Optimierung der Exact-Match-Anfrage eines Lokal Sensitiven Hashverfahrens

Sarah Heckel
Fakultät für Mathematik
Otto-von-Guericke Universität
Universitätsplatz 2
39106 Magdeburg
sarah.heckel@st.ovgu.de

Abstract: Hochdimensionale Indexverfahren sind wichtig um einen schnellen Zugriff auf Multimediadaten zu gewährleisten. Eine Klasse dieser Verfahren ist das Lokal Sensitive Hashen (LSH). Beim LSH können sehr unterschiedlich ausgelastet Bereiche entstehen. Um die Exakt-Match-Anfrage beim Permutationsansatz, einer Variante des LSHs, effizient bearbeiten zu können, ist eine gleichmäßige Raumaufteilung von Vorteil. Dazu ist die Wahl der Protoypen von großer Bedeutung.
Im Folgenden wird ein mathematisches Optimierungsproblem aufgestellt, welches die Prototypen bestimmt. Die Idee dabei ist Kugeln mit minimalem gleichem Radius um die Prototypen zu legen, sodass jeder Datenpunkt in mindestens einer Kugel enthalten ist.
Werden optimierte Prototypen für die permutationsbasierte Variante des LSHs gewählt, so ist die Abweichung der Raumaufteilung gegenüber der Aufteilung bei zufällig gewählten Prototypen stabiler.

1 Einleitung

Hochdimensionale Indexstrukturen werden benötigt, um Daten mit vielen Attributen verwalten zu können. Dabei wird der Raum in mehrere kleinere Bereiche, im Folgenden Buckets genannt, aufgeteilt, damit Datenpunkte schneller gefunden werden können. Es gibt verschiedene Indexstrukturen um hochdimensionale Daten gut verwalten zu können. Dabei wird zwischen exakten und approximativen Indexverfahren unterschieden [AI08]. Exakte Verfahren geben bei Nachbarschaftsanfragen nach den k nächsten Nachbarn (kNN) genau die k nächsten Datenpunkte eines Datenpunktes aus, während die approximativen Verfahren zu einem Datenpunkt k ähnliche Elemente ausgeben. Zu den exakten Verfahren gehören zum Beispiel verschiedene Baumverfahren, wie der R-Baum und X-Baum [GG97]. Die Klasse der LSH-Verfahren gehört zu den approximierenden Verfahren [GG97]. Das LSH ist ein hochdimensionales Hashverfahren. Es wurde vorgestellt, um die kNN-Suche gegenüber anderen Hashverfahren zu verbessern [IM98]. Im Gegensatz zu anderen Hashverfahren, werden ähnliche Datenpunkte auf dasselbe Bucket abgebildet, statt sie über die Buckets zu streuen. Um die kNN eines Datenpunktes zu finden werden nur noch die angrenzenden Buckets durchsucht.

Ein erster Schritt für eine Verbesserung dieses Verfahrens stellt die Optimierung der Exact-Match-Anfrage dar. Die Erkenntnisse, die sich daraus gewinnen lassen, können später für die Anfrage der kNN genutzt werden.
Im Folgenden geht es um die permutationsbasierte Variante des LSHs. Diese wird in Kapitel 2 näher erläutert. Dabei wird der Raum anhand von einigen Datenpunkten, den sogenannten Prototypen, aufgeteilt. Untersucht man die dadurch entstehende Raumaufteilung, so ist festzustellen, dass sehr ungleichmäßige Aufteilungen entstehen können. So erhält man beispielsweise einige Bereiche, die im Vergleich zu anderen sehr viele Elemente enthalten. Um diese dann sequenziell zu durchsuchen, wird mehr Zeit benötigt, als wenn schwach belegte Buckets durchsucht werden.
Für die verschiedenen Anfragetypen Exact-Match-Anfrage, kNN-Suche und Bereichsanfrage sind unterschiedliche Raumaufteilungen wünschenswert. So soll bei der Exact-Match-Anfrage eine möglichst gleichmäßige Raumaufteilung erreicht werden, so dass in jedem Bucket annähernd die selbe Anzahl an Elementen enthalten ist. Es müssen dann im Schnitt die gleiche Anzahl an Elementen sequenziell durchsucht werden. Um eine gute Raumaufteilung zu erzielen, sollen Prototypen so gewählt werden, dass die Datenpunkte möglichst gleichmäßig auf die Buckets verteilt werden. Genau um dieses Problem geht es in dieser Arbeit.

