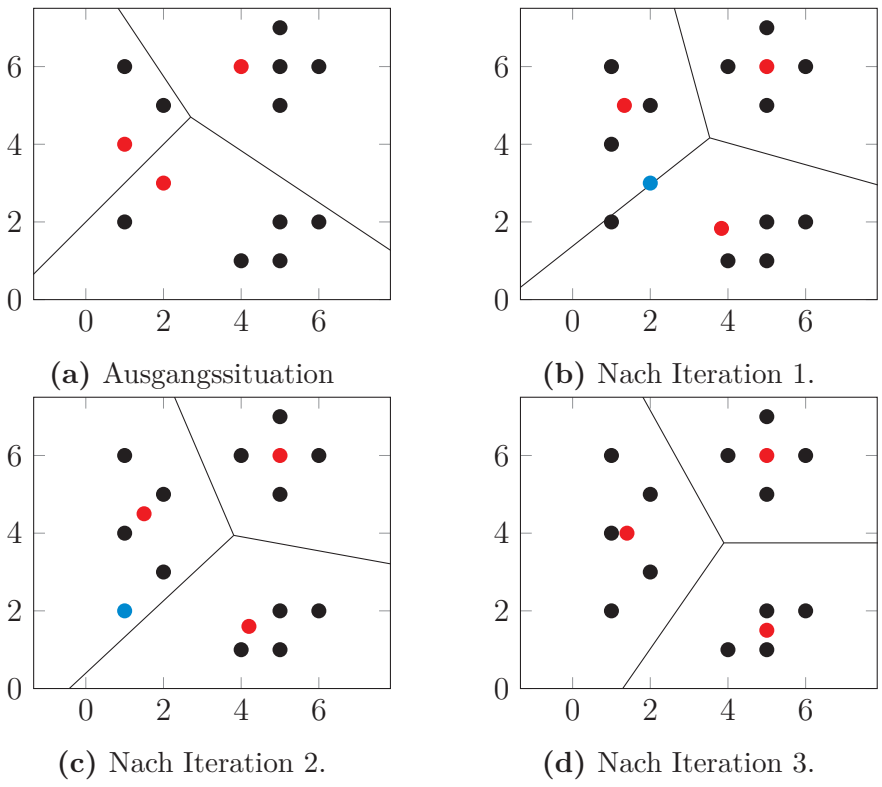
Darüber hinaus ist leicht zu sehen, dass beide Phasen jeweils monoton die Zielfunktion verringern: Die Zuordnung eines Punktes zum nächstgelegenen Zentrum minimiert die (quadratische) Distanz für den Punkt. Die Verschiebung des Clusterzentrums in den Schwerpunkt minimiert die Summe der quadratischen Distanzen innerhalb des Clusters. Die Zielfunktion ist nicht-negativ, die monotone Verringerung dieser führt daher ebenfalls zwangsläufig zu einer Konvergenz in endlicher Zeit.

[1] Die Stirling-Zahl zweiter Art gibt an, auf wie viele Arten eine n-elementige Menge in k nicht-leere, disjunkte Teilmengen aufgeteilt werden kann: $S_{n,k} = \frac{1}{k!} \sum_{j=1}^{k} (-1)^{k-j} \binom{k}{j} j^n$.

2.1.4. Beispiel

In Abbildung 2.1 findet sich ein beispielhafter Verlauf von k-means-Clustering mit Hilfe des Lloyd-Algorithmus für einen simplen Datensatz mit 14 Datenpunkten. Diese sind in drei natürliche, konvexe und etwa gleich große Cluster aufgeteilt und die Anzahl der gewünschten Clusterzentren ist als $k = 3$ gewählt. Es ist daher zu erwarten, dass die finalen Positionen der Clusterzentren genau in den natürlichen Clustern liegen.

Dies ist nach drei Iterationen der Fall. In Iteration 3 hat keine Neuzuordnung stattgefunden (es gibt keinen blau markierten Datenpunkt), daher terminiert der Algorithmus.



(a) Ausgangssituation

(b) Nach Iteration 1.

(c) Nach Iteration 2.

(d) Nach Iteration 3.

Abbildung 2.1.: Die aktuellen Clusterzentren sind rot markiert. Zu Beginn liegen sie genau auf drei Datenpunkten. Die blau markierten Datenpunkte haben nach der Verschiebung der Clusterzentren in der jeweiligen Iteration ihre Zuordnung geändert.

2.1.5. Initialisierung

Die Auswahl der initialen Clusterzentren hat einen entscheidenden Einfluss auf die Qualität des resultierenden Clusterings und auf die Anzahl der benötigten Iterationen bis zur Konvergenz.

Abbildung 2.2 zeigt beispielhaft den Einfluss der Initialisierung auf die Qualität des Ergebnisses. Die Datenpunkte liegen in diesem Beispiel auf den Eckpunkten eines Rechtecks. Abhängig von der Initialisierung liegen die resultierenden Clusterzentren entweder auf den Mittelpunkten der langen Seiten (lokales Minimum) oder auf den Mittelpunkten der kurzen Seiten (globales Minimum).

Eine Initialisierung, die als initiale Clusterzentren Punkte auswählt, die nah an der konvergierten Position des Zentrums liegen, reduziert die Anzahl der benötigten Iterationen. Eine Veranschaulichung findet sich in Abbildung 2.3. In 2.3a wurden beide initialen Clusterzentren im gleichen natürlichen Cluster platziert. Dies führt dazu, dass die Grenze der beiden Voronoi-Zellen den linken Cluster teilt. Diese fehlerhafte Zuordnung kann erst in der dritten Iteration vollständig korrigiert werden.

Beide Zielsetzungen korrelieren miteinander. Eine Auswahl von initialen Clusterzentren, die die Wahrscheinlichkeit eines „schlechten“ lokalen Minimums verringert, führt dazu, dass sich die Zentren bereits im Bereich eines möglichen Zielclusters und damit in der Nähe ihrer endgültigen Position befinden.

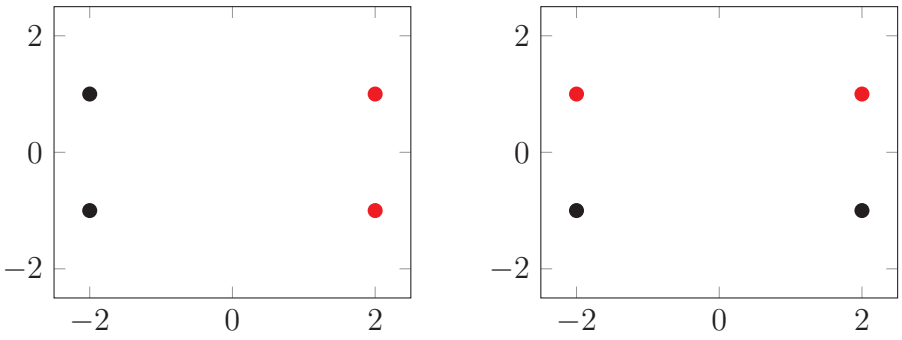
Das in dieser Arbeit genutzte Initialisierungsverfahren k-means++ wird in Abschnitt 3.1 näher vorgestellt.

2.1.6. Einschränkungen und Probleme

Die Einfachheit des k-means-Problems und des Lloyd-Algorithmus bringt einige Einschränkungen und Probleme mit sich. Einige Einschränkungen sind inherent mit dem k-means-Problem verknüpft (etwa „arithmetisches Mittel notwendig“), andere sind in der Konstruktion des Lloyd-Algorithmus zu verorten (etwa „Leere Cluster können entstehen“).

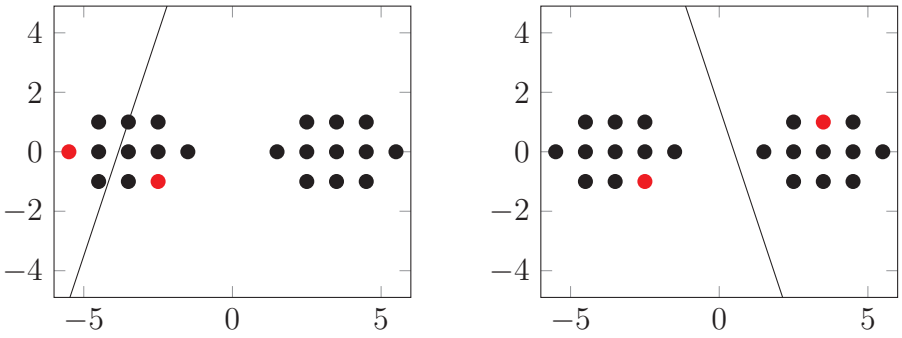
Arithmetisches Mittel notwendig

Das k-means-Problem setzt die Existenz des namensgebenden arithmetischen Mittels beziehungsweise der euklidischen Distanz voraus. Dadurch sind die



(a) Konvergiert zu $(0, 1)$ und $(0, -1)$. **(b)** Konvergiert zu $(-2, 0)$ und $(2, 0)$.

Abbildung 2.2.: Abhängig von der Initialisierung erreicht die Zielfunktion bei diesen Datenpunkten entweder $4 \cdot 2^2 = 16$ oder $4 \cdot 1^2 = 4$. Die „rot“en Punkte stellen die als initiale Clusterzentren gewählten Punkte dar.



(a) Benötigt drei Iterationen zur Konvergenz.

(b) Benötigt eine Iteration zur Konvergenz.

Abbildung 2.3.: In beiden Fällen konvergiert der Algorithmus zum gleichen Ergebnis $(-3.5, 0)$ und $(3.5, 0)$. Im ersten Fall werden aber zwei zusätzliche Iterationen benötigt, da der Großteil der Punkte des linken Clusters zunächst dem rechten Cluster zugeordnet wird. Die Grenze der Voronoi-Zerlegung ist mit einer Linie gekennzeichnet.

Datenpunkte auf numerische Werte beschränkt, ein Clustering von nicht-numerischen Daten ist nicht möglich.

Clustermodell

Bedingt dadurch, dass das k-means-Problem die Cluster durch einen Stellvertreter, das Clusterzentrum, beschreibt und die Clusterzugehörigkeit von der Distanz zu diesem Stellvertreter statt der Distanz zum nächstgelegenen Datenpunkt abhängig macht, ist die Lösung des k-means-Problems stets eine Menge von konvexen Clustern (Abbildung 2.4a).

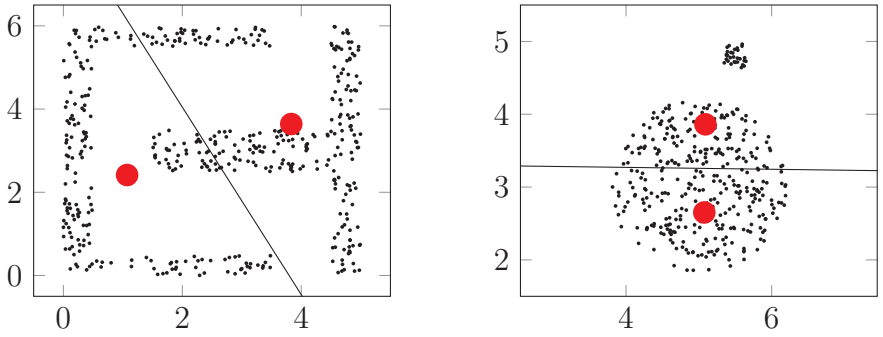
Darüber hinaus führt dies dazu, dass bei starken Größenunterschieden geometrisch benachbarter Cluster die Randpunkte des größeren Clusters aufgrund der hohen Distanz zum Clusterzentrum des größeren Clusters dem kleineren, aber dennoch deutlich separierten Cluster zugeordnet werden (Abbildung 2.4b).

Ausreißer

Ein weiterer Effekt des verwendeten Clustermodells ist die fehlende Behandlung von Ausreißern. Jeder Punkt wird einem Cluster zugeordnet und nimmt an der Berechnung des Clusterzentrums teil. In Abbildung 2.5 ist beispielhaft dargestellt, wie Ausreißer das Ergebnis verändern können.

k muss bekannt sein

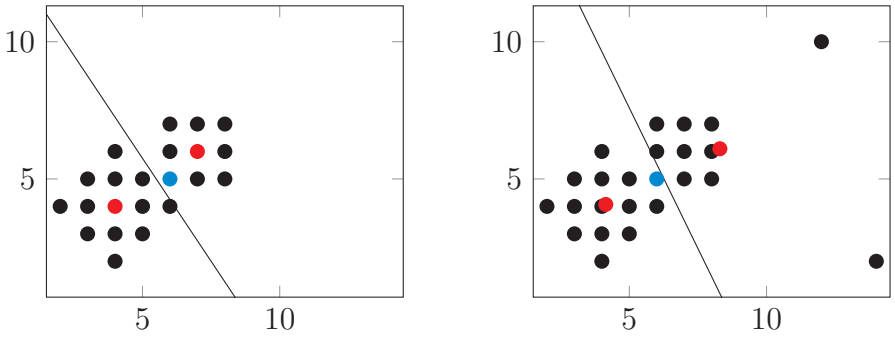
Eine weitere Einschränkung, die sich aus der fehlenden Berücksichtigung der intra-Cluster-Distanzen ergibt, ist, dass die Anzahl der gewünschten Cluster k explizit als Eingabewert übergeben werden muss. Eine Lösung für diese Problematik ist es, den Algorithmus iterativ mit steigendem k durchzuführen, bis die Verbesserung der Zielfunktion J abflacht (Abbildung 2.6). An diesem Punkt ist es sehr wahrscheinlich, dass ein großer natürlicher Cluster in zwei kleinere Cluster aufgeteilt wurde und keine weiteren natürlichen Cluster mehr gefunden werden. Alternativ kann für jedes resultierende Clustering der Silhouettenkoeffizient berechnet werden. Dieser ist in der Berechnung deutlich aufwändiger. Distanzen müssen von jeden Datenpunkt zu jedem anderen Datenpunkt berechnet werden. Für weniger klar separierte natürliche Cluster ist dieser aber genauer als die Betrachtung der k-means-Zielfunktion.



(a) Für die beiden Cluster lässt sich keine Trenngerade finden.

(b) Das Zentrum des kleineren Clusters ist weniger als der Durchmesser des größeren Clusters von dessen Randpunkten entfernt.

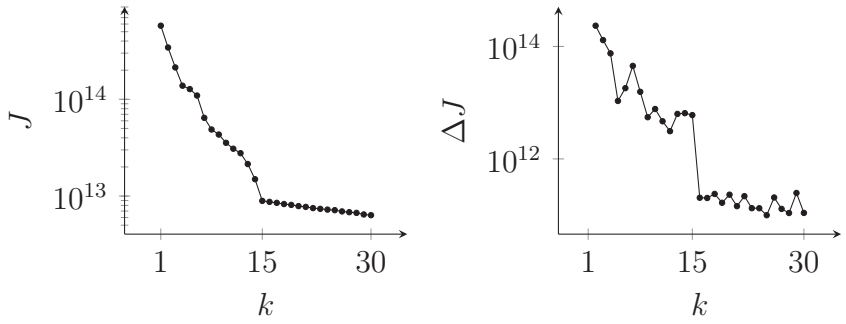
Abbildung 2.4.: In beiden Fällen sind die Cluster optisch klar voneinander separiert. Die Lösung des k-means-Problems sind aber nicht die natürlichen Cluster.



(a) Ohne Ausreißer.

(b) Mit Ausreißern.

Abbildung 2.5.: Die Ausreißer führen dazu, dass das rechte Clusterzentrum sich außerhalb des Clusters befindet. Der „blau“ markierte Punkt an der Clustergrenze wird daher falsch zugeordnet.



(a) Fehler in Abhängigkeit der Clusteranzahl k.

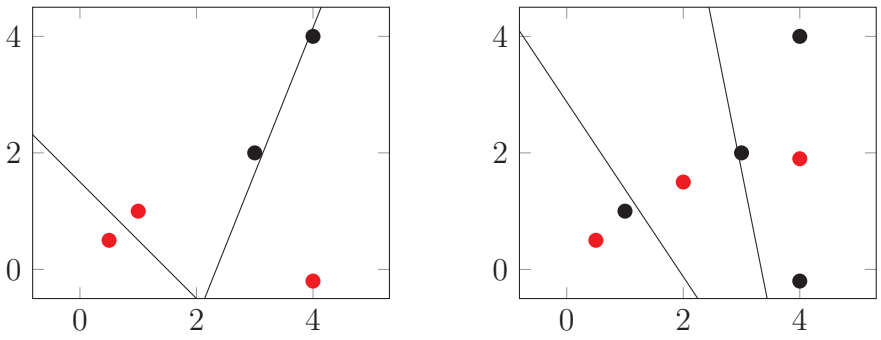
(b) Verbesserung des Fehlers im Vergleich zum vorherigen k.

Abbildung 2.6.: Fehler beim Clustering des s1-Datensatzes (Seite 56) für steigende k. Ein Abflachen bei $k = 15$ ist deutlich zu erkennen und spiegelt sich auch in der Verbesserung des Fehlers wieder.

Leere Cluster können entstehen

Als Folge des vorzugebenden k ist es bei einer schlechten Wahl und ungünstiger Initialisierung möglich, dass ein Clusterzentrum im Laufe des Algorithmus für keinen Punkt das nächstgelegene Zentrum ist und daher einen leeren Cluster repräsentiert. Bei der Verschiebung in den Schwerpunkt dieses leeren Clusters kommt es zu einer Division durch 0. Als Lösung könnten betroffene Clusterzentren auf einen zufällig gewählten Datenpunkt gesetzt werden. Dies reduziert in jedem Falle die Zielfunktion, da die quadratische Distanz von diesem Datenpunkt zu seinem nächstgelegenen Zentrum den Wert 0 annimmt.

Ein Beispiel findet sich in Abbildung 2.7. Die Datenpunkte bilden entweder zwei oder vier natürliche Cluster, abhängig davon, ob die drei rechten Punkte



(a) Vor der 1. Iteration.

(b) Nach der 1. Iteration.

Abbildung 2.7.: Nach der 1. Iteration wird dem mittleren Clusterzentrum kein Punkt mehr zugeordnet.

jeweils einen Cluster bilden oder nicht. Es wurde aber $k = 3$ gewählt und zwei der initialen Clusterzentren befinden sich im linken Cluster. Dies führt nach der ersten Iteration dazu, dass ein Clusterzentrum genau zwischen allen Punkten liegt und daher von jedem Punkt zu weit entfernt ist.

2.2. Die Optimierung

Der unveränderte Lloyd-Algorithmus führt in der in Abschnitt 2.1.1 vorgestellten Zuordnungsphase viele redundante Distanzberechnungen durch. Ein Datenpunkt, der eine geringe Entfernung zu seinem derzeit zugeordneten Clusterzentrum besitzt, wird in der nächsten Zuordnungsphase in der Regel *nicht* neu zugeordnet, sondern ist weiterhin näher an seinem derzeit zugeordneten Zentrum, verglichen mit allen anderen Zentren.

Wenn sich Clusterzentren mit geringem Zusatzaufwand ausschließen („prunen“) lassen, können, potentiell teure, Distanzberechnungen eingespart werden.

Definition 1 (Pruning)
Mit „Pruning“ wird die Möglichkeit bezeichnet, ohne exakte Distanzberechnungen zu bestimmen, ob ein Clusterzentrums potentiell das einem Punkt nächstgelegene ist.

Allen in dieser Arbeit untersuchten Algorithmen ist gemein, dass sie schrankenbasiert prunen und exakt sind. Wir möchten diese beiden Eigenschaften definieren.

Definition 2 (Schrankenbasierter Algorithmus)
Ein schrankenbasierter Algorithmus pruned Clusterzentren auf Basis

- einer oberen Schranke der Entfernung zum nächstgelegenen Clusterzentrum
- einer Menge von unteren Schranken, die Aussagen über die Entfernung zu anderen Clusterzentren treffen
- und gegebenenfalls zusätzlichen Informationen, die in Verbindung mit den Schranken ein Pruningkriterium herleiten.

Schrankenbasierte Algorithmen bieten den Vorteil, dass diese für die Speicherung der zusätzlichen Informationen simple Datenstrukturen nutzen kön-

nen. Eine obere Schranke pro Datenpunkt könnte effizient durch ein lineares Feld[2] von Distanzwerten repräsentiert werden. Darüber sind schrankenbasierte Indexstrukturen weniger stark als Bäume vom „Fluch der Dimensionalität“ betroffen [Elk03, Abschnitt 6; Ham10, Abschnitt 2.2].

Eine nicht-schrankenbasierte exakte k-means-Variante auf Basis von kd-Bäumen wurde in [Kan+02] vorgestellt.

Definition 3 (Exakter Algorithmus)
Ein exakter Algorithmus liefert nach jeder Iteration die gleichen Positionen der Clusterzentren wie der Lloyd-Algorithmus.

Exakte Algorithmen bieten den Vorteil, dass diese die erprobten Eigenschaften des Lloyd-Algorithmus beibehalten. Anwendungen, die Wert auf die bewährte Qualität der Ergebnisse des Lloyd-Algorithmus legen, können auf diese Weise beschleunigt werden, ohne Gefahr zu laufen, dass die Ergebnisse des Clusterings fehlerhaft oder von unzureichender Genauigkeit sind. Es muss keine langwierige Prüfung des Algorithmus auf Praxistauglichkeit für den gewünschten Einsatzzweck durchgeführt werden.

Im Gegensatz zu exakten Algorithmen stehen beispielsweise das probabilistische Mini-batch k-means, das bei jeder Iteration nur einen kleinen Teil der Datenpunkte auswählt [Scu10], und Recursive Partition Based K-Means, das die Datenpunkte in immer kleinere Partitionen aufteilt und die Ergebnisse der vorherigen Iteration als Startwert für die nächste Iteration verwendet [CPL17].

2.3. Abgrenzung zu anderen Clustering-Verfahren

Neben den beschleunigten k-means-Varianten im vorherigen Abschnitt möchten wir an dieser Stelle noch einige andere Algorithmen zum Clustering von Datensätzen nennen und die Unterschiede zu k-means kurz herausstellen.

k-median

Der k-median [BMS96] kann als Variante von k-means, bei dem die verwendete Distanz in der Zielfunktion nicht die quadratierte euklidische Distanz,

[2] Auch bezeichnet als Array oder Vektor.

sondern die Manhattan-Distanz ist, aufgefasst werden. Statt durch eine Verschiebung in den geometrischen Schwerpunkt wird diese durch Auswahl des komponentenweisen (geometrischen) Medians minimiert.

Entsprechend kann der k-median auch dann verwendet werden, wenn die Bildung eines arithmetischen Mittels nicht, der Median und die Manhattan-Distanz aber berechnet werden können. Ein weiterer Vorteil ist die Robustheit des Median gegenüber Ausreißern. In geringer Anzahl ist die Beeinflussung der Position der Clusterzentren zu vernachlässigen. Dem gegenüber steht die erhöhte Komplexität bei der Berechnung des Medians. Als holistische Funktion kann dieser im Gegensatz zum algebraischen arithmetischen Mittel nicht inkrementell und schwieriger parallel berechnet werden.

Da der Großteil der in dieser Arbeit vorgestellten Optimierungen rein auf die Eigenschaften einer Metrik aufbaut[3] und der k-median lediglich die Distanzfunktion austauscht, können diese auf den k-median angewendet und dieser analog beschleunigt werden.

Partitioning Around Medoids

Der auch als k-medoid bezeichnete, in [KJ09, Kapitel 2] vorgestellte, Partitioning Around Medoids (PAM) ermittelt analog zu k-means und k-median k durch Clusterzentren repräsentierte Cluster. Das Clustermodell entspricht dem Clustermodell des k-means. Als Zielfunktion wird die Summe der Distanzen von allen Punkten zu ihrem nächstgelegenen Clusterzentrum verwendet.

Genau wie beim k-means werden die Datenpunkte in jeder Iteration ihrem jeweils nächstgelegenen Clusterzentrum zugeordnet. Die Aktualisierung der Clusterzentren unterscheidet sich aber deutlich: Jedes Clusterzentrum wird mit einem diesem Clusterzentrum zugeordneten Datenpunkt so vertauscht, dass die Zielfunktion minimiert wird.

Daraus folgt unmittelbar, dass die einzige an die Datenpunkte gestellte Anforderung ist, dass eine Distanzfunktion existiert. Dadurch kann PAM für beliebige Daten eingesetzt werden. Es entsteht aber eine im Vergleich zum k-means deutlich erhöhte Laufzeitkomplexität.

[3] Einige Optimierungen beschleunigen die Neuberechnung des Clusterzentrums und sind daher nicht anwendbar.

Density-based spatial clustering of applications with noise

Ein Clusteringalgorithmus, der es nicht erfordert, dass die Anzahl der Cluster vorgegeben wird ist der Density-based spatial clustering of applications with noise (DBSCAN) [Est+96]. Dieser ermittelt auf Basis einer als Eingabe übergebenen maximalen Distanz („dichtebasiert“) zu den nächstgelegenen Datenpunkten selbstständig die Clusterzugehörigkeiten, bis alle Datenpunkte entweder einem Cluster zugeordnet oder als Ausreißer ausgeschlossen wurden. Auf diese Weise ist DBSCAN von keiner der in Abschnitt 2.1.6 genannten Probleme und Einschränkungen betroffen.

Aufgrund dieser Funktionsweise besitzt der DBSCAN im Vergleich zum k-means und auch PAM eine erhöhte Laufzeitkomplexität. Distanzen müssen nicht nur zu Clusterzentren oder Punkten innerhalb eines Clusters, sondern potentiell zu jedem anderen Datenpunkt berechnet werden.

3

Beschleunigung von k-means

Nachdem in Kapitel 2 die grundsätzliche Funktionsweise des Lloyd-Algorithmus zur Lösung des k-means-Problems hinreichend erläutert wurde, soll in den Abschnitten 3.1 bis 3.5 die Funktionsweise der Einzelkomponenten schrankenbasierter exakter k-means-Algorithmen vorgestellt werden. Anschließend wird in Abschnitt 3.6 diskutiert, wie diese Einzelkomponenten in konkret benennbaren Algorithmen zum Einsatz kommen.

3.1. Initialisierung

Wie in Abschnitt 2.1.5 bereits untersucht, ist die Initialisierung von hoher Wichtigkeit für die Qualität des Ergebnisses, sowie für die Anzahl der benötigten Iterationen und somit die Laufzeit des Clusterings. Ein populäres Verfahren, das nach aktuellem Kenntnisstand eine überdurchschnittlich gute Initialisierung liefert, ist k-means++, vorgestellt in [AV07]. [CKV12] empfiehlt unter anderem die Verwendung von „Greedy k-means++“, einer k-means++-Variante, die den Worst Case von k-means++ verbessern soll.

Da alle in dieser Arbeit betrachteten Algorithmen exakte Algorithmen sind, hat die Initialisierung auf alle Algorithmen den gleichen Einfluss. Wir stellen in dieser Arbeit daher nur den unmodifizierten k-means++-Algorithmus vor.

k-means++ ist ein probabilistischer Algorithmus. Nach der Auswahl eines zufälligen ersten Clusterzentrums werden alle weiteren Clusterzentren mit einer Wahrscheinlichkeit proportional zu ihrer quadratischen Distanz zum nächstgelegenen, bereits ausgewählten Clusterzentrum ausgewählt. Dieses Verfahren führt dazu, dass Punkte in möglichst weit entfernten Clustern ausgewählt werden, ohne dass die Auswahl auf einen Ausreißer fällt. Letztere besitzen zwar eine hohe Distanz zu anderen Datenpunkten, dies wird aber durch die höhere Anzahl an Punkten innerhalb der natürlichen Cluster ausgeglichen, sodass jeder Punkt eines Clusters für sich genommen zwar eine niedrige Chance zur Auswahl

hat, der Cluster in Summe aber eine hohe. Ein Beispiel dafür findet sich in Abbildung 3.1.

Insgesamt vermeidet k-means++ es auf diese Weise, mehrere initiale Clusterzentren im selben natürlichen Cluster auszuwählen. Diese müssten anschließend aufwändig aufgetrennt werden (Abbildung 2.3), was abhängig von der geometrischen Beschaffenheit nicht immer gelingt und dadurch die Qualität des Ergebnisses mindert. Abbildung 3.2 zeigt dies beispielhaft. Die Anzahl der zu findenden Cluster k entspricht genau der Anzahl der natürlichen Cluster. Dadurch, dass alle initialen Cluster aus dem natürlichen Cluster um $(0, 0)$ gewählt wurden, konnten die Cluster nicht sauber aufgetrennt werden.

Stattdessen werden die initialen Clusterzentren mit hoher Wahrscheinlichkeit in ihren Zielclustern platziert, sodass bei klar separierten natürlichen Clustern lediglich die Verschiebung in den Schwerpunkt von Nöten ist.

k-means++ verbessert auf diese Weise sowohl die Qualität des Ergebnisses[2] als auch die benötigte Laufzeit.

3.2. Schranken

Grundlage der Beschleunigung von k-means durch Schranken sind die drei Eigenschaften einer metrischen Distanzfunktion (d), insbesondere der Dreiecksungleichung (3.3).

Definition 4 (Metrische Distanzfunktion und Dreiecksungleichung)
Folgende Gleichungen gelten für eine metrische Distanzfunktion $d(\cdot, \cdot)$:

$$d(x, y) \geq 0 \wedge d(x, y) \iff x = y \tag{3.1}$$

$$d(x, y) = d(y, x) \tag{3.2}$$

$$d(x, z) \leq d(x, y) + d(y, z) \tag{3.3}$$

Die Dreiecksungleichung erlaubt es bei Kenntnis der Distanzen von zwei Punkten x und z zu einem gemeinsamen Punkt y Aussagen über die Distanz zwischen x und z zu treffen.

[1] $4 \cdot 1 + 4 \cdot 2 + 10.25 + 13 + 16.25 + 15.25 + 29.25$

[2] [AV07] beweist, dass die Zielfunktion J des für das resultierende Clustering innerhalb von $O(\log k)$ des optimalen Clusterings liegt.

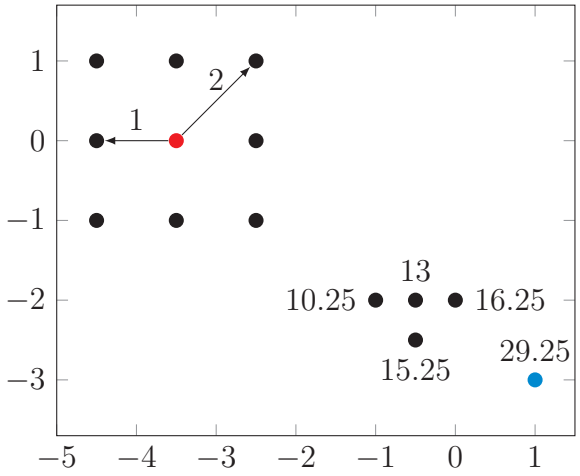


Abbildung 3.1.: Der rot markierte Punkt ist bereits ausgewählt. Die Summe der quadratischen Distanzen beträgt 96[1]. Mit einer Wahrscheinlichkeit von $\frac{54.75}{96} \approx 57\%$ wird ein Punkt im unteren Cluster ausgewählt, mit einer Wahrscheinlichkeit von $\frac{12}{96} \approx 13\%$ ein Punkt im gleichen Cluster und mit einer Wahrscheinlichkeit von $\frac{29.25}{96} \approx 30\%$ der blau markierte Ausreißer.

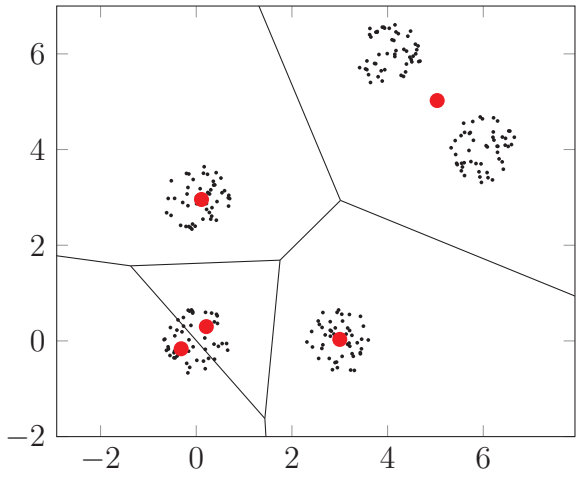


Abbildung 3.2.: Durch Auswahl aller initialen Clusterzentren im Cluster bei $(0, 0)$ nicht weiter auftrennbare Cluster.

In Verbindung mit der Annahme, dass Datenpunkte, insbesondere bei zunehmender Anzahl an Iterationen, nicht neu zugeordnet werden [Kan+00, Abschnitt 5; KFN00, Abschnitt IV] ist dies oftmals ausreichend, um zu ermitteln, ob ein Punkt *potentiell* neu zugeordnet werden muss.

Dazu lässt sich die Annahme in zwei Teilannahmen zerlegen:

1. Die Clusterzentren bewegen sich nur wenig.
2. Die Distanz von einem Datenpunkt zu seinem zugeordneten (und somit nächsten) Clusterzentrum ist deutlich kleiner als die Distanz zu dem zweitnächsten Clusterzentrum.

Die erste Annahme wird mit zunehmender Anzahl an Iterationen ausgeprägter, da sich die Clusterzentren dann bereits nahe an ihrer Zielposition befinden [Kan+00, Abschnitt 3]. Wenn keine Veränderung in einem Cluster stattfindet, dann bewegt sich das Clusterzentrum gar nicht.

Definition 5 (Statische und aktive Cluster)
Ein statischer Cluster ist ein Cluster, dessen Menge von zugeordneten Punkten ($A(c)$) sich in der aktuellen Iteration nicht geändert hat. Dem gegenüber steht ein aktiver Cluster, dessen Menge von zugeordneten Punkten sich geändert hat.

In [KFN00] waren in späteren Iterationen bis zu 80 % der Cluster statische Cluster. Eine Verletzung der Annahme ergibt sich, wenn die Bewegung dazu führt, dass weit entfernte Punkte einem Cluster neu zugeordnet werden, also die zweite Teilannahme verletzt wird.

Die zweite Annahme verlangt klar separierte natürliche Cluster, also dass die intra-Cluster-Distanzen deutlich kleiner als die inter-Cluster-Distanzen sind. In diesem Fall gibt es nur wenige Punkte nahe den Grenzen der Voronoi-Zellen, die bei der Verschiebung der Clusterzentren ihre Zuordnung ändern könnten. Das Zutreffen dieser Annahme ist daher abhängig von dem zu clusternden Datensatz.

Falls die erste Teilannahme zutreffend ist, erlauben es die namensgebenden Schranken, die Distanzen zwischen Datenpunkten und Clusterzentren mit einem geringen Fehler zu approximieren. Die Details dieser Approximierung werden in Abschnitt 3.2.1 diskutiert.

Falls die zweite Teilannahme zutreffend ist, erlaubt es ein simpler Vergleich zwischen den approximierten Schranken für einen Großteil der Punkte zu

ermitteln, dass diese ihre Zuordnung nicht ändern. Dies führt zu unserem ersten Pruningkriterium:

Pruningkriterium 1 (Vergleich von unterer und oberer Schranke)
Wenn eine untere Schranke für die Distanz zwischen Datenpunkt und Clusterzentrum (A) eine obere Schranke für die Distanz zwischen Datenpunkt und einem anderen Clusterzentrum (B) überschreitet, dann muss Clusterzentrum A weiter vom Datenpunkt entfernt sein als Clusterzentrum B. *Insbesondere kann Clusterzentrum A nicht das dem Datenpunkt am nächsten gelegene Clusterzentrum sein.*

Im Sinne der besseren Verständlichkeit soll an dieser Stelle noch die inverse Dreiecksungleichung (3.4) zur Verwendung in späteren Abschnitten definiert werden.

