UNIVERSITÄT ZU LÜBECK

Aus dem Institut für Informationssysteme

der Universität zu Lübeck

Direktor: Prof. Dr. Ralf Möller

Algorithmen und Indexstrukturen für Top-k-Anfragen mit quasi-konvexen oder quadratischen Bewertungsfunktionen

Inauguraldissertation

zur

Erlangung der Doktorwürde

der Universität zu Lübeck

- Aus der Sektion Informatik / Technik -

vorgelegt von

Dipl.-Math. Peter K. Poensgen

aus Marmagen

Lübeck, 2019

Erster Berichterstatter:	Prof. Dr. Ralf Möller
Zweiter Berichterstatter:	Prof. Dr. Bernhard Thalheim

Tag der mündlichen Prüfung:	17.06.2019

Zum Druck genehmigt,
Lübeck, den 18.06.2019

…wer traversiert, verliert.

Kurzfassung

Eine Anfrage an ein Datenbanksystem spezifiziert für die Attribute von Objekten Qualifikationsbedingungen. Werden diese Bedingungen erfüllt, so wird das entsprechende Objekt in die Ergebnismenge aufgenommen. Bei einer Top-k-Anfrage werden Attribute mit Hilfe einer Bewertungsfunktion zu einem skalaren Wert zusammengefasst. Die Ergebnissemenge wird im Anschluss aufsteigend, oder absteigend sortiert, um dann die gemäß der Anfragebedingung k besten Objekte auszugeben.

Um Top-k-Anfragen effizient beantworten zu können, sollte vermieden werden, dass die gesamte Objektmenge durchsucht und jeder Rang berechnet wird. Man kann versuchen, die Objekte selbst, oder aber ihren Datenraum zu partitionieren, um so den Suchbereich zu verkleinern. Die entstandenen Partitionen lassen sich den Ebenen eines Entscheidungsbaumes zuordnen, mit dessen Hilfe die Ergebnismenge bestimmt werden kann.

Eine bekannte Methode effizienter Verarbeitungsstrategien auf der Basis von Entscheidungsbäumen ist das Branch-and-Bound-Prinzip. Damit es funktioniert, sind Schranken für jede Partition anzugeben, um eine begrenzende Suche zu ermöglichen. Das zugehörige Verfahren im Kontext von Top-k-Anfragen wird Branch-and-Bound Ranked Search, kurz BRS genannt.

Die verwendete Bewertungsfunktion ist unabhängig von ihrer praktischen Relevanz nur dann für das BRS-Verfahren geeignet, wenn erstens für das Bild jeder Partition unter der Funktion Schranken existieren, die sich dann zweitens auch effizient berechnen lassen. Die Bestimmung solcher Schranken spezieller Funktionsklassen bildet den Kern der vorliegenden Arbeit. Es wird untersucht, welche Funktionen die oben genannten beiden Bedingungen erfüllen und es wird gezeigt, wie sich die Schranken explizit berechnen lassen. Die angeführten Beispiele machen deutlich, warum diese Funktionen für die Praxis so bedeutsam sind. Neben dem Problem der Maximierung einer Bewertungsfunktion wird auch ihre Minimierung, also die Suche nach den k kleinsten Objekten im Fokus stehen. Beide vorgestellten Funktionsklassen sind geeignete Kandidaten für das Branch-and-Bound-Prinzip.

Einerseits wird in der vorliegenden Arbeit der notwendige theoretische Rahmen für den BRS-Algorithmus entwickelt, das Verfahren also sozusagen in einen mathematischen Kontext überführt, zum anderen erhält der Anwender des Verfahrens eine Anleitung, mit der sich entscheiden lässt, ob und wie seine Bewertungsfunktion genutzt werden kann.

Abstract

A request to a database system specifies qualifying conditions for the attributes of an object. If these conditions are met, the object will be included in the result. For a top-k query, attributes are aggregated to a scalar value, using a ranking function. The result is then sorted in ascending or descending order in order to single out the k best objects according to the request condition.

In order to answer a top-k query efficiently, it should be avoided that the entire object set is searched and the score of each data point is calculated. One can try to partition the object space, or the data points, in order to make the search area smaller. The resulting partitions can be assigned to the levels of a decision tree, with the help of which the result set can be determined path-wise.

A well-known method for efficient processing strategies based on decision trees is the branch-and-bound-principle. For this purpose, one must specify bounds for each partition to enable bounded search. The associated procedure in the context of top-k queries is called Branch-and-Bound Ranked Search, BRS for short.

Regardless of its practical relevance, the evaluation function used is only suitable for the BRS method if, firstly, there are bounds for the image of each partition under the function, which secondly can be calculated efficiently. The determination of such bounds for special functional classes forms the core of this work. We examine which functions fulfil these two conditions and we show how the bounds can be explicitly calculated. Further we will demonstrate why those functions are important for practical purposes. In addition to the problem of maximizing ranking functions, its minimization, i.e. the search for the k smallest objects, will also be in focus. Both featured functional classes are suitable candidates for the Branch-and-Bound-Principle.

On the one hand, the necessary theoretical framework for the BRS algorithm is developed in the present work, so that the procedure is transferred into a mathematical context. On the other hand, we give a hand to practitioners, to decide whether and how the evaluation function can be applied.

Danksagungen

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.
Als erstes bedanke ich mich bei meinem Doktorvater Prof. Dr. Ralf Möller für die Betreuung und Unterstützung bei auftretenden Fragen, sowie für viele hilfreiche Anregungen, die maßgeblich zum Gelingen beigetragen haben. Herrn Prof. Dr. Bernhard Thalheim danke ich für die kurzfristige Übernahme des zweiten Gutachtens, das mit einem nicht unerheblichen Arbeitsaufwand verbunden ist. Bei Prof. Dr. Martin Leucker möchte ich mich ebenfalls bedanken, der so freundlich war, den Vorsitz in der Prüfungskommission zu übernehmen.
Für anregende Diskussionen geht ein besonderer Dank an meinen Kollegen Dr. Stefan-M. Heinemann. Abschließend möchte ich mich ganz besonders bei meiner Frau Iris bedanken, die in den vergangenen Monaten auf viel gemeinsame Zeit verzichten musste und ohne deren Unterstützung diese Arbeit nicht möglich gewesen wäre.

Inhalt

1 Einleitung

Eine Top-k-Anfrage an ein Informationssystem ist eine Anfrage, die nach den „besten“ k Ergebnissen einer Datenmenge sucht. Der Nutzer einer solchen Anfrage ist demnach nicht an allen, möglicherweise sehr vielen Ergebnissen interessiert, sondern spezifiziert explizit, dass lediglich die besten k Objekte berechnet und angezeigt werden sollen. Zur Ermittlung einer solchen Bestenliste ist eine Form der Bewertung der zugrundeliegenden Objekte erforderlich, bei der alle Objekte anhand eines Bewertungskriteriums angeordnet werden. Das bedeutet, dass solche Anfragen die zu ermittelnde Ergebnismenge anhand einer zuvor bestimmten Sortierung (Rangfolge) liefern.

Typische Top-k-Anfragen an ein Onlinebuchungssystem für Hotels suchen beispielweise nach den drei günstigsten, oder den zehn gefragtesten Hotels einer Stadt, gemessen an der Anzahl ihrer Buchungen. Diese beiden einfachen Beispiele zeigen schon, dass Bewertungen sowohl eine aufsteigende, als auch absteigende Sortierung erforderlich machen können, so dass man es entweder mit einem Maximierungs- oder Minimierungsproblem zu tun hat.

Werden mehrere Attribute in eine Bewertung einbezogen, so kann sich die Bestimmung der Bestenliste als schwierig erweisen, sofern jedes Attribut für sich betrachtet werden soll. Ein typisches Beispiel wäre die Frage nach einem möglichst großen Apartment mit bestmöglicher Bewertung. Es stellt sich alsdann die Frage, ob der Anwender gegebenenfalls ein großes Apartment mit etwas niedrigerer Bewertung einem kleineren mit höherer Bewertung vorziehen möchte.

Um die Präferenzen von Nutzern für Anfragen mit mehreren Attributen abbilden zu können, werden sogenannte Bewertungsfunktionen (engl. scoring/ranking functions) genutzt, die die betreffenden Attribute zu einem einzigen Wert zusammenfassen. Dieser aggregierte Wert lässt sich dann wieder wie im Falle eines einzelnen Attributes auf- oder absteigend sortieren. Im oben genannten Onlinebuchungssystem würde man beispielweise die Zimmergröße und die Bewertung, gemessen an Sternen, additiv, also mit der Summenfunktion zusammenfassen und aus der absteigend sortierten Liste die k besten auswählen.

Die Komplexität einer Bewertungsfunktion steigt naturgemäß mit der Vielfalt an Anforderungen, die mit ihr modelliert werden sollen. Hat der Anwender eine genaue Vorstellung darüber, wie groß das gesuchte Apartment sein soll, was es höchstens kosten darf und bevorzugt er die Nähe zum Zentrum, wobei ihm der Preis wichtiger ist, als die Lage des Hotels, so hat man es mit nicht-monotonen Bewertungsfunktionen zu tun, die im Fokus der vorliegenden Arbeit stehen.

Eine der wesentlichen Anforderungen an die Verarbeitungsstrategie einer Top-k-Anfrage ist es, die Ergebnismenge auf effiziente Weise zu bestimmen. Effizienz ist neben Korrektheit eine der wesentlichen Anforderungen an jeden Algorithmus. Bei der Effizienzbetrachtung steht neben dem Speicherplatzbedarf (Speicher-Effizienz) meist die Frage nach der Laufzeit (Laufzeit-Effizienz) im Vordergrund. Auch in Zeiten immer schneller wachsender Rechnerkapazitäten sind Algorithmen gefragt, die ein Ergebnis in angemessener Laufzeit liefern können, da ineffiziente Verfahren auch Systeme mit noch so großen Kapazitäten an ihre Grenzen bringen können.

Eine naive Strategie zur Beantwortung einer Top-k-Anfrage besteht darin, alle Datenpunkte in die Anfragebearbeitung einzubeziehen, also den Funktionswert derselben zu berechnen und in dieser Werteliste die k größten oder kleinsten auszuwählen. Diese Strategie wird in Datenbanksystemen als *sequentielles Suchen* bezeichnet. Sie erfordert keine Sortierung der Gesamtliste der Objekte. Die Laufzeit zur Ermittlung der Top-k-Objekte einer N-elementigen unsortierten Liste wird beschrieben in einer Zeitfunktion von

(L1) $$\mathcal{O}(kN).$$

Das Ziel eines Algorithmus zur Beantwortung einer Top-k-Anfrage muss es also sein, möglichst wenige Datenpunkte zur Beantwortung der Anfrage heranziehen zu müssen, um die Laufzeit der sequentiellen Suche unterbieten zu können. Beim Laufzeitverhalten legen wir den Fokus stets auf die Daten- und nicht auf die Anfragekomplexität.

Um eine Top-k-Anfrage in effizienter Weise beantworten zu können, sind geeignete Zugriffspfade und nicht selten eine damit verbundene Partitionierung des Suchraumes erforderlich. Da man es im Allgemeinen mit Mehrattributzugriffen zu tun hat, werden mehrdimensionale Suchstrukturen, wie z.B. k-d-Bäume oder R-Bäume verwendet (siehe [1] und [2]), die bekanntlich ihren eindimensionalen Pendants, wie etwa den B-Bäumen überlegen sind. Für mehrdimensionale Zugriffspfade gelten im Allgemeinen ähnliche Anforderungen hinsichtlich der praktischen Tauglichkeit, wie bei eindimensionalen. In Falle der Top-k-Anfragen ist es allerdings sehr wichtig, dass die topologische Struktur, also die geometrische Anordnung der Datenpunkte im Datenraum berücksichtigt wird und erhalten bleibt. In der Praxis von Geodatenbanksystemen hat sich als Zugriffsstrategie der Einsatz sogenannter räumlicher Indizes in Kombination mit der Approximation der Datenpunkte bzw. der daraus resultierenden Geometrien durch achsenparallele, die Geometrien minimal umgebende Hyperrechtecke durchgesetzt. Ihre englische Bezeichnung *minimal bounding rectangle*, kurz MBR (Definition 3.1), verdanken sie der zweidimensionalen Variante, die Rechtecke im Raum bilden.

Welche der auf Basis dieser MBRs basierenden Zugriffsmethoden die beste ist, lässt sich wie so oft pauschal nicht beantworten. Zur Beantwortung von Anfragen zur Ermittlung einer Bestenliste (First-n-, Top-k-, oder Skyline-Anfragen) unter Verwendung des Branch-and-Bound-Prinzips hat sich die Verwendung von R-Bäumen als geeignet erwiesen, so dass der Trend in diesem speziellen Kontext in ihre Richtung geht. Bei der Ermittlung der Ergebnismenge mit Hilfe eines R-Baumes werden von der Wurzel ausgehend bis hin zur Blattebene und anhand der in jedem Knoten hinterlegten oberen oder unteren Schranke Verzweigungsschritte ausgeführt. Diese Schranken (engl. bounds) sind dabei abhängig vom zugrundegelegten MBR und der gewählten Bewertungsfunktion. Eine grundlegende Voraussetzung an jedes Verfahren ist es, dass solche Schranken überhaupt existieren und effizient berechnet werden können.

1.1 Zielsetzung

In der Arbeit „Branch-and-Bound Processing of Ranked Queries“ von Tao, Vagelis, Papadias und Papakonstantinou [3] wurde gezeigt, dass der Branch-and-Bound Ranked Search Algorithmus (BRS) eine effiziente Strategie zur Beantwortung von Top-k-Anfragen sowohl für monotone, als auch für nicht-monotone Bewertungsfunktionen darstellt. Der BRS-Algorithmus ist allerdings dann und nur dann anwendbar, wenn sich für jedes MBR des zugrundeliegenden R-Baumes im Falle der Maximierung eine obere Schranke, im Falle der Minimierung entsprechend, eine untere Schranke der Funktionswerte für die sich in den MBRs befindenden Datenpunkte angeben lässt. Mit verschiedenen Suchverfahren in Bäumen, wie etwa der Tiefensuche, der Breitensuche, oder unter Zuhilfenahme einer Prioritätswarteschlange (engl. priority queue) können die Top-k-Kandidaten ermittelt werden. Die Ausführungen in [3] lassen allerdings einige wesentliche Fragen offen, so z.B. wie man erkennen kann, ob die verwendete Bewertungsfunktion für das Verfahren überhaupt geeignet ist. Diese Fragen sollen im Kontext der hier vorliegenden Arbeit beantwortet werden.

Durch die in dieser Arbeit vorgestellten Methoden und Ergebnisse wird dem Anwender des BRS-Verfahrens ein Konzept an die Hand gegeben, welches u.a. die folgenden Fragen beantwortet:

(1) Welche Anforderungen sind an eine Bewertungsfunktion zu stellen, damit die Existenz eines Maximums bzw. Minimums gesichert ist?
(2) Welche Funktionsklassen eignen sich für das BRS-Verfahren in dem Sinne, dass sich eine obere (untere) Schranke effizient bestimmen lässt, und sie gleichzeitig eine Fülle für die Praxis relevante Szenarien abdecken können?
(3) Wie sollten ein R-Baum und seine MBRs konstruiert sein, damit unterschiedliche Punktverteilungen berücksichtigt werden und die Suche nach den Top-k-Kandidaten effizient erfolgen kann?

Damit besteht das wesentliche Ziel dieser Arbeit darin, Funktionsklassen anzugeben, die sich für die Verwendung des BRS-Verfahrens besonders eignen, weil ihre Funktionen auf nur endlich vielen Punkten eines Hyperrechtecks maximal bzw. minimal sind.

1.2 Ergebnisse

Ein Algorithmus wird für die Anwendung dann interessant sein, wenn er eine Vielzahl aus der Praxis relevanter Szenarien abdecken kann. Die zu betrachtende Klasse der Bewertungsfunktionen sollte demnach nicht zu speziell sein. Je weniger (mathematische) Bedingungen an sie gestellt werden, desto mehr Anwendungsszenarien sind mit ihren Funktionen darstellbar. Für jede dieser Funktionsklassen sollte allerdings stets eine einheitliche Methodik zur Verbesserung der Laufzeit (Indexstruktur und Partitionierung der Datenmenge) angegeben werden können, welche die gesamte Klasse und nicht nur einzelne Vertreter derselben unterstützt.

Stetige Funktionen

Frage (1) kann aus mathematischer Sicht mit dem Satz vom Minimum und Maximum (Satz 2.2) beantwortet werden. Ist eine Bewertungsfunktion stetig (Definition 2.12), so ist sichergestellt, dass sie stets ihre lokalen Extremwerte, also Maximum und Minimum, auf einer kompakten Menge annimmt, die sich aber im Allgemeinen nur mit viel Aufwand berechnen lassen. Für den BRS-Algorithmus genügt aber schon die Angabe einer oberen oder unteren Schranke der Funktion auf einem MBR. Da MBRs, wie wir zeigen werden, kompakt sind und Stetigkeit auf kompakten Mengen gleichmäßige Stetigkeit impliziert, kann man obere und untere Schranken explizit bestimmen. Allerdings wird sich zeigen, dass diese Schranken lediglich theoretische Relevanz besitzen, da ihre Berechnung in Abhängigkeit der Bewertungsfunktion im Allgemeinen sehr komplex werden kann und sie zusätzlich noch von mehreren Parametern abhängig sind.

Es liegt also nahe, neben der Stetigkeit noch andere Anforderungen an die zugrundeliegende Bewertungsfunktion zu stellen (Frage (2)), um die Berechnung der notwendigen Schranken zu vereinfachen bzw. erst zu ermöglichen. Wir untersuchen daher Funktionsklassen, deren Funktionen ihr Maximum (Minimum) auf dem Rand ihrer Definitionsmenge annehmen, was man als Maximumprinzip (Minimumprinzip) bezeichnet (siehe [4] und [5]).

Quasi-konvexe Funktionen

Eine monoton steigende Bewertungsfunktion nimmt ihr Maximum stets auf der „oberen rechten“ Ecke, also der Ecke eines Hyperrechtecks mit den größten Koordinaten an, was die Bestimmung der oberen Schranke für die jeweiligen Hyperrechtecke eines R-Baumes sehr vereinfacht. Ist Monotonie all ihrer Komponentenfunktionen (Definition 3.4) gegeben, so nimmt die Funktion ihr Maximum ebenfalls auf einer Ecke eines Hyperechteckes an (Maximumprinzip komponentenweiser-monotoner Funktionen auf Hyperrechtecken), die aber im Allgemeinen nicht die obere rechte sein muss.

Es stellt sich die Frage, ob es auch nicht-monotone Funktionen gibt, die vergleichbare Eigenschaften besitzen, die ihr Maximum an speziell ausgezeichneten und damit leicht identifizierbaren Punkten eines Hyperrechtecks annehmen. Es wird sich zeigen, dass quasi-konvexe Funktionen ihr Maximum stets auf den Ecken eines Hyperrechteckes annehmen und sie damit in Analogie zum einfachen Fall der Monotonie gute Kandidaten für die Anwendung des BRS-Algorithmus darstellen.

Ist eine Bewertungsfunktion quasi-konvex, nicht notwendigerweise stetig und ihr Definitionsbereich ein (kompaktes) Hyperrechteck, so lässt sich zeigen, dass sie ihr Maximum auf mindestens einer der Ecken dieses Hyperrechteckes annimmt (Theorem 2. Maximumprinzip quasi-konvexer Funktionen-Teil I). Theorem 2. (Maximumprinzip quasi-konvexer Funktionen-Teil II) zeigt, dass die Eigenschaft auf den Ecken eines jeden allgemeinen Hyperrechteckes maximal zu sein, für die Funktionsklasse der quasi-konvexen Funktionen charakteristisch ist.

Quadratische Funktionen

Ein Analogon zur Minimierung quasi-konvexer (quadratischer) Funktionen existiert leider nicht. Kompaktheit in Kombination mit der Quasi-Konvexität einer Funktion auf einem Hyperrechteck garantiert ein Maximumprinzip. Der Satz vom Minimum und Maximum sichert für quadratische Funktionen lediglich die Existenz eines Minimums, lässt aber keinerlei Rückschlüsse darauf zu, wo es zu finden ist. In dieser Arbeit wird erstmalig ein Verfahren zur Minimierung nicht-monotoner Bewertungsfunktionen entwickelt.

Um ein Minimumprinzip und damit eine geeignete untere Schranke für einen Top-k-Anfrage-Algorithmus herleiten zu können, betrachten wir aus der Klasse der Quadriken diejenigen quadratischen Polynome, deren Funktionsgraph ein elliptisches, oder ein hyperbolisches Paraboloid ist. Solche Funktionen lassen sich ebenfalls auf R-Bäume anwenden, da auch für den Algorithmus zur Minimierung eine Partitionierung erforderlich ist, deren Cluster aus Hyperrechtecken (MBRs) bestehen. Da die Komponentenfunktionen einer quadratischen Bewertungsfunktion in Top-k-Anfragen positiv, aber auch negativ gewichtet sein können, sind sie eine Verallgemeinerung der bekannten Nächste-Nachbarn-Anfragen.

Konstruktionsverfahren für R-Bäume

Wie bereits erwähnt, verwendet [3] einen Standard-R-Baum und verzichtet auf die Angabe eines Konstruktionsverfahrens. Um Frage (3) nach geeigneten Konstruktionsverfahren für R-Bäume nachzugehen, werden wir in dieser Arbeit drei verschiedene Partitionierungsverfahren auf der Basis einer „medianen Aufteilung“ vorgeschlagen, die den Raum in allen Dimensionen verfeinern. Bei der ersten wird der Raum auf dem Median der Koordinaten der Punkte geteilt, was den Vorteil hat, dass per Definition des Medians (Definition 5.1) man als Ergebnis eine nahezu Gleichverteilung der Datenpunkte auf den Hyperrechtecken des R-Baumes der letzten inneren Knotenebene erhält, sofern die Koordinaten der Datenpunkte paarweise verschieden voneinander sind. Bei der zweiten Variante wird der Raum und jedes MBR hälftig (mittig) geteilt, wodurch eine gleichmäßige Gitterpartitionierung entsteht. Im Anschluss werden für die in den Partitionen enthaltenen Datenpunkte MBRs gebildet. Das dritte Verfahren verläuft wie Verfahren 2, verzichtet allerdings auf die Bildung der MBRs, sofern man von einer gegebenen Gleichverteilung und einer zugleich großen Anzahl von Datenpunkten ausgehen kann. Eine konkrete Implementierung des letzten Verfahrens in einem relationalen Datenbanksystem, dem Microsoft SQL-Server, erfolgt im letzten Kapitel.

Das Neuartige an dieser Arbeit ist, dass das Branch-and-Bound-Prinzip im Kontext der Top-k-Anfragen erstmals im mathematischen Kontext beleuchtet wird. Es werden einerseits die konkreten Rahmenbedingungen angegeben, in dem das in [3] vorgestellte Verfahren sicher funktioniert, andererseits werden konkrete Funktionsklassen benannt und Methoden entwickelt, die eine explizite und effiziente Berechnung der Schranken für Funktionen dieser Funktionsklassen ermöglichen. Sowohl für das Maximierungs-, als auch erstmals für das Minimierungsproblem wird jeweils eine Funktionsklasse vorgestellt, die, wie wir zeigen werden, von großer praktischer Bedeutung ist. Mit den vorgestellten Ergebnissen lässt sich Quasi-Konvexität als eine Verallgemeinerung der Monotonie verstehen.

1.3 Veröffentlichungen

Zu den in dieser Arbeit vorgestellten Ergebnissen zu quasi-konvexen Bewertungsfunktionen wurde ein Paper mit dem Titel „Quasi-Convex Scoring Functions in Branch-and-Bound Ranked Search“ erstellt, das im Open Journal of Databases (OJDB) veröffentlicht wird.

1.4 Gliederung

Die vorliegende Arbeit besteht im Wesentlichen aus drei Hauptteilen (siehe Abbildung 1.1). Der erste Teil stellt die für das Verständnis der Arbeit notwendigen Grundlagen vor und ordnet das Thema in den Kontext der aktuellen Forschung ein. Im zweiten Teil, dem Kern der Arbeit, wird untersucht, welche Funktionsklassen für das BRS-Verfahren geeignet sind. Es werden zwei konkrete Klassen vorgestellt und auf das Verfahren angewandt, gefolgt vom dritten und letzten Teil, in dem eine Implementierung der vorgestellten Methoden in einem Microsoft SQL-Server Datenbanksystem mit anschließender Evaluierung erfolgt. Die Arbeit selbst schließt mit einem kurzen Ausblick auf mögliche Erweiterungen und weitere Forschungsthemen.

Algorithmen und Indexstrukturen für Top-k-Anfragen mit quasi-konvexen oder quadratischen Bewertungsfunktionen		
• Mathematische und algorithmische Grundlagen	• Stetige, quasi-konvexe und quadratische Bewertungs-funktionen	• Implementierung und Evaluation • Zusammenfassung und Ausblick

Abbildung 1.1: Aufbau der Arbeit

Die Gliederung orientiert sich an den im letzten Abschnitt vorgestellten Ergebnissen und wird im Folgenden kurz vorgestellt.

In **Kapitel 2** werden die für das Verständnis der Arbeit notwendigen mathematischen Konzepte und Begriffe vorgestellt. Der erste Teil erläutert die aus der Literatur bekannten Definitionen relationaler Datenbanken und Datenbanksysteme, die notwendig sind, um den zentralen Begriff der Top-k-Anfrage einführen zu können. Im zweiten Teil erfolgt die Einführung grundlegender Definitionen und Sätze aus der Geometrie und Analysis, die zur Herleitung der in dieser Arbeit vorgestellten Theoreme erforderlich sind.

Kapitel 3 stellt die algorithmischen Grundlagen bereit. Zu Beginn wird der R-Baum, sowie einer seiner Spezialisierungen, der R+-Baum, eingeführt, die im Kontext von Datenbanken beide der Kategorie der mehrdimensionalen Indexstrukturen angehören. Im Anschluss werden drei Verfahren zur Bestimmung von Bestenlisten vorgestellt, die das Branch-and-Bound-Prinzip verwenden: Die k-Nächste-Nachbarn-Suche, Skyline-Anfragen und Top-k-Anfragen. Von besonderem Interesse für unsere Arbeit ist das in [3] vorgestellte Verfahren, welches sich mit „Ranked Search“ und dem bekannten BRS-Verfahren befasst und zeigt, wie sich der BRS-Algorithmus zur Beantwortung von Top-k-Anfragen verwenden lässt. In ersten Teil dieses Aufsatzes wird gezeigt, wie dieser Ansatz

bekanntermaßen auf monotone Bewertungsfunktionen angewendet werden kann. Später werden neue Ansätze für nicht-monotone Bewertungsfunktionen diskutiert.