2 Lokal Sensitives Hashen

Viele Hashverfahren streuen die Daten auf verschiedene Bereiche und zerstören so die Nachbarschaftsbeziehungen. Diese Verfahren sind daher nicht geeignet für die Suche nach den nächsten Nachbarn eines Datenpunktes. Bei der exakten Suche nach den kNN müssen daher alle Bereiche durchsucht werden. Im Gegensatz dazu bildet das LSH ähnliche Datenpunkte auf dasselbe Bucket ab.

2.1 Definition

Beim LSH gibt es statt nur einer Hashfunktion mehrere Hashtabellen, für die jeweils verschiedene Hashfunktionen gelten [IM98]. Dabei kommen die Hashfunktionen aus einer Familie $\mathcal{H}$ von Funktionen. Jede Funktion h aus $\mathcal{H}$ ist dabei (P1, P2, r, cr)-sensitiv [IM98], das heißt für je zwei Datenpunkte $p, q \in \mathcal{R}^d$ und $P1 > P2$ gilt:

1. $wenn \; \|p - q\| < r, \; dann \; Pr[h(p) = h(q)] > P1$
2. $wenn \; \|p - q\| > cr, \; dann \; Pr[h(p) = h(q)] < P2.$

Das heißt, wenn der Abstand von p und q kleiner als r ist werden p und q mit einer Wahrscheinlichkeit größer als $P1$ in dasselbe Bucket abgebildet. Dagegen werden p und q mit einer Wahrscheinlichkeit kleiner als $P2$ in dasselbe Bucket abgebildet, wenn ihr Abstand größer als cr ist. Diese Definition sorgt dafür, dass ähnliche Elemente mit hoher Wahrscheinlichkeit in dasselbe Bucket abgebildet werden.

2.2 Permutationsansatz

Chavez et al. stellen in [CFN08] den Permutationsansatz vor, der die Definition einer Funktionenfamilie $\mathcal{H}$ umgeht. Dazu werden zufällig l, so genannte Prototypen, aus der vorhandenen Datenmenge $\mathcal{D}$ gewählt und der Abstand von $D \in \mathcal{D}$ zu jedem der l Prototypen berechnet. Anhand dieser werden die Prototypen aufsteigend sortiert. Die so entstehende Reihenfolge der Prototypen gibt den Hashwert von D an.

Beispiel:
$dim = 2,\ \#Prototypen = 3$
$Prototypen:\ P1 = (1, 2),$
$P2 = (20, 200),\ P3 = (150, 60)$
$gesuchter\ Datenpunkt:$
$D = (15, 30)$
$\|D - P1\|_2 \approx 31,3$
$\|D - P2\|_2 \approx 170,1$
$\|D - P3\|_2 \approx 138,3$
$\Rightarrow Aufsteigend\ sortiert\ nach\ den$
$Abständen\ ergibt\ sich\ für\ den$
$Hashwert\ von\ D\ :\ P1P3P2$

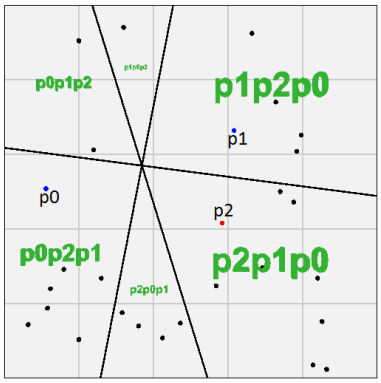

Abbildung 1: links: Ein Beispiel für die Berechnung des Hashwertes eines Datenpunktes mit drei Prototypen. rechts: Raumaufteilung mit dem Permutationsansatz im zweidimensionalen Raum mit drei Prototypen [B12].

