
Non-Standard-Datenbanken

Approximationstechniken

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme



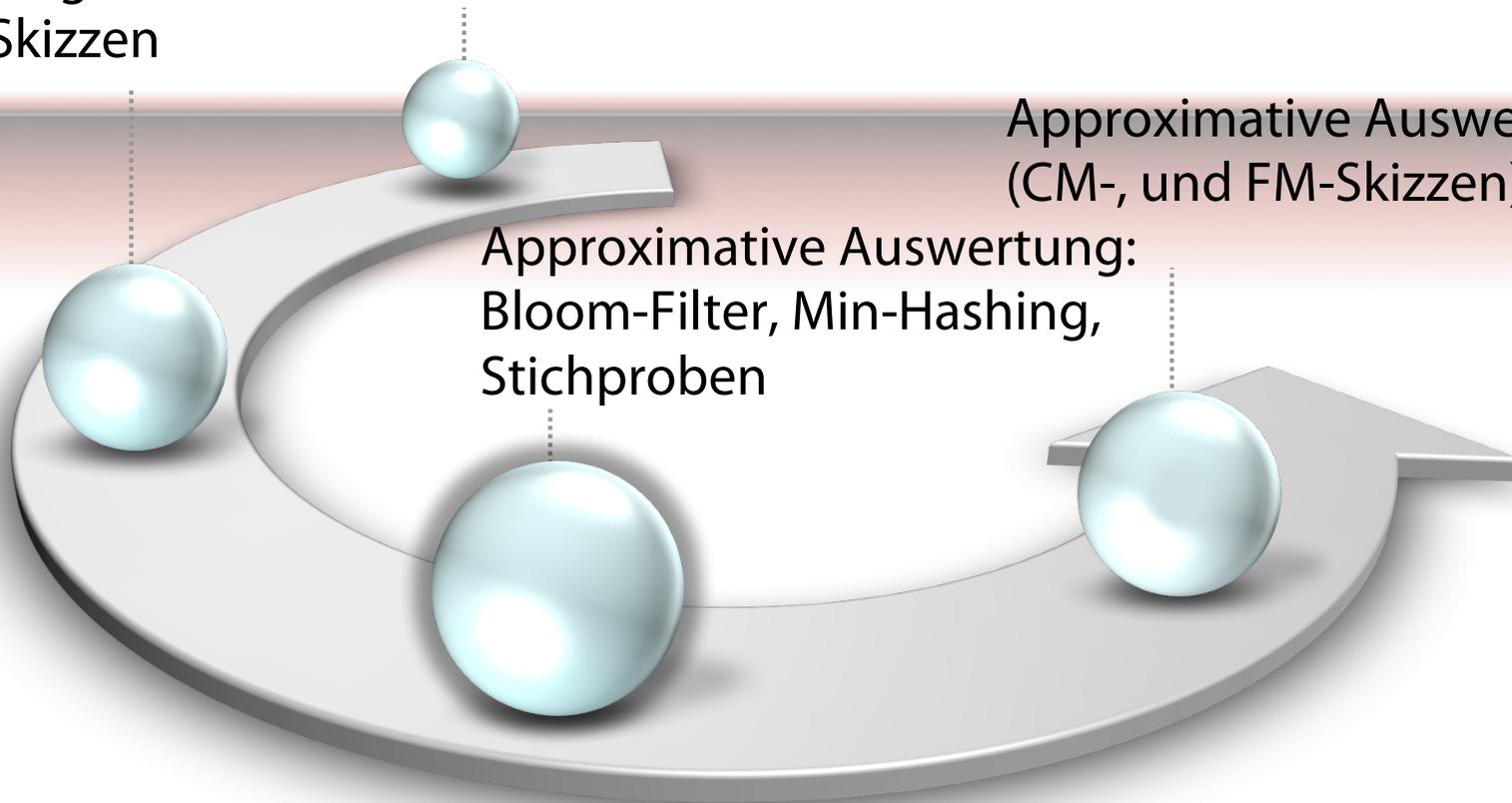
Übersicht

Intervall-Skyline,
inkrementelle
Auswertungstechniken
Skizzen

Stromkonzept, relationale Anfragesprachen,
exakte Auswertung, QoS-Angaben

Approximative Auswertung
(CM-, und FM-Skizzen)

Approximative Auswertung:
Bloom-Filter, Min-Hashing,
Stichproben



Filterung von Datenströmen

- **Jedes Element eines Datenstroms ist ein Tupel**
- Gegeben sei eine Liste von Schlüsseln S
- **Bestimme, welche Tupel eines Strom in S sind**
- **Offensichtliche Lösung: Hashtabelle**
 - Aber was machen wir, wenn **nicht genügend Hauptspeicher bereitsteht**, um alle Elemente von S in einer Hashtabelle zu platzieren
 - Z.B. könnten wir Millionen von Filtern auf einem Strom evaluieren

Anwendungen

- **Email-Spamfilterung**

- Nehmen wir an, es sind 10^9 “gute” E-Mail-Adressen bekannt
- Falls eine E-Mail von einer dieser Adressen kommt, ist sie kein Spam

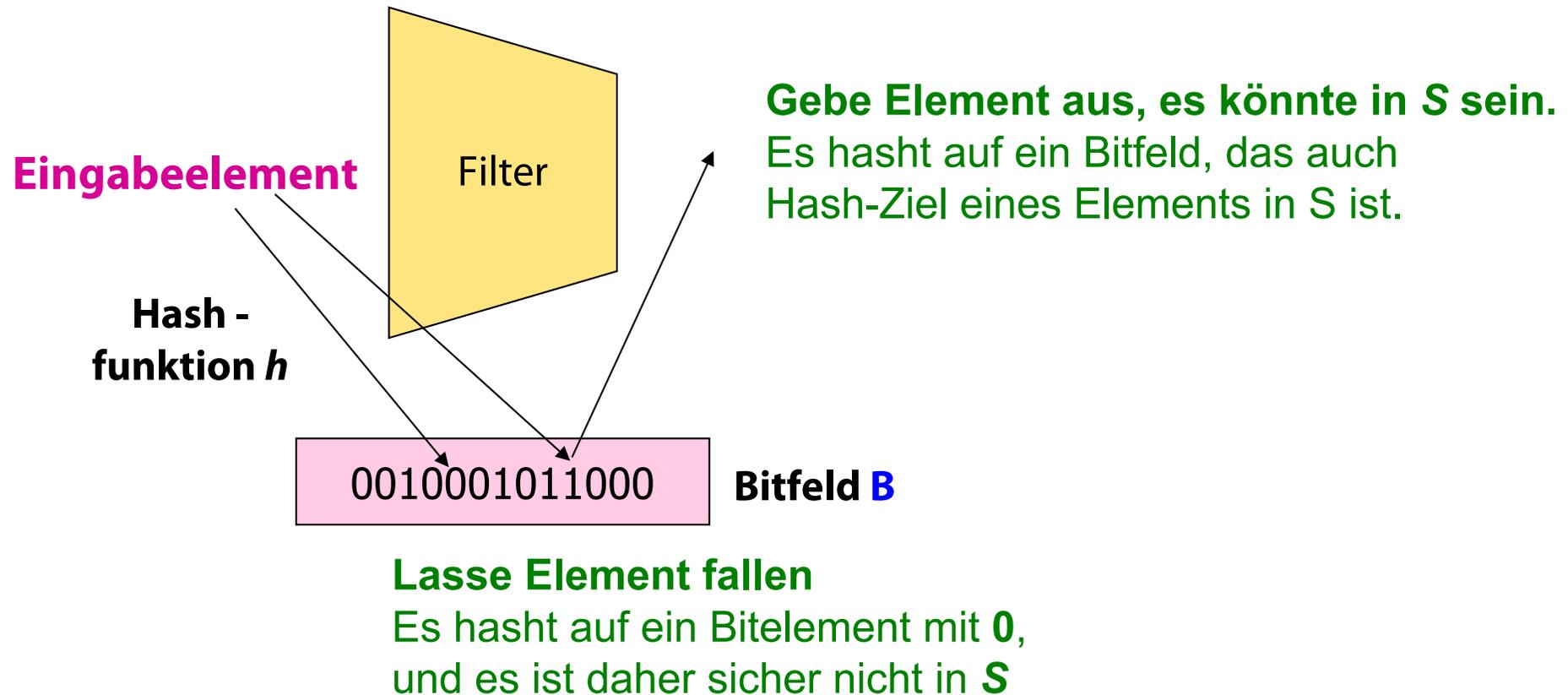
- **Publish-subscribe-Systeme**

- Nehmen wir an, jemand möchte Nachrichten zu bestimmten Themen zugesandt bekommen
- Interessen werden durch Mengen von Schlüsselworten repräsentiert
- Schlüsselworte von Interessenten müssen mit Nachrichtenelementen abgeglichen werden

Erster Ansatz (1)

- Gegeben eine Menge von Schlüsseln S , die zur Filterung dienen
- Erzeuge **Bitfeld B** aus n Bits, anfangs alle **0**
- Wähle eine **Hash-Funktion h** mit Bereich $[0, n)$
- Hashe jedes Mitglied von $s \in S$ auf eines der n Bitfeldelemente und setze das Bit auf **1**: $B[h(s)] := 1$
- Hashe jedes Element a des Stroms und gebe nur diejenigen aus, die auf ein Bit hashen, das auf **1** gesetzt ist
 - **Ausgabe a** falls $B[h(a)] = 1$

Erster Ansatz (2)



Verfahren erzeugt falsch-positive aber keine falsch-negativen Ausgaben

- Falls Element in S , wird es sicher ausgegeben, ansonsten möglicherweise auch

Erster Ansatz (3)

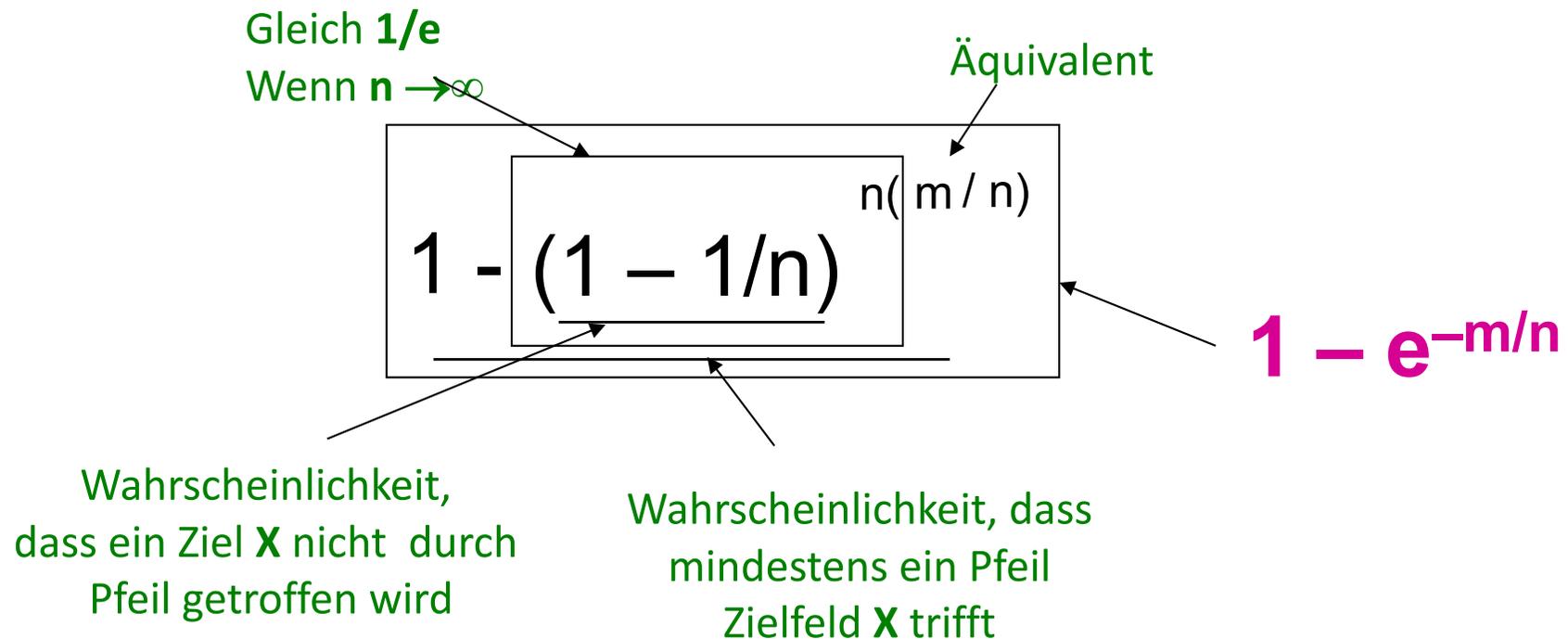
- $|S| = 1$ Milliarde E-Mail-Adressen
- $|B| = 1\text{GB} = 8$ Milliarden Bits
- Falls die E-Mail-Adresse in S ist, wird sie sicher auf ein Bitfeldelement gehasht, das mit **1** besetzt ist, so wird sie immer durchgelassen (*keine falsch-negativen Resultate*)
- Ungefähr **1/8** der Bits sind mit **1** besetzt, also werden ca. **1/8** der Adressen nicht in S durchgelassen (*falsch-positive Resultate*)
 - Tatsächlich sind es weniger als **1/8**, weil mehr als eine Adresse auf das gleiche Bit gehasht wird

Analyse: Werfen von Pfeilen (1)

- **Genauere Analyse der Anzahl falsch positiver Ausgaben**
- **Betrachtung:** Wenn wir m Pfeile in n gleichermaßen wahrscheinliche Zielfelder werfen, **was ist die Wahrscheinlichkeit, dass ein Zielfeld mindestens einen Pfeil abbekommen?**
- **In unserem Fall:**
 - **Zielfelder** = Bits/Elemente des Hashfeldes
 - **Pfeile** = Hashwerte der Eingabeelemente

Analyse: Werfen von Pfeilen(2)

- Wir haben m Pfeile, n Zielfelder
- **Was ist die Wahrscheinlichkeit, dass ein Zielfeld mindestens einen Pfeil abbekommt?**



Analyse: Werfen von Pfeilen(3)

- **Anteil der 1en im Bitfeld B =**
= **Wahrscheinlichkeit eines falsch-positiven Resultats**
= **$1 - e^{-m/n}$**
- **Beispiel: 10^9 Pfeile, $8 \cdot 10^9$ Zielfelder**
 - Anteil der **1en** in **B** = **$1 - e^{-1/8} = 0.1175$**
 - Vergleiche mit Schätzung: **$1/8 = 0.125$**

Bloom-Filter

- Betrachte: $|\mathbf{S}| = m, |\mathbf{B}| = n$
- Verwende k unabhängige Hash-Funktionen h_1, \dots, h_k
- **Initialisierung:**
 - Besetze \mathbf{B} mit **0en**
 - Hashe jedes Element $s \in \mathbf{S}$ mit jeder Hash-Funktion h_i ,
setze $\mathbf{B}[h_i(s)] = 1$ (für jedes $i = 1, \dots, k$)
- **Laufzeit:**
 - Wenn ein Stromelement mit Schlüssel x erscheint
 - Falls $\mathbf{B}[h_i(x)] = 1$ für alle $i = 1, \dots, k$ dann deklarieren, dass x in \mathbf{S}
 - Also, x hasht auf ein Hashfeld mit **1** für jede Hash-Funktion $h_i(x)$
 - Sonst, ignoriere Element x

Es gibt nur
ein Bitfeld \mathbf{B} !

