J. W. Goethe-Universität Frankfurt
Seminar Data Mining WS98/99

Thema:
Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications

von Stefan Steinhaus
(07 November 1999)

Inhaltsverzeichnis

1. Einleitung

Im Bereich des Data-Mining finden heutzutage eine Reihe verschiedener Denkansätze zur Findung von Informationszusammenhängen in Algorithmen Anwendung. Viele dieser Algorithmen verwenden bekannte Methoden der Mathematik oder Statistik und erweitern diese für die eigenen Belange. Eine interessante Anwendung versprechen hierbei vor allem, die aus der Statistik bekannten Methoden der Clusteranalyse.

Die Clusteranalyse erkennt die Gruppenzugehörigkeit von n-dimensionalen Objekten. Unter einem Objekt versteht man in diesem Zusammenhang einen Datenpunkt im n-dimensionalen Raum der durch n Attribute (Koordinaten) beschrieben wird. Die Art und Weise nach der Gruppen festgelegt werden, hängt in erster Linie vom Clusteralgorithmus ab. Die zwei bekanntesten Kategorien sind die partitionierenden und die hierarchischen Clusteralgorithmen. Die partitionierenden Clusteralgorithmen gehen dabei so vor, daß aus n Objekten Repräsentanten für k Cluster gewählt werden und die restlichen Objekte gemäß ihrer Ähnlichkeiten zu einem dieser Cluster zugeordnet werden (Ähnlichkeiten können z.B. durch Distanzmaße berechnet werden). Die hierarchischen Algorithmen nehmen zu Anfang an, daß alle n Objekte jeweils n Cluster darstellen. Diese n Cluster werden dann an Hand von Ähnlichkeiten mit anderen zusammengefaßt, so daß weniger Cluster mit mehr Objekten existieren. Diese werden wiederum zusammengefaßt usw. Am Ende erhält man ein großes Cluster welches alle Objektebeinhaltet.

Die Verarbeitung von großen Datenmengen wird von vielen Clusteralgorithmen nicht zufriedenstellend gelöst. Viele Clusteralgorithmen können vor allem mit hochdimensionalen Objekten nicht mehr effizient umgehen, da dann z.B. Störungen oder gleichverteilte Werte zu falschen Clusterzuordnungen führen können. Auch die Interpretierbarkeit und die Skalierbarkeit der Rohdaten und der Ergebnisse weist bei großen Datenbeständen mit vielen Attributen signifikante Probleme auf.

Aufgabe dieser Seminararbeit wird es daher sein den Clusteralgorithmus CLIQUE* vorzustellen, der diese Probleme angeht und löst. CLIQUE setzt zur Lösung der Probleme auf eine Reduzierung der Dimensionalität, indem es automatisch in Unterräumen mit hoher Objektdichte, die für eine Gruppenbildung besser geeignet sind, nach Clustern sucht. Außerdem stellt der Anstieg der Verarbeitungsgeschwindigkeit in Abhängigkeit mit der Datenmenge und die Reihenfolge in der die Daten verarbeitet werden, einen Schwachpunkt bei vielen Algorithmen dar. Bei CLIQUE nimmt die Verarbeitungsgeschwindigkeit mit der Größe des Datenbestandes in einem linearen Verhältnis ab [siehe Kapitel 4] und zeigt somit im Vergleich zu anderen Algorithmen ein sehr gutes Verhältnis. Zudem ist die Reihenfolge in der die Rohdaten verarbeitet werden, ebenfalls ein signifikantes Problem. Viele Clusteralgorithmen (z.B. die hierarchische Clusteranalyse) sind abhängig von der Eingabereihenfolge der Rohdaten, wird diese geändert so ändern sich auch die Endergebnisse. CLIQUE hingegen ist unabhängig von der Reihenfolge der Daten, erzeugt somit immer gleiche Ergebnisse.

---

* *Der Clusteringalgorithmus CLIQUE (Abkürzung für Clustering in Quest) wurde von Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos und Prabhakar Raghavan im Rahmen eines Data-Mining Projektes bei IBM entwickelt.*

2. Clusterbildung in Unterräumen

Der CLIQUE Algorithmus sucht, wie einige andere Data-Mining Verfahren auch, nach Clustern in niedriger dimensionierten Unterräumen des Originalraumes. Bevor also der CLIQUE Algorithmus im Detail besprochen werden kann, soll dieser Abschnitt nun erst einmal die Grundlagen der Clusterbildung in Unterräumen erklären.

Als erstes stellt sich einem hier die Frage, warum man nach Clustern in Unterräumen sucht bzw. welchen Vorteil dies bringt. Der wichtigste Vorteil der sich aus diesem Verfahren ergibt, ist die Reduzierung der Komplexität des Modells. Dies wirkt sich vor allem positiv auf die Präsentationsmöglichkeit der Ergebnisse, sowie die Interpretierbarkeit aus. Cluster in Räumen mit nur geringer Dimension lassen sich leichter grafisch darstellen und sind in Zahlen ausgedrückt auch leichter zu verstehen. Besonders zu erwähnen sei hierbei auch, daß Unterräume des Originalraumes, im Gegensatz zu neu erstellten Räumen (z.B. durch Linearkombination), noch eine echte Bedeutung haben.

Sei nun eine Menge von Attributen A = {A_1, A_2, ..., A_n} gegeben, so definiert sich ein Raum S (oder auch Unterraum) als $S = A_1 \times A_2 \times ... \times A_n$. Jedes Attribut A_i beschreibt dabei jeweils die Komponenten einer Dimension des Raumes S. Zur Clusterfindung in Unterräumen nehmen wir nun einen Algorithmus an, der diese an Hand von Regionen mit höherer Punktdichte als deren Umgebung erkennt. Die Punktdichte stellen wir fest, indem wir jede Dimension in ξ Intervalle $u_i = [l_i, h_i)$ gleicher Länge einteilen. Durch die Einteilung solcher Intervalle wird der Unterraum in n-dimensionale Würfel u = {u_1, u_2, ..., u_n} gleichen Volumens aufgeteilt. Die Dichte ist nun durch einfaches Abzählen der Punkte V = {v_1, v_2,..., v_n} je Würfel leicht feststellbar. Zur Entscheidung, ob die so festgestellte Dichte eines Würfels „hoch" oder „niedrig" ist, führen wir den Eingabeparameter τ ein. Ist die Dichte eines Würfels u_i größer als τ (die Anzahl der darin enthaltenen Punkte), so wird dieser als „hohe" Dichte erkannt.