Die Raumaufteilung entsteht dann dadurch, dass man jeweils zwischen zwei Prototypen eine imaginäre Strecke legt und orthogonal zu deren Mittelpunkt eine Hyperebene zieht. Dieses wird mit allen Paaren von Prototypen wiederholt. Die Buckets werden dann mit den entsprechenden Permutationen beschriftet (Abbildung 1 rechts).
Um eine effiziente Raumaufteilung zu erhalten, ist die Wahl der Prototypen, mit denen der Raum aufgeteilt wird, wichtig. Um Cluster innerhalb der Datenmenge gut zu unterteilen hat Broneske in [B12] herausgearbeitet, dass die Prototypen um das Cluster herum gelegt werden sollten.

2.3 Exact-Match-Anfrage beim Lokal Sensitiven Hashen

Beim LSH können sehr unterschiedlich ausgelastete Bereiche entstehen. Daher sollen die Prototypen so gesetzt werden, dass eine möglichst gleichmäßige Auslastung entsteht.
Damit die Exact-Match-Anfrage von einem Datenpunkt D effizient verarbeitet werden kann, wird der zugehörige Hashwert berechnet. Anschließend wird das dazugehörige Bucket mittels sequentieller Suche durchsucht, um festzustellen, ob D in der Datenmenge $\mathcal{D}$ enthalten ist.
Daher ist es von Vorteil, wenn jedes Bucket die selbe Anzahl an Elementen enthält. Sind

zum Beispiel in einem Bucket nur zwei Datenpunkte und in einem anderen 50, ist die Suche im zweiten Bucket im Vergleich zum ersten rechenintensiver. Deswegen soll eine ungleichmäßige Aufteilung der Datenpunkte auf die Buckets vermieden werden. Daher stellt sich die Frage, wie die l Prototypen zu wählen sind.
Betrachtet man die zufällige Wahl der Prototypen, wie sie in [CFN08] vorgestellt wird, so ist festzustellen, dass sehr unterschiedliche Raumaufteilungen entstehen können, die von unterschiedlicher Güte sind.
Deshalb wird im Folgenden ein mathematisches Optimierungsproblem aufgestellt, welches möglichst optimale Prototypen aus der Datenmenge $\mathcal{D}$ auswählt.

3 Optimierung der Exact-Match-Anfrage

Damit die Exact-Match-Anfrage effizient bearbeitet werden kann, ist es vorteilhaft, wenn die Buckets annähernd gleich viele Elemente enthalten. In diesem Fall lässt sich das Bucket in dem ein Datenpunkt D liegt leicht berechnen und das dazugehörige Bucket kann dann sequenziell durchsucht werden.