Definition 6 (Inverse Dreiecksungleichung)
Die inverse Dreiecksungleichung folgt durch einfaches Umformen aus der Dreiecksungleichung (3.3):

$$\begin{aligned} & d(x,y) \leq d(x,z) + d(z,y) \\ \iff \quad & d(x,y) - d(z,y) \leq d(x,z) \\ \iff \quad & d(x,y) - d(y,z) \leq d(x,z) \\ \iff \quad & d(x,z) \geq d(x,y) - d(y,z) \end{aligned} \tag{3.4}$$

3.2.1. Aktualisierung der Schranken

Bevor die einzelnen Schranken vorgestellt werden, soll zunächst diskutiert werden, auf welche Weise die tatsächlichen Distanzen durch Schranken approximiert werden und was für einen Einfluss die Art der Approximierung hat, da dies für das Verständnis der Unterschiede zwischen den und der Leistung der unterschiedlichen unteren Schranken essentiell ist.

Neben Anzahl und Art unterscheidet sich insbesondere auch der Zeitpunkt der Aktualisierung der Schranken je nach Algorithmus. Allen Algorithmen ist aber gemein, dass es nach der Verschiebung der Clusterzentren in der zweiten Phase notwendig ist, alle gespeicherten Schranken zu aktualisieren, damit die Invarianten der Schranken gültig bleiben. Eine scharfe obere Schranke würde

beispielsweise verletzt, wenn sich das Clusterzentrum von einem Datenpunkt entfernt.

Rein mit Hilfe der Eigenschaften der Distanzfunktion, insbesondere der Dreiecksungleichung, kann keine Aussage über die Bewegungsrichtung relativ zu dem Datenpunkt beziehungsweise den Datenpunkten, auf die sich eine Schranke bezieht, getroffen werden. Die Dreiecksungleichung garantiert lediglich, dass die Distanz zum Datenpunkt sich nicht um mehr als die zurückgelegte Distanz geändert haben kann. Dies folgt durch einfaches Einsetzen in die Dreiecksungleichung (3.3) beziehungsweise die inverse Dreiecksungleichung (3.4):

$$d(p, c_{new}) \leq d(p, c_{old}) + d(c_{old}, c_{new}) \tag{3.5}$$

$$d(p, c_{new}) \geq d(p, c_{old}) - d(c_{old}, c_{new}) \tag{3.6}$$

Aus diesem Grunde ist es bei der Aktualisierung typischerweise notwendig, die zurückgelegte Distanz „pessimistisch“ auf die oberen Schranken zu addieren, beziehungsweise von den unteren Schranken zu subtrahieren, um sicher zu stellen, dass die Schrankeninvariante nicht verletzt wird.

In [RH] werden zusätzliche geometrische Eigenschaften genutzt, die eine Aktualisierung erlauben, die präziser als die pessimistische Aktualisierung ist. Diese Eigenschaften sollen aber nicht Bestandteil dieser Arbeit sein.

Da die Schranken nach jeder Verschiebung der Clusterzentren um die zurückgelegte Distanz aktualisiert werden, summiert sich der Fehler durch die „pessimistische“ Annahme auf, wenn die Schranke zwischenzeitlich nicht durch eine notwendige exakte Distanzberechnung scharf aktualisiert werden kann. Diese Art der Aktualisierung wird daher als „Sum of Norms“, Summe der Bewegungen, bezeichnet.

Insbesondere dann, wenn ein Clusterzentrum sich auf einer elliptischen Bahn um einen Punkt bewegt, führt diese Art der Aktualisierung dazu, dass ein großer aufsummierter Fehler entsteht, obwohl sich das Zentrum seit der letzten exakten Aktualisierung der Schranke nur wenig bewegt hat. Es wird also die erste Annahme aus Abschnitt 3.2 verletzt. In [NF16] wird daher eine alternative Aktualisierung vorgeschlagen, die ebenfalls rein auf den Eigenschaften der Dreiecksungleichung aufbaut. Diese wird in Abschnitt 3.4.1 näher vorgestellt.

3.2.2. Obere Schranke

Als Vergleichswert in jedem der Pruningkriterien wird die Distanz zwischen einem Datenpunkt und dem bislang zugeordneten Clusterzentrum benötigt. Schließlich ist diese Distanz die Distanz, die es für eine Neuzuweisung zu unterschreiten gilt.

Anstelle diese Distanz in jedem Zuweisungsschritt erneut zu berechnen, bietet es sich an, diese durch eine obere Schranke zu approximieren. In jeder Iteration, in der die obere Schranke ausreichend präzise ist, um ein Pruning zu ermöglichen, kann so eine Distanzberechnung eingespart werden.

Für die Speicherung der oberen Schranke entsteht ein zusätzlicher linearer Speicherbedarf in der Anzahl der Datenpunkte: $\Theta(N)$. Für jeden Datenpunkt muss eine Distanz[3] gespeichert werden.

3.2.3. Untere Schranke

Neben einer oberen Schranke für das nächstgelegene Clusterzentrum setzen alle Algorithmen auf untere Schranken für die Distanzen zu den anderen Clusterzentren. Die untere Schranke ist der essentielle Aspekt bei der Beschleunigung. Der Großteil der Rechenzeit des Lloyd-Algorithmus wird für die (redundante) Berechnung der Distanzen zu den Clusterzentren aufgewendet. Um diese Berechnungen einzusparen, müssen entsprechend Informationen über diese Distanzen effizient vorgehalten werden.

Im Gegensatz zu der oberen Schranke gibt es hier durch die größere Anzahl an Clusterzentren drei grundsätzliche Alternativen der Ausgestaltung. Diese benennen wir nach dem (primären) Autor des Algorithmus, in denen die jeweilige Alternative vorgestellt wurde.

Elkan „k“

Der von Charles Elkan in [Elk03] vorgestellte „Elkan-Algorithmus“ speichert pro Datenpunkt und Clusterzentrum eine untere Schranke der Distanz. Diese untere Schranke bezieht sich ausschließlich auf das Paar und ist diesem fest zugeordnet.

Auf diese Weise erlaubt die Schranke mit Hilfe von Pruningkriterium 1 eine feingranulare Entscheidung, ob ein spezifisches Clusterzentrum ein potentiell

[3] Beispielsweise eine 64-bit double precision IEEE 754-Gleitkommazahl.

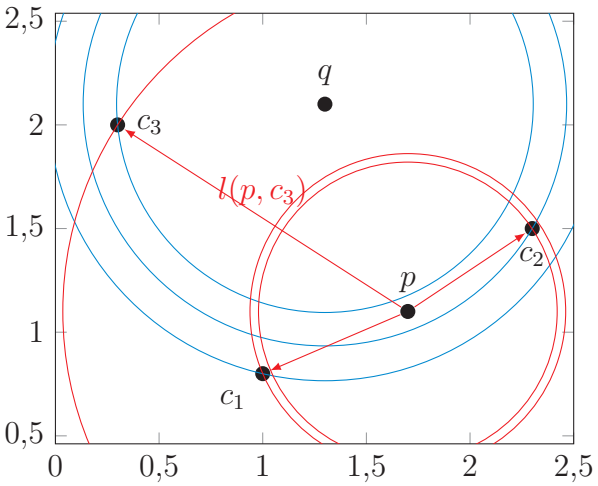


Abbildung 3.3.: (Scharfe) untere Schranken des Elkan. Die Clusterzentren müssen weiter als der Kreisradius von den Punkten entfernt sein. Die Abstandsvektoren für den Punkt q wurden im Sinne der Lesbarkeit ausgelassen.

nächstgelegenes Clusterzentrum ist oder nicht. Darüber hinaus kann die Schranke immer dann scharf aktualisiert werden, wenn eine exakte Distanzberechnung zwischen Datenpunkt und Clusterzentrum durchgeführt wird. Die Aktualisierung nach der Verschiebung der Clusterzentren (Abschnitt 3.2.1) verwendet das Delta des zugeordneten Clusterzentrums.

Die unteren Schranken des Elkan benötigen einen zusätzlichen Speicher in Höhe von $\Theta(N \cdot k)$, es wird eine Distanz pro Paar aus Datenpunkt und Clusterzentrum gespeichert.

Hamerly „1“

Im von Greg Hamerly in [Ham10] vorgestellten „Hamerly-Algorithmus“ wird pro Datenpunkt genau eine untere Schranke gespeichert. Dies soll den größten Nachteil des Elkan, die hohe Anzahl der unteren Schranken und den damit verbundenen hohen Speicherbedarf und hohen Aufwand bei der Korrektur der Schranken verbessern.

Diese repräsentiert die Distanz zum *zweit*nächsten Clusterzentrum. Mit Hilfe von Pruningkriterium 1 können auf diese Weise mit *einer* Prüfung potentiell alle Clusterzentren ausgeschlossen und somit alle Distanzberechnungen eingespart werden. Wenn dies nicht möglich ist, ist es, sofern kein anderes Pruningkriterium greift, hingegen notwendig, dass für alle Clusterzentren eine exakte Distanz

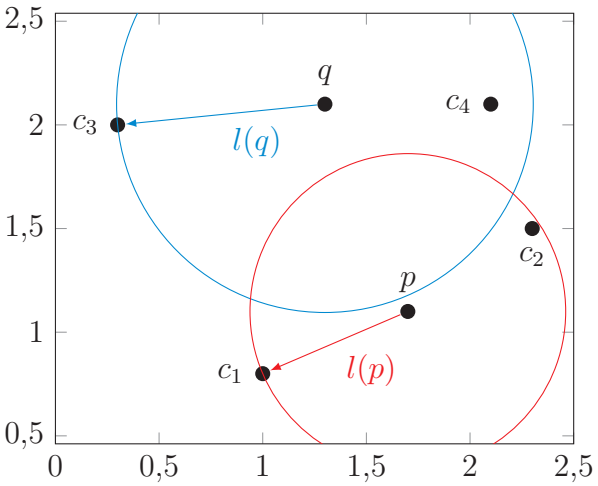


Abbildung 3.4.: (Scharfe) untere Schranke des Hamerly. Die Clusterzentren müssen weiter als der Kreisradius von den Punkten entfernt sein. c_2 beziehungsweise c_4 sind das nächstgelegene Clusterzentrum für p beziehungsweise q und haben daher einen geringeren Abstand als die scharfe untere Schranke.

berechnet wird. Nach dieser exakten Berechnung werden sowohl die obere als auch die untere Schranke scharf aktualisiert. Um die Korrektheit der Schranke nach der Verschiebung der Clusterzentren zu gewährleisten, wird diese um die Clusterbewegung mit dem größten Betrag nach unten korrigiert. Eine große Bewegung eines einzelnen Clusterzentrums („Big Mover“) hat dadurch einen großen Einfluss auf die Pruning-Möglichkeiten aller Clusterzentren.

Definition 7 (Big Mover)
Als „Big Mover“ werden Clusterzentren bezeichnet, die im Vergleich zu den durchschnittlichen inter-Cluster-Distanzen eine große Distanz zurücklegen und somit einen großen Approximationsfehler bei der pessimistischen Aktualisierung der Schranken verursachen.

Der Hamerly benötigt für seine untere Schranke einen Speicher von $\Theta(N)$ zusätzlichen Distanzen.

Drake „$1 \leq b \leq k$“

Die untere Schranke in Drakes Algorithmus, vorgestellt von Jonathan Drake und Greg Hamerly in [Dra12], lässt sich als Kombination von Elkan- und von

Hamerly-Schranke auffassen[4]. Diese soll die Nachteile beider Algorithmen, den hohen Overhead des Elkan und die Problematik der Big Mover des Hamerly, vermeiden.

Der Drake verwendet b sortierte Schranken, die Anzahl wird über die Laufzeit des Algorithmus variiert. Wie genau die Anzahl angepasst wird, wird im Abschnitt 3.6.3 zum konkreten Algorithmus „Drake“ näher betrachtet.

Die ersten $b - 1$ dieser Schranken beziehen sich, wie die untere Schranke des Elkan, auf jeweils ein konkretes Clusterzentrum, nämlich auf die $b - 1$ nächstgelegenen Clusterzentren, beginnend mit dem zweitnächsten. Die letzte untere Schranke ist, wie die untere Schranke des Hamerly, eine gemeinsame untere Schranke aller verbleibenden Clusterzentren.

Die Sortierung der Schranken reduziert den Aufwand beim Pruning. Ein erfolgreiches Pruning mit Hilfe von Pruningkriterium 1 führt unmittelbar dazu, dass auch alle verbleibenden Clusterzentren ausgeschlossen werden können. Einerseits kann dadurch die Prüfung des Pruningkriteriums für eine Vielzahl von Clusterzentren eingespart werden, wenn bereits eine der ersten Schranken ein Pruning ermöglicht. Auf der anderen Seite ist es dadurch nicht notwendig, die Clusterzentren der letzten (gemeinsamen) Schranke explizit zu speichern. Beim Prüfen des Pruningkriteriums kann beispielsweise eine Datenstruktur mit allen nicht-ausschließbaren Clusterzentren gefüllt werden. Wenn die letzte Schranke ein Pruning erlaubt, dann ist keine weitere Berechnung notwendig. Wenn auch die letzte Schranke kein Pruning erlaubt, dann sind *alle* Clusterzentren zu prüfen und die erzeugte Datenstruktur kann mit der Liste aller Clusterzentren ersetzt werden.

Für alle Clusterzentren, die nicht ausgeschlossen werden können, wird eine exakte Distanzberechnung durchgeführt und die zugehörigen Schranken werden scharf aktualisiert und neu sortiert.

Bei der Aktualisierung der Schranken nach der Bewegung der Clusterzentren ist es möglich, dass ein weiter entferntes Clusterzentrum eine größere Bewegung als ein näher gelegenes Clusterzentrum durchführt. Durch die pessimistische Aktualisierung würde dadurch potentiell die Ordnung der Schranken verletzt. Eine echte Sortierung der Schranken führt aufgrund der zusammengefassten letzten Schranke potentiell zu Problemen. Es könnte passieren, dass diese Schranke nach der Sortierung nicht mehr die letzte Schranke ist. Dies macht

[4]Beziehungsweise können umgekehrt Elkan- und Hamerly-Schranke als Spezialfall der Drake-Schranke aufgefasst werden.

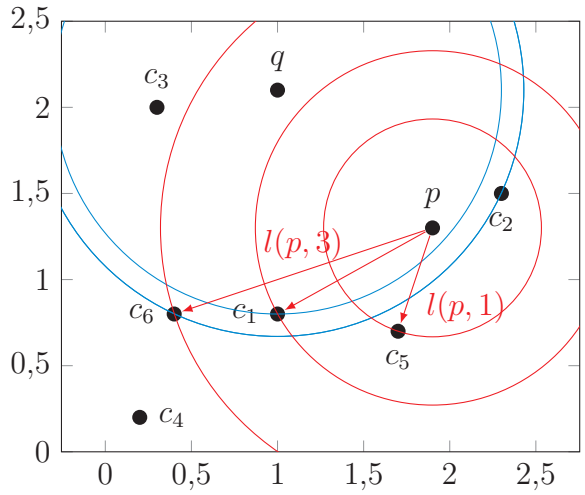


Abbildung 3.5.: (Scharfe) untere Schranke des Drake für $b = 3$. Die letzte Schranke bezieht sich jeweils auf alle verbleibenden Clusterzentren. Im Falle von Datenpunkt p sind dies c_6, c_3 und c_4. Für das nächstgelegene Clusterzentrum wird keine untere Schranke gespeichert.

den Vorteil der Sortierung, das effiziente Pruning, zu nichte. Die Bewegungen der Clusterzentren der ersten $b - 1$ Schranken müssten also mit den Bewegungen der Clusterzentren der letzten Schranke verglichen und die Clusterzentren möglicherweise ausgetauscht werden. Der Pflegeaufwand wäre deutlich erhöht.

Um dieses Problem zu umgehen, werden die Schranken bei der Aktualisierung nicht *sortiert*, sondern auf ihre Nachfolgeschranke *beschränkt*. Eine weitere Verringerung von unteren Schranken verletzt keine Invarianten, sondern macht diese lediglich weniger scharf. Da die in der letzten Schranke zusammengefassten Clusterzentren eine mutmaßlich große Distanz zu dem jeweiligen Punkt besitzen, besteht auch im Falle von Big Movern unter diesen ein ausreichender Sicherheitsabstand zu der oberen Schranke, wodurch das Pruningkriterium 1 im Vergleich zum Hamerly häufiger wirksam bleibt.

3.3. Weitere Pruningkriterien

Zusätzlich zum Vergleich von oberer und unterer Schranke verwenden die Algorithmen weitere Informationen, die es bei bekannter oberer Schranke erlauben, Clusterzentren zu prunen. Im Gegensatz zur unteren Schranke ist der Pflegeaufwand für diese zusätzlichen Pruningkriterien nicht von der Anzahl der

Datenpunkte abhängig. Die zusätzlichen Metadaten sind schnell zu ermitteln, im Gegenzug sind die entstehenden Pruningkriterien weniger stark.

3.3.1. Center-Center-Distanzen

Neben der unteren Schranke werden in [Elk03] die Distanzen zwischen den Clusterzentren als weiteres Pruningkriterium verwendet. Dazu werden zu Beginn jeder Iteration die paarweisen Distanzen aller Clusterzentren berechnet. Für jedes Clusterzentrum wird die Distanz zum nächstgelegenen Clusterzentrum gespeichert. Mit diesen Informationen lässt sich Pruningkriterium 2 konstruieren.

Pruningkriterium 2 (Center-Center-Distanz)
Alle Clusterzentren c, deren halbe Distanz zu dem, einem Datenpunkt p nächstgelegenen, Clusterzentrum a_p die obere Schranke u überschreitet, können ausgeschlossen werden.

$$\begin{aligned} \forall c \in C : \frac{d(c, a_p)}{2} &> u(p) \\ \implies c &\text{ kann ausgeschlossen werden.} \end{aligned} \tag{3.7}$$

In der Praxis wird der Aufruf der Distanzfunktion einmalig zu Beginn der Iteration und nicht pro Datenpunkt durchgeführt.

Eine beispielhafte Veranschaulichung findet sich in Abbildung 3.6. c_3 ist das dem Punkt p in der letzten Iteration nächstgelegene und daher aktuell zugeordnete Clusterzentrum. Aufgrund der Verschiebung der Clusterzentren ist die obere Schranke $u(p)$ nicht mehr scharf und daher größer als die tatsächliche Distanz zwischen p und c_3. Sie überschreitet die halbe Distanz zwischen c_3 und c_1 beziehungsweise c_4 und „ragt“ somit in die Voronoi-Zelle dieser Clusterzentren. Daher können c_1 und c_4 nicht gepruned werden. Die halbe Distanz zwischen c_3 und c_2 überschreitet die obere Schranke. Ein Pruning von c_2 ist daher möglich. Eine genaue Distanzberechnung würde aber zeigen, dass c_3 weiterhin das nächstgelegene Clusterzentrum ist. Eine Neuzuweisung findet nicht statt.

Zu Beginn jeder Iteration müssen die paarweisen Distanzen aller Clusterzentren berechnen werden. Es werden $\frac{k \cdot (k-1)}{2}$ zusätzliche Distanzberechnungen und ein ebenso großer zusätzlicher Speicherbedarf von $\Theta\left(\frac{k \cdot (k-1)}{2}\right)$ benötigt.

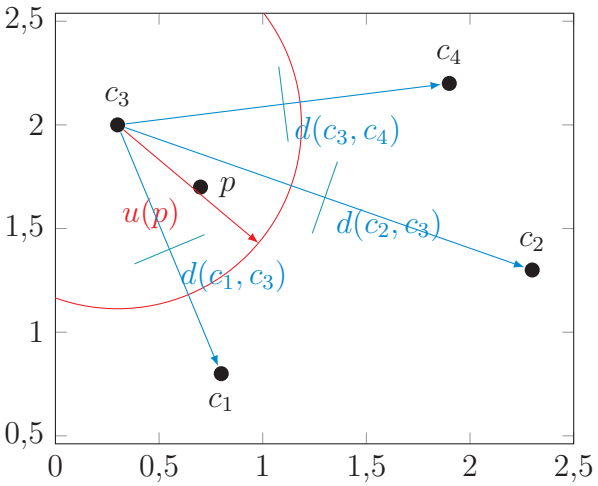


Abbildung 3.6.: Das Pruning mit Hilfe von Center-Center-Distanzen nutzt die obere Schranke $u(p)$ (rot) und die halbe Distanz zwischen dem aktuell zugeordneten und allen anderen Clusterzentren (blau).

3.3.2. Distanz zu einem Fixpunkt („Norm“)

Der im Rahmen von Drakes Masterarbeit [Dra13, Abschnitt 3.2] entwickelte und später in [HD17, Abschnitt 2.4.5] publizierte Annulus-Algorithmus[5] führt die Distanz zu einem Fixpunkt[6] als zusätzliches Pruningkriterium ein.

Pruningkriterium 3 (Distanz zu einem Fixpunkt)
Alle Clusterzentren c, deren Distanz zum Fixpunkt[7] von der Distanz des Datenpunktes p zum Fixpunkt um mehr als die obere Schranke u abweicht, können ausgeschlossen werden[8]:

$$\begin{aligned} &\forall c \in C : \big|\, \|c\| - \|p\| \,\big| > u(p) \\ &\qquad \Longrightarrow \; c \text{ kann ausgeschlossen werden.} \end{aligned} \tag{3.8}$$

Bei der Visualisierung (Abbildung 3.7) dieses Pruningkriteriums entsteht ein Ring („Annulus“) mit der doppelten oberen Schranke als Durchmesser um den Fixpunkt, sodass der Datenpunkt auf dem Kreis in der Mitte des

[5] In späteren Veröffentlichungen auch als Annular-Algorithmus bezeichnet.
[6] Beispielsweise der Ursprung des Koordinatensystems, also die Norm des Punktes.
[7] Im Sinne der Lesbarkeit wird hier der Ursprung und somit die Norm gewählt.
[8] Da der Annulus eine Erweiterung des Hamerly ist, verwendet das Originalpaper das Maximum aus u und l, um eine Aktualisierung der unteren Schranke zu ermöglichen. Diese Anpassung wird in Abschnitt 3.6.4 diskutiert.

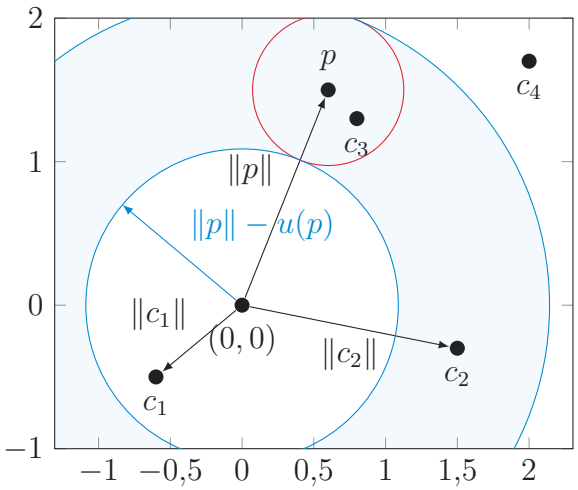


Abbildung 3.7.: Die obere Schranke (rot) des Datenpunktes p spannt den blau markierten Annulus auf.

Ringes liegt. Clusterzentren, die außerhalb des Annulus liegen, in diesem Fall c_1 und c_4, können ausgeschlossen werden. Clusterzentrum c_2 liegt innerhalb des Annulus und kann daher nicht ausgeschlossen werden, obwohl die Distanz zum Datenpunkt p die obere Schranke überschreitet.

Zur Verwendung dieses Pruningkriteriums müssen einmalig die Distanzen aller Datenpunkte zum Fixpunkt berechnet und gespeichert werden. Zu Beginn jeder Iteration müssen die k Distanzen der Clusterzentren zum Fixpunkt berechnet werden. Insgesamt entsteht so ein zusätzlicher Speicherbedarf von $\Theta(n + k)$.

3.4. Sonstige Verbesserungen

Neben den Pruningkriterien, die den Algorithmus durch Vermeidung von Distanzberechnungen beschleunigen, möchten wir auch einige weitere Vorschläge zur Beschleunigung schrankenbasierter, exakter k-means-Algorithmen betrachten. Diese haben unterschiedliche Ansatzpunkte. Teilweise haben sie zum Ziel, die Aktualisierung der Schranken zu verbessern, teilweise ist das Ziel, den Verwaltungsoverhead zu reduzieren und dadurch Rechenzeit einzusparen.

3.4.1. Norm of Sums

Wie in Abschnitt 3.2.1 bereits festgestellt, führt die aufsummierte Aktualisierung der Schranken nach jeder Iteration potentiell zu einem großen Approximations-

fehler. In [NF16, Abschnitt 3.2] wird eine alternative Aktualisierung vorgestellt, die anstatt der aufsummierten Bewegung aller Iterationen seit der letzten scharfen Aktualisierung einer Schranke die tatsächliche Abweichung von der Position zu diesem Zeitpunkt verwendet.

Der praktische Unterschied zwischen der „Sum of Norms“ und dieser „Norm of Sums“-Aktualisierung ist in Abbildung 3.8 dargestellt. Bei Sum of Norms wird die obere Schranke zwischen Punkt p und dem Clusterzentrum immer weiter pessimistisch erhöht. Bei Norm of Sums wird nur die tatsächliche Distanz zwischen der Position des Zentrums zum Zeitpunkt der letzten scharfen Aktualisierung c_{tight} und der aktuellen Position c_{cur} zur scharfen Schranke addiert.

Um die Abweichung von der derzeitigen Position zu bestimmen, ist es notwendig, die historischen Positionen der Clusterzentren zu speichern und für jede Schranke zu hinterlegen, in welcher Iteration diese zuletzt scharf aktualisiert wurde.

Zu Beginn jeder Iteration wird für jedes Paar aus Clusterzentrum und gespeicherter Position die Distanz der Abweichung berechnet. Bei der Verwendung einer Schranke wird diese um die Abweichung zur Position der Iteration der letzten scharfen Aktualisierung modifiziert. Der Approximationsfehler ist dann auf die pessimistische Aktualisierung beschränkt.

Um bei einer größeren Anzahl an benötigten Iterationen zu verhindern, dass der Speicherbedarf unbegrenzt wächst und um zu verhindern, dass die Abweichung für historische Positionen, die nur noch von wenigen Schranken referenziert werden, berechnet werden muss, werden die historischen Werte in regelmäßigen Abständen geleert. Dabei wird für jede Schranke hinterlegt, dass diese in der aktuellen Iteration zuletzt scharf aktualisiert wurde. Die gespeicherte Distanz wird, wie bei „Sum of Norms“, dauerhaft um die aktuelle Abweichung angepasst. Anschließend können alle historischen Distanzen verworfen werden. [NF16] schlägt vor, dass dies alle $t_{clear} = \frac{N}{\min(k,d)}$ Iterationen geschieht.

3.4.2. Delta Updates

Für die Berechnung des neuen Clusterzentrums (Abschnitt 2.1.2) ist es notwendig, das arithmetische Mittel der Datenpunkte eines Clusters zu bestimmen. Die naheliegende Lösung ist es, diese Zuordnung explizit zu speichern, alle Datenpunkte des zu aktualisierenden Clusters aufzusummieren und durch die

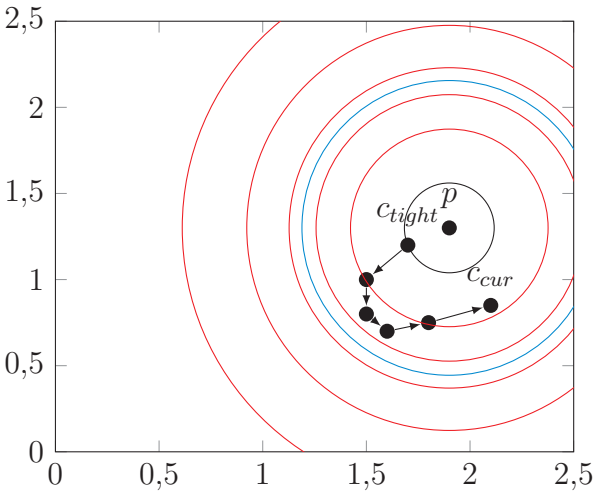


Abbildung 3.8.: Sum of Norms (rot) im Vergleich zu Norm of Sums (blau). Der Wert der letzten scharfen Aktualisierung der Schranke ist schwarz markiert.

Anzahl der Datenpunkte zu teilen. Diese Lösung hat den Nachteil, dass ein, in der Anzahl der Datenpunkte linearer, erhöhter Speicherbedarf entsteht. Auch ist es hier nicht möglich, auf ein lineares Feld zur Speicherung zurückzugreifen, da es im Falle einer Neuzuordnung nicht effizient aktualisiert werden könnte.

Eine Alternative ist es, über alle Datenpunkte zu iterieren und alle Cluster gleichzeitig aufzusummieren. Es wird zwar nur ein Speicher für k Zwischenergebnisse benötigt, aber auch für diese Lösung ist es notwendig, über alle Punkte zu iterieren, auch wenn nur wenige Neuzuweisungen stattgefunden haben.

Stattdessen schlägt [Ham10, Abschnitt 3.2.2] vor, sich zu nutze zu machen, dass das arithmetische Mittel algebraisch aggregiert werden kann. Für jeden Cluster wird die aktuelle Vektorsumme der Datenpunkte und Anzahl der zugeordneten Datenpunkte gespeichert. Bei einer Neuzuordnung eines Datenpunktes wird die Summe und Anzahl des alten Clusters verringert, die Summe und Anzahl des neuen Clusters erhöht. Die neue Position des Clusterzentrum kann so durch eine einfache Division der Vektorsumme durch die Anzahl berechnet werden.

Bei der Neuzuweisung eines Datenpunktes sind dadurch pro Punkt zwei Vektoradditionen notwendig, allerdings können N Vektoradditionen bei der Aktualisierung der Clusterzentren eingespart werden. Der zusätzliche Speicherbedarf von einem d-dimensionalen Vektor und einer Ganzzahl pro Clusterzentrum ist

im Vergleich zu einer expliziten Speicherung der einem Cluster zugeordneten Datenpunkte zu vernachlässigen.

3.4.3. Nebenläufigkeit

Die Berechnungen *innerhalb* der beiden Phasen (Abschnitt 2.1) des Lloyd-Algorithmus und daher auch der betrachteten beschleunigten Algorithmen sind nicht voneinander abhängig. Die Positionen der Datenpunkte sind über die gesamte Laufzeit des Algorithmus unverändert. Die Positionen der Clusterzentren verändern sich nur innerhalb der zweiten Phase (Abschnitt 2.1.2). Die vorgestellten Schranken beziehen sich immer auf einen konkreten Datenpunkt, möglicherweise aber auf mehrere Clusterzentren.

Eine nebenläufige Ermittlung des nächstgelegenen Clusterzentrums *unterschiedlicher Datenpunkte* ist so auch ohne den Einsatz von Sperren sicher möglich, sofern die verwendeten Datenstrukturen einen sicheren nebenläufigen Zugriff auf *unterschiedliche* Elemente erlauben. Lediglich bei der Cluster-Neuzuweisung ist es notwendig, auf Sperren oder atomare Schreiboperationen zu setzen, damit eine nebenläufige Zuweisung zum gleichen Cluster das korrekte Ergebnis liefert. Da schon die Verwendung von Schranken auf dem Grundsatz fußt, dass eine Neuzuweisung nur selten stattfindet, ist zu erwarten, dass auf die Erlangung dieser Sperren nicht gewartet werden muss.

Aus diesem Grund ist das k-means-Problem „embarrassingly parallel“. Eine Parallelisierung ist ohne nennenswerten zusätzlichen Rechenaufwand zur Sicherstellung der Korrektheit möglich, die Beschleunigung verhält sich dauerhaft nahezu linear zu der Anzahl der verwendeten Recheneinheiten.

3.5. Eckdaten der Techniken

Damit ergeben sich die in Tabelle 3.1 dargestellten Eckdaten der vorgestellten Techniken zur Beschleunigung in exakten, schrankenbasierten k-means-Clustering-Algorithmen.

Distanzen in der Spalte zum Speicherbedarf beziehen sich auf zu speichernde Rückgabewerte der Distanzfunktion. In der Regel werden diese in [IEEE 754] standardisierte binary64-Gleitkommazahlen[9] mit Bitlänge 64 sein. Punkte sind d-dimensionale Vektoren. Der Speicherbedarf bezieht sich rein auf die Nutzdaten,

[9] Üblicherweise als „double precision“-Gleitkommazahlen bezeichnet.

ohne Beachtung des Overheads der eingesetzten Datenstruktur oder etwaiger temporärer Variablen.

Der Pflegeaufwand bezieht sich auf den fixen, inhärenten zusätzlichen Pflegeaufwand pro Iteration. Der entstehende Aufwand zur Prüfung eines Pruningkriteriums ist nicht berücksichtigt. Ebenfalls keine Berücksichtigung findet die scharfe Aktualisierung von Schranken falls kein Pruning möglich ist und eine exakte Distanzberechnung durchgeführt werden muss. In diesem Fall muss nur der ohnehin berechnete Wert innerhalb der Datenstruktur aktualisiert werden. Die Anpassung der Distanzen als Pflegeaufwand bezieht sich auf die Aktualisierung der Schranken (Abschnitt 3.2.1). Eine Anpassung ist eine Addition beziehungsweise Subtraktion von zwei Distanzen: Der aktuellen Schranke und der Bewegung des Clusterzentrums.

Für Delta Updates (Abschnitt 3.4.2) und Nebenläufigkeit (Abschnitt 3.4.3) lässt sich kein fixer Speicherbedarf und kein fixer Pflegeaufwand angeben.

Technik	Maximaler Speicherbedarf	Pflegeaufwand pro Iteration
Obere Schranke	N Distanzen	Anpassung von N Distanzen
Elkan-Schranke	$N \cdot k$ Distanzen	Anpassung von $N \cdot k$ Distanzen
Hamerly-Schranke	N Distanzen	Anpassung von N Distanzen
Drake-Schranke	$N \cdot k$ Distanzen	Anpassung von $N \cdot m$ Distanzen
Center-Center	$\frac{k \cdot (k-1)}{2}$ Distanzen	Berechnung von $\frac{k \cdot (k-1)}{2}$ Distanzen
Norm	$N + k$ Distanzen	Berechnung von k Distanzen
Sum Of Norms	-	Berechnung von k Distanzen
Norm Of Sums	$k \cdot t_{clear}$ Punkte, N Ganzzahlen	Berechnung von im Mittel $k \cdot \frac{t_{clear}}{2}$ Distanzen

Tabelle 3.1.: Eckdaten der Techniken zur Beschleunigung in exakten, schrankenbasierten k-means-Clustering-Algorithmen.

3.6. Einsatz in den Algorithmen

Die reine *Existenz* von Pruningkriterien erlaubt noch keine Beschleunigung des Lloyd-Algorithmus. Nachfolgend soll daher untersucht werden, wie die zuvor vorgestellten Pruningkriterien in unterschiedlichen Algorithmen zum Einsatz kommen, wie die zusätzlich zu speichernden Informationen (etwa die Schranken) im Detail gepflegt und aktualisiert werden und welche weiteren Prozesse stattfinden müssen, damit der Algorithmus beschleunigt, aber *insbesondere auch exakt* bleibt.

3.6.1. Elkan

Im Elkan [Elk03] werden, neben der Elkan-Schranke (Seite 23) und einer oberen Schranke, Center-Center-Distanzen zum Pruning verwendet. Die Definition der Pruningkriterien kann im Elkan unmittelbar übernommen und abgeprüft werden. Als weitergehende Berechnungen sind lediglich die Aktualisierung der Schranken nach Verschiebung der Clusterzentren (Abschnitt 3.2.1) und die Berechnung der Center-Center-Distanzen notwendig.

