FM-DBSCAN: Ein effizienter, dichte-basierter Clustering-Algorithmus

Philipp Egert
Brandenburgische Technische Universität Cottbus–Senftenberg
Institut für Informatik, Informations- und Medientechnik
Fachgebiet Datenbank- und Informationssysteme
03013 Cottbus, Deutschland
Philipp.Egert@b-tu.de

ABSTRACT

DBSCAN ist ein dichte-basierter Clustering-Algorithmus, der Cluster beliebiger Form auffindet und diese von Rauschen trennt. Aufgrund des quadratischen Aufwands ist DBSCAN für große Datenmengen jedoch oft ungeeignet. In dieser Arbeit wird deshalb ein effizienterer Algorithmus namens FM-DBSCAN vorgestellt, der für eine beliebige Distanzfunktion (Metrik) dasselbe Ergebnis wie DBSCAN liefert. Hierfür partitioniert FM-DBSCAN die Datenkollektion in Leader-Umgebungen, auf denen anschließend das Clustering durchgeführt wird. Erste Experimente mittels synthetischen Datenkollektionen zeigen, dass FM-DBSCAN um einen Faktor > 990 schneller als DBSCAN ist und auch wesentlich besser mit der Kollektionsgröße skaliert.

Kategorien und Themenbeschreibungen

I.5.3 [**PATTERN RECOGNITION**]: Clustering—*Algorithms*

Schlüsselwörter

Density-based Clustering, DBSCAN, Leaders

1. EINLEITUNG

Clustering ist ein wichtiges Teilgebiet des Data Mining, welches bspw. für die Segmentierung von Kunden oder der Strukturierung von Textdokumenten eingesetzt wird. Ziel des Clustering ist es, Objekte so in Gruppen (Cluster) einzuteilen, dass Objekte eines Cluster einander möglichst ähnlich und Objekte verschiedener Cluster einander möglichst unähnlich sind. Die Gruppe der dichte-basierten Clustering-Algorithmen lösen dieses Problem, indem sie dichte Regionen von Regionen mit geringer Dichte trennen. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) ist ein dichte-basierter Clustering-Algorithmus mit dem es möglich ist, Cluster beliebiger Form zu erkennen [1]. Aufgrund dieser Eigenschaft und, dass die Anzahl der zu suchenden Cluster nicht vorgegeben werden muss, ist DBSCAN für viele Einsatzgebiete attraktiv. Ein großer Nachteil von DBSCAN ist allerdings seine Laufzeit von $O(n^2)$, wobei n die Größe der Datenkollektion ist. Dies macht DBSCAN für große Datenkollektionen nicht praktikabel. Um den Aufwand zu reduzieren, wurden verschiedene Ansätze entwickelt [2, 3, 4, 5, 6]. Die bisherigen Verfahren sind jedoch entweder nur approximativ [2, 4, 6] oder nur für den euklidischen Raum $\mathbb{R}^d$ ausgelegt [2, 3, 5]. Für allgemeine Distanzfunktionen existiert bisher kein exakter Ansatz. Als wesentlicher Beitrag dieser Arbeit wird daher ein exakter und effizienter Algorithmus mit dem Namen *Fast Metric* DBSCAN vorgestellt, der für beliebige Distanzfunktionen einsetzbar ist.

Die Arbeit ist wie folgt gegliedert: In Abschnitt 2 wird DBSCAN vorgestellt. Danach wird in Abschnitt 3 der Stand der Technik beleuchtet. FM-DBSCAN wird in Abschnitt 4 eingeführt. Die experimentelle Evaluierung von FM-DBSCAN im Vergleich zu DBSCAN erfolgt in Abschnitt 5. In Abschnitt 6 werden die Zusammenfassung der Arbeit und ein Ausblick über zukünftige Forschungsvorhaben gegeben.

2. DBSCAN

In dem folgenden Abschnitt wird der von Ester et al. entwickelte, dichte-basierte Clustering-Algorithmus DBSCAN eingeführt [1]. Hierzu werden in Abschnitt 2.1 die wesentlichen Definitionen und Lemmata von Ester et al. zusammengefasst. Auf DBSCAN wird im Abschnitt 2.2 eingegangen.

2.1 Definitionen und Lemmata

Im Nachfolgenden sei O eine endliche Menge von Objekten, mit $|O| = n$. Des Weiteren sei d eine Distanzfunktion (Metrik) auf der Menge X und $O \subseteq X$. $minPts \in \mathbb{N}_{\geq 0}$ und $\epsilon \in \mathbb{R}_{\geq 0}$ sind die beiden Parameter von DBSCAN. Alle folgenden Definitionen und Lemmata sind immer abhängig von ϵ und $minPts$.

Definition 1. Ein Objekt $o \in O$ heißt *Kernobjekt*, wenn $|N_\epsilon(o)| \geq minPts$ ist, mit $N_\epsilon(o) = \{\hat{o} \in O \mid d(o, \hat{o}) \leq \epsilon\}$.

ϵ und $minPts$ spezifizieren hierbei einen minimalen Dichtegrenzwert, der festlegt, ob ein Objekt ein Kernobjekt ist. Alle Objekte, welche keine Kernobjekte sind, werden *Nicht-Kernobjekte* genannt.

Definition 2. Ein Objekt $p \in O$ ist *direkt dichte-erreichbar* von $q \in O$, wenn q ein Kernobjekt und $p \in N_\epsilon(q)$ ist.

Nach Definition 2 sind alle Objekte der ϵ-Umgebung eines Kernobjektes q direkt dichte-erreichbar von q. Um die Er-

28^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 24.05.2016 - 27.05.2016, Nörten-Hardenberg, Germany.

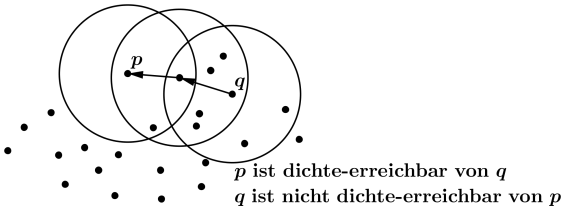


Abb. 1: Dichte-Erreichbarkeit für $minPts = 5$

reichbarkeit über die Grenzen der ϵ-Umgebung eines Kernobjektes hinauszuführen, wird der Begriff der *Dichte-Erreichbarkeit* eingeführt.

