Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

Bachelorarbeit

Visuelle Analyse der Raumaufteilung und Bucketauslastung von permutationsbasierten Indexverfahren

Verfasser:

David Broneske

16. April 2012

Betreuer:

Prof. Dr. rer. nat. habil. Gunter Saake,
Dipl.-Inform. Martin Schäler,
Dipl.-Wirtsch.-Inf. Thomas Leich

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg

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

Danksagung

Bei all denjenigen, die mich sowohl bei meinem Praktikum als auch beim Erstellen dieser Bachelorarbeit unterstützt und geführt haben, möchte ich mich herzlich bedanken.
Ich möchte mich vorallem bei Prof. Dr. rer. nat. habil. Gunter Saake bedanken, weil er mich in dieser Zeit unterstützt hat, wo er nur konnte und auch darüber hinweggesehen hat, wenn zu erledigende Arbeiten aufgeschoben werden mussten. Besonderer Dank gebührt weiterhin Dipl.-Inform. Martin Schäler für die Betreuung während des Schreibens dieser Arbeit. Seine Verbesserungsvorschläge haben diese Arbeit und meinen Schreibstil wesentlich voran gebracht.
Auch möchte ich mich bei Dipl.-Wirtsch.-Inf. Thomas Leich und der METOP GmbH für die Betreuung und Durchführung meines Praktikums bedanken. Weiteren Dank möchte ich meinen Freunden und meiner Familie aussprechen, die mich unterstützt haben, so gut es ihnen möglich war und welche mir durch ihre Geduld und ihr Verständnis in dieser Zeit einen starken Rückhalt gegeben haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

kNN-Anfragen . . . . . k-Nearest-Neighbor-Anfragen

LSH . . . . . Locality Sensitive Hashing

MBR . . . . . Minimum Bounding Rectangle

MBS . . . . . Minimum Bounding Sphere

UML . . . . . Unified Modeling Language

VBR . . . . . Virtual Bounding Sphere

Kapitel 1

1 Einleitung

Datenbanken haben seit jeher mit steigenden Anforderungen an die Datenhaltung zu kämpfen. Mit der Weiterentwicklung der Computersysteme und dem Preisverfall für Speichermedien [SSH11] wird die Speicherung vieler Daten zunehmend einfacher. Aufgrund der dadurch zunehmenden Digitalisierung von Daten werden Datenbanken mehr und mehr für die Speicherung großer Datenmengen genutzt. Auch die Beschaffenheit der Daten entwickelt sich von der Speicherung einfacher Daten mit wenigen Dimensionen zu komplexen Daten mit mehreren Dimensionen [HK98]. Diese Multimediadaten führen zur Entwicklung von Multimediadatenbanken, die die Speicherung von Daten mit vielen Dimensionen bewältigen müssen. Die Anforderungen liegen dabei nicht nur in der persistenten Speicherung, sondern ebenso in der schnellen Verfügbarkeit der Daten. Mit dem Ziel eines schnellen, performanten Zugriffs auf die Daten wurden in der Literatur zahlreiche Indexstrukturen vorgeschlagen [BKSS90; LJF94; BKK96; KS97]. Mit diesen Indexverfahren wird durch unterschiedliche Strategien versucht die Daten anhand ihrer einzelnen Eigenschaften einzuteilen, um bei Anfragen der Daten einen schnellen Zugriff auf diese zu bieten. Mit dem Steigen der Dimensionen beeinflusst jedoch der sogenannte *Fluch der hohen Dimensionen* [WSB98] immer weiter die Anfragegeschwindigkeit der Daten. Dieser Fluch besagt, dass die Indexstrukturen bei steigenden Dimensionen ihren Vorteil verlieren, der aufgrund der Einteilung der Daten zu stande kam. Der Aufwand beim Suchen in diesen Indexstrukturen kann bei hohen Dimensionen aufwändiger als ein sequenzielles Vergleichen jedes einzelnen Datums in der Datenbank mit dem Anfragedatum sein. Deshalb werden in dem Bereich der Datenbanken zunehmend neue Indexstrukturen vorgeschlagen, die den Fluch der hohen Dimensionen bewältigen sollen. Ein Ansatz stellt hierbei die Nutzung von Hashverfahren als approximierendes Verfahren dar, die mit Hilfe einer Hashfunktion den Raum einteilen und durch diese Funktion einen direkten Zugriff auf die einzelnen Regionen ermöglichen. Eine vielversprechende Idee zur Gewinnung der Hashwerte stellt hierbei der permutationsbasierte Ansatz dar [CGFN08]. Durch verschiedene Positionierung von Prototypen wird die Permutation als Ordnung der Abstände zu diesen Prototypen erstellt, welche als Hashwert benutzt wird. Die dabei entstehenden Regionen sind mitunter nicht trivial und sollen deshalb in dieser Arbeit näher betrachtet werden.

1.1 Zielstellung

Zur Auswertung der Raumaufteilung des permutationsbasierten Ansatzes soll ein Tool entwickelt werden, mit dem die Raumaufteilung bei verschiedenen Datenverteilungen und verschiedener Positionierung der Prototypen im Raum dargestellt werden soll. Dafür soll ein zwei- bis dreidimensionaler Raum dargestellt werden, in den die Prototypen und verschiedene Datenbasen eingefügt werden können. Anhand der gesetzten Prototypen soll die Raumaufteilung visualisiert werden und beim Einfügen von Datenpunkten soll der Füllgrad an Datenpunkten der einzelnen Regionen angezeigt werden können. Durch die Umsetzung dieser Anforderungen soll die Raumaufteilung ausgewertet werden können und damit Schlüsse über das vorteilhafte Setzen der Prototypen in den Raum aufgezeigt werden.

1.2 Gliederung der Arbeit

Diese Arbeit beginnt im nun folgenden Kapitel 2 mit den Grundlagen. Dabei soll auf die verschiedenen Eigenschaften von zu speichernden Daten eingegangen und verschiedene Anfragetypen vorgestellt werden, die die in der Datenbank gespeicherten Daten wieder zurückgeben. Weiterhin werden schon bestehende Indexverfahren vorgestellt und von dem hier behandelten Permutationsansatz für Hashverfahren abgegrenzt. In Kapitel 3 wird dann anhand verschiedener Raumaufteilung, die durch zufälliges Setzen der Prototypen entstanden sind, eine Analyse einer vorteilhaften Positionierung der Prototypen motiviert. Zur besseren Durchführung dieser Analyse wird vorgeschlagen ein Tool zu entwickeln, mit dem die entstehende Raumaufteilung automatisiert dargestellt werden kann. Als Zielsetzung für die Implementierung des Tools wurden dann in Kapitel 4 verschiedene Anforderungen an das Tool festgelegt. Diese beziehen sich unter anderem auf das Setzen der Prototypen, die Darstellung der Raumaufteilung und das Laden von Datenbasen. Anhand dieser Anforderungen wird dann ein Tool implementiert, das es ermöglicht die Raumaufteilung durch die Prototypen darzustellen. Einige Implementierungsdetails werden dafür in Kapitel 5 erläutert, um dann im darauf folgenden Kapitel die Umsetzung der gestellten Anforderungen auszuwerten. Dabei wird erläutert inwieweit die einzelnen Anforderungen erfüllt wurden und welche weiteren Arbeitspakete für weitere Projekte an dem Tool offen bleiben. Außerdem werden erste Erkenntnisse zur Raumaufteilung vorgestellt, die bei der Arbeit mit dem Tool entstanden sind. Dafür werden Vorschläge für ein vorteilhaftes Setzen der Prototypen für verschiedene Anfragetypen gemacht. Abschließend wird im Kapitel 7 die Arbeit zusammengefasst und ein Ausblick gegeben. Dabei wird darauf eingegangen welche Anforderungen in späteren Arbeiten an dem Tool umgesetzt werden können und welche Betrachtungen der Raumaufteilung in Bezug auf permutationsbasierte Indexverfahren noch ausgewertet werden können.

Kapitel 2

2 Grundlagen

In diesem Kapitel wird ein Einblick in die Herausforderungen beim Indexieren von hochdimensionalen Räumen gegeben. Dafür wird zuerst die Beschaffenheit der Daten und die dadurch entstehenden Räume vorgestellt. Im Abschnitt 2.2 werden dann die möglichen Anfragetypen eingeführt. Danach wird ein Einblick in das Thema der hochdimensionalen Indexstrukturen gegeben und schließlich Vertreter verschiedener Klassen von Indexstrukturen vorgestellt und deren Raumaufteilung gegenübergestellt.

2.1 Aufbau eines Datensatzes

Als erstes soll an dieser Stelle ein Einstieg in die Daten und ihre speziellen Eigenschaften gegeben werden, sowie ein Einblick in die Datenräume, die sich aufgrund der Daten ergeben, verschafft werden. Dies dient dazu die Notwendigkeit von hochdimensionalen Indexstrukturen zu erkennen und darzustellen. Jedes einzelne Datum besteht aus mehreren Attributen, die es identifizieren. Diese Attribute können zum Beispiel bei Bildern die Farbwerte der einzelnen Pixel darstellen, oder bei Personen Angaben wie Name, Vorname, Geburtsdatum und Wohnort. Die Menge an identifizierenden Attributen wird auch *Schlüssel* genannt und Nutzdaten, die nicht zur Identifikation beitragen, werden als *Payload* bezeichnet. Vereinfacht werden in dieser Arbeit die Attributsausprägungen als Zahlenwerte dargestellt. Es besteht natürlich die Möglichkeit jegliche Datentypen auf Zahlenwerte abzubilden, dadurch gelten die hier vorgestellten Betrachtungen auch für diese Daten. Besteht der Schlüssel aus einem Attribut, so sind diese als eindimensionale Daten zu bezeichnen. Besitzen die Daten eine Schlüsselgröße zwischen zwei und circa 45 Attributen, so wird von mehrdimensionalen bzw. multidimensionalen Daten gesprochen. Bei mehr als 45 Attributen werden diese laut Weber et al. als hochdimensionale Daten bezeichnet [WSB98]. Valle et al. legen diese Grenze etwas höher und nennen Daten erst mit mehr als 100 Attributen hochdimensional [VCPF08]. Die vorhandenen Daten werden von Datenbanken gespeichert und effizient indexiert, um schnellen Zugriff auf diese Daten zu ermöglichen. Beim Suchen nach Datenpunkten in der Datenbank gibt es verschiedene Anfragetypen mit verschiedenen Eigenschaften, die im nachfolgenden Abschnitt erläutert werden.

2.2 Anfragetypen

Bei der Suche in einem eindimensionalen Raum haben sich sowohl *Exact-Match-Anfragen* als auch *Bereichsanfragen* als wichtig herausgestellt [SSH11].

Bei Exact-Match-Anfragen werden die Ausprägungen aller vorhandenen Attribute eines Datums angegeben und genau dieses Datum wird gefunden, wenn es in der Datenbank vorhanden ist. Diese Anfrage dient zumeist dazu, Duplikate beim Einfügen von neuen Datenpunkten zu verhindern [SSH11].

Bereichsanfragen hingegen geben einen bestimmten Bereich an, in dem sich die Attributsausprägungen der einzelnen Daten befinden müssen, um in die Ergebnismenge aufgenommen zu werden [SSH11]. Eine Bereichsanfrage wäre zum Beispiel wie viele Preise in der Datenbank zwischen 100 € und 300 € liegen.

Komplexere Anfragen lassen sich auf Daten mit mehreren Dimensionen ausführen. Neben den Bereichsanfragen und Exact-Match-Anfragen, gibt es im mehrdimensionalen Raum noch weitere Anfragen. Zum einen gibt es *Partial-Match-Anfragen*, die nur für einige der vorhandenen Attribute die Wertausprägungen der Anfrageobjekte vorgeben [SSH11]. Zum anderen nimmt die Suche nach den k nächsten Nachbarn *(kNN-Anfragen genannt)* an Relevanz im hochdimensionalem Raum zu [BBK01; HS03]. Das Ergebnis dieser Suche sind die k nächsten Elemente zum Anfragepunkt q aus der Menge der Datenpunkte S. Formal muss die Ergebnismenge A also folgenden Bedingungen genügen [Nav02]:

- $A \subseteq S$,
  trivialer Weise muss die Ergebnismenge A eine Teilmenge der Menge der Datenpunkte sein, auf der die Anfrage durchgeführt wird.

- $|A| = k$,
  die Ergebnismenge A soll die Mächtigkeit k besitzen, da k nächste Nachbarn gesucht werden. Hierbei muss beachtet werden, dass auch mindestens k Elemente in der Menge S vorhanden sein müssen, um die Anfrage ausführen zu können.

- $\forall x \in A, y \in (S - A) : \|x, q\| \leq \|y, q\|$,
  unter der Voraussetzung, dass $\|\cdot, \cdot\|$ die *Distanzfunktion* darstellt, bedeutet diese Bedingung, dass es keinen Punkt gibt, der nicht in der Ergebnismenge ist, aber näher zum Anfragepunkt liegt als ein Punkt in der Ergebnismenge.

2.3 Anforderungen in hohen Dimensionen

Im vorigen Abschnitt wurde unter anderem auf die kNN-Anfragen eingegangen. Für die Berechnung der nächsten Nachbarn zu einem Anfragepunkt wird der Abstand zwischen den Punkten im Raum benötigt, damit eine gewisse Nähe von Punkten bestimmt werden kann. Dabei gibt es verschiedene Formen der Distanzmessung, die im folgenden Abschnitt genauer betrachtet werden.

2.3.1 Distanzfunktionen

Die Wahl der passenden Distanzfunktion, auch Norm genannt, für eine vorliegende Aufgabe stellt hierbei eine Anforderung in hohen Dimensionen dar [AHK01]. Die meisten Normen gehören zu der Familie der L_p-Normen, für die gilt:

- $x, y \in \mathbb{R}^d$, $p \in \mathbb{N}$, $L_p(x,y) = \sqrt[p]{\sum_{i=1}^{d} (\|x^i - y^i\|^p)}$.

Hierbei wird angenommen, dass die Attributwerte Zahlenwerte im d-dimensionalen Raum sind. Aggarwal et al. zeigen in [AHK01], dass p kleiner als 3 gewählt werden soll, da bei hohen Dimensionen für steigende p der Kontrast zwischen dem weitesten und dem nächsten Punkt abnimmt.

Für $k = 1$ ergibt sich die *Manhattan Distanzmetrik*:

- $L_1 = \sum_{i=1}^{d} (\|x^i - y^i\|)$

dies ergibt eine quadratische Form der Distanzbereiche. Um kreisförmige Distanzabstände zu bekommen, wurde die *Euklidische Distanzmetrik* eingeführt:

- $L_2 = \sqrt[2]{\sum_{i=1}^{d} (\|x^i - y^i\|)^2}$

Das Verhalten der Manhattan und Euklidschen Metrik wurden in Abbildung 2.1 noch einmal veranschaulicht.

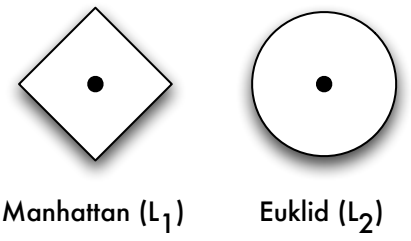


Abbildung 2.1: L_1- und L_2-Norm im Datenraum

2.3.2 Fluch der hohen Dimensionen

Eine weitere Anforderung, die der hochdimensionale Raum stellt, ist der sogenannte *Fluch der hohen Dimensionen* (engl.: Curse of dimensionality). Er besagt, dass die Performanz der Indexstrukturen beim Indexieren von hochdimensionalen Daten abnimmt und sogar zu einer sequenziellen Suche ausarten kann [WSB98]. Existierende Indexstrukturen verlieren, laut Weber et al. [WSB98], ihren Vorteil gegenüber der sequenziellen Suche schon bei 10 dimensionalen Datenräumen.

Deshalb wurde in der Forschung das Thema des Fluchs der hohen Dimensionen oftmals aufgegriffen und versucht neue Indexstrukturen aufzuzeigen, die den Fluch dämpfen [SRF87; LJF94; WJ96; KS97; WSB98; SYUK00].

Eine Herausforderung, die im hochdimensionalem Raum besteht, ist zum einen, dass der Raum nur spärlich besetzt ist. Angenommen ein 10-dimensionaler Raum, wobei jede Dimension zehn Ausprägungung besitzt, wird mit 100.000 Datenpunkten befüllt. Insgesamt gibt es im Raum 10^{10} mögliche Punkte. Somit ist gerade einmal 0,001% des Raumes gefüllt. Bei steigenden Dimensionen verschärft sich das Problem, denn um einen gleichbleibenden Füllgrad zu gewährleisten, muss die Anzahl an Datenpunkten exponentiell steigen [WSB98].

Weiterhin sind die einzelnen Dimensionen bei Realdaten nicht immer gleichmäßig auf die Atrributsausprägungen verteilt. Dadurch entstehen oft sogenannte *Cluster*, also Anhäufungen von Datenpunkten auf einen kleinen Bereich des Datenraums. Weiterhin kann es dazu kommen, dass sich viele Datenpunkte vorallem an den Rändern des Raumes befinden [BBB+97]. Dies führt oft zu einem Entarten der Indexverfahren.

Ein weiteres Problem stellt der vergrößerte Abstand zu den nächsten Nachbarn dar. Der steigende Abstand zwischen den Datenpunkt führt dazu, dass sich auch die nächsten Nachbarn immer weiter voneinander entfernen [BBB+97]. Bei kNN-Anfragen muss somit mehr Raum durchsucht werden und die Anfragegeschwindigkeit kann beeinträchtigt werden.

Nachdem in diesem Abschnitt die Anforderungen vorgestellt wurden, die auf Indexstrukturen im hochdimensionalen Raum zu kommen, werden im nächsten Abschnitt konkrete Vertreter hochdimensionaler Indexstrukturen vorgestellt.

2.4 Hochdimensionale Indexstrukturen

Zum Indexieren von Objekten anhand von vielen verschiedenen Attributen werden hochdimensionale Indexstrukturen benötigt. Diese versuchen den Raum, den die Attribute aufspannen, in bestimmte Bereiche einzuteilen, um in den entstandenen Unterräumen die gesuchten Datenpunkte schneller zu finden. Bei dieser Raumaufteilung wird unterschieden zwischen *raumpartitionierende Verfahren (engl. space partitioning methods)* und *datenpartitionierende Verfahren (engl. data partitioning methods)* [SWS+00; BBK01; WSB98].

- Raumpartitionierende Verfahren teilen den gesamten Raum entlang vordefinierter oder berechneter Grenzen. Hierbei können sehr dichtbesetzte aber auch sehr dünnbesetzte Regionen entstehen. Dieses Verhalten wirkt sich negativ auf die Selektivität aus.

- Datenpartitionierende Verfahren bilden die Regionen aufgrund der Verteilung der Daten im Raum. Dadurch besitzen die Regionen unterschiedliche Ausdehnungen im Raum; je dichter eine Region besetzt ist, desto feingranularer muss sie unterteilt werden. Der Vorteil ergibt sich daraus, dass Teile des Datenraums, die keine Datenpunkte enthalten, nicht indexiert werden.

Weiterhin ist es für den Anwendungsfall wichtig, ob die Indexstruktur exakt oder approximativ arbeitet. Bei exakten Verfahren werden bei der Suche immer die k nächsten Nachbarn gefunden. Für approximative Verfahren wird diese harte Definition der k nächsten Nachbarn modifiziert. Es werden hierbei k Datensätze geliefert, die dem Anfrageobjekt sehr ähnlich sind, jedoch nicht zwangsweise zu der Menge der k nächsten Nachbarn äquivalent sind.

2.4.1 Klassen von Indexstrukturen

Indexstrukturen lassen sich aufgrund ihrer Funktionsweise in *Baumverfahren, Optimierte Sequenzielle Suche, Space-Filling-Curves und Hashverfahren* einteilen [GG98]. Um die Indexstrukturen auch klassenübergreifend zu vergleichen, wird deren Raumaufteilung anhand des zweidimensionalen Beispielraums aus Abbildung 2.2 dargestellt. Dieser Raum wurde mit 26 zufällig verteilten Datenpunkten befüllt.

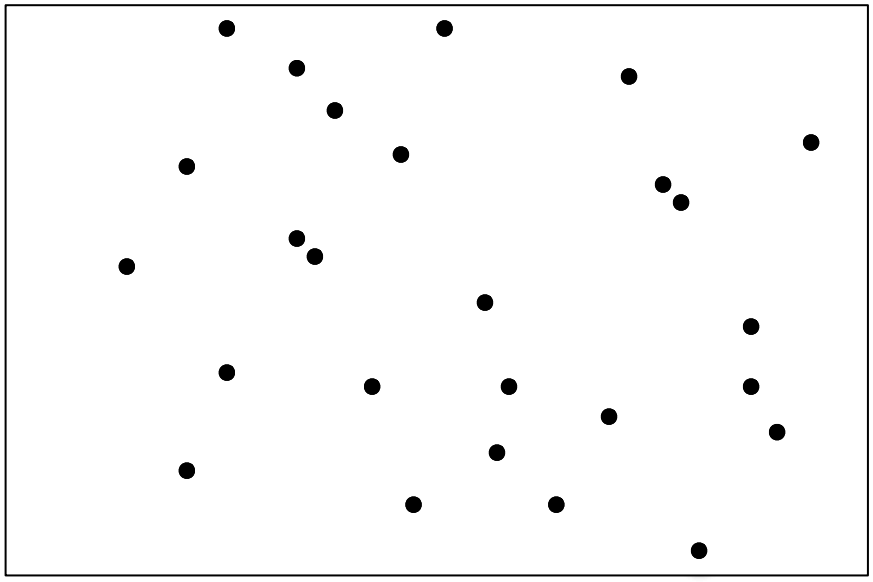


Abbildung 2.2: Zweidimensionaler Beispielraum

Die einzelnen Schwächen der Indexstrukturen sind im niederdimensionalen Raum, wie dem Beispielraum, nur ansatzweise nachvollziehbar, es reicht aber aus, um auf Missstände hinweisen zu können und Verbesserungen unter den Indexstrukturen zu veranschaulichen. Die hier vorgestellten problematischen Eigenschaften verschärfen sich mit steigenden Dimensionen und wirken sich auf die Anfragegeschwindigkeit der jeweiligen Indexstruktur aus.

2.4.2 Baumverfahren

Baumverfahren stellen in Datenbankmanagementsystemen eine der wichtigsten Indexstrukturen dar [GG98]. Dabei wird versucht, durch hierarchisches Einteilen des Suchraums, die Anfragekosten von $O(n)$ auf $O(log(n))$ zu senken [WSB98]. Einer der ersten mehrdimensionalen Vertreter der Baumverfahren ist der *k-d-Baum* [BF79].

k-d-Baum

Der k-d-Baum teilt den Raum anhand der Attributsausprägungen der eingetragenen Punkte. Dafür wird die zu teilende Dimension zyklisch gewechselt. Die entstehende Raumaufteilung kann anhand der Abbildung 2.3 nachvollzogen werden, die durch die 26 Datenpunkte erzeugt wird. Auch die daraus resultierende Baumstruktur wurde anhand der Aufteilung der oberen rechten Region dargestellt. Hieran ist erkennbar, dass der k-d-Baum leicht entartet und die Suche dadurch verhältnismäßig lange dauern kann. Der aufgespannte Baum enthält die Regionen in hierarchischer Ordnung von der obersten Ebene, auch Wurzelebene genannt, bis hin zur untersten Ebene, der Blattebene.

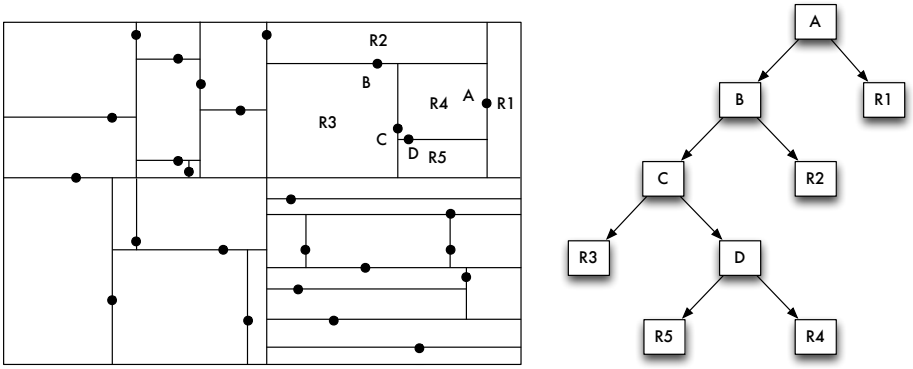


Abbildung 2.3: Raumaufteilung des k-d-Baums und dessen Aufbau

Der k-d-Baum gehört zu den raumpartitionierenden Verfahren und ermöglicht gute Performanz bei Anfragen auf Daten im mehrdimensionalen Raum. Bei steigenden Dimensionen wirkt jedoch der Fluch der Dimensionen und die Suchgeschwindigkeit sinkt [BL97].

R-Baum

Im Jahre 1984 wurde der *R-Baum* entwickelt, um die Probleme des k-d-Baums auszubessern. Mit diesem ist es möglich nicht nur Datenpunkte, sondern auch Objekt mit einer Ausdehnung im Raum zu indexieren [Gut84]. Ursprünglich wurde er als multidimensionale Indexstruktur entworfen, wird aber heutzutage auch in höheren Dimensionen eingesetzt.