Ein Unterraum des Raumes S wird, wie bereits angedeutet, äquivalent definiert als $S_k = A_{t_1} \times A_{t_2} \times ... \times A_{t_k}$, wobei k < n ist und für t_i gilt $t_i < t_j$ und i < j. Ein Cluster in einem Unterraum S_k ist dann festgelegt als maximale Anzahl von zusammenhängenden k-dimensionalen Würfeln mit hoher Dichte. Man versteht zwei k-dimensionale Würfel $u_1 = \{r_{t_1}, ..., r_{t_k}\}$ und $u_2 = \{r'_{t_1}, ..., r'_{t_k}\}$ als zusammenhängend, wenn für mindestens k-1 Dimensionen eine Überstimmung der Intervalle $r_{t_j} = r'_{t_j}$ existiert und für ein k-tes Intervall entweder $h_{t_k} = l'_{t_k}$ oder $h'_{t_k} = l_{t_k}$ gilt. Eine Region R kann nach dieser Definition auch als DNF Ausdruck über die Intervalle dargestellt werden.

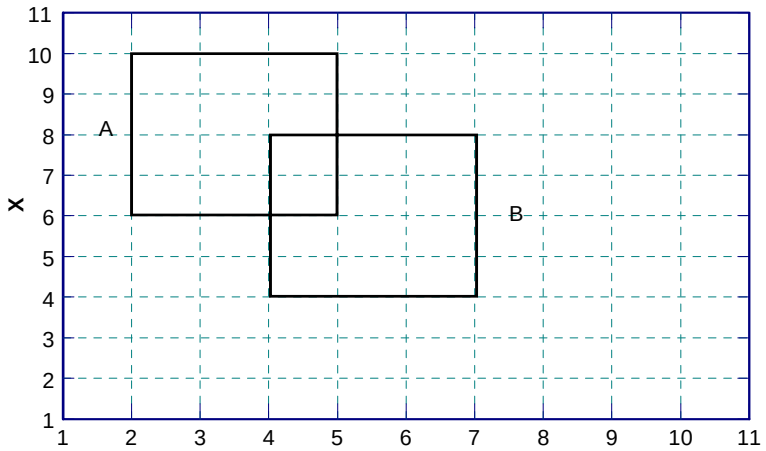

Abbildung 2.1: Beispiel zweier Regionen.

Eine minimale Beschreibung eines Clusters ist eine nichtredundante Überdeckung von maximalen Regionen. Unter maximale Regionen versteht man die Vereinigungen der Würfel u_i in der Form, daß für ein Cluster C und eine maximale Region R immer gilt $R \cap C = R$. Es gibt somit keinen Würfel u_i der zwar zum Cluster C nicht aber zu einer Region R gehört.

Abbildung 2.1 zeigt wie ein Cluster C aus zwei Regionen A und B bestehen kann. Die Beschreibung der Regionen lautet dabei

$$A = (2 \le Y < 5) \quad (6 \le X < 10)$$

und

$$B = (4 \le Y < 7) \quad (4 \le X < 8)$$

und ergibt somit eine DNF von

$$R = ((2 \le Y < 5) \quad (6 \le X < 10) \quad ((4 \le Y < 7) \wedge (4 \le X < 8)).$$

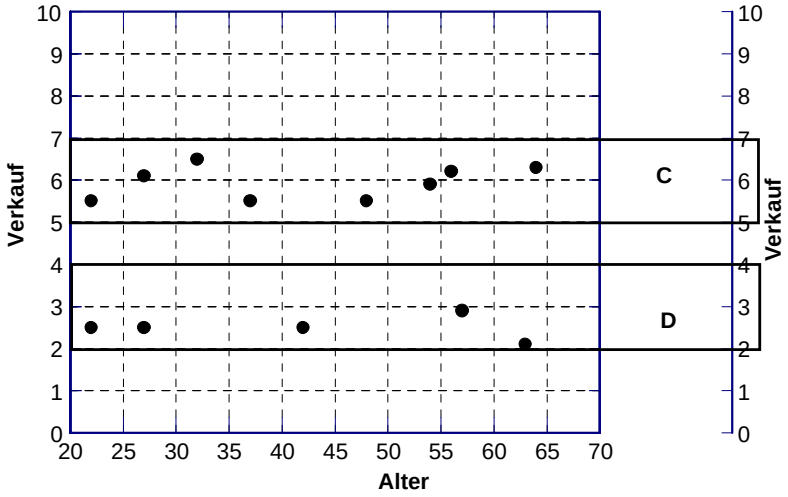

Abbildung 2.2: Beispiel für Clusterbildung in Unterräumen.

Abbildung 2.2 zeigt bei vorgegebenem ξ ein Beispiel für die Clusterfindung in Unterräumen. Sei der Eingabeparameter $\tau = 20\%$, dann ist bei vorgegebener Intervalleinteilung kein Cluster im Ursprungsraum zu finden, da alle Bereiche gemäß dem Eingabeparameter nicht über eine „hohe" Punktdichte verfügen. Abgebildet auf den Unterraum Verkauf können allerdings zwei Cluster C und D gefunden werden. Im Unterraum Alter ist ebenfalls kein Cluster zu finden, da hier wieder die Punktdichte zu gering ist.

3. Der CLIQUE Algorithmus

Der CLIQUE Algorithmus benötigt zur Erstellung des gewünschten Ergebnisses drei Arbeitsschritte. Der erste Schritt ist, wie bereits der vorhergehende Abschnitt vermuten läßt, die Identifizierung von Unterräumen die Cluster enthalten. Der zweite Schritt dient dann zur Findung der Cluster und im dritten Schritt werden dann die minimalen Beschreibungen für diese Clusters erstellt. Die nachfolgenden Unterabschnitte werden sich jeweils getrennt mit diesen Arbeitsschritten befassen.

3.1. Identifizierung von Unterräumen

Zur Identifikation von Unterräumen die Cluster enthalten verwendet CLIQUE einen Bottom-Up Algorithmus. Dieser basiert auf der Annahme, daß, wenn eine Punktmenge S ein Cluster im k-dimensionalen Raum ist, S auch Teil eines Clusters in einer (k-1)-dimensionalen Projektion dieses Raumes ist.

Beweis der Annahme: Beschreiben die Punkte S ein k-dimensionales Cluster C, so haben gemäß Definition die Einheiten u eine hohe Dichte (nach Schwellenwert τ). Alle Projektionen von u in C haben ebenfalls eine hohe Dichte, da deren Punkte wieder in u liegen. Da außerdem die Einheiten des Clusters, und somit auch der Projektionen, zusammenhängend sind, kann man folgern, daß alle Punkte in C auch im selben Cluster jeglicher (k-1)-dimensionalen Projektionen liegen.

Zurückkommend auf den in CLIQUE angewendeten Algorithmus kommt dies nun wie folgt zur Anwendung. Zuerst werden durch die bereits vorgestellte Methode, dichte Einheiten in einem (k-1)-dimensionalen Raum identifiziert. Die Menge aller (k-1)-dimensionalen Einheiten mit hoher Dichte nennen wir D_{k-1}. Da D_{k-1} während des Durchlaufens des Algorithmuses mehrmals benötigt wird und auf Grund seiner Größe nicht immer im Hauptspeicher gehalten werden kann, muß dieser auf der Festplatte gespeichert werden. Dadurch wird unter anderem das Arbeiten mit beliebig großen Datenbeständen ermöglicht und zudem Pufferüberläufe vermieden.