Definition 3. Ein Objekt $p \in O$ ist *dichte-erreichbar* von einem Objekt $q \in O$, wenn es eine Folge von Objekten $p_1, \ldots, p_m \in O$ gibt, sodass $p_1 = q, p_m = p$ und p_{i+1} direkt dichte-erreichbar von p_i ist, für $1 \leq i < m$.

Wie man leicht sieht, bildet die Dichte-Erreichbarkeit den transitiven Abschluss über der Relation der direkten Dichte-Erreichbarkeit. Die Veranschaulichung der Dichte-Erreichbarkeit ist in Abb. 1 zu sehen. Mittels der Dichte-Erreichbarkeit kann man einen Cluster wie folgt konstruieren [1].

Lemma 1. Sei p ein Kernobjekt in O. Dann ist die Menge $C = \{o \in O | o \text{ ist dichte-erreichbar von } p\}$ ein Cluster in O.

Aus Lemma 1 folgt, dass ein Cluster C vollständig berechnet werden kann, indem man ausgehend von einem beliebigen Kernobjekt $p \in C$, alle dichte-erreichbaren Objekte von p ermittelt. Alle Objekte, die zu keinem solchen Cluster gehören, werden als *Rauschen* bezeichnet.

2.2 Algorithmus

Ziel von DBSCAN ist es nun, alle solche Cluster zu finden. Die Erkenntnis aus Lemma 1 fließt dabei in den Alg. 1 ein.

Alg. 1 DBSCAN($O, \epsilon, minPts$)

```
1:  clusterId := 1;
2:  for all o ∈ O do
3:      if o.clId = −1 then
4:          if expandCluster(O, o, clusterId, ε, minPts) then
5:              clusterId := clusterId + 1;
```

Alg. 2 expandCluster(O, o, $clusterId$, ϵ, $minPts$)

```
1:  neighborhood := N_ε(o);
2:  if |neighborhood| < minPts then
3:      o.clId := 0;
4:      return false;
5:  seeds := ∅;
6:  setClusterId(neighborhood, clusterId);
7:  expandSeeds(neighborhood, o, clusterId, seeds);
8:  while seeds ≠ ∅ do
9:      Wähle Objekt ô aus der Menge seeds aus;
10:     seeds := seeds \ {ô};
11:     neighborhood := N_ε(ô);
12:     if |neighborhood| ≥ minPts then
13:         setClusterId(neighborhood, clusterId);
14:         expandSeeds(neighborhood, ô, clusterId, seeds);
15: return true;
```

In DBSCAN (Alg. 1) erhält jedes Objekt $o \in O$ ein Attribut *clId*, welches angibt zu welchem Cluster es gehört. Der Wert 0 bedeutet, das Objekt gehört zum Rauschen und ein Wert größer 0 spezifiziert die zugehörige Clusternummer. Initial ist für alle Objekte *clId* auf -1 (nicht klassifiziert) gesetzt. In DBSCAN wird nun versucht, von einem Objekt o mit $clId = -1$, einen neuen Cluster zu finden (Z. 3–5). Hierzu wird in expandCluster zuerst die ϵ-Umgebung von o berechnet (Z. 1) und überprüft, ob es sich bei o um ein Kernobjekt handelt. Ist o ein Nicht-Kernobjekt, wird es als Rauschen markiert und die Suche nach einem Cluster von o aus abgebrochen (Z. 2–4). Es sei angemerkt, dass o später noch einem Cluster zugewiesen werden kann, wenn es in der ϵ-Umgebung eines Kernobjekts liegt. Andernfalls ist o ein Kernobjekt und es kann laut Lemma 1 ein neuer Cluster mit der Nummer *clusterId* konstruiert werden. Hierzu werden die direkt dichte-erreichbaren Objekte von o, also $N_\epsilon(o)$, dem Cluster hinzugefügt (Z. 6). Anschließend werden für alle Objekte aus $N_\epsilon(o)$ deren direkt dichte-erreichbaren Objekte ermittelt und dem Cluster zugewiesen. Die Schritte werden solange wiederholt, bis keine neuen Objekte des Clusters mehr gefunden werden können (Z. 8–14). expandSeeds fügt dabei nur die Objekte aus $N_\epsilon(o) \setminus \{o\}$ für das aktuelle Kernobjekt o in die temporäre Menge *seeds* ein, für die $clId = -1$ ist. setClusterId markiert die Objekte aus $N_\epsilon(o)$ mit der Nummer *clusterId*, für die $clId < 1$ gilt. Die Laufzeit von Alg. 1 beträgt $O(n^2)$.

3. STAND DER TECHNIK

Zur Beschleunigung von DBSCAN existieren bereits mehrere Ansätze, welche sich in exakte (liefern dasselbe Ergebnis wie DBSCAN) und approximative Verfahren (garantieren kein exaktes Ergebnis) unterteilen lassen. Exakte Verfahren sind in den Arbeiten [2, 3, 5] und approximative Verfahren in den Arbeiten [2, 4, 6] zu finden.

GriDBSCAN ist ein exaktes Verfahren für den euklidischen Raum $\mathbb{R}^d$ [5]. Dabei zerlegt GriDBSCAN den Datenraum mittels eines Gitters und führt auf den einzelnen Gitterzellen DBSCAN aus, um anschließend deren Ergebnisse in einer Mischphase zusammenzuführen.

Ein exaktes Verfahren, welches ebenfalls auf einem Datengitter basiert, wurde von Gunawan vorgestellt [3]. Zwar ist der Algorithmus nur für den $\mathbb{R}^2$ ausgelegt, jedoch konnte dafür eine Laufzeit von $O(n \log n)$ gezeigt werden.