Die Raumaufteilung beim R-Baum wird durch minimale umschreibende Rechtecke *(engl. minimum bounding rectangles, MBRs)* vorgenommen. Um die Ausdehnung von MBRs im Raum zu bestimmen, werden nur zwei, im Rechteck gegenüberliegende, Punkte benötigt. Die Koordinaten dieser Punkte werden dann in dem entsprechenden Vaterknoten abgelegt, dessen MBR die Kind-MBRs vollständig umschließt. Die Raumaufteilung und der dazugehörige Baum wurden in Abbildung 2.4 noch einmal veranschaulicht. Die Datenpunkte werden von den MBRs (R1-R12) vollständig umschlossen. Hierbei kann es durchaus zu Überlappungen der MBRs kommen. Der dafür aufgespannte Baum enthält alle MBRs in hierarchischer Ordnung bis zur Blattebene, die schließlich die Datenpunkte enthält, die das übergeordnete MBR umschließt. Aufgrund der besseren Anschaulichkeit wurden hier nur die Datenpunkte A-C eingetragen.

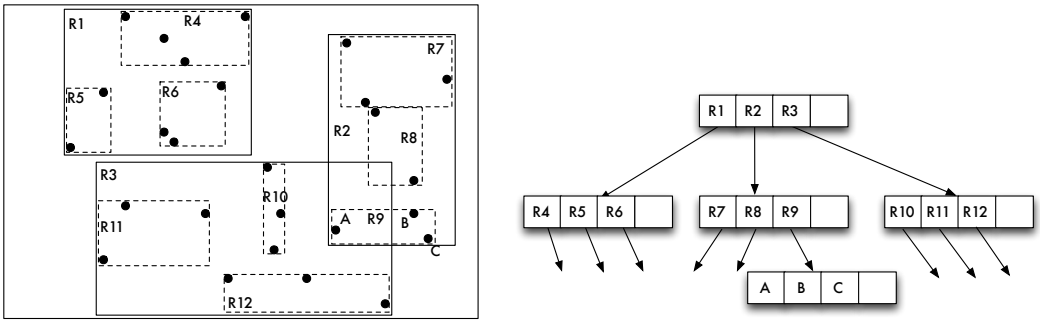


Abbildung 2.4: Raumaufteilung des R-Baums und dessen Aufbau

Wie in Abbildung 2.4 zu sehen ist, indexiert der R-Baum nicht den gesamten Raum. Die Minimum Bounding Rectangles umfassen nur die nötigen Regionen, um die Datenpunkte einzuschließen. Somit ist der R-Baum zu den datenpartitionierenden Verfahren zu zählen.

Bei der Suche nach Datenpunkten im R-Baum, zum Beispiel der Exact-Match-Anfrage, wird auf der Wurzelebene mit der Suche begonnen und sukzessiv die jenigen Kind-MBRs untersucht, dessen Vater-MBR den Anfragepunkt überdecken. Diese Vorgehensweise kann bei einer schlechten Datenverteilung die Suche in mehreren Unterbäumen nach sich ziehen. In Abbildung 2.4 führt die Suche nach dem Datenpunkt A zu einer Suche im Teilbaum des MBRs R2 und R3. Da im MBR R3 kein Kind gefunden werden kann, dass den Anfragepunkt weiter überdeckt, endet hier die Suche für diesen Teilbaum. Hingegen kann der Pfad R2 $\rightarrow$ R9 erfolgreich ausgewertet werden und schließlich der Anfragepunkt A zurück gegeben werden.

Aus diesem Suchbeispiel ist ersichtlich, dass der Grad der Überlappung indirekt proportional zur Anfragegeschwindigkeit ist [BKK96]. Sogar bei einfachen Punktanfragen kann es dazu kommen, dass mehrere Pfade des R-Baums durchsucht werden müssen, weil an dieser Stelle mehrere MBRs übereinander liegen. Dieses Verhalten schränkt die Performanz des R-Baums ein. Deshalb wurde in den darauf folgenden Jahren der R^+*-Baum* [SRF87] und der R^**-Baum* [BKSS90]

entwickelt, die dieses Problem aufgreifen. Der R^+-Baum verbietet hierbei Überlappungen der MBRs, sodass alle Regionen disjunkt sind. Sollten im R^+-Baum nicht nur Punktdaten, sondern auch geometrische Objekte gespeichert werden, so kann es dazu kommen, dass sie getrennt behandelt und in mehrere Knoten abgelegt werden *(engl. clipping)*. Diese Eigenschaft bezieht sich jedoch nur auf Körper mit einer Ausdehnung im Raum, weshalb die speziell dafür entwickelten Algorithmen hier weniger von Bedeutung sind. Weiterhin ist die Regionenaufteilung unseres Beispieldatenraums mit ausschließlich Punktdaten äquivalent zu der des R^*-Baum, der im folgendem Abschnitt genauer betrachtet wird.

R^*-Baum

Die Entwickler des R^*-Baum sahen mehrere Faktoren, die den R-Baum in der Anfragegeschwindigkeit einschränken. Zum einen sind die MBRs zu groß und indizieren zu viel unbesetzten Raum. Zum anderen schränken die Überlappungen die Performanz ein [BKSS90]. Die Raumaufteilung des R-Baums und somit die entstehende Baumstrukur wird durch die Einfügereihenfolge bestimmt. Durch das zufällige Einfügen von Datensätzen kann die Baumstruktur unausgeglichen werden [BKSS90]. Dies lässt sich auch beim B-Baum, auf den der R-Baum basiert, erkennen. Um einen ausgewogenen R^*-Baum zu erhalten, wird bei einem bevorstehenden Split eines Knotens zuerst ein *Wiedereinfügen (engl. reinsert)* vollzogen. Dafür werden Datenpunkte aus dem Baum herausgenommen und dann in besserer Reihenfolge wieder eingefügt. Dadurch wird der Baum ausgeglichener und Splits können verhindert werden. Kommt es jedoch trotzdem zu einem Split eines Knotens, so versucht der Algorithmus entstehende Überlappungen möglichst zu vermeiden, wie in Abbildung 2.5 angedeutet. Der Punkt A wurde hierbei durch das Wiedereinfügen nicht in R9, sondern in R10 eingeordnet. Weiterhin wanderte ein Punkt aus R10 zu R9, wodurch die vorhandene Überlappung eliminiert wurde. Dieser Algorithmus benötigt zwar mehr Verarbeitungszeit, im Gegenzug ist aber die Anfragegeschwindigkeit optimiert [BKSS90], weil bei Exact-Match-Anfragen immer nur ein Pfad durch den Baum genommen werden kann.

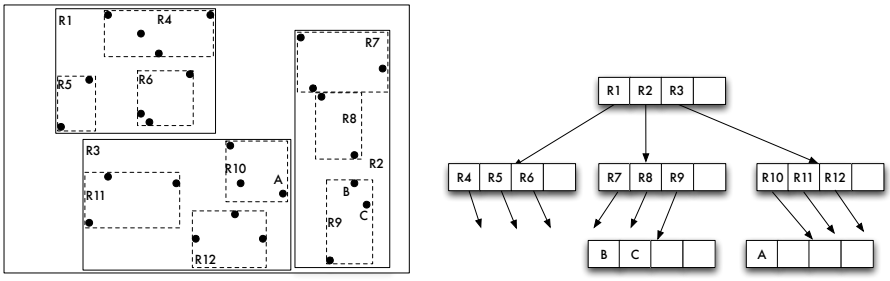


Abbildung 2.5: Raumaufteilung des R^*-Baum

Die Überlappungen sind im zweidimensionalen Raum zwar relativ gut zu vermeiden, bei höheren Dimensionen kann es jedoch unvermeidlich zu Überlappungen kommen. Dieses Problem wird vom X-Baum adressiert [BKK96].

X-Baum

Der *X-Baum* wurde im Jahre 1996 entwickelt, um die Performanz bei Überlappungen zu verbessern [BKK96]. Er basiert auf dem R*-Baum und dessen Splitalgorithmus, mit dem entstehende Überlappungen vermieden werden sollen. Entstehen beim Einfügen von neuen Datensätzen trotzdem Überlappungen von MBRs, so werden die betroffenen MBRs verschmolzen und der korrespondierende Knoten wird zum *Superknoten*. Dadurch müssen bei der Suche nicht mehrere Pfade ausgewertet werden, sondern nur der Superknoten sequenziell durchsucht werden.

Wie schon erwähnt, ist es im zweidimensionalen Raum möglich die Überlappungen zu verhindern, wodurch keine Superknoten entstehen werden. Deshalb ist hier eine Visualisierung der Raumaufteilung nicht zielführend und würde der Abbildung 2.5 entsprechen. Der Aufbau eines vorstellbaren X-Baums in höheren Dimensionen kann in Abbildung 2.6 nachempfunden werden.

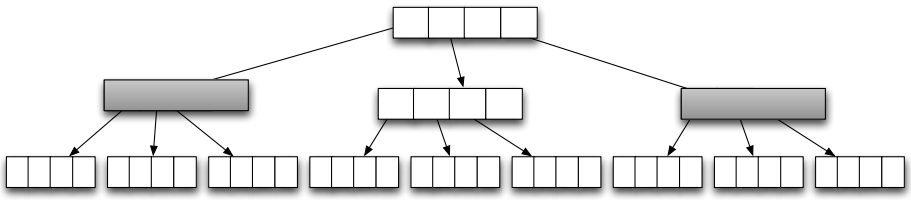


Abbildung 2.6: Aufbau des X-Baums

Die schwarzen dargestellten Knoten, die Superknoten, besitzen keine einzelnen MBRs, sondern sind große Knoten, die nur einmal sequenziell durchsucht werden müssen.

SS-Baum

Ein anderer Ansatz, den Raum aufzuteilen, verfolgt der *SS-Baum* [WJ96]. Anstatt auf minimal umschließende Rechtecke zurückzugreifen, wird hierbei der Raum durch minimal umschließende Kugeln *(engl. minimum bounding spheres, MBSs)* eingeteilt, wie in Abbildung 2.7 zu sehen ist. Ein Vorteil des SS-Baums ist, dass beim Speichern der Region in den Knoten nur der Mittelpunkt und der Radius abzulegen sind. MBRs hingegen benötigen mindestens zwei gegenüberliegende Eckpunkte des Rechtecks um dieses im Raum zu rekonstruieren. Dieser Gewinn an Speicherplatz ermöglicht es mehr Einträge pro Knoten abzuspeichern und somit die Baumbreite *(engl. fanout)* zu erhöhen. Dadurch wird die Tiefe des Baums geringer, weil die untersten Knoten aufgrund der Platzersparnis eine Ebene höher eingeordnet werden können. Sparen wir eine Baumebene ein, so muss eine Leseoperation von der Festplatte weniger ausgeführt werden, um die Blattebene und somit die indexierten Daten zu erreichen.

Im Vergleich schneidet der SS-Baum besser als der R*-Baum ab [KS97], jedoch teilt der SS-Baum den Raum nicht optimal auf. Wie auch in Abbildung 2.7 nachvollzogen werden kann, haben die MBSs zwar einen kleinen Durchmesser und

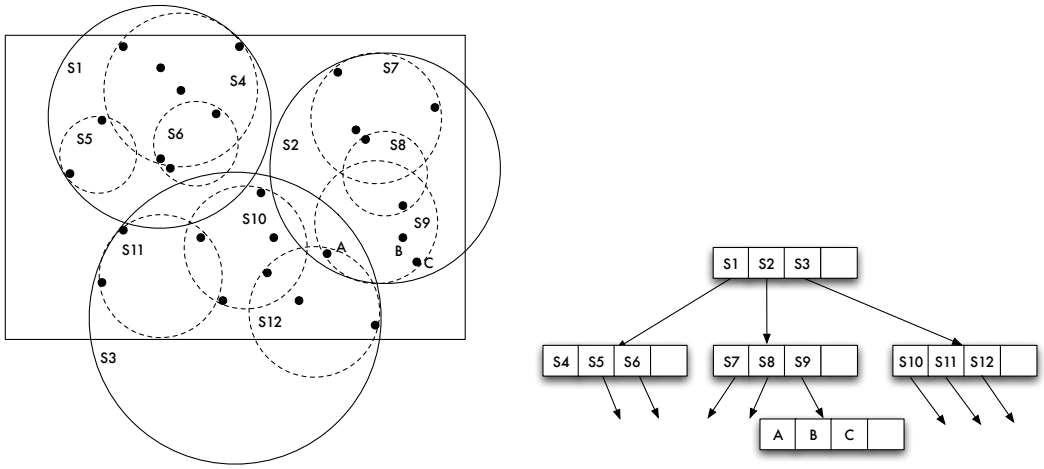


Abbildung 2.7: Raumaufteilung und Aufbau des SS-Baums

somit eine geringe Ausbreitung im Raum, jedoch besitzen sie ein großes Volumen und indexieren somit zu viel unbenutzten Raum [KS97]. Beispielsweise befindet sich ungefähr ein drittel von S3 außerhalb des Datenraums.

SR-Baum

Die Raumaufteilung des SS-Baums ist, wie beschrieben, immer noch nicht optimal, deshalb wurde im Jahre 1997 der SR-Baum von Katayama und Satoh vorgeschlagen [KS97]. Die Regionenaufteilung des SR-Baums wird durch den Durchschnitt eines MBRs und eines MBSs erzielt. Zur Veranschaulichung sind neben der Raumaufteilung durch die MBRs und MBSs auch die entstehenden Regionen der Wurzelebene in Abbildung 2.8 dargestellt. Die zugehörige Baumstruktur ist äquivalent zu der des SS-Baums aus Abbildung 2.7.

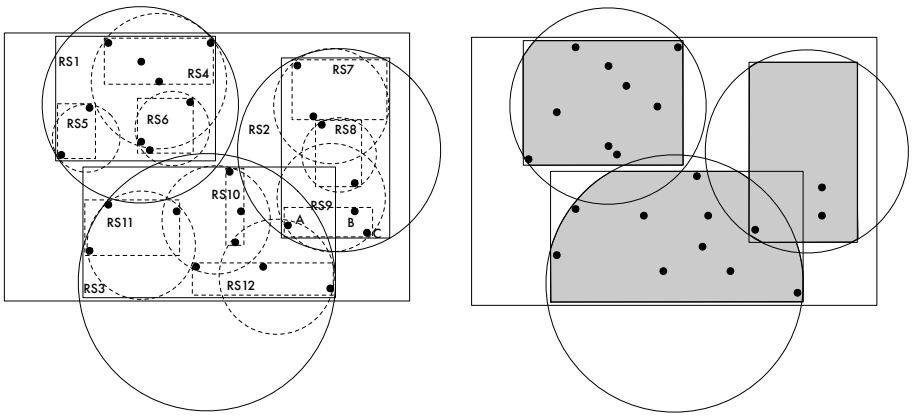


Abbildung 2.8: Raumaufteilung des SR-Baums und die entstehenden Bereiche für die Wurzelebene des SR-Baums

Minimum Bounding Spheres haben zwar einen kleinen Durchmesser, jedoch besitzen die Kugeln ein sehr großes Volumen. Minimum Bounding Rectangles nehmen ein minimales Volumen ein, haben jedoch eine große Ausdehnung im Raum aufgrund der Diagonalen. Für die Suche der nächsten Nachbarn ist es von Vorteil Regionen mit minimalem Durchmesser zu erstellen, da dadurch Überlappungen verringert werden und keine Teilräume betrachtet werden müssen, in denen keine nächsten Nachbarn des Anfragepunktes liegen [KS97]. Weiterhin ist aber auch das Ziel, den Raum in Regionen mit minimalem Volumen einzuteilen, weil dadurch kein *"leerer Raum"* indexiert wird. Deshalb bestehen die Regionen des SR-Baums aus dem Durchschnitt eines MBRs und eines MBSs, um somit ein minimales Volumen und einen minimalen Durchmesser der Regionen zu erzielen.

Beim SR-Baum nehmen die Knoten jedoch verhältnismäßig viel Speicherplatz ein, da sowohl das MBR als auch das MBS abgespeichert werden muss. Dies verringert die Baumbreite und somit werden mehr Ebenen bis zu den Blattknoten durchlaufen. Bei den MBRs und MBSs verhält sich die Knotengröße aufgrund der zu speichernden Punkte direkt proportional zur Dimensionszahl [SYUK00].

A-Baum

Mit dem Ziel den benötigten Speicherplatz zu verringern, wurden von Sakurai et al. *Virtual Bounding Rectangles (VBRs)* eingeführt und der A-Baum entwickelt [SYUK00]. Dieser vereint Ansätze des R-Baums und des VA-Files, welches im Abschnitt 2.4.3 genauer betrachtet wird. Durch die VBRs wird eine relative Approximation ausgedrückt; hierbei wird jeweils das Kind-MBR zum Vater-MBR in Abhängigkeit gesetzt und als Bitstring gespeichert. Dies hat den Vorteil, dass in einem Knoten nur das aktuelle MBR und als Verweis auf die Kinder nur die Bitstring der VBRs gespeichert werden muss. Dadurch sinkt die Speichergröße der Einträge im Knoten, wodurch wiederum mehr Einträge pro Knoten speicherbar sind. Dadurch kann die Baumtiefe gesenkt werden. In Abbildung 2.9 wurde die Raumaufteilung für das MBR R2 dargestellt.

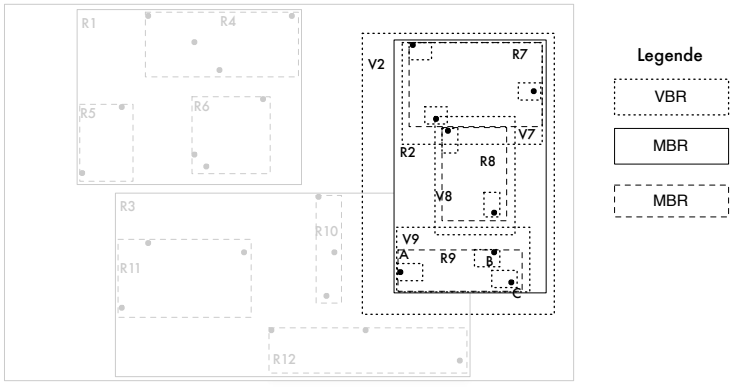


Abbildung 2.9: Raumaufteilung des A-Baums anhand MBR R2 und dessen untergeordneten Rechtecken

Die verwendete Approxiation setzt hierbei die Diagonalen des übergeordneten Rechtecks zur Diagonale des zu approximierenden Rechtecks ins Verhältnis. Ein Datenpunkt kann hierbei als eine Diagonale, dessen Anfangs- und Endpunkt in einem gemeinsamen Punkt liegen, aufgefasst werden [SYUK00], sodas auch für jeden Datenpunkt ein VBR erstellt werden kann (Abbildung 2.9).

Der entstehende Baum kann in Abbildung 2.10 betrachtet werden. Der A-Baum hat als Wurzelknoten eine Menge von VBRs, welche dann auf die darunter liegenden MBRs verweisen. Bei der Suche wird nun, als Vorfilterphase, die Ausdehnung des entsprechenden VBRs im Raum mit dem Anfragepunkt verglichen. Überdeckt das VBR den Anfragepunkt, so werden die exakten Koordinaten des zugehörigen MBRs geladen und mit diesen der Abgleich durchgeführt. Der Vorteil liegt darin, dass ohne Laden und Vergleichen der MBR-Knoten ermittelt werden kann, ob das Kind-MBR für eine weitere Betrachtung sinnvoll ist, oder außer Acht gelassen werden kann.

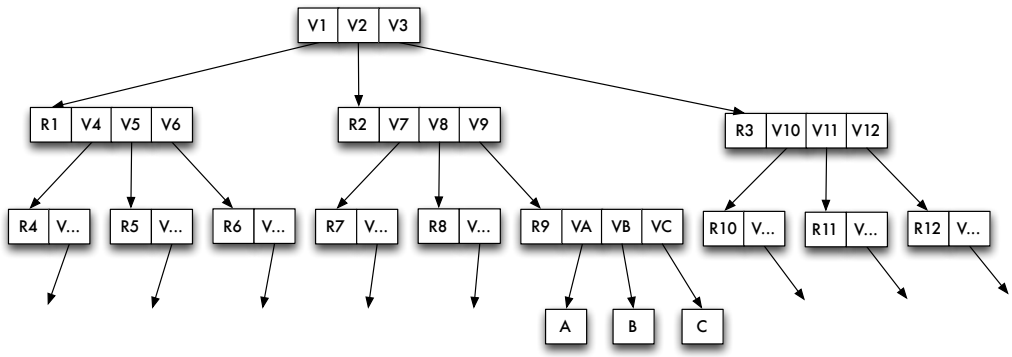


Abbildung 2.10: Aufbau des A-Baums

Weiterhin werden die VBRs als Bitstrings gespeichert und nehmen dadurch weniger Speicherplatz weg, als die Koordinaten eines MBRs. Dieses Verhalten begünstigt den Speicherverbrauch einzelner Knoten, sodass mehr Knoten im Arbeitsspeicher gehalten werden können [SYUK00]. Des Weiteren stellen die VBRs, wie in Abbilung 2.9 ersichtlich, eine übergroße Approximation des MBRs dar, sodass auf keinen Fall mögliche Kandidaten zu unrecht verworfen werden. Die Speicherplatzersparnis durch Benutzung von Bitstrings wurde hierbei dem VA-File entnommen, das im nächsten Abschnitt näher untersucht wird.

Die vorgestellten Baumverfahren versuchen die Herausforderungen der Indexierung von hochdimensionalen Daten so gut wie möglich zu bewältigen. Jedoch können Baumverfahren noch nicht vollständig dem Fluch der hohen Dimensionen trotzen und entarten bei sehr hohen Dimensionen zu einer Sequenziellen Suche [WSB98]. Deshalb schlagen Weber et al. ein Verfahren vor, dass die Sequentielle Suche beschleunigen soll, um damit den Fluch der hohen Dimensionen zu überwinden [WB97].

2.4.3 Optimierte Sequenzielle Suche

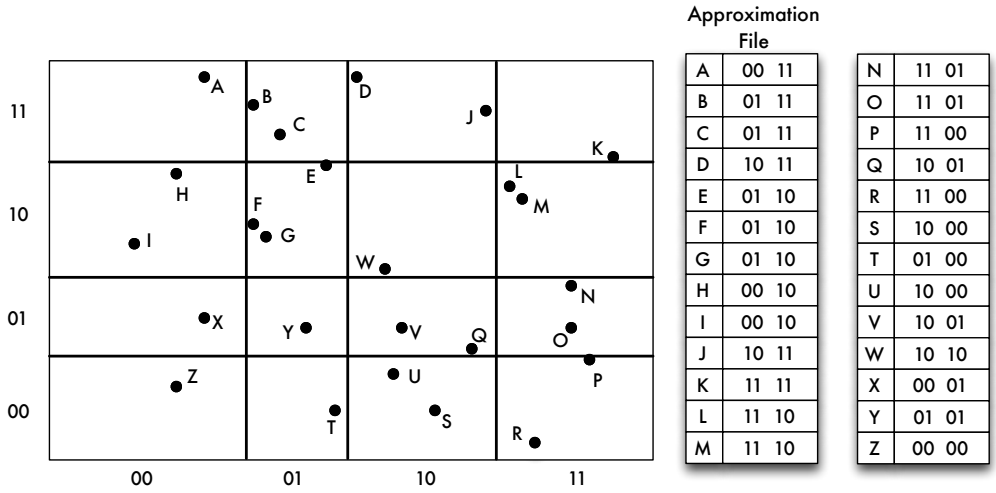
Die vorgestellten Baumverfahren sind ein guter Weg einen hochdimensionalen Datenraum zu indexieren. Jedoch leiden viele der vorgestellten Verfahren weiterhin unter dem Fluch der hohen Dimensionen. Zur Begründung dieses Sachverhalts führen Weber et al. verschiedene Beobachtungen an [WSB98]:

- In hochdimensionalen Räumen gibt es vielen "leeren Raum". Dies führt dazu, dass vorallem bei raumpartitionierenden Verfahren viele Regionen leer sind oder zumindest dünn besetzt und somit der Speicherplatz, den die jeweilige Region einnimmt, nicht effektiv genutzt werden kann.
- Bei der Suche in hohen Dimensionen mit raumpartitionierenden Verfahren müssen fast alle Regionen besucht werden.
- Sogar bei datenpartitionierenden Verfahren mit rechteckigen Regionen werden laut Weber et al. durchschnittlich alle Regionen durchsucht, um vorallem bei der Suche nach den nächsten Nachbarn alle Datenpunkte zu finden.
- Datenpartitionierende Verfahren mit spherischen Regionen leiden hingegen etwas weniger am Fluch der hohen Dimensionen. Nach den Beobachtungen in [WSB98] verlieren sie ihren Vorteil gegenüber einer sequenziellen Suche erst ab 45 Dimensionen, wohingegen Verfahren mit MBRs schon ab 26 Dimensionen zu einer sequenziellen Suche ausarten sollen.

Aufgrund der Beobachtung, dass sowohl raumpartitionierende als auch datenpartitionierende Verfahren ab einer bestimmten Dimensionszahl zu einer sequenziellen Suche ausarten [WSB98], wurde die Verbesserung der sequenziellen Suche in Betracht gezogen. Eine Datenstruktur, die die sequenzielle Suche verbessert, ist das *Vector Approximation File (oder VA-File)*. Das VA-File arbeitet mit einer Kompression und Approximation der realen Vektordaten, um somit bei Vergleichen einen Geschwindigkeitsvorteil zu erhalten.