Angenommen die Attribute des (k-1)-dimensionalen Raumes liegen geordnet vor, dann kann jetzt mit Hilfe von D_{k-1} an die Auswahl der sogenannten „Kandidaten" C_k gegangen werden. Die „Kandidaten" C_k sind Einheiten mit hoher Dichte im k-dimensionalen Raum mit deren Hilfe und unter Verwendung der ursprünglichen Annahme, wir auf für uns interessante Unterräume zurückschließen können die Cluster enthalten. Mit Hilfe des folgenden Pseudocodes lassen sich die „Kandidaten" C_k generieren:

insert into C_k
select $u_1 \cdot [l_1, h_1), u_1 \cdot [l_2, h_2), ..., u_1 \cdot [l_{k-1}, h_{k-1}), u_2 \cdot [l_{k-1}, h_{k-1})$
from $D_{k-1} \cdot u_1, D_{k-1} \cdot u_2$
where $u_1 \cdot a_1 = u_2 \cdot a_1, u_1 \cdot l_1 = u_2 \cdot l_1, u_1 \cdot h_1 = u_2 \cdot h_1,$
$u_1 \cdot a_2 = u_2 \cdot a_2, u_1 \cdot l_2 = u_2 \cdot l_2, u_1 \cdot h_2 = u_2 \cdot h_2, ...,$
$u_1 \cdot a_{k-2} = u_2 \cdot a_{k-2}, u_1 \cdot l_{k-2} = u_2 \cdot l_{k-2}, u_1 \cdot h_{k-2} = u_2 \cdot h_{k-2},$
$u_1 \cdot a_{k-1} < u_2 \cdot a_{k-1}$

Ziel dieses Algorithmus ist es durch Verknüpfung zweier (k-1)-dimensionaler Einheiten, die eine „gemeinsame Oberfläche" im (k-2)-dimensionalen haben, Einheiten mit hoher Dichte im k-dimensionalen Raum zu erzeugen. Die Erzeugung von k-dimensionalen Einheiten wird dabei dadurch gewährleistet, indem die (k-1)ten Attribute der beiden (k-1)-dimensio-

nalen Einheiten jeweils andere Dimensionen beschreiben. Dieser Pseudocode wird nun solange durchlaufen bis alle verfügbaren C_k festgestellt wurden. Diese werden dann ebenfalls auf der Festplatte zwischengespeichert.

Im nächsten Schritt wird nun von der zuvor bewiesenen Annahme Gebrauch gemacht. Für jede dichte Einheit in C_k müssen also alle Projektionen in (k-1)-Dimensionen ebenfalls eine hohe Dichte haben. Dies bedeutet, daß für jede Einheit mit hoher Dichte in C_k alle seine in (k-1)-Dimensionen verfügbaren Projektionen in C_{k-1} liegen müssen. Alle Einheiten aus C_k für die dies nicht gilt werden gelöscht. Alle verbliebenen Einheiten sind Elemente von Clustern und werden zur weiteren Verarbeitung benötigt.

Der so verwendete Bottom-up Algorithmus hat allerdings einen gravierenden Nachteil bzgl. dessen Laufzeitverhaltens. Bei der Kandidatensuche werden jeweils 2 Einheiten mit hoher Dichte kombiniert. Da immer k-1 Attribute aus einem k-dimensionalen Raum genommen werden, gibt es somit 2^k Kombinationsmöglichkeiten. Die Komplexität der Kandidatenwahl beträgt somit $O(2^k)$. Danach werden im schlimmsten Fall für m Einheiten (wenn m die Punktanzahl gleich der Einheitenanzahl ist) k Projektionen im (k-1)-dimensionalen überprüft, so daß hierfür noch einmal ein Aufwand von O(m*k) anfällt. Die Komplexität des Bottom-up Algorithmus ist somit exponentiell und beträgt$O(2^k+m*k)$.

Eine Verbesserung des Algorithmuses kann durch Verwendung der MDL* Methode erreicht werden. Diese dient zur Reduzierung der Anzahl der dichten Einheiten, auf die die in „interessanten" Unterräumen liegen. Die Idee hinter diesem Algorithmus ist dabei, daß zur Erzeugung der Kandidaten C_k nur Unterräume verwendet werden, in denen sich mit hoher Wahrscheinlichkeit Cluster befinden. Es werden somit bereits vor der Kandidatenerzeugung die Anzahl der Unterräume und somit auch der dichten Einheiten nach einem bestimmten Kriterium reduziert. Die Anzahl der Kandidaten und der Unterräume die Cluster enthalten wird somit ebenfalls verringert.

Als Kriterium zur Auswahl „interessanter" Unterräume wird eine Kodierung erstellt, deren Minimum über die Auswahl entscheidet. Hierzu benötigen wir zuerst einmal die Anzahl der Datenpunkte x_{S_j} pro Einheit u_i bzw. pro Unterraum S_j. Die Unterräume S_j werden dann nach x_{S_j} in absteigender Folge sortiert. Zur Aufteilung der Menge aller S_j in eine Menge von akzeptierten Unterräumen I und einer Menge von nichtverwendeten Unterräumen P, wird nun das Minimum für folgendeKodierung gesucht:

$$CL(i) = \log_2(\mu_I(i)) + \sum_{1\le j\le i} \log_2\left(\left|x_{S_j} - \mu_I(i)\right|\right) + \log_2(\mu_P(i)) + \sum_{i+1\le j\le n} \log_2\left(\left|x_{S_j} - \mu_P(i)\right|\right)$$

Hierbei stellt μ_I und μ_P jeweils den Mittelwert der Anzahl an Datenpunkten in den Mengen I und P dar. μ_I und μ_P definieren sich als

$$\mu_I(i) = \left\lceil \frac{\sum_{1\le j\le i} x_{S_j}}{i} \right\rceil \qquad \text{und} \qquad \mu_P(i) = \left\lceil \frac{\sum_{i+1\le j\le n} x_{S_j}}{n-i} \right\rceil.$$

Als Einflußfaktor wird in der Funktion CL(i) der Speicherplatz, der zur Speicherung der Mittelwerte und der Differenzen benötigt wird, gewichtet bzw. dieser gibt die Länge des Kodes an. Dies erfolgt durch die Logarithmierung zur Basis 2, da man davon ausgeht daß die Werte ganzzahlig sind und einen Speicherbedarf zur Basis 2 in Bit benötigen. Ein Minimum wird für diese Funktion gefunden, wenn beide Mengen I und P ausgewogen aufgeteilt sind. Das für das Minimum festgestellte i stellt nun die Grenze zwischen I und P für alle sortierten S_i dar, somit also die gewünschte Aufteilung und reduzierte Anzahl an Einheiten für die Kandidatensuche.

---

* *MDL, Abkürzung für „Minimal Description Length".*

Als problematisch erweist sich bei diesem verbesserten Algorithmus allerdings der Tatbestand, daß eventuell auch Unterräume verworfen werden die Cluster enthalten. Es können also Cluster nicht gefunden werden die für den ursprünglichen Algorithmus existent sind. Es ist somit auf jeden Fall wichtig sich vor der Anwendung des „verbesserten" Algorithmuses über dessen Folgen bewußt zu sein, da ansonsten Ergebnisse eventuell falsch interpretiert würden.

