# Unsupervised Learning

**Problemstellung**: Es sind nur Datenobjekte gegeben, aber keine Beispiele die Werte der Zielvariable mit Werten aus der Darstellung im Feature-Raum verknüpfen.

**Aufgaben**:

- Finde Gruppen von ähnlichen Objekten (Clustering)
- Finde Objekte, die ungewöhnlich sind (Outlier Detection)
- Finde Teile von Objektbeschreibungen, die häufig vorkommen (Pattern Mining)

**Vorteile**:

- Ergebnis ist generischer da es weniger Angaben über die Zielvariablen gibt
- Keine manuelle Bewertung von Trainingsbeispielen notwendig (vorher)

**Nachteile**:

- Es ist schwierig die Ergebnisse in Relation zum Vorhersageziel zu setzen.
- höhere Flexibilität hat häufig auch höherer Komplexität zur Folge
- Manuelle Bewertung wird nach dem Data Mining notwendig, um das Ergebnis zu evaluieren.



# Beispielanwendungen

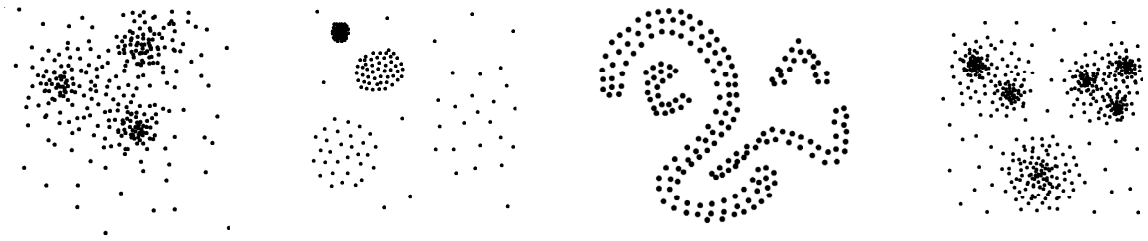
- **Clustering**: Welche Taktiken werden beim Kampf gegen einen Boss-Gegner eingesetzt ?
  (Darstellung der Taktik und Clustering der Ergebnisse)

- **Outlier Detection**: Welche Spieler verhalten sich verdächtig?
  (Beschreibung der Spieler durch Online-Zeiten und Aktivitätsverteilung => feststellen der Abweichung von „normalen Spielern“)

- **Pattern Mining**: Finde Standardrotationen bei einem RPG?
  (Abbilden des Spielerverhaltens durch die Reihenfolge der eingesetzten Fähigkeiten => häufige Muster entsprechen einem typischen Verhalten)

# Clustering Verfahren

- Identifikation einer endlichen Menge von Kategorien, Klassen oder Gruppen (*Cluster*) in den Daten.
- *Ähnliche* Objekte sollen im *gleichen* Cluster sein, *unähnliche* Objekte sollen in *unterschiedlichen* Clustern sein.
- Clustering-Modelle sollen sowohl Klassen(Cluster) finden, als auch Methoden liefern Objekte den Klassen (Clustern) zu-zuweisen.





# Clustering

**Gegeben**:

- Eine Objektmenge: $DB \subseteq F$
- Eine diskrete Zielvariable $C \subseteq IN_0$ (Menge der Clusterindizes)
- ggf. ist $|C|$ bekannt (Anzahl der Cluster)

**Gesucht**: Funktion $f: F \rightarrow C$, die Objektdarstellungen auf Werte der Zielvariable abbilden und dabei möglichst „*gute Cluster*“ findet.

**Güte eines Clusters:**

- Abhängig vom Clustermodell:
  - Wie wird festgestellt das Objek o zu Cluster c gehört?
  - Wie kann man entscheiden ob 2 Objekte im gleichen Cluster liegen?
- Optimiert werden sollen:
  - Kompaktheit des Clusters
  - Trennung der Cluster

# Partitionierendes Clustering (1)

**Idee**:

- Es gibt k cluster und jeder Cluster $c$ hat ein repräsentatives Objekt $o_c$
- Zuordnung von Objekten $o$ zu $c$ über den Abstand $dist(o_c,o)$:
  $cluster(o) = \arg\min_{c \in C}(dist(o_c, o))$
- Kompaktheit eines Cluster über:
  - durchschnittlichen Abstand der zugeordneten Elemente:
    $$compact(c) = \sum_{o \in \{o \in DB | cluster(o)=c\}} dist(o_c, o)$$
  - durchschnittlichen quadratischen Abstand der zugeordneten Elemente:
    $$sqrComp(c) = \sum_{o \in \{o \in DB | cluster(o)=c\}} dist(o_c, o)^2$$
- Güte des Clusterings: $TD(C) = \sum_{c \in C} compact(c)$

  oder $TD^2(C) = \sum_{c \in C} sqrComp(c)$



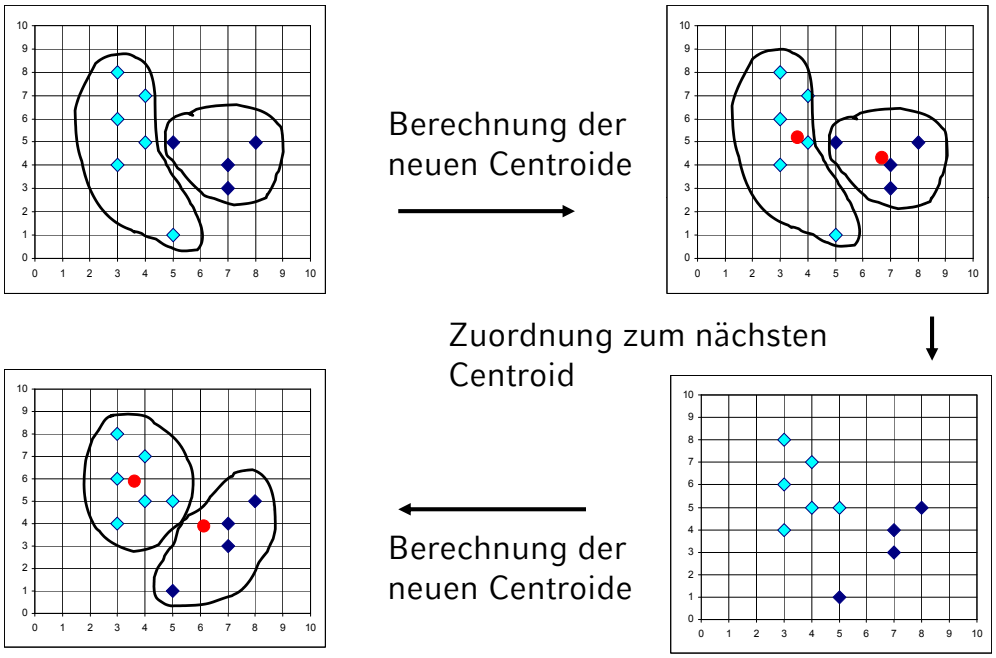
# Partitionierendes Clustering (2)

- typische Repräsentanten:
  - Centroid: $centroid(c) = \frac{1}{|\{o \in DB \mid cluster(o) = c\}|} \sum_{o \in \{o \in DB | cluster(o)=c\}} o$
  - Medoid: $medoid(c) = \arg\min_{o \in \{o \in DB | cluster(o)=c\}} \left( \sum_{p \in \{p \in DB | cluster(o)=c\}} dist(o, p) \right)$

