Speed up Entity Resolution with Bit Arrays

Big Data Praktikum
SS 17

Moritz Engelmann
Maik Fröbe

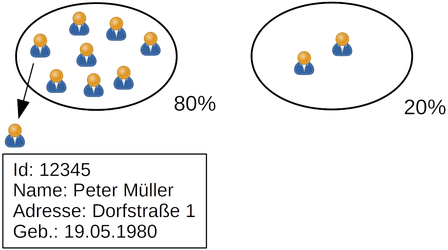
31.07.2017

1. Einführung
2. Ansätze zur Entity-Resolution
   - Trivialer Ansatz
   - Sortier-Ansatz
   - Bit-Array-Ansatz
   - Vergleich
3. Optimierungen des Bit-Array-Ansatzes
   - Bit-Array als Filter
   - Filtern in 2 Phasen
4. Rückblick + Ausblick

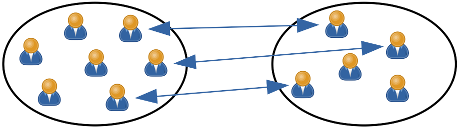
Einführung

Problembeschreibung

- Eingabe
  - 2 Mengen von Personen
  - Verhältnis 80:20



- Ziel
  - finden ähnlicher Personen in beiden Datensätzen

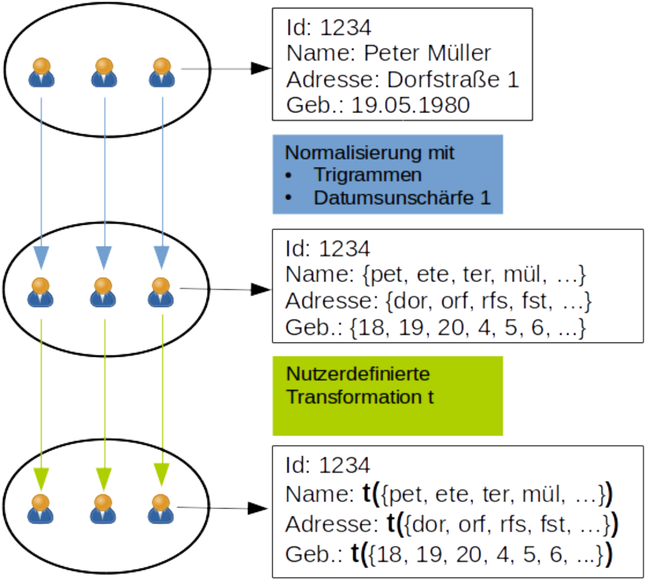


- Anforderungen
  - Bestimmung der Ähnlichkeit mit Jaccard-Index
  - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$

Framework zur Entity-Resolution

- vollständig Parametrisierbar
- Modular
- Start der Entity-Resolution mit:
  - Transformation: Person $\rightarrow V$
  - Ähnlichkeit: $V \times V \rightarrow [0, 1]$
  - n (Größe der n-Gramme)
  - Threshold
  - ...
- sequentieller Nested-Loop
  - Vollständige Berechnung der Ähnlichkeit für kartesisches Produkt

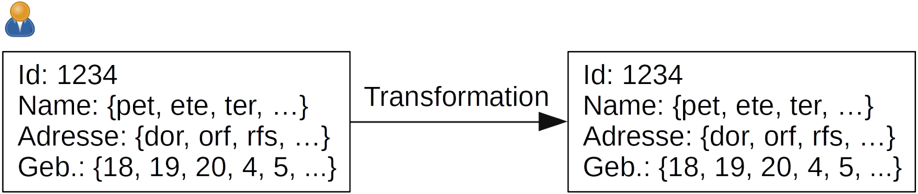
Importphase



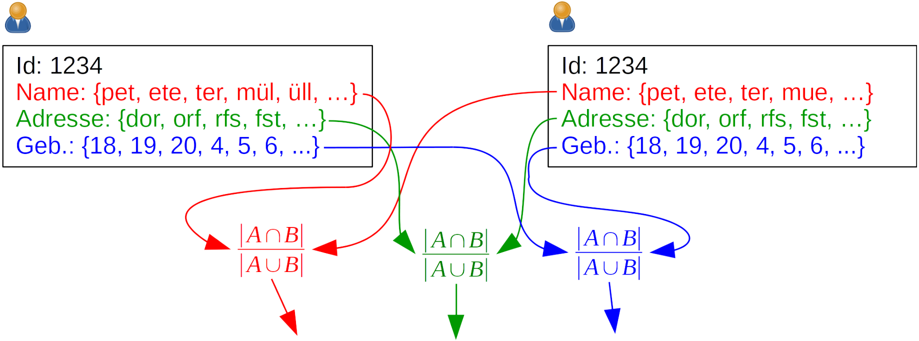
Ansätze zur Entity-Resolution

Trivialer Ansatz

Transformation:



Ähnlichkeitsfunktion:



$$\mathrm{sim} = avg(\mathrm{sim}_{Jacc}(Name), \mathrm{sim}_{Jacc}(Adresse), \mathrm{sim}_{Jacc}(Geb.))$$

Sortier-Ansatz

- Analog zu Sort-Merge-Verbund[1]
    - Während Import: Sortiere Mengen
    - Während ER: Berechne Kardinalität der Schnittmenge in $\mathcal{O}(n)$
        - Schritthaltende Traversierung der sortierten Mengen

[1] Siehe Vorlesung Implementierung von Datenbanksystemen

Bit-Array-Ansatz

Einschub: Bloom-Filter

pet ete ter

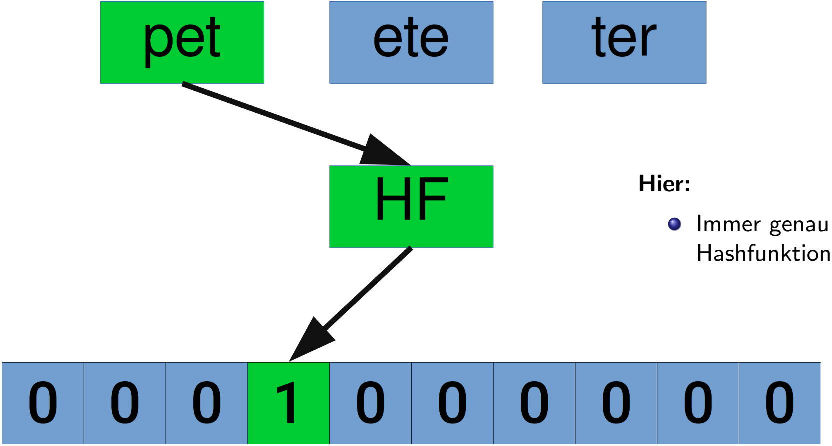
HF

Hier:

- Immer genau eine Hashfunktion!

0	0	0	0	0	0	0	0	0	0

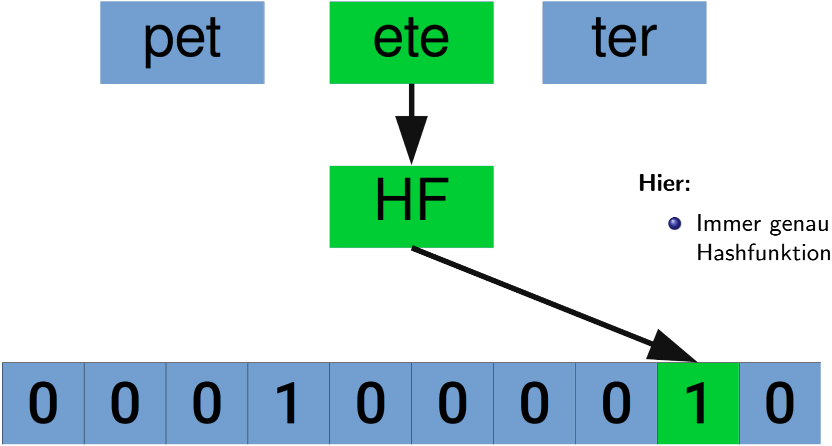
Einschub: Bloom-Filter



Hier:

- Immer genau eine Hashfunktion!

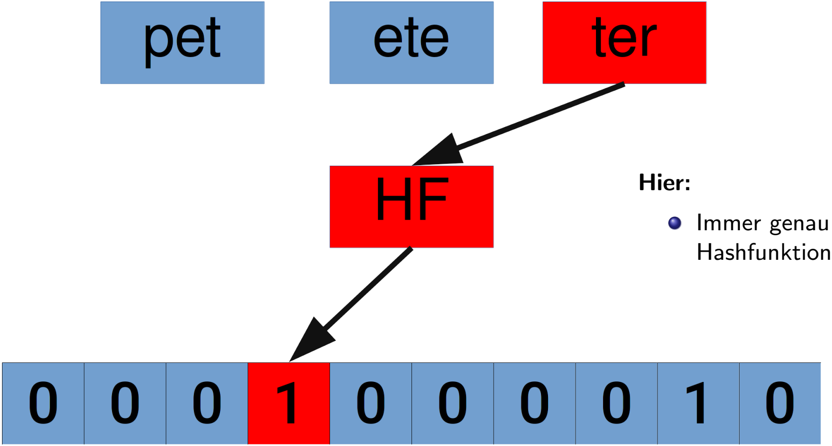
Einschub: Bloom-Filter



Hier:

- Immer genau eine Hashfunktion!