3.2. Identifizierung der Cluster

Sind nun in Abschnitt 3.1 die gewünschten Unterräume ausgewählt worden in denen sich Cluster befinden, so ist es nun im nächsten Schritt notwendig diese Cluster zu identifizieren. Nach Definition sind Cluster Regionen in denen Einheiten mit hoher Dichte über eine gemeinsame Oberfläche miteinander verbunden sind. Einheiten mit hoher Dichte die nicht mit einer Einheit aus der Region verbunden sind, sind nicht Bestandteil dieses Clusters. Exakt definiert bedeutet dies für Regionen D^i, daß alle Einheiten u^i in D^i zusammenhängen. Es existiert aber keine Einheit u^j außerhalb von D^i die mit einer Einheit aus D^i zusammenhängt, da diese ansonsten nach Definition innerhalb von D^i liegen muß.

Diese Definition von Clustern läßt sich ohne Probleme mit Graphen vergleichen bei denen jeder Knoten zu einem Graph gehört, wenn diese von einem anderen Knoten des Graphen aus erreichbar ist. Zur Suche von Clustern kann man daher einfach einen Tiefensuchealgorithmus (DFS) verwenden, da dieser jeden Knoten (jede Einheit) eines Graphen (eines Clusters) durchläuft. Der Pseudokode für einen solchen DFS Algorithmus angewendet in CLIQUE läßt sich wie folgt definieren:

Eingabe: *Ausgangsknoten der Tiefensuche* $u = \{[l_1,h_1), \ldots , [l_k,h_k)\}$
Clusternummer

```
dfs(u,n)
u.num = n
for (j = 1; j < k; j++) do begin
        // Suche im linken Nachbarn von u //
        u^l = {[l_1,h_1), ... , [(l_j^l ),(h_j^l )) , ... , [l_k,h_k)}
        if (u^l == hohe Dichte) and (u^l.num == undefiniert)
                dfs(u^l,n)
        // Suche im rechten Nachbarn von u //
        u^r = {[l_1,h_1), ... , [(l_j^r ),(h_j^r )) , ... , [l_k,h_k)}
        if (u^r == hohe Dichte) and (u^r.num == undefiniert)
                dfs(u^r,n)
end
```

Nach dem DFS Algorithmus wird ausgehend von einer Anfangseinheit rekursiv jede mit dieser Einheit zusammenhängende Einheit besucht und mit der Nummer des aktuellen Clusters markiert. Werden keine weiteren mit zusammenhängenden Einheiten mehr gefunden, so sind alle Einheiten eines Clusters definiert. Es wird dann mit der nächsten noch nicht markierten Einheit fortgefahren. Dies wird solange durchgeführt bis alle Einheiten mit hoher Dichte einem entsprechenden Cluster zugewiesen worden sind.

Da davon auszugehen ist, daß die Anzahl der Einheiten deren Dichte höher als der Schwellenwert τ ist (ausgehend vom Originaldatenbestand), relativ gering ist, kann man sagen daß die Clusteridentifikation komplett im Speicher durchgeführt werden kann. Die Zugriffe auf die einzelnen Einheiten können daher recht schnell stattfinden. Mit Vorsicht ist

dabei zu bedenken, daß der Schwellenwert τ eventuell benutzerdefiniert sein kann. In diesem Fall kann ein sehr kleiner Schwellenwert (oder großer Wert in Prozent) zu einer zu großen Anzahl an Einheiten mit hoher Dichte führen. Eine Fehlbesetzung des Schwellenwertes, also ein Benutzerfehler, kann sich also sehr negativ auf die Laufzeit des DFS Algorithmuses auswirken (da dann auf der Festplatte gearbeitet werden muß) und führt zudem zu ungünstigen Clusterbildungen.

3.3. Erstellung einer minimalen Clusterbeschreibung

Im letzten Arbeitsschritt werden nun, nach der Definition aus Abschnitt 2, minimale Beschreibungen der Cluster erstellt. Dies ist vor allem notwendig, um eine möglichst einfache, allgemein verwendbare Beschreibung der Cluster zu erhalten (im Gegensatz zur Statistik, wo exakte numerische Zuweisungen der Cluster erwartet werden). Hierzu werden als Eingabe alle k-dimensionalen Einheiten der Cluster aus dem Unterraum S zur Verfügung gestellt. Aufgabe ist es nun Regionen mit möglichst maximaler Ausdehnung zu erstellen, so daß nur eine minimale Anzahl an Regionen benötigt wird um jeweils ein Cluster abzudecken. Diese Regionen müssen aber alle dichten Einheiten enthalten, da sie sonst gegen die Definition aus Abschnitt 2 verstoßen würden.

Zur Erstellung von minimalen Clusterbeschreibungen bietet sich eine Aufteilung in zwei Arbeitsgänge auf. Im ersten Durchgang sollten maximale Regionen erstellt werden und im zweiten Durchgang sollten die redundanten Überdeckungen, die im ersten Durchgang entstanden sind, beseitigt werden.

Für die Zuweisung von maximalen Regionen wird ein greedy Algorithmus verwendet, da dieser der beste bekannte Algorithmus für diese Art von Problemen ist. Dies gilt insbesondere in Hinblick auf hochdimensionale Unterräume. Die Vorgehensweise des Algorithmus sieht dabei wie folgt aus. Man nehme aus einem Cluster C eine Einheit u_1 als Anfangspunkt. Von u_1 ausgehend soll nun eine maximale Region R_1 konstruiert werden. Hierfür wird zufallsmäßig in einer beliebigen Dimension angefangen. In der ausgewählten Dimension des Unterraumes S wird nun die Region R_1 durch weitere, mit der Einheit u_1 links oder rechts zusammenhängenden Einheiten erweitert. Dieser Erweiterungsprozeß läuft solange bis es keine neuen Einheiten links oder rechts der bereits in R_1 befindlichen Einheiten gibt, die noch mit diesen zusammenhängen. R_1 ist somit auf der Ebene der ausgewählten Dimension maximal. Nach selbigem Schema arbeitet man nun mit der nächsten Dimension weiter und sucht auch dort nach zusammenhängenden Einheiten „links" und „rechts" der bereits inkludierten Einheiten. Sind alle k Dimensionen durchlaufen worden, so ist R_1 eine maximale Region. Abbildung 3.1 zeigt an Hand einer Region R_1 wie der Algorithmus funktioniert.

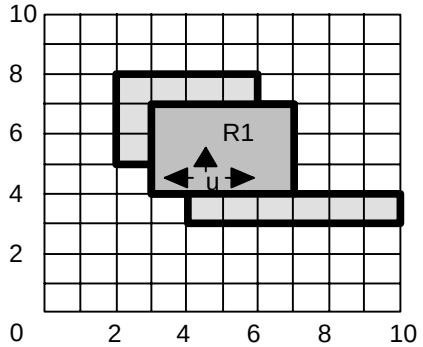

Abbildung 3.1: Funktionsweise des Greedy-Algorithmus.