Bloom Filter – Analyse (1)

- **Welcher Anteil des Bitfeldes ist mit 1en gefüllt?**
 - Werfe $k \cdot m$ Pfeile auf n Zielfelder
 - Der Anteil an **1**en ist also $(1 - e^{-km/n})$
- Aber wir haben k unabhängige Hash-Funktionen und wir lassen x nur durch, falls **alle** k Hashvorgängen das Element x auf eine Bitfeld mit Wert **1** abbilden
- Also ist die **Wahrscheinlichkeit für ein falsch-positives** Resultat $= (1 - e^{-km/n})^k$

Burton H. Bloom: *Space/Time Trade-offs in Hash Coding with Allowable Errors*. In: *Communications of the ACM*. Band 13, Nr. 7, Juli 1970

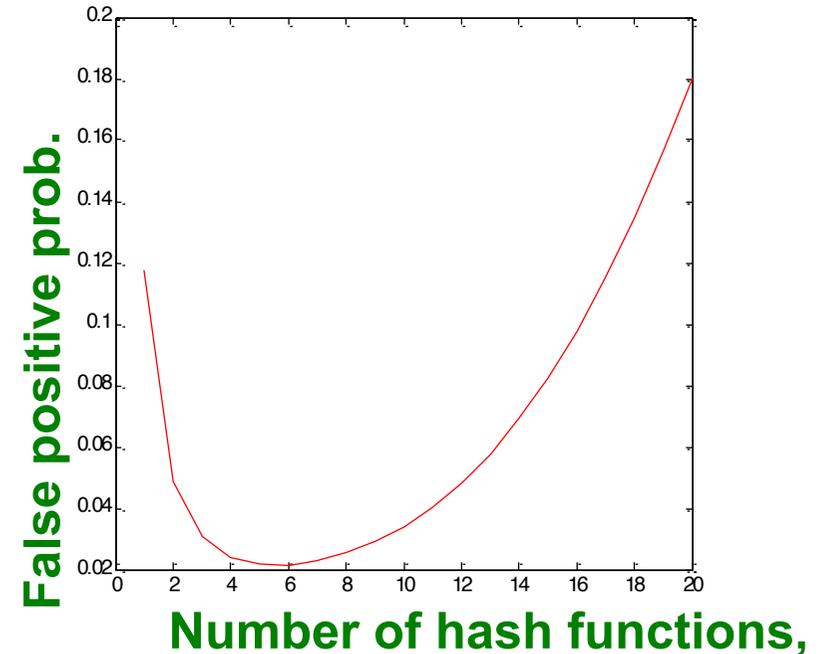
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmids.org>

IM FOCUS DAS LEBEN 12



Bloom-Filter – Analyse (2)

- **$m = 1$ billion, $n = 8$ billion**
 - **$k = 1$: $(1 - e^{-1/8}) = 0.1175$**
 - **$k = 2$: $(1 - e^{-1/4})^2 = 0.0493$**
- **Was passiert, wenn k steigt?**
- **“Optimaler” Wert für k : $n/m \ln(2)$**
 - **In unserem Fall:** Optimal ist **$k = 8 \ln(2) = 5.54 \approx 6$**
 - **Fehler bei $k = 6$: $(1 - e^{-6/8})^6 = 0.0215$**



Bloom Filter: Zusammenfassung

- **Bloom-Filter garantieren, dass falsch-positive Ausgaben selten sind und verwenden begrenzten Speicher**
 - Gut zur Vorverarbeitung bevor teure Tests durchgeführt werden
- **Geeignet für Hardware-Implementation**
 - Hash-Vorgänge können parallelisiert werden
- **Ist es besser 1 großes B zu haben oder k kleine Bs?**
 - **Es ist gleich: $(1 - e^{-km/n})^k$ vs. $(1 - e^{-m/(n/k)})^k$**
 - **Aber 1 großes B ist einfacher**

Filterung von Datenströmen

- **Fenster-weise Verarbeitung eines Datenstroms**
- Gegeben sei eine Liste von Schlüsselworten **S**
- **Bestimme Ähnlichkeit von Tupeln im Fenster mit Schlüsseln in S**
- **Anwendung: Erkennen von bestimmten Mustern**

Min-Hashing

- Gegeben zwei Mengen A und B von Objekten
- Zum Beispiel sind A und B Text-Dokumente mit Wörtern.
- Ein oft benutztes Ähnlichkeitsmaß ist der **Jaccard Koeffizient**:

$$J(A,B) := \frac{|A \cap B|}{|A \cup B|}$$

Idee hinter Min-Hashing

- Berechne für alle Elemente einer Menge A eine Hashfunktion, die $a \in A$ einem Integer-Wert zuweist.
- Betrachte den kleinsten dieser Werte $h_{min}(A)$
- Wie groß ist die Wahrscheinlichkeit, dass zwei Mengen A und B dieser min-Wert identisch ist?

Beispiel

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$

$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$

$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

Objekt	Hashwert
Apfel	263228505
Birne	1512322680
Tomate	2655545330
Orange	2426202636
Zitrone	4196103473
Gurke	3877529293
Kokosnuss	2846776306
Aprikose	41486361
Banane	4138599105

Beispiel

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$

$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$

$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

Objekt	Hashwert
Apfel	263228505
Birne	1512322680
Tomate	2655545330
Orange	2426202636
Zitrone	4196103473
Gurke	3877529293
Kokosnuss	2846776306
Aprikose	41486361
Banane	4138599105

Wir haben also

$$h_{min}(S_1) = 263228505$$

$$h_{min}(S_2) = 2426202636$$

$$h_{min}(S_3) = 41486361$$

Min-Hashing (2)

- Die Wahrscheinlichkeit $Pr[h_{min}(A) = h_{min}(B)]$ steht in direkter Verbindung zum Jaccard-Koeffizienten:

$$Pr[h_{min}(A) = h_{min}(B)] = \frac{|A \cap B|}{|A \cup B|}$$

Anwendung

- Suche nach ähnlichen Mengen: Betrachte nur Paare von Mengen, deren min-Wert identisch ist
- Bzw., nehme $Pr[h_{min}(A) = h_{min}(B)]$ als Näherung für den Jaccard-Koeffizienten
- Wie gut funktioniert das?

Beispiel

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$

$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$

$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

Objekt	Hashwert
Apfel	263228505
Birne	1512322680
Tomate	2655545330
Orange	2426202636
Zitrone	4196103473
Gurke	3877529293
Kokosnuss	2846776306
Aprikose	41486361
Banane	4138599105

Wir haben also

$$h_{min}(S_1) = 263228505$$

$$h_{min}(S_2) = 2426202636$$

$$h_{min}(S_3) = 41486361$$

Die Min-Werte sind für alle drei Mengen unterschiedlich. D.h. wir bekommen eine **Schätzung von 0** für den Jaccard-Koeffizienten, obwohl dieser für $S_1 S_2$ nicht 0 ist.

Beispiel: zweite Hashfunktion

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$

$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$

$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

Objekt	Hashwert 1	Hashwert 2
Apfel	263228505	3747123490
Birne	1512322680	2691314150
Tomate	2655545330	618073562
Orange	2426202636	167471787
Zitrone	4196103473	3040259855
Gurke	3877529293	2452364051
Kokosnuss	2846776306	2613259214
Aprikose	41486361	2319295075
Banane	4138599105	765635320

Beispiel: zweite Hashfunktion

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$

$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$

$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

Objekt	Hashwert 1	Hashwert 2	Wir haben also
Apfel	263228505	3747123490	$h1_{min}(S_1) = 263228505$
Birne	1512322680	2691314150	$h1_{min}(S_2) = 2426202636$
Tomate	2655545330	618073562	$h1_{min}(S_3) = 41486361$
Orange	2426202636	167471787	$h2_{min}(S_1) = 167471787$
Zitrone	4196103473	3040259855	$h2_{min}(S_2) = 167471787$
Gurke	3877529293	2452364051	$h2_{min}(S_3) = 765635320$
Kokosnuss	2846776306	2613259214	
Aprikose	41486361	2319295075	
Banane	4138599105	765635320	

Beispiel 2

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$

$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$

$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

$$h1_{min}(S_1) = 263228505$$

$$h1_{min}(S_2) = 2426202636$$

$$h1_{min}(S_3) = 41486361$$

$$h2_{min}(S_1) = 167471787$$

$$h2_{min}(S_2) = 167471787$$

$$h2_{min}(S_3) = 765635320$$

	Schätzung	Echt
$J(S_1, S_2)$	1/2	2/6
$J(S_2, S_3)$	0	0
$J(S_1, S_3)$	0	0

Min-Hashing - Mehrere min-Werte bzw. Hashfunktionen

Mehrere (k) Hash-Funktionen mit je einem min-Wert

- Betrachte k (unabhängige) Hashfunktionen, die jeweils einen min-Wert liefern.
- Approximiere $J(A,B)$ mit Anteil der übereinstimmenden min-Werte.
- Wie im Beispiel zuvor.

Mehrere (k) min-Werte & eine Hashfunktionen

- Betrachte nur eine Hashfunktion, aber nehme von dieser die k kleinsten Werte.

Der Fehler ist bei beiden Schemata $O(1/\sqrt{k})$. Ohne Beweis

Approximation mit
Stichproben und Skizzen
Probabilistische Garantien



Stromverarbeitungsmodelle – interner Speicher A[·]

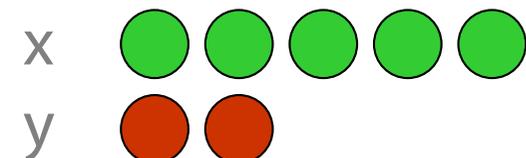
- **Absolute Werte (Zeitreihenmodell)**

- Jeder neue Wert pro Zeiteinheit x wird gespeichert
(x aus Menge von Zeitpunkten)

- **Akkumulierte Werte**

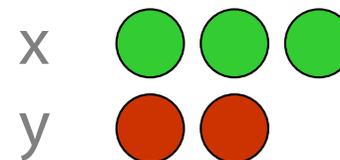
- **Nur Eingänge (Akkumulation von Zählern)**

- Zähler immer ≥ 0
- Multimengenmodell
- (x, y sind Objekte bzw. Objekttypen)



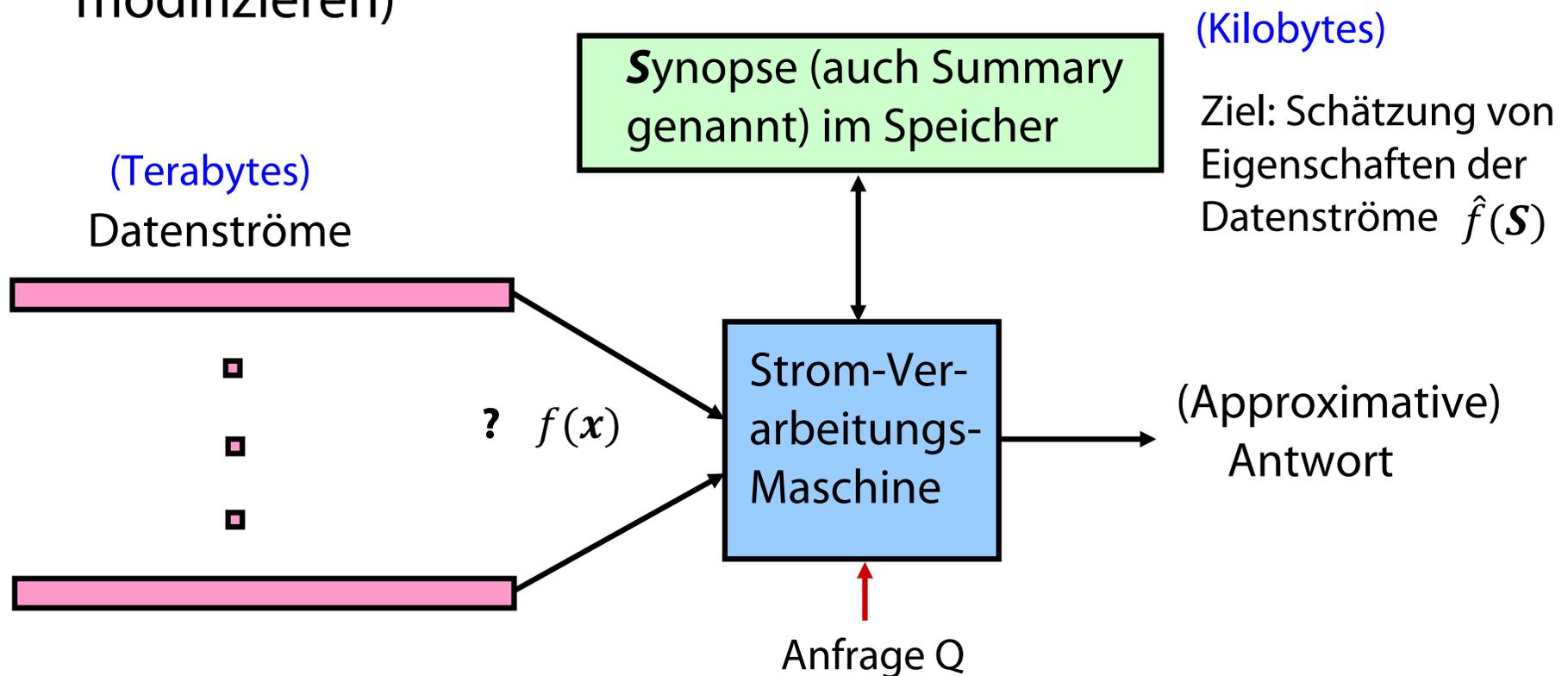
- **Eingänge und Abgänge (Drehkreuzmodell)**