Kapitel 4 untersucht drei spezielle Funktionsklassen und zeigt auf, ob und wenn ja, wie sich die Funktionen dieser Klassen auf den BRS-Algorithmus anwenden lassen.

Die erste Klasse sind die stetigen Funktionen. Es wird gezeigt, dass Stetigkeit alleine nicht ausreichend ist, um obere und untere Schranken auf kompakten Mengen angeben zu können. Dazu bedarf es der strikteren Form der gleichmäßigen Stetigkeit einer Funktion. Wir werden untersuchen, ob die aus der gleichmäßigen Stetigkeit abgeleiteten Schranken auch für den praktischen Gebrauch geeignet sind.

Die zweite Klasse ist die der quasi-konvexen Bewertungsfunktionen, die nicht notwendigerweise stetig sein müssen. Sie erfüllen das Maximumprinzip auf den Ecken eines (konvexen und kompakten) Hyperrechtecks und ermöglichen damit eine effiziente Methode zur Berechnung von oberen Schranken für die Datenpunkte einer Partitionierung.

Quadratische (stetige) Bewertungsfunktionen (Klasse 3) eignen sich zur Bestimmung von Objekten, wo ein Teil der Attribute möglichst nahe, der andere Teil möglichst weit weg von den entsprechenden Koordinaten eines Anfragepunktes entfernt liegen soll. Zur Lösungsfindung muss die Bewertungsfunktion minimiert, also das ORDER BY-Kriterium absteigend sortiert werden. Sie sind auf dem Rand von achsenparallelen Hyperrechtecken minimal. Einzige Ausnahme: Das Minimum liegt im Inneren des Hyperrechteckes und ist Null. Für solche Funktionen kann die benötigte untere Schranke (Minimumprinzip) explizit angeben werden.

In **Kapitel 5** werden die in dieser Arbeit vorgestellten Methoden in einem Microsoft SQL-Server Datenbanksystem implementiert und das BRS-Verfahren für quasi-konvexe und stetige Bewertungs-funktionen der sequentiellen Suche gegenübergestellt. Die Untersuchungen machen deutlich, welche Parameter zu beachten sind, damit der BRS-Algorithmus der sequentiellen Suche überlegen ist.

Die Arbeit schließt mit der Zusammenfassung der wesentlichen Ergebnisse und einem Ausblick auf mögliche Erweiterungen und zukünftige Arbeiten (**Kapitel 6**).

2 Mathematische Grundlagen

Das Grundlagenkapitel besteht aus zwei Teilen. Im ersten Teil wird eine kurze Einführung in das Gebiet der relationalen Datenbanken gegeben und es werden Grundbegriffe bereitgestellt, die zur Einführung des zentralen Begriffes der Top-k-Anfrage erforderlich sind. In zweiten Teil werden grundlegende Konzepte der Geometrie und Analysis eingeführt, mit denen sich der Begriff der Stetigkeit von Funktionen definieren lässt. Es wird ferner darauf eingegangen, wie sich Funktionen mehrerer Variablen geometrisch interpretieren und veranschaulichen lassen. Es werden diejenigen topologischen Begriffe und Konzepte vorgestellt, die für das Verständnis des „Extremwertsatzes“ und des „Satzes über stetige Funktionen auf kompaktem Definitionsbereich“ erforderlich sind. Für eine Beweisführung der beiden Sätze wird auf die entsprechende Literatur verwiesen. Der erste Satz sichert die Existenz von Extrempunkten stetiger Funktionen, der zweite ist erforderlich, um für stetige Funktionen konkrete Schranken angeben zu können. Es werden all die mathematischen Begriffe aufgeführt, die in der Arbeit verwendet werden. Die Geometrie von Bewertungsfunktionen soll helfen, ein besseres Verständnis für die Ergebnisse aus Kapitel 4.2 und 4.3, also für quasi-konvexe und quadratische Bewertungsfunktionen zu bekommen. Sie bietet aber auch für die Bewertung möglicher Suchverfahren, wie sie in Kapitel 5 erfolgt, eine große Hilfestellung. Am Ende von Teil 2 wird gezeigt, wie man ein beliebiges Hyperrechteck mit affinen Abbildungen auf den Einheitswürfel abbilden kann. Diese Abbildungen sind besonders nützlich, da sie den Datenraum verkleinern und das Top-k-Anfrage-Problem auf den Einheitswürfel reduzieren. Ferner werden wir zeigen, dass die Rangeigenschaften der Datenpunkte invariant unter solchen affinen Abbildungen sind.

2.1 Relationales Datenmodell

Der folgende Abschnitt stellt der Vollständigkeit halber die Grundbegriffe des relationalen Datenmodells vor, die in der vorliegenden Arbeit Verwendung finden. Für eine ausführliche Darstellung verweisen wir auf die einschlägige Literatur, wie beispielsweise [6], [7], [8] oder [9].

Beispiel 2.1 Immobiliendatenbank

Als Beispiel für die Anwendung und die Motivation diverser Beispiele von Top-k-Anfragen wählen wir ein Onlinedatensystem für Immobilien, welches die folgenden Attribute für ein Immobilienobjekt bereithält: *Miete*, *Kaufpreis*, *Typ*, *Standort*, *Postleitzahl* (PLZ) *Breitengrad* (BG), *Längengrad* (LG), den *Wohnraum* (in m²), sowie eine eindeutige *Objektnummer* (ONr) eines jeden Tupels. Es sei ferner angenommen, dass diese Informationen dem Anwender über die Attribute der Relation *Immobilien* zur Verfügung gestellt werden.

2.1.1 Grundlagen des relationalen Datenmodells

Im relationalen Modell werden Daten als eine Sammlung von *Relationen* dargestellt. Sei $A = \{A_1, \dots, A_n\}$ eine endliche Menge von *Attributen,* D_i der zu jedem Attribut $A_i \in A$ zugehörige Wertebereich (Domäne), der ausschließlich aus atomaren, also nicht zusammengesetzten Werten besteht, dann ist eine *n-stellige Relation* R eine Teilmenge des kartesischen Produktes der Domänen, also

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n.$$

Ein *Tupel* ist ein Vektor $t = (t_1, \ldots, t_n) \in R$, der die Objekte unseres Interesses, z.B. eine spezielle Immobilie, über die Kombination seiner *Komponenten* $t_i \in D_i$, den Attributwerten beschreibt.

Beschränkt man eine Relation auf die Attribute mit reellwertigem Definitionsbereich, also $D_i \subseteq \mathbb{R}$, so kann man jedes Tupel der Relation als Punkt des Raumes interpretieren.

Definition 2.1 Datenpunkt und Datenraum

Sei R eine n-stellige Relation (Tabelle), $A = \{A_1, \ldots, A_m\}$ die Menge ihrer $m \leq n$ reellwertigen Attribute. Dann wird durch die Abbildung

$$\rho: R \to \mathbb{R}^m;\ (t_1, \ldots, t_n) \mapsto (x_1, \ldots, x_m)$$

jedes Tupel eindeutig auf einen m-dimensionalen Punkt projiziert, so dass sich jedes t mit seinem Bild $\rho(t)$ als *Datenpunkt* identifizieren lässt. Wir bezeichnen $\mathbb{R}^m$ als den zur Relation R zugehörigen *Datenraum*.

Man beachte, dass zwei verschiedene Tupel einer Relation auf denselben Datenpunkt abgebildet werden können, die Abbildung ρ im Allgemeinen also nicht injektiv ist. Aufgrund der vorangegangenen Definition werden wir im Kontext von Top-k-Anfragen häufig anstelle des Tupelbegriffs, den Begriff des Datenpunktes verwenden. Der Rest dieses Kapitels führt die Grundlagen der relationalen Abfragesprache SQL ein, die notwendig sind, um den in dieser Arbeit zentralen Begriff der Top-k-Anfrage vorzustellen.

2.1.2 Top-k-Anfragen

Die Structured Query Language (SQL) basiert auf der relationalen Algebra. Ihr Name suggeriert, dass es sich um eine reine Anfragesprache handelt. In Wahrheit ist SQL aber eine Anfrage-, Definitions- und Transaktionsprotokollsprache für relationale Datenbanken.

Der Sprachumfang von SQL lässt sich im Wesentlichen in drei Bereiche aufteilen:

- Data Definition Language (DDL)
- Data Control Language (DCL)
- Data Manipulation Language (DML)

Die DDL enthält die Befehle, die auf dem Schemakatalog des Datenbanksystems (DBS) agieren, hierzu gehören etwa Anweisungen wie CREATE TABLE, ALTER TABLE und DROP TABLE. Die Befehle aus DCL werden verwendet, um Transaktionen und Zugriffsrechte zu verwalten. Das sind Kommandos wie BEGIN, ROLLBACK und GRANT. Für uns ist die dritte Gruppe, d.h. die Data Manipulation Language entscheidend, welche auch das bekannte SELECT-Statement beinhaltet, das im Folgenden näher beschrieben werden wird.

Die nun folgenden Begriffe sind, wenn nicht anders angegeben, der Online-Dokumentation zum relationalen Datenbankmanagementsystem (RDBMS) DB2 von IBM[1] entnommen. Sie sind aber nicht nur für DB2, sondern ebenso für alle anderen kommerziellen im Einsatz befindlichen relationalen Datenbankmanagementsysteme, wie z.B. Oracle oder den Microsoft SQL-Server anwendbar.

Eine SELECT-Anweisung besteht aus sieben Klauseln, wovon lediglich die SELECT- und FROM-Anweisungen notwendig sind, die restlichen sind optional. Die Reihenfolge ist wie folgt festgelegt:

Syntax 2.1 Die sieben Hauptklauseln einer SQL-Anweisung

```
1  SELECT...
2  FROM …
3  WHERE …
4  GROUP BY …
5  HAVING …
6  ORDER BY …
7  FETCH FIRST …
```

Wir wollen im Folgenden nur auf die für diese Arbeit wesentlichen Klauseln 1, 2, 3 und 6 näher eingehen. Für alle übrigen sei der Leser auf die entsprechende Literatur verwiesen.

Durch den **SELECT**-Befehl erfolgt die Auswahl der Spalten, die der Ergebnistabelle angehören sollen und gegebenenfalls wie diese benannt werden sollen. Durch * oder Tabelle.* werden alle Spalten der Tabellen in die Ergebnistabelle übernommen. Da im Gegensatz zu einer Relation, die stets eine Menge aus Tupeln darstellt, eine Ergebnistabelle Duplikate enthalten kann, müssen diese, wenn gewünscht, durch die Anweisung SELECT DISTINCT eliminiert werden.

Im **FROM**-Teil erfolgt die Spezifikation der Tabelle(n), auf die sich die Anfrage bezieht (*Ausgangstabellen*), also die Spezifikation der für die Anfrage herangezogenen Relationen. Werden mehrere Relationen angebenden, so ist das Ergebnis auf dem kartesischen Produkt derselben definiert.

Mit Hilfe der **WHERE**-Klausel erfolgt die Spezifikation der Bedingung(en), welche die Zeilen der Ausgangstabelle erfüllen müssen, damit sie in die Ergebnistabelle übernommen werden (*Selektion*). Diese drei Klauseln einer SQL-Anfrage für sich allein werden auch als *SELECT-FROM-WHERE-Block*, (S*FW-Block*) bezeichnet.

Möchte man die Ergebniszeilen nach bestimmten Merkmalen sortieren, so geschieht dies in einer separaten Anweisung, der sogenannten **ORDER BY**-Klausel. Über eine solche wird angegeben, in welcher Reihenfolge die Tupel der Ergebnistabelle im Ergebnis angeordnet sein sollen. Es können mehrere Spalten angegeben werden, nach denen sortiert werden soll. Die Sortierung erfolgt dann entsprechende der Nennung der Spalten von links nach rechts, d.h. zuerst nach dem ersten, dann nach dem zweiten Attribut, usw. Zusammen mit dem SFW-Block ergibt sich die folgende Syntax.

[1] IBM, “DB2 Version 9 - SQL Reference Volume 2,” 1993-2006. [Online]. Verfügbar: ftp://ftp.software.ibm.com/ps/products/db2/info/vr9/pdf/letter/en_US/db2s2e90.pdf. [Zugriff: 19 März 2018].

Syntax 2.2 SFW-Block mit ORDER BY

SELECT *Projektionsliste* (Festlegung der Projektionsattribute)
FROM *Relationenliste* (zu verwendende Relationen, evtl. Umbenennungen)
[WHERE (Selektions-, Verbundbedingungen)]
ORDER BY *Attributliste* [ASC|DESC]

Zu beachten ist, dass die zu sortierenden Attribute in der SELECT-Klausel enthalten sein müssen.

Es können auch mehrere Spalten als Variablen in eine Funktion eingehen, mit diesen gerechnet und im Anschluss die Teilergebnisse zu einem einzigen Wert aggregiert werden, nach dem dann sortiert werden kann. Solche Funktionen nennt man multivariate Funktionen, die zusammen mit dem SWF-Block wie folgt notiert werden.

Syntax 2.3 SWF-Block mit Bewertungsfunktion

SELECT...
FROM…
[WHERE…]
ORDER BY $f(a_1, \dots, a_n)$ [ASC | DESC]

Viele Szenarien aus der Praxis erfordern solch eine Funktion $f(a_1, \dots, a_n)$, um die Anforderungen des Anwenders abbilden zu können. Ist ein Anwender an einem Apartment mit einer Wohnfläche von 120 m² mit einer monatlichen Miete von 600 Euro interessiert, so lassen sich entsprechende Objekte durch die folgende Anfrage ermitteln:

Beispiel 2.2

SELECT $*$
FROM $Immobilien$
WHERE $Typ = 'Appartment'$
ORDER BY $(Wohnraum - 120)^2 + (Miete - 600)^2$ ASC

Anfragen dieser Art werden von einem relationalen Datenbanksystem standardmäßig nicht unterstützt und es wird im weiteren Verlauf dieser Arbeit deutlich, dass dieses noch recht einfache Beispiel einer Bewertungsfunktion bereits hohe Anforderungen an die Anfragebeantwortungs-strategie stellt.

Sollen nicht alle zuvor sortierten Zeilen ausgeben werden, sondern lediglich eine bestimmte kleine Anzahl $k \in \mathbb{N}$, so hat man es mit einer Top-k-Anfrage zu tun. Top-k-Anfragen sind Anfragen, die eine Ergebnistabelle auf eine bestimmte Zeilenzahl beschränken.

Eine naive Möglichkeit nur die „besten“ k Ergebnisse auszugeben, also die kleinsten oder größten, ist es, alle Tupel zu sortierten und die Ausgabe nach den ersten k abzubrechen. Um eine limitierte Datenausgabe zu ermöglichen, wurde der SQL-Standard um die *fetch first*-Anweisung erweitert, die in IBM DB2, PostgreSQL, SQL-Server und nicht zuletzt auch in Oracle verfügbar ist. Die entsprechende SQL-Syntax variiert je nach DBMS und ist in der folgenden Abbildung dargestellt. Die Ausführung wird abgebrochen, sobald k Zeilen geladen wurden.

SELECT … FROM … [WHERE…] ORDER BY… FETCH FIRST k ROWS ONLY	SELECT TOP k ... FROM… [WHERE…] ORDER BY…	SELECT ... FROM (SELECT ... FROM ... ORDER BY ...) WHERE ROWNUM < 10	SELECT... FROM... [WHERE…] LIMIT k
DB2	SQL-Server	Oracle *	MySQL, PostgreSQL

Abbildung 2.1: Top-k-Anweisungen diverser RDBMSe

* Das Oracle DBS unterstützt die *fetch first*-Anweisung seit der Version 12c. Die hier dargestellte Anweisung kann alternativ unter Verwendung der Pseudo-Spalte ROWNUM genutzt werden, die die Zeilen einer Tabelle numeriert.

Nach diesen Vorbereitungen können wir den zentralen Begriff einer Top-k-Anfrage vorstellen. Wir übernehmen die vom SQL-Server verwendete Notation.

Definition 2.2 Top-k-Anfrage

Es sei R eine gegebene Relation mit $m \geq 1$ reellwertigen Attributen $A_1, \dots, A_m$. Es sei $f: \mathbb{R}^m \to \mathbb{R}$ eine Bewertungsfunktion, $f(t) = f(t_1, \dots, t_m)$ der aggregierte Wert eines Tupels $t \in R$. Dann verstehen wir unter einer Top-k-Anfrage eine SQL-Anfrage der Form

SELECT TOP k *Projektionsliste*
FROM R
[WHERE …]
ORDER BY $f(t)$ [ASC | DESC]

Es sei an dieser Stelle darauf hingewiesen, dass im Gegensatz zur angegebenen Definition aus [10] sowohl eine Minimierung als auch eine Maximierung der Bewertungsfunktion zugelassen wird. Die Ergebnismenge kann also entweder aufsteigend (ASC), oder absteigend (DESC) sortiert werden. Im Fall $m = 1$ liegt zwar keine Aggregation von Werten vor, es kann aber dennoch erforderlich sein, auch auf nur ein Attribut eine Funktion anzuwenden.

Im Allgemeinen besteht die Möglichkeit, auch mehrere Relationen in einer solchen Anfrage zu verwenden, was zum Begriff der Top-k-Join-Anfrage führt, oder auch Bewertungsfunktionen in Kombination mit der GROUP BY-Klausel zuzulassen (Top-Aggregate-Anfrage). Beide Anfragemodelle werden in [10], Kapitel 2.1 dargestellt und ihre Definitionen (Definition 2.2 und 2.3) aufgeführt. In dieser Arbeit wird stets die Syntax aus Definition 2.2 verwendet. Die vorgestellten Funktionsklassen und Ergebnisse lassen sich aber auch auf Join- und Aggregate-Anfragen übertragen.

Eine Top-k-Anfrage besteht also (mindestens) aus einer natürlichen Zahl $k \in \mathbb{N}$, der SELECT-Anweisung und einer Funktion f. Bezeichnet man eine solche Anfrage als Q, das ORDER BY-Kriterium stellvertretend mit der Bewertungsfunktion f und die Selektionsbedingung mit S, so lässt sich eine Top-k-Anfrage vereinfacht als Tripel $Q = (S, k, f)$ schreiben (vgl. [11]).

2.2 Grundlagen der Geometrie und Analysis

Die in dieser Arbeit vorgestellten Algorithmen zur Beantwortung von Top-k-Anfragen nutzen spezielle Eigenschaften der zugrundeliegenden Bewertungsfunktion und der Struktur ihres Definitionsbereiches. Neben der Einführung notwendiger mathematischer Begriffe und Konzepte, werden diese Funktionen zunächst geometrisch und später analytisch untersucht.

2.2.1 Geometrie von Bewertungsfunktionen

Wir fokussieren uns auf Funktionen mehrerer (reeller) Veränderlicher, die im Kontext von Top-k-Anfragen auch als Bewertungsfunktion bezeichnet werden. Die Ausführungen dieses Kapitels erfolgen weitestgehend in Anlehnung an [12].

Eine Funktion in n Veränderlichen, auch *multivariate Funktion* genannt, ist eine Abbildung, die jedem Vektor $x = (x_1, \dots, x_n) \in D \subseteq \mathbb{R}^n$ eine reelle Zahl zuordnet, also

$$f: D \to \mathbb{R}, x \mapsto f(x_1, \dots, x_n).$$

Sofern keine weiteren Anforderungen an eine solche Funktion gestellt werden, wollen wir sie im Folgenden immer als *Bewertungsfunktion* bezeichnen.

Verwendet man f in der ORDER BY-Klausel einer Top-k-Anfrage, so werden die n Attributwerte eines Tupels t zu einer reellen Zahl aggregiert. Erst durch die Verwendung einer Sortierungsfunktion ASC oder DESC erhält das Tupel t seinen Rang (engl. score). Für das Verständnis der in Kapitel 4.2 und 4.3 dargestellten Algorithmen ist es wichtig, sich die Geometrie solcher Funktionen zu veranschaulichen. Dazu sind die folgenden beiden Definitionen der multivariaten Analysis besonders hilfreich. Funktionen in zwei Variablen lassen sich durch ihren Graphen veranschaulichen, der entsteht, indem man zu jedem Punkt (x_1, x_2) der x_1x_2-Ebene eines (rechtwinkligen) Koordinatensystems den Funktionswert $x_3 = f(x_1, x_2)$ in x_3-Richtung einzeichnet. Diesen Graphen kann man sich als Fläche im Raum vorstellen, wie z.B. für die Funktion $f: \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto f(x, y) = exp(-x^2 - y^2)$ von Abbildung 2.2.

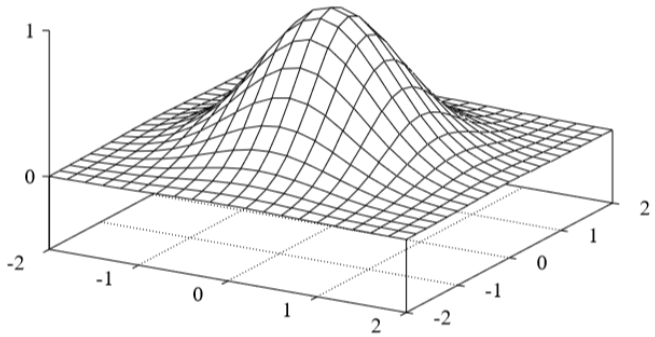


Abbildung 2.2: Graph einer Funktion in zwei Variablen

Quelle[2]

[2] J. Leydold, „Multivariate Analysis," 2006. [Online]. Verfügbar: http://statmath.wu.ac.at/courses/mmwi-finmath/Grundkurs/handouts/handout-6-Funktionen_in_mehreren_Variablen.pdf. [Zugriff am 19 März 2018].

Für Funktionen in $n \in \mathbb{N}$ Variablen ergibt sich allgemein die folgende Definition.

Definition 2.3 Graph einer Funktion

Der *Graph* einer multivariaten Funktion $f: D \to \mathbb{R}, D \subseteq \mathbb{R}^n$ ist definiert als Teilmenge

$$G_f = \{(x, f(x)) \in \mathbb{R}^{n+1} \mid x \in D\}.$$

Für $n = 1$ ist der Graph eine Kurve der zweidimensionalen Ebene. Für $n = 2$, wie oben erwähnt, eine Fläche im dreidimensionalen Raum.

Eine weitere Methode, Funktionen von zwei Variablen anschaulich darzustellen, beruht auf der Verwendung von sogenannten Niveaulinien. Eine Niveaulinie einer Funktion f ist die Menge aller Punkte (x_1, x_2), für die die Funktion einen konstanten Wert annimmt, also $f(x_1, x_2) = \alpha \in \mathbb{R}$ gilt. Abbildung 2.3 verdeutlicht den Sachverhalt.

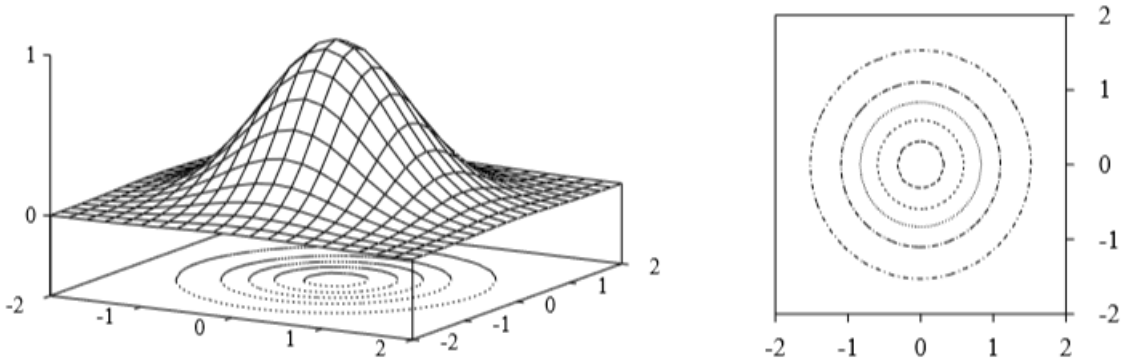


Abbildung 2.3: Niveaulinien einer Funktion in zwei Variablen

Quelle (wie Abbildung 2.2)

Die Verallgemeinerung des zweidimensionalen Falles ergibt die folgende Definition.

Definition 2.4 Niveaumenge und Subniveaumenge

Es sei $D \subseteq \mathbb{R}^n, f: D \to \mathbb{R}$ eine Funktion und $\alpha \in \mathbb{R}$. Dann ist die *Niveaumenge zum Niveau* α die Menge aller Punkte $x \in D$, für die $f(x) = \alpha$ gilt, d.h. die Menge

$$N_f(\alpha) = \{x \in D \mid f(x) = \alpha\} \subset \mathbb{R}^n.$$

Ist $n = 2$, so sprechen wir von einer Niveau- oder Höhenlinie (zum Wert α). Ist $n = 3$, so sprechen wir von Niveaufläche. Die *Subniveaumenge zum Niveau* α schließt zusätzlich zur Niveaumenge auch noch die „darunter liegende“ Fläche mit ein, also

$$\bar{N}_f(\alpha) = \{x \in D \mid f(x) \leq \alpha\} \subset \mathbb{R}^n.$$

Subniveaumengen lassen sich nutzen, um den Begriff einer quasi-konvexen Funktion zu definieren. Das folgende Beispiel veranschaulicht beide Begriffe für eine quadratische Funktion mit ausschließlich positiven Gewichten und zeigt exemplarisch, was wir später allgemein für alle quasi-

konvexen Funktionen beweisen werden, dass eine quasi-konvexe Funktion ihr Maximum stets auf den Ecken eines Hyperrechteckes annimmt.

Beispiel 2.3

Sei I^2 das Einheitsquadrat. Wir betrachten die Funktion

$$f: I^2 \to \mathbb{R};\ (x_1, x_2) \mapsto (x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2.$$

Die Niveaulinie für $\alpha < 0$ ist leer und besteht für $\alpha = 0$ nur aus einem Punkt. In allen anderen Fällen ist

$$N_f(\alpha) = \{(x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2 = \alpha \mid x_1, x_2 \in I\},$$

was gerade der Kreisfläche um $M = (0{,}5; 0{,}5)$ mit Radius α entspricht. Abbildung 2.4 zeigt den Graphen sowie die Niveaulinien der Funktion für $\alpha \in \{0{,}02; 0{,}05; 0{,}1; 0{,}15\}$.