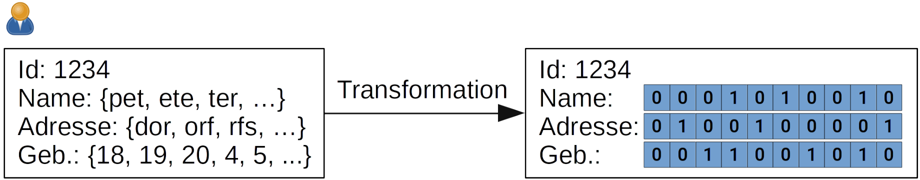
Einschub: Bloom-Filter



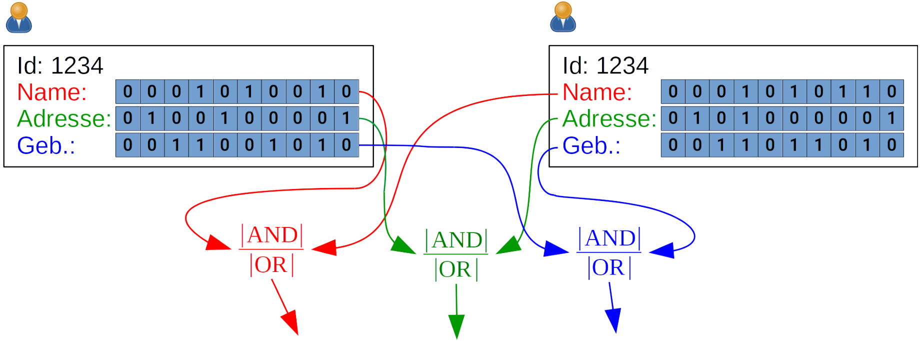
Hier:

- Immer genau eine Hashfunktion!

Transformation:



Ähnlichkeitsfunktion:

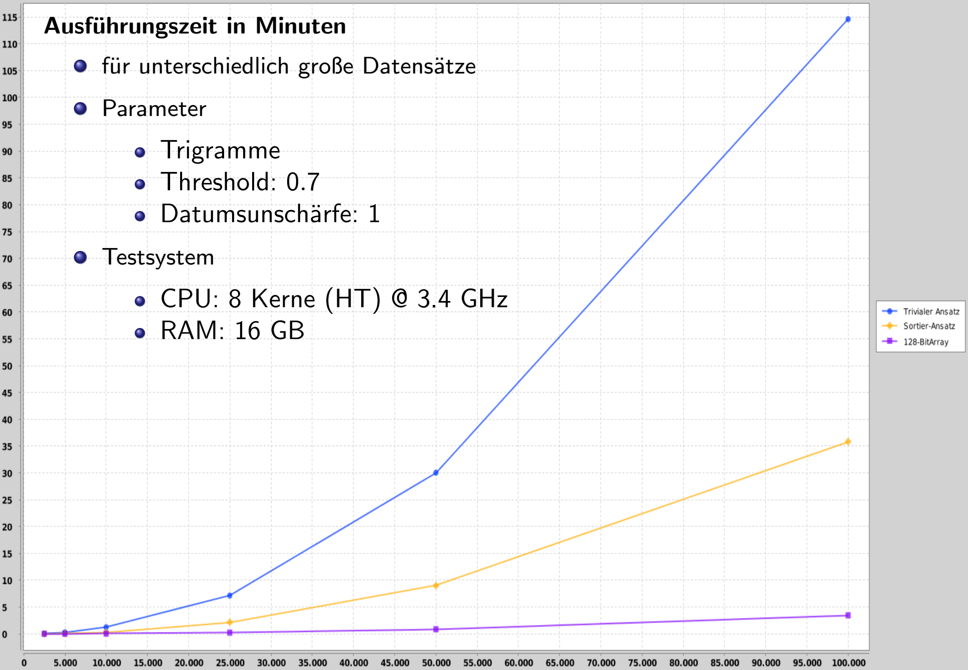


$$sim = avg(sim_{Jacc}(Name), sim_{Jacc}(Adresse), sim_{Jacc}(Geb.))$$

Vergleich

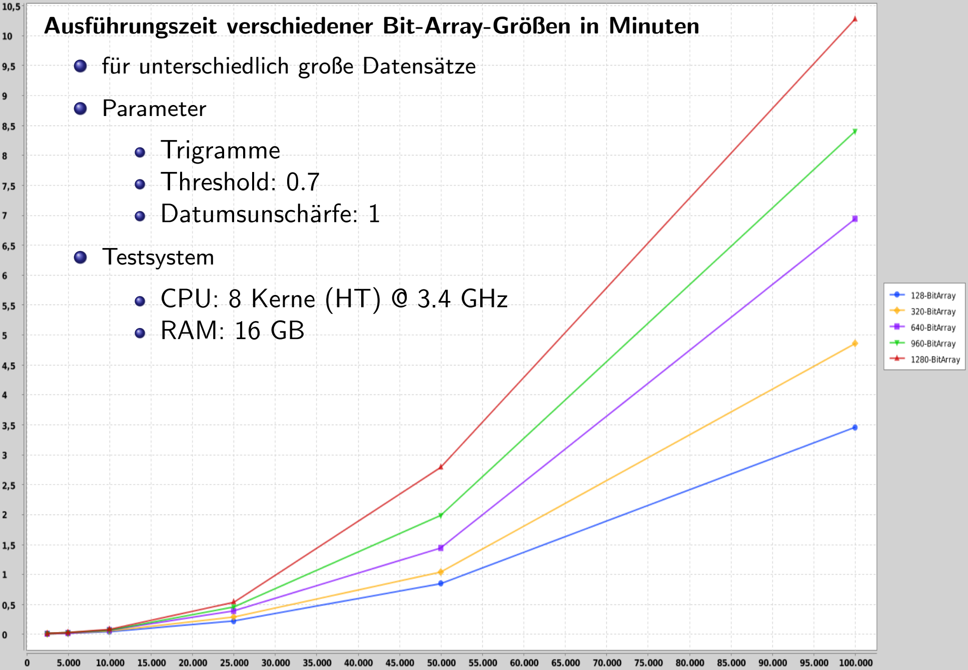
Ausführungszeit in Minuten

- für unterschiedlich große Datensätze
- Parameter
  - Trigramme
  - Threshold: 0.7
  - Datumsunschärfe: 1
- Testsystem
  - CPU: 8 Kerne (HT) @ 3.4 GHz
  - RAM: 16 GB

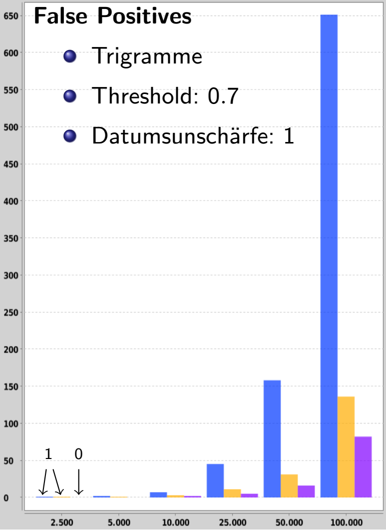


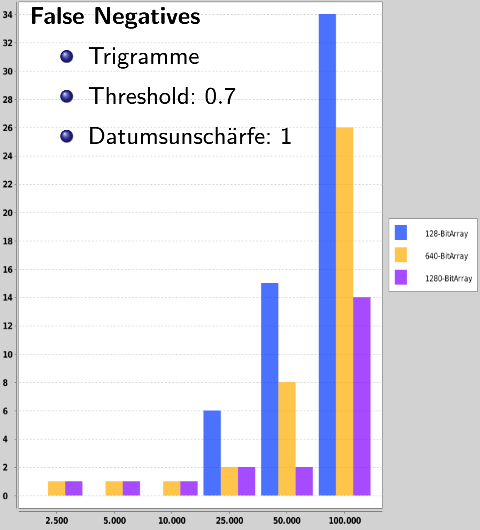
Ausführungszeit verschiedener Bit-Array-Größen in Minuten

- für unterschiedlich große Datensätze
- Parameter
  - Trigramme
  - Threshold: 0.7
  - Datumsunschärfe: 1
- Testsystem
  - CPU: 8 Kerne (HT) @ 3.4 GHz
  - RAM: 16 GB

