Ein skalierbares agglomeratives Clusterverfahren basierend auf erste Nachbarschaftsbeziehungen

Dissertation

zur Erlangung des Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

der Technischen Universität Dortmund an der Fakultät für Informatik

von

Alireza Moradpour

Dortmund

2017

Tag der mündlichen Prüfung: 28.04.2017
Dekan: Prof. Dr.-Ing. Gernot A. Fink
Gutachter/Gutachterinnen: Prof. Dr. Günter Rudolph
Prof. Dr. Kristian Kersting

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen verwendet habe.

Danksagung

Diese Arbeit ist an der Fakultät für Informatik der Universität Dortmund enstanden.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr. Günter Rudolph, Prof. Dr. Kristian Kersting und Prof. Dr. Miriam Minor für deren Anregungen und Unterstützung bei der Ausarbeitung meiner Dissertation. Sie standen mir mit hilfreichen Korrekturen zur Seite und haben mich motiviert, diese Arbeit zu Ende zu schreiben.

Außerdem möchte ich meiner Familie für ihren seelischen Rückhalt danken. Sie waren mir in schwierigen Zeiten eine sehr große Hilfe und haben mich ermuntert, weiterzumachen.

Inhaltsverzeichnis

1. Einleitung

Eine der revolutionären Veränderungen des 20. und 21. Jahrhunderts der Menschheit ist die Computerisierung und Digitalisierung der Welt. Tagtäglich werden große Menge von Daten erzeugt.

Es ist unmöglich für uns Menschen, ohne anspruchsvolle Verfahren und Algorithmen, aus diesen enorm großen Datenmengen versteckte Strukturen (Wissen) herauszufiltern. Datamining, Machine Learning und Pattern Recognition sind ähnliche Disziplinen, mit denen man versuchen will, diese Herausforderung zu bewältigen.

Da das Volumen der Daten von Sekunde zu Sekunde größer wird, brauchen wir wie nie zuvor Algorithmen, die skalierbar sind.

In dieser Arbeit konzentrieren wir uns auf zwei wichtige, interdisziplinäre Lernverfahren, nämlich Supervised (Klassifikation) und Unsupervised (Clustering) Learning.

Unser Ziel in dieser Arbeit ist die schrittweise Entwicklung von skalierbaren Algorithmen. Dazu erklären wir erst die Grundidee der Algorithmen und beweisen diese anschließend formal. Im nächsten Schritt beweisen wir die Korrektheit der Algorithmen. Dann analysieren wir entweder ihre Laufzeit, oder untersuchen ihr Verhalten empirisch.

Wir benutzen die erste Nachbarschaftsbeziehung zwischen Punkten als Grundoperation, mit der wir weitere Algorithmen für beide Lernverfahren, nämlich Klassifikation und Clustering, schreiben.

1.1. Supervised und Unsupervised Learning

In diesem Kapitel möchten wir Supervised (Klassifikation) und Unsupervised Learning (Clustering) jeweils in ihrer Art erklären und anschließend deren Methodik erläutern.

Das Verfahren zur Entdeckung von Strukturen in Datenbeständen nennt man Clusteranalyse. Damit werden Objekte innerhalb einer Gruppe (Struktur) bestmöglichst ähnlich und Objekte der verschiedenen Gruppen möglichst unähnlich zueinander zusammengefasst [1, 2, 3, 4].

Im Folgenden beschreiben wir die am meisten benutzten Clusteringverfahren:

1. Partitionierende Methoden:

   Gegeben seien n Datensätze und k mit $k \leq n$. Eine partitionierende Methode partitioniert nun die Datenmenge in k Anfangscluster, wobei jedes Cluster mindestens einen Datensatz hat. Dann vertauscht man, um eine bessere

Cluster-Qualität zu erreichen, die Datensätze der Cluster solange miteinander, bis eine durch eine Heuristik festgelegte globale Bedingung nicht mehr gilt. Partitionierende Methoden sind meist distanzbasiert, der Nachteil dieser Methoden ist, dass man nur vom Distanzmaß abhängige Formen von Cluster finden kann. Es gibt auch Varianten, die Punktwolken in Form von Ellipsoiden oder Ringen identifizieren. Die bekanntesten partitionierenden Algorithmen sind k-Mean [5], PAM und CLARA [4].

2. Hierarchische Methoden:

   Wir können im allgemeinen hierarchische Methoden in zwei Arten unterteilen, agglomerativ und teilend. Bei beiden Methoden konstruiert man eine hierarchische, baumartige Zusammensetzung von Datenmengen.

   Detaillierter kann gesagt werden, dass man bei dem agglomerativen Ansatz mit der Gruppierung der Datensätze beginnt, indem am Anfang jeder Datensatz eine Gruppe repräsentiert. Dann vereinigt man sukzessiv Gruppen, die einer Ähnlichkeitsfunktion nach nah zueinander sind, bis entweder alle Datensätze in einer Gruppe sind oder eine allgemeine Bedingung nicht mehr gilt.

   Umgekehrt ist dies bei der teilenden Methode, bei der man erst alle Datensätze der Datenmenge in eine Gruppe setzt, und daraufhin die Datensätze der Gruppe nach ein Unähnlichkeitsfunktion in verschiedene Gruppen sukzessiv zerlegt, bis entweder in jeder Gruppe ein Datensatz vorhanden ist oder eine allgemeine Bedingung nicht mehr gilt. Die bekanntesten hierarchischen Methoden sind CURE [6], CHAMELEON [7] und BIRCH [8].

3. Dichtenbasierte Methoden:

   Bei dichtenbasierten Methoden betrachtet man die Cluster als Bereiche oder Punktwolken innerhalb des Eingaberaumes, die im Vergleich mit anderen Bereichen des Eingaberaumes mehr Datensätze oder Punkte besitzen, oder, in anderen Worten, dichter sind.

   Detaillierter bedeutet dies, dass man zwei Parameter definiert, einen Radius r und eine Dichte d, damit jedes Element eines Clusters innerhalb des Radius r mindestens d Nachbarn haben muss. Der Vorteil dieser Art von Betrachtung der Cluster, im Gegenteil zu partitionierenden Methoden, die nur sphärisch geformte Cluster finden können, ist, dass man beliebige Formen von Clustern unterscheiden oder finden kann. Außerdem kann man so auch Ausreißer innerhalb der Datensätze feststellen. Die bekanntesten dichtenbasierten Methoden sind OPTICS [9] und DBSCAN [10].

Beim Supervised Learning haben wir eine andere Art von Datenmengen oder Samples. Gegeben sind die Datensätze $x_i \in X$, $i = 1, 2, \ldots, n$, für die jeweils ein Klassenlabel oder Wert y_i zugeordnet ist, sodass $X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ ist. Unser Ziel ist es, dass wir erst durch eine Stichprobe S der Datenmenge $S \subseteq X$ den Zusammenhang zwischen x und y verstehen. D.h. wir lernen eine Funktion $f : y = f(x)$, mit deren Hilfe wir dann in der Lage sind, für einen beliebigen neuen Datensatz $x_{n+t} \in X$, $t = 1, 2, \ldots$ dessen unbekanntes Klassenlabel oder Wert y_{n+t}

festzustellen oder vorhersagen zu können [11, 12, 13, 14].

Die zurzeit am meisten genutzten Klassifikationstechniken sind Entscheidungsbäume [15, 16, 17, 18, 19], Bayesian classification [12, 14, 20] und Bayesian belief networks [23, 24, 25, 26], neuronale Netze [27, 28, 29, 30], K-nächster Nachbar reasoning [20, 31, 32, 33], SVM [59, 60], Random Forest [61, 62], rough sets [38, 39, 40] und fuzzy Logiktechniken [41, 42, 43].

Da wir in dieser Arbeit für die K-nächste Nachbar-Klassifikation einen Algorithmus präsentieren, beschreiben wir diesen Klassifikator hier:

Bei der K-nächsten Nachbar-Klassifikation geht man von einem d-dimensionalen euklidischen Datenraum aus, in dem jeder Datensatz einen Punkt repräsentiert. Falls wir einen neuen Datensatz x_u erhalten, dessen Klassenlabel oder Wert y_u unbekannt ist, wird bei dieser Klassifikation wie folgt vorgegangen:

Wir finden die K-nächsten Nachbarn $x_1, x_2, \ldots, x_k$ mittels einer Distanzfunktion, deren Klassenlabel oder Wert jeweils $y_1, y_2, \ldots, y_k$ sind, zu diesem Punkt x_u im Eingaberaum und entscheiden im Falle eines Klassenlabels mittels einer Mehrheitsabstimmung. D.h. innerhalb der K-nächsten Nachbarn $x_1, x_2, \ldots, x_k$, bei denen ein Klassenlabel überwiegt, wird die Unbekannte y_u diesem Klassenlabel zugeordnet. Wenn aber der neue Punkt x_u anstelle eines Klassenlabels einen Wert y_u hat, wird y_u gleich dem Mittelwert aus allen zugeordneten Werten $y_1, y_2, \ldots, y_k$ der K-nächsten Nachbarn $y_u = \frac{\sum_{i=1}^{k} y_i}{k}$ gesetzt.

Das K-nächsten Nachbar-Problem hat eine signifikante Bedeutung in verschiedenen Gebieten der Informatik, wie Pattern Recognition, Suchen in Multimedia-Daten, Vector Compression, Computational Statistics, und Data Mining, für deren Anwendungen große Datenmengen existieren. Diese Tatsache macht den K-nächsten Nachbar-Ansatz sehr ansprechend. Aber auf der anderen Seite wächst die Sorge im Bezug auf dessen Berechnungsaufwand. Daher ist es sehr wichtig, Algorithmen für das K-nächsten Nachbar-Problem zu entwickeln, die immer noch effizient sind, wenn die Anzahl der Datensätze und die Dimension der Daten wachsen. Falls wir die Datenmenge X mit N-Datensätzen $X = \{x_1, x_2, \ldots, x_N\} : x_i = (a_1, a_2, \ldots, a_d) \in \mathbb{R}^d$, $i = 1, 2, \ldots, N$ haben, dann lautet ein naiver Algorithmus für dieses Problem wie folgt:

Gegeben ist ein Punkt $x_i \in X$, dann berechne die Distanz zwischen x_i und allen anderen Punkten in X und finde einen Punkt $x_j \in X$, der diese Distanz minimiert. Diese lineare Suche hat die Query-Zeit von $O(dN)$. Dieser Aufwand ist für große Datenmenge unakzeptabel. Das "Ziel" der Forschung auf diesem Gebiet ist es, einen Algorithmus für das Problem zu entwerfen, der eine sublineare (oder sogar logarithmische) Worst-Query-Laufzeit für Datensätze mit beliebiger Dimensionsgröße erreicht. Das K-nächste Nachbar-Problem wurde ausgiebig auf dem Gebiet der algorithmischen Geometrie untersucht.

Als Ergebnis dieser Forschungen wurden viele effiziente Lösungen für den Fall erzielt, wenn die Datensätze eine konstante und kleine Dimension haben. Wenn $d = 1$, d.h wenn die Datensätze auf einer reellen Linie sind, wird das nächste Nachbar-Problem das klassische Vorgängerproblem. Hierfür kann man alle Datensätze sortieren und dann

eine binäre Suche darauf anwenden. Damit brauchen wir nur $O(N)$ Speicherplatz und $O(log\, N)$ Query-Zeit. Falls Datensätze in der Ebene liegen, kann das nächste Nachbar-Problem mit einer Laufzeit von $O(logN)$ pro Query gelöst werden, wobei nur $O(N)$ Speicherplatz gebraucht wird [47, 48]. Insbesondere hat das nächste Nachbar-Problem eine Lösung in $O(d^{O(1)}logN)$ Query-Zeit, jedoch unter Verwendung von $N^{O(d)}$ Speicherplatz [49, 50]. Ein solcher Speicherplatz ist z.B. für eine Millionen Datensätze mit einer Dimensiongröße $d > 3$ unpraktisch.

Es existieren praktikable Algorithmen für Datensätze mit bis zu 10 oder 20 Dimensionen, die auf eine Datenstruktur angewendet werden. Die erste derartige Datenstruktur, genannt kd-Baum, wurde 1975 von Jon Bentley [51] eingeführt. Seitdem wurden viele andere multidimensionale Datenstrukturen entwickelt, einschließlich dem R-Baum, R*-Baum, X-Baum, SS-Baum, SR-Baum, VP-Baum und dem Metrik-Baum [52].

Das Prinzip von all diesen Algorithmen, die auf die oben genannten Datenstrukturen angewendet werden, ist es, dass man von der Wurzel bis zu einem Blatt travesiert. Gleichzeitig versucht man, so weit wie möglich, zunächst mittels Heuristiken einen Teilbaum auszuwählen und damit andere Teilbäume aus der Suche auszuschließen, um durch eine kleinere zu durchsuchende Menge eine gezieltere Suche zu erreichen.

Leider werden die Algorithmen immer weniger effizient, wenn die Anzahl der Dimensionen wächst. Genauer gesagt wächst der Suchraum und damit die Anzahl der untersuchten Datensätze exponentiell in Abhängigkeit zur Dimension. Um dieses Problem zu überwinden[1], wurden in den letzten Jahren von Forschern Methoden vorgeschlagen, Näherungslösungen anstelle der exakten Lösung zu finden [53, 54]. In dieser Formulierung ist es dem Algorithmus erlaubt, einen Punkt, dessen Abstand von dem Query-Punkt höchstens c-Mal größer ist als der Abstand von dem Query-Punkt zu seinem exakten nächsten Nachbar-Punkt, zurückzugeben. Man bezeichnet $c : c > 1$ als Annäherungsfaktor. Der Reiz dieses Ansatzes ist es, dass in vielen Fällen ein ungefährer nächster Nachbar fast so gut ist wie der genaue. Einer der populärsten Algorithmen für die approximative Lösung des nächsten Nachbarproblems basiert auf dem Konzept von locality-sensitive hashing (LSH) [59]. Die Grundidee bei diesem Ansatz ist, dass man mehrmals hintereinander die Datensätze durch verschiedene Hash-Funktionen $h_i : h_i \in H$, $i = 1, 2, \ldots, t$ hasht, damit sich nach der Anwendung jeder Hash-Funktion $h_i(.)$ die Wahrscheinlichkeit, dass Kollisionspunkte die nächsten Nachbarspunkte zueinander sind, im Vergleich zu der Wahrscheinlichkeit, dass Nicht-Kollisionspunkte die weit voneinander liegenden Punkte sind, erhöht.

Man kann den nächsten Punkt für den Query-Punkt q finden, indem man den Query-Punkt q mehrmals hasht und innerhalb der Kollisionspunkte denjenigen Punkt mit dem kleinsten Abstand zu q sucht. Die Wahl der Hash-Funktionen ist sehr wichtig und setzt Vorkenntnisse über die Datenmenge voraus.

In dieser Arbeit haben wir für das nächste Nachbarproblem zwei deterministische Algorithmen (Algorithmus 1 und 2) entwickelt. Unsere Algorithmen funktionieren mit

[1] Die Wirkung der Dimension auf den Suchraum.

einer einfachen Datenstruktur (Matrix). Algorithmus 1 kann parallel durchgeführt werden.

2. Begriffsdefinitionen

In diesem Kapitel definieren wir Begriffe, die in der gesamten Arbeit benutzt werden.
Sei im Folgenden $X = \{x_1, x_2, \ldots, x_N\}$, N-Punkte in einem d-dimensionalen Raum mit $x = (a_1, a_2, \ldots, a_d)$, wobei a_j Werte von Attributen A_j sind, d.h.
$x \in A_1 \times A_2 \times \ldots \times A_d$, $a_j \in A_j$, $j = 1, 2, \ldots, d$. Über die Verteilung der Punkte im Raum wissen wir nichts. Es gibt nur ein vernünftiges Unähnlichkeitsmaß, mit dem wir den Abstand zwischen den zwei Punkten x_i und x_j im Raum feststellen können.
Seien $dist(x, y)$ und $n(x, k)$ wie folgt definiert:

Definition 1 *-$dist(x, y)$*
$dist(x, y)$ gibt den Abstand zwischen den Punkten x und y zurück.

Definition 2 *-$n(x, k)$*
$n(x, k)$ gibt den k-ten Nachbarn von Punkt x zurück $k = 1, 2, \ldots, m$, so dass:
$dist(x, n(x, 1)) < dist(x, n(x, 2)) < dist(x, n(x, 3)) < \ldots$

Falls mehrere k-te Nachbarn existieren, ist diese Definition mathematisch nicht sauber. Wir benutzen trotzdem die Funktion $n(x, k)$ in unserer Arbeit, erstens weil unsere Ergebnisse weiterhin korrekt bleiben, zweitens um den Formalismus soweit wie möglich einfach zu halten.

Bevor wir uns Gedanken über einen Algorithmus zur Bestimmung der ersten Nachbarschaftsbeziehung machen, versuchen wir zunächst die beiden Funktionen $dist(.)$ und $n(.)$ besser zu verstehen, da dies die einzigen Informationen sind, welche wir bezüglich der Menge X haben.

$dist(.)$ ist eine Funktion, die bedingt durch ihre Anwendung verschieden definierbar ist und folgende Eigenschaften haben kann:

1. $dist(.,.) \geq 0$ und $dist(x, x) = 0$
2. $dist(x, y) = dist(y, x)$
3. $dist(x, y) \leq dist(x, z) + dist(z, y)$
4. Die erste und zweite Bedingung sind eine Grundvoraussetzung unserer Arbeit. Die dritte Bedingung allerdings kann je nach Anwendung nicht mehr berücksichtigt werden.

Für die Nachbarschaftsfunktion $n(.)$ definieren wir ein paar Begriffe:

1. Wir setzen $n(x, 0) = x$.

2. Sei $N_k(x)$, $k \geq 1$ die Menge der k-ten Nachbarn von x definiert als: $N_k(x) := \{y \,|\, n(x,k) = y\}$. Hier folgern wir sofort, falls $y_1 \in N_{k-1}(x)$ und $y_2 \in N_k(x)$ dann $dist(x, y_1) < dist(x, y_2)$.
3. Für die Punkte x, y, falls $n(x,k) = y$ und $n(y,k) = x$, $k \geq 1$, dann sagen wir x und y haben symmetrische Nachbarschaftsbeziehungen mit Grad k zueinander und wir bezeichnen diese Beziehung als $x \sim^k y$ und $x \nsim^k y$, falls sie unter x und y nicht besteht.
4. Sei $N_k^s(x)$, $k \geq 1$ die Menge der Nachbarn von x, die eine symmetrische Nachbarschaftsbeziehung mit Grad k zu x haben.
   $N_k^s(x) := \{y \,|\, n(x,k) = y,\ x \sim^k y\}$
5. Sei $\overrightarrow{N_k}(x)$, $k \geq 1$ die Menge der ausgehenden Nachbarschaft von x definiert als:
   $\overrightarrow{N_k}(x) := \{y \,|\, n(x,k) = y,\ x \nsim^k y\}$
6. Folglich setzen wir die Menge der k-ten Nachbarschaft von x gleich:
   $N_k(x) = N_k^s(x) \bigcup \overrightarrow{N_k}(x)$
7. Seien $D(N_k^s(x))$, $D(\overrightarrow{N_k}(x))$ jeweils die Distanz zwischen x und seinem k-ten symmetrischen, seinem k-ten ausgehenden Nachbarn y.
   $D(N_k^s(x)) = dist(x,y)$, $n(x,k) = y$, $x \sim^k y$
   $D(\overrightarrow{N_k}(x)) = dist(x,y)$, $n(x,k) = y$, $x \nsim^k y$

Hier beweisen wir noch ein paar Eigenschaften der Nachbarschaftsfunktion $n(.)$.

Korollar 1 -

1. *Für die $k-$te Nachbarschaft kann man eine Ordnung definieren. D.h. für Punkte x_1, x_2, x_3, x_4, die paarweise ungleich sind, mit $n(x_1,k) = x_2$, $n(x_3,k) = x_4$ gilt: $dist(x_1,x_2)$ ist entweder kleiner/gleich oder größer als $dist(x_3,x_4)$.*
2. *Falls $n(x_i,s) = x_p$ und $dist(x_i,x_p) \leq dist(x_j,x_p)$ dann gilt für alle $k = 1,2,\ldots,s-1$:*
   $dist(x_i, n(x_i,k)) < dist(x_j,x_p)$
3. *Falls $n(x_i,k) = x_j$, $n(x_j,l) = x_i$, $k \neq l$ und $n(x_j,k) = x_p$ dann folgt:*
   $$\begin{cases} dist(x_j,x_p) < dist(x_i,x_j) & falls\, l > k \\ dist(x_i,x_j) < dist(x_j,x_p) & sonst \end{cases}$$
4. *Falls $n(x_i,s) = x_j$, $n(x_j,t) = x_i$ und $s < t$, $t \geq 2$. Wenn wir x_i und x_j mit einer gerichteten Kante verbinden wollen, dann erreicht die Kante, die von der Seite x_i abgeht, zuerst x_j. x_i gehört zur Menge $N_t(x_j)$ und für $k = t-1$ gibt es noch Zwischen-Nachbarmengen $N_p(x_j)$, $1 \leq p \leq k$. Die Punkte, die zur diesen Zwischen-Nachbarmengen gehören, haben eine kleinere Distanz mit x_j im Vergleich zur Distanz von x_i mit x_j. Man kann dies auch so interpretieren, dass x_j um k Zwischen-Nachbarmengen dichter ist als x_i.*

Beweis:

1. *Diese Eigenschaft ist völlig nachvollziehbar und braucht keinen Beweis.*
2. *Das folgt aus der Definition der Nachbarschaftsfunktion. Wir haben:*
   $dist(x_i, n(x_i, 1)) < \ldots < dist(x_i, n(x_i, s-1)) < dist(x_i, x_p) \leq dist(x_j, x_p)$
3. *Das folgt aus der Definintion der Nachbarschaftsfunktion und zweite Eigenschaft der dist(.)-Funktion* $dist(x_j, x_i) = dist(x_i, x_j)$.
   $l > k \Rightarrow dist(x_j, n(x_j, k) = x_p) < dist(x_j, n(x_j, l) = x_i) = dist(x_i, x_j)$
   $k > l \Rightarrow dist(x_i, x_j) = dist(x_j, n(x_j, l) = x_i) < dist(x_j, n(x_j, k) = x_p)$
4. *Das folgt aus den Definitionen der Nachbarschaftsfunktion und der Menge der k-ten Nachbarn* $N_k(x)$, $k \geq 1$.

3. Die erste Nachbarschaft

Für das Clustering und die Klassifikation in unserer Arbeit benutzen wir im Haupteil nur die erste Nachbarschaftsbeziehung. Deswegen entwerfen wir zunächst Algorithmen, um den ersten Nachbarn zu bestimmen.

In diesem Kapitel beschäftigen wir uns zuerst mit den Begriffsdefinitionen. Dann beweisen wir ein Theorem, auf dem unsere Kernidee für die Entwicklung des ersten Nachbarschaftsalgorithmus basiert. Weiterhin entwickeln wir zunächst einen einfachen ersten Nachbarschaftsalgorithmus (Algorithmus 1), dessen Schritte alle parallel durchführbar sind. Wir erweitern Algorithmus 1 und entwickeln einen sequentiellen ersten Nachbarschaftsalgorithmus (Algorithmus 2), der im Falle der sequentiellen Ausführung des Algorithmus 1 effizienter ist als Algorithmus 1. Danach beweisen wir die Korrektheit der Algorithmen 1 und 2 und bestimmen abschließend die Laufzeit des Algorithmus 1 und untersuchen das Verhalten der Algorithmen 1 und 2 empirisch. Außerdem vergleichen wir den Algorithmus 2 mit einem auf dem R-Baum basierten RKV-Algorithmus ([55]) .

3.1. Begriffsdefinitionen und unsere Kernidee des ersten Nachbarschaftsalgorithmus

In diesem Abschnitt definieren wir zunächst wieder ein paar Begriffe und beweisen die Kernidee unseres ersten Nachbarschaftsalgorithmus als ein Theorem.
Es existiert für $x_i \in X$, $|X| \geq 2$ mindestens ein
$x_j \in X : dist(x_i, x_j) \leq dist(x_i, x_p)$, $x_j \neq x_p$ und damit auf jeden Fall $n(x_i, 1) = x_j$.
$n(x_j, 1) = x_i$ kann nur dann gelten, wenn kein anderer Punkt
$x_p \in X : dist(x_j, x_p) < dist(x_j, x_i)$ existiert.

Korollar 2 - *Existenz der ersten Nachbarschaft*
Für jeden Punkt $x \in X$, $|X| \geq 2$ gibt es mindestens einen Punkt $y : n(x, 1) = y$.

Wir beweisen dieses Korollar nicht, weil es eindeutig und nachvollziehbar ist.

Die Frage ist, wie aufwändig die Berechnung von der ersten Nachbarschaft ist. Gibt es irgendeine Möglichkeit, mit der man die erste Nachbarschaft effizienter berechnen kann?

Wir gehen davon aus, dass wir überhaupt keine Vorkenntnisse über die Punkte in der Menge X haben. Wenn wir den naiven Algorithmus benutzen, dann müssen wir für jeden Query-Punkt x die Abstände zu allen anderen $N - 1$ Punkten berechnen und

jedes Mal den minimalen Abstand von den berechneten Abständen finden. Der Aufwand ist dazu für einen Query-Punkt $O(dN)$. Dieser Aufwand ist inakzeptabel, weswegen wir uns Gedanken machen müssen, wie wir diese Situation verbessern können.

Wir sortieren jede Dimension der Punkte $x_j = (a_{1_j}, a_{2_j}, \ldots, a_{d_j}) \in X$, $j \in \{1, 2, \ldots, N\}$. D.h. für Attribut A_i, $i = 1, 2, \ldots, d$ sei $[A_i] := \left(a_{i_{s_1}}, a_{i_{s_2}}, \ldots, a_{i_{s_N}}\right)$ eine $1 \times N$-Matrix, wobei $a_{i_{s_k}}$, $s_k \in \{1, 2, \ldots N\}$ die sortierten Werte der Attribute A_i sind, d.h. $a_{i_{s_1}} \leq a_{i_{s_2}} \leq \ldots \leq a_{i_{s_N}}$. Hier besagt Index i_{s_k}, dass $a_{i_{s_k}}$ der Wert der i-ten Dimension des Punktes $x_{s_k} \in X$ ist. Wir transformieren jede unsere $1 \times N$-Matritzen $[A_i]$, $i = 1, 2, \ldots, d$ in einer Form, so dass die Attributswerte in $[A_i]$ nicht miteinander gleich sind. D.h. anstelle von $a_{i_{s_1}} \leq a_{i_{s_2}} \leq \ldots \leq a_{i_{s_N}}$ haben wir $a_{i_{s_1}} < a_{i_{s_2}} < \ldots < a_{i_{s_p}}$, $p \leq N$. Falls in $[A_i]$ ein Attributswert a k-Mal vorkommt, dann speichern wir hierfür a nur ein Mal in $[A_i]$ und merken uns, welche Punkte diesen gemeinsamen Attributswert a an dieser Stelle in der $1 \times N$-Matrix $[A_i]$ haben. Wir speichern $k - 1$-Mal leeren Wert $(-)$ am Ende der $1 \times N$-Matrix $[A_i]$.

Definition 3 *-Matrix M*

Sei im Folgenden $M := \begin{pmatrix} [A_1] \\ [A_2] \\ \vdots \\ [A_d] \end{pmatrix}$ *die* $d \times N$*-sortierte Matrix.*

Beispiel 1 -

Sei $X = \{x_1, x_2, x_3, x_4\}$*, wobei* x_1, x_2, x_3 *und* x_4 *gleich sind:*

$x_1 = (3, 6)$, $x_2 = (4, 6)$, $x_3 = (3, 9)$, $x_4 = (5, 8)$

Dann ergibt sich Folgendes:

$[A_1] = (3, 4, 5, -)$

$[A_2] = (6, 8, 9, -)$

$M = \begin{bmatrix} 3 & 4 & 5 & - \\ 6 & 8 & 9 & - \end{bmatrix}$

Definition 4 *-*$[x]$

Für Punkt $x = (a_1, a_2, \ldots, a_d) \in X$ *sei*
$[x] := (j_1, j_2, \ldots, j_d)^T$, $j_1, j_2, \ldots j_d \in \{1, 2, \ldots, N\}$ *die Positionen der Attributswerte* a_i, $i = 1, 2, \ldots, d$ *in Matrix* M*, wobei* j_s, $s = 1, 2, \ldots, d$ *für* (s, j_s) *Zeile* s *und Spalte* $j_s \in \{1, 2, \ldots, N\}$ *in Matrix* M *steht.*

Definition 5 *-*$M_{(i,j)}$

Sei die Menge $M_{(i,j)}$ *definiert als:*

$M_{(i,j)} := \{x \in$
$X \mid x$ *hat seinen Attributswert an* $i - te$ *Zeile und* $j - te$ *Spalte in Matrix* $M\}$

Definition 6 *-A(.)*

Sei $A : \{1, 2, \ldots, d\} \times \{1, 2, \ldots, N\} \to A$, $(i, j) \mapsto a$ *die Funktion, die als Eingabe* (i, j)[1] *hat und als Ausgabe den Attributswert an i-te Zeile und j-te Spalte in Matrix M zurückgibt.*

Definition 7 *-dist(.)*

Für $x_1 = (a_1, a_2, \ldots, a_d)$, $x_2 = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d)$ *sei*
$dist_j(x_1, x_2) := |a_j - \acute{a}_j|$, $j = 1, 2, \ldots, d$ *der lokale Abstand zwischen* x_1 *und* x_2 *in Dimension j. Wir definieren* $dist(x_1, x_2) := \kappa(\sum_{j=1}^{d} dist_j(x_1, x_2))$ *mit* $\kappa : \mathbb{R}^+ \to \mathbb{R}^+$.

Definition 8 -

1. *Für Punkt* $x = (a_1, a_2, \ldots, a_d) \in X$,
   $[x] := (j_1, j_2, \ldots, j_d)^T$, $j_1, j_2, \ldots j_d \in \{1, 2, \ldots, N\}$ *seien*
   $A^s_k(x)$, $k \in \{1, 2, \ldots\}$, $s = 1, 2, \ldots, d$ *die Attributswerte in Matrix M definiert als:*
   $A^s_k(x) := \{A^s_{k^+}(x), A^s_{k^-}(x)\}$, *wobei* $A^s_{k^+}(x)$ *und* $A^s_{k^-}(x)$ *definiert sind als:*

   $$A^s_{k^+}(x) := \begin{cases} +\infty & falls\, j_s + k > N \text{ oder } A(s, j_s + k) = (-) \\ A(s, j_s + k) & sonst \end{cases}$$

   $$A^s_{k^-}(x) := \begin{cases} +\infty & falls\, j_s - k < 1 \\ A(s, j_s - k) & sonst \end{cases}$$

2. *Wir bezeichnen* $k : k \in \{1, 2, \ldots\}$ *als den Index des k-ten Intervallradius, der aber weiterhin in dieser Arbeit kurz als k-ter Intervallradius in Bezug auf Punkt* $x = (a_1, a_2, \ldots, a_d) \in X$ *bezeichnet wird. Wir müssen k so festlegen, dass mindestens einer von beiden* $A^s_{k^+}(x)$ *oder* $A^s_{k^-}(x)$ *in jeder Dimension* $s = 1, 2, \ldots, d$ *ungleich* $+\infty$ *ist.*
3. *Seien* p^+_j, $p^-_j \in \{1, 2, \ldots, N\}$, $j = 1, 2, \ldots, d$ *Spalten, in denen jeweils Attributswerte* $A^j_{k^+}(x)$ *und* $A^j_{k^-}(x)$ *in Dimension j in Bezug auf Punkt x in Matrix M vorkommen.*
4. *Wir definieren* $Min(A^s_k(x))$ *und* $Max(A^s_k(x))$, $s = 1, 2, \ldots, d$ *in Bezug auf Punkt* $x = (a_1, a_2, \ldots, a_d) \in X$ *als:*
   $Min(A^s_k(x)) := \min(A^s_{k^+}(x), A^s_{k^-}(x))$
   $Max(A^s_k(x)) := \max(A^s_{k^+}(x), A^s_{k^-}(x))$

Definition 9 *-Intervallbildung*

1. *Wir sagen Punkt* $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in X$ *hat in Dimension* $j = 1, 2, \ldots, d$ *die Eigenschaft* min_j *in Bezug auf den Punkt* $x = (a_1, a_2, \ldots, a_d) \in X$ *und den k-ten Intervallradius* $k \in \{1, 2, \ldots\}$, *wenn die folgende Bedingung gilt:*
   $dist_j(x, x_M) \leq |Min(A^j_k(x)) - a_j|$

[1] i steht für Zeile und j für Spalte in Matrix M.

2. *Wir bezeichnen* $!min_j$ *als das Kennzeichen für den Punkt* $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in X$, *wenn er nicht in Dimension* $j = 1, 2, \ldots, d$ *die Eigenschaft* min_j *in Bezug auf den Punkt* $x = (a_1, a_2, \ldots, a_d) \in X$ *und den* k*-ten Intervallradius* $k \in \{1, 2, \ldots\}$ *hat. Damit haben wir folgende Bedingung:*
   $dist_j(x, x_M) > |Min(A_k^j(x)) - a_j|$
3. *Seien* $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$ *Dimensionen, in denen Punkt* $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in X$ *die Eigenschaft* $!min_{j_b}$, $b = 1, 2, \ldots, s$ *in Bezug auf den Punkt* $x = (a_1, a_2, \ldots, a_d) \in X$ *und den* k*-ten Intervallradius* $k \in \{1, 2, \ldots\}$ *hat. Falls* $A_{k^+}^{j_b}$ *und* $A_{k^-}^{j_b}$ *beide ungleich* $+\infty$ *sind, kann es auch möglich sein, dass folgende Bedingung gilt:*
   $dist_{j_b}(x, x_M) > |Max(A_k^{j_b}(x)) - a_{j_b}|$
   Wir bilden Intervalle in Dimensionen j_b. *Hierfür setzen wir das rechte Intervall gleich* $[p_{j_b}^+, p_{j_b}^r]$, *wenn die Bedingungen* $A_{k^+}^{j_b} \neq +\infty$ *und*
   $dist_{j_b}(x, x_M) > |A_{k^+}^{j_b}(x) - a_{j_b}|$ *gelten. Die rechte Seite* $p_{j_b}^r$ *des rechten Intervalls hat folgende Eigenschaften:*
   - $p_{j_b}^r > p_{j_b}^+$ *und* $dist_{j_b}(x, x_M) \geq |A(j_b, p_{j_b}^r) - a_{j_b}|$
   - *Falls* $p_{j_b}^r + 1 \leq N$, *dann gilt:*
     $dist_{j_b}(x, x_M) < |A(j_b, p_{j_b}^r + 1) - a_{j_b}|$

   Weiterhin setzen wir das linke Intervall gleich $[p_{j_b}^l, p_{j_b}^-]$, *wenn die Bedingungen* $A_{k^-}^{j_b} \neq +\infty$ *und* $dist_{j_b}(x, x_M) > |A_{k^-}^{j_b}(x) - a_{j_b}|$ *gelten. Die linke Seite des linken Intervalls hat folgende Eigenschaften:*
   - $p_{j_b}^l < p_{j_b}^-$ *und* $dist_{j_b}(x, x_M) \geq |A(j_b, p_{j_b}^l) - a_{j_b}|$
   - *Falls* $p_{j_b}^l - 1 \geq 1$, *dann gilt:*
     $dist_{j_b}(x, x_M) < |A(j_b, p_{j_b}^l - 1) - a_{j_b}|$

   Für $v_{j_b}^l : p_{j_b}^l \leq v_{j_b}^l \leq p_{j_b}^-$ *und* $v_{j_b}^r : p_{j_b}^+ \leq v_{j_b}^r \leq p_{j_b}^r$ *gilt Folgendes:*
   Alle Punkte $x^l = (a_1^l, a_2^l, \ldots, a_d^l) : x^l \in M_{(j_b, v_{j_b}^l)}$ *und*
   $x^r = (a_1^r, a_2^r, \ldots, a_d^r) : x^r \in M_{(j_b, v_{j_b}^r)}$, *deren Attributswerte in diesen Intervallen liegen, haben die Eigenschaft:*
   $dist_{j_b}(x, x^l) \leq dist_{j_b}(x, x_M)$
   $dist_{j_b}(x, x^r) \leq dist_{j_b}(x, x_M)$
   Wir bezeichnen die Punkte x^l *und* x^r *als Intervallpunkte und die Konstruktion dieser Intervalle als Intervallbildung.*

Folgend ein Beispiel:

Beispiel 2 -
Sei $X = \{x_1, x_2, x_3, x_4\}$, *wobei* x_1, x_2, x_3 *und* x_4 *gleich sind:*
$x_1 = (1, 9, 3)$, $x_2 = (-1, 5, 12)$

$x_3 = (20, -5, -5)$, $x_4 = (-1, 15, 25)$
Dann ergibt sich Folgendes:

$$M = \begin{pmatrix} -1 & 1 & 20 & - \\ -5 & 5 & 9 & 15 \\ -5 & 3 & 12 & 25 \end{pmatrix}$$

$[x_1] = (2, 3, 2)^T$, $[x_2] = (1, 2, 3)^T$
$[x_3] = (3, 1, 1)^T$, $[x_4] = (1, 4, 4)^T$
$M_{(1,1)} = \{x_2, x_4\}$, $M_{(1,2)} = \{x_1\}$, $M_{(1,3)} = \{x_3\}$, $M_{(1,4)} = \emptyset$
$M_{(2,1)} = \{x_3\}$, $M_{(2,2)} = \{x_2\}$, $M_{(2,3)} = \{x_1\}$, $M_{(2,4)} = \{x_4\}$
$M_{(3,1)} = \{x_3\}$, $M_{(3,2)} = \{x_1\}$, $M_{(3,3)} = \{x_2\}$, $M_{(3,4)} = \{x_4\}$
$A(1,1) = -1$, $A(1,2) = 1$, $A(1,3) = 20$, $A(1,4) = (-)$
$A^1_{1^+}(x_1) = 20$, $A^1_{1^-}(x_1) = -1$, $A^1_1(x_1) = \{-1, 20\}$
$A^2_{1^+}(x_1) = 15$, $A^2_{1^-}(x_1) = 5$, $A^2_1(x_1) = \{5, 15\}$
$A^3_{1^+}(x_1) = 12$, $A^3_{1^-}(x_1) = -5$, $A^3_1(x_1) = \{-5, 12\}$
$A^1_{1^+}(x_4) = 1$, $A^1_{1^-}(x_4) = +\infty$, $A^1_1(x_4) = \{1, +\infty\}$
$A^2_{1^+}(x_4) = +\infty$, $A^2_{1^-}(x_4) = 9$, $A^2_1(x_4) = \{9, +\infty\}$
$A^3_{1^+}(x_4) = +\infty$, $A^3_{1^-}(x_4) = 12$, $A^3_1(x_4) = \{12, +\infty\}$
$A^1_{2^+}(x_4) = 20$, $A^1_{2^-}(x_4) = +\infty$, $A^1_2(x_4) = \{20, +\infty\}$
$A^2_{2^+}(x_4) = +\infty$, $A^2_{2^-}(x_4) = 5$, $A^2_2(x_4) = \{5, +\infty\}$
$A^3_{2^+}(x_4) = +\infty$, $A^3_{2^-}(x_4) = 3$, $A^3_2(x_4) = \{3, +\infty\}$

Nachdem wir ein paar Begriffe eingeführt haben, beschäftigen wir uns wieder mit der Frage, wie man die Menge des ersten Nachbarn $N_1(x)$ effizient bestimmen kann. Hierfür beweisen wir zunächst ein Theorem, auf welchem die Kernidee unserer ersten Nachbarschaftsalgorithmen basiert.

Theorem 1 -
Sei $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$. Wir berechnen den Abstand zwischen x und den Punkten $x^* : x^* \in M_{(j,i_j \pm l)}$, $1 \leq l \leq k$, $k \in \{1, 2, \ldots\}$ in Matrix M, wobei k der k-te Intervallradius in Bezug auf Punkt x ist. Sei $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in X$ der Punkt, der unter diesen höchstens $2(T(k \times d))$ Punkten den kleinsten Abstand zu x hat. Hierbei sei T definiert als:

$T := \max_{1 \leq l \leq k} |M_{(j,i_j \pm l)}|_{\, j=1,2,\ldots,d}$

Zunächst stellen wir zwei Hypothesen auf, die wir im Nachhinein beweisen.

- *a- Falls x_M in jeder Dimension $j = 1, 2, \ldots, d$ die Eigenschaft min_j in Bezug auf x und den k-ten Intervallradius $k \in \{1, 2, \ldots\}$ hat, dann ist er das globale Minimum. Hier folgt $n(x, 1) = x_M$ und damit $N_1(x) = \{x_M\}$.*

b- *Seien* $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$ *Dimensionen, in denen Punkt* x_M *die Eigenschaft* $!min_{j_b}$, $b = 1, 2, \ldots, s$ *in Bezug auf Punkt* x *und den* k*-ten Intervallradius hat. Wenn wir jeweils rechte und linke Intervalle* $[p^+_{j_b}, p^r_{j_b}]$, $[p^l_{j_b}, p^-_{j_b}]$, $b = 1, 2, \ldots, s$ *wie in Definition 9 bilden und falls für alle Intervallpunkte* $x^* \in X$*, deren Attributswerte in diesen Intervallen liegen, gilt:* $dist(x, x_M) < dist(x, x^*)$*, dann ist* x_M *globales Minimum. Hieraus folgt* $n(x, 1) = x_M$ *und damit* $N_1(x) = \{x_M\}$.