Sind zu diesem Zeitpunkt noch Einheiten des Clusters C (also zusammenhängende Einheiten) nicht in R_1 eingebunden, so fährt der Algorithmus mit einer zweiten Region R_2 fort und maximiert diese von einem Anfangspunkt u_2 aus. Der ganze Prozeß der Erstellung von Regionen läuft nun solange bis es keine Einheiten in C mehr gibt, die nicht zumindest einer Region R_i gehören (leider können nach diesem Prozeß Einheiten auch zu mehreren Regionen R_i gehören).

Für die praktische Anwendung des Greedy Algorithmus in CLIQUE ist allerdings zu bedenken, daß das Laufzeitverhalten des Algorithmuses relativ schlecht ist. Schaut man sich den Algorithmus im Detail an, so stellt man fest, daß jede in einer Region R befindliche Einheit u mindestens einmal angesprochen werden muß. Außerdem muß von jeder Einheit ausgehend auch dessen linker und rechter Nachbar pro Dimension angesprochen werden. Man kann also davon ausgehen, daß die Zugriffe auf die Einheiten pro Region durch $O(2k|R|)$ begrenzt sind, wobei k die Anzahl der Dimensionen angibt und $|R|$die Anzahl an Einheiten für die Region R ist. Bezogen auf die Gesamtanzahl an Zugriffen zur Festlegung aller benötigten maximalen Regionen kann man weiterhin folgern, daß die Anzahl an Regionen in C durch O(n) begrenzt ist. n ist hierbei die Anzahl an dichten Einheiten im Cluster C. Daraus läßt sich für das gesamte Cluster eine Begrenzung der Anzahl der Zugriffe auf die Einheiten durch $O(n^2)$ angeben.

Da durch die Erstellung der maximalen Regionen viele Einheiten zu mehreren Regionen zugeordnet wurden, ist es jetzt noch nötig diese redundanten Überdeckungen zu eliminieren. Hierzu ist es nötig alle minimalen Regionen, die Einheiten enthalten die bereits durch andere Regionen abgedeckt werden, zu entfernen.

Um diese Arbeit zu erledigen werden für CLIQUE zwei heuristische Algorithmen vorgeschlagen die beide eine Sortierung der Regionen R voraussetzen. Als erstes müssen also die Regionen der Größe nach sortiert (Größe bedeutet in diesem Fall die Anzahl der abgedeckten Einheiten) werden. Der Aufwand für eine optimale Sortierung ist, wie allgemein bekannt, O(n log n). Nun gibt es zwei gegensätzliche Methoden wie die redundanten Regionen entfernt werden können, beide Methoden haben aber auf jeden Fall denselben Kostenaufwand. Die erste Methode geht davon aus, aus einer Liste aller Regionen minimale redundante Regionen zu entfernen. Hierbei wird bei der kleinsten Region angefangen, überprüft ob deren Einheiten bereits vollständig von anderen Regionen abgedeckt werden und falls ja werden diese aus der Liste entfernt. Wird dieser Algorithmus für alle Regionen durchgeführt, so bleiben am Ende nur Regionen übrig, die nicht redundant sind. Die zweite Methode geht einen umgekehrten Weg und geht von einem leeren Cluster aus. In diesem Cluster werden, angefangen von der größten Region, versucht alle Einheiten durch möglichst große Regionen abzudecken. Zuerst wird also die größte Region eingefügt, dann die nächstkleinere soweit diese nicht redundant zu der bereits eingefügten ist und dann wieder die nächstkleinere und so weiter. Der Algorithmus läuft solange bis alle Regionen in das Cluster eingepaßt wurden. Alle Regionen die nicht eingepaßt werden konnten, da diese Redundanzen erzeugt hätten, werden verworfen.

Beide Methoden gewährleisten, daß ausschließlich große Regionen zur Abdeckung der dichten Einheiten verwendet werden. Der Aufwand der für diese Methoden nötig ist, ergibt sich aus der Anzahl der Zugriffe pro Region. Für jede Region R_i müssen alle bereits in das Cluster eingefügten Regionen gescannt werden oder nach der ersten Methode alle noch in der Liste befindlichen. Da die darin enthaltenen dichten Einheiten durch n angegeben sind, läßt sich dadurch eine obere Grenze für die Zugriffe mit $O(n^2)$ angeben. Leider ist die Methode der Auffüllung von leeren Clustern mit Regionen technisch schwer zu handhaben,

insbesondere je höher die Dimensionalität der Daten ist, so daß diese in aller Regel in der Praxis nicht verwendet wird.

Abschließend soll eine stochastische Analyse darüber Aufschluß geben wie gut die Abdeckung der dichten Einheiten im Vergleich zum Optimum ist. Nehmen wir τ als Schwellenwert für die Dichte der Einheiten und die Anzahl der Datenpunkte pro Einheit als Zufallsvariable, so können wir mit einer Wahrscheinlichkeit von p davon ausgehen daß eine Einheit unabhängig und dicht ist. Umgekehrt gesehen bedeutet diese Definition, daß wenn Einheiten eine sehr hohe Dichte haben, die Wahrscheinlichkeit p recht klein wird. Da CLIQUE aber vor allem für Datenbestände mit einer relativ hohen Dichte vorgesehen ist, kann man in der Regel von kleinen Wahrscheinlichkeiten ausgehen. Geht man davon aus, daß die Wahrscheinlichkeit p approximativ durch $p = \frac{1}{2d} - \varepsilon$ festgelegt ist (d ist die Dimension des Datenbestandes), wobei $0 < \varepsilon < \frac{1}{2d}$ ist, so wird angenommen, daß die Qualität der erreichten Überdeckung innerhalb eines konstanten Faktors um das Optimum liegt.

Beweis: Sei die Wahrscheinlichkeit für die Größe i eines Clusters durch a^{-i} festgelegt (a ist nur von ε abhängig). Bringen wir diese nun in Verbindung mit den maximalen Regionen eines Clusters und gesehen aus der Perspektive aller Cluster, so läßt sich die Wahrscheinlichkeit als $\sum_{Cluster} \sum_{i} ia^{-i}$ ausdrücken. Da die Wahrscheinlichkeit auf ein Cluster bezogen immer durch einen konstanten Faktor θ begrenzt ist (also $\sum_{i} ia^{-i} \leq \theta$ ist) gilt:

$$\sum_{Cluster} \sum_{i} ia^{-i} \leq \theta n$$

n stellt hierbei die Anzahl der gefunden Cluster dar. Somit steht fest, daß die erreichte Clusterbeschreibung bzw. die Größe der Überdeckungen nur um einen konstanten Faktor um das Optimum variieren kann.