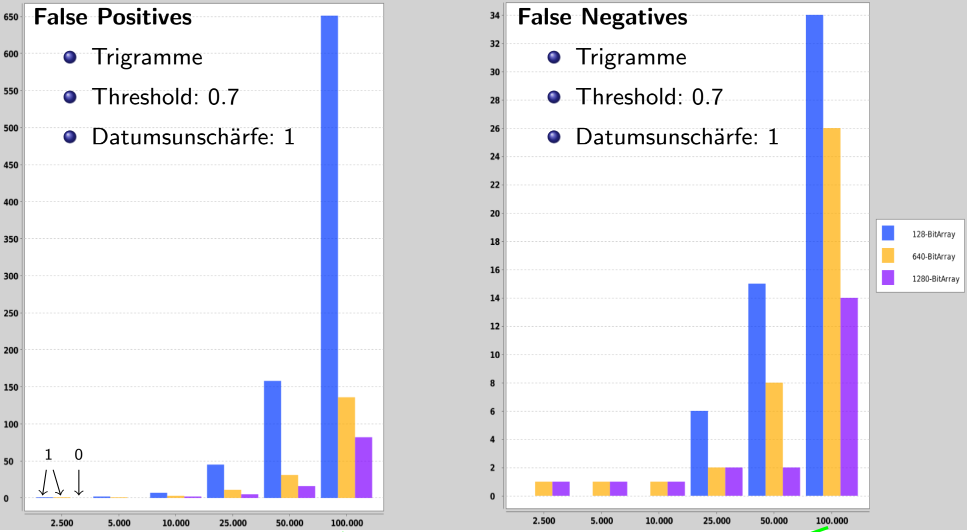


Absolute Werte:





Absolute Werte:



	128-Bit	640-Bit	1280-Bit
Precision	0.955	0.990	0.994
Recall	0.998	0.998	0.999

Optimierungen des Bit-Array-Ansatzes

Bit-Array als Filter

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A| + |B| - |A \cup B|}{|A \cup B|} \leq \frac{|A| + |B| - |A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A| + |B| - |A \cup B|}{|A \cup B|} \leq \frac{|A| + |B| - |A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

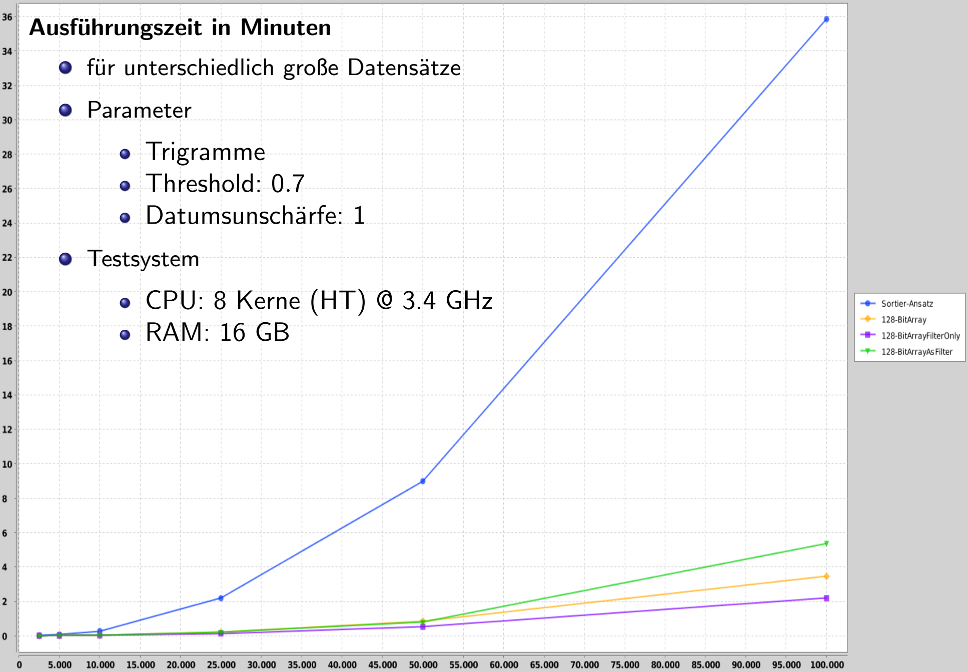
- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A|+|B|-|A \cup B|}{|A \cup B|} \leq \frac{|A|+|B|-|A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