3.1 Verschiedene Optimierungsprobleme in der Übersicht

Es gibt verschiedene Möglichkeiten wie das Problem bearbeitet werden kann. Zum einen kann das Problem als Modell intuitiv geometrisch (1) formuliert werden, indem passende Beschreibungen für die entstehenden Polytope gefunden werden. Die Polytope entstehen dadurch, dass man zwischen je zwei Prototypen in der Mitte orthogonal eine Hyperebene legt. Betrachtet man nun die Schnitte von mehreren solcher Hyperebenen, entstehen Polytope, die sich im Verlauf des Optimierungsprozesses stetig ändern können. Weiterhin muss sichergestellt werden, dass in jedem dieser Polytope annähernd gleich viele Datenpunkte enthalten sind. Die Beschreibung der Polytope ist kompliziert, da diese sich im Verlauf der Optimierung ändern können. Daher ist dieses Modell rechnerisch sehr aufwändig.
Eine weitere Möglichkeit besteht darin, das Volumen der Polytope (2) zu optimieren. Dieses macht Sinn, wenn die Daten über denen optimiert wird gleichverteilt sind. Es soll erreicht werden, dass jedes Polytop annähernd dasselbe Volumen hat. Allerdings ist es mit der selben Begründung wie oben rechnerisch sehr aufwändig.
Eine weitere Idee besteht darin, dass eigentliche Problem approximativ zu formulieren (3). Somit ist das folgende Optimierungsproblem losgelöst von den Beschreibungen der Polytope. Dadurch ist es leichter zu verarbeiten.
Man legt jeweils gleich große Kugeln um die l Prototypen, sodass deren Mittelpunkt einer der l Prototypen ist. Nun besteht die Aufgabe darin, den Radius der Kugeln so zu minimieren, dass jeder beliebige andere Datenpunkt $D \in \mathcal{D}$ in mindestens einer dieser l Kugeln enthalten ist. In Abbildung 2 ist das Optimierungsproblem graphisch dargestellt. Die Kreuze stellen die Prototypen dar, um welche Kugeln gelegt werden sollen.

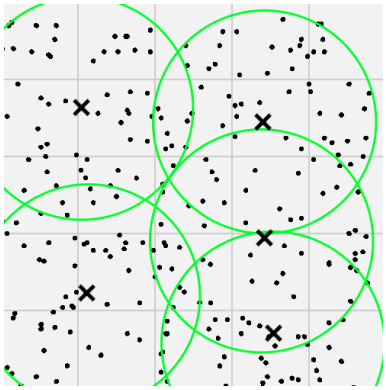

Abbildung 2: Optimierungsproblem (3) mit $l = 5$.

3.2 Mathematische Beschreibung des Optimierungsproblems

Es wird eine Datenmenge $\mathcal{D} \subseteq \mathcal{R}^d$ betrachtet, die n Datenpunkte enthält. Mathematisch sieht das Problem wie folgt aus:

$$\min_{r,x,y} r$$

s.t.

$$\sum_{i=1}^{n} x_i = l \tag{1}$$

$$\sum_{i=1}^{n} y_{i,j} \geq 1 \quad \forall j \in \mathcal{I} \tag{2}$$

$$y_{i,j}\sqrt{\| D_i - D_j \|_2} \leq r \quad \forall i,j \in \mathcal{I};\ D_i, D_j \in \mathcal{D} \tag{3}$$

$$y_{i,j} \leq x_i \tag{4}$$

$$r \geq 0 \tag{5}$$

$$x_i \in \{0,1\} \tag{6}$$

$$y_{i,j} \in \{0,1\} \tag{7}$$

Dabei ist x ein Entscheidungsvektor, für den $x_i = 1$ gilt, wenn der Datenpunkt D_i zu den l Prototypen gehört, ansonsten gilt $x_i = 0$. Die Matrix $y \in \{0,1\}^{n \times n}$ ist eine Entscheidungsmatrix, die angibt, ob ein Datenpunkt D_j von einer Kugel um D_i umgeben wird. Die Parameter $i,\ j$ sind aus der Menge $\mathcal{I} = \{1, 2, ..., n\}$. Zusammenfassend gilt:

$$x_i = \begin{cases} 1, & \textit{wenn } D_i \textit{ ein Prototyp ist} \\ 0, & \textit{sonst} \end{cases}$$

$$y_{i,j} = \begin{cases} 1, & \textit{wenn } D_j \textit{ in Kugel um } D_i \textit{ enthalten ist} \\ 0, & \textit{sonst} \end{cases} .$$

Die Nebenbedingungen des Optimierungsproblems werden im Folgenden erläutert: (1) gibt an, dass genau l Prototypen gesucht werden. In (2) wird beschrieben, dass jeder Datenpunkt in mindestens einer Kugel um einen Prototypen enthalten sein muss. Die Kugeldefinition mit positiven Radius r in (3) kommt genau dann zum Tragen, wenn $y_{i,j}$ eins ist. Dagegen sichert (4), dass $y_{i,j}$ nicht eins sein darf, wenn D_i nicht zu den Prototypen gehört, das heißt D_i ist dann kein Mittelpunkt einer Kugel.
Das oben genannte Optimierungsproblem ist linear und hängt nicht von der Dimension der Datenpunkte in $\mathcal{D}$ ab. Die verwendete Euklidische Norm lässt sich durch andere Normen ersetzen, zum Beispiel durch die Summennorm oder die Maximumsnorm. Man beachte, dass $y_{i,j}$ für große Datenmengen sehr groß wird.