4. CLIQUE in der Praxis

Nach der Entwicklungsphase von CLIQUE wurden von R. Agrawal, J. Gehrke, D. Gunopulos und P. Raghavan einige anwendungsorientierte Tests durchgeführt um festzustellen wie sich CLIQUE bzgl. seiner Laufzeit verhält und vor allem um die Effizienz der Clustersuche festzustellen. Für diese Tests wurde mit einem künstlich erzeugten Datenbestand gearbeitet, deren Eigenschaften und Cluster bereits vor dem Test feststanden. Auf diese Art und Weise konnten die Ergebnisse der Tests einfach mit den tatsächlichen Begebenheiten verglichen und bewertet werden.

Für die künstlich erzeugten Daten wurde als Grundvoraussetzung definiert, daß die Werte für alle Attribute im Wertebereich [0,100] liegen sollten. Alle Werte werden durch einen Zufallsgenerator nach einer unabhängigen Normalverteilung erzeugt. Die Werte der Cluster sollen alle in einem hyperdimensionalen Rechteck liegen während hingegen die restlichen Werte über den gesamten Defintionsraum verteilt werden können. Eine weitere Bedingung, die zur Simulation einer möglichst praxisnahen Anwendung notwendig ist, ist die Festlegung, daß mindestens 10% aller Punkte als zufällige „Störung" zwischen den Clustern verteilt sein müssen. Alle Tests wurden mit einer Intervalleinteilung von $\xi = 10$ ausgeführt.

4.1. Eigenschaften und Laufzeitverhalten von CLIQUE

Laufzeitverhalten in Abhängigkeit zur Größe des Datenbestandes:

Ein Test der Laufzeit in Abhängigkeit von der Größe des Datenbestandes ergab, wie in Abbildung 4.1 ersichtlich, ein lineares Verhalten. Verwendet wurde für diesen Test ein 50-dimensionaler Datenbestand von 100.000 Datensätzen bis zu 500.000 Datensätzen, der 5 Cluster in 5 verschiedenen 5-dimensionalen Unterräumen enthält. Das Laufzeitverhalten erwies sich in diesem Test als linear, da sich das Größenverhältnis der Suchdurchläufe durch die Datenbestände nicht änderte. Dies ist darauf zu beziehen, daß nur der Arbeitsaufwand in Arbeitsschritt 1 von der Vergrößerung des Datenbestandes betroffen ist, nicht aber der in Schritt 2 und 3. Die Arbeit in Schritt 2 und 3 bezieht sich bei alleiniger Vergrößerung des Datenbestandes immer noch auf dieselbe Anzahl an Unterräumen.

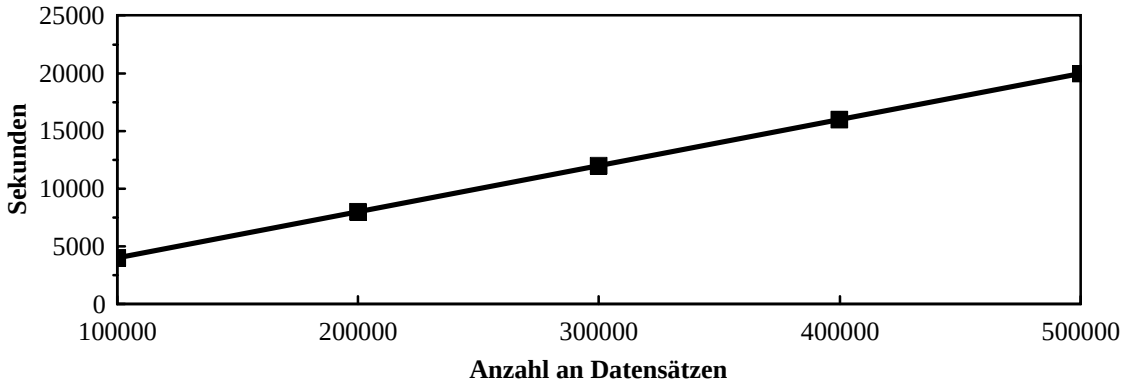

Abbildung 4.1: Laufzeitverhalten in Abhängigkeit zur Größe des Datenbestandes

Laufzeitverhalten in Abhängigkeit zur Dimension des Datenbestandes:

Wird die Dimension eines Datenbestandes anstatt deren Größe erhöht, so ergibt sich ein anderes Laufzeitverhältnis. Für diese Zwecke wurde wieder ein Datenbestand von 100.000 Datensätzen verwendet, dessen Dimension von 10-dimensional bis 100-dimensional gesteigert wurde. Es wurden auch hier wieder 5 Cluster in 5 verschiedenen 5-dimensionalen

Unterräumen definiert. Wie Abbildung 4.2 zeigt ergab die Durchführung dieses Tests, daß sich die Laufzeit im Verhältnis $O(d^5)$ erhöht. d stellt hierbei die Dimension des Datenbestandes dar und 5 ist die Dimension des Unterraumes der die Cluster enthält.

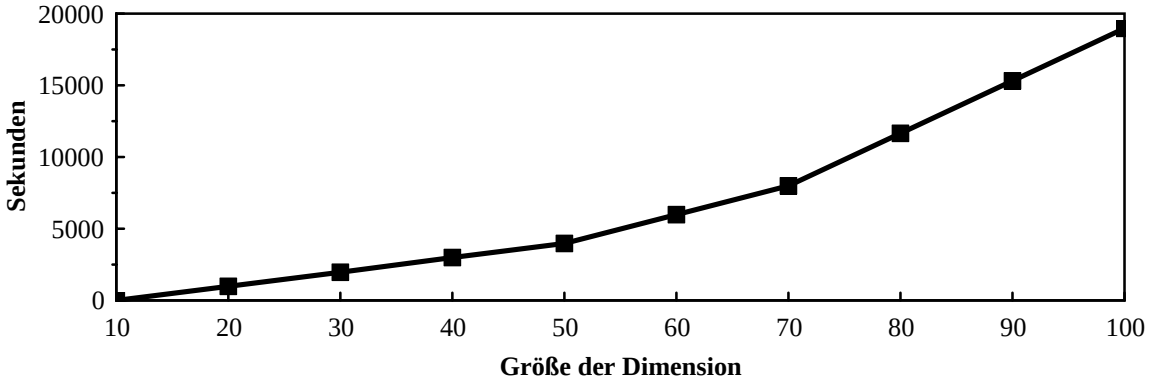

Abbildung 4.2: Laufzeitverhalten in Abhängigkeit zur Dimension des Datenbestandes

Bei der Durchführung des Testes wurde allerdings auch ein weiterer Effekt beobachtet, der auf den MDL Algorithmus zurückzuführen ist. Wie in Abbildung 4.3 gezeigt werden durch den MDL Algorithmus viele niedrigdimensionale Unterräume verworfen (in diesem speziellen Fall 86% aller 2-dimensionalen Unterräume und 38% aller 3-dimensionalen Unterräume). Es besteht somit die Gefahr, daß auch Unterräume verworfen werden die Cluster enthalten.