Beweis:

Da die Distanz zu x *gleich der Summe der lokalen Abstände in jeder Dimension ist, minimiert ein Punkt diese Distanz, falls er diese Summe minimiert. Dies bedeutet für unseren Punkt* $x = (a_1, a_2, \ldots, a_d)$ *und* $\dot{x} = (\dot{a}_1, \dot{a}_2, \ldots, \dot{a}_d)$ *je mehr die Attributswerte* $\dot{a}_j$ *ähnlich wie* a_j, $j = 1, 2, \ldots, d$ *sind, desto kleiner ist die Distanz zwischen* x *und* $\dot{x}$.

a- *Weil* x_M *in jeder Dimension die Eigenschaft* min_j *hat, haben wir in jeder Dimension* $j = 1, 2, \ldots, d$ *die folgende Bedingung:*

$dist_j(x, x_M) \leq |Min(A^j_k(x)) - a_j|$

x_M *hat unter den höchstens* $2(T(k \times d))$ *Punkten den kleinsten Abstand zu* x*. Die Attributswerte aller anderen* $N - 2(T(k \times d))$ *Punkte* $x^* = (a^*_1, a^*_2, \ldots, a^*_d) \in X$ *können nur in jeder Dimension* $j = 1, 2, \ldots, d$ *in Matrix* M *in Spalten*

$$\begin{cases} \alpha & falls\, A^j_{k^+}(x) \neq +\infty\, und\, A^j_{k^-}(x) \neq +\infty \\ \beta & falls\, A^j_{k^+}(x) \neq +\infty\, und\, A^j_{k^-}(x) = +\infty \\ \gamma & falls\, A^j_{k^+}(x) = +\infty\, und\, A^j_{k^-}(x) \neq +\infty \end{cases}$$

vorkommen.

Wobei α, β *und* γ *definiert sind als:*

$\alpha := 1, 2, \ldots, i_j - k - 2, i_j - k - 1, i_j + k + 1, i_j + k + 2, \ldots, N - 1, N$

$\beta := i_j + k + 1, i_j + k + 2, \ldots, N - 1, N$

$\gamma := 1, 2, \ldots, i_j - k - 2, i_j - k - 1$

Für alle drei Fälle α, β *und* γ *gelten folgende Ungleichungen:*

$dist_j(x, x^* \in M_{(j,t)})_{1 \leq t \leq i_j - k - 1} > |Min(A^j_k(x)) - a_j|$

$dist_j(x, x^* \in M_{(j,t)})_{N \geq t \geq i_j + k + 1} > |Min(A^j_k(x)) - a_j|$

Deswegen haben x^* *einen lokalen Abstand* $dist_j(x, x^*)$ *zu* x*, der größer ist als* $|Min(A^j_k(x)) - a_j|$*. Da die Distanz zu* x *die Summe der lokalen Abstände zu* x *ist, folgt:*

$dist(x, x_M) = \kappa\left(\sum_{j=1}^d dist_j(x, x_M)\right) \leq \kappa\left(\sum_{j=1}^d |Min(A^j_k(x)) - a_j|\right)$

$\kappa\left(\sum_{j=1}^d |Min(A^j_k(x)) - a_j|\right) < \kappa\left(\sum_{j=1}^d dist_j(x, x^*)\right) = dist(x, x^*)$

Hieraus folgt $n(x, 1) = x_M$ *und damit* $N_1(x) = \{x_M\}$.

b- *Wir bilden in den Dimensionen* j_b*, bei denen Attributswerte von* x_M *die Eigenschaft* $!min_{J_b}$ *haben, wie in Definition 9 beschrieben, Intervalle. Innerhalb*

dieser Intervalle haben Intervallpunkte $x^* = (a_1^*, a_2^*, \ldots, a_d^*) \in X$ einen lokalen Abstand kleiner/gleich mit x im Vergleich zu x_M mit x, d.h. $dist_{j_b}(x, x^*) \leq dist_{j_b}(x, x_M)$. Ein Punkt x_t kann eine kleinere Distanz mit x im Vergleich zu x_M haben, falls er eine kleinere Summe der Abstände hat. In Dimensionen $* \neq j_b$, in denen Attributswerte von x_M die Eigenschaft min_* haben, hat x_M den kleineren lokalen Abstand mit x. Deswegen kann x_t nur dann eine kleinere Summe der Abstände bilden, falls er mindestens einen von seinen Attributswerten innerhalb der Intervalle in Dimensionen j_b, wo die Attributswerte von x_M die Eigenschaft $!min_{j_b}$ haben, hat. Aus diesem Grund ergibt sich, für den Fall, dass alle Intervallpunkte $x^* \in X$ eine größere Distanz mit x im Vergleich zu x_M mit x haben, dass x_M definitiv ein globales Minimum ist.

3.2. Algorithmen für die erste Nachbarschaftsbeziehung

Nachdem wir Theorem 1 bewiesen haben, entwickeln wir in diesem Abschnitt zunächst einen leicht verständlichen ersten Nachbarschaftsalgorithmus. Daraufhin erweitern wir diesen Algorithmus.
Sei $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ ein beliebiger Punkt, für den wir die Menge des ersten Nachbarn $N_1(x)$ bestimmen wollen. Wir setzen den k-ten Intervallradius gleich 1 und schreiben folgenden Algorithmus:

Algorithm 1 *-Bestimmung der Menge des ersten Nachbarn $N_1(x)$*

1. *Bestimme sortierte Matrix M.*
2. *Berechne parallel den Abstand zwischen $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ und Punkten $x^* : x^* \in M_{(j,i_j \pm 1)}$, $j = 1, 2, \ldots, d$ in Matrix M. Sei W die Menge des Punktes/der Punkten, der/die unter diesen höchstens $2(T \times d)$ Punkten den kleinsten Abstand zu x hat/haben.*

   $W := \{x_M \,|\, x_M \in M_{(j,i_j \pm 1)} \, und \, dist(x, x_M) \, ist \, minimal\}$.

   Hier sei T definiert als:

   $T := \max_{j=1,2,\ldots,d} |M_{(j,i_j \pm 1)}|$
3. *Die Menge W ist nicht leer und hat mindestens ein Element. Abhängig davon, wieviele Elemente die Menge W hat (eins oder mehr als eins), gehen wir wie folgt vor:*

   1- *Sei $|W| = 1$ und $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in W$, der Punkt mit dem kleinsten Abstand zu x. Hier haben wir zwei Möglichkeiten.*

      a- *x_M kommt d-mal vor und hat in jeder Dimension j die Eigenschaft min_j. D.h. Attributswerte von x_M kommen in jeder Dimension in der Spalte $i_j + 1$ oder $i_j - 1$ vor und x_M hat in jeder Dimension den kleinsten lokalen Abstand zu x. Deswegen ist x_M globales Minimum. Hier folgt $n(x, 1) = x_M$ und damit $N_1(x) = \{x_M\}$.*

b- *Seien $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$, $1 \leq s \leq d$ Dimensionen, in denen x_M die Eigenschaft $!min_{j_b}, b = 1, 2, \ldots, s$ hat. x_M ist für uns aber ein Kandidat, der ein lokales Minimum sein kann. Um zu prüfen, ob x_M ein globales Minimum ist, müssen wir Intervalle bilden, womit wir den Abstand von x_M zu x mit den anderen Abständen der Punkte, die in diesen Intervallen vorkommen, mit x vergleichen. Wenn x_M immer noch den kleinsten Abstand zu x hat, dann ist x_M ein globales Minimum. In Matrix M bilden wir wie in Definition 9 beschrieben linke und rechte Intervalle $[p^l_{j_b}, p^-_{j_b}]$, $[p^+_{j_b}, p^r_{j_b}]$. Für $v^l_{j_b} : p^l_{j_b} \leq v^l_{j_b} \leq p^-_{j_b}$ und $v^r_{j_b} : p^+_{j_b} \leq v^r_{j_b} \leq p^r_{j_b}$ gilt Folgendes:*

Alle Intervallpunkte $x^l = (a^l_1, a^l_2, \ldots, a^l_d) : x^l \in M_{(j_b, v^l_{j_b})}$ und

$x^r = (a^r_1, a^r_2, \ldots, a^r_d) : x^r \in M_{(j_b, v^r_{j_b})}$, deren Attributswerte in diesen Intervallen liegen, haben die Eigenschaft:

$dist_{j_b}(x, x^l) \leq dist_{j_b}(x, x_M)$

$dist_{j_b}(x, x^r) \leq dist_{j_b}(x, x_M)$

Sei in folgenden o.B.d.A. x^* ein Variable, die wir anstelle von x^r und x^l benutzen. Wir berechnen parallel in jedem Intervall den Abstand zwischen x und x^*. Falls $dist(x, x^*) < dist(x, x_M)$, markieren wir x^* als K (kleiner) und falls $dist(x, x^*) = dist(x, x_M)$ als G (gleich). Mit der Markierung haben wir den Vorteil, dass wir den Abstand zu x^* nicht erneut berechnen müssen, falls Attributswerte des markierten x^* in mehr als einem Intervall vorkommen.

Wir definieren $K(x)$ und $G(x)$ als:

$$\begin{aligned} K(x) &:= \{x^* \mid x^* \, wurde \, markiert \, als \, K\} \\ G(x) &:= \{x^* \mid x^* \, wurde \, markiert \, als \, G\} \end{aligned}$$

Dann ergeben sich folgende Situationen:

- Falls $K(x) = \emptyset$ und $G(x) = \emptyset$, dann ist es $N_1(x) = \{x_M\}$.

- Falls $K(x) = \emptyset$ und $G(x) \neq \emptyset$, dann ist es $N_1(x) = \{x_M\} \bigcup G(x)$.

- Für $K(x) \neq \emptyset$ sortieren wir die Abstände von diesen Punkten zu x und Punkte mit dem kleinsten gleichen Abstand zu x bilden die erste Nachbarschaftsmenge $N_1(x)$.

2- *Sei $|W| = p$, $p \geq 2$ und seien $x_{M_1}, x_{M_2} \ldots, x_{M_p} \in W$ Punkte mit dem gleichen kleinsten Abstand zu x. Hier haben wir zwei Möglichkeiten:*

a- *Sei $p = 2$, x_{M_1} und x_{M_2} kommen d-mal vor und haben in jeder Dimension j die Eigenschaft min_j. Deswegen sind sie beide globales Minimum. Hier ist es $N_1(x) = \{x_{M_1}, x_{M_2}\}$.*

b- *Sonst wähle beliebig einer von x_{M_b}, $b = 1, 2, \ldots, p$ und gehe genau wie Schritt 1 Teil b vor.*

Wir erweitern Algorithmus 1 in zwei Hinsichten, erstens betrachten wir den k-ten Intervallradius als einen Parameter in unserem Algorithmus. Zweitens, um den Suchraum so weit wie möglich einzuschränken, verkleinern wir solange unsere Intervalle, wie wir einen neuen ersten Nachbarskandidaten finden, der einen kleineren Abstand zu unserem Punkt hat, bis wir den globalen ersten Minimumsnachbarn finden.

Sei $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ ein beliebiger Punkt, für den wir die Menge des ersten Nachbarn $N_1(x)$ bestimmen wollen.

In unserem sequenziellen erweiterten ersten Nachbarschaftsalgorithmus verwenden wir folgende globale Parameter und Variablen:

Globale Parameter

- k-ter Intervallradius $k \in \{1, 2, \ldots\}$.

Globale Variablen

- $W :=$ Wir speichern Punkte in Menge W.
- $gibtMinimum :=$ boolesche Variable.
- $x.behandelt :=$ Eine boolesche Variable für jeden Punkt $x \in X$, die am Anfang als Falsch initialisiert wird.

Algorithm 2 *-Erweiterte erste Nachbarschaftsalgorithmus*

Initialisierungsphase

1- Bestimme sortierte Matrix M.

2- Bestimme den k-ten Intervallradius.

3- Berechne den Abstand zwischen $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ und Punkten $x^* : x^* \in M_{(j,i_j \pm l)}$, $1 \leq l \leq k$, $j = 1, 2, \ldots, d$ in Matrix M und setze boolesche Variable aller x^* gleich Wahr $x^*.behandelt = Wahr$.

4- Bestimme den Punkt oder die Punkte, der/die unter diesen höchstens $2(T(k \times d))$ Punkten den kleinsten Abstand zu x hat/haben und speichere ihn/sie in Menge W.

$W := \{x_M \,|\, x_M \in M_{(j,i_j \pm t)},\, t \in \{1, 2, \ldots, k\} \,und\, dist(x, x_M)\, ist\, minimal\}$

Hier sei T definiert als:

$T := \max_{1 \leq l \leq k} |M_{(j,i_j \pm l)}|_{\, j=1,2,\ldots,d}$

5- Bestimme $A_k^j(x) = \{A_{k^+}^j(x), A_{k^-}^j(x)\}$, $j = 1, 2, \ldots, d$.

6- Setze boolesche Variable $gibtMinimum$ gleich Wahr $gibtMinimum = Wahr$.

Hauptphase

BEGIN

1- $While(gibtMinimum)$ {

2- Setze boolesche Variable $gibtMinimum$ gleich Falsch $gibtMinimum = Falsch$.

3- Wähle einen beliebigen Punkt $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in W$.

4- Prüfe, ob x_M in jeder Dimension $j = 1, 2, \ldots, d$ die Eigenschaft min_j hat. Wenn " Ja ", dann haben wir globales Minimum gefunden. Deswegen breche die While-Schleife in Zeile 1 ab.

5- For alle Dimensionen $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$*, in denen* x_M *die Eigenschaft* $!min_{j_b}$, $b = 1, 2, \ldots, s$ *hat, tue folgendes {*

6- Setze boolesche Variable b_1 *gleich Wahr, falls die folgende Bedingung gilt:*

- $B_1 := (A^{j_b}_{k^+}(x) \neq +\infty) \,\&\, (p^+_{j_b} + 1 \leq N) \,\&\, (|A(j_b, p^+_{j_b} + 1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$

6- Setze boolesche Variable b_2 *gleich Wahr, falls die folgende Bedingung gilt:*

- $B_2 := (A^{j_b}_{k^-}(x) \neq +\infty) \,\&\, (p^-_{j_b} - 1 \geq 1) \,\&\, (|A(j_b, p^-_{j_b} - 1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$

7- Falls b_1 *gleich Wahr ist, dann inkrementiere* $p^+_{j_b}$, $p^+_{j_b} = p^+_{j_b} + 1$ *und aktualisiere den Wert von* $A^{j_b}_{k^+}(x)$, $A^{j_b}_{k^+}(x) = A(j_b, p^+_{j_b})$, $k^+ = k^+ + 1$.

8- Falls b_2 *gleich Wahr ist, dann dekrementiere* $p^-_{j_b}$, $p^-_{j_b} = p^-_{j_b} - 1$ *und aktualisiere den Wert von* $A^{j_b}_{k^-}(x)$, $A^{j_b}_{k^-}(x) = A(j_b, p^-_{j_b})$, $k^- = k^- + 1$.

9- $!min_{k^+}(j_b, b_1)$.

10- $!min_{k^-}(j_b, b_2)$.

11- Falls boolesche Variable gibtMinimum gleich Wahr ist, dann bereche $For - Schleife$ *in Zeile 5 ab.*

}// end $For - Schleife$ *Zeile 5*

12- $if\,(gibtMinimum)${

13- For alle Dimensionen $j = 1, 2, \ldots, d$ *tue folgendes {*

14- $min_{k^+}(j)$.

15- $min_{k^-}(j)$.

}//end For-Schleife Zeile 13

}//end if *Zeile 12*

}// end $While - Schleife$ *Zeile 1*

Endphase

16- Setze $N_1(x)$ *gleich* W, $N_1(x) = W$.

END

Hier schreiben wir die benutzten Methoden in unseren erweiterten ersten Nachbarschaftsalgorithmus:

$!min_{k^+}(int\, j, boolean\, a)$ {

1- $While(a)${

2- Setze boolesche Variable a gleich Falsch $a = Falsch$.

3- For alle $x^* : x^* \in M_{(j,p^+_j)}$ tue folgendes {

4- $if\,(x^*.behandelt == Falsch)${

5- Setze $x^*.behandelt$ gleich Wahr $x^*.behandelt = Wahr$.

6- $if(dist(x, x^*) \leq dist(x, x_M))${

7- $if(dist(x, x^*) == dist(x, x_M))$ dann $W = W \cup \{x^*\}$.

8- $if\,(dist(x, x^*) < dist(x, x_M))${

9- Setze W gleich leere Menge $W = \emptyset$.

10- Füge neuen ersten Nachbarschaftskandidaten in Menge W ein $W = W \cup \{x*\}$.

11- Setze boolesche Variable $gibtMinimum$ gleich Wahr $gibMinimum = Wahr$.

}// end if Zeile 8

}// end if Zeile 6

}// end if Zeile 4

}// end if Zeile 3

12- Falls $gibtMinimum$ gleich Wahr ist, dann bereche $While - Schleife$ in Zeile 1 ab.

13- $if\,((p_j^+ + 1 \leq N)\,\&\,(|A(j, p_j^+ + 1) - a_j| \leq dist_j(x, x_M))\{$

14- Inkrementiere p_j^+, $p_j^+ = p_j^+ + 1$.

15- Aktualisiere $A^j_{k^+}(x)$, $A^j_{k^+}(x) = A(j, p_j^+)$, $k^+ = k^+ + 1$.

16- Setze boolesche Variable a gleich $Wahr$ $a = Wahr$.

}// end if Zeile 13

}// end $while - Schleife$ Zeile 1

}//end Methode

$!min_{k^-}(int\,j, boolean\,a)$ {

1- $While(a)\{$

2- Setze boolesche Variable a gleich Falsch $a = Falsch$.

3- For alle $x^* : x^* \in M_{(j,p_j^-)}$ tue folgendes {

4- $if\,(x^*.behandelt == Falsch)\{$

5- Setze $x^*.behandelt$ gleich Wahr $x^*.behandelt = Wahr$.

6- $if\,(dist(x, x^*) \leq dist(x, x_M))\{$

7- $if\,(dist(x, x^*) == dist(x, x_M))$ dann $W = W \cup \{x^*\}$.

8- $if\,(dist(x, x^*) < dist(x, x_M))\{$

9- Setze W gleich leere Menge $W = \emptyset$.

10- Füge neuen erste Nachbarschaftskandidat in Menge W ein $W = W \cup \{x*\}$.

11- Setze boolesche Variable $gibtMinimum$ gleich Wahr $gibMinimum = Wahr$.

}// end if Zeile 8

}// end if Zeile 6

}// end if Zeile 4

}// end if Zeile 3

12- Falls $gibtMinimum$ gleich Wahr ist, dann bereche $While - Schleife$ in Zeile 1 ab.

13- $if\,((p_j^- - 1 \geq 1)\,\&\,(|A(j, p_j^- - 1) - a_j| \leq dist_j(x, x_M))\{$

14- Dekrementiere p_j^-, $p_j^- = p_j^- - 1$.

15- Aktualisiere $A^j_{k^-}(x)$, $A^j_{k^-}(x) = A(j, p_j^-)$, $k^- = k^- + 1$.

16- Setze boolesche Variable a gleich $Wahr$ $a = Wahr$.

}// end if Zeile 13

}// end $while - Schleife$ Zeile 1

}// end Methode

$min_{k^+}(int\, j)\{$

1- $if\ (A^j_{k^+}(x) \neq +\infty)\{$

2- $While(p_j^+ + 1 \leq N\ \&\ Behandelt(j, p_j^+ + 1))\{$

3- Inkrementiere p_j^+, $p_j^+ = p_j^+ + 1$.

4- Aktualisiere $A^j_{k^+}(x)$, $A^j_{k^+}(x) = A(j, p_j^+)$, $k^+ = k^+ + 1$.

}// end $While - Schleife$ Zeile 2

}// end if Zeile 1

}// end Methode

$min_{k^-}(int\, j)\{$

1- $if\ (A^j_{k^-}(x) \neq +\infty)\{$

2- $While(p_j^- - 1 \geq 1\ \&\ Behandelt(j, p_j^- - 1))\{$

3- Dekrementiere p_j^-, $p_j^- = p_j^- - 1$.

4- Aktualisiere $A^j_{k^-}(x)$, $A^j_{k^-}(x) = A(j, p_j^-)$, $k^- = k^- + 1$.

}// end $While - Schleife$ Zeile 2

}// end if Zeile 1

}//end Methode

$Behandelt(int\, j_1, int\, j_2)\{$

1- Falls für alle $x^* \in M_{(j_1,j_2)}$ gilt:

$x^*.behandelt$ gleich Wahr $x^*.behandelt$==Wahr, dann gib Wahr zurück.

2- Sonst gib Falsch zurück.

}// end Methode

Hier beschreiben wir unseren sequentiellen erweiterten ersten Nachbarschaftsalgorithmus detaillierter. Wir haben unseren Algorithmus in drei Phasen eingeteilt.

Initialisierungsphase

In Schritt 4 bestimmen wir die Menge W, die die Punkte/den Punkt, die/der unter höchstens $2(T(k \times d))$ Punkten den kleinsten Abstand zu x haben/hat und Kandidat der ersten Nachbarschaft für x sind/ist, hat.

Weiterhin bestimmen wir in Schritt 5 in Bezug auf den Punkt x und k-ten Intervallradius die Menge $A^j_k(x) = \{A^j_{k^+}(x), A^j_{k^-}(x)\}$, $j = 1, 2, \ldots, d$. Damit legen wir mit p_j^+ und p_j^- die linke und rechte initiale Seite von dem rechten Intervall $[p_j^+, p_j^r]$ und dem linken Intervall $[p_j^l, p_j^-]$ in Definition 9 fest. p_j^r und p_j^l werden dynamisch in der Hauptphase in der For-Schleife Zeile 5 bestimmt.

Hauptphase

Die Hauptphase besteht aus einer $While-Schleife$ mit globaler Bedingung $While(gibtMinimum)$. $gibtMinimum$ ist eine boolesche Variable, die in Schritt 6 der Initialisierungsphase gleich Wahr gesetzt wird. Diese globale Bedingung ist beim ersten Eintritt der $While-Schleife$ richtig, weil die Menge W die Kandidaten für die gesuchte erste Nachbarschaft für x hat. In Schritt 2 setzen wir die boolesche Variable $gibtMinimum$ gleich Falsch. In Schritt 3 wählen wir einen beliebigen Punkt $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in W$. In Schritt 4 wollen wir wissen, ob x_M ein lokales oder ein globales Minimum ist. Wenn x_M in jeder Dimension $j = 1, 2, \ldots, d$ die Eigenschaft min_j hat, dann ist er nach dem Teil a des Theorems 1 globales Minimum. Ansonsten machen wir mit der $For-Schleife$ in Schritt 5 weiter. Innerhalb der $For-Schleife$ in Schritt 6 und 7 prüfen wir, ob die zwei Bedingungen B_1 und B_2 Wahr sind. Die Bedingung B_1 und B_2 bestehen jeweils aus drei booleschen Ausdrücken, die sukzessiv von links nach rechts ausgewertet werden und mittels (&)-Verknüpfung aneinander gefügt sind. (&)-Verknüpfung hat die Eigenschaft, dass wenn der boolesche Ausdruck vor der (&)-Verknüpfung Falsch ist, der boolesche Ausdruck nach der (&)-Verknüpfung nicht ausgewertet wird. Hier müssen alle mit (&) verknüpften booleschen Ausdrücke Wahr sein, damit die Bedingung als Wahr ausgewertet wird. Hier beschreiben wir die booleschen Ausdrücke der Bedingungen B_1 und B_2:

$B_1 := (A^{j_b}_{k^+}(x) \neq +\infty)\,\&\,(p^+_{j_b} + 1 \leq N)\,\&\,(|A(j_b, p^+_{j_b} + 1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$

$B_2 := (A^{j_b}_{k^-}(x) \neq +\infty)\,\&\,(p^-_{j_b} - 1 \geq 1)\,\&\,(|A(j_b, p^-_{j_b} - 1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$

- Wenn die Ausdrücke $(A^{j_b}_{k^+}(x) \neq +\infty)$ und $(A^{j_b}_{k^-}(x) \neq +\infty)$ Wahr sind, dann existieren jeweils Spalten $p^+_{j_b}$ und $p^-_{j_b}$ in Matrix M. Falls $p^+_{j_b}$ und $p^-_{j_b}$ existieren, dann sind alle Punkte $x^r \in M_{(j_b, p^+_{j_b})}$ und $x^l \in M_{(j_b, p^-_{j_b})}$ entweder in der Initialisierungsphase oder in der vorherigen $While-Schleife$ bereits verarbeitet worden.
- Wenn die Ausdrücke $(p^+_{j_b} + 1 \leq N)$ und $(p^-_{j_b} - 1 \geq 1)$ Wahr sind, dann existieren jeweils Spalten $p^+_{j_b} + 1$ und $p^-_{j_b} - 1$ in Matrix M.
- Wenn die beiden Ausdrücke $(|A(j_b, p^+_{j_b} + 1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$ und $(|A(j_b, p^-_{j_b} - 1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$ Wahr sind, dann haben die Punkte $x^r = (a^r_1, a^r_2, \ldots, a^r_d) \in M_{(j_b, p^+_{j_b}+1)}$ und $x^l = (a^l_1, a^l_2, \ldots, a^l_d) \in M_{(j_b, p^-_{j_b}-1)}$ jeweils die Eigenschaft, dass sie in Dimension j_b einen kleineren/gleichen lokalen Abstand zu x im Vergleich x_M zu x haben und damit wahrscheinlich eine kleinere globale Distanz zu x haben. $dist_{j_b}(x, x_M)$ ist eine dynamische obere Schranke für unsere linke und rechte Seite $p^l_{j_b}$, $p^r_{j_b}$ vom linken Intervall $[p^l_{j_b}, p^-_{j_b}]$ und rechten Intervall $[p^+_{j_b}, p^r_{j_b}]$. Sie ist dynamisch, weil $\acute{a}_{j_b}$ der Attributswert der j_b-ten Dimension des Punktes x_M sich verändert, falls wir einen neuen Kandidaten für die erste Nachbarschaft finden.

In Schritt 7 und 8 aktualisieren wir $A^{j_b}_{k^+}(x)$, $p^+_{j_b}$, $A^{j_b}_{k^-}(x)$ und $p^-_{j_b}$, falls boolesche Variablen b_1 und b_2 jeweils Wahr sind. In Schritt 9 und 10 rufen wir die Methoden $!min_{k^+}(j_b, b_1)$ und $!min_{k^-}(j_b, b_2)$ auf. Hier bilden wir Intervalle in Dimensionen j_b, $b = 1, 2, \ldots, s$, in denen x_M die Eigenschaft $!min_{j_b}$ hat, und prüfen, ob es in diesen

Intervallen einen anderen Punkt gibt, der einen kleineren Abstand zu x im Vergleich x_M zu x hat. D.h. in Schritt 9 und 10 werden die Intervallpunkte, die wahrscheinlich eine kleinere Distzanz zu x im Vergleich x_M zu x haben, untersucht. Dazu wird ihre Distanz zu x berechnet und wenn diese Distanz kleiner im Vergleich x zu x_M ist, wird die Menge W mit neuen ersten Nachbarschaftskandidaten für den Punkt x initialisiert. Es wird in Zeile 11 der Hauptsphase die $For - Schleife$ in Zeile 5 abgebrochen, falls wir einen neuen Nachbarschaftskandidaten in den Methoden $!min_{k^+}(j_b, b_1)$ oder $!min_{k^-}(j_b, b_2)$ gefunden haben.

In Zeile 12 bis 15, für den Fall, dass die Variable $gibtMinimum$ gleich Wahr ist, aktualisieren wir $A^j_{k^+}(x)$, p^+_j, $A^j_{k^-}(x)$ und p^-_j in Dimensionen $j = 1, 2, \ldots, d$. Wir aktualisieren $A^j_{k^+}(x)$, p^+_j, $A^j_{k^-}(x)$ und p^-_j für jeden neuen ersten Nachbarschaftskandidaten mit den Methoden $min_{k^+}(.)$ und $min_{k^-}(.)$ in jeder $While - Schleife$.

Endphase

In Schritt 16 setzen wir die Menge der ersten Nachbarschaft $N_1(x)$ gleich der Menge W, $N_1(x) = W$.

3.3. Beweis der Korrektheit der ersten Nachbarschaftsalgorithmen und ihre Laufzeitanalyse

Hier beweisen wir die Korrektheit der Algorithmen 1 und 2.

Theorem 2 -

Algorithmus 1 findet für einen beliebigen Punkt $x = (a_1, a_2, \ldots, a_d) \in X$ die Menge des ersten Nachbarn $N_1(x)$.

Beweis:

1. *Schritt 1 Teil a folgt aus dem Teil a des Theorems 1.*
2. *Schritt 1 Teil b folgt aus dem Teil b des Theorem 1.*
3. *Schritt 2 Teil a folgt aus dem Teil a des Theorems 1.*
4. *Schritt 2 Teil b folgt aus dem Teil b des Theorems 1 und wir wählen einen beliebigen Punkt y aus den Punkten $x_{M_1}, x_{M_2}, \ldots, x_{M_p}$ und gehen wie in Schritt 1 Teil b vor, weil:*
   - *$x_{M_1}, x_{M_2}, \ldots, x_{M_{p-1}}$ und x_{M_p} haben den gleichen kleinsten Abstand zu x.*
   - *Falls es bei der Intervallbildung in Schritt 2 Teil b keinen Intervallpunkt gibt, der einen kleineren Abstand zu x im Vergleich y zu x hat, dann sind alle $x_{M_1}, x_{M_2}, \ldots, x_{M_{p-1}}$ und x_{M_p} globales Minimum. Aus diesem Grund kann y beliebig gewählt werden.*
   - *Falls es bei der Intervallbildung in Schritt 2 Teil b mindestens einen Intervallpunkt x^* gibt, der einen kleineren Abstand zu x im Vergleich y zu x hat, dann befindet sich mindestens einer von den Attributswerten von x^**

auch bei der Intervallbildung von allen anderen Punkten $x_{M_b} : x_{M_b} \neq y$, $b \in \{1, 2, \ldots, p\}$ in dessen Intervallen, da es nicht möglich ist, dass x^* auf einer Seite den Abstand zu x im Vergleich y mit x minimiert, aber auf der anderen Seite nicht den Abstand zu x im Vergleich zu den anderen x_{M_b} Punkten mit x minimiert. Aus diesem Grund kann y beliebig gewählt werden.

Wir beweisen die Korrektheit des Algorithmus 2.

Theorem 3 -

Sei k der k-te Intervallradius definiert wie in Definition 8. Dann findet Algorithmus 2 für einen beliebigen Punkt $x = (a_1, a_2, \ldots, a_d) \in X$ die Menge des ersten Nachbarn $N_1(x)$.

Beweis:

Algorithmus 2 funktioniert im Prinzip wie Algorithmus 1 mit folgenden Unterschieden:

- *Bei dem Algorithmus 2 muss der k-te Intervallradius nicht gleich 1 sein.*
- *Bei dem Algorithmus 1 legen wir nur einmal in Schritt 1 Teil b oder Schritt 2 Teil b die linken und rechten Intervalle fest und suchen mühselig in diesen Intervallen nach einem Punkt x^*, der einen kleineren Abstand zu x im Vergleich $x_M \in W$ mit x hat. Beim Algorithmus 2 hingegen setzen wir die linke- und die rechte Intervalle jedes Mal erneut fest, wenn wir in der Hauptphase innerhalb der Methoden $!min_{k^+}(.)$ oder $!min_{k^-}(.)$ einen neuen ersten Nachbarschaftskandidaten x^* finden, der einen kleineren Abstand zu x im Vergleich $x_M \in W$ mit x hat. Da jeder neue erste Nachbarschaftskandidat x^* einen kleineren Abstand zu x hat und wir entsprechend zu x^* die Intervalle in der Hauptphase mit den Methoden $min_{k^+}(.)$ und $min_{k^-}(.)$ dynamisch festlegen, verkleinern wir den Suchraum, so dass wir im Vergleich zu dem Algorithmus 1 nicht mühsam den Suchraum durchsuchen müssen.*

Um die Korrektheit des Algorithmus 2 zu beweisen, zeigen wir, dass die folgende Aussage richtig ist:

Die gesuchte erste Nachbarschaftsmenge $N_1(x)$ für den Punkt x ist in Endphase gleich der Menge W.

In der Initialisierungsphase bestimmen wir die Mengen

$$W = \{x_M \,|\, x_M \in M_{(j,i_j \pm t)},\ t \in \{1, 2, \ldots, k\}\, und\, dist(x, x_M)\, ist\, minimal\}$$

und $A_k^j(x) = \{A_{k^+}^j(x), A_{k^-}^j(x)\}$, $j = 1, 2, \ldots, d$ und setzen in Zeile 6 die boolesche Variable $gibtMinimum$ gleich Wahr.

Die globale Bedingung $While(gibtMinimum)$ ist beim ersten Eintritt der While-Schleife in Zeile 1 in der Hauptphase gleich Wahr. Das ist richtig, da die Menge W den potentiellen ersten Nachbarschaftskandidaten hat. In Zeile 2 setzen wir die

boolesche Variable gibtMinimum gleich Falsch. In der Hauptphase wählen wir in Zeile 3 beliebig[2] *einen Punkt* $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in W$ *und testen, ob* x_M *in jeder Dimension* $j = 1, 2, \ldots, d$ *die Eigenschaft* min_j *hat. Wenn " ja ", dann ist* x_M *nach dem Teil a des Theorems 1 globales Minimum. Damit ist die Menge* W *gleich der ersten Nachbarschaftsmenge* $N_1(x)$ *und wir verlassen die While-Schleife in der Hauptphase. Ansonsten machen wir in Zeile 5 mit der For-Schleife weiter. Hier bilden wir durch die Methoden* $!min_{k^+}(.)$ *und* $!min_{k^-}(.)$ *dynamisch Intervalle in Dimensionen* j_b, $b = 1, 2, \ldots s$, *in denen* x_M *die Eigenschaft* $!min_{j_b}$ *hat und suchen in diesen Intervallen nach dem Intervallpunkt* x^*, *der einen kleineren Abstand zu* x *im Vergleich* x_M *mit* x *hat. Hier haben wir zwei Möglichkeiten:*

1. *Wir finden keinen Intervallpunkt* x^*, *der einen kleineren Abstand zu* x *im Vergleich* x_M *mit* x *hat. Damit ist nach dem Teil b des Theorems 1 die Menge* W *gleich der Menge* $N_1(x)$. *Hier verlassen wir die* $While - Schleife$, *weil die globale Bedingung* $While(gibtMinimum)$ *nicht mehr gilt.*
2. *Wir finden einen Intervallpunkt* x^*, *der einen kleineren*[3] *Abstand zu* x *im Vergleich* x_M *mit* x *hat. Hier setzen wir die Menge* W *innerhalb der Methoden* $!min_{k^+}(.)$ *oder* $!min_{k^-}(.)$ *gleich leere Menge* $W = \emptyset$ *und fügen den neuen ersten Nachbarschaftskandidaten* x^* *in Menge* W, $W = W \cup \{x^*\}$ *ein. Wir setzen auch hier die boolesche Variable gibtMinimum gleich Wahr und aktualisieren* $A^j_{k^+}(x)$, p^+_j, $A^j_{k^-}(x)$ *und* p^-_j *für den neuen ersten Nachbarschaftskandidaten mit den Methoden* $min_{k^+}(.)$ *und* $min_{k^-}(.)$. *Dann fangen wir wieder mit der While-Schleife in Zeile 1 an.*

Wie wir oben beschrieben haben. Innerhalb jeder While-Schleife in Zeile 1 in Hauptphase haben wir folgende Möglichkeiten:

- *Wir finden das globales Minimum entweder nach Teil a des Theorem 1 in Zeile 4 der Hauptphase oder nach Teil b des Theorem 1 innerhalb der For-Schleife in Zeile 5 der Hauptphase, wenn wir keinen Intervallpunkt finden, der einen kleineren Abstand zu* x *hat.*
- *Wir finden einen Intervallpunkt* x^*, *der einen kleineren Abstand zu* x *im Vergleich* x_M *mit* x *hat. Hier aktualisieren wir erneut* $A^j_{k^+}(x)$, p^+_j, $A^j_{k^-}(x)$ *und* p^-_j *mit den Methoden* $min_{k^+}(.)$ *und* $min_{k^-}(.)$. *Dann fangen wir erneut mit einer neuer While-Schleife an, um zu prüfen, ob* x^* *ein lokales- oder globales Minimum ist.*

Damit haben wir bewiesen, dass wenn wir die While-Schleife in der Hauptphase verlassen, in der Endphase die Menge $N_1(x)$ *gleich der Menge* W *ist.*

Nachdem wir die Korrektheit des Algorithmus 1 bewiesen haben, versuchen wir, seine sequentielle und seine parallele Laufzeit für n Prozessoren $P_0, P_1, \ldots, P_{n-1}$ zu

[2] Der Grund, warum wir einen beliebigen Punkt $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in W$ wählen können, ist, dass er dem Theorem 2 Item 4 entspricht.

[3] x^* kann auch den gleichen Abstand zu x wie x_M mit x haben. In diesem Fall fügen wir x^* innerhalb der Methoden $!min_{k^+}(.)$ oder $!min_{k^-}(.)$ in Menge $W, W = W \cup \{x^*\}$ ein.

bestimmen, wobei wir, um bessere Optimalität bei der parallelen Berechnung zu erreichen, setzen wir n größer als J und kleiner/gleich d, $J < n \leq d$. Hier gilt $J := \max(J^1, J^2)$ (siehe Anhang).

Laufzeitanalyse 1 *-Algorithmus 1*

Sei im Folgenden $x \in X$, $x = (a_1, a_2, \ldots, a_d)$, $[x] = (i_1, i_2, \ldots, i_d)^T$ der Punkt, für den wir die Menge $N_1(x)$ bestimmen wollen. Sei $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in W$ der Kandidatspunkt für die erste Nachbarschaft mit x.

Dieser Algorithmus hat den Vorteil, alle Schritte parallel durchzuführen zu können. In Schritt 1 sortieren wir unsere Punkte in Menge X und bilden eine sortierte $(d \times N)$-Matrix M. Man muss $d \times N$ Attributswerte in d-Zeilen sortieren. Mit einem sequentiellen Sortieralgorithmus wie Merge-Sort können wir die Matrix M in $O(d \times N \log N)$ sortieren. In Schritt 2 berechnen wir für höchstens $2(T \times d)$ Punkte die Distanz zu unserem Punkt x und bilden das Minimum, wobei $T := \max_{j=1,2,\ldots d} |M_{(j,i_j \pm 1)}|$. Da die Distanzfunktion je nach Anwendung unterschiedlich sein kann, setzen wir allgemein den Aufwand für sie gleich $A(.)$. Hiermit haben wir für die Distanzberechnung von $2(T \times d)$ Punkten zu unserem Punkt x den Aufwand $2(T \times d \times A(.)) = O(T \times d \times A(.))$ und für die Bestimmung des Minimums brauchen wir $2(T \times d) - 1 = O(T \times d)$ Vergleiche. Insgesamt ist der Aufwand für den sequentiellen Fall $O(T \times d \times A(.)) + O(T \times d) = O(T \times d \times A(.))$ und für den parallelen Fall mit n Prozessoren $O(J^1 \times A(.))$ (siehe dazu PRAM 1 im Anhang), wobei J^1 die maximale Anzahl der berechneten Distanzen durch einen Prozessor ist. Bei der Intervallbildung vergleichen wir die Distanz von höchstens S Punkten, wobei $S := \left(\sum_j \left(|p_j^l - p_j^-| + |p_j^+ - p_j^r| + 2\right)\right)$, $j = j_1, j_2, \ldots, j_s$, und x_M hat die Eigenschaft $!min_{j_b}$, $b = 1, \ldots, s$, zu x mit unserem Minimum. Hier beträgt der Aufwand für den sequentiellen Fall $O(S \times A(.)) + O(S) = O(S \times A(.))$ und für den parallelen Fall mit n Prozessoren $O(J^2 \times A(.))$ (siehe dazu PRAM 3 im Anhang), wobei J^2 die maximale Anzahl der berechneten Distanzen durch einen Prozessor ist.

Bei der Laufzeitanalyse des Algorithmus 1 verzichten wir auf den Aufwand der Bestimmung der sortierten Matrix M, weil diese nicht nur für einen Punkt, sondern für alle Punkte $x \in X$ berechnet wird. Deswegen betrachten wir nur die Laufzeit der Schritte 2 und 3/item 1/Option b, weil die Laufzeit des Algorithmus 1 durch diese Schritte dominiert wird. Der Aufwand der Bestimmung der Menge der ersten Nachbarn für jeden Punkt $x \in X$ ist insgesamt im sequentiellen Fall $O(T \times d \times A(.)) + O(S \times A(.)) = O(\max(T \times d, S) \times A(.))$ und im parallelen Fall für n Prozessoren $O(J \times A(.))$. Wir haben auch bei der Analyse des Speicherverbrauchs gezeigt (siehe dazu PRAM 1,2 und 3 im Anhang), dass unser Algorithmus zusätzlich höchstens $O(J^1 \times n)$ Speicherplatz benötigt. Hier wird der Speicherverbrauch durch die Speicherung der Matrizen M und X dominert.