Eine um die Prüfung der Pruningkriterien erweiterte Version der Prozedur „`assignPointsToCluster`“ des Lloyd (Algorithmus 2.3) für einen konkreten Punkt p findet sich in Algorithmus 3.1. Die Variable `nearest_dist` konnte entfallen. Sie wurde durch eine Abfrage der oberen Schranke des Punktes ersetzt.

Zunächst wird die minimale Center-Center-Distanz für das aktuell derzeit zugeordnete Clusterzentrum mit Hilfe von Pruningkriterium 2 überprüft. Wenn dieses relativ schwache Pruningkriterium ein Pruning erlaubt, dann können alle Clusterzentren ausgeschlossen werden. Die Bearbeitung des Datenpunktes ist abgeschlossen.

Andernfalls werden für jedes Clusterzentrum die untere Schranke (Zeile 11; Pruningkriterium 1) und die exakte Center-Center-Distanz (Zeile 14; Pruningkriterium 2) überprüft.

Wenn auch damit kein Pruning möglich ist, dann wird die obere (und untere) Schranke, die durch die Aktualisierung der Position der Clusterzentren nicht mehr scharf ist, mit einer Distanzberechnung scharf aktualisiert und die Prüfung wiederholt. Diese Distanzberechnung ist nur einmalig pro Punkt notwendig.

Algorithmus 3.1 assignPointsToCluster für einen konkreten Datenpunkt p im Elkan

```
 1: procedure assignPointsToClusterElkan(p)
 2:     nearest ← getAssignedCluster(p)
 3:     loose ← True
 4:     if cc_g(nearest) · 0.5 > u(p) then          ▷ Globales Center-Center
 5:         return
 6:     end if
 7:     for all c ∈ C do
 8:         if c = nearest then
 9:             continue with next c
10:         end if
11:         if l(p, c) > u(p) then                   ▷ Untere Schranke
12:             continue with next c
13:         end if
14:         if cc(nearest, c) · 0.5 > u(p) then      ▷ Center-Center
15:             continue with next c
16:         end if
17:         if loose then
18:             u(p) ← d(p, nearest)                 ▷ Obere Schranke scharf
19:             l(p, nearest) ← u(p)                 ▷ Untere Schranke scharf
20:             loose ← False
21:             if l(p, c) > u(p) then               ▷ Untere Schranke
22:                 continue with next c
23:             end if
24:             if cc(nearest, c) · 0.5 > u(p) then  ▷ Center-Center
25:                 continue with next c
26:             end if
27:         end if
28:         l(p, c) ← d(p, c)                        ▷ Untere Schranke scharf
29:         if l(p, c) < u(p) then                   ▷ Prüfung
30:             nearest ← c
31:             u(p) ← l(p, c)
32:         end if
33:     end for
34:     assignPointToCluster(p, nearest)
35: end procedure
```

Da im Falle einer Neuzuweisung eine Distanzberechnung stattgefunden haben muss, bleibt die obere Schranke bis zur Verschiebung der Clusterzentren scharf.

Wenn auch dann kein Pruning möglich ist, dann ist für das aktuelle Clusterzentrum eine exakte Distanzberechnung notwendig. Die untere Schranke kann in diesem Zuge scharf aktualisiert werden.

Abschließend erfolgt in Zeile 29 analog zum Lloyd eine Prüfung, ob das aktuelle Clusterzentrum näher als das aktuell zugeordnete am Datenpunkt liegt. In diesem Falle ist eine Neuzuweisung notwendig. Die obere Schranke wird entsprechend angepasst.

Da den Schranken entweder unmittelbar (untere Schranke) oder mittelbar (zugeordnetes Clusterzentrum für obere Schranke) ein konkretes Clusterzentrum zugeordnet ist, können diese nach Verschiebung der Clusterzentren mit der konkreten Bewegung eines Clusterzentrums aktualisiert werden.

Eine Variante des Elkan ohne den Einsatz von Pruningkriterium 2 wird auch als „Simplified Elkan“ bezeichnet [NF16, Abschnitt 2.2].

3.6.2. Hamerly

Der Hamerly [Ham10] verwendet, genau wie der Elkan, obere und untere Schranke, sowie Center-Center-Distanzen. Das Pruning beim Hamerly erfolgt binär: Entweder können alle Clusterzentren gepruned werden oder keines. Die Hamerly-Schranke erlaubt kein präziseres Pruning, die Center-Center-Distanzen würden es analog zum Elkan erlauben.

Die Struktur des Prunings im Hamerly ist sehr ähnlich dem Pruning im Elkan. Der Unterschied besteht darin, dass das Pruning nicht innerhalb der Schleife über alle Zentren stattfindet, sondern, wie die globalen Center-Center-Distanzen des Elkans, unmittelbar vor der Schleife. Wie beim Elkan werden die Pruningkriterien 1 (Zeile 3) und 2 (Zeile 6) gegen die aktuelle obere Schranke geprüft. Wenn kein Pruning möglich ist, wird ganz analog die obere Schranke scharf aktualisiert und die Pruningkriterien werden erneut geprüft.

Wenn auch das kein Pruning erlaubt, dann müssen, analog zum Lloyd, die Distanzen zu allen Clusterzentren ermittelt werden, um das nächstgelegene Clusterzentrum zu bestimmen. Da alle Distanzen berechnet werden müssen, wird in diesem Zuge die untere Schranke des Hamerly scharf aktualisiert. Sie entspricht der Distanz zum zweitnächsten Clusterzentrum.

Algorithmus 3.2 assignPointsToCluster für einen konkreten Datenpunkt p im Hamerly

```
1: procedure assignPointsToClusterHamerly(p)
2:     nearest ← getAssignedCluster(p)
3:     if l(p) > u(p) then                                  ▷ Untere Schranke
4:         return
5:     end if
6:     if cc_g(nearest) · 0.5 > u(p) then                   ▷ Globales Center-Center
7:         return
8:     end if
9:     u(p) ← d(p, nearest)                                 ▷ Obere Schranke scharf
10:    if l(p) > u(p) then                                  ▷ Untere Schranke
11:        return
12:    end if
13:    if cc_g(nearest) · 0.5 > u(p) then                   ▷ Globales Center-Center
14:        return
15:    end if
16:    nearest ← ⊥
17:    u(p) ← ∞
18:    l(p) ← ∞
19:    for all c ∈ C do
20:        dist ← d(p, c)
21:        if dist < u(p) then
22:            l(p) ← u(p)
23:            nearest ← c
24:            u(p) ← dist
25:        else if dist < l(p) then
26:            l(p) ← dist
27:        end if
28:    end for
29:    assignPointToCluster(p, nearest)
30: end procedure
```

Die Anpassung der oberen Schranke erfolgt wie beim Elkan. Da sich die untere Schranke nicht auf ein konkretes Clusterzentrum, sondern auf alle Clusterzentren außer dem nächstgelegenen bezieht, muss die untere Schranke um die maximale Bewegung innerhalb der Menge dieser Clusterzentren angepasst werden.

Analog zum „Simplified Elkan“ gibt es einen „Simplified Hamerly“, der keine Center-Center-Distanzen nutzt. Die später in dieser Arbeit betrachteten Algorithem „Annulus“ (Abschnitt 3.6.4) und „Exponion“ (Abschnitt 3.6.5) sind unmittelbare Erweiterungen des Hamerly um leichtgewichtige Pruningkriterien zur Vermeidung des binären Prunings.

3.6.3. Drake

Beim Drake [Dra12] kommt ausschließlich Pruningkriterium 1 auf Basis der Drake-Schranke zum Einsatz.

Als erstes prüft der Drake mit der Schleife in Zeile 3, ab welcher unteren Schranke ein Pruning mit Hilfe von Pruningkriterium 1 möglich ist. Wenn die erste Schranke ein Pruning erlaubt (Zeile 9), dann ist das zweitnächste Clusterzentrum weiter entfernt als das nächste Clusterzentrum. Entsprechend kann die Prüfung an dieser Stelle abgebrochen werden. Wenn keine Schranke ein Pruning erlaubt (Zeile 11), dann müssen alle Clusterzentren geprüft werden. Diese sind Kandidaten das nächstgelegene Clusterzentrum zu sein. In allen anderen Fällen sind das aktuell zugeordnete sowie alle Clusterzentren, die nicht gepruned werden können, Kandidaten das nächstgelegene Clusterzentrum zu sein.

Anschließend werden die Kandidaten nach ihrer Distanz zum Punkt p sortiert (Zeile 19). Hierbei wird die exakte Distanz zwischen p und dem Kandidaten benötigt. Diese Berechnung ist im Algorithmus nicht dargestellt, sie könnte aber im Rahmen der Erstellung der Kandidatenliste erfolgen. In jedem Fall muss die Distanz auch nach der Sortierung zur Verfügung stehen, da sie in den Zeilen 21 und 23 noch zur scharfen Aktualisierung der Schranken benötigt wird.

Die sortierte Kandidatenliste enthält nun die z nächstgelegenen Clusterzentren. Das erste Clusterzentrum wird dem Punkt zugewiesen und zur scharfen Aktualisierung der oberen Schranke genutzt. Alle anderen werden genutzt, um die unteren Schranken scharf zu aktualisieren.

Algorithmus 3.3 assignPointsToCluster für einen konkreten Datenpunkt p im Drake

```
1: procedure assignPointsToClusterDrake(p)
2:     z ← ⊥
3:     for i ← 1..b do                                  ▷ Prüfung des Prunings
4:         if l(p, i) > u(p) then
5:             z ← i
6:             break loop
7:         end if
8:     end for
9:     if z = 1 then                ▷ Alle Clusterzentren können gepruned werden
10:        return
11:    else if z = ⊥ then           ▷ Kein Clusterzentrum kann gepruned werden
12:        candidates ← C
13:    else                         ▷ Manche Clusterzentren können gepruned werden
14:        candidates[1] ← getAssignedCluster(p)
15:        for i ← 1..(z − 1) do
16:            candidates[i + 1] ← l_c(p, i)
17:        end for
18:    end if
19:    sort candidates by their distance to p
20:    assignPointToCluster(p, candidates[1])   ▷ Aktualisierung der oberen Schranke
21:    u(p) ← d(p, candidates[1])
22:    for i ← 2..length(candidates) do   ▷ Aktualisierung der unteren Schranke
23:        l(p, i − 1) ← d(p, candidates[i])
24:    end for
25: end procedure
```

Nach den Annahmen in Abschnitt 3.2 ist zu erwarten, dass nach wenigen Iterationen nur geringe Änderungen an der Clusterzuordnung notwendig sind und Datenpunkte aufgrund des Verhältnisses aus intra-Cluster- und inter-Cluster-Distanzen nicht plötzlich einem bislang weit entfernten Clusterzentrum zugeordnet werden. Entsprechend führen primär die ersten unteren Schranken zu einem Pruning. Die Pflege präziser Schranken zu weiter entfernten Clusterzentren ist daher nicht mehr notwendig und führt zu einem vermeidbaren Overhead. Aus diesem Grund wird in jeder Iteration ermittelt, wie viele Schranken zum Pruning *maximal* notwendig waren und die Anzahl der eingesetzten Schranken wird nach unten auf diesen Wert angepasst. Die weiter entfernten Clusterzentren werden damit in der letzten Hamerly-Schranke zusammengefasst. Auf diese Weise variiert der Drake die Anzahl der Schranken von anfänglichen $\frac{k}{4}$ auf potentiell nur $\frac{k}{8}$ Schranken.

Die ersten $b-1$ Schranken beziehen sich auf ein konkretes Clusterzentrum. Entsprechend werden diese, wie beim Elkan, um die Bewegung dieses Clusterzentrums angepasst. Die letzte Schranke bezieht sich auf alle verbleibenden Clusterzentren. Da diese nicht explizit gespeichert werden sollen und schon bei der Zuordnung der Datenpunkte zur Verringerung der Anzahl der Schranken angenommen wird, dass die letzte Schranke einen hohen Betrag hat, ist auch der Einfluss von Big Movern auf die Pruningleistung begrenzt. Entsprechend ist es praktikabel, die letzte Schranke um die maximale Bewegung *aller* Clusterzentren anzupassen, um Rechenaufwand zur Auswahl des passenden Clusterzentrums einzusparen.

Wie schon in der Vorstellung der Drake-Schranke genannt (Seite 26), führt die Aktualisierung der Schranken potentiell zu einer Verletzung der aufsteigenden Ordnung. Eine echte Sortierung würde einen großen Aufwand bedeuten, entsprechend werden die Schranken auf ihren Nachfolger beschränkt. Sie sind dadurch weniger scharf, aber nach den getroffenen Annahmen hat dies keine Auswirkung auf die Pruningleistung. Wenn die Aktualisierung der Schranken in umgekehrter Reihenfolge, beginnend mit Schranke b, stattfindet, dann kann diese Beschränkung unmittelbar bei der Aktualisierung berücksichtigt werden. Als zusätzlicher Berechnungsaufwand wird lediglich ein zusätzlicher Vergleich zur Prüfung, ob die Schranke ihren Nachfolger überschreitet, benötigt. Im Vergleich zu einer echten Sortierung ist dies zu vernachlässigen.

3.6.4. Annulus

Der Annulus-Algorithmus [Dra13] ist eine strikte Erweiterung des in Abschnitt 3.6.2 vorgestellten Hamerly und führt die Distanz zu einem Fixpunkt (Pruningkriterium 3, Abschnitt 3.3.2) als zusätzliches Pruningkriterium ein.

Dies erlaubt ein feingranulareres Pruning im Vergleich zum binären Pruning nur mit der unteren Schranke.

Um sicher zu stellen, dass die untere Schranke, die sich auf alle Clusterzentren bezieht, auch beim Pruning einzelner Clusterzentren korrekt scharf aktualisiert werden kann, dürfen Clusterzentren, die potentiell die untere Schranke unterschreiten, nicht gepruned werden. Entsprechend muss Pruningkriterium 3 so angepasst werden, dass nicht der Betrag der oberen Schranke, sondern das Maximum aus oberer Schranke und der Distanz zum zweitnächsten bekannten Clusterzentrum für den Durchmesser des Annulus verwendet wird.

Die *obere* Schranke für Distanz zum *zweit*nächsten Clusterzentrum muss also bekannt sein. Diese wird im Gegensatz zur *unteren* Schranke im Hamerly nicht explizit gespeichert. Aus diesem Grunde speichert der Annulus zusätzlich die Identität des zweitnächsten Clusterzentrums. Diese wird dazu genutzt, um bei nicht möglichem Pruning mit Hilfe von Pruningkriterium 1 eine obere Schranke für die Distanz zum zweitnächsten Clusterzentrum zu berechnen. Die Berechnung dieser Distanz wäre, da kein weiteres Pruning mehr möglich wäre, im Hamerly ohnehin erfolgt.

Die Zuordnung der Datenpunkte verläuft nahezu identisch zum Hamerly. Die Anwendung der Pruningkriterien des Hamerly verläuft identisch. Nur dann, wenn der Hamerly nicht prunen kann, kommt ab Zeile 16 das zusätzliche Pruningkriterium zum Einsatz. Wie oberhalb diskutiert, wird zunächst eine obere Schranke für das zweitnächste Clusterzentrum bestimmt. Diese wird als „Startwert“ für die neue, scharfe untere Schranke des Hamerly verwendet (Zeile 17) und bei der exakten Berechnung der Distanzen zu nicht ausschließbaren Clusterzentren analog zum Hamerly weiter verringert.

Nachdem die Größe des Annulus mit Hilfe der Distanz zum nächsten und zweitnächsten Clusterzentrum festgelegt wurde, findet in Zeile 19 das eigentliche Pruning mit Hilfe des zusätzlichen Pruningkriteriums 3 statt. Die Notation mittels Pseudocode übergeht im Sinne der Lesbarkeit einige Aspekte, die in einer konkreten Implementierung berücksichtigt werden können und sollten. Diese sollen daher an dieser Stelle genannt werden.

Algorithmus 3.4 assignPointsToCluster für einen konkreten Datenpunkt p im Annulus

```
 1: procedure assignPointsToClusterAnnulus(p)
 2:     nearest ← getAssignedCluster(p)
 3:     if l(p) > u(p) then
 4:         return
 5:     end if
 6:     if cc_g(nearest) · 0.5 > u(p) then
 7:         return
 8:     end if
 9:     u(p) ← d(p, nearest)
10:     if l(p) > u(p) then
11:         return
12:     end if
13:     if cc_g(nearest) · 0.5 > u(p) then
14:         return
15:     end if                          ▷ Bis zu dieser Stelle unverändert
16:     nearest2 ← getNearest2(p)
17:     l(p) ← d(p, nearest2)           ▷ Obere Schranke der Distanz zum
    zweitnächsten Clusterzentrum
18:     annulus_size ← max(u(p), l(p))
19:     candidates ← {c ∈ C | | ||c|| − ||p|| | ≤ annulus_size}   ▷ Zusätzliches
    Pruning
20:     for all c ∈ candidates do
21:         dist ← d(p, c)
22:         if dist < u(p) then
23:             nearest2 ← nearest
24:             l(p) ← u(p)
25:             nearest ← c
26:             u(p) ← dist
27:         else if dist < l(p) then
28:             nearest2 ← c
29:             l(p) ← dist
30:         end if
31:     end for
32:     assignPointToCluster(p, nearest)
33:     assignNearest2(p, nearest2)
34: end procedure
```

Der erste Aspekt ist die redundante Berechnung der Normen innerhalb der Gleichung. Die Position des Datenpunktes bleibt über die gesamte Laufzeit des Algorithmus unverändert. Entsprechend sollte $\|p\|$ einmalig zu Beginn des Algorithmus berechnet und anschließend nur abgerufen werden. Analog bleibt die Position der Clusterzentren innerhalb einer Iteration für alle Punkte unverändert, sodass auch diese nur einmalig pro Iteration berechnet werden sollten.

Der zweite Aspekt ist die Verwendung einer binären Suche zur effizienten Ermittlung der nicht prunbaren Clusterzentren. Wenn die Clusterzentren nach Berechnung der Normen aufsteigend nach ihrer Norm sortiert werden, dann kann das erste Clusterzentrum, auf das die Bedingung $\big|\, \|c\| - \|p\| \,\big| \leq annulus\_size$ zutrifft, mit Hilfe einer binären Suche bestimmt werden. Aufgrund der Stetigkeit der Betragsfunktion und Monotie auf beiden Seiten der Nullstelle erfüllen alle folgenden Clusterzentren die Bedingung, bis diese das erste mal verletzt wird. Anschließend erfüllt kein weiteres Clusterzentrum die Bedingung. Entsprechend können nach der binären Suche des ersten Clusterzentrums alle weiteren Clusterzentren linear geprüft und bei Verletzung der Bedingung die Suche abgebrochen werden.

Im Vergleich zum Hamerly sind bei der Aktualisierung der Schranken keine Änderungen notwendig. Die zusätzlichen Informationen werden entweder einmalig zu Beginn des Algorithmus (Distanz der Datenpunkte zum Fixpunkt) ermittelt oder müssen in jeder Iteration scharf berechnet werden (Distanz der Clusterzentren zum Fixpunkt), sodass diese Berechnung bei der Zuordnung der Datenpunkte stattfinden kann.

Eine Variante des Annulus auf Basis des „Simplified Hamerly“ bezeichnen wir als „Simplified Annulus“.

3.6.5. Exponion

Genau wie der Annulus-Algorithmus ist auch der Exponion [NF16] eine strikte Erweiterung des Hamerly (Abschnitt 3.6.2). Der Exponion kann als Weiterentwicklung des Annulus mit dem Ziel, das zusätzliche Pruningkriterium, den namensgebenden Ring, effektiver zu machen, verstanden werden[10].

[10] Tatsächlich ist der Exponion dem Hamerly aber näher als der Annulus.

Dazu verschiebt der Exponion den Mittelpunkt des Ringes[11] von einem Fixpunkt zu dem jeweils aktuell zugeordneten Clusterzentrum. Diese Änderung ist dadurch motiviert, dass das Volumen einer Hyperkugel in Dimension d vom Radius r als d-te Potenz abhängt [DLMF, S. 5.19.iii]. Durch die Verschiebung des Mittelpunktes soll eine Reduzierung des Radius erzielt werden.

Das resultierende Pruningkriterium ist das bereits bekannte Pruningkriterium 2, die Center-Center-Distanzen. Diese kommen im Hamerly bereits als globales Pruningkriterium zum Einsatz. Um die Center-Center-Distanzen zusätzlich lokal, das heißt für ein spezifisches Clusterzentrum, einsetzen zu können, ist eine kleine Modifikation erforderlich. Es muss zusätzlich der minimale Abstand aller Clusterzentren zum aktuell zugeordneten Clusterzentrum a_p berücksichtigt werden. Andernfalls würde potentiell das zweitnächste Clusterzentrum ausgeschlossen (Abbildung 3.9b). Dieses wird aber, wie bei der Vorstellung des Annulus diskutiert, zur scharfen Aktualisierung der unteren Schranke benötigt.

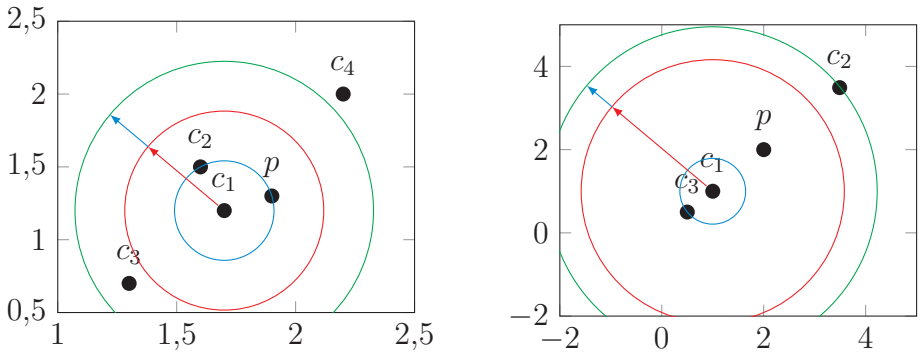
Pruningkriterium 4 (Center-Center-Distanz (Exponion))
Alle Clusterzentren c, deren Distanz zu dem, einem Datenpunkt p nächstgelegenen, Clusterzentrum a_p die Summe aus der doppelten oberen Schranke u und der minimalen Distanz zwischen a_p und einem anderen Clusterzentrum überschreitet, können weder das nächstgelegene, noch das zweitnächste Clusterzentrum sein und somit ausgeschlossen werden:

$$\begin{aligned} &\forall c \in C : d(c, a_p) > 2 \cdot u(p) + \min_{c' \in C \setminus \{a_p\}} \{d(c', a_p)\} \\ &\implies c \text{ kann ausgeschlossen werden.} \end{aligned} \tag{3.9}$$

In der Praxis wird der Aufruf der Distanzfunktionen einmalig zu Beginn der Iteration und nicht pro Datenpunkt durchgeführt.

Eine einfache Umformung der Bedingung von Pruningkriterium 4 zeigt die Abstammung von der Center-Center-Distanz. Auf der rechten Seite kommt der

[11] Der beim Exponion ein regulärer Kreis, beziehungsweise eine reguläre Hyperkugel ist.



(a) c_2 und c_3 befinden sich im grünen Kreis und können daher nicht ausgeschlossen werden. c_4 kann ausgeschlossen werden.

(b) Der Ring des Exponion ist scharf. c_2 befindet sich infinitesimal näher an p als c_3.

Abbildung 3.9.: c_1 ist das dem Punkt p nächstgelegene Clusterzentrum. c_2 ist das zweitnächste Clusterzentrum und darf daher nicht ausgeschlossen werden. Die regulären Center-Center-Distanzen entsprechen dem roten Kreis.

zweite Summand hinzu.

$$\begin{aligned} \forall c \in C : \frac{d(c, a_p)}{2} &> u(p) + \min_{c' \in C \setminus \{a_p\}} \left\{ \frac{d(c', a_p)}{2} \right\} \\ \implies & \; c \text{ kann ausgeschlossen werden.} \end{aligned} \tag{3.10}$$

Die Bedingung ist scharf. Eine beispielhafte Situation findet sich in Abbildung 3.9b. Die doppelte obere Schranke zum Datenpunkt für sich genommen entspricht Pruningkriterium 2. Der Ring des Exponions muss dementsprechend mindestens so groß sein, damit alle Clusterzentren, die potentiell näher als das aktuell zugeordnete Clusterzentrum c_1 am Datenpunkt p liegen, innerhalb des Ringes liegen. Damit mindestens ein weiteres Clusterzentrum innerhalb des Ringes liegt, muss die minimale Distanz zu einem anderen Clusterzentrum zu der doppelten oberen Schranke addiert werden. Diese Distanz wird genau dann erreicht, wenn das dem Zentrum c_1 nächstgelegene Clusterzentrum c_3 auf der Verlängerung der Strecke zwischen p und c_1 liegt. Eine Berücksichtigung der geometrischen Anordnung im Raum erlaubt die Dreiecksungleichung bekanntermaßen nicht. Das tatsächlich zweitnächste Clusterzentrum c_2 liegt genau gegenüber von p und befindet sich infinitesimal näher an p als c_3. Ein formaler Beweis der Korrektheit findet sich in [NF16, SM-B.4.].

Die Identität des zweitnächsten Clusterzentrums ist im Gegensatz zum Annulus für den Exponion nicht von Belang. Im Vergleich zum Hamerly müssen für Pruningkriterium 4 zusätzlich alle paarweisen Center-Center-Distanzen bekannt sein, anstatt nur die betragsmäßig kleinste Distanz pro Zentrum.

In Algorithmus 3.5 ist anhand der Kommentare direkt zu sehen, dass der Exponion im Vergleich zum Hamerly anstatt über alle Clusterzentren C zu iterieren nur noch über die, durch das zusätzliche Pruningkriterium 4 ausschließbaren, Kandidaten *candidates* iterieren und nur für diese eine exakte Distanzberechnung durchführen muss. Weitere Anpassungen sind nicht notwendig, alle notwendigen Informationen stehen bereits zur Verfügung.

Wie auch beim Annulus unterschlägt die Notation in Pseudocode in Zeile 20 einige Details zur effizienten Implementierung. [NF16, Abschnitt 3.1] diskutiert die Möglichkeit und die Schwierigkeiten einer binären Suche zur Ermittlung der Kandidaten. Diese gestaltet sich im Vergleich zum Annulus schwieriger, da die Sortierung der Clusterzentren abhängig vom aktuell zugeordneten Clusterzentrum ist. Die Clusterzentren müssen daher einmal pro Clusterzentrum sortiert werden, entsprechend ergeben sich k Sortiervorgänge. Darüber hinaus kann der CPU-Cache weniger effizient arbeiten, da für jeden Datenpunkt ein nicht vorhersagbares Clusterzentrum das aktuell zugeordnete ist. Entsprechend wird für jeden Datenpunkt eine unterschiedliche Zeile in der Matrix der sortierten Clusterzentren benötigt.

Als Lösung wird in [NF16] vorgeschlagen, dass keine vollständige Sortierung der Clusterzentren durchgeführt wird, sondern die Clusterzentren so angeordnet werden, dass alle Clusterzentren der „Gruppe“ mit Indizes aus dem halboffenen Intervall $[2^n, 2^{n+1})$ weiter entfernt als alle Clusterzentren mit Index kleiner 2^n sind. Innerhalb einer Gruppe ist die Ordnung der Clusterzentren nicht definiert. Jede Gruppe ist dabei doppelt so groß wie die vorherige Gruppe und enthält ein Clusterzentrum mehr als *alle* vorherigen Gruppen zusammen. Zusätzlich werden die Distanz-Grenzen zwischen den Gruppen gespeichert, diese entsprechen jeweils dem größten Eintrag einer Gruppe. Die erste Gruppe, in der alle Clusterzentren außerhalb des Annulus liegen, kann bei dieser Ordnung mit Hilfe einer binären Suche gefunden werden, ohne dass eine vollständige Sortierung erforderlich ist.

Für die beispielhafte Tabelle 3.2 würde die Suche für einen Annulus der Größe 5 die binäre Suche mit einer Prüfung der zweiten Gruppe ($[2, 4)$) beginnen.

Algorithmus 3.5 assignPointsToCluster für einen konkreten Datenpunkt p im Exponion

```
 1: procedure assignPointsToClusterExponion(p)
 2:     nearest ← getAssignedCluster(p)
 3:     if l(p) > u(p) then
 4:         return
 5:     end if
 6:     if cc_g(nearest) · 0.5 > u(p) then
 7:         return
 8:     end if
 9:     u(p) ← d(p, nearest)
10:     if l(p) > u(p) then
11:         return
12:     end if
13:     if cc_g(nearest) · 0.5 > u(p) then
14:         return
15:     end if
16:     nearest ← ⊥
17:     u(p) ← ∞
18:     l(p) ← ∞                          ▷ Bis zu dieser Stelle unverändert
19:     annulus_size ← 2 · u(p) + cc_g(nearest)
20:     candidates ← {c ∈ C | cc(nearest, c) ≤ annulus_size}          ▷
    Zusätzliches Pruning
21:     for all c ∈ candidates do
22:         dist ← d(p, c)                    ▷ Ab dieser Stelle unverändert
23:         if dist < u(p) then
24:             l(p) ← u(p)
25:             nearest ← c
26:             u(p) ← dist
27:         else if dist < l(p) then
28:             l(p) ← dist
29:         end if
30:     end for
31:     assignPointToCluster(p, nearest)
32: end procedure
```

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Gruppe	$[1, 2)$	$[2, 4)$		$[4, 8)$				$[8, 16)$					
Distanz	**1**	**6**	4	7	11	**13**	10	32	20	27	35	23	**40**

Tabelle 3.2.: Beispielhafte Sortierung der Clusterzentren für den Exponion. Die Distanzgrenzen der Gruppen sind fett gedruckt.

Die maximale Distanz in dieser Gruppe beträgt 6. Die binäre Suche fährt in der linken Hälfte fort, da diese Distanz größer als der Annulus ist und alle „rechtsseitigen“ Clusterzentren ausgeschlossen werden können. Entsprechend wird als nächstes die erste Gruppe überprüft. Die maximale Distanz beträgt 1. Diese ist kleiner als der Annulus, es wird in der rechten Hälfte fortgefahren. Es verbleiben keine weiteren Gruppen zur Prüfung, die Grenze liegt in Gruppe zwei. Die Clusterzentren mit Index zwischen 1 und 3 müssen überprüft werden.

Im schlechtesten Fall[12] müssen aufgrund der gewählten Gruppengrößen im Vergleich zu einem exakten Pruning für maximal doppelt so viele Clusterzentren exakte Distanzberechnungen durchgeführt werden.

[NF16] geht allerdings nicht darauf ein, wie die *Erzeugung* der Gruppen in der Praxis aussehen könnte. Es wird lediglich die Suche bei gegebenen Gruppen und die oberhalb vorgestellten Anforderungen an die Gruppen betrachtet.

Genau wie beim Annulus sind auch beim Exponion im Vergleich zum Hamerly bei der Aktualisierung der Schranken keine Änderungen notwendig.

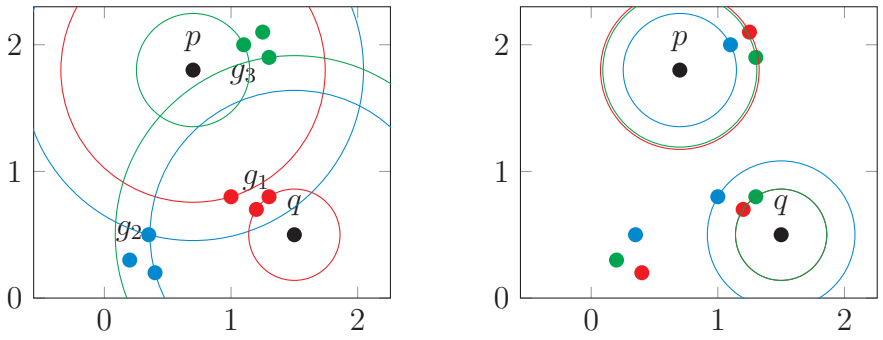
Im Gegensatz zum Annulus gibt es keine „Simplified Exponion“-Variante. Die Berechnung der Center-Center-Distanzen ist essentiell für die Verwendung des zusätzlichen Pruningkriteriums 4. Der Overhead bei der zusätzlichen Verwendung als globales Pruningkriterium ist zu vernachlässigen.

3.6.6. Yinyang

Besonderes Merkmal des in [Din+15] vorgestellten Yinyang-Algorithmus ist, dass dieser die Cluster zu Beginn in Gruppen aufteilt. Jede dieser Gruppen hat *eine* gemeinsame untere Schranke, die sich wie eine Hamerly-Schranke (Seite 24) verhält. Bei einer scharfen Aktualisierung entspricht sie der geringsten Distanz eines Clusters der Gruppe, der nicht der dem Datenpunkt nächstgelegene Cluster ist.

Die Aufteilung der Clusterzentren in Gruppen kann grundsätzlich beliebig erfolgen. Es bietet sich aber an geometrisch benachbarte Cluster der selben Gruppe zuzuweisen, da diese, unter der Annahme, dass sie sich bereits nah an ihrer finalen Position befinden (Abschnitt 3.1), eine ähnliche Größenordnung für die Distanz zu den einzelnen Datenpunkten besitzen (Abbildung 3.10). Bei der guten Aufteilung sind alle Cluster einer Gruppe benachbart und haben daher ähnliche Distanzen zu den Datenpunkten. Die untere Schranke erlaubt

[12] Nur ein Clusterzentrum der letzten Gruppe liegt innerhalb des Annulus.



(a) Gute Aufteilung in Gruppen. **(b)** Schlechte Aufteilung der Gruppen.

Abbildung 3.10.: Gegenüberstellung einer guten und einer schlechten Aufteilung der Gruppen.

ein effektives Pruning. Bei der schlechten Aufteilung sind die Cluster einer Gruppe über den gesamten Raum verteilt. Die unteren Schranken haben fast identische Werte, ein effektives Pruning ist nicht möglich.

[Din+15, Abschnitt 3, Step 1] schlägt dazu vor, die initialen *Clusterzentren* mit Hilfe von k-means für 5 Iterationen zu clustern. Die Anzahl der Cluster wird als $\lceil \frac{k}{10} \rceil$ gewählt, jeder Cluster entspricht einer Gruppe.

Zunächst prüft der Yinyang in Zeile 4 die globale untere Schranke, dem Minimum der unteren Schranken aller Gruppen, gegen die obere Schranke. Diese Prüfung entspricht der Prüfung der unteren Schranke im Hamerly (Algorithmus 3.2, Zeile 3). Nach der scharfen Aktualisierung der oberen Schranke wird diese Prüfung wiederholt, bevor der Yinyang die Kandidatengruppen ermittelt (Zeile 11). Diese Gruppen sind die Gruppen, die nicht über die untere Schranke ausgeschlossen werden können.

Für jede Kandidatengruppen wird die untere Schranke auf den maximalen Wert gesetzt, damit sie bei der Prüfung der einzelnen Clusterzentren nach unten auf die Distanz zwischen Punkt und dem nächstgelegenen[13] Clusterzentrum einer Gruppe korrigiert werden kann.

Es folgt die einzelne Prüfung aller Clusterzentren. Zunächst wird geprüft, ob die Gruppe des Zentrums eine Kandidatengruppe ist (Zeile 20). Anschließend wird geprüft, ob das Clusterzentrum innerhalb seiner Gruppe das nächstgelegene Clusterzentrum sein kann (Zeile 23). Dazu benötigt der Yinyang den Wert der unteren Schranke *vor* der Aktualisierung der Schranken um die maximale Bewegung innerhalb der Gruppe. Von dieser wird die Bewegung des konkret

[13] Ausgenommen das dem Punkt zuzuweisende Clusterzentrum.