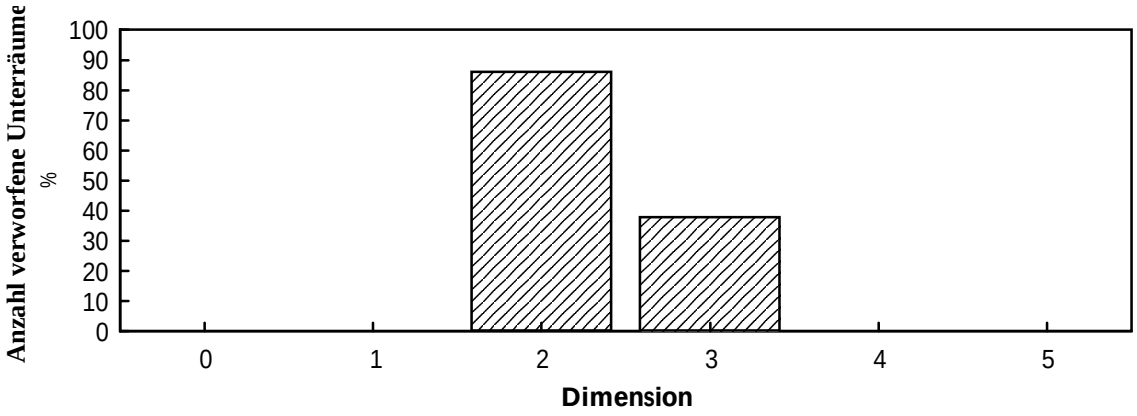

Abbildung 4.3: Anzahl verworfener Unterräume

Laufzeitverhalten in Abhängigkeit zur Dimension der Unterräume mit Clustern:

Um das Laufzeitverhalten in Abhängigkeit zur Dimension der Unterräume mit Clustern zu testen, wurde wieder ein 50-dimensionaler Datenbestand bestehend aus 100.000 Datensätzen erzeugt. Diesmal wurden Cluster in Unterräumen der Dimension 3 bis 10 verteilt, wobei die maximale Dimension dieser Unterräume in diesem Rahmen jeweils erhöht wurde. Das Laufzeitverhalten ist aus Abbildung 4.4. ersichtlich und ergab $O(mk+c^k)$, wobei m die Anzahl der Datensätze ist, c eine Konstante und k die maximale Dimension der Unterräume ist. Die so erhaltene Laufzeit entspricht dem Algorithmus aus Abschnitt 3.1. Es sei hierbei allerdings zu bemerken, daß für diesen Test, zusätzlich zur Anhebung der Dimension der Unterräume, auch der Schwellenwert τ erhöht werden mußte. Grund hierfür ist, daß, je höher die Dimension eines Unterraumes ist, um so niedriger wird die Dichte der Punkte je

Einheit. Es muß also auf jeden Fall auch der Schwellenwert τ erhöht werden, da ansonsten gewisse Cluster bei der Suche eventuell nicht erkannt werden.

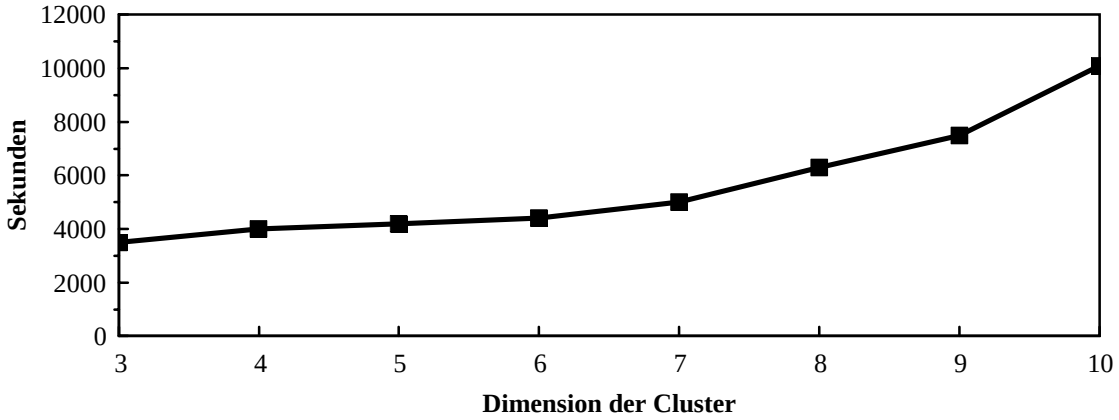

Abbildung 4.4: Laufzeitverhalten in Abhängigkeit zur Dimension Unterräume mit Clustern

Zusätzlich zu den Tests mit künstlich erzeugten Datenbeständen, wurde ebenfalls ein Test mit echten, aus der praktischen Anwendung stammenden Datenbeständen durchgeführt. Hierzu wurden Daten von Versicherungen, Banken und aus dem Einzelhandel verwendet, die eine Dimension von 9, 7, 24 und 52 hatten. Die Ergebnisse aus der Clustersuche in diesen Datenbeständen konnten nicht genauso wie künstliche Datenbestände gehandhabt werden, da in diesen die Cluster nicht von vornherein feststanden. Es läßt sich in diesem Fall allerdings etwas über die Aussagekraft der gefundenen Cluster sagen. Im Test wurden in jedem Datenbestand Cluster gefunden, die bezogen auf die Herkunft eine echte Bedeutung hatten.

4.2. Vergleich der Eigenschaften von CLIQUE zu anderen Verfahren

Zusätzlich zu den Tests von CLIQUE wurden von IBM auch einige Vergleiche zu den Algorithmen BIRCH, DBSCAN und SVD durchgeführt. Die Absicht dieser Tests war es festzustellen ob auch Algorithmen die für die Clusterfindung in kompletten Räumen entwickelt wurden für die Suche in Unterräumen geeignet sind. BIRCH verwendet zur Suche von Clustern einen sogenannten „Mode seeking" Algorithmus, der sich als Erweiterung zum K-means Clusteralgorithmus versteht. Die K-means Clusteralgorithmus geht dabei so vor, daß eine gewisse Anzahl an Anfangspunkten vom Benutzer vorher definiert werden muß. Diese Anfangspunkte stellen jeweils den ersten Punkt eines Clusters dar. Nun werden alle verbleibenden Datenpunkte gemäß ihrer Distanzen zu den Clustern, diesen zugeordnet. Die Clusterzugehörigkeit entscheidet sich jeweils durch das Minimum der Quadratsummen der Distanzen (nähere Definitionen siehe A K-Means Clustering Algorithm (1979) von J.A. Hartigan und M. A. Wong). DBSCAN verwendet einen Algorithmus, der darauf basiert das Regionen mit einer hohen Punktdichte durch Regionen mit einer sehr geringen Dichte abgegrenzt sind und dadurch zu erkennen sind. Der SVD Algorithmus (Singular Value Decomposition) wird zuletzt ebenfalls zum Vergleich eingesetzt. Dieser dient allerdings nicht zur Clusterfindung, sondern zur Reduktion der Dimension des Raumes durch eine Linearkombination einzelner Attribute. Es soll in diesem Fall überprüft werden, inwieweit der SVD Algorithmus geeignet ist durch Linearkombinationen Unterräume zu erstellen die Cluster enthalten und somit wieder durch Standardclusteralgorithmen verarbeitet werden können.