Um eine obere Schranke für die Größe S bestimmen zu können, nehmen wir die starke Voraussetzung an, dass die Attributswerte in Matrix M zeilen- und spaltenweise unabhängig voneinander vorkommen. Deswegen kommt jeder Attributswert in jeder Zeile in Matrix M mit der gleichen Wahrscheinlichkeit $\frac{1}{N}$ vor. D.h. der Attribustwert

$\acute{a}_{j_b}, b = 1, 2, \ldots, s$ des Punktes $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d)$ in Dimensionen j_b, in denen x_M die Eigenschaft $!min_{j_b}$ hat, kann an $N-1$ Stellen in der Matrix M vorkommen. Hier berechnen wir o.B.d.A., wie viele Attributswerte durschnittlich in Dimension j_1 zwischen Attributswert a_{j_1} des Punktes x, $[x] = (i_1, i_2, \ldots, i_d)^T$ und Attributswert $\acute{a}_{j_1}$ des Punktes x_M vorkommen können.

$\frac{1}{N-1} \sum_{r=1, r \neq i_{j_1}}^{N} |i_{j_1} - r| =$

$\frac{1}{N-1} \sum_{r=1}^{i_{j_1}-1} (i_{j_1} - r) + \frac{1}{N-1} \sum_{r=i_{j_1}+1}^{N} (r - i_{j_1}) =$

$\frac{1}{N-1} \left(\frac{i_{j_1}(i_{j_1}-1)}{2} \right) + \frac{1}{N-1} \left(\frac{(N-i_{j_1})(N-i_{j_1}+1)}{2} \right) =$

Da $i_{j_1}(i_{j_1}-1) + (N-i_{j_1})(N-i_{j_1}+1) \equiv N(N+1) - 2i_{j_1}(N-i_{j_1}+1)$ eine Identität ist, folgt:

$\frac{i_{j_1}(i_{j_1}-1)+(N-i_{j_1})(N-i_{j_1}+1)}{2(N-1)} = \frac{N(N+1)-2i_{j_1}(N-i_{j_1}+1)}{2(N-1)}$

$\leq \frac{N(N+1)}{2(N-1)} \approx \frac{N}{2}$

D.h für die Dimension $j_!$ wird höchstens die Distanz zwischen $\frac{N}{2}$-Punkten x^* und x berechnet. Da die Dimension j_1 eine beliebige Dimenson ist, wird für die Rest der Dimensionen auch höchstens die Distanz zwischen x und $\frac{N}{2}$-Punkten berechnet. Aber wir markieren jeden Punkt x^*, wenn wir die Distanz zwischen x^* und x berechnen und ignorieren x^*, wenn er in anderen Dimensionen vorkommt. Damit halbiert sich von Dimension zu Dimension sukzessiv die Anzahl der berechneten Distanzen zwischen x und anderen Punkten wie folgt:

Für Dimension j_1, $\frac{N}{2}$-Punkten

Für Dimension j_2, $\frac{N}{2^2}$-Punkten

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$

Für Dimension j_s, $\frac{N}{2^s}$-Punkten

Hiermit ist die Anzahl der gesamten berechneten Distanzen zwischen x und anderen Punkten gleich:

$\frac{N}{2} + \frac{N}{2^2} + \ldots + \frac{N}{2^s} = N \left(\sum_{b=1}^{s} \frac{1}{2^b} \right) < N$

Daraus folgern wir, dass der Algorithmus 1 effizient sein kann, wenn die Anzahl der Dimensionen s, in denen x_M die Eigenschaft $!min_{j_b}$, $b = 1, 2, \ldots, s$ hat, sehr viel kleiner ist als die gesamte Anzahl der Datensätze N, d.h. $s \ll N$.

Das ist ein großer Vorteil des Algorithmus 1, weil s nur abhängig ist von dem gesuchten ersten Nachbarschaftskandidaten x_M und nicht von der Anzahl der gesamten Dimensionen d von Punkt $x = (x_1, x_2, \ldots, x_d)$.

Bei der Untersuchung des Verhaltens des Algorithmus 2 machen wir eine empirische Untersuchung. Wir haben die Algorithmen 1 und 2 mittels der Programmiersprache Java implementiert. Damit können wir die Effizienz der Algorithmen 1 und 2 miteinander vergleichen.

Empirische Untersuchung 1 *-Algorithmus 2*

Wir erzeugen die Attributswerte jedes Punktes $x = (a_1, a_2, \ldots, a_d) \in X$, $|X| = N$ zufällig und unabhängig voneinander. Weiterhin definieren wir folgende Größen:
$D^1(x)$:=*Die Anzahl der berechneten Distanzen durch Algorithmus 1, um die Menge $N_1(x)$, $x \in X$ zu bestimmen.*
$D^2(x)$:=*Die Anzahl der berechneten Distanzen durch Algorithmus 2, um die Menge $N_1(x)$, $x \in X$ zu bestimmen.*
Seien in folgenden h^1, h^2, h^1_m, h^2_m, W^1, W^2, E^1 und E^2 definiert als:

Definition 10 *-h^1*
Wir definieren die durchschnittlich berechneten Distanzen durch Algorithmus 1 als:
$h^1 := \frac{1}{N} \sum_{x \in X} D^1(x)$

Definition 11 *-h^2*
Wir definieren die durchschnittlich berechneten Distanzen durch Algorithmus 2 als:
$h^2 := \frac{1}{N} \sum_{x \in X} D^2(x)$

Definition 12 *-h^1_m*
Wir definieren die maximal berechneten Distanzen für einen Punkt in Menge X durch Algorithmus 1 als:
$h^1_m := \max_{x \in X} D^1(x)$

Definition 13 *-h^2_m*
Wir definieren die maximal berechneten Distanzen für einen Punkt in Menge X durch Algorithmus 2 als:
$h^2_m := \max_{x \in X} D^2(x)$

Definition 14 *-W^1*
Wir definieren das Verhältnis von h^1 zu h^1_m als:
$W^1 := \frac{h^1}{h^1_m}$

Definition 15 *-W^2*
Wir definieren das Verhältnis von h^2 zu h^2_m als:
$W^2 := \frac{h^2}{h^2_m}$

Je mehr das Verhältnis der durchschnittlich berechneten Distanzen zu der maximal berechneten Distanz schrumpft, desto größer wird der Anteil der Datensätze, deren berechnete Distanzen im Vergleich zur berechneten maximalen Distanz signifikant kleiner sind.

Um für einen Datensatz $x \in X$ die Menge der ersten Nachbarschaft $N_1(x)$ zu bestimmen, braucht der naive Algorithmus $N - 1$ Distanzberechnung. Das zeigt, je niedriger die Anzahl der berechneten Distanzen für einen Datensatz ist, desto schneller und effizienter ist dieser Algorithmus. Hierzu definieren wir die Effizienz des Nachbarschaftsalgorithmus allgemein als:

Definition 16 *-Effizienz*

$Effizienz := \frac{Anzahl\,der\,Datensätze - durchschnittlich\,berechnete\,Distanzen}{Anzahl\,der\,Datensätze}$

Da wir aus der Statistik schon wisssen, dass der Mittelwert sehr empfindlich und variant gegenüber dem Maximum ist, haben wir absichtlich die durchschnittlich berechneten Distanzen für alle Datensätze der Eingabemenge als Parameter gewählt, um die Effizienz so sensibel und empfindlich wie nur möglich zu machen.

Um eine Vorstellung von dem definierten Begriff "Effizienz" zu bekommen, schreiben wir erst ein Beispiel:

Beispiel 3 *-*

Wir vergleichen zuerst die Effizienz des naiven Algorithmus mit dem besten Algorithmus, der für jeden Datensatz nur eine Distanz berechnet, um die Menge des ersten Nachbarn zu bestimmen. Hierfür haben beide Algorithmen gleiche Eingabemenge $X : |X| = N$.

Die Effizienz des naiven Algorithmus ist gleich:

$\frac{N - \frac{1}{N} \sum_{x \in X} berechnete\,Distanzen\,für\,x}{N} = \frac{N - \frac{1}{N} \sum_{x \in X} (N-1)}{N} =$

$\frac{N - \frac{N(N-1)}{N}}{N} = \frac{N-(N-1)}{N} = \frac{N-N+1}{N} = \frac{1}{N}$

Die Effizenz des besten Algorithmus ist gleich:

$\frac{N - \frac{1}{N} \sum_{x \in X} berechnete\,Distanzen\,für\,x}{N} = \frac{N - \frac{1}{N} \sum_{x \in X} 1}{N} =$

$\frac{N - \frac{N}{N}}{N} = \frac{N-1}{N}$

Hier sehen wir sofort:

$\lim_{N \to +\infty} \frac{1}{N} = 0$

$\lim_{N \to +\infty} \frac{N-1}{N} = \lim_{N \to +\infty} \frac{N(1-\frac{1}{N})}{N} = \lim_{N \to +\infty} (1 - \frac{1}{N}) = 1$

D.h. beide Algorithmen sind im Bezug auf die Effizienz extrem. Alle andere Algorithmen können eine Effizienz größer als Null und kleiner als Eins haben.

Als nächstes betrachten wir einen Algorithmus A, bei dem wir annehmen, dass er bei der Eingabemenge $X : |X| = 100$ *und wenn die Anzahl der durchschnittlich berechneten Distanzen bei 10 liegt eine Effizienz von* $\frac{100-10}{100} = 0.9$ *besitzt. Falls wir die Anzahl der Datensätze*[4] *verzehnfachen und somit die Anzahl der durchschnittlich berechneten Distanzen 100 beträgt, dann hat der Algorithmus A eine Effizienz von* $\frac{1000-100}{1000} = 0.9$. *D.h. wenn die Anzahl der Datensätze vervielfach wird, verbleibt die Effizienz unverändert, falls die Anzahl der durchnittlich berechneten Distanzen nicht eine obere Schranke überschreitet (hier für Algorithmus A 100). Wenn Algorithmus A weniger als 100 durchschnittlich berechnete Distanzen braucht, z.B. 99, dann liegt ein Effizienzgewinn entsprechend*

$\frac{1000-99}{1000} - \frac{100-10}{100} = 0.901 - 0.9 = 0.001$

[4]Hier haben alle Datensätze die gleiche Dimensionsanzahl.

vor.
Um Umkehrschuss bedeutet dies, falls Algorithmus A mehr als 100 durchschnittlich berechnete Distanzen gebraucht, z.B. 101, dann hat Algorithmus A einen Effizienzverlust gleich:
$\frac{1000-101}{1000} - \frac{100-10}{100} = 0.899 - 0.9 = -0.001$
Daraus können wir folgendes schließen:

1. *Da der Wert der Effizienz normiert ist, können wir die Effizienz von zwei Nachbarschaftsalgorithmen bei identischer Eingabe miteinander vergleichen.*
2. *Wir können das Verhalten von einem Nachbarschaftsalgorithmus im Bezug auf seine Effizienz beobachten, falls wir die Anzahl der Datensätze in der Eingabemenge vervielfachen.*

Weiterhin definieren wir hier E^1 und E^2 als:

Definition 17 *-E^1*
Wir definieren E^1 als:
$E^1 := \frac{N-h^1}{N}$

Definition 18 *-E^2*
Wir definieren E^2 als:
$E^2 := \frac{N-h^2}{N}$

Wir charakterisieren die Menge der Datensätze X mit zwei Merkmalen N (Anzahl der Datensätze) und d (Anzahl der Dimensionen eines Datensatzes). Wir untersuchen das Verhalten der Algorithmen 1 und 2[5] mit gleicher Eingabe X im Bezug auf die oben definierten Merkmale h^1, h^2, h^1_m, h^2_m, W^1, W^2, E^1 und E^2 folgenden Situationen aus:

1. *Wir setzen d gleich eine Konstante Zahl d_c und lassen N linear Wachsen. Hierfür setzen wir jedes Mal d_c gleich 2, 3, 4, 5 und lassen jeweils N von 100 bis 100000 wachsen. Wir bezeichnen die Gleichsetzung von d mit einer konstante d_c und das Wachstum von N als eine Periode. Dann erstellen wir Tabelle 1 wie folgt:*

[5]Wir setzen den k-ten Intervallradius für den Algorithmus 2 in unserer Untersuchung gleich 1.

(N, d_c)	h^2	h_m^2	W^2	E^2	h^1	h_m^1	W^1	E^1
(100,2)	*14.6*	*38*	*0.3842*	*0.854*	*22.6*	*75*	*0.3013*	*0.774*
(1000,2)	*45.6*	*177*	*0.2576*	*0.9544*	*206.9*	*842*	*0.2457*	*0.7931*
(10000,2)	*147.5*	*598*	*0.2466*	*0.9852*	*2026.5*	*9578*	*0.2116*	*0.7973*
(100000,2)	*467.2*	*2187*	*0.2136*	*0.9953*	*19141.8*	*99942*	*0.1915*	*0.8086*
(100,3)	*33.3*	*64*	*0.5203*	*0.667*	*44.9*	*88*	*0.5102*	*0.551*
(1000,3)	*167.6*	*361*	*0.4643*	*0.8324*	*434.4*	*963*	*0.4511*	*0.5656*
(10000,3)	*815.6*	*2164*	*0.3769*	*0.9184*	*4421*	*9978*	*0.4431*	*0.5579 (*)*
(100000,3)	*3841*	*10994*	*0.3494*	*0.9615*	*42559.6*	*99977*	*0.4257*	*0.5744*
(100,4)	*52.8*	*85*	*0.6212*	*0.4715*	*63.4*	*97*	*0.6536*	*0.3664*
(1000,4)	*341.1*	*638*	*0.5346*	*0.6589*	*633.3*	*995*	*0.6365*	*0.3667*
(10000,4)	*2013.6*	*4721*	*0.4265*	*0.7986*	*6284.6*	*9999*	*0.6285*	*0.3715*
(100000,4)	*11681.3*	*27611*	*0.4231*	*0.8832*	*61567.7*	*99997*	*0.6157*	*0.3843*
(100,5)	*68.9*	*94*	*0.733*	*0.3109*	*75.2*	*98*	*0.7673*	*0.2478*
(1000,5)	*505.5*	*894*	*0.5654*	*0.4945*	*760*	*999*	*0.7608*	*0.2398*
(10000,5)	*3496.8*	*6019*	*0.581 (*)*	*0.6503*	*7598.2*	*9998*	*0.7600*	*0.2402*
(100000,5)	*23215.3*	*46408*	*0.5002*	*0.7678*	*75351*	*99998*	*0.7535*	*0.2465*

Tabelle 1: Die Outlier wurden mit (*) markiert. Aus der Tabelle 1 können wir ersehen, dass bei der gleich gebliebenden Anzahl der Dimensionen und dem Wachstum der Anzahl der Datensätze die Effizienz der Algorithmen 1 und 2 steigt und das Verhältnis der durchschnittlichen Anzahl der berechneten Distanzen zu der maximal berechneten Distanz (W^1 und W^2) fällt. Hier sehen wir, dass für jede Perionde $d_c = 2, 3, 4, 5$ die Anzahl der berechneten Distanzen linear wächst.

h^1 :=*siehe Definition 10 Seite 43*

h^2 :=*siehe Definition 11 Seite 43*

h_m^1 :=*siehe Definition 12 Seite 44*

h_m^2 :=*siehe Definition 13 Seite 44*

W^1 :=*siehe Definition 14 Seite 44*

W^2 :=*siehe Definition 15 Seite 44*

E^1 :=*siehe Definition 17 Seite 46*

E^2 :=*siehe Definition 18 Seite 47*

Folgende Phänomene können wir bei Tabelle 1 feststellen:

Schlussfolgerung 1 -

a- E^1 *und* E^2 *steigen:*
Wir sehen für jede Periode d_c gleich 2, 3, 4 und 5 wachsen die Werte von E^1 und E^2spaltenweise. Hier wächst E^2 im Vergleich zu E^1 stärker. Weiterhin sehen wir, dass h^1, h_m^1, h^2 und h_m^2 spaltenweise linear wachsen.

b- W^1 *und* W^2 *fallen:*
Wir sehen für jede Periode d_c gleich 2, 3, 4 und 5 fallen die Werte von W^1 und W^2spaltenweise. Hier fällt W^2 insgesamt stärker als W^1.

c- *In jeder Zeile haben wir folgende Ordnungen:*
$h^2 < h^1$, $h^2_m < h^1_m$, $E^2 > E^1$

d- *Es gibt nur zwei Outlier, welche durch (*) markiert wurden.*

2. *Wir nutzen die Werte der Tabelle 1 und stellen die Tabelle 1 um. Hierfür setzen wir N gleich einer konstanten Zahl N_c und setzen jedes Mal N_c gleich 100, 1000, 10000, 100000 und lassen jeweils d von 2 bis 5 wachsen. Wir bezeichnen die Gleichsetzung von N mit einer Konstante N_c und das Wachstum von d als eine Periode. Daraus erstellen wir Tabelle 2 wie folgt:*

(N_c, d)	h^2	h^2_m	W^2	E^2	h^1	h^1_m	W^1	E^1
(100,2)	14.6	38	0.3842	0.854	22.6	75	0.3013	0.774
(100,3)	33.3	64	0.5203	0.667	44.9	88	0.5102	0.551
(100,4)	52.8	85	0.6212	0.4715	63.4	97	0.6536	0.3664
(100,5)	68.9	94	0.733	0.3109	75.2	98	0.7673	0.2478
(1000,2)	45.6	177	0.2576	0.9544	206.9	842	0.2457	0.7931
(1000,3)	167.6	361	0.4643	0.8324	434.4	963	0.4511	0.5656
(1000,4)	341.1	638	0.5346	0.6589	633.3	995	0.6365	0.3667
(1000,5)	505.5	894	0.5654	0.4945	760	999	0.7608	0.2398
(10000,2)	147.5	598	0.2466	0.9852	2026.5	9578	0.2116	0.7973
(10000,3)	815.6	2164	0.3769	0.9184	4421	9978	0.4431	0.5579
(10000,4)	2013.6	4721	0.4265	0.7986	6284.6	9999	0.6285	0.3715
(10000,5)	3496.8	6019	0.581	0.6503	7598.2	9998	0.7600	0.2402
(100000,2)	467.2	2187	0.2136	0.9953	19141.8	99942	0.1915	0.8086
(100000,3)	3841	10994	0.3494	0.9615	42559.6	99977	0.4257	0.5744
(100000,4)	11681.3	27611	0.4231	0.8832	61567.7	99997	0.6157	0.3843
(100000,5)	23215.3	46408	0.5002	0.7678	75351	99998	0.7535	0.2465

Tabelle 2: Aus Tabelle 2 können wir ersehen, dass bei einer gleichbleibenden Anzahl der Datensätze und dem Wachstum der Anzahl der Dimensionen die Effizienz der Algorithmen 1 und 2 fällt und das Verhältnis der durchschnittlichen Anzahl der berechneten Distanzen zu der maximal berechneten Distanz (W^1 und W^2) steigt. Hier sehen wir, dass für jede Perionde $N_c = 100, 1000, 10000, 100000$ die Anzahl der berechneten Distanzen linear wächst.

h^1 :=*siehe Definition 10 Seite 43*

h^2 :=*siehe Definition 11 Seite 43*

h^1_m :=*siehe Definition 12 Seite 44*

h^2_m :=*siehe Definition 13 Seite 44*

W^1 :=*siehe Definition 14 Seite 44*

W^2 :=*siehe Definition 15 Seite 44*

E^1 := *siehe Definition 17 Seite 46*

E^2 := *siehe Definition 18 Seite 47*

Folgende Phänomene können wir bei Tabelle 2 feststellen:

Schlussfolgerung 2 -

a- *E^1 und E^2 Fallen:*
Wir sehen für jede Periode N_c gleich 100, 1000, 10000 und 100000 fallen die Werte von E^1 und E^2spaltenweise. Hier fällt E^1 stärker als E^2. Weiterhin sehen wir, dass h^1, h^1_m, h^2 und h^2_m spaltenweise linear wachsen.

b- *W^1 und W^2 steigen:*
Wir sehen für jede Periode N_c gleich 100, 1000, 10000 und 100000 wachsen die Werte von W^1 und W^2spaltenweise. Hier wächst W^1 stärker als W^2.

c- *In jeder Zeile haben wir folgende Ordnungen:*
$h^2 < h^1$, $h^2_m < h^1_m$, $E^2 > E^1$

Aus unseren beobachteten Phänomenen der Tabellen 1 und 2 können wir folgende Schlussfolgerungen ziehen:

Schlussfolgerung 3 -

1. *Wegen Item c der Schlussfolgerung 1 und 2 ist Algorithmus 2 im Vergleich zu Algorithmus 1 effizienter und schneller.*
2. *Wegen der Items a und b der Schlussfolgerung 1 werden Algorithmen 1 und 2[6] effizienter, je mehr die Anzahl der Datensätze N im Vergleich zur Anzahl der Dimensionen d steigt, d.h. für $d \ll N$. Wir sehen weiterhin ein lineares Wachstum in Bezug auf die Anzahl der berechneten Distanzen.*
3. *Wegen der Items a und b der Schlussfolgerung 2 werden Algorithmus 1[7] und 2 bei der gleich gebliebenden Anzahl der Datensätze N und Wachstum der Dimensionen d ihre Effizienz verlieren. Wir sehen weiterhin ein lineares Wachstum im Bezug auf die Anzahl der berechneten Distanzen.*
4. *Um die Menge der ersten Nachbarschaftskandidatenpunkte W zu bestimmen, brauchen wir die Distanz von höchstens $2(T(k \times d))$ Punkten zu unserem Query-Punkt zu berechnen. Das ist eine sehr pessimistische obere Schranke. Praktisch sind die Anzahl der berechneten Distanzen linear zur Anzahl der Dimensionen und dem Intervallradius. Das heißt $O(kd)$.*
5. *Wir sehen eindeutig, dass Algorithmus 1 und 2 beide skalierbar sind. Beide Algorithmen haben für den Fall, dass $d \ll N$ ist, ein sehr gutes Verhalten.*

3.4. Vergleich des Algorithmus 2 mit dem RKV-Algorithmus

In diesem Abschnitt skizzieren wir zunächst den R-Baum[8] [52]. Dann beschreiben wir den RKV-Algorithmus und vergleichen ihn mit unserem Algorithmus 2.

[6] Hier wird Algorithmus 2 besonders effizient.

[7] Hier verliert Algorithmus 1 besonders im Vergleich zu Algorithmus 2 seine Effizienz stärker. Insgesamt können wir sagen, dass Algorithmus 2 im Vergleich zu Algorithmus 1 stabiler ist.

[8] Unter der großen Anzahl der Datenstrekturen, die für hoch dimensionale Daten entwickelt wurden, wurde der R-Baum wegen seiner Einfachheit, seiner durchschnittlichen Performanz und seiner Fähigkeit, hoch dimensonale Daten (bis zu 20 Dimensionen) zu behandeln, populär.

Hier wir skizzieren kurz die Grundeigenschaften des R-Baumes. Der R-Baum ist eine hierarchische Datenstruktur auf der Grundlage des B^+-Baumes [56]. Er wurde als eine festplattenbasierte Zugriffsmethode entwickelt, um Rechtecke zu organisieren. Man kann effizient mehrdimensionale Daten in einem R-Baum abspeichern und suchen. Die Datensätze werden hierarchisch in Rechtecken verteilt, so dass im Falle einer Wurzel oder eines inneren Knotens die Rechtecke alle Datensätze aller Kindsknoten beinhalten. Diese Rechtecke (im Folgenden minimum bounding d-dimensional rectangles, kurz MBRs genannt) sind dabei möglichst klein gehalten, um größtmögliche Effizienz zu gewährleisten. Dabei beinhaltet jeder innere Knoten sowie die Wurzel ein Rechteck und einen Pointer, welcher auf den nächsten Kindsknoten zeigt. Ein Blatt beinhaltet mindestens ein Rechteck, wovon jedes nur ein Datenobjekt beinhaltet. Hierbei ist es sehr wahrscheinlich, dass sich die MBRs überlappen, was bedeutet, dass bei einer Suche mehrere Teilbäume durchsucht werden müssen, was der Nachteil dieser Datenstruktur ist.

Der R-Baum hat demnach folgende grundlegende Eigenschaften:

- Der Baum ist balanciert.
- Jedes Blatt enthält mindestens ein Paar der Form (MBR, O), wobei O dem abgespeicherten Datenobjekt entspricht.
- Jeder innere Knoten beinhaltet mindestens ein Paar der Form (MBR, P), wobei P der Pointer zum nächsten Kindsknoten ist.
- Jeder Knoten eines R-Baumes der Klasse (k, K), wobei die Wurzel dabei eine Ausnahme darstellen kann, beinhaltet zwischen k und K Paare, wobei gilt, dass $k \leq \left\lceil \frac{K}{2} \right\rceil$.
- Die Wurzel beinhaltet mindestens zwei Paare, solange sie nicht gleichzeitig ein Blatt ist.

Nachdem wir den R-Baum nun kurz umrissen haben, versuchen wir, den RKV Algorithmus zu beschreiben.

Das Prinzip des RKV ist Folgendes:

Um den nächsten Nachbarn für einen Query-Punkt zu finden, traversieren wir ab der Wurzel bis zu einem Blatt. Währendessen versuchen wir, soweit möglich, Teilbäume, die für die Suche irrelevant sind, aus der Suche auszuschließen. Dafür haben die Entwickler des RKV Algorithmus zwei Distanzfunktionen $MinDist$ (kleinstmögliche Distanz zum MBR, optimistische Ansicht) und $MinMaxDist$ (die "kleinste" pessimistische Ansicht) von Query-Punkt zum MBR aufgestellt. Diese nutzen für den Query-Punkt $x = (a_1, a_2, \ldots, a_d)$ und den MBR mit den Koordinaten von der linken unteren Ecke $l = (l_1, l_2, \ldots, l_d)$ und der rechten oberen Ecke $r = (r_1, r_2, \ldots, r_d)$. Diese Distanzfunktionen sind wie folgt definiert:

$MinDist(x, MBR) := \sqrt{\sum_{j=1}^{n} |x_j - s_j|^2}$

wobei

$$s_j := \begin{cases} l_j & x_j < l_j \\ r_j & x_j > r_j \\ x_j & sonst \end{cases}$$

sowie

$$MinMaxDist(x, MBR) := \sqrt{\min_{1 \le i \le d}(|x_i - rm_i|^2 + \sum_{1 \le j \le d,\, j \neq i} |x_j - rM_j|^2)}$$

wobei

$$rm_i := \begin{cases} l_i & falls\, x_i \le \frac{l_i + r_i}{2} \\ r_i & sonst \end{cases}$$

$$rM_j := \begin{cases} l_j & falls\, x_j \ge \frac{l_j + r_j}{2} \\ r_j & sonst \end{cases}$$

Damit ergibt sich, dass $MinDist$ der optimistischen Metrik entspricht, da es die kleinstmögliche Distanz zum nächsten Nachbarn des Query-Punktes in MBR ist. Parallel dazu entspricht $MinMaxDist$ der pessimistischen Metrik, da es die längste mögliche Distanz zum nächsten Nachbarn vom Query-Punkt bezüglich MBR darstellt. Daraus ergibt sich, dass der nächste Nachbar des Query-Punktes nicht weiter entfernt liegt als $MinMaxDist$ angibt.

Die folgenden drei Regeln werden dazu genutzt, bei einer Suche überflüssige Suchbäume aus der Suche auszugrenzen:

1. Ein spezifischer MBR M mit der der Wert $MinDist(x, M)$ größer als der Wert $MinMaxDist(x, M')$ eines anderen MBR M' wird ausgeschlossen, da er nicht den nächsten Nachbarn beinhalten kann. Wir verwenden diese Regel beim Traversieren zum Blatt.
2. Ein tatsächlicher Abstand von x zu einem gegebenen Datenobjekt O, welcher größer ist als die $MinMaxDist(x, M)$ für MBR M wird ersetzt durch den Wert von $MinMaxDist(x, M)$. Wir verwenden diese Regel beim Traversieren zum Blatt.
3. Jeder MBR M mit $MinDist(x, M)$ größer als die tatsächliche Distanz von x zu einem gegebenen Datenobjekt O wird ausgeschlossen, da es nicht näher an das Objekt kommen kann als O. Wir verwenden diese Regel beim Traversieren zur Wurzel.

Wenn wir von einer Wurzel zu einem Blatt traversieren, verwenden wir Regel 1 und 2, um unnötige Teilbäume auszuschließen und dann wird nach einer Prioritätenregel (entweder nach $MinMaxDist$ oder $MinDist$) ein Suchpfad ausgewählt.

Wenn wir von einem Blatt aufwärts traversieren, dann verwenden wir Regel 3, um andere Pfade auszuschließen (insofern welche vorhanden sind).

Wir geben jetzt in Pseudocode den RKV an [57]:

float pruning_dist /* Die aktuelle Distanz vom ausgeschlossenen Zweig */
= $+\infty$; /* Initialisierung vor dem Start des RKV Algorithmus */
Point cpc; /* Der nächste Kandidatenpunkt. Diese Variable wird den nächsten Nachbarn beinhalten, wenn der RKV Algorithmus beendet wurde. */

```
1- RKV_algorithm (Point q, Metrik m, PageAdr pa) {
2- int i ; float h;
3- Page p = LoadPage(pa) ;
4- if( isDatapage(p) ) {
5- for ( i = 0 ; i < p.num_objects; i++){
6- h = PointToPointDist( q, p.object[i], m );
7- if ( pruning_dist >= h) {
8- pruning_dist = h ;
9- cpc = p.object[i] ;
10- } // Zeile 7
11- } // Zeile 5
12- } // Zeile 4
13- if ( isDirectoryPage ( p ) ){
14- sort ( p, CRITERION);      /*CRITERION ist MinDist oder MinMaxDist */
15- for ( i = 0; i < p.num_objects ; i++) {
16- if ( MinDist ( q, p.region[i]), m) <= pruning_dist)
17- RKV_algorithm ( q, m, p.childpage[i] );
18- h = MinMaxDist ( q, p.region[i], m );
19- if (pruning_dist >= h)
20- pruning_dist = h;
21- } // Zeile 15
22- } // Zeile 13
23- } // Zeile 1
```

Die Entwickler des RKV Algorithmus haben in ihrer Veröffentlichung als Zusammenfassung empirisch gezeigt, dass der Algorithmus im Bezug auf k, d.h. die Anzahl der gesuchten Nachbarn und auf die Anzahl der Datensätze beliebig skalierbar ist. Außerdem geben sie keine Garantie, dass der Algorithmus im Bezug auf die Dimension und die Anzahl der Datensätze immer eine sublineare oder logarithmische Laufzeit hat. Wir haben im letzten Abschnitt auch empirisch gezeigt, dass unser Algorithmus 2 in Hinsicht der Anzahl der Datensätze und der Dimensionen ebenfalls beliebig skalierbar ist und haben gezeigt, wenn die Anzahl der Dimensionen sehr viel kleiner als die Anzahl der Datensätze ist, dass der Algorithmus ein sehr gutes Verhalten zeigt. Der RKV Algorithmus wird auf den R-Baum angewendet, der eigentlich den Vorteil hat, ihn ebenfalls gut für dynamische Datenbanken zu nutzen, allerdings auch den Nachteil hat, dass es sich dabei um eine sehr komplexe Datenstuktur handelt. Bei unserem Algorithmus haben wir eine Matrix benutzt, was an sich eine sehr simple Datenstruktur ist, die jedoch den Nachteil hat, dass sie sich nicht für dynamische Datenbanken eignet.

4. K-nächste Nachbarn

In diesem Kapitel beschäftigen wir uns mit der Entwicklung eines Algorithmus zum K-nächste Nachbarproblem. Hierfür erweitern wir Algorithmus 2. Danach beweisen wir die Korrektheit des Algorithmus und untersuchen dessen Verhalten empirisch.

4.1. Entwicklung des Algorithmus

In diesem Abschnitt entwickeln wir einen Algorithmus zum K-nächsten Nachbarproblem. Um dies zu bewältigen, erweitern wir Algorithmus 2.

Algorithmus 2 findet die Menge der ersten Nachbarn für einen beliebigen Datensatz $x = (a_1, a_2, \ldots, a_d) \in X$. Für diesen Zweck findet er zunächst in der Initialisierungsphase einen/mehrere Kandidat/Kandidaten, der/die den kleinsten Abstand zu dem Punkt x hat/haben. Dann untersucht er in der Hauptphase, ob dieser Kandidat/diese Kandidaten ein globales- oder lokales Minimum darstellt/darstellen. Dafür wählt er einen beliebigen Kandidaten x_M und untersucht, ob dieser in allen Dimensionen die Eigenschaft min_j, $j = 1, 2, \ldots, d$ besitzt. Wenn dies der Fall ist, dann hat Algorithmus 2 den ersten Nachbarn gefunden, ansonsten bildet Algorithmus 2 Intervalle in Dimensionen, in denen x_M die Eigenschaft $!min_j$ besitzt und durchsucht diese Intervalle nach einem Intervallpunkt, welcher einen kleineren Abstand zu x hat. Wenn Algorithmus 2 keinen Intervallpunkt x^* findet, der einen kleineren Abstand zu x hat, dann ist x_M globales Minimum, ansonsten wird x^* als neuer erster Nachbarschaftskandidat gewählt und der oben geschilderte Prozess für x^* in der Hauptphase wird wiederholt. Während dieses Vorgangs werden alle Intervallpunkte als behandelt markiert.

Unsere Idee ist es, den Algorithmus 2 K-Mal mit entsprechenden Nachbarschaftskandidaten aufzurufen. Hierfür verwenden wir die zwei Mengen W und $minimal$. In der Menge W sind alle Kandidaten enthalten. Wir entfernen sukzessiv die i-ten Nachbarschaftskandidaten $i = 1, 2, \ldots, k$ aus der Menge W und fügen sie der Menge $minimal$[1] hinzu. Dann rufen wir Algorithmus 2 zur Bestimmung der i-ten Nachbarschaftsmenge mit den entsprechenden i-ten Nachbarschaftskandidaten in der Menge $minimal$ auf.

Sei $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ ein beliebiger Punkt, für den wir die K-nächsten Nachbarschaftsmengen $N_i(x)$, $i = 1, 2, \ldots, K$ bestimmen wollen.

[1] Vor der Initialisierung mit den i-ten Nachbarschaftskandidaten i=2,3,...,K wird die Menge minimal gleich leere Menge gesetzt.

In unserem K-nächsten Nachbarschaftsalgorithmus verwenden wir folgende globale Parameter und Variablen:

Globale Parameter

- K, der Grad der Nachbarschaftsuche.
- K_1 Intervallradius $K_1 \in \{1, 2, \ldots\}$.

Globale Variablen

- $W :=$ Wir speichern Punkte in der Menge W.
- $minimal :=$ Wir speichern Punkte in der Menge $minimal$.
- $p :=$ Zähler in While-Schleifen
- $gibtMinimum :=$ boolesche Variable
- $x.behandelt :=$ Eine boolesche Variable für jeden Punkt $x \in X$, die zu Beginn als Falsch initialisiert wird.

Algorithm 3 - *K-nächste Nachbarschaftsalgorithmus*

Initialisierungsphase

1- Bestimme sortierte Matrix M.

2- Bestimme K *und* K_1.

3- Berechne den Abstand zwischen $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ *und den Punkten* $x^* : x^* \in M_{(j,i_j \pm l)}$, $1 \leq l \leq K_1$, $j = 1, 2, \ldots, d$ *in Matrix* M *und setze boolesche Variable aller* $x^*.behandelt$ *gleich Wahr* $x^*.behandelt = Wahr$. *Füge alle* x^* *in Menge* W *ein* $W = W \cup \{x^*\}$.

4- Bestimme $A^j_{K_1}(x) = \{A^j_{K_1^+}(x), A^j_{K_1^-}(x)\}$, $j = 1, 2, \ldots, d$.

5- Setze p gleich eins $p = 1$.

Hauptphase

BEGIN

1- $While(p \leq K)$ *{*

2- Finde Punkte $x_M \in W$ *mit dem kleinsten Abstand zu* x. *Füge alle* x_M *in Menge* $minimal$ *ein* $minimal = minimal \cup \{x_M\}$.

3- Setze boolesche Variable $gibtMinimum$ *gleich Wahr* $gibtMinimum = Wahr$.

4- $While(gibtMinimum)$ *{*

5- Setze boolesche Variable $gibtMinimum$ *gleich Falsch* $gibtMinimum = Falsch$.

6- Wähle einen beliebigen Punkt $x_M = (\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_d) \in minimal$.

7- Prüfe, ob x_M *in jeder Dimension* $j = 1, 2, \ldots, d$ *die Eigenschaft* min_j *besitzt. Wenn " Ja ", dann haben wir das globale p-te Minimum gefunden. Deswegen breche die While-Schleife in Zeile 4 ab.*

8- For alle Dimensionen $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$, *in denen* x_M *die Eigenschaft* $!min_{j_b}$, $b = 1, 2, \ldots, s$ *hat, tue folgendes {*

9- Setze boolesche Variable b_1 gleich Wahr, falls die folgende Bedingung gilt:

- $B_1 := (A^{j_b}_{K_1^+}(x) \neq +\infty)\,\&\,(p^+_{j_b}+1 \leq N)\,\&\,(|A(j_b, p^+_{j_b}+1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$

10- Setze boolesche Variable b_2 gleich Wahr, falls die folgende Bedingung gilt:

- $B_2 := (A^{j_b}_{K_1^-}(x) \neq +\infty)\,\&\,(p^-_{j_b}-1 \geq 1)\,\&\,(|A(j_b, p^-_{j_b}-1) - a_{j_b}| \leq dist_{j_b}(x, x_M))$

11- Falls b_1 gleich Wahr ist, dann inkrementiere $p^+_{j_b}$, $p^+_{j_b} = p^+_{j_b}+1$ und aktualisiere den Wert von $A^{j_b}_{K_1^+}(x)$, $A^{j_b}_{K_1^+}(x) = A(j_b, p^+_{j_b})$, $K_1^+ = K_1^+ + 1$.

12- Falls b_2 gleich Wahr ist, dann dekrementiere $p^-_{j_b}$, $p^-_{j_b} = p^-_{j_b} - 1$ und aktualisiere den Wert von $A^{j_b}_{K_1^-}(x)$, $A^{j_b}_{K_1^-}(x) = A(j_b, p^-_{j_b})$, $K_1^- = K_1^- + 1$.

13- $!MIN_{K_1^+}(j_b, b_1)$.

14- $!MIN_{K_1^-}(j_b, b_2)$.

15- Falls boolesche Variable gibtMinimum gleich Wahr ist, dann bereche $For-Schleife$ in Zeile 8 ab.

}// end $For-Schleife$ Zeile 8

16- $if\,(gibtMinimum)\{$

17- For alle Dimensionen $r = 1, 2, \ldots, d$ tue folgendes {

18- $min_{K_1^+}(r)$.

19- $min_{K_1^-}(r)$.

}//end For-Schleife Zeile 17

}//end if Zeile 16

}// end $While-Schleife$ Zeile 4

Endphase

20- Setze $N_p(x)$ gleich minimal, $N_p(x) = minimal$.

21- Entferne alle $x_M \in minimal$ aus der Menge W, $W = W - \{x_M\}$.

22- Setze minimal gleich leere Menge $minimal = \emptyset$.

23- Inkrementiere p, $p = p + 1$.

} //end While-Schleife Zeile 1

END

Hier schreiben wir die verwendeten Methoden, die sich von denen im Algorithmus 2 unterscheiden und sich in unserem K-nahsten Nachbarschaftsalgorithmus finden:

$!MIN_{K_1^+}(int\,j, boolean\,a)$ {

1- $While(a)\{$

2- Setze boolesche Variable a gleich Falsch $a = Falsch$.

3- For alle x^* : $x^* \in M_{(j,p^+_j)}$ tue folgendes {

4- $if\,(x^*.behandelt == Falsch)\{$

5- Setze $x^*.behandelt$ gleich Wahr $x^*.behandelt = Wahr$.

6- $if(dist(x, x^*) \leq dist(x, x_M))\{$

7- $if(dist(x, x^*) == dist(x, x_M))$ {

8- $W = W \cup \{x^*\}$.

9- $minimal = minimal \cup \{x^*\}$

}// end if Zeile 7

10- $if\,(dist(x, x^*) < dist(x, x_M))\{$

11- $W = W \cup \{x^*\}$.

12- Setze $minimal$ gleich leere Menge $minimal = \emptyset$.

13- Füge neuen p-ten Nachbarschaftskandidaten in Menge $minimal$ ein $minimal = minimal \cup \{x*\}$.

14- Setze boolesche Variable $gibtMinimum$ gleich Wahr $gibMinimum = Wahr$.

}// end if Zeile 10

}// end if Zeile 6

}// end if Zeile 4

}// end if Zeile 3

15- Falls $gibtMinimum$ gleich Wahr ist, dann breche $While - Schleife$ in Zeile 1 ab.

16- $if\,((p_j^+ + 1 \leq N)\,\&\,(|A(j, p_j^+ + 1) - a_j| \leq dist_j(x, x_M))\{$

17- Inkrementiere p_j^+, $p_j^+ = p_j^+ + 1$.

18- Aktualisiere $A^j_{K_1^+}(x)$, $A^j_{K_1^+}(x) = A(j, p_j^+)$, $K_1^+ = K_1^+ + 1$.

19- Setze boolesche Variable a gleich $Wahr$ $a = Wahr$.

}// end if Zeile 16

}// end $while - Schleife$ Zeile 1

}//end Methode

$!MIN_{K_1^-}(int\,j, boolean\,a)$ {

1- $While(a)\{$

2- Setze boolesche Variable a gleich Falsch $a = Falsch$.