- Multimengenmodell
- (x, y sind Objekte bzw. Objekttypen)



Approximation

- Daten werden nur einmal betrachtet und
- Speicher für Zustand (stark) begrenzt: $O(\text{poly}(\log(|\text{Strm}|)))$
- Rechenzeit pro Tupel möglichst klein (auch um Zustand zu modifizieren)



Synopsen

Eine kleine Zusammenfassung großer Datenmengen, die ziemlich gut die interessierenden statistischen Größen abschätzen

Nützliche Eigenschaften:

- Es ist leicht, ein Element hinzuzufügen
- Unterstützung für Lösungen ("undo")
- Zusammenführung mehrerer Synopsen sollte leicht sein
Wichtig für horizontale Skalierung
- Flexibler Einsatz: Unterstützung mehrerer Anfragetypen

Was kann über einem Strom berechnet werden?

Einige Funktionen sind leicht:

min, max, sum, ...

Es wird jeweils nur eine einzige Variable benötigt:

- Maximum: **Initialisiere $s \leftarrow 0$**

For element x do $s \leftarrow \max s, x$

- Sum: **Initialisiere $s \leftarrow 0$**

For element x do $s \leftarrow s + x$

Die "Synopsis" ist ein einzelner Wert

Eine solche Synopsis kann zusammengeführt werden

Approximation und Randomisierung

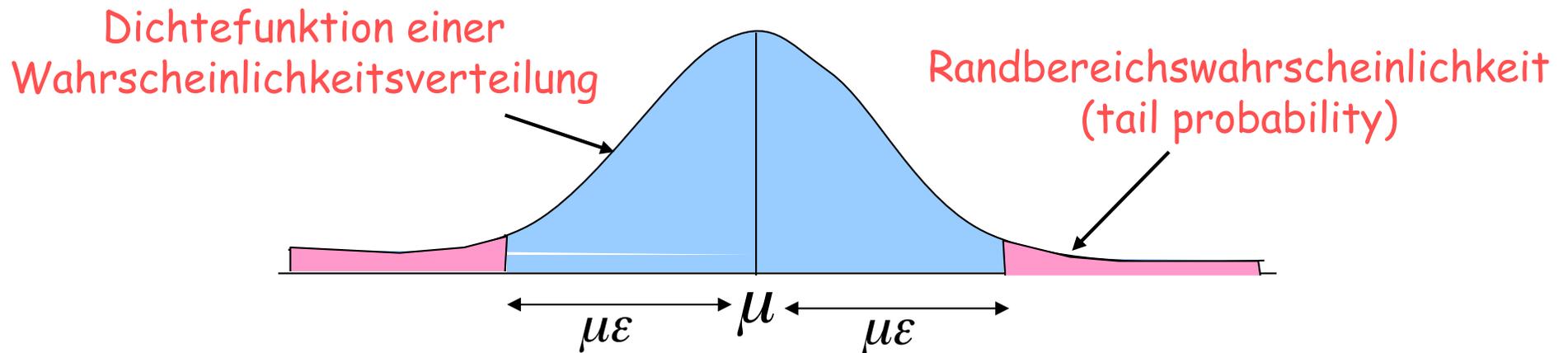
- Viele Probleme sind schwierig, exakt zu berechnen
 - Anzahl der Elemente einer gegebenen Menge von Elementklassen identisch in zwei verschiedenen Strömen?
 - Platzbedarf linear bezogen auf Anzahl der Elementklassen
- **Approximation:** Finde eine Antwort, die korrekt ist bezogen auf einen gegebenen Faktor
 - Finde Antwort im Bereich von $\pm 10\%$ des korrekten Ergebnisses
 - **Genereller:** finde $(1 \pm \varepsilon)$ -Faktor-Approximation
- **Randomisierung:** Erlaube Fehler mit kleiner Wahrscheinlichkeit
 - Eine von 10000 Antworten darf falsch sein
 - **Genereller:** Erfolgswahrscheinlichkeit $(1-\delta)$
- *Approximation **und** Randomisierung:* (ε, δ) -Approximationen

Probabilistische Garantien

- Benutzerbestimmte (ε, δ) -Approximationen
 - Beispiel: Tatsächliche Antworten innerhalb von Faktor 1.1 mit Wahrscheinlichkeit ≥ 0.8 ($\varepsilon = 0.1, \delta = 0.2$)
- Wie können wir prüfen, ob durch ein spezielles Verfahren die Gütekriterien eingehalten werden?
 - Verwendung von Randbereichsabschätzungen

Exkurs: Randbereichsabschätzungen

- Generelle Grenzen der Randbereichswahrscheinlichkeit einer Zufallsvariable (Wahrscheinlichkeit, dass Wert weit vom Erwartungswert abweicht)



- Basisungleichungen: Sei X eine Zufallsvariable mit Erwartungswert μ und Varianz $\text{Var}[X]$. Dann gilt für alle $\epsilon > 0$

Markov:

$$\Pr(X \geq (1 + \epsilon)\mu) \leq \frac{1}{1 + \epsilon}$$

Chebyshev:

$$\Pr(|X - \mu| \geq \mu\epsilon) \leq \frac{\text{Var}[X]}{\mu^2 \epsilon^2}$$

Stichproben (Abtastung, sampling): Grundlagen

Idee: Ein kleiner Ausschnitt (Stichprobe) S einer Datenmenge repräsentiert die Eigenschaften aller Daten

- Zur schnellen Approximierung, wende “modifizierte” Anfrage auf S an
- Beispiel: `select agg from R where R.e is odd` (n=12)

Datenstrom: 9 3 5 2 7 1 6 5 8 4 9 1

Ausschnitt S : 9 5 1 8

- agg = avg \rightarrow Mittel der ungeraden Elemente in S Antwort: 5
- agg = count \rightarrow Mittel über Summanden abgel. aus e in S mit
 - n falls e ungerade
 - 0 falls e geradeAntwort: $12 * 3/4 = 9$

Randbereichsschätzungen für Summen

- **Hoeffding-Ungleichung:** Seien X_1, \dots, X_m unabhängige Zufallsvariablen mit $0 \leq X_i \leq r$.

Sei $\bar{X} = \frac{1}{m} \sum_i X_i$ und μ der Erwartungswert von \bar{X}

Dann gilt für alle $\varepsilon > 0$

$$\Pr(|\bar{X} - \mu| \geq \varepsilon) \leq 2 \exp \frac{-2m\varepsilon^2}{r^2}$$

- Anwendung für Anfragen mit Mittelwert-Operation (avg):
 - m ist Größe der Untermenge der Stichprobe S , die das Prädikat erfüllt (3 im Beispiel)
 - r ist Bereich der Elemente in der Stichprobe (8 im Beispiel)
- Anwendung auf Zählfragen (count):
 - m ist Größe der Stichprobe S (4 im Beispiel)
 - r ist Anzahl der Elemente n im Strom (12 im Beispiel)

Randbereichsschätzungen für Summen (2)

Für Bernoulli-Versuche (zwei mögliche Erg.) sogar starke Eingrenzung möglich:

- **Chernoff Abschätzung:** Seien X_1, \dots, X_m unabhängige Zufallsvariable für Bernoulli-Versuche, so dass $\Pr[X_i=1] = p$ ($\Pr[X_i=0] = 1-p$)
- Sei $X = \sum_i X_i$ und $\mu = mp$ der Erwartungswert von X
- Dann gilt, für jedes $\varepsilon > 0$

$$\Pr(|X - \mu| \geq \mu\varepsilon) \leq 2 \exp^{\frac{-\mu\varepsilon^2}{2}}$$

- Verwendung für Anzahlanfrage (count queries):
 - m ist Größe der Stichprobe S (4 im Beispiel)
 - p ist Anteil der ungeraden Elemente im Strom (2/3 im Beispiel)
- Anmerkung: Chernoff-Abschätzung liefert engere Grenzen für Anzahlanfragen als die Hoeffding-Ungleichung

Stichproblem für Datenströme

- **(1)** Betrachtung einer **festgelegten Teilmenge** der Elemente, die im Datenstrom auftreten (z.B. 1 von 10)
- **(2)** Extraktion einer **Teilmenge fester Größe** über einem potentiell unendlichen Strom
 - Zu jedem Zeitpunkt k liegt eine Teilmenge der Größe s vor
 - **Was sind die Eigenschaften der Teilmenge, die verwaltet wird?**
Für alle Zeitpunkte k , soll jedes der k Elemente, die betrachtet wurden, die gleiche Wahrscheinlichkeit haben, in die Teilmenge übernommen zu werden

Verwendung einer festgelegten Teilmenge

- **Szenario:** Auswertung von Anfragen an eine Suchmaschine
 - **Strom von Tupeln:** (user, query, time)
 - **Stromanfrage: How often did a user run the same query in a single day**
 - Ann: Platz für **1/10** des Stroms
- **Naive Lösung:**
 - Generiere ganzzahlige Zufallszahl [**0..9**] für jede Anfrage
 - Speichere Anfrage falls Zahl = **0**, sonst ignoriere Stromelement (eine Anfrage)

Probleme des naiven Ansatzes

- **Auswertung von: Welcher Bruchteil von Anfragen sind Duplikate?**
 - Nehme an, jeder Benutzer sendet x Anfragen einmal und d Anfragen zweimal (insgesamt also $x+2d$ Anfragen)
 - **Korrekte Antwort: $d/(x+d)$**
 - **Vorgeschlagene Lösung: Verwende 10% der auftretenden Anfragen aus dem Strom**
 - Stichprobe enthält $x/10$ der Einfachanfragen und mindestens $2d/10$ der Mehrfachanfragen
 - Aber es werden nur $d/100$ der Paare wirklich als Duplikate erkannt
 - $d/100 = 1/10 \cdot 1/10 \cdot d$
 - Von d "Duplikaten" kommen $18d/100$ nur genau einmal in der Stichprobe vor
 - $18d/100 = ((1/10 \cdot 9/10) + (9/10 \cdot 1/10)) \cdot d$
 - **Also wäre die Antwort mit der naiven Stichprobe:**
 $(d/100)/(x/10) + d/100 + 18d/100 = d/10x + 19d/100$

Lösung: Stichprobe mit Benutzern

- Wähle **1/10** der **Benutzer** und werte alle ihre Anfragen aus
- Verwende Hash-Funktion, mit der (name, user id) gleichverteilt auf 10 Werte abgebildet wird, speichere Anfragen in Hashtabelle

Generalisierte Lösung

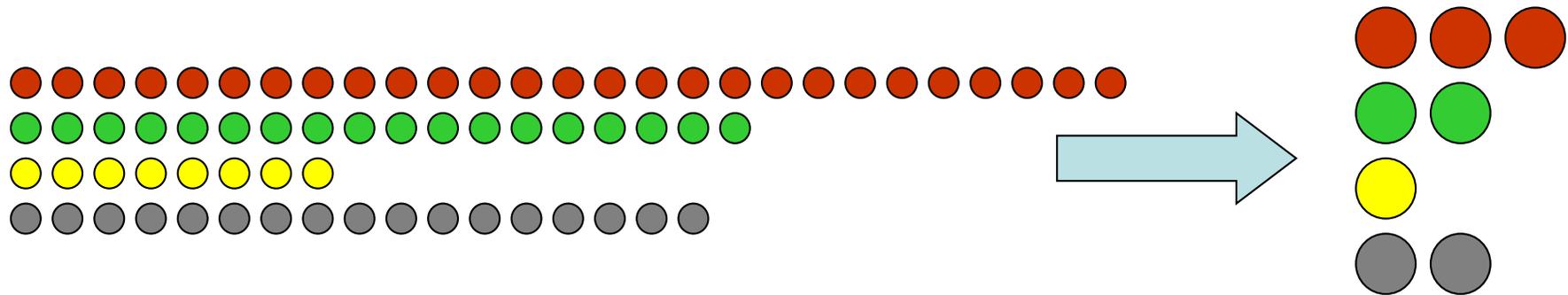
- **Strom von Tupeln mit Schlüsseln:**
 - Schlüssel = Teilmenge der Tupelkomponenten
 - Z.B.: Tupel = (user, search, time); Schlüssel ist **user**
 - Wahl der Schlüssel hängt von der Anwendung ab
- **Bestimmung einer Stichprobe als a/b Bruchteil des Stroms:**
 - Hashen der Schlüssel auf **b** Hashwerte (Eimer)
 - Wähle das Tupel, falls Hashwert höchstens **a**



Wie 30% Stichproblem generieren?