Zur Lösung des Problems wird AMPL („A Mathematical Programming Language“) benutzt. AMPL ist eine mathematische Modellierungssprache, mit der ein mathematisches Optimierungsproblem in abstrakter Form formuliert werden kann [FGK03]. Dabei übersetzt AMPL das Optimierungsproblem für Optimierungsalgorithmen, die dieses dann lösen. Man beachte, dass passende Optimierungsalgorithmen gewählt werden müssen. Bei dem oben genannten Optimierungsproblem handelt es sich um ein lineares Programm, dafür ausgewählte Solver sind beispielsweise CPLEX und Gurobi.

4 Evaluation

In diesem Abschnitt wird die Auswertung des in 3.2 genannten Optimierungsproblems betrachtet. Dazu wird eine reale Datenmenge aus [AA96] verwendet. In einem Vergleich der Raumaufteilung mit optimierten und zufällig gewählten Prototypen wird die Relevanz des Optimierungsproblems dargestellt.

4.1 Setup und Durchführung

Die Testdaten bestehen aus 10.992 Datenpunkten mit jeweils 16 Dimensionen. Die Einträge der Daten sind ganzzahlige Werte zwischen 0 und 100. Aufgrund von Platzproblemen wird exemplarisch $l = 5$ gewählt, es sollen also 5 Prototypen aus den Datenpunkten so gewählt werden, dass der Raum möglichst optimal aufgeteilt wird. Als Optimierungssolver wurde CPLEX gewählt.
Bei der Berechnung des Optimierungsproblems hat sich herausgestellt, dass die Berechnungszeit von einer großen Datenmenge sehr hoch ist. So benötigt man für die Optimierung über einer Datenmenge mit 1000 Elementen mehrere Tage. Aufgrund der Berechnungszeit werden approximative Lösungen betrachtet, bei denen jeweils nur ein Teil der Daten zur Lösung des Problems gewählt wird. Daraus ergibt sich eine weitere Fragestellung, die später noch untersucht werden muss. Wie viele Daten müssen aus der Datenmenge betrachtet werden, damit keine großen Änderungen bezüglich der Abweichung auftreten. Exemplarisch wird die Auswirkung auf die Raumaufteilung durch den Permutationsansatz beim LSH betrachtet. Hierzu werden 100, 200 und 300 Daten aus der vorliegenden

Datenmenge ausgewählt. Auf diesen kleineren Datenmengen wird, mittels oben genanntem Optimierungsproblems, optimiert. Anschließend werden die ermittelten Prototypen auf die gesamte Datenmenge angewandt.

4.2 Ergebnisse

Als Erstes wird die Auswirkung der Optimierung der Prototypen auf den kleinen Datenmengen, mit 100, 200 und 300 Datenpunkten aus der gesamten Datenmenge, untersucht. Werden die Ergebnisse der Optimierung auf der Datenmenge mit 100 Elementen mit Ergebnissen vergleichen, die durch zufällige Wahl von 5 Prototypen aus der Datenmenge auswählt werden, ist zu erkennen, dass unterschiedlich viele Buckets entstehen können. Dies liegt daran, wie die Hyperebenen im Raum zueinander liegen. In diesem Beispiel entstehen 35, 30 und 24 Buckets für die Raumaufteilung in Abbildung 3. Der Durchschnitt gibt an, wie viele Datenpunkte in einem Bucket liegen müssen, damit die Aufteilung optimal ist. Die Balken des Diagramms stehen dafür, wie viele Buckets es in der entsprechenden Raumaufteilung gibt.
Bei der auf den optimierten Prototypen basierenden Raumaufteilung (a) in Abbildung 3 ist zu erkennen, dass die Mehrheit der Buckets dicht beim Durchschnitt liegt. Das bedeutet, dass die Mehrheit der Buckets annähernd optimal viele Elemente beinhalten. Vergleicht man dazu die Raumaufteilung (b), ist festzustellen, dass die Mehrheit der Buckets weiter vom Durchschnitt entfernt sind. Hierbei ist zu beachten, dass beide Verteilungen die gleiche Abweichung haben. Als Abweichung wird