Der Raum wird vom VA-File in 2^b rechteckige Zellen eingeteilt, dafür wird jede Dimension in 2^{b_i} gleichbefüllte Abschnitte partitioniert, die auch nach dem Einfügen neuer Punkte konstant bleiben. Jede Zelle hat somit einen eindeutigen Bitstring der Länge b, der sich durch die Aneinanderreihung der einzelnen Bits der Dimensionsabschnitte zusammensetzt [WB97]. Für jeden Punkt werden nun nicht nur die Vektordaten im *Vector File*, sondern auch der Bitstring der Zelle, in die er eingeordnet werden kann, im sogenannten *Approximation File* gespeichert. Die Raumaufteilung und das Vector sowie Approximation File wurden in Abbildung 2.11 visualisiert.

Bei einer Exact-Match-Anfrage wird dann die zugehörige Zelle des Anfragepunktes ermittelt und anhand dieser das ganze Approximation File durchsucht. Punkte, die zur gleichen Zelle gehören, werden dann als Kandidaten für einen exakten Abgleich der Vektordaten herausgefiltert. Hierbei kann zu Hilfe genommen werden, dass die Reihenfolge im Approximation File äquivalent zu der im Vector File ist. Wird zum Beispiel der Punkt F gesucht, so wird der Bitstring 0110 berechnet und somit beim Durchlaufen des Approximation Files die Kandidaten



A	00 11
B	01 11
C	01 11
D	10 11
E	01 10
F	01 10
G	01 10
H	00 10
I	00 10
J	10 11
K	11 11
L	11 10
M	11 10

N	11 01
O	11 01
P	11 00
Q	10 01
R	11 00
S	10 00
T	01 00
U	10 00
V	10 01
W	10 10
X	00 01
Y	01 01
Z	00 00

Abbildung 2.11: Aufbau des VA-Files

E,F,G gefunden. Danach werden die genauen Punktdaten des Anfragepunktes mit allen Kandidaten abgeglichen und schließlich der Datenpunkt F zurückgegeben.

Für die Suche der nächsten Nachbarn, wird wieder das ganze Approximation File durchsucht. Wird hierbei, aufgrund eines Eintrages im Approximation File, eine Zelle gefunden, die näher zur Zelle des Anfragepunktes liegt als eine der bisherigen gefundenen, so wird dieser Kandidat eingefügt. Weiterhin wird dafür der Kandidat, dessen Zelle am weitesten vom Anfragepunkt entfernt ist, aus der Kandidatenliste herausgenommen.

2.4.4 Space-filling Curves

Eine andere Möglichkeit den Raum einzuteilen stellen Space-filling Curves dar. Die Idee hinter Space-filling Curves ist, dass der hochdimensionale Raum anhand einer Funktion auf einen eindimensionalen Raum abgebildet wird. Die entstehende Kurve besucht jede Region eines n-dimensionalen Gitters ohne sich dabei zu überschneiden.

Eine Anforderung besteht darin sowohl Punktanfragen, als auch Suchen nach dem nächsten Nachbarn im hochdimensionalen Raum zu unterstützen. Dies setzt voraus, dass Nachbarschaftsbeziehungen erhalten bleiben. Dadurch müssen Punkte, die sich im Raum nah beieinander befinden, auch auf der entstehenden Kurve dicht beieinander abgebildet werden.

Z-Kurve

Mit dem Ziel die Nachbarschaftsbeziehungen möglichst gut zu erhalten, wurden verschiedene Space-filling Curves entwickelt und vorgestellt [BBBK00; BBK01; BM97; GG98]. Die meisten Space-filling Curves teilen den Raum rekursiv in einzelne Teile ein. Pro Iteration, den die Rekursion vollzieht, wird die Aufteilung

feingranularer und somit werden große Räume in immer kleinere Teilabschnitte unterteilt. Somit gibt es Kurven verschiedener *Ordnung* [VCPF08]. In Abbildung 2.12 wurde die Z-Kurve erster und zweiter Ordnung abgebildet.

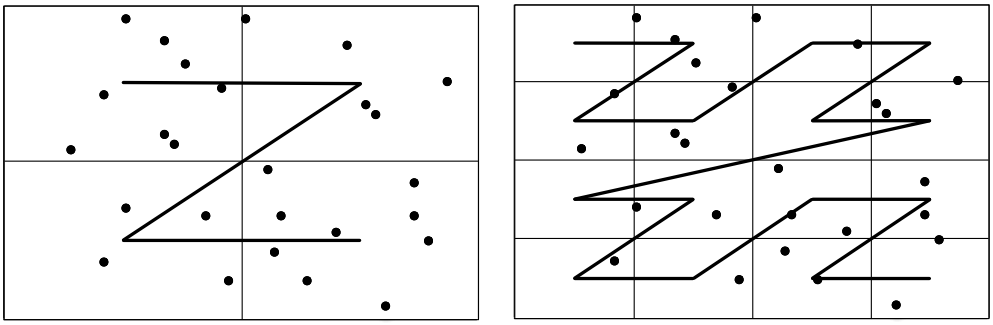


Abbildung 2.12: Z-Kurve erster und zweiter Ordnung

Die räumliche Ausdehnung der Z-Kurve entspricht dem Buchstaben "Z" und kann auch durch Bit-Interleaving erzeugt werden [OM84]. Aus der Abbildung 2.12 ist ersichtlich, dass die Z-Kurve zwar für die Bereiche in einem Z nachbarschaftserhaltend ist, jedoch wird dies nicht automatisch für benachbarte Regionen unterschiedlicher Zs gewährleistet. Dieser Fall muss bei der Suche nach den nächsten Nachbarn beachtet werden.

Hilbert Kurve

Die großen Sprünge in der Z-Kurve behindern laut Faloutsos et al. die Anfragebearbeitung, weshalb sie die Hilbert Kurve vorschlagen [FR89]. Der Aufbau der Hilbert Kurve wurde in Abbildung 2.13 veranschaulicht.

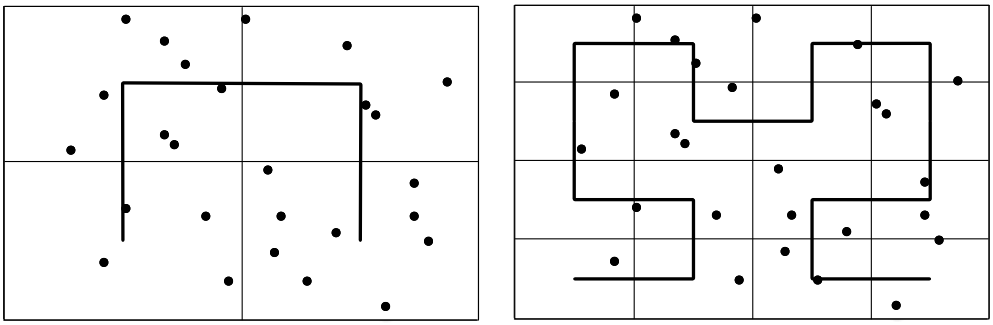


Abbildung 2.13: Hilbert Kurve erster und zweiter Ordnung

Die Konstruktion der Hilbert Kurve ist dabei etwas aufwändiger als die der Z-Kurve. Um höhere Ordnungen der Hilbert Kurve zu erhalten, muss die Kurve reflektiert und rotiert werden. Dafür ist die Erhaltung von Nachbarschaftsbeziehungen besser gewährleistet als bei der Z-Kurve [FR89].

Space-filling Curves werden in verschiedenen Indexstrukturen zu Hilfe genommen [Bay97; TYSK10]. Ein Vertreter ist der *UB-Baum (engl. univeral B-Tree)*. Der UB-Baum ist ein B-Baum, dessen Schlüssel dem Wert der Space-filling Curve für den jeweiligen Datenpunkt entspricht. Beim Einfügen muss also zuerst die Zelle, in die er fällt bestimmt werden und wird dann mit diesem Wert als Schlüssel in den B-Baum eingefügt. Weitere Verfahren sind unter anderem durch die Hinzunahme von Hashverfahren, wie LSH aus dem folgenden Kapitel, realisert und als LSB-Bäume vorgestellt worden [TYSK10].

Der generelle Vorteil von Space-filling Curves ist, dass sie den hochdimensionalen Raum in einen eindimensionalen Raum überführen und somit jede eindimensionale Indexstruktur zum Indexieren genutzt werden kann. Aufgrund der Kompression der Dimensionen gehen jedoch Nachbarschaftsbeziehungen verloren und für die Suche nach den nächsten Nachbarn müssen spezielle Vorkehrungen getroffen werden.

2.4.5 Hashverfahren

Hashverfahren werden schon seit einigen Jahren in Datenbankenmanagementsystemen angewandt [GG98]. Sie versuchen anhand einer Hashfunktion die Datenpunkte den sogenannte Buckets zu zuteilen, um eine Anfragegeschwindigkeit von $O(1)$ zu erzielen. Dabei soll eine ausgeglichene Verteilung über die vorhandenen Buckets erreicht werden, weil bei Anfragen das jeweilige Bucket sequenziell durchsucht wird [SKS01]. Deshalb ist die Aufgabe dieser Hashfunktionen für niedrige Dimensionen die Daten möglichst gut zu streuen. Das heißt, dass zwei Datenpunkte mit sehr ähnlichen Attributsausprägungen einen völlig unterschiedlichen Hashwert besitzen sollen.

Locality Sensitive Hashing

Neben den eindimensionalen Hashverfahren wie lineares und dynamisches Hashen [SKS01] gibt es auch einige mehrdimensionale Hashverfahren [GG98; SSH11]. Diese mehrdimensionalen Verfahren lassen sich ebenso für hochdimensionale Daten verwenden, jedoch haben sie einen Nachteil bei der Suche der nächsten Nachbarn. Aufgrund der Streuung der Hashfunktionen werden zumeist die Nachbarschaftsbeziehungen zerstört, sodass bei kNN-Anfragen fast alle Buckets durchsucht werden müssen, um die exakten nächsten Nachbarn zu finden. Deshalb wurde *Locality Sensitive Hashing (kurz LSH)* vorgestellt [IM98].

Im Gegensatz zu bekannten Hashfunktionen, die ähnliche Datensätze über die vorhandenen Buckets verstreuen, bewirkt LSH bei ähnlichen Datensätzen, dass diese Punkte in das gleiche Bucket eingeordnet werden. Dadurch werden kNN-Anfragen beschleunigt. Bei LSH ist die Suche der nächsten Nachbarn jedoch nicht exakt, sondern nur approximativ, weil nicht alle Buckets durchsucht werden. Deshalb wird zumeist die Anforderung der kNN-Anfrage aufgeweicht. Es werden dadurch nicht die nächsten Nachbarn ausgegeben, sondern Punkte, die dem Anfragepunkt sehr ähnlich sind.

Zur Verbesserung der Güte der Ergebnisse, wird beim Locality Sensitive Hashing nicht nur eine Hashfunktion verwendet, sondern gleich mehrere sogenannte Hashtabellen angelegt, für die unterschiedliche Hashfunktionen gelten [IM98]. Die gewählten Hashfunktionen entstammen aus einer Funktionsfamilie $\mathcal{H}$, für die gilt, dass jede Funktion h $(P1, P2, r, cr)$-sensitiv ist [IM98]. Diese Bedingung ist wie folgt definiert:

> Für alle Datensätze in unserem d-dimensionalem Raum $p, q \in \mathcal{R}^d$ gilt:
>
> 1. *wenn* $||p - q|| < r$, *dann* $Pr[h(p) = h(q)] > P1$
> 2. *wenn* $||p - q|| > cr$, *dann* $Pr[h(p) = h(q)] < P2$

Hierbei stellt Pr die Wahrscheinlichkeitsfunktion dar, sodass $Pr[h(p) = h(q)]$ die Wahrscheinlichkeit angibt, für die die Datenpunkte das gleiche Hashbucket zugewiesen bekommen. In der vorgestellten Formel geht es primär um den Abstand der beiden Punkte aufgrund der Distanzmetrik. Ist dieser Abstand kleiner als r, so soll die Wahrscheinlichkeit, dass die Punkte in das gleiche Bucket eingeordnet werden, größer als $P1$ sein. Dementgegen sollen Punkte, deren Abstand größer als cr ist, mit einer Wahrscheinlichkeit kleiner als $P2$ in das gleiche Bucket einsortiert werden. Dabei sollte die Wahrscheinlichkeit $P1$ größer als $P2$ sein, sonst arbeiten die gewählten Funktionen nicht lokal sensitiv.

Klassen von LSH-Verfahren

Die Herausforderung stellt hierbei jedoch die Definition passender $(P1, P2, r, cr)$-sensitiver Funktionen dar. Eine dieser Funktionsfamilien kann aus p-stabilen Verteilungen gewonnen werden [DIIM04].

Die genutzte Eigenschaft von p-stabilen Verteilungen ist, dass die Summe von unabhängigen, identisch verteilten Zufallsvariablen aus einer p-stabilen Verteilung genauso verteilt ist wie eine andere Variable aus einer p-stabile Verteilung. Formal bedeutet das, dass für ein $p \geq 0$ die Verteilung $\mathcal{D}$ p-stabil ist, wenn für n reelle Zahlen $v_1, ..., v_n$ und unabhängig, identisch verteilte Zufallsvariablen $X_1, ..., X_n$ folgendes gilt:

> $\sum_{i=1}^{n}(v_i X_i) \sim (\sum_{i=1}^{n}(\|v_i\|^p))^{1/p} X$, wobei $\sim$ gleich verteilt bedeutet und X eine Zufallsvariable aus der Verteilung $\mathcal{D}$ ist [Nol09].

Wird nun ein d-dimensionaler Vektor $\vec{a}$ mit Werten aus einer p-stabilen Verteilung gewählt, so ist das Skalarprodukt des Vektors $\vec{a}$ mit einem Datenpunkt $\vec{v}$ eine Zufallsvariable, die $(\sum_{i=1}^{d}(\|v_i\|^p))^{1/p} X$ verteilt ist. Mehrere dieser Skalarprodukte mit verschiedenen Vektoren $\vec{a}$ ergibt den Aufriss des Punktes $\vec{v}$ und kann benutzt werden um $\|\vec{v}\|_p$ abzuschätzen, also den Abstand vom Nullpunkt unter der jeweiligen L_p-Norm. Um die, im Abschnitt 2.3.1 vorgestellten, L_p-Norm anzunähern, können Vektoren mit Werten aus der Cauchy Verteilung für die L_1-Norm und der Gauss (Normal) Verteilung für die L_2-Norm entnommen werden. Diese Verteilungen sind wie folgt definiert [DIIM04]:

- Die Cauchy Verteilung $\mathcal{D}_{\mathcal{C}}(0,1)$, mit der Dichtefunktion $c(x) = \frac{1}{(\pi(1+x^2))}$ ist 1-stabil.
- Die Gauss (Normal) Verteilung $\mathcal{D}_{\mathcal{G}}(0,1)$, mit der Dichtefunktion $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ ist 2-stabil.

Das gezeigte Skalarprodukt kann für LSH genutzt werden, indem für jede Hashtabelle ein Vektor $\vec{a}$ mit Werten aus p-stabilen Verteilungen festgelegt wird. Dann wird für jeden Datenpunkt $\vec{v}$ und für jede Tabelle i das Skalarprodukt $\langle \vec{a}_i, \vec{v} \rangle$ berechnet. Diese Wertberechnung bildet den hochdimensionalen Datenpunkt auf einen eindimensionalen Raum ab [DIIM04]. Wird dieser dann in kleine Bereiche eingeteilt, so repräsentieren diese Bereiche die einzelnen Buckets, in die die Datenpunkte eingeordnet werden können.

Der Vorteil des Skalarprodukts $\langle \vec{a}, \vec{v} \rangle$ ist hierbei, dass die Distanz der Projektion zweier Vektoren $(\langle \vec{a}, \vec{v}_1 \rangle - \langle \vec{a}, \vec{v}_2 \rangle)$ genauso verteilt ist, wie $\|\vec{v}\|_p X$. Somit ist gewährleistet, dass Punkte, die nah beieinander liegen auch mit hoher Wahrscheinlichkeit in das gleiche Bucket eingeordnet werden.

Permutationsansatz

Eine Möglichkeit die Definition einer Funktionsfamilie $\mathcal{H}$ zu umgehen ist der Permutationsansatz [CGFN08]. Hierbei wird aus den Datenpunkten eine bestimmte Anzahl an zufällig gewählten Prototypen ausgesucht. Dann wird für jeden Datenpunkt der Abstand zu jedem Prototyp berechnet und damit die Prototypen mit aufsteigender Distanz sortiert. Die entstehende Reihenfolge der Prototypen wird dann als Hashwert benutzt. Diese Raumaufteilung kann in Abbildung 2.14 noch einmal nachempfunden werden. Die Prototypen P1 bis P3 spannen hier 6 verschiedene Teilräume auf, die mit dem jeweiligen Hashwert beschriftet wurden.

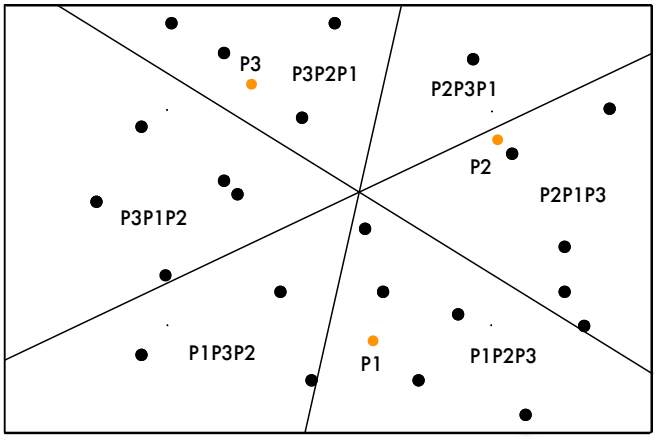


Abbildung 2.14: Raumaufteilung durch den Permutationsansatz

Für die exakte Suche mit dem Permutationsansatz wird nun zuerst für den Anfragepunkt die entsprechende Permutation anhand der Abstände zu den Prototypen bestimmt. Dadurch ergibt sich das zu durchsuchende Bucket, in dem

der Datenpunkt enthalten sein muss, wenn er existiert. Dieses Bucket wird dann sequenziell durchsucht. Die Suche nach den nächsten Nachbarn ist dem entsprechend etwas aufwändiger, da es möglich ist, dass das Zielbucket weniger als k Einträge umfasst. Werden zum Beispiel die zwei nächsten Nachbarn des einzigen Punktes in Bucket P2P3P1 gesucht (Abbildung 2.14), so müssen auch die umliegenden Buckets durchsucht werden.

Der Permutationsansatz hilft die Nutzung von Hashes auch im hochdimensionalen Raum zu ermöglichen. Jedoch verschiebt sich hierbei die Herausforderung vom Finden einer passenden Hashfunktion zur richtigen Wahl der Prototypen. In [CGFN08] wird vorgeschlagen die Prototypen zufällig aus den Datenpunkten zu wählen. Dadurch passt sich die Wahl an die vorhandene Datenverteilung an. Jedoch ist die entstehende Raumaufteilung nicht trivial. Deshalb soll in dieser Arbeit die Wahl der richtigen Punkte im Raum als Kandidaten für einen Prototyp untersucht werden.

2.5 Zusammenfassung

Ziel dieses Kapitels war es einen Einblick in die Daten und Anfragetypen einer Datenbank zu geben und zu beschreiben, wie es Indexstrukturen ermöglichen den Zugriff auf die Daten zu beschleunigen. Sind die zu speichernden Daten nicht nur ein- bis mehrdimensional sondern sogar hochdimensional, so haben die Indexstrukturen weitere Herausforderungen zu bewältigen, wie die Wahl der passenden Distanzfunktion und der Fluch der hohen Dimensionen. Nach einer Einführung von hochdimensionalen Indexstrukturen und einer kurzen Charakterisierung dieser in approximative und exakte Verfahren sowie raumpartitionierende und datenpartitionierende Verfahren wurde eine Klasseneinteilung der Indexstrukturen vorgestellt. Die Indexstrukturen gliedern sich in Baumverfahren, Optimierte Sequenzielle Suche, Space-filling Curves und Hashverfahren, wobei für jede Klasse verschiedene Vertreter von Indexstrukturen vor- und gegenübergestellt wurden. Als Hashverfahren bietet der prototypbasierte Ansatz einen interessanten Vertreter, wobei die Raumaufteilung anhand der Prototypen noch nicht vollständig ergründet wurde. Deshalb beschäftigt sich das nächste Kapitel mit dem Setzten der Prototypen in den Raum und die damit verbundenen Herausforderungen.

Kapitel 3

3 Motivation

Im vorherigen Kapitel wurden anschaulich die Raumaufteilungen und Eigenschaften spezieller hochdimensionaler Indexstrukturen vorgestellt. Dabei wurde auch die Raumaufteilung der permutationsbasierten Variante des Indexverfahrens LSH verdeutlicht. Leider ist die entstehende Raumaufteilung aufgrund der Verteilung der Prototypen im Raum derzeit nicht eindeutig nachvollziehbar. In diesem Kapitel werden die Schwierigkeiten beim Setzen der Prototypen aufgezeigt und damit herausgestellt, dass eine nähere Betrachtung dieses Themas sinnvoll ist.

3.1 Raumaufteilung durch Prototypen

Der Algorithmus aus [CGFN08] setzt die Prototypen anhand der Datenverteilung. Dafür werden aus der Menge der Datenpunkte zufällig Prototypen ausgewählt. Dadurch richtet sich die Verteilung der Prototypen nach der Verteilung der Datenpunkte im Raum. Im zweidimensionalen Raum entsteht pro Prototyenpaar eine Gerade, die den Raum aufteilt. In höheren Dimensionen wird diese Gerade zu einer Hyperebene, dessen Dimensionalität um eins kleiner ist als die Dimensionalität des Raumes. Vereinfacht wird im Folgenden der zweidimensionale Raum betrachtet, wobei die Ausführungen auch auf höhere Dimensionen übertragbar sind. Die entstehende Gerade zweier Prototypen liegt in der Mitte der Verbindungslinie zwischen den beiden Prototypen und verläuft senkrecht zu dieser Verbindungslinie im Raum. In Abbildung 3.1 wurde dies im zweidimensionalen Datenraum mit 150 Punkten dargestellt.

Es wurden zufällig zwei Prototypen P1 und P2 aus dem Datenraum ausgewählt. Einer dieser Punkte befindet sich sogar im vorhandenen Cluster im unteren rechten Teil des Bildes. Weiterhin wurde die Verbindungslinie der beiden Prototypen gestrichelt angedeutet und die dazu senkrecht im Raum verlaufende Gerade dargestellt. Sie teilt den Raum in der Mitte von P1 und P2 in zwei Regionen. Alle Punkte auf der rechten Seite dieser Gerade befinden sich näher zu P1 als zu P2 und werden beim Indexieren mit dem Hashwert P1P2 versehen. Punkte, die sich auf der linken Raumhälfte der Gerade befinden, werden in das Hashbucket mit dem Wert P2P1 eingeordnet. Bei der dargestellten Aufteilung aus Abbildung 3.1 werden die Hashbuckets jedoch relativ ungleichmäßig befüllt. Das

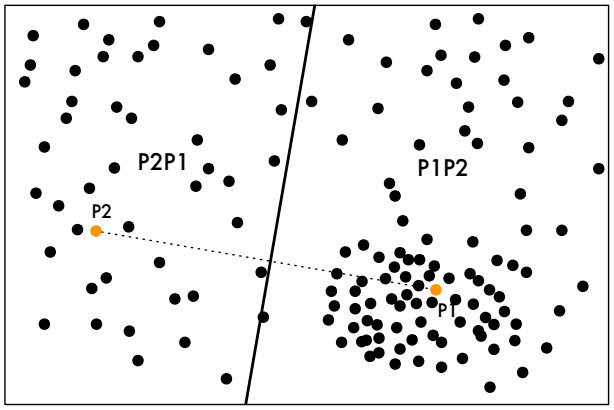


Abbildung 3.1: Raumaufteilung durch den Permutationsansatz

Bucket mit dem Hashwert P1P2 enthält 102 Datenpunkte und das zweite Bucket dementsprechend 48 Datenpunkte. Werden nun weiterhin zufällig Datenpunkte als Prototypen hinzugenommen, so kann die Raumaufteilung weiter entarten.