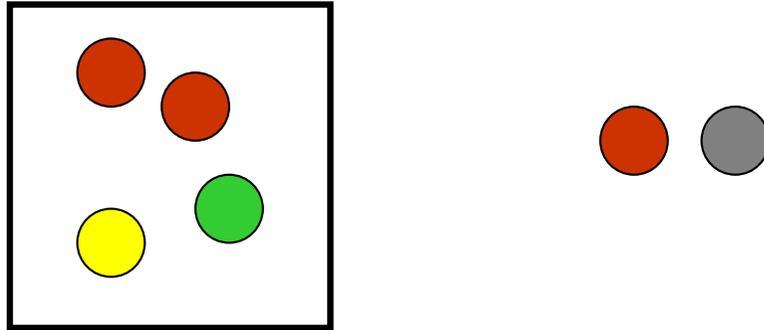
Hash auf $b=10$ Eimer, wähle Tupel, falls in einen der ersten drei Eimer gehasht

Stichproben für Datenströme



- Kernproblem: Wähle Stichprobe von m Elementen gleichverteilt aus dem Strom
- **Herausforderung:** Länge des Stroms nicht bekannt
 - Wann/wie oft Stichprobe entnehmen?
- Zwei Lösungsvorschläge, für verschiedene Anwendungen:
 - Reservoir Sampling (aus den 80ern?)
 - Min-wise Sampling (aus den 90ern?)

Reservoir Sampling (Reservoir bildet Synopse)



- Übernahme erste m Elemente
- Wähle i -tes Element ($i > m$) mit Wahrscheinlichkeit m/i
- Wenn neues Element gewählt, ersetze beliebiges Element im Reservoir
- Optimierung: Wenn i groß, berechne jeweils nächstes Element (überspringe Zwischenelemente)

Reservoir Sampling – Analyse

- Analysieren wir einen einfachen Fall: $m = 1$
- Wahrscheinlichkeit, dass i -tes Element nach n Elementen die Stichprobe bildet:
 - Wahrscheinlichkeit, dass i bei Ankunft gewählt wird
× Wahrscheinlichkeit dass i “überlebt”

$$\begin{aligned} & \frac{1}{i} \times \left(1 - \frac{1}{i+1}\right) \times \left(1 - \frac{1}{i+2}\right) \dots \left(1 - \frac{1}{n-1}\right) \times \left(1 - \frac{1}{n}\right) \\ &= \frac{1}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \dots \frac{\cancel{n-2}}{\cancel{n-1}} \times \frac{\cancel{n-1}}{n} = 1/n \end{aligned}$$

- Analyse für $m > 1$ ähnlich, Gleichverteilung leicht zu zeigen
- Nachteil: Nicht einfach parallelisierbar

Min-wise Sampling

- Für jedes Element: Wähle Anteil zwischen 0 und 1
- Wähle Element mit kleinstem Anteilswert



- Jedes Element hat gleiche Chance, kleinstes Element zu werden, also gleichverteilt
- Anwendung auf mehrere Ströme, dann Zusammenführung

S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson.
Synopsis diffusion for robust aggregation in sensor
networks. In ACM SenSys, 2004

Non-Standard-Datenbanken

Approximationstechniken

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme



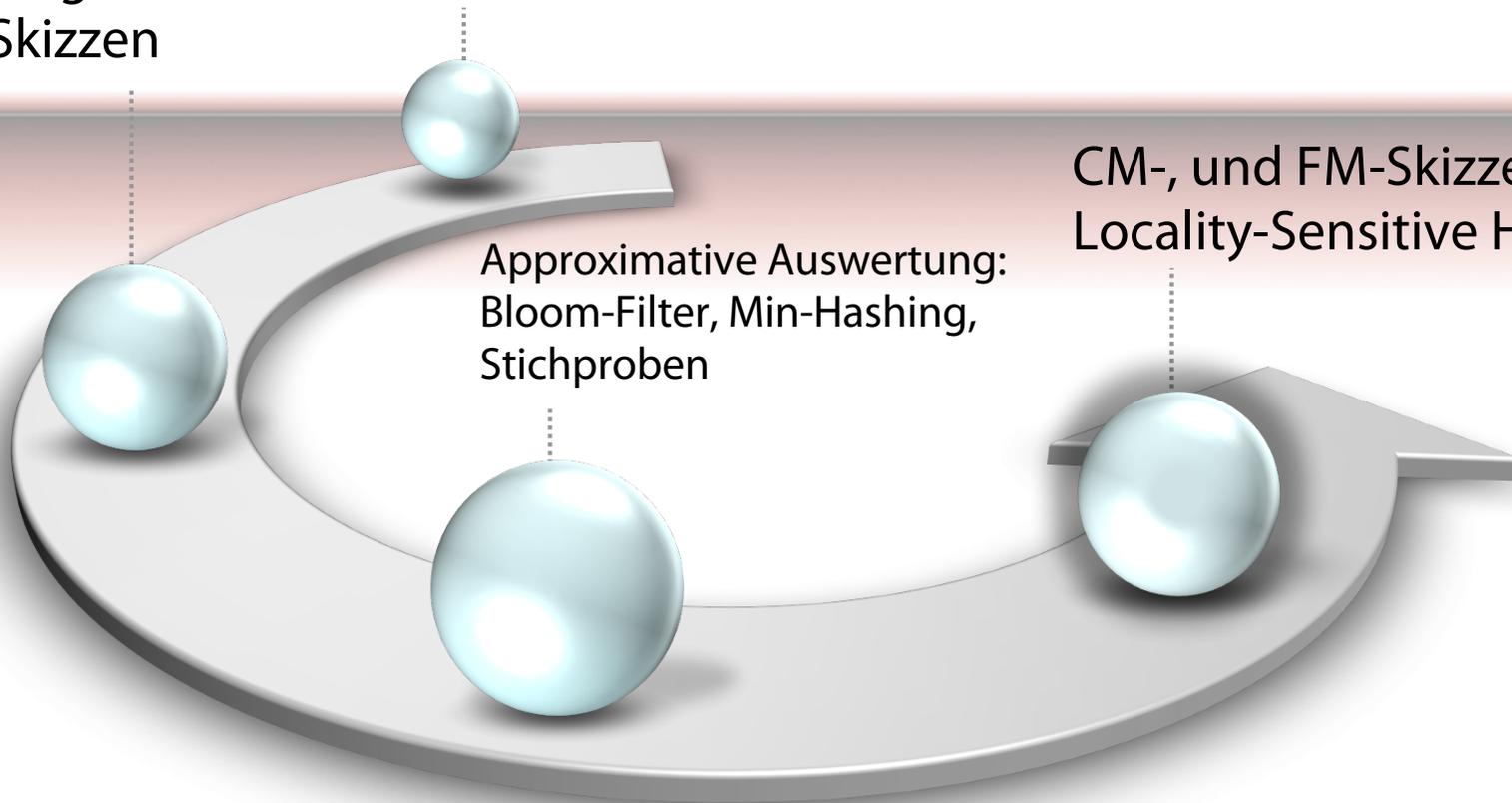
Übersicht

Intervall-Skyline,
inkrementelle
Auswertungstechniken
Skizzen

Stromkonzept, relationale Anfragesprachen,
exakte Auswertung, QoS-Angaben

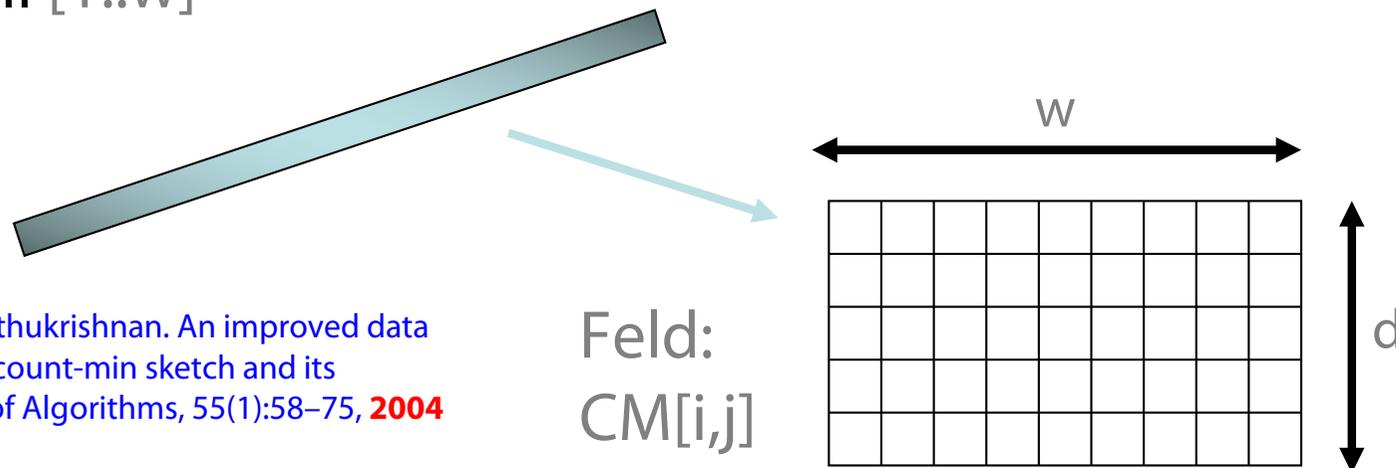
CM-, und FM-Skizzen,
Locality-Sensitive Hashing

Approximative Auswertung:
Bloom-Filter, Min-Hashing,
Stichproben



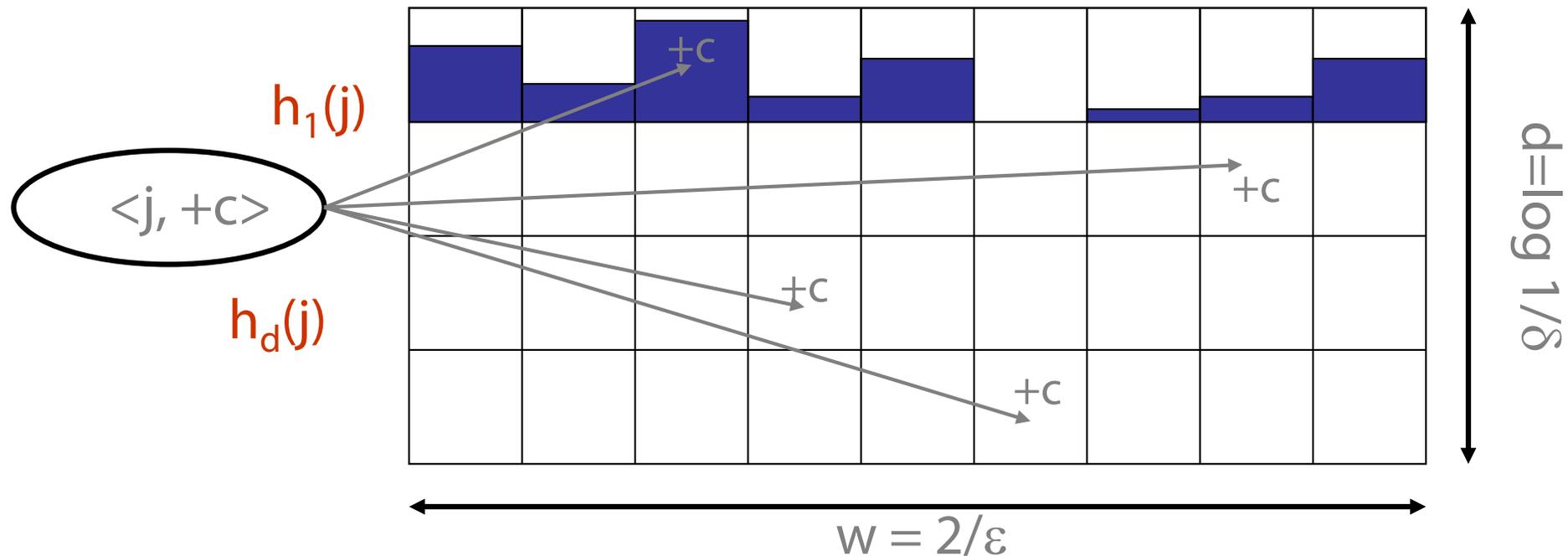
Count-Min-Skizzen (CM-Skizzen)

- Einfache Synopse (Skizze), Basis für viele Strom-Untersuchungsaufgaben (stream mining tasks)
 - hauptsächlich für "Anzahl Elemente pro Typ" (item frequencies)
 - für Zählerakkumulierung und Drehkreuzmodell
- Eingabestrom intern als Vektor A der Dimension N repräsentiert
- Skizze als kleines Feld CM der Größe $w \times d$ dargestellt
- Verwende d Hashfunktionen zur Abbildung der A -Elemente auf Intervall $[1..w]$



G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2004

CM-Skizzen – Struktur



- Jedes $A[j]$ wird auf CM-Eintrag abgebildet
 - $h()$'s paarweise unabhängig
- Schätze $A[j]$ über den Ausdruck $\min_k \{ CM[k, h_k(j)] \}$
- Parallelisierung: Verschmelzung eintragsweise durch Summierung verschiedener CM-Matrizen möglich