3- For alle $x^* : x^* \in M_{(j,p_j^-)}$ tue folgendes {

4- $if\,(x^*.behandelt == Falsch)\{$

5- Setze $x^*.behandelt$ gleich Wahr $x^*.behandelt = Wahr$.

6- $if(dist(x, x^*) \leq dist(x, x_M))\{$

7- $if(dist(x, x^*) == dist(x, x_M))$ {

8- $W = W \cup \{x^*\}$.

9- $minimal = minimal \cup \{x^*\}$

}// end if Zeile 7

10- $if\,(dist(x, x^*) < dist(x, x_M))\{$

11- $W = W \cup \{x^*\}$.

12- Setze $minimal$ gleich leere Menge $minimal = \emptyset$.

13- Füge neuen p-ten Nachbarschaftskandidaten in Menge $minimal$ ein $minimal = minimal \cup \{x*\}$.

14- Setze boolesche Variable $gibtMinimum$ gleich Wahr $gibMinimum = Wahr$.

}// end if Zeile 10

}// end if Zeile 6

}// end if Zeile 4

}// end if Zeile 3

15- Falls $gibtMinimum$ gleich Wahr ist, dann breche $While - Schleife$ in Zeile 1 ab.

16- $if\,((p_j^- - 1 \geq 1)\,\&\,(|A(j, p_j^- - 1) - a_j| \leq dist_j(x, x_M))\{$

17- Dekrementiere p_j^-, $p_j^- = p_j^- - 1$.

18- Aktualisiere $A^j_{K_1^-}(x)$, $A^j_{K_1^-}(x) = A(j, p_j^-)$, $K_1^- = K_1^- + 1$.

19- Setze boolesche Variable a gleich $Wahr$ $a = Wahr$.

}// end if Zeile 16

}// end $while - Schleife$ Zeile 1

}//end Methode

$min_{K_1^+}(int\, j)\{$

1- $if\,(A^j_{K_1^+}(x) \neq +\infty)\{$

2- $While(p_j^+ + 1 \leq N\,\&\,Behandelt(j, p_j^+ + 1))\{$

3- Inkrementiere p_j^+, $p_j^+ = p_j^+ + 1$.

4- Aktualisiere $A^j_{K_1^+}(x)$, $A^j_{K_1^+}(x) = A(j, p_j^+)$, $K_1^+ = K_1^+ + 1$.

}// end $While - Schleife$ Zeile 2

}// end if Zeile 1

}//end Methode

$min_{K_1^-}(int\, j)\{$

1- $if\,(A^j_{K_1^-}(x) \neq +\infty)\{$

2- $While(p_j^- - 1 \geq 1\,\&\,Behandelt(j, p_j^- - 1))\{$

3- Dekrementiere p_j^-, $p_j^- = p_j^- - 1$.

4- Aktualisiere $A^j_{K_1^-}(x)$, $A^j_{K_1^-}(x) = A(j, p_j^-)$, $K_1^- = K_1^- + 1$.

}// end $While - Schleife$ Zeile 2

}// end if Zeile 1

}//end Methode

Da Algorithmus 3 sich sehr ähnlich zu Algorithmus 2 verhält, beschreiben wir Algorithmus 3 an dieser Stelle nicht detaillierter. Erläutert werden ausschließlich die Unterschiede, die sich von Algorithmus 2 zu Algorithmus 3 finden:

1. In der Initialisierungsphase des Algorithmus 2 fügen wir den Punkt/die Punkte, der/die unter den $2(T(k \times d))$ Kandidatenpunkten den kleinsten Abstand zu x hat/haben, in die Menge W ein. In Algorithmus 3 hingegen fügen wir alle Kandidatenpunkte in Menge W ein und fügen erst in der Hauptphase in Schritt 2 alle Punkte $x_M \in W$, der/die den kleinsten Abstand zu x haben, in die Menge $minimal$ ein.
2. Die Hauptphase des Algorithmus 2 besteht aus einer While-Schleife und die Endphase schließt an die Hauptphase an:

   Hauptphase

   BEGIN

   While($gibtMinimum$){

   $\vdots \quad \vdots \quad \vdots$

   }

   Endphase

   ... ...

   END

   Allerdings besteht die Hauptphase des Algorithmus 3 aus zwei geschachtelten While-Schleifen und die Endphase beginnt am Ende der äußeren While-Schleife:

   Hauptphase

   BEGIN

   While($p \leq K$){

   // p-ter Nachbarschaftskandidat wird hier bestimmt

   $\vdots \quad \vdots \quad \vdots$

   While($gibtMinimum$){

   // Hier wird die Hauptphase des Algorithmus 2 aufgerufen, allerdings mit dem Unterschied, dass in Menge W alle Nachbarschaftskandidaten gespeichert werden und in der Menge $minimal$ der p-te Nachbarschaftskandidat gespeichert wird.

   $\vdots \quad \vdots \quad \vdots$

   }// end innere While-Schleife

   Endphase

   // Hier wird die p-te Nachbarschaftsmenge des Punktes x gleich der Menge $minimal$ gesetzt $N_p(x) = minimal$, $p = 1, 2, \ldots, K$.

   $\vdots \quad \vdots \quad \vdots$

   }// end äußere While-Schleife

   END

4.2. Beweis der Korrektheit des Algorithmus 3

In diesem Abschnitt beweisen wir die Korrektheit des Algorithmus 3.

Theorem 4 -

Algorithmus 3 findet die K-nächsten Nachbarn $N_p(x)$, $p = 1, 2, \ldots, K$ für einen beliebigen Punkt $x = (a_1, a_2, \ldots, a_d) \in X$ bei Eingabe von K und K_1.

Beweis:

Wir beweisen die Korrektheit des Algorithmus 3 durch vollständige Induktion nach dem Parameter K. Zu diesem Zweck zeigen wir, dass in jedem While-Schleifendurchlauf $p = 1, 2, \ldots, K$ die folgende Aussage richtig ist:

Die gesuchte p-te Nachbarschaftsmenge $N_p(x), p = 1, 2, \ldots, K$ für den Punkt x ist in Endphase gleich der Menge minimal.

Induktionsanfang:

Für den Induktionsanfang zeigen wir, dass die oben stehende Aussage für p gleich 1 und 2 korrekt ist.

Sei $p = 1$ und gesucht ist die Menge der ersten Nachbarn $N_1(x)$. Innerhalb des ersten While-Schleifendurchlaufs der Hauptphase des Algorithmus 3 wird in Schritt 2 die Menge minimal mit dem/den ersten Nachbarschaftskandidat/en $x_M \in W$, der/die unter allen Punkten der Menge W den kleinsten Abstand zu dem Punkt x hat/haben, initialisiert. Hier ist Menge minimal äquivalent zu der Menge W in der Initialisierungsphase des Algorithmus 2. Weiterhin wird die Hauptphase des Algorithmus 2 aufgerufen, allerdings mit dem Unterschied, dass anstelle der Menge W die Menge minimal verwendet wird. In Schritt 6 wird ein beliebiger Punkt $x_M \in$ minimal gewählt. In Schritt 7 prüfen wir, ob x_M in jeder Dimension $j = 1, 2, \ldots, d$ die Eigenschaft min_j hat. Wenn "Ja", dann ist x_M nach dem Teil a des Theorem 1 globales Minimum. Sonst werden in den Dimensionen $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$, in denen x_M die Eigenschaft $!min_{j_b}$, $b = 1, 2, \ldots, s$ hat, Intervalle gebildet. Dazu wird, anders als in den Methoden $!min_{k^+}(.)$ und $!min_{k^-}(.)$ des Algorithmus 2 innerhalb der if-Anweisung in Schritt 6 der Methoden $!MIN_{k_1^+}(.)$ und $!MIN_{k_1^-}(.)$ die Menge W und minimal mit dem/den neuen Nachbarschaftskandidat/en x^*, der/die einen kleineren/gleichen Abstand mit x im Vergleich x_M zu x hat/haben, vereinigt[2].

$\vdots \quad \vdots \quad \vdots$

6- $if(dist(x, x^*) \leq dist(x, x_M))\{$

7- $if(dist(x, x^*) == dist(x, x_M))\ \{$

8- $W = W \cup \{x^*\}$.

[2]Bei den Methoden $!min_{k^-}(.)$ und $!min_{k^+}(.)$ wird nur die Menge W mit dem/den neuen Nachbarschaftskandidat/en x^*, der/die einen kleineren/gleichen Abstand mit x im Vergleich x_M zu x hat/haben, vereinigt.

9- $minimal = minimal \cup \{x^*\}$

}// end if Zeile 7

10- if $(dist(x, x^*) < dist(x, x_M))\{$

11- $W = W \cup \{x^*\}$.

12- Setze minimal gleich leere Menge $minimal = \emptyset$.

13- Füge neuen p-ten Nachbarschaftskandidaten in Menge minimal ein $minimal = minimal \cup \{x*\}$.

14- Setze boolesche Variable $gibtMinimum$ gleich Wahr $gibMinimum = Wahr$.

}// end if Zeile 10

}// end if Zeile 6

$\vdots \quad \vdots \quad \vdots$

Hier merken wir uns, dass alle Nachbarschaftskandidaten[3] Elemente der Menge W sind und alle p-ten Nachbarschaftskandidaten $p = 1, 2, \ldots, K$ mit kleinstem Abstand zu x Element der Menge minimal sind.

Da die Menge minimal in Algorithmus 3 im ersten While-Schleifendurchlauf äquivalent zu Menge W in Algorithmus 2 ist und wir die Hauptphase des Algorithmus 2 innerhalb der Hauptphase des Algorithmus 3 aufrufen, sind alle Punkte $x_M \in minimal$ innerhalb des ersten While-Schleifendurchlaufs der Endphase nach dem Theorem 3 die ersten Nachbarn des Punktes x. Deswegen setzen wir in Schritt 20 der Endphase $N_1(x) = minimal$ und entfernen in Schritt 21 alle $x_M \in minimal$ von der Menge W. Damit haben alle verbleibende Punkte $x^* \in W$ einen größeren Abstand zu x in Vergleich $x_M \in minimal$ mit x. Dann setzen wir in Schritt 22 die Menge minimal gleich leere Menge $minimal = \emptyset$ und inkrementieren in Schritt 23 While-Schleifendurchlaufsvariable p um eins.

Sei $p = 2$ und wir sind im zweiten While-Schleifendurchlauf. In Schritt 2 initialisieren wir erneut die Menge minimal mit dem/den Punkt/Punkten, der/die den kleinsten Abstand zu dem Punkt x hat/haben. Da wir im ersten While-Schleifendurchlauf alle ersten Nachbarn des Punktes x von der Menge W entfernt haben, können nur die zweiten Nachbarschaftskandidaten in der Menge minimal in Schritt 2 vorhanden sein. In Schritt 6 wählen wir einen beliebigen Punkt $x_M \in minimal$. Dann in Schritt 7 prüfen wir, ob x_M in jeder Dimension $j = 1, 2, \ldots, d$ die Eigenschaft min_j hat. Wenn "Ja", dann ist x_M aus folgenden Gründen der zweite Nachbar des Punktes x:

1. *Nur alle Punkte $y \in N_1(x)$ haben eine Distanz $dist(y, x)$ kleiner als $dist(x_M, x)$ und alle $y \in N_1(x)$ haben die Eigenschaft $y.behandelt = true$.*
2. *Wir wissen schon nach unserer Wahl aller Punkte $y \in W$, $y \neq x_M$ gilt: $dist(x_M, x) < dist(y, x)$ und alle $y \in W$ haben die Eigenschaft $y.behandelt = true$.*

[3] Hier sind alle Nachbarschaftskandidaten innerhalb der Initialisierungs- und Hauptphase des Algorithmus 3 Element der Menge W.

3. *Alle verbleibenden Punkte* $r \in R := X - \{W \cup N_1(x)\}$ *haben in jeder Dimension* $j = 1, 2, \ldots, d$ *eine lokale Distanz* $dist_j(r, x)$*, die größer als* $|Min(A^j_{K_1}(x)) - a_j|$ *ist. Aus diesem Grund ergibt sich Folgendes:*
   $dist(x_M, x) = \kappa\left(\sum_{j=1}^{d} dist_j(x_M, x)\right) \leq \kappa\left(\sum_{j=1}^{d} |Min(A^j_{K_1}(x)) - a_j|\right) <$
   $\kappa\left(\sum_{j=1}^{d} dist_j(r, x)\right) = dist(r, x)$
   Alle Punkte r *haben die Eigenschaft* $r.behandelt = false$*.*

Ansonsten werden in den Dimensionen $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$*, in denen* x_M *die Eigenschaft* $!min_{j_b}$*,* $b = 1, 2, \ldots, s$ *hat, Intevalle gebildet. Innerhalb dieser Intervalle liegen Intervallpunkte* x^**, die die folgenden Eigenschaften besitzen:*

1. *Alle* x^* *haben eine lokale Distanz* $dist_{j_b}(x^*, x)$*, die kleiner oder gleich* $dist_{j_b}(x_M, x)$ *ist.*
2. *Alle* $x^* \notin W$ *haben die Eigenschaft* $x^*.behandelt = false$ *und haben eine Distanz* $dist(x^*, x)$*, die größer ist als* $dist(y, x)$*,* $y \in N_1(x)$*.*
3. *Alle* $x^* \in W$ *werden durch die Methoden* $!MIN_{K_1^+}()$ *und* $!MIN_{K_1^-}()$ *ignoriert und haben eine Distanz* $dist(x^*, x)$ *größer/gleich* $dist(x_M, x)$*.*

Hier haben wir zwei Möglichkeiten:

1. *Falls kein Intervallpunkt* $x^* : x^*.behandelt = false$ *gefunden wird, der eine kleinere Distanz zu* x *im Vergleich* x_M *zu* x *hat, dann ist* x_M *nach dem folgenden Grund der zweite Nachbar.*

   - *Ein Punkt* $r \in R := X - \{W \cup N_1(x)\}$ *kann eine kleinere Distanz* $dist(r, x)$ *zu* x *im Vergleich* x_M *mit* x *haben, falls* r *die Summe der lokalen Distanzen* $dist_j(r, x)$*,* $j = 1, 2, \ldots, d$ *minimiert. In Dimensionen, in denen* x_M *die Eigenschaft* min_j *hat, hat* r *eine lokale Distanz größer als* x_M*. Daher kann* r *nur in den Dimensionen* j_b*,* $b = 1, 2, \ldots, s$*, in denen* x_M *die Eigenschaft* $!min_{j_b}$ *hat, eine lokale Distanz kleiner/gleich* x_M *haben. Deswegen muss* r *zuerst einen Intervallpunkt sein, um überhaupt eine Distanz* $dist(r, x)$ *kleiner als* $dist(x_M, x)$ *haben zu können. Wenn sich kein Intervallpunkt findet, der eine kleinere Distanz zu* x *hat, dann ist* x_M *definitiv der zweite Nachbar von* x*.*

   Da die boolesche Variable gibtMinimum innerhalb der Methoden $!MIN_{k_1^+}(.)$ *und* $!MIN_{k_1^-}(.)$ *nicht gleich true gesetzt wird, wird die innere While-Schleife in Zeile 4 (Hauptphase des Algorithmus 2) abgebrochen. Weiterhin werden in der Endphase Schritt 20,21,22 und 23 durchgeführt. Im Anschluss wird mit dem nächsten While-Schleifendurchlauf begonnen.*
2. *Wir finden einen Intervallpunkt* x^**, der einen kleineren*[4] *Abstand zu* x *im Vergleich* x_M *zu* x *hat. Hier setzen wir die Menge minimal innerhalb der*

[4] x^* kann auch den gleichen Abstand zu x wie x_M mit x haben. In diesem Fall fügen wir x^* innerhalb der Methoden $!MIN_{K_1^+}(.)$ oder $!MIN_{K_1^-}(.)$ in Mengen *minimal* und W ein $minimal = minimal \cup \{x^*\}, W = W \cup \{x^*\}$.

Methoden $!MIN_{K_1^+}(.)$ oder $!MIN_{K_1^-}(.)$ gleich leere Menge minimal $= \emptyset$ und fügen einen neuen zweiten Nachbarschaftskandidaten x^* in Mengen minimal und W ein minimal $=$ minimal $\cup \{x^*\}$, $W = W \cup \{x^*\}$. Wir setzen auch hier die boolesche Variable gibtMinimum gleich Wahr und aktualisieren $A^j_{K_1^+}(x)$, p_j^+, $A^j_{K_1^-}(x)$ und p_j^- für den neuen zweiten Nachbarschaftskandidaten mit den Methoden $min_{K_1^+}(.)$ und $min_{K_1^-}(.)$. Dann fangen wir wieder mit einem neuen While-Schleifendurchlauf in Zeile 4 an, um zu prüfen, ob x^* ein lokales- oder globales Minimum ist.

Induktionsannahme:

Die oben stehende Aussage gilt für p gleich $K - 1$.

Induktionschritt:

Wir beweisen, falls die oben stehende Aussage für p gleich $K - 1$ gilt (Induktionsannahme), dann gilt die Aussage für p gleich K.

Beweis:

Mit gleichen Argumenten und Gründen wie beim Beweis für die zweite Nachbarschaft $p = 2$ gilt die oben stehende Aussage für p gleich K.

4.3. Empirische Untersuchung des Verhaltens des Algorithmus 3

In diesem Abschnitt untersuchen wir empirisch das Verhalten des Algorithmus 3. Unsere Untersuchung ähnelt sehr der Untersuchung der Verhaltens der Algorithmen 1 und 2. Für diesen Zweck haben wir Algorithmus 3 in mittels der Programmiersprache Java implementiert.

Empirische Untersuchung 2 *-Algorithmus 3*

Wir erzeugen die Attributswerte jedes Punktes $x = (a_1, a_2, \ldots, a_d) \in X$, $|X| = N$ zufällig und unabhängig voneinander. Weiterhin definieren wir folgende Größen:

Definition 19 *-$D_i(x)$*

Wir definieren $D_i(x)$ als:

$D_i(x)$:=Die Anzahl der berechneten Distanzen durch Algorithmen 3, um die Menge $N_i(x)$, $x \in X$, $i = 1, 2, \ldots, K$ zu bestimmen.

Definition 20 *-$D_s(x)$*

Wir definieren die Summe der berechneten Distanzen für die Bestimmung der K-nächsten Nachbarn eines Datensatzes x als:

$D_s(x) := \sum_{i=1}^{K} D_i(x)$

***Definition 21** -H*

Wir definieren die durchschnittlichen berechneten Distanzen durch Algorithmus 3 als:
$H := \frac{\sum_{x \in X} D_s(x)}{N}$

***Definition 22** -E*

Wir definieren die Effizienz des Algorithmus 3 als:
$E := \frac{N-H}{N}$

Wir charakterisieren hier genau wie in der ersten empirischen Untersuchung die Menge der Datensätze X mit zwei Merkmalen N (Anzahl der Datensätze) und d (Anzahl der Dimensionen eines Datensatzes). Weiterhin Untersuchen wir das Verhalten des Algorithmus 3 im Bezug auf K gleich 10 und die oben definierten Merkmale H und E unter folgenden Situationen:

1. *Wir setzen d gleich eine konstante Zahl d_c und lassen N linear wachsen. Hierfür setzen wir jedes Mal d_c gleich 2, 3, 4, 5 und lassen jeweils N von 100 bis 100000 wachsen. Wir bezeichnen die Gleichsetzung von d mit einer Konstante d_c und das Wachstum von N als eine Periode. Dann erstellen wir Tabelle 3 wie folgt:*

(N, d_c)	H	E
(100,2)	73.56	0.2644
(1000,2)	486.87	0.5131
(10000,2)	1832.14	0.8168
(100000,2)	5375.93	0.9642
(100,3)	93.68	0.0632
(1000,3)	869.36	0.1306
(10000,3)	7222.82	0.2777
(100000,3)	42546.92	0.5745
(100,4)	97.9	0.0210
(1000,4)	968.82	0.0312
(10000,4)	9308.99	0.0691
(100000,4)	82516.32	0.1748
(100,5)	98.76	0.0124
(1000,5)	992.54	0.0074 (*)
(10000,5)	9820.35	0.0180
(100000,5)	95105.26	0.0489

Tabelle 3: Die Outlier wurden mit (*) markiert. Aus Tabelle 3 können wir ersehen, dass bei gleichbleibender Anzahl der Dimensionen und dem Wachstum der Anzahl der Datensätze die Effizienz des Algorithmus 3 steigt. Hier sehen wir, dass für jede Perionde $d_c = 2, 3, 4, 5$ die Anzahl der berechneten Distanzen linear wächst.

H :=siehe Definition 21 Seite 72

E :=siehe Definition 22 Seite 72

Folgendes Phänomen können wir in der Tabelle 3 beobachten:

Schlussfolgerung 4 *-*

a- E steigt:

Wir sehen, dass für jede Periode d_c gleich 2, 3, 4 und 5 die Werte von E spaltenweise wachsen.

2. *Wir nutzen die Werte der Tabelle 3 und stellen die Tabelle 3 um. Hierfür setzen wir N gleich einer konstanten Zahl N_c und setzen jedes Mal N_c gleich 100, 1000, 10000, 100000 und lassen jeweils d von 2 bis 5 wachsen. Wir bezeichnen die Gleichsetzung von N mit einer konstanten N_c und dem Wachstum von d als eine Periode. Dann erstellen wir Tabelle 4 wie folgt:*

(N_c, d)	H	E
(100,2)	73.56	0.2644
(100,3)	93.68	0.0632
(100,4)	97.9	0.0210
(100,5)	98.76	0.0124
(1000,2)	486.87	0.5131
(1000,3)	869.36	0.1306
(1000,4)	968.82	0.0312
(1000,5)	992.54	0.0074
(10000,2)	1832.14	0.8168
(10000,3)	7222.82	0.2777
(10000,4)	9308.99	0.0691
(10000,5)	9820.35	0.0180
(100000,2)	5375.93	0.9642
(100000,3)	42546.92	0.5745
(100000,4)	82516.32	0.1748
(100000,5)	95105.26	0.0489

Tabelle 4: Aus Tabelle 4 können wir ersehen, dass bei gleichbleibender Anzahl der Datensätze und dem Wachstum der Anzahl der Dimensionen die Effizienz des Algorithmus 3 fällt. Hier sehen wir, dass für jede Perionde $N_c = 100, 1000, 10000, 100000$ die Anzahl der berechneten Distanzen linear wächst.

H :=siehe Definition 21 Seite 72

E :=siehe Definition 22 Seite 72

Folgendes Phänomen können wir bei der Betrachtung von Tabelle 4 feststellen:

Schlussfolgerung 5 *-*

a- E Fällt:

Wir sehen, dass für jede Periode N_c gleich 100, 1000, 10000 und 100000 die Werte von E spaltenweise fallen.

Aus unseren beobachteten Phänomenen der Tabellen 3 und 4 können wir folgende Schlussfolgerungen ziehen:

Schlussfolgerung 6 *-*

1. *Wegen Item a der Schlussfolgerung 4 wird Algorithmus 3 effizienter, je mehr die Anzahl der Datensätze N im Vergleich zur Anzahl der Dimensionen d steigt, also für $d \ll N$.*
2. *Wegen Item a der Schlussfolgerung 5 wird Algorithmus 3 bei gleichbleibender Anzahl der Datensätze N und Wachstum der Dimensionen d seine Effizienz verlieren.*

5. Cluster-Algorithmen

In diesem Kapitel beschäftigen wir uns mit der Entwicklung eines partitionierenden agglomerativen Bottom-up Cluster-Algorithmus (Algorithmus 4). Zuerst beweisen wir ein Theorem, auf welchem die Kernidee unseres Cluster-Algorithmus basiert. Dann beschreiben wir schrittweise die Grundideen des Algorithmus. Weiterhin erweitern wir Algorithmus 4 und entwickeln Algorithmus 5. Wir beweisen die Korrekheit der Algorithmen 4 und 5 und bestimmen die Laufzeit des Algorithmus 4. Weiterhin vergleichen wir den Algorithmus 4 mit den PAM und CLARA Algorithmen erst konzeptuell und dann empirisch.

5.1. Kernidee der Clusteralgorithmen

Im Folgenden beweisen wir einige Eigenschaften der ersten Nachbarschaftsbeziehung.

Theorem 5 *Die erste Nachbarschaftsbeziehung partitioniert die Menge* $X = (x_1, x_2, \ldots, x_N)$ *in* X_i, $i = 1, 2, \ldots k$, $k \geq 1$ *Teilmengen mit den folgenden Eigenschaften:*

1. *Jede Teilmenge* $X_i \neq \emptyset$, $i = 1, 2, \ldots, k$ *ist nicht leer und hat mindestens zwei Elemente. Wir bezeichnen die Teilmengen* X_i, $i = 1, 2, \ldots, k$ *als erste Nachbarschaftsfolge oder vereinfacht Einsfolge.*
2. *Für alle zwei Einsfolgen* X_i *und* X_j *gilt:* $X_i \cap X_j = \emptyset$, $i \neq j$, $i, j = 1, 2, \ldots k$ *und damit* $X = \bigcup_{b=1}^{k} X_b$.
3. *In jeder Einsfolge* X_i, $i = 1, 2, \ldots, k$ *gibt es mindestens zwei Punkte* x_s *und* x_t, *die eine symmetrische erste Nachbarschaftsbeziehung zueinander haben. Wir bezeichnen diese Punkte als Kernpunkte oder Repräsentanten der Einsfolge und die restlichen Punkte der Einsfolge als Normalpunkte.*

Beweis:

1. *Wie oben in Korollar 2 geschrieben, exsistiert auf jeden Fall für jeden Punkt* $x_s \in X$, $|X| \geq 2$ *mindestens ein Punkt* $x_t : n(x_s, 1) = x_t$. *Die beiden Punkte* x_s *und* x_t *bilden dann eine Menge* X_i, *die mindestens zwei Elemente hat.*
2. *Das folgt aus Teil 1 des Beweises und der Existenz der höheren Nachbarschaftsbeziehung zwischen Punkten. D.h. irgendwann ist die Bildung der ersten Nachbarschaftsbeziehung zwischen ein paar Punkten abgeschlossen. Diese Punkte bilden dann eine Einsfolge miteinander. Damit gibt es für die Punkte zweier Einsfolgen* X_i *und* X_j, *aufgrund der Räumlichkeitsbedingung (Verteilung*

der Punkte), keine erste Nachbarschaftsbeziehung mehr. Daher ist $X_i \cap X_j = \emptyset$ *und damit* $X = \bigcup_{b=1}^{k} X_b$.

3. *Seien* $x_n \to x_{n-1} \to \ldots \to x_3 \to x_2 \leftrightarrow x_1$ *eine Folge von ersten Nachbarschaftsbeziehungen (.* $\to$ *. steht für* $n(x_i, 1) = x_{i-1}$ *und .* $\leftrightarrow$ *. steht für symmetrische Nachbarschaftsbeziehung, d.h.* $x_2 \sim^1 x_1$. *Da in einer Sequenz der ersten Nachbarschaftsbeziehungen jeder Punkt* x_i, $i = 1, 2, \ldots, n$ *sich den zunächst am nahe liegensten Punkt als seinen ersten Nachbarn sucht und die Einsfolge für endlich viele Punkte endlich ist, haben wir die Distanzverhältnisse* $dist(x_n, x_{n-1}) > \ldots > dist(x_2, x_1)$. *Hier nimmt die Distanz zwischen* x_j *und* x_{j-1}, $j = n, \ldots, 2$ *ab, bis sie ihr Minimum zwischen den Punkten* x_2 *und* x_1 *erreicht. Deswegen haben wir eine symmetrische erste Nachbarbeziehung zwischen* x_2 *und* x_1.

   Wir sehen hier ganz eindeutig, dass jede Einsfolge die Tendenz hat, die Distanz zwischen ihren Mitgliedern zu minimieren und auch, dass sie eine Richtung hat, indem sie einen lokalen oder globalen Minimumsabstand im Eingaberaum zeigt.

5.2. Entwicklung der Clusteralgorithmen

Das Theorem 5 hat uns einige Eigenschaften der ersten Nachbarschaftsbeziehung dargestellt, welche wir benutzen können, um einen Algorithmus zur Lösung des Clustering-Problems zu entwickeln.

Wir können uns Folgendes vorstellen: Falls ein Cluster sich aus mehr als einer Einsfolge zusammensetzt, sind die Einsfolgen im Eingaberaum innerhalb eines Clusters näher zueinander gelegen als Einsfolgen in verschiedenen Clustern. Diese Vorstellung bringt uns insofern einen Schritt weiter, als dass, falls wir den Mittelpunkt der Punkte einer Einsfolge bilden und sie durch diesen Mittelpunkt im Eingaberaum ersetzen, diese Mittelpunkte innerhalb eines Clusters dann wieder Einsfolgen bilden. Falls wir diesen Schritt zur Bildung des Mittelpunkts iterativ fortsetzen, nimmt die Anzahl der Mittelpunkte mit jedem Schritt ab. Zudem nähern sich die Mittelpunkte innerhalb eines Clusters immer mehr dem Zentrum des Clusters. Dies ist darauf zurückzuführen, dass sich die Einsfolgen innerhalb des Clusters nebeneinander bzw. nah beieinander befinden. Das Ergebnis dieser iterativen Mittelpunktsbildung innerhalb eines Clusters ist ein Mittelpunkt, der je nach Form des Clusters zentriert ist und welchen wir als Zentrum des Clusters bezeichnen. Ein Zentrum z in unserem Cluster-Algorithmus hat einfache Eigenschaften :

- z ist nicht Element unserer Eingabemenge $X = \{x_1, x_2, \ldots, x_N\}$.

- Sei z das Zentrum des Clusters C, dann muss z nicht unbedingt den gleichen Abstand zu jedem Punkt $x \in C$ haben.

- Sei z das Zentrum des Clusters C und sei $z_j : z_j \neq z$ ein durch unseren Cluster-Algorithmus beliebig gefundenes Zentrum, dann gilt :

$\forall x \in C : dist(x, z) \leq dist(x, z_j)$

Diese Idee verwenden wir in unserem Cluster-Algorithmus.

Weiterhin verfeinern wir die Eigenschaften der Punkte einer Einsfolge. Wir haben in Theorem 5 bewiesen, dass es in jeder Einsfolge Kernpunkte gibt, die eine symmetrische erste Nachbarschaftsbeziehung zueinander haben und Normalpunkte, für welche keine symmetrische Nachbarschaftsbeziehung zu ihrem ersten Nachbarn vorliegt. D.h. wir haben folgende Situation:

$$\begin{cases} x\,ist\,ein\,Kernpunkt & falls\,N_1^s(x) \neq \emptyset \\ x\,ist\,ein\,Normalpunkt & sonst \end{cases}$$

Damit können wir die Normalpunkte von den Kernpunkten unterscheiden.

Basierend auf Theorem 5 und unserer obigen Überlegung schreiben wir einen partitionierenden agglomerativen Bottom-up Cluster-Algorithmus mit zwei Parametern K und α.

In unserem Cluster-Algorithmus verwenden wir folgende globale Parameter und Variablen:

Globale Parameter

- $K :=$ Anzahl der Cluster.
- $\alpha : \alpha \in (0, 1]$ eine untere Schranke, mit der wir die Auswahl der Kernpunkte steuern und beeinflussen.

Globale Variablen

- $L :=$ Wir speichern Kernpunkte in Array L.
- $P :=$ Wir speichern Mittelpunkte in Menge P.

1. $Mittelpunktanzahl :=$ Wir benutzen diese Variable innerhalb der While-Schleife, um die Anzahl der Mittelpunkte zu speichern.
2. $min :=$ Wir benutzen diese Variable, um das Minimum zu speichern.

Wir schreiben unseren Cluster-Algorithmus.

Algorithm 4 - *Cluster-Algorithmus*

Initialisierungsphase:

1- Bestimme globale Parameter $K, \alpha : K \in \mathbb{N}\,und\,\alpha \in (0, 1]$.

2- Bestimme für jeden Punkt $x \in X$ *die Menge des ersten Nachbarn* $N_1(x)$.

3- Füge alle Kernpunkte $x \in X : x \sim^1 y$ *und nicht seinen ersten symmetrischen Nachbarn* $y \in X$ *in Array* L *ein* [1]. *Setze* $min := \min_{x \in L} D(N_1^s(x))$.

4- Bilde jeweils von den Kernpunkten $x \in L$, *Mittelpunkt* x_m *zwischen Kernpunkt* x *und seinem symmetrischen Nachbarn* y *falls* $\frac{min}{D(N_1^s(x))} > \alpha$ *und füge* x_m *in der Menge* $P := \{x_m \mid x_m\,Mittelpunkt\,von\,x, y : x \sim^1 y\}$ *ein.*

5- Setze $Mittelpunktanzahl = |P|$.

[1] Sei $x \in X : x \sim^1 y$, dann wird o.B.d.A. nur x in L eingefügt und nicht sein erster symmetrischer Nachbar y.

6- Setze $L = Null$.

Hauptsphase:

- BEGIN

1- $While(Mittelpunktanzahl > K)\{$

2- Bestimme für jeden Punkt $x_m \in P$ *die Menge des ersten Nachbarn* $N_1(x_m)$.

3- Seien $T_1 := \{x_m \in P | \, x_m \, ist \, Normalpunkt\}$ *und*

$T_2 := \{x_m \in P \, | \, x_m \, ist \, Kernpunkt\}$, *wir definieren min als:*

$$min := \begin{cases} \min_{x_m \in T_1} D(\overrightarrow{N}_1(x_m)) & falls \, T_1 \neq \emptyset \\ \min_{x_m \in T_2} D(N_1^s(x_m)) & sonst \end{cases}$$

4- Füge jeden Kernpunkt $x_m \in P : D(N_1^s(x_m)) \leq min$ *und nicht seinen ersten symmetrischen Nachbarn* $y_m \in P$ *in Array* L *ein. Sortiere diese Kernpunkte aufsteigend nach* $D(N_1^s(x_m))$ *in* L *und entferne sie und ihren ersten symmetrischen Nachbarn aus der Menge* P[2].

5- for alle Kernpunkte $x_m \in L$ *tue Folgendes {*

6- $if(Mittelpunktanzahl \leq K)$ *{*

7- Vereinige P *mit* x_m *und seinem symmetrischen ersten Nachbarn* y_m, $P = P \cup \{x_m, y_m\}$.

8- Fahre mit nächsten For-Schleifendurchlauf fort.

}// end if

9- Setze Punkt z *als Mittelpunkt von* x_m *und seinen ersten symmetrischen Nachbarn* y_m *und vereinige* P *mit dem Mittelpunkt* z, $P = P \cup \{z\}$.

10- Dekrementiere Anzahl der Mittelpunkte um 1

$Mittelpunktanzahl = Mittelpunktanzahl - 1$.

}// end for

11- Setze $L = Null$.

}// end while

Endphase:

12- Ordne jeden Punkt $x \in X$ *dem am nah gelegensten Zentrum* $x_m \in P$ *zu.*

13- Bilde $|P|$ *Cluster und füge alle Punkte* $x \in X$, *die dem gleichen Zentrum* x_m *zugeordnet sind, in ein gemeinsames Cluster* C *ein.*

- END

Hier beschreiben wir unseren Cluster-Algorithmus detailierter. Wir haben unseren Algorithmus in drei Phasen eingeteilt.

Initialisierungsphase

[2] Als Beispiel falls $x_m, y_m \in P : x_m \sim^1 y_m$ fügen wir o.B.d.A. x_m in Array L ein und entfernen beide x_m und y_m aus der Menge P.

In Zeile 1 bestimmen wir die globalen Parameter K und α, deren richtige Auswahl einen direkten Einfluss auf die Korrektheit des Algorithmus hat. In Zeile 2 legen wir für jeden Punkt $x \in X$ die Menge des ersten Nachbarn fest. In Zeile 3 fügen wir alle Kernpunkte $x \in X : x \sim^1 y$ und nicht ihren ersten symmetrischen Nachbarn y in Array L ein. Außerdem setzen wir $min := \min_{x \in L} D(N_1^s(x))$. In Zeile 4 wählen wir die Kernpunkte $x \in L : \frac{min}{D(N_1^s(x))} > \alpha$ aus und bilden den Mittelpunkt x_m zwischen dem Kernpunkt x und seinem symmetrischen Nachbarn y. Dann fügen wir x_m in die Menge $P := \{x_m \mid x_m \, Mittelpunkt \, von \, x, y : x \sim^1 y\}$ ein. Schritt 4 ist unter vier Aspekten sehr bedeutsam.

1. Die Outliers bilden ebenso Einsfolgen, jedoch mit dem Unterschied, dass die Outlier-Kernpunkte im Vergleich zu den Normalkernpunkten meistens eine größere Distanz zueinander haben. Durch die richtige Auswahl der unteren Schranke α können wir verhindern, dass Outlier-Kernpunkte in Schritt 4 gewählt werden. Wenn die untere Schranke $\alpha : \alpha \in (0, 1]$ genügend scharf ist, dann wird, aufgrund der Tatsache, dass Outlier-Kernpunkte eine größere $D(N_1^s(x))$ im Vergleich zu den Normalkernpunkten haben, die Erfüllung der Bedingung $\frac{min}{D(N_1^s(x))} > \alpha$ für die Outlier-Kernpunkte schwieriger. Hier folgt, dass die gewählten Kernpunkte mit hoher Wahrscheinlichkeit Normalkernpunkte sind.
2. Wir reduzieren die Anzahl der Datensätze in der Initialisierungsphase zweimal. Zum Einen werden in Schritt 3 unter allen Punkten $x \in X$ nur Kernpunkte in Array L eingefügt. Zum Anderen werden in Schritt 4 Kernpunkte $x \in L$ gewählt, welche die Bedingung $\frac{min}{D(N_1^s(x))} > \alpha$ erfüllen.
3. In Schritt 4 bilden wir Mittelpunkte zwischen den ausgewählten Kernpunkten $x \in L : \frac{min}{D(N_1^s(x))} > \alpha$ und ihrem symmetrischen ersten Nachbarn y und fügen sie in die Menge P ein. Anschließend wird in der Hautphase des Algorithmus auf diese Mittelpunkte $x_m \in P$ operiert , nicht jedoch auf die Punkte $x \in X$. Die Menge P hat normalerweise wesentlich weniger Elemente als die Eingabemenge X, wodurch unser Algorithmus effizienter und schneller wird.
4. Die Struktur und die Form der Cluster wird durch die Mittelpunkte $x_m \in P$ weiterhin beibehalten.

In Zeile 5 wird $Mittelpunktanzahl$ gleich der Anzahl der Mittelpunkte in Menge P gesetzt. In Zeile 6 wird die Liste L gleich $Null$ gesetzt, um dieses Array innerhalb der While-Schleife erneut zu nutzen.

Hauptphase

Unser Hauptteil besteht aus einer While-Schleife, welche solange durchgeführt wird, bis die globale Bedingung ($Mittelpunktanzahl > K$) wahr ist. In Zeile 2 bestimmen wir für jeden Mittelpunkt $x_m \in P$ die Menge des ersten Nachbarn $N_1(x_m)$. In Zeile 3 bestimmen wir zuerst die Menge $T_1 := \{x_m \in P \mid x_m \, ist \, Norm\text{alpunkt}\}$. Falls T_1 nicht leer ist, setzen wir min gleich $min = \min_{x_m \in T_1} D(\overrightarrow{N}_1(x_m))$, sonst bestimmen wir die Menge $T_2 := \{x_m \in P \mid x_m \, ist \, Kernpunkt\}$ und setzen min gleich $min = \min_{x_m \in T_2} D(N_1^s(x_m))$, wobei dieses Minimum min in jedem

While-Schleifendurchlauf erneut berechnet wird. In Zeile 4 fügen wir jeden Kernpunkt $x_m \in P : D(N_1^s(x_m)) \leq min$ und nicht seinen ersten symmetrischen Nachbarn $y_m \in P$ in Array L ein. Im vierten Schritt sortieren wir diese Kernpunkte aufsteigend nach $D(N_1^s(x_m))$ in Array L und entfernen sie und ihren ersten symmetrischen Nachbarn [3] aus der Menge P. Dies hat zur Folge, dass die Normal- und Kernpunkte $x_m : D(N_1^s(x_m)) > min$ weiterhin in Menge P verbleiben. Desweiteren (Schritt 4) setzen wir in jedem While-Schleifendurchlauf eine dynamische und binäre Eigenschaft $B(.)$ für jeden Kernpunkt $x_m \in P$ wie folgt fest:

Definition 23 *-Binäre Eigenschaft B(.)*
Wir definieren die binäre Eigenschaft
$B(x_m) : D(N_1^s(x_m)) \mapsto \{positiv, negativ\}$*, die dynamisch ist, für jeden Kernpunkt* x_m *als:*

$$B(x_m) := \begin{cases} positiv & falls\, D(N_1^s(x_m)) \leq min \\ negativ & falls\, D(N_1^s(x_m)) > min \end{cases}$$

Die binäre Eigenschaft $B(.)$ ist dynamisch, weil min in jedem While-Schleifendurchlauf erneut berechnet wird. Es besteht die Möglichkeit, dass diese Eigenschaft für einen Kernpunkt im aktuellen While-Schleifendurchlauf negativ ist, im nächsten While-Schleifendurchlauf jedoch positiv wird.