Algorithmus 3.6 assignPointsToCluster für einen konkreten Datenpunkt p im Yinyang

```
1: procedure assignPointsToClusterYinyang(p)
2:     nearest ← getAssignedCluster(p)
3:     global_lower ← min {l(p, g) | g ∈ G}
4:     if global_lower > u(p) then ▷ Globale Prüfung der unteren Schranke
5:         return
6:     end if
7:     u(p) ← d(p, nearest)
8:     if global_lower > u(p) then ▷ Wiederholung mit scharfer oberer Schranke
9:         return
10:    end if
11:    candidates ← {g | g ∈ G, l(p, g) ≤ u(p)} ▷ Ermittlung der nicht-prunebaren Gruppen
12:    for all g ∈ candidates do ▷ Startwert für untere Schranke
13:        l(p, g) ← ∞
14:    end for
15:    for all c ∈ C do
16:        if c = nearest then
17:            continue with next c
18:        end if
19:        g ← getGroup(c)
20:        if g ∉ candidates then ▷ Cluster-Pruning, wenn Gruppe gepruned
21:            continue with next c
22:        end if
23:        if old_l(p, g) − δ(c) > l(p, g) then ▷ Pruning innerhalb der Gruppe
24:            continue with next c
25:        end if
26:        dist ← d(p, c)
27:        if dist < u(p) then
28:            l(p, getGroup(nearest)) ← u(p)
29:            nearest ← c
30:            u(p) ← dist
31:        else if dist < l(p, g) then
32:            l(p) ← dist
33:        end if
34:    end for
35:    assignPointToCluster(p, nearest)
36: end procedure
```

betrachteten Clusterzentrums subtrahiert und somit eine untere Schranke für das Clusterzentrum gebildet. Diese wird mit der kleinsten bekannten Distanz der Gruppe zum Punkt verglichen und das Zentrum gepruned, wenn es nicht das nächstgelegene Zentrum der Gruppe ist.

Wenn keine Prüfung ein Pruning erlaubt, dann wird in Zeile 26 die exakte Distanz für das Clusterzentrum berechnet und geprüft, ob das Clusterzentrum entweder das dem Punkt nächstgelegene Zentrum oder das nächstgelegene Zentrum der Gruppe ist.

Die Aktualisierung der Schranken läuft analog zum Hamerly ab. Die obere Schranke wird um die Bewegung des zugeordneten Clusterzentrums korrigiert. Die unteren Schranken werden um die maximale Bewegung aller Clusterzentren einer Gruppe korrigiert.

Für das clusterspezifische Pruning innerhalb des Zuordnungsschritts ist zusätzlich der „unkorrigierte“ Wert der unteren Schranke und die Bewegungen der einzelnen Clusterzentren von Interesse. Damit kein doppelter Speicherbedarf für die untere Schranke anfällt, bietet es sich an, die Korrektur der Schranken in die Neuzuordnung der Datenpunkte zu integrieren. Auf diese Weise müssen jeweils nur die alten unteren Schranken eines einzelnen Datenpunktes vorgehalten werden, eine Reduktion des Speicherbedarfs um den Faktor N.

4

Verwendete Testdatensätze

Zur Vergleichbarkeit mit bestehender Literatur und zur Sicherstellung einer repräsentativen Auswahl soll die Implementierung gegen die in [NF16] verwendeten Datensätzen überprüft werden. Im Gegensatz zu [NF16] wurde aber keine Vorverarbeitung zur statistischen Standardisierung der Daten vorgenommen.

Es war nicht möglich, alle Datensätze eindeutig zu identifizieren und zu beziehen. Mitunter war es gar nicht möglich, den Datensatz zu recherchieren, etwa weil der Name zu unspezifisch war (beispielsweise „mv“ oder „tsn“). In anderen Fallen konnten durch einen hinreichend eindeutigen Namen Datensätze gefunden werden, mitunter bestanden hier aber Abweichungen in Dimension oder Anzahl.

4.1. Beschreibung der Datensätze

Nachfolgend sollen die genutzten Datensätze kurz vorgestellt werden. Die genauen Bezugsquellen, der in dieser Arbeit konkret genutzten Datensätze, finden sich in Anhang A.

birch1

Der birch1-Datensatz ist ein synthetisch generierter 2-dimensionaler Datensatz bestehend aus 100 000 Datenpunkten. Es finden sich 100 natürliche Cluster auf den Eckpunkten eines regelmäßigen Gitters. Er wurde für den „BIRCH“-Clustering-Algorithmus [ZRL97] konstruiert.

In [NF16] wurde dieser Datensatz als „birch“ bezeichnet. Neben dem gitterförmigen Datensatz wurden in [ZRL97] auch sinusförmig verteilte und zufällig verteilte natürliche Cluster untersucht. Darüber hinaus ist unklar, ob der Datensatz in [NF16] auf Basis der Eckdaten in [ZRL97] generiert wurde oder

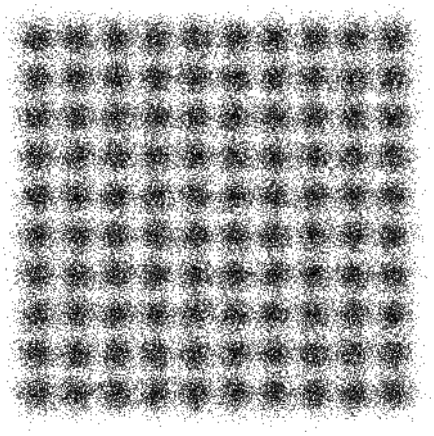


Abbildung 4.1.: Der birch1-Datensatz.

ob ein bestehender Datensatz genutzt wurde. Eine sichere Identifizierung des Datensatzes ist daher nicht möglich.

colormoments

Colormoments ist ein 9-dimensionaler Datensatz generiert aus 68 040 Bildern der „Corel Image Collection“. Die Dimensionen entsprechen dem durchschnittlichen Wert, der Standardabweichung und der statistischen Schiefe der drei Komponenten des HSV-Farbraums [DG17].

Da dieser Datensatz auf realen Testdaten basiert, entspricht er mit an Sicherheit grenzender Wahrscheinlichkeit dem Datensatz in [NF16].

conflongdemo

[RA15] beschreibt den Datensatz als „Data contains recordings of five people performing different activities. Each person wore four sensors (tags) while performing the same scenario five times.“. Die Bezugsquelle [FS18] stellt eine auf drei numerische Attribute reduzierte Version zur Verfügung. Laut der Angaben dort enthält der Datensatz 164 860 Datenpunkte in 11 natürlichen Clustern.

Da dieser Datensatz auf realen Testdaten basiert, entspricht er mit an Sicherheit grenzender Wahrscheinlichkeit dem Datensatz in [NF16].

covtype

Covertype basiert auf Messdaten unterschiedlicher Wälder in den Vereinigten Staaten von Amerika. Insgesamt besteht der Datensatz aus 581 012 Datenpunkten mit 54 Dimensionen. Die Einzelwerte enthalten unter anderem die Erhebung in Metern, die Distanz zur nächsten oberflächlichen Wasserquelle, binäre Werte für verschiedene Eigenschaften des Bodens und die namensgebende Waldbedeckung („Cover Type“) als Ganzzahl zwischen 1 und 7 [DG17].

Da dieser Datensatz auf realen Testdaten basiert, entspricht er mit an Sicherheit grenzender Wahrscheinlichkeit dem Datensatz in [NF16].

house16h

House16h ist ein Datensatz, der zum Test von Algorithmen zur Klassifizierung entwickelt wurde. Er besteht aus 22 784 Datenpunkten mit namensgebenden 16 Dimensionen von 1990 ermittelten Daten des US Census Bureau. Ziel der Klassifizierung ist es, auf Basis dieser Daten, den Median-Preis eines Hauses zu bestimmen. Zur Nutzung mit k-means ist der zu klassifizierende Preis als 17. Dimension Bestandteil des Datensatzes.

Da dieser Datensatz auf realen Testdaten basiert, entspricht er mit an Sicherheit grenzender Wahrscheinlichkeit dem Datensatz in [NF16].

kddcup04

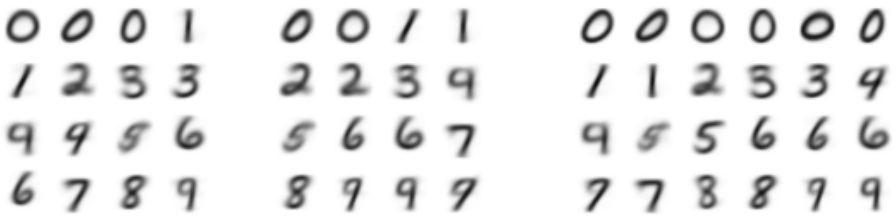
Kddcup04 ist der „Protein Homology“-Trainingsdatensatz des Knowledge Discovery and Data Mining Competition (KDDCUP) aus 2004. Er enthält 145 751 Datenpunkte mit jeweils 74 Dimensionen. Die Dimensionen beschreiben unterschiedliche Eigenschaften von Proteinsequenzen [Elb]. Im Rahmen des KDDCUP soll für diese Sequenzen klassifiziert werden, ob diese homolog[1] sind. Der Datensatz enthält 303 Abstammungslinien, der die einzelnen Datenpunkte zugeordnet sind. Nicht alle Datenpunkte entstammen aber tatsächlich der zugeordneten Abstammungslinie, sondern sind negative Trainingsdaten. Entsprechend sind im Datensatz mindestens 303 natürliche Cluster zu finden. [FS18] gibt $k = 2000$ als Anzahl der natürlichen Cluster an.

Dieser Datensatz basiert auf realen Testdaten und entspricht daher wahrscheinlich dem Datensatz in [NF16]. Der genutzte Datensatz in [NF16] wird

[1] Homologe Proteine haben die gleiche Abstammung.

allerdings mit 145 750 Datenpunkten angegeben und ist somit um einen Datenpunkt kleiner.

mnist784



(a) k = 16 **(b)** k = 16 **(c)** k = 24

Abbildung 4.2.: Mögliche Clusterzentren im mnist784-Datensatz für unterschiedliche k und unterschiedliche Initialisierung.

Der mnist784-Datensatz besteht aus 60 000 Graustufen-Bildern handgeschriebener Ziffern. Die Bilder haben eine Höhe und Breite von jeweils 28 Pixeln, entsprechend kann jedes Bild als Vektor, bestehend aus 784 ganzzahligen Werten aufgefasst werden. Für jede Ziffer und Variante einer Ziffer[2] findet sich ein natürlicher Cluster. Abbildung 4.2 zeigt einige beispielhafte Clusterzentren. Insbesondere die 4 und die 9 sind schwierig auseinander zu halten. Die 0 hingegen ist klar zu erkennen, besitzt aber viele mögliche Formen.

Da dieser Datensatz auf realen Testdaten basiert, entspricht er mit an Sicherheit grenzender Wahrscheinlichkeit dem Datensatz in [NF16].

s1

Der s1-Datensatz ist ein synthetischer 2-dimensionaler Datensatz bestehend aus 5000 Datenpunkten in 15 natürlichen Clustern [FV06].

Dieser Datensatz wurde in [NF16] nicht verwendet. In dieser Arbeit wurde er zur effizienten Sicherstellung der Exaktheit der Implementierung genutzt, da er aufgrund der geringen Größe in jeder Algorithmenvariante schnell geclustert werden kann.

[2] Die 1 wird beispielsweise im Englischen typischerweise als senkrechter Strich geschrieben, während sich im Deutschen der Haken am oberen Ende findet.

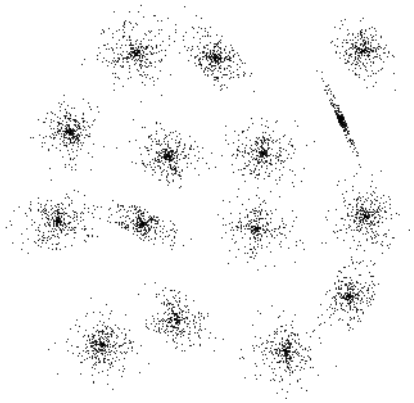


Abbildung 4.3.: Der s1-Datensatz.

uscensus

Der uscensus-Datensatz basiert auf 1 % bei der 21. Volkszählung der Vereinigten Staaten von Amerika erhobenen Daten. Insgesamt enthält der Datensatz 2 458 285 Datenpunkte bestehend aus 68 Dimensionen diskreter Einzelwerte. Diese Einzelwerte sind, unter anderem, das Alter, der Familienstand, die Staatsbürgerschaft, das Geschlecht, die Abstammung und das Einkommen [DG17].

Da dieser Datensatz auf realen Testdaten basiert, entspricht er mit an Sicherheit grenzender Wahrscheinlichkeit dem Datensatz in [NF16].

4.2. Eckdaten der Datensätze

Tabelle 4.1 gibt eine Übersicht über die Eckdaten der genutzten Datensätze. N ist die Anzahl der Datenpunkte, d die Anzahl der Dimensionen und k ist, sofern bekannt, die Anzahl der natürlichen Cluster. Der Typ gibt die Arten der Datentypen in den einzelnen Dimensionen an. Reelle und ganzzahlige Dimensionen sind im Sinne reeller und ganzer Zahlen in der Mathematik zu interpretieren. Distanzen und das arithmetische Mittel sind wohldefiniert. Binäre Dimensionen enthalten einen von zwei möglichen Werten. Diese sind nicht notwendigerweise 0 und 1. Distanzen sind wohldefiniert, das arithmetische Mittel hingegen nicht. Werte von kategorischen Komponenten enthalten einen Wert aus einer zuvor definierten Menge von Zahlen. Die Werte in dieser Menge sind Stellvertreter für die repräsentierte Eigenschaft, wie etwa den Familienstand des uscensus-Datensatzes. Entsprechend sind weder Distanzen noch das arithmetische Mittel wohldefiniert.

Bezeichner	N	d	k	Typ
birch1	100 000	2	100	positiv ganzzahlig
colormoments	68 040	9		reell
conflongdemo	164 860	3	11	reell
covtype	581 012	54		ganzzahlig, binär, kategorisch
house16h	22 784	17		positiv ganzzahlig, positiv reell
kddcup04	145 751	74	≥ 303	reell
mnist784	60 000	784	~ 50	positiv ganzzahlig
s1	5000	2	15	positiv ganzzahlig
uscensus	2 458 285	68		positiv ganzzahlig, binär, kategorisch

Tabelle 4.1.: Eckdaten der genutzten Datensätze.

5 Praktische Umsetzung

Zur empirischen Untersuchung und Beurteilung der Leistung der Algorithmen in Kapitel 6 erfolgte eine praktische Implementierung der in Kapitel 3 beschriebenen Konzepte. Dieses Kapitel motiviert in Abschnitt 5.1 die getroffenen Designentscheidungen der praktischen Implementierung. Abschnitt 5.2 erklärt, wie diese Designentscheidungen zur Implementierung der in Kapitel 3 vorgestellten Konzepten konkret berücksichtigt wurden.

5.1. Designentscheidungen

Grundsätzlich ist die Entwicklung nach dem Maßstab „so simpel wie möglich, so komplex wie notwendig“ erfolgt.

Oberstes Ziel war es, dass die Vergleichbarkeit der Messdaten gewahrt bleibt. Etwaige Änderungen und Optimierungen müssen daher in gleichem Maße Anwendung auf alle Algorithmen und Algorithmenkonfigurationen Anwendung finden.

Unter dieser primären Zielsetzung sollte eine effiziente Implementierung geschaffen werden, um möglichst viele Messdaten innerhalb kurzer Zeit zu erlangen. Auf der anderen Seite soll die Lesbarkeit und Verständlichkeit nicht gefährdet werden. Eine Optimierung, die alle Algorithmen beispielsweise um 5 % beschleunigt und dafür kryptische Anpassungen[1] erfordert, wahrt zwar die Vergleichbarkeit, führt aber zu einer schlechteren Verständlichkeit und erschwert damit eine Nachvollziehbarkeit der Ergebnisse.

Um eine Vergleichbarkeit der Algorithmen zu gewährleisten, wurde Wert darauf gelegt, dass ähnliche oder identische Abläufe in allen Algorithmen identisch implementiert wurden. Die Berechnung der neuen Position eines Clusterzentrums ist lediglich vom Einsatz der Delta Updates (Abschnitt 3.4.2)

[1] Beispielsweise der Einsatz von Inline-Assembler.

abhängig und unterscheidet sich nicht pro Algorithmus. Dementsprechend befindet sich diese in einer Funktion, die von jedem Algorithmus identisch verwendet wird.

Die Struktur der Implementierung der Hauptschleife ist unmittelbar an dem in Kapitel 3 vorgestellten Pseudocode orientiert. Bereits bei der Darstellung der Algorithmen in Pseudocode wurde Wert auf möglichst geringe Änderungen im Vergleich zum Lloyd-Algorithmus gelegt.

Die Implementierung der Algorithmen erfolgte in C++14 auf Basis der Standard Template Library (STL) und ohne Verwendung externer Bibliotheken. C++ liefert eine gute Leistung in Bezug auf Laufzeit und Speicherverbrauch und ist daher eine naheliegende Wahl für Algorithmen, die große Datenmengen verarbeiten sollen.

Wie eingangs genannt soll das resultierende Programm effizient sein, ohne dass der Programmcode übermäßig komplex wird. So ist beispielsweise die Verwendung der C++ Move Semantics [cpp] oder die Verwendung von Referenzen eine naheliegende Optimierung zur Vermeidung von Kopieroperationen. Diese Techniken sollten jedem erfahrenen C++-Programmierer bekannt sein und wurden daher wann immer möglich eingesetzt.

Die manuelle Verwendung von beispielsweise Streaming SIMD Extensions (SSE) hingegen würde zwar alle Algorithmen gleichermaßen beeinflussen, die Verständlichkeit aber gefährden. Entsprechend wurde darauf verzichtet[2], das resultierende Programm ist lediglich etwas langsamer.

Ganz analog liefert die STL alle notwendigen Datenstrukturen, die für die Algorithmen notwendig sind und diese ist jedem erfahrenen C++-Entwickler bekannt. Die Verwendung externer Bibliotheken, etwa einer Bibliothek zur effizienteren Berechnung von Distanzen zwischen zwei Vektoren, erfordert es, dass der Leser mit ebenjener Bibliothek vertraut ist. Entsprechend wurde auch hier der Fokus auf die Verständlichkeit und Kompatibilität gesetzt.

Diese Entscheidung ist unter anderem durch die Referenzimplementierung des Yinyang „yykmeans“ [Din+15] motiviert. yykmeans baut auf dem GraphLab Framework [Low+12] auf. Zur Nachvollziehbarkeit und Reproduktion der Ergebnisse sind Kenntnisse von GraphLab erforderlich.

Die Wahl des Algorithmus, die Wahl der Initialisierung und die Algorithmenkonfiguration müssen zum Zeitpunkt der Kompilierung getroffen werden. Das

[2]Im Idealfall vektorisiert der Compiler die Schleifen selbstständig. Im Falle der Distanzfunktion wurde dies durch Disassemblierung des resultierenden Programms verifiziert.

endgültige ausführbare Programm („Binary“) enthält nur den Maschinencode, der zur Ausführung der gewählten Konfiguration notwendig ist.

Dadurch, dass die Menge der Instruktionen in der Binary kleiner ist, passen potentiell mehr Instruktionen in den CPU-Cache. Die Chance auf Cache-Misses wird reduziert. Weiterhin wird die Anzahl der Verzweigungen („Branches“) reduziert. Anstatt beispielsweise bei jeder Zuweisung prüfen zu müssen, ob Delta Updates (Abschnitt 3.4.2) zum Einsatz kommen sollen, enthält die Binary entweder nur den Code für Delta Updates oder nur den Code für die explizite Speicherung der Zuordnungen.

Die Erfassung von Messdaten, wie der Anzahl durchgeführter Distanzberechnungen, kann auf diese Weise ebenfalls vollständig entfernt werden, wenn Speicherverbrauch und Laufzeit extern gemessen werden. Letzteres kann beispielsweise durch die Verwendung des Programms GNU „time“ geschehen.

Die Entscheidung ist darin begründet, dass eine derartige Flexibilität zur Ausführungszeit zwar für wissenschaftliche Untersuchungen hilfreich sein mag, diese in einem realen Einsatz aber nicht notwendig ist und die Leistung senkt.

5.2. Konkrete Umsetzung der Designentscheidungen

Wie in Abschnitt 5.1 motiviert, ist die Umsetzung nahe an dem vorgestellten Pseudocode orientiert, etwaige Schleifen und Bedingungen finden sich in gleicher Form im C++-Quellcode wieder.

Identische Funktionsweise aller Algorithmen ist in eine gemeinsame abstrakte Basisklasse `Algorithm` ausgelagert worden von der alle Algorithmen erben. Diese Basisklasse enthält etwa die Methode zur Neuzuweisung eines Datenpunktes zu einem Clusterzentrum, Methoden, die den aktuellen Zustand zurückliefern und eine Methode `cluster()`, die die von den abgeleiteten Klassen zu implementierende Methode `round()` so lange aufruft, bis der Algorithmus konvergiert. Sie nimmt ebenfalls die zu clusternden Datenpunkte und die Positionen der initialen Clusterzentren entgegen. Letztere werden vorab durch einen `Initialization`-Algorithmus ausgesucht, zur Verfügung steht k-means++ (Abschnitt 3.1) und ein naiver Algorithmus, der die ersten k Punkte des Datensatzes als initiale Clusterzentren wählt.

Die abgeleiteten Klassen implementieren `round()`, ergänzen benötigte Klassenattribute zur Speicherung der verwendeten Schranken und anderer Metadaten und erweitern den Konstruktor um die algorithmenspezifische Initialisierung.

Die Wahl der eingesetzten Datenstrukturen ist, wann immer möglich, auf `std::vector` gefallen. Insbesondere dann, wenn die Datenstruktur nur einmalig aufgebaut wird und anschließend lediglich die Werte der einzelnen Elemente verändert werden müssen, bietet ein `vector` einen sehr schnellen Zugriff mit geringem Speicheroverhead. Dies ist beispielsweise bei der Speicherung der Schranken der Fall. Es muss einmalig eine Schranke pro Datenpunkt im `vector` gespeichert werden, anschließend wird dieses Element nur noch modifiziert. Wann immer die Anzahl der zu speichernden Elemente bekannt war, wurde der benötigte Speicher im `vector` unmittelbar reserviert, damit beim Einfügen von Elementen keine aufwändigen Vergrößerungen des Speicherbereichs notwendig sind.

Falls die Verwendung eines `vector` nicht sinnvoll möglich war, etwa um die Zugehörigkeit zu einer Menge dynamisch zu speichern, dann wurde ein `std::unordered_set` genutzt, um Elemente effizient entfernen zu können.

Zur Speicherung mehrerer zusammengehöriger Werte, wie beispielsweise die Identität und die Distanz bei der Ermittlung des Clusterzentrums, das sich am weitesten bewegt hat, wird zur Vermeidung von Fehlern auf eine Struktur gesetzt. Die Zusammngehörigkeit der Werte ist auf diese Weise unmittelbar ersichtlich. Indem zur Aktualisierung jeweils die komplette Struktur ersetzt wird, ist es unwahrscheinlich, dass die Einzelwerte zueinander inkonsistent sind. Im häufigsten Fall, der Speicherung von zwei Werten, ist ein `std::pair` die Struktur der Wahl.

Die Realisierung der Konfiguration bei Kompilierung erfolgt durch die Verwendung von C++-Preprocessor-Direktiven. Nicht gewünschte Funktionalität wird auf diese Weise vollständig entfernt oder ersetzt.

Nebenläufigkeit kommt im Interesse der Verständlichkeit des Quellcodes nicht zum Einsatz. Wie in Abschnitt 3.4.3 diskutiert, ist bei Verwendung von mehreren Threads zu erwarten, dass alle Algorithmen gleichermaßen beschleunigt werden. Der Einsatz von Nebenläufigkeit erfordert aber ein großes Maß an Sorgfältigkeit zur Vermeidung von subtilen Fehlern und dem effizienten Einsatz der trotz guter Parallelisierbarkeit notwendigen Sperren.

Verifikation

Zur Verifikation der Exaktheit kann konfiguriert werden, dass nach jeder Iteration für jeden Cluster die Anzahl der zugeordneten Datenpunkte und ein Hashwert dieser Datenpunkte ausgegeben werden soll. Diese Ausgabe kann anschließend mit der Ausgabe des Lloyd-Algorithmus als Referenzwert verglichen werden. Der Lloyd ist, gemäß Definition 3, der Maßstab für die Exaktheit eines beschleunigten k-means-Algorithmus.

Die Ausgabe verwendet einen Hashwert, um zu vermeiden, dass bei größeren Datensätzen in jeder Iteration mehrere Megabyte Text ausgegeben und potentiell in eine Datei geschrieben werden müssen. Die Gefahr von Kollisionen ist zu vernachlässigen, da ein Fehler in der Regel dazu führt, dass zu viele Clusterzentren ausgeschlossen werden und der Algorithmus daher zu früh konvergiert. Im Falle von subtilen Abweichungen, etwa durch eine andere Sortierung bei Clusterzentren mit gleicher Distanz, bietet die eingesetze Hashfunktion eine hinreichende Kollisionssicherheit, insbesondere, da der Algorithmus mit unterschiedlichen Datensätzen verifiziert wird.

Darüber hinaus werden `asserts` zur Sicherstellung der Korrektheit und zur erleichterten Fehlerdiagnose im Falle eines inkorrekten Resultats eingesetzt. So wird beispielsweise geprüft, dass spätere Pruningkriterien nicht alle Clusterzentren ausschließen, wenn dies bereits durch ein früheres Pruningkriterium hätte erkannt werden müssen. Ein weiteres Beispiel ist die Binärsuche für die Ermittlung der Clusterzentren im Annulus. Es wird durch ein `assert` sichergestellt, dass das Clusterzentrum, das vor dem ersten ermittelten Clusterzentrum in der sortierten Liste steht, nicht im Annulus liegt.

k-means++

Zur Vergleichbarkeit der Ergebnisse der unterschiedlichen Algorithmen wird der k-means++-Initialisierung ein Zufallsgenerator (Random Number Engine) übergeben, der mit einem auf der Kommandozeile zu übergebenen Seed initialisiert wird. Auf diese Weise kann sicher gestellt werden, dass alle Algorithmen trotz der probabilistischen Natur von k-means++ die identischen initialen Clusterzentren nutzen und somit gleich viele Neuzuweisungen und Iterationen benötigen.

Lloyd

Für den Lloyd-Algorithmus ist bei der Implementierung nichts zu beachten gewesen. Die Implementierung besteht im wesentlichen aus zwei Schleifen, die für jeden Punkt und jedes Clusterzentrum prüfen, welches Zentrum dem Punkt an nächsten gelegen ist.

Elkan

Der Elkan folgt dem Pseudocode sehr nahe. Wie eingangs erwähnt wird zur Speicherung der oberen und unteren Schranken jeweils ein `std::vector` genutzt. Auch die ermittelten Center-Center-Distanzen werden für die Iteration in einem `std::vector` gespeichert. Die zwei Parameterdimensionen werden auf die eine Dimension des `std::vector` mit Hilfe einer „row-major“-Speicherung abgelegt.

Die „Simplified Elkan“-Variante ist dadurch realisiert, dass die Verwendung der Center-Center-Distanzen mit Hilfe des C++-Preprocessors entfernt werden kann.

Hamerly

Die Hinweise zum Elkan sind nahezu unverändert auf den Hamerly anwendbar. Ein Unterschied ist, dass die Center-Center-Distanzen nicht mehr paarweise benötigt werden, sondern lediglich die geringste Distanz pro Zentrum. Entsprechend sind die Center-Center-Distanzen bereits eindimensional.

Drake

Im Vergleich zum Pseudocode 3.3 enthält die Implementierung des Drake, wie in Abschnitt 3.6.3 genannt, zusätzlich noch eine Speicherung der maximalen Anzahl an benötigten Schranken (z) innerhalb einer Iteration. In der nächsten Iteration wird dieser Wert genutzt, um die Prüfung der Pruningmöglichkeit früher abzubrechen.

Die Berechnung der exakten Distanzen erfolgt im Rahmen der Ermittlung der „Kandidaten“. Die exakte Distanz wird zusammen mit dem Kandidaten in einem `std::pair` gespeichert und der `std::vector` der Kandidaten anschließend auf Basis dieser Distanz sortiert. Anschließend enthält der Vektor die

z nächstgelegenen Clusterzentren in der Reihenfolge ihrer Distanz und kann unmittelbar genutzt werden, um die unteren Schranken zu aktualisieren.

Die Aktualisierung der unteren Schranke erfolgt, wie in Abschnitt 3.6.3 vorgeschlagen, in umgekehrter Reihenfolge, um die aufsteigende Ordnung ohne zusätzlichen Rechenaufwand sicherzustellen.

Annulus

Der Annulus ermittelt zu Beginn einmalig die Norm aller Datenpunkte. Die Implementierung von `round()` erfolgt anschließend identisch zum Hamerly, mit der Ergänzung des zusätzlichen Pruningkriteriums 3.

Neben der Möglichkeit zur Deaktivierung der binären Suche erlaubt es die im Rahmen dieser Arbeit erstellte Implementierung die Auswahl des Fixpunktes zu konfigurieren. [Dra13] legt als Fixpunkt den Ursprung des Koordinatensystems fest. Alternative Fixpunkte werden im Rahmen von [Dra13] nicht untersucht. Die Position des Fixpunktes hat über die Distanz zu den Datenpunkten Einfluss auf die Fläche des Annulus und daher auf die Pruningleistung. Um den Einfluss der Position zu untersuchen, bietet die Implementierung die Möglichkeit den Fixpunkt als Schwerpunkt (Mean) aller Datenpunkte zu wählen. Eine weitere Alternative ist die Wahl des Fixpunktes als das komponentenweise Minimum aller Datenpunkte. Im zweidimensionalen Fall befindet sich der Fixpunkt somit in der linken unteren Ecke des minimal umgebenden Rechtecks (Bounding Box).

Exponion

Analog zum Annulus erweitert der Exponion ebenfalls den Hamerly. Im Unterschied zum Hamerly sind hier, wie beim Elkan, die paarweisen Center-Center-Distanzen und nicht nur die geringste Distanz pro Zentrum notwendig. Entsprechend verwendet der Exponion eine Implementierung analog zum Elkan.

Die in [NF16] genannte binäre Suche zur Ermittlung der Kandidaten ist aus den in Abschnitt 3.6.5 genannten Gründen nicht umgesetzt. Das ELKI Framework [SZ19] enthält eine Implementierung des Exponion in Java. Auch diese ist bezüglich der binären Suche unvollständig. Die Referenzimplementierung des [NF16] ist diesbezüglich ebenfalls nicht hilfreich, da sie eine geringe Kommentardichte enthält, genutzte Bezeichner häufig Abkürzungen enthal-

ten und die Implementierung in komplexen Klassenhierarchien organisiert ist. Entsprechend ist es schwierig, ein angemessenes Verständnis für die Implementierung zu entwickeln, um den im Paper genannten Vorschlag zur binären Suche nachvollziehen und insbesondere auch die Korrektheit prüfen zu können.

Yinyang

Der Yinyang weicht in der Implementierung am stärksten von den anderen Algorithmen ab. Eine Besonderheit ist hier, dass die Aktualisierung der Schranken, wie in Abschnitt 3.6.6 vorgeschlagen, während jeder Iteration erfolgt und nicht nach der Iteration. Dies ist darin begründet, dass sowohl der Betrag der Schranken vor der Aktualisierung, als auch der Betrag der Schranken nach der Aktualisierung zum Pruning benötigt wird. Indem die Aktualisierung Bestandteil der jeweiligen Iteration ist, müssen lediglich die Werte eines einzelnen Datenpunktes doppelt vorgehalten werden.

Das Clustering der initialen Clusterzentren zur Bestimmung der Gruppenzusammensetzung erfolgt mit Hilfe des Lloyd-Algorithmus. Die Initialisierung des Lloyd-Algorithmus erfolgt auf naive Weise. Es werden die ersten $\lceil \frac{k}{10} \rceil$ Clusterzentren als initiale Clusterzentren gewählt. Wenn der Yinyang selbst durch k-means++ initialisiert wurde, dann ist auch mit dieser naive Initialisierung eine gute Verteilung zu erwarten, da die zuerst gewählten Clusterzentren bei k-means++ weit voneinander entfernt sind und somit nicht innerhalb der gleichen Gruppe sein sollten.

Es erfolgten zwei Implementierungen von Yinyang, die wir als „yinyang“ und „yinyang2“ bezeichnen. Die erste Implementierung („yinyang“) erfolgte rein auf Basis der Beschreibung in [Din+15]. Diese Implementierung enthält eine fehlerhafte Aktualisierung der unteren Schranke, die durch eine Unklarheit in der textuellen Beschreibung entstanden ist. Anstatt das zweitnächste Clusterzentrum *pro* Gruppe separat scharf zu aktualisieren, wird das global betrachtete zweitnächste Clusterzentrum genutzt, um die untere Schranke der nicht-prunebaren Gruppen zu aktualisieren. Die separate untere Schranke ist dadurch lediglich im Falle eines erfolgreichen Prunings von Relevanz, in allen anderen Fällen degeneriert der „yinyang“ zu einem Hamerly mit deutlich mehr Overhead. Entsprechend liefert der „yinyang“ eine Leistung vergleichbar mit dem Lloyd. In vielen Fällen sogar eine schlechtere Leistung.

Der „yinyang2“ ist eine Implementierung unter Sichtung der Referenzimplementierung. In diesem wird korrekt über das nächstgelegene Clusterzentrum jeder Gruppe Buch geführt und dieses zur Aktualisierung der unteren Schranke genutzt. Entsprechend sind die Schranken im Vergleich zum „yinyang“ deutlich schärfer und es wird eine gute Leistung erzielt.

Der „yinyang“ ist im Sinne der Vollständigkeit enthalten, enthält aber einen Hinweis auf das Missverständnis von [Din+15].

Aktualisierung der Schranken

Zur Aktualisierung der Schranken müssen die Bewegungen der Clusterzentren bekannt sein. Dazu wird die aktuelle Position der Clusterzentren zu Beginn jeder Iteration abgespeichert. Nach der Verschiebung in den Schwerpunkt kann für jedes Clusterzentrum die Distanz zu seiner alten Position berechnet und die Schranken entsprechend nach unten oder nach oben korrigiert werden.

Zur Realisierung von „Norm Of Sums“ wird für jede Schranke neben der Distanz zusätzlich ein Integer gespeichert. Dieser gibt die Iteration der letzten exakten Aktualisierung an. Bei der Korrektur der Schranken wird die Position des Clusterzentrums dann nicht mit der zuletzt bekannten Position verglichen, sondern mit der Position zu der jeweiligen exakten Aktualisierung.

Delta Updates

Delta Updates sind wie in Abschnitt 3.4.2 diskutiert umgesetzt. Statt pro Clusterzentrum ein `std::unordered_set` mit den zugeordneten Punkten zu speichern, wird ein `std::pair` bestehend aus der Vektorsumme aller zugeordneten Datenpunkte und der Anzahl der zugeordneten Datenpunkte gespeichert. Zur Aktualisierung der Position des Clusterzentrums wird die Vektorsumme durch die Anzahl geteilt, um das arithmetische Mittel zu erhalten.

6 Empirische Untersuchung

Dieses abschließende Kapitel untersucht die Leistung der Algorithmen im praktischen Einsatz unter Verwendung der in Kapitel 5 entwickelten Implementierung und der in Kapitel 4 vorgestellten Datensätze. Um eine Reproduktion und Überprüfung der Ergebnisse zu ermöglichen, beginnen wir mit einer umfassenden Erklärung der Methodik und Testumgebung in Abschnitt 6.1. Nach einer Vorstellung der verwendeten Beurteilungskriterien in Abschnitt 6.2 folgt in Abschnitt 6.3 die eigentliche Auswertung der gewonnenen Messdaten.