CM-Skizzen – Garantien

- CM-Approximierungsfehler bei Punktanfragen kleiner als $\epsilon \|A\|_1$ mit Platzbedarf $O(1/\epsilon \log 1/\delta)$ [Cormode, Muthukrishnan '04]
 - Wahrscheinlichkeit eines größeren Fehlers kleiner als $1-\delta$
 - Ähnliche Garantien für Bereichsanfragen, Quantile, Verbundgrößen (join size), ...
- Hinweise
 - Zähler überschätzen durch Hash-Kollisionen
 - Wie begrenztbar in jedem Feld?
 - **Nutze Unabhängigkeit über Zeilen,** um Konfidenz für $\min\{\}$ Schätzung zu erhöhen

G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. LATIN 2004, J. Algorithm 55(1): 58-75, **2004**

Minos Garofalakis A Quick Intro to Data Stream Algorithmics – CS262



CM-Skizzen – Analyse

Schätze $A'[j] = \min_k \{ CM[k, h_k(j)] \}$



CM-Skizzen – Analyse

Schätze $A'[j] = \min_k \{ CM[k, h_k(j)] \}$

- Analyse: In k 'ter Zeile, $CM[k, h_k(j)] = A[j] + X_{k,j}$
 - $X_{k,j} = \sum A[i] \mid h_k(i) = h_k(j)$
 - $E[X_{k,j}] = \sum A[i] \cdot \Pr[h_k(i) = h_k(j)] \leq 1/w \cdot \sum A[i]$
 $\leq (\epsilon/2) \cdot \sum A[i] = \epsilon \|A\|_1 / 2$ (paarweise Unabh. von h)

Fehler, wenn

- $\Pr[X_{k,j} \geq \epsilon \|A\|_1] = \Pr[X_{k,j} \geq 2E[X_{k,j}]] \leq 1/2$ über **Markov Ungl.**
- Also, $\Pr[A'[j] \geq A[j] + \epsilon \|A\|_1] = \Pr[\forall k. X_{k,j} > \epsilon \|A\|_1] \leq 1/2^{\log 1/\delta} = \delta$
- Endergebnis: $A[j] \leq A'[j]$ und
mit Wahrscheinlichkeit $1-\delta$ gilt $A'[j] < A[j] + \epsilon \|A\|_1$

Tafelbild der Abschätzung $A'[j]$ für $A[j]$

$$A'[j] = \min_k \{ CM[k, h_k(j)] \}$$

Analyse der k -ten Zeile

$$CM[k, h_k(j)] = A[j] + X_{k,j}$$

$$X_{k,j} := \sum_{h_k(i)=h_k(j)} A[i]$$

$$E[X_{k,j}] = \sum_i A[i] \cdot \Pr(h_k(i)=h_k(j))$$

$$\Pr(h_k(i)=h_k(j)) \leq \frac{1}{w} = \frac{\epsilon}{2} \quad (h \text{ paarw. unabh.})$$

Tafelbild der Abschätzung $A'[j]$ für $A[j]$ (2)

$$E[X_{k,j}] \leq \frac{\epsilon}{2} \sum_i A[i] = \frac{\epsilon}{2} \|A\|_1$$

Fehler wenn $X_{k,j} \geq \epsilon \|A\|_1$

$$P[X_{k,j} \geq \epsilon \|A\|_1] = P[X_{k,j} \geq \underbrace{2E[X_{k,j}]}_{(1+\epsilon)\mu}] \leq \frac{1}{2}$$

Markov-
Abschätzung: $P(X \geq (1+\epsilon)\mu) \leq \frac{1}{1+\epsilon}$

$$P[\forall k. X_{k,j} \geq \epsilon \|A\|_1] \leq \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \dots \cdot \frac{1}{2}}_d = \frac{1}{2^d} = \frac{1}{2^{\log_2 \frac{1}{\delta}}} = \delta$$

Zählen der Anzahl der verschiedenen Elemente

- **Problem:**

- Datenstrom enthält Elemente aus Grundmenge der Größe N
- Gesucht ist Anzahl der verschiedenen Elemente in einem gesamten bisherigen Strom zu einem Zeitpunkt, zu dem Fenster ausgewertet wird

- **Naiver Ansatz:**

Speichere gesehene Elemente in Hashtabelle als Synopse

Anzahl **verschiedener** Werte abschätzen

- **Aufgabe**: Finde Anzahl der verschiedenen Werte in einem Strom von Werten aus $[1, \dots, N]$ (**count distinct**)
 - **Statistik**: Anzahl von Arten oder Klassen in Population
 - **Datenbanken**: Selektivitätsschätzung in Anfrageoptimierung
 - **Netzwerkbeobachtung**: IP-Adressen mit unterschiedlichem Ziel, Quelle/Ziel-Paare, angeforderte URLs usw

- Beispiel (N=64)

Datenstrom:

3	2	5	3	2	1	7	5	1	2	3	7
---	---	---	---	---	---	---	---	---	---	---	---

Anzahl der verschiedene Werte: 5

- Naiver Ansatz: Hash-Tabelle für alle gesehenen Elemente
- Schwierig auch für CM (gedacht für Multimengen)

Flajolet-Martin Approach

- Wähle Hashfunktion h zur Abbildung von N Elementen auf mindestens $\log_2 N$ Bits
- Für jedes Stromelement a , sei $r(a)$ die Anzahl der Nullen am Ende von $h(a)$
 - $r(a)$ = Position der ersten 1 von rechts
 - Z.B. sei $h(a) = 12$, dann ist 12 gleich 1100 binär, also $r(a) = 2$
- Speichere $R = \text{maximales } r(a) \text{ bisher}$
 - $R = \max_a r(a)$, über alle Stromelemente a bisher
- **Geschätzte Anzahl unterschiedlicher Elemente: $\approx 2^R$**

P. Flajolet and G. N. Martin. Probabilistic counting. In IEEE Conference on Foundations of Computer Science, pages 76–82, 1983. Journal version in Journal of Computer and System Sciences, 31:182–209, 1985.

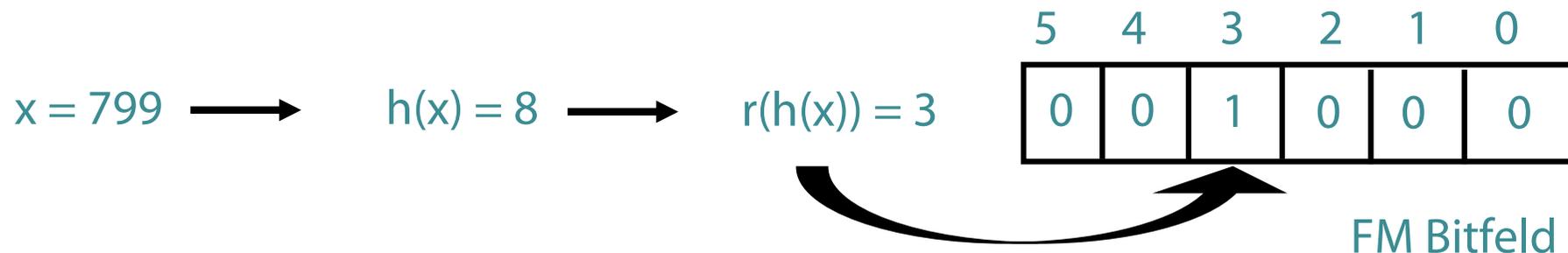


Warum funktioniert das? Intuition

- $h(a)$ bildet a mit **gleicher Wahrscheinlichkeit** auf jeden von N möglichen Werten ab
- Dann ist $h(a)$ eine Sequenz von $\log_2 N$ Bits, wobei 2^{-r} der Anteil aller a s mit r Nullen am Ende ist
 - Ca. 50% der a s hashen auf *****0**
 - Ca. 25% der a s hashen auf ****00**
 - Wenn also $r(a)=2$ Nullen hinten stehen (i.e., Hash ergibt ***100**), dann haben wir wahrscheinlich **ca. 4** verschiedene Elemente gesehen
- **Also braucht es ein Hash auf 2^r Elementen bevor man eines mit 0-Suffix der Länge r sieht**

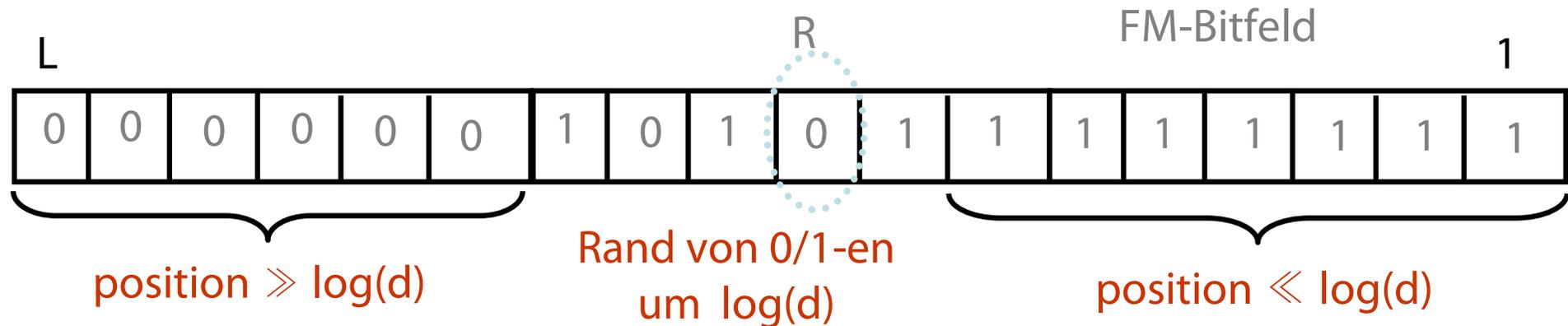
FM-Skizzen [Flajolet, Martin 85]

- Verwende Hashfunktion h' zur Abbildung von Eingabeelementen auf i mit Wahrscheinlichkeit 2^{-r}
 - also $\Pr[h'(x) = 1] = 1/2, \Pr[h'(x) = 2] = 1/4, \Pr[h'(x)=3] = 1/8 \dots$
 - Konstruiere $h'()$ aus gleichverteilter Hashfunktion und anschließend dem “Zählen der Nullen am Ende” durch r
- Aufbau FM-Skizze= Bitfeld von $L = \log N$ Bits
 - Initialisiere Bitfelder of 0
 - Für jeden neuen Wert x , setze $FM[r(x)] = 1$



FM-Skizzen – Analyse

- Bei d verschiedenen Werten, erwarte Abbildung von $d/2$ Werten nach $FM[1]$, $d/4$ nach $FM[2]$...



- Sei $R =$ Position der rechtesten 0 in FM, Indikator für $\log(d)$
- [FM85] zeigen, dass $E[R] = \log(\phi d)$, mit $\phi = .7735$
- Schätzung $d = c2^R$ für Skalierungskonstante $c \approx 1.3$
- Mittelung mit verschiedenen Hashfunktionen dient zur Verbesserung des Ergebnisses
- Wieviele Hashfunktionen für best. Anforderung benötigt?

Herleitung der Schätzung von d

$$E(R) = \log(\Phi d) \quad [FM85]$$

$$= \log \Phi + \log d$$

$$E[R] - \log \Phi = \log d$$

$$\log 2^{E[R]} - \log \Phi = \log d$$

$$\log \frac{2^{E[R]}}{\Phi} = \log d$$

$$d \approx \frac{2^R}{\Phi} \approx c 2^R \quad c \approx 1.3..$$

FM-Skizzen – Eigenschaften

- Mit $O(1/\varepsilon^2 \log 1/\delta)$ Hashfunktionen, $(1 \pm \varepsilon)$ Genauigkeit mit Wahrscheinlichkeit mindestens $1-\delta$

Ohne Beweis: siehe [Bar-Yossef et al.' 02], [Ganguly et al.' 04]

- 10 Funktionen \approx 30% Fehler, 100 Funkt. $<$ 10% Fehler

- *Bei Löschung:* Verwende Zähler statt Bits

- +1 für Einfügungen, -1 für Löschungen

- *Komposition:* komponentenweise OR bzw. +

$$\begin{array}{cccccc} 6 & 5 & 4 & 3 & 2 & 1 \\ \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{1} \end{array} + \begin{array}{cccccc} 6 & 5 & 4 & 3 & 2 & 1 \\ \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & \boxed{0} & \boxed{1} \end{array} = \begin{array}{cccccc} 6 & 5 & 4 & 3 & 2 & 1 \\ \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{1} \end{array}$$

- Schätze $|S_1 \cup \dots \cup S_k| = \text{Kardinalität der Vereinigungsmenge}$

Stichproben und Skizzierung: Zusammenfassung

- Zentrale Idee für viele Stromanalyseverfahren
 - Momente/Verbundaggregate, Histogramme, top-k, Meistfrequentierte Elemente, andere Analyseprobleme, ...
- Stichproben eher generelle Repräsentation eines Datensatzes
 - Einfache Stichproben (sampling) kaum für Strommodelle mit Ein- und Abgängen (Drehkreuzmodell) geeignet (aber es gibt auch hier neue Arbeiten)
- Skizzierung eher für speziellen Zweck
 - FM-Skizze für “Anzahl der Typen”,
 - CM-Skizze für “Anzahl Elemente pro Typ” (auch: Verbundgröße bzw. Momentenschätzung ...)