Durch Definition dieser binären Eigenschaft legen wir eine dynamische Heuristik $H(.)$ für jeden Kernpunkt $x_m \in P$ im aktuellen While-Schleifendurchlauf fest. Wir gehen davon aus, dass falls $B(x_m) = positiv$, dann gehören x_m und sein erster symmetrischer Nachbar y_m zum gleichen Cluster C_i und falls $B(x_m) = negativ$, dann gehören x_m und y_m jeweils zu den verschiedenen Clustern C_i und C_j. Hier schreiben wir entsprechend eine Definition:

Definition 24 *-Dynamische Heuristik H(.)*
Wir definieren dynamische Heuristik $H(.)$ *für jeden Kernpunkt* x_m *und seinen symmetrischen Nachbarn* y_m *wie folgt:*

$$H(x_m) := \begin{cases} \{x_m, y_m\} \in C_i & falls\, B(x_m) = positiv \\ x_m \in C_i, y_m \in C_j & falls\, B(x_m) = negativ \end{cases}$$

Unsere Heuristik ist für einen Kernpunkt x_m dynamisch, weil die binäre Eigenschaft $B(.)$ für x_m dynamisch ist.

Die Heuristik $H(.)$ stellt zusammen mit unserer Sortierung aus Schritt 4 eine scharfe und gemischte Heuristik innerhalb der for-Schleife in Zeile 5 dar. Innerhalb der for-Schleife in Zeile 6 gibt es eine if-Anweisung mit der Bedingung $(Mittelpunktanzahl \leq K)$, welche eine globale Abbruchbedingung für die While-Schleife darstellt. Wenn diese if-Anweisung für einen Kernpunkt $x_m \in L$ wahr

[3] Wir entfernen alle Kernpunkte $x_m, y_m \in P : x_m \sim^1 y_m, x_m \in L, y_m \notin L$ aus der Menge P.

ist, dann setzen wir voraus, dass x_m und sein symmetrischer Nachbar y_m Zentren verschiedener Cluster sind. Deswegen bilden wir keinen Mittelpunkt mit ihnen und vereinigen sie in Zeile 7 mit der Menge P, $P = P \cup \{x_m, y_m\}$. Nach der Vereinigung setzen wir in Zeile 8 den nächsten for-Schleifendurchlauf fort. In Zeile 9 besteht der implizite Zustand $(Mittelpunktanzahl > K)$. In diesem Schritt ist die globale Abbruchbedingung $(Mittelpunktanzahl \leq K)$ noch nicht erreicht und die binäre Eigenschaft $B(x_m) = positiv$ bewirkt, dass unsere Heuristik $H(.)$ x_m und seinen symmetrischen Nachbarn y_m als Kernpunkte des gleichen Clusters erkennt. Der Mittelpunkt z der Kernpunkte x_m und y_m wird in Zeile 9 gebildet, worauf wir ihn mit der Menge P, $P = P \cup \{z\}$ vereinigen. In Zeile 10 aktualisieren wir die Anzahl der Mittelpunkte, indem wir sie um 1 dekrementieren $Mittelpunktanzahl = Mittelpunktanzahl - 1$. Aufgrund der Sortierung der Kernpunkte nach $D(N_1^s(x_m))$ in Array L in Zeile 4 wird der Dekrementierungsschritt in Zeile 10 im aktuellen for-Schleifendurchlauf immer durch den Kernpunkt x_m, der die minimale $D(N_1^s(x_m)) : D(N_1^s(x_m)) \leq min$ hat, durchgeführt. Ferner ist zu berücksichtigen, dass die globale Bedingung $(Mittelpunktanzahl > K)$ immer vor dem Dekrementierungsschritt gültig ist. Die scharfe und gemischte Heuristik ist eine gleichzeitige Verknüpfung der globalen Bedingung, unserer Heuristik $H(.)$ und der Sortierung. In Zeile 11 setzen wir $L = Null$, um sie erneut im nächsten While-Schleifendurchlauf zu nutzen.

Endphase

Wenn die While-Schleife abgebrochen wird, haben wir unsere gesuchten Clusterzentren in der Menge P. In der Endphase, d.h. in Schritt 12 und 13, ordnen wir jedem Datensatz $x \in X$ das zunächst nah gelegenste Zentrum in der Menge P zu und bilden für die Punkte $x \in X$, welche das gleiche Zentrum haben, ein Cluster.

Im Folgenden sei noch einmal kurz zusammengefasst, warum es sich bei unserem Algorithmus um einen partitionierenden agglomerativen Bottom-up Algorithmus handelt. Zunächst bildeten wir in der Initialisierungsphase die Menge P und setzten in Schritt 5 die $Mittelpunktanzahl = |P|$. Im Anschluss daran bestimmten wir in Schritt 2 die Menge $N_1(x_m)$ für jeden Punkt $x_m \in P$ innerhalb der While-Schleife, womit wir nach unserem Theorem 5 die Menge P in Einsfolgen partitionierten. Dies ist die partitionierende Eigenschaft unseres Algorithmus. Darüberhinaus bildeten wir in Schritt 9, in welchem die globale Bedingung $Mittelpunktanzahl > K$ gilt, den Mittelpunkt z zwischen dem Kernpunkt x_m und seinem symmetrischen ersten Nachbarn y_m und vereinigten diesen Mittelpunkt mit der Menge P, $P = P \cup \{z\}$. Dies ist der agglomerative Bottom-up[4] Schritt unseres Algorithmus.

Für manche Eingabemengen $X = \{x_1, x_2, \ldots, x_N\}$ ist es deutlicher und präziser, wenn bei der Zuordnung der Datensätze $x \in X$ in der Endphase nicht nur Zentren von Clustern, sondern auch zusätzlich andere zu den jeweiligen Clustern gehörende Hilfspunkte vorhanden sind. Deswegen beschäftigen wir uns mit einer neuen Überlegung. Wir wollen unseren Cluster-Algorithmus soweit modifizieren, dass wir

[4]Es handelt sich um Bottom-up, weil wir die Menge P solange mit einem Mittelpunkt z vereinigen, bis die globale Bedingung nicht mehr gilt.

Mittelpunkte, mit denen wir ein Zentrum bilden, ebenso im Cluster haben. D.h. in unserem modifizierten Cluster-Algorithmus bilden wir K-Cluster, indem jedes Cluster aus dem Zentrum z und denjenigen Mittelpunkten besteht, mit welchen wir das Zentrum z bilden. Dann ordnen wir in der Endphase jeden Datensatz $x \in X$ dem nah gelegensten Cluster $\grave{C}$ zu. Dazu definieren wir für jeden Datensatz $x \in X$ und jedes Cluster $\grave{C}_i$, $i = 1, \ldots, K$ das nah gelegenste Cluster $C(x)$ als:

$C(x) := \arg\min_{\grave{C}_i,\, i=1,\ldots,K} dist(x, \grave{C}_i)$

$dist(x, \grave{C}_i) := \min_{x_m \in \grave{C}_i} dist(x, x_m)$

Zuerst führen wir eine neue Variable für jeden Punkt x ein:

$x.P :=$ In dieser Variable speichern wir Mittelpunkte für den Punkt x.

Hier schreiben wir unseren modifizierten Cluster-Algorithmus, indem wir nur die zusätzlichen Zeilen, nicht jedoch die ähnlichen Schritte im Vergleich zu Algorithmus 4 schreiben.

Algorithm 5 - *modifizierter Cluster-Algorithmus*

Initialisierungsphase

1- ...

⋮ ⋮ ⋮

Hauptphase

BEGIN

1- While(Mittelpunktanzahl > K){

⋮ ⋮ ⋮

5- for alle Kernpunkte $x_m \in L$ tue Folgendes {

⋮ ⋮ ⋮

9- Setze Punkt z als Mittelpunkt von x_m und seinen ersten symmetrischen Nachbarn y_m.

10- $z.P := \begin{cases} \{x_m, y_m\} & falls\, x_m.P = \emptyset,\ y_m.P = \emptyset \\ x_m.P \cup \{y_m\} & falls\, x_m.P \neq \emptyset,\ y_m.P = \emptyset \\ \{x_m\} \cup y_m.P & falls\, x_m.P = \emptyset,\ y_m.P \neq \emptyset \\ x_m.P \cup y_m.P & falls\, x_m.P \neq \emptyset,\ y_m.P \neq \emptyset \end{cases}$

11- Vereinige P mit dem Mittelpunkt z, $P = P \cup \{z\}$.

12- Dekrementiere die Anzahl der Mittelpunkte um eins $Mittelpunktanzahl = Mittelpunktanzahl - 1$.

}// end for

13- Setze $L = Null$.

}// end while

Endphase:

14- Bilde $|P|$ Cluster $\grave{C}_i := \{z_i\} \cup z_i.P$, $z_i \in P$, $i = 1, 2, \ldots, K$, wobei z_i das i-te Zentrum der Menge P ist.

15- Ordne jedem Punkt $x \in X$ das Cluster $C(x) \in \{\grave{C}_1, \grave{C}_2, \ldots, \grave{C}_K\}$ zu.

16- Bilde Cluster C_i, $i = 1, 2, \ldots K$ und füge alle Punkte $x \in X$, die dem gleichen Cluster $C(x)$ zugeordnet sind, in ein gemeinsames Cluster C_i ein.

- END

Bei dem modifizierten Cluster-Algorithmus bleibt jede Zeile, bis auf die Zeile 10 in der Haupt- und Endphase, wie zuvor. In Schritt 9 bilden wir, wie bereits vorher, den Mittelpunkt z vom Kernpunkt x_m und seinem ersten symmetrischen Nachbarn y_m. In Zeile 10 speichern wir die Kernpunkte, welche in der Initialisierungsphase bestimmt worden sind, in Variable $z.P$ des Mittelpunktes z. In der Initialisierung- und Hauptphase bilden wir zwar einen Mittelpunkt, jedoch unterscheiden sich die Hauptphasen- und Initialisierungsphasenmittelpunkte dahingehend, dass die Variable $x_m.P$ eines Initialisierungsphasenmittelpunktes im Gegensatz zu einem Hauptphasenmittelpunkt leer[5] ist. Der Mittelpunkt z ist in der Hauptphase in Zeile 9 Mittelpunkt einer der folgenden Mittelpunktbildungskombinationen:

x_m	y_m
IM	IM
IM	HM
HM	IM
HM	HM

Tabelle 5: Mittelpunktsbildungsmöglickkeiten innerhalb der for-Schleife in der Hauptphase in Zeile 9.

$IM :=$ Initialisierungsphasenmittelpunkt

$HM :=$ Hauptphasenmittelpunkt

Weiterhin betrachten wir die Zeile 10 genauer:

$$z.P := \begin{cases} \{x_m, y_m\} & falls\, x_m.P = \emptyset,\ y_m.P = \emptyset \\ x_m.P \cup \{y_m\} & falls\, x_m.P \neq \emptyset,\ y_m.P = \emptyset \\ \{x_m\} \cup y_m.P & falls\, x_m.P = \emptyset,\ y_m.P \neq \emptyset \\ x_m.P \cup y_m.P & falls\, x_m.P \neq \emptyset,\ y_m.P \neq \emptyset \end{cases}$$

Wie wir hier sehen, besteht $z.P$ letztendlich nur aus den Initialisierungsphasenmittelpunkten[6], von denen die Hauptphasenmittelpunkte [7] gebildet werden.

In der Endphase enthält die Menge P hiermit die Zentren z_i, $i = 1, 2, \ldots, k$ und jede $z_i.P$ enthält, wie oben ausführlich beschrieben, die Initialisierungsphasenmittelpunkte.

[5] Weil wir erst in Hauptphase in Zeile 10 die Variable $z.P$ initialisieren.

[6] Weil, wenn x_m ein HM Mittelpunkt ist, enthält $z.P$ nicht x_m, sondern $x_m.P$ und jeder $x_m.P$ wird am Anfang durch Initialisierungsphasenmittelpunkte intialisiert.

[7] Wir bilden auch von diesen Hauptphasenmittelpunkten ein Zentrum.

5.3. Beweis der Korrektheit der Cluster-Algorithmen und Laufzeitanalyse des Algorithmus 4

Unserer Cluster-Algorithmus ist deterministisch und hat zwei Parameter K und α. Im Folgenden beweisen wir, dass unser Cluster-Algorithmus Zentren der K-Cluster unter bestimmten Annahmen finden kann.

Wir formulieren zuerst die Vorraussetzungen, die erfüllt sein müssen, damit unserer Algorithmus korrekt funktionieren kann. Diese Annahmen bezeichnen wir als erforderliche Bedingungen:

1. Die Auswahl der richtigen Prameter K und α ist essenziell. D.h. in Initialisierungsphase soll kein Mittelpunkt $x_m \in P$ aus Outlier-Kernpunkten gebildet worden sein. Zudem soll mindestens ein Mittelpunkt jedes Clusters in der Menge P vorhanden sein.
2. Die Lage der Cluster im Eingaberaum soll Folgendes gewährleisten:

   - Falls die globale While-Schleifenbedingung ($Mittelpunktanzahl > K$) innerhalb der for-Schleife (in Hauptsphase) für einen beliebigen Kernpunkt $x_m \in L : B(x_m) = positiv$ wahr ist, dann sollen x_m und sein symmetrischer Nachbar y_m zu dem gleichen Cluster gehören.

Unsere erforderlichen Bedingungen sind starke Vorraussetzungen, mit deren Hilfe wir die Korrektheit unseres Cluster-Algorithmus beweisen können.

Hier beweisen wir die Korrektheit unseres Algorithmus:

Theorem 6 *Algorithmus 4 findet Zentren der K-Cluster für die Eingabemenge $X = \{x_1, x_2, \ldots, x_N\}$, falls die erforderlichen Bedingungen erfüllt sind.*

Beweis:

In unserem Algorithmus wollen wir durch Mittelpunktsbildung von den aus den Datensätzen $x \in X$ erzeugten künstlichen Punkten $x_m \in P$ Zentren der Cluster finden. In der Endphase des Algorithmus ordnen wir jedem $x \in X$ das zunächst nah gelegenste Zentrum $x_m \in P$ zu und fügen alle Punkte $x \in X$ mit dem gleichen Zentrum in ein gemeinsames Cluster ein.

Nachdem wir in der Initialisierungsphase die Menge P bestimmt haben, legen wir im zweiten Schritt in der Hauptphase innerhalb der While-Schleife für jeden Mittelpunkt $x_m \in P$ seine erste Nachbarschaftsmenge $N_1(x_m)$ fest. Folglich partitionieren wir laut Theorem 5 die Menge P in Einsfolgen und haben für einen beliebigen Punkt $x_m \in P$ zwei Fälle:

1. *x_m ist ein Kernpunkt einer Einsfolge. Hier haben wir entsprechend zwei Möglichkeiten.*

   a- Die binäre Eigenschaft $B(x_m)$ ist in den aktuellen While-Schleifendurchlauf positiv. Weiterhin haben wir hier zwei Optionen:

a- Innerhalb der for-Schleife gilt die globale While-Schleifenbedingung* $(Mittelpunktanzahl > K)$ *für* x_m*. Folglich bilden wir in Schritt 9 den Mittelpunkt z zwischen* x_m *und seinem ersten symmetrischen Nachbarn* y_m *und vereinigen ihn mit der Menge P,* $P = P \cup \{z\}$*.*

Hier garantieren die erste und zweite Vorraussetzung der erforderlichen Bedingungen, dass unsere Heuristik $H(.)$ *korrekt funktioniert und der Mittelpunkt z weiter innerhalb des Clusters bleibt. Er bewegt sich damit mehr zum Zentrum des Clusters hin und entfernt sich zunehmend von den Mittelpunkten der anderen Cluster. Wir können nicht unterscheiden, ob der Punkt z das gesuchte Zentrum ist oder nicht. Als Normalpunkt oder Kernpunkt bildet er weiterhin im nächsten While-Schleifendurchlauf*[8] *mit den anderen Mittelpunkten eine Einsfolge. Allerdings haben sowohl die Mitglieder der neuen Einsfolge, als auch z eine größere Distanz zu ihren ersten Nachbarn. D.h. in jedem While-Schleifendurchlauf wächst die Distanz zwischen den Mittelpunkten und ihren ersten Nachbarn allmählich an. Wir können uns diese Erscheinung wie folgt erklären: Nach der Bestimmung der Menge P hat diese in der Initialisierungsphase* $|P| = Mittelpun-$

ktanzahl Elemente. Solange die globale Bedingung $(Mittelpunkt-$

$anzahl > K)$ *wahr ist, wird innerhalb der for-Schleife der Mittelpunkt der Kernpunkte gebildet. Dadurch nimmt die Anzahl der Mittelpunkte in Menge P ab, womit die Distanz zwischen den Mittelpunkten und ihren ersten Nachbarn größer wird. Dieses Phänomen bezeichnen wir als Wachstum der Distanz der ersten*

Nachbarschaft und nutzen es dynamisch bei der Feststellung des Minimuns min in Schritt 3 der Hauptphase. Wegen dieses Phänomens wächst das Minimum min auch entsprechend langsam in jedem While-Schleifendurchlauf an. In Schritt 10 aktualisieren wir die Mittelpunktanzahl und dekrementieren sie innerhalb der for-Schleife um 1 $Mittelpunktanzahl = Mittelpunktanzahl - 1$*.*

b- Innerhalb der for-Schleife gilt die globale Abbruchbedingung*

$(Mittelpunktanzahl \leq K)$ *in Zeile 6 bei der if-Anweisung für* x_m*. Die Variable Mittelpunktanzahl simuliert die gesamte Anzahl der Mittelpunkte in jedem While-Schleifendurchlauf dynamisch und ist noch vor Schritt 4 in der Hauptphase gleich* $Mittel-$

$punktanzahl = |P|$*. Da wir in Schritt 4 alle Kernpunkte* $x_m : B(x_m) = positiv$ *und nicht ihren ersten symmetrischen Nachbarn in Array L einfügen und diese und ihre erste symmetrische Nachbarn anschließend aus der Menge P entfernen, ist die Mittelpunktanzahl nach der Schritt 4 und vor Eintreten der for-Schleife gleich*

[8] Falls die globale Bedingung (Mittelpunktanzahl>K) immernoch gültig ist.

Mittelpunktanzahl $= (2 \times |L|) + |P|$. *Innerhalb der for-Schleife in Zeile 9 bilden wir den Mittelpunkt z aus zwei in Schritt 4 von der Menge P entfernten Kernpunkten* $x_m^*, y_m^* : x_m^* \sim^1 y_m^*, x_m^* \in L$, $y_m^* \notin L$ *und vereinigen ihn mit der Menge* $P = P \cup \{z\}$. *Wir dekrementieren ebenso die Mittelpunktanzahl in Zeile 10 um 1. Deswegen ist die Mittelpunktanzahl, wenn die Bedingung der if-Anweisung in Zeile 6 im i-ten for-Schleifendurchlauf* $1 < i \leq |L|$ *für Kernpunkt* x_m *wahr ist, gleich Mittelpunktanzahl* $= K = (2 \times (|L| - j)) + (|P| + j)$, $j = i - 1$. *Daraus folgt, dass sich die restlichen gesuchten Zentren in Array L befinden und wir trotz der Posivivität der binären Eigenschaft von* x_m x_m *und seinen symmetrischen Nachbarn* y_m *ohne Mittelpunktsbildung direkt in die Menge P einfügen müssen. Hiermit wird insgesamt garantiert, dass unserer Algorithmus genau K Zentren findet.*

b- *Die binäre Eigenschaft* $B(x_m)$ *ist im aktuellen While-Schleifendurchlauf negativ. Nach unserer Definition ist* $B(x_m)$ *negativ, falls* $D(N_1^s(x_m)) > min$ *wahr ist. Laut unserer Heuristik* $H(.)$ *gehen wir davon aus, dass* x_m *und sein symmetrisch erster Nachbar* y_m *zu den verschiedenen Clustern gehören. Deswegen bilden wir keinen Mittelpunkt von ihnen und belassen sie unverändert in Menge P. In Möglichkeit a haben wir das Phänomen Wachstum der Distanz der ersten Nachbarschaft eingeführt und seine Wirkung auf das Wachstum des Minimums min erklärt. Da unser Minimum min in jedem While-Schleifendurchlauf wächst, haben wir zwei Zustände für* $x_m : D(N_1^s(x_m)) > min$. *Wir nummerieren o.B.d.A. die While-Schleifendurchläufe von t bis* t_{end} *durch* $(t, \ldots, t_{end})$, *so dass t der aktuelle- und* t_{end} *der letzte While-Schleifendurchlauf ist.*

erster Zustand:

Im $t + s$*-ten While-Schleifendurchlauf mit* $1 \leq s \leq t_{end} - t$ *wird* $D(N_1^s(x_m)) \leq min$. *D.h.* $D(N_1^s(x_m)) > min$ *bleibt von dem aktuellen t-ten- bis zum* $t + s - 1-$*ten While-Schleifendurchlauf unverändert. Zudem verbleibt* x_m *zusammen mit seinem symmetrischen Nachbarn weiterhin unverändert in Menge P. Innerhalb dieser s While-Schleifen-durchläufe wird die Mittelpunktsbildung für die Kernpunkte, deren binäre Eigenschaft positiv ist, durchgeführt. Diese Kernpunkte haben eine kleinere Distanz zu ihren ersten symmetrischen Nachbarn, womit sie dichter beieinander liegen. Innerhalb dieser s While-Schleifendurch-läufe nimmt die Anzahl der Mittelpunkte in Menge P durch die Mittelpunktsbildung ab, womit die Distanz der ersten Nachbarschaft größer wird. Dies hat zur Folge, dass* $D(N_1^s(x_m)) \leq min$ *für unseren Kernpunkt* x_m *erst im* $t + s$*-ten While-Schleifendurchlauf wahr wird. Damit ist* $B(x_m) = positiv$, *so dass Fall 1 Möglichkeit a eintritt. Aufgrund der oben beschriebenen Tatsachen ist unser Algorithmus fähig, Zentren der heterogen besiedelten Cluster zu finden. Die Mittelpunktsbildung wird erst innerhalb dicht besiedelter Cluster vorgenommen, wodurch diese Cluster auch allmählich dünner besiedelt sind. Als Folge wird die Mittelpunktsbildung*

schließlich innerhalb der ursprünglich dünn besiedelten Cluster durchgeführt.

zweiter Zustand:

Die binäre Eigenschaft $B(x_m)$ bleibt vom aktuellen t-ten While-Schlei-fendurchlauf bis zum letzten t_{end}-ten While-Schleifendurchlauf negativ. Damit sind x_m und sein erster symmetrischer Nachbar y_m bis zum Schluss unverändert in der Menge P enthalten. Die beiden Mittelpunkte x_m und $y_m$sind nach unserer Heuristik $H(.)$ Zentren verschiedener Cluster.

2. *x_m ist ein Normalpunkt einer Einsfolge. Der Mittelpunkt wird in unserem Algorithmus nur aus den Kernpunkten gebildet, weil die erste Nachbarschaftsbeziehung für die Normalpunkte einseitig ist. D.h. sei $y_m \in \overrightarrow{N}_1(x_m)$ der erste Nachbar für unseren Normalpunkt x_m. Hier haben wir die Beziehung $n(x_m, 1) = y_m$ und $n(y_m, 1) = x_m^* \neq x_m$. Wir nummerieren wieder wie Fall eins Teil b While-Schleifendurchläufe von dem aktuellen t-ten bis zu letztem $t_{end}-$ten While-Schleifendurchlauf durch $(t, \ldots, t_{end})$. Wir haben zwei Zustände:*

   erster Zustand:

   In den $t + s$-ten While-Schleifendurchlauf für $1 \leq s \leq t_{end} - t$ wandelt sich x_m von dem Normalpunkt zu dem Kernpunkt. Er bleibt für s While-Schleifendurchläufe als Normalpunkt und innerhalb diese s While-Schleifendurchläufe wächst die Distanz der ersten Nachbarschaft. Dadurch ändert sich seine einseitige- zu doppelseitige erste Nachbarschaftsbeziehung in den $t + s$-ten While-Schleifendurchlauf und er wandelt sich von dem Normalpunkt zu dem Kernpunkt. Hier haben wir dann den ersten Fall für den Kernpunkt x_m.

   zweiter Zustand:

   x_m bleibt unverändert von dem aktuellen t-ten- bis zu dem letzten t_{end}-ten While-Schleifendurchlauf als Normalpunkt. Dadurch bleibt er in Menge P erhalten und ist eigentlich Zentrum eines Cluster.

Da die Mittelpunktsbildung für jeden Kernpunkt $x_m \in P$ in der Hauptphase nur innerhalb Fall eins/Möglichkeit a/Option a durchgeführt wird und die Voraussetzungen innerhalb Fall eins/Möglichkeit a/Option a* sich nicht mit unseren erforderlichen Bedingungen widersprechen, folgt daraus, dass K gesuchte Zentren sich in Menge P befinden. In der Endphase wird jeder Punkt $x \in X$ dem zunächst nächsten Zentrum $x_m \in P$ zugeordnet und es werden alle Punkte $x \in X$, die das gleiche Zentrum haben, in ein gemeinsames Cluster eingefügt.*

Im Folgenden beweisen wir die Korrektheit unseres modifizierten Cluster-Algorithmus:

Theorem 7 *Der modifizierte Cluster-Algorithmus findet K Cluster $\grave{C}_1, \ldots, \grave{C}_K$ für die Eingabemenge $X = \{x_1, x_{2,}, \ldots, x_N\}$, falls die erforderlichen Bedingungen erfüllt sind.*

Beweis:

Da der modifizierte Cluster-Algorithmus sehr ähnlich zu Algorithmus 4 ist, läuft unserer Beweis, ähnlich wie in Theorem 6 beschrieben, ab. In Theorem 6 haben wir

bewiesen, dass der Algorithmus 4 Zentren der K-Cluster findet. Unser modifizierter Cluster-Algorithmus tut dies ebenso und findet gleichfalls Zentren. Deswegen steht noch aus, zu beweisen, dass unser modifizierter Cluster-Algorithmus zusätzlich zu den Zentren eventuell noch andere Punkte für jedes Cluster findet.

Wir haben bei den erforderlichen Bedingungen verlangt, dass in der Initialisierungsphase in Schritt 4 für jedes Cluster in der Menge P mindestens ein Mittelpunkt x_m vorhanden sein muss. Wenn es in der Initialisierungsphase nur einen Mittelpunkt für ein Cluster in Menge P gibt, dann finden der modifizierte Cluster-Algorithmus und Algorithmus 4 diesen Mittelpunkt als Zentrum. Ist in der Initialisierungsphase jedoch mehr als ein Mittelpunkt für ein Cluster in Menge P vorhanden, dann bilden diese Mittelpunkte eine oder mehrere Einsfolgen. Nach unserer Heuristik $H(.)$ bilden wir hier genau wie innerhalb der for-Schleife in Zeile 9 in der Hauptphase des Algorithmus 4 den Mittelpunkt z vom Kernpunkt x_m und seinem ersten symmetrischen Nachbarn y_m, falls $B(x_m) = positiv$ und $(Mittelpunktanzahl > K)$ Wahr sind. Dies widerspricht nicht den erforderlichen Bedingungen. D.h. hier müssen x_m und y_m zum gleichen Cluster gehören[9], was unabhängig davon ist, ob x_m und y_m Initialisierungsphasenmittelpunkt oder Hauptphasenmittelpunkt sind. Wir haben oben ausführlich beschrieben, dass Initialisierungsphasenmittelpunkte in der Hauptphase in Zeile 10 nur in Variable z.P gespeichert werden. Die Initialisierungsmittelpunkte gehören nach unserer Heuristik $H(.)$ und den erforderlichen Bedingungen zum gleichen Cluster. "Die Eigenschaft im gleichen Cluster zu sein" wird von den Initialisierungsphasenmittelpunkten geerbt und weitergeleitet. Als Beispiel sei z_1 in Schritt 9 der Mittelpunkt der Initialisierungsphasenmittelpunkte x_{m_1} und y_{m_1} und sei z_2 in Schritt 9 der Mittelpunkt der Initialisierungsphasenmittelpunkte x_{m_2} und y_{m_2}, dann ergeben sich folgende Verhältnisse:

$z_1.P = \{x_{m_1}, y_{m_1}\} \Rightarrow z_1, x_{m_1}$ *und* y_{m_1} *befinden sich im gleichen Cluster.*

$z_2.P = \{x_{m_2}, y_{m_2}\} \Rightarrow z_2, x_{m_2}$ *und* y_{m_2} *befinden sich im gleichen Cluster.*

Hier o.B.d.A., falls z_3 der Mittelpunkt der Hauptphasenmittelpunkte z_1 und z_2 in Schritt 9 ist, folgt daraus, dass z_3, z_1 und z_2 im gleichen Cluster sind.

Wir haben außerdem :

$z_3.P = z_1.P \cup z_2.P = \{x_{m_1}, y_{m_1}, x_{m_2}, y_{m_2}\}$

Hieraus folgt:

z_3, x_{m_1}, y_{m_1}, x_{m_2} *und* y_{m_2} *befinden sich alle im gleichen Cluster.*

Deswegen, wenn wir in der Endphase in Schritt 14 die Cluster $\grave{C}_i := \{z_i\} \cup z_i.P$, $z_i \in P$, $i = 1, 2, \ldots, K$ bilden, gehören das Zentrum z_i und alle in Menge $z_i.P$ enthaltenen Initialisierungsphasenmittelpunkte zum gleichen Cluster.

Hiermit findet unser modifizierter Cluster-Algorithmus K Cluster $\grave{C}_1, \ldots, \grave{C}_K$.

Nachdem wir die Korrektheit der Algorithmen 4 und 5 bewiesen haben, versuchen wir, die Laufzeit des Algorithmus 4 zu analysieren. Wir konzentrieren uns auf Algorithmus

[9] Damit bleibt der Mittelpunkt z natürlich auch innerhalb des gleichen Clusters.

4, weil Algorithmus 4 und 5 einander sehr ähnlich sind. Bei der Zeitanalyse gehen wir davon aus, dass die Mengen X, P und L als Array vorliegen.

Laufzeitanalyse 2 *-Algorithmus 4*

Wir haben Algorithmus 4 in drei Phasen eingeteilt und analysieren die Laufzeit jeder Phase getrennt.

Initialisierungsphase

Schritt 1 ist eine einfache Zuweisung, die wir in konstanter Zeit ausführen können. In Schritt 2 bestimmen wir die Menge der ersten Nachbarn $N_1(x)$. *Sei im Folgenden* $E(x) : x \in X$ *der Aufwand für die Bestimmung der Menge der ersten Nachbarn* $N_1(x)$ *und sei* β *definiert als:*

$\beta := \max_{x \in X} E(x)$.

Dann können wir in Schritt 2 für alle Punkte $x \in X$ *die Menge der ersten Nachbarn* $N_1(x)$ *in* $O(N \times \beta)$ *bestimmen. In Schritt 3 fügen wir alle Kernpunkte* $x \in X$, $N_1^s(x) \neq \emptyset$ *und nicht ihre symmetrischen ersten Nachbarn in Array L ein und bestimmen* $min = \min_{x \in L} D(N_1^s(x))$. *Wir können dies in* $O(N)$ *tun. Der Aufwand für die Mittelpunktsbildung*[10] *der Kernpunkte* $x \in L : \frac{min}{D(N_1^s(x))} > \alpha$ *und die Einfügung in Menge P in Schritt 4 ist gleich* $O(|L| \times d) + O(|L|) = O(|L| \times d)$. *Schritt 5 und 6 sind zwei einfache Zuweisungen, die in konstanter Zeit* $O(1)$ *ausgeführt werden koennen.*

Der Zeitaufwand der Initialisierungsphase I_I *ist gleich:*

$I_I = O(N \times \beta) + O(N) + O(|L| \times d) =$

$\max(O(N \times \beta), O(N), O(|L| \times d)) = O(\max(N \times \beta, |L| \times d)) = O(N \times \beta)$, *hier gilt* $|L| < N$ *und* $d < \beta$.

Damit wird der Hauptaufwand der Initialisierungsphase durch die erste Nachbarschaftsberechnung verursacht.

Hauptphase

Die Hauptphase besteht aus einer While-Schleife mit der globalen Bedingung $While(Mittelpunktanzahl > K)$. *Wir setzen in Schritt 5 der Initialisierungsphase* $Mittelpunktanzahl = |P|$. *Bei der Analyse der Hauptphase stellen wir uns zwei Modelle vor:*

1. *Bei diesem Modell gehen wir davon aus, dass die Anzahl der maximalen While-Schleifendurchläufe gegeben ist. Solange die globale Bedingung gültig ist, gibt es mindestens einen Kernpunkt* $x_m \in L$, *der die binäre Eigenschaft* $B(x_m)$ *gleich positiv hat. Deswegen wird innerhalb der For-Schleife in Schritt 9 der Mittelpunkt z zwischen dem Kernpunkt* x_m *und seinem symmetrischen Nachbarn gebildet und mit der Menge P vereinigt* $P = P \cup \{z\}$. *Da der Kernpunkt* x_m *und sein symmetrischer Nachbar* y_m *in Schritt 4 aus der Menge P entfernt wurden und ihr Mittelpunkt z in Schritt 9 mit der Menge P vereinigt wird, wird die*

[10] Die Mittelpunktsbildung ist abhängig von der Anzahl der Dimensionen d. Wenn wir den Aufwand der Addition der beiden Zahlen und ihre Division durch 2 als konstant betrachten, dann können wir den Aufwand der Mittelpunktsbildung gleich d setzen.

Anzahl der Mittelpunkte in Menge P in jeder While-Schleife mindestens um eins verringert. Deswegen haben wir höchstens $Mittelpunktanzahl - K = t$ While-Schleifendurchläufe. In Schritt 2 bestimmen wir für jeden Mittelpunkt $x_m \in P$ die Menge der ersten Nachbarn $N_1(x_m)$. Seien P_i, β_i und β im Folgenden definiert als:

$P_i := \{P \quad Menge\, P\, in\, i - ten\, While - Schleifendurchlauf,\, i = 1, \ldots, t$, *wobei P_1 gleich der Menge P in der Initialisierungsphase ist.*

$\beta_i := \max_{x_m \in P_i} E(x_m),\, i = 1, \ldots, t$

$\beta := \max_{i=1,\ldots,t} \beta_i$

Damit ist der Aufwand der Bestimmung der ersten Nachbarschaft für die Mittelpunkte $x_m \in P_i$ höchstens gleich $O(\beta_i \times |P_i|)$. Deswegen ist der gesamte Aufwand für Schritt 2 innerhalb von t While-Schleifendurchläufen höchstens gleich $O(t \times \beta \times |P_1|)$. Der gesamte Aufwand jeweils für Schritt 3 und 4 innerhalb von t Schleifendurchläufen ist gleich $O(t \times |P_1|)$. Da wir in unserer Analyse von der maximalen Anzahl der While-Schleifendurchläufe ausgegangen sind, kann in jedem Schleifendurchlauf nur ein Kernpunkt x_m existieren, der die binäre Eigenschaft $B(x_m)$ gleich positiv hat. Deswegen ist $|L|$ in jedem While-Schleifendurchlauf gleich eins. Damit ist der gesamte Aufwand[11] *von Schritt 9 innerhalb von t While-Schleifendurchläufen gleich $O(t \times d) + O(t \times |P_1|) = O(t \times \max(d, |P_1|)) = O(t \times d)$, unter der Annahme, dass Datensätze mit einer großen Anzahl an Dimensionen vorliegen.*

Der Aufwand in der Hauptphase mit den maximal t Schleifendurchläufen ist gleich:

$O(t \times \beta \times |P_1|) + O(t \times |P_1|) + O(t \times d) = O(t \times \max(\beta \times |P_1|, d)) = O(t \times \beta \times |P_1|)$, *wobei $\beta > d$ ist.*

Hier gilt auch $|P_1| > |P_2| > \ldots > |P_t|$.

2. *Bei diesem Modell gehen wir davon aus, dass wir s While-Schleifendurchläufe haben, wobei $1 < s < t$ ist und t für die maximale Anzahl der Schleifendurchläufe steht. Da wir nicht von der maximalen Anzahl der Durchläufe ausgegangen sind, könnte in jedem Durchlauf mehr als ein Kernpunkt x_m existieren, der die binäre Eigenschaft $B(x_m)$ gleich positiv hat. Deswegen musste $|L|$ nicht in jedem Schleifendurchlauf gleich eins sein. Seien im Folgenden P_i, L_i, β_i, β und γ definiert als:*

   $P_i := \{P \quad Menge\, P\, in\, i - ten\, While - Schleifendurchlauf,\, i = 1, \ldots, s$, *wobei P_1 gleich der Menge P in der Initialisierungsphase ist.*

   $\beta_i := \max_{x_m \in P_i} E(x_m),\, i = 1, \ldots, s$

   $\beta := \max_{i=1,\ldots,s} \beta_i$

   $L_i := \{L \quad Array\, L\, in\, i - ten\, While - Schleifendurchlauf,\, i = 1, \ldots, s$

[11] Wir betrachten für die Laufzeitanalyse der For-Schleife nur Schritt 9, weil alle anderen Schritte geringeren Aufwand haben.

$\gamma := \max_{i=1,2,\ldots,s} |L_i|$

Damit ist der Aufwand der Bestimmung der ersten Nachbarschaft für die Mittelpunkte $x_m \in P_i$ *höchstens gleich* $O(\beta_i \times |P_i|)$. *Deswegen ist der gesamte Aufwand für Schritt 2 innerhalb von* s *While-Schleifendurchläufen höchstens gleich* $O(s \times \beta \times |P_1|)$. *Der gesamte Aufwand jeweils für die Schritte 3 und 4 innerhalb von* s *While-Schleifendurchläufen ist gleich* $O(s \times |P_1|)$ *(Schritt 3) und* $O(s \times |P_1|) + O(s \times (\gamma \log \gamma)) = O(s \times \max(|P_1|, \gamma \log \gamma))$ *(Schritt 4), wobei* $\gamma \log \gamma$ *der Sortieraufwand der Arrays* L_i *ist.*

Der gesamte Aufwand von Schritt 9 innerhalb von s *Durchläufen ist gleich* $O(s \times \gamma \times d) + O(s \times |P_1|) = O(s \times \max(\gamma \times d, |P_1|)) = O(s \times \gamma \times d)$, *unter der Annahme, dass Datensätze mit einer großen Anzahl an Dimensionen vorliegen.*

Damit ist der Aufwand der Hauptphase gleich:

$O(s \times \beta \times |P_1|) + O(s \times \gamma \times d) + O(s \times (\gamma \log \gamma)) =$

$O(\max(s \times \beta \times |P_1|, s \times \gamma \times d), s \times (\gamma \log \gamma)) =$

$O(s \times \max(\beta \times |P_1|, \gamma \times d, \gamma \log \gamma)) = O(s \times \beta \times |P_1|)$, *weil* $\gamma \ll |P_1|$, $\gamma \ll d$ *und* $d < \beta$ *sind.*

Hier gilt auch $|P_1| > |P_2| > \ldots > |P_s|$.

Der Zeitaufwand in der Hauptphase in Modell 1 und 2 wird durch die Berechnung der ersten Nachbarschaft dominiert. Da $t > s$ *ist, ist der Zeitaufwand der Hauptphase* I_H *im schlimmsten Fall gleich* $I_H = O(t \times \beta \times |P_1|)$.

Endphase

In Schritt 12 ordnen wir jeden Punkt $x \in X$ *dem nahegelegensten Zentrum* $x_m \in P$, $|P| = K$ *zu. Hierfür müssen wir* K *mal die Distanz zwischen* x *und* $x_M \in P$ *berechnen und deren Minimum bestimmen. Damit ist der gesamte Aufwand von Schritt 12 gleich* $O(N \times K \times A(.)) + O(N \times K) = O(N \times K \times A(.))$, *wobei* $A(.)$ *der sequentielle Aufwand der Distanzberechnung ist. Schritt 13 kann in* $O(N)$ *ausgeführt werden.*

Der Hauptaufwand der Endphase wird durch die Zuordnung der Zentren verursacht. Deswegen ist der Zeitaufwand der Endphase I_E *gleich* $I_E = O(N \times K \times A(.))$.

Wir können folgende Schlussfolgerungen aus unserer Laufzeitanalyse ziehen:

1. *Insgesamt ist die Laufzeit des Algorithmus 4 gleich:*

   $I_I + I_H + I_E = O(N \times \beta) + O(t \times \beta \times |P_1|) + O(N \times K \times A(.)) =$

   $O(\max(N \times \beta, t \times \beta \times |P_1|, N \times K \times A(.))) = O(N \times \beta)$

   Dieses Ergebnis gilt für viele Anwendungen, aber nicht immer, weil normalerweise $|P_1| \ll N \Rightarrow t \times |P_1| < N$ *und* $K \times A(.) < \beta$ *ist.*

2. *Wir haben bei allen drei Phasen festgestellt, dass der Hauptaufwand des Algorithmus durch die Berechnung der ersten Nachbarschaft und die Zuordnung der Zentren verursacht wird. Das ist eigentlich positiv, weil wir für die erste Nachbarschaftsberechnung effiziente Algorithmen präsentiert haben.*

5.4. Konzeptueller Vergleich des Algorithmus 4 mit den PAM und CLARA Algorithmen

In diesem Abschnitt vergleichen wir das Konzept unseres Cluster-Algorithmus (Algorithmus 4) mit den PAM und CLARA Algorithmen. Dazu beschreiben wir zuerst, wie die PAM und CLARA Algorithmen funktionieren.

Zunächst zum PAM-Algorithmus:

Algorithm 6 *-PAM Algorithmus*

Initialisierungsphase

1- Wähle zufällig (ohne Zurücksetzung) k von N Datensätzen als Medoid und bilde von diesen k-Medoids k Anfangscluster .

Hauptphase

WIEDERHOLE:

1- Ordne jedem normalen Datensatz (nicht Medoid) o dem Cluster mit dem nächsten Medoid m als Zentrum zu.