6.1. Methodik

Durch die Implementierung von sieben Algorithmen[1] mit sieben Featureflags, die sich teilweise gegenseitig ausschließen und nicht immer auf jeden Algorithmus anwendbar sind, und zwei möglichen Initialisierungen[2] ergeben sich insgesamt 156 Varianten von durch Schranken beschleunigten k-means-Algorithmen[3]. Listing 6.1 zeigt die genaue Zusammenstellung der Algorithmen mit ihren jeweiligen Featureflags als Auszug des Python-Skripts zur Generierung des `Makefile`s.

Für den praktischen Test sind lediglich die Varianten mit k-means++ als Initialisierungsmethode zum Einsatz gekommen. Die naive Initialisierung ist nicht repräsentativ für einen realen Einsatz und findet lediglich im Yinyang zur Initialisierung der Gruppen Anwendung. Ebenfalls nicht genutzt wurden die Varianten *ohne* den Einsatz von Delta Updates. Letztere waren in Vorabtests konsequent und deutlich schneller als die explizite Speicherung der Zuordnung von Cluster zu Datenpunkt. Eine detaillierte Untersuchung würde keine über-

[1] Acht unter Berücksichtigung des fehlerhaften „yinyang“
[2] Naiv und k-means++
[3] 156 inkludiert die vier nicht-beschleunigten Lloyd-Varianten

```
class Flags(Enum):
  NsBound = ("USE_NS_BOUND", "ns")
  RollingSum = ("USE_ROLLING_SUM", "rolling")
  SimplifiedElkan = ("SIMPLIFIED_ELKAN", "simplified")
  SimplifiedHamerly = ("SIMPLIFIED_HAMERLY", "simplified")
  AnnulusNoBinarySearch = ("ANNULUS_NO_BSEARCH", "noBsearch")
  AnnulusOriginMean = ("ANNULUS_ORIGIN_MEAN", "originMean")
  AnnulusOriginEdge = ("ANNULUS_ORIGIN_EDGE", "originEdge")

class Algorithms(Enum):
  Annulus = Algorithm(
    "ANNULUS",
    [
      Flags.NsBound,
      Flags.RollingSum,
      Flags.SimplifiedHamerly,
      Flags.AnnulusNoBinarySearch,
      [ Flags.AnnulusOriginMean, Flags.AnnulusOriginEdge ],
    ],
  )
  Drake = Algorithm("DRAKE", [Flags.NsBound, Flags.RollingSum])
  Elkan = Algorithm("ELKAN", [Flags.NsBound, Flags.RollingSum, Flags.
      SimplifiedElkan])
  Exponion = Algorithm("EXPONION", [Flags.NsBound, Flags.RollingSum])
  Hamerly = Algorithm(
    "HAMERLY", [Flags.NsBound, Flags.RollingSum, Flags.SimplifiedHamerly]
  )
  Lloyd = Algorithm("LLOYD", [Flags.RollingSum])
  # YinYang = Algorithm("YINYANG", [Flags.RollingSum])
  YinYang2 = Algorithm("YINYANG2", [Flags.NsBound, Flags.RollingSum])

class Initialization(Enum):
  Naive = "NAIVE"
  KmeansPP = "KMEANSPP"
```

Listing 6.1: Die möglichen Varianten ergeben sich als kartesisches Produkt aus Initialisierung, Algorithmus und den Featureflags des jeweiligen Algorithmus.

raschenden Ergebnisse zu Tage führen und wäre eine Verschwendung in Form von Rechenleistung und Strom.

Insgesamt verbleiben 39 Varianten von denen mit 24 der Großteil aufgrund der großen Anzahl an unterstützten Featureflags auf den Annulus entfällt.

Diese wurden unter Deaktivierung der `asserts` (`-DNDEBUG=1`) und mit aktivierter Erhebung von Statistiken (`-DENABLE_STATS=1`) mit clang++ 3.8 kompiliert. Der vollständige Compiler-Aufruf findet sich in Listing 6.2.

```
clang++ -Isrc -std=c++14 -O2 -Wall -pedantic -DENABLE_STATS=1 -DNDEBUG=1
```

Listing 6.2: Verwendete Kompilierungsparameter

Zur Erhebung des maximalen Hauptspeicherverbrauchs wurde die Erhebung der Statistiken deaktiviert, das `-DENABLE_STATS=1` entfällt.

Die Tests erfolgten unter Verwendung eines KVM-virtualisierten Cloudservers mit 4 dedizierten CPU-Kernen, 16 GiB Hauptspeicher und NVM Express (NVMe)-Festspeicher. Die CPU-Kerne werden innerhalb des Cloudservers als Intel Xeon der Skylake-Architektur mit 2100 MHz Taktfrequenz angegeben (Listing 6.3), die im physischen Wirt tatsächlich verwendete CPU ist nicht bekannt.

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel Xeon Processor (Skylake, IBRS)
stepping        : 4
microcode       : 0x1
cpu MHz         : 2100.000
cache size      : 16384 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
    pat pse36 clflush mmx fxsr sse sse2 ht syscall nx pdpe1gb rdtscp lm
    constant_tsc rep_good nopl xtopology pni pclmulqdq ssse3 fma cx16 pcid sse4_1
    sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
    hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb kaiser
    fsgsbase bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx avx512f avx512dq rdseed
    adx smap clwb avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 arat md_clear
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
bogomips        : 4200.00
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

Listing 6.3: Ausgabe von /proc/cpuinfo für einen CPU-Kern des virtualisierten Testsystems.

Das eingesetzte Betriebssystem war ein minimales Debian GNU/Linux 9.9 („Stretch“) mit Linux 4.9. Neben grundlegenden Systemdiensten und dem OpenSSH-Dienst lief zu jedem Zeitpunkt lediglich eine der 39 Binarys zum k-means-Clustering. Entsprechend stand die komplette Leistung des Systems dieser Binary zur Verfügung, eine Verfälschung des Messergebnisses durch etwaige Kontextwechsel oder von anderen Prozessen angeforderte Rechenzeit wird dadurch minimiert. Von den 4 dedizierten CPU-Kernen wurde aufgrund der nicht implementierten Nebenläufigkeit nur einer genutzt. Die Wahl eines Modells mit weniger CPU-Kernen war nicht möglich, da einige Algorithmen

zum Clustern der größeren Datensätze mehr als die 8 GiB Hauptspeicher des nächstkleineren Modells benötigten.

Die während des Clusterings erfassten Messdaten werden durch die Binary im JSON Lines-Format [War] über die Standardausgabe (`std::cout`) ausgegeben. Diese Ausgabe wurde zur Weiterverarbeitung in eine Datei umgeleitet. Aufgrund der geringen Datenmengen und des schnellen Festspeichers ist keine Beeinflussing des Messergebnisses zu erwarten gewesen.

Bedingt durch die Leistungsfähigkeit der Hardware, den verfügbaren Hauptspeicher und die zur Verfügung stehende Zeit wurde die Anzahl von Clustern in Schritten von 16 zwischen 16 und 96 gewählt. Eine maximale Anzahl von Clustern über 96 hätte mehr Arbeitsspeicher erfordert[4]. Eine kleinere Schrittgröße hätte die benötigte Laufzeit ohne zu erwartenden Kenntnisgewinn vervielfacht.

Auf gleiche Weise wurden pro Datensatz und Clusteranzahl nur jeweils fünf unterschiedliche Initialisierungen mit k-means++ getestet. Diese Anzahl erlaubt eine Mittelung der resultierenden Werte mit einem angemessenen Zeitaufwand.

Ein vollständiges Clustering aller Datensätze für alle Clusteranzahlen, Initialisierungen und Algorithmenvarianten benötigte mit diesen Beschränkungen etwa zehn Tage.

6.2. Beurteilungskriterien

Primäres Kriterium zur Beurteilung der Leistung der einzelnen Varianten ist die Realzeit („Wall Clock Time“) und dadurch implizit auch die CPU-Zeit. Die Realzeit ist der *nach außen sichtbare* Effekt der Beschleunigung von k-means. Eine reduzierte Realzeit bedeutet, dass Ergebnisse schneller zur Verfügung stehen und dass Hardware eingespart werden kann.

Sekundäres Kriterium ist die Anzahl der Distanzberechnungen. Die Anzahl der Distanzberechnungen ist *der* Bestandteil von k-means, der sich algorithmisch verbessern lässt. Der Lloyd verbringt den Großteil der Rechenzeit mit der Berechnung von Distanzen. Die benötigte Rechenzeit für eine einzelne Distanzberechnung steigt etwa linear mit der Anzahl der Dimensionen (Tabelle 6.1). Der Overhead der vorgestellten Pruningkriterien hingegen ist konstant in der Anzahl der Dimensionen, da für einzelne Punkte jeweils nur zwei Gleitkommawerte ver-

[4] Für die 2-dimensionalen Datensätze waren auch Clusteranzahlen bis 1024 mit der zur Verfügung stehenden Hardware möglich.

glichen werden. Die zusätzlichen Distanzberechnungen, wie etwa zur Ermittlung der Zentrenbewegungen, haben die gleiche Dimension wie andere Distanzberechnungen und können gegen die eingesparten Distanzberechnungen aufgerechnet werden. Somit dominiert die Berechnung der Distanzen mit steigender Anzahl an Dimensionen das Laufzeitverhalten. Die Anzahl der Distanzberechnungen ist dementsprechend ein Indikator, aber nicht das entscheidende Kriterium für die benötigte Realzeit. Eine Einsparung von Distanzberechnungen ist nicht unmittelbar für den Benutzer sichtbar.

Der benötigte Hauptspeicher schränkt die Auswahl der möglichen Varianten abhängig vom Zielsystem möglicherweise ein, wenn der Speicher nicht ausreichend ist, um die genutzten Datenstrukturen vollständig aufzunehmen. Dieser *sollte* daher auch Bestandteil der Beurteilung sein. Eine verlässliche externe Messung stellte sich in der Praxis allerdings als schwierig heraus. Beispielsweise hatte der Lloyd in einigen Fällen eine größere maximale Resident Set Size (RSS) als beschleunigte Algorithmen, obwohl der Lloyd keine zusätzlichen Daten speichert und alle beschleunigten Algorithmen mindestens obere und untere Schranken speichern. Aus diesem Grund ist der benötigte Hauptspeicher hier nicht Teil der Beurteilung.

Zur Ermittlung des Anteils der Distanzberechnungen an der Gesamtlaufzeit wurde die Dauer einer einzelnen Distanzberechnung auf dem zuvor vorgestellten Testsystem bestimmt. Um den Einfluss des Messfehlers gering zu halten, wurden jeweils 50 000 000 Distanzberechnungen durchgeführt und die Gesamtzeit durch die Anzahl der Distanzberechnungen geteilt. Die Ergebnisse finden sich in Tabelle 6.1.

In niedrigen Dimensionen dominiert der Overhead des Funktionsaufrufs, die benötigte Zeit pro Dimension nimmt daher anfangs ab. In höheren Dimensionen sind mehr Hauptspeicherzugriffe notwendig, die Zeit pro Dimension nimmt daher wieder zu. Auffällig ist der Sprung der Laufzeit pro Dimension bei exakt 32 Dimensionen. Ein Test mit dem Programm `perf` auf einem nicht-virtualisierten System mit Intel Core i5 der Sandy Bridge-Architektur legt nahe, dass diese Anomalie durch schlechte Sprungvorhersage (englisch „Branchprediction“) verursacht wird (Listing 6.4). Eine Verifizierung auf dem Testsystem ist nicht möglich, die entsprechenden Daten stehen durch die Virtualisierung nicht zur Verfügung.

d	Zeit pro Distanz	Zeit pro Dimension
2	5,35 ns	2,67 ns
4	6,82 ns	1,70 ns
8	11,12 ns	1,39 ns
16	19,59 ns	1,22 ns
31	36,52 ns	1,18 ns
32	42,36 ns	1,32 ns
33	39,05 ns	1,18 ns
64	75,92 ns	1,19 ns
128	165,21 ns	1,29 ns
256	338,01 ns	1,32 ns
512	681,56 ns	1,33 ns
1024	1366,72 ns	1,33 ns
2	4,71 ns	2,35 ns
3	5,95 ns	1,98 ns
9	11,84 ns	1,32 ns
17	21,08 ns	1,24 ns
54	63,38 ns	1,17 ns
68	83,41 ns	1,23 ns
74	93,60 ns	1,26 ns
784	1039,61 ns	1,33 ns

Tabelle 6.1.: Benötigte Zeit für eine Distanzberechnung abhängig von der Anzahl der Dimensionen d. Die Anzahl der Dimensionen ist einmal als Zweierpotenzen und einmal als die Größe der tatsächlich verwendeten Datensätze gewählt.

```
 Performance counter stats for 'target/benchmark 31':

[...]
 3,397,612,451  branches        # 1482.261 M/sec           (83.43%)
        64,148  branch-misses   #    0.00% of all branches (83.33%)

   2.293598268 seconds time elapsed

 Performance counter stats for 'target/benchmark 32':

[...]
 3,504,524,873  branches        # 1436.719 M/sec           (83.45%)
    49,411,428  branch-misses   #    1.41% of all branches (83.33%)

   2.442455416 seconds time elapsed

 Performance counter stats for 'target/benchmark 33':

[...]
 3,602,623,395  branches        # 1545.174 M/sec           (83.36%)
        40,967  branch-misses   #    0.00% of all branches (83.24%)

   2.331939048 seconds time elapsed
```

Listing 6.4: Schlechte Sprungvorhersage bei 32 Dimensionen.

6.3. Beurteilung

Wir beginnen die Auswertung der Messdaten mit einer Sicht von außen und vergleichen die Algorithmenvarianten *als Ganzes gegeneinander* anhand der zuvor vorgestellten Beurteilungskriterien. Ziel soll es sein, interessante Ansätze für die nachfolgende Untersuchung der Algorithmen zu finden. Darüber hinaus soll, falls möglich, abhängig von Datensatz und Anzahl der gewünschten Cluster, eine Empfehlung für die Wahl einer Algorithmenvariante gegeben werden. Als Vorbereitung für die Beurteilung der Leistung der Einzelkomponenten innerhalb einer Algorithmenvariante erfolgt nach dieser Gegenüberstellung der Algorithmen eine Untersuchung des Clusteringverlaufs für die einzelnen Datensätze, um beispielsweise das Verhältnis der aktiven und statischen Cluster (Definition 5) zu bestimmen. Abschließend wird die „Blackbox“ geöffnet, um zu untersuchen, welche Einzelkomponenten zu welchem Anteil die Leistung der Algorithmen beeinflussen. Ziel soll es sein, die zu Beginn gewonnenen Ergebnisse besser erklären zu können und Ansätze zur weiteren Verbesserung der Leistung zu finden.

6.3.1. Vergleich der Algorithmen

Zum Vergleich der Algorithmen führen wir den Begriff der Dominanz ein.

Definition 8 (Dominanz)
Eine Algorithmenvariante A dominiert eine Algorithmenvariante B, wenn A konsequent eine bessere Leistung als B liefert.

Zunächst betrachten wir die Dominierungen unter Verwendung der Anzahl der Distanzberechnungen als Kriterium.

Ein Blick auf Tabelle 6.2 zeigt, dass die Situation dort eindeutig ist. Lediglich drei Varianten des Elkan sind undominiert. Der reguläre Elkan und der Elkan unter Verwendung von Norm of Sums dominieren nahezu alle anderen Algorithmen und Algorithmenvarianten. Dies ist wenig überraschend, die große Anzahl an unteren Schranken des Elkan erlaubt eine feingranulare Aktualisierung der Schranken nach einer Bewegung der Clusterzentren.

Auffällig ist hingegen, dass der Simplified Elkan mit Norm of Sums undominiert ist. Der Einsatz eines zusätzlichen Pruningkriteriums sollte intuitiv entweder die Anzahl der Distanzberechnungen reduzieren oder, falls es zu schwach ist, keine Veränderung bewirken. Entsprechend ist zu erwarten, dass der reguläre Elkan mit Norm of Sums diesen dominiert. Ein näherer Blick zeigt, dass diese Auffälligkeit nur beim Clustern des mnist784-Datensatzes auftritt. Dort hat sie aber einen nicht unerheblichen Einfluss. Beim Clustern des mnist784 in 96 Cluster benötigte der Simplified Elkan in einigen Fällen über 4 % weniger Distanzberechnungen als der reguläre Elkan. Die Ursache ist der konstante Overhead von k^2 Distanzberechnungen zur Ermittlung der Center-Center-Distanzen. Pruningkriterium 2 kann zwar sowohl in Zeile 18 als auch in Zeile 28 von Algorithmus 3.1 Distanzen einsparen. Die Anzahl der eingesparten Distanzen ist allerdings geringer als der konstante Overhead von k^2. Wie wir bei der detaillierten Analyse des Elkan (Seite 97) feststellen werden, ist der Unterschied zwischen Elkan und Simplified Elkan in Bezug auf Distanzberechnungen aber generell kleiner als bei einer naiven Betrachtung der Leistung von Pruningkriterium 2 anzunehmen wäre.

In hohen Dimensionen sollte die Wahl daher immer auf eine Elkanvariante fallen. Dies bestätigt sich mit einem Blick auf die Dominierungen mit Realzeit als Vergleichskriterium beim Clustering des mnist784-Datensatzes (Tabelle 6.3) und entspricht den Erkenntnissen der aktuellen Literatur [Din+15, Abschnitt 5].

Dieser besitzt mit 784 Dimensionen unter den verwendeten Testdatensätzen die meisten Dimensionen. Auch hier dominieren die Elkanvarianten alle anderen Algorithmen. Die Simplified-Variante des Elkan erbringt, wie zuvor diskutiert,

Algorithmus	Dominierte	Dominiert von
ELKAN_ns	36	**0**
ELKAN	35	**0**
ELKAN_ns_simplified	21	**0**
ELKAN_simplified	21	1
ANNULUS_ns_noBsearch_originEdge	8	2
EXPONION_ns	8	2
ANNULUS_ns_simplified_noBsearch_originEdge	8	4
ANNULUS_ns_originEdge	7	3
ANNULUS_ns_simplified_originEdge	7	5
ANNULUS_ns_noBsearch	6	2
ANNULUS_ns_simplified_noBsearch	6	4
ANNULUS_ns	5	3
ANNULUS_ns_simplified	5	5
ANNULUS_noBsearch_originEdge	4	2
YINYANG2_ns	4	4
ANNULUS_ns_noBsearch_originMean	4	4
ANNULUS_simplified_noBsearch_originEdge	4	6
ANNULUS_ns_simplified_noBsearch_originMean	4	6
ANNULUS_originEdge	3	3
DRAKE_ns	3	4
DRAKE	3	4
ANNULUS_ns_originMean	3	5
YINYANG2	3	5
HAMERLY_ns	3	5
ANNULUS_simplified_originEdge	3	7
ANNULUS_ns_simplified_originMean	3	7
ANNULUS_noBsearch	2	2
EXPONION	2	2
ANNULUS_simplified_noBsearch	2	6
HAMERLY_ns_simplified	2	8
ANNULUS_noBsearch_originMean	2	9
ANNULUS_simplified_noBsearch_originMean	2	19
ANNULUS	1	3
ANNULUS_simplified	1	7
HAMERLY	1	7
ANNULUS_originMean	1	10
HAMERLY_simplified	1	11
ANNULUS_simplified_originMean	1	20
LLOYD	0	38

Tabelle 6.2.: Dominierungen unter Verwendung der Anzahl der Distanzberechnungen als Kriterium.

Algorithmus	Dominierte	Dominiert von
ELKAN_ns_simplified	37	**0**
ELKAN_ns	35	**0**
ELKAN	35	1
ELKAN_simplified	35	1
YINYANG2_ns	32	4
DRAKE_ns	31	4
DRAKE	31	4
YINYANG2	31	5
HAMERLY_ns_simplified	17	8
HAMERLY_ns	16	8
EXPONION_ns	16	8
ANNULUS_ns_simplified_noBsearch	4	8
ANNULUS_ns_simplified_originMean	4	8
ANNULUS_ns_simplified_originEdge	4	8
ANNULUS_ns_simplified	4	8
ANNULUS_ns_noBsearch_originMean	4	8
ANNULUS_ns_originEdge	4	8
ANNULUS_ns_simplified_noBsearch_originMean	4	8
ANNULUS_ns_simplified_noBsearch_originEdge	4	8
ANNULUS_ns_noBsearch_originEdge	4	8
ANNULUS_ns_noBsearch	4	8
ANNULUS_ns_originMean	4	9
ANNULUS_ns	3	8
HAMERLY_simplified	2	11
HAMERLY	1	11
EXPONION	1	11
ANNULUS_simplified_originEdge	1	11
ANNULUS_simplified_noBsearch_originMean	1	11
ANNULUS_simplified	1	11
ANNULUS_simplified_noBsearch_originEdge	1	11
ANNULUS_originEdge	1	11
ANNULUS_simplified_originMean	1	11
ANNULUS_simplified_noBsearch	1	11
ANNULUS_noBsearch_originMean	1	11
ANNULUS_originMean	1	12
ANNULUS_noBsearch_originEdge	1	22
ANNULUS	1	23
ANNULUS_noBsearch	1	23
LLOYD	0	38

Tabelle 6.3.: Dominierungen unter Verwendung der Realzeit für mnist784.

die bessere Leistung in Bezug auf Distanzberechnungen. In Verbindung mit dem geringeren Overhead wird so auch eine bessere Leistung in Bezug auf die Realzeit erreicht.

Bei den niedrigdimensionalen Datensätzen[5] in Tabelle 6.4 ist die Situation weniger eindeutig. Praktisch alle Varianten von Hamerly, Annulus und Exponion sind undominiert. Dies ändert sich auch nicht, wenn die Annulus-Varianten mit geändertem Ursprung unberücksichtigt bleiben. Umgekehrt sind aber alle Algorithmen und Varianten, die nicht auf dem Hamerly basieren, dominiert. Dies entspricht den Aussagen der jeweiligen Originalveröffentlichungen von Annulus und Exponion [Dra13; NF16].

Der Hamerly selbst ist nur im Falle des s1 mit 16 Clustern undominiert. Dies entspricht den Erwartungen. Annulus und Exponion besitzen jeweils nur kleine Anpassungen, die speziell den schlechten Fall des Hamerly optimieren sollen. Im Durchschnitt beträgt die Differenz zwischen Hamerly und Exponion für den s1 mit 16 Clustern 70 µs, ein Unterschied von ungefähr 2 % im Vergleich zur durchschnittlichen Clusteringzeit des Hamerly (3320,9405 µs). Darüber hinaus ist der Hamerly nur für zwei von fünf Initialisierungen besser als der Exponion. Für die anderen drei Initialisierungen ist entweder der Exponion schneller oder gleichauf. Falls diese geringe Differenz in der Clusteringzeit nicht ausschließlich auf Messungenauigkeiten zurück zu führen ist, ist diese dennoch so klein, dass diese keinen relevanten Effekt auf einen Einsatz in der Praxis hätte. Beispielsweise benötigt das Lesen eines 4 kB-Blocks von einer Solid-state Drive (SSD) bereits 150 µs und das Lesen von 1 MB Daten aus dem Hauptspeicher 250 µs [Bon]. Die Dauer des Einlesens des Datensatzes dominiert die Laufzeitdifferenz also deutlich.

Die durchschnittlichen Realzeiten bis zur Konvergenz (Tabellen 6.5 und 6.6) geben etwas mehr Aufschluss über die Leistung der undominierten Varianten. Der Simplified Annulus erzielt insgesamt die beste Leistung. Im Falle des conflongdemo ist dieser für die *durchschnittliche* Zeit in allen getesteten Dimensionen der beste. Er dominiert die anderen Algorithmen aber nicht, da er in *Einzelfällen* bis zu 55 % langsamer als andere Varianten war. Für den birch-Datensatz erzielt der Simplified Annulus bei großem k eine bis zu 50 % schlechtere Leistung.

[5] 2 und 3 Dimensionen

Algorithmus	Dominierte	Dominiert von
EXPONION	10	**0**
ANNULUS_ns_simplified	9	**0**
ANNULUS_simplified_originEdge	9	**0**
ANNULUS_ns_simplified_originMean	9	**0**
ANNULUS_noBsearch_originEdge	9	**0**
ANNULUS_simplified	8	**0**
ANNULUS_ns_originEdge	8	**0**
ANNULUS	8	**0**
ANNULUS_ns_originMean	8	**0**
ANNULUS_originMean	8	**0**
ANNULUS_ns_noBsearch_originEdge	8	**0**
ANNULUS_noBsearch	8	**0**
ANNULUS_ns_noBsearch	8	**0**
EXPONION_ns	8	**0**
ANNULUS_ns_simplified_originEdge	7	**0**
ANNULUS_ns	7	**0**
ANNULUS_originEdge	7	**0**
ANNULUS_noBsearch_originMean	7	**0**
ANNULUS_ns_simplified_noBsearch	7	**0**
ANNULUS_ns_simplified_noBsearch_originEdge	7	**0**
ANNULUS_simplified_noBsearch	7	**0**
ANNULUS_simplified_noBsearch_originEdge	7	**0**
HAMERLY_ns	7	**0**
HAMERLY	7	**0**
ANNULUS_ns_noBsearch_originMean	7	1
ANNULUS_ns_simplified_noBsearch_originMean	7	1
ANNULUS_simplified_originMean	6	**0**
ANNULUS_simplified_noBsearch_originMean	6	1
HAMERLY_ns_simplified	5	**0**
YINYANG2	5	3
HAMERLY_simplified	4	**0**
YINYANG2_ns	4	14
ELKAN	3	27
DRAKE	1	30
ELKAN_simplified	1	32
DRAKE_ns	0	30
LLOYD	0	31
ELKAN_ns	0	32
ELKAN_ns_simplified	0	35

Tabelle 6.4.: Dominierungen für niedrigdimensionale Datensätze (s1 / birch / conflongdemo).

Algorithmus	16	32	48	64	80	96
ANN	0,38 s (1.19)	0,40 s (1.12)	0,62 s (1.14)	0,64 s (1.14)	0,45 s (1.02)	0,36 s (1)
ANN_noBs	0,38 s (1.2)	0,41 s (1.16)	0,68 s (1.23)	0,71 s (1.26)	0,52 s (1.18)	0,42 s (1.19)
ANN_noBs_oEdge	0,38 s (1.21)	0,41 s (1.16)	0,68 s (1.23)	0,72 s (1.27)	0,52 s (1.17)	0,44 s (1.25)
ANN_noBs_oMean	0,41 s (1.29)	0,45 s (1.27)	0,77 s (1.4)	0,81 s (1.44)	0,61 s (1.38)	0,50 s (1.4)
ANN_ns	0,41 s (1.29)	0,41 s (1.15)	0,64 s (1.16)	0,65 s (1.15)	0,46 s (1.04)	0,36 s (1.03)
ANN_ns_noBs	0,42 s (1.31)	0,43 s (1.22)	0,71 s (1.3)	0,74 s (1.31)	0,53 s (1.21)	0,43 s (1.21)
ANN_ns_noBs_oEdge	0,41 s (1.3)	0,44 s (1.23)	0,71 s (1.29)	0,74 s (1.31)	0,54 s (1.21)	0,43 s (1.23)
ANN_ns_noBs_oMean	0,44 s (1.38)	0,48 s (1.34)	0,78 s (1.42)	0,84 s (1.48)	0,63 s (1.43)	0,51 s (1.44)
ANN_ns_oEdge	0,41 s (1.28)	0,42 s (1.17)	0,65 s (1.18)	0,65 s (1.14)	0,45 s (1.03)	0,36 s (1.03)
ANN_ns_oMean	0,43 s (1.35)	0,45 s (1.26)	0,71 s (1.29)	0,72 s (1.27)	0,51 s (1.15)	0,41 s (1.15)
ANN_ns_simp	0,35 s (1.09)	0,38 s (1.07)	0,58 s (1.06)	0,59 s (1.04)	0,48 s (1.08)	0,44 s (1.24)
ANN_ns_simp_noBs	0,35 s (1.09)	0,41 s (1.14)	0,66 s (1.2)	0,71 s (1.26)	0,60 s (1.36)	0,57 s (1.62)
ANN_ns_simp_noBs_oEdge	0,35 s (1.11)	0,41 s (1.15)	0,66 s (1.2)	0,70 s (1.24)	0,61 s (1.38)	0,59 s (1.66)
ANN_ns_simp_noBs_oMean	0,38 s (1.18)	0,46 s (1.29)	0,76 s (1.39)	0,84 s (1.48)	0,74 s (1.69)	0,75 s (2.12)
ANN_ns_simp_oEdge	0,34 s (1.08)	0,38 s (1.06)	0,57 s (1.05)	0,60 s (1.06)	0,47 s (1.06)	0,44 s (1.25)
ANN_ns_simp_oMean	0,36 s (1.14)	0,43 s (1.2)	0,67 s (1.22)	0,70 s (1.24)	0,57 s (1.28)	0,55 s (1.55)
ANN_oEdge	0,38 s (1.2)	0,39 s (1.1)	0,61 s (1.11)	0,63 s (1.11)	0,44 s **(1)**	0,35 s **(1)**
ANN_oMean	0,41 s (1.29)	0,44 s (1.24)	0,69 s (1.26)	0,72 s (1.27)	0,51 s (1.16)	0,40 s (1.14)
ANN_simp	0,32 s (1.01)	0,36 s **(1)**	0,55 s (1)	0,57 s **(1)**	0,47 s (1.06)	0,44 s (1.23)
ANN_simp_noBs	0,33 s (1.05)	0,40 s (1.13)	0,65 s (1.17)	0,69 s (1.22)	0,60 s (1.37)	0,60 s (1.69)
ANN_simp_noBs_oEdge	0,33 s (1.04)	0,41 s (1.15)	0,65 s (1.18)	0,70 s (1.24)	0,60 s (1.37)	0,61 s (1.72)
ANN_simp_noBs_oMean	0,36 s (1.12)	0,47 s (1.32)	0,79 s (1.44)	0,90 s (1.59)	0,83 s (1.87)	0,85 s (2.4)
ANN_simp_oEdge	0,32 s **(1)**	0,36 s (1.02)	0,55 s **(1)**	0,57 s (1)	0,47 s (1.06)	0,44 s (1.24)
ANN_simp_oMean	0,34 s (1.08)	0,42 s (1.17)	0,65 s (1.18)	0,68 s (1.2)	0,56 s (1.28)	0,54 s (1.54)
DRAKE	0,93 s (2.92)	1,45 s (4.09)	2,49 s (4.53)	3,36 s (5.93)	2,61 s (5.91)	2,53 s (7.15)
DRAKE_ns	1,25 s (3.91)	1,99 s (5.6)	3,58 s (6.52)	3,92 s (6.92)	3,00 s (6.79)	2,64 s (7.46)
ELKAN	0,99 s (3.11)	1,54 s (4.34)	3,20 s (5.83)	3,94 s (6.95)	2,79 s (6.32)	2,11 s (5.96)
ELKAN_ns	1,45 s (4.56)	2,34 s (6.59)	5,12 s (9.32)	6,39 s (11.29)	4,44 s (10.07)	3,44 s (9.73)
ELKAN_ns_simp	1,53 s (4.81)	2,48 s (6.98)	5,54 s (10.07)	7,15 s (12.62)	5,49 s (12.45)	4,90 s (13.86)
ELKAN_simp	1,12 s (3.53)	1,76 s (4.95)	3,76 s (6.85)	4,77 s (8.42)	3,85 s (8.73)	3,39 s (9.57)
EXPONION	0,32 s (1)	0,36 s (1.01)	0,60 s (1.09)	0,64 s (1.14)	0,49 s (1.12)	0,42 s (1.18)
EXPONION_ns	0,34 s (1.08)	0,39 s (1.09)	0,63 s (1.14)	0,67 s (1.18)	0,52 s (1.18)	0,44 s (1.24)
HAMERLY	0,40 s (1.26)	0,48 s (1.36)	0,84 s (1.54)	0,92 s (1.63)	0,71 s (1.61)	0,59 s (1.66)
HAMERLY_ns	0,43 s (1.36)	0,50 s (1.42)	0,85 s (1.55)	0,94 s (1.66)	0,71 s (1.61)	0,58 s (1.64)
HAMERLY_ns_simp	0,37 s (1.18)	0,51 s (1.44)	0,86 s (1.57)	0,98 s (1.73)	0,89 s (2.02)	0,92 s (2.6)
HAMERLY_simp	0,35 s (1.1)	0,50 s (1.42)	0,88 s (1.6)	1,01 s (1.79)	0,95 s (2.15)	0,98 s (2.77)
LLOYD	1,59 s (5.01)	2,67 s (7.5)	5,82 s (10.58)	7,37 s (13)	5,48 s (12.43)	4,84 s (13.69)
YINYANG2	0,44 s (1.38)	0,53 s (1.49)	0,88 s (1.6)	0,97 s (1.71)	0,82 s (1.86)	0,72 s (2.03)
YINYANG2_ns	0,49 s (1.55)	0,64 s (1.79)	1,06 s (1.93)	1,24 s (2.19)	1,02 s (2.31)	0,88 s (2.5)

Tabelle 6.5.: Durchschnittliche Zeiten zum Clustering des birch-Datensatzes. In Klammern ist der Faktor im Vergleich zur besten Zeit angegeben.