Die Erweiterung des Verfahrens von Gunawan auf höhere Dimensionen d wurde von Gan und Tao gezeigt [2]. Für $d \geq 4$ konnte eine Laufzeit von $O(n^{2-\frac{2}{\lceil d/2 \rceil + 1} + \delta})$ nachgewiesen werden, für eine beliebig kleine Konstante $\delta > 0$. Für $d = 3$ kann die Laufzeit auf $O((n \log n)^{\frac{4}{3}})$ verbessert werden.

Gan und Tao stellen ebenfalls noch das ρ-approximative DBSCAN für den $\mathbb{R}^d$ vor, mit einer Laufzeit von $O(n)$ [2]. $\rho > 0$ ist eine beliebig kleine Konstante, welche die Qualität des Ergebnisses beeinflusst. Kleinere Werte für ρ liefern bessere Ergebnis als große, benötigen aber mehr Rechenzeit.

Rough-DBSCAN ist ein approximatives Verfahren, das wie FM-DBSCAN auf dem Prinzip der Leader-Umgebungen basiert [6]. Rough-DBSCAN führt auf der Menge der Leader DBSCAN aus, um die Objekte zu clustern und somit den Aufwand zu reduzieren. Hierfür wird DBSCAN nur geringfügig geändert. Um die Größe der ϵ-Umgebung eines Leaders zu bestimmen, wird eine leicht zu berechnende Abschätzungsformel auf Basis der Leader-Umgebungen verwendet. Rough-DBSCAN ist für beliebige Metriken anwendbar. Eine Laufzeit von $O(n)$ wurde jedoch nur für den $\mathbb{R}^d$ gezeigt.

Das approximative Verfahren FDBSCAN für den $\mathbb{R}^d$ hat

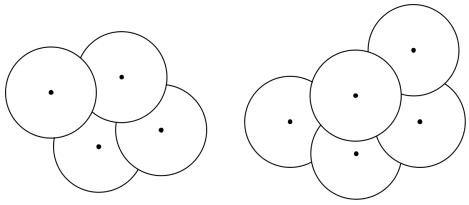


Abb. 2: Beispiel für eine Leader-Partition

einen, im Vergleich zu DBSCAN, veränderten Ablauf [4]. Nach dem Auffinden eines Kernobjektes wird nicht in dessen ϵ-Umgebung nach weiteren Kernobjekten gesucht, sondern außerhalb. Hierfür sind die Objekte nach einer ausgewählten Dimension sortiert, sodass mit dem nächsten Objekt in der Sortierung, welches nicht in der ϵ-Umgebung war und noch keinem Cluster zugewiesen wurde, fortgesetzt wird. Cluster werden bei Überschneidungen ggf. zusammengefasst.

Im Gegensatz zu den hier aufgeführten Verfahren, die entweder approximativ oder nur für den $\mathbb{R}^d$ exakt sind, liefert FM-DBSCAN für beliebige Distanzfunktionen ein exaktes Ergebnis und stellt somit einen universelleren Ansatz dar.

4. FM-DBSCAN

In diesem Abschnitt wird der Clustering-Algorithmus *Fast Metric* DBSCAN präsentiert. FM-DBSCAN besteht aus zwei Phasen. In der ersten Phase werden die Objekte mittels des Algorithmus COUNTED-LEADERS [6] zu Leader-Umgebungen aggregiert, die zusammen eine Leader-Partition bilden. Anhand dieser Leader-Partition und unter Ausnutzung der Dreiecksungleichung, kann in der zweiten Phase ein effizientes Clustering, ähnlich dem von DBSCAN, durchgeführt werden. Hierzu werden in Abschnitt 4.1 die Definitionen und Lemmata eingeführt, die für FM-DBSCAN benötigt werden. Die Berechnung der Leader-Partition und das anschließende Clustering erfolgt in den Abschnitten 4.2 und 4.3.

4.1 Definitionen und Lemmata

Seien O, d, *minPts* und ϵ gegeben wie in Abschnitt 2.1. Im Folgenden sind die Definitionen und Lemmata immer abhängig von ϵ und *minPts*.

Definition 4. Ein Tupel $(l, N_\epsilon^L(l))$ ist eine Leader-Umgebung in O, wenn $N_\epsilon^L(l) \subseteq N_\epsilon(l)$ und $l \in N_\epsilon^L(l)$ ist. l ist der Repräsentant der Menge $N_\epsilon^L(l)$ und wird *Leader* genannt.

Die Leader-Umgebung mit dem Leader l ist also ein Ausschnitt der ϵ-Umgebung von l. Mittels der Leader-Umgebungen soll nun die Menge der Objekte O partitioniert werden.

Definition 5. Eine *Leader-Partition* von O ist eine Menge von Leader-Umgebungen $L = \{(l_1, N_\epsilon^L(l_1)), \ldots, (l_m, N_\epsilon^L(l_m))\}$, mit $d(l_i, l_j) > \epsilon \wedge N_\epsilon^L(l_i) \cap N_\epsilon^L(l_j) = \emptyset$ für $i \neq j$ und $\bigcup_{i=1}^m N_\epsilon^L(l_i) = O$.

Ein Beispiel für eine Leader-Partition, die auch der Alg. 3 aus Abschnitt 4.2 berechnet, ist in Abb. 2 dargestellt. Im Anschluss sollen die Begriffe der direkten Dichte-Erreichbarkeit und der Dichte-Erreichbarkeit auf Leader-Umgebungen übertragen werden.

Definition 6. Eine Leader-Umgebung $(l_p, N_\epsilon^L(l_p))$ ist *direkt dichte-erreichbar* von einer Leader-Umgebung $(l_q, N_\epsilon^L(l_q))$ in O, wenn $p_1, \ldots, p_m \in N_\epsilon^L(l_p) \cup N_\epsilon^L(l_q)$ gibt, sodass $p_1 = l_q, p_m = l_p$ ist und es gilt:

1. p_{i+1} ist direkt dichte-erreichbar von p_i für $1 \leq i < m$,
2. l_p ist ein Kernobjekt in O oder $|N_\epsilon^L(l_p)| = 1$.