***Minimieren von TD oder $TD^2$:***

- *TD* und $TD^2$ sind nicht konvex und haben lokale Minima
- *TD* und $TD^2$ sind nicht stetig (Sprungstellen bei Clusterwechsel möglich)
- Verfahren mit Greedy-Suche die TD/ $TD^2$ lokal minimieren
  1. Schritt: Gegeben ist cluster(o) für alle o∈ DB
     => bilde die Menge der Clusterrepräsentaten $\{o_{c1},.., o_{cn}\}$
  2. Schritt: Gegeben ist $Centroids=\{o_{c1},.., o_{cn}\}$
     =>ordne alle Objekt *o* dem nächsten Element von *Centroids* zu
  3. Abbruch falls sich TD/ $TD^2$ nicht mehr ändert

# Beispiel für partitionierendes Clustering






# Algorithmus

```
ClusteringDurchVarianzMinimierung(Objektmenge DB, Integer k)
    Erzeuge eine „initiale" Zerlegung
      der Objektmenge DB in k Cluster;
    Berechne die Menge C'={C_1, ..., C_k} der
      Centroide für die k Cluster;
    C = {};
    TD2 = sqrTD(C',DB);
    repeat
        TD2old = TD2;
        C = C';
        Bilde k Cluster durch Zuordnung jedes Punktes zum
          nächstliegenden Centroid aus C;
        Berechne die Menge C'={C'_1, ..., C'_k} der Centroide für
          die neu bestimmten Cluster;
        TD2 = sqrTD(C',DB);
    until TD2 == TD2old;
    return C;
```

# partitionierendes Clustering

**Varianten**:

- k-Means: in jedem Schritt wird nur ein Objekt ggf. neu zugewiesen.
  Der alte und der neue Cluster-Centroid wird danach gleich geändert.
- Expectation Maximation Clustering (EM)
  Statitische Variante die Cluster durch Verteilungfunktionen beschreibt.
- k-Medoid Clusterings:
  - Cluster-Repräsentanten sind Medoide
  - Anpassung der Cluster durch Vertauschen von Medoid und Nicht-Medoid Objekten.

**Eigenschaften**:

- Algorithmus ist stark von der initialen Partitionierung abhängig.
- Centroid-basierte Verfahren sind sehr schnell O(i*n*k). (#Iterationen i)
- Medoid Verfahren sind relativ langsam $O(i*n^2*k)$ (#Iterationen i)

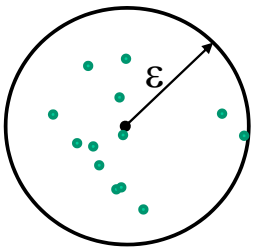


# Dichtebasiertes Clustering

**Idee**: Cluster sind dichte Regionen im Featureraum F.

**Dichte**:

$$\frac{\text{Anzahl Objekte}}{\text{Volumen}}$$




**Hier**:

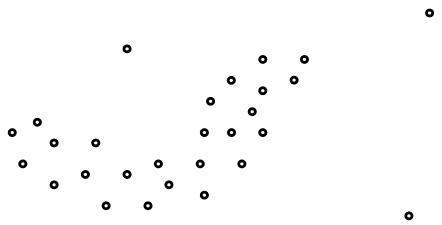
- ***Volumen***: ε-Umgebung bzgl. Distanzmaß *dist(x,y)* um Objekt o
- ***dichte Region:*** es liegen MinPts Objekte in der ε-Umgebung von o
  => o wird Kernpunkt genannt
- „zuhängende Kernpunkte“ bilden ***Cluster***
- Punkte, die in keinem Cluster liegen nennt man Rauschen (***Noise***)

# Dichtebasiertes Clustering

## Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren Dichtegrenzwert

$\varepsilon$ $MinPts = 4$





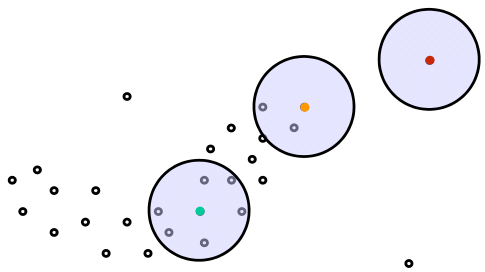
# Dichtebasiertes Clustering

## Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren Dichtegrenzwert

$\varepsilon$ $MinPts = 4$

- *Kernpunkt*



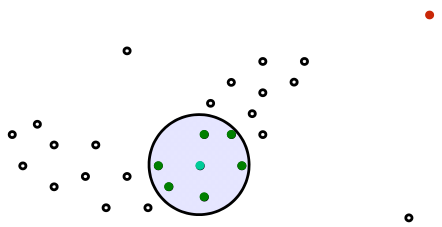
# Dichtebasiertes Clustering

## Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren Dichtegrenzwert

$\varepsilon$ $MinPts = 4$

- *Kernpunkt*
- *Direkte Dichte-Erreichbarkeit*





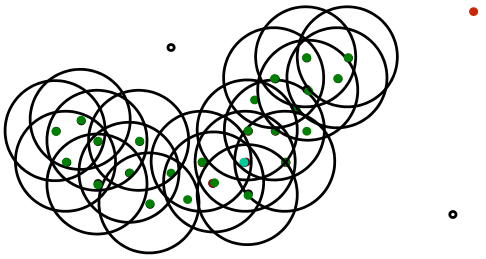
# Dichtebasiertes Clustering

## Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren Dichtegrenzwert

$\varepsilon$ $MinPts = 4$

- *Kernpunkt*
- *Direkte Dichte-Erreichbarkeit*
- *Dichte-Erreichbarkeit*



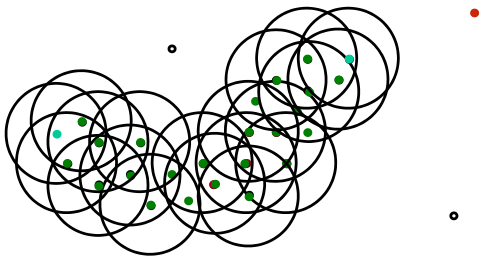
# Dichtebasiertes Clustering

## Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren Dichtegrenzwert

$\varepsilon$ $MinPts = 4$

- *Kernpunkt*
- *Direkte Dichte-Erreichbarkeit*
- *Dichte-Erreichbarkeit*
- *Dichte-Verbundenheit*

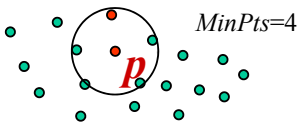




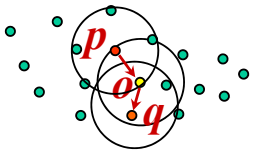
# Dichtebasiertes Clustering

## Formalisierung [Ester, Kriegel, Sander & Xu 1996]

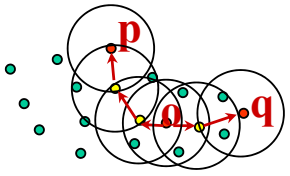
- Ein Objekt $p \in DB$ heißt *Kernobjekt*, wenn gilt:
  $|RQ(p,\varepsilon)| \geq MinPts$
  $RQ(p,\varepsilon) = \{o \in DB \mid dist(p,o) <= \varepsilon\}$




- Ein Objekt $p \in DB$ ist *direkt dichte-erreichbar* von $q \in DB$ bzgl. $\varepsilon$ und *MinPts,* wenn gilt:
  $p \in RQ(q,\varepsilon)$ und $q$ ist ein Kernobjekt in *DB*.




- Ein Objekt $p$ ist *dichte-erreichbar* von $q$,
  wenn es eine Kette von direkt erreichbaren Objekten von $q$ nach $p$ gibt.
- Zwei Objekte $p$ und $q$ sind *dichte-verbunden*,
  wenn sie beide von einem dritten Objekt $o$ aus dichte-erreichbar sind.



## Formalisierung

Ein (*dichtebasierter*) *Cluster* $C$ bzgl. ε und *MinPts* ist eine nicht-leere Teilmenge von *DB* für die die folgenden Bedingungen erfüllt sind:

*Maximalität*: $\forall p,q \in DB$: wenn $p \in C$ und $q$ dichte-erreichbar von $p$ ist, dann ist auch $q \in C$.