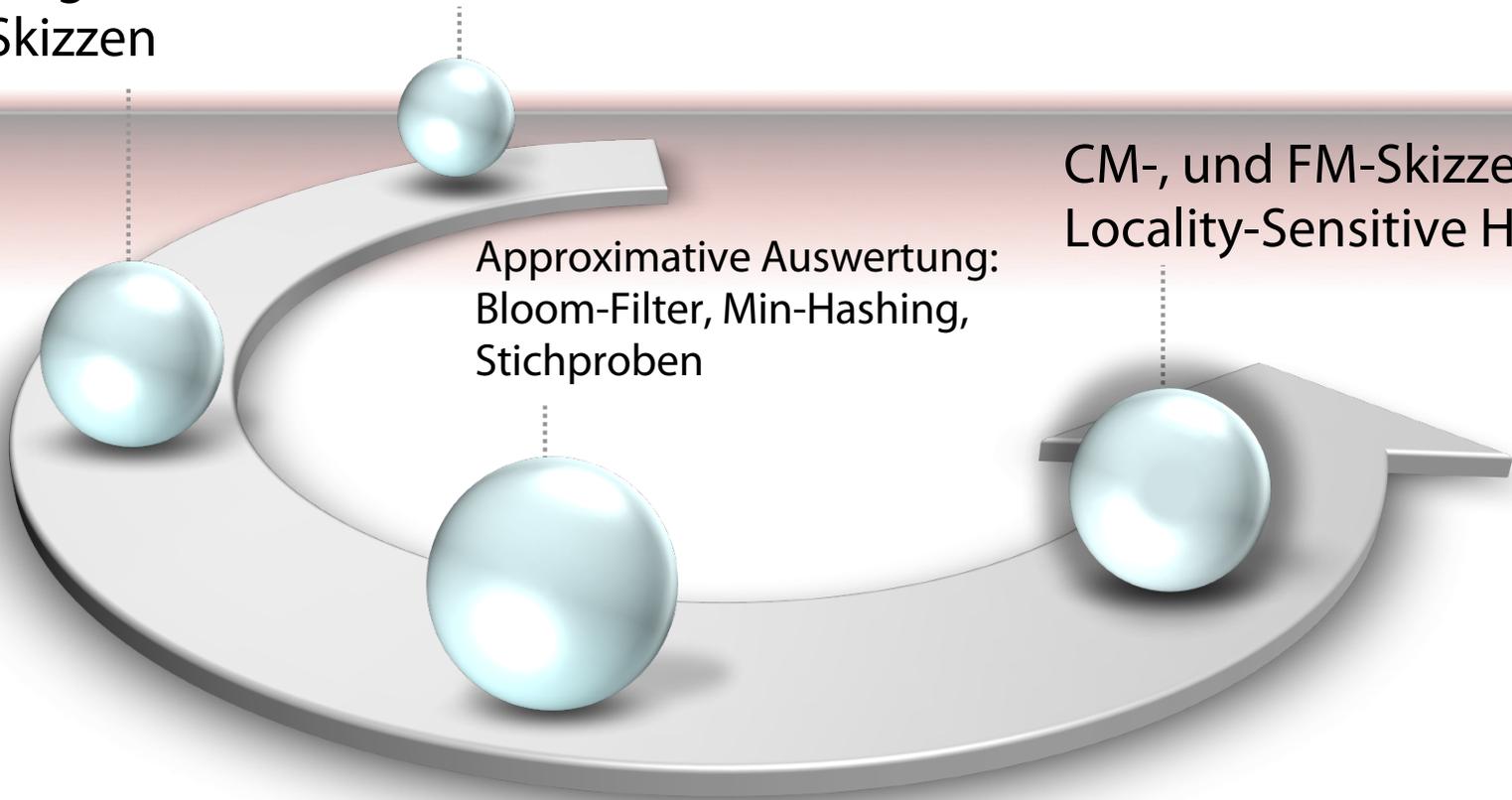
Übersicht

Intervall-Skyline,
inkrementelle
Auswertungstechniken
Skizzen

Stromkonzept, relationale Anfragesprachen,
exakte Auswertung, QoS-Angaben

CM-, und FM-Skizzen,
Locality-Sensitive Hashing

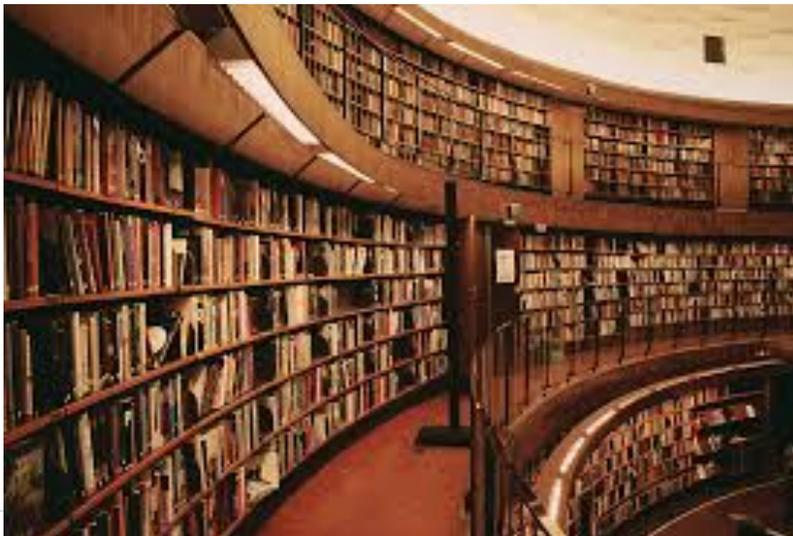
Approximative Auswertung:
Bloom-Filter, Min-Hashing,
Stichproben



Approximative Ähnlichkeitssuche in hohen Dimensionen

- Vergleiche Daten im Strom mit Daten aus großer Kollektion
- Suche k ähnlichste Dateneinheiten in Kollektion
- Anzahl der Vergleichsmerkmale kann sehr groß werden

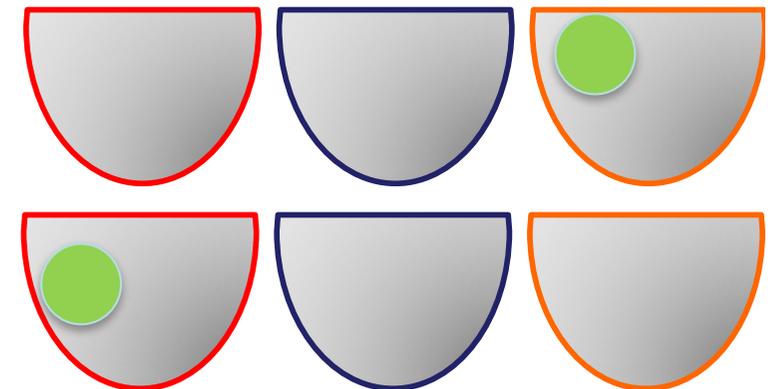
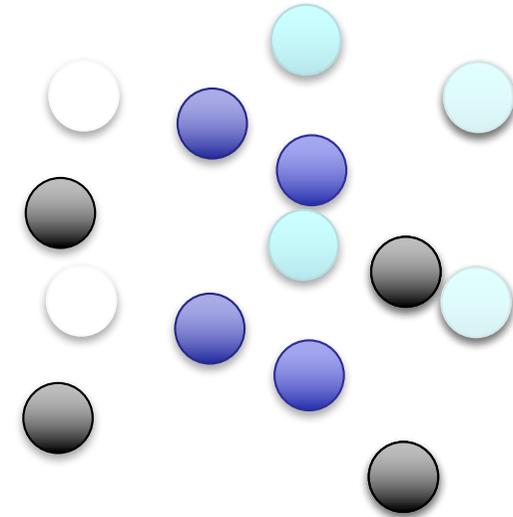
Gionis, A.; Indyk, P.; Motwani, R. (1999). "Similarity Search in High Dimensions via Hashing". Proceedings of the 25th Very Large Database (VLDB) Conference



Approximative Reduktion der Vergleiche

Locality Sensitive Hashing (LSH):

- Jedes Objekt wird in Partition ghasht
- Objekte in der gleichen Partition (Kollision) häufig ähnlich
- Es gibt falsch-negative Ergebnisse (zwei ähnliche Objekten in verschiedenen Partitionen)
- Fehler kann klein gehalten werden
- Um die Wahrscheinlichkeit von Kollisionen zu erhöhen, werden mehrere Hashtabellen aufgebaut
- Vergleiche Objekte mit Objekten aus gleicher Partition in jeder Hashtabelle



Was bedeutet "locality-sensitive"

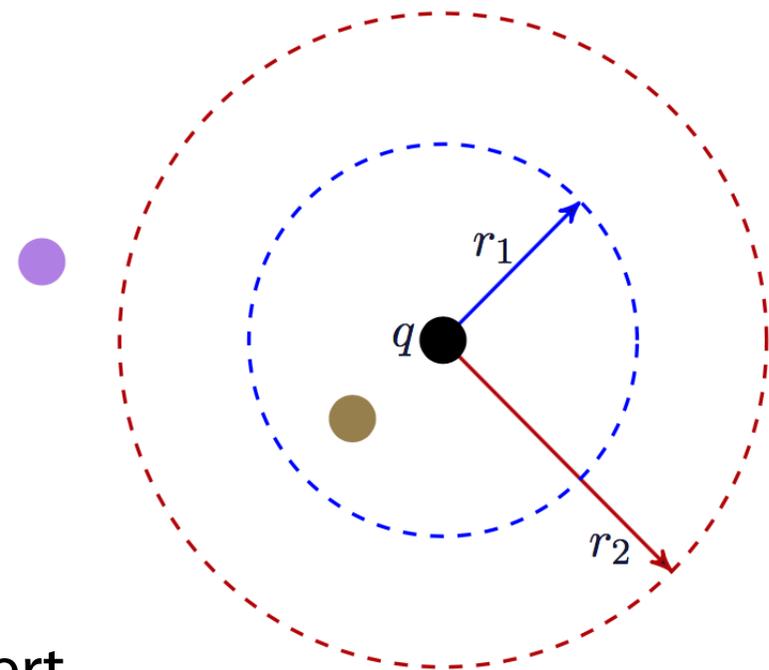
- Eine Familie von Hashfunktionen $H = \{h : S \rightarrow U\}$ heißt (r_1, r_2, p_1, p_2) -sensitiv falls die folgenden beiden Bedingungen für beliebige Punkte $q, v \in S$ gelten:

- Falls $\text{dist}(q, v) \leq r_1$
dann $\Pr_H(h(q) = h(v)) \geq p_1$

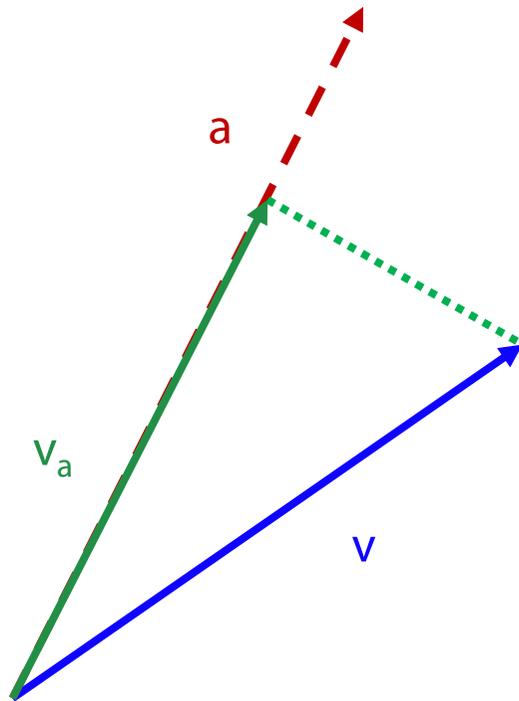
- Falls $\text{dist}(q, v) > r_2$
dann $\Pr_H(h(q) = h(v)) \leq p_2$

- Analyse:

Falls $r_1 < r_2$ und $p_1 > p_2$ gilt:
Ähnliche Objekte bekommen
häufiger den gleichen Hash-Wert
als weniger ähnliche

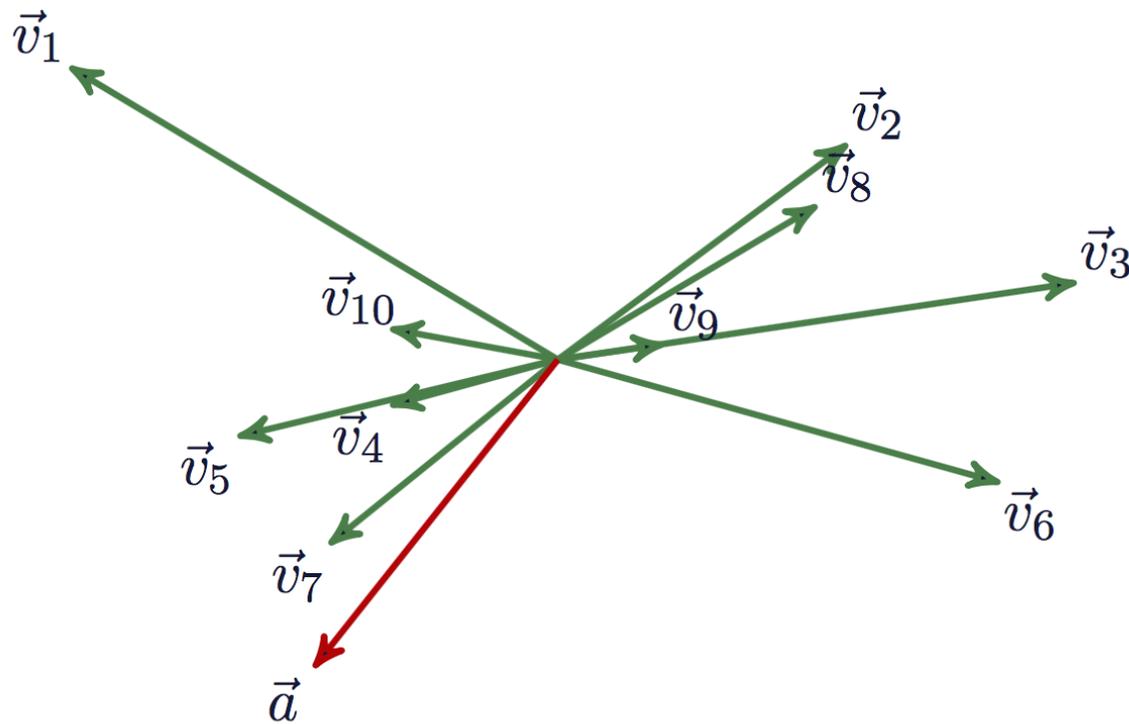


D-dimensionaler Vektorraum: Idee für Hashfunktion



- Gegeben Vektor v im d -dimensionalen Raum
- Wie kann man v mit weniger als d Werten beschreiben?
- Nehme weiteren Vector a
- Betrachte Skalarprodukt $a \cdot v$
- Ähnliche Vektoren v haben ähnliche Werte $a \cdot v$
- Also, erste Idee für Hashfunktion: $h(v) := \lfloor a \cdot v \rfloor$