Definition 7. Eine Leader-Umgebung $(l_p, N_\epsilon^L(l_p))$ ist *dichte-erreichbar* von einer Leader-Umgebung $(l_q, N_\epsilon^L(l_q))$ in O, wenn es eine Folge von Leader-Umgebungen $(l_1, N_\epsilon^L(l_1)), \ldots, (l_m, N_\epsilon^L(l_m))$ in O gibt, sodass $(l_1, N_\epsilon^L(l_1)) = (l_q, N_\epsilon^L(l_q))$, $(l_m, N_\epsilon^L(l_m)) = (l_p, N_\epsilon^L(l_p))$ und $(l_{i+1}, N_\epsilon^L(l_{i+1}))$ direkt dichte-erreichbar von $(l_i, N_\epsilon^L(l_i))$ in O ist, für $1 \leq i < m$.

In der Definition 6 Punkt 2 wurde explizit gefordert, dass l_p ein Kernobjekt oder $|N_\epsilon^L(l_p)| = 1$ ist. Dadurch wird erreicht, dass für eine Umgebung $(l_p, N_\epsilon^L(l_p))$, die von einer Umgebung $(l_q, N_\epsilon^L(l_q))$ direkt dichte-erreichbar ist, gilt, dass alle Objekte $o \in N_\epsilon^L(l_q) \cup N_\epsilon^L(l_p)$ dichte-erreichbar von l_q sind. Dies wäre für eine Umgebung $(l_p, N_\epsilon^L(l_p))$, mit l_p ist ein Nicht-Kernobjekt und $|N_\epsilon^L(l_p)| > 1$, nicht der Fall.

Im Anschluss werden noch drei Lemmata vorgestellt, die es dem Alg. 4 aus Abschnitt 4.3 ermöglichen, erheblichen Berechnungsaufwand einzusparen.

Lemma 2. Seien $(l_p, N_\epsilon^L(l_p))$ und $(l_q, N_\epsilon^L(l_q))$ zwei Leader-Umgebungen und $(l_p, N_\epsilon^L(l_p))$ ist direkt dichte-erreichbar von $(l_q, N_\epsilon^L(l_q))$ in O, dann gelten folgende Aussagen:

1. Ist l_p ein Kernobjekt, dann existieren zwei Kernobjekte $p \in N_\epsilon^L(l_p)$ und $q \in N_\epsilon^L(l_q)$ mit $d(p, q) \leq \epsilon$.
2. Ist $|N_\epsilon^L(l_p)| = 1$, dann existiert ein Kernobjekt $q \in N_\epsilon^L(l_q)$ in O mit $d(l_p, q) \leq \epsilon$.

Lemma 3. Seien p, q zwei Objekte in O. Ist $d(p, q) > 2\epsilon$, so gilt für alle $o \in N_\epsilon(p) : d(o, q) > \epsilon$.

Lemma 4. Seien p, q zwei Objekte in O. Ist $d(p, q) > 3\epsilon$, so gilt für alle $o_1 \in N_\epsilon(p)$ und $o_2 \in N_\epsilon(q) : d(o_1, o_2) > \epsilon$.

Lemma 2 beschreibt eine effiziente Möglichkeit, um zu testen, ob eine Leader-Umgebung $(l_p, N_\epsilon^L(l_p))$ von einer Leader-Umgebung $(l_q, N_\epsilon^L(l_q))$ direkt dichte-erreichbar ist. Hierfür müssen wir im ersten Fall nur nachweisen, dass l_q und l_p Kernobjekte sind und dass zwei Kernobjekte $p \in N_\epsilon^L(l_p)$ und $q \in N_\epsilon^L(l_q)$ existieren mit $d(p, q) \leq \epsilon$. Der Test auf $d(p, q) \leq \epsilon$ kann dabei mit Lemma 3 beschleunigt werden. Für den zweiten Fall reicht es sogar aus, nachzuweisen, dass l_q ein Kernobjekt ist und das ein Kernobjekt $q \in N_\epsilon^L(l_q)$ existiert mit $d(q, l_p) \leq \epsilon$. Um die Anzahl dieser Tests so gering wie möglich zu halten, verwenden wir die Lemmata 3 und 4. Hierzu muss lediglich die Distanz $d(l_p, l_q)$ berechnet werden, um zu entscheiden, ob für zwei Objekte $p \in N_\epsilon^L(l_p)$ und $q \in N_\epsilon^L(l_q)$ die Distanz $d(p, q) \leq \epsilon$ sein kann. Sollte das nicht der Fall sein, so ist keine der Leader-Umgebungen voneinander direkt dichte-erreichbar. Auf die Beweise der Lemmata wird aufgrund des eingeschränkten Platzes verzichtet. Im Wesentlichen basieren sie auf der von d erfüllten Dreiecksungleichung, der direkten Dichte-Erreichbarkeit und der Dichte-Erreichbarkeit von Objekten.

4.2 Berechnung der Leader-Partition

Um eine Leader-Partition zu berechnen, nutzen wir eine angepasste Variante des Algorithmus COUNTED-LEADERS (Alg. 3) von Viswanath und Suresh Babu [6].

Alg. 3 computeLeaderPartition(O, ϵ)

```
1:  leaderPartition := ∅;
2:  for all o ∈ O do
3:      leaderFound := false;
4:      for all (l, N_ε^L(l)) ∈ leaderPartition do
5:          if d(o, l) ≤ ε then
6:              N_ε^L(l) := N_ε^L(l) ∪ {o};
7:              leaderFound := true;
8:              break;
9:      if leaderFound = false then
10:         leaderPartition := leaderPartition ∪ {(o, {o})};
11: return leaderPartition;
```

Die Leader-Partition (*leaderPartition*) wird schrittweise aufgebaut. Hierzu wird jedes Objekte $o \in O$ der ersten Leader-Umgebung $(l, N_\epsilon^L(l)) \in leaderPartition$ zugewiesen, für die $d(o,l) \leq \epsilon$ gilt (Z. 4–8). Existiert keine solche Leader-Umgebung, so wird $(o, \{o\})$ als neue Leader-Umgebung zu *leaderPartition* hinzugefügt (Z. 9–10). Die Reihenfolge, in der über *leaderPartition* iteriert wird (Z. 4), ist durch die Einfügereihenfolge der Leader-Umgebungen festgelegt.