Algorithmus	16	32	48	64	80	96
ANN	0,68 s (1.17)	1,13 s (1.11)	1,54 s (1.07)	1,88 s (1.1)	2,33 s (1.02)	2,24 s (1.02)
ANN_noBs	0,71 s (1.22)	1,29 s (1.26)	1,92 s (1.33)	2,40 s (1.4)	3,24 s (1.42)	3,21 s (1.46)
ANN_noBs_oEdge	0,72 s (1.24)	1,35 s (1.32)	2,06 s (1.43)	2,60 s (1.52)	3,55 s (1.56)	3,55 s (1.62)
ANN_noBs_oMean	0,75 s (1.28)	1,43 s (1.4)	2,19 s (1.52)	2,78 s (1.62)	3,91 s (1.71)	3,89 s (1.77)
ANN_ns	0,72 s (1.23)	1,15 s (1.13)	1,60 s (1.11)	1,93 s (1.13)	2,42 s (1.06)	2,28 s (1.04)
ANN_ns_noBs	0,75 s (1.28)	1,34 s (1.31)	2,00 s (1.39)	2,47 s (1.44)	3,34 s (1.47)	3,28 s (1.5)
ANN_ns_noBs_oEdge	0,77 s (1.32)	1,41 s (1.38)	2,06 s (1.43)	2,61 s (1.52)	3,54 s (1.55)	3,52 s (1.61)
ANN_ns_noBs_oMean	0,79 s (1.34)	1,47 s (1.44)	2,24 s (1.56)	2,83 s (1.65)	3,92 s (1.72)	3,95 s (1.8)
ANN_ns_oEdge	0,73 s (1.25)	1,19 s (1.17)	1,65 s (1.15)	2,01 s (1.17)	2,54 s (1.12)	2,43 s (1.11)
ANN_ns_oMean	0,75 s (1.28)	1,25 s (1.22)	1,79 s (1.24)	2,16 s (1.26)	2,78 s (1.22)	2,64 s (1.21)
ANN_ns_simp	0,63 s (1.08)	1,06 s (1.04)	1,49 s (1.04)	1,78 s (1.04)	2,34 s (1.03)	2,29 s (1.04)
ANN_ns_simp_noBs	0,66 s (1.12)	1,23 s (1.21)	1,90 s (1.32)	2,38 s (1.39)	3,31 s (1.45)	3,35 s (1.53)
ANN_ns_simp_noBs_oEdge	0,67 s (1.15)	1,31 s (1.29)	2,01 s (1.39)	2,56 s (1.49)	3,65 s (1.6)	3,76 s (1.71)
ANN_ns_simp_noBs_oMean	0,70 s (1.2)	1,40 s (1.37)	2,22 s (1.54)	2,84 s (1.66)	4,14 s (1.82)	4,29 s (1.96)
ANN_ns_simp_oEdge	0,66 s (1.12)	1,09 s (1.07)	1,58 s (1.09)	1,88 s (1.1)	2,48 s (1.09)	2,43 s (1.11)
ANN_ns_simp_oMean	0,67 s (1.15)	1,20 s (1.17)	1,73 s (1.2)	2,08 s (1.21)	2,79 s (1.22)	2,73 s (1.25)
ANN_oEdge	0,69 s (1.17)	1,15 s (1.13)	1,61 s (1.12)	1,94 s (1.13)	2,51 s (1.1)	2,37 s (1.08)
ANN_oMean	0,73 s (1.24)	1,23 s (1.2)	1,78 s (1.24)	2,12 s (1.24)	2,77 s (1.22)	2,63 s (1.2)
ANN_simp	0,59 s **(1)**	1,02 s **(1)**	1,44 s **(1)**	1,71 s **(1)**	2,28 s **(1)**	2,19 s **(1)**
ANN_simp_noBs	0,62 s (1.06)	1,23 s (1.21)	1,91 s (1.33)	2,36 s (1.38)	3,43 s (1.51)	3,44 s (1.57)
ANN_simp_noBs_oEdge	0,65 s (1.11)	1,30 s (1.28)	2,06 s (1.43)	2,55 s (1.49)	3,71 s (1.63)	3,79 s (1.73)
ANN_simp_noBs_oMean	0,66 s (1.14)	1,38 s (1.36)	2,25 s (1.56)	2,84 s (1.66)	4,17 s (1.83)	4,31 s (1.97)
ANN_simp_oEdge	0,60 s (1.03)	1,08 s (1.06)	1,56 s (1.08)	1,84 s (1.08)	2,48 s (1.09)	2,42 s (1.11)
ANN_simp_oMean	0,63 s (1.07)	1,16 s (1.14)	1,71 s (1.19)	2,03 s (1.18)	2,78 s (1.22)	2,73 s (1.24)
DRAKE	1,52 s (2.6)	3,48 s (3.41)	6,22 s (4.32)	9,40 s (5.48)	11,94 s (5.24)	12,52 s (5.71)
DRAKE_ns	1,99 s (3.4)	4,66 s (4.57)	7,53 s (5.23)	10,86 s (6.34)	13,31 s (5.84)	13,52 s (6.17)
ELKAN	1,76 s (3.01)	4,11 s (4.03)	7,23 s (5.02)	11,05 s (6.45)	14,81 s (6.5)	14,78 s (6.74)
ELKAN_ns	2,37 s (4.06)	5,66 s (5.55)	10,37 s (7.2)	16,01 s (9.34)	21,09 s (9.26)	21,23 s (9.69)
ELKAN_ns_simp	2,10 s (3.59)	4,74 s (4.65)	8,79 s (6.1)	13,92 s (8.12)	18,55 s (8.15)	19,08 s (8.7)
ELKAN_simp	1,57 s (2.68)	3,40 s (3.34)	6,05 s (4.2)	9,43 s (5.5)	12,87 s (5.65)	12,97 s (5.92)
EXPONION	0,62 s (1.07)	1,31 s (1.29)	2,07 s (1.44)	2,54 s (1.48)	3,55 s (1.56)	3,46 s (1.58)
EXPONION_ns	0,66 s (1.13)	1,38 s (1.35)	2,08 s (1.45)	2,60 s (1.51)	3,58 s (1.57)	3,52 s (1.6)
HAMERLY	0,72 s (1.22)	1,48 s (1.45)	2,33 s (1.62)	3,03 s (1.77)	4,39 s (1.93)	4,42 s (2.02)
HAMERLY_ns	0,76 s (1.3)	1,53 s (1.5)	2,39 s (1.66)	3,05 s (1.78)	4,42 s (1.94)	4,50 s (2.05)
HAMERLY_ns_simp	0,69 s (1.17)	1,52 s (1.49)	2,48 s (1.72)	3,17 s (1.85)	4,84 s (2.12)	5,09 s (2.32)
HAMERLY_simp	0,65 s (1.1)	1,46 s (1.44)	2,44 s (1.69)	3,14 s (1.83)	4,79 s (2.1)	5,06 s (2.31)
LLOYD	2,43 s (4.16)	5,68 s (5.57)	10,60 s (7.35)	16,64 s (9.71)	22,07 s (9.69)	22,67 s (10.34)
YINYANG2	0,75 s (1.29)	1,44 s (1.41)	2,26 s (1.57)	2,83 s (1.65)	3,83 s (1.68)	3,86 s (1.76)
YINYANG2_ns	0,82 s (1.4)	1,64 s (1.61)	2,52 s (1.75)	3,27 s (1.91)	4,48 s (1.97)	4,49 s (2.05)

Tabelle 6.6.: Durchschnittliche Zeiten zum Clustering des conflongdemo-Datensatzes. In Klammern ist der Faktor im Vergleich zur besten Zeit angegeben.

In der Blackbox-Beurteilung verbleiben die mittleren Dimensionen[6]. Für diese ist die Situation noch weniger eindeutig, als für die niedrigen Dimensionen. Varianten des Annulus erreichen auch hier regelmäßig sehr gute Leistungen. Eine pauschalisierte Empfehlung des Annulus wäre hier allerdings nicht angemessen. Ein Blick auf die Entwicklung der Rangliste für steigende k (Abbildungen 6.1, 6.2 und 6.4 bis 6.6) und die Rangliste für steigende Größen des zu clusternden Datensatzes (Abbildung 6.8) zeigt interessante Trends.

Zunächst fällt auf, dass der Exponion für $k = 16$ oftmals auf den niedrigen Plätzen zu finden ist, mit zunehmendem k aber an Leistung einbüßt. Dies ist wohl in der fehlenden Implementierung der binären Suche der Kandidatenzentren begründet. Abbildung 6.3 zeigt beispielhaft, dass der Exponion für den house16h-Datensatz in Bezug auf Distanzberechnungen eine gleichbleibend gute Pruningleistung relativ zu den anderen Algorithmen erreicht. Daraus folgt, dass der Anteil des Overheads angestiegen ist.

Der zweite interessante Trend ist, dass der Yinyang mit zunehmender Größe der zu bearbeitenden Daten an Leistung gewinnt. Zu bearbeitende Daten ist als Produkt aus Clusteranzahl k, Dimension d und Anzahl der Datenpunkte N zu verstehen. Dieser Effekt ist besonders ausgeprägt für eine steigende Clusteranzahl zu beobachten. In allen Abbildungen 6.1 bis 6.7 ist zu sehen, dass der Rang des Yinyang mit zunehmender Anzahl von Clustern steigt. Insbesondere ist der Yinyang der einzige Algorithmus, der seine Platzierung in Bezug auf Distanzberechnungen deutlich steigern konnte. Alle anderen Algorithmen und Varianten behalten relativ zueinander ihre Platzierungen. Beispielhaft ist dies für den kddcup04-Datensatz in Abbildung 6.7 zu sehen. Analog tritt dies auch für andere Datensätze auf. Diese Beobachtung deckt sich mit den Aussagen der Originalveröffentlichung des Yinyang [Din+15, Abschnitt 5]. Bei einer geringen Anzahl von Clustern arbeitet der Yinyang mit einer geringen Anzahl von Gruppen und ist entsprechend stark von Big Movern betroffen. Mit zunehmender Anzahl an Clustern nimmt die Anzahl der Gruppen proportional zu, sodass bei einer gleichbleibenden Menge von Big Movern ein geringerer Anteil von Clustern durch diese betroffen ist. Darüber hinaus sorgt die Gruppierung räumlich benachbarter Cluster dafür, dass sich Paare von aktiven Clustern[7] wahrscheinlich in der gleichen Gruppe befinden,

[6] 9 bis 74 bei unseren Datensätzen

[7] Ein Paar von aktiven Clustern ist so zu verstehen, dass ein Datenpunkt von einem der beiden Cluster in den anderen wechselt.

da Punkte ihre Zuordnung in der Regel nur zwischen unmittelbar benachbarten Clustern bewegen. Umgekehrt folgt daraus, dass die Wahrscheinlichkeit sinkt, dass sich aktive Cluster in Gruppen aus statischen Clustern befinden. Wenn eine Gruppe ausschließlich aus statischen Clustern besteht, dann ist keine Anpassung der Schranke erforderlich. Für Cluster in statischen Gruppen wird eine gute Pruningleistung erzielt.

Abbildung 6.8 zeigt ein ähnliches, aber weniger ausgeprägtes Verhalten, wenn man die Anzahl der Datensätze nach dem Produkt von Dimension und Anzahl der Datenpunkte sortiert.

- This area intentionally left blank -

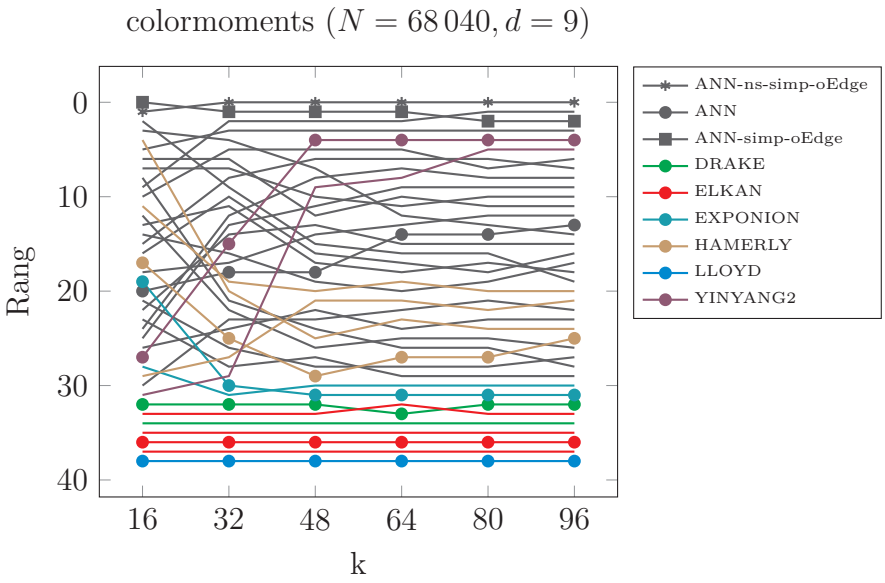


Abbildung 6.1.: Rangliste der durchschnittlichen Zeiten zum Clustering des colormoments-Datensatzes.

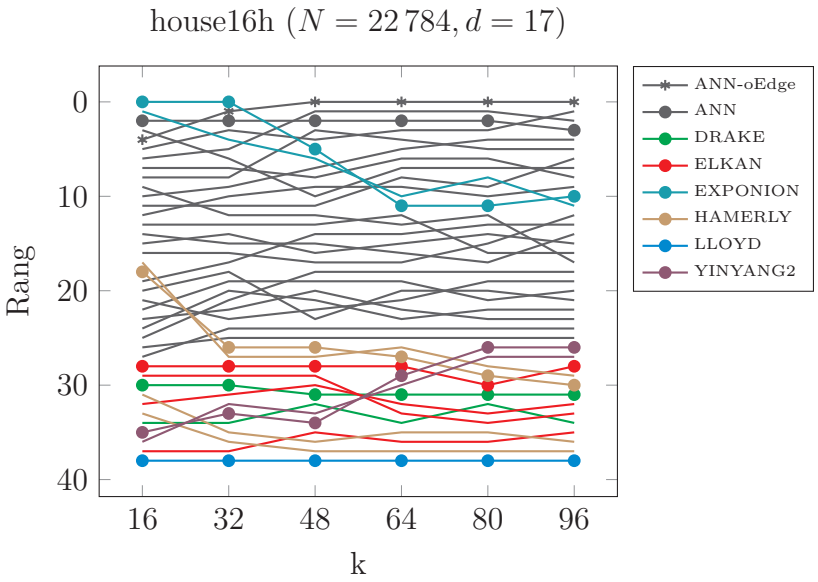


Abbildung 6.2.: Rangliste der durchschnittlichen Zeiten zum Clustering des house16h-Datensatzes.

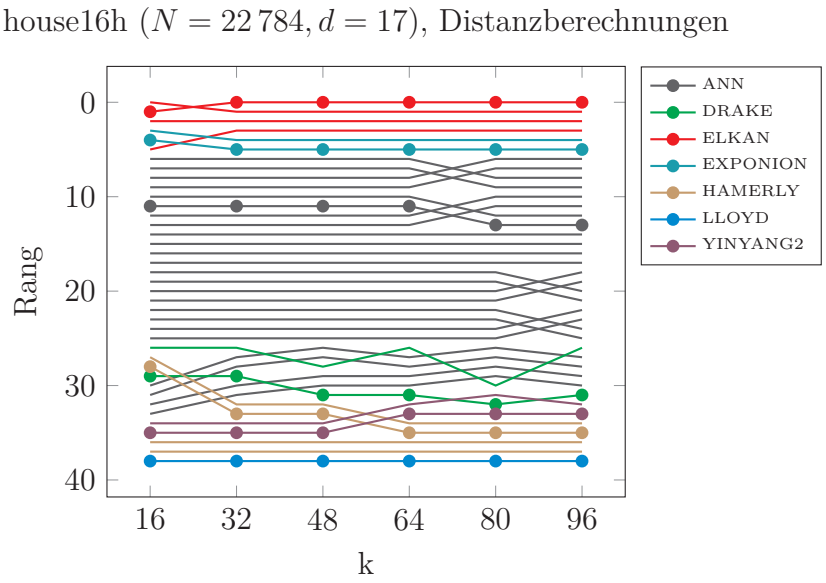


Abbildung 6.3.: Rangliste der durchschnittlichen Distanzberechnungen zum Clustering des house16h-Datensatzes.

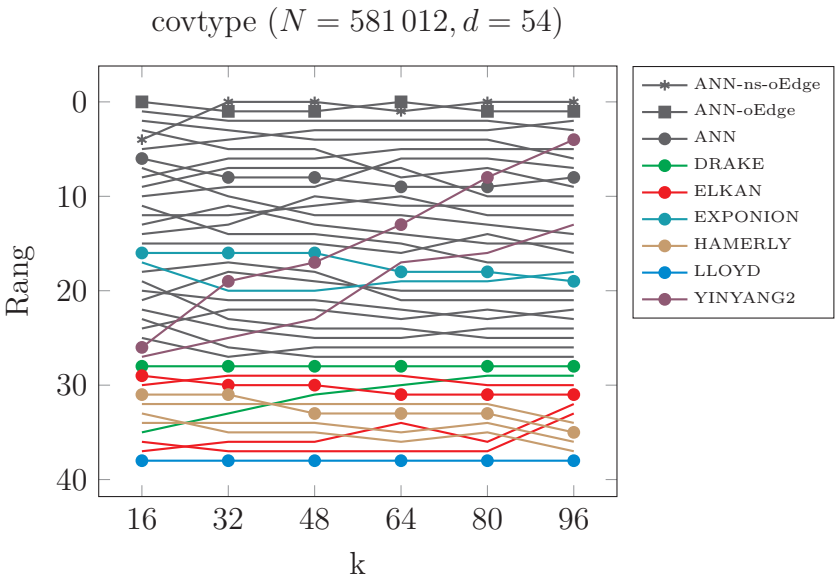


Abbildung 6.4.: Rangliste der durchschnittlichen Zeiten zum Clustering des covtype-Datensatzes.

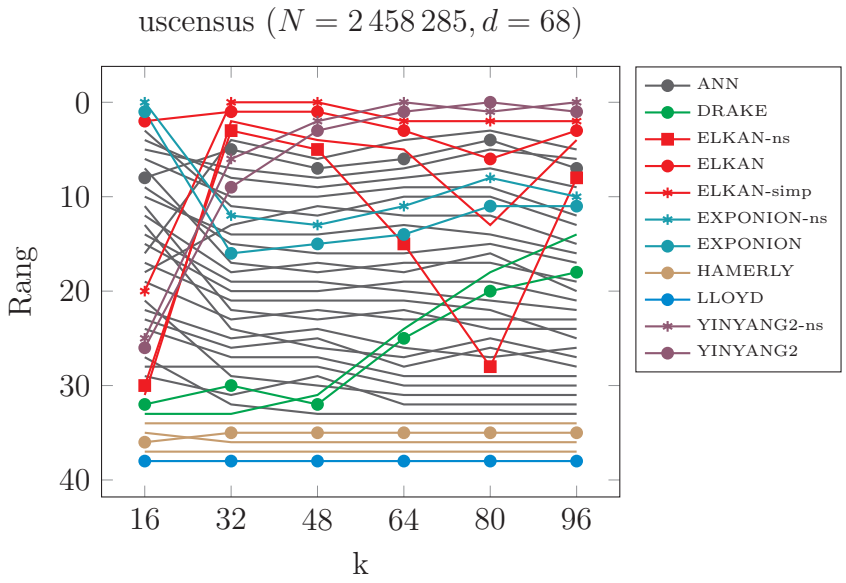


Abbildung 6.5.: Rangliste der durchschnittlichen Zeiten zum Clustering des uscensus-Datensatzes.

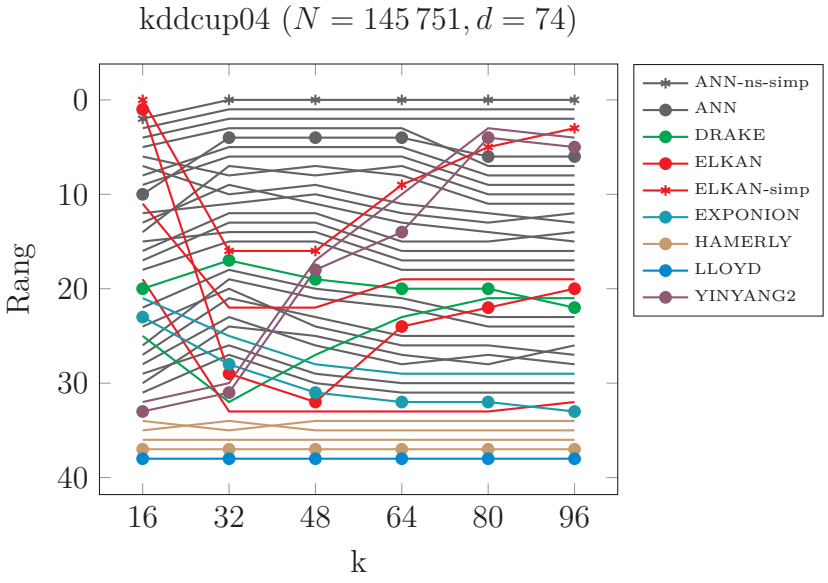


Abbildung 6.6.: Rangliste der durchschnittlichen Zeiten zum Clustering des kddcup04-Datensatzes.

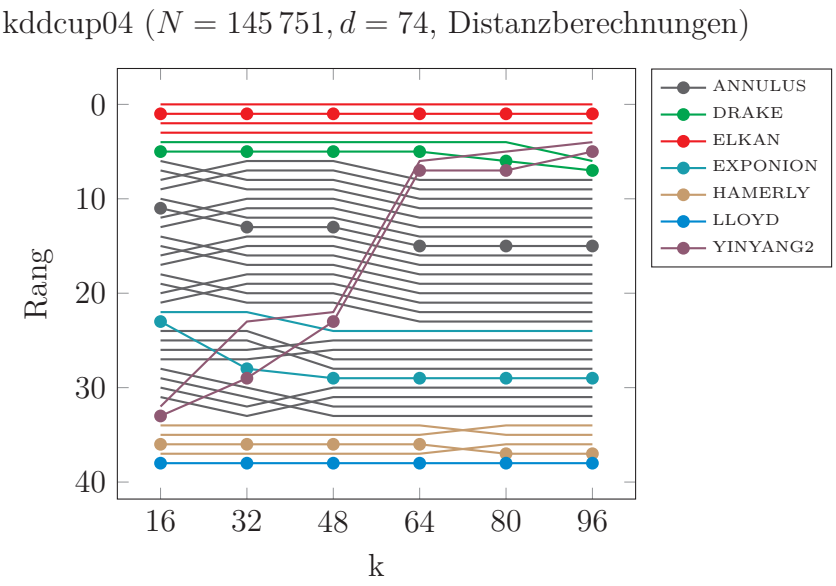


Abbildung 6.7.: Rangliste der durchschnittlichen Distanzberechnungen zum Clustering des kddcup04-Datensatzes.

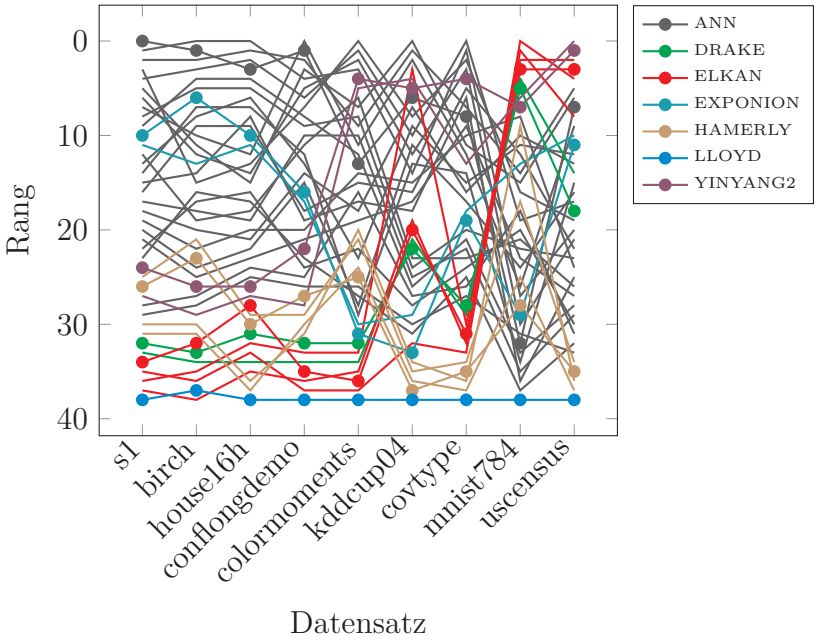


Abbildung 6.8.: Rangliste der durchschnittlichen Zeiten zum Clustering in 96 Cluster abhängig vom Datensatz. Die Datensätze sind aufsteigend nach dem Produkt aus der Anzahl der Datenpunkte und der Dimension geordnet.

6.3.2. Verhalten der Datensätze

Die Leistung der Pruningkriterien hängt entscheidend von der Genauigkeit der eingesetzten Schranken ab. Je genauer und somit kleiner die obere Schranke ist, desto wahrscheinlicher ist es, dass diese eine untere Schranke unterschreitet. Auch die Fläche des Annulus von Pruningkriterium 3 ist unmittelbar mit der Größe der oberen Schranke verbunden. Die Bewegung der Clusterzentren ist der Aspekt des Clusterings, der es erfordert, die Schranken anzupassen. Dieser Abschnitt untersucht daher die Bewegungen der Clusterzentren in den eingesetzten Datensätzen.

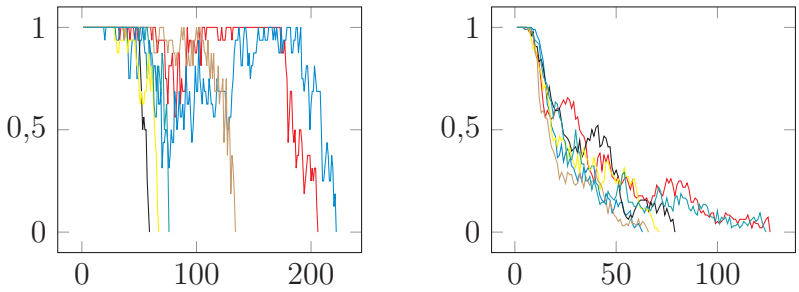
Zunächst möchten wir einen Fokus auf das Verhältnis aus statischen und aktiven Clustern setzen. Die Clusterzentren eines statischen Clusters (Definition 5) bewegen sich nicht. Für diese ist keine weitere Anpassung der Schranken notwendig. Abbildungen 6.9 bis 6.12 zeigen für ausgewählte Datensätze den Anteil der aktiven Cluster, also der Cluster, die an der Anpassung der Schranken beteiligt sind, abhängig von der Iteration. Die einzelnen farbigen Verlaufslinien entsprechen jeweils einer unterschiedlichen Initialisierung mit k-means++.

Es zeigt sich, dass nicht angenommen werden kann, dass statische Cluster der aktuellen Iteration auch in zukünftigen Iterationen statische Cluster sind. In mehreren Beispielen ist zu erkennen, dass nach lokalen Minima aktiver Cluster wieder deutlich höhere Maxima folgen. Besonders ausgeprägt ist dies für die schwarze Initialisierung in Abbildung 6.11b. Nachdem in Iteration 78 bereits über 96 % der Cluster statische Cluster waren, waren noch rund 200 weitere Iterationen erforderlich, bis der Zustand konvergiert ist. Die Anzahl der statischen Cluster reduzierte sich stellenweise wieder auf lediglich 40 %.

Je größer die Anzahl der Cluster, desto früher sinkt die Anzahl der aktiven Cluster. Abbildung 6.13 zeigt, dass die Anzahl der aktiven Cluster bei geringer Anzahl erst nach etwa 50 % der Iterationen beginnt abzunehmen. Bei größerer Anzahl von Clustern ist die Krümmung deutlich geringer ausgeprägt. Die Anzahl der aktiven Cluster sinkt etwa linear. Die Verlaufsgraphen in [KFN00] konnten mit den in dieser Arbeit verwendeten Datensätzen, Initialisierungen und Clusteranzahlen nicht bestätigt werden. In [KFN00] verhält sich die Anzahl der aktiven Cluster ungefähr umgekehrt proportional zu der Anzahl der Iterationen. Insbesondere nahm die Anzahl der aktiven Cluster zu Beginn stark ab, was gegensätzlich zu den Ergebnissen dieses Abschnitts ist. Am ehesten entspricht Abbildung 6.9b dem in [KFN00] beobachteten Verlauf.

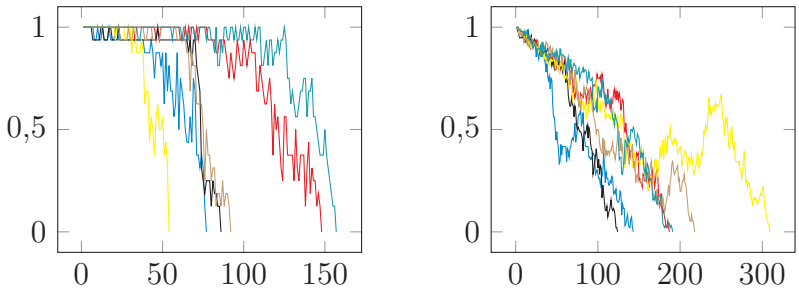
Neben dem Verhältnis von aktiven und statischen Cluster zeigen die Verlaufsgraphen auch den Einfluss der Initialisierung auf die Anzahl der benötigten Iterationen. Dieser Einfluss ist deutlich zu erkennen. Abbildung 6.10b zeigt, dass die schlechteste Initialisierung des conflongdemo für 96 Cluster (gelb) fast drei Mal so viele Iterationen zur Konvergenz benötigt wie die beste (schwarz).

- This area intentionally left blank -



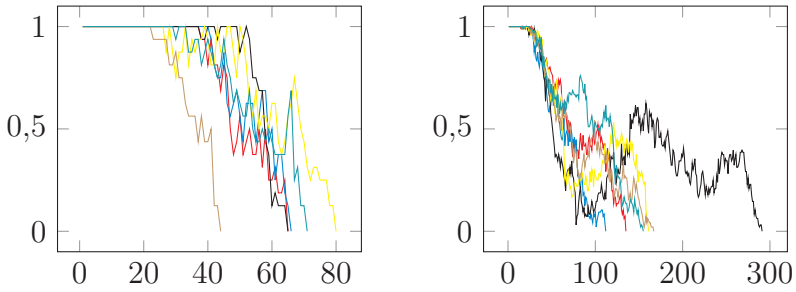
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.9.: Anteil der aktiven Cluster pro Iteration im birch-Datensatz.



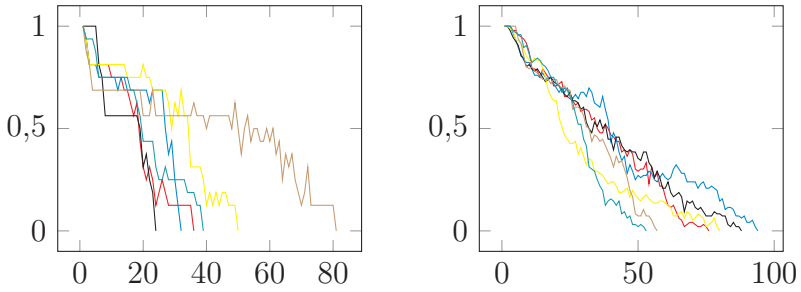
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.10.: Anteil der aktiven Cluster pro Iteration im conflongdemo-Datensatz.



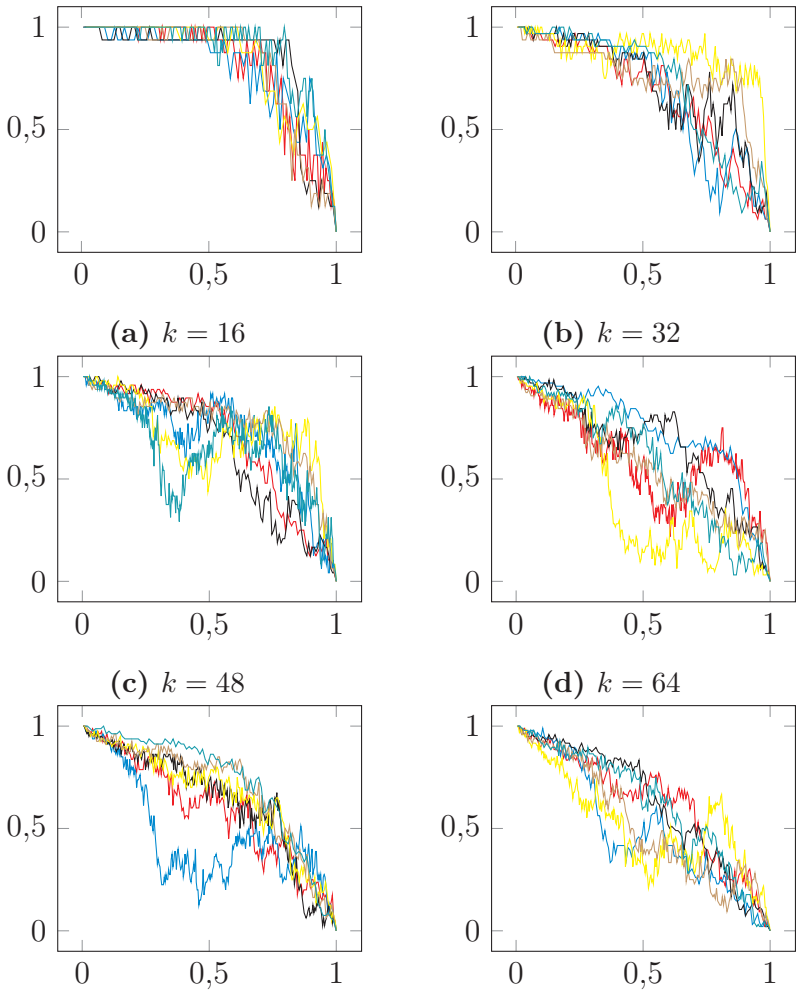
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.11.: Anteil der aktiven Cluster pro Iteration im mnist784-Datensatz.



(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.12.: Anteil der aktiven Cluster pro Iteration im uscensus-Datensatz.



(a) $k = 16$ **(b)** $k = 32$

(c) $k = 48$ **(d)** $k = 64$

(e) $k = 80$ **(f)** $k = 96$

Abbildung 6.13.: Anteil der aktiven Cluster nach Clusteringfortschritt im conflongdemo-Datensatz. Die X-Achse repräsentiert die normalisierte Iteration. Bei 1 ist das Clustering abgeschlossen.

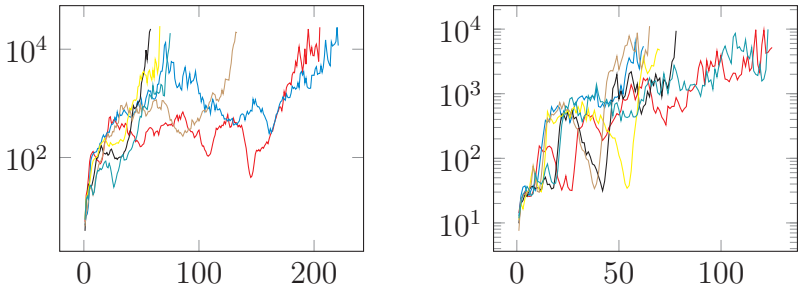
Statische Cluster können als Spezialfall des Betrags der Zentrenbewegung aufgefasst werden. Letztere ist das eigentlich Entscheidende für die Genauigkeit der Schranken. Abbildungen 6.14 und 6.15 zeigen, dass die maximale Bewegung der Clusterzentren relativ zu der durchschnittlichen Distanz der Clusterzentren, insbesondere in späteren Iterationen, zu vernachlässigen ist. Der in diesen Abbildungen dargestellte Faktor entspricht in einer idealisierten Situation[8] der Anzahl der Iterationen, nach denen für einen Datenpunkt, der sich direkt neben einem Clusterzentrum befindet und somit eine sehr kleine obere Schranke besitzt, eine neue Distanzberechnung spätestens erforderlich wäre. Für einen Datenpunkt, der auf der Hälfte der Distanz zu einem anderen Clusterzentrum und somit direkt an der Grenze der Voronoizelle liegt, ist in nahezu jeder Iteration eine Distanzberechnung erforderlich, unabhängig davon, wie stark sich die Clusterzentren bewegen. Zwischen diesen beiden Grenzfällen verhält sich die Anzahl der Iterationen, die mit einer ursprünglich scharfen unteren Schranke ohne zusätzliche Distanzberechnungen überbrückt werden können linear.

Die zuvor betrachtete maximale Bewegung der Clusterzentren ist der relevante Aspekt für den Einfluss der Clusterbewegungen beim Einsatz der Hamerly-Schranke (Seite 24). Diese ist, wie dort diskutiert, stark anfällig für Big Mover. Abschließend soll für die Datensätze daher noch ermittelt werden, ob beim Clustering Big Mover auftreten oder ob die meisten Cluster ähnliche Distanzen zurücklegen. Im Falle der Existenz von Big Movern soll zusätzlich untersucht werden, wie stark die Bewegung der Big Mover von der durchschnittlichen Zentrenbewegung abweicht. Abbildungen 6.16 bis 6.18 setzen die maximalen Zentrenbewegung mit der durchschnittlichen Zentrenbewegung in Verhältnis. Während sich das Verhältnis bei $k = 16$ im einstelligen Bereich befindet, bewegen sich Big Mover, unabhängig von der Größe und Dimension des Datensatzes, bei größeren Clusterzahlen deutlich stärker als das durchschnittliche Clusterzentrum. Wenn man nur die Bewegung der aktiven Clusterzentren in Verhältnis zu der maximalen Zentrenbewegung setzt, also eine Bewegung von 0 unberücksichtigt lässt, dann weicht die Bewegung der Big Mover weniger stark von denen eines durchschnittlichen Clusterzentrums ab (Abbildungen 6.19 bis 6.21), bleibt in einigen Fällen aber weiterhin im mittleren zweistelligen

[8] Alle Zentren besitzen zueinander genau die durchschnittliche Center-Center-Distanz. Das derzeit zugeordnete Clusterzentrum bewegt sich nicht, wodurch die obere Schranke unverändert bleibt.