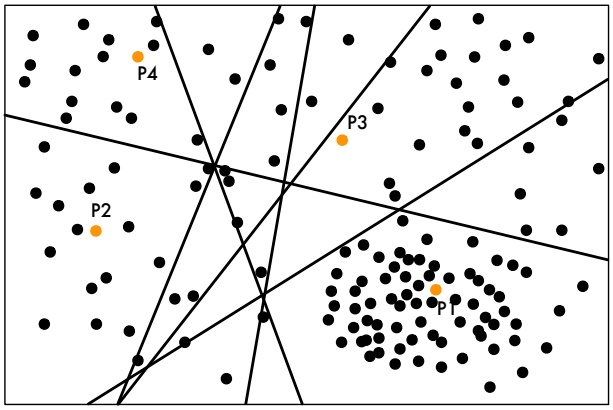


Abbildung 3.2: Beispielaufteilung mit vier Prototypen

In Abbildung 3.2 wurden zwei weitere Punkte als Prototypen gewählt und die entstehende Raumverteilung dargestellt. Es ist zu erkennen, dass immer noch ein Großteil der Punkte in der unteren rechten Region liegt, wobei es sogar leere Regionen gibt. Dieses Beispiel zeigt, dass die zufällige Wahl an Prototypen zwar den Raum weiter aufteilt, jedoch ist die Wahrscheinlichkeit groß, dass die feingranularen Regionen an den falschen Stellen des Raumes sind. Sinnvoll wäre es bei diesem Beispiel vorallem das Cluster weiter zu unterteilen, da hier eine große Anzahl an Datenpunkten gehäuft auftreten und dadurch eine feingranulare Einteilung die Performanz von Exact-Match-Anfragen steigern würde. Somit ist die zufällige Wahl von Prototypen nicht immer zielführend und bedarf Verbesserungen. Im nächsten Abschnitt wird deshalb darauf eingegangen, ob durch einfache Operationen, wie die Wahl eines anderen Punktes als Prototypen oder das Hinzufügen eines weiteren Prototypens eine Verbesserung der Raumaufteilung bewirkt.

3.2 Einfache Verbesserungen

Wie schon erwähnt, entsteht für jedes Prototypenpaar eine Gerade im Raum. Dadurch beeinflusst ein Prototyp $|p| - 1$ Geraden, die den Raum teilen. Aufgrund dieser Annahem wäre es denkbar, dass beim Verschieben eines Prototypen die Raumaufteilung verbessert werden kann. Dieses Verhalten wurde in Abbildung 3.3 dargestellt.

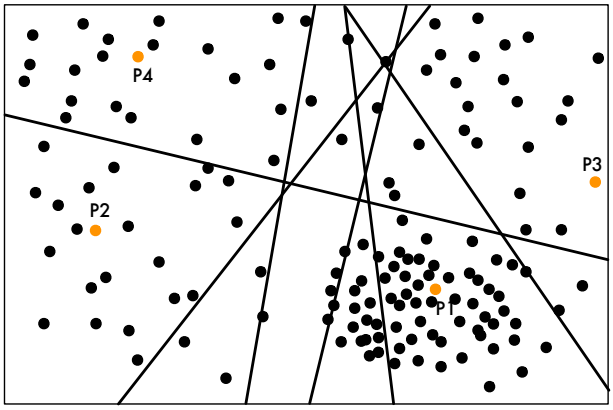


Abbildung 3.3: Umsetzen vom Prototyp P3

Hierfür wurde für den Prototyp P3 ein anderer Punkt, der weiter am rechten Rand des Raumes liegt, ausgewählt. Dadurch verschieben sich drei der Geraden im Raum in Richting P3, wodurch das Cluster besser unterteilt wird, wobei sich jedoch immer noch ein großer Teil der Datenpunkte im Bucket befindet, in dem auch P1 enthalten ist. Zu sehen ist hierbei weiterhin, dass der feingranular unterteilte Bereich am oberen Rand des Raumes liegt und nur sehr wenige Datenpunkte unterteilt. Dieser Bereich ist durch die Veränderung noch nicht in die Mitte des Clusters gelangt.

Eine andere Möglichkeit die Raumaufteilung durch wenige Veränderungen zu verbessern, könnte darin liegen einfach einen weiteren Prototypen in den Raum zu setzen. Dadurch entstehen weitere Geraden, die den Raum besser einteilen können und somit das Cluster feingranularer unterteilen kann.

In der Abbildung 3.4 wurde ein weiterer zufällig gewählter Prototyp eingefügt. Dieser befindet sich in dem vorhandenen Cluster. Dadurch entsteht eine Gerade zwischen P5 und P1, die das Cluster unterteilt. Die restlichen drei entstandenen Gerade tragen jedoch nicht dazu bei, die Indexierung des Clusters zu verbessern. Dementgegen werden die feingranularen Einteilungen in der Mitte nur noch weiter verfeinert, sodass im mittleren Bereich des Raumes eine große Anzahl an leeren Buckets entsteht.

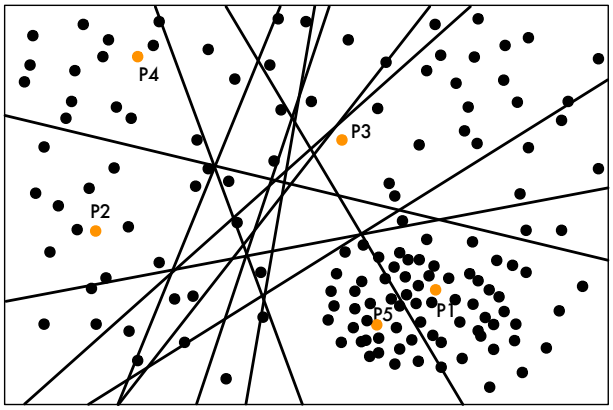


Abbildung 3.4: Hinzufügen eines Prototypen

3.3 Gewähltes Einfügen

Wie im vorherigen Abschnitt gezeigt wurde, ist das richtige Setzen der Prototypen nicht trivial und kann nicht immer effektiv durch einfache Schritte verbessert werden. Deshalb ist es wichtig, von vornherein die Prototypen an günstige Stellen des Raumes zu platzieren, um zum Beispiel Cluster gut zu unterteilen. Dafür wurde in Abbildung 3.5 von Anfang an darauf geachtet, dass die entstehende Raumaufteilung das Cluster besser unterteilt.

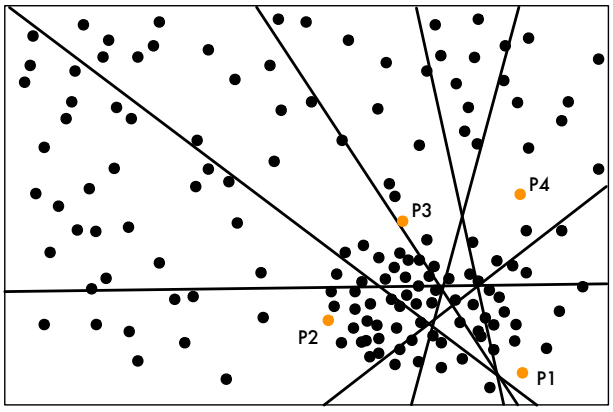


Abbildung 3.5: Verbesserte Raumaufteilung durch intelligentes Setzen

Wie zu sehen ist, liegen dabei die meisten Schnittpunkte der Geraden in der Mitte des Clusters. Dadurch wird zwar das Cluster gut unterteilt, doch die Bereiche außerhalb des Clusters können trotzdem eine große Anzahl an Datenpunkten beinhalten. Das Einfügen eines weiteren Punktes könnte hierbei nützlich sein. Eine gleichmäßige Raumaufteilung ist vorallem für Exact-Match-Anfragen von Vorteil, weil bei der Suche im Durchschnitt immer die gleiche Anzahl an Punkten durchsucht werden muss. Für Anfragetypen wie Bereichsanfragen könnte eine andere Raumaufteilung vorteilhaft sein, denn hierbei werden anhand betimmter

Grenzen die Punkte herausgesucht. Dadurch kann es von Nutzen sein, wenn der Raum möglichst orthogonal zu den Raumgrenzen aufgeteilt wird. Dies bewirkt, dass weniger Buckets die Anfrageregion überschneiden und somit das unnötige Durchsuchen von einigen Buckets erspart werden kann.

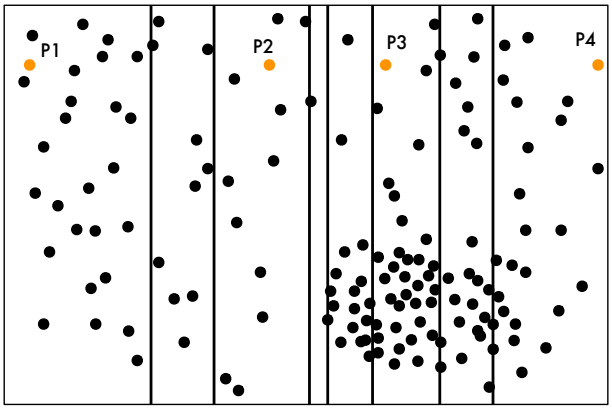


Abbildung 3.6: Verbesserungen für Bereichsanfragen

In Abbildung 3.6 wurden die Prototypen so gewählt, dass sie fast exakt auf einer Linie liegen. Dadurch wird der Raum orthogonal zu einer der beiden Dimensionen eingeteilt. Um die andere Dimensionen für Bereichsanfragen zu unterstützen, müssen jedoch weitere Punkte eingefügt werden.

3.4 Fazit

Wie in diesem Kapitel gezeigt wurde, führt das zufällige Setzen von Prototypen anhand der Datenverteilung nicht zwangsläufig zum Erfolg. So kann es passieren, dass kleine Bereiche entstehen, die wenige Datenpunkte enthalten, wie in Abbildung 3.4 gezeigt wurde. Deshalb ist ein intelligentes Setzen der Prototypen von großer Wichtigkeit. Jedoch ist die Betrachtung der Raumaufteilung durch mehrere, von Hand erstellte, Abbildung nicht zielführend, da es mit viel Aufwand und Ausprobieren verbunden ist, wenn die entstehende Raumaufteilung optimal sein soll. Deshalb soll im folgenden Kapitel die Anforderungen an ein Tool vorgestellt werden, das es ermöglicht die Darstellung der Raumaufteilungen zu visualisieren.

Kapitel 4

4 Anforderungsanalyse

Im letzten Kapitel wurde die Raumaufteilung des Permutationsansatzes anhand ausgewählter Prototypenverteilungen dargestellt, jedoch ist die Untersuchung der Raumaufteilung ohne eine toolgestützte Darstellung der entstehenden Regionen sehr mühsam und aufwendig. Deshalb wird ein Tool entwickelt, dass die Aufteilung des Raumes anhand der gewählten Prototypen darstellt. In diesem Kapitel wird darauf eingegangen, welches Ziel verfolgt wird und welche Anforderungen das Tool erfüllen muss. Dafür werden die Anforderungen an den darzustellenden Raum, die Manipulation der Prototypen, die Darstellung der Raumaufteilung und das Einbinden von Beispieldaten formuliert.

4.1 Darzustellender Raum

Das entstehende Tool soll dazu dienen, die Raumaufteilung durch den Permutationsansatz darzustellen. Dafür werden bestimmte Anforderungen an den Datenraum gestellt, die in der Tabelle 4.1 zusammengefasst wurden.

Anforderung	**Erläuterung**
2-D	Punkte und Prototypen sollen im zweidimensionalen Raum angeordnet werden können und die Raumaufteilung visualisiert werden.
3-D	Eine Visualisierung im dreidimensionalen Raum wäre von Vorteil, weil dadurch Schlüsse über das Verhalten der Indexstruktur bei steigenden Dimensionen gezogen werden können.
Zoom	Mit dem Ziel einer besseren Übersichtlichkeit, sollte es möglich sein in den Raum hineinzuzoomen, damit auch kleine Regionen oder dicht besetzte Datenräume besser untersucht werden können.
Raster	Ein weiterer Belang stellt die dezente Darstellung eines Gitters dar. Dies dient der verbesserten Orientierung im Raum und ist vorallem beim Zoomen in den Raum von Nutzen.

Tabelle 4.1: Durch das Tool darzustellender Datenraum

Die hier genannten Anforderungen legen die räumliche Darstellung des Datenraums fest und geben Vorgaben für das Aussehen des Tools. Das Setzen und Verschieben der Prototypen wird im nächsten Abschnitt beschrieben und einzelne Anforderungen herausgearbeitet.

4.2 Manipulation der Prototypen

Zur Darstellung der Raumaufteilung anhand von Prototypen, müssen diese individuell in den Raum gesetzt werden können. Das Setzen soll sowohl durch Mausklicks in den Datenraum, als auch durch Angeben von einzelnen Koordinaten ermöglicht werden. Die sich ergebenden Anforderungen wurden in Tabelle 4.2 aufgelistet und erläutert.

Anforderung	**Erläuterung**
Setzen von Prototypen	Durch einfaches Klicken mit der linken Maustaste in den Datenraum soll ein Prototyp an der ausgewählten Stelle gesetzt werden.
Verschieben eines Prototyps • per Maus • per Koordinaten • per Pfeiltasten	Ein Prototyp soll im Datenraum versetzt werden können. Dies soll sowohl per Maus durch verschieben des Punktes im Raum, als auch per Eingabe der neuen Koordinaten erfolgen können. Die Koordinaten der einzelnen Prototypen sollen dafür separat angezeigt werden. Außerdem soll es möglich sein einen markierten Prototyp per Pfeiltasten im Raum zu verschieben. Dadurch kann interaktiv verfolgt werden, wie sich die Regionenaufteilungen verändern und welche Geraden mit dem verschobenen Prototyp in Beziehung stehen.
Löschen eines Prototyps • per Maus • per Schaltfläche	Eine weitere Anforderung stellt das Löschen eines Prototypens dar. Es sollen Prototypen einerseits durch einen Klick mit der rechten Maustaste auf den Prototypen entfernt werden können, andererseits sollte auf der Benutzeroberfläche eine Schaltfläche vorhanden sein, die den zur Zeit ausgewählten Prototypen löscht. Dadurch können dünn besetze Bereiche zu größeren Regionen wieder verschmolzen werden.
Kennzeichnung der Prototypen	Mit dem Ziel der Unterscheidung der Prototypen von den anderen Datenpunkten, sollen die Protoypen anders eingefärbt werden. Weiterhin soll immer der jeweils angewählte Prototyp eine andere Färbung erhalten, als die restlichen Prototypen, damit der verschobene Prototyp besser von den anderen Prototypen zu unterscheiden ist.

Tabelle 4.2: Anforderungen an die möglichen Manipulationen von Prototypen

Nach der Umsetzung der hier beschriebenen Arbeitspaketen, ist ein Tool entstanden, mit dem Prototypen in einen Datenraum gesetzt werden können und deren Lage verändert werden kann. Beim Setzen neuer Prototypen wäre es praktisch, sofort die entstehenden Regionenaufteilungen darzustellen. Mit den Anforderungen an diese Visualisierung beschäftigt sich der nächste Abschnitt.

4.3 Darstellung der Raumaufteilung

Ein Ziel stellt die Visualisierung der entstehenden Raumaufteilung anhand der Prototypen dar. Ein weiteres Ziel wurde aufgrund folgender Beobachtung gesetzt. Bei der Betrachtung der Raumaufteilung durch die Prototypen ist auffällig, dass diese den Voronoi-Diagrammen ähnlich ist, wobei hierbei die Linien beim Aufeinandertreffen nicht weiter gezeichnet werden würden. Eine Voronoi-Zelle beschreibt somit den Raum, dessen Punkte am nächsten zu einem der Prototypen ist. Deshalb soll das Tool ermöglichen den Hashwert, also die Permutation der Prototypen, Schritt für Schritt verkürzen zu können. In der Abbildung 4.1 ist die Raumaufteilung bei drei gewählten Prototypen zu sehen. Es entstehen dabei 6 Regionen mit unterschiedlichem Hashwert. Wird nun der Hashwert um eine Stelle verkürzt, so gibt es immer noch sechs verschiedene Hashbuckets, da sich weiterhin die Regionenbezeichnungen unterscheiden. Wird jedoch nur die erste Stelle des Hashes betrachtet, so haben wir jeweils zwei Regionen mit dem gleichen Hashwert, sodass nur noch die farbig hinterlegten Bereiche übrig bleiben und die Hashbuckets bilden.

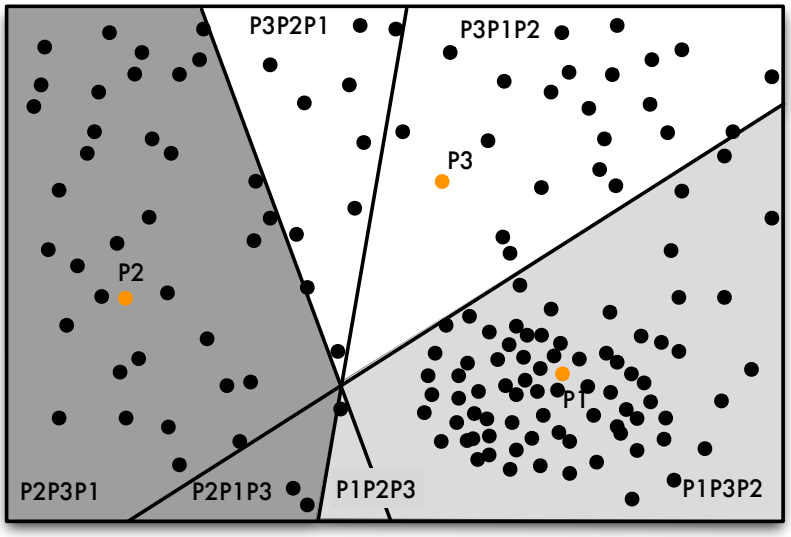


Abbildung 4.1: Voronoi Zellen der Prototypen

Für die Darstellung der Raumaufteilung wurden drei Anforderungen definiert. Diese sind in Tabelle 4.3 zusammengefasst worden.

Anforderung	Erläuterung
Darstellung der Geraden	Durch Einfügen eines Prototyps sollen automatisch die entstehenden Geraden im Datenraum gezeichnet werden. Weiterhin sollen sie automatisch verändert werden, wenn sich Prototypen in Anzahl oder Position ändern.
Darstellung des Hashwertes	Eine weitere Anforderung soll die Möglichkeit darstellen, die einzelnen entstandenen Regionen mit ihrem Hashwert zu beschriften. Dadurch kann nachvollzogen werden, welche Regionen benachbart sind und der Zusammenhang zwischen Lage der Prototypen und Beschriftung der Regionen verdeutlicht werden.
Verkürzung des Hashwertes	Weiterhin soll es möglich sein in der Benutzeroberfläche die Länge des Hashwertes zu bestimmen. Bei Veränderungen der Länge sollen automatisch die Beschriftungen der Regionen angepasst werden.

Tabelle 4.3: Anforderungen an die Darstellung der Raumaufteilung

Nach Abarbeitung der hier genannten Anforderungen kann die Raumaufteilung automatisch dargestellt werden. Um nun Aussagen über die Güte der Raumaufteilung treffen zu können, muss das entstehende Tool Datenbasen laden und im Raum darstellen können. Anhand dieser soll dann die Raumaufteilung bewertet werden. Die zu bewältigende Aufgaben, die das Einbinden der Beispieldaten ermöglichen, werden im nächsten Abschnitt erläutert.

4.4 Einbinden von Beispieldaten

Das entstehende Tool soll mit dem Framework QuEval[1] harmonieren und die Datenbasen, die vom Datengenerator aus QuEval erzeugt werden, sollen in den Datenraum geladen werden können. Der Datengenerator kann sowohl Datenbasen mit unterschiedlichen Verteilungen als auch geclusterte Daten erzeugen. Weiterhin kann die Anzahl der Datenpunkte eingestellt werden und die Anzahl an Dimensionen, die die Datensätze besitzen. Ein weiterer Vorteil des Datengenerators ist, dass der Datenbereich eingestellt werden kann, dadurch können die Datensätze an den Datenbereich anpassen werden, der für das Tool gewählt wurde. Anhand der eingefügten Punkte kann dann die resultierende Raumaufteilung besser analysiert und die Bucketauslastung visualisiert werden. Dadurch kann analysiert werden, unter welchen Umständen viele leere Buckets entstehen und wie die Prototypen zu setzen sind, um dem entgegen zu wirken. Auch die Veränderung der Bucketauslastung beim Verschieben eines Prototypens soll angezeigt werden können. Eine Auflistung und Erläuterung der hier genannten Anforderungen beinhaltet Tabelle 4.4.

[1] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/iJudge/index_en.php

Anforderung	Erläuterung
Einfügen von Beispieldaten	Das Framework QuEval bietet einen komfortablen Datengenerator. Die erzeugten Datenbasen sollen in den Datenraum geladen werden können und als Datenpunkte angezeigt werden. Dadurch ist es möglich Auswertungen der Raumaufteilung anhand verschiedener Datenbasen vorzunehmen.
Visualisierung der Bucketauslastung	Die Bucketauslastung soll anhand der geladenen Datenbasis und der Raumaufteilung visualisiert werden. Dabei soll anhand eines Balkendiagramms gezeigt werden, wie viele Buckets eine bestimmte Anzahl an Datenpunkten enthält. Diese Darstellung hilft vorallem herauszufinden, wie viele Buckets wenige Datenpunkte enthalten und wie viele Buckets überfüllt sind. Dabei wird natürlich eine gleichmäßige Bucketauslastung angestrebt, um Performanzeinbuße zu minimieren.

Tabelle 4.4: Einfügen von Beispieldatenbasen

Werden die Anforderungen dieses Kapitels umgesetzt, so wurde ein Tool entwickelt mit dem einfach und komfortabel die Raumaufteilung und Bucketauslastung visualisiert werden kann. Im nun folgendem Kapitel soll die Umsetzung dieser Anforderungen anhand von Codebeispielen gezeigt werden und das entstehende Tool vorgestellt werden.

4.5 Zusammenfassung

Dieses Kapitel setzt sich mit den Anforderungen an ein Tool zur Visualisierung der Raumaufteilung von prototypbasierten Indexverfahren wie LSH auseinander. Dafür wurde als Anforderung zu erst auf den darzustellenden Raum eingegangen. Dieser soll sowohl zwei- als auch dreidimensional sein und in diesen Raum sollen Prototypen einfach durch Mausklicks in den Raum gesetzt werden können und die Raumaufteilung visualisiert werden. Weiterhin soll beim weiteren Hinzufügen, Entfernen oder Verschieben von Prototypen die Raumaufteilung angepasst werden. Schließlich soll auch eine Verkürzung des Hashwertes und das Laden von Datenbasen inklusive der Darstellung der Bucketauslastung ermöglicht werden. Im nun folgendem Kapitel soll die Umsetzung dieser Anforderungen anhand von Codebeispielen gezeigt werden und das entstehende Tool vorgestellt werden.

Kapitel 5

5 Implementierung

Ziel dieses Kapitels ist die Beschreibung der Implementierung des Tools. Dabei soll exemplarisch aufgezeigt werden, wie die Anforderungen, die im letzten Kapitel beschrieben wurden, umgesetzt worden sind. Die Umsetzung erfolgte mit der Programmiersprache Java als universelle Sprache, dessen Programme plattformunabhängig sind. Ein weiterer Entscheidungsgrund für Java liegt in der guten Dokumentation und der Verfügbarkeit vieler Bibliotheken, die das Programmieren unterstützen und bereits bestehende Funktionalität wie die benutzten `MouseListener` bereitstellt. Weiterhin ist das Framework QuEval in Java geschrieben und eine eventuelle Verschmelzung des Tools mit diesem Framework sollte gewährleistet sein. Mit dem Ziel der Einführung in das Tool und dessen Implementierung soll im nun folgenden Kapitel auf den strukturellen Aufbau des Tools eingegangen werden, bevor in den weiteren Kapiteln ein detailierterer Einblick in die Implementierung gegeben wird.

5.1 Aufbau des Tools

Das Design des Tools wurde möglichst modular gestaltet, da der Austausch von Funktionalität gewährleistet werden soll. Dafür muss klar sein, welche Funktionalität wie gekapselt werden soll. In diesem Zusammenhang bedeutet Kapselung, dass Implementierungsdetails hinter einer Schnittstelle versteckt werden [SS10]. Diese Schnittstelle bestimmt zwar die Ein- und Ausgaben; die Implementierung ist jedoch austauschbar und kann verbessert oder erweitert werden. Ein weiterer Aspekt der Modularisierung ist Kohäsion [SS10]. Das bedeutet, dass Programmkonstrukte, die zueinander in Beziehung stehen in einer Einheit zusammengefasst werden sollen. Dadurch sollen die Funktionsaufrufe innerhalb der Einheit erfolgen und nur wenige nach außen zu anderen Einheiten durchgeführt werden. Bei der objektorientierten Programmierung in Java wird die Funktionalität einzelner Objekte in so genannten Klassen gespeichert. Die Objektorientierung modelliert diese Klassen wie reale Objekte mit eigenen Eigenschaften, die in so genannten Feldern gespeichert werden, und eigenen Fähigkeiten, die in Methoden gespeichert werden. Somit wurde die Entscheidung getroffen jeweils für Prototypen und für die entstehenden Geraden eine eigene Klasse anzulegen. Weiterhin wurde eine

Klasse für die Indexstruktur angelegt und eine für die Metrik, da verschiedene Metriken ermöglicht werden sollten. Somit sollte gewährleistet werden, dass nicht nur die Funktionalität ausgetauscht werden kann, sondern mehrere verschiedene Metriken nebeneinander existieren können. Der konzeptuelle Aufbau des Tools wurde in Abbildung 5.1 als UML-Grafik visualisiert.