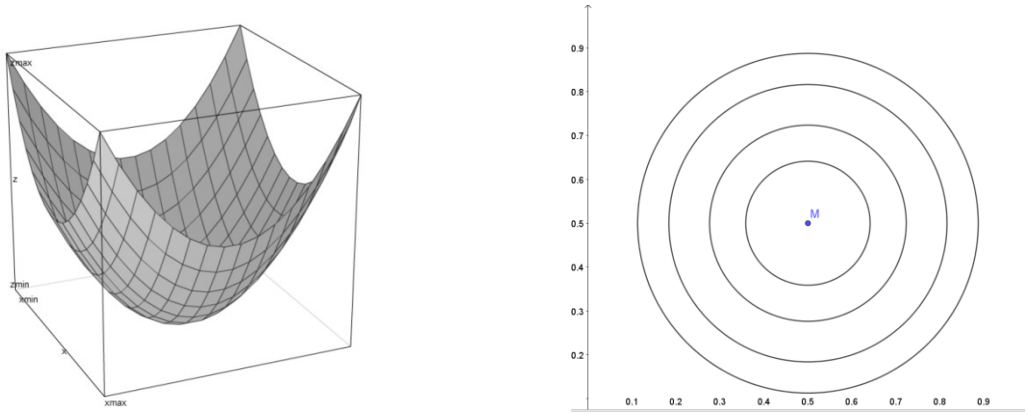


Abbildung 2.4: Graph und Niveaulinien

Man erhält die Niveaulinien für verschiedene Höhen α durch Schnitte des Graphen mit einer Hyperebene parallel zur Grundebene des Hyperwürfels.

Abschließend sei noch der aus der Analysis bekannte Begriff der Monotonie angeführt, der nicht zuletzt zur Unterscheidung monotoner und generischer, also nicht-monotoner Verfahren, benötigt werden wird (siehe [10]).

Definition 2.5 Monotonie

Eine Bewertungsfunktion $f: \mathbb{R}^n \to \mathbb{R}$ ist *monoton steigend*, wenn $f(x_1, \dots, x_n) \leq f(x_1', \dots, x_n')$ gilt, für $x_i \leq x_i'$ und jedes $i \in \{1, \dots, n\}$. Sie heißt *monoton fallend*, falls $f(x_1, \dots, x_n) \geq f(x_1', \dots, x_n')$, für $x_i \leq x_i'$ und jedes $i \in \{1, \dots, n\}$ gilt. Gelten jeweils echte Größer- oder Kleiner-Beziehungen, so ist die Funktion *streng* monoton steigend bzw. fallend. Ist eine Funktion entweder monoton fallend oder monoton steigend, so heißt sie *monoton*, anderseits *nicht-monoton*.

2.2.2 Stetigkeit in metrischen Räumen

In diesem Kapitel werden Grundbegriffe und Konzepte aus der Topologie metrischer Räume vorgestellt. Die Begriffe werden der Einfachheit halber allgemein für jeden metrischen Raum definiert. Im weiteren Verlauf des Kapitels und insbesondere der Folgekapitel, wird der Spezialfall des euklidischen Raumes als metrischer Raum im Vordergrund stehen.

Es wird der aus der Analysis bekannte Begriff der Stetigkeit eingeführt und seine für diese Arbeit wesentlichen Eigenschaften diskutiert. Begriffe, wie etwa der Durchmesser von Mengen und ihr Abstand zu Punkten, die im Folgenden immer wieder verwendet werden, sind ebenfalls Bestandteil dieses Kapitels. Die Ausführungen erfolgen im Wesentlichen in Anlehnung an das Standardwerk der „Grundzüge der modernen Analysis“ von J. Dieudonné [13], sowie an das Buch „From Real to Complex Analysis“ von R.H. Dyer und D.E. Edmunds [14].

Wir beginnen mit dem zentralen Begriff der Abstandsfunktion und des metrischen Raumes.

Definition 2.6 Abstandsfunktion, Metrischer (Teil-)Raum

Sei X eine Menge. Eine *Abstandsfunktion* (kurz: ein *Abstand*) auf X ist eine Funktion $d: X \times X \to \mathbb{R}_0^+$ in die Menge der positiven reellen Zahlen, mit den folgenden Eigenschaften:

(M1) $d(x,y) = 0$ genau dann, wenn $x = y$
(M2) $d(x,y) = d(y,x)$ (Symmetrie)
(M3) $d(x,y) \leq d(x,z) + d(z,y)$ (Dreiecksungleichung)

Das Paar (X,d) heißt *metrischer Raum*. Die Elemente von X werden *Punkte* genannt.
Jede Teilmenge $U \subseteq X$ wird zu einem (metrischen) *Teilraum* $(U, d|_{U\times U})$ durch Einschränkung der Abstandsfunktion von $X \times X$ auf $U \times U$.

Der für den Kontext von Top-k-Anfragen grundlegende Spezialfall eines metrischen Raumes ist der aus der linearen Algebra und analytischen Geometrie bekannte euklidische Raum.

Beispiel 2.4 Euklidischer Raum

Die Funktion $d: \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+; (x,y) \mapsto |x-y|$ ist eine Abstandsfunktion auf der Menge $\mathbb{R}$ der reellen Zahlen, der entsprechende metrische Raum $(\mathbb{R}, |\cdot|)$, ist die Zahlengerade. Die Abbildung $d_E: \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}_0^+; (x,y) \mapsto \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2 + (x_3-y_3)^2}$ definiert den euklidischen Abstand zwischen zwei Punkten $x = (x_1, x_2, x_3)$ und $y = (y_1, y_2, y_3)$. Dieser Abstand lässt sich für jede Dimension, also für jeden n-dimensionalen Raum festlegen. Das Paar $(\mathbb{R}^n, d_E)$ ist der aus der linearen Algebra bekannte *n-dimensionale euklidische Raum* (vgl. [13], 3.2.1 und 3.2.2).

Wenn die Angabe der Abstandsfunktion vernachlässigt werden kann, bezeichnen wir ihn im Folgenden einfach als $\mathbb{R}^n$ und ohne Dimension als euklidischen Raum. Es gibt bekanntlich viele Abstandsfunktionen auf metrischen Räumen und speziell auch auf $\mathbb{R}^n$. Eine weitere, die in Skyline-Anfragen Anwendung findet, ist die sogenannte *Manhattan-Distanz*, die sich aus der Summe der

absoluten Differenzen ihrer Einzelkoordinaten ergibt: $d_M(x,y) = \sum_{i=1}^{n}|x_i - y_i|$ (siehe [15]). Wenn nicht ausdrücklich anders angegeben, verwenden wir den euklidischen Abstand im $\mathbb{R}^n$.

Abbildung 2.5: Euklidischer Abstand (links) und Manhattan-Distanz (rechts)

Obwohl die Abstandsfunktionen geometrisch betrachtet paarweise verschieden sind, so sind sie aus topologischen Gesichtspunkten häufig äquivalent, was bedeutet, dass sich die nun folgenden Begriffe unabhängig von der Wahl der Abstandsfunktion definieren lassen. Ist eine Funktion stetig unter Verwendung der Abstandsfunktion d, so ist sie auch stetig unter d', wenn d, d' äquivalent sind (vgl. [16]).

Es folgen nun einige Definitionen, die für die Einführung kompakter Mengen notwendig sind.

Definition 2.7 Durchmesser und Kugel

Unter dem *Durchmesser* einer nichtleeren Teilmenge A eines metrischen Raumes (X, d) versteht man die Zahl $\delta(A) = sup_{x,y\in A}d(x,y)$. Dieser kann im Allgemeinen positiv reell als auch unendlich sein.

Die *offene Kugel mit Zentrum* $a \in X$ *und Radius* $r > 0$ ist die Menge $K(a,r) = \{x \in X \mid d(x,a) < r\}$. Analog ist die *abgeschlossene* Kugel die Menge $\overline{K}(a,r) = \{x \in X \mid d(x,a) \leq r\}$.

Zwei wichtige Klassen von Mengen aus der Topologie sind die offenen und abgeschlossenen Mengen, mit deren Hilfe sich die für uns wichtige dritte Klasse der kompakten Mengen einführen lässt.

Definition 2.8 Offene, abgeschlossene Menge und beschränkte Mengen

Sei $A \subseteq X$ eine Teilmenge des metrischen Raumes (X, d). A heißt *offen*, wenn zu jedem beliebigen Punkt der Menge eine offene Kugel um diesen Punkt existiert, die in der Menge enthalten ist, also:

$$\forall\, x \in A \text{ existiert ein } r > 0 \text{ mit } K(x,r) \subset A.$$

Das Komplement $\bar{A} := X \backslash A$ einer offenen Teilmenge A heißt *abgeschlossen* (vgl. [13], 3.5).

A heißt durch $o \in X$ *nach oben beschränkt*, wenn die Abstände je zweier Elemente von A kleiner oder gleich der oberen Schranke sind. Wenn also $d(x,y) \leq o\ \forall\, x,y \in A$. Analog ist A durch $u \in X$ *nach unten beschränkt*, wenn $d(x,y) \geq u\ \forall\, x,y \in A$. Ist eine Menge nach oben und unten beschränkt, so ist sie *beschränkt*.

Man kann zeigen, dass eine Menge A beschränkt ist, wenn sie einen endlichen Durchmesser besitzt (siehe [13], 3.4). Nach diesen Vorbereitungen lässt sich der für uns wesentliche Begriff der Kompaktheit einer Menge in dem für uns grundlegenden euklidischen Raum beschreiben.

Definition 2.9 Kompakte Mengen im euklidischen Raum

Eine Teilmenge $C \subseteq \mathbb{R}^n$ des euklidischen Raumes $(\mathbb{R}^n, d)$ heißt *kompakt* genau dann, wenn sie abgeschlossen und beschränkt ist.

Die für diese Arbeit wichtigen Beispiele für Mengen sind die Intervalle (siehe [13], Kap. 3.8).

Beispiel 2.5

i) Jedes offene Intervall $I = (a, b) \subset \mathbb{R}$ ist eine offene, beschränkte Teilmenge des metrischen Raumes $(\mathbb{R}, |\cdot|)$. Entsprechend ist jedes abgeschlossene Intervall $I = [a, b] \subset \mathbb{R}$ eine abgeschlossene, beschränkte Teilmenge von $\mathbb{R}$ für jedes $a, b \in \mathbb{R}$.
ii) Die Vereinigung endlich vieler abgeschlossener Mengen ist abgeschlossen.
iii) Das endliche kartesische Produkt von kompakten Mengen ist kompakt.

Die folgende vierte und letzte Klasse von Mengen bildet den Definitionsbereich der (quasi-)konvexen Funktionen, deren Eigenschaften wir in Kapitel 4.2 untersuchen wollen.

Definition 2.10 Konvexe Menge

Es seien $x_1 \neq x_2$ zwei Punkte des $\mathbb{R}^n$. Dann nennt man die Menge

$$L_{x_1 x_2} = \{z \in \mathbb{R}^n \mid z = \lambda x_1 + (1 - \lambda) x_2, \lambda \in [0,1]\}$$

die *Verbindungslinie* von x_1 zu x_2. Eine Teilmenge $C \subseteq \mathbb{R}^n$ heißt *konvex*, wenn für je zwei beliebige Punkte $x_1, x_2 \in C$ auch deren Verbindungslinie in C liegt.

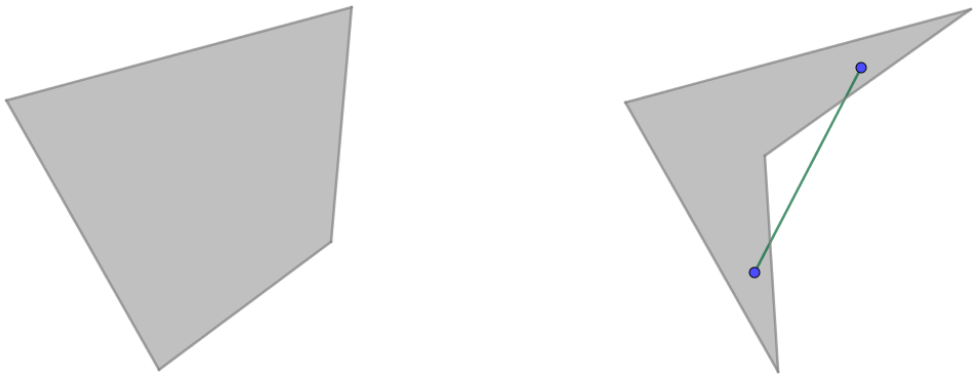


Abbildung 2.6: Konvexe und Nicht-konvexe Menge

Das für uns zentrale Beispiel einer kompakten Menge ist das Hyperrechteck. Ein MBR, als das ein Objekt minimal begrenzendes Hyperrechteck, bildet zusammen mit einer Schranke die zentrale Komponente eines R-Baumes. Für den Beweis des Maximumprinzips quasi-konvexer Funktionen werden wir ein Verfahren zur Konstruktion von Hyperrechtecken verwenden. Dieses Verfahren wird auch zur Definition dienen und soll nun genauer dargestellt werden.

In der Geometrie sind Hyperrechtecke (engl. hyper rectangles) $H \subset \mathbb{R}^n$, die auch als *Hyperquader* bezeichnet werden, n-dimensionale Analogien zum Rechteck für $n = 2$, oder zum Quader für $n = 3$. Mengentheoretisch lassen sie sich als kartesisches Produkt von Intervallen definieren.

Definition 2.11 Hyperrechteck

Eine *achsenparalleles Hyperrechteck* ist eine Teilmenge $H \subseteq \mathbb{R}^n$, die sich als endliches kartesisches Produkt

$$H = I_1 \times I_2 \times \ldots \times I_n$$

von geschlossenen Intervallen $I_i = [a_i, b_i]$ mit $a_i \leq b_i$ für $i = 1, \ldots, n$ darstellen lässt.

Ein *allgemeines Hyperrechteck* ist eine Geometrie, die kongruent mit einem achsenparallelen Hyperrechteck ist.

Man erhält das *Innere* von H, wenn man die offenen Intervalle $I_i = (a_i, b_i)$ statt der abgeschlossenen $[a_i, b_i]$ verwendet. Diejenigen Punkte von H, die nicht zum Inneren gehören, bilden seinen *Rand*. Der *Durchmesser* ist gegeben durch den Abstand $d_E(p, q)$ der beiden gegenüberliegenden Punkte $p, q \in H$, mit den jeweils kleinsten und größten Koordinaten (Definition 3.1). Jedes Hyperrechteck ist, als endliches kartesisches Produkt kompakter und konvexer Mengen (Intervalle) ebenfalls wieder eine kompakte und konvexe Teilmenge des $\mathbb{R}^n$.

Im Folgenden wird, wenn die Angabe der Dimension nicht explizit erforderlich ist oder eindeutig aus dem Kontext hervorgeht, statt n-dimensionalem Hyperrechteck, kurz der Begriff Hyperrechteck verwendet. Die im Kontext von R-Bäumen verwendeten MBRs haben zusätzlich noch die Eigenschaft, dass sie Datenobjekte minimal begrenzen. Strukturell sind MBRs und Hyperechtecke identisch.

Beispiel 2.6

Einfache Beispiele für Hyperechtecke sind der Einheitswürfels $I^n := [0,1]^n$ und die Teilmenge $A_R \subset R \subseteq D_1 \times D_2 \times \ldots \times D_n$ einer jeden n-stelligen Relation R, die man erhält durch

$$A_R = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n],$$

wobei a_i die minimale und b_i die maximale Ausprägungen der Domäne D_i für $i = 1, \ldots, n$ ist.

Für den Beweis des Maximumprinzips quasi-konvexer Funktionen (Kapitel 4.2.2) werden wir das aus der Geometrie bekannte Konstruktionsverfahren eines n-dimensionalen Hyperrechteckes heranziehen und die Eigenschaft verwenden, dass ein solches Hyperrechteck aus zwei $(n-1)$-dimensionalen Hyperrechtecken konstruiert wird. Das Konstruktionsverfahren eines Hyperechteckes ist also iterativ. Beim Übergang in die nächsthöhere Dimension wird das n-dimensionale Hyperrechteck dupliziert und die entsprechenden Punkte der beiden $(n-1)$-dimensionalen Hyperrechtecke durch Strecken miteinander verbunden. Die folgende Abbildung veranschaulicht das Verfahren bis Dimension vier.

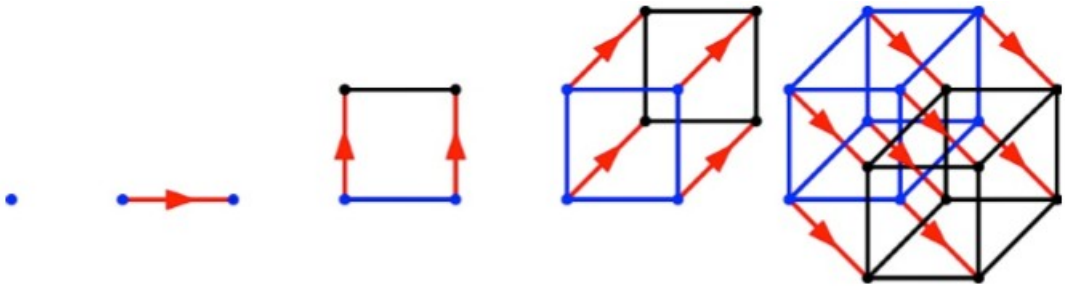


Abbildung 2.7: Hyperrechtecke

Quelle: [17]

Im Falle des vierdimensionalen Hyperrechteckes (rechte Figur von Abbildung 2.7) werden zwei Quader (dreidimensionale Hyperrechtecke) parallel in der Ebene verschoben und entsprechende Punkte gemäß den roten Pfeilen miteinander verbunden. So entsteht ein Schrägbild des vierdimensionalen Hyperrechteckes. Abbildung 2.8 zeigt das in den Ursprung verschobene vierdimensionale Einheitsrechteck mit Angabe der Koordinaten seiner Ecken. Man erhält die Eckenkoordinaten, indem man alle Viererkombinationen aus 0 und 1 erzeugt, bzw. die in Abbildung 2.8 eingezeichnete Menge $\{x_1, x_2, x_3, x_4\}$ der roten Vektoren als Erzeugendensystem (Basis) wählt.

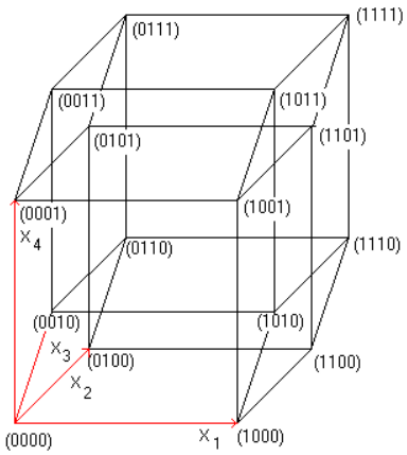


Abbildung 2.8: 4-dimensionales Einheitsrechteck

Quelle:[3]

Dieses Verfahren lässt sich auf höhere Dimensionen verallgemeinern. Jedes n-dimensionale Hyperrechteck mit $n \geq 2$ hat 2^n Ecken, $n2^{n-1}$ Kanten und $2n$ Seitenflächen der Dimension $n-1$. Wir wollen es mit der Geometrie der Hyperrechtecke damit bewenden lassen und verweisen für weiterführende Darstellungen auf das Buch von Coxeter [17].

Es wird jetzt noch gezeigt, dass ein Hyperrechteck durch zwei seiner gegenüberliegenden Ecken eindeutig bestimmt ist. Diese Aussage ist für die Definition von Hyperrechtecken, insbesondere aber auch zur (algorithmischen) Konstruktion der MBRs besonders wichtig.

[3] „Der Hypercubus“ [Online]. Verfügbar: http://www.mathematische-basteleien.de/hyperkubus.htm [Zugriff am 01 September 2018].

Lemma 1 Minimale und Maximale Ecke eines achsenparallelen Hyperrechtecks

Ein achsenparalleles Hyperrechteck ist durch zwei seiner gegenüberliegenden Ecken eindeutig bestimmt.

Beweis: Gegeben sei ein n-dimensionales Hyperrechteck H mit der Menge seiner Ecken $M_H = \{P_1, \dots, P_{2^n}\}$. Die ausgezeichnete „minimale Ecke“ ist die Ecke $P_{min} = \left(x_1^{min}, \dots, x_n^{min}\right) \in M_H$, deren Koordinaten kleiner als die der anderen Punkte sind, für die also $x_1^{min} \leq x_1^i, \dots, x_n^{min} \leq x_n^i$ für jedes $i \in \{1, \dots, n\}$ gilt. Analog zeichnet sich die „maximale Ecke“ P_{max} dadurch aus, dass ihre Komponenten größer als die der anderen Punkte sind, für die also $x_1^{max} \geq x_1^i, \dots, x_n^{max} \geq x_n^i$ für jedes $i \in \{1, \dots, n\}$ gilt.

Es bleibt zu zeigen, wie sich aus P_{min} und P_{max} das Hyperechteck erzeugen lässt. Dazu definieren wir die n Intervalle $I_1 = \left[x_1^{min}, x_1^{max}\right], \dots, I_n = \left[x_n^{min}, x_n^{max}\right]$. Dann gilt $H = I_1 \times I_2 \times \dots \times I_n$. ■

Wir sagen, dass das Hyperechteck durch P_{min} und P_{max} *aufgespannt* wird.

Wir wollen die Ausführungen kurz an einem zweidimensionalen Rechteck verdeutlichen. Gegeben sei ein Rechteck mit den Punkten P_1, P_2, P_3 und P_4 wie in der folgendem Abbildung dargestellt.

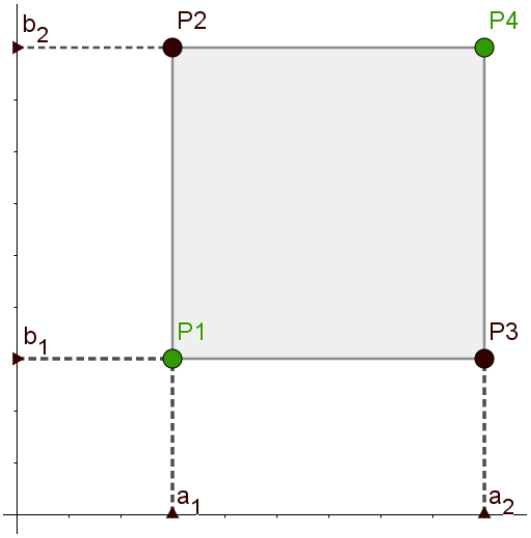


Abbildung 2.9: Minimale und maximale Ecke eines 2-D Hyperrechtecks

Die minimale Ecke ist dann $P_{min} = (x_1, x_2)$ mit $x_1 = \min_{i=1,2} a_i$ und $x_2 = \min_{i=1,2} b_i$, was gerade den Punkt P_1 ergibt. P_1 ist demnach der Punkt des Rechtecks mit den jeweils kleinsten Koordinaten in jeder Dimension. Analog erhält man durch Maximierung der Koordinaten den Punkt $P_{max} = P_4$, der die größten Werte je Dimensionskoordinate besitzt. Offenbar ergibt sich sowohl durch Maximierung, wie auch durch Minimierung immer ein sich gegenüberliegendes Punktepaar des Rechtecks, wodurch dieses damit eindeutig bestimmt ist.

2.2.3 Eigenschaften stetiger Funktionen

Der Stetigkeitsbegriff, wie man ihn aus der Analysis kennt, lässt sich für metrische Räume mit Hilfe einer Abstandsfunktion und dem ε-δ-Kriterium definieren.

Definition 2.12 Stetigkeit

Seien (X, d) und (X', d') zwei metrische Räume. Eine Abbildung $f: X \to X'$ ist *stetig im Punkte* $x_0 \in X$, wenn zu jedem $\varepsilon > 0$ ein $\delta_\varepsilon > 0$ gefunden werden kann, so dass f die Kugel $K(x_0, \delta_\varepsilon)$ um x_0 mit Radius δ_ε in die Kugel $K(f(x_0), \varepsilon)$ abbildet. Wenn also gilt:

$$\forall\, \varepsilon > 0\ \exists\, \delta_\varepsilon > 0\ \forall\, x \in X\ \big(d(x, x_0) < \delta_\varepsilon \Rightarrow d'\big(f(x), f(x_0)\big) < \varepsilon\big).$$

$f: X \to X'$ heißt *stetig auf X*, wenn f stetig in jedem Punkt $x_0 \in X$ ist.

Wie man der Definition entnimmt, ist die Angabe des δ_ε im Allgemeinen abhängig von der Wahl des Punktes x_0, weswegen man daher auch von *punktweiser* Stetigkeit spricht. Wenn δ_ε nur von ε und nicht vom Punkt x_0 abhängt, so bezeichnet man dies als gleichmäßige Stetigkeit, was eine stärkere Form von Stetigkeit darstellt. Wir werden in Kapitel 4.1 zeigen, dass erst dann, wenn eine Bewertungsfunktion nicht nur stetig, sondern auch gleichmäßig stetig ist und man keine weiteren Eigenschaften voraussetzt, bereits eine obere und untere Schranke auf einem Hyperrechteck nur in Abhängigkeit von ε angeben werden kann. Die genaue Definition ist die Folgende:

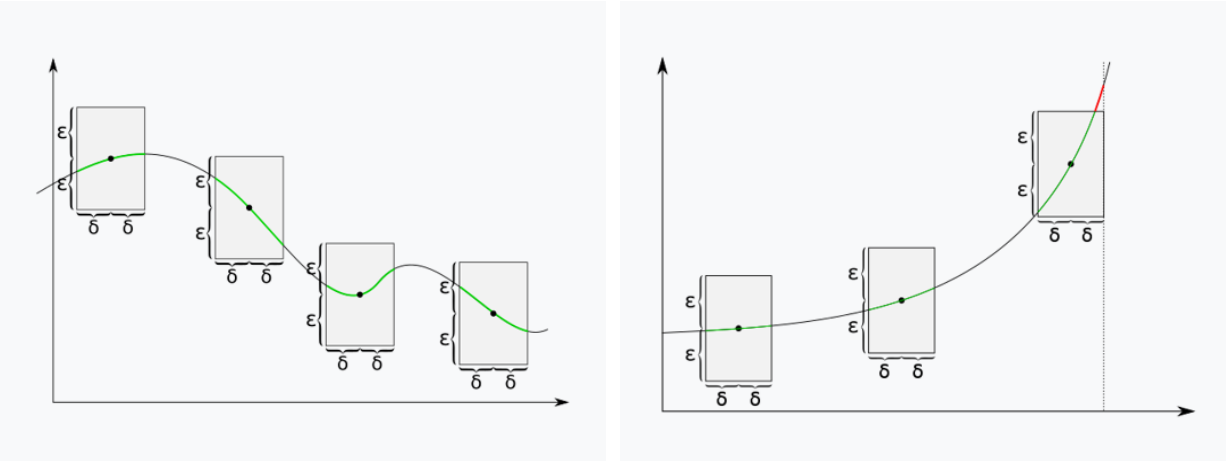
Definition 2.13 Gleichmäßige Stetigkeit

Seien (X, d) und (X', d') zwei metrische Räume. Eine Abbildung $f: X \to X'$ heißt *gleichmäßig stetig* wenn

$$\forall\, \varepsilon > 0\ \exists\, \delta_\varepsilon > 0\ \forall\, x, y \in X\ \big(d(x, y) < \delta_\varepsilon \Rightarrow d'\big(f(x), f(y)\big) < \varepsilon\big).$$

In der folgenden eindimensionalen Darstellung soll der Unterschied zwischen (einfacher) Stetigkeit und der gleichmäßigen Stetigkeit noch einmal verdeutlicht werden.