4.3 Clustering anhand der Leader-Partition

In diesem Abschnitt wird beschrieben, wie anhand einer Leader-Partition die Menge der Objekte O effizient in Cluster unterteilt werden kann. Die Grundidee ist dabei analog zu DBSCAN. Ausgehend von einer Leader-Umgebung, deren Leader ein Kernobjekt ist, werden alle dichte-erreichbaren Leader-Umgebungen ermittelt. Diese bilden dann zusammen einen Cluster. Probleme bereiten dabei jedoch Leader-Umgebungen $(l, N_\epsilon^L(l))$, bei denen l ein Nicht-Kernobjekt und $|N_\epsilon^L(l)| > 1$ ist. Von diesen Umgebungen aus kann kein Cluster konstruiert werden. Dies ist besonders problematisch, wenn in der Leader-Partition nur solche Umgebungen auftreten. Außerdem sind diese Umgebungen laut Definition 6 nicht direkt dichte-erreichbar von anderen Umgebungen, wodurch Objekte eines Clusters verloren gehen könnten. Die Lösung dieses Problems besteht darin, jedes Mal, wenn auf so eine Umgebung getroffen wird, deren Objekte auf noch nicht geclusterte Umgebungen zu verteilen, deren Leader nicht als Nicht-Kernobjekt markiert wurden. Existiert für eines der Objekte eine solche Umgebung nicht, so wird eine neue Umgebung mit dem Objekt als Leader erzeugt. Ist der neue Leader l_1 noch nicht als Nicht-Kernobjekt markiert, so werden ihm alle als Nicht-Kernobjekte markierten Leader l_2, die noch nicht geclustert wurden und für die $d(l_1, l_2) \leq \epsilon$ gilt, zugeordnet. Dadurch erreicht man, in mehreren Ausführungen der Neuverteilung, dass alle Objekte entweder einer Umgebung zugeordnet sind, deren Leader ein Kernobjekt ist, oder dass Leader-Umgebungen $(l, N_\epsilon^L(l))$ entstehen, mit $|N_\epsilon^L(l)| = 1$. Hierdurch wird die direkte Dichte-Erreichbarkeit wieder anwendbar. Die entsprechende Umsetzung dieses Ansatzes ist in Alg. 4 dargestellt.

Alg. 4 FM-DBSCAN(O, ϵ, *minPts*)

```
1:  L_N := computeLeaderPartition(O, ε);
2:  L_C := ∅;
3:  clusterId := 1;
4:  while L_N ≠ ∅ do
5:      (l, N_ε^L(l)) = argmax_{(l̃, N_ε^L(l̃)) ∈ L_N} |N_ε^L(l̃)|;
6:      if expandCluster((l, N_ε^L(l)), L_N, L_C, clusterId, ε, minPts
        ) then
7:          clusterId := clusterId + 1;
```

Alg. 5 expandCluster($(l, N_\epsilon^L(l))$, L_N, L_C, *clusterId*, ϵ, *minPts*)

```
1:  if isCoreObject(l, L_N, L_C, ε, minPts) then
2:      setClusterId(N_ε^L(l), clusterId);
3:      L_N := L_N \ {(l, N_ε^L(l))};
4:      L_C := L_C ∪ {(l, N_ε^L(l))};
5:      seeds := ∅;
6:      ddrLeaders := findDirectDensityReachableLeaders(
        (l, N_ε^L(l)), L_N, L_C, ε, minPts);
7:      setClusterId(ddrLeaders, clusterId);
8:      expandSeeds(ddrLeaders, L_N, L_C, clusterId, seeds);
9:      while seeds ≠ ∅ do
10:         (l̃, N_ε^L(l̃)) = argmax_{(l̃2, N_ε^L(l̃2)) ∈ seeds} |N_ε^L(l̃2)|;
11:         seeds := seeds \ {(l̃, N_ε^L(l̃))};
12:         ddrLeaders :=
            findDirectDensityReachableLeaders((l̃, N_ε^L(l̃)), L_N,
            L_C, ε, minPts);
13:         setClusterId(ddrLeaders, clusterId);
14:         expandSeeds(ddrLeaders, L_N, L_C, clusterId, seeds);
15:     return true;
16: else
17:     if |N_ε^L(l)| = 1 then
18:         setClusterId(N_ε^L(l), 0);
19:         L_N := L_N \ {(l, N_ε^L(l))};
20:         L_C := L_C ∪ {(l, N_ε^L(l))};
21:         return false;
22:     L_N := L_N \ {(l, N_ε^L(l))};
23:     for all o ∈ N_ε^L(l) do
24:         leaderFound := false;
25:         for all (l̃, N_ε^L(l̃)) ∈ L_N do
26:             if l̃.type ≠ 0 ∧ d(o, l̃) ≤ ε then
27:                 N_ε^L(l̃) := N_ε^L(l̃) ∪ {o};
28:                 leaderFound := true;
29:                 break;
30:         if leaderFound = false then
31:             L_N := L_N ∪ {(o, {o})};
32:             if o.type ≠ 0 then
33:                 for all (l̃, N_ε^L(l̃)) ∈ L_N do
34:                     if l̃.type = 0 ∧ d(o, l̃) ≤ ε then
35:                         N_ε^L(o) := N_ε^L(o) ∪ {l̃};
36:                         L_N := L_N \ {(l̃, N_ε^L(l̃))}
37:     return false;
```