- Zwischenschritt:
  - Obere Schranke des Jaccard-Index mit Bit-Arrays effizient bestimmen
- Idee:
  - A, B - Mengen
  - $A_F = bloom(A)$, $B_F = bloom(B)$ - Bit-Arrays der Mengen
  - Es gilt:
    - $|A_F| \leq |A|$
    - $|A_F \vee B_F| \leq |A \cup B|$
    - $jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A| + |B| - |A \cup B|}{|A \cup B|} \leq \frac{|A| + |B| - |A_F \vee B_F|}{|A_F \vee B_F|}$
- Vorgehen:
  - Schätze Jaccard-Index
  - Größer Threshold? $\Rightarrow$ berechne Jaccard-Index

Ausführungszeit in Minuten

- für unterschiedlich große Datensätze
- Parameter
  - Trigramme
  - Threshold: 0.7
  - Datumsunschärfe: 1
- Testsystem
  - CPU: 8 Kerne (HT) @ 3.4 GHz
  - RAM: 16 GB



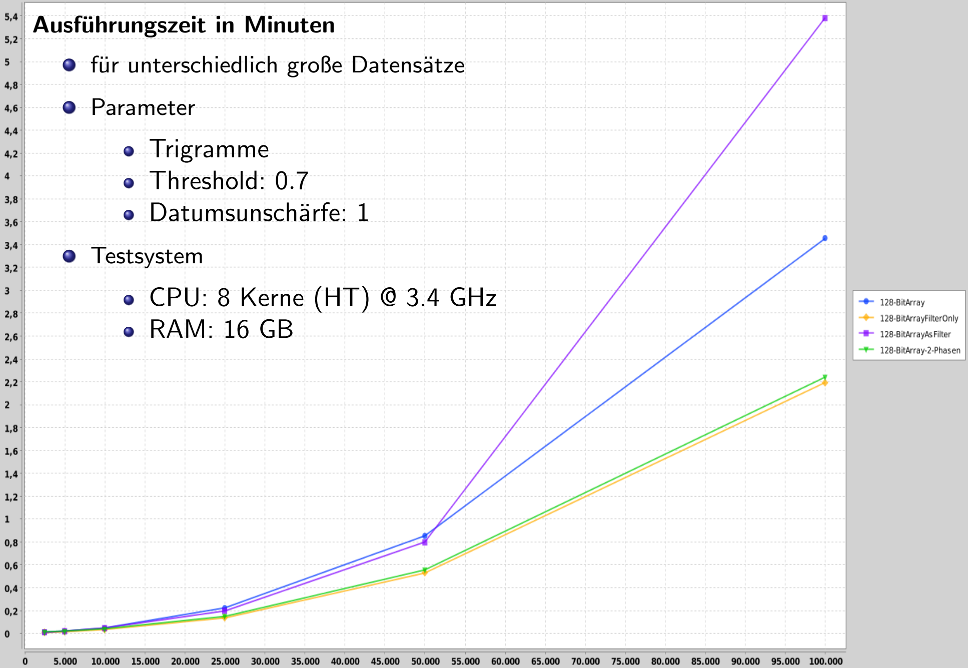
Filtern in 2 Phasen

Filtern in 2 Phasen

- Phase 1
    - ER mit Bit-Array-Filter-Only
- Phase 2
    - Eingabe: Id-Paare der Kandidaten aus Phase 1
    - Import (Normalisierung + Transformation) für Sortier-Ansatz aus
    - ER nur für Kandidaten-Paare

Ausführungszeit in Minuten

- für unterschiedlich große Datensätze
- Parameter
  - Trigramme
  - Threshold: 0.7
  - Datumsunschärfe: 1
- Testsystem
  - CPU: 8 Kerne (HT) @ 3.4 GHz
  - RAM: 16 GB



Rückblick + Ausblick

- Erfahrungen:
  - Abstraktion + große Datenstrukturen sind teuer
- Tipps für unbekannte Datensätze:
  - Untersuche Datensatz mit „Phase 1“
  - Schrittweise Anpassung der Parameter
    - Ziel: sehr hohe Selektivität trotz kleinem Bit-Array
  - Abschließend: ER mit „Phase 1 + 2“
- mögliche Nächste Schritte:
  - Parallelisierung
  - Ein Partitionierter Bit-Array
  - Vergleich mit weiteren ER-Ansätzen
  - Integration unterschiedlicher ER-Ansätze als „Phase 2“