*Verbundenheit*: $\forall p,q \in C$: $p$ ist dichte-verbunden mit $q$.





## Formalisierung

- Definition Clustering
  Ein *dichtebasiertes Clustering* *CL* der Menge *DB* bzgl. ε und *MinPts* ist eine „vollständige“ Menge von dichtebasierten Clustern bzgl. ε und *MinPts* in *DB*.

- Definition Noise
  Die Menge $Noise_{CL}$ („*Rauschen*“) ist definiert als die Menge aller Objekte aus *DB*, die nicht zu einem der dichtebasierten Cluster $C \in CL$ gehören.

- Grundlegende Eigenschaft
  Sei $C$ ein dichtebasierter Cluster und sei $p \in C$ ein Kernobjekt.
  Dann gilt:
  $C = \{o \in DB \mid o$ dichte-erreichbar von $p$ bzgl. ε und *MinPts*$\}$.

  => ermöglicht effiziente Suche (WARUM?)

# Dichtebasiertes Clustering

## Algorithmus DBSCAN

```
DBSCAN(Punktmenge DB, Real ε, Integer MinPts)
  // Zu Beginn sind alle Objekte unklassifiziert,
  // o.ClId = UNKLASSIFIZIERT für alle o ∈ DB

   ClusterId := nextId(NOISE);
   for i from 1 to |DB| do
      Objekt := DB.get(i);
      if Objekt.ClId = UNKLASSIFIZIERT then
         if ExpandiereCluster(DB, Objekt, ClusterId, ε, MinPts)
         then ClusterId:=nextId(ClusterId);
```



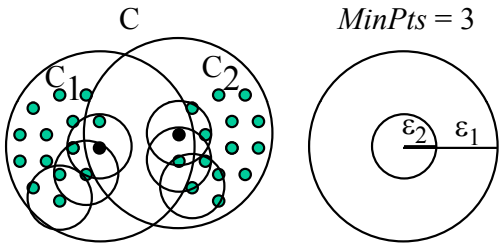
# Dichtebasiertes Clustering

```
ExpandiereCluster(DB, StartObjekt, ClusterId, ε, MinPts): Boolean
seeds:= RQ(StartObjekt, ε);
if |seeds| < MinPts then // StartObjekt ist kein Kernobjekt
  StartObjekt.ClId := NOISE;
  return false;
// sonst: StartObjekt ist ein Kernobjekt
forall o ∈ seeds do o.ClId := ClusterId;
  entferne StartObjekt aus seeds;
  while seeds ≠ Empty do
    wähle ein Objekt o aus der Menge seeds;
    Nachbarschaft := RQ(o, ε);
    if |Nachbarschaft| ≥ MinPts then // o ist ein Kernobjekt
        for i from 1 to |Nachbarschaft| do
              p := Nachbarschaft.get(i);
              if p.ClId in {UNCLASSIFIED, NOISE} then
                    if p.ClId = UNCLASSIFIED then
                          füge p zur Menge seeds hinzu;
                    p.ClId := ClusterId;
    entferne o aus der Menge seeds;
return true;
```

# Dichtebasiertes hierarchisches Clustering

[Ankerst, Breunig, Kriegel & Sander 1999]

- für einen konstanten *MinPts*-Wert sind dichte-basierte Cluster bzgl. eines kleineren ε vollständig in Clustern bzgl. eines größeren ε enthalten




- in einem DBSCAN-ähnlichen Durchlauf gleichzeitig das Clustering für verschiedene Dichte-Parameter bestimmen
- zuerst die dichteren Teil-Cluster, dann den dünneren Rest-Cluster



# Dichtebasiertes hierarchisches Clustering

## Grundbegriffe

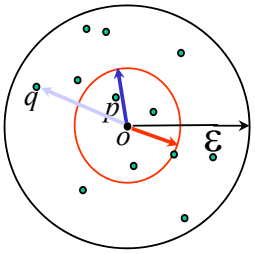
- *Kerndistanz* eines Objekts *o* bzgl. ε und *MinPts*

$$Kerndistanz_{\varepsilon,MinPts}(o) = \begin{cases} UNDEFINIERT, & wenn\ |RQ(o,\varepsilon)| < MinPts \\ MinPtsDistanz(o), & sonst \end{cases}$$

- *Erreichbarkeitsdistanz* eines Objekts *p* relativ zu einem Objekt *o*

$$Erreichbarkeitsdistanz_{\varepsilon,MinPts}(p,o) = \begin{cases} UNDEFINIERT & wenn\ |RQ(o,\varepsilon)| < MinPts \\ \max\{Kerndistanz(o), dist(o,p)\}, & sonst \end{cases}$$

*MinPts = 5*




*Kerndistanz*(*o*)

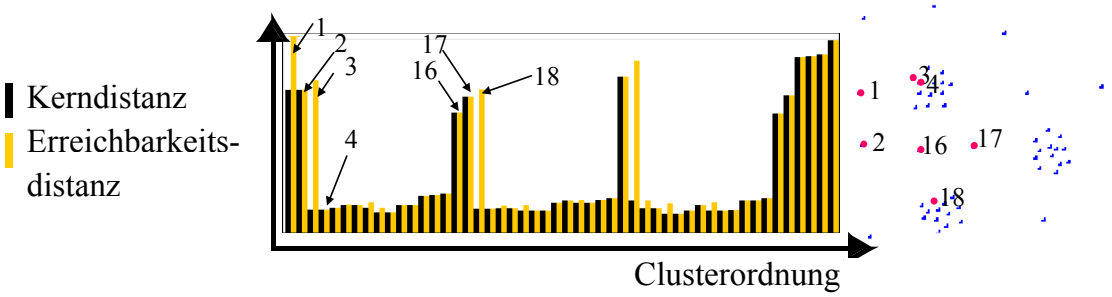
*Erreichbarkeitsdistanz*(*p*,*o*)

*Erreichbarkeitsdistanz*(*q*,*o*)

# Algorithmus OPTICS

## Clusterordnung

- OPTICS liefert nicht direkt ein (hierarchisches) Clustering, sondern eine „Clusterordnung“ bzgl. ε und *MinPts*
- *Clusterordnung* bzgl. ε und *MinPts*
  - beginnt mit einem beliebigen Objekt
  - als nächstes wird das Objekt besucht, das zur Menge der bisher besuchten Objekte die minimale Erreichbarkeitsdistanz besitzt






# Algorithmus OPTICS

## Datenstrukturen

- SeedList : speichert Punkte mit „aktueller“ Erreichbarkeitsdistanz in aufsteigender Reihenfolge
- ClusterOrder: resultierende Clusterordnung wird schrittweise aufgebaut

## Hauptschleife

```
SeedList = ∅;
WHILE es gibt noch unmarkierte Objekte in DB DO
   IF SeedList = ∅
   THEN
     füge beliebiges noch unmarkiertes Objekt in ClusterOrder ein mit
     Erreichbarkeitsdistanz ∞;
   ELSE
     füge erstes Objekt aus der SeedList mit aktueller
   Erreichbarkeitsdistanz in  ClusterOrder ein;
   // sei obj das zuletzt in ClusterOrder eingefügte Objekt
   markiere obj als bearbeitet;
   FOR ALL neighbor ∈ RQ(obj, ε) DO
     // insert/update neighbor in SeedList mit referenzobjekt obj
     SeedList.update(neighbor, obj);
```

# Algorithmus OPTICS

Einfügen/Updaten eines Objekts *o* in SeedList:

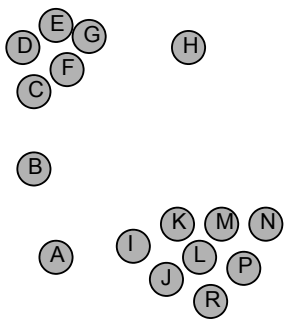
- Beachte: Für alle Objekte *p* in SeedList ist die „aktuelle“ Erreichbarkeitsdistanz *p*.rdist gespeichert.
- SeedList ist nach *p*.rdist aufsteigend sortiert (als Heap organisiert)
- Referenzobjekt: *obj*

```
SeedList :: update(o, obj)
Berechne Erreichbarkeitsdistanz_{ε,MinPts}(o, obj) =: rdistneu_o;
IF o ist bereits in SeedList THEN
   IF rdistneu_o ≤ o.rdist THEN
     o.rdist := rdistneu_o;
     verschiebe o in SeedList (nach vorne); // aufsteigen im
   Heap
ELSE
   // o ist noch nicht in SeedList => normales Einfügen in Heap
   füge o mit o.rdist := rdistneu_o in SeedList ein;
```



# Beispiellauf mit OPTICS

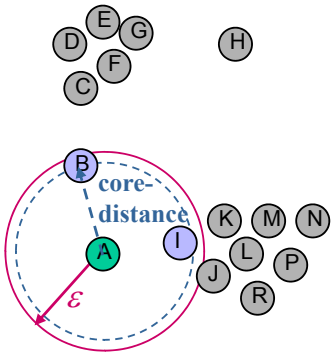
- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



seed list:

## Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

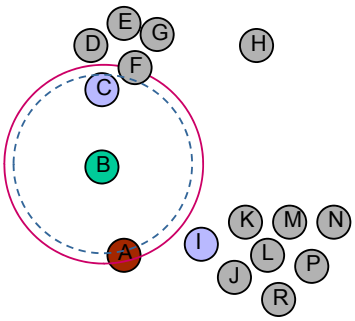


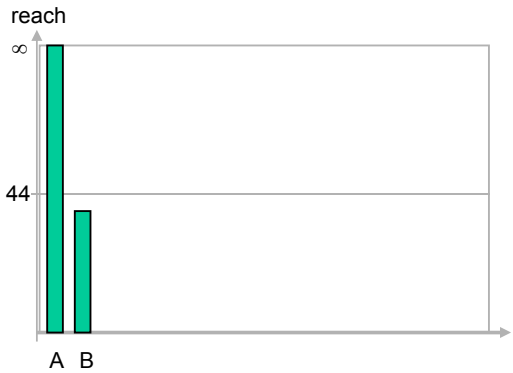
seed list: (B,40) (I, 40)



## Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

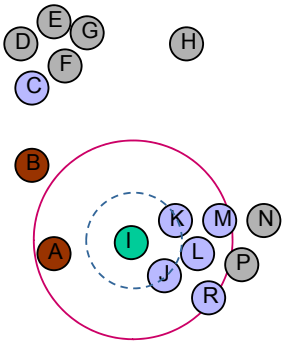



seed list: (I, 40) (C, 40)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

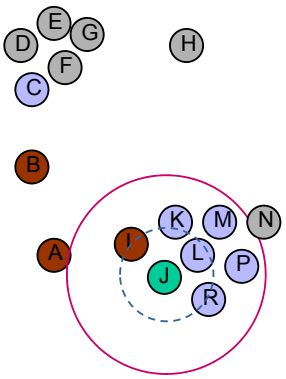


seed list: (J, 20) (K, 20) (L, 31) (C, 40) (M, 40) (R, 43)



# Beispiellauf mit OPTICS

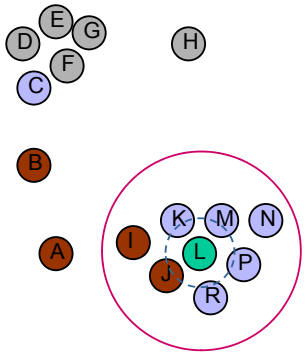
- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



seed list: (L, 19) (K, 20) (R, 21) (M, 30) (P, 31) (C, 40)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

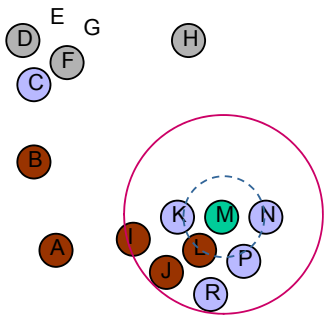


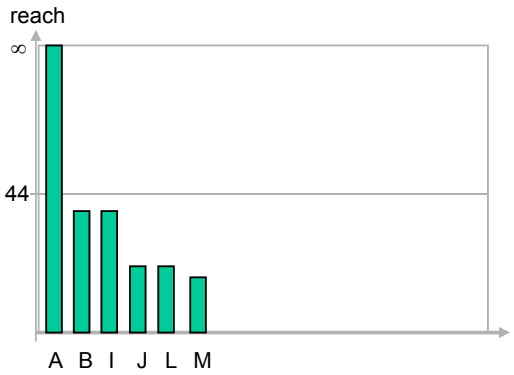
seed list: (M, 18) (K, 18) (R, 20) (P, 21) (N, 35) (C, 40)



# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

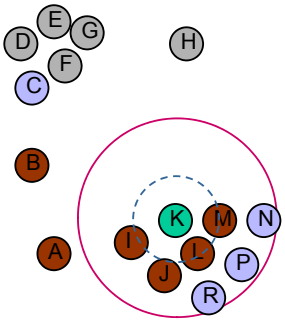


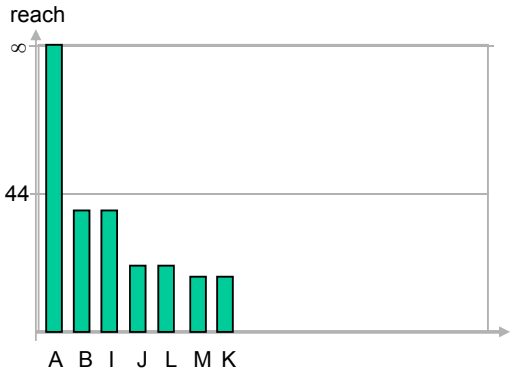




seed list: (K, 18) (N, 19) (R, 20) (P, 21) (C, 40)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



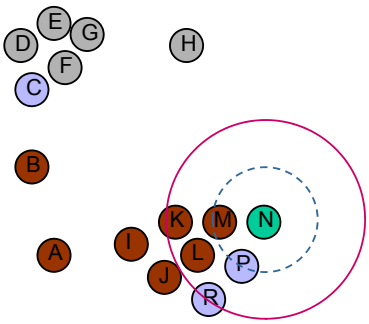



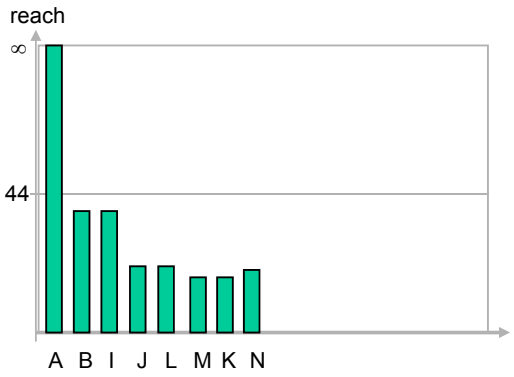

seed list: (N, 19) (R, 20) (P, 21) (C, 40)



# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

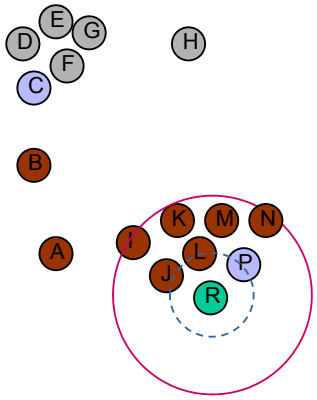


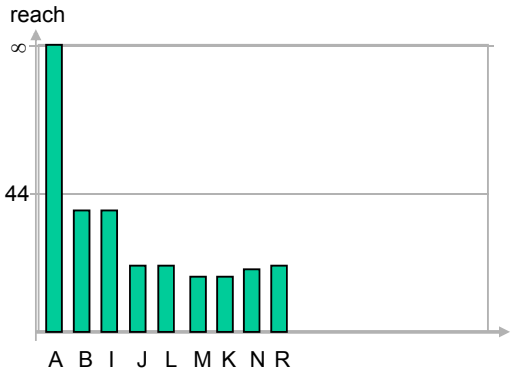




seed list: (R, 20) (P, 21) (C, 40)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



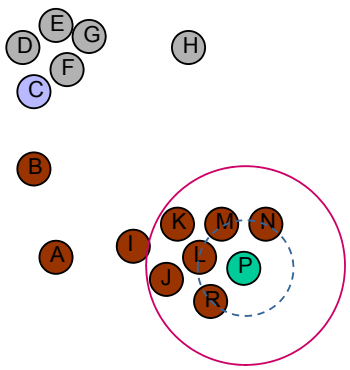




seed list: (P, 21) (C, 40)



# Beispiellauf mit OPTICS

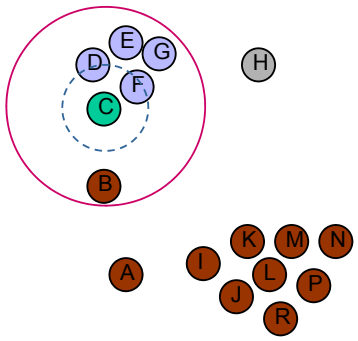
- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

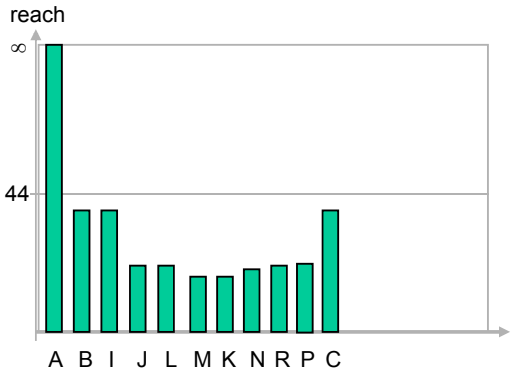


seed list: (C, 40)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



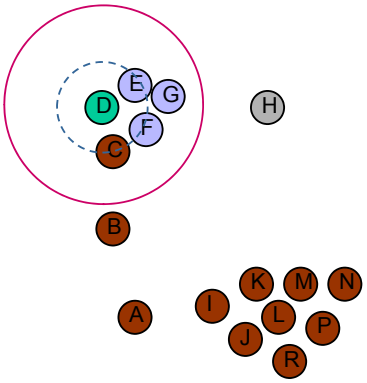




seed list: (D, 22) (F, 22) (E, 30) (G, 35)



# Beispiellauf mit OPTICS

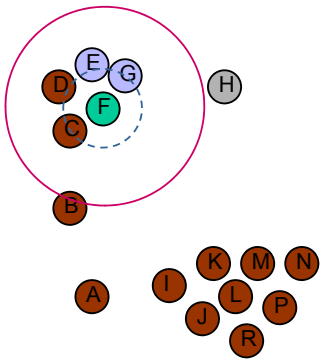
- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

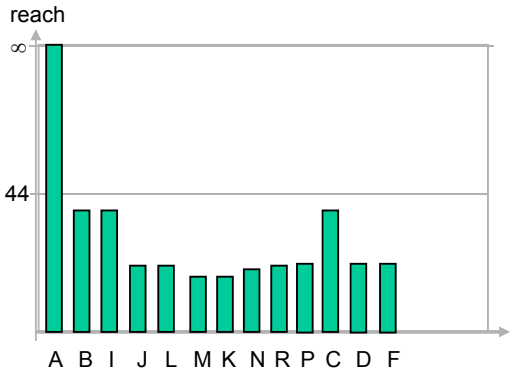


seed list: (F, 22) (E, 22) (G, 32)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



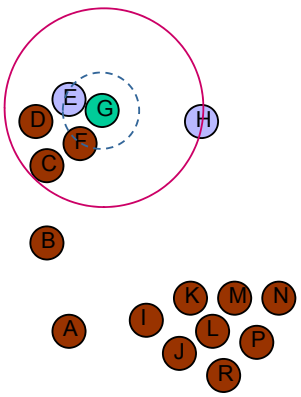



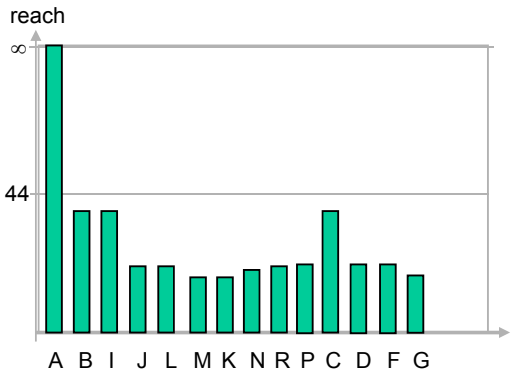

seed list: (G, 17) (E, 22)



# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

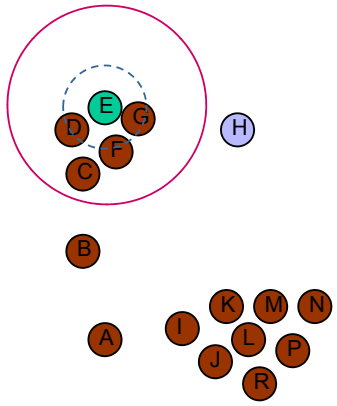


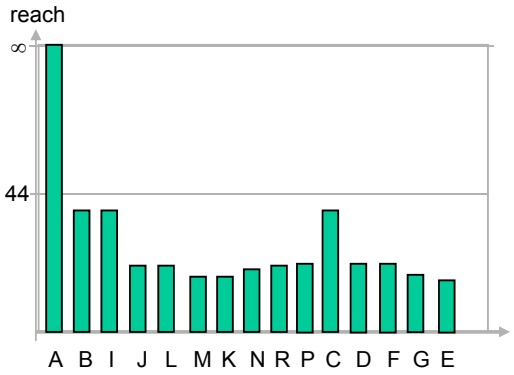




seed list: (E, 15) (H, 43)

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3



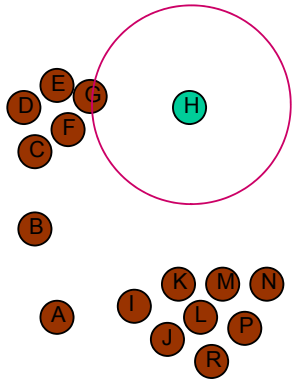




seed list: (H, 43)



# Beispiellauf mit OPTICS

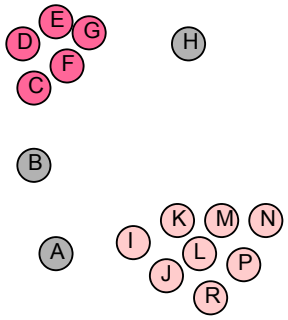
- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