Im eindimensionalen euklidischen Raum $(\mathbb{R}, |\cdot|)$ mit der vom Absolutbetrag induzierten Abstandsfunktion ist die Kugel $K(x, r)$ um einen Punkt x ein Quadrat mit Seitenlänge $2r$. Bei einer gleichmäßig stetigen Funktion kann für jedes $\varepsilon > 0$ ein $\delta_\varepsilon > 0$ gefunden werden, so dass sich die entsprechenden Funktionswerte $f(x)$ und $f(y)$ für $|x - y| < \delta_\varepsilon$ um maximal ε unterscheiden. Also lässt sich um jeden Punkt $\big(x, f(x)\big)$ des Graphen von f ein Rechteck der Breite δ_ε und Höhe ε legen, so dass der Graph im Inneren dieses Rechteckes liegt. Wenn man das Rechteck entlang des Graphen einer nur stetigen, aber nicht gleichmäßig stetigen Funktion verschiebt, so kann es Punkte geben, die nicht innerhalb des Rechteckes liegen. Abbildung 2.10 veranschaulicht die Situation.



Links: Gleichmäßig stetige Funktion, Rechts: Stetige aber nicht gleichmäßig stetige Funktion
Abbildung 2.10: Veranschaulichung der gleichmäßigen Stetigkeit

Quelle: Wikipedia

Als zwei einfache weitere Beispiele betrachte man im zweidimensionalen euklidischen Raum die Addition und Multiplikation der Koordinaten der Punkte, also die beiden Abbildungen $add: \mathbb{R}^2 \to \mathbb{R};\ (x_1, x_2) \mapsto x_1 + x_2$, sowie $mult: \mathbb{R}^2 \to \mathbb{R};\ (x_1, x_2) \mapsto x_1 \cdot x_2$. add ist gleichmäßig stetig, $mult$ ist stetig, aber nicht gleichmäßig stetig.

Im Folgenden werden zwei fundamentale Eigenschaften stetiger Funktionen auf kompakten Mengen aufgeführt. Beide zusammen sind grundlegend für die Anwendung des BRS-Algorithmus der nachfolgenden Kapitel.

Satz 2.1 (Satz von Heine) Jede stetige Funktion auf einer kompakten Menge ist gleichmäßig stetig.

Beweis: (siehe [14], Theorem 2.3.30). ■

Damit für die Verwendung von stetigen Funktionen überhaupt die Top-k-Punkte einer Anfrage bestimmt werden können, muss zuvor sichergestellt sein, dass diese Punkte überhaupt existieren. Dies garantiert der folgende Satz.

Satz 2.2 Satz vom Minimum und Maximum (Extremwerttheorem)

Es sei $C \subseteq \mathbb{R}^n$ eine kompakte Menge und $f: C \to \mathbb{R}$ eine reellwertige stetige Funktion. Dann ist f auf C nach unten und oben beschränkt und es existieren zwei Punkte $p, q \in C$ derart, dass $f(p) = sup_{x \in C} f(x)$ und $f(q) = inf_{x \in C} f(x)$.

Mit anderen Worten: Jede stetige Funktion f nimmt auf einer kompakten Menge ein Maximum und Minimum an.

Beweis: (siehe [18], Korollar 8.2.5). ■

Zum Schluß von Kapitel 2 wollen wir noch zeigen, wie mit sogenannten affin-linearen Abbildungen geometrische Objekte in andere überführt werden und dabei die für uns wesentliche Eigenschaft der Ranginformationen der Punkte erhalten bleibt. Das Objekt unseres Interesses wird dabei die aus Beispiel 2.6 gegebene Menge $A_R \subset R$ sein. Diese soll aus Gründen einer einheitlichen Methodik, aber auch um die Datenpunkte einfacher handhaben zu können, auf den Einheitswürfel abgebildet werden.

2.2.4 Projektion auf den Einheitswürfel

Wir wollen eine gegebene Punktmenge, genauer, das diese Punktmenge umgebende Hyperrechteck so auf den Einheitswürfel gleicher Dimension abbilden, dass dabei der Informationsgehalt über den Rang (Ranginvarianz) der einzelnen Datenpunkte erhalten bleibt. Wir stellen den aus der linearen Algebra bekannten Begriff einer affin-linearen Abbildung für den euklidischen Raum $\mathbb{R}^n$ vor, da er genau diese Anforderung erfüllt. Das Rechnen mit Matrizen (Addition und Multiplikation), wie man es aus der linearen Algebra (siehe [19]) kennt, setzen wir als bekannt voraus.

Definition 2.14 Affin-lineare Abbildungen

Sei A eine quadratische $(n \times n)$-Matrix und $b \in \mathbb{R}^n$ ein n-dimensionaler Vektor. Dann ist eine *affin-lineare Abbildung* des $\mathbb{R}^n$ (kurz affine Abbildung) eine Abbildung der Form

$$\phi: \mathbb{R}^n \to \mathbb{R}^n; x \mapsto Ax + b.$$

Mit anderen Worten: Eine affine Abbildung setzt sich aus einer *linearen Abbildung* $x \mapsto Ax$ und einer anschließenden *Translation* (Verschiebung um den Vektor b) zusammen.

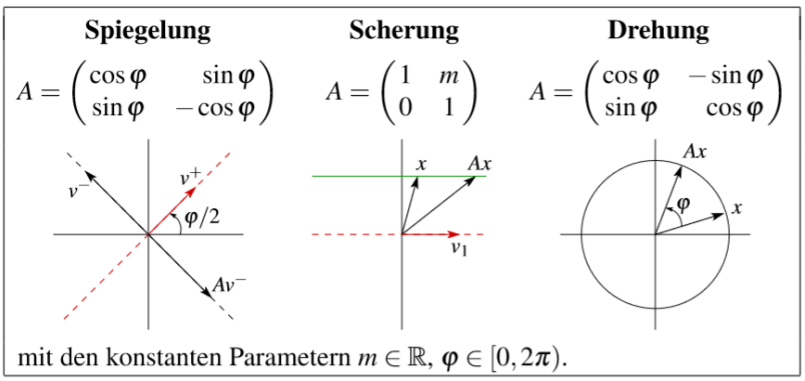


Abbildung 2.11: Wichtige lineare Abbildungen im $\mathbb{R}^2$

Quelle: [4]

Eine affine Abbildung ist bijektiv, wenn die Determinante der Koeffizientenmatrix $det(A) \neq 0$ ist. Aus der Geometrie bzw. der linearen Algebra weiß man ([19] Seite 139 f.), dass affine Abbildungen folgende Eigenschaften besitzen:

[4] A. de Vries, „Affine Abbildungen - Definition und Anwendungsbeispiele," 22 März 2017. [Online]. Verfügbar: http://haegar.fh-swf.de/TBW/Affine_Abbildungen/Affine-Abbildungen_Folien.pdf. [Zugriff am 19 März 2018].

(A1) Bilder von Punkten, die auf einer Geraden liegen, werden wieder auf Punkte einer Geraden abgebildet.

(A2) Parallele Geraden werden in parallele Geraden überführt.

(A3) Drei verschiedene Punkte, die auf einer Geraden liegen werden so abgebildet, dass das Teilverhältnis der Bildpunkte mit dem der Urbildpunkte übereinstimmt.

Wenn man also eine affine Abbildung angeben kann, welche jede Kante $[a, b]$ eines Hyperrechtecks auf das Einheitsintervall abbildet, so folgt aus der Eigenschaft (A3) unmittelbar die gewünschte Ranginvarianz. Die gesuchte Abbildung erhält man durch Kombination einer Verschiebung des Hyperwürfels mit der Ecke der kleinsten Koordinaten in den Nullpunkt und der Streckung bzw. Stauchung desselben, was wir als (Einheits-)Würfelprojektion bezeichnen wollen.

Beispiel 2.7 Würfelprojektion

Wir betrachten die kleinste und größte Ecke eines gegebenen Hyperrechteckes gemäß den Ausführungen von Kapitel 2 (Lemma 1). Es seien $(a_1, \dots, a_n)$ die Koordinaten der kleinsten und $(b_1, \dots, b_n)$ die Koordinaten der größten Ecke. Wählt man $r_j = \frac{1}{b_j - a_j}$ für $j = 1, \dots, n$ und die Diagonalmatrix $A = \begin{bmatrix} r_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_n \end{bmatrix}$, dann ist $x \mapsto Ax - Aa$ die gesuchte Abbildung, die wir als *Würfelprojektion* bezeichnen wollen.

Wir können daher von nun an, ohne Beschränkung der Allgemeinheit, bei einer vorgegebenen Relation einer Datenbank bei ihren Tupeln mit reellem Wertebereich immer von einer Punktmenge ausgehen, die eine Teilmenge des Einheitswürfels bildet. Wir werden also im Folgenden immer den Einheitshyperwürfel als Datenraum voraussetzen. Als konkretes durchgängiges Beispiel einer dreistelligen Relation mit den beiden reellwertigen Attributen (Attribut 1 und Attribut 2), wird in den folgenden Kapiteln das aus [3], also das in Abbildung 2.12 angegebene Beispiel verwendet. Die Vergabe einer eindeutigen Bezeichnung (hier: ID) für jeden Datenpunkt wird im BRS-Algorithmus von entscheidender Bedeutung sein.

ID	1	2	3	4	5	6	7	8	9	10	11	12
Attribut 1	0,2	0,1	0,3	0,2	0,3	0,5	0,4	0,6	0,7	0,6	0,7	0,7
Attribut 2	0,2	0,5	0,3	0,9	0,8	0,7	0,3	0,1	0,2	0,5	0,6	0,5

Abbildung 2.12: Beispiel 2-dimensionale Punktmenge

Zu beachten ist, dass durch Anwendung einer solchen Projektion eine gegebene Problemstellung nicht verändert werden soll, man also mit der „neuen“ Punktmenge zum gleichen Ergebnis gelangt. Wir werden bei der Anwendung spezieller Funktionsklassen in Kapitel 4 noch einmal genauer auf diese Problematik eingehen.

3 Algorithmische Grundlagen

Nachdem die mathematischen Grundlagen eingeführt sind, widmet sich das nun folgende Kapitel den algorithmischen Methoden und Konzepten. Zunächst stehen R-Bäume im Vordergrund, die als Zugriffspfade für ausgedehnte mehrdimensionale Objekte (Geometrien) entwickelt wurden. Zur Evaluierung der Ergebnisse in Kapitel 5 werden wir einen speziellen binären R-Baum konstruieren, indem wir den Datenraum mit Hilfe eines regelmäßigen Gitters partitionieren, dessen Partitionen die MBRs des R-Baumes bilden.

R-Bäume werden für verschiedene Branch-and-Bound-Anfragebeantwortungsverfahren verwendet, die eine Bestenliste ermitteln, wie etwa für die Nächste-Nachbarn-Suche, für Skyline- und Top-k-Anfragen. Alle drei Problemstellungen können mit unterschiedlichen Methoden gelöst werden. Wir werden für jedes dieser drei Verfahren einen prominenten Vertreter vorstellen, der zur Lösung des Problems das Branch-and-Bound-Prinzip einsetzt und der uns für die Einordnung unserer Arbeit als wesentlich erscheint.

3.1 Mehrdimensionale Suchstrukturen

Anwender von Datenbanksystemen erwarten, dass ihre Anfragen vom System möglichst schnell bearbeitet und beantwortet werden. Diese Anforderung kann nur erreicht werden, wenn nicht der gesamte Datenbestand durchsucht werden muss, was sich durch die Verwendung geeigneter Indexstrukturen und einer Partitionierung der Punktmenge in Kombination mit einem Index erreichen lässt.

Grundsätzlich lässt sich der Zugriff auf mehrere Attribute auch durch eindimensionale Indizes, wie etwa durch den klassischen B-Baum realisieren. Untersuchungen haben allerdings gezeigt, dass für Suchvorgänge über mehrere Attribute, wie für Top-k-Anfragen, Zugriffspfade über einzelne Attribute den mehrdimensionalen unterlegen sind. Mehrdimensionale Suchstrukturen, wie k-d- und R-Bäume, sind nicht auf nur ein Attribut beschränkt, sondern berücksichtigen „beliebig“ viele Dimensionen gleichzeitig. Anzumerken ist dabei, dass alle Attribute gleichrangig berücksichtigt werden, weil die Indizierung hierbei über die verschiedenen Dimensionen zugleich erfolgt. Aus diesen Gründen und da im Kontext der Branch-and-Bound basierten Verfahren meist R-Bäume zum Einsatz kommen, werden wir uns im Folgenden auf den R-Baum und einer seiner Spezialisierungen, den R+-Baum fokussieren.

Zuerst wollen wir allgemeine Grundbegriffe für Bäume aufführen, die im Folgenden immer wieder Verwendung finden. Unter einem *Baum* versteht man ein Tupel aus zwei Mengen von *Knoten* und *Kanten*. Jeder (nichtleere) Baum besitzt einen ausgezeichneten Knoten, die *Wurzel* (engl. root), und jeder Knoten, außer der Wurzel, ist durch genau eine Kante mit seinem *Elternknoten*, auch Vaterknoten genannt verbunden. Er wird als *Kindknoten* oder auch als *Sohn* bezeichnet. Ein Knoten, der keine Kinder besitzt, ist ein *Blattknoten* und wird kurz als *Blatt* bezeichnet. Alle Nicht-Blattknoten, außer der Wurzel, heißen *innere* Knoten des Baumes. Ein *Pfad* eines Baumes ist eine Folge paarweise unterschiedlicher Knoten, die durch Kanten verbunden sind. Ein wesentliches Merkmal eines Baumes ist, dass es zwischen Wurzel und jedem Knoten genau einen Pfad gibt, so dass man die *Tiefe eines*

Knotens als die Anzahl der Kanten von der Wurzel bis zum besagten Knoten definieren kann. Die *Höhe eines Baumes* ist die maximale Tiefe eines Knotens.

Ein Baum heißt *binär* oder *Binärbaum*, wenn jeder Knoten maximal zwei Kindknoten besitzt. Ist die absolute Höhendifferenz der beiden Teilbäume eines Binärbaumes maximal gleich 1, so heißt er *balanciert*. Er ist *perfekt balanciert*, wenn sich alle Blätter auf höchsten zwei benachbarten Ebenen befinden.

Eine wichtige Klasse von Bäumen sind die sogenannten (binären) *Entscheidungs-* bzw. *Suchbäume*. In einem solchen Baum hat jeder innere Knoten einen Schlüsselwert, für den gilt, dass alle Schlüsselwerte seines linken Teilbaumes kleiner als sein eigener Schlüsselwert, die der rechten dementsprechend größer sind. Beim Durchsuchen des Baumes wird bei jedem besuchten Knoten der entsprechende Schlüsselwert abgefragt und verglichen, also eine Entscheidung über das weitere Vorgehen getroffen, wonach der Name Entscheidungsbaum rührt. Dieser Vorgang wird solange fortgesetzt, bis man die Blattebene erreicht hat.

Eine Möglichkeit, mehrdimensionale Suchstrukturen zu klassifizieren, besteht in der Unterscheidung zwischen der *Organisation nach Datenpunkten* und der *Organisation des* diesen Datenpunkten zugeordneten *Datenraumes* (siehe Definition 2.1). Man spricht auch von daten- und raumbezogenen Indexstrukturen. Bei raumbezogenen Indexstrukturen wird der Datenraum selbst, ohne Berücksichtigung der sich darin enthaltenen Objekte, partitioniert. Bei der zweiten Variante, der datenbezogenen Indexstruktur, erfolgt eine Partitionierung der Datenpunkte im Raum. Der hier vorgestellte R-Baum und seine bekannten Varianten sind raumbezogene Indexstrukturen.

Die in diesem Kapitel aufgeführten Beispiele sind der Anschauung halber alle zweidimensional. Sie lassen sich aber ohne Weiteres auf höhere Dimensionen verallgemeinern. Die Ausführung in den nun folgenden Kapiteln erfolgen weitestgehend in Anlehnung an [1] und [20].

3.1.1 R-Baum

Der Begriff des R-Baumes (engl. r-tree) wurde von Antonin Guttman eingeführt [21] und für die schnelle Suche in räumlich ausgedehnten Objekten konzipiert, die auch als Geometrien bezeichnet werden. Die zu indizierenden räumlichen Objekte sind in den Blattknoten des Baumes enthalten. Unter einer *Geometrie* verstehen wir eine Menge von Datenpunkten im euklidischen Raum mit einem rechtwinkligen kartesischen Koordinatensystem, die aus nur einem einzigen Punkt, endlich vielen, oder unendlich vielen Punkten bestehen kann.

Die Indizierung von Geometrien erfolgt beim R-Baum durch mehrdimensionale achsenparallele (Hyper-)Rechtecke, welche ein Objekt minimal begrenzen. Aufgrund ihrer Begrenzungseigenschaft wird ein solches mehrdimensionales Rechteck auch *minimal bounding rectangles*, kurz MBR genannt.

Um für eine vorgegebene endliche Menge von Geometrien im Raum einen R-Baum aufzubauen, bildet man im ersten Schritt die diese Geometrien umgebenden MBRs und ordnet sie der ersten inneren Knotenebene zu. Als nächstes werden die entstandenen MBRs der Blattebene einem übergeordneten MBR zugeordnet, der sie umschließt und mehrere MBRs zusammenfasst. Diese

Struktur setzt sich so bis zur Wurzel fort. Als Ergebnis entsteht ein balancierter Baum mit einer Wurzel, die aus einer Menge von MBRs besteht, die alle Objekte umfassen.

Abbildung 3.1 veranschaulicht einen R-Baum mit drei MBRs (R1, R2, R3) auf Knotenebene und einer inneren Ebene mit den MBRs R4, R5 bis R12, sowie den R9 zugeordneten Blättern A, B und C.

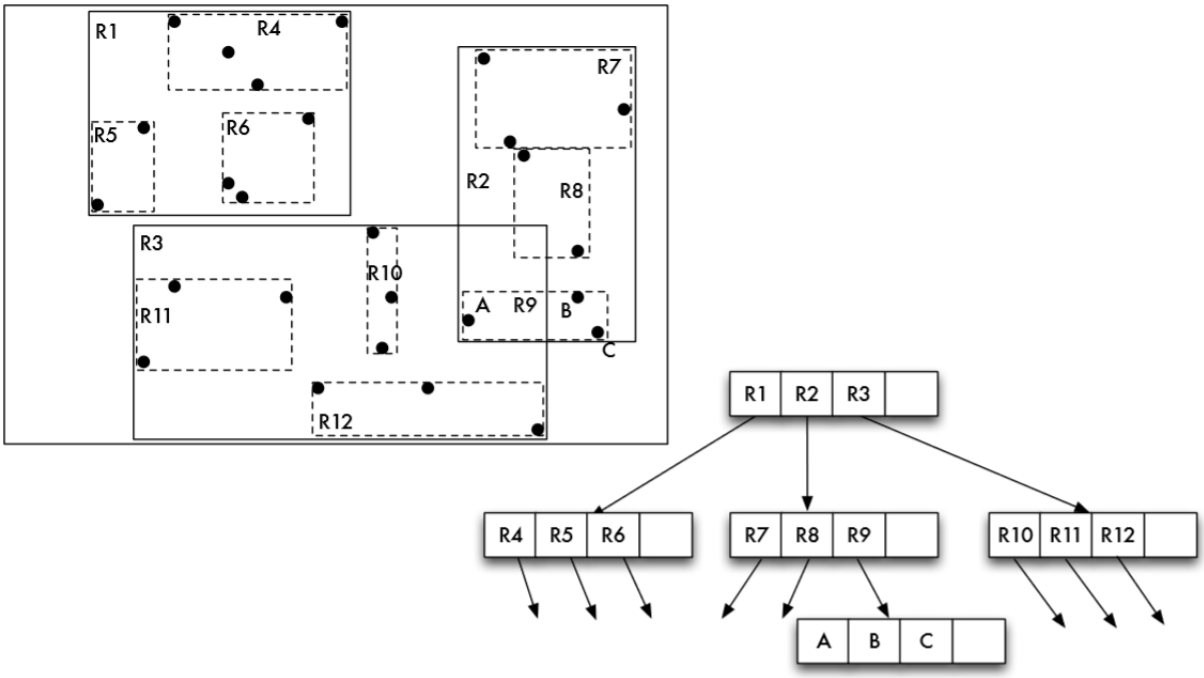


Abbildung 3.1: Struktur eines R-Baumes

Quelle: [5]

Beim R-Baum werden räumlich „zusammengehörige“ Objekte zu neuen, übergeordneten Objekten durch MBRs zusammengefasst. Nach welcher Vorschrift eine solche Zusammenfassung erfolgt, bleibt zunächst einmal offen. Des Weiteren sei darauf hingewiesen, dass es sich bei Abbildung 3.1 schon um die spezielle Form des R-Baumes handelt, der später für Top-k-Anfragen verwendet wird, da seine Blätter aus Datenpunkten und nicht aus Datenrechtecken bestehen.

Im Gegensatz zum bekannten B*-Baum, der auf jeder Baumebene einen eindimensionalen Raum in disjunkte Intervalle zerlegt, zerlegt ein R-Baum einen n-dimensionalen Raum in n-dimensionale nicht notwendigerweise disjunkte Hyperrechtecke, die im Folgenden unabhängig von ihrer Dimension immer nur als MBRs bezeichnet werden (vgl. [22]). Da die Geometrien im Raum im Allgemeinen überlappen können, gilt das gleiche auch für die MBRs eines R-Baumes. Allerdings ist die Zuordnung der Geometrien zu einem MBR auf Blattebene eindeutig.

[5] K.-U. Sattler, G. Saake und V. Köppen, „Data-Warehouse-Technologien, Teil 4, Indexstrukturen für Data Warehouse,“ 13 Dezember 2017. [Online]. Verfügbar: www.dbse.ovgu.de/dbse_media/Dateien/DWT/06_Index.pdf. [Zugriff am 19 März 2018].

Lemma 1 aus vorigem Kapitel hat gezeigt, dass jedes Hyperechteck und damit auch jedes MBR durch zwei Punkte eindeutig bestimmt ist. Die Blattebene der R-Bäume, die zur Beantwortung von Top-k-Anfragen genutzt wird, enthält die Datenpunkte, die zu MBRs der übergeordneten Ebene zusammengefasst werden. Lemma 1 lässt sich nutzen, um zu beschreiben, wie sich ein MBR aus einer endlich gegebenen Menge von Datenpunkten explizit konstruieren und definieren lässt.

Definition 3.1 Minimum Bounding Rectangle

Sei $M = \{P_1, \dots, P_N\}, N \in \mathbb{N}$ eine Menge von Punkten im n-dimensionalen euklidischen Raum mit den Koordinaten $P_j = \left(x_1^j, \dots, x_n^j\right)$ für jedes $j \in \{1, \dots, N\}$. Das von diesen Punkten erzeugte MBR ist das von den beiden Punkten $P = \left(\min_j\left(x_1^j\right), \dots, \min_j\left(x_n^j\right)\right)$ und $Q = \left(\max_j\left(x_1^j\right), \dots, \max_j\left(x_n^j\right)\right)$ aufgespannte achsenparallele Hyperrechteck. Wir nennen P die *untere* oder *minimale Ecke*, Q die *obere* oder auch *maximale Ecke* eines MBRs. Besteht M aus nur einem Punkt, so ist $P = Q$.

Beispiel 3.1

Die Menge $A_R = [a_1, b_1] \times \dots \times [a_n, b_n]$ aus Beispiel 2.6 ist das MBR *aller* Datenpunkte einer gegebenen Relation *R*, also das kleinste Hyperechteck, dass alle Datenpunkte der Relation umfasst.

Nicht zu verwechseln ist ein MBR mit der konvexen Hülle einer endlichen Punktmenge des Raumes, die stets eine Untermenge des MBRs bildet. Zu beachten ist auch, dass die beiden Punkte P und Q nicht notwendigerweise in der Punktmenge M enthalten sein müssen, wie folgende Abbildung verdeutlicht. Die linke Figur zeigt die MBRs für jeweils drei (zusammengehörige) Datenpunkte. Rechts sind für jedes dieser MBRs die unteren und oberen Ecken dargestellt.