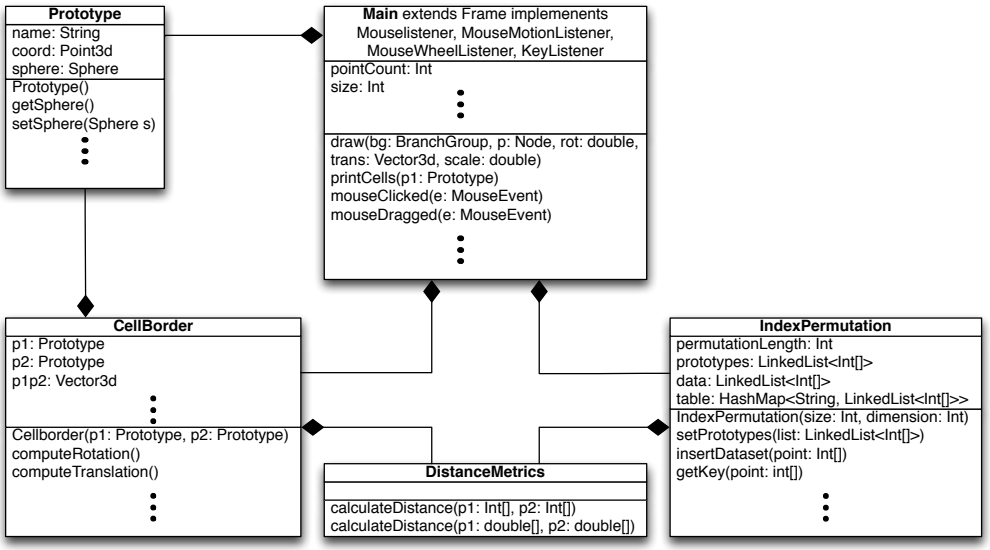


Abbildung 5.1: UML-Klassendiagramm des Tools

Die Benutzeroberfläche wird in der Klasse `Main` implementiert. Deshalb erbt diese von der Klasse `Frame`, wodurch alle Methoden und Felder der Klasse `Frame` auch in der Klasse `Main` verfügbar sind. Diese Vererbung erfolgt durch das Schlüsselwort ***extends*** hinter der Klassendeklaration [SS10]. Durch diese Vererbung wird die Funktionalität auf die Klasse `Main` übertragen, ein Fenster darzustellen und unter anderem Schaltflächen und Eingabefelder auf diesem zu positionieren.

Ein weiteres Schlüsselwort ist ***implements*** [SS10]. Dadurch müssen die Methoden, die in der übergeordneten Klasse definiert werden, in der eigenen Klasse implementiert werden. In dem vorhandenen Tool implementiert die Klasse `Main` eine Anzahl verschiedener `Listener`, die unterschiedliche Funktionalität bieten. Zum einen gibt es den `MouseListener`, der es ermöglicht das Klicken oder Drücken der Maustaste zu überwachen und beim Eintritt eines Ereignisses eine Aktion auszuführen. Analog dazu funktioniert auch der `MouseWheelListener` für das Mausrad und der `KeyListener` für die Tasten der Tastatur. Der `MouseMotionListener` hingegen bearbeitet komplexere Abläufe, wie zum Beispiel das Ziehen der Maus bei gedrückter Maustaste (engl. drag). Diese `Listener` helfen bei der Umsetzung der Anforderungen des Abschnittes 4.2 Manipulation der Prototypen, wo die Interaktionen mit der Maus und den Tasten der Tastatur umgesetzt werden muss. Die Klasse `Main` benutzt weiterhin Instanzen der Klassen `IndexPermutation`, `Prototype` und `CellBorder`, wobei letztere die Funktionalität der Geraden, die durch Einfügen der Prototypen entstanden sind, beschreibt.

Die grafische Darstellung des Datenraums, inklusive der Prototypen, der Geraden und der Datenpunkte, wurde mit Java 3D [1] implementiert. Die Bibliothek Java 3D enthält viele nützliche Funktionen zum Entwickeln von 3D-Anwendungen. Dazu gehört zum einen, dass geometrische Körper im dreidimensionalen Raum platziert werden können und das räumliche Aussehen bestimmter Körper vorbestimmt werden kann. Weiterhin ist es mit wenigen Schritten möglich, sich im Raum zu bewegen und zum Beispiel hineinzuzoomen. Auch das Bewegen der Körper durch die Maus kann mit Java 3D einfach umgesetzt werden. Um zu unterscheiden, was verändert werden darf und was geschützt ist, wird bei der Entwicklung mit Java 3D ein so genannter Szenegraph aufgebaut. Auf die einzelnen Bestandteile und den Aufbau wird im nächsten Abschnitt eingegangen.

5.2 Aufbau des Szenegraphen

In Java 3D werden die verschiedenen geometrischen Körper in einem Szenegraphen angeordnet [SD99]. Ein beispielhafter Aufbau eines Szenegraphen kann in Abbildung 5.2 nachempfunden werden.

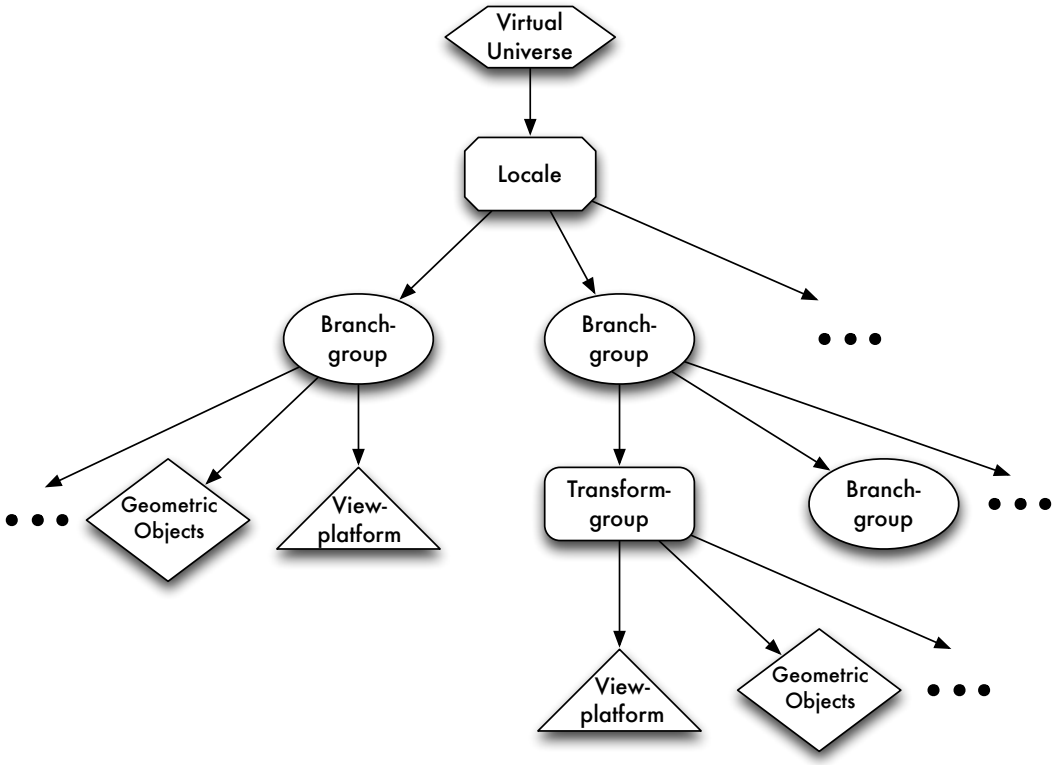


Abbildung 5.2: Beispiel eines Szenegraphen aus Java 3D

[1] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html

Der Wurzelknoten ist immer ein virtuelles Universum. Darunter befindet sich ein `Locale`, dass alle anderen Knoten enthält. Dieses `Locale` beschreibt die Darstellungsfläche, auf der die darunter liegenden Figuren des Szenegraphen gezeichnet werden. An das `Locale` können ein oder mehrere `BranchGroups` gehängt werden. An diese können gleich die geometrischen Figuren gehangen werden oder durch weitere `BranchGroups` zusätzliche Verzweigungen eingeführt werden. Wichtig ist weiterhin die Angabe der `Viewplatform`. Diese muss nicht zwingend einer `BranchGroup` untergeordnet sein, sondern kann auch für das gesamte Universum gelten. Dadurch wird unter anderem die Anschauungsplattform definiert, also der Standpunkt, von dem aus die Szene betrachtet wird. An eine `BranchGroup` kann weiterhin eine `TransformGroup` gehangen werden. Mit dieser kann die Lage der untergeordneten Objekte im `Locale` verändert werden. Mögliche Veränderungen sind hierbei das Rotieren, Translatieren und Skalieren der Kind-Objekte. Diese Elemente bilden den groben Aufbau eines Java 3D Szenegraphen.

Das Tool besteht aus zwei Szenegraphen, da wir zwei Bereiche haben, die etwas darstellen. Zum einen gibt es den Datenraum, der die Datenpunkte, Prototypen, Geraden und Zellenbeschriftungen zeigt. Zum anderen wird ein Diagramm gefordert, dass die Bucketauslastung visualisiert und konsistenter Weise als Java 3D Grafik angezeigt wird. Der Aufbau des Szenegraphens, mit dem die Bucketauslastung dargestellt wird, ist in Abbildung 5.3 nachvollziehbar.

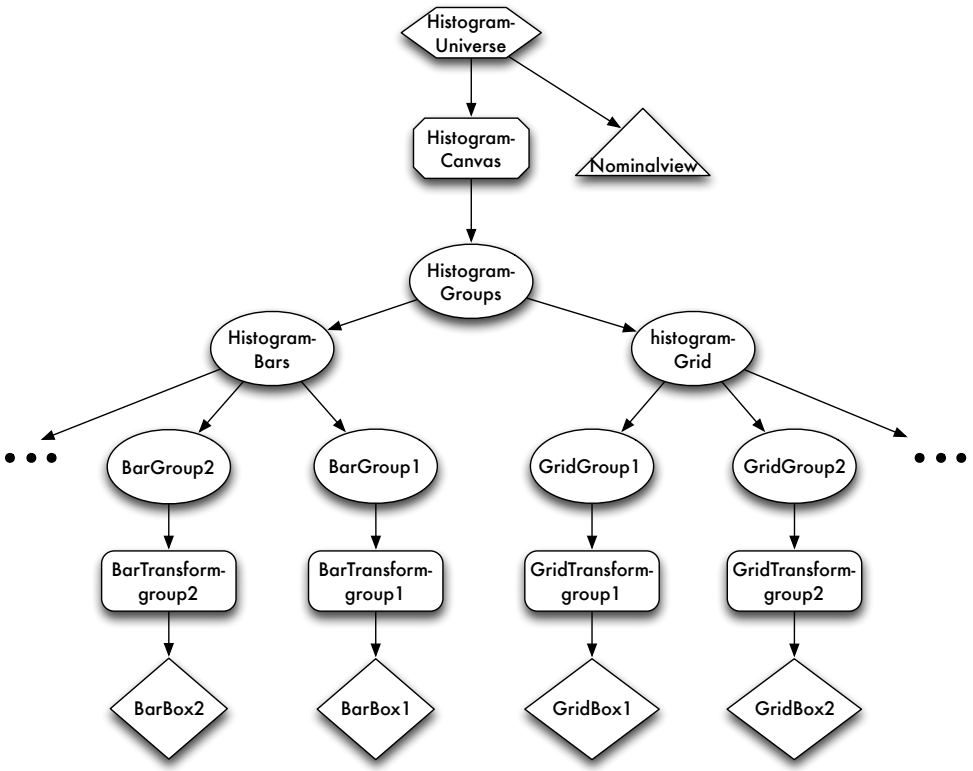


Abbildung 5.3: Szenegraph zur Darstellung der Bucketauslastung

Der Szenegraph für die Bucketauslastung besteht als aller erstes aus einem virtuellen Universum, das `histogramUniverse` genannt wurde. Darunter wird das `Locale histogramCanvas` und die Anschauungsplattform als `Nominalview` definiert und somit eine Art Vogelperspektive ermöglicht. Darunter hängt eine `BranchGroup` namens `histogramGroups`, die wiederum zwei `BranchGroups` als Kinder hat. Die `BranchGroup histogramBars` beinhaltet zum einen die Balken des Diagramms und auch noch weitere `textGroups`, die die Beschriftungen enthalten(Abbildung 5.4). Die `BranchGroup histogramGrid` hingegen beinhaltet die Abgrenzungen der einzelnen Balken des Diagramms.

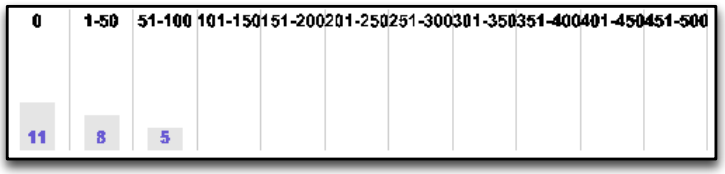


Abbildung 5.4: Darstellung der Bucketauslastung durch ein Balkendiagramm

Durch die Aufteilung in `histogramBars` und `histogramGrid` ist es möglich, beim Ausschalten der Visualisierung der Bucketauslastung, alle Balken durch Ausführung eines Befehls auszublenden und das Gitter weiter bestehen zu lassen. Weiterhin muss jeder Balken seine eigene `BranchGroup` besitzen, weil pro `BranchGroup` nur eine `TransformGroup` erlaubt ist. Würden mehrere Objekte mit einer `TransformGroup` positioniert, so befinden sich ihre Mittelpunkte im gleichen Punkt und die Objekte würden sich überlagern.

Der Szenegraph für die Darstellung des Datenraums (Abbildung 5.5) inklusive der Prototypen, Datenpunkten und Geraden ist durch die Anzahl an Objekten komplizierter aufgebaut.

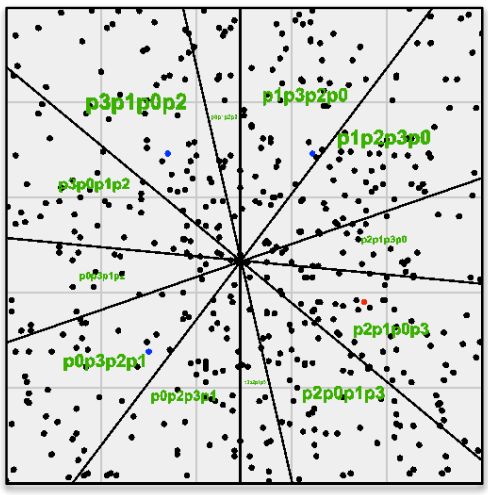


Abbildung 5.5: Darstellung des Datenraums

Hierbei müssen die einzelnen Körper in verschiedene `BranchGroups` einsortiert werden. Es wird jeweils eine `BranchGroup` für die Prototypen, die Geraden, die Datenpunkte, das angedeutete Gitternetz und die Beschriftungen der Zellen angelegt, um individuell die einzelnen Körper behandeln zu können. Durch die Aufteilung ist es wieder möglich einzelne `BranchGroups` zu leeren. Dies ist hilfreich beim Löschen aller Datenpunkte oder Verbergen der Zellbeschriftungen.

Der entstehende Szenegraph des Datenraums (Abbildung 5.6) enthält im Gegensatz zum Szenegraphen der Bucketauslastung weitere Besonderheiten. Damit der Hintergrund die graue Farbe erhält, wurde unter die `BranchGroup Group` ein Objekt gehangen, dass den Hintergrund festlegt und sich über den Datenraum ausbreitet. Weiterhin wurde ein `PickCanvas` eingefügt, dass es ermöglicht die Kinder der `PrototypeGroup` zu „picken“. Das bedeutet, dass die Objekte in der `PrototypeGroup` mit der Maus ausgewählt werden können. In Kombination mit den `Mouse-` und `MouseMotionListener` ist es dann möglich die ausgewählten Objekte zu bewegen oder zu löschen. Wird beim Klicken in den Datenraum jedoch kein Prototyp ausgewählt, so soll an dieser Stelle ein Prototyp eingefügt werden. Ein weiterer Unterschied zum Szenegraphen aus Abbildung 5.3 ist eine zusätzliche `TransformGroup`, die unter der obersten `BranchGroup` hängt. Mit dieser wird das Zoomen in den Datenraum ermöglicht werden, weil die Änderungen an der `TransformGroup` alle untergeordneten Objekte betrifft.

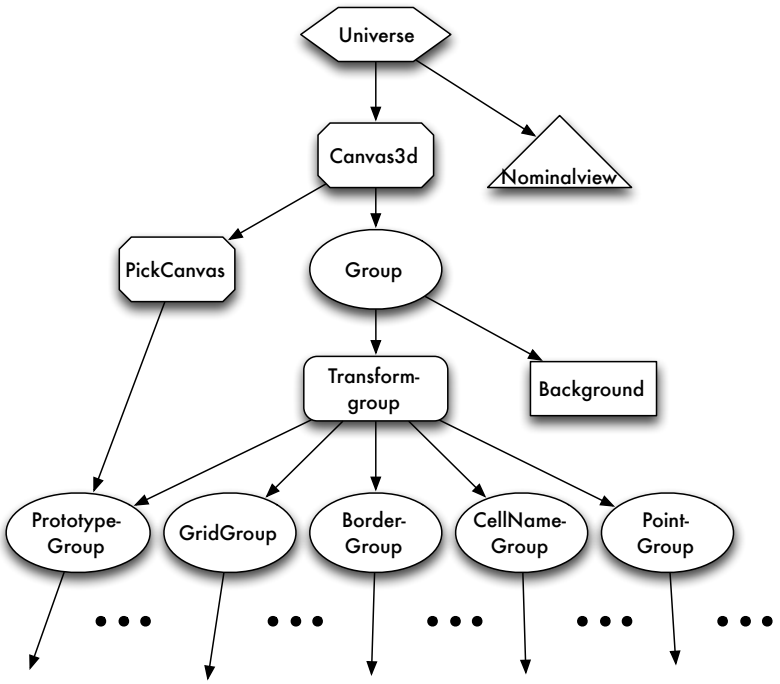


Abbildung 5.6: Szenegraph des Datenraums

Unter den fünf untersten `BranchGroups` des Szenegraphen aus Abbildung 5.6 hängen wiederum pro geometrischem Objekt eine `BranchGroup`, darunter eine `TransformGroup` und schließlich das darzustellende Objekt. Dies geschieht somit

analog zu den Objekten in Abbildung 5.3. Aufgrund der mehrmaligen Verwendung ein und desselben Algorithmus zum Darstellen von Objekten in Java 3D, liegt es nahe, diese Funktionalität in eine eigene Funktion zu kapseln. Diese Funktionen werden im nächsten Abschnitt beschrieben und der entstandene Code gezeigt.

5.3 Ausgewählte Funktionen

In diesem Abschnitt soll anhand von einzelnen Codebeispielen ein Einblick in die Funktionsweise des Tools gegeben und die Umsetzung einzelner Anforderungen aus dem Kapitel 4 beschrieben werden.

5.3.1 Funktionen der Klasse Main

Wie im Abschnitt 5.2 beschrieben wurde, ist es von Vorteil den Algorithmus zum Erstellen einer `BranchGroup` mit dazugehöriger `TransformGroup` und dem darzustellenden geometrischen Objekt in eine eigene Funktion auszulagern. Der Quellcode dieser Methode wurde in Abbildung 5.7 dargestellt.

```
1  private BranchGroup draw(BranchGroup bg, Node p, double rot, Vector3d
       trans, double scale) {
2
3      BranchGroup newBranchGroup = new BranchGroup();
4      newBranchGroup.setCapability(BranchGroup.ALLOW_DETACH);
5      TransformGroup newTransformGroup = new TransformGroup();
6
7      Transform3D newTransform = new Transform3D();
8      newTransform.rotZ(rot);
9      newTransform.setTranslation(trans);
10     newTransform.setScale(scale);
11     newTransformGroup.setTransform(newTransform);
12     newTransformGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
13     newTransformGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
14     newTransformGroup.addChild(p);
15     newBranchGroup.addChild(newTransformGroup);
16     bg.addChild(newBranchGroup);
17     return newBranchGroup;
18
19   }
```

Abbildung 5.7: Funktion `Draw` zum Einfügen von Körpern in den Szenegraphen

Als Übergabewerte werden eine `BranchGroup bg`, ein `Node p`, ein `double rot`, ein `Vector3d trans` und ein weiterer `double` namens `scale` übergeben. Die `BranchGroup bg` ist hierbei die `BranchGroup`, in die der Punkt eingeordnet werden soll. Bei Prototypen wäre dies zum Beispiel die `PrototypeGroup` (vgl. Abbildung 5.6). Der Übergabeparameter `Node p` stellt hierbei die Superklasse eines einzufügenden geometrischen Objektes dar. Durch die Wahl des Typs `Node` als Übergabeparametertyp können verschiedene Klassen von Objekten mit dieser Methode gezeichnet werden, wenn diese Klassen die Klasse `Node` als Superklasse haben. Dies gilt unter anderem für die Subklassen der Klasse `Primitives`, die einfache geometrische Objekte wie Boxen und Kugeln und Zylinder darstellen,

aber auch für die Subklassen der Klasse `Shape3D`, die unter anderem die Darstellung von Text ermöglichen. Als weitere Übergabewerte werden die Rotation `rot` und die Skalierung `scale` als `double`, sowie die Translation `trans` vom Typ `Vector3d` übergeben.

Die erste Anweisung in der Methode legt eine neue `BranchGroup` an, die die untergeordnete `TransformGroup` und das geometrische Objekt enthält. Java 3D ist sehr restriktiv, deshalb muss explizit angegeben werden, dass es erlaubt ist diese `BranchGroup` vom Elternknoten abzukapseln. Dies wird in der Zeile 4 umgesetzt. Damit Rotation, Translation und Skalierung angewandt werden können, muss eine `Transform3D` angelegt werden, die der `TransformGroup` zugeordnet wird. Danach können die jeweiligen Veränderungen der `Transform3D` übergeben werden (vgl. Zeile 8-10). Eine weitere Restriktion wird in Zeile 12-13 aufgehoben, wodurch es ermöglicht wird, die angegebene `Transform3D` im nachhinein auszulesen und neu zu belegen. Dies ist hilfreich, wenn zum Beispiel Prototypen verschoben werden und somit die Position im Datenraum verändert werden muss. Schließlich wird der `Node p` als Kindknoten an die `TransformGroup` gehangen. Abschließend wird diese dann an die erstellte `BranchGroup` und das gesamte dann an die übergebene `BranchGroup` angehangen. Als Rückgabewert wird die erstellte `BranchGroup` zurückgegeben, weil unter Umständen weitere Veränderungen im nachhinein gemacht werden sollen.

Mit der Funktion aus Abbildung 5.7 können nun verschiedene Objekte in Java 3D dargestellt werden und Rotation, Translation und Skalierung auf diese angewandt werden. Diese Funktion wird in der Methode `printCells` benutzt, um beim Hinzufügen eines Datenpunktes die entstehenden Geraden, hier `CellBorder` genannt, darzustellen und in den Szenegraphen einzufügen. Der Quellcode der Methode `printCells` wurde in Abbildung 5.8 dargestellt.

```
1  private void printCells(Prototype P1) {
2      for (Prototype P2 : prototypeList) {
3        if (!P1.getName().equals(P2.getName())) {
4
5          CellBorder cb = new CellBorder(P1, P2, new TransformGroup());
6          cellBorderList.add(cb);
7          //neue Appearance mit der Farbe Schwarz
8          Appearance lineApp = getAppearance(0f, 0f, 0f);
9          Box border = new Box(3f, borderwidth, 0f, lineApp);
10         BranchGroup borderbg = draw(BorderGroup, border,
               cb.computeRotation(), cb.computeTranslation(), 1.0);
11         cb.setTransformGroup((TransformGroup) borderbg.getChild(0));
12
13       }
14     }
15   }
```

Abbildung 5.8: Funktion `printCells` zum Erstellen der Geraden

Als Eingabeparameter für die Funktion dient der eingefügte Prototyp `P1`. Aufgrund der Eigenschaft, dass jedes Prototypenpaar eine Gerade beeinflusst, muss beim Einfügen eines Prototyps für jeden schon vorhandenen Prototypen eine Gerade eingefügt werden. Deshalb wird für jedes Paar aus `P1` und `P2`, wobei `P2` aus der Liste der Prototypen ist, eine Gerade angelegt (vgl. Zeile 2-3). Zum Erstellen

einer Instanz einer Geraden wird hierbei der Konstruktor der Klasse `CellBorder` verwendet. Dieser hat als Eingabe die beiden Prototypen `P1` und `P2`, die die Lage der Geraden beeinflussen und die `TransformGroup`, die die Lage im Datenraum mit Java 3D definiert. Da die neu erstellte Gerade noch keine `TransformGroup` hat, wird hier zu allererst eine neue `TransformGroup` übergeben.

Die erstellte Gerade wird nun in die Liste der `CellBorder` eingefügt, um bei Veränderungen einen einfachen Zugriff zu erhalten. Wird im Programmverlauf die Lage eines Prototyps verändert, so muss nur die Liste der `CellBorder` durchgegangen werden und die betroffenen Geraden anhand der zugewiesenen Prototypen bestimmt und ihre zugeordnete `TransformGroup` bearbeitet werden.

In Zeile 8 wird nun das Aussehen der Geraden erstellt. Die selbstdefinierte Funktion `getAppearance` bekommt drei Werte vom Typ `float` im Intervall $[0, 1]$, die der Konzentration von Rot, Gelb, Blau entsprechen. Aufgrund der übergebenen Nullen, entsteht die Farbe Schwarz für die Geraden. In der darauffolgenden Zeile wird eine `Box` erstellt, die die Länge, Breite und Tiefe, sowie eine `Appearance` übergeben bekommt. Diese `Box` wird dann in der nächsten Zeile in den Szenegraphen durch die Methode `draw` eingefügt.