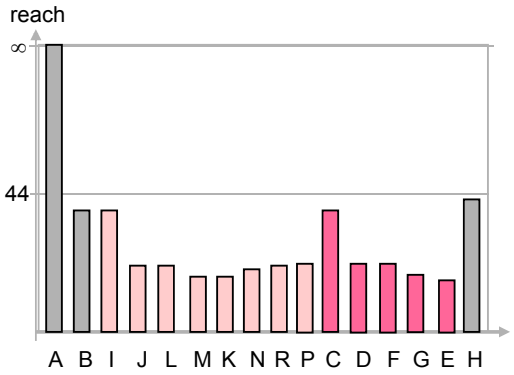


seed list: -

# Beispiellauf mit OPTICS

- Example Database (2-dimensional, 16 points)
- ε= 44, *MinPts* = 3

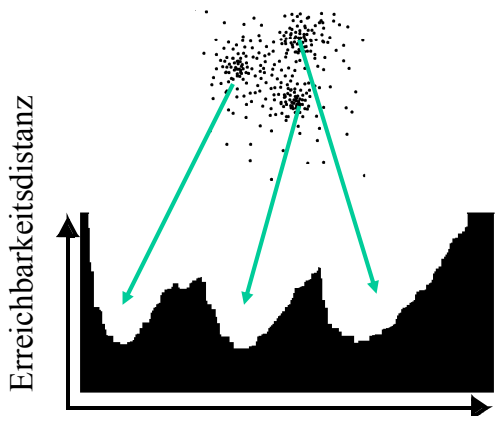


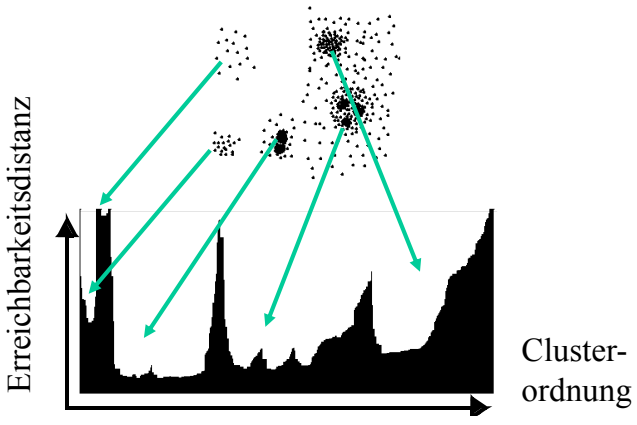





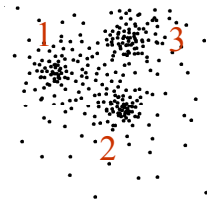

# Erreichbarkeits-Diagramm

- Zeigt die Erreichbarkeitsdistanzen (bzgl. ε und *MinPts*) der Objekte
- als senkrechte, nebeneinanderliegende Balken
- in der durch die Clusterordnung der Objekte gegebenen Reihenfolge



# Parameter-Sensitivität



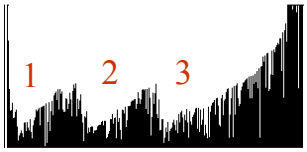

*MinPts* = 10, ε = 10

optimale Parameter

*MinPts* = 10, ε = 5

kleineres ε

*MinPts* = 2, ε = 10




kleineres *MinPts*

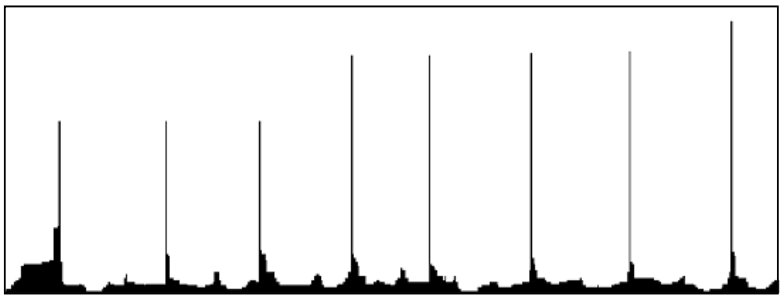
Clusterordnung ist robust gegenüber den Parameterwerten
gute Resultate wenn Parameterwerte „groß genug“



# Manuelle Analyse der Cluster

## Mit Erreichbarkeits-Diagramm

- gibt es Cluster?
- wieviele Cluster?
- sind die Cluster hierarchisch geschachtelt?
- wie groß sind die Cluster?



Erreichbarkeits-Diagramm

# Diskussion dichtebasiertes Clustering

- Anzahl der Cluster wird automatisch bestimmt
- Parameter ε und MinPts sind relativ unkritisch (zumindest bei OPTICS)
- Komplexität ist $O(n^2)$ im allgemeinen Fall
- dichtebasierte Verfahren benötigen nur ein Distanzmaß zur Anwendung
- Sowohl DBSCAN als auch OPTICS erlauben Noise-Objekte, die zu keiner Gruppe gehören
- DBSCAN und OPTICS sind reihenfolgeabhängig (Randpunkte)
- OPTICS ist zur visuellen Analyse geeignet
- Keine Zielfunktion wird optimiert
- kein  kompaktes Clustermodell
- keine Zuordnungsanweisung für neue Objekte (unsupervised classification)



# Outlier Detection

**Definition nach Hawkins [Hawkins 1980]:**

"Ein Outlier ist eine *Beobachtung*, die sich von den anderen *Beobachtungen* so deutlich unterscheidet, dass man denken könnte, sie sei von einem anderen Mechanismus generiert worden."

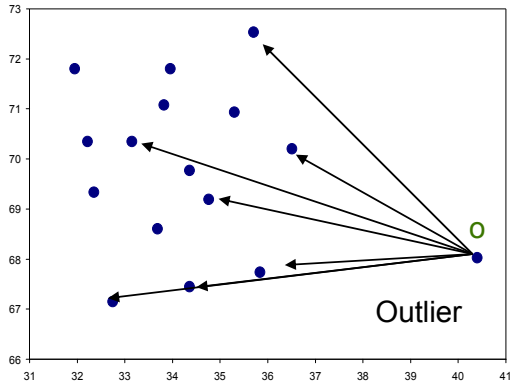
**Was meint "Mechanismus"?**

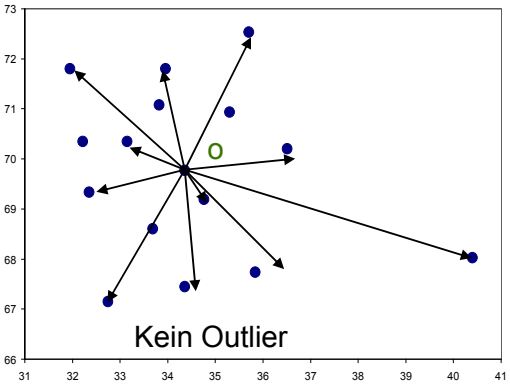
- Intuition aus der Statistik: "erzeugender Mechanismus" ist ein (statistischer) Prozess.
- Abnormale Daten (Outlier) zeigen eine verdächtig geringe Wahrscheinlichkeit, aus diesem Prozess zu stammen.
- Zusammenhang mit dem Clustering: Wenn das Clustering die Verteilung der Daten im Raum beschreibt, dann sind Outlier Objekte die nicht oder nur sehr schlecht Clustern zugeordnet werden können:
  - Maximaler Abstand zu allen Cluster Zentren (Part. Clustering)
  - Noise Punkte im Dichtebasiertem Clustering

# ABOD – angle-based outlier degree

**Idee**:

- Winkel sind in hochdimensionalen Daten stabiler als Distanzen
- Ein Objekt ist ein Outlier, wenn andere Objekte alle in derselben Richtung liegen.
- Ein Objekt ist kein Outlier, wenn es in der Mitte aller Objekte liegt






# ABOD

## Grundannahme

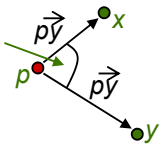
- Outlier sind am Rand der Datenverteilung
- Normale Punkte liegen in der Mitte der Verteilung

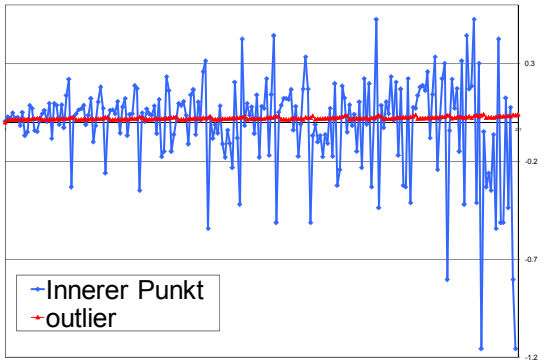
## Modell

Für ein Objekt p und jeweils zwei ander Objekt aus DB *x* und *y*, wird der Winkel zwischen $\overrightarrow{px}$ und $\overrightarrow{py}$ bestimmt.