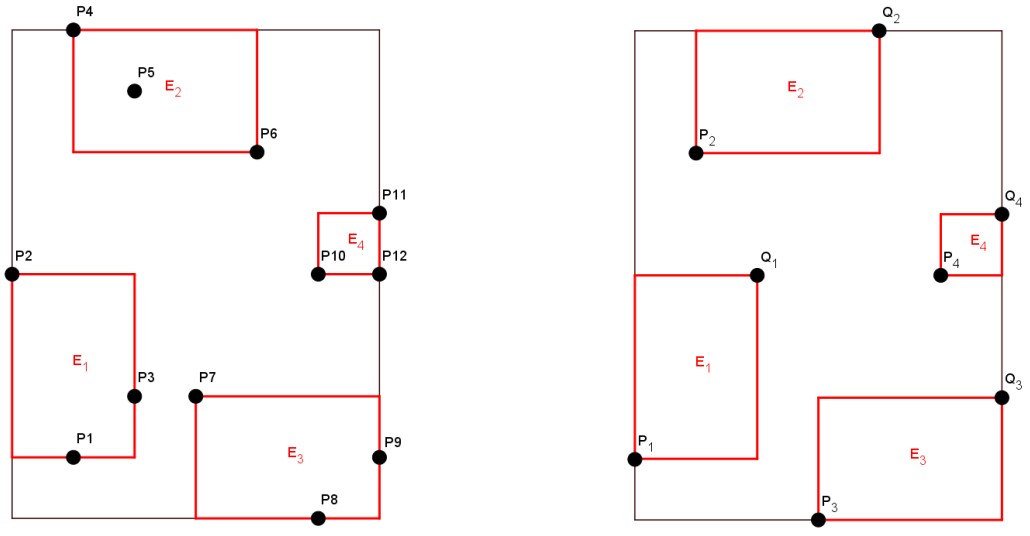


Abbildung 3.2: Zweidimensionale MBRs über Datenpunkten

Der Eintrag eines inneren Knotens eines R-Baumes besteht aus dem Paar (MBR, node-id), wobei node-id ein Verweis ist, der auf einen Kindknoten zeigt. Ein Eintrag eines Blattes ist ein Paar aus einem MBR und einem Bezeichner, oder Verweis auf den Datenpunkt (z.B. eine ID-Nummer): (MBR, o-id). An den Baum selbst, also seine Struktur, werden die folgenden Bedingungen gestellt:

(R1) Jeder Knoten außer der Wurzel hat höchstens $\hat{a}$, aber wenigstens a Einträge, mit $a \leq \hat{a}/2$.

(R2) Wenn die Wurzel kein Blatt ist, so hat sie mindestens 2 Söhne.

(R3) Jeder Eintrag in einem inneren Knoten hat als MBR das kleinste umbeschriebene MBR, das alle MBRs des entsprechenden Sohnes enthält.

(R4) Jeder innere Knoten hat eine Anzahl von Einträgen der Form (mbb, node-id) und jedes Blatt der Form (mbb, o-id).

(R5) Alle Blätter befinden sich auf der gleichen Ebene.

Da der R-Baum als Sekundärspeicherstruktur konzipiert ist, orientieren sich die konkreten Werte für $\hat{a}$ und a an der Größe einer Speicherseite. Ein Knoten soll also möglichst genau die Größe einer solchen Seite haben. Der letzte Punkt (R5) besagt, dass der R-Baum stets perfekt balanciert ist.

Abschließend wollen wir der Vollständigkeit halber noch die „gewöhnliche“ Suche nach Gebieten in R-Bäumen beschreiben. Für Top-k-Anfragen werden wir zur Ermittlung der Ergebnismenge allerdings eine andere Suche durchführen.

Unter einem Gebiet verstehen wir ein n-dimensionales MBR, welches eine Geometrie gleicher Dimension minimal umfasst (siehe Abbildung 3.4). Das Gebiet ist zur Durchführung einer Suche bekannt und muss im Allgemeinen nicht in nur einem Knoten des R-Baumes enthalten sein. Bei der Suche nach nur einem Punkt schrumpft das MBR auf diesen einen Punkt zusammen. Wir folgen den Ausführungen von [20].

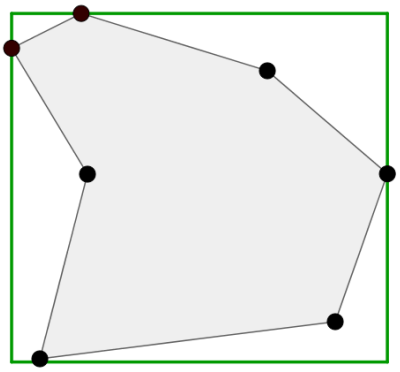


Abbildung 3.3: 2D Geometrie mit zugehörigem Gebiet (MBR)

Gebietsuche

Gesucht sei ein Gebiet G. Beginnend bei der Wurzel werden alle Einträge des aktuellen Knotens auf einen Schnitt mit G untersucht. Bei allen Einträgen, bei denen $G \cap MBR = \emptyset$ wird der Sohnknoten rekursiv durchsucht. Ist man an einem Blattknoten angekommen, so werden alle Einträge betrachtet. Im Falle $G \cap MBR = \emptyset$ werden, über die in den Blättern enthaltenen Zeiger zu den Datenobjekten, die gefundenen Datenobjekte als Ergebnis der Suche zurückgegeben.

Beispiel 3.2 Gebietsuche im R-Baum (**[20]**, Beispiel für Gebietsuche)

In der Abbildung 3.4 ist das Rechteck S das Suchziel, wobei der Baum mit $\hat{a}$ = 4 und a = 2 ist. Der Algorithmus sucht nach den MBRs, die sich mit S überschneiden. In der unteren Figur von Abbildung 3.4 ist der vom Algorithmus gewählte Suchvorgang gezeigt.

Da sich die Geometrie S mit den Wurzeleinträgen R1 und R2 überschneidet, sucht der Algorithmus in beiden MBRs weiter. In R1 existiert ein Rechteck R4, das sich mit S schneidet. Die Einträge des Rechtecks R4 werden im Anschluss überprüft. So erreicht der Algorithmus den Blattknoten, der nach geeigneten Einträgen untersucht wird. R11 ist der einzige passende Kandidat zur Ergebnismenge. In R2 gibt es zwei Rechtecke, die sich mit S schneiden: R5 und R6. Die weitere Untersuchung wird vom Algorithmus in R5 und R6 geführt. Dies liefert die Einträge R13 bzw. R15 und R16. Somit bilden R11, R13, R15 und R16 die Ergebnismenge der Suche.

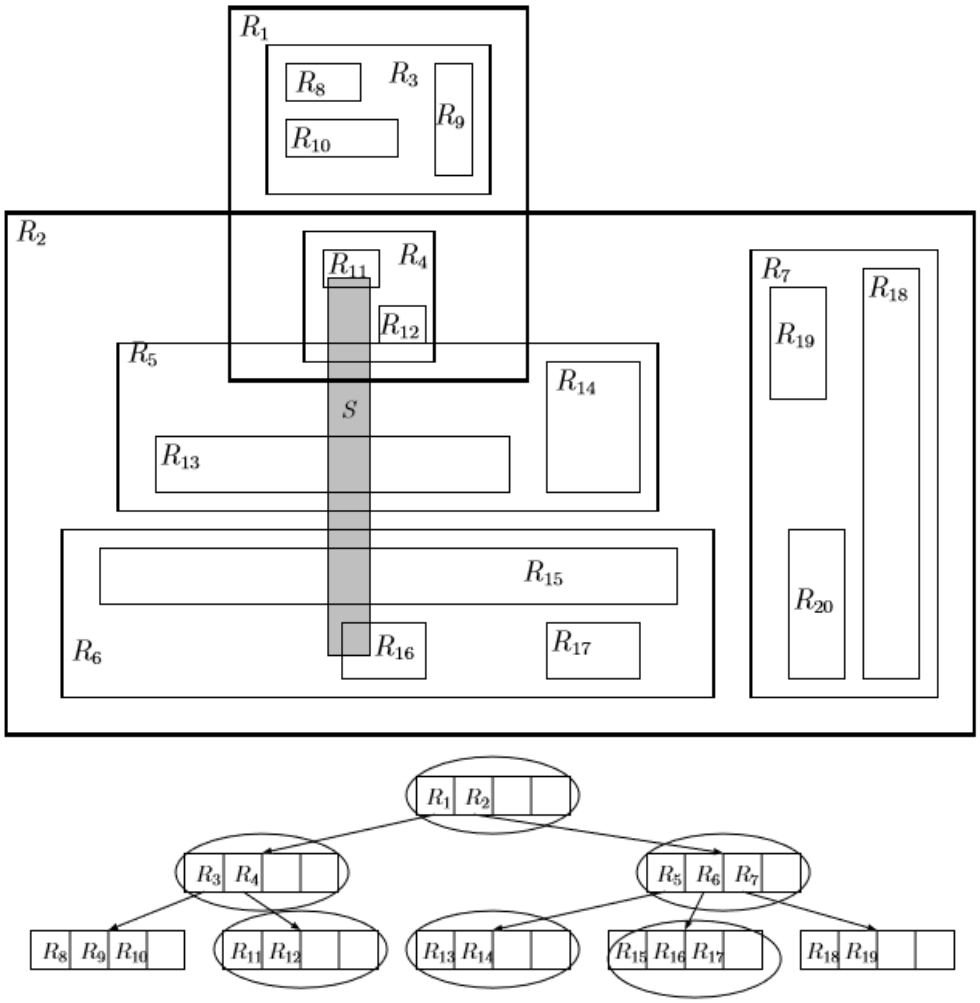


Abbildung 3.4: Gebietsuche im R-Baum

Quelle: [20]

Da eine Top-k-Anfrage nach Datenpunkten sucht, haben wir es in diesem Fall mit R-Bäumen zu tun, deren Blattebene aus Datenpunkten bestehen, die den MBRs der letzten Nicht-Blattebene zugeordnet sind.

3.1.2 R+-Baum

Der in [23] vorgestellte R+-Baum ist eine Spezialisierung des R-Baumes, die eine schnellere Suchlaufzeit anstrebt. Ein R+-Baum unterscheidet sich im Wesentlichen von einem R-Baum in den folgenden Eigenschaften:

a. Im R-Baum können sich die Suchbereiche überlappen. Das ist im R+-Baum für die Nicht-Blattknoten nicht erlaubt, was zur Folge hat, dass der Suchraum disjunkt aufgeteilt wird und jede Suche über einen eindeutigen Suchpfad erfolgt.
b. Um Disjunktheit auf den Suchbereichen der inneren Blattebenen zu erreichen, kann es nötig sein, bestimmte Datenobjekte redundant auf Blattebene zuzuweisen.
c. Zuletzt wird beim R+-Baum auf die Forderung nach einer Mindestanzahl a an Einträgen pro Knoten verzichtet.

Punkt b. bedeutet für die Punktsuche, dass man ggf. mehrere Blätter in die Suche einbeziehen muss.

Die Struktur des R+-Baumes ist wie folgt festgelegt (vgl. [24]):

(R1+) Die Wurzel hat mindestens zwei Einträge, außer wenn sie ein Blattknoten ist.

(R2+) Die MBRs der Knoten, die sich auf einer gleichen Baumebene befinden, dürfen einander nicht überlappen.

(R3+) Wenn ein Knoten kein Blatt ist, umfasst sein MBR alle MBRs, die sich in einem Unterbaum mit der Wurzel dieses Knotens befinden.

(R4+) Die Objekte, die nicht vollständig in eine Region fallen, werden in allen Blattknoten, mit denen sie sich überschneiden, abgelegt. Das bedeutet, dass solche MBRs redundant gespeichert werden.

Ein Beispiel des R+-Baums zeigt die folgende Abbildung 3.5.

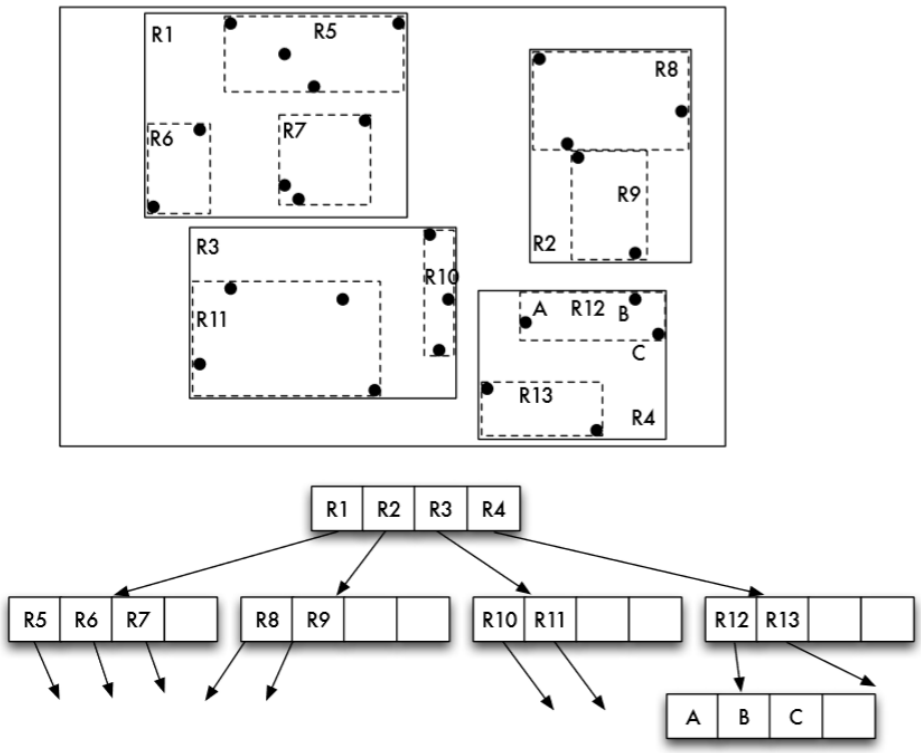


Abbildung 3.5: Struktur eines R+-Baumes

Quelle: [25]

Da der R+-Baum keine Überlappungen in den inneren Knotenebenen aufweist, sind die Suchpfade zur Blattebene stets eindeutig. Dies führt insbesondere bei Punktanfragen zu einer deutlichen Verbesserung der Laufzeit gegenüber der Suche mit R-Bäumen. Um Top-k-Anfragen effizient beantworten zu können, werden wir in dieser Arbeit eine weitere Variante eines R-Baumes konstruieren, die Disjunktheit auf den Suchbereichen der inneren Blattebenen sicherstellt *und* gleichzeitig auch die Datenobjekte, in unserem Fall die Datenpunkte des Suchraumes, disjunkt den Blättern bzw. MBRs des Baumes zuweist. Eine Variante, die hinsichtlich der Punktsuche die Vorteile der beiden oben dargestellten Varianten (R-Baum und R+-Baum) in sich vereint.

Als nächstes soll das Branch-and-Bound-Prinzip vorgestellt werden. Ausgangsbasis ist eine Relation R mit n verschiedenen reellwertigen Attributen $A_1, \dots, A_n$. Es sei $P = \{P_1, \dots, P_N\}, N \in \mathbb{N}$ eine gegebene Punktmenge, deren Elemente wir, gemäß den Ausführungen des vorangegangen Kapitels nach Anwendung einer entsprechenden Projektion auf den Einheitswürfel I^n, stets auch als multidimensionale Datenpunkte $P_i \in I^n$ auffassen dürfen.

3.2 Das Branch-and-Bound-Prinzip

Das Branch-and-Bound-Prinzip ist ein mathematisches Verfahren aus dem Bereich des Operations Research und wurde erstmals in den 60-ziger Jahren vorgestellt ([26] und [27]).

Branch-and-Bound (Verzweigen und Begrenzen) ist kein spezieller Algorithmus, sondern vielmehr eine mathematische Methode oder ein Prinzip, dem ein Entscheidungsbaum zugrundeliegt. Dabei werden Informationen über die Optimalität von Teillösungen herangezogen, um Untersuchungen

von potentiellen Lösungen zu vermeiden, die nicht optimal sein können. Der *Branch-Schritt* zerlegt ein gegebenes Problem in Teilprobleme mit dem Ziel, eine Vereinfachung des ursprünglichen Problems zu erreichen. Der *Bound-Schritt* hat das Ziel, die einzelnen Teile mit einer Schranke zu versehen, um entscheiden zu können, ob dieser für die Lösung herangezogen werden muss oder eben nicht. Man benötigt eine obere Schranke zur Lösung von Maximierungsproblemen, entsprechend eine untere zur Lösung einer Minimierungsaufgabe.

Die nun folgende kurze Darstellung des Verfahrens basiert auf den Ausführungen in [28]. Das Verfahren wird, wie oben bereits erwähnt, auf die Verwendung eines Baumes reduziert.

Von der Wurzel wird ein Verzweigungsschritt durchgeführt, der zu neuen Knoten führt, für die eine obere oder untere Grenze bestimmt werden muss. Die Berechnung einer solchen Grenze ist problemspezifisch. Wir werden bei der Darstellung verschiedener Anwendungen weiter unten sehen, was genau darunter zu verstehen ist. Grundsätzlich besagt diese Grenze, dass ab dieser keine Lösung mehr gefunden werden kann, die besser ist als von dieser Grenze angegeben.

Dies impliziert im Falle des Maximierungsproblems, dass man einen Pfad nicht weiterverfolgen wird, wenn auf der gleichen Ebene ein anderer Knoten existiert, der eine höhere obere Schranke besitzt. Es wird also immer der Knoten mit der höchsten Schranke gewählt und weiterverfolgt. Die Suche bricht erfolgreich ab, d.h. eine Lösung des Problems ist gefunden, wenn der Wert der Lösung größer im Vergleich zu den Grenzen der übrigen Knoten ist.

Es drängen sich sofort zwei Fragen auf. Wie lässt sich die Existenz einer solchen Schranke sicherstellen? Und wie kann man eine (obere oder untere) Schranke für die jeweiligen Knoten ermitteln? Die Antwort auf die erste Frage wird in Kapitel 4 gegeben.

In den nun folgenden Darstellungen wollen wir Verfahren vorstellen, mit denen man für die drei oben genannten Problemstellungen zu solchen Schranken gelangen kann.

Alle drei nun dargestellten Algorithmen bedienen sich zur Bestimmung der Ergebnisliste eines R-Baumes. Für das Durchsuchen der Ebenen eines Baumes gibt es mehrere Möglichkeiten, wovon wir zwei vorstellen werden. Die Erste ist eine *Tiefensuche* (engl. depth-first search, kurz DFS), die einen Pfad bis auf Blattebene durchsucht, bevor abgezweigte Pfade besucht werden. Das zweite Verfahren besucht nicht die Knoten auf den einzelnen Ebenen, sondern wählt aus einer Prioritätswarteschlange (engl. priority queue), welches Element als nächstes untersucht werden soll.

Bei einer *Prioritätswarteschlange*, auch Vorrangwarteschlange genannt, handelt es sich um eine erweiterte Form einer einfachen Warteschlange insofern, als das den zu verwaltenden Daten-objekten ein Schlüssel mitgegeben werden muss, mit dessen Hilfe sich eine auf- oder auch absteigende Reihenfolge der Objekte bestimmen lässt. Da eine solche Prioritätswarteschlange in den Verfahren zur Beantwortung von Skyline- und Top-k-Anfragen verwendet wird, wollen wir auf ihre Struktur etwas genauer eingehen. Die Darstellungen sind entnommen aus [29].

Die grundsätzliche Idee einer Prioritätswarteschlange ist es, Elemente mit Prioritäten einzufügen und das Element mit der jeweils höchsten Priorität, also dasjenige mit der kleinsten oder größten

Schranke zu entfernen. Sei eine Menge $\{e_1, \dots, e_n\}$ von Elementen mit Priorität $key(e_i)$ gegeben und wird das Element mit niedrigstem Wert gesucht, so benötigt man zur Erzeugung und Abarbeitung einer Warteschlange pq die folgenden Operationen:

(PQ1) Die Funktion $build(\{e_1, \dots, e_n\})$, um eine Warteschlange pq zu erzeugen.

(PQ2) Eine Prozedur $insert(e, pq)$, die ein neues Element e in die Warteschlange pq mit Priorität $key(e)$ einfügt.

(PQ3) Die Funktion $min(pq)$, die das Element mit der niedrigsten Priorität zurückgibt.

(PQ4) Eine Prozedur $deleteMin(pq)$, die das Element mit der niedrigsten Priorität aus der Warteschlange entfernt.

Soll nach maximalen Elementen gesucht werden, so benötigt man die beiden Funktionen $max(pq)$ und $deleteMax(pq)$. Prioritätswarteschlangen können auf verschiedene Weisen implementiert werden (siehe [30], [31], [32] und [33]). Die unterschiedlichen Methoden der Implementierung führen zu unterschiedlicher Leistungsfähigkeit für die oben aufgeführten Operationen (PQ1) bis (PQ4). In Abbildung 3.6 sind für eine Anzahl von N Elementen einige der gängigen Implementierungen und die zugehörigen Laufzeiten der Operationen dargestellt.

	Unsortierte Liste	**Sortierte Liste**	**Binärer Heap**	**Binominal-Heap**	**Fibonacci-Heap**
$insert$	$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(log(N))$	$\mathcal{O}(log(N))$	$\mathcal{O}(1)$
$merge$	$\mathcal{O}(1)$	$\mathcal{O}(N+M)$	$\mathcal{O}(N)$	$\mathcal{O}(log(N))$	$\mathcal{O}(1)$
$deleteMin$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(log(N))$	$\mathcal{O}(log(N))$	$\mathcal{O}(log(N))$ $*$

$(*)$ amortisierte Kosten

Abbildung 3.6: Implementierungsformen und Laufzeiten von Prioritätswarteschlangen

Quelle: [29]

Anzumerken ist, dass den guten Laufzeiten der Operationen einer sortierten Liste die Laufzeit von $\mathcal{O}(N \cdot log(N))$ für die Erzeugung der Liste (PQ1) entgegensteht. Das Erzeugen einer unsortierten Liste erfordert eine Laufzeit von $\mathcal{O}(N)$.

3.3 Nächste-Nachbarn-Suche

Das Ziel einer Nächste-Nachbarn-Suche ist die Bestimmung der zu einem gegebenen Anfrageobjekt nächstgelegenen Nachbarn. Bevor wir das Verfahren vorstellen, muss zuerst einmal festgelegt werden, was wir unter den nächsten Nachbarn verstehen möchten.

Definition 3.2 k-Nächste-Nachbarn

Sei M eine (Grund-)Menge mit $|M| \geq k$, $q \notin M$ ein Anfragepunkt, $k \in \mathbb{N}$ die gewünschte Anzahl gesuchter nächster Nachbarn von q und d eine Metrik, dann ist die Menge $NN(k, q, M)$ der k nächsten Nachbarn von q definiert als diejenige Teilmenge $S \subseteq M$, mit $|S| = k$, so dass es keine Menge $S' \subseteq M$mit $S' \cap S = \emptyset$ gibt, so dass ein $p \in S$ und $p' \in S$ existiert und $d(q, p) > d(q, p')$.

Eine k-NN-Anfrage ist entsprechend der Definition 3.2 eine Anfrage, welche zu einem gegebenen Anfragepunkt q und einer (kleinen) Zahl $k \in \mathbb{N}$ die (Ergebnis-)Menge $NN(k, q, M)$ bestimmt.

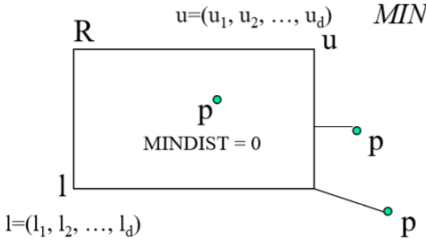
3.3.1 Nächste-Nachbarn-Suche im R-Baum mit Tiefensuche

Wir stellen nun die wesentlichen Ergebnisse der Arbeit „Next Neighbor Queries“ von Roussopoulos, Kelly und Vincent [34] vor, die das Branch-and-Bound-Prinzip im Kontext der Nächste-Nachbarn-Suche verwenden. Das vorgestellte Verfahren löst das Problem des nächsten Nachbarn, also für $k = 1$, zu einem gegebenen Anfragepunkt und ist als RKV-Algorithmus bekannt. Die nun folgenden Ausführungen erfolgen in Anlehnung an [35]. In der Arbeit von Roussopoulos werden MBRs immer mit R oder mit $R_i, i \in \mathbb{N}$ bezeichnet.

Aufgrund von Lemma 1 aus Kapitel 2 wissen wir, dass ein MBR im $\mathbb{R}^d$ durch seine kleinste Ecke $l = (l_1, \dots, l_d)$ und seine größte Ecke $u = (u_1, \dots, u_d)$ eindeutig bestimmt ist.

Mit Hilfe dieser beiden Punkte lassen sich zwei Abstandsfunktionen $MINDIST$ und $MINMAXDIST$ definieren, die in unterschiedlicher Weise die Distanz eines Punktes $P \in \mathbb{R}^d$ zu einem MBR R gleicher Dimension messen.

$MINDIST(P, R)$ ist die minimale Distanz zwischen $P \in \mathbb{R}^d$ zu einem MBR R. Wenn $P \in R$, also der Punkt innerhalb des MBR liegt, so ist der $MINDIST(P, R) = 0$. Liegt P außerhalb, gilt also $P \notin R$, so gibt die Metrik den Abstand von P zum nahegelegensten Punkt $Q \in R$ an.



$$MINDIST(P,R) = \sqrt{\sum_{i=1}^{d} (p_i - r_i)^2}$$

$$r_i = l_i \text{ if } p_i < l_i$$
$$= u_i \text{ if } p_i > u_i$$
$$= p_i \text{ otherwise}$$

Quelle: [34]

$MINMAXDIST(P, R)$ ist die minimale Distanz aller maximalen Distanzen von Punkten $P \in \mathbb{R}^d$ zu einer Seite des MBRs R. Sie gibt an, welchen maximalen Abstand vom Anfragepunkt P der nächste Nachbar aus dem MBR haben kann. Bei der Berechnung wird die minimal begrenzende Eigenschaft eines MBRs ausgenutzt, d.h. das jede $(n-1)$-dimensionale Hyperfläche an mindestens einen Datenpunkt stößt, der als möglicher nächster Nachbar innerhalb des MBRs in Frage kommt. Daher berechnet $MINMAXDIST$ für jede dem Anfragepunkt zugewandte Hyperfläche die maximale Distanz eines Punktes auf dieser Fläche. Von all diesen maximalen Distanzen wird als Ergebnis die minimale ermittelt. Die Definitionen der beiden Abstandsfunktionen lauten wie folgt: Sei $P \in \mathbb{R}^n$ zu einem MBR R gleicher Dimension, dann ist

$$MINMAXDIST(P,R) = \min_{1 \le k \le n} \left(|p_k - rm_k|^2 + \sum_{\substack{i \neq k \\ 1 \le i \le n}} |p_i - rM_i|^2 \right),$$

wobei

$$rm_k = \begin{cases} s_k, wenn\ p_k \leq \dfrac{s_k + t_k}{2} \\ t_k\ sonst \end{cases}$$

und

$$rM_i = \begin{cases} s_i, wenn\ p_i \geq \dfrac{s_k + t_k}{2} \\ t_i\ sonst \end{cases}.$$

Abbildung 3.7 veranschaulicht beide Metriken im zweidimensionalen Raum.