Für die Berechnung der Translation und Rotation sind in der Klasse `CellBorder` eigene Methoden erstellt worden, dessen mathematischer Hintergrund in Folge noch betrachtet wird. Die durch die Funktion `draw` erstellte und zurückgegebene `BranchGroup` wird nun in Zeile 11 dazu genutzt, dem `CellBoder` die `TransformGroup` zuzuordnen. Dafür wird das erste Kind der `BranchGroup` zur `TransformGroup` gecastet und per Methode `setTransformGroup` der erstellten Gerade zugeordnet.

Die Darstellung der Geraden nimmt einen wichtigen Teil der Visualisierung ein. Deshalb soll im nächsten Abschnitt einige Methoden der Klasse `CellBorder` erläutert werden.

5.3.2 Funktionen der Klasse CellBorder

Die Klasse `CellBorder` enthält die Variablen und Methoden, die benötigt werden um eine Gerade zu definieren, die durch zwei Prototypen beeinflusst wird. Deshalb werden in der Klasse die beiden Prototypen gespeichert, von denen die jeweilige Instanz abhängt. Weiterhin bietet die Klasse `CellBorder` Funktionen, die zum Berechnen der Lage der Geraden im Raum genutzt werden können.

Zum Erstellen einer Instanz der Klasse `CellBorder` werden dem Konstruktor zwei Prototypen `P1` und `P2` übergeben. Damit wird der `Vector3d P1P2` erstellt, der dem mathematischen Vektor von `P1` zu `P2` entspricht und dessen Berechnung beim Konstruktor, so wie beim Verschieben der Prototypen ausgeführt wird. Berechnet wird hierfür die Differenz der x-,y- und z-Werte von `P1` und `P2`:

```
1 P1P2 = new Vector3d(P2.getCoord().x - P1.getCoord().x, P2.getCoord().y -
      P1.getCoord().y, P2.getCoord().z - P1.getCoord().z);
```

Abbildung 5.9: Berechnung des Vektors `P1P2`

Mit Hilfe der Variablen `P1P2` kann nun zum Beispiel die Position der Geraden berechnet werden, wie in Abbildung 5.10 nachvollzogen werden kann. In die Berechnung des Vektors `P1P2` und der Translation gehen drei Dimensionen ein, für den zweidimensionalen Fall steht jedoch in der z-Koordinate eine Null. Die dreidimensionale Umsetzung des Tools wurde zwar an vielen Stellen vorbereitet, jedoch wurde sie nicht vollständig im Rahmen dieser Arbeit realisiert und somit bleibt diese Anforderung für weitere Projekte offen.

```
1 public Vector3d computeTranslation() {
2     return new Vector3d(P1.getCoord().x + 0.5 * P1P2.x, P1.getCoord().y +
          0.5 * P1P2.y, P1.getCoord().z + 0.5 * P1P2.z);
3   }
```

Abbildung 5.10: Funktion `computeTranslation` zur Berechnung der Lage der Geraden im Raum

Die entstehenden Geraden befinden sich genau in der Mitte der beiden Prototypen, die sie beeinflussen. Mathematisch bedeutet dies, dass auf den Ortsvektor von `P1` die Hälfte des Vektors von `P1` zu `P2` addiert werden muss.

Nachdem nun die Position der Geraden bestimmt wurde, wird in Abbildung 5.11 die Berechnung der Rotation der Geraden gezeigt.

```
1   public double computeRotation() {
2     Vector3d normal = new Vector3d(-P1P2.y, P1P2.x, P1P2.z);
3     double numerator = normal.x * 1 + normal.y * 0 + normal.z * 0;
4     double denominator = metrik.calculateDistance(new double[]{0,0,0}, new
          double[]{normal.x,normal.y,normal.z});
5     double angle = numerator / denominator;
6     double cosangle = Math.acos(angle);
7     if (P1P2.x < 0) {
8       return (2 * Math.PI - cosangle);
9     } else {
10      return cosangle;
11
12    }
13
14  }
```

Abbildung 5.11: Funktion `computeRotation` zur Berechnung der Rotation der Geraden zur x-Achse

Zur Berechnung des Winkels der Drehung, wird die Winkelberechnung zwischen zwei Vektoren benutzt. Einer dieser Vektoren ist die x-Achse mit den Koordinaten (1,0,0). Der andere Vektor ist der Richtungsvektor der entstehenden Geraden. Dieser verläuft orthogonal zum Vektor von `P1` zu `P2` und ist somit die Normale zum Vektor $\overrightarrow{P1P2}$. Für die zweidimensionale Umsetzung des Tools, lässt sich der Normalenvektor berechnen, indem die ersten beiden Vektoren vertauscht und einer davon negiert wird, wie in Zeile 2 zu sehen ist. Zur Berechnung des Winkels λ zwischen zwei Vektoren $\overrightarrow{v_1}, \overrightarrow{v_2}$, gilt folgende Formel: $cos\lambda = \frac{\overrightarrow{v_1} * \overrightarrow{v_2}}{\left\|\overrightarrow{v_1}\right\| * \left\|\overrightarrow{v_2}\right\|}$. Der Zähler wird durch das Skalarprodukt aus den beiden Vektoren berechnet, wie in Codezeile 3 zu sehen ist. In der nachfolgenden Zeile wird der Nenner berechnet,

der aus dem Produkt der Länge der Vektoren gebildet wird. Da die Länge des Einheitsvektors (1,0,0) Eins ergibt, musste dieser in der Rechnung nicht weiter berücksichtigt werden, sodass nur noch die Länge des Normalenvektors bestimmt werden muss. Die Länge dieses Vektors berechnet sich aus der Distanz zwischen dem Koordinatenursprung und dem Vektor unter Verwendung der gewählten Metrik (vgl. Zeile 4). Nach der Formel für die Winkelberechnung, muss nun der Arkuskosinus aus dem Quotienten gebildet werden. Dieser befindet sich im Bereich $[0, \pi]$ und deckt somit nicht die volle Drehung ab. Deswegen muss, um den größeren der beiden Winkel zu berechnen λ von 2π abgezogen werden. Dieser Fall wurde in Abbildung 5.13 nachgestellt.

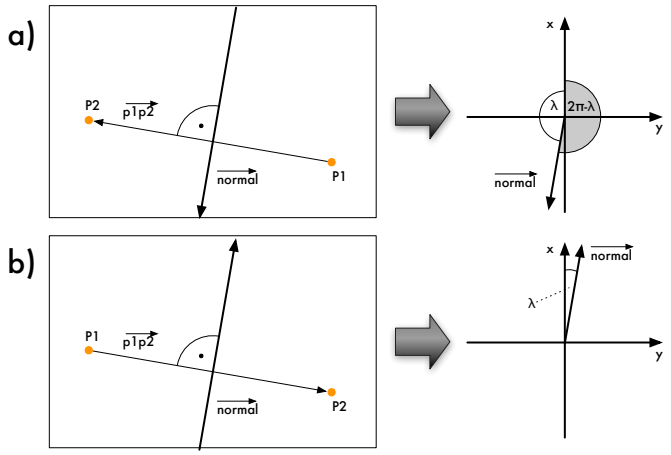


Abbildung 5.12: Winkelberechnung der Geraden zur x-Achse

Wie hier dargestellt wurde, ist die x-Koordinate des Vektors $\overrightarrow{P1P2}$ im Beispiel a) negativ. Dadurch verläuft der resultierende Normalenvektor in die negative y-Richtung. Da bei der Berechnung des Winkels immer der eingeschlossene Winkel berechnet wird, ergibt sich der falsche Winkel. Wird dieser von der x-Achse im Uhrzeigersinn abgetragen, so entsteht ein Normalenvektor, der an der x-Achse gespiegelt ist. Im Beispiel b) hingegen wird der richtige Winkel berechnet, der von der x-Achse im Uhrzeigersinn abgetragen den richtigen Normalenvektor ergibt. Hierbei ist aber auch der x-Wert des Vektors $\overrightarrow{P1P2}$ positiv.

Nachdem nun die einzelnen Prototypen per Mausklick in den Datenraum gesetzt werden können und auch die Geraden dazu gezeichnet werden können, ist es weiterhin von Interesse, wie die Bucketauslastung bestimmt werden kann. Für diese Berechnung musste der permutationsbasierte Index umgesetzt werden, von dem einige Funktionen im nächsten Abschnitt beschrieben werden.

5.3.3 Funktionen der Klasse IndexPermutation

Der permutationsbasierte Index wurde als Hashverfahren mit einer `Java-Hashmap` umgesetzt. Dadurch kann anhand eines Schlüssels, der die Permutation darstellt, Werte in der `Hashmap` gespeichert und auch wieder extrahiert werden. Da pro

Schlüssel mehrere Einträge gespeichert werden müssen, steckt hinter jedem Wert eine verkettete Liste, die die Punkte speichert.

Das Laden von Datenbasen in den Index wird durch Auswählen der richtigen Datei mit dem `JFileChooser` durchgeführt. Die Punkte werden Zeilenweise aus der Datei gelesen und können dann einzeln wie folgt eingefügt werden:

```
1  try {
2    BufferedReader br = new BufferedReader(new FileReader filePath));
3    String line;
4      while ((line = br.readLine()) != null) {
5        String[] s = line.split("␣");
6        int x = Integer.parseInt(s[0]);
7        int y = Integer.parseInt(s[1]);
8        indexPermutation.insertDataset(new int[] { x, y });
9        ...
10       }
11       catch (FileNotFoundException e) {
12         e.printStackTrace();
13       } catch (IOException e) {
14         e.printStackTrace();
15       }
16   }
17 }
```

Abbildung 5.13: Auslesen der einzelnen Datenpunkte aus der generierten Datei mit den Datensätzen

Es wird Zeile für Zeile ausgelesen, die entstehende Zeichenkette wird an den Leerzeichen gesplittet und diese dann zu einem Zahlenwert geparset. Dadurch kann der Datenpunkt als Array aus Zahlen in den Index eingefügt werden. Die aufgerufene Methode `insertDataset` wurde in Abbildung 5.14 dargestellt.

```
1  public boolean insertDataset(int[] point) {
2          //berechnet die Permutation zum gegebenen Punkt
3          String key = getKey(p);
4          //gibt die Liste aufgrund der Permutation zurück
5          LinkedList<int[]> insertInto = table.get(key);
6          if (insertInto == null) {
7            //ist die Liste leer-> erstelle eine Neue
8                  insertInto = new LinkedList<int[]>();
9          }
10         insertInto.add(p);
11         //Wiedereinfügen der Liste in die Hashmap
12         table.put(key, insertInto);
13   return true;
14 }
```

Abbildung 5.14: Einfügen von Punkten in den permutationsbasierten Index mittels der Funktion `insertDataset`

Für das Einfügen der Datensätze in die `Hashmap` muss zu allererst der Schlüssel des Buckets, in die der Datensatz einzuordnen ist, bestimmt werden. Dieser Schlüssel wird durch die Funktion `getKey` berechnet, die im weiteren Verlauf noch erläutert wird. Mit der Funktion `get` auf unsere `Hashmap` namens `table`, bekommen wir nun die `LinkedList` in die der Punkt eingeordnet werden muss. Ist diese

null, so wurde sie noch nicht initialisiert und der Konstruktor der `LinkedList` wird aufgerufen. In Zeile 10 wird dann der Punkt in die Liste eingereiht und in Zeile 12 dann die neue `LinkedList` in die Tabelle eingefügt und die vorhergehende damit überschrieben.

Für das Einfügen von Punkten, sowie für die Bestimmung der Beschriftung der einzelnen Zellen ist eine Funktion nötig, die den einzelnen Punkten anhand ihrer Entfernung zu den Prototypen den richtigen Hashwert zuweist. Die Berechnung der Permutation wird durch die Funktion `getKey` berechnet (Abbildung 5.15).

```
15   public String getKey(int[] point) {
16         ArrayList<Integer> Indices = getPermutation(point);
17         String key = "";
18          //forms the key e.g. "p1p2p5p4p3"
19             for (int i = 0; i< permutationLength;i++) {
20                 key += "p"+ prototypes.get(Indices.get(i)).getName();
21             }
22         return key;
23     }
```

Abbildung 5.15: Berechnung des Schlüsselwerts eines Datenpunktes

Die Berechnung des Schlüssels beginnt mit der Berechnung der Reihenfolge der Prototypen. Dafür wird in der Funktion `getPermutation` der Abstand zu jedem Prototypen anhand der Metrik bestimmt und dann die Indizes der Prototypen anhand dieser Distanzen durch sortiertes Einfügen geordnet. Danach wird der `Key` zusammengesetzt, indem immer ein „p“ vorangestellt wird und dahinter die Zahl des Prototypen gesetzt wird. Dadurch sind auch bei mehr als zehn Prototypen die Indizes eindeutig voneinander unterscheidbar.

Mit diesem Codebeispiel soll nun die Betrachtung von Implementierungsdetails abgeschlossen werden, auch wenn es noch unzählige Funktionen gibt, die exemplarisch gezeigt werden können. Es wurde ein guter Überblick über einige Funktionsweisen gegeben; sei es allgemein aus Java, oder aber auch aus der Erweiterung Java 3D mit der dreidimensionale Objekte dargestellt werden können.

5.4 Zusammenfassung

Zu Beginn dieses Kapitels wurde der Aufbau des Tools vorgestellt, um an die Umsetzung des Tools heranzuführen. Weiterhin wurden die Konzepte der Modularität erklärt und beschrieben, wie diese in dem vorliegenden Tool umgesetzt wurden. Im Kapitel 5.2 wurde dann eine Einführung in Java 3D gegeben und vorallem auf den Aufbau von Szenegraphen hingewiesen. Im weiteren Verauf des Kapitels wurde anhand von weiteren Abbildungen, die entwickelten Szenegraphen für die Umsetzung der Visualisierung des Tools mit Java 3D vorgestellt. Anschließend wurden ausgewählte Funktionen der Klassen `Main`, `CellBorder` und `IndexPermutation` gezeigt. Anhand dieser konnte der Implementierungsaufwand nachvollzogen werden und die Umsetzung einzelner Anforderungen aufgezeigt werden. Aufbauend auf diesem Kapitel ist es nun möglich die Umsetzungen der Anforderungen zu betrachten. Diese Betrachtung soll im folgenden Kapitel durchgeführt werden.

Kapitel 6

6 Auswertung der erfüllten Anforderungen

In den vorherigen Kapiteln 4 und 5 wurde anhand von ausführlich formulierten Anforderungen die Implementierung eines Tools vorgestellt, mit dem die Visualisierung der Raumaufteilung und Bucketauslastung erleichtert werden soll. Dieses Kapitel soll als erstes einen Überblick über die umgesetzten Anforderung geben. Dabei wird darauf eingegangen, welche Anforderungen vollständig, teilweise oder nicht umgesetzt werden konnten, um dann in späteren Kapiteln weitere Anregungen zur Verbesserung des Tools zu geben und ausstehende Arbeitspakete für weitere Arbeiten am Tool zu definieren. Weiterhin soll die Nutzbarkeit des Tools gezeigt werden, indem erste Ergebnisse der Arbeit mit dem Tool gezeigt werden sollen. Dafür wird eine Empfehlung für das Setzen der Prototypen für die Anfragetypen Exakt-Match, Bereichsanfragen und der Suche nach den nächsten Nachbarn gegeben.

Der folgende Abschnitt zeigt als erstes das Benutzerinterface und dient der Klärung aller vorhandenen Schaltflächen zur besseren Auswertung der umgesetzten Anforderungen.

6.1 Das entstandene Tool

Als erstes wird in diesem Kapitel das entstandene Tool vorgestellt, weil dadurch die Auswertung der Anforderungen erleichtert werden soll. Im linken oberen Bereich des Tools (Abbildung 6.1) befindet sich die Darstellung des Datenraums inklusive der Datenpunkte, Prototypen und der Geraden, die den Raum aufteilen. Auch die Beschriftung der Regionen ist ersichtlich und wurde an die Größe der Regionen angepasst. Die schwarzen Punkte stellen die Daten aus einer Datenbasis dar, die insgesamt 500 Datenpunkte enthält. Die blauen Punkte stellen die Prototypen P0 bis P3 dar, wobei im Moment der Prototyp P2 ausgewählt ist und somit eine rote Färbung besitzt. Links unten wurde die Bucketauslastung anhand des Balkendiagramms visualisiert, wobei zur Zeit elf Buckets leer sind, acht Buckets enthalten zwischen 1 und 50 Datenpunkte und fünf Buckets beinhalten 51 bis 100 Datenpunkte. Der Bereich auf der rechten Seite, beinhaltet weitere Steue-

relemente. Ganz oben befindet sich eine Auswahlliste, die eine Liste der aktuellen Prototypen enthält. Bei der Auswahl eines Prototypen in dieser Liste, erscheinen seine Koordinaten in den Feldern darunter. Dies geschieht ebenso, wenn der jeweilige Prototyp mit der Maus angeklickt wird. Die Koordinaten in den Feldern können verändert werden und somit ist es möglich die Protoypen punktgenau zu setzen, indem die Schaltfläche „Set“ betätigt wird. Mit der Schaltfläche „Del“ ist es möglich den aktuell gewählten Prototypen zu löschen und somit auch alle Geraden, die er beeinflusst. Unter diesen beiden Schaltflächen befindet sich ein Kontrollkästchen, mit dem die Beschriftung der Regionen ein- beziehungsweise ausgeblendet werden kann. Die Anforderung, die Länge der Permutation zu beeinflussen, wurde auch umgesetzt. Durch Betätigen der Schaltflächen „+“ oder „-“ kann die Länge der Permutation verändert werden. Die aktuelle und die maximale Länge der Permutation werden in dem Textbereich daneben angezeigt. Schließlich befinden sich rechts unten noch zwei weitere Schaltflächen. Mit der Schaltfläche „Load“ kommt ein Auswahlfenster zum Vorschein, um die Datei mit den Beispieldatensätzen zu laden. Es ist ebenfalls möglich die eingefügten Datenpunkte mit der Schaltfläche „Unload“ aus dem Datenraum zu entfernen.

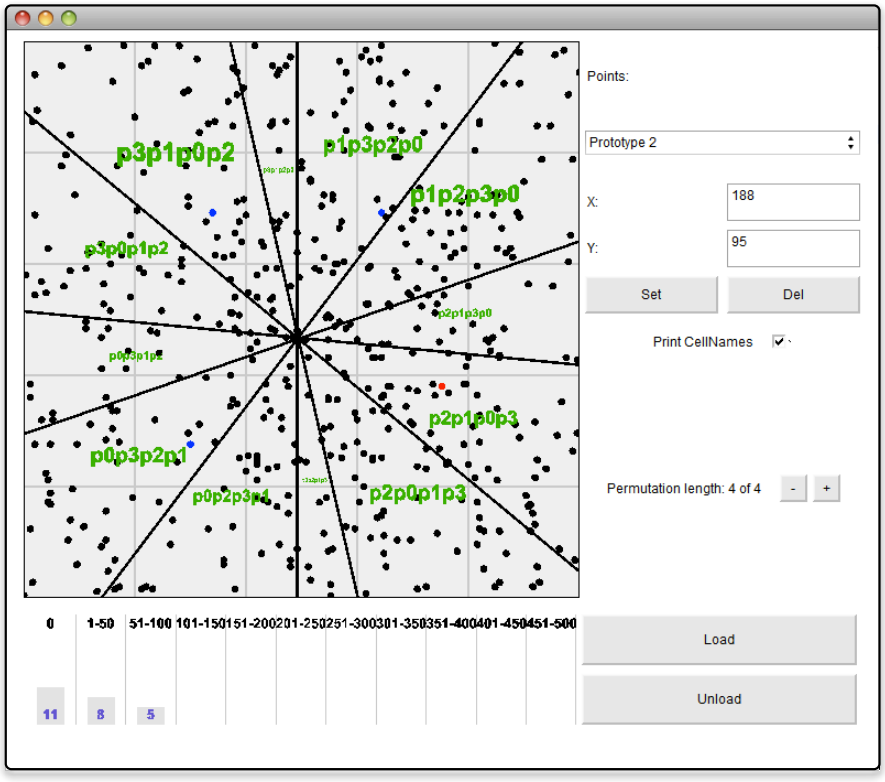


Abbildung 6.1: Das entstandene Tool

Nachdem das entstandene Tool vorgestellt wurde, soll in den nächsten Abschnitten ausgewertet werden, inwieweit die Anforderungen aus dem Kapitel 4 umgesetzt wurden.

6.2 Auswertung der Anforderungen

In den nun folgenden Abschnitten wird die Umsetzung der definierten Anforderungen beschrieben. Dafür wird zuerst auf den darzustellenden Raum eingegangen, danach auf die Manipulation der Prototypen und die damit verbundene Visualisierung der Raumaufteilung und abschließend wird die Umsetzung der Anforderungen an das Einbinden der Beispieldaten betrachtet. Um einen übersichtlichen Überblick über den Stand der Umsetzung zu geben, wurden folgende Symbole verwendet: ein „✓“ bedeutet, dass die Anforderung vollständig umgesetzt wurde. Das Symbol „X“ bedeutet, dass die Anforderung nicht umgesetzt wurde und wenn die Anforderung zumindest teilweise umgesetzt wurde, wird ein „~“ vergeben.

6.2.1 Darzustellender Raum

Als grundlegende Anforderung an das Tool wurde definiert, wie die räumliche Darstellung des Datenraums aussehen soll. Dafür wurden in Abschnitt 4.1 verschiedene Anforderungen definiert. Der Datenraum sollte sowohl zweidimensional, als auch dreidimensional dargestellt werden können. Weiterhin sollte es möglich sein, mittels Zoom den Datenraum zu vergrößern, um einige Bereiche genauer betrachten zu können. Damit dabei die Übersichtlichkeit erhalten bleibt, wurde weiterhin die Anforderung formuliert, dass ein Gitternetz vorhanden sein soll, das die Orientierung im Raum erleichtert. Diese Anforderungen wurden in Abbildung 6.1 ihrer Umsetzung gegenübergestellt.

Anforderung	**Umsetzung**	
2-D	Die Visualisierung im zweidimensionalen Raum wurde umgesetzt. Dafür wurde das Koordinatensystem aus Java 3D genutzt, wobei nur die x- und y-Werte betrachtet wurden. Somit blieb die Tiefe des Raumes ungenutzt. Der Wertebereich der Punkte beschränkt sich auf Werte zwischen 0 und 250, wobei diese Grenzen einfach adaptierbar sind.	✓
3-D	Eine dreidimensionale Darstellung des Datenraumes und der Raumaufteilung wurde in dieser Arbeit nicht umgesetzt und bleibt für weitere Arbeiten als Anforderung erhalten.	X
Zoom	Das Tool ermöglicht das Hineinzoomen sowohl per Mausrad, als auch mit den „+ “ und „-“ Tasten auf der Tastatur. Diese Funktionalität wurde mit Hilfe der `Listener` Klassen realisiert.	✓
Raster	Ein Raster des Datenraums wurde auch dargestellt (siehe Abbildung 6.1).	✓

Tabelle 6.1: Umsetzung des darzustellenden Datenraums

Die Umsetzung der zweidimensionalen Visualisierung, sowie das Zoomen und erstellen eines Gitternetzes wurde vollständig umgesetzt. Als Aufgabe für spätere Projekte bleibt noch die Visualisierung des Datenraumes und der Regionenaufteilung im dreidimensionalen Raum. Generell ist dabei an einigen Stellen eine Umstrukturierung vonnöten, wie zum Beispiel bei der Berechnung der Geraden. Jedoch bietet Java 3D alle Voraussetzungen für eine dreidimensionale Visualisierung. Für die Visualisierung der Raumaufteilung müssen in den Datenraum Prototypen gesetzt und verschoben werden können. Diese Umsetzung wird im folgenden Abschnitt betrachet.

6.2.2 Manipulation der Prototypen

Für die Darstellung der Raumaufteilung anhand der Prototypen, muss es möglich sein, Prototypen in den Raum zu setzen und ihre Lage zu verändern. Die dafür geltenden Anforderungen wurden in Tabelle 6.2 zusammengefasst.

Anforderung	**Umsetzung**	
Setzen von Prototypen	Das Setzen der Prototypen geschieht durch einen Mausklick in den Datenraum an eine Stelle, an der sich noch kein Prototyp befindet. Wird jedoch auf einen Prototypen geklickt, so wird dieser markiert und seine Koordinaten angezeigt.	✓
Verschieben eines Prototyps • per Maus • per Koordinaten • per Pfeiltasten	Durch die einzelnen `Listener` ist es möglich, dass die Prototypen durch Ziehen mit der Maus oder die Pfeiltasten verschoben werden können. Dadurch ist es einfach die vorhandenen Prototypen zu verschieben und die Veränderung der Raumaufteilung zu verfolgen. Weiterhin ist ein punktgenaues Verschieben der Prototypen durch Eingabe der Koordinaten in der Benutzeroberfläche möglich.	✓
Löschen eines Prototyps • per Maus • per Schaltfläche	Das Löschen eines Prototyps kann mit einem Klick mit der rechten Maustaste auf den Prototypen vollzogen werden. Außerdem kann die Schaltfläche „Del“ den aktuell gewählten Prototypen löschen.	✓
Kennzeichnung der Prototypen	Zur Unterscheidung der Datenpunkte und Prototypen, wurden die Prototypen blau eingefärbt, wobei der aktuell ausgewählte Prototyp eine rote Färbung besitzt.	✓

Tabelle 6.2: Umsetzung der Manipulation der Prototypen

In diesem Anforderungsbereich wurden die gestellten Anforderungen vollständig umgesetzt. Durch die Möglichkeit der Manipulation der Prototypen mit der Maus, den Pfeiltasten und auch teilweise durch Schaltflächen und Textfelder, ist ein breites Spektrum an Benutzeraktionen abgedeckt und implementiert worden. Ein weiterer Anforderungsbereich zielt auf die Darstellung der Raumaufteilung anhand der gesetzten Prototypen ab. Die Umsetzung dieser Anforderungen soll im nächsten Abschnitt erläutert werden.