=> Summe über alle Winkel zu p heiß Winkelspektrum

Winkel zwischen $\overrightarrow{px}$ und $\overrightarrow{py}$



# ABOD

- Das Winkelspektrum von $p$ gibt ab wie diverse die Winkel bzgl. $p$ sind.
- Bei Outliern kann man überwiegend spitze Winkel beobachten.
- Die Varianz des Winkelspektrum fasst das Spektrum auf einen Wert zusammen

$$ABOD(p) = \underset{x,y \in DB}{VAR} \left( \frac{\left\langle \vec{xp}, \vec{yp} \right\rangle}{\left\| \vec{xp} \right\|^2 \cdot \left\| \vec{yp} \right\|^2} \right)$$

**Interpretation**

- kleiner ABOD => Outlier
- großer ABOD => No Outlier



# ABOD

## Algorithmus

- Naïve Algorithmus hat eine Zeitkomplexität in $O(n^3)$
- Es gibt einen effizienteren Allgorithmus zur Bestimmung der die top-$n$ Outlier berechnet $O(n^2)$
- Das Winkelspektrum wird dabei durch eine Stichprobe approximiert und auf deren Grundlage eine konservative Approximation berechnet wird.
- Auf Basis der konservativen Approximation müssen dann nur noch wenige Spektren exakt bestimmt werden.

# Frequent Pattern Mining

**Bisher**: Muster beschreiben Menge aus vollständigen Objektbeschreibungen.

**Idee**: Muster können auch nur aus Teilen der gesamten Objektbeschreibung bestehen.

**Hier**: Muster ist ein bestimmter Teil der Objektbeschreibung, der in vielen Objekten der Datenmenge vorkommt.

**Beispiel**: Gruppenzusammensetzungen in MMORPG:

| | |
|---|---|
| Grp1 | Priester, Magier, Druide, Schurke, Krieger |
| Grp3 | Magier, Magier, Jäger , Priester, Krieger |
| Grp3 | Priester, Priester, Paladin, Druide, Krieger |
| Grp4 | Hexer, Paladin, Schamane, Krieger, Priester |
| Grp5 | Priester, Krieger, Jäger, Schurke, Krieger |

Krieger, Priester,?,?,?



# Varianten des Frequent Pattern Mining

- **Frequent Itemset Mining** (=> KDD1 und auf den nächsten Folien)
  Datenobjekte sind Teilmenge einer Menge von Items
  => Bestimme Teilmengen, die in vielen Objekten vorkommen

- **Frequent Substring Mining** (=> nächstes Kapitel)
  Datenobjekte sind Strings über einem endlichen Alphabet
  => Bestimme häufige Substrings

- **Frequent Subgraph Mining** (=> KDD2)
  Datenobjekte sind Graphen mit kategorischen Labeln/Knotentypen
  => Bestimme häufige isomorphe Teilgraphen

- **Regel Mining**: Auf Mustern lassen sich Regeln ableiten:
  *Wenn Teilmuster A vorhanden ist,*
  *dann ist auch die Erweiterung A+B vorhanden*

# Grundlagen Frequent Itemsets (1)

- *Items* $I = \{i_1, ..., i_m\}$ eine Menge von Literalen
  z.B. Waren/Artikel bei einem Einkauf
- *Itemset* $X$: Menge von Items $X \subseteq I$
  z.B. ein kompletter Einkauf
- *Datenbank DB*: Menge von *Transaktionen* $T$ mit $T = (tid, X_T)$
  z.B. Menge aller Einkäufe (=Transaktionen) in einem bestimmten Zeitraum
- Transaktion $T$ enthält Itemset $X$: $X \subseteq T$
- Items in Transaktionen oder Itemsets sind lexikographisch sortiert:
  Itemset $X = (x_1, x_2, ..., x_k)$, wobei $x_1 \leq x_2 \leq ... \leq x_k$
- *Länge des Itemsets*: Anzahl der Elemente in einem Itemset
- $k$-Itemset: ein Itemset der Länge $k$
  {Butter, Brot, Milch, Zucker} ist ein 4-Itemset
  {Mehl, Wurst} ist ein 2-Itemset



# Grundlagen Frequent Itemsets

- *Cover* eines Itemset $X$: Menge der Transaktionen $T$, die $X$ enthalten:
  $cover(X) = \{tid \mid (tid, X_T) \in DB, X \subseteq X_T\}$
- *Support des Itemset X in DB*: Anteil der Transaktionen in $DB$, die $X$ enthalten:
  $support(X) = |cover(X)|$
  Bemerkung: $support(\emptyset) = |DB|$
- *Häufigkeit eines Itemsets X in DB*:
  Wahrscheinlichkeit, daß $X$ in einer Transaktion $T \in DB$ auftritt:
  $frequency(X) = P(X) = support(X) / |DB|$
- *Häufig auftretendes (frequent) Itemset X in DB*:
  $support(X) \geq s \qquad (0 \leq s \leq |DB|)$
  $s$ ist ein absoluter support-Grenzwert
- Alternativ: $frequency(X) \geq s_{rel}$ wobei $s = \lceil s_{rel} \cdot |DB| \rceil$

# Problemstellung: Frequent Itemset Mining

**Gegeben**:

- eine Menge von Items $I$
- eine Transaktionsdatenbank $DB$ über $I$
- Ein absoluter support-Grenzwert $s$

**Aufgabe**: Finde alle frequent Itemsets in $DB$, d.h. $\{X \subseteq I \mid support(X) \geq s\}$

| TransaktionsID | Items |
|---|---|
| 2000 | A,B,C |
| 1000 | A,C |
| 4000 | A,D |
| 5000 | B,E,F |

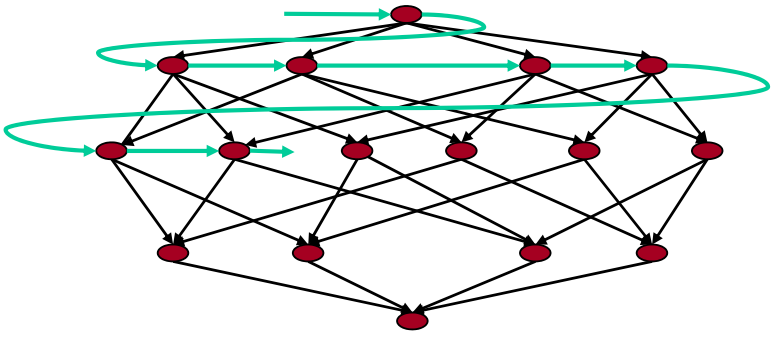
Support der 1-Itemsets:
(A): 75%, (B), (C): 50%, (D), (E), (F): 25%,
Support der 2-Itemsets:
(A, C): 50%,
(A, B), (A, D), (B, C), (B, E), (B, F), (E, F): 25%



# Itemset Mining

Apriori Algorithmus [Agrawal & Srikant 1994]

- zuerst die einelementigen Frequent Itemsets bestimmen, dann die zweielementigen und so weiter (Breitensuche)