Wie bei DBSCAN hat jedes Objekt das Attribut *clId* (intial -1). Zusätzlich bekommt jedes Objekt ein Attribut *type*, was angibt, ob es sich bei dem Objekt um ein Kernobjekt ($type = 1$) oder ein Nicht-Kernobjekt ($type = 0$) handelt. Initial ist $type = -1$ (undefiniert) für alle Objekte $o \in O$. Die Menge L_C enthält alle geclusterten und als Rauschen markierten Leader-Umgebung. L_N hingegen enthält alle nicht geclusterten Umgebungen ($clId = -1$). FM-DBSCAN läuft solange, bis L_N leer ist (Z. 4), also alle Umgebungen geclustert wurden. Das iterative Einsammeln von dichte-erreichbaren Leader-Umgebungen ist in den Z. 1–15 von expandCluster (Alg. 5) beschrieben. Dabei weist setClusterId der übergebenen Menge von Objekten die übergebene Clusternummer zu. expandSeeds fügt der temporären Menge *seeds* lediglich die direkt dichte-erreichbaren Umgebungen hinzu, deren Leader ein Kernobjekt ist. Sollte die Startumgebung in Alg. 5 ein Nicht-Kernobjekt sein, so erfolgt die angesprochene Neuverteilung der Objekte (Z. 22–36).

findDirectDensityReachableLeaders (Alg. 6) ermittelt für die übergebene Leader-Umgebung $(l, N_\epsilon^L(l))$ alle direkt dichte-erreichbaren Leader-Umgebungen, indem mit einer Kandidatenmenge (*candidates*) (Z. 1) gearbeitet wird. *candidates* wird mit den Leader-Umgebungen aus L_N initialisiert, deren Leader eine Distanz kleiner gleich 3ϵ zu l aufweisen (Lemma 4). Die Beschränkung auf Umgebungen aus L_N ist korrekt, da nur nicht geclusterte, direkt dichte-er-

Alg. 6 findDirectDensityReachableLeaders($(l, N_\epsilon^L(l))$, L_N, L_C, ϵ, $minPts$)

```
1: candidates := N_{3ε}((l, N_ε^L(l)), L_N);
2: ddrLeaders := ∅;
3: while candidates ≠ ∅ do
4:     (l̃, N_ε^L(l̃)) = argmax_{(l̃2, N_ε^L(l̃2)) ∈ candidates} |N_ε^L(l̃2)|;
5:     candidates := candidates \ {(l̃, N_ε^L(l̃))};
6:     if isDirectDensityReachable((l, N_ε^L(l)), (l̃, N_ε^L(l̃)), L_N,
           L_C, ε, minPts) then
7:         ddrLeaders := ddrLeaders ∪ {(l̃, N_ε^L(l̃))};
8:     else if l̃.type = 0 ∧ |N_ε^L(l̃)| > 1 then
9:         L_N := L_N \ (l̃, N_ε^L(l̃));
10:        for all o ∈ N_ε^L(l̃) do
11:            leaderFound := false;
12:            for all (l̃2, N_ε^L(l̃2)) ∈ L_N do
13:                if l̃2.type ≠ 0 ∧ d(o, l̃2) ≤ ε then
14:                    N_ε^L(l̃2) := N_ε^L(l̃2) ∪ {o};
15:                    leaderFound := true;
16:                    if (l̃2, N_ε^L(l̃2)) ∉ candidates ∧
                       (l̃2, N_ε^L(l̃2)) ∉ ddrLeaders ∧ d(l, l̃2) ≤ 3ε then
17:                        candidates := candidates ∪ {(l̃2, N_ε^L(l̃2))};
18:                    break;
19:            if leaderFound = false then
20:                if d(l, o) ≤ 3ε then
21:                    candidates := candidates ∪ {(o, {o})};
22:                L_N := L_N ∪ {(o, {o})};
23:                if o.type ≠ 0 then
24:                    for all (l̃2, N_ε^L(l̃2)) ∈ L_N do
25:                        if l̃2.type = 0 ∧ d(o, l̃2) ≤ ε then
26:                            N_ε^L(o) := N_ε^L(o) ∪ {l̃2};
27:                            L_N := L_N \ {(l̃2, N_ε^L(l̃2))}
28: return ddrLeaders;
```

reichbare Umgebungen gefunden werden sollen. Die Menge *candidates* wird nun schrittweise abgearbeitet. Ist die aktuelle Leader-Umgebung $(\tilde{l}, N_\epsilon^L(\tilde{l}))$ direkt dichte-erreichbar von $(l, N_\epsilon^L(l))$, so wird sie zu der Ergebnismenge *ddrLeaders* hinzugefügt (Z. 6–7). Ansonsten werden, falls $\tilde{l}.type \neq 0$ und $|N_\epsilon^L(\tilde{l})| > 1$ gilt, die Objekte neuverteilt (Z. 8–27). Wird eine neue Umgebung erzeugt oder eine bestehende erweitert, muss sie ggf. in *candidates* eingefügt werden, da sie nun doch direkt dichte-erreichbar sein kann (Z. 16–17 und 20–21).

Die Rauscherkennung erfolgt in Alg. 5 in den Z. 17–21. An dieser Stelle kann sichergestellt werden, dass es sich bei der Leader-Umgebung $(l, N_\epsilon^L(l))$ um Rauschen handelt. Da $(l, N_\epsilon^L(l)) \in L_N$ ist, gehört $(l, N_\epsilon^L(l))$ zu keinem der bisherigen Cluster. Zusätzlich gilt, aufgrund der absteigenden Bearbeitung der Umgebungen aus L_N bzgl. deren Mächtigkeiten (Z. 5 aus Alg. 4), der initialen Partitionierung und der Art der Neuverteilung von Objekten, für alle weiteren Leader-Umgebungen $(l_2, N_\epsilon^L(l_2)) \in L_N$: $|N_\epsilon^L(l_2)| = 1 \wedge (d(l, l_2) > \epsilon \vee l_2.type = 0)$. Somit gehört $(l, N_\epsilon^L(l))$ zum Rauschen.