2- Für jedes Medoid m tue folgendes {

3- Für jeden normal Datensatz o tue folgendes {

4- berechne den durchschnittlichen Abstand $d_m(.)$ zwischen o und allen anderen Datensätzen des Clusters mit dem Medoid m als Zentrum

5- } // end for-Schleife Zeile 3

6- Wähle den Datensatz o mit dem kleinsten $d_m(o) < d_m(m)$ und vertausche o mit m im Cluster.

7- } // ende for-Schleife Zeile 2

SOLANGE SICH NICHTS ÄNDERT

Wie sich aus dem Ablauf des PAM-Algorithmus eindeutig rauslesen lässt, hat dieser Algorithmus für große k und N einen stark erhöhten Aufwand. Für jeden Datensatz o müssen wir den Abstand zwischen o und allen anderen Datensätzen berechnen. Das heißt, für N Datensätze brauchen wir mindestens $\frac{N(N-1)}{2}$ Distanzberechnungen. In Schritt 1 der Hauptphase ordnen wir jedem normalen Datensatz o das Cluster mit dem nächsten Medoid m, falls in Schritt 6 der Hauptphase innerhalb eines Clusters ein Medoid m durch einen normalen Datensatz vertauscht wurde, zu. Das heißt, dass Schritt 1, die Zuordnung der Datensätze zu den Clustern, solange wiederholt wird, bis der PAM-Algorithmus alle optimalen Medoids findet.

Um das Problem, das sich ergibt, wenn man den PAM-Algorithmus mit einer großen Datenmenge$X$ ausführen will, in den Griff zu kriegen, kann man den PAM-Algorithmus auf eine Teilmenge $T \subset X$, $|T| = s$ (sample) der gesamten Datenmenge X beschränken. Das ist die Idee, die hinter dem CLARA-Algorithmus steckt. Bei dem haben wir wie beim PAM-Algorithmus mindestens $\frac{s(s-1)}{2}$

Distanzberechnungen und eine mehrmalige Durchführung des Zuordnungsschrittes jedes Datensatzes o zu dem Cluster mit dem nächsten Medoid m. Der Aufwand des CLARA-Algorithmus mit sampel $T \subset X$, $|T| = s$, $|X| = N$ ist gleich $O(ks^2 + k(N-k))$ ([46] Seite 354). Wir sehen sofort, dass dieser Aufwand immer noch von der Kardinalität N der Datenmenge X abhängt.

Es gibt noch ein weiteres Problem beim CLARA-Algorithmus. Beim PAM-Algorithmus suchen wir nach den besten k-Medoids innerhalb der gesamten Datenmenge X. Aber beim CLARA-Algorithmus suchen wir innerhalb des sampel T nach den besten k-Medoids und falls ein- oder mehrere beste Medoids innerhalb der sampel T fehlen, dann findet der CLARA-Algorithmus nicht das beste Clustering.

Nachdem wir PAM und CLARA Algorithmen beschrieben haben, vergleichen wir unseren Cluster-Algorithmus (Algorithmus 4) mit den PAM und CLARA Algorithmen im Bezug auf die oben dargestellten Themen.

Wie wir bei der Laufzeitsanalyse des Algorithmus 4 gesehen haben, basiert der Hauptaufwand des Algorithmus 4 auf der ersten Nachbarschaftsberechnung. Im Gegensatz zu den PAM und CLARA Algorithmen berechnen wir in der Initialisierungsphase für jeden Datensatz $x \in X$ nur den Abstand zu den ersten Nachbarn. Das ist erheblich weniger Aufwand im Vergleich zu der Berechnung des Abstandes zu allen anderen Datensätzen. Nachdem wir in der Initialisierungsphase für jeden Datensatz $x \in X$ seine erste Nachbarschaftsmenge $N_1(x)$ festgestellt haben, bilden wir den Mittelpunkt x_M zwischen dem Kernpunkt $x \in L : \frac{min}{D(N_1^s(x))} > \alpha$ und seinem symmetrischen ersten Nachbarn $y \notin L : x \sim^1 y$, wobei min ist definiert als:

$min := \min_{x \in L} D(N_1^s(x))$

Dann fügen wir alle Mittelpunkte x_M in die Menge P ein. Um die Zentren der Cluster zu finden, wird in der Hauptphase von den Mittelpunkten $x_M \in P : |P| \ll |X|$ und nicht von Datensätzen $x \in X$ Gebrauch gemacht. Normalerweise ist die Anzahl der Mittelpunkte in der Menge P sehr viel geringer als die Anzahl der Datensätze in Menge X.

Ferner wird in der Hauptphase unseres Clustering-Algorithmus für jeden Mittelpunkt $x_M \in P$ seine erste Nachbarschaftsmenge $N_1(x_M)$ festgelegt. Hierbei ergibt sich durch die Mittelpunktsbildung zwischen den Kernmittelpunkten $x_m \in P : D(N_1^s(x_m)) \leq min$, wobei min definiert ist als:

$$min := \begin{cases} \min_{x_m \in T_1} D(\overrightarrow{N}_1(x_m)) & falls\, T_1 \neq \emptyset \\ \min_{x_m \in T_2} D(N_1^s(x_m)) & sonst \end{cases}$$

, mit T_1 und T_2 definiert ist als:

$T_1 := \{x_m \in P | \, x_m \, ist \, Normalpunkt\}$

$T_2 := \{x_m \in P \, | \, x_m \, ist \, Kernpunkt\}$

Wodurch die Zentren der Cluster konstruiert werden. Im Gegensatz zu den PAM- und CLARA-Algorithmen wird in der Endphase unseres Clusteringalgorithmus jeder Datensatz $x \in X$ nur ein mal zu dem nächsten Zentrum eines Clusters zugeordnet.

Unserer Clusteringalgorithmus weist folgende Eigenschaften auf:

1. Skalierbarkeit:

   Aufgrund der oben geschriebenen Fakten und der Tatsache, dass fast alle Schritte des Algorithmus 4 parallel durchführbar sind, können wir Algorithmus 4 auf ganz großen Datenmengen laufen lassen.

2. Entdeckung der Cluster einer beliebigen Gestalt:

   Unserer Clusteringalgorithmus kann beliebige Clusterformen (nur nicht geschachtelte Cluster) entdecken.

3. Fähigkeit Outlier zu ignorieren:

   Wenn wir den Parameter α richtig einschätzen, dann können wir den Einfluss der Outlier auf unseren Clusteringalgorithmus beseitigen.

4. Die Reihenfolge der Daten:

   Da unserer Clusteringalgorithmus deterministisch ist, ist er unabhängig von der Verarbeitung der Datenmenge in beliebigen Reihenfolge.

5. Entdeckung der Cluster von unterschiedlichen Dichten:

   Unserer Clusteringalgorithmus ist fähig, Cluster mit verschiedenen Dichten zu entdecken.

Wir haben unseren Clusteringalgorithmus (Algorithmus 4) mithilfe der Programmiersprache Java implementiert und präsentieren in Abb. 1 das Ergebnis für eine Eingabemenge X mit 3000 Datensätzen, von denen 600 Datensätze Outlier sind. Wir setzen die Parameter K und α jeweils gleich 5 und 0.2. In der Initialisierungsphase konstruiert unser Clusteringalgorithmus 32 Mittelpunkte, von denen in der Hauptphase die Zentren der Cluster gebildet werden. Innerhalb von 10 While-Schleifendurchläufen in der Hauptphase werden die Cluster durch unseren Algorithmus erkannt.

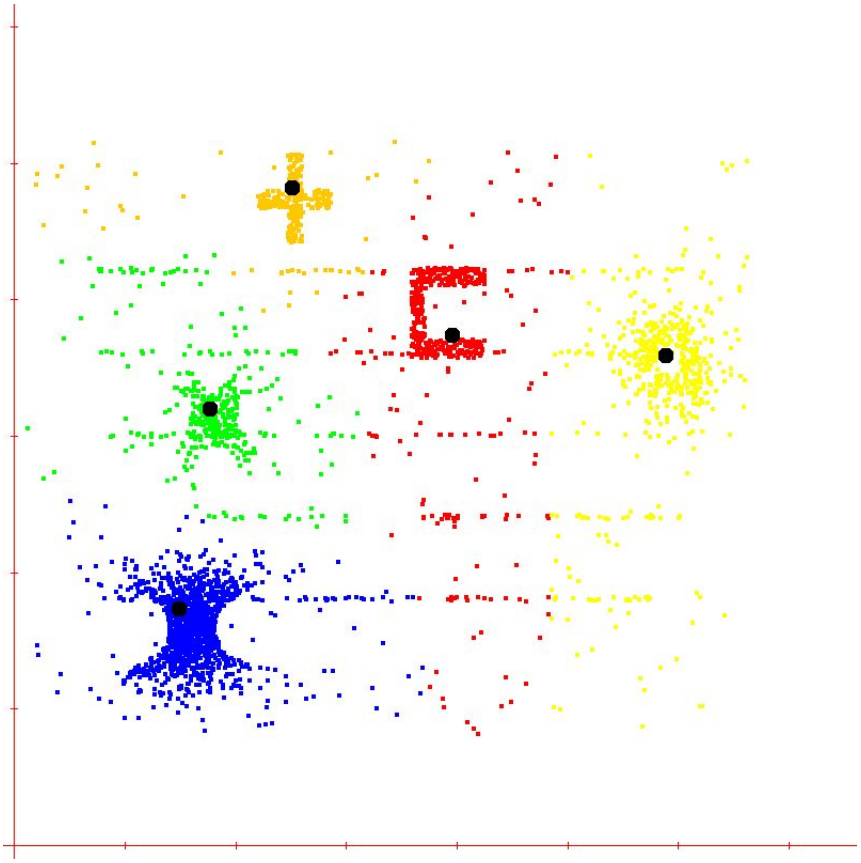

Abbildung 1: die Zentren der Cluster sind als große schwarze Punkte dargestellt. Wie deutlich in dem Bild zu sehen ist, haben die Cluster beliebige Formen und weisen verschiedene Dichten auf.

5.5. Empirischer Vergleich der Algorithmen 4 und 5 mit den PAM und CLARA Algorithmen

In diesem Abschnitt vergleichen wir empirisch unsere Algorithmen 4 und 5 mit den PAM und CLARA Algorithmen. Um einen Überblick über die Qualität unserer Cluster-Algorithmen gegenüber den PAM und CLARA Algorithmen zu gewinnen, benutzen wir Silhouettenkoeffizienten ([46] Seite 523). Der Silhouettenkoeffizient gibt eine von der Cluster-Anzahl unabhängige Maßzahl für die Qualität eines Clusterings an. Deswegen wird er ganz häufig benutzt. Der Silhouettenkoeffizient eines Objektes x_i wird berechnet als gewichteter Wert $s(x_i)$ des Mittelwerts $a(x_i)$ der Abstände zwischen x_i und allen anderen Objekten desselben Clusters und dem Mittelwert $b(x_i)$ der Distanz zwischen x_i und allen Objekten des nächst gelegenen Clusters. Hierzu beschreiben wir diese kurz.

Wie bisher sei $X = \{x_1, x_2, \ldots, x_N\}$ unsere Datenmenge. Nachdem wir unseren Cluster-Algorithmus auf die Datenmenge X angewendet haben und K Cluster $C_1, C_2, \ldots, C_K$ bestimmt haben, sei $x_i \in X : x_i \in C_m$. Dann seien $a(x_i)$ und $b(x_i)$ definiert wie folgt:

$a(x_i) := \frac{\sum_{x_j \in C_m : x_j \neq x_i} dist(x_i, x_j)}{|C_m| - 1}$

$b(x_i) := \min_{C_n, n=1,2,\ldots,k, m \neq n} \left\{ \frac{\sum_{x_j \in C_n} dist(x_i, x_j)}{|C_n|} \right\}$

Dann sei $s(x_i)$ als Silhouettenkoeffizient von x_i definiert als:

$s(x_i) := \frac{b(x_i) - a(x_i)}{\max(b(x_i), a(x_i))}$, wobei $s(x_i) \in [-1, 1]$

Wie man einfach nachvollziehen kann, je kleiner $a(x_i)$ wird, desto kompakter wird das Cluster C_m und je größer $b(x_i)$ wird, desto weiter entfernt liegt x_i vom anderen Cluster. Hieraus können wir sofort schließen, dass je mehr sich $s(x_i)$ der Eins annähert, desto kompakter ist das Cluster C_m und desto weiter weg liegt x_i entfernt von anderen Clustern.

Wir können jetzt ganz einfach die Qualität von einem Clustering-Verfahren A durch die Silhouettenkoeffizienten definieren als:

$Q(A) := \frac{\sum_{i=1}^{N} s(x_i)}{|X|}$

Es ist ganz offensichtlich, je mehr sich $Q(A)$ der Eins annähert, desto besser wird die Qualität von Algorithmus A.

Um unsere empirische Untersuchung durchzuführen, brauchen wir Daten. Hierzu haben wir vier Datenmengen X_1, X_2, X_3 und X_4, deren Datensätze entweder normal- oder gleichverteilt sind, durch den Datengenerator [58] erzeugt und 120 Datenmengen von einer einschlägigen Internetseite [65] heruntergeladen. Von 120 Datenmengen haben jeweils die ersten 30 Datenmengen D_1 zwei Dimensionen, die zweiten 30 Datenmengen D_2 10 Dimensonen, die dritten 30 Datenmengen D_3 50 Dimensionen und die vierten 30 Datenmengen D_4 100 Dimensionen. Die Datensätze in D_1, D_2, D_3 und D_4 haben 4, 10 oder 20 Cluster. Die Datensätze in D_1 und D_2 sind normal verteilt und haben beliebige

Form. Die Cluster der Datensätze in D_1 liegen dicht beieinander. Die Cluster der Datensätze in D_3 und D_4 haben ellipsoidische Form. Wir haben aber für unsere Cluster-Algorithmen keine bestimmte Verteilung für Datensätze vorausgesetzt. Das heißt, unsere Cluster-Algorithmen funktionieren unabhängig von einer bestimmten Verteilung der Datensätze.

Unsere Datenmengen X_1, X_2, X_3 und X_4 haben folgende Eigenschaften:

Datenmenge	Anzahl der Datensätze	Anzahl der Outlier
X_1	3000	150
X_2	3000	600
X_3	10000	2000
X_4	10000	2000

Tabelle 6: Eigenschaften der Mengen X_1, X_2, X_3 und X_4.
Die Cluster der Datenmengen X_1, X_2, X_3, X_4 liegen weit voneinander und haben beliebige Form.

Wir haben unsere Algorithmen mittels der Programmiersprache JAVA implementiert. PAM und CLARA sind schon in der Programmiersprache R (Version 3.2.3) im Package "cluster" vorhanden. Deswegen haben wir sie nicht mehr in JAVA erneut implementiert. Unsere Datensätze, welche wir für PAM und CLARA benutzt haben, hatten die Form einer Text-Datei (text-Tabelle). Nach dem Aufruf der PAM und CLARA Algorithmen in R sind $Q(PAM)$ und $Q(CLARA)$ in der Variable "silinfo$avg.width" der beiden Algorithmen vorhanden. Nach der Durchführung unseres Tests stellen wir unsere Resultate in folgenden Tabellen dar:

D_1	$Q(PAM)$	$Q(CLARA)$	$Q(A4)$	α_4	$Q(A_5)$	α_5
2d.4c.no0	0.6664575	0.7135286	0.6707964	0.1	0.6707451	0.1
2d.4c.no1	0.5997583	0.5801188	0.5968652	0.15	0.5972807	0.15
2d.4c.no2	0.611111	0.5665504	0.6000325	0.09	0.5884556	0.09
2d.4c.no3	0.7360709	0.7520457	0.7321040	0.1	0.7321264	0.1
2d.4c.no4	0.6179145	0.7522576	0.6183258	0.009	0.6223994	0.01
2d.4c.no5	0.5566542	0.5255293	0.5442842	0.08	0.5365884	0.08
2d.4c.no6	0.6526822	0.6489885	0.6686615	0.01	0.6628621	0.01
2d.4c.no7	0.6967681	0.7122679	0.7024078	0.004	0.7021738	0.004
2d.4c.no8	0.7102453	0.7219184	0.7168793	0.004	0.7173728	0.004
2d.4c.no9	0.6406503	0.6594804	0.6401203	0.03	0.6367011	0.03
2d.10c.no0	0.6253635	0.6714356	0.6089275	0.1	0.6051931	0.1
2d.10c.no1	0.532567	0.5895617	0.5095579	0.05	0.4994639	0.05
2d.10c.no2	0.6080502	0.5962525	0.6485830	0.03	0.6454818	0.03
2d.10c.no3	0.5903884	0.6870443	0.5261952	0.06	0.5248743	0.06
2d.10c.no4	0.6292947	0.6668754	0.5737291	0.01	0.5273665	0.01
2d.10c.no5	0.5418937	0.5894089	0.5466591	0.1	0.5120263	0.15
2d.10c.no6	0.5707784	0.6645607	0.5241513	0.05	0.5035999	0.05
2d.10c.no7	0.5593725	0.6713172	0.5716582	0.005	0.5614940	0.005
2d.10c.no8	0.5960342	0.614463	0.6210289	0.01	0.6137216	0.01
2d.10c.no9	0.6523734	0.7076644	0.6109686	0.01	0.6191172	0.01
2d.20c.no0	0.6616822	0.6807964	0.6118839	0.009	0.6103314	0.009
2d.20c.no1	0.6413621	0.6491859	0.5931882	0.01	0.5727598	0.01
2d.20c.no2	0.6044125	0.6151463	0.5726993	0.01	0.5607400	0.01
2d.20c.no3	0.6141647	0.6105132	0.5561467	0.03	0.5387897	0.03
2d.20c.no4	0.5743511	0.5801354	0.5614001	0.02	0.5397631	0.02
2d.20c.no5	0.5960883	0.6602949	0.6150063	0.02	0.6198083	0.02
2d.20c.no6	0.595397	0.5569747	0.5658389	0.01	0.5560229	0.01
2d.20c.no7	0.6224214	0.6603736	0.5555188	0.01	0.5636132	0.01
2d.20c.no8	0.64547	0.6490223	0.6040221	0.01	0.5983218	0.01
2d.20c.no9	0.6140971	0.6495616	0.6123221	0.009	0.5975692	0.009

Tabelle 7: Algorithmenqualität für Datenmenge D_1

$Q(.)$:= Qualität des Algorithmus im Zusammenhang mit der Datenmenge.

α_4 := Parameter α bezüglich Algorithmus 4.

α_5 := Parameter α bezüglich Algorithmus 5.

$2d.ic.noj$:= "2d" steht für zweidimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

D_1	$A(.)$	$z(A_4)$	$a(A_4)$	$z(A_5)$	$a(A_5)$
2d.4c.no0	1572	218	44	184	44
2d.4c.no1	1623	274	10	180	10
2d.4c.no2	1064	196	80	173	80
2d.4c.no3	1123	247	122	252	122
2d.4c.no4	863	520	245	479	243
2d.4c.no5	1638	189	30	170	30
2d.4c.no6	1670	353	127	288	127
2d.4c.no7	1028	334	206	386	206
2d.4c.no8	1078	515	318	518	318
2d.4c.no9	876	305	136	330	136
2d.10c.no0	2972	647	153	601	153
2d.10c.no1	2525	392	84	390	84
2d.10c.no2	3073	742	224	627	224
2d.10c.no3	3359	561	110	563	110
2d.10c.no4	3291	1120	516	1050	516
2d.10c.no5	3630	521	188	540	103
2d.10c.no6	3408	797	216	749	216
2d.10c.no7	2534	628	271	577	271
2d.10c.no8	2830	1254	622	1227	622
2d.10c.no9	3580	1597	676	1339	676
2d.20c.no0	1517	487	342	506	342
2d.20c.no1	1231	474	294	521	294
2d.20c.no2	1084	419	305	439	305
2d.20c.no3	1129	351	229	397	229
2d.20c.no4	1446	291	156	289	156
2d.20c.no5	1178	376	246	430	246
2d.20c.no6	1177	548	328	526	328
2d.20c.no7	1237	622	365	685	365
2d.20c.no8	1152	396	271	434	271
2d.20c.no9	1206	366	215	425	215

Tabelle 8: Zeitverlauf für Datenmenge D_1

$2d.ic.noj$:= "2d" steht für zweidimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

$A(.)$:=Anzahl der Datensätze der Datenmenge $2d.ic.noj$.

$z(.)$:= Laufzeit in Millisekunden der Algorithmen 4 und 5 in Bezug auf Hauptphase und Endphase.

$a(.)$:= Anzahl der Anfangsmittelpunkte der Algorithmen 4 und 5 innerhalb der While-Schleife in der Hauptphase, mit denen die Zentren der Cluster berechnet werden.
Folgende Phänomene können wir aus den Tabellen 7 und 8 herauslesen:

Schlussfolgerung 7 -

1. *Wie aus Tabelle 7 zu ersehen ist, hat bei 27 von 30 Datenmengen PAM oder CLARA eine bessere Qualität. So steht CLARA als Sieger der Gruppe mit durchschnittlicher Qualität von 0.6467758 fest, gefolgt in absteigender Reihenfolge von PAM mit 0.6187958, Algorithmus 4 mit 0.6026654 und schließlich Algorithmus 5 mit 0.5945588. Die Cluster der Datenmengen in Tabelle 7 liegen dicht beieinander.*
2. *Wenn die Cluster der Datenmenge dicht beieinander liegen, haben Algorithmen 4 und 5 schlechtere Qualität im Vergleich zu den PAM und CLARA Algorithmen. Der Grund dafür ist, dass die Algorithmen 4 und 5 auf erster Nachbarschaftsberechnung basieren. Wenn die Cluster ganz dicht nebeneinnder liegen, kann der erste Nachbar im anderen Cluster liegen. Hierzu haben PAM und CLARA ein anderes Konzept, nämlich sie berechnen die Distanz für jeden Punkt zu allen anderen Punkten der Datenmenge. Durch diesen Mehraufwand haben PAM und CLARA bessere Qualität, wenn die Cluster dicht beieinander liegen.*
3. *Mathematisch können wir weit voneinander liegende Cluster für Algorithmen 4 und 5 so definieren:*

   Zwei Cluster liegen weit voneinander, wenn der maximale innere erste Nachbarschaftsabstand zwischen je zwei Punkten des gleichen Clusters kleiner als der Abstand zwischen zwei Clustern ist.
4. *Aus Tabelle 8 können folgende Werte hergeleitet werden:*

   $\max(z(A_4))_{2d.4c.noj,\, j=0,\ldots,9} = 520$

   $\max(z(A_5))_{2d.4c.noj,\, j=0,\ldots,9} = 518$

   $\max(z(A_4))_{2d.10c.noj,\, j=0,\ldots,9} = 1597$

   $\max(z(A_5))_{2d.10c.noj,\, j=0,\ldots,9} = 1339$

   $\max(z(A_4))_{2d.20c.noj,\, j=0,\ldots,9} = 622$

   $\max(z(A_5))_{2d.20c.noj,\, j=0,\ldots,9} = 685$

   $\max(z(A_4))_{2d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 1597$

   $\max(z(A_5))_{2d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 1339$

   Da die Datenmengen mit 10 Clustern mehr Datensätze enthalten als die mit 20, ist die maximale Laufzeit der zehnclusterigen Datenmengen höher als die der Datenmengen mit 20 Clustern.

D_2	$Q(PAM)$	$Q(CLARA)$	$Q(A4)$	α_4	$Q(A_5)$	α_5
10d.4c.no0	0.2332177	0.3380201	0.4277786	0.5	0.4280505	0.5
10d.4c.no1	0.395933	0.3993719	0.4435493	0.6	0.4409053	0.6
10d.4c.no2	0.3069376	0.236141	0.3551645	0.3	0.3590993	0.3
10d.4c.no3	0.2120943	0.2052464	0.3383952	0.5	0.3387290	0.5
10d.4c.no4	0.3594542	0.3709419	0.4102659	0.5	0.4123332	0.5
10d.4c.no5	0.3390801	0.2200771	0.3748333	0.4	0.3720782	0.4
10d.4c.no6	0.3432847	0.2546784	0.3868606	0.5	0.3938722	0.5
10d.4c.no7	0.4219774	0.4420135	0.4744399	0.7	0.4754985	0.7
10d.4c.no8	0.3541207	0.3908021	0.3862363	0.6	0.3851339	0.6
10d.4c.no9	0.2949453	0.3019512	0.3496955	0.6	0.3482009	0.6
10d.10c.no0	0.2204524	0.2277932	0.2826937	0.5	0.2767946	0.5
10d.10c.no1	0.2785265	0.3148209	0.2878002	0.6	0.2791276	0.6
10d.10c.no2	0.261595	0.2500278	0.2715924	0.5	0.2763600	0.5
10d.10c.no3	0.2626311	0.2996543	0.2358882	0.3	0.2182136	0.3
10d.10c.no4	0.2603872	0.2604487	0.2576343	0.4	0.2754726	0.4
10d.10c.no5	0.2603851	0.2218917	0.3138984	0.6	0.3141478	0.6
10d.10c.no6	0.2280292	0.2992682	0.2704906	0.3	0.1919738	0.3
10d.10c.no7	0.2632586	0.2835675	0.2940822	0.4	0.2895827	0.4
10d.10c.no8	0.2419483	0.2565455	0.3195277	0.5	0.2970156	0.5
10d.10c.no9	0.2363112	0.2434491	0.2545474	0.4	0.2461732	0.4
10d.20c.no0	0.5235798	0.4742154	0.5190298	0.4	0.5190544	0.4
10d.20c.no1	0.4886391	0.4802519	0.5249398	0.3	0.5249398	0.3
10d.20c.no2	0.5005436	0.517877	0.4996672	0.4	0.4997420	0.4
10d.20c.no3	0.5063699	0.4740745	0.4976448	0.3	0.4992509	0.3
10d.20c.no4	0.5115438	0.4993789	0.5123078	0.5	0.5117683	0.5
10d.20c.no5	0.528852	0.4866492	0.5177531	0.4	0.5177531	0.4
10d.20c.no6	0.4661547	0.4604818	0.4639862	0.3	0.4642661	0.3
10d.20c.no7	0.5345036	0.4472443	0.5190673	0.3	0.5190727	0.3
10d.20c.no8	0.4955943	0.4511459	0.4827332	0.4	0.4832702	0.4
10d.20c.no9	0.5048057	0.4694119	0.5079303	0.4	0.5089017	0.4

Tabelle 9: Algorithmenqualität für Datenmenge D_2

$Q(.)$:= Qualität des Algorithmus im Zusammenhang mit der Datenmenge.

α_4 := Parameter α bezüglich Algorithmus 4.

α_5 := Parameter α bezüglich Algorithmus 5.

$10d.ic.noj$:= "10d" steht für zehndimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

D_2	$A(.)$	$z(A_4)$	$a(A_4)$	$z(A_5)$	$a(A_5)$
10d.4c.no0	1289	1131	127	1123	127
10d.4c.no1	958	451	47	444	47
10d.4c.no2	838	734	93	679	93
10d.4c.no3	1318	1092	100	1084	100
10d.4c.no4	933	886	116	850	116
10d.4c.no5	1139	1186	122	1327	122
10d.4c.no6	977	814	99	773	99
10d.4c.no7	1482	874	67	813	67
10d.4c.no8	966	833	78	723	78
10d.4c.no9	1183	766	55	775	55
10d.10c.no0	2729	2750	183	2475	183
10d.10c.no1	3056	2520	64	2074	64
10d.10c.no2	3618	5010	306	4899	306
10d.10c.no3	2594	5024	425	5347	425
10d.10c.no4	3034	4576	346	4950	346
10d.10c.no5	3788	4293	131	3582	131
10d.10c.no6	2161	3921	377	4226	377
10d.10c.no7	3222	3659	162	2984	162
10d.10c.no8	2666	2305	157	2277	157
10d.10c.no9	2691	4696	397	5175	397
10d.20c.no0	1013	1118	180	1227	180
10d.20c.no1	904	828	180	873	180
10d.20c.no2	1164	1315	186	1464	186
10d.20c.no3	1201	1543	249	1723	249
10d.20c.no4	1228	1107	181	1402	181
10d.20c.no5	1248	1163	208	1325	208
10d.20c.no6	1279	1425	250	1561	250
10d.20c.no7	1082	1331	216	1466	216
10d.20c.no8	1284	1506	236	1637	236
10d.20c.no9	1316	1734	241	1738	241

Tabelle 10: Zeitverlauf für Datenmenge D_2

$10d.ic.noj$:= "10d" steht für zehndimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

$A(.)$:=Anzahl der Datensätze der Datenmenge $10d.ic.noj$.

$z(.)$:= Laufzeit in Millisekunden der Algorithmen 4 und 5 in Bezug auf Hauptphase und Endphase.

$a(.)$:= Anzahl der Anfangsmittelpunkte der Algorithmen 4 und 5 innerhalb der While-Schleife in der Hauptphase, mit denen die Zentren der Cluster berechnet werden.
Folgende Phänomene können wir aus den Tabellen 9 und 10 herauslesen:

Schlussfolgerung 8 -

1. *Wie aus Tabelle 9 zu ersehen ist, hat bei 19 von 30 Datenmengen Algorithmus 4 oder 5 eine bessere Qualität. So steht Algorithmus 4 als Sieger der Gruppe mit durchschnittlicher Qualität von 0.3926811, gefolgt in absteigender Reihenfolge von Algorithmus 5 mit 0.3888927, PAM mit 0.3611719 und schließlich CLARA mit 0.3525814.*
2. *Bei zehndimensionalen Datenmengen mit normalverteilten Datensätzen haben Algorithmus 4 und 5 eine bessere Qualität im Vergleich zu den PAM und CLARA Algorithmen.*
3. *Aus Tabelle 10 können folgende Werte hergeleitet werden:*

   $\max(z(A_4))_{10d.4c.noj,\, j=0,\ldots,9} = 1186$

   $\max(z(A_5))_{10d.4c.noj,\, j=0,\ldots,9} = 1327$

   $\max(z(A_4))_{10d.10c.noj,\, j=0,\ldots,9} = 5024$

   $\max(z(A_5))_{10d.10c.noj,\, j=0,\ldots,9} = 5347$

   $\max(z(A_4))_{10d.20c.noj,\, j=0,\ldots,9} = 1734$

   $\max(z(A_5))_{10d.20c.noj,\, j=0,\ldots,9} = 1738$

   $\max(z(A_4))_{10d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 5024$

   $\max(z(A_5))_{10d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 5347$

   Da die Datenmengen mit 10 Clustern mehr Datensätze enthalten als die mit 20, ist die maximale Laufzeit der zehnclusterigen Datenmengen höher als die der Datenmengen mit 20 Clustern.

D_3	$Q(PAM)$	$Q(CLARA)$	$Q(A4)$	α_4	$Q(A_5)$	α_5
50d.4c.no0	0.4369275	0.4725565	0.4536527	0.2	0.4736734	0.2
50d.4c.no1	0.351983	0.6072022	0.5223248	0.2	0.5190304	0.2
50d.4c.no2	0.3381868	0.4434176	0.5004352	0.005	0.5001275	0.005
50d.4c.no3	0.4338542	0.5027491	0.4080962	0.005	0.4146479	0.005
50d.4c.no4	0.3442666	0.5380844	0.3957126	0.005	0.3056529	0.005
50d.4c.no5	0.2938772	0.3551136	0.5435591	0.005	0.5545777	0.005
50d.4c.no6	0.3639612	0.2764178	0.3728913	0.005	0.3743652	0.005
50d.4c.no7	0.2595016	0.3542916	0.4068285	0.005	0.4014367	0.005
50d.4c.no8	0.5506588	0.5718784	0.5079248	0.005	0.4610841	0.005
50d.4c.no9	0.3190298	0.3867558	0.2691414	0.005	0.4975760	0.005
50d.10c.no0	0.3133994	0.4845042	0.4105316	0.05	0.3858556	0.05
50d.10c.no1	0.3457535	0.4222939	0.4213232	0.04	0.3928517	0.04
50d.10c.no2	0.3432877	0.4181818	0.4005900	0.04	0.3965434	0.04
50d.10c.no3	0.3319446	0.4348819	0.4131695	0.01	0.3542816	0.01
50d.10c.no4	0.3091894	0.4098111	0.4212837	0.04	0.3841909	0.04
50d.10c.no5	0.3416078	0.4336601	0.4109176	0.05	0.3874646	0.05
50d.10c.no6	0.3134239	0.4441542	0.3713079	0.03	0.3319661	0.03
50d.10c.no7	0.2716947	0.4093493	0.3848217	0.01	0.3739946	0.01
50d.10c.no8	0.2937504	0.4538051	0.3986155	0.05	0.3686483	0.05
50d.10c.no9	0.3460646	0.351057	0.4265446	0.05	0.3910622	0.05
50d.20c.no0	0.3105436	0.3770526	0.3641836	0.1	0.3073012	0.1
50d.20c.no1	0.3424154	0.3690901	0.3786553	0.01	0.3312144	0.01
50d.20c.no2	0.3182129	0.4371463	0.3840825	0.03	0.3666070	0.03
50d.20c.no3	0.3552297	0.4014186	0.3859843	0.03	0.3616704	0.03
50d.20c.no4	0.2917679	0.4492841	0.3834490	0.03	0.3708254	0.03
50d.20c.no5	0.3494053	0.5089849	0.4000144	0.08	0.3420084	0.08
50d.20c.no6	0.3660176	0.4508114	0.4088171	0.08	0.3760995	0.08
50d.20c.no7	0.3141197	0.415295	0.3998850	0.05	0.3520446	0.05
50d.20c.no8	0.2892178	0.3356343	0.3559681	0.06	0.3250657	0.06
50d.20c.no9	0.3058164	0.378223	0.3904395	0.06	0.3754037	0.06

Tabelle 11: Algorithmenqualität für Datenmenge D_3

$Q(.)$:= Qualität des Algorithmus im Zusammenhang mit der Datenmenge.

α_4 := Parameter α bezüglich Algorithmus 4.

α_5 := Parameter α bezüglich Algorithmus 5.

$50d.ic.noj$:= "50d" steht für fünfzigdimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

D_3	$A(.)$	$z(A_4)$	$a(A_4)$	$z(A_5)$	$a(A_5)$
50d.4c.no0	1064	2161	65	2526	65
50d.4c.no1	984	1791	59	1635	59
50d.4c.no2	1371	5138	83	2801	83
50d.4c.no3	1098	3507	70	2091	70
50d.4c.no4	351	648	39	829	39
50d.4c.no5	1331	5960	82	2806	82
50d.4c.no6	1152	**3053**	64	2526	64
50d.4c.no7	949	1828	57	1760	57
50d.4c.no8	683	1866	52	1447	52
50d.4c.no9	1469	6467	89	3172	89
50d.10c.no0	2698	19001	131	7199	131
50d.10c.no1	3219	30687	161	9833	161
50d.10c.no2	2919	24338	149	8183	149
50d.10c.no3	2713	23028	159	7769	159
50d.10c.no4	2462	15399	131	6285	131
50d.10c.no5	2950	23430	155	8697	155
50d.10c.no6	2328	14137	101	5554	101
50d.10c.no7	3140	28791	165	10013	165
50d.10c.no8	2242	13810	141	5969	141
50d.10c.no9	3293	31792	168	10884	168
50d.20c.no0	1254	4150	110	2885	110
50d.20c.no1	1153	4006	145	2971	145
50d.20c.no2	1308	5215	148	3391	148
50d.20c.no3	1246	4875	154	3076	154
50d.20c.no4	1243	4752	153	3239	153
50d.20c.no5	1062	3180	128	2801	128
50d.20c.no6	1007	3148	126	2600	126
50d.20c.no7	1181	3899	115	2366	115
50d.20c.no8	1115	4269	131	2890	131
50d.20c.no9	1142	3724	122	2814	122

Tabelle 12: Zeitverlauf für Datenmenge D_3

$50d.ic.noj$:= "50d" steht für fünfzigdimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

$A(.)$:=Anzahl der Datensätze der Datenmenge $50d.ic.noj$.

$z(.)$:= Laufzeit in Millisekunden der Algorithmen 4 und 5 in Bezug auf Hauptphase und Endphase.

$a(.)$:= Anzahl der Anfangsmittelpunkte der Algorithmen 4 und 5 innerhalb der While-Schleife in der Hauptphase, mit denen die Zentren der Cluster berechnet werden.
Folgende Phänomene können wir aus den Tabellen 11 und 12 herauslesen:

Schlussfolgerung 9 -

1. *Wie aus der Tabelle 11 zu ersehen ist, hat bei 19 von 30 Datenmengen CLARA oder PAM eine bessere Qualität. So steht CLARA als Sieger der Gruppe mit durchschnittlicher Qualität von 0.4297698, gefolgt in absteigender Reihenfolge von Algorithmus 4 mit 0.4097050, Algorithmus 5 mit 0.3925757 und schließlich PAM mit 0.3381703.*
2. *Aus Tabelle 12 können folgende Werte hergeleitet werden:*

   $\max(z(A_4))_{50d.4c.noj,\, j=0,\ldots,9} = 6467$

   $\max(z(A_5))_{50d.4c.noj,\, j=0,\ldots,9} = 3172$

   $\max(z(A_4))_{50d.10c.noj,\, j=0,\ldots,9} = 31792$

   $\max(z(A_5))_{50d.10c.noj,\, j=0,\ldots,9} = 10884$

   $\max(z(A_4))_{50d.20c.noj,\, j=0,\ldots,9} = 5215$

   $\max(z(A_5))_{50d.20c.noj,\, j=0,\ldots,9} = 3391$

   $\max(z(A_4))_{50d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 31792$

   $\max(z(A_5))_{50d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 10884$

   Da die Datenmengen mit 10 Clustern mehr Datensätze enthalten als die mit 20, ist die maximale Laufzeit der zehnclusterigen Datenmengen höher als die der Datenmengen mit 20 Clustern.

D_4	$Q(PAM)$	$Q(CLARA)$	$Q(A4)$	α_4	$Q(A_5)$	α_5
100d.4c.no0	0.2283912	0.5317558	0.3664888	0.001	0.2561936	0.001
100d.4c.no1	0.3065385	0.4524103	0.2647033	0.001	0.4144693	0.001
100d.4c.no2	0.3260648	0.2806662	0.1128758	0.001	0.1950251	0.001
100d.4c.no3	0.3992755	0.4534713	0.2420320	0.001	0.4089170	0.001
100d.4c.no4	0.1883858	0.2749245	0.2553797	0.001	0.2689498	0.001
100d.4c.no5	0.364292	0.1970926	0.4666119	0.001	0.4399175	0.001
100d.4c.no6	0.3676876	0.2779428	0.3663120	0.001	0.3220417	0.001
100d.4c.no7	0.3432546	0.3825747	0.4027930	0.001	0.4326823	0.001
100d.4c.no8	0.2840828	0.7622504	0.3468573	0.001	0.3590279	0.001
100d.4c.no9	0.2442781	0.272957	0.2454284	0.001	0.4838280	0.001
100d.10c.no0	0.1145056	0.4201479	0.3740085	0.07	0.2984394	0.07
100d.10c.no1	0.20567	0.1526175	0.4006398	0.07	0.3645500	0.07
100d.10c.no2	0.20984	0.2975219	0.3721843	0.05	0.3457343	0.05
100d.10c.no3	0.2585536	0.3823045	0.4295343	0.02	0.3537344	0.02
100d.10c.no4	0.1568048	0.3451905	0.3642182	0.02	0.3430702	0.02
100d.10c.no5	0.1592546	0.2663006	0.3202574	0.02	0.3021440	0.02
100d.10c.no6	0.2725011	0.436426	0.4295983	0.02	0.4143024	0.02
100d.10c.no7	0.2079653	0.2548121	0.3609420	0.02	0.3076606	0.02
100d.10c.no8	0.2313363	0.2823984	0.3814274	0.02	0.3597833	0.02
100d.10c.no9	0.2260411	0.3857339	0.4021359	0.04	0.3016454	0.04
100d.20c.no0	0.08695608	0.4510733	0.3614153	0.04	0.2964990	0.04
100d.20c.no1	0.2140723	0.4310246	0.3678745	0.04	0.3416479	0.04
100d.20c.no2	0.2314339	0.434135	0.3948136	0.04	0.2929110	0.04
100d.20c.no3	0.06987857	0.2988164	0.3331199	0.04	0.2918659	0.04
100d.20c.no4	0.1754313	0.3646763	0.3804628	0.04	0.3201306	0.04
100d.20c.no5	0.1711963	0.2792105	0.3527252	0.04	0.3281149	0.04
100d.20c.no6	0.2080261	0.2398585	0.3705687	0.04	0.3333615	0.04
100d.20c.no7	0.06302427	0.3770909	0.3534772	0.04	0.2770750	0.04
100d.20c.no8	0.181684	0.4144442	0.3746540	0.1	0.3483406	0.1
100d.20c.no9	0.2128979	0.3571465	0.3749281	0.1	0.3590385	0.1

Tabelle 13: Algorithmenqualität für Datenmenge D_4

$Q(.)$:= Qualität des Algorithmus im Zusammenhang mit der Datenmenge.

α_4 := Parameter α bezüglich Algorithmus 4.

α_5 := Parameter α bezüglich Algorithmus 5.

$100d.ic.noj$:= "100d" steht für hundertdimensionale Datensätze, "ic, i=4,10,20" steht für die Anzahl der Cluster, "noj, j=0,1,...,9" steht für die Nummer der Datenmenge.