Für den Test werden dem Algorithmus jeweils Datenbestände von 100.000 Datensätzen übergeben. Der Gesamtraum hat dabei eine Dimension von 5, 10, 20, 30, 40 oder 50 und

beinhaltet 5 Cluster in 5-dimensionalen Unterräumen. DBSCAN wird aus technischen Gründen auf 10.000 Datensätzen mit einer maximalen Dimension von 10 begrenzt.

BIRCH:

Wie Tabelle 4.1 zeigt findet BIRCH in allen 6 verschiedenen Datenbeständen Cluster. Allerdings kann BIRCH die echten Cluster nur finden, solange die Dimension des Gesamtraumes kleiner gleich 10 ist. Dieses Problem basiert darauf, daß der BIRCH Algorithmus eine Distanzmethode zur Clusterfindung verwendet, welcher bei hochdimensionalen, gleichverteilten Datenbeständen nicht funktioniert.

Dimension der Daten	Dimension der Cluster	Anzahl der Cluster	Gefundene Cluster	Wirklich identifizierte Cluster
5	5	5	5	5
10	5	5	5	5
20	5	5	3,4,5	0
30	5	5	3,4	0
40	5	5	3,4	0
50	5	5	3	0

Tabelle 4.1: Clusterfindung mit BIRCH

DBSCAN:

DBSCAN ist im Gegensatz zu BIRCH auf maximal 10-dimensionale Datenbestände beschränkt. Wie aus Tabelle 4.2 ersichtlich ist identifiziert DBSCAN in Unterräumen des 5- und 7-dimensionalen Gesamtraumes alle 5 Cluster einwandfrei. Probleme treten bei 8- sowie 10-dimensionalen Räumen auf, da hier zwar noch Cluster gefunden werden, diese aber nicht den vorgegebenen Clustern entsprechen. Die Probleme bzw. das Versagen von DBSCAN bei höheren Dimensionen des Gesamtraumes basieren auf der, im Algorithmus verwendeten Methode, der Suche nach Regionen mit hoher Dichte, getrennt durch Regionen mit geringer Dichte. Bei Datenbeständen mit sehr hoher Dimension reduziert sich die Dichte in dem Maße, daß DBSCAN die Regionen mit hoher Dichte nur noch unvollkommen oder gar nicht mehr erkennen kann.

Dimension der Daten	Dimension der Cluster	Anzahl der Cluster	Gefundene Cluster	Wirklich identifizierte Cluster
5	5	5	5	5
7	5	5	5	5
8	5	5	3	1
10	5	5	1	0

Tabelle 4.2: Clusterfindung mit DBSCAN

SVD:

Durch die Verwendung des SVD Algorithmus sollte überprüft werden, ob es möglich ist die Dimension eines hochdimensionalen Raumes durch Linearkombinationen derart zur Verringern, daß ein Standard-Clusteralgorithmus Cluster in dem reduzierten neuen Raum sinnvoll erkennen kann.

Die Tests ergaben, daß sich der neue reduzierte Raum nicht sinnvoll für eine Clustersuche verwenden läßt. Der Grund hierfür läßt sich durch folgende Tatbestände erklären. Basierend darauf, daß beim SVD Algorithmus eine Linearkombination aus Attributen des Originalraumes zur Erstellung eines reduzierten Raumes verwendet wird, verändert sich, wie aus Tabelle 4.3 ersichtlich, die Varianz des jeweiligen reduzierten Raumes. Aus diesem Grund lassen sich keine Unterräume mit Clustern mit Hilfe des SVD Algorithmus erkennen. Zudem sind Linearkombinationen von Attributen nur noch schwer zu interpretieren.

Dimension der Daten	Dimension der Cluster	Anzahl der Cluster	$r_{d/2}$	$r_{(d-5)}$	$r_{(d-1)}$
10	5	5	0,647	0,647	0,937
20	5	5	0,606	0,827	0,969
30	5	5	0,563	0,858	0,972
40	5	5	0,557	0,897	0,981
50	5	5	0,552	0,919	0,984

Tabelle 4.3: Clusterfindung unter zu Hilfenahme des SVD-Algorithmuses

5. Abschließende Analyse von CLIQUE

Zusammenfassend kann man sagen, daß CLIQUE im Vergleich zu den bereits bekannten Algorithmen, wirklich sinnvolle Unterräume mit Clustern erkennen kann. Die Qualität der Ergebnisse ist hierbei unabhängig von der Dimension des Originalraumes und auch der der Unterräume. Bei Datenbeständen mit hoher Dimension neigt CLIQUE geringfügig dazu mehr Cluster zu finden als tatsächlich vorhanden sind. Dieser Tatbestand läßt sich aber auch auf das Problem der optimalen Wahl des Schwellenwertes τ sowie der Intervalleinteilung ξ zurückführen. Werden diese beiden Werte nicht optimal gewählt so kann die Anzahl der Cluster zu hoch oder zu niedrig ausfallen, da Regionen mit hoher Dichte eventuell nicht als solche erkannt werden oder aber wesentlich größer eingeteilt werden. Diesem ganz speziellen Problem wollen sich R. Agrawal, J. Gehrke, D. Gunopulos und P. Raghavan allerdings noch in einer späteren Arbeit widmen.

Ein weiteres Problem welches im Algorithmus von CLIQUE zu finden ist, ist die verbesserte Kandidatenwahl durch den MDL Algorithmus. Dieser kann zum Verwerfen von niedrigdimensionalen Unterräumen führen, die aber eventuell Cluster enthalten. Dem ist allerdings gegenüber zu halten, daß der MDL Algorithmus eine erhebliche Verbesserung der Laufzeit mit sich bringt. Nur aus diesem Grund ist es CLIQUE möglich ein lineares Verhalten bei der Vergrößerung der Datenbestände aufzuweisen.

Außerdem sind CLIQUE noch zwei weitere Vorteile anzurechnen. CLIQUE findet die gewünschten Cluster ohne daß der Benutzer ihm angeben muß wieviele Cluster er im Datenbestand erwartet. Dies ist eine erhebliche Verbesserung gegenüber Algorithmen wie der hierarchischen Clusteranalyse oder z.B. BIRCH, bei denen die erwartete Clusteranzahl vorher angegeben werden muß. Als weiteren Vorteil ergibt sich aus dem Algorithmus, daß CLIQUE keinen Unterschied macht in welcher Anordnung die Eingabedaten vorliegen. Unabhängig von der Reihenfolge der Eingabedaten erzeugt CLIQUE immerdasselbe Endergebnis.

Anhang: Literatur

Agrawal, R. / Gehrke, J. / Gunopulos, D. / Raghavan, P.: Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications (http://www.almaden.ibm.com/cs/people/ragrawal/pubs.html#clustering)