6.2.3 Darstellung der Raumaufteilung

Anhand der gesetzten Prototypen soll die Raumaufteilung visualisiert werden. Dabei wurden verschiedene Anforderungen umgesetzt, die in Tabelle 6.3 zusammengefasst wurden.

Anforderung	**Umsetzung**	
Darstellung der Geraden	Wie in Abschnitt 5.3.2 dargestellt wurde, erfolgt die Berechnung der Position und Drehung der entsprechenden Gerade durch die Methoden der Klasse `CellBorder`. Diese wird bei Veränderungen der Prototypen aktualisiert und die Gerade wird angepasst.	✓
Darstellung des Hashwertes	Auch die Darstellung des Hashwertes kann in der Benutzeroberfläche aktiviert werden und wird beim Verändern der Prototypen automatisch angepasst.	✓
Verkürzung des Hashwertes	Generell wurde es ermöglicht den Hashwert zu verkürzen. Dies beeinflusst das Balkendiagramm zur Darstellung der Bucketauslastung und die Beschriftungen der entstandenen Regionen. Jedoch wurde es noch nicht umgesetzt, dass benachbarte Zellen, die nach dem Verkürzen den gleichen Hashwert besitzen, verschmolzen werden. Wird diese Anforderung noch umgesetzt, so können weitere Schlüsse über den Zusammenhang zwischen Anzahl der Prototypen, Länge des Hashes und der Raumaufteilung gezogen werden.	~

Tabelle 6.3: Umsetzung der Darstellung der Raumaufteilung

Mit der Umsetzung der Darstellung der Geraden, der Hashwerte einzelner Regionen und der Möglichkeit der Verkürzung des Hashes ist es nun möglich zu beobachten, wie der Raum durch die Prototypen aufgeteilt wird. Eine weitere positive Eigenschaft des entstandenen Tools ist, dass beim Verschieben der Prototypen sofort die Geraden verändert werden und somit Schritt für Schritt die Veränderung der Raumaufteilung sichtbar wird. Eine neue Anforderung für spätere Arbeiten stellt hierbei die Darstellung der entstehenden Regionen beim Verkürzen der Hashwerte dar. Dieses Thema ist jedoch sehr anspruchsvoll und rechenintensiv. Zur Verdeutlichung dieses Fakts kann als Beispiel die Darstellung

der Hashwerte der Regionen genommen werden. Zum Bestimmen der richtigen Position der Beschriftung und weiterhin zur Berechnung des Volumens der Region wurde ein Geometrie-Paket benutzt, dass durch den Schnitt des Raumes mit den Geraden schließlich eine Liste von Regionen liefert, die entstanden sind. Diese werden dann einzeln beschriftet. Wird diese Berechnung auch ausgeführt, wenn mit der Maus ein Prototyp verschoben wird, so beginnt die Darstellung, aufgrund der Berechnung der entstehenden Region, zu stocken. Deshalb wird die Darstellung der Beschriftung während des Verschiebens ausgeschalten und beim Loslassen des Punktes wieder aktiviert. Eine Berechnung der entstehenden Regionen bei verkürzten Hashwerten würde einen ähnlichen Algorithmus benutzen und somit würde beim Verschieben der Prototypen wiederum ein Performanzverlust entstehen und zu einer stockenden Darstellung führen. Deshalb sind weitere Überlegungen und neue Herangehensweisen nötig, um das Verschmelzen der Regionen umzusetzen und somit die reale Raumaufteilung beim Verkürzen des Hashes darzustellen.

Nachdem nun die Anforderungen an die Darstellung der Raumaufteilung umgesetzt wurden und die Raumaufteilung analysiert werden kann, bleibt als letzter Anforderungsbereich noch das Einbinden von Beispieldaten und damit die Darstellung der Bucketauslastung auszuwerten.

6.2.4 Einbinden von Beispieldaten

An die Auswertung der Bucketauslastung wurden zwei Anforderungen gestellt, die zum einen das Einbinden von Beispieldatenbasen ermöglichen sollen und zum anderen anhand dieser Daten die Bucketauslastung dargestellt werden sollen. Die Umsetzung dieser beiden Anforderungen wurde in Tabelle 6.4 ausgewertet.

Anforderung	**Umsetzung**	
Einfügen von Beispieldaten	Mit den Schaltflächen „Load“ und „Unload“ können Datenbasen in den Datenraum geladen werden und wieder gelöscht werden. Die Auswahl der Datenbasen erfolgt mit Hilfe eines Java-Auswahldialogs, dem `JFileChooser`.	✓
Visualisierung der Bucketauslastung	Die Umsetzung der Visualisierung der Bucketauslastung wurde als Balkendiagramm visualisiert. Dieses zeigt die Anzahlen an Buckets mit einer bestimmten Menge an Datenpunkten. Dadurch ist ersichtlich wie viele Buckets übermäßig viele oder nur wenige Datenpunkte enthalten.	✓

Tabelle 6.4: Auswertung der Anforderungen an das Einfügen von Beispieldaten

Die beiden Anforderungen an das Einbinden von Beispieldaten wurden vollständig umgesetzt und somit ermöglicht das Tool eine Analyse der Bucketauslastung anhand der Beispieldaten durchzuführen. Um dem Nutzer das Auswählen

der Dateien zu erleichtern, wurde weiterhin ein Filter auf den Auswahldialog angewendet, der nur Ordner und Dateien mit der Dateiendung einer Datenbasis von QuEval zulassen. Dadurch wird das versehentliche Laden von nicht unterstützten Dateien vorgebeugt.

In den vorherigen Abschnitten wurde noch einmal die Umsetzung der einzelnen Anforderungen betrachtet und somit wurden die funktionalen Anforderungen an das Tool ausgewertet. Diese Betrachtung zeigt zwar, dass das Tool die Funktionen umsetzt, jedoch nicht, wie einfach es zu bedienen ist und ob es auch wirklich bedeutende Erkenntnisse gewinnen lässt. Deshalb sollen erste Erkenntnisse über die Raumaufteilung bei permutationsbasierten Indexverfahren, die durch das Tool gewonnen wurden, betrachtet werden.

6.3 Analyse der Raumaufteilung

In diesem Abschnitt soll eine Analyse der Raumaufteilung vorgenommen werden. Dafür wurden anhand des Tools verschiedene Szenarien konstruiert, die Aufschluss über eine optimierte Raumaufteilung für bestimmte Anfragetypen geben soll. Die dabei betrachteten Anfragetypen beschränken sich dafür zunächst auf die Anfragetypen Exakt-Match, Bereichsanfragen und die Suche nach den nächsten Nachbarn. Zuerst werden dafür im nächsten Kapitel einige grundlegende Erkenntnisse vorgestellt, die bei der Arbeit mit dem Tool entstanden sind.

6.3.1 Grundlegende Erkenntnisse

Bei der Betrachtung der Raumaufteilung anhand der Visualisierung durch das Tool gibt es einige Eigenschaften der entstehenden Regionen. Es ist zu beobachten, dass durch verschiedene Positionen der Prototypen verschiedene Anzahlen an Regionen entstehen. Von der mathematischen Seite her, gibt es bei n Prototypen $n!$ mögliche Permutationen. In Abbildung 6.2 wurden beispielsweise drei und vier Prototypen in den Datenraum eingefügt und die Raumaufteilung dargestellt.

Im linken oberen Beispiel gibt es genau 12 Bereiche in die sich der Raum teilt. Bei einer anderen Positionierung der einzelnen Prototypen können jedoch 18 Bereiche entstehen, wie im rechten oberen Beispiel zu sehen ist. Theoretisch gibt es jedoch 24 verschiedene Permutationen, die durch die vier Prototypen entstehen können.

Eine nähere Betrachtung dieser Angelegenheit zeigt, dass dieses Phänomen nur auftreten kann, wenn die Anzahl der Prototypen um mindestens eins größer ist als die vorliegende Dimension des Raums. Werden nur drei Prototypen eingefügt, so sind alle möglichen Permutationen als Beschriftung vorhanden (vgl. Abbildung 6.2 unten).

Dieses Phänomen lässt sich durch die Lage der Prototypen erklären und schließlich auch auf die Dimensionalität zurückführen. Im zweidimensionalen Raum gibt es nur diese eine Ebene. Wird darin der Abstand eines Punktes zu den Prototypen bestimmt, so gibt es im linken oberen Beispiel in Abbildung 6.2 keinen Bereich, der mit P2P3 beginnt. Das ist darauf zurückzuführen, dass es immer

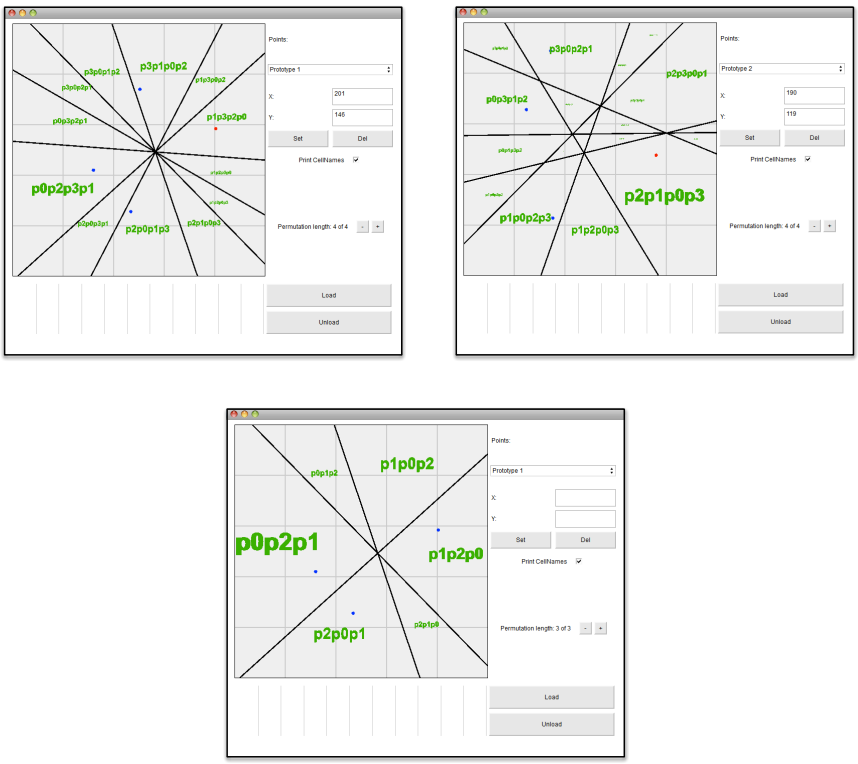


Abbildung 6.2: Verschiedene Anzahlen an Regionen

zwei andere Prototypen gibt, die näher zu P2 sind als P3. Wird gedanklich ein ähnliches Szenario im dreidimensionalen Raum betrachtet, bei dem die Prototypen nicht in einer Ebene liegen, so wird es Bereiche geben, deren Permutation mit P2P3 beginnt. Wird im dreidimensionalen Raum ein weiterer Prototyp eingefügt, so werden wahrscheinlich wieder einige Permutationen nicht vorhanden sein. Mathematisch lässt sich dieser Sachverhalt folgendermaßen beschreiben: sind die Vektoren von einem Prototyp zu allen anderen Prototypen linear abhängig, so entstehen weniger Regionen als mögliche Permutationen vorhanden sind und es gibt Permutationen, die nicht als Region vorhanden sind. Dieser Sachverhalt sollte jedoch weiter untersucht werden, wenn eine dreidimensionale Visualisierung der Regionenaufteilung im Tool umgesetzt wurde.

Dieser Fakt ist jedoch wichtig bei der Auswertung der Bucketauslastung. Wird davon ausgegangen, dass alle Permutationen als Bereiche vertreten sein sollen, so würde es bei steigenden Prototypanzahlen übermäßig viele leere Buckets geben. Dieser Fakt wurde bei der Entwicklung des Tools nicht betrachtet und somit ist die reale Anzahl an leeren Buckets wesentlich kleiner als bei der Anzeige des Tools. Dieser Fakt sollte bei der nun folgenden Auswertung mit einbezogen werden.

Eine weitere generelle Beobachtung bezieht sich auf die Funktionalität die Länge der Permutation zu verkürzen. Wie in Abschnitt 4.3 dargestellt wurde, führt die Verkürzung der Permutation dazu, dass sich einige Bereiche verschmelzen und sich die Regionen den Voronoi-Zellen annähern. Die Verkürzung der Permutation

um eine Stelle hat keine Auswirkung auf die Bucketauslastung und auch nicht auf die Raumaufteilung, da der letzte Prototyp der Permutation eindeutig durch die restlichen Prototpyen bestimmt wird. Diese Veränderung wirkt sich einzig und allein auf die Beschriftung der Bereiche aus, wie in Abbildung 6.3 zu sehen ist.

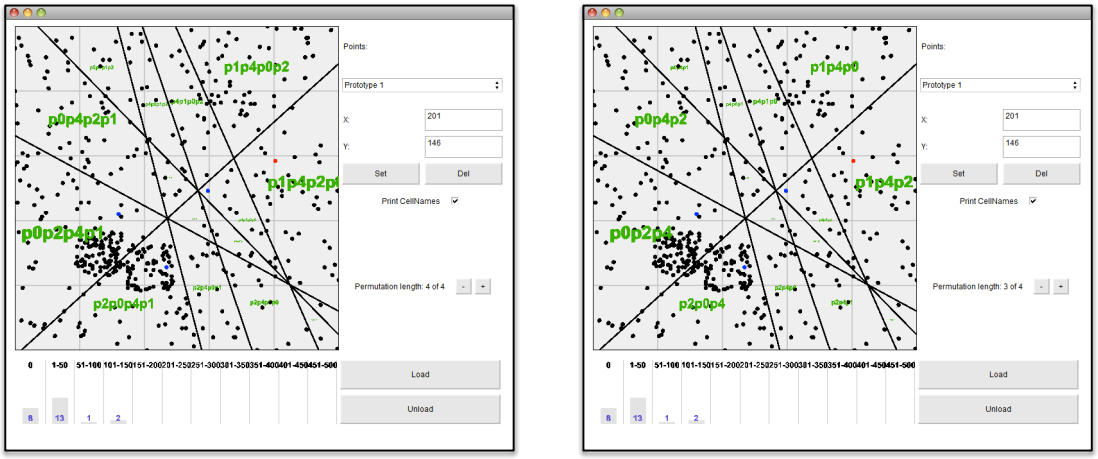


Abbildung 6.3: Verkürzung der Permutation

Wird die Länge der Permutationen jedoch weiter verkürzt, so sinkt die Anzahl an leeren Buckets, wie im linken Bild der Abbildung 6.4 zu sehen ist. Durch die Verkürzung der Permutation wurden weiterhin dünn besiedelte Regionen verschmolzen, was daran zu erkennen ist, dass jetzt ein Bucket mehr als zuvor 51-100 Datensätze enthält und die Anzahl an Regionen mit 1-50 Datenpunkte gesunken ist. Auch die Anzahl an leeren Buckets ist gesunken, wobei schon in Abbildung 6.3 keine realen leeren Buckets vorhanden waren, sondern nur die Permutation nicht vorhanden war.

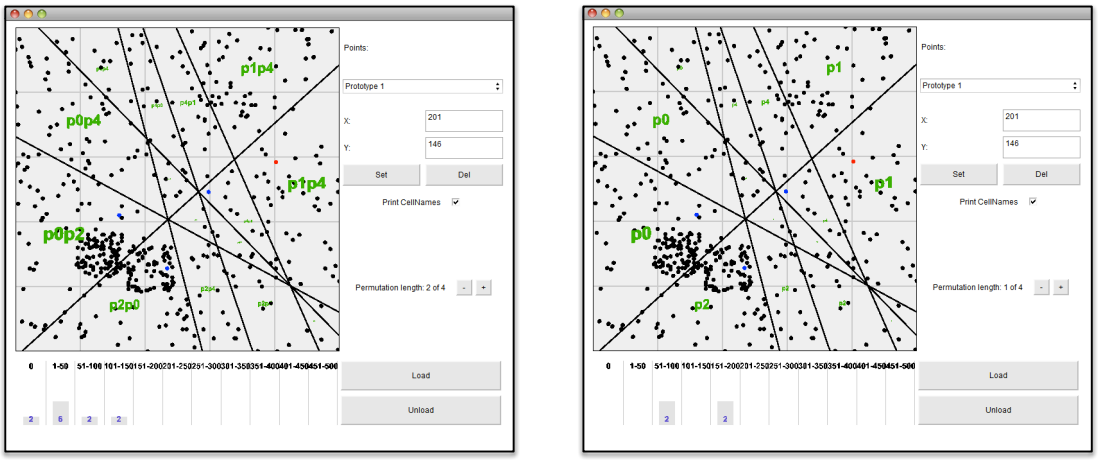


Abbildung 6.4: Weitere Verkürzung der Permutation

Beim weiteren Verkürzen der Permutation auf die Länge 1 verschwinden alle leeren Buckets, da jetzt nur noch Voronoi Zellen vorliegen. Diese vier Regionen beinhalten zweimal 51-100 Datenpunkte und zweimal 151-200 Datenpunkte. Dadurch ist eine gleichmäßigere Befüllung der Bereiche gegeben, als bei voller Länge der Permutation. Durch das Verkürzen der Permutation sinkt die Selektivität, da kleinere Regionen zu größeren Regionen zusammengesetzt werden und somit mehr Punkte in den Buckets enthalten sind. Diese sinkende Selektivität kann für einige Anfragetypen positiv sein, kann sich aber auch negativ auf die Anfragegeschwindigkeit auswirken. Diese und weitere Erkenntnisse sollen in den nächsten Abschnitten für jeden der drei betrachteten Anfragetypen zusammengefasst werden.

6.3.2 Exakt-Match

Die Anfragebearbeitung vom permutationsbasierten Locality Sensitive Hashing findet in zwei Stufen statt. Zuerst wird mittels des Hashwertes des Anfragepunktes je nach Anfragetyp ein oder mehrere Buckets ausgewählt und die darin enthaltenen Punkte in eine Kandidatenliste eingeordnet. Als zweiter Schritt folgt dann ein sequenzielles Durchsuchen der Kandidatenliste, um die richtigen Anfrageergebnisse aus der Liste zu filtern und auszugeben.

Bei Exakt-Match soll als Ergebnis nur ein Datenpunkt ausgegeben werden, auf den die Anfrage passt. Deshalb ist es von Vorteil, wenn beim Suchen des passenden Punktes in der Kandidatenliste nur wenige weitere Punkte vorhanden sind. Somit ist es von Vorteil, wenn das benutze Verfahren eine hohe Selektivität besitzt und dadurch nur kleine Kandidatenlisten durchsucht werden müssen. Als Erkenntnis aus dem vorherigen Abschnitt, sinkt die Selektivität beim Verkürzen des Hashwertes und somit sollte das Verkürzen der Permutation nicht bei Exakt-Match Anfragen genutzt werden.

Eine weitere Erkenntnis bezieht sich auf das Setzen der Prototypen. Vorallem bei vorliegenden Clustern ist das richtige Setzen der Prototypen von großer Wichtigkeit für Exakt-Match Anfragen. Aufgrund der Lage der Prototypen können verschiedenste Raumaufteilungen entstehen. Raumaufteilungen, die das Cluster gut unterteilen, wirken sich hierbei aufgrund der daraus resultierenden großen Selektivität gut auf die Anfragegeschwindigkeit aus. Liegen die Prototypen jedoch ungünstig im Raum kann es passieren, dass das gesamte Cluster in einer Region liegt und somit Anfragen auf diese Region mit viel Aufwand verbunden sind.

Normalerweise richtet sich das permutationsbasierte Verfahren nach der Datenverteilung. Deshalb ist es relativ wahrscheinlich, dass auch Punkte innerhalb des Clusters als Prototypen bestimmt werden. Dieses Verhalten ist jedoch nicht immer zielführend. In Abbildung 6.5 wurden einige Raumaufteilungen von Datenbasen mit Clustern dargestellt.

In Bild a) und b) ist zu sehen, dass durch das Setzen der Prototypen genau in das Cluster nur dessen Mitte gut unterteilt ist und viele dicht besetzte Bereiche entstanden sind. Dies ist auch im Balkendiagramm ersichtlich. Beim Verschieben der Punkte aus dem Cluster heraus, werden die inneren kleinen Bereiche groß-

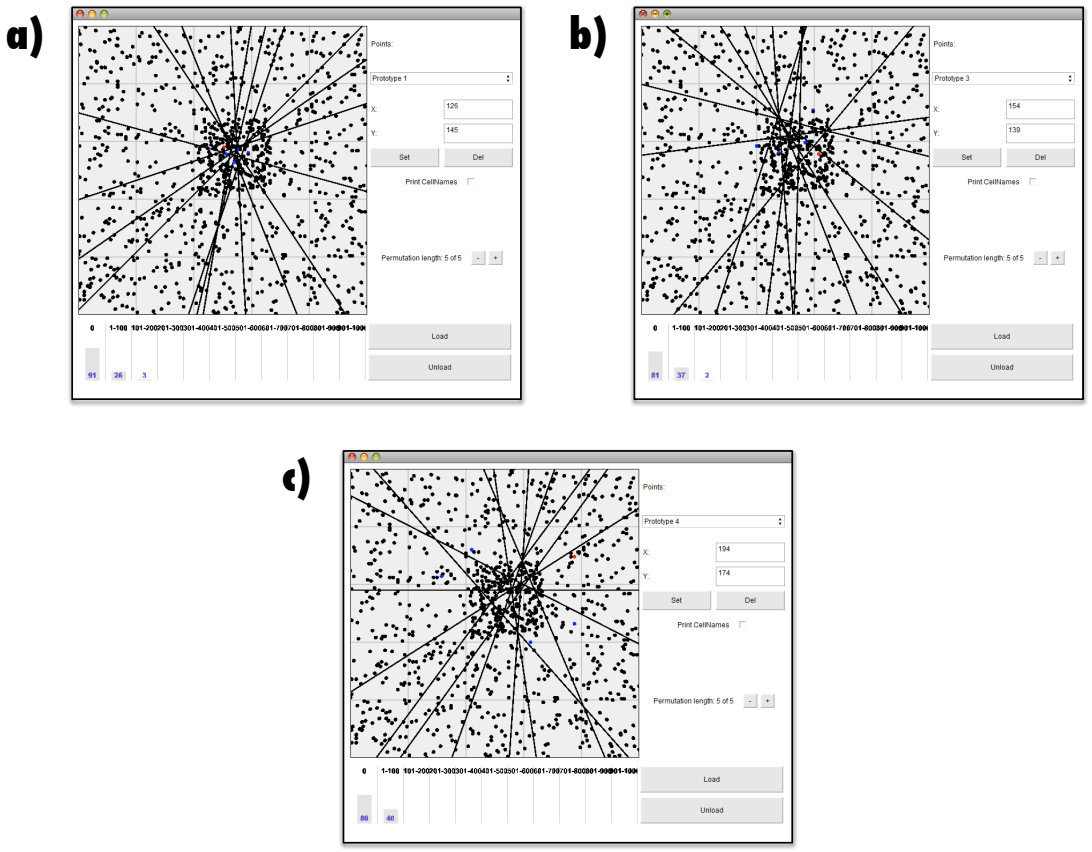


Abbildung 6.5: Setzen der Prototypen bei Clustern

räumiger und die Bucketauslastung wird ausgeglichener. Werden die Prototypen außerhalb des Cluster gesetzt, aber immer noch um dieses herum, so wird sowohl das Cluster als auch der restliche Raum gut unterteilt (vgl. c) aus Abbildung 6.5). Dadurch gibt es schließlich 40 Bereiche von denen jeder 1-100 Datenpunkte enthält. Als Erkenntnis aus diesem Sachverhalt, soll gezeigt werden, dass es für Exakt-Match Anfragen von Vorteil ist, wenn die Prototypen um ein vorhandenes Cluster gesetzt werden, anstatt genau in dieses gesetzt zu werden.

Das Behandeln von Clustern nimmt bei der Optimierung der Raumaufteilung bei Exakt-Match Anfragen eine sehr wichtige Rolle ein, da die Cluster die Anfragegeschwindigkeit stark beeinflussen können. Diese Cluster sind jedoch für Bereichsanfragen relativ uninteressant, weil es bei diesen Anfragen nur darum geht, ob die Punkte in einem bestimmten Bereich liegen.

6.3.3 Bereichsanfragen

Bereichsanfragen haben als Ergebnis eine Menge an Punkten, die in einem bestimmten Bereich liegen. Befindet sich innerhalb dieses Bereiches ein Cluster, so müssen ebenso alle Punkte des Clusters zurückgegeben werden. Somit ist eine feine Unterteilung des Clusters für Bereichsanfragen nicht vonnöten. Zur Unter-

stützung der Bereichsanfragen sollte eine Unterteilung des Raumes durch Geraden vorgenommen werden, die orthogonal zu einer Dimension stehen. Die Unterteilung des Raumes für Bereichsanfragen wurde in Abbildung 6.6 dargestellt.