D_4	$A(.)$	$z(A_4)$	$a(A_4)$	$z(A_5)$	$a(A_5)$
100d.4c.no0	1286	10537	81	6846	81
100d.4c.no1	1013	5847	63	4469	63
100d.4c.no2	1331	9884	80	7072	80
100d.4c.no3	1506	12471	88	8549	88
100d.4c.no4	1117	8046	85	5640	85
100d.4c.no5	849	3601	61	3283	61
100d.4c.no6	1311	8367	80	6976	80
100d.4c.no7	1166	6429	61	5909	61
100d.4c.no8	629	2883	51	2281	51
100d.4c.no9	967	5143	67	4212	67
100d.10c.no0	2892	67385	155	24909	155
100d.10c.no1	2986	71641	97	41420	97
100d.10c.no2	3100	72503	84	46149	84
100d.10c.no3	2450	41547	133	19994	133
100d.10c.no4	2987	77173	166	26303	166
100d.10c.no5	2803	63934	166	22388	166
100d.10c.no6	3152	80763	183	26455	183
100d.10c.no7	2716	57812	139	26858	139
100d.10c.no8	2103	29425	128	15215	128
100d.10c.no9	2550	48703	136	23554	136
100d.20c.no0	1338	11264	122	7699	122
100d.20c.no1	1007	6788	128	5061	128
100d.20c.no2	1079	7569	123	5728	123
100d.20c.no3	1151	7747	95	5945	95
100d.20c.no4	1137	8123	117	5976	117
100d.20c.no5	1124	8540	136	6187	136
100d.20c.no6	1029	6896	119	5394	119
100d.20c.no7	1013	7520	130	5540	130
100d.20c.no8	1151	8787	125	6236	125
100d.20c.no9	1220	10205	130	6961	130

Tabelle 14: Zeitverlauf für Datenmenge D_4

$100d.ic.noj$:= “100d” steht für hundertdimensionale Datensätze, “ic, i=4,10,20” steht für die Anzahl der Cluster, “noj, j=0,1,...,9” steht für die Nummer der Datenmenge.

$A(.)$:=Anzahl der Datensätze der Datenmenge $100d.ic.noj$.

$z(.)$:= Laufzeit in Millisekunden der Algorithmen 4 und 5 in Bezug auf Hauptphase und Endphase.

$a(.)$:= Anzahl der Anfangsmittelpunkte der Algorithmen 4 und 5 innerhalb der While-Schleife in der Hauptphase, mit denen die Zentren der Cluster berechnet werden.
Folgende Phänomene können wir aus den Tabellen 13 und 14 herauslesen:

Schlussfolgerung 10 -

1. *Wie aus der Tabelle 13 zu sehen ist, hat bei 16 von 30 Datenmengen Algorithmus 4 oder Algorithmus 5 eine bessere Qualität. So steht CLARA als Sieger der Gruppe mit durchschnittlicher Qualität von 0.3585660, gefolgt in absteigender Reihenfolge von Algorithmus 4 mit 0.3522822, Algorithmus 5 mit 0.3387034 und schließlich PAM mit 0.2236441.*
2. *Aus der Tabelle 14 können folgende Werte hergeleitet werden:*

   $\max(z(A_4))_{100d.4c.noj,\, j=0,\ldots,9} = 12471$

   $\max(z(A_5))_{100d.4c.noj,\, j=0,\ldots,9} = 8549$

   $\max(z(A_4))_{100d.10c.noj,\, j=0,\ldots,9} = 80763$

   $\max(z(A_5))_{100d.10c.noj,\, j=0,\ldots,9} = 46149$

   $\max(z(A_4))_{100d.20c.noj,\, j=0,\ldots,9} = 11264$

   $\max(z(A_5))_{100d.20c.noj,\, j=0,\ldots,9} = 7699$

   $\max(z(A_4))_{100d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 80763$

   $\max(z(A_5))_{100d.ic.noj,\, i=4,10,20\, j=0,\ldots,9} = 46149$

   Da die Datenmengen mit 10 Clustern mehr Datensätze enthalten als die mit 20, ist die maximale Laufzeit der zehnclusterigen Datenmengen höher als die der Datenmengen mit 20 Clustern.

Datenmenge	$Q(PAM)$	$Q(CLARA)$	$Q(A4)$	α_4	$Q(A_5)$	α_5
X_1	0.6592006	0.7076226	0.7905039	0.15	0.7905421	0.15
X_2	0.4479615	0.4694805	0.5913158	0.2	0.5912459	0.2
X_3	0.5764851	0.5178545	0.6137964	0.16	0.6186731	0.16
X_4	0.5151939	0.5526228	0.6097399	0.1	0.6106528	0.1

Tabelle 15: $Q(X_i)$:= Qualität des Algorithmus im Zusammenhang mit den Datenmengen X_i, $i = 1, 2, 3, 4$. α_4 := Parameter α bezüglich Algorithmus 4.
α_5 := Parameter α bezüglich Algorithmus 5.
Folgendes Phänomen können wir aus der Tabelle 15 herauslesen:

Schlussfolgerung 11 -

Wie aus Tabelle 15 zu ersehen ist, haben Algorithmus 4 und 5 bessere Qualität in Bezug auf die Datenmengen X_1, X_2, X_3 und X_4. So steht Algorithmus 5 als Sieger der Gruppe mit durchschnittlicher Qualität von 0.6527785 fest, gefolgt in absteigender Reihenfolge von Algorithmus 4 mit 0.651339, CLARA mit 0.5618951 und schließlich PAM mit 0.5497103. Wir können dieses Ergebnis so interpretieren, dass die Algorithmen 4 und 5, wenn die Cluster der Datenmenge weit voneinander entfernt liegen, besser als PAM und CLARA funktionieren.

Aus den Schlussfolgerungen 7 bis 11 können wir folgendes schließen:

Schlussfolgerung 12 -

1. *Wir haben oben folgende Werte festgestellt:*

   a- $\max(z(A_4))_{2d.ic.noj} = 1597$ $\max(z(A_4))_{10d.ic.noj} = 5024$
   $\max(z(A_4))_{50d.ic.noj} = 31792$ $\max(z(A_4))_{100d.ic.noj} = 80763$
   Hierbei sind $i = 4, 10, 20$ *und* $j = 1, \ldots, 9$. *Wenn wir diese Werte miteinander vergleichen, sehen wir, dass sie ungefähr von gleicher Ordnung sind.*

   b- $\max(z(A_5))_{2d.ic.noj} = 1339$ $\max(z(A_5))_{10d.ic.noj} = 5347$
   $\max(z(A_5))_{50d.ic.noj} = 10884$ $\max(z(A_5))_{100d.ic.noj} = 46149$
   Hierbei sind $i = 4, 10, 20$ *und* $j = 1, \ldots, 9$. *Wenn wir diese Werte miteinander vergleichen, sehen wir, dass sie ungefähr von gleicher Ordnung sind.*

   c- *Damit können wir hier schließen, dass die Algorithmen 4 und 5 skallierbar sind. Man kann sie für hochdimensionale Datensätze benutzen.*

2. *Wir können auch hier schließen, dass es paarweise direkte Verhältnisse zwischen Anzahl der Datensätze der Datenmenge* $A(.)$, *Laufzeit des Algorithmus* $z(.)$ *und Anzahl der Anfangsmittelpunkte innerhalb der While-Schleife in der Hauptphase* $a(.)$ *gibt. D.h. je größer* $A(.)$ *wird, desto größer werden* $z(.)$ *und* $a(.)$.

An dieser Stelle versuchen wir, die Frage "Wie sensibel ist der Parameter α in Algorithmus 4?" zu beantworten. Dazu rufen wir den Algorithmus 4 mit den Datenmengen X_1, X_2, X_3 und X_4 im Zusammenhang mit verschiedenen Werten des Parameter α auf. Wir sehen, wie sich die unterschiedlichen Werte von α auf die Qualität $Q(.)$ des Algorithmus 4 auswirken.

Datenmenge	$Q/\alpha = 0.01$	$Q/\alpha = 0.02$	$Q/\alpha = 0.03$	$Q/\alpha = 0.04$	$Q/\alpha = 0.05$
X_1	0.7054876	0.7905372	0.7904267	0.7905262	0.7902881
X_2	0.4217204	0.4593777	0.5742594	0.4108919	0.5371754
X_3	0.4240458	0.3911393	0.5465156	0.4497173	0.5534120
X_4	0.3240185	0.4480308	0.4529959	0.5985937	0.5088783

Tabelle 16: Qualität des Algorithmus 4 für $\alpha = 0.01, 0.02, 0.03, 0.04, 0.05$.

$Q/\alpha :=$ Qualität $Q(.)$ des Algorithmus 4 für α gleich 0.01,0.02,0.03,0.04 und 0.05.

Datenmenge	$Q/\alpha = 0.06$	$Q/\alpha = 0.07$	$Q/\alpha = 0.08$	$Q/\alpha = 0.09$	$Q/\alpha = 0.1$
X_1	0.7898462	0.7903721	0.7903680	0.7901059	0.7904851
X_2	0.5214926	0.5402056	0.5313584	0.5312204	0.5893966
X_3	0.3467154	0.4415500	0.4520723	0.5540819	0.5496588
X_4	0.6054729	0.6027981	0.6071311	0.6092747	0.6097399

Tabelle 17: Qualität des Algorithmus 4 für $\alpha = 0.06, 0.07, 0.08, 0.09, 0.1$.

$Q/\alpha :=$ Qualität $Q(.)$ des Algorithmus 4 für α gleich 0.06,0.07,0.08,0.09 und 0.1.

Datenmenge	$Q/\alpha = 0.11$	$Q/\alpha = 0.12$	$Q/\alpha = 0.13$	$\alpha = 0.14$	$Q/\alpha = 0.15$
X_1	0.7904851	0.7900703	0.7904851	0.7903235	0.7905039
X_2	0.5655648	0.5927846	0.5870755	0.5707833	0.5825765
X_3	0.5949270	0.5970336	0.6053733	0.6075694	0.6167057
X_4	0.6103098	0.6110142	0.6108068	0.6108673	0.6107716

Tabelle 18: Qualität des Algorithmus 4 für $\alpha = 0.11, 0.12, 0.13, 0.14, 0.15$

$Q/\alpha :=$ Qualität $Q(.)$ des Algorithmus 4 für α gleich 0.11,0.12,0.13,0.14 und 0.15.

Datenmenge	$Q/\alpha = 0.16$	$Q/\alpha = 0.17$	$Q/\alpha = 0.18$	$Q/\alpha = 0.19$	$Q/\alpha = 0.2$
X_1	0.7905039	0.7905039	0.7903874	0.7899742	0.7899742
X_2	0.5872922	0.5838085	0.5802809	0.5846368	0.5913158
X_3	0.6137964	0.6117325	0.6161501	0.6163143	0.6194990
X_4	0.6096249	0.6096249	0.6099211	0.6071906	0.6071906

Tabelle 19: Qualität des Algorithmus 4 für $\alpha = 0.16, 0.17, 0.18, 0.19, 0.2$

$Q/\alpha :=$ Qualität $Q(.)$ des Algorithmus 4 für α gleich 0.16,0.17,0.18,0.19 und 0.2.

Die Tabellen 16 bis 19 sind die Grundlage für Diagramm 1. Dabei wurden die Qualitätswerte $Q(.)$ auf die 2. Nachkommastelle gerundet.

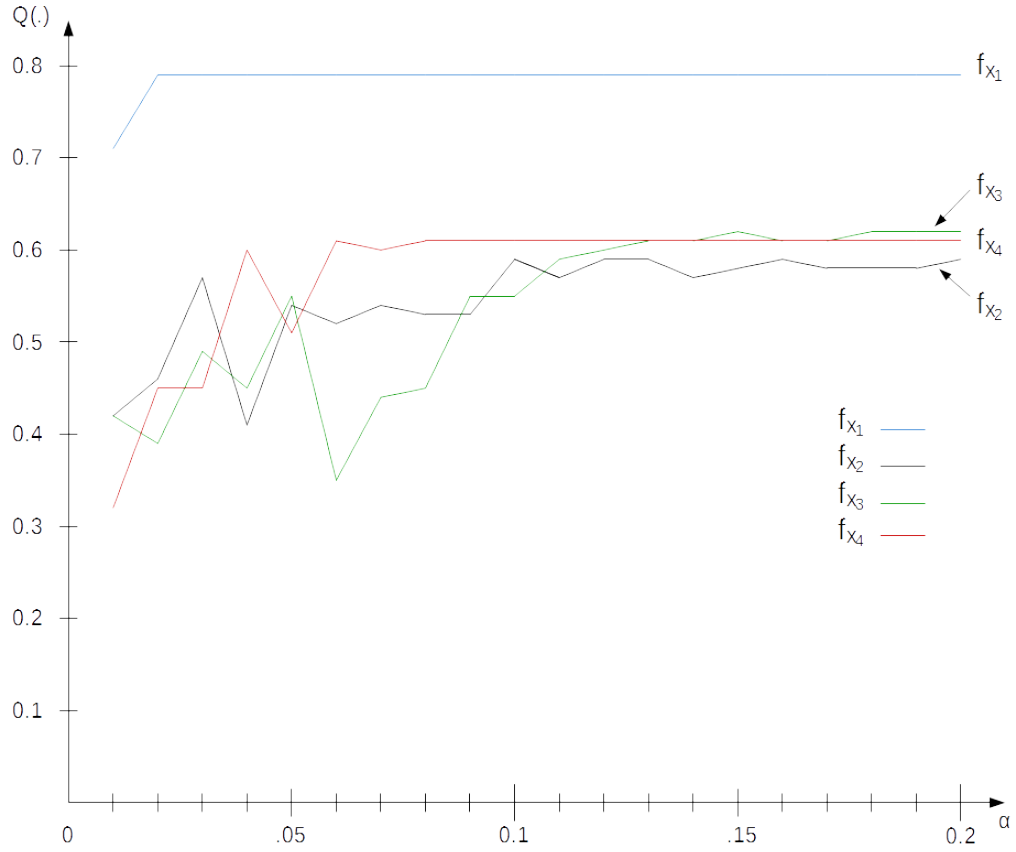

Diagramm 1: Verlauf der Qualitätsfunktionen für die Mengen X_1, X_2, X_3 und X_4 bezüglich Parameter α

Aus Diagramm 1 können folgende Schlussfolgerungen gezogen werden:

Schlussfolgerung 13 -

1. *Hier können wir die Frage "Wie sensibel ist der Parameter α in Algorithmus 4?" beantworten. Die Wahl des richtigen Wertes des Parameter α hat direkten Einfluss auf die Qualität $Q(.)$ des Algorithmus 4. Denn je mehr der Wert des Prameter α im richtigen Bereich liegt, desto stärker reduziert sich die negative Wirkung der Outlier-Punkte auf die Qualität $Q(.)$ des Algorithmus 4. Weiterhin sehen wir, dass die Qualitätsfunktion für Menge X_1 sich stabil verhält, weil es nur wenige Outlier-Punkte in der Menge X_1 gibt. Die Qualitätsfunktionen für die Mengen X_2, X_3 und X_4 variieren stark für $\alpha < 0.1$. Für $\alpha > 0.1$ ändern sich die Werte nur wenig.*
2. *Aus dem Diagramm ersehen wir, dass die Qualität $Q(.)$ des Algorithmus 4 sich, wenn der Wert des Parameter α im richtigen Bereich liegt, ganz gering verändert. Das heißt, wenn der Wert des Parameter α richtig gewählt wird, verhält sich der Algorithmus 4 stabiler in Bezug auf die Qualität $Q(.)$.*
3. *Je mehr sich die Anzahl der Cluster erhöht oder je mehr die Cluster nebeneinander liegen, desto schwieriger und sensibler wird die Bestimmung des Parameter α.*
4. *Wir sehen den höchsten Wert der Qualität $Q(.)$ des Algorithmus 4 für die Datenmenge X_1. Das ist nachvollziehbar, weil die Datenmenge X_1 die minimale Anzahl der Outlier-Punkte hat.*

Im Folgenden stellen wir die Ergebnisse unserer Untersuchung in Bezug auf alle vier Datenmengen X_1, X_2, X_3, X_4 und unseren Algorithmus 4 bildlich dar. Die Zentren der Cluster sind als große schwarze Punkte dargestellt.

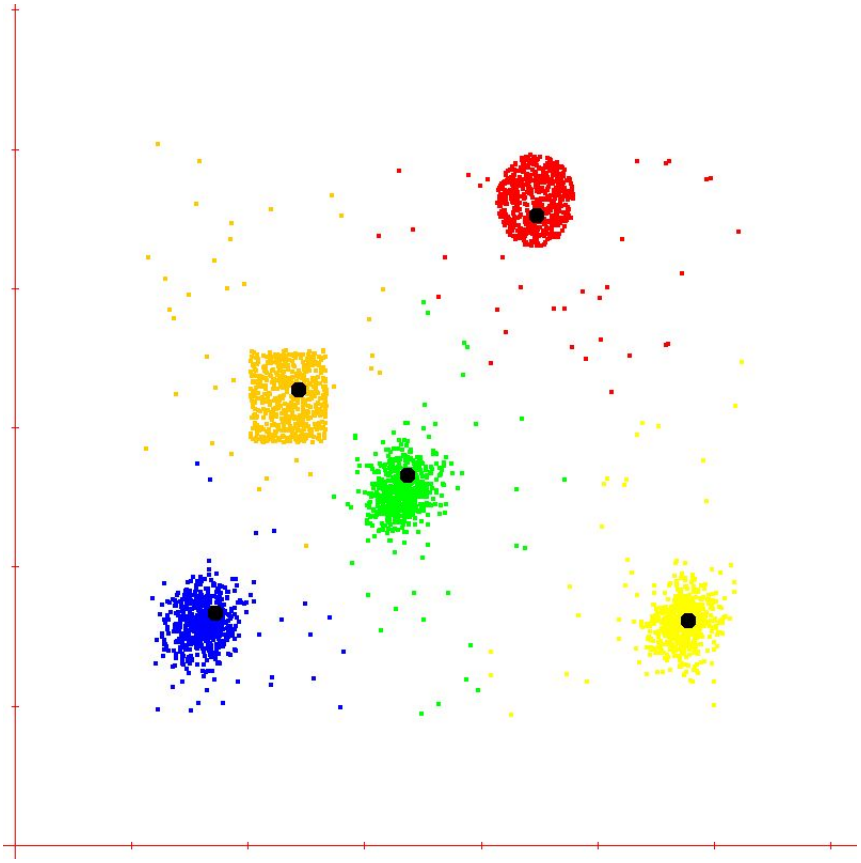

Abbildung 2: Datenmenge X_1, Der Algorithmus 4 wurde mit den Parametern $K = 5$, $\alpha = 0.15$ aufgerufen. Nach 5 While-Schleifen mit 18 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert.

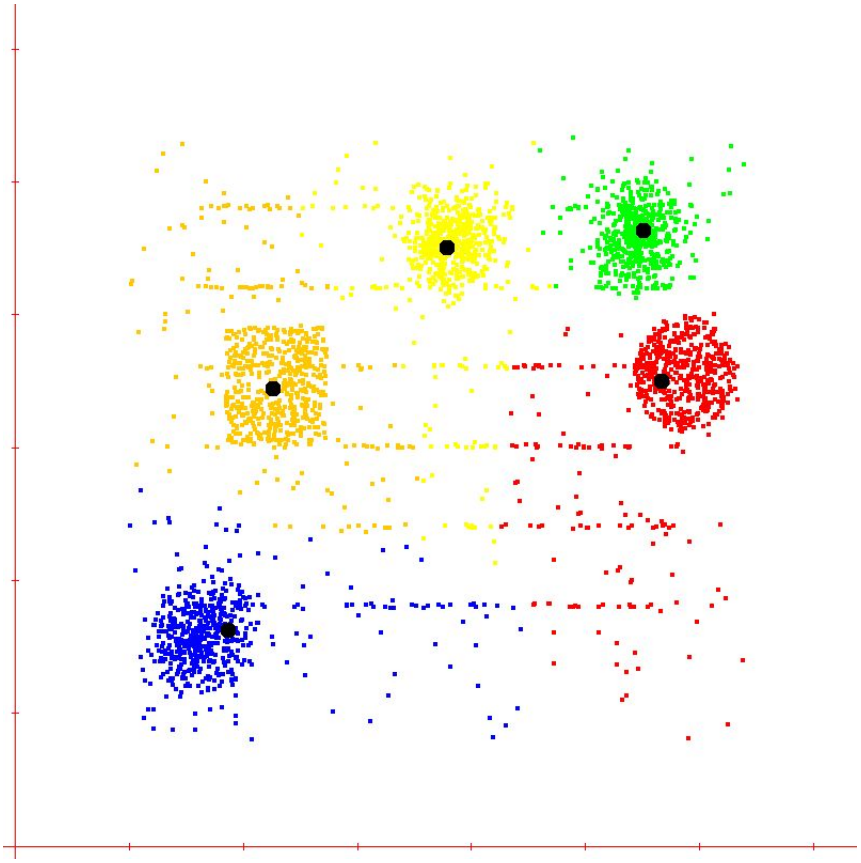

Abbildung 3: Datenmenge X_2, Der Algorithmus 4 wurde mit den Parametern $K = 5$, $\alpha = 0.2$ aufgerufen. Nach 8 While-Schleifen mit 21 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert.

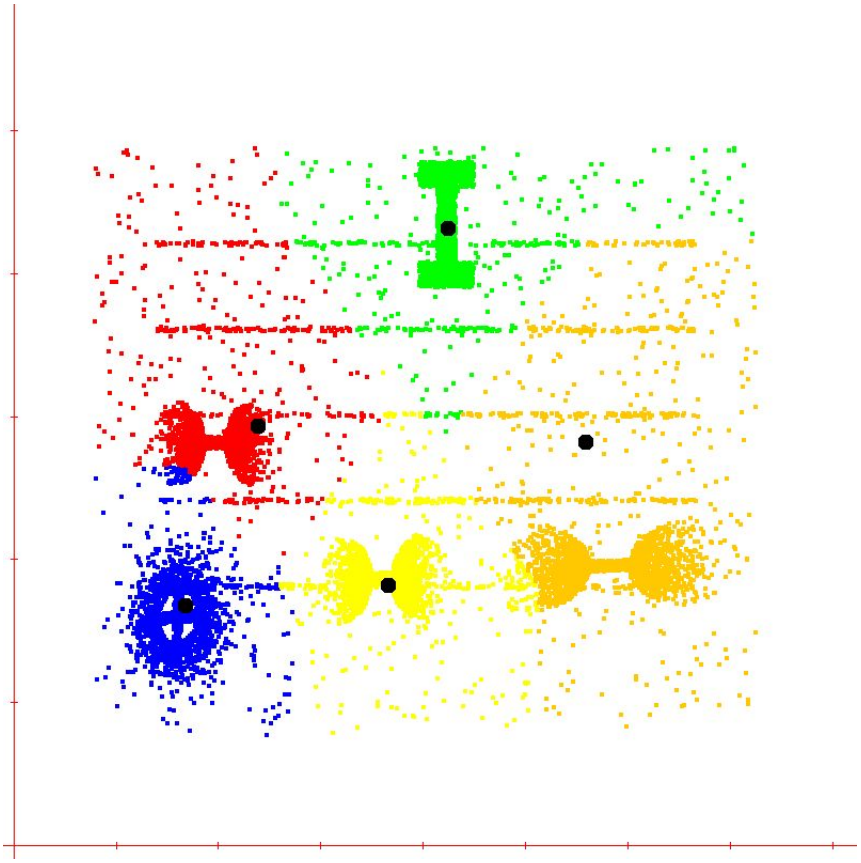

Abbildung 4: Datenmenge X_3, Der Algorithmus 4 wurde mit Parameter $K = 5$, $\alpha = 0.16$ aufgerufen. Nach 24 While-Schleifen mit 95 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert. Wir sehen in Abb. 4, dass ein Zentrum nicht genau in der Mitte des Clusters liegt. Die Outlier-Punkte und die Lage der Cluster sind der Grund für diese Anomalie.

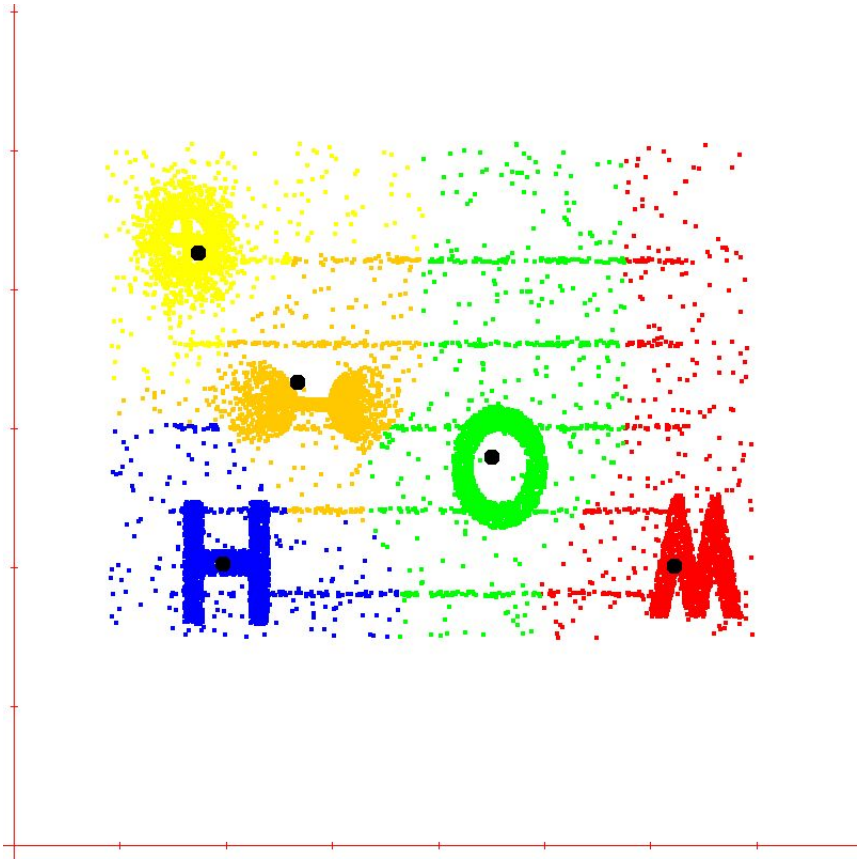

Abbildung 5: Datenmenge X_4, Der Algorithmus 4 wurde mit Parameter $K = 5$, $\alpha = 0.1$ aufgerufen. Nach 9 While-Schleifen mit 37 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert.

Weiterhin stellen wir die Ergebnisse unserer Untersuchung im Bezug auf die Datenmengen 2d-4c-no0, 2d-10c-no0 und 2d-20c-no0 und unseren Algorithmus 4 bildlich dar. Die Zentren der Cluster sind als schwarze Punkte dargestellt.

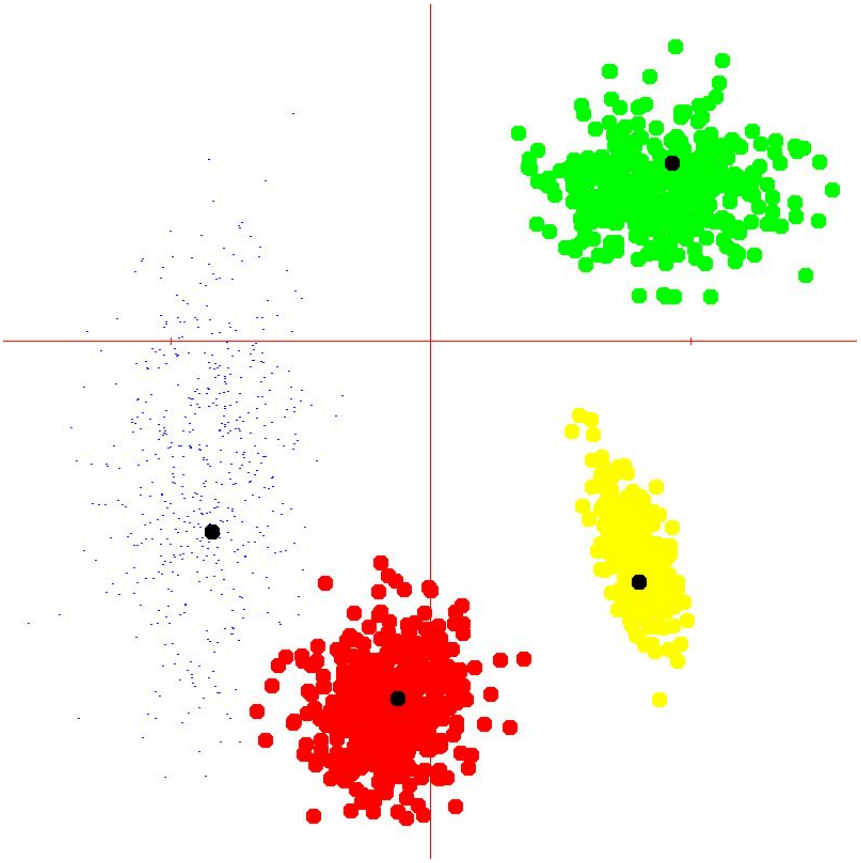

Abbildung 6: Datenmenge 2d-4c-no0 mit 1572 Datensätzen, der Algorithmus 4 wurde mit Parametern $K = 4$, $\alpha = 0.1$ aufgerufen. Nach 14 While-Schleifen mit 44 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert.

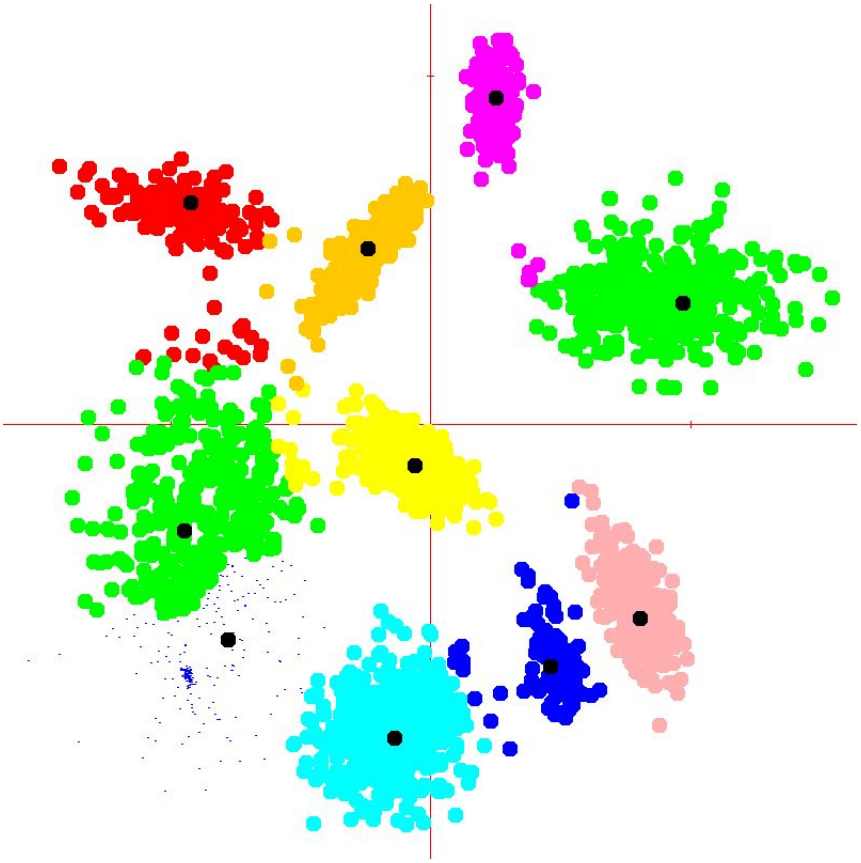

Abbildung 7: Datenmenge 2d-10c-no0 mit 2972 Datensätzen, der Algorithmus 4 wurde mit Parametern $K = 10$, $\alpha = 0.1$ aufgerufen. Nach 34 While-Schleifen mit 153 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert.

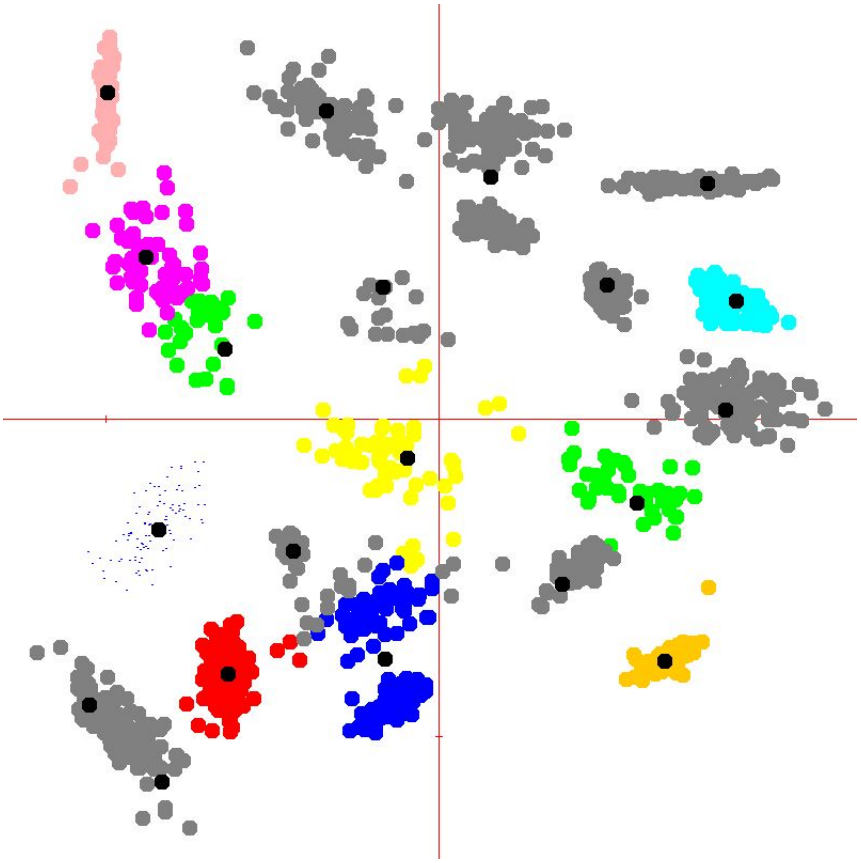

Abbildung 8: Datenmenge 2d-20c-no0 mit 1517 Datensätzen, der Algorithmus 4 wurde mit Parametern $K = 20$, $\alpha = 0.009$ aufgerufen. Nach 43 While-Schleifen mit 342 Anfangsmittelpunkten in der Hauptphase hat der Algorithmus terminiert.

6. Zusammenfassung

In diesem Kapitel möchten wir einen Überblick über das geben, was in dieser Arbeit gemacht wurde und verweisen auf weitere mögliche Forschungsziele, die sich aus dieser Arbeit ergeben.

Wir haben zwei erste Nachbarschaftsalgorithmen (Algorithmus 1 und 2) entwickelt. Algorithmus 1 ist komplett parallel durchführbar, während Algorithmus 2 sequentiell ist. Durch die Analyse des Verhaltens der beiden Algorithmen haben wir gezeigt, dass beide mit der Datenmenge skalieren. Um für das K-nächste Nachbarnproblem einen Algorithmus (Algorithmus 3) zu entwickeln, haben wir Algorithmus 2 ergänzt. Daraufhin haben wir gezeigt, dass Algorithmus 3 ebenso skalierbar ist. Die Kernidee bei Algorithmus 3 war es, Algorithmus 2 K-mal mit entspechenden Nachbarschaftskandidaten aufzurufen. Für diesen Zweck haben wir die zwei Mengen W und *minimal* verwendet: in der Menge W waren alle Kandidaten enthalten. Davon haben wir dann sukzessiv die i-ten Nachbarschaftskandidaten $i = 1, 2, \ldots, k$ aus der Menge W entfernt und haben sie der Menge *minimal* hinzugefügt. Anschließend haben wir Algorithmus 2 zum Zweck der Bestimmung der i-ten Nachbarschaftsmenge mit den entsprechenden Nachbarschaftskandidaten aus der Menge *minimal* aufgerufen. Ein Optimierungsschritt zu Algorithmus 3 könnte die Überlegung sein, dass wir erst für alle Datensätze $x \in X$ ihre erste Nachbarschaftsmenge $N_1(x)$ bestimmen. Dann, anstelle, dass wir sukzessiv die i-ten Nachbarschaftskandidaten $i = 2, \ldots, k$ von der Menge W entfernen und in die Menge *minimal* einfügen, wählen wir einen beliebigen Punkt $y \in N_{i-1}(x)$. Dann fügen wir von seiner ersten Nachbarschaftsmenge nur einen beliebigen Punkt $x^+ \in N_1(y)$ als i-ten Nachbarschaftskandidaten für den Punkt x in die Menge *minimal* ein. Damit wird die Menge W nicht mehr benötigt und wir haben eventuell weniger Distanzberechnungen. Weiterhin haben wir zwei skalierbare Clusteralgorithmen (Algorithmus 4 und 5) entwickelt. Wir haben in Abschnitt 5.4 gezeigt, dass unsere Clusteralgorithmen konzeptuell besser als die PAM- und CLARA-Algorithmen sind. Diese Algorithmen basieren auf der Berechnung der Distanz zwischen allen Datensätzen der Datenmenge und einer mehrmaligen Zuordnung der Datensätze zu den nächsten Cluster, um die optimalen Medoide zu finden. Aber bei unseren Clusteralgorithmen berechnen wir nur die erste Nachbarschaftsmenge und in der Endphase ordnen wir nur einmal die Datensätze den künstlich erzeugten nächsten Zentren der Cluster zu.

Eine Neuerung unserer Clusteralgorithmen ist die Möglichkeit der starken Reduzierung der Anzahl der Datensätze, ohne damit die Struktur der Cluster der Eingabemenge verändern zu müssen. Dafür wurde ein Parameter α definiert. Wir haben in Abschnitt 5.5 bei der empirischen Untersuchung gezeigt, dass unsere Clusteralgorithmen mit der

Datenmenge skalieren und die hier vorgestellten Verfahren qualitativ mindestens ebenbürtig zu PAM und CLARA sind.

A. Anhang

A.1. Maschinenmodelle

Wir haben verschiedene Rechnerarchitekturen [66], die nach der Anzahl der vorhandenen Befehls- und Datenströme unterteilt sind. Es gibt vier Klassen: SISD, SIMD, MISD und MIMD.

SISD (single instruction, single data):

Ein Prozessor arbeitet ein Programm sequentiell ab.

SIMD (single instruction, multiple data):

Mehrere Prozessoren arbeiten ein Programm im gemeinsamen Speicher ab. Jede Instruktion wird synchron (gleichzeitig) auf unterschiedlichen Daten ausgeführt. Die Synchronisierung erfolgt mittels eines globalen Takts.

MIMD (multiple instruction, multiple data):

Mehrere Prozessoren arbeiten unterschiedliche Instruktionen mit verschiedenen Daten asynchron (nicht gleichzeitig) im gemeinsamen Speicher ab.

MISD (Multiple Instruction, Single Data):

Mehrere Prozessoren arbeiten unterschiedliche Instruktionen auf den gleichen Daten ab.

A.2. PRAM

Einen SIMD-Rechner mit unbeschränkter Anzahl von Prozessoren und gemeinsamen Spreicher bezeichnet man als PRAM (Parallel Random Access Machine) [67]. Vergleichbar der RAM [68] verfügt die PRAM über eine abzählbar unendliche Menge von einzeln adressierbaren Speicherzellen. Jede Speicherzelle kann eine beliebig große reelle Zahl aufnehmen. Die Prozessoren sind von 0 bis n-1 durchnumeriert. Jeder Prozessor kennt seine Nummer und kann auf jede Speicherzelle in konstanter Zeit zugreifen. Die von einem Prozessor ausgeführten Instruktionen entsprechen dem Befehlssatz einer RAM. Typische Befehle sind:

x_i : x steht für den Variablennamen und Index i für die Adresse der Variable im Speicher.

$x_{i \to i+1}$: Index (Adresse) i wird um eins erhöht, dies ermöglicht den Zugriff auf die nächste Speicherzelle.

$x_i = x_j \otimes x_k$, $\otimes \in \{+, -, *, /\}$: arithmetische Operationen.

$x_{x_i} = x_j$: Speichern in indirekt adressierten Speicherzellen.

$x_i = x_{x_j}$: Laden von indirekt adressierten Speicherzellen.

$if(x_i \otimes x_j)$ then : bedingte Verzweigungen $\otimes \in \{<, >, =\}$.

$for\, i = r\, to\, s\, do$: for-Schleifen.

$while\, i \neq r\, do$: while-Schleifen.

$break$: bricht den aktuellen for- oder while-Schleifendurchlauf ab.

Elementare Operationen kosten unabhängig von der Größe der referierten Operanden konstante Zeit (uniformes Kostenmaß). Alle Prozessoren arbeiten synchron dasselbe Programm ab.

Man unterscheidet vier Varianten der PRAM bzgl. der Gleichzeitigkeit von Lese- und Schreiboperationen:

EREW: exclusive read, exclusive write

CREW: concurrent read, exclusive write

ERCW: exclusive read, concurrent write

CRCW: concurrent read, concurrent write

Wir benutzen zur Analyse unseres Algorithmus 1 das (CREW)-PRAM Modell. Um exclusives Schreiben durch Prozessoren zu erreichen, benutzen wir Semaphore [69]. Ein Semaphor ist eine Datenstruktur mit einer Initialisierungsoperation $Init(.)$ und den beiden Nutzungsoperationen $Wait()$ und $Release()$. Die Datenstruktur besteht aus einem Zähler und einer Warteschlange für die Aufnahme der Nummer blockierter Prozessoren.