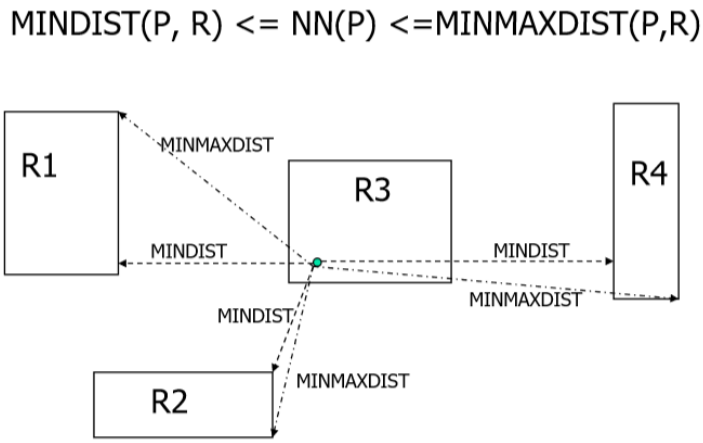


Abbildung 3.7: MINDIST und MINMAXDIST im $\mathbb{R}^2$

Quelle: [34]

Der RKV-Algorithmus realisiert eine Tiefensuche. Die initiale Distanz ($obereGrenze$) von P zu seinem nächsten Nachbarn wird auf „Unendlich“ festgesetzt. Beim Abstieg wird für jeden Nicht-Blattknoten die $MINDIST$ seiner Kindknoten berechnet und die Kindknoten anhand dieser Distanz aufsteigend sortiert in eine Liste, die vom Autor als Active Branch List (ABL) bezeichnet wird, eingefügt. Mit der Sortierung soll erreicht werden, dass zuerst die Kindknoten aufgesucht werden, von denen ausgegangen wird, dass sie den nächsten Nachbarn am ehesten enthalten.

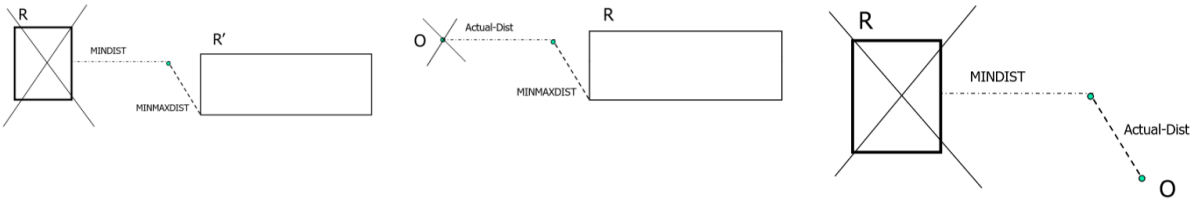
Um eine Traversierung des Baumes zu vermeiden, werden drei Regeln (Strategien) zur Reduzierung des Suchaufwandes angewandt.

(P1) $MINDIST(P, R_1) > MINMAXDIST(P, R_2)$: In diesem Fall kann R_1 keinen näheren Punkt als R_2 liefern und kann somit vernachlässigt werden.

(P2) $obereGrenze > MINMAXDIST(P, R)$: $obereGrenze$ steht für die Distanz zu einem bis jetzt ermittelten Nächste-Nachbarn-Kandidaten. Da R einen Punkt enthält, der näher zu P liegt, kann die Distanz $obereGrenze$ auf den Wert $MINMAXDIST(P, R)$ reduziert und R nach Regel (P3) ausgeschlossen werden.

(P3) $obereGrenze < MINDIST(P, R)$: Da R keinen besseren Kandidaten als den aktuellen Nächste-Nachbarn-Kandidaten liefern kann, braucht R nicht aufgesucht zu werden.

Die folgende Abbildung stellt alle drei Kriterien schematisch dar (Actual-Dist = $obereGrenze$).



P1 P2 P3

Abbildung 3.8: Verwerfungsregeln für k-NN-Suche

Quelle: [35]

Abbildung 3.9 zeigt den konkreten Algorithmus. Der Parameter $naechsterNachbar$ ist der Ausgabeparameter und enthält als Ergebnis den ermittelten nächsten Nachbarn.

```
    Algorithmus RKV (Punkt P, Knoten T, obereGrenze, naechsterNachbar)
    /* Lokale Variablen */
1   neuerKnoten  knoten
2   branchList   array
3   distanz      real
4   o            object
        /* Auf Blattebene – berechne Distanzen zum aktuellen Objekt */
5       If T ist Blattknoten Then
6           For Each o In T Do
7               Distanz ist Distanz zwischen P und o
8               If (distanz < obereGrenze) Then
9                   obereGrenze = dist
10                  naechsterNachbar = o
11              End If
12          End For
        /* Nicht-Blattebene – Sortieren und Pruning */
13      Else
14          branchList ist Liste von Kindknoten aus T
15          branchList nach MINDDIST sortieren
16          branchList nach Regel (P1) kürzen
17          obereGrenze nach Regel (P2) reduzieren
18          branchList nach Regel (P3) kürzen
19          For Each neuerKnoten In branchList Do
                /* Rekursiver Aufruf */
20              RKV (P, neuerKnoten, obereGrenze, naechsterNachbar)
21              branchList nach Regel (P3) kürzen
22          End For
23      End If
    End RKV
```

Abbildung 3.9: RKV-Algorithmus Pseudocode

Quelle: [34]

Das Verfahren lässt sich zur Berechnung von $k > 1$ nächsten Nachbarn mit einem gewissen Aufwand modifizieren. Dazu benötigt man eine Prioritätswarteschlange zur Verwaltung der k Nächste-

Nachbarn-Kandidaten, die diese Kandidaten nach ihrer Distanz zum Anfragepunkt sortiert verwaltet. Für eine genauere Darstellung sei auf [35] verwiesen.

Fazit

Im Gegensatz zum einfachen Tiefensuche-Algorithmus, der mit einem beliebigen Pfad beginnt und wodurch die ersten ermittelten Kandidaten sehr weit weg vom Anfragepunkt liegen können, schränkt der RKV-Algorithmus den Suchraum durch Verwendung der $MINMAXDIST$ Funktion schneller ein. Dennoch kann es passieren, dass der RKV Tiefendurchlauf stark fehlgeleitet wird und unnötige Knoten des Baumes in die Suche einbezogen werden. Dies ist ein prinzipielles Problem der Tiefensuche. Hinzu kommt, dass das Erzeugen sortierter Listen im Allgemeinen nicht optimal ist, da das Sortieren von N Objekten $\mathcal{O}\big(N \cdot log(N)\big)$ Schritte erforderlich macht.

Alle nun folgenden Verfahren nutzen zur Bestimmung der Ergebnisliste einen R-Baum, deren MBRs auf jeder Nichtblattebene mit entsprechenden Grenzen versehen wird. Statt eines rekursiven Durchlaufs mit einer Tiefensuche, entscheidet das Objekt mit der höchsten Priorität, d.h. das mit der größten oberen oder der kleinsten unteren Schranke, einer Prioritätswarteschlange über den weiteren Suchverlauf. Lemma 4.3 in [36] und die Ausführungen in [37] zeigen, dass diese Verfahren optimal hinsichtlich der zu besuchenden Knoten sind.

Im folgenden Kapitel stellen wir die Nächste-Nachbarn-Suche im R-Baum mit Hilfe des Best-First-Verfahrens (BF) von Hjaltason und Samet [36] an einem Beispiel vor. Die Beschreibung und Darstellung der Beispiele der beiden darauffolgenden Verfahren können wir dann kürzer halten, da sie sich im Wesentlichen nur durch die Methode zur Bestimmung der Grenzen der MBRs unterscheiden. Datenpunkte werden als P_i, MBRs als E_i für $i \in \mathbb{N}$ dargestellt. Die verwendete Darstellungsform der Warteschlangen kann von der einer konkret realisierten Implementierung abweichen. Die Werte in den Knoten und auf der Blattebene sind in den folgenden Beispielen lediglich zu Illustrationszwecken aufgeführt.

3.3.2 Nächste-Nachbarn-Suche im R-Baum mit dem Best-First-Verfahren

Zur Bestimmung der oberen Grenzen für jedes MBR wird die $MINDIST$-Metrik verwendet, die den minimalen Abstand $MINDIST(q, o)$ eines Objektes o (Datenpunkt oder MBR) zu einem gegebenen Anfragepunkt q angibt. Die zugrundeliegende Punktmenge sei wieder durch das Beispiel von Abbildung 2.12 gegebenen. Abbildung 3.10 enthält zusätzlich noch den $MINDIST(P, q)$ Abstand eines jeden Datenpunktes P mit ID $\in \{1, \dots, 12\}$ zum Anfragepunkt q. Abbildung 3.10 macht damit schon deutlich, dass P_{10} der gesuchte nächste Nachbar (ID = 10) zum Anfragepunkt sein wird.

ID	1	2	3	4	5	6	7	8	9	10	11	12
Attribut 1	0,2	0,1	0,3	0,2	0,3	0,5	0,4	0,6	0,7	0,6	0,7	0,7
Attribut 2	0,2	0,5	0,3	0,9	0,8	0,7	0,3	0,1	0,2	0,5	0,6	0,5
Abstand	0,39	0,35	0,25	0,47	0,33	0,2	0,21	0,43	0,39	0,15	0,27	0,25

Abbildung 3.10: Punktmenge mit MINDIST-Abstand zum Anfragepunkt

Die Punktmenge sei mit Hilfe des Median-Split-Verfahrens in jeder Koordinate einmal partitioniert. Der Anfragepunkt sei gegeben durch $q = (0{,}45; 0{,}5)$. Die Partitionierungen, sowie die $MINDIST$-Abstände der rot markierten MBRs zu q sind in Abbildung 3.11, links dargestellt. Der zugehörige R-Baum (Abbildung 3.11, rechts) zeigt alle MBRs und die entsprechenden Abstände $MINDIST(E_j, q)$ für jedes MBRs $E_j, j = 1, \dots, 6$ auf Nicht-Blattebene, sowie die Datenpunkte P_i und ihre Abstände $MINDIST(P_i, q)$ für jedes $i = 1, \dots, 12$.

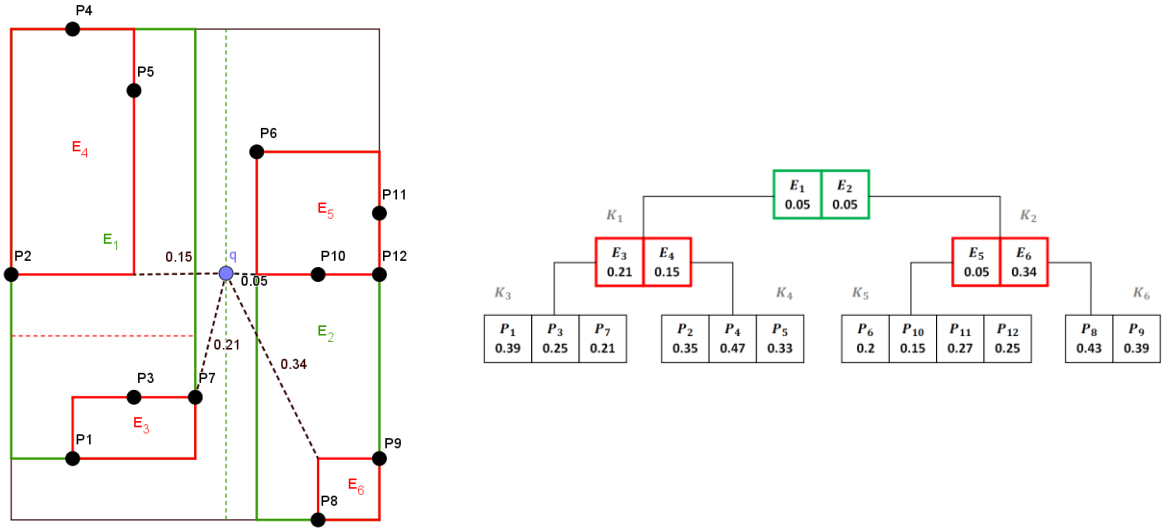
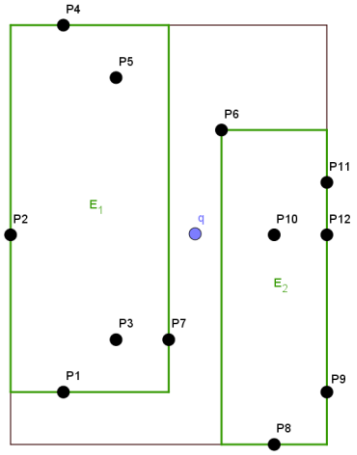


Abbildung 3.11: Median Split und R-Baum

Jeder innere Knoten des Baumes enthält den Knotenschlüssel, den Abstand zum Anfragepunkt, sowie die Referenz auf seine Kindknoten. Zur Ermittlung des nächsten Nachbarn zum Anfragepunkt q sind für dieses Beispiel vier Schritte erforderlich, die in den folgenden Abbildungen dargestellt sind.

Im ersten Schritt werden beide MBRs E_1 und E_2 des Wurzelknotens in die Prioritätswarteschlange aufgenommen. Die bisherige Ergebnismenge ist leer, da der Wurzelknoten keine Datenpunkte enthalten kann.



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$

Abbildung 3.12: Schritt 1 BF-Algorithmus

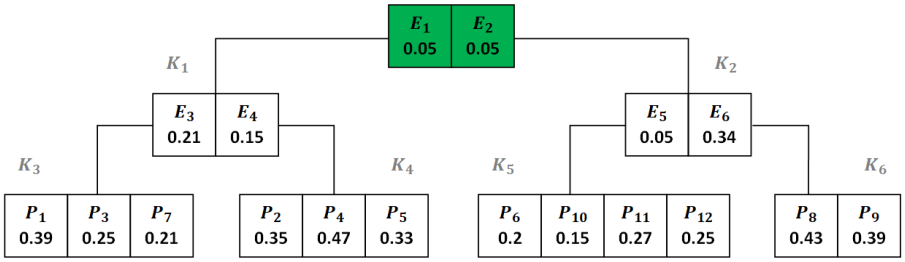
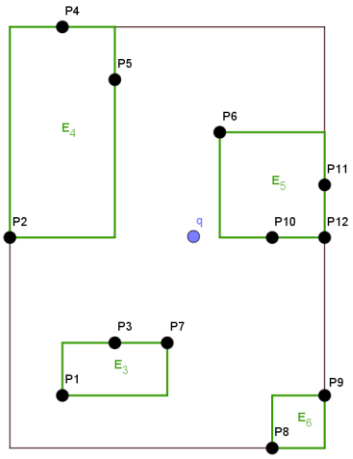


Abbildung 3.13: Schritt 1 BF-Algorithmus (R-Baum)

Da beide MBRs E_1 und E_2 denselben Abstand zum Anfragepunkt besitzen, folgt man im zweiten Schritt beiden auf die (nächste) Ebene ihrer Kindknoten. Beide Elternknoten werden aus der Warteschlange entfernt (*deleteMin*-Funktion) und durch ihre Kindknoten ersetzt (*insert*-Prozedur).



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$
Folge E_1 und E_2	$(E_5, 0.05), (E_4, 0.15), (E_3, 0.21), (E_6, 0.34)$	$\emptyset$

Abbildung 3.14: Schritt 2 BF-Algorithmus

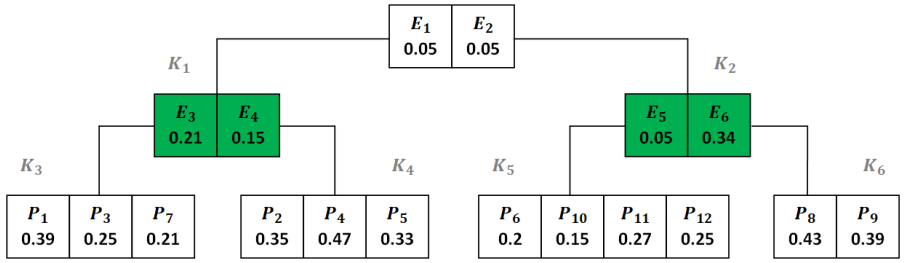
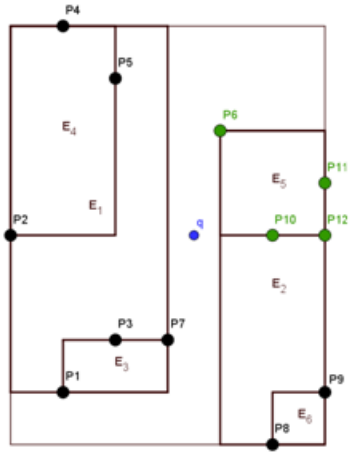


Abbildung 3.15: Schritt 2 der k-NN-Suche (R-Baum)

E_5 hat den geringsten Abstand zum Anfragepunkt q, so dass man im nächsten Schritt (Schritt 3) diesem MBR und nur diesem auf die nächste Knotenebene der Blattknoten folgt. E_5 sind die vier Punkte des Knotens K_5 zugeordnet, die den Vaterknoten in der Warteschlage ersetzen (Aufruf von *insert* und *deleteMin*).



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$
Folge E_1 und E_2	$(E_5, 0.05), (E_4, 0.15), (E_3, 0.21), (E_6, 0.34)$	$\emptyset$
Folge E_5	$(E_4, 0.15), (P_{10}, 0.15), (P_6, 0.2), (E_3, 0.21), (P_{12}, 0.25), (P_{11}, 0.27), (E_6, 0.34)$	$\emptyset$

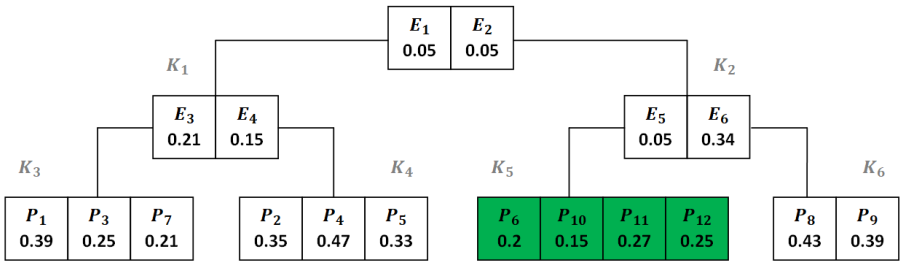
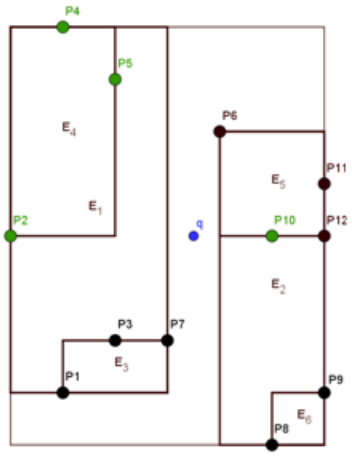


Abbildung 3.16: Schritt 3 BF-Algorithmus (oben), zugehöriger R-Baum (unten)

E_4 und P_{10} haben beide den kleinsten Abstand zu q, als alle bislang aufgenommenen Objekte, so dass man auch dem MBR E_4 auf die Blattknotenebene folgen muss (Schritt 4). E_4 wird dann entsprechend durch die Datenpunkte des Knotens K_4 ersetzt. Nach erneuter Bestimmung des Objektes mit dem kleinsten Abstand zum Anfragepunkt, ergibt sich damit P_{10} als der gesuchte nächste Nachbar zu q, so dass das Verfahren an dieser Stelle erfolgreich abbricht.



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$
Folge E_1 und E_2	$(E_5, 0.05), (E_4, 0.15), (E_3, 0.21), (E_6, 0.34)$	$\emptyset$
Folge E_5	$(E_4, 0.15), (P_{10}, 0.15), (P_6, 0.2), (E_3, 0.21), (P_{12}, 0.25), (P_{11}, 0.27), (E_6, 0.34)$	$\emptyset$
Folge E_4	$(P_{10}, 0.15), (P_6, 0.2), (E_3, 0.21), (P_{12}, 0.25), (P_{11}, 0.27), (P_5, 0.33), (P_2, 0.35), (E_6, 0.34), (P_4, 0.47)$	$\{P_{10}\}$

Abbildung 3.17: Schritt 4 BF-Algorithmus

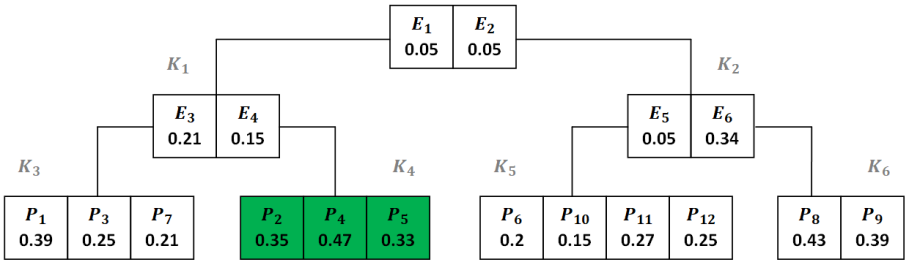


Abbildung 3.18: Schritt 4 BF-Algorithmus (R-Baum)

Fazit

Das Best-First-Verfahren ist optimal hinsichtlich besuchter Knoten. Es werden nur diejenigen MBRs besucht, die notwendig sind, um die Ergebnismenge bestimmen zu können. Dafür kann aber die Prioritätswarteschlange recht umfangreich werden. Effizienzsteigernd ist es, wenn man so wenig Objekte wie nötig in die Warteschlange überträgt. Ob und wie das möglich ist, hängt vom jeweiligen Verfahren ab. In Kapitel 3.5.2 werden wir zeigen, wie diese Fragestellung für den in dieser Arbeit verwendeten BRS-Algorithmus gelöst werden kann.

3.4 Skyline-Anfragen

Das Skyline-Problem ist ebenso wie die Suche nach den Top-k-Objekten auf die Ermittlung einer Bestenliste mit mehreren Elementen ausgerichtet. Beide Probleme können mit dem Branch-and-Bound-Prinzip gelöst werden. Die Skyline ist definiert als die Menge derjenigen Datenpunkte, die von keinem anderen Punkt des Datenraumes dominiert werden. Ein Datenpunkt dominiert einen anderen, wenn er in allen Dimensionen entweder einen gleichen, oder einen besseren Wert besitzt, in mindestens einer dieser Dimensionen aber besser ist. Man erhält die folgende Definition.

Definition 3.3 Dominanz, Skyline

Es sei $P = \{P_1, \dots, P_N\}, N \in \mathbb{N}$ die gegebene Punktmenge im $\mathbb{R}^d$ mit $P_i = (p_{i1}, \dots, p_{id})$ für jedes i. Dann *dominiert* der Punkt $P_i = (p_{i1}, \dots, p_{id})$ den Punkt $P_j = (p_{j1}, \dots, p_{jd})$, in Zeichen $P_i \succ P_j$, falls $p_{is} \geq p_{js}$ für jedes $s = 1, \dots, d$ gilt und mindestens ein $s = 1, \dots, d$ existiert, so dass $p_{is} \neq p_{js}$ ist.

Die *Skyline* $S(P)$ einer Menge P ist die Menge derjenigen Punkte von P, die von keinem Punkt anderen Punkt dominiert wird. Also:

$$S(P) = \{P_j \in P \mid \nexists\, P_i \in P : P_i \succ P_j\}.$$

Jeder Punkt der Skyline spannt einen Raum über Punkten auf, die er dominiert. Gemeinsam decken die Punkte der Skyline alle Punkte des Datensatzes ab. Abbildung 3.19 zeigt die Skyline für die Punktmenge des Beispiels (Abbildung 2.12) und veranschaulicht, welcher Punkt welchen Bereich dominiert.

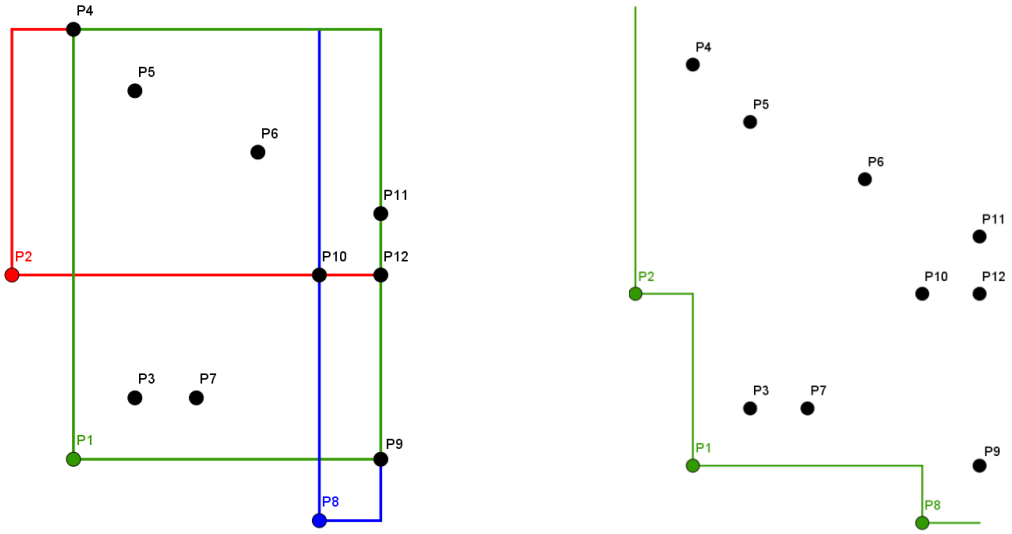


Abbildung 3.19: Dominierter Raum (links), Skyline von Punkten (rechts)

Es gibt mehrere Möglichkeiten, die Skyline zu einer gegebenen Punktmenge zu bestimmen. In [38] werden einige Methoden vorgestellt. Von besonderem Interesse im Kontext unserer Arbeit ist das auf der Basis des Branch-and-Bound-Prinzips entwickelte BBS-Verfahren von Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, und Chun Kit Hui [39], das nun vorgestellt wird.