LSH: Beispiel



$$\vec{v}_1 = (-2.1, -1.1)$$

$$\vec{v}_2 = (2.4, -1.0)$$

$$\vec{v}_3 = (-1.6, -0.5)$$

$$\vec{v}_4 = (-3.9, -0.9)$$

$$\vec{v}_5 = (-1.0, -0.2)$$

$$\vec{v}_6 = (-2.6, 0.0)$$

$$\vec{v}_7 = (-0.7, 1.2)$$

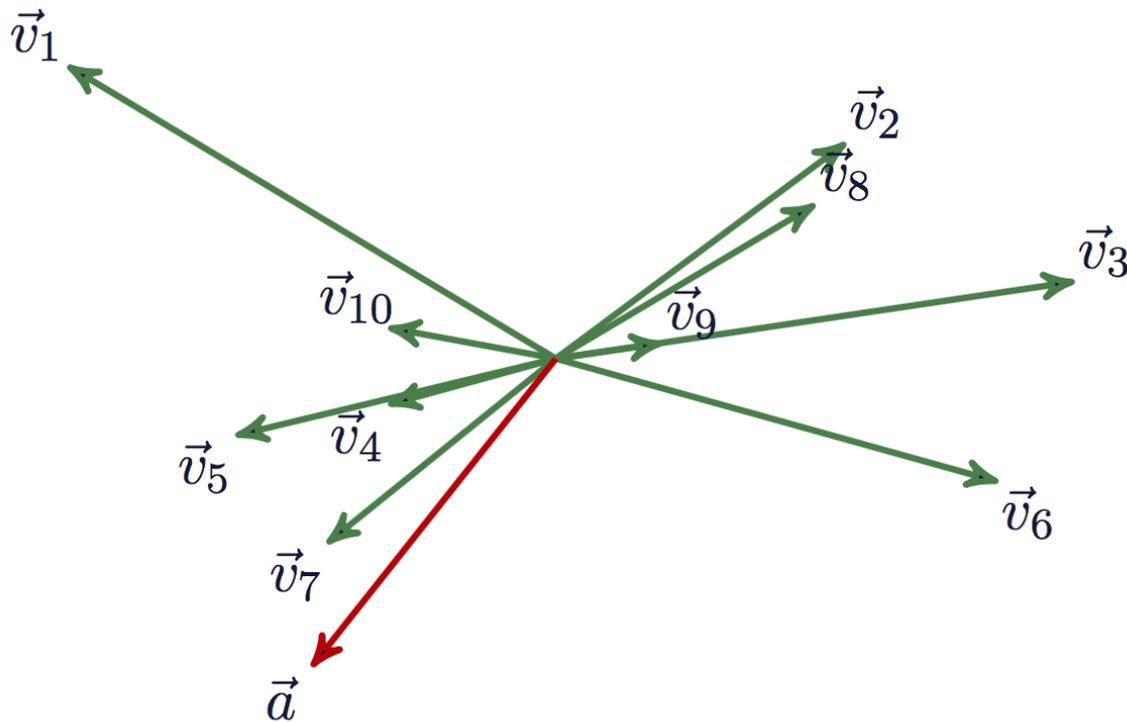
$$\vec{v}_8 = (1.2, -0.7)$$

$$\vec{v}_9 = (-2.6, 0.5)$$

$$\vec{v}_{10} = (-0.8, 1.1)$$

$$\vec{a} = (-1.6, -2)$$

LSH: Beispiel



Mit Hashfunktion

$$h(v) = \lfloor \vec{a} \cdot \vec{v} \rfloor$$

erhalten wir

$$h(\vec{v}_1) = 1$$

$$h(\vec{v}_2) = -6$$

$$h(\vec{v}_3) = -7$$

$$h(\vec{v}_4) = 2$$

$$h(\vec{v}_5) = 4$$

$$h(\vec{v}_6) = -4$$

$$h(\vec{v}_7) = 4$$

$$h(\vec{v}_8) = -5$$

$$h(\vec{v}_9) = -2$$

$$h(\vec{v}_{10}) = 1$$

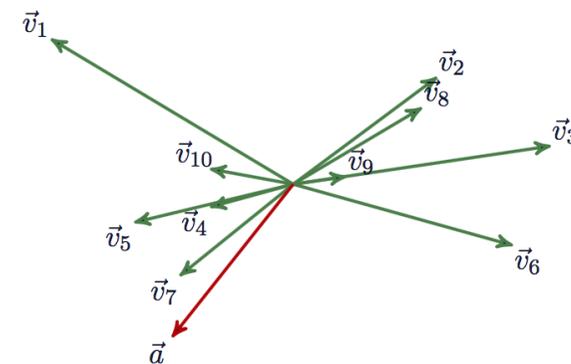
Wir interpretieren die Hashwerte als Label der Buckets, in die wir die Objekte (Vektoren) stecken....

LSH: Beispiel

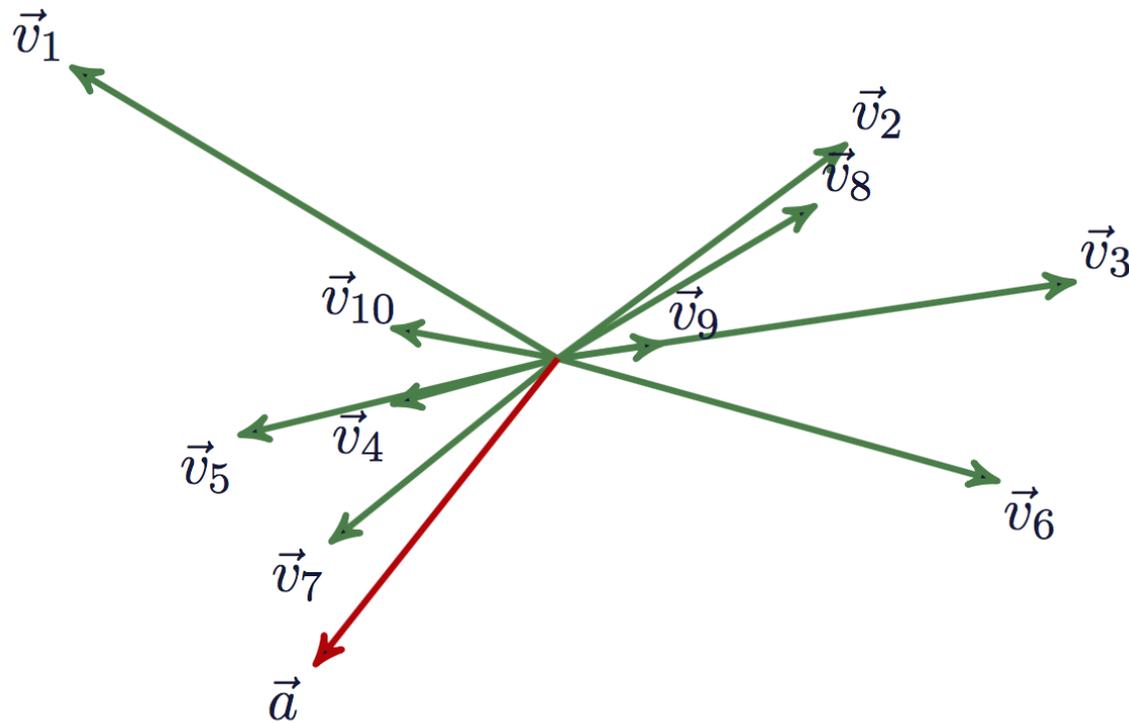
Bucket -7	Bucket -6	Bucket -5
\vec{v}_3	\vec{v}_2	\vec{v}_8
Bucket -4	Bucket -3	Bucket -2
\vec{v}_6		\vec{v}_9
Bucket -1	Bucket 0	Bucket 1
		\vec{v}_1 \vec{v}_{10}
Bucket 2	Bucket 3	Bucket 4
\vec{v}_4		\vec{v}_5 \vec{v}_7
Bucket 5	Bucket 6	Bucket 7

Kaum Kollisionen, viele leere Buckets. Was können wir tun?

$$\begin{aligned}
 h(\vec{v}_1) &= 1 \\
 h(\vec{v}_2) &= -6 \\
 h(\vec{v}_3) &= -7 \\
 h(\vec{v}_4) &= 2 \\
 h(\vec{v}_5) &= 4 \\
 h(\vec{v}_6) &= -4 \\
 h(\vec{v}_7) &= 4 \\
 h(\vec{v}_8) &= -5 \\
 h(\vec{v}_9) &= -2 \\
 h(\vec{v}_{10}) &= 1
 \end{aligned}$$



LSH: Beispiel



Betrachte nun
Hashfunktion

$$h(v) = \lfloor \frac{\vec{a} \cdot \vec{v}}{2} \rfloor$$

$$h(\vec{v}_1) = 0$$

$$h(\vec{v}_2) = -3$$

$$h(\vec{v}_3) = -4$$

$$h(\vec{v}_4) = 1$$

$$h(\vec{v}_5) = 2$$

$$h(\vec{v}_6) = -2$$

$$h(\vec{v}_7) = 2$$

$$h(\vec{v}_8) = -3$$

$$h(\vec{v}_9) = -1$$

$$h(\vec{v}_{10}) = 0$$

LSH: Beispiel

Bucket -4	Bucket -3	Bucket -2
\vec{v}_3	\vec{v}_2 \vec{v}_8	\vec{v}_6
Bucket -1	Bucket 0	Bucket 1
\vec{v}_9	\vec{v}_1 \vec{v}_{10}	\vec{v}_4
Bucket 2	Bucket 3	Bucket 4
\vec{v}_5 \vec{v}_7		

Hashwerte:

$$h(\vec{v}_1) = 0$$

$$h(\vec{v}_2) = -3$$

$$h(\vec{v}_3) = -4$$

$$h(\vec{v}_4) = 1$$

$$h(\vec{v}_5) = 2$$

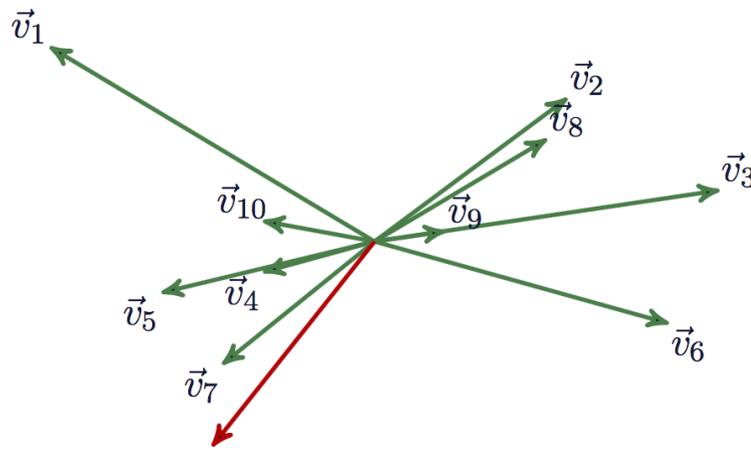
$$h(\vec{v}_6) = -2$$

$$h(\vec{v}_7) = 2$$

$$h(\vec{v}_8) = -3$$

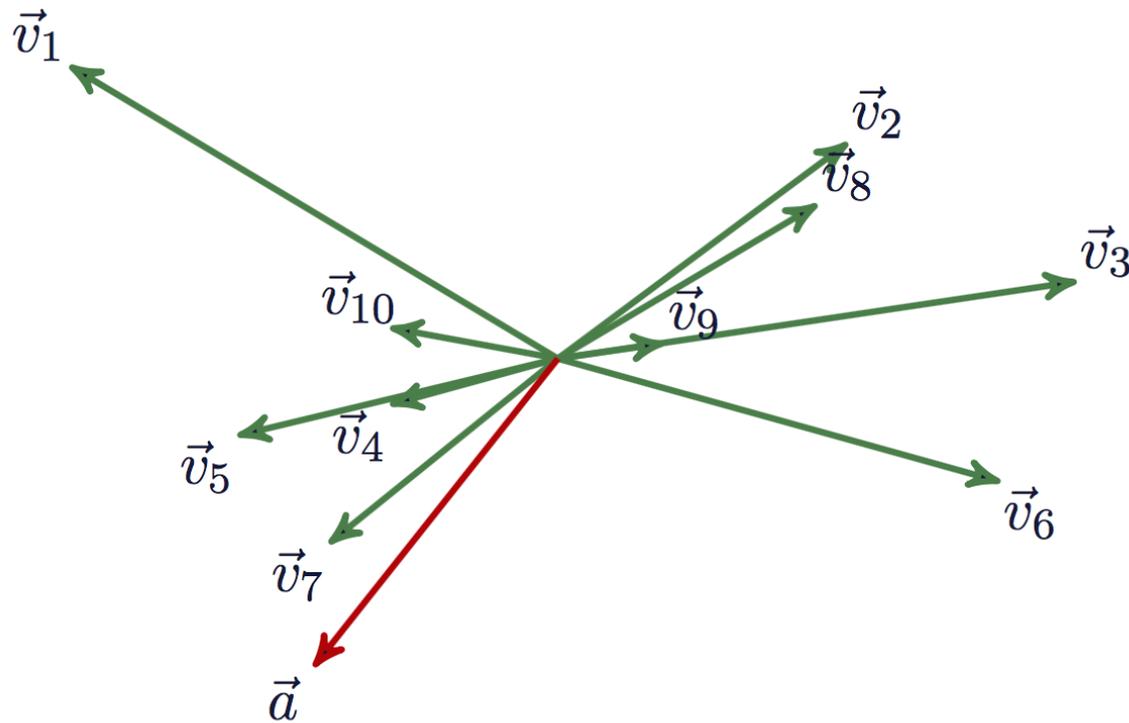
$$h(\vec{v}_9) = -1$$

$$h(\vec{v}_{10}) = 0$$



Wir sehen den Effekt: Weniger Buckets, mehr Kollisionen.