$$\sum_{i=1}^{Anzahl\ der\ Buckets} |Anzahl\ Elemente\ in\ Bucket\ i - Durchschnitt|$$

bezeichnet.

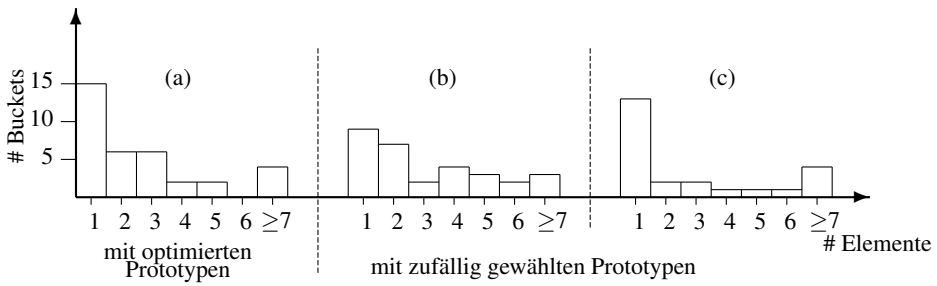

Abbildung 3: Verteilung der Daten auf die entstehenden Buckets für 100 Datenpunkte. Durchschnitt der optimierten Raumaufteilung (a): 2; Durchschnitt der Raumaufteilungen mit zufälligen Prototypen (b) und (c): 3 und 4.

Allerdings ist die optimierte Raumaufteilung besser verteilt, da es weniger Ausreißer nach oben hin gibt. Es gibt also weniger Buckets die viele Elemente enthalten. Die Raumaufteilung (c) in Abbildung 3 weist eine noch größere Streuung zu den Rändern, damit sind die

Buckets mit minimal beziehungsweise maximal vielen Datenpunkten gemeint, auf. Hier besitzt die Mehrheit der Buckets ein oder mehr als sieben Elemente, während das Optimum bei vier liegt. Ähnlich verhält sich die Raumaufteilung, wenn über einer Menge von 200 beziehungsweise 300 Daten optimiert wird.

Im Folgenden werden die Ergebnisse der Raumaufteilung der gesamten Datenmenge mit optimierten (Abbildung 5) und zufälligen Prototypen (Abbildung 6) betrachtet. Dazu werden die Prototypen, die zu den optimierten Raumaufteilungen in Abbildung 3 und die zu den optimierten Raumaufteilungen auf 200 und 300 Daten führen, gewählt. Wird über einer immer größer werdenden Menge optimiert, ist zu erkennen, dass die Schwankungen um das Optimum geringer werden. Je größer die Datenmenge über der optimiert wird ist, desto kleiner sollte die Abweichung werden. Weiterhin kommt es darauf an, welche Elemente man aus der gesamten Datenmenge wählt, um über diesen zu optimieren. In diesem Beispiel werden die ersten 100, 200 und 300 Elemente aus der gesamten Datenmenge gewählt.
Betrachtet man nun die zufälligen Ergebnisse in Abbildung 6, so ist festzustellen, dass die durch zufällige Wahl der Prototypen entstehenden Raumaufteilungen große Schwankungen aufweisen. So können durchaus bessere Lösungen als die in Abbildung 5 dargestellten Raumaufteilungen gefunden werden, jedoch entstehen auch schlechtere. Durch die optimierten Prototypen kann man die Schwankungen durch zufällige Wahl der Prototypen verringern.