3.4.1 BBS-Algorithmus mit R-Bäumen

Grundlage ist wieder eine Indizierung der Datenpunkte mit Hilfe eines R-Baumes und die Verwendung einer Prioritätswarteschlange, dessen Einträge nach der Manhattan-Distanz $d(e, 0)$ (siehe Abbildung 2.5) eines Objektes e (Datenpunkt oder MBR) zum Usprung 0 verwaltet werden. Ziel ist es, die Skyline gemäß obiger Definition zu ermitteln. Speziell in diesem Verfahren wird die Liste der vorläufig gefundenen Skylineobjekte genutzt, um Datenregionen ausschließen (pruning), die von diesen Objekten dominiert werden. Das Verfahren wird solange fortgeführt, bis die Prioritäts-warteschlange leer ist. Der Pseudocode ist in Abbildung 3.20 dargestellt.

```
    Algorithmus BBS (R-Baum R)
1   S ≔ ∅ /* Skyline */
2   Insert alle Einträge der Wurzel in die Warteschlange PQ
3   While die Warteschlange nicht leer Do
4       Delete ersten Eintrag e aus PQ /* d(e,0) ist minimal */
5       If e von einem Punkt in S dominiert wird Then
6         Ignore e
7       Else /* e wird nicht dominiert */
8          If e kein Datenpunkt ist Then
9                For Each Kind e_i von e
10                    If e_i not von einem Punkt in S dominiert wird Then
11                          Insert e_i in PQ
12                    End If
13               End For
14         Else /* e ist ein Datenpunkt */
15           Delete diejenigen Elemente aus S, die von e dominiert werden
16           Insert e in S
17         End If
18       End If
19  WEnd
20  Return S
    End BBS
```

Abbildung 3.20: Pseudocode BBS-Algorithmus

Quelle: [40]

3.4.2 BBS am Beispiel

Auch dieses Verfahren soll mit Hilfe eines einfachen Beispiels veranschaulicht werden. Wir organisieren die Datenpunkte unseres durchgängigen Beispiels, mit je einer Teilung pro Koordinate, mit dem Median-Split-Verfahren und ordnen die Punkte den MBRs gemäß Abbildung 3.21 zu. Im Folgenden werden nur diejenigen Suchschritte graphisch dargestellt, die sich wesentlich von der Vorgehensweise zu denen aus Kapitel 3.3.1 unterscheiden. Wie im Beispiel von oben werden Datenpunkte wieder mit P_i und MBRs mit E_i für $i \in \mathbb{N}$ bezeichnet. Die eingetragenen Werte der MBRs und der Punkte bezeichnen die Manhattan-Distanz der jeweiligen Objekte zum Ursprung.

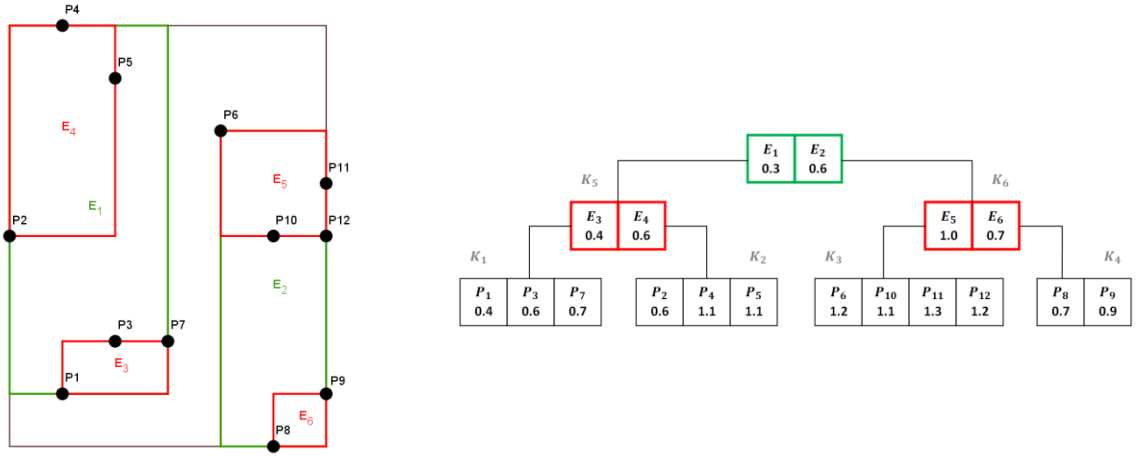
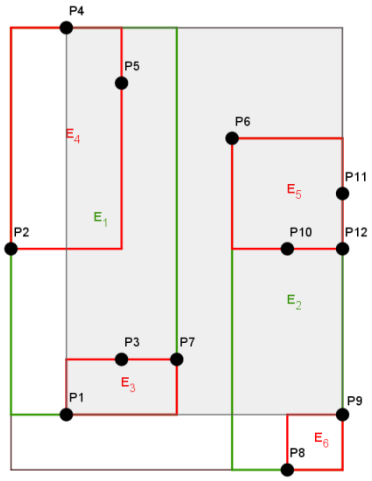


Abbildung 3.21: R-Baum für BBS-Verfahren

Im ersten Schritt werden die beiden MBRs E_1 und E_2 in die Vorrangwarteschlage geladen, und man folgt dem MBR mit dem kleinsten Abstand zum Ursprung, also E_1 auf die nächste Knotenebene und ersetzt dieses MBR durch seine zugeordneten Kindknoten E_3 und E_4. Danach folgt man dem MBR mit dem kleinsten Abstand, also E_3, und erreicht erstmals die Blattebene. E_3 wird durch den Datenpunkt P_1 ersetzt, da er von den Punkten P_1, P_3 und P_7 den geringsten Abstand zum Ursprung besitzt. P_1 gehört damit zur Lösungsmenge. Man betrachtet den von P_1 dominierten Bereich des MBRs aller Datenpunkte, dargestellt als graue Fläche in Abbildung 3.22, links.



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.3), (E_2, 0.6)$	$\emptyset$
Folge E_1	$(E_3, 0.4), (E_2, 0.6), (E_4, 0.6)$	$\emptyset$
Folge E_3	$(P_1, 0.4), (E_2, 0.6), (E_4, 0.6)$	$\{P_1\}$

Abbildung 3.22: Dominierter Bereich des Datenpunktes P1 (links), Warteschlange (rechts)

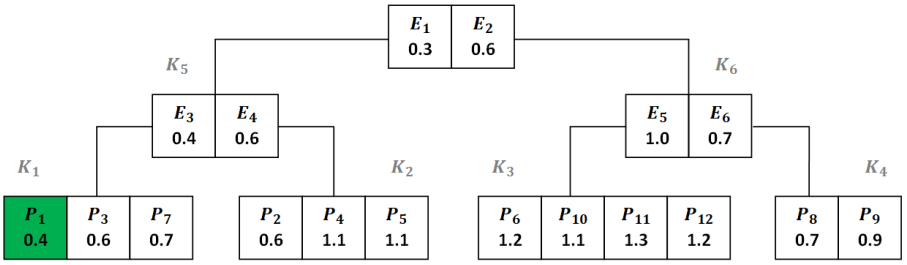
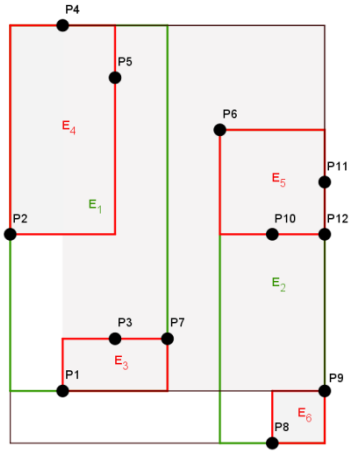


Abbildung 3.23: Blattknoten mit P1 (R-Baum)

Das MBR E_5 liegt in dem vom Punkt P_1 dominierten Bereich und würde, sofern es sich in der Warteschlange befände, entfernt werden können. Jetzt folgt man E_2 auf die nächste Ebene, das dadurch durch seine beiden Kinder E_5 und E_6 ersetzt wird. E_4 hat nun die höchste Priorität in der Warteschlange, d.h. den geringsten Abstand zum Ursprung und wird im nächsten Schritt durch P_2 in der Warteschlange ersetzt. Da P_2 das MBR E_5 dominiert, kann E_5 aus der Warteschlange entfernt werden. Als letztes wird E_6 durch den Datenpunkt P_8 ersetzt. Damit sind alle MBRs der Vorrangwarteschlage verarbeitet, so dass der Algorithmus erfolgreich abbricht und die Datenpunkte P_1, P_2 und P_8 die gesuchte Skyline bilden (siehe Abbildung 3.25).



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.3), (E_2, 0.6)$	$\emptyset$
Folge E_1	$(E_3, 0.4), (E_2, 0.6), (E_4, 0.6)$	$\emptyset$
Folge E_3	$(P_1, 0.4), (E_2, 0.6), (E_4, 0.6)$	$\{P_1\}$
Folge E_2	$(E_4, 0.6), (E_6, 0.7), (E_5, 1.0)$	$\{P_1\}$
Folge E_4	$(P_2, 0.6), (E_6, 0.7), (E_5, 1.0)$	$\{P_1, P_2\}$
Lösche E_5	$(E_6, 0.7)$	$\{P_1, P_2\}$
Folge E_6	$(P_8, 0.7)$	$\{P_1, P_2, P_8\}$

Abbildung 3.24: Dominierter Bereich aller Datenpunkte der Skyline (links), Warteschlange (rechts)

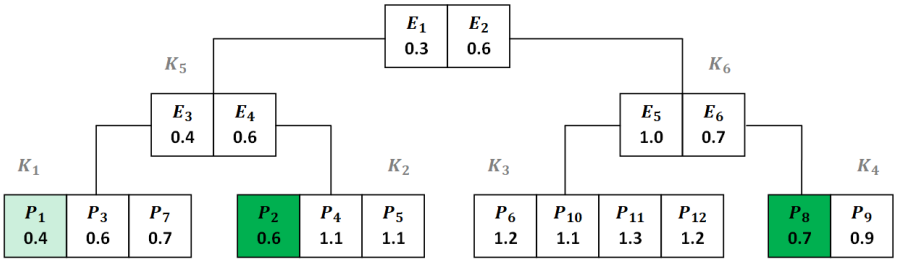


Abbildung 3.25: Punkte der Skyline (R-Baum)

Fazit:

Durch den Einsatz eines R-Baumes wird vermieden, dass in „leeren“ Datenregionen gesucht werden muss. Zusätzlich können im BBS-Verfahren durch bereits gefundene Kandidaten Teilbäume für die weitere Suche ausgeschlossen werden. Untersuchungen haben gezeigt, dass der BBS-Algorithmus in Bezug auf CPU- und IO-Zeit anderen Skyline-Varianten überlegen ist und der BBS optimal bzgl. der Seitenzugriffe im R-Baum (I/O-optimal) ist.

3.5 Top-k-Anfragen

In der Arbeit von Tao, Hristidis, Papadias und Papakonstantinou [3] aus dem Jahre 2007 wurde gezeigt, wie sich die Branch-and-Bound-Methode auch zur Beantwortung von Top-k-Anfragen anwenden lässt. Der daraus abgeleitete Algorithmus zur Bestimmung der Top-k-Objekte wird von den Autoren *Branch-and-Bound Ranked Search*, kurz *BRS* genannt.

Diese Arbeit und speziell der BRS-Algorithmus stehen im Fokus unserer Untersuchungen. Wir werden insbesondere auf das Verfahren zur Bestimmung der oberen Schranken und die Ausführungen, die zwischen monotonen und nicht-monotonen Bewertungsfunktionen differenzieren, ausführlicher eingehen. Die Darstellung des Algorithmus selbst lässt sich allerdings kurzhalten, da er, wie die Autoren selbst anmerken, nach entsprechender Bestimmung geeigneter Schranken schlussendlich analog zum oben dargestellten Branch-and-Bound-Ansatz für k-NN-Anfragen verläuft.

Die Problemstellung wird in [3] wie folgt beschrieben:

Gegeben sei eine Relation R mit m numerischen Attributen $A_1, \dots, A_m$. Mit Verweis auf [41] und [42] wird ohne Beschränkung der Allgemeinheit vorausgesetzt, dass die Menge der Datenpunkte eine Teilmenge des Einheitswürfels ist. Die Punkte des Datenraumes entsprechen wieder denjenigen aus unserem Beispiel (Abbildung 2.12). Weiter wird darauf hingewiesen, dass es sich um eine dreistellige Relation mit zwei reellwertigen Attributen handelt und dass sich das Verfahren ganz analog auf beliebige (endliche) Dimensionen erweitern lässt.

Zu beachten ist, dass es grundsätzlich möglich ist, dass die Ergebnismenge der Top-k-Datenpunkte mehr als k Punkte enthält, d.h. dass Ränge mehrfach vorkommen können. Wählt man beispielweise $k = 1$ und die zweidimensionale Bewertungsfunktion $f(x_1, x_2) = x_2$, so haben die drei Datenpunkte P_9, P_{11} und P_{12} denselben (größten) Rang bei 0,7. Diese Erkenntnis wird natürlich auch bei nicht-monotonen Bewertungsfunktionen eine wichtige Rolle spielen. Es wird aber darauf hingewiesen, dass das Verfahren auch in diesen Fällen das richtige Ergebnis, also alle Lösungen liefert.

Dem Verfahren selbst wird die bekannte Struktur eines R-Baumes zugrundegelegt, allerdings wird in [3] keine spezielle Methode zur Konstruktion angegeben, sondern lediglich ein Standard-R-Baum vorausgesetzt. Der BRS-Algorithmus selbst funktioniert in Analogie zur Nächste-Nachbarn-Suche unter Verwendung einer Prioritätswarteschlange. Der wesentliche Unterschied besteht im Verfahren zur Bestimmung der Schranken für jedes MBR.

Der BRS-Algorithmus benötigt zur Maximierung der Bewertungsfunktion eine obere Schranke für jedes MBR. Genauer: Sei H ein MBR und f eine Bewertungsfunktion, dann verwendet man als obere Schranke den größten Funktionswert, den f auf H annehmen kann. Analog ist die untere Schranke der kleinste Funktionswert von f auf H, die wir mit $f_{max}(H)$ und $f_{min}(H)$ bezeichnen wollen. Die wesentliche Aufgabe besteht also in der Entwicklung eines Verfahrens zur effizienten Bestimmung dieser Grenzen, so dass der Fokus der weiteren Untersuchungen die Bewertungsfunktion sein wird.

3.5.1 Maximumprinzip monotoner Bewertungsfunktionen

Ist die Bewertungsfunktion f einer Top-k-Anfrage monoton (Definition 2.5), dann nimmt sie auf einem achsenparallelen Hyperrechteck (MBR) ihr Maximum stets auf einer ausgezeichneten Ecke an. Ist sie monoton fallend, so ist es die kleinste Ecke, ist sie monoton steigend, so ist es die größte Ecke gemäß Definition 3.1 eines MBR.

Die Autoren bezeichnen diese Ecke auch als *dominierende* Ecke e eines n-dimensionalen Hyperrechteckes H, da $f(e) \geq f(x)$ für jedes $x \in H$ gilt und $f(e)$ sich damit als Obergrenze eines MBR verwenden lässt.

Die einfachste Möglichkeit zur Bestimmung der dominierenden Ecke eines n-dimensionalen Hyperrechtecks für eine monotone Bewertungsfunktion besteht natürlich darin, die Funktionswerte aller 2^n Ecken zu bestimmen und den größten auszugeben. Dieses Vorgehen lässt sich, wie oben erwähnt, für den monotonen Fall optimieren und auf diejenigen nicht-monotonen Funktionen erweitern, deren Komponentenfunktionen gemäß Definition 3.4 monoton sind.

Es sei $x^0 = (x_1^0, \dots, x_n^0) \in D$ ein beliebiger, aber fest gewählter Punkt des Definitionsbereiches einer multivariaten Funktion $f: D \to \mathbb{R}, x \mapsto f(x_1, \dots, x_n)$. Verändert man nur das Argument in der i-ten Komponente und hält alle übrigen fest, so ist durch $x_i \mapsto f_i(x_1^0, x_2^0, \dots, x_{i-1}^0, x_i, x_{i+1}^0, \dots x_n^0)$ eine Funktion in einer Variablen definiert. Sie heißt *i-te Komponentenfunktion* von f.

Definition 3.4 Monotonie in einer Komponente

Eine Funktion $f: D \to \mathbb{R}, x \mapsto f(x_1, \dots, x_n), D \in \mathbb{R}^n$ heißt steigend (fallend) *in der i-ten Komponente*, wenn f_i, unabhängig von der Wahl von x^0, eine steigende (fallende) Funktion von x_i ist. Wenn eine Funktion steigend (fallend) ist, spricht man allgemein von einer monotonen Funktion in der i-ten Komponente.

Im Falle einer Bewertungsfunktion f, die auf einem n-dimensionalen achsenparallelen Hyperrechteck H definiert ist und deren Komponentenfunktionen f_i *alle* monoton sind, lässt sich die dominierende Ecke wie folgt bestimmen.

Wird H (gemäß Lemma 1) durch die untere Ecke $l = (l_1, \dots, l_n)$ und die obere Ecke $h = (h_1, \dots, h_n)$ aufgespannt, dann besitzt die gesuchte dominierende Ecke die Koordinaten

$$e = (e_1, \dots, e_n),$$

mit

$$e_i = \begin{cases} l_i, \text{falls } f_i \text{ monoton fallend,} \\ h_i, \text{falls } f_i \text{ monoton steigend.} \end{cases}$$

In [3] wird der folgende Pseudocode zur Bestimmung der dominierenden Ecke angegeben:

```
    Algorithm getMaxScore (H = (I_1, h_1, I_2, h_2, ..., I_n, h_n), f)
    /* M is a MBR with extent [I_i, h_i] along the i-th dimension (1 ≤ i ≤ n), and f the
    ranking function */
1   Initiate a point p whose coordinates are not decided yet
2   For i = 1 To n /* examine each dimension in turn */
3       If f is increasingly monotone on this dimension Then
4         the i-th coordinate of p is Set to h_i
5       Else /* f is decreasingly monotone on this dimension */
6         the i-th coordinate of p is Set to l_i
7       End If
8   End For
9   Return f(p)
    End getMaxScore
```

Abbildung 3.26: Pseudocode getMaxScore für MBR

Quelle:[3], Fig. 4

Ist auch nur eine Komponentenfunktion von f nicht monoton, so lässt sich getMaxScore nicht mehr anwenden.

3.5.2 Der BRS-Algorithmus

Ist der Algorithmus zur Berechnung der Schranken festgelegt, so lässt sich das BRS-Verfahren gemäß [3], Fig. 5 darstellen. Eingabevariablen sind ein R-Baum, eine Bewertungsfunktion und die Anzahl zu berechnender Top-k-Kandidaten. Die in dieser Arbeit verwendete Form des BRS (siehe Abbildung 3.27) ist eine modifizierte Darstellung, die sich dahingehend von der Originalversion aus [3] unterscheidet, dass beim Besuch der Blattebene immer nur die Top-k-Kandidaten des jeweiligen MBRs in die Warteschlange übertragen werden (siehe Zeile 6) und die Warteschlange als unsortierte Liste implementiert wird. Dadurch wird erreicht, dass beim Besuch eines MBRs auf unterster Ebene für nur k Datenpunkte die entsprechenden Funktionswerte berechnet werden müssen.

In Zeile 11 erfolgt der Aufruf der Funktion getMaxScore mit Übergabe eines MBRs H und der Bewertungsfunktion f, welche $f_{max}(H)$ bestimmt. Wir wollen dieses leicht modifizierte Verfahren im Folgenden als BRS+-Algorithmus bezeichnen.

```
    Algorithm BRS+ (RTree, f, k)
 1  Unsortierte Liste PQ
 2  Lade Wurzel von RTree in die PQ
 3  While die top-k Elemente der PQ nicht nur Punkte sind Do
 4        For Each MBR H von PQ mit maximaler Schranke Do
 5            If H ist aus der letzten innerer Ebene Then
 6                For Each top-k Punkt von H.Punkte Do
 7                   Insert(Punkt, Punkt.f(Punkt), PQ)
 8                End For
 9            Else
10                For Each Kind von H.Kindknoten Do
11                   Kind.maxscore = getMaxScore(Kind, f)
12                   Insert(Kind, Kind.maxscore, PQ)
13                End For
14            End If
15            Delete H
16        End For
17  WEnd
18  Return top-k Elemente der PQ
    End BRS+
```

Abbildung 3.27: Pseudocode des BRS+-Algorithmus
Angelehnt an Quelle:[3], Fig. 5

Es bleibt zu zeigen, dass der BRS+-Algorithmus die korrekten Ergebnisse liefert.

Lemma 2 Der BRS+-Algorithmus ist korrekt

Beweis: Da wir lediglich die Übertragungsroutine der Datenpunkte in die unsortierte Warteschlange geändert haben, bleibt zu zeigen, dass sich dadurch am Ergebnis gegenüber dem ursprünglichen BRS nichts ändert. Dazu bedenke man, dass wenn ein Datenpunkt ein möglicher Top-k-Kandidat ist, er auf jeden Fall einer der besten k ihm zugeordneten MBRs sein muss. Erfüllen weniger als k Kandidaten das Abbruchkriterium (im schlimmsten Fall alle k), so kommen als nächste Kandidaten wieder nur die besten k des kommenden MBRs in Frage. Dieses Verfahren setzt sich bis dann zum letzten möglichen MBR fort. ■

Wenn von nun an die Rede von BRS ist, so ist mit dem zugehörigen Algorithmus immer der von uns in Abbildung 3.27 dargestellte BRS+-Algorithmus gemeint. Das grundsätzliche Verfahren, also die Bestimmung der Lösungsmenge erfolgt dabei wie in [3] beschrieben und wird im Folgenden als BRS-Verfahren bezeichnet. Es soll zum Schluss an einem Beispiel veranschaulicht werden.

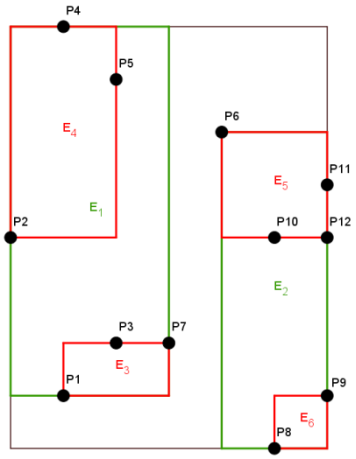
3.5.3 BRS am Beispiel

Der Algorithmus zur Bestimmung des besten Datenpunktes $(k = 1)$ soll nun an einem einfachen Beispiel der monoton steigenden Bewertungsfunktion $f: \mathbb{R}^2 \to \mathbb{R};\ (x_1, x_2) \mapsto x_1 + x_2$ dargestellt werden. Die Menge der Datenpunkte sei wieder durch die 12 Punkte unseres durchgängigen Beispiels gegeben. Abbildung 3.28 enthält zusätzlich noch die zugehörigen Funktionswerte von f.

ID	1	2	3	4	5	6	7	8	9	10	11	12
Attribut 1	0,2	0,1	0,3	0,2	0,3	0,5	0,4	0,6	0,7	0,6	0,7	0,7
Attribut 2	0,2	0,5	0,3	0,9	0,8	0,7	0,3	0,1	0,2	0,5	0,6	0,5
Score	0,4	0,6	0,6	1,1	1,1	1,2	0,7	0,7	0,9	1,1	1,3	1,2

Abbildung 3.28: Datenpunkte mit Rang für BRS

Die Organisation der Datenpunkte erfolgt wie im vorherigen Verfahren durch Anwendung des Median-Split-Verfahrens durch eine Teilung je Koordinate. Die Zuordnung der Punkte zu den MBRs, der sich daraus ergebende R-Baum und die (schematische) Zusammenfassung aller Suchschritte sind in der folgenden Abbildung 3.29 dargestellt.



Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 1.3), (E_2, 1.4)$	$\emptyset$
Folge E_2	$(E_5, 1.4), (E_1, 1.3), (E_6, 0.9)$	$\emptyset$
Folge E_5	$(E_1, 1.3), (P_{11}, 1.3), (P_{12}, 1.2), (P_6, 1.2), (P_{10}, 1.1), (E_6, 0.9)$	$\emptyset$
Folge E_1	$(P_{11}, 1.3), (E_4, 1.2), (P_{12}, 1.2), (P_6, 1.2), (P_{10}, 1.1), (E_6, 0.9), (E_3, 0.7)$	$\{P_{11}\}$

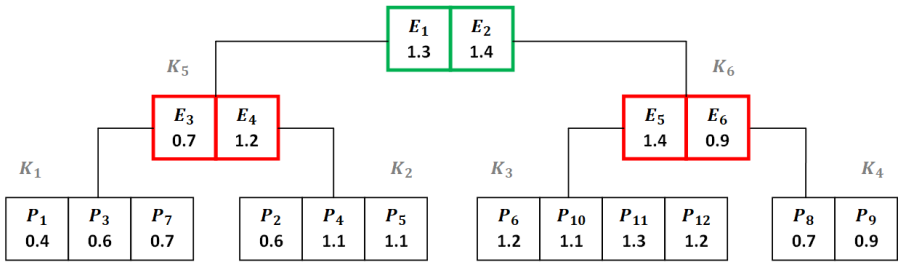


Abbildung 3.29: BRS am Beispiel einer monotonen Bewertungsfunktion mit R-Baum

Im ersten Schritt werden initial die beiden MBRs E_1 und E_2 in die Warteschlange geladen. Dann wird dem MBR E_2 gefolgt und dieses durch die beiden MBRs E_5 und E_6 des Knotens K_6 ersetzt. Man folgt E_5 auf die Blattebene und ersetzt E_5 durch die ihm zugeordneten Datenpunkte P_6, P_{10}, P_{11} und P_{12}. Da die obere Schranke von E_1 genauso groß ist wie die von P_{11}, muss man auch E_1 auf die nächste Ebene folgen und ersetzt das MBR durch seine beiden Kinder E_3 und E_4. Die im Anschluss durchgeführte Bestimmung des Datenobjektes mit der höchsten Priorität der Warteschlange setzt nur noch den Datenpunkt P_{11} auf die erste Position, so dass der gesuchte Top-1-Kandidat gefunden ist und der Algorithmus erfolgreich beendet werden kann.

Fazit

Wie für jedes Problem, so stellt sich aus mathematischer Sicht auch für das BRS-Verfahren zunächst einmal die Frage, ob und unter welchen Bedingungen obere oder untere Schranken für ein jeweiliges zu besuchendes MBR eines R-Baumes überhaupt existiert, bevor man eine Methode entwickeln wird, mit der sie gefunden werden kann. Wir werden uns im nun folgenden Kapitel im ersten Schritt mit der Frage beschäftigen, welche Eigenschaft eine Bewertungsfunktion grundsätzlich erfüllen muss, damit sie für das BRS-Verfahren geeignet ist. Im Anschluss werden wir dann prüfen, ob sich aus den gefundenen Eigenschaften auch die für das Verfahren notwendigen unteren oder oberen Schranken ableiten lassen.

Die Autoren aus [3] weisen darauf hin, dass das BRS-Verfahren grundsätzlich auch auf nicht-monotone Bewertungsfunktionen angewendet werden kann, sofern sich für jedes MBR H der Wert $f_{max}(H)$ bzw. $f_{min}(H)$ bestimmen lässt.