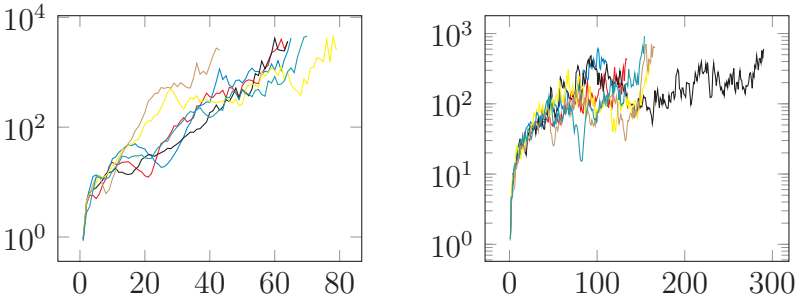
Bereich (Abbildung 6.21b). Diese Beobachtungen ist konsistent zu den zuvor betrachteten Ranglisten der Algorithmenvarianten. Für Datensätze bei denen der Hamerly für kleine k niedrigere Ränge erreicht hat, sinkt der Rang mit zunehmendem k ab (Abbildungen 6.1 und 6.2 auf Seite 85).

- This area intentionally left blank -



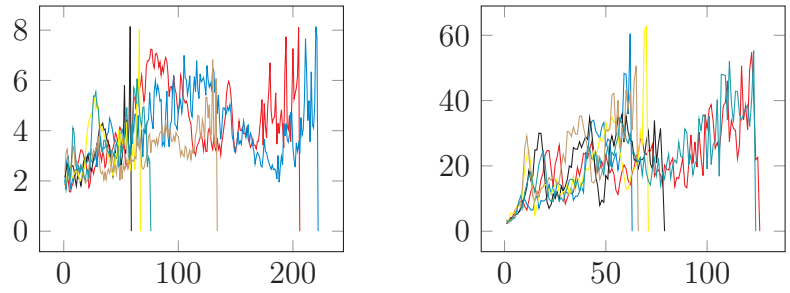
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.14.: Durchschnittlicher Zentrenabstand geteilt durch maximale Zentrenbewegung abhängig von der Iteration im birch-Datensatz.



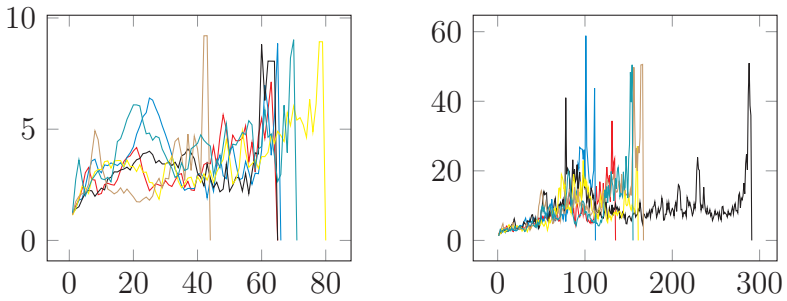
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.15.: Durchschnittlicher Zentrenabstand geteilt durch maximale Zentrenbewegung abhängig von der Iteration im mnist784-Datensatz.



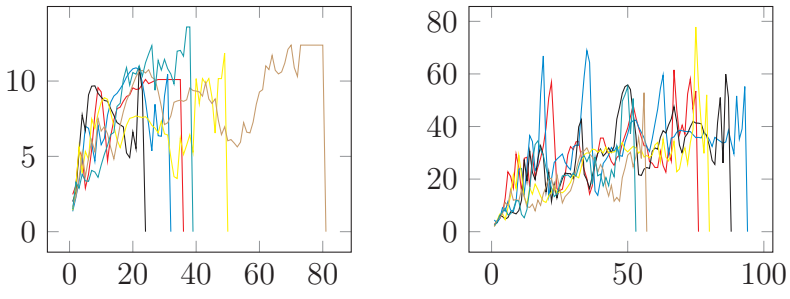
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.16.: Maximale Zentrenbewegungen geteilt durch die durchschnittliche Zentrenbewegung abhängig von der Iteration im birch-Datensatz.



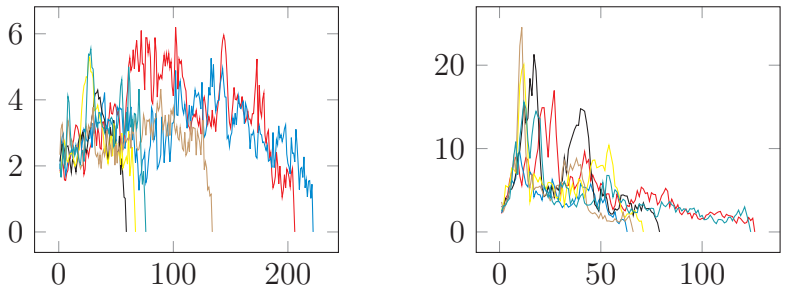
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.17.: Maximale Zentrenbewegungen geteilt durch die durchschnittliche Zentrenbewegung abhängig von der Iteration im mnist784-Datensatz.



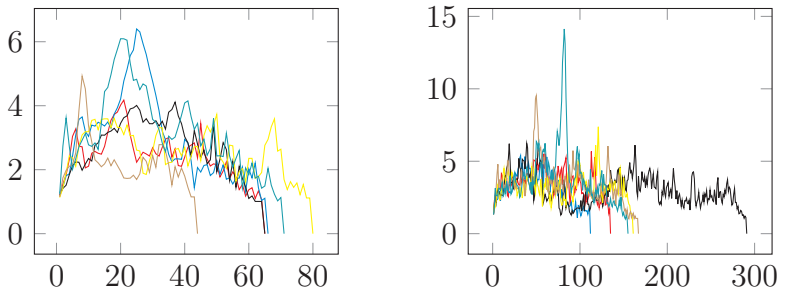
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.18.: Maximale Zentrenbewegungen geteilt durch die durchschnittliche Zentrenbewegung abhängig von der Iteration im uscensus-Datensatz.



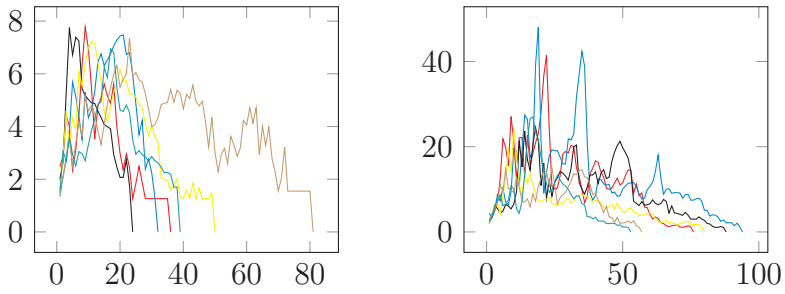
(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.19.: Maximale Zentrenbewegungen geteilt durch die durchschnittliche Zentrenbewegung aktiver Cluster abhängig von der Iteration im birch-Datensatz.



(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.20.: Maximale Zentrenbewegungen geteilt durch die durchschnittliche Zentrenbewegung aktiver Cluster abhängig von der Iteration im mnist784-Datensatz.



(a) $k = 16$ **(b)** $k = 96$

Abbildung 6.21.: Maximale Zentrenbewegungen geteilt durch die durchschnittliche Zentrenbewegung aktiver Cluster abhängig von der Iteration im uscensus-Datensatz.

6.3.3. Analyse der Algorithmen

Nachdem im Vergleich der Algorithmen untersucht wurde, *wie* gut die Leistung der einzelnen Algorithmen im Vergleich zu anderen Algorithmen ist, möchten wir in diesem Abschnitt untersuchen, *warum* die Algorithmen diese Ergebnisse erreichen und welche Parameter die Leistung der Algorithmen beeinflussen. Ein besonderer Fokus soll dabei auf die Alleinstellungsmerkmale der einzelnen Algorithmen gesetzt werden.

Elkan

Die Funktionsweise, Vor- und Nachteile des Elkan sind Grundlage bei der Entwicklung der anderen Algorithmen gewesen, wodurch dieser kein Alleinstellungsmerkmal besitzt. Aus diesem Grund soll die Leistung der einzelnen Pruningkriterien und insbesondere die Unterschiede der Pruningleistung zwischen dem regulären Elkan mit Pruningkriterium 2 und dem Simplified Elkan ohne Einsatz von Pruningkriterium 2 untersucht werden.

Bereits in dem zuvor erfolgten Vergleich der Algorithmenvarianten hat sich herausgestellt, dass der reguläre Elkan im Vergleich zum Simplified Elkan durch das zusätzliche Pruningkriterium nicht in jedem Fall Distanzberechnungen einsparen kann (Seite 76). Dies ist nicht ausschließlich darin begründet, dass Pruningkriterium 2 für bestimmte Konstellationen aus Datensatz und Clusteranzahl eine schlechte Leistung erbringt. Stattdessen ist es so, dass der Simplified Elkan in der Lage ist, den Wegfall von Pruningkriterium 2 durch eine bessere Leistung der unteren Schranke zum größten Teil zu kompensieren.

Abbildungen 6.22 bis 6.27 zeigen, dass der Verlaufsgraph für jeweils einen beispielhaften Clusteringlauf mit unterschiedlichen Clusteranzahlen auf dem mnist784- und dem uscensus-Datensatz für Elkan und Simplified Elkan nahezu identisch ist. Auf der vertikalen Achse ist die Anzahl der eingesparten Distanzberechnungen gestapelt aufgetragen. Die Pruningkriterien werden nacheinander abgeprüft, ein erfolgreiches Pruning mit Hilfe eines früheren Pruningkriteriums führt dazu, dass ein späteres Pruningkriterium nicht mehr abgeprüft werden muss. Unsere Implementierung des Elkan prüft die Kriterien innerhalb der im Diagramm aufgetragenen Reihenfolge. Insbesondere wird die untere Schranke vor Pruningkriterium 2, aber nach der globalen Variante von Pruningkriterium 2 überprüft. Auf diese Weise ergibt sich die Summe der eingesparten Distanzberechnungen aus der Anzahl der eingesparten Distanzberechnungen der

einzelnen Pruningkriterien. Die horizontale Linie am oberen Ende des Graphen gibt die Anzahl der Distanzberechnungen des Lloyd und damit die maximal einzusparende Anzahl an Distanzberechnungen an.

Für den uscensus mit 96 Clustern ist in Abbildung 6.27 deutlich zu erkennen, dass Pruningkriterium 2 für den regulären Elkan deutlich mehr als die Hälfte der Pruningleistung erbringt. Wenn man aber die Differenz der kombinierten Pruningleistung aller Pruningkriterien zwischen Simplified Elkan und regulärem Elkan betrachtet, dann liegt diese im Bereich von 0,1 %. Im Vergleich zum regulären Elkan vervierfacht sich die Pruningleistung der unteren Schranke für den Simplified Elkan.

Der größte und relevante Unterschied zwischen dem regulären Elkan und dem Simplified Elkan befindet sich in der ersten Iteration. Innerhalb der ersten Iteration sind noch keine unteren Schranken bekannt, ein Pruning ist daher nur auf Basis der Center-Center-Distanzen möglich[9]. Hier hat der Simplified Elkan einen deutlichen Nachteil, der *zum Teil* in den folgenden Iterationen wieder ausgeglichen wird. Ein erfolgreiches Pruning führt dazu, dass für diese Kombination aus Datenpunkt und Clusterzentrum keine untere Schranke berechnet wird. Insbesondere führt ein erfolgreiches Pruning mit Hilfe von Pruningkriterium 2 in der ersten Iteration dazu, dass die untere Schranke für diese Kombination aus Datenpunkt und Clusterzentrum auf dem initialen Wert von 0 verbleibt. Wenn in der zweiten Iteration *nicht* erneut ein Pruning mit Pruningkriterium 2 möglich ist, dann kann aus diesem Grund auch nicht mit Hilfe der unteren Schranke gepruned werden. Der Simplified Elkan musste innerhalb der ersten Iteration notwendigerweise alle unteren Schranken berechnen. Aufgrund der feingranularen unteren Schranke ist ein Pruning in der zweiten und folgenden Iterationen in der Regel möglich.

Eine notwendige Bedingung für die Ersparnis von Distanzberechnungen ist, dass der reguläre Elkan identische Paare aus Clusterzentrum und Datenpunkt über *alle* Iterationen hinweg mit Hilfe von Pruningkriterium 2 prunen kann. Andernfalls verschiebt sich im Vergleich zum Simplified Elkan lediglich der Zeitpunkt der erstmaligen Berechnung der unteren Schranke. Abbildung 6.28 zeigt als Gegenstück zu Abbildung 6.27, dass der Großteil der unteren Schranken für diese konkrete Initialisierung für den uscensus mit 96 Clustern nie benötigt wird. Von ungefähr 240 Millionen unteren Schranken werden 60 Mil-

[9] Und für die Datenpunkte, die als initiales Clusterzentrum gewählt wurden, da die obere Schranke für diese 0 beträgt.

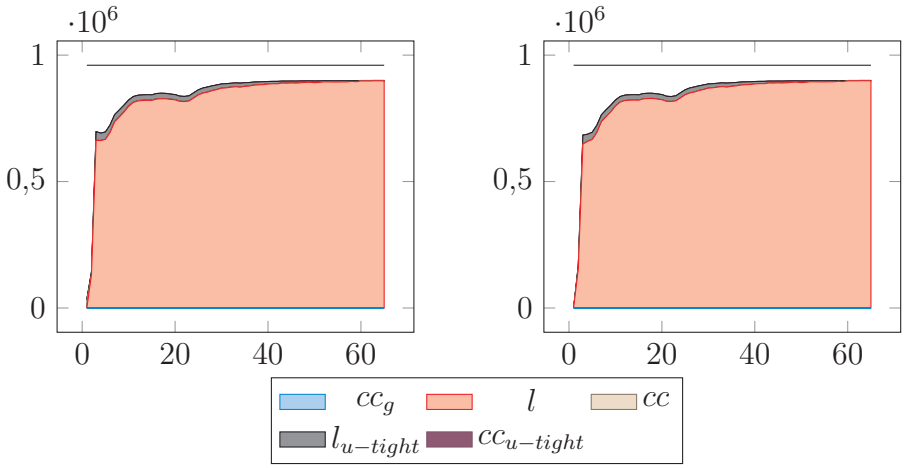


Abbildung 6.22.: Pruningkriterien im Vergleich für Elkan und Simplified Elkan für mnist784 mit 16 Clustern.

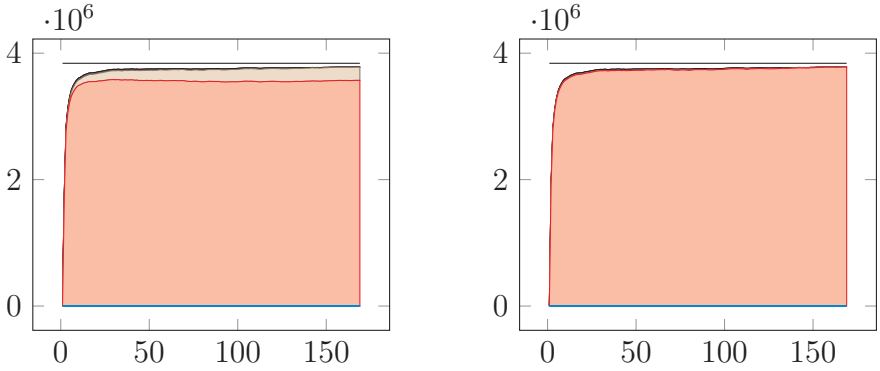


Abbildung 6.23.: Pruningkriterien im Vergleich für Elkan und Simplified Elkan für mnist784 mit 64 Clustern.

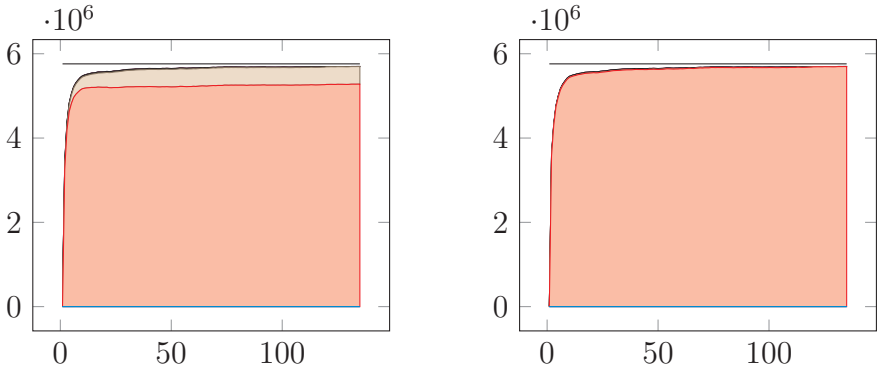


Abbildung 6.24.: Pruningkriterien im Vergleich für Elkan und Simplified Elkan für mnist784 mit 96 Clustern.

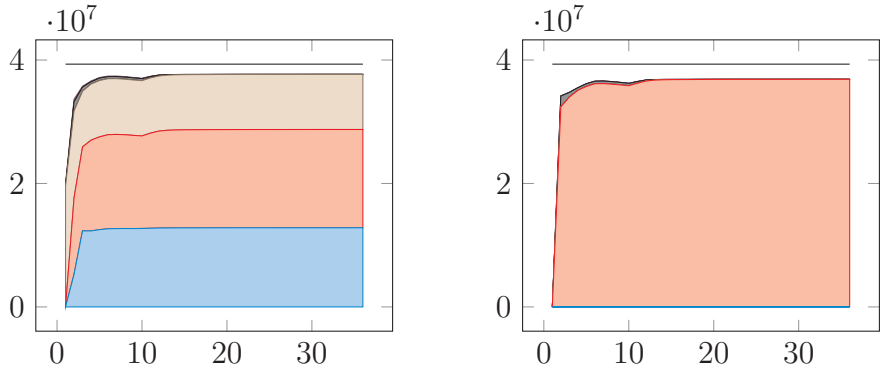


Abbildung 6.25.: Pruningkriterien im Vergleich für Elkan und Simplified Elkan für uscensus mit 16 Clustern.

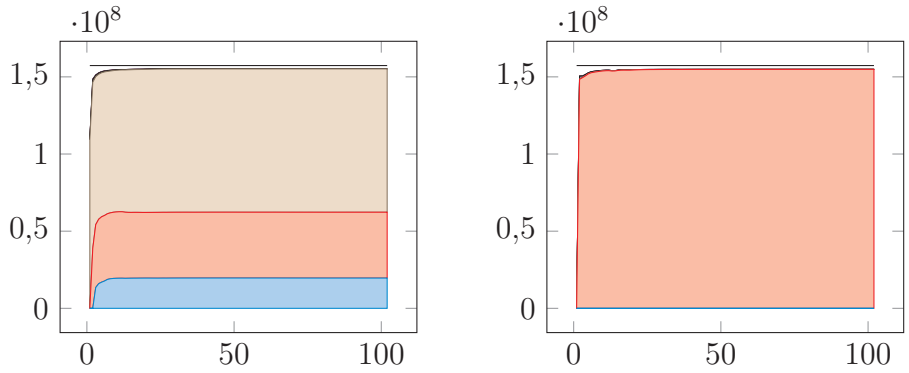


Abbildung 6.26.: Pruningkriterien im Vergleich für Elkan und Simplified Elkan für uscensus mit 64 Clustern.

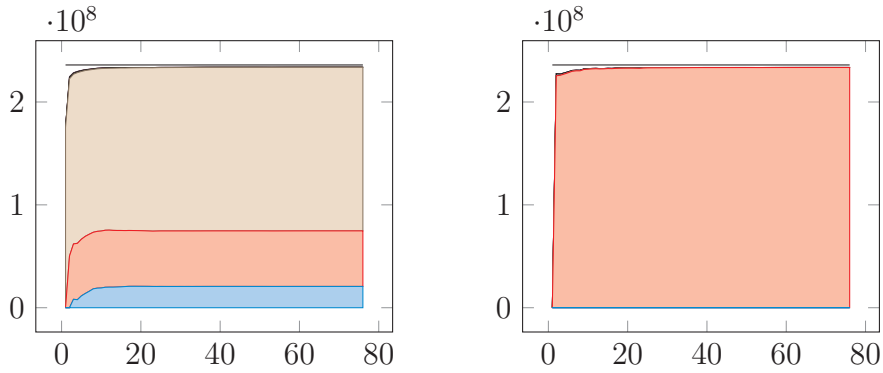


Abbildung 6.27.: Pruningkriterien im Vergleich für Elkan und Simplified Elkan für uscensus mit 96 Clustern.

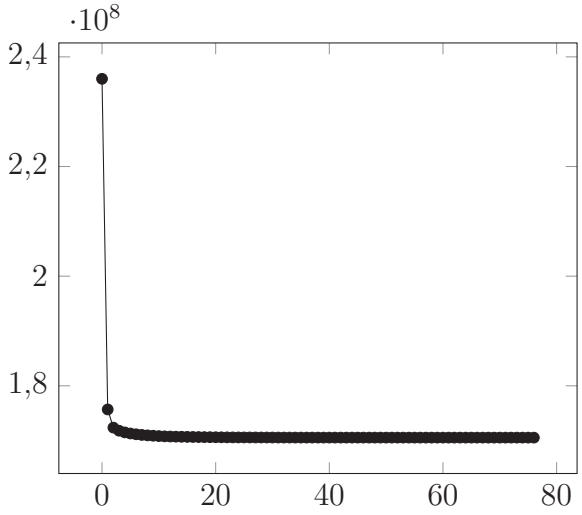


Abbildung 6.28.: Anzahl der nie aktualisierten unteren Schranken des regulären Elkan für uscensus mit 96 Clustern.

lionen innerhalb der ersten Iteration initial gesetzt, da kein Pruning möglich war. Weitere 3 Millionen werden innerhalb der zweiten Iteration gesetzt. Die restlichen Iterationen bewirken nur geringe Änderungen an dieser Anzahl. Am Ende verbleiben ungefähr 170 Millionen untere Schranken, die auf dem initialen Wert von 0 verbleiben, beziehungsweise durch die Bewegung der Clusterzentren in den negativen Bereich angepasst werden. Dies deckt sich mit der Verteilung der Pruningleistung in Abbildung 6.27.

Die durch die gute Leistung von Pruningkriterium 2 nicht erfolgte Berechnung der unteren Schranke ist also ursächlich an der scheinbar schlechten Pruningleistung der unteren Schranke. Eine weitere Verbesserung der Leistung von Pruningkriterium 2 würde die Leistung der unteren Schranke weiter verringern.

Für den praktischen Einsatz bedeutet das also, dass zur Berechnung der Center-Center-Distanzen zusätzliche Distanzberechnungen und zur Prüfung zusätzliche Verzweigungen erfolgen, nur um das Pruning von einem Pruningkriterium auf ein anderes zu verschieben. Insbesondere die größere Anzahl an Verzweigungen hat einen negativen Einfluss auf die benötigte Realzeit. Ein Test mit dem bereits zuvor betrachteten beispielhaften Clustering des uscensus in 96 Cluster zeigt, dass der Simplified Elkan zwar 5 % mehr Instruktionen

ausführt und 10 % mehr Verzweigungen überprüfen muss[10], die Anzahl der falsch vorhergesagten Verzweigungen durch die Sprungvorhersage sinkt aber auf weniger als die Hälfte (Listing 6.5). Das führt zu einer besseren Auslastung der CPU, da mehr Instruktionen pro Taktzyklus ausgeführt werden können. Trotz der höheren Anzahl an ausgeführten Instruktionen kommt es dadurch zu einer reduzierten Anzahl von Taktzyklen und somit reduzierten Realzeit.

```
1  Performance counter stats for 'target/kmeans_ELKAN_KMEANSPP_rolling Ascii ../
       datasets/target/uscensus.txt 96 1':
2
3  [...]
4    847,758,833,256  cycles           #    2.904 GHz                      (83.33%)
5  [...]
6  1,778,210,811,027  instructions     #    2.10  insns per cycle
7                                      #    0.12  stalled cycles per insn  (83.35%)
8    280,317,574,653  branches         #  960.226 M/sec                    (83.33%)
9      3,697,319,986  branch-misses    #    1.32% of all branches          (83.33%)
10 [...]
11
12 Performance counter stats for 'target/kmeans_ELKAN_KMEANSPP_rolling_simplified
       Ascii ../datasets/target/uscensus.txt 96 1':
13
14 [...]
15   783,348,635,053  cycles           #    2.915 GHz                      (83.34%)
16 [...]
17 1,871,025,410,795  instructions     #    2.39  insns per cycle
18                                     #    0.10  stalled cycles per insn  (83.34%)
19   309,145,314,405  branches         # 1150.346 M/sec                    (83.34%)
20     1,584,765,985  branch-misses    #    0.51% of all branches          (83.33%)
21 [...]
```

Listing 6.5: Sprungvorhersage im Vergleich zwischen Elkan und Simplified Elkan

Drake

Besonderes Merkmal des Drake ist die variable Anzahl von unteren Schranken. Diese sollen im Verlauf des Algorithmus reduziert werden können, um Rechenzeit einzusparen. In der Praxis funktioniert dies auf den verwendeten Datensätzen mit den verwendeten Clusteranzahlen allerdings nicht. *Wenn* eine Anpassung der Schrankenanzahl erfolgt, dann erfolgt diese nahezu immer innerhalb der ersten 10 % des Clusterings auf den minimalen Wert $\frac{k}{8}$ (Abbildung 6.29). Eine weitere Häufung der Anpassungen befindet sich kurz vor Abschluss des Clusterings. Am weitaus häufigsten erfolgt aber überhaupt keine Anpassung der Anzahl der Schranken. Die Heatmap in Abbildung 6.30 zeigt, dass für den Großteil der

[10] Vermutlich bedingt durch die Prüfung in Zeile 4 von Algorithmus 3.1, die das Betreten der Schleife über alle Clusterzentren für einen Datenpunkt vermeidet.

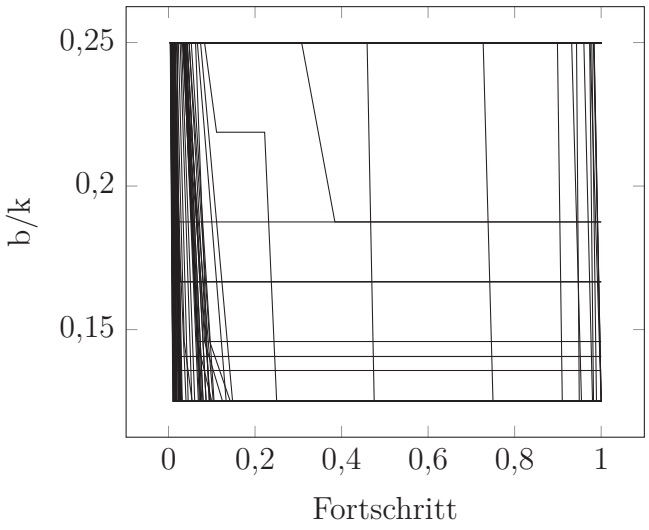


Abbildung 6.29.: Verlauf des Anteils $\frac{b}{k}$ der genutzten Schranken b im Drake als Übersicht aller Datensätze, Clusteranzahlen und Seeds.

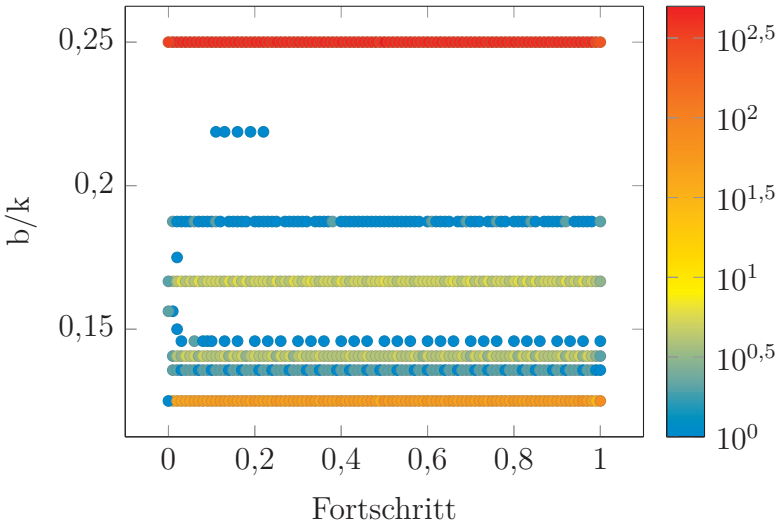


Abbildung 6.30.: Heatmap des Anteils $\frac{b}{k}$ der genutzten Schranken b im Drake als Übersicht aller Datensätze, Clusteranzahlen und Seeds.

Läufe der Startwert von $\frac{k}{4}$ nicht oder erst kurz vor Abschluss des Clusterings verlassen wird.

In vielen Fällen verhindern einzelne Datenpunkte dadurch, dass für diese kein Pruning möglich ist, eine Anpassung der Schrankenanzahl. Beispielhaft haben rund $4{,}96 \cdot 10^{-5}\,\%$ der Datenpunkte für ein Clustering des uscensus-Datensatzes in 96 Cluster[11] bis zuletzt verhindert, dass die Anzahl der Schranken reduziert werden konnte. Nach der Hälfte der Iterationen war für diesen konkreten Clusteringlauf für rund 99 % der Datenpunkte eine Schranke ausreichend, um ein Pruning zu ermöglichen.

Das andere Extrem tritt beim Clustering von niedrigdimensionalen Datensätzen in viele Cluster, beispielhaft dem birch in 1024 Cluster[12], auf. Nach der zweiten Iteration war die maximale Anzahl benötigter Schranken für keinen Datenpunkt größer als 10. Die Anzahl der Schranken wurde daher auf $\frac{1024}{8} = 128$ reduziert und verblieb für die verbleibenden 70 Iterationen auf diesem Wert. 110 Schranken wurden unnötig weiterhin aktualisiert und es muss Hauptspeicher für den maximalen Speicherbedarf von 256 initialen Schranken bereitgestellt werden.

Dadurch, dass die Bestimmung der Anzahl der genutzten Schranken sehr leichtgewichtig implementiert werden kann, kommt es für Clusteringläufe, in denen keine Reduktion stattfinden kann, zu keiner relevanten Erhöhung der benötigten Realzeit. Wenn eine Reduktion der Anzahl der Schranken stattfinden kann, dann kann die für die Aktualisierung der Schranken benötigte Zeit potentiell halbiert werden. Im Vergleich zu der benötigten Zeit für die restlichen Berechnungen ist aber auch dies zu vernachlässigen.

Zusammenfassend lässt sich sagen, dass eine Anpassung der Anzahl der unteren Schranken nicht schädlich ist, in der Praxis aber auch keine deutlich spürbaren Vorteile bringt. Der Drake erreicht seine Leistung vielmehr durch die im Vergleich zum Elkan reduzierte Anzahl von Schranken, nicht durch die dynamische Anpassung der Anzahl dieser bereits reduzierten Anzahl.

[11] seed = 2
[12] seed = 1

Annulus

Der Annulus ist bei unseren Tests der Algorithmus mit der größten Anzahl an Varianten gewesen. Eine Stellschraube ist die mögliche Deaktivierung der binären Suche. Dies ist nicht sinnvoll. Die Deaktivierung der binären Suche ist, abgesehen von einzelnen Ausnahmen, nur für 16 Cluster und zum Clustering des mnist784-Datensatzes schneller gewesen. Für 16 Cluster ist die Verbesserung aber unter 5 % für eine ohnehin schon kleine Realzeit. Im Falle des mnist784 erreicht Pruningkriterium 3 generell nur eine geringe Leistung, der weitaus größte Teil der Clusterzentren liegt auf dem Annulus. Nach der binären Suche muss daher trotzdem über alle Clusterzentren iteriert werden, genau so, wie es bei der linearen Suche der Fall wäre.

Die zweite annulusspezifische Stellschraube ist die Auswahl des Fixpunkts. Die Auswahl des Fixpunkts als Schwerpunkt aller Datenpunkte benötigt mit Ausnahme des colormoments-Datensatzes mehr Distanzberechnungen, als die Auswahl des Fixpunkts im Ursprung. In Fällen, in denen die Auswahl der benötigten Distanzberechnungen gesunken ist, beträgt der Unterschied weniger als 2 %. Wenn sich der Schwerpunkt im Fixpunkt befindet, dann befinden sich Datenpunkte und somit Clusterzentren in allen 4 Quadranten. Entsprechend deckt der Annulus hier eine deutlich größere Fläche, in der sich potentiell Clusterzentren befinden können, ab.

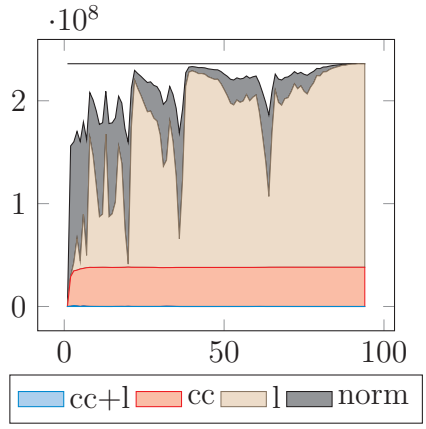
Den Fixpunkt hingegen als das komponentenweise Minimum aller Datenpunkte zu wählen, bringt große Leistungsunterschiede sowohl in die positive als auch in die negative Richtung. Beim Clustering von colormoments und covtype konnten bis zu 25 % der Distanzberechnungen eingespart werden. Für diese beiden Datensätze war der Fixpunkt im Minimum in keinem Fall schlechter als der Fixpunkt im Ursprung. Beim colormoments befinden sich die Datenpunkte relativ gleichmäßig um den Ursprung verteilt und alle Komponenten besitzen eine ähnliche Standardabweichung. Dies entspricht der Situation bei der Auswahl des Fixpunkts im Schwerpunkt aller Datenpunkte, die wie oben diskutiert, eine schlechte Leistung bringt. Entsprechend verbessert sich die Leistung bei Änderung der Position des Fixpunkts.

Für birch, house16h, mnist784 und s1 sind die Unterschiede zu vernachlässigen, für viele Komponenten ist das Minimum hier nahe bei 0, sodass sich die Position des Fixpunkts nur marginal ändert.

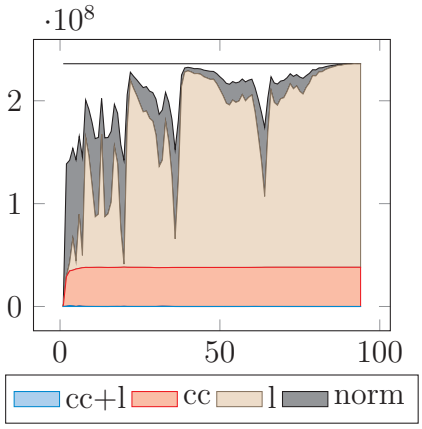
Für den kddcup04, conflongdemo und uscensus wurden bis zu 20 % mehr Distanzberechnungen benötigt. Die Leistungsabnahme beim uscensus ist hier von besonderem Interesse. Nur zwei der 68 Komponenten haben ein Minimum das nicht 0 ist. Komponente 3 bewegt sich ganzzahlig im Bereich von 1 bis 12. 70 % der Datensätze haben dort das Minimum 1. Komponente 50 bewegt sich mit Lücken ganzzahlig im Bereich von 10 bis 52. 63 % der Datensätze haben dort das Minimum 10. Es wäre daher anzunehmen, dass der Einfluss durch die geänderte Position des Fixpunkts nur gering ist. Ursächlich an der deutlichen Abnahme der Leistung ist die Verschiebung des Fixpunkts in der 50. Komponente. Diese besitzt mit 11,56 die zweitgrößte Standardabweichung aller Komponenten und ist für einen beispielhaften Lauf mit 96 Clustern eine der beiden Komponenten, deren Standardabweichung größer als die durchschnittliche obere Schranke ($\approx 5{,}94$) ist.