In Abbildung 4 sind die Abweichungen vom Optimum graphisch dargestellt. Es ist gut zu erkennen, dass die Abweichung bei zufälliger Wahl der Prototypen stark variieren kann. Weiterhin wird deutlich, dass die Abweichungen geringer werden, wenn optimierte Prototypen gewählt werden, die über einer größer werdenden Teilmenge der gesamten Daten optimiert wurden.

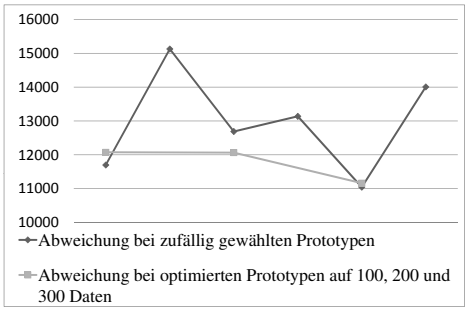

Abbildung 4: Vergleich der Abweichung mit zufällig gewählten und optimierten Prototypen.

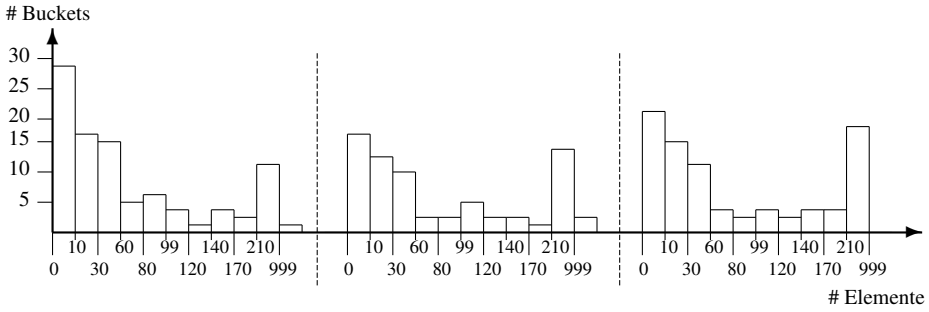

Abbildung 5: Verteilung der gesamten Datenmenge auf die entstehenden Buckets bei Anwendung der optimierten Prototypen auf 100, 200 und 300 Datenpunkten. Durchschnitt der jeweiligen Datenverteilungen auf den Raum: 109, 150 und 124.