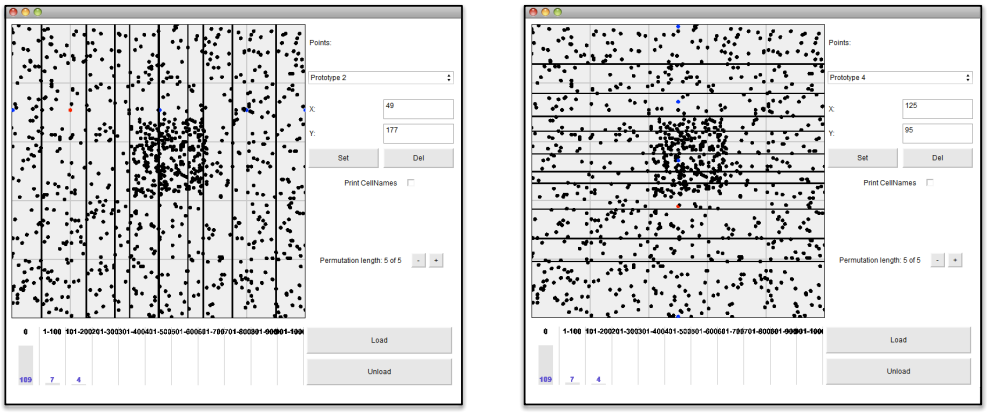


Abbildung 6.6: Setzen der Prototypen für Bereichsanfragen für eine Dimension

Damit die Geraden auch senkrecht im Raum verlaufen, müssen sie auf einer Linie angeordnet werden. Zum Finden der richtigen Position muss nur bedacht werden, dass sich die entstehende Gerade genau in der Mitte der beiden Prototypen befindet. Ist eine ungefähre Verteilung der Anfragebereiche der Bereichsanfragen bekannt, so kann durch gezieltes Setzen der Prototypen die Anfragegeschwindigkeit für den Spezialfall optimiert werden. Die dargestellten Raumaufteilungen behandeln jedoch nur jeweils eine Dimension. Die Frage ist jedoch, ob eine Unterstützung von Bereichsanfragen gleichzeitig über beide Regionen möglich ist. Zur Aufteilung des Raumes in Quadrate sollten die Prototypen selbst als Ecken eines Quadrates angeordnet werden. Wie beim Betrachten der entstehenden Raumaufteilung in Abbildung 6.7 zu sehen ist, teilen die Geraden den Raum nicht nur entlang der einzelnen Dimensionen, sondern auch diagonal.

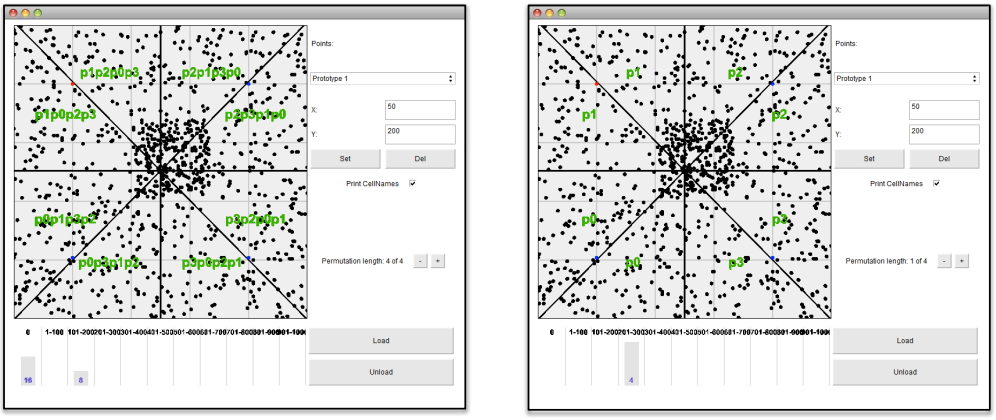


Abbildung 6.7: Prototypenposition für beide Dimensionen von Bereichsanfragen

Bei der Aufteilung der Region mit den vier quadratisch angeordneten Prototypen entstehen acht dreieckige Regionen, wobei jeweils zwei von diesen ein Viereck bilden. Dadurch ist es naheliegend, dass ein Verschmelzen dieser Dreiecke zum Vorteil für Bereichsanfragen wäre. Beim Verkürzen der Permutation auf die Länge Eins, ist ersichtlich, dass nur noch die Vierecke übrig bleiben würden. Somit haben wir jede Dimension in zwei Bereiche unterteilt. Werden nun weitere Prototypen hinzugefügt, so wird das Gitter noch feiner, wobei jedoch die Übersichtlichkeit sinkt, da auch weitere Diagonalen hinzukommen. Zur Bewahrung der Übersichtlichkeit sollte beim Verbessern des Tools die Auswirkung des Verkürzens der Permutation auf die entstehenden Geraden dargestellt werden. Als letztes soll bei der Betrachtung der Verbesserung der Raumaufteilung durch gezieltes Setzen der Prototypen die Erkenntnisse in Bezug auf die Suche nach den nächsten Nachbarn ausgewertet werden.

6.3.4 kNN-Anfragen

Für die Suche nach den nächsten Nachbarn ist ein feingranulares Aufteilen des Raumes nicht unbedingt von Vorteil. Dies liegt daran, dass viele umliegende Bereiche nach weiteren nächsten Nachbarn durchsucht werden müssen, wenn in der Region Anfrage nicht genug Punkte enthalten sind. Diese Suche in benachbarten Regionen ist jedoch nur möglich, wenn bekannt ist, welche Permutation die umliegenden Regionen haben. Deshalb ist die Möglichkeit der Beschriftung der Regionen von großer Bedeutung für diese Analyse. In Abbildung 6.8 wurde eine beispielhafte Raumaufteilung inklusive der Beschriftung dargestellt.

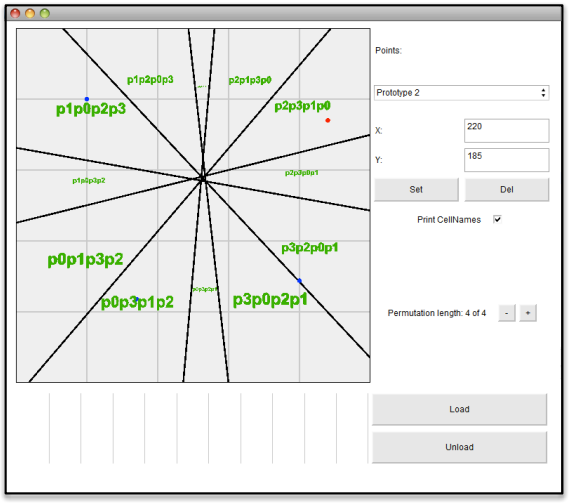


Abbildung 6.8: Setzen der Prototypen für kNN-Anfragen

Es ist ersichtlich, dass sich benachbarte Regionen in ihrer Permutation sehr ähnlich sind. Die Permutationen unterscheiden sich nur in einem getauschten Prototypenpaar, dessen Prototypen sich in der Permutation direkt nebeneinander befinden. So hat zum Beispiel P3P0P2P1 die benachbarte Zelle P3P2P0P1. Dies

lässt sich darauf zurückführen, dass die Gerade zwischen den beiden Zellen die Gerade zwischen den Punkten P0 und P2 ist. Nach dem Überqueren der Geraden ist der Abstand zum Punkt P2 kleiner als der zum Punkt P0. Dadurch tauschen diese beiden Prototypen nur die Position in der Permutation.

Werden nun die nächsten Nachbarn eines Anfragepunktes gesucht, so wird als erstes die Region untersucht, in die der Punkt fällt. Sind in diesem nicht genügend Punkte vorhanden, so können weiterhin die Regionen zur Kandidatenliste hinzugefügt werden, dessen Permutation sich um zwei getauschte Prototypen von der Permutation des Anfragepunktes unterscheiden.

Liegen die Anfragepunkte jedoch ungünstig, so kann es passieren, dass mehr Regionen durchsucht werden müssen, als die mit dem Zweiertausch der Prototypen. In Abbildung 6.8 müssen alle Bereiche durchsucht werden, wenn der Anfragepunkt in der Mitte des Raumes liegt und die exakten nächsten Nachbarn zu finden sind. Das ist dem Umstand geschuldet, dass sich die Geraden in der Mitte kreuzen und somit in jedem der Bereiche ein nächster Nachbar liegen kann.

6.4 Zusammenfassung

In diesem Kapitel wurde zu allererst das entstandene Tool vorgestellt und die einzelnen Elemente der Benutzeroberfläche erläutert. Anhand dieser Vorstellung und weiteren Implementierungsdetails wurden die Anforderung an den darzustellenden Raum, die Manipulation der Prototypen, die Darstellung der Raumaufteilung und das Einbinden von Beispieldaten nocheinmal aufgegriffen und deren Umsetzung beschrieben. Es wurden elf der insgesamt dreizehn Anforderungen vollständig umgesetzt, wobei Verbesserungen der einzelnen Umsetzungen im späteren Verlauf notwendig werden können. Die Anforderung an eine dreidimensionale Darstellung des Raumes wurde innerhalb dieser Arbeit nicht umgesetzt und benötigt noch weitere Betrachtungen. Auch die Verkürzung des Hashes wurde nur teilweise umgesetzt, so beeinflusst diese Funktionalität nur die Beschriftungen der Raumaufteilung und die Darstellung der Bucketauslastung.

Im darauf folgenden Abschnitt wurden dann erste Erkenntnisse in Bezug auf das Setzen der Prototypen zur Unterstützung der Anfragetypen Exakt-Match, Bereichsanfragen und kNN-Anfragen vorgestellt. Für Exakt-Match Anfragen wurde erläutert, dass das Verkürzen der Permutation aufgrund der sinkenden Selektivität nicht von Vorteil ist. Anhand von Beispielen wurde weiterhin gezeigt, dass Cluster am besten unterteilt werden, wenn die Prototypen um sie herum gesetzt werden, anstatt sie genau innerhalb der Cluster zu positionieren. Bereichsanfragen können durch das permutationsbasierte Verfahren unterstützt werden, indem die Prototypen auf einer Linie angeordnet werden, um eine Dimension zu unterstützen. Sollen jedoch zwei Dimensionen für Bereichsanfragen partitioniert werden, so müssen die Prototypen quadratisch im Raum positioniert werden und die Permutation auf die Länge Eins verkürzt werden. Für höhere Dimensionen sollte dann die Anordnung einem oder mehreren Hyperwürfel ähneln.

Schließlich wurde auch noch eine Erkenntnis in Bezug auf die Ausnutzung der Raumaufteilung bei der Suche nach den nächsten Nachbarn formuliert. Beim Per-

mutationsansatz unterscheiden sich die Permutation zweier benachbarter Zellen nur in einem, in der Permutation nebeneinander liegenden, Prototypenpaar, dass getauscht wurde. Dadurch ist es möglich den Suchraum sukzessiv zu erweitern, indem benachbarte Regionen betrachtet werden, wenn zuvor nicht k Kandidaten gefunden wurden.

Kapitel 7

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die vorliegende Arbeit behandelt das Thema der Darstellung der Raumaufteilung von permutationsbasierten Indexstrukturen. Es sollte ein Tool entwickelt werden, mit dem die entstehende Raumaufteilung durch die gesetzten Prototypen dargestellt werden kann. Dafür wurden zuerst verschiedene Anforderungen definiert, anhand derer der Funktionsumfang des Tools festgelegt wurde. Darin wurde unter anderem vorgestellt, wie der Raum dargestellt werden soll und wie die Prototypen in diesen gesetzt werden können. Weiterhin soll die Darstellung der Raumaufteilung und das Einbinden von Datenbasen ermöglicht werden, um damit die Bucketauslastung zu bestimmen und zu visualisieren.

Im darauf folgenden Kapitel wurde die Implementierung vorgestellt. Zuerst wurde das Design des Tools vorgestellt und eine Einführung in Java 3D gegeben. Java 3D wurde dafür genutzt den Datenraum, die Punkte, Prototypen und Raumaufteilung darzustellen, sowie die Bucketauslastung zu visualisieren. Des weiteren wurde ein Einblick in ausgewählte Funktionen verschiedener implementierter Klassen gegeben, durch die die formulierten Anforderungen umgesetzt wurden.

Nachdem dann die Implementierung vorgestellt wurde, konnte die Umsetzung der Anforderungen ausgwertet werden. Insgesamt sind elf der dreizehn formulierten Anforderungen umgesetzt wurden, wobei eine Anforderung nur teilweise und eine Anforderung überhaupt nicht umgesetzt wurde. Diese Arbeitspakete bleiben offen für weitere Projekte.

Abschließend wurden einige Erkenntnisse über das vorteilhafte Setzen der Prototypen für verschiedene Anfragetypen vorgestellt. Dabei wurde festgestellt, dass für Exakt-Match Anfragen bei vorhandenen Clustern die Prototypen möglichst um das Cluster herum gesetzt werden müssen, anstatt mitten in das Cluster herein. Weiterhin ist das Verkürzen der Permutation für Punktanfragen nicht von Vorteil. Für Bereichsanfragen hingegen ist das Verkürzen der Permutation sinnvoll, weil dadurch ein Gitternetz ähnlich dem des VA-File in den Raum gelegt werden kann. Diese Raumaufteilung entsteht, wenn die Prototypen quadratisch

in den Raum gesetzt werden und die Länge der Permutation auf einen Prototyp verkürzt wird. Das entstandene Tool zeigte weiterhin für die Suche nach den nächsten Nachbarn, dass die Permutationen benachbarter Regionen sich nur darin unterscheiden, dass zwei, in der Permutation nebeneinanderliegende, Prototypen getauscht wurden. Dadurch ist es möglich bei der Suche nach den k nächsten Punkten zum Anfragepunkt auch benachbarte Zellen zu durchsuchen. Diese ersten Ergebnisse und das entstandene Tool legen einen Grundstein für die Analyse der Raumaufteilung bei permutationsbasierten Indexverfahren. Ergänzende Betrachtungen des Themas benötigen weitere Arbeiten an der Implementierung und weitere Analysen der Raumaufteilung.

7.2 Ausblick

7.2.1 Verbesserung des Tools

Bei der Implementierung des Tools sind derzeit elf der dreizehn Anforderungen vollständig umgesetzt worden. Somit kann das Tool durch nachfolgende Projekte weiterentwickelt werden. Eine offen gebliebene Anforderung bezieht sich auf die Darstellung des Raumes durch das Tool. Es soll mit dem Tool ermöglicht werden, dass ein dreidimensionaler Raum dargestellt wird, in den dreidimensionale Daten eingefügt werden können. Diese Anforderung benötigt jedoch genauere Betrachtungen und sollte in Teilziele geteilt werden. Aufgrund der Nutzung von Java 3D ist eine Positionierung von Objekten im Raum nicht sehr aufwendig, da die meisten Funktionen vorliegen. Bei der näheren Betrachtung der entstehenden Raumaufteilung wird jedoch klar, dass nun keine Geraden mehr den Raum teilen, sondern ganze Flächen im Raum liegen. Diese Eigenschaft stellt hohe Anforderungen an eine übersichtliche Visualisierung der Raumaufteilung, dabei sollte jedoch der Zoom und das Gitternetz unterstützend wirken.

Eine weitere unvollständig umgesetzte Anforderung stellt die Darstellung der Raumaufteilung beim Verkürzen der Permutation dar. Zur Zeit wirkt sich das Verkürzen der Permutation nur auf die Beschriftung der Regionen und die Bucketauslastung aus. Wird nun auch noch die Auswirkung auf die Raumaufteilung dargestellt, so können bessere Schlüsse über den Einfluss der Verkürzung der Permutation betrachtet werden. Zur Umsetzung dieser Anforderung braucht es jedoch bessere Algorithmen, als die zur Zeit benutzen Funktionen zur Beschriftung der Zellen. Diese Funktionen sind sehr rechenintensiv und bedürfen einer weiteren Überarbeitung.

Eine weitere Anforderung betrifft die Darstellung der Bucketauslastung. Das jetzige Tool berechnet die Anzahl der Buckets durch die vorhandenen Protoypen und der daraus resultierenden Anzahl an Permutationen. Der Umstellung der Bucketauslastung wird jedoch eine intensive Recherche und Analyse vorausgehen. Dabei sollte bestimmt werden, welche Regionen wirklich entstehen können.

Neben den hier genannten zu implementierenden Anforderungen ist noch ein umfangreicheres Ziel denkbar, das die Verwendbarkeit des Tools erweitern soll. Das große Ziel ist, dass das entstandene Tool umgestaltet wird und es ermöglicht

wird, dass weitere Indexstrukturen eingebaut werden können und deren Raumaufteilung visualisiert werden kann. Mit Abschluss dieser Anforderung kann das Tool in das Framework QuEval eingebunden werden und die darin enthaltenen Indexstrukturen in das Tool hereingeladen werden können.

7.2.2 Auswertung der Raumaufteilung

Neben den vielen möglichen Erweiterungen des Tools, bleiben noch weitere Betrachtungen in Bezug auf das vorteilhafte Setzen der Prototypen offen. Dafür können zum Beispiel die Raumaufteilungen für bestimmte Datenverteilungen ausgewertet werden. Die vorliegende Arbeit beschränkt sich hierbei auf gleichverteilte Daten oder Datenbasen mit Clustern. Denkbar wäre also zum Beispiel günstige Positionen der Prototypen bei normalverteilten Daten zu bestimmen.

Außerdem sollte mit der Verbesserung der Implementierung heraus gefunden werden können, wie sich das Verkürzen auf die entstehende Raumaufteilung auswirkt. In dieser Arbeit wurde nur eine Tendenz der Bucketauslastung betrachtet; nicht jedoch wie die wirkliche Raumaufteilung aussieht. Dabei wäre es von Interesse, wenn bei bestimmten Anzahlen an Prototypen die optimale Länge der Permutation bestimmt werden könnte. Dadurch könnte das Verfahren gleichzeitig für mehrere Anfragetypen optimiert werden, indem die Prototypen in den Raum gesetzt werden und für Exakt-Match Anfragen die volle Permutationslänge benutzt wird und für beispielsweise Bereichsanfragen die Permutationslänge verkürzt wird. Wird weiterhin eine dreidimensionale Darstellung des Tools ermöglicht, so kann ermittelt werden, wie sich die Raumaufteilung bei steigenden Dimensionen verhält.

Literaturverzeichnis

[AHK01] Aggarwal, Charu C. ; Hinneburg, Alexander ; Keim, Daniel A.: On the surprising behavior of distance metrics in high dimensional space. In: *Proceedings of the International Conference on Database Theory (ICDT)*, Springer-Verlag, 2001, S. 420–434

[Bay97] Bayer, Rudolf: The universal B-tree for multidimensional indexing: general concepts. In: *Proceedings of the International Conference on Worldwide Computing and its Application (WWCA)*, Springer-Verlag, 1997, S. 198–209

[BBB+97] Berchtold, Stefan ; Böhm, Christian ; Braunmüller, Bernhard ; Keim, Daniel A. ; Kriegel, Hans-Peter: Fast parallel similarity search in multimedia databases. In: *SIGMOD Record* 26 (1997), Nr. 2, S. 1–12. – ISSN 0163–5808

[BBBK00] Böhm, Christian ; Braunmüller, Bernhard ; Breunig, Markus ; Kriegel, Hans-Peter: High performance clustering based on the similarity join. In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, ACM, 2000, S. 298–305

[BBK01] Böhm, Christian ; Berchtold, Stefan ; Keim, Daniel A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. In: *ACM Computing Surveys* 33 (2001), Nr. 3, S. 322–373

[BF79] Bentley, Jon L. ; Friedman, Jerome H.: Data structures for range searching. In: *ACM Computing Surveys* 11 (1979), Nr. 4, S. 397–409

[BKK96] Berchtold, Stefan ; Keim, Daniel A. ; Kriegel, Hans-Peter: The X-tree: An index structure for high-dimensional data. In: *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., 1996, S. 28–39

[BKSS90] Beckmann, Norbert ; Kriegel, Hans-Peter ; Schneider, Ralf ; Seeger, Bernhard: The R*-tree: An efficient and robust access method for points and rectangles. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*, ACM, 1990, S. 322–331

[BL97] Beis, Jeffrey S. ; Lowe, David G.: Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: *Proceedings*

of the International Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 1997, S. 1000–1006

[BM97] Bayer, Rudolf ; Markl, Volker: The UB-tree: Performance of multidimensional range queries / Institut für Informatik, TU München. 1997 (11). – Forschungsbericht

[CGFN08] Chavez Gonzalez, Edgar ; Figueroa, Karina ; Navarro, Gonzalo: Effective proximity retrieval by ordering permutations. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30 (2008), Nr. 9, S. 1647–1658

[DIIM04] Datar, Mayur ; Indyk, Piotr ; Immorlica, Nicole ; Mirrokni, Vahab S.: Locality-sensitive hashing scheme based on p-stable distributions. In: *Proceedings of the Symposium on Computational Geometry (SCG)*, ACM, 2004, S. 253–262

[FR89] Faloutsos, Christos ; Roseman, Shari: Fractals for secondary key retrieval. In: *Proceedings of the Symposium on Principles of Database Systems (PODS)*, ACM, 1989, S. 247–252

[GG98] Gaede, Volker ; Günther, Oliver: Multidimensional access methods. In: *ACM Computing Surveys* 30 (1998), Nr. 2, S. 170–231

[Gut84] Guttman, Antonin: R-trees: A dynamic index structure for spatial searching. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*, ACM, 1984, S. 47–57

[HK98] Hinneburg, Alexander ; Keim, Daniel A.: An efficient approach to clustering in large multimedia databases with noise. In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, AAAI Press, 1998, S. 58–65

[HS03] Hjaltason, Gisli R. ; Samet, Hanan: Index-driven similarity search in metric spaces. In: *ACM Transactions on Database Systems* 28 (2003), Nr. 4, S. 517–580

[IM98] Indyk, Piotr ; Motwani, Rajeev: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: *Proceedings of the Symposium on Theory of Computing (STOC)*, ACM, 1998

[KS97] Katayama, Norio ; Satoh, Shin'ichi: The SR-tree: An index structure for high-dimensional nearest neighbor queries. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*, ACM, 1997, S. 369–380

[LJF94] Lin, King I. ; Jagadish, Hosagrahar V. ; Faloutsos, Christos: The TV-tree: An index structure for high-dimensional data. In: *VLDB Journal* 3 (1994), Nr. 4, S. 517–542

[Nav02] Navarro, Gonzalo: Searching in metric spaces by spatial approximation. In: *VLDB Journal* 11 (2002), Nr. 1, S. 28–46

[Nol09] Nolan, John P.: *Stable distributions: Models for heavy tailed data.* Springer-Verlag, 2009

[OM84] Orenstein, Jack A. ; Merrett, Tim H.: A class of data structures for associative searching. In: *Proceedings of the Symposium on Principles of Database Systems (PODS)*, ACM, 1984, S. 181–190

[SD99] Sowizral, Henry A. ; Deering, Michael F.: The java 3D API and virtual reality. In: *IEEE Computer Graphics and Applications* 19 (1999), S. 12–15

[SKS01] Silberschatz, Abraham ; Korth, Henry F. ; Sudarshan, S.: *Database system concepts.* Bd. 4. The McGraw-Hill Companies, 2001

[SRF87] Sellis, Timos K. ; Roussopoulos, Nick ; Faloutsos, Christos: The R+-tree: A dynamic index for multi-dimensional objects. In: *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., 1987, S. 507–518

[SS10] Saake, Gunter ; Sattler, Kai-Uwe: *Algorithmen und Datenstrukturen: Eine Einführung mit Java.* 4. Auflage. dpunkt.Verlag, 2010

[SSH11] Saake, Gunter ; Sattler, Kai-Uwe ; Heuer, Andreas: *Datenbanken - Implementierungskonzepte.* 3. Auflage. MITP, Bonn, 2011

[SWS+00] Smeulders, Arnold W. M. ; Worring, Marcel ; Santini, Simone ; Gupta, Amarnath ; Jain, Ramesh: Content-based image retrieval at the end of the early years. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nr. 12, S. 1349–1380

[SYUK00] Sakurai, Yasushi ; Yoshikawa, Masatoshi ; Uemura, Shunsuke ; Kojima, Haruhiko: The A-tree: An index structure for high-dimensional spaces using relative approximation. In: *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., 2000, S. 516–526

[TYSK10] Tao, Yufei ; Yi, Ke ; Sheng, Cheng ; Kalnis, Panos: Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. In: *ACM Transactions on Database Systems* 35 (2010), Nr. 3, S. 1–46

[VCPF08] Valle, Eduardo ; Cord, Matthieu ; Philipp-Foliguet, Sylvie: High-dimensional descriptor indexing for large multimedia databases. In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, ACM, 2008, S. 739–748

[WB97] Weber, Roger ; Blott, Stephen: An approximation-based data structure for similarity search. 1997. – Forschungsbericht

[WJ96] White, David A. ; Jain, Ramesh: Similarity indexing with the SS-tree. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, IEEE Computer Society, 1996, S. 516–523

[WSB98] Weber, Roger ; Schek, Hans-Jörg ; Blott, Stephen: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., 1998, S. 194–205

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 16. April 2012

David Broneske