Semaphor{

1- int z

2- Queue q // Warteschlange

}

$Init(Semaphor\, s)${

1- $s.z = 0$

2- s.$q = empty()$

}

$Wait()${

1- $z = z - 1$

2- $if(z < 0)${

3- $selbst\_blockieren(q)$ // Blockieren des Prozessors, Einreihung in Warteschlange

}// end if

}

$Release()${

1- $s = s + 1$

2- $if(s \leq 0)${

3- $Einen\_entblocken(q)$ // Entblockieren eines Prozessors aus der Warteschlange

}// end if
}
Für ein besseres Verständnis unseres (CREW)-PRAM Pseudocodes definieren wir noch eine weitere Datenstruktur Punkt wie folgt:

Punkt{

int index

double distanz

}

Soll eine Instruktion durch einen bestimmten Prozessor P_i ausgeführt werden, benutzen wir folgende Schreibweise:

P_i : *Instruktion*. Hiermit werden alle anderen Prozessoren P_j, $j \neq i$ blockiert und müssen auf Prozessor P_i warten. Wenn Prozessor P_i seine Instruktion ausgeführt hat, werden alle Prozessoren mit dem nächsten globalen Takt die nächste Instruktion abarbeiten.

Es ist auch möglich, dass alle Prozessoren gleichzeitig eine For-Schleife mit unterschiedlichem Parameter r (for i=0 to r) abarbeiten. Hierfür benutzen wir die Funktion $Versammlung()$, die wieder die Arbeit der Prozessoren synchronisiert.

$Versammlung()$:= die Prozessoren, die zuerst diese Instruktion ausführen, müssen warten, bis die anderen Prozessoren auch diese Instruktion ausführen. Dann wird die nächste Instruktion gemeinsam durch alle Prozessoren abgearbeitet.

A.3. (CREW)-PRAM Pseudocode

In diesem Abschnitt schreiben wir (CREW)-PRAM Pseudocode für die wichtigen Teile von Algorithmus 1. Wir analysieren die Laufzeit $T(.)$ und den Speicherverbrauch $S(.)$ jeder Funktion getrennt und benutzen dafür die $O(.)$ Notation [70]. Hierfür setzen wir die Laufzeit jeder Instruktion als t_i und den Speicherverbrauch als s_i, wobei sich i auf die Instruktionsnummer bezieht. Für unsere Analyse gehen wir davon aus, dass die Matrizen M (Def. 3) und $X = \{x_1, x_2, \ldots, x_N\}$, beide zweidimensionale Matrizen, als globale Variablen in einem gemeinsamen Speicher abgespeichert sind. Da die Distanzfunktion anwendungsabhängig ist, setzen wir den sequentiellen Aufwand zur Berechnung der Distanz zwischen zwei Punkten x_s und x_t gleich $A(.)$. Wenn wir den Algorithmus 1 mit n Prozessoren ausführen, müssen wir die Arbeit soweit wie möglich gleich zwischen den Prozessoren verteilen. Hierfür übergeben wir jedem Prozessor P_i $\frac{d}{n}$ Zeilen der $(d \times N)$-Matrix M. Weiterhin, um die Menge der ersten Nachbarn $N_1(x)$ für den Punkt $x \in X$ zu bestimmen, berechnen wir beim Algorithmus 1 in Instruktion 2 die Distanz zwischen $x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ und Punkten $x_t : x_t \in M_{(j,i_j \pm 1)}$, $j = 1, 2, \ldots, d$ und in Instruktion 3/item 1/Option b die Distanz zwischen x und Intervallpunkten $x^* = (a_1^*, a_2^*, \ldots, a_d^*) \in X$. Die Wahrscheinlichkeit, dass die Attributswerte der Punkte x_t und x^* in Matrix M in Dimension i im Falle x_t

in Spalten $i_j \pm 1$ und im Falle x^* innerhalb des Intervalls vorkommen, ist gleich $\frac{2}{N}$ und $\frac{In_i}{N}$, wobei In_i die Anzahl der Intervallpunkte in Dimension i ist.
Wenn die Attributswerte der Punkte x_t und x^* gleichzeitig auch in Dimension j vorkommen, dann ist diese Wahrscheinlichkeit gleich $(\frac{2}{N})^2$ und $\frac{In_i}{N} \times \frac{In_j}{N}$. Diese Wahrscheinlichkeit ist sehr klein und wird für noch weitere Dimensionen kleiner. Wir benutzen für unsere Analyse ein (CREW)-PRAM Modell, d. h. wir müssen beim gleichzeitigen Schreiben der Prozessoren auf eine globale Variable Schreibkonflikte vermeiden, indem wir sequentielles Schreiben organisieren. Wenn wir vermeiden wollen, dass zum Beispiel die Distanz zwischen den Punkten x und x_t mehrmals durch Prozessoren berechnet wird, können wir eine globale boolische Variable für jeden Punkt $x_t \in X$ definieren und folgenden Pseudocode schreiben:

$if(x_t.bool = false)\{$

- setze $x_t.bool = true$

- berechne $dist(x, x_t)$

$\}$

Aber hier müssen wir die Schreibkonflikte zwischen Prozessoren, die gleichzeitig in die if-Anweisung eintreten, vermeiden, indem wir versuchen, nur einen Prozessor Schreibzugriff zu erlauben und die anderen zu blockieren. Obwohl, wie wir oben gesehen haben, die Wahrscheinlichkeit, dass mehrere Prozessoren gleichzeitig in die if-Anweisung eintreten, sehr gering ist, können wir nicht einschätzen, wie viele Prozessoren dies tatsächlich tun. Da wir bei unserer Laufzeitsanalyse die worst-case Analyse benutzen, müssen wir theoretisch davon ausgehen, dass für jede Distanzberechnug alle anderen Prozessoren blockieren müssen. Dies bringt aber für parallele Verarbeitung keine Effizienz mehr. Deswegen definieren wir globale Variablen und vermeiden eventuelle Schreibkonflikte durch Semaphore, wenn wir damit die Effizienz der parallelen Verarbeitung nicht verlieren.

Um Schreibkonflikte gering zu halten, führen wir globale Variablen g VAR (für alle Prozessoren) und private Variablen p VAR (für jeden Prozessor separat) ein.

(CREW)-PRAM 1 - *Instruktion 2*

Verwendet werden $n \leq d$ Prozessoren P_0, P_1, ..., P_{n-1}.

Aufgabe: Berechne parallel den Abstand zwischen
$x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ *und Punkten*
$x_t : x_t \in M_{(j,i_j \pm 1)}$ $(Def. 5)$, $j = 1, 2, \ldots, d$ *in Matrix M. Bestimme der Menge W der Punkte, die unter diesen Punkten den kleinsten Abstand zu x haben. Die Menge W könnte auch nur einen Punkt beinhalten.*

$W := \{x_M \mid x_M \in M_{(j,i_j \pm 1)} \, und \, dist(x, x_M) \, ist \, minimal\}$.

Um die Lesbarkeit unserers Pseudocodes zu verbessern, definieren wir eine for-Schleife, mit der wir auf die Punkte $x_t \in M_{(j,i_j \pm 1)}$ zugereifen können:

for $x_t \in M_{(j,k)}$, $k = i_j + 1, i_j - 1$

p VAR p_{ij}*: of PUNKT // $i = 0, 1, \ldots, n-1$, $j = 0, 1, 2, \ldots$*

p VAR min_i*: of INT* $i = 0, 1, \ldots, n-1$
p VAR J_i*: of INT* $i = 0, 1, \ldots, n-1$
p VAR W_{ij} *: of PUNKT* $i = 0, 1, \ldots, n-1,\ j = 0, 1, 2, \ldots$
g VAR MIN : of INT
g VAR W_j *: of PUNKT* $j = 0, 1, 2, \ldots$
VAR s : Semaphor
BEGIN
1- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*
  1- P_i : $Init(i)$
END
2- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*
  1- P_i : $Dist(i)$
END
3- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*
  1- P_i : $Min(i)$
END
4- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*
  1- P_i : $W(i)$
END
5- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*
  1- P_i : $Ergebnis(i)$
END
END
Sei $J^1 := \max_{i=0,\ldots,n-1} J_i$ *und seien im Folgenden* t_i *die Laufzeit und* s_i *der Speicherverbrauch (wie oben in A.3 erklärt) jeder Instruktion. Der Index i bezieht sich auf die Instruktionsnummer.*
$Init(i)$ *{*
1- P_0 : $\{Init(s), MIN = -1\}$ *//* $t_1 = O(1)$, $s_1 = O(1)$
2- $min_i = -1$ *//* $t_2 = O(1)$, $s_2 = O(n)$
3- $J_i = 0$ *//* $t_3 = O(1)$, $s_3 = O(n)$
$T(Init(i)) = O(1)$, $S(Init(i)) = O(n)$
}
$Dist(i)$*{*
1- for $j = (f \times i) + 0, (f \times i) + 1$ *to* $(f \times i) + (f-1)$*{ // wobei* $f = \frac{d}{n}$
2- for $x_t \in M_{(j,k)}$, $k = i_j + 1, i_j - 1$*{*
3- $p_{iJ_i}.dist = dist(x, x_t)$ *//* $t_3 = O(J^1 \times A(.))$, $s_3 = O(J^1 \times n)$
4- $p_{iJ_i}.index = t$ *//* $t_4 = J^1 \times O(1) = O(J^1)$, $s_4 = O(J^1 \times n)$

5- $if(min_i < p_{iJ_i}.dist)\, min_i = p_{iJ_i}.dist$ *//* $t_5 = J^1 \times O(1) = O(J^1)$, $s_5 = O(J^1 \times n)$
6- $J_i = J_i + 1$ *//* $t_6 = J^1 \times O(1) = O(J^1)$, $s_6 = O(n)$
}// end for
}// end for
7- $Versammlung()$
Hier gilt $f < J_i \leq J^1$.
$T(Dist(i)) = O(J^1 \times A(.))$, $S(Dist(i)) = O(J^1 \times n)$
}
$Min(i)\{$
// höchstens n Prozessoren können hier eintreten.
1- $s.Wait()$ *// hier werden Prozessoren blockiert.*
2- $s.Release()$ *// Um Schreibkonflikte bei globalen Variablen zu vermeiden, werden hier durch das System die blockierten Prozessoren einer nach dem anderen entblockiert.*
3- $if(min_i < MIN)\, MIN = min_i$ *//* $t_3 = n \times O(1) = O(n)$, $s_3 = O(n)$
4- $Versammlung()$
$T(Min(i)) = O(n)$, $S(Min(i)) = O(n)$
}
$W(i)\{$
1- $if(min_i = MIN)\{$ *//* $t_1 = O(1)$, $s_1 = O(n)$
2– for k=0 to J_i *do{ //* $t_2 = O(J^1)$
3– $if(p_{ik}.dist = MIN)\{$ *//* $t_3 = J^1 \times O(1) = O(J^1)$, $s_3 = O(J^1 \times n)$
4- $W_{ij} = p_{ik}$ *//* $t_4 = |W_{ij}| \times O(1) = O(|W_{ij}|)$, $s_4 = O(J^1 \times n)$
5- $W_{ij \to ij+1}$*//* $t_5 = |W_{ij}| \times O(1) = O(|W_{ij}|)$, $s_5 = O(|W_{ij}|)$
Hier gilt: $|W_{ij}| = j + 1$*, Anzahl der gespeicherten Punkte in* W_{ij}
}// end $if(p_{ik}.dist = MIN)$
$T(for) = O(\max(t_3, t_4, t_5)) = O(J^1)$
$S(for) = O(\max(s_3, s_4, s_5)) = O(J^1 \times n)$
}//end for
}//end if $if(min_i = MIN)$
$T(W(i)) = O(J^1)$, $S(W(i)) = O(J^1 \times n)$
}
$Ergebnis(i)\{$
1- $if(|W_{ij}| > 0)\{$ *//* $t_1 = O(1)$
// höchstens n Prozessoren können hier eintreten.
2- $s.Wait()$
3- $s.Release()$

4- for $k = 0$ *to* $|W_{ij}|\{$ *//* $t_4 = O(|W_{ij}|)$

6- $W_j = W_{ik}$ *//* $t_6 = |W_{ij}| \times O(1) = O(|W_{ij}|)$, $s_6 = O(J^1)$

7- $W_{j \to j+1}$ *//* $t_7 = O(J^1)$, $s_7 = O(J^1)$

$T(for) = O(J^1)$, $S(for) = O(J^1)$

}// end for $k = 0$ *to* $|W_{ij}|$

}// end $if(|W_{ij}| > 0)$

Hier gilt:

$|W_{0j}| + |W_{1j}| + \ldots + |W_{n-1j}| < c \times J^1 = O(J^1)$, $c \in \mathbb{N}^+ = \{1, 2, 3, \ldots\}$. *Das lässt den folgenden Schluss zu:* $|W_j| < c \times J^1 = O(J^1)$, *wobei* J^1 *die maximale Anzahl der berechneten Distanzen durch einen Prozessor ist.*

$T(Ergebnis(i)) = O(n)$, $S(Ergebnis(i)) = O(J^1)$

}

Um bessere Optimalität bei der parallelen Berechnung zu erreichen, setzen wir n *größer als* J *und kleiner/gleich* d, $J < n \leq d$, *wobei* $J := \max(J^1, J^2)$ *ist (siehe PRAM 2 und 3).*

Wir wissen, dass die sequentielle Aufwandsfunktion $A(.)$ *abhängig von der Anzahl der Dimensionen* d *ist. Hierfür müssen wir* d*-mal lokale Abstände* $dist_j(x, x_t)$ *berechnen. Unter der Annahme, dass der Aufwand zur Berechnung des lokalen Abstands konstant ist, ist die sequentielle Aufwandsfunktion mindestens gleich* d. *Hiermit gilt* $A(.) \geq d$. *Daraus kann* $A(.) > n$ *geschlossen werden.*

$T(PRAM_1) = O(\max(J^1, n, J^1 \times A(.))) = O(J^1 \times A(.))$

$S(PRAM_1) = O(J^1 \times n)$

(CREW)-PRAM 2 - *Instruktion 3/item 1/Option a*

Verwendet werden $n \leq d$ *Prozessoren* P_0, P_1, $\ldots$, P_{n-1}.

Aufgabe: prüfe ob $x_M \in W$ *in jeder Dimension in den Spalten* $i_j \pm 1$ *vorkommt und die Eigenschaft* min_j *(Def. 9 item 1) hat. Hier ist*
$x = (a_1, a_2, \ldots, a_d) \in X$, $[x] = (i_1, i_2, \ldots, i_d)^T$ *wie in (CREW)-PRAM 1.*

p VAR b_i*: of BOOLEAN* $i = 0, 1, \ldots, n-1$

p VAR J_i*: of INT* $i = 0, 1, \ldots, n-1$

p VAR v_i*: of INT* $i = 0, 1, \ldots, n-1$

p VAR min_i*: of INT* $i = 0, 1, \ldots, n-1$

g VAR b_1, b_2 *: of BOOLEAN*

VAR s : Semaphor

BEGIN

1- Init()

2- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- P_i : $prüfe_1(i)$

END

3- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- $P_i : if(b_1)\, prüfe_2(i)$

2- $P_0 : if(b_2)\, N_1(x) = \{x_M\}$

END

END

Sei J^1 *definiert wie PRAM 1.*

$Init()\{$

1- $P_0 : \{b_1 = false,\, b_2 = false, Init(s)\}$ *//* $t_1 = O(1),\, s_1 = O(1)$

$T(Init()) = O(1),\, S(Init()) = O(1)$

$\}$

$prüfe_1(i)\{$

1- for $j = (f \times i) + 0,\, (f \times i) + 1$ *to* $(f \times i) + (f-1)\{$ *// wobei* $f = \frac{d}{n}$

2- $b_i = false$ *//* $t_2 = f \times O(1) = O(f),\, s_2 = O(n)$

3- for $x_t \in M_{(j,k)},\, k = i_j + 1, i_j - 1\{$

4- $J_i = J_i + 1$ *//* $t_4 = J^1 \times O(1) = O(J^1),\, s_4 = O(n)$

5- $if(x_M = x_t)\{$ *//* $t_5 = J^1 \times O(1) = O(J^1),\, s_5 = O(1)$

6- $b_i = true$ *//* $t_6 = f \times O(1) = O(f),\, s_6 = O(n)$

}// end $if(x_M = x_t)$

}// end for $x_t \in M_{(j,k)},\, k = i_j + 1, i_j - 1$

7- $if(b_i = false)\, break$ *//* $t_7 = f \times O(1) = O(f),\, s_7 = O(n)$

}// end for $j = (f \times i) + 0,\, (f \times i) + 1$ *to* $(f \times i) + (f-1)$

8- $Versammlung()$

9- $s.Wait()$

10– $s.Release()$

11- $b_1 = b_i$ *//* $t_{11} = n \times O(1) = O(n),\, s_{11} = O(n)$

12- $if(b_1)\, Versammlung()$

Hier gilt $f < J_i \leq J^1 < n$, *wobei* J^1 *die maximale Anzahl der berechneten Distanzen durch einen Prozessor ist.*

$T(prüfe_1(i)) = O(\max(f, J^1, n)) = O(n)$

$S(prüfe_1(i)) = O(n)$

$\}$

$prüfe_2(i)\{$

1- for $j = (f \times i) + 0,\, (f \times i) + 1$ *to* $(f \times i) + (f-1)\{$ *// wobei* $f = \frac{d}{n}$

// Aufwand des lokalen Abstands $dist_j(x, x_M)$ *ist gleich* $\frac{A()}{d}$

2- $min_i = dist_j(x, x_M)$ *//* $t_2 = f \times \frac{A()}{d} = O(f \times \frac{A()}{d}),\, s_2 = O(n)$

3- $b_i = true$ *//* $t_3 = f \times O(1) = O(f)$, $s_3 = O(n)$
4- for $x_t \in M_{(j,k)}$, $k = i_j + 1, i_j - 1\{$
5- $v_i = dist_j(x, x_t)$ *//* $t_5 = J^1 \times \frac{A()}{d} = O(J^1 \times \frac{A()}{d})$, $s_5 = O(n)$
6- $if(min_i > v_i)\{$ *//* $t_6 = J^1 \times O(1) = O(J^1)$, $s_6 = O(n)$
7- $b_i = false$ *//* $t_7 = f \times O(1) = O(f)$, $s_7 = O(n)$
}// end $if(min_i > v_i)$
}// end for $x_t \in M_{(j,k)}$, $k = i_j + 1, i_j - 1$
8- $if(b_i = false)\, break$ *//* $t_8 = f \times O(1) = O(f)$, $s_8 = O(n)$
}// end for $j = (f \times i) + 0$, $(f \times i) + 1$ *to* $(f \times i) + (f - 1)$
9- $Versammlung()$
10- $s.Wait()$
11– $s.Release()$
12- $b_2 = b_i$ *//* $t_{12} = n \times O(1) = O(n)$, $s_{12} = O(n)$
Hier gilt $f < J_i \leq J^1 < n$ *und* $\frac{A()}{d} \geq O(1)$.
$T(prüfe_2(i)) = O(\max(J^1 \times \frac{A()}{d}, n))$
$S(prüfe_2(i)) = O(n)$
}
$T(PRAM_2) = O(\max(J^1 \times \frac{A()}{d}, n))$
$S(PRAM_2) = O(n)$

(CREW)-PRAM 3 - *Instruktion 3/item 1/Option b*
Verwendet werden $n \leq d$ *Prozessoren* P_0, P_1, $\ldots$, P_{n-1}.
Aufgabe: Seien $j_1, j_2, \ldots, j_s \in \{1, 2, \ldots, d\}$, $1 \leq s \leq d$ *Dimensionen, in denen* $x_M \in W$ *die Eigenschaft* $!min_{j_b}, b = 1, 2, \ldots, s$ *(Def. 9 item 2) hat. Bestimme die Distanz zwischen* x *und den Intervallpunkten* $x^l \in M_{(j_b, v^l_{j_b})}$ *und* $x^r \in M_{(j_b, v^r_{j_b})}$, *wobei* $v^l_{j_b}$ *und* $v^r_{j_b}$ *sind definiert als:*
$v^l_{j_b} : p^l_{j_b} \leq v^l_{j_b} \leq p^-_{j_b}$
$v^r_{j_b} : p^+_{j_b} \leq v^r_{j_b} \leq p^r_{j_b}$
Hier sind wie in Def. 9 item 3 $[p^l_{j_b}, p^-_{j_b}]$, $[p^+_{j_b}, p^r_{j_b}]$ *linke und rechte Intervalle in Matrix* M. *Sei in folgenden o.B.d.A.* x^* *eine Variable, die wir anstelle von* x^r *und* x^l *benutzen. Wir definieren* $K(x)$ *und* $G(x)$ *als:*
$K(x) := \{x^* \,|\, dist(x, x^*) < dist(x, x_M)\}$
$G(x) := \{x^* \,|\, dist(x, x^*) = dist(x, x_M)\}$
Dann ergeben sich folgende Situationen:
- Falls $K(x) = \emptyset$ *und* $G(x) = \emptyset$, *dann ist es* $N_1(x) = \{x_M\}$.
- Falls $K(x) = \emptyset$ *und* $G(x) \neq \emptyset$, *dann ist es* $N_1(x) = \{x_M\} \bigcup G(x)$.

- Für $K(x) \neq \emptyset$ sortieren wir die Abstände von diesen Punkten zu x und Punkte mit dem kleinsten gleichen Abstand zu x bilden die erste Nachbarschaftsmenge $N_1(x)$.

Um die Lesbarkeit unseres Pseudocodes zu verbessern, definieren wir eine for-Schleife, mit der wir auf die Indexmenge $\{J_1, J_2, \ldots, J_s\}$ zugereifen können:

for $m \in \{J_1, J_2, \ldots, J_s\}$.

p VAR k_{ij}*: of PUNKT // i* $= 0, 1, \ldots, n-1$, *j* $= 0, 1, 2, \ldots$

p VAR g_{ij}*: of PUNKT i* $= 0, 1, \ldots, n-1$, *j* $= 0, 1, 2, \ldots$

p VAR in_{ij}*: of INT i* $= 0, 1, \ldots, n-1$, *j* $= 0, 1, 2, \ldots$

p VAR p_i *: of PUNKT i* $= 0, 1, \ldots, n-1$

p VAR k_i*: of BOOLEAN i* $= 0, 1, \ldots, n-1$

p VAR g_i*: of BOOLEAN i* $= 0, 1, \ldots, n-1$

p VAR J_i*: of INT i* $= 0, 1, \ldots, n-1$

g VAR b_g, b_k *: of BOOLEAN*

g VAR K_j *: of PUNKT j* $= 0, 1, 2, \ldots$

g VAR G_j *: of PUNKT j* $= 0, 1, 2, \ldots$

g VAR d : of REAL

VAR s : Semaphor

BEGIN

1- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- P_i : $Init(i)$

END

2- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- P_i : $Index(i)$

END

3- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- P_i : $Dist(i)$

END

4- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- P_i : $G(i)$

END

5- For ALL P_i, $i = 0, 1, \ldots n-1$ *DO IN PARALLEL*

1- P_i : $K(i)$

END

6- $Ergebnis()$

END

$Init(i)$ *{*

1- P_0 : $\{Init(s), d = dist(x, x_M), b_g = false, b_k = false\}$ *//* $t_1 = A(.)$, $s_1 = O(1)$

2- $J_i = 0$ *//* $t_2 = O(1)$, $s_2 = O(n)$
3- $k_i = false$ *//* $t_3 = O(1)$, $s_3 = O(n)$
4- $g_i = false$ *//* $t_4 = O(1)$, $s_4 = O(n)$
$T(Init(i)) = A(.)$, $S(Init(i)) = O(n)$
}
Sei $J^2 := \max_{i=0,1,\ldots,n-1} J_i$
$Index(i)\{$
1- for $j = (f \times i) + 0, (f \times i) + 1$ *to* $(f \times i) + (f - 1)\{$ *// wobei* $f = \frac{d}{n}$
2- for $m \in \{J_1, J_2, \ldots, J_s\}\{$ *//* $t_2 = O(s)$
3- $if(m = j)\{$ *//* $t_3 = O(f)$
4- $in_{ij} = m$ *//* $t_4 = O(f)$, $s_4 = O(f \times n)$
5- $in_{ij \to ij+1}$ *//* $t_5 = O(f)$, $s_5 = O(f \times n)$
}// end $if(m = j)$
}// for $m \in \{J_1, J_2, \ldots, J_s\}$
}// end for $j = (f \times i) + 0, (f \times i) + 1$ *to* $(f \times i) + (f - 1)$
// Die Instruktionen 3 bis 5 werden höchstens f*-mal ausgeführt.*
$T(Index(i)) = O(f)$
$S(Index(i)) = O(f \times n)$
}
$Dist(i)\{$
1- for $j \in in_{ij}\{$ *//* $t_1 = O(f)$
2- for $x^* \in M_{(j,k)}$, $k = v_j^l, v_j^r\{$
3- $p_i.dist = dist(x, x^*)$ *//* $t_3 = O(J^2 \times A(.))$, $s_3 = O(n)$
4- $p_i.index = t$ *//* $t_4 = J^2 \times O(1) = O(J^2)$, $s_4 = O(n)$
5- $if(p_i.dist = d)\{$ *//* $t_5 = J^2 \times O(1) = O(J^2)$, $s_5 = O(n)$
6- $g_i = true$ *//* $t_6 = |g_{ij}| \times O(1) = O(|g_{ij}|)$, $s_6 = O(n)$
7- $g_{ij} = p_i$ *//* $t_7 = |g_{ij}| \times O(1) = O(|g_{ij}|)$, $s_7 = O(|g_{ij}|)$
8- $g_{ij \to ij+1}$ *//* $t_8 = |g_{ij}| \times O(1) = O(|g_{ij}|)$, $s_8 = O(|g_{ij}|)$
}// end $if(p_i.dist = d)$
9- $if(p_i.dist < d)\{$ *//* $t_9 = J^2 \times O(1) = O(J^2)$, $s_9 = O(n)$
10- $k_i = true$ *//* $t_{10} = |k_{ij}| \times O(1) = O(|k_{ij}|)$, $s_{10} = O(n)$
11- $k_{ij} = p_i$ *//* $t_{11} = |k_{ij}| \times O(1) = O(|k_{ij}|)$, $s_{11} = O(|k_{ij}|)$
12- $k_{ij \to ij+1}$ *//* $t_{12} = |k_{ij}| \times O(1) = O(|k_{ij}|)$, $s_{12} = O(|k_{ij}|)$
}// end $if(p_i.dist < d)$
13- $J_i = J_i + 1$ *//* $t_{13} = J^2 \times O(1) = O(J^2)$, $s_{13} = O(n)$
}// end for $x_t \in M_{(j,k)}$, $k = v_j^l, v_j^r$

}// end for $j \in in_{ij}$
7- Versammlung()
Hier gilt:
$|g_{0j}| + |g_{1j}| + \ldots + |g_{n-1j}| < c \times J^2 = O(J^2)$, $c \in \mathbb{N}^+ = \{1, 2, \ldots\}$
$|k_{0j}| + |k_{1j}| + \ldots + |k_{n-1j}| < c \times J^2 = O(J^2)$
Wobei J^2 *die maximale Anzahl der berechneten Distanzen durch einen Prozessor ist.*
$T(Dist(i)) = O(J^2 \times A(.)))$
$S(Dist(i)) = O(\max(J^2, n)) = O(n)$
}
$G(i)\{$
1- $s.Wait()$
2- $s.Release()$
3- $if(g_i)\{$ *//* $t_3 = O(n)$, $s_3 = O(n)$
4- for $k = 0$ *to* $|g_{ij}|\{$ $t_4 = O(|g_{ij}|)$
5- $G_j = g_{ik}$ *//* $t_5 = |g_{ij}| \times O(1) = O(|g_{ij}|)$, $s_5 = O(J^2)$
6- $G_{j \to j+1}$ *//* $t_6 = |g_{ij}| \times O(1) = O(|g_{ij}|)$, $s_6 = O(J^2)$
7- $b_g = true$ *//* $t_7 = |g_{ij}| \times O(1) = O(|g_{ij}|)$, $s_7 = O(1)$
$T(for) = O(J^2)$, $S(for) = O(J^2)$
}// end for $k = 0$ *to* $|g_{ij}|$
}// end $if(g_i)$
8- $Versammlung()$
Hier gilt:
$|g_{0j}| + |g_{1j}| + \ldots + |g_{n-1j}| < c \times J^2 = O(J^2)$, $c \in \mathbb{N}^+ = \{1, 2, \ldots\}$
Wobei J^2 *die maximale Anzahl der berechneten Distanzen durch einen Prozessor ist.*
$T(G(i)) = O(\max(J^2, n)) = O(n)$
$S(G(i)) = O(\max(J^2, n)) = O(n)$
}
$K(i)\{$
1- $s.Wait()$
2- $s.Release()$
3- $if(k_i)\{$ *//* $t_3 = O(n)$, $s_3 = O(n)$
4- for $k = 0$ *to* $|k_{ij}|\{$ *//* $t_4 = O(|k_{ij}|)$
5- $K_j = k_{ik}$ *//* $t_5 = |k_{ij}| \times O(1) = O(|k_{ij}|)$, $s_5 = O(J^2)$
6- $K_{j \to j+1}$ *//* $t_6 = |k_{ij}| \times O(1) = O(|k_{ij}|)$, $s_6 = O(J^2)$
7- $b_k = true$ *//* $t_7 = |k_{ij}| \times O(1) = O(|k_{ij}|)$, $s_7 = O(1)$
$T(for) = O(J^2)$, $S(for) = O(J^2)$

}// end for $k = 0$ *to* $|k_{ij}|$
}// end $if(k_i)$
8- $Versammlung()$
Hier gilt:
$|k_{0j}| + |k_{1j}| + \ldots + |k_{n-1j}| < c \times J^2 = O(J^2)$, $c \in \mathbb{N}^+ = \{1, 2, \ldots\}$
$T(K(i)) = O(\max(J^2, n)) = O(n)$
$S(K(i)) = O(\max(J^2, n)) = O(n)$
}
$Ergebnis()\{$
1- $P_0 : if(b_k = falls \,\&\&\, b_g = false)\, N_1(x) = \{x_M\}$ *//* $t_1 = O(1)$, $s_1 = O(1)$
2- $P_0 : if(b_k = falls \,\&\&\, b_g = true)\, N_1(x) = \{x_M\} \cup G_j$ *//* $t_2 = O(1)$, $s_2 = O(J^2)$
3- $P_0 : if(b_k = true)\{$ *//* $t_3 = O(1)$, $s_3 = O(1)$
4- $MergeSort(K_j)$ *//* $t_4 = O(J^2 \times \log\, J^2)$, $s_4 = O(J^2)$
5- $p_0 = K_0$ *//* $t_5 = O(1)$, $s_5 = O(1)$
6- $d = p_0.dist$ *//* $t_6 = O(1)$, $s_6 = O(1)$
7- *for* $m = 0$ *to* $|K_j|$ $\{$ *//* $t_7 = O(J^2)$
8- $p_0 = K_m$ *//* $t_8 = O(1)$, $s_8 = O(J^2)$
9- $if(p_0.dist > d)\{$ *//* $t_9 = O(1)$, $s_9 = O(1)$
10- *Lösche* $K_m, K_{m+1}, \ldots, K_j$ *//* $t_{10} = O(J^2)$, $s_9 = O(J^2)$
11- $k_{j \to m-1}$ *//* $t_{11} = O(1)$, $s_{11} = O(J^2)$
12- *break*
}// end $if(p_0.dist > d)$
$T(for) = O(J^2)$, $S(for) = O(J^2)$
}// end for $j = 0$ *to* $|K_j|$
13- $N_1(x) = K_j$ *//* $t_{13} = O(1)$, $s_{13} = O(J^2)$
}// end $if(b_k = true)$
$T(Ergebnis()) = O(J^2 \times \log\, J^2)$
$S(Ergebnis()) = O(J^2)$
}
$T(PRAM_3) = O(\max(J^2 \times A(.), J^2 \times \log\, J^2, n)) = O(J^2 \times A(.))$
$S(PRAM_3) = O(f \times n)$

Hier wollen wir insgesamt die Laufzeit und den Speicherverbrauch des Algorithmus 1 mit n Prozessoren analysieren. Schritt 2 (PRAM 1) und 3/item 1/Option b (PRAM 3) haben den maximalen Aufwand in Laufzeit und Speicherverbrauch in Algorithmus 1. Hiermit ist die Laufzeit des Algorithmus 1 $O(J \times A(.))$ und sein Speicherverbrauch gleich $O(J^1 \times n)$.

Literaturverzeichnis

[1] J. A. Hartigan. Clustering Algorithms. New York: John Wiley & Sons, 1975.

[2] A. K. Jain und R. C. Dubes. Algorithms for Clustering Data. Englewood Cliffs, NJ: Prentice Hall, 1988.

[3] A. K. Jain, M.N. Murty, und P. J. Flynn. Data clustering: A survey. ACM Comput. Surv., 31:264-323, 1999.

[4] L. Kaufman und P. J. Rousseeuw. Finding Groups in Data: An introduction to Cluster Analysis. New York: John Wiley & Sons, 1990.

[5] J. MacQueen. Some Methods for Classification and analysis of Multivariate observations. Proc. 5th Berkeley Symp. Math. Statist, Prob., 1:281-297, 1967.

[6] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 98), pages 73-84, Seattle, WA, June 1998.

[7] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering Algorithm using dynamic modeling. COMPUTER, 32:68-75, 1999.

[8] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 96) , pages 103-114. Montreal, Canada, June 1996.

[9] M. Ankerst, M. Breunig, H.-R Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 99), pages 49-60, Philadelphia, PA, June 1999.

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD 96), Pages 226-231, Portland, OR, Aug. 1996.

[11] P. Langley Elements of Machine Learning. San Franciso: Morgan Kaufmann, 1996.

[12] T. M. Mitchell. Machine Learning. New York: McGraw-Hill, 1997.

[13] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. Machine Learning, Neural and Statistical Classification. New York: Ellis Horwood, 1994.

[14] S. M. Weiss and C. A. Kulikowski. Computer System That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. San Mateo, CA: Morgan Kaufmann, 1991.

[15] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In Proc. 1996 Int. Conf. Extending Database Technology (EDBT 96), Avignon, France, Mar. 1996

[16] S.K. Murthy. Automatic construction of decision trees from data: A multidisciplinary survey. Data Mining and Knowledge Discovery, 2:345-389, 1998.

[17] W. Y. Loh and Y. S. Shih. Split selection methods for classification trees. Statistica Sinica, 7:815-840, 1997.

[18] J. R. Quinlan. Induction of decision trees. Machine Learning, 1:81-106, 1986

[19] J. R. Quinlan. C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann, 1993

[20] R. Duda and P. Hart. Pattern Classification and Scene Analysis. New York: John Wiley & Sons, 1973.

[21] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simpel Bayesian classifier. In Proc. 13th Intl. Conf. Machine Learning, page 105-112, 1996.

[22] G. H. John. Enhancements to the Data Mining Process. Ph.D. Thesis, Computer Science Dept., Stanford University, 1997.

[23] D. Heckerman. Bayesian networks for knowledge discovery. In U. M. Fayyad, G. Piatetsky-Shaprio, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 273-305. Cambridge, MA:MIT Press, 1996

[24] S. Russell and P. Norvig. Artificial Inteligence: A Modern Approach. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[25] F.V. Jensen. An Introduction to Bayesian Networks. New York: Springer-Verlag, 1996.

[26] S. Russel, J. Binder, D. Koller, and K. Kanazawa. Local Learning in probabilistic networks with hidden variables. In Proc. 14th Joint Int. Conf. on Artificial Intelligence (IJCAI95), volume 2, pages 1146-1152, Montral, Canada, Aug. 1995.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, parallel Distributed Processing. Cambridge, MA: MIT Press, pp. 318-362, 1986.

[28] S. J. Hanson ans D. J. Burr. Minkowski back-propagation: Learning in connectionist models with non-euclidean error signal. In Neural Information Processing System, America Institute of Physics, 1988.

[29] S. Fahlman and C. Lebiere. The cascade-correlation learning algorithm. In Technical Report CMU-CS-90-100, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1990.

[30] Y. LeCun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D. Touretzky, editor, Advances in Neural Information Processing Systems, 2. San Mateo, CA: Morgan Kaufmann, pp. 598-605, 1990.

[31] M. James. Classification Algorithms. New York: John Wiley & Sons, 1985.

[32] T. Cover and P. Hart. Nearest neighbor pattern classification. IEEE Trans. Information Theory, 13:21-27, 1967.

[33] K. Fukunaga and D. Hummels. Bayes error estimation using parzen and knn procedure. In IEEE Trans. Pattern Analysis and Machine Learning, pages 634-643, 1987.

[34] C. Riesbeck and R. Schank. Inside Case-Based Reasoning. Hillsdale, NJ: Lawrence Erlbaum, 1989.

[35] J. L. Kolodner. Case-Based Reasoning. San Francisco: Morgan Kaufmann, 1993.

[36] D. B. Leake. GBR in context: The present and future. In D. B. Leake, editor, Cased-Based Reasoning: Experience, Lessons, and Future Direction, page 3-30. Menlo Park: AAAI Press, 1996.

[37] A. Aamodt and E. Plazas. Case-based reasoning: Foundational issues, methodological variation, and system approaches. AI Comm., 7:39-52, 1994.

[38] D. Goldberg. Genetic Algotithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[39] Z. Michalewicz. Genetic Algorithms + Data Structures= Evolution Programs. New York: Springer-Verlag, 1992.

[40] M. Mitchell. An Introduction to Genetic Algorithms. Cambridge, MA:MIT Press, 1996.

[41] Z. Pawlak. Rough Sets, Theoretical Aspects of Reasoning about Data. Boston: Kluwer Academic Publishers, 1991.

[42] W. Ziarko. Rough Sets, Fuzzy Sets and Knowledge Discovery. New York: Springer-Verlag, 1994.

[43] K. Cios, W. Pedrycz, and R. Swiniarski. Data Mining Methods for Knowledge Discovery. Boston: Kluwer Academic Publishers, 1998.

[44] L. A. Zadeh. Fuzzy sets. Information and Control, 8:338-353, 1965.

[45] A. Berson and S. J. Smith. Data Warehousing, Data Mining, and OLAP. New York: McGraw-Hill, 1997.

[46] J. Han and M. Kamber. Data mining: concepts and techniques. Morgan Kaufmann, 2012, ISBN 1-55860-489-8

[47] R. Lipton and R. Tarjan. Applications of a planar separator theorem. SIAM Journal on Computing, 9:615-627, 1980.

[48] M. I. Shamos and D. Hoey. Closest point problems. Proceedings of the Sixteen IEEE Symposium of Foundations of Computer Science, pages 152-162, 1975.

[49] K. Clarkson. A randomized algorithm for closest-point queries. SIAM Journal on Computing, 17:830-847, 1988.

[50] S. Meiser. Point location in arrangements of hyperplanes. Information and Computation, 106:286-303, 1993.

[51] John L. Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18:509-517, 1975.

[52] Hannan Samet. Foundations of Multidimensional and Metrik Data Structures. Elsevier, 2006.

[53] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching. J. ACM, 6(45):891-923, 1998.

[54] Jon Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, 1997.

[55] Roussopoulos N., Kelley S., Vincent F.: 'Nearest Neighbor Queries', Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71-79.

[56] D. Knuth: "The Art of Computer Programming: Sorting and Searching", Vol.3, Addison- Wesley, 1973

[57] Christian Böhm, Stefan Berchtold, Daniel A. Keim: "Searching in High-dimensional Spaces - Index Structures for Improving the Performance of Multimedia Databases", 2001

[58] Yaling Pei, Osmar Zaiane. "A Synthetic Data Generator for Clustering and Outlier Analysis". Computing Science Department University of Alberta, Edmonton, Canada.

[59] Indyk, P. and Motwani, R. 1998. Approximate nearest neighbor: Towards removing the curse of dimensionality. In Proceedings of the Symposium on Theory of Computing, pp. 604-613.

[60] Ben-Hur, Asa, Horn, David, Siegelmann, Hava, and Vapnik, Vladimir; "Support vector clustering" (2001) Journal of Machine Learning Research, 2: 125–137.

[61] Bilwaj Gaonkar, Christos Davatzikos Analytic estimation of statistical significance maps for support vector machine based multi-variate image analysis and classification.

[62] Breiman L., Random forests. In Machine Learning, Seiten 5-32, 2001.

[63] Tin Kam Ho, Random Decision Forests, Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, Canada, August 14-18, 1995, 278-282.

[64] Alan Gibbons: "Efficient Parallel Algorithms", Verlag: Cambridge University Press; Auflage: Reprint (1. Februar 1990), ISBN-13: 978-0521388412, Seite 180.

[65] "http://personalpages.manchester.ac.uk/mbs/julia.handl/generators.html", letzte Zugriffszeit 01.09.2016.

[66] M. Flynn: Some Computer Organizations and Their Effectiveness, IEEE Trans. Comput., Band C-21, S. 948–960, 1972.

[67] Neil Immerman, Expressibility and parallel complexity. Siam Journal on Computing, vol. 18, no. 3, pp. 625-638, 1989.

[68] J. Hartmanis (1971), "Computational Complexity of Random Access Stored Program Machines," Mathematical Systems Theory 5, 3 (1971) pp. 232–245.

[69] M. Raynal, D. Beeson: Algorithms for mutual exclusion. MIT Press, Cambridge MA 1986, ISBN 0-262-18119-3.

[70] N. G. de Bruijn (1958). Asymptotic Methods in Analysis. Amsterdam: North-Holland. pp. 5–7. ISBN 978-0-486-64221-5.