Die Umsetzungen von isCoreObject und isDirectDensityReachable soll anschließend kurz erklärt werden. In isCoreObject wird für das übergebene Objekt o über die Umgebungen $L_N \cup L_C$ iteriert und gezählt (Variable *counter*), für wie viele Objekte $\hat{o}$ aus den Umgebungen $d(o, \hat{o}) \leq \epsilon$ gilt. Dabei werden Umgebungen mittels Lemma 3 von der Suche ausgeschlossen. Ist o ein Leader einer Leader-Umgebung, so wird *counter* um $|N_\epsilon^L(o)|$ erhöht, ohne eine Distanz zu berechnen. Sollte *counter* $\geq$ *minPts* sein, so wird frühzeitig abgebrochen. Wurde isCoreObject schon einmal mit o aufgerufen, so werden keine Distanzen berechnet und das Ergebnis anhand von *o.type* ermittelt. isDirectDensityReachable wird durch zwei verschachtelte Schleifen über die Leader-Umgebungen $(l, N_\epsilon^L(l))$ und $(l_2, N_\epsilon^L(l_2))$ realisiert, wenn Fall 1 von Lemma 2 anwendbar ist. Ein Objekt $o \in N_\epsilon^L(l)$ kann als eines der verbindenden Kernobjekte p, q (siehe Lemma 2 Fall 1) ausgeschlossen werden, wenn Lemma 3 für die Distanz $d(o, l_2)$ gilt. Für den Fall 2 aus Lemma 2, also $|N_\epsilon^L(l_2)| = 1$, wird für l_2 nur über die Objekte $N_\epsilon^L(l)$ iteriert, wenn Lemma 3 für $d(l, l_2)$ nicht erfüllt ist, um ein Kernobjekt $q \in N_\epsilon^L(l)$ zu finden mit $d(q, l_2) \leq \epsilon$. Vorher muss mittels isCoreObject überprüft werden, ob l_2 ein Kernobjekt ist, um den entsprechenden Fall zu wählen.

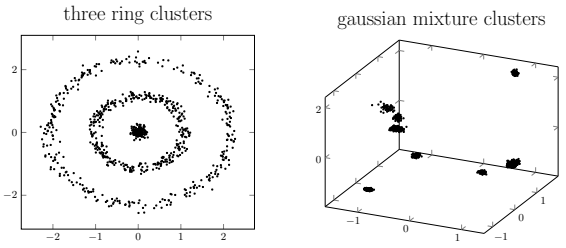


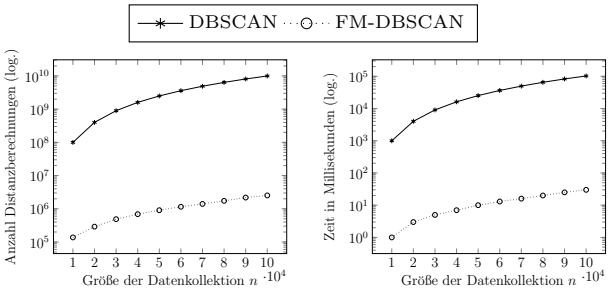
Abb. 3: Evaluierte Datenverteilungen

Aufgrund des Umfangs eines vollständigen Beweises der Exaktheit von FM-DBSCAN soll hier nur die Idee des Beweises erfolgen. Hierbei wird vorausgesetzt, dass computeLeaderPartition, isDirectDensityReachable und isCoreObject korrekt sind. Da im Alg. 6 in *candidates* immer die Leader-Umgebungen gehalten oder aufgrund der Neuverteilung eingefügt werden (Z. 1, 16–17 und 20–21), die potentiell direkt dichte-erreichbar sein können (Lemma 4), werden alle direkt dichte-erreichbaren Umgebungen gefunden. Wie in Abschnitt 4.1 erläutert, folgt daraus, dass alle dichte-erreichbaren Objekte des Leaders der Startumgebung gefunden werden. Somit wird in Alg. 5 (Z. 1–15) ein vollständiger Cluster erzeugt (Lemma 1). Aufgrund der Neuverteilung in Alg. 5 (Z. 22–36) und 6 (Z. 8–27) wird nach endlich viele Schritten für jeden Cluster mindestens eine Leader-Umgebung in L_N existieren, deren Leader ein Kernobjekt ist und zu dem entsprechenden Cluster gehört. Von diesen Umgebungen wird in Alg. 5 (Z. 4–7) ein Cluster erzeugt, sodass wir alle Cluster finden. Die korrekte Erkennung von Rauschen wurde weiter oben erläutert. Somit berechnet FM-DBSCAN dasselbe Ergebnis wie DBSCAN.

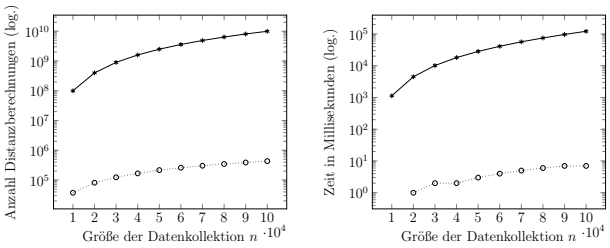
5. EVALUATION

In diesem Abschnitt wird FM-DBSCAN mit DBSCAN verglichen. Beide Algorithmen wurden in C++11 implementiert. Sämtliche Experimente liefen auf einem PC mit einem Intel® Core™ i7-2600K Prozessor, 16 GB RAM und Debian 8. Zur Evaluation wurden zwei synthetische Datenverteilungen herangezogen. Die beiden Datenverteilungen *three ring clusters* (2-dimensional) und *gaussian mixture clusters* (3-dimensional) sind für eine Kollektionsgröße von 1000 Objekten in Abb. 3 dargestellt. Die euklidische Distanz diente als Distanzfunktion zum Vergleich der Objekte. Sämtliche Daten wurden im Hauptspeicher gehalten.