LSH: Beispiel



Mit Hashfunktion

$$h(v) = \lfloor \frac{\vec{a} \cdot \vec{v}}{4} \rfloor$$

$$h(\vec{v}_1) = 0$$

$$h(\vec{v}_2) = -2$$

$$h(\vec{v}_3) = -2$$

$$h(\vec{v}_4) = 0$$

$$h(\vec{v}_5) = 1$$

$$h(\vec{v}_6) = -1$$

$$h(\vec{v}_7) = 1$$

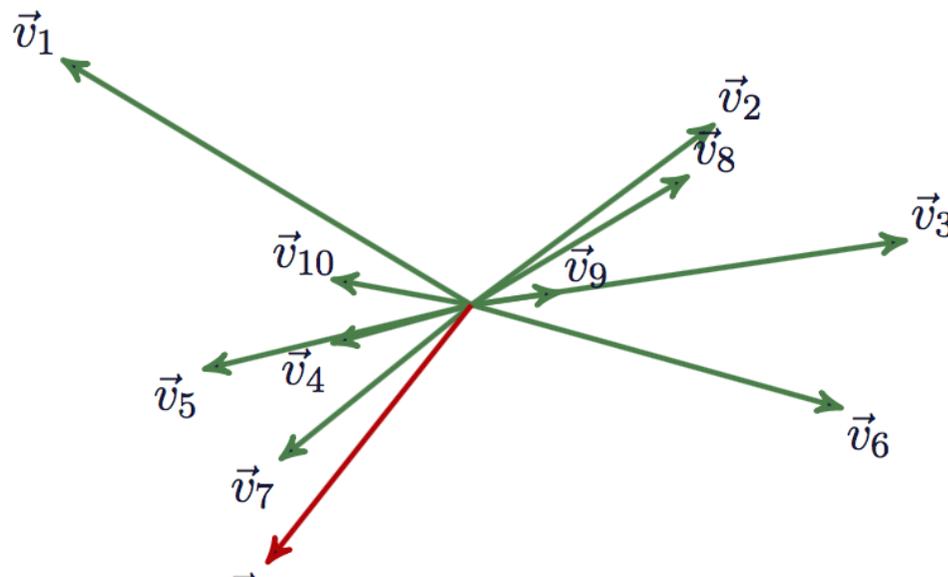
$$h(\vec{v}_8) = -2$$

$$h(\vec{v}_9) = -1$$

$$h(\vec{v}_{10}) = 0$$

LSH: Beispiel

Bucket -2	Bucket -1	Bucket 0
\vec{v}_2	\vec{v}_6	\vec{v}_1
\vec{v}_3	\vec{v}_9	\vec{v}_4
\vec{v}_8		\vec{v}_{10}
Bucket 1	Bucket 2	
\vec{v}_5		
\vec{v}_7		



Hashwerte:

$$h(\vec{v}_1) = 0$$

$$h(\vec{v}_2) = -2$$

$$h(\vec{v}_3) = -2$$

$$h(\vec{v}_4) = 0$$

$$h(\vec{v}_5) = 1$$

$$h(\vec{v}_6) = -1$$

$$h(\vec{v}_7) = 1$$

$$h(\vec{v}_8) = -2$$

$$h(\vec{v}_9) = -1$$

$$h(\vec{v}_{10}) = 0$$

Anfrage: Finde
Nachbarn von \vec{v}_8 .

Vorgehensweise:

Schaue in Bucket mit
Label -2 ($= h(\vec{v}_8)$).

Finde zwei Objekte.

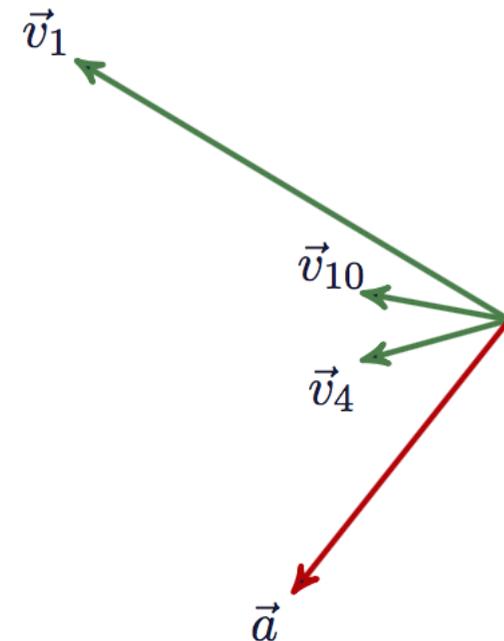
Evaluere diese.

LSH: Beispiel

Beobachtung: \vec{v}_{10} , \vec{v}_4 und \vec{v}_1 fallen in einen Bucket. Ok, \vec{v}_1 und \vec{v}_4 sind in der Tat ähnlich, aber \vec{v}_{10} passt nicht gut.

- **Wo ist das Problem?**

Vektor \vec{a} ist nicht in der Lage, gut zwischen diesen drei Vektoren zu unterscheiden.



Bucket 0
\vec{v}_1
\vec{v}_4
\vec{v}_{10}

LSH: Beispiel

Beobachtung: \vec{v}_{10} , \vec{v}_4 und \vec{v}_1 fallen in einen Bucket. Ok, \vec{v}_1 und \vec{v}_4 sind in der Tat ähnlich, aber \vec{v}_{10} passt nicht gut.

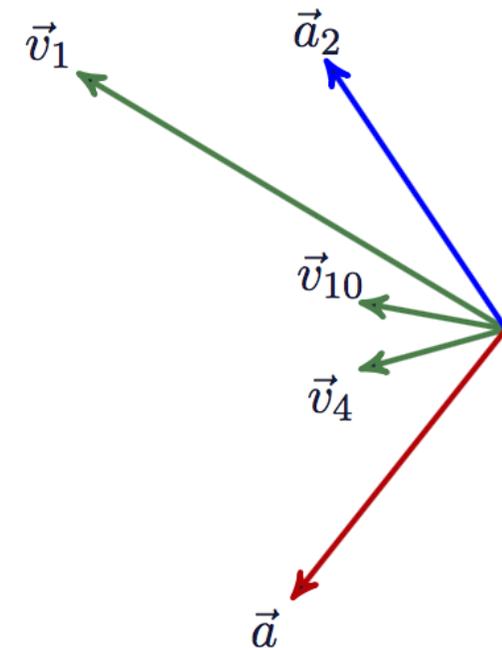
- **Wo ist das Problem?**

Vektor \vec{a} ist nicht in der Lage, gut zwischen diesen drei Vektoren zu unterscheiden.

- **Lösung:** Wir nehmen einen zweiten Vektor \vec{a}_2 hinzu.

$$h_{a_2}(\vec{v}) := \lfloor \frac{\vec{a}_2 \cdot \vec{v}}{4} \rfloor$$

Mit $h_{a_2}(\vec{v}_{10}) = 0$, $h_{a_2}(\vec{v}_4) = 0$, $h_{a_2}(\vec{v}_1) = 2$



LSH: Beispiel

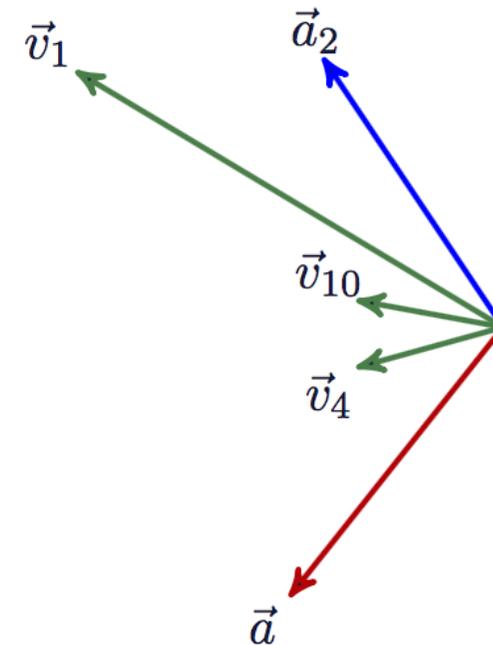
Beobachtung: \vec{v}_{10} , \vec{v}_4 und \vec{v}_1 fallen in einen Bucket. Ok, \vec{v}_1 und \vec{v}_4 sind in der Tat ähnlich, aber \vec{v}_{10} passt nicht gut.

- **Wo ist das Problem?**
Vektor \vec{a} ist nicht in der Lage, gut zwischen diesen drei Vektoren zu unterscheiden.
- **Lösung:** Wir nehmen einen zweiten Vektor \vec{a}_2 hinzu.

$$h_{a_2}(\vec{v}) := \lfloor \frac{\vec{a}_2 \cdot \vec{v}}{4} \rfloor$$

Mit $h_{a_2}(\vec{v}_{10}) = 0$, $h_{a_2}(\vec{v}_4) = 0$, $h_{a_2}(\vec{v}_1) = 2$

- Wie sehen dann die Labels der Buckets aus?
Konkatenation der einzelnen Hashwerte.
- Wir haben also die "Genauigkeit" der Buckets erhöht.
- Was ist der potentielle Nachteil?



Bucket (0,0)	
\vec{v}_4 \vec{v}_{10}	...

Eine bekannte LSH-Variante

Für jeden d -dimensionalen Punkt \vec{v} betrachten wir k unabhängige Hashfunktionen der Form:

$$h_{\vec{a},B}(\vec{v}) = \left\lfloor \frac{\vec{a} \cdot \vec{v} + B}{W} \right\rfloor$$

\vec{a} : d -dimensionaler Vektor, zufällig anhand Wahrscheinlichkeitsverteilung ausgewählt.

$W \in \mathbb{R}$, und $B \in [0, W]$. \vec{v} wird auf \vec{a} “projiziert” (Skalarprodukt).

“Beschriftung” der LSH Buckets

Mit k Hashfunktionen ist die Beschriftung des Buckets ein Integer-Vektor der Länge k :

$$g(\vec{v}) = (h_{\vec{a}_1, B_1}(\vec{v}), \dots, h_{\vec{a}_k, B_k}(\vec{v}))$$

Welchen Einfluss hat k auf die Suche?

Objekte den Hash-Buckets zuweisen

LSH Bucket "Labels"

Mit k Hashfunktionen ist ein Label ein Integer-Vektor der Länge k :

$$g(\vec{v}) = (h_{\vec{a}_1, B_1}(\vec{v}), \dots, h_{\vec{a}_k, B_k}(\vec{v}))$$

Objekt:



Anwendung von 4 Hashfunktionen:

$$h_1(\dots) = 0$$

$$h_2(\dots) = 1$$

$$h_3(\dots) = 1$$

$$h_4(\dots) = 1$$

Ergibt das Label: $g(\dots) = (0, 1, 1, 1)$

Suche nach ähnlichen Objekten: Beispiel

		Bucket-Label
Anfrage:		0,1,0,1
Potentieller Treffer:		0,1,1,1

Nicht gefunden!

- Um Trefferwahrscheinlichkeit zu erhöhen, werden Objekte nicht nur in einen Bucket gesteckt anhand von g indexiert, sondern anhand von verschiedenen g in mehrere Buckets in verschiedenen Hashtabellen.

Mehrere Hashtabellen

Benutze mehrere Hashtabellen, um die Wahrscheinlichkeit der Kollisionen zu vergrößern

	Hashtabelle 1	Hashtabelle 2
	0,1,0,1	1,0,0,1
	0,1,1,1	1,0,0,1
Ergebnis:	Kein Treffer	Treffer!

Anfrageverarbeitung: Suche der K nächsten Nachbarn

Gegeben ein Anfrage-Punkt q

- Berechne Bucket-Labels $g_i(q)$ für jede Hashtabelle i .
- Hole Objekte aus diesen Buckets
- Berechne echte Distanz und ordne Objekte entsprechend

Trotzdem: LSH ist eine approximative Technik

- Keine harte Garantie, dass alle Treffer (und nur diese) gefunden werden ...
- ... das ist oftmals aber akzeptabel.
- Es gibt theoretische Ansätze die Ergebnisgüte voraus zu sagen

Beobachtungen

Tuning

- Tradeoff zwischen Größe der Hash-Buckets und der Effektivität
- Mehrere Hashtabellen: Höhere Trefferwahrscheinlichkeit aber größerer Platzverbrauch
- Achtung: auch hier gilt wieder: Ab einem bestimmten Punkt kann ein Full-Scan günstiger sein (und dieser ist sogar noch exakt!)

Erweiterungen

- Schaue in mehrere Hash-Buckets *per* Hashtabelle (aka. multi-probe LSH)
- Auch: verteilte Implementierungen von LSH (z.B. in Peer-to-Peer-Systemen) oder in MapReduce

Zusammenfassung

Intervall-Skyline,
inkrementelle
Auswertungstechniken
Skizzen

Stromkonzept, relationale Anfragesprachen,
exakte Auswertung, QoS-Angaben

CM-, und FM-Skizzen,
Locality-Sensitive Hashing

Approximative Auswertung:
Bloom-Filter, Min-Hashing,
Stichproben