- Finden von $k$+1-elementigen Frequent Itemsets:
- nur solche $k$+1-elementigen Itemsets betrachten, für die alle $k$-elementigen Teilmengen häufig auftreten
- Bestimmung des Supports durch Zählen auf der Datenbank (ein Scan)

# Apriori Algorithmus

$C_k$: die zu zählenden Kandidaten-Itemsets der Länge $k$

$L_k$: Menge aller häufig vorkommenden Itemsets der Länge $k$

```
Apriori(I, DB, minsup)
L1 := {frequent 1-Itemsets aus I};
k := 2;
while Lk-1 ≠ ∅ do
    Ck := AprioriKandidatenGenerierung(Lk-1);
    for each Transaktion T ∈ DB do
        CT := Subset(Ck, T);// alle Kandidaten aus Ck, die
                            // der Transaktion T enthalten sind;
        for each Kandidat c ∈ CT do c.count++;
    Lk := {c ∈ Ck | c.count ≥ minsup};
    k++;
return ∪k Lk;
```



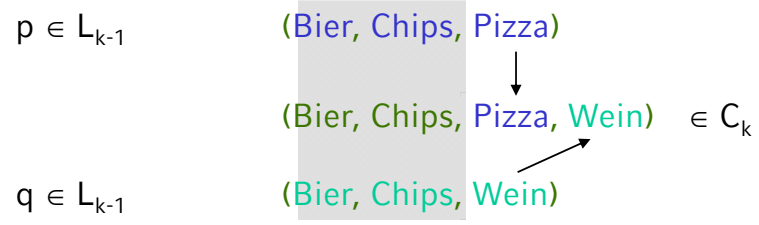
# Kandidatengenerierung im Apriori Algorithmus (1)

Anforderungen an Kandidaten-Itemsets $C_k$

- Obermenge von $L_k$
- wesentlich kleiner als die Menge aller k-elementigen Teilmengen von I

Schritt 1: Join

- k-1-elementige Frequent Itemsets p und q
- p und q werden miteinander verbunden, wenn sie in den ersten k–2 Items übereinstimmen



# Kandidatengenerierung im Apriori Algorithmus (2)

Schritt 2: Pruning

entferne alle Kandidaten-k-Itemsets, die eine k–1-elementige Teilmenge enthalten, die nicht zu $L_{k-1}$ gehört

Beispiel:

$L_3$ = {(1 2 3), (1 2 4), (1 3 4), (1 3 5), (2 3 4)}

nach dem Join-Schritt: Kandidaten = {(1 2 3 4), (1 3 4 5)}

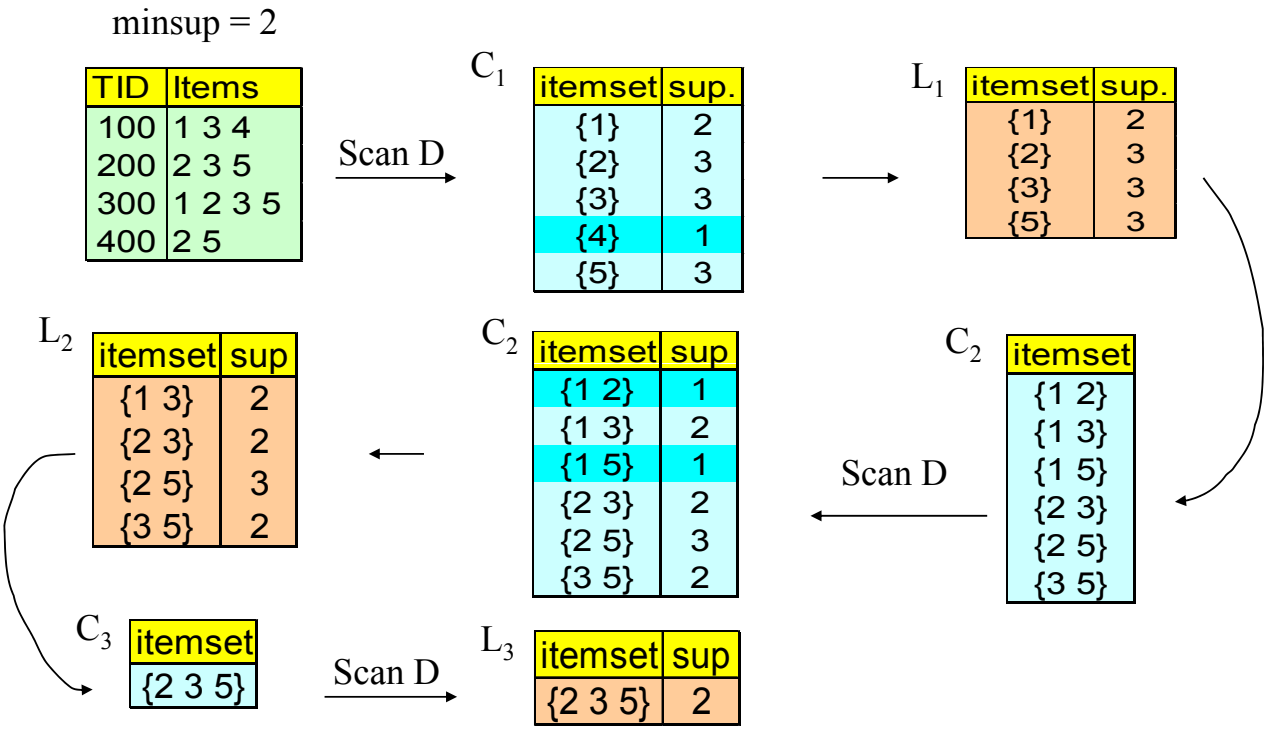
im Pruning-Schritt:

lösche (1 3 4 5)

⇨ $C_4$ = {(1 2 3 4)}



# Beispiel für den Apriori Algorithmus



# Frequent Itemset Mining

**Eigenschaften**:

- Benötigt für alle Itemsets der Länge $k$ einen Datenbank-Scan

  $\Rightarrow$ $O(l \cdot |D|)$

- Menge der generierten Kandidaten, die nicht frequent sind entspricht dem negativen Rand

  $\Rightarrow O\left(\binom{m}{\lfloor m/2 \rfloor}\right)$ (Sperners Theorem)

- Wenn nicht alle Kandidaten Itemsets in den Hauptspeicher passen, werden Kandidaten blockweise auf min. Support überprüft



# Lernziele

- Was ist Data Mining und KDD?
- Welche Schritte hat der KDD Prozeß?
- Was ist Supervised Learning?
  - Linear Regression
  - kNN Klassifikation
  - Bayes Learner
  - Bewertung von Supervised Verfahren
- Was ist Unsupervised Learning
  - Clustering (partitionierende Verfahren, DBSCAN, OPTICS)
  - Outlier Detection (ABOD)
  - Frequent Pattern Mining (Frequent Itemset Mining mi Apriori-Alg.)

# Literatur


- Skripte zu KDD I: http://www.dbs.informatik.uni-muenchen.de/cms/Knowledge_Discovery_in_Databases_I_(KDD_I)
- Ester M., Sander J.:
  **Knowledge Discovery in Databases: Techniken und Anwendungen**
  Springer Verlag, September 2000
- Han J., Kamber M., Pei J.:
  Data Mining: Concepts and Techniques
  3rd edition, Morgan Kaufmann Publishers, March 2011
- M. M. Breunig, H.-P. Kriegel, R. Ng, J. Sander
  **OPTICS-OF: Identifying Local Outliers**
  In Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), Prague, Czech Republic: 262–270, 1999.
- H.-P. Kriegel, M. Schubert, A. Zimek
  **Angle-Based Outlier Detection in High-dimensional Data**
  In Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Las Vegas, NV: 444–452, 2008.