Im ersten Experiment wurde der Einfluss der Kollektionsgröße auf die Effizienz, gemessen anhand der Anzahl der Distanzberechnungen und der Zeit, evaluiert. Hierfür wurden mittels RapidMiner (https://rapidminer.com/) Datenkollektionen in den Größen $n = 10.000, 20.000, \ldots, 100.000$ für beide Datenverteilungen generiert. Über alle n wurden

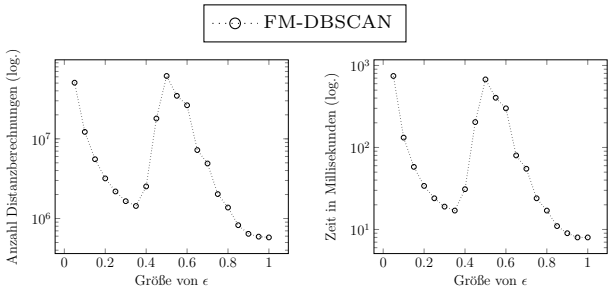


(a) *three ring clusters* ($minPts = 5$, $\epsilon = 0,4$)

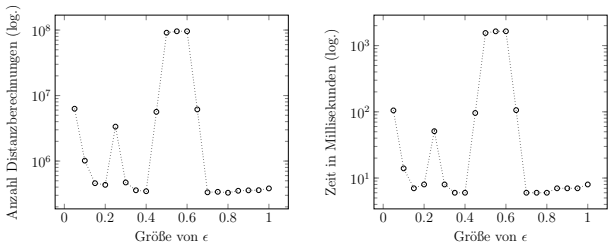


(b) *gaussian mixture clusters* ($minPts = 5$, $\epsilon = 0,2$)

Abb. 4: Clustering-Performanz für variierendes n



(a) *three ring clusters* ($minPts = 5$, $n = 100.000$)



(b) *gaussian mixture clusters* ($minPts = 5$, $n = 100.000$)

Abb. 5: Clustering-Performanz für variierendes ϵ

die selben Werte für ϵ und *minPts* verwendet. Dabei wurden die Werte für ϵ und *minPts* so festgelegt, dass die in Abb. 3 dargestellten Cluster gefunden wurden. Das Experiment wurde für jedes n zehnmal durchgeführt und anschließend die Werte gemittelt. Die Ergebnisse sind in Abb. 4 zu sehen.

Auf beiden Verteilungen konnte im Vergleich zu DBSCAN eine erhebliche Anzahl an Distanzberechnungen eingespart und die Rechenzeit drastisch reduziert werden. Für *three ring clusters* berechnet FM-DBSCAN um einen Faktor 727 bis 3956 weniger Distanzberechnungen als DBSCAN, wodurch FM-DBSCAN um einen Faktor 999 bis 3418 weniger Zeit benötigt. Bei *gaussian mixture clusters* sind die Einsparungen noch deutlich höher. Dort kann ein Faktor 2664 bis 23.075 an Distanzberechnungen eingespart und die Rechenzeit um einen Faktor 1128 bis 17.404 reduziert werden.

Als zweites wurde der Einfluss des Parameters ϵ auf die Effizienz von FM-DBSCAN evaluiert. Auch hier wurde das Experiment zehnmal ausgeführt und die Werte anschließend gemittelt. Die Ergebnisse sind in der Abb. 5 zu finden.

Für *three ring clusters* ist für wachsendes ϵ eine fallende Tendenz sowohl für die Anzahl der Distanzberechnungen, als auch für die Zeit ersichtlich. Lediglich im Bereich $0,4$ bis $0,5$ ist ein Steigen zu erkennen. Dies liegt daran, dass sich die ϵ-Umgebungen zweier Leader aus unterschiedlichen Clustern immer mehr in einem großen leeren Bereich überschneiden und somit die Lemmata 3 und 4 nicht mehr effektiv angewendet werden können. Vereinigen sich diese Cluster aufgrund des größeren ϵ, fallen die Werte wieder. Das gleiche Verhalten tritt auch bei *gaussian mixture clusters* auf, nur öfter, da es dort mehr Cluster gibt.

6. ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde der Clustering-Algorithmus FM-DBSCAN vorgestellt, der für beliebige Distanzfunktionen dasselbe Ergebnis wie DBSCAN liefert. Dabei nutzt FM-DBSCAN das Prinzip der Leader-Umgebungen, um die Objekte zusammenzufassen, sodass anschließend auf den Umgebungen ein Clustering durchgeführt werden kann. Erste Experimente zeigen, dass FM-DBSCAN um ein Vielfaches schneller als DBSCAN ist (Faktor > 990), wodurch FM-DBSCAN für große Datenkollektionen vorteilhaft wird.

In zukünftigen Arbeiten sollen weitere Datenkollektionen (synthetisch und real) evaluiert werden, um die Effizienz von FM-DBSCAN zu bestätigen. Außerdem soll untersucht werden, wie ϵ und *minPts* zu wählen sind, sodass zum einen eine gute Qualität (Clustering-Ergebnis) und zum anderen eine hohe Effizienz erzielt wird. Die durchgeführten Experimente zu ϵ geben hierfür erste Aufschlüsse. Ebenfalls sollen neue Partitionierungsalgorithmen untersucht werden, um die Partitionierung noch weiter zu beschleunigen.

Literatur

[1] Ester, M., H. Kriegel, J. Sander und X. Xu: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* In: *Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, S. 226–231, 1996.

[2] Gan, J. und Y. Tao: *DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation.* In: *Proc. of the 2015 ACM SIGMOD Int. Conf. on Management of Data*, S. 519–530, 2015.

[3] Gunawan, A.: *A faster algorithm for DBSCAN.* Diplomarbeit, Technische Universität Eindhoven, 2013.

[4] Liu, B.: *A Fast Density-Based Clustering Algorithm for Large Databases.* In: *Proc. of the 2006 Int. Conf. on Machine Learning and Cybernetics*, S. 996–1000, 2006.

[5] Mahran, S. und K. Mahar: *Using grid for accelerating density-based clustering.* In: *Proc. of 8th IEEE Int. Conf. on Computer and Information Technology*, S. 35–40, 2008.

[6] Viswanath, P. und V. S. Babu: *Rough-DBSCAN: A fast hybrid density based clustering method for large data sets.* Pattern Recognition Letters, 30(16):1477–1488, 2009.