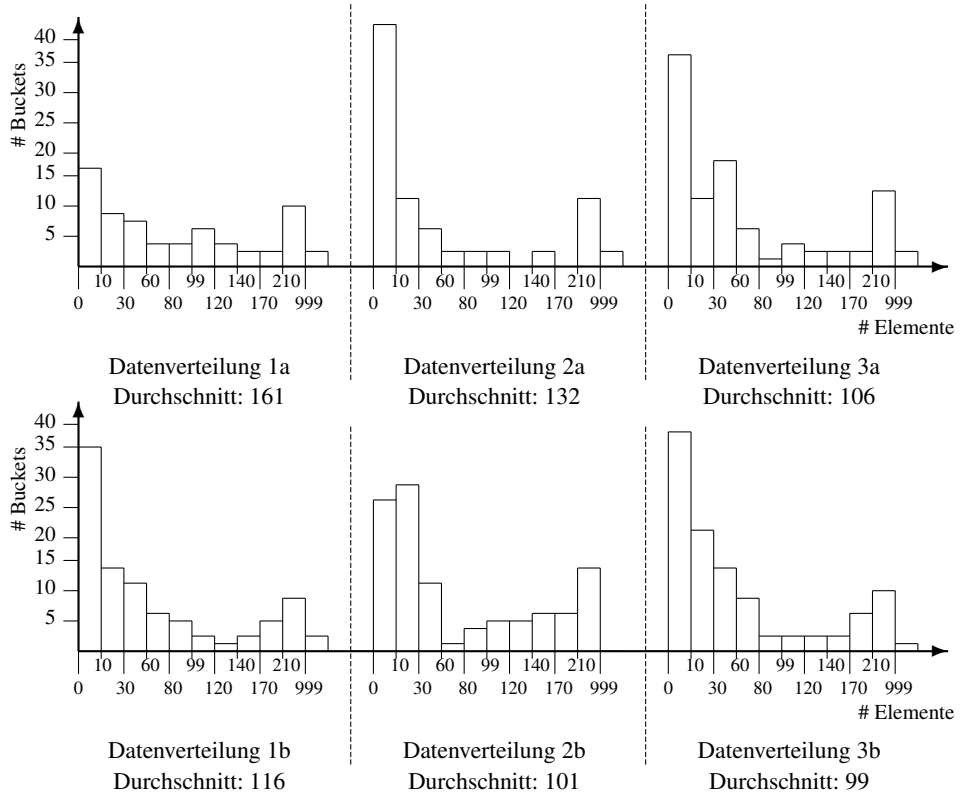

Abbildung 6: Verteilung der gesamten Daten auf die entstehenden Buckets für zufällig gewählte Prototypen.

5 Fazit und Ausblick

Es wurde ein Optimierungsproblem vorgestellt, welches die Wahl der Prototypen der Exact-Match-Anfrage beim permutationsbasiertem Ansatz des Lokal Sensitiven Hashens optimiert. Dazu wurden aufgrund von hohen Rechenzeiten Teilmengen aus dem gesamten

Datensatz ausgewählt, über denen optimiert wurde. Die so bestimmten Prototypen wurden anschließend auf die gesamte Datenmenge angewandt. Werden die Ergebnisse der Datenverteilung auf die Buckets mit Verteilungen der Daten bei zufälliger Wahl der Prototypen verglichen, so ist zu erkennen, dass die Abweichungen zum Optimum bei ersteren stabiler sind. Dabei sinkt die Abweichung bei einer größer werdenden Menge über der optimiert wird.
Da es bei der Wahl der Teilmenge aus der gesamten Datenmenge darauf ankommt, wie gut diese die gesamte Menge repräsentieren, bleibt zu untersuchen welche Elemente aus der Menge ausgewählt werden sollten.

Danksagung

Teile dieser Veröffentlichung beruhen auf Ergebnissen aus dem Forschungsvorhaben Digi-Dak (FKZ:13N10817), gefördert vom Bundesministerium für Bildung und Forschung (BMBF).

Literatur

[AA96] Alimoglu, F.; Alpaydin, E.: Methods of combining multiple classifiers based on different representations for pen-based handwriting recognition, In: TAINN. IEEE, pp.637-640, 1996.

[AI08] Andoni, A.; Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, Commun. ACM, 51(1): pp. 117-122, 2008.

[B12] Broneske, D.: Bachelorarbeit: Visuelle Analyse der Raumaufteilung und Bucketauslastung von permutationsbasierten Indexverfahren, Bachelorarbeit, Otto-von-Guericke Universität Magdeburg, 2012.

[CFN08] Chavez, E.; Figueroa, K.; Navarro, G.: Effective proximity retrieval by ordering permutations. In: IEEE Trans. on Pattern Analysis and Machine Intelli. 30, Nr. 9, pp. 1647-1658, 2008.

[FGK03] Fourer, R.; Gay, D. M.; Kernighan, B. W.: AMPL: A Modeling Language for Mathematical Programming, Second Edition, Brooks/Cole, Canada, 2003.

[GG97] Gaede, V.; Günther, O.: Multidimensional access methods, ACM Comp. Surveys , vol. 30, pp. 170-231, 1997.

[IM98] Indyk, P.; Motwani, R.: Approximate nearest neighbors: Towards removing the course of dimensionality. In: STOC, ACM, pp. 604-613, 1998.