Die Leistung der einzelnen Pruningkriterien dieses beispielhaften Laufs ist in Abbildung 6.31 für die zwei Positionen des Fixpunkts im Annulus und für den Exponion dargestellt. Analog zu der Analyse des Elkan sind auf der vertikalen Achse die Anzahl der eingesparten Distanzberechnungen gestapelt aufgetragen. Da sowohl die untere Schranke als auch Pruningkriterium 2 global arbeiten, erfasst unsere Implementierung ebenfalls die Anzahl der Distanzen, bei denen beide Pruningkriterien ein Pruning erlauben ($cc + l$). Die Werte für $cc + l$, cc und l entsprechen der Pruningleistung des Hamerly und sind in allen Varianten identisch. Das zusätzliche Pruningkriterium des Annulus und des Exponion kommt nur dann zum Einsatz, wenn diese nicht erfolgreichen prunen können. Unabhängig von der Leistung dieses zusätzlichen Pruningkriteriums werden die obere und untere Schranke identisch zum Hamerly aktualisiert. Es wird deutlich, dass die größte Pruningleistung durch die unteren Schranken erreicht wird. Diese dominieren daher die Form des Graphen. Abbildung 6.31d zeigt die Leistung des jeweils zusätzlichen Pruningkriteriums daher noch einmal im direkten Vergleich. Der Exponion erreicht die beste Leistung, gefolgt von dem Fixpunkt im Ursprung. Der Fixpunkt als komponentenweises Minimum erreicht, wie zuvor diskutiert, die schlechteste Leistung.

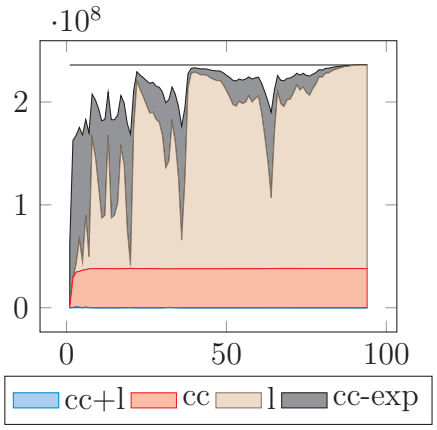
Der Grund in der Leistungsabnahme bei Verschiebung des Fixpunkts zum Minimum aller Komponenten besteht darin, dass für einen Datenpunkt die durch den Annulus abgedeckte Fläche beginnend an diesem Datenpunkt entlang einer Dimension zunimmt je näher der Fixpunkt dem Wert des Datenpunkts in



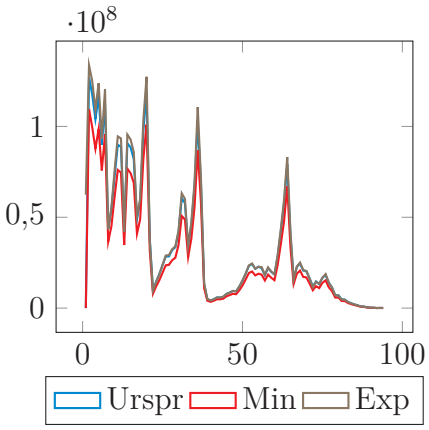
(a) Fixpunkt im Ursprung



(b) Fixpunkt als Minimum aller Komponenten



(c) Exponion



(d) Vergleich der Leistung des zusätzlichen Pruningkriteriums.

Abbildung 6.31.: Einfluss der Position des Fixpunkts auf die Pruningleistung für einen Clusteringlauf auf dem uscensus-Datensatz mit 96 Clustern.

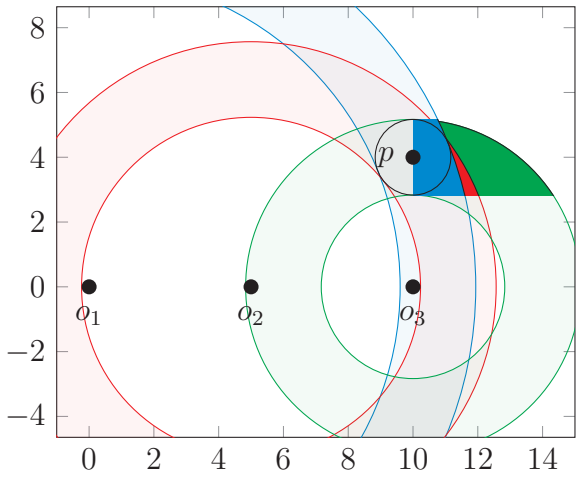


Abbildung 6.32.: Ansteigende Fläche im Annulus je näher der Fixpunkt auf der X-Achse dem Punkt p kommt.

dieser Dimension kommt. Umgekehrt nimmt die abgedeckte Fläche der anderen Dimensionen ab. Abbildung 6.32 zeigt dies beispielhaft. Der kleinste Wert für die horizontale Achse im Datensatz beträgt 10 (analog zur Komponente 50 des uscensus) und wird repräsentiert durch den Datenpunkt p. Je näher der Fixpunkt o_n auf der horizontalen Achse dem Wert 10 kommt, desto größer ist die durch den Annulus abgedeckte Fläche in horizontaler Richtung für Werte beginnend bei 10.

Im konkreten Fall des uscensus-Datensatzes führt dies dazu, dass die abgedeckte Fläche entlang einer Komponente mit großer Varianz und Spannweite zunimmt, während die abgedeckte Fläche entlang der Komponenten mit kleiner Varianz und Spannweite abnimmt. Insbesondere ist die Spannweite der meisten Komponenten durch die obere Schranke nahezu vollständig abgedeckt, sodass unabhängig von der Position des Fixpunkts der Großteil der Werte dieser Komponenten auf dem Annulus liegt. Dies führt dazu, dass durch den Annulus auf der entscheidenen Komponente 50 eine größere Fläche abgedeckt wird. Als direkte Folge davon nimmt die Leistung von Pruningkriterium 3 ab.

Eine Verschiebung des Fixpunkts zur Verbesserung der Pruningleistung müsste also auch Varianz und Spannweite der einzelnen Dimensionen in Verbindung mit der durchschnittlichen oberen Schranke beziehungsweise der Durchmesser der natürlichen Cluster berücksichtigen, um konsistent eine verbesserte Leistung zu erbringen. Der durchschnittliche Betrag der oberen Schranke und die Durchmesser der natürlichen Cluster stehen a priori allerdings nicht zur Verfü-

gung, sodass diese nicht in die Auswahl der Position des Fixpunkts einfließen können. Eine statistische Standardisierung der Werte aller Dimensionen auf den Mittelwert 0 mit Varianz 1 würde dieses Problem umgehen. Diese kommt aber, obwohl die Standardisierung reversibel ist, zu anderen Ergebnissen. Ein Clustering auf Basis standardisierter Werte ist somit kein exaktes Clustering im Sinne von Definition 3.

7

Fazit und Ausblick

Im Rahmen dieser Arbeit über exakte, durch Schranken beschleunigte k-means-Clustering-Verfahren haben wir neben dem unbeschleunigten Lloyd-Algorithmus sechs Algorithmen der k-means-Familie vergleichend untersucht.

Untersuchungen der k-means-Familie in bestehender Literatur vergleichen die Leistung der unterschiedlichen Algorithmenvarianten anhand von extern gemessenen Daten, wie beispielsweise die benötigte Real- oder CPU-Zeit und dem benötigten Speicherverbrauch. Bei der Entwicklung neuer Algorithmenvarianten wird zum Teil begründet, warum die Neuentwicklung besser als ein bestehender Algorithmus sein soll, diese Verbesserung und Herangehensweise aber nicht anhand von Messdaten innerhalb des Algorithmus, sondern nur anhand der vorgenannten externen Messdaten belegt.

Ziel dieser Arbeit war es daher, systematisch zu untersuchen, *welchen Einfluss* die einzelnen Eingabeparameter, wie beispielsweise die Größe und Beschaffenheit des Datensatzes und die Anzahl der gewünschten Cluster, auf die Clusteringleistung der unterschiedlichen Algorithmen und Algorithmenvarianten haben und insbesondere, *warum* die einzelnen Algorithmenvarianten unterschiedlich durch diese Parameter beeinflusst werden.

Unsere Untersuchung besteht dabei aus zwei Teilen. Im ersten Teil wurden sechs Algorithmen aus bestehender Literatur auf Gemeinsamkeiten und Unterschiede untersucht, wiederverwendete Einzelkomponenten, die an der Beschleunigung beteiligt sind, extrahiert und die Funktionsweise auf theoretischer Basis untersucht. Dabei wurden in den Algorithmen eine obere Schranke, drei Arten von unteren Schranken und zwei Arten von ergänzenden Metadaten identifiziert, die es mit Hilfe der Dreiecksungleichung erlauben, Aussagen über die Distanzen zwischen Datenpunkten und Clusterzentren zu treffen. Mit Kombination einer Art von unterer Schranke und einer Menge der ergänzenden Metadaten lassen sich alle Pruningkriterien in den untersuchten Algorithmen abbilden. Die ergänzenden Metadaten werden auf unterschiedliche Art und

Weise in den Algorithmen genutzt. Algorithmen, die die gleichen Daten vorhalten, unterscheiden sich teilweise nur in der Bedingung, die zur Prüfung des möglichen Prunings genutzt wird. Die genaue Funktionsweise der einzelnen Pruningkriterien und die Zusammensetzung zu einem vollständigen Algorithmus wurden im Detail vorgestellt. Der Ablauf des Zuweisungsschritts für einen einzelnen Punkt wurde für alle Algorithmen mit Hilfe von Pseudocode in konkreter Form dargestellt. Basis dieses Pseudocodes ist die Beschreibung des Algorithmus innerhalb der Originalveröffentlichung, um sicherzustellen, dass alle Pruningkriterien in der korrekten Reihenfolge überprüft werden. Ein erfolgreiches Pruning führt dazu, dass spätere Pruningkriterien nicht mehr überprüft werden. Bei der Entwicklung dieses Pseudocodes wurde Wert darauf gelegt, dass identische Abläufe und Bedingungen im Pseudocode in allen zutreffenden Algorithmen in identischer Form umgesetzt wurden. Dabei stellt sich heraus, dass sich das Pruning in den verschiedenen Algorithmen zum Teil nur durch die Ergänzung einzelner Anweisungen unterscheidet. Details der Originalveröffentlichung, die im Sinne der Lesbarkeit nicht sinnvoll als Pseudocode abgebildet werden konnten, wurden textuell erklärt.

Im zweiten Teil wurden die zuvor vorgestellten Algorithmen in C++ auf Basis der STL ohne Verwendung externer Abhängigkeiten implementiert. Basis der Implementierung ist der im ersten Teil entwickelte Pseudocode in Verbindung mit den textuell beschriebenen Details. Der C++-Programmcode ist möglichst nah am Pseudocode orientiert. Insbesondere die Struktur der Kontrollstrukturen findet sich identisch in Pseudocode und C++-Programmcode. Falls die Beschreibung des Algorithmus in der Originalveröffentlichung Raum für Interpretationen ließ wurden mehrere Varianten des Algorithmus implementiert. Auf gleiche Weise wurden mehrere Varianten implementiert, wenn durch geringfügige Anpassungen, die den Ablauf nicht wesentlich verändern, ein anderes Laufzeitverhalten zu erwarten war. Als letzte Form der Variante wurden Techniken zur Verbesserung der Leistung, die den Ablauf des Zuweisungsschritts nicht verändern, für alle Algorithmen implementiert, auch wenn diese nicht Bestandteil der Originalveröffentlichung waren.

Der Exponion konnte nicht vollständig gemäß der Originalveröffentlichung implementiert werden, der Einsatz der binären Suche fehlt. Die Funktionsweise dieser binären Suche wurde zwar textuell beschrieben, es ist allerdings offen

geblieben, wie die partielle Sortierung der Center-Center-Distanzen für die binäre Suche in der geforderten Komplexitätsklasse erfolgen soll.

Durch diese strukturierte Herangehensweise wurde sichergestellt, dass kein Algorithmus durch seine Implementierung gegenüber anderen Algorithmen bevorteilt und die Ergebnisse dadurch verfälscht wurden.

Der so entwickelte Programmcode wurde an einer Vielzahl von Stellen um die Erhebung von Messdaten erweitert. Eine nicht abschließende Aufzählung dieser Messdaten inkludiert die Häufigkeit des Prunings mit Hilfe der einzelnen Pruningkriterien, die Anzahl der Distanzberechnungen und die benötigte Realzeit pro Iteration. Diese Messdaten wurden für alle Algorithmenvarianten für unterschiedliche Datensätze, Clusteranzahlen und Initialisierungen erhoben und anschließend ausgewertet. Ein Vergleich der Algorithmenvarianten mit Realzeit und der Anzahl der durchgeführten Distanzberechnungen als Leistungsmerkmal konnte die Erkenntnisse bestehender Literatur im Wesentlichen bestätigen. Für den Einsatz mit vielen Dimensionen empfiehlt sich die Verwendung des Elkan, für den Einsatz in niedrigen Dimensionen die Verwendung einer Hamerly-Variante. Der Yinyang erreicht mit zunehmender Anzahl an Clustern eine bessere Leistung. Auffälligkeiten in diesem Vergleich der externen Messdaten wurden als Basis für die Untersuchung der internen Messdaten genutzt.

Als Vorbereitung für die Untersuchung der internen Messdaten haben wir die Bewegungen der Clusterzentren in den verwendeten Datensätzen untersucht. Da alle untersuchten Algorithmen exakte Algorithmen sind, bewegen sich die Clusterzentren für alle Algorithmen auf identische Weise. Dabei konnten wir feststellen, dass die maximale Bewegung einzelner Clusterzentren mit zunehmender Anzahl von Clustern zunehmend stärker von der durchschnittlichen Zentrenbewegung abweicht. Derartige Big Mover sorgen bei Algorithmen ohne feingranulare untere Schranke für eine schlechte Pruningleistung.

Bei der abschließenden Betrachtung der *internen* Messdaten konnte über die Analyse der Pruningleistung der einzelnen Pruningkriterien in Abhängigkeit von der Iteration für den Elkan festgestellt werden, dass weniger mehr ist und warum weniger mehr ist. Der Simplified Elkan, der im Vergleich zum Elkan auf ein Pruningkriterium verzichtet, kann in der Praxis eine bessere Leistung erzielen.

Für den Drake stellt sich heraus, dass das „adaptive Tuning“ der Schrankenanzahl in der Praxis auf den von uns verwendeten Datensätzen in der Regel nicht sinnvoll funktioniert, aber auch keine Nachteile bietet.

Der Annulus erlaubte die Implementierung einer großen Anzahl von Varianten. Für die Auswertung von besonderem Interesse und in der uns vorliegenden Literatur bislang noch völlig unberücksichtigt war die Verschiebung des verwendeten Fixpunkts. Eine Verschiebung innerhalb einer einzelnen Dimension sorgt für bis zu 25 % Differenz in der Pruningleistung des Annulus. Wir haben die Ursache für diesen großen Einfluss der Position des Fixpunkts auf die Pruningleistung im Detail untersucht. Eine deutlich abweichende Beschaffenheit dieser einzelnen Dimension im Vergleich zu allen Dimensionen führt dazu, dass durch die Verschiebung des Fixpunkts weniger Clusterzentren gepruned werden können.

In dieser Untersuchung der internen Messdaten nicht berücksichtigt wurden Hamerly, Exponion und Yinyang. Für den Hamerly hat sich bereits bei dem direkten Vergleich der Algorithmen herausgestellt, dass dieser als echte Teilmenge des Annulus und Exponion in praktisch allen Fällen eine schlechtere Leistung als selbige erbringt. Darüber hinaus treffen Leistungsmerkmale des Hamerly identisch auf Annulus und Exponion zu. Im Falle des Exponion zeigte der direkte Vergleich keine Auffälligkeiten, die als Startpunkt für die detaillierte Untersuchung genutzt werden konnten. Die Leistung des Exponion entsprach im Wesentlichen der Erwartung. In Bezug auf die Realzeit sorgte die fehlende binäre Suche für eine schlechte Leistung und damit eine Verfälschung des Ergebnisses. Der Yinyang erreichte bei dem direkten Vergleich der Algorithmen mit zunehmenden k eine bessere relative Leistung. Die Auswertung der internen Messdaten lieferte aber keine eindeutigen Ergebnisse für den Grund dieser Leistungszunahme. Es konnte auch nicht klar festgestellt werden, ob die Leistung des Yinyang zunimmt oder ob die Leistung der anderen Algorithmen abnimmt.

Im Rahmen dieser Arbeit haben wir durch die systematische Herangehensweise in Bezug auf die Untersuchung von Algorithmen der k-means-Familie und der daraus resultierenden praktischen Implementierung ein stabiles und faires Grundgerüst zur Erfassung von Messdaten während des Clusterings geschaffen. Dadurch konnten wir bestehende Erkenntnisse einerseits bestätigen und auf der anderen Seite konnten wir die Gründe für das Verhalten der Algorithmen

anhand der erfassten Messdaten ermitteln und klar belegen. Für den Exponion und den Yinyang sind noch Fragen offen geblieben. Diese sollten im Rahmen zukünftiger Forschung auf Basis des von uns entwickelten Grundgerüsts, das um die Erfassung weiterer Messdaten erweitert wird, geklärt werden.

Eine Einschränkung unserer Resultate ist die durch die Bearbeitungszeit beschränkte geringe Anzahl an unterschiedlichen Initialisierungen mit k-means++. Ein Clustering aller Datensätze mit allen Algorithmenvarianten für die gewählten Clusteranzahlen benötigt für jeweils einen k-means++-Seed über 24 Stunden. Durch Fehler in der Implementierung, die erst durch die Auswertung aufgefallen sind, und durch Erweiterung der erfassten Messdaten im Rahmen der Auswertung ist mehrfach ein erneutes Clustering erforderlich gewesen, um korrekte und konsistente Resultate zu gewährleisten. Auf gleiche Weise ist die maximale Anzahl von Clustern mit 96 relativ gering gewählt. Für größere Clusteranzahlen wäre bei den größeren Datensätzen für die schwergewichtigeren Algorithmenvarianten mehr Hauptspeicher erforderlich gewesen. Das benötigte finanzielle Budget für die genutzte Cloud-Hardware hätte sich dadurch verdoppelt. Eine Beurteilung, ob die beobachteten Trends, wie beispielsweise die Leistungszunahme des Yinyang, sich für größere k fortsetzen, ist daher nicht abschließend möglich gewesen.

Den Einfluss der Position des Fixpunkts auf die Pruningleistung des Annulus konnten wir belegen und die Ursache dahinter ermitteln. Es konnte im Rahmen dieser Arbeit aber kein Verfahren entwickelt werden, wie auf Basis des genutzten Datensatzes die Position des Fixpunkts so gewählt werden kann, dass sich die Leistung des Annulus gegenüber einer Position im Ursprung im Regelfall verbessert.

Es wird deutlich, dass die Möglichkeiten der Beschleunigung von k-means-Clustering durch exakte, schrankenbasierte Algorithmen noch nicht ausgeschöpft sind. Diese Arbeit konnte aufgrund ihres Fokus und der zur Verfügung stehenden Ressourcen nicht alle Fragen abschließend klären, liefert aber einen entscheidenden Beitrag für zukünftige Forschung in diesem Bereich.

A

Datensätze

Eine besondere Schwierigkeit, die sich bei der Überprüfung der Resultate von anderen Veröffentlichungen ergibt, ist, dass Testdaten und / oder die eingesetzen Programme nicht zur Verfügung stehen. An dieser Stelle finden sich daher die Quellen der verwendeten Datensätze und ihre kryptografischen Prüfsummen. Es kann so leicht überprüft werden, ob der identische Datensatz vorliegt. Falls eine Quelle zukünftig nicht mehr zur Verfügung steht, kann ein Datensatz mit Hilfe der Prüfsumme möglicherweise auch über Magnet-Links für BitTorrent oder das InterPlanetary File System (IPFS) bezogen werden.

birch

http://cs.joensuu.fi/sipu/datasets/ [FS18]

```
1 MD5 (birch1.txt) = 98a199d85ad4fc3cae4437008712a294
2 SHA1 (birch1.txt) = 486d2f0becdc6265f0c91cf26f52ef3ab0241a0f
3 SHA256 (birch1.txt) = 95230d302b2ffbe15de77f732af7002b037887
4 c30c19b1539999dedfe3587400
5 SHA512 (birch1.txt) = 9db82d02d0771c3c7b922f4320215f26a81d77
6 d5ea05a7e19ad8d0d0e832f11937f66186d0a7cc898e5813764522d45842
7 717b543d35f53aedfab014a3f083c8
```

colormoments

https://archive.ics.uci.edu/ml/datasets/corel+image+features [DG17]

```
1 MD5 (ColorMoments.asc) = 8c05f6e9e4a15baa07d61ab736dfdd07
2 SHA1 (ColorMoments.asc) = cdf1b9a8ba5bbb30d1954b7b104f84dbdf
3 0b7870
4 SHA256 (ColorMoments.asc) = 4353b76cd32d4e52617e287f05e0cf72
5 b6d3dff58542af55ec36a5300ecbf07d
6 SHA512 (ColorMoments.asc) = afbef12d829193003cfe57d74cb855d5
```

```
76b0464bdfe78ea650bf57531454bf54c0752fe082be874a2899d8072b0e
013e2879ee2e58555989250a1c3ca15c3e8e
```

conflongdemo

http://cs.joensuu.fi/sipu/datasets/ [FS18]

```
MD5 (ConfLongDemo_JSI_164860.txt) = 4adae4725b198b7b2cf5b052
73821fa5
SHA1 (ConfLongDemo_JSI_164860.txt) = 5edd02bd3956fc23331e388
2c1c2ec7b996e02ce
SHA256 (ConfLongDemo_JSI_164860.txt) = 75628aef681b9cdcf1abc
08a392abf094d374e779b8d5566c4a8ebe064f52b37
SHA512 (ConfLongDemo_JSI_164860.txt) = cccc7147ddd1d0ed4d18e
6117ec257cfe11ec7561aaed81c05bb62485d2973f8afea3c704d21b6ed1
291a4d8dc324abae30b8f9a7e1d7e62a4a6a00642164213
```

covtype

http://archive.ics.uci.edu/ml/datasets/covertype [DG17]

```
MD5 (covtype.data) = 71df19898bd3e11db15ae0faf4159f2c
SHA1 (covtype.data) = 701f84a08505d5aea6870e48294585f6c7326a
f1
SHA256 (covtype.data) = 0a9371cef7c964b5475d6053cc3e0894a5aa
6f65ad1ed3ecb01c45aa96217945
SHA512 (covtype.data) = 4e77e1705c2040512ade4456bcb5422a2e00
b7a6ad2abc324a72a249d7a174a32a09368c76faee951ecd983871d17b0a
7e198654257d2eba4d542cad9bd1f2f7
```

house16h

http://funapp.cs.bilkent.edu.tr/DataSets/

```
MD5 (HH.dat) = c0d06d7e29bb86b12b5fac5748b92244
SHA1 (HH.dat) = 894893e21d257bcab4c6e5c123585af08b0a0551
SHA256 (HH.dat) = 8c85bdaf6bb3b096fe06cd50bbf9fbbd5513a179ad
b81465b73d0bebd4b845c8
SHA512 (HH.dat) = 6a0b89a2db1bf2f7cd014cc4c68944f31f3ac3ce0a
a7ce82d1d72bda9c9f50774e85f3ac2bea5af744137932dee2b031826971
8e5da07c12a6af038515a23f18
```

kddcup04

http://cs.joensuu.fi/sipu/datasets/ [FS18]

```
MD5 (KDDCUP04Bio.txt) = 7042bef84dc482c8e1e40af1c8388eae
SHA1 (KDDCUP04Bio.txt) = d5ee968039e3859903517fc2b819b433a4f
76e1f
SHA256 (KDDCUP04Bio.txt) = 32e82eb6afba4072a3858b63503ae8ab0
34a90d24e03495a499d0ccb5b4f1dd3
SHA512 (KDDCUP04Bio.txt) = f8efa8cd8e85a1e7e21f63de000ebc0ae
bc52e3454c1e23cf20c32313cb326c5d4136af219877d5b3aa4a1bc1aa08
c6b5a1646228fe4e02fb2fe61030b189111
```

mnist784

http://yann.lecun.com/exdb/mnist/

```
MD5 (train-images-idx3-ubyte) = 6bbc9ace898e44ae57da46a32403
1adb
SHA1 (train-images-idx3-ubyte) = c3557c10f29b266e19b3eeee155
3c85e0ef4a8ea
SHA256 (train-images-idx3-ubyte) = ba891046e6505d7aadcbbe256
80a0738ad16aec93bde7f9b65e87a2fc25776db
SHA512 (train-images-idx3-ubyte) = 0c574eb011cd10a30a29887cb
7614a092e948881c3fa6a94b2c840413ba5363a99ff10274cb1790852e4b
f8fa2aa6c29d2bba3fa3b20b58ca8f381cf004fd478
```

s1

http://cs.joensuu.fi/sipu/datasets/ [FS18]

```
MD5 (s1.txt) = aadba37ab4b91b43320b6f1a44d61768
SHA1 (s1.txt) = a40ff718983669d42546366b241b981e86c19bbb
SHA256 (s1.txt) = d98ddffe6ad4ff67babf1aacd1b679e7b858c8c97b
e1e51c818fe5f4552664af
SHA512 (s1.txt) = 6a0f93fd661f3801623746757ee4cb842ab6f688db
ff2acdd7e3d240dbd20c792f01ab0adbe5226b7895634f08d286aead2792
113e6bf3b02763dac26da57454
```

uscensus

http://archive.ics.uci.edu/ml/datasets/us+census+data+(1990) [DG17]

```
1 MD5 (USCensus1990.data.txt) = 13852241ffbd1bced75d4625e6ce9a
2 86
3 SHA1 (USCensus1990.data.txt) = ce03021c23ec87a90404ff772e522
4 ec5df248d5f
5 SHA256 (USCensus1990.data.txt) = 38e99f50855ddc03167b715b365
6 19ef40b77295ae4e85dfef0c6429ac79ed799
7 SHA512 (USCensus1990.data.txt) = a905c1a43d21c1330f26f3a4cab
8 c8971fadf6859a759eb9b3dca995028be83be21a28c866191311cfd4005f
9 6cada28d5e57693dc1f671071d209f1086305b1ee
```

B

Inhalt der beigelegten CD-ROM

Auf der beigelegten CD-ROM finden sich, neben einer digitalen Version dieser Arbeit, die in Kapitel 4 vorgestellten Datensätze, der Quelltext der Implementierung, die Rohdaten der Auswertung und die zur Auswertung verwendeten Hilfsprogramme.

Die Quelldateien sind UTF-8-kodiert und haben Unix-Zeilenenden (LF).

Vergl_Untersuchung_von_exaktem_durch_Schr_beschl_k_means_Clustering.pdf Dieses Dokument.

datasets.tar.gz gzip-komprimiertes Tar-Archiv der Datensätze.

impl/ Quelltext der Implementierung.

impl.tar.gz gzip-komprimiertes Tar-Archiv des `impl/`-Ordners.

results.tar.gz gzip-komprimiertes Tar-Archiv der Rohdaten der Auswertung.

Verzeichnis der Pruningkriterien

Definitionsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Listings

Tabellenverzeichnis

Abkürzungsverzeichnis

CPU	Central Processing Unit
DBSCAN	Density-based spatial clustering of applications with noise
GNU	GNU's Not Unix
HSV	Farbwert (Hue), Sättigung (Saturation), Hellwert (Value)
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
KDDCUP	Knowledge Discovery and Data Mining Competition
KVM	Kernel-based Virtual Machine
NVMe	NVM Express
PAM	Partitioning Around Medoids
RSS	Resident Set Size
SSD	Solid-state Drive
SSE	Streaming SIMD Extensions
STL	Standard Template Library

Literatur

[AV07] David Arthur und Sergei Vassilvitskii. „K-means++: The Advantages of Careful Seeding“. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. New Orleans, Louisiana: Society for Industrial und Applied Mathematics, 2007, S. 1027–1035. ISBN: 978-0-898716-24-5. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

[BMS96] P. S. Bradley, O. L. Mangasarian und W. N. Street. „Clustering via Concave Minimization“. In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. NIPS'96. Denver, Colorado: MIT Press, 1996, S. 368–374. URL: http://dl.acm.org/citation.cfm?id=2998981.2999033.

[Bon] Jonas Bonér. *Latency Numbers Every Programmer Should Know*. URL: https://gist.github.com/jboner/2841832 (besucht am 05. 12. 2019).

[CKV12] M. Emre Celebi, Hassan A. Kingravi und Patricio A. Vela. „A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm“. In: *CoRR* abs/1209.1960 (2012). arXiv: 1209.1960. URL: http://arxiv.org/abs/1209.1960.

[CPL17] Marco Capó, Aritz Pérez und Jose A. Lozano. „An efficient approximation to the K-means clustering for massive data“. In: *Knowledge-Based Systems* 117 (2017). Volume, Variety and Velocity in Data Science, S. 56–69. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2016.06.031. URL: http://www.sciencedirect.com/science/article/pii/S0950705116302027.

[cpp] cppreference.com. *Move constructors*. URL: https://en.cppreference.com/w/cpp/language/move_constructor (besucht am 16. 09. 2019).

[DEC] DE-CIX Management GmbH. *DE-CIX Frankfurt statistics*. URL: https://www.de-cix.net/de/locations/germany/frankfurt/statistics (besucht am 20. 08. 2019).

[DG17] Dheeru Dua und Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[Din+15] Yufei Ding u. a. „Yinyang K-means: A Drop-in Replacement of the Classic K-means with Consistent Speedup“. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, S. 579–587. URL: http://dl.acm.org/citation.cfm?id=3045118.3045181.

[DLMF] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.23 of 2019-06-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds. URL: http://dlmf.nist.gov/.

[Dra12] Jonathan Drake. „Accelerated k-means with adaptive distance bounds“. In: 2012.

[Dra13] Jonathan Drake. „Faster k-means Clustering“. Magisterarb. Baylor University, Sep. 2013.

[Elb] Ron Elber. *Description of the KDD-Cup 2004 Protein Data*. URL: http://osmot.cs.cornell.edu/kddcup/protein_description.pdf (besucht am 24. 10. 2019).

[Elk03] Charles Elkan. „Using the Triangle Inequality to Accelerate K-means“. In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, S. 147–153. ISBN: 1-57735-189-4. URL: http://dl.acm.org/citation.cfm?id=3041838.3041857.

[Est+96] Martin Ester u. a. „A density-based algorithm for discovering clusters in large spatial databases with noise“. In: AAAI Press, 1996, S. 226–231.

[FS18] Pasi Fränti und Sami Sieranoja. *K-means properties on six clustering benchmark datasets*. 2018. URL: http://cs.uef.fi/sipu/datasets/.

[FV06] P. Fränti und O. Virmajoki. „Iterative shrinking method for clustering problems“. In: *Pattern Recognition* 39.5 (2006), S. 761–765. DOI: 10.1016/j.patcog.2005.09.012. URL: http://dx.doi.org/10.1016/j.patcog.2005.09.012.

[Ges15] Gesamtverband der Deutschen Versicherungswirtschaft e.V. *Die Geschichte der Datenanalyse*. Nov. 2015. URL: https://www.gdv.de/de/themen/news/infografik---die-geschichte-der-datenanalyse-17774 (besucht am 19. 08. 2019).

[Gz] Cristhian (https://math.stackexchange.com/users/129028/cristhian-gz) Gz. *Formal proof that mean minimize squared error function*. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/967182 (version: 2014-10-10). eprint: https://math.stackexchange.com/q/967182. URL: https://math.stackexchange.com/q/967182 (besucht am 01. 09. 2019).

[Ham10] Greg Hamerly. „Making k-means even faster“. In: *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM International Conference on Data Mining, 2010, S. 130–140. DOI: 10.1137/1.9781611972801.12. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611972801.12. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972801.12.

[HD17] Greg Hamerly und Jonathan Drake. „Chapter 2 Accelerating Lloyd' s Algorithm for k-Means Clustering“. In: 2017.

[IEEE 754] „IEEE Standard for Floating-Point Arithmetic“. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (Juli 2019), S. 1–84. DOI: 10.1109/IEEESTD.2019.8766229.

[Kan+00] Tapas Kanungo u. a. *The analysis of a simple k-means clustering algorithm*. Techn. Ber. MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE, 2000.

[Kan+02] Tapas Kanungo u. a. „An Efficient k-Means Clustering Algorithm: Analysis and Implementation“. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.7 (Juli 2002), S. 881–892. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2002.1017616. URL: http://dx.doi.org/10.1109/TPAMI.2002.1017616.

[KFN00] T. Kaukoranta, P. Franti und O. Nevalainen. „A fast exact GLA based on code vector activity detection“. In: *IEEE Transactions on Image Processing* 9.8 (Aug. 2000), S. 1337–1342. DOI: 10.1109/83.855429.

[KJ09] Leonard Kaufman und Peter J Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Sep. 2009. ISBN: 9780470317488.

[Low+12] Yucheng Low u. a. „Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud“. In: *Proc. VLDB Endow.* 5.8 (Apr. 2012), S. 716–727. ISSN: 2150-8097. DOI: 10.14778/2212351.2212354. URL: https://doi.org/10.14778/2212351.2212354.

[Mar18] Bernard Marr. *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. Mai 2018. URL: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#1131172760ba (besucht am 20. 08. 2019).

[MNV12] Meena Mahajan, Prajakta Nimbhorkar und Kasturi Varadarajan. „The planar k-means problem is NP-hard“. In: *Theoretical Computer Science* 442 (2012). Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009), S. 13–21. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2010.05.034. URL: http://www.sciencedirect.com/science/article/pii/S0304397510003269.

[NF16] James Newling und François Fleuret. „Fast K-means with Accurate Bounds“. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, S. 936–944. URL: http://dl.acm.org/citation.cfm?id=3045390.3045490.

[RA15] Ryan A. Rossi und Nesreen K. Ahmed. „The Network Data Repository with Interactive Graph Analytics and Visualization“. In: *AAAI*. 2015. URL: http://networkrepository.com.

[RH] Petr Ryšavý und Greg Hamerly. „Geometric methods to accelerate k-means algorithms“. In: *Proceedings of the 2016 SIAM International Conference on Data Mining*, S. 324–332. DOI: 10.1137/1.9781611974348.37. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611974348.37. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611974348.37.

[Scu10] D. Sculley. „Web-scale K-means Clustering“. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, S. 1177–1178. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772862. URL: http://doi.acm.org/10.1145/1772690.1772862.

[Sim17] Radu Simion. Jan. 2017. URL: https://www.bnrbeurs.nl/2017/01/20/algorithmic-trading-the-milliseconds-that-bring-millions/ (besucht am 29. 08. 2019).

[SZ19] Erich Schubert und Arthur Zimek. *ELKI: A large open-source library for data analysis - ELKI Release 0.7.5 "Heidelberg"*. 2019. arXiv: 1902.03616 [cs.LG].

[War] Ian Ward. *JSON Lines - Documentation for the JSON Lines text file format*. URL: http://jsonlines.org/ (besucht am 31. 10. 2019).

[Wu+08] Xindong Wu u. a. „Top 10 algorithms in data mining“. In: *Knowledge and Information Systems* 14.1 (Jan. 2008), S. 1–37. ISSN: 0219-3116. DOI: 10.1007/s10115-007-0114-2. URL: https://doi.org/10.1007/s10115-007-0114-2.

[ZRL97] T. Zhang, R. Ramakrishnan und M. Livny. „BIRCH: A new data clustering algorithm and its applications“. In: *Data Mining and Knowledge Discovery* 1.2 (1997), S. 141–182.

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *Eine vergleichende Untersuchung von exaktem, durch Schranken beschleunigtem k-means-Clustering* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Tim Wolfgang Düsterhus, Münster, 9. Januar 2020

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Tim Wolfgang Düsterhus, Münster, 9. Januar 2020