Der getMaxScore-Algorithmus aus [3] ist nur dann anwendbar, wie bereits oben gesagt, wenn die Monotonie aller Komponentenfunktionen gemäß Definition 3.4 für eine multivariate Funktion f gegeben ist. Ist das nicht der Fall, so lassen sich aus der Analysis bekannte Verfahren zur Bestimmung lokaler Extrempunkte (differenzierbarer) Funktionen anwenden, die in Abhängigkeit von f im Allgemeinen sehr aufwendig sein können. Wie eine solche Berechnung konkret aussieht, wird in Kapitel 4.2.4 vorgestellt.

Der nun folgende zweite Teil unserer Arbeit fokussiert sich auf das BRS-Verfahren und beantwortet die Frage, wie sich dieses Verfahren auch auf nicht-monotone Bewertungsfunktionen anwenden lässt.

4 Maximum- und Minumumprinzip spezieller Funktionsklassen

Sollen die Ergebnismenge einer Top-k-Anfrage aufsteigend sortiert werden, so ist die Bewertungsfunktion zu minimieren und man hat es mit einem Minimuproblem zu tun. Bei absteigender Sortierung entsprechend mit einem Maximumproblem. Im Kontext des BRS-Verfahrens haben wir die Situation, dass eine Bewertungsfunktion

$$f: H \to \mathbb{R}$$

auf einem (konvexen und kompakten) MBR H zu betrachten ist. Im Falle eines Minimumproblems verwendet das Verfahren als untere Schranke den kleinstmöglichen Wert, den die Funktion auf diesem MBR annehmen kann. Im Falle der Maximierung den größtmöglichen Wert. Wir werden nun für drei verschiedene Funktionsklassen untersuchen, ob sich aus den Funktionseigenschaften obere und/oder untere Schranken auf solchen MBRs bestimmen lassen. Dabei wird es je nach Funktionsklasse wichtig sein, dass man gemäß Definition 2.11 und Definition 3.1 zwischen achsenparallelen Hyperrechtecken, MBRs und allgemeinen Hyperrechtecken unterscheidet. Bei monotonen und den im Folgenden vorgestellten quadratischen Funktionen ist das Hyperrechteck immer achsenparallel. Bei quasi-konvexen Funktionen (Kap. 4.2) werden auch allgemeine Hyperrechtecke zugelassen.

4.1 Stetige Bewertungsfunktionen

In Kapitel 2 wurde der aus der Topologie zentrale Begriff der Kompaktheit eingeführt und das für diese Arbeit wichtige Beispiel des Hyperrechtecks als kompakter Teilraum des euklidischen Raumes vorgestellt. Desweiteren wurden zwei grundlegende Aussagen über reellwertige, stetige Funktionen eingeführt, nämlich der Satz von Heine, nach welchem stetige Funktionen auf kompakten Mengen bereits gleichmäßig stetig sind und der Satz vom Maximum und Minimum, der uns die Existenz einer oberen und einer unteren Schranke von stetigen Funktionen auf Hyperrechtecken garantiert.

Wir wollen nun untersuchen, ob sich konkrete, für das BRS-Verfahren verwendbare Schranken aus der Stetigkeit ableiten lassen. Es wird sich zeigen, dass zur Herleitung dieser Schranken die striktere Form der gleichmäßigen Stetigkeit erforderlich sein wird.

4.1.1 Schranken stetiger Funktionen auf Hyperrechtecken

Aufgrund des Satzes vom Minimum und Maximum (Satz 2.2) wissen wir zwar, dass jede stetige Funktion mit einem Hyperrechteck als Definitionsmenge ein Maximum und Minimum besitzt, allerdings liefert er keine konkreten Hinweise darüber, wie sich die Extrempunkte berechnen lassen. Da für die Anwendung des BRS-Verfahrens eine Schranke für jedes MBR gefordert ist, werden wir nun ein Theorem beweisen, mit dem sich explizite untere und obere Schranken für jede stetige Funktion auf einem Hyperrechteck angeben lassen.

Theorem 1 Schranken stetiger Funktionen auf Hyperrechtecken

Es sei $f: H \to \mathbb{R}$ eine stetige Bewertungsfunktion, $H \subset \mathbb{R}^n$ ein Hyperrechteck, $x_{max} \in H$ der Datenpunkt, auf dem f maximal, $x_{min} \in H$ der Datenpunkt, auf dem f minimal ist. Dann gelten die beiden folgenden Ungleichungen:

(1.1)
$$f(x_{max}) \leq f(M_H) + \frac{D(H)}{2\delta} \cdot \varepsilon,$$

(1.2)
$$f(x_{min}) \geq f(M_H) - \frac{D(H)}{2\delta} \cdot \varepsilon,$$

wobei M_H der Mittelpunkt von H und $D(H)$ sein Durchmesser ist. Statt δ_ε schreiben wir einfach δ.

Beweis: Da die Funktion f stetig auf H ist, ist sie wegen Satz 2.1 auch gleichmäßig stetig. Demnach gilt per Definition 2.12 das für ein gewähltes $\varepsilon > 0$ und für alle $x, y \in H$ ein $\delta > 0$ gefunden werden kann, so dass gilt

$$d_E(x, y) < \delta \Rightarrow d_E\big(f(x), f(y)\big) < \varepsilon.$$

Also gilt für den Mittelpunkt M_H des Hyperrechtecks H und für $n \in \mathbb{N}$, dass

$$d_E(M_H, y) \leq n\delta \Rightarrow d_E\big(f(M_H), f(y)\big) \leq n\varepsilon$$

für jedes $y \in H$. Wählt man speziell $n = \frac{D(H)}{2\delta}$ und $y = x_{max}$, so erhält man

$$d_E(M_H, x_{max}) \leq \frac{D(H)}{2} \Rightarrow d_E\big(f(M_H), f(x_{max})\big) \leq \frac{D(H)}{2\delta}\ \varepsilon$$

und demnach

$$f(x_{max}) \leq f(M_H) + \frac{D(H)}{2\delta}\ \varepsilon,$$

bzw.

$$f(x_{min}) \geq f(M_H) - \frac{D(H)}{2\delta}\ \varepsilon.\ \blacksquare$$

Um also eine obere oder untere Schranke für ein Hyperrechteck angeben zu können, benötigt man den Funktionswert des Mittelpunktes, seinen Durchmesser und das Zahlenpaar (ε, δ).

Es sei noch einmal angemerkt, dass nur mit der gleichmäßigen Stetigkeit sichergestellt ist, dass die angegebenen Schranken für *alle* Punkte des Hyperrechtecks gelten, da ansonsten (bei lediglich einfacher Stetigkeit) das angegebene δ mit jedem Punkt variieren kann.

Das nun folgende Rechenbeispiel zur Bestimmung eines δ zu vorgegebenem ε für lineare Funktionen lässt erahnen, warum die beiden Schranken (1.1) und (1.2) des obigen Theorems einen eher theoretischen Charakter besitzen. Denn die Berechnung muss für jede Funktion auf einem Hyperechteck individuell erfolgen und der Berechnungsaufwand kann in Anhängigkeit von der Komplexität der Funktion im Allgemeinen sehr aufwendig werden.

Beispiel 4.1

Wir berechnen die Schranken einer einfachen linearen Funktion. Gegeben sei also eine Abbildung der Form

$$f: \mathbb{R}^n \to \mathbb{R}; (x_1, \dots x_n) \mapsto \sum_{i=1}^{n} a_i x_i + b,$$

wobei $a_i, b \in \mathbb{R}$.

Es sei $x = (x_1, \dots x_n)$ ein fester, aber beliebiger Punkt des $\mathbb{R}^n$ und $\varepsilon > 0$ eine positive reelle Zahl. Für einen weiteren Punkt $y = (y_1, \dots y_n) \in \mathbb{R}^n$ gilt dann

$$|f(y) - f(x)| = \left| \sum_{i=1}^{n} a_i (y_i - x_i) \right| \leq \sum_{i=1}^{n} |a_i| \cdot |y_i - x_i| \leq \sum_{i=1}^{n} |a_i| \cdot \|y - x\|_\infty,$$

wobei $\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$ ist.

Wenn alle a_i verschwinden, ist der rechte Term gleich Null und damit auch kleiner als ε. Ansonsten ist er kleiner als ε, falls $\|y - x\|_\infty < \delta = \frac{\varepsilon}{\sum_{i=1}^{n} |a_i|}$ gilt, womit das gesuchte δ gefunden ist. Durch Einsetzen von ε und δ in die Ungleichungen (1.1) bzw. (1.2) des Theorems von oben erhält man die gesuchte untere und obere Schranke.

4.1.2 Bewertung

Die Definition der (gleichmäßigen) Stetigkeit und das einfache Beispiel machen deutlich, dass die beiden Schranken, obere wie untere, von den vier Parametern ε, δ, M_H und $D(H)$ abhängen und für jede stetige Bewertungsfunktion individuell zu bestimmen sind, womit die Motivation ihrer Anwendung für das BRS-Verfahren eher gering ist.

Das sehr einfache Beispiel einer linearen Bewertungsfunktion verdeutlicht den Aufwand, der zur Bestimmung einer geeigneten Schranke für allgemeine stetige Funktionen erforderlich ist. Es liegt die Vermutung nahe, dass die Berechnung solcher Schranken für komplexere, nicht-lineare Funktionen deutlich schwieriger sein wird. Mit zunehmender Komplexität der Funktion steht der Rechenaufwand dann in keinem Verhältnis mehr zu deren Anwendung. Man wird nach einer einheitlichen, für alle Funktionen der verwendeten Funktionsklassen anwendbaren Methode suchen.

Man wird daher gut beraten sein, nach weiteren Eigenschaften von Funktionen zu suchen, also „mehr“ als nur Stetigkeit zu verlangen, um die Funktionsklasse weiter einzugrenzen. Eigenschaften von Funktionen stehen oft im direkten Zusammenhang mit den Eigenschaften der zugrundeliegenden Definitionsmenge der Funktion. Im Kontext von Top-k-Anfragen unter der Verwendung von R-Bäumen haben wir es stets mit Hyperechtecken (MBRs) zu tun. Für die Entwicklung eines Verfahrens zur Berechnung von Schranken wäre es vorteilhaft, wenn man wüsste, in welchem Bereich der Definitionsmenge, also des Hyperechteckes, die gesuchten Punkte zu finden sind.

Dieser Gedankengang leitet über zu Funktionen, die ein sogenanntes Maximumprinzip erfüllen. Vereinfacht gesprochen, genügt eine Funktion genau dann einem *Maximumprinzip*, wenn sie ihr Maximum auf dem Rand ihres Definitionsbereiches annimmt. In Analogie ist der Begriff des *Minimumprinzips* definiert.

Wir werden in den beiden folgenden Kapiteln Funktionsklassen vorstellen, die ihr Maximum oder Minimum auf dem Rand eines Hyperechteckes annehmen, oder (im Falle der Minimierung) für die, wenn sie im Inneren des MBRs minimal sind, das Minimum gleich Null ist.

4.2 Quasi-konvexe Bewertungsfunktionen

Konvexen Mengen und konvexen Funktionen begegnet man in vielen Teilgebieten der Mathematik. Konvexe Mengen findet man in der Geometrie (siehe [43]), aber auch in der Analysis (siehe [44]) sind sie Bestandteil vieler Anwendungen. Konvexe Funktionen werden häufig in der Optimierungstheorie verwendet, da sich mit ihnen viele praktisch relevante Problemstellungen modellieren und angemessen bearbeiten lassen (siehe [45] und [46]). Um ein Maximumprinzip auf Hyperrechtecken herleiten zu können, benötigt man allerdings die allgemeinere Eigenschaft der Quasi-Konvexität.

Unsere Untersuchungen werden zeigen, dass quasi-konvexe Funktionen mit einem n-dimensonalen Hyperrechteck als Definitionsbereich die schöne Eigenschaft besitzen, auf seinen Ecken maximal zu sein. Ist also $f_{max}(H)$ das Maximum von f auf einem Hyperrechteck H, so gilt stets $f_{max}(H) = max\{f(e_1), \dots, f(e_{2^n})\}$, wobei die e_i die Ecken von H sind. Da H aus genau 2^n Eckpunkten besteht, lässt sich eine obere Schranke in endlich vielen Schritten für jedes MBR berechnen.

Umgekehrt werden wir zeigen, dass die Eigenschaft auf den Ecken eines jeden allgemeinen Hyperrechtecks maximal zu sein auch schon hinreichend für eine Funktion ist, quasi-konvex zu sein. Eine quasi-konvexe Funktion ist also durch diese Eigenschaft charakterisiert. Quasi-Konvexität verallgemeinert also das Maximumprinzip der Monotonie auf Hyperechtecken immer auf einer ausgezeichneten Ecke desselben maximal zu sein.

4.2.1 Motivation

Zunächst einmal wollen wir die praktische Relevanz dieser Funktionsklasse an einem Beispiel zeigen. Wir wählen dazu das Beispiel 2.1 der Immobiliendatenbank und suchen die drei besten Immobilien, deren Standort der nördliche Stadtrand von München (Nord / Ost oder Nord / West) ist und die sich möglichst nahe am Flughafen (Nord) von München befinden.

Wir setzen eine Würfelprojektion, also die Abbildung der Domänen beider Attribute auf das Einheitsintervall voraus, welche das Zentrum (48,13; 11,58) von München auf das Paar (0,5; 0) abbildet. Dass die Anwendung einer solchen Projektion unproblematisch ist setzt voraus, dass die Klasse der quasi-konvexen Funktionen invariant unter affinen Abbildungen ist. Wir werden diese Aussage direkt nach Einführung der dafür notwendigen Definition einer quasi-konvexen Funktion beweisen (siehe Lemma 3).

Die entsprechende Anfrage kann dann wie in Beispiel 4.2 geschrieben werden.

Beispiel 4.2:

SELECT $Top\ 3\ *$
FROM $Immobilien$
WHERE PLZ BETWEEN 80331 AND 81929
ORDER BY $(BG - 0.5)^2 + LG$ DESC

Die verwendete Bewertungsfunktion $f(x_1, x_2) = (x_1 - 0{,}5)^2 + x_2$ ist, wie sich sofort an ihrem Graphen bzw. der Höhenlinien erkennen lässt (siehe Abbildung 4.1) nicht monoton und wir haben es mit einem Maximumproblem zu tun.

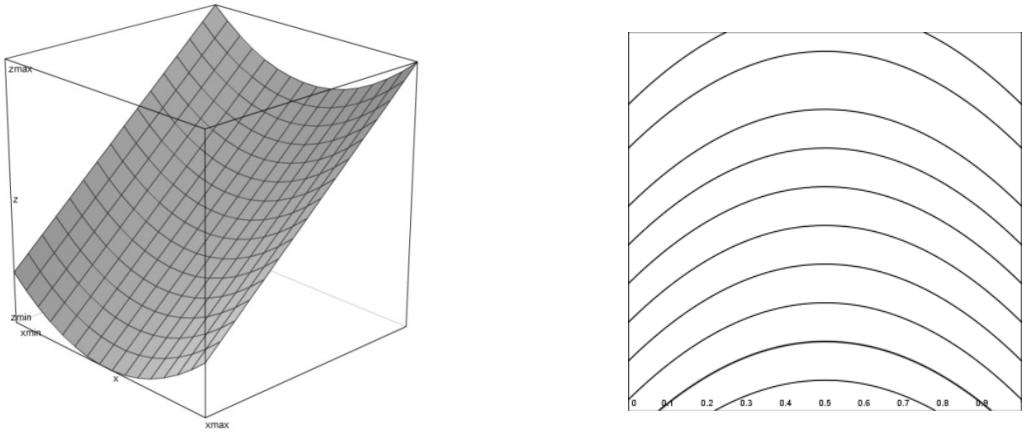


Abbildung 4.1: Graph und Höhenlinie der Funktion aus Beispiel 5.1

Andererseits lässt sich die gewünschte Anfrage auch nicht mit einer monotonen Bewertungsfunktion darstellen, da eine Abstandsmessung, d.h. die Verwendung einer Norm im Allgemeinen nicht monoton ist. Sie ist aber, wie wir im Folgenden sehen werden, wie so viele andere praktische Beispiele dieser Art auch, konvex und damit quasi-konvex.

Wir beginnen mit der Definition der in diesem Kapitel zu betrachtenden Funktionenklasse.

Definition 4.1 Konvexe und quasi-konvexe Funktionen

Eine reellwertige, multivariate Funktion $f: C \to \mathbb{R}$ auf einer konvexen Teilmenge $C \subseteq \mathbb{R}^n$ heißt *konvex*, wenn für je zwei Punkte $x, y \in C$ und alle $\lambda \in [0,1]$ gilt, dass

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

Sie heißt quasi-konvex, wenn für jedes $x, y \in C$ und alle $\lambda \in [0,1]$ gilt, dass

$$f(\lambda x + (1-\lambda)y) \leq max\{f(x), f(y)\}.$$

Alternativ lassen sich quasi-konvexe Funktionen auch dadurch definieren, dass man verlangt, dass all ihre Subniveaumengen (Definition 2.4) konvexe Mengen sind.

Geometrisch bedeutet Konvexität, dass die jeweilige Verbindungslinie zwischen $\big((x), f(x)\big)$ und $\big((y), f(y)\big)$ stets oberhalb des Graphen von f liegt, was für eine quasi-konvexe Funktion im Allgemeinen nicht zutrifft. Bei einer quasi-konvexen Funktionen kann der Graph der Funktion oberhalb der Verbindungslinie liegen, allerdings nicht so „hoch“, dass er den größten der beiden Werte $f(x)$ und $f(y)$ überschreitet. Abbildung 4.2 veranschaulicht den Sachverhalt.

Der Teil des Graphen der rechten Funktion, der oberhalb der rot gestrichelten Linie $y = f(y)$ liegt, verletzt die Eigenschaft der Quasi-Konvexität. So ist beispielweise der Funktionswert des rot dargestellten Punktes größer als $f(x)$ und $f(y)$.

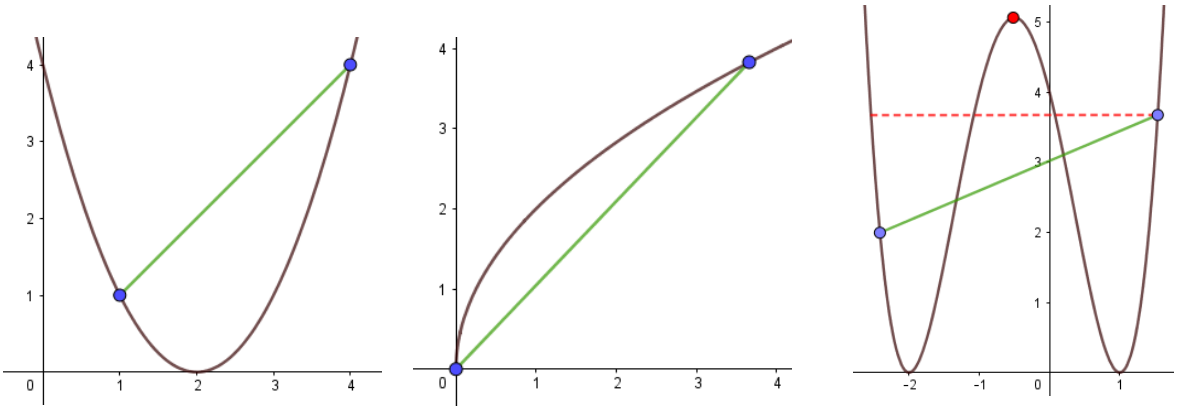


Abbildung 4.2: Konvex, quasi-konvex und nicht-konvex

Man überlegt sich leicht, dass jede konvexe Funktion auch quasi-konvex ist, die Umkehrung aber im Allgemeinen nicht gilt. So ist die Funktion $(x, y) \mapsto |x^2 - y^2|$ auf $[0,1]^2$ quasi-konvex, aber nicht konvex, während $(x, y) \mapsto |x - y|$ beide Eigenschaften erfüllt (siehe Abbildung 4.3).

Da wir im Folgenden unterscheiden müssen zwischen globalem und lokalem Maximum einer Funktion, sei der Vollständigkeit halber noch einmal die Unterscheidung beider Begriffe dargestellt.

Definition 4.2 Globale und lokale Extrempunkte

Sei $f: D \to \mathbb{R}$ eine multivariate Funktion, $D \subseteq \mathbb{R}^n$, dann heißt $x_0 \in D$ *globales* Maximum von f, falls $f(x_0) \geq f(x)$ für jedes $x \in D$. $x_0 \in D$ ist ein *lokales* Maximum, falls $f(x_0) \geq f(x)$ für jedes $x \in U$ einer Umgebung U von x_0 gilt. Analog definiert man globales und lokales Minimum.

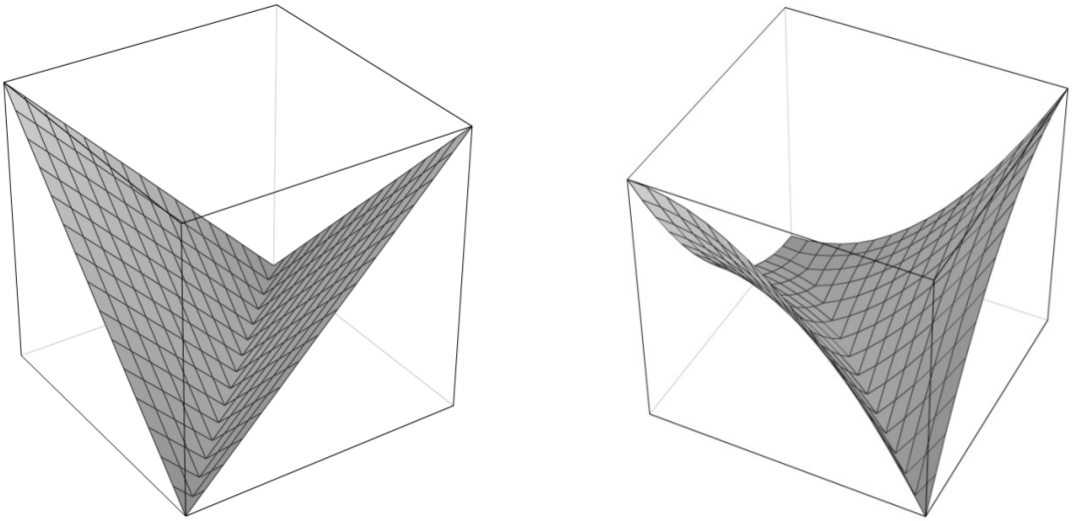


Abbildung 4.3: Konvexe Funktion (links) und Nicht-konvexe, aber quasi-konvexe Funktion (rechts)

Wir wollen nun zeigen, dass die Anwendung einer Würfelprojektion im Falle einer quasi-konvexen Bewertungsfunktion ein gegebenes Top-k-Anfrageproblem nicht verändert. Genauer:

Lemma 3 Quasi-konvexe Funktionen sind invariant unter affinen Abbildungen

Beweis: Wenn f eine quasi-konvexe Funktion ist, die die Datenpunkte des Datenraumes einer gegebenen Relation R in die reellen Zahlen abbildet und ϕ eine (beliebige) affine Transformation ist, so muss man zeigen, dass es eine quasi-konvexe Funktion $\bar{f}$ gibt, so dass gilt:

(A) $$f = \bar{f} \circ \phi.$$

Da jede Strecke mit den Endpunkten s_1 und s_2 durch eine affine Transformation ϕ aufgrund ihrer Eigenschaften (A1) bis (A3) aus Definition 2.14 wieder auf die Eckpunkte der Bildgeraden und das Innere wieder auf das Innere der Bildgeraden abgebildet wird, folgt die Behauptung. ■

Für die praktische Anwendung bedeutet dieses Lemma, dass wenn die Bewertungsfunktion f eine Problemstellung auf einem Datenraum einer Relation R modelliert und der Anwender diese Relation durch eine Würfelprojektion ϕ auf den Einheitswürfel projiziert, dass das Problem mit Bewertungsfunktion $\bar{f} = f \circ \phi^{-1}$ auf dem Einheitswürfel gelöst werden kann. Die Funktion $\bar{f}$ ist wegen Lemma 3 immer quasi-konvex, so dass sich das in diesem Kapitel entwickelten Verfahren auch auf $\bar{f}$ anwenden lässt.

4.2.2 Maximumprinzip quasi-konvexer Funktionen

Quasi-konvexe Funktionen müssen im Gegensatz zu konvexen Funktionen nicht automatisch stetig sein, wie schon das einfache Beispiel der Gaußklammer (Abrundungsfunktion) in einer Variablen $f: \mathbb{R} \to \mathbb{R}, x \mapsto \lfloor x \rfloor$ zeigt, die weder stetig noch konvex ist. Um die Existenz eines Maximums für quasi-konvexe Funktionen auf Hyperrechtecks beweisen zu können, ist allerdings die Kompaktheit des Definitionsbereiches, also des Hyperrechtecks, bereits ausreichend.

Wir werden nun zeigen, dass eine quasi-konvexe Funktion ein Maximumprinzip besitzt, wenn man ihren Definitionsbereich auf ein (allgemeines) Hyperrechteck beschränkt.

Theorem 2 Maximumprinzip quasi-konvexer Funktionen (Teil 1)

Es sei $f: D \to \mathbb{R}$ eine Bewertungsfunktion, $D \subset \mathbb{R}^n$ konvex, dann gilt:

Ist die Funktion quasi-konvex, dann ist sie auf mindestens einer der 2^n Ecken $e_1, \dots, e_{2^n}$ eines jeden allgemeinen Hyperechteckes $H \subset D$ maximal, d.h. es gilt:

$$f_{max}(H) = max\{f(e_1), \dots, f(e_{2^n})\}.$$

Man beachte, dass obige Aussage für jedes allgemeine Hyperechteck und damit insbesondere auch für jedes MBR gilt, was für die Anwendung dieser Funktionen für den BRS entscheidend ist.

Beweis: Wir führen den Beweis als Induktion über die Dimensionen.

Induktionsanfang $n = 2$:

Wir nehmen an, dass der maximale Punkt P im Inneren eines gegebenen Rechteckes liegt. Würde er auf dem Rand, oder gar auf einer Ecke liegen, sind wir fertig. Wir ziehen eine Verbindungslinie einer beliebigen Ecke, nennen wie sie A durch P, welche das Rechteck in einer Seite in B schneidet (siehe Abbildung 4.4, links). Da die Funktion f quasi-konvex ist, ist sie per Definition entweder auf A, oder auf B maximal. Wäre es A, so ist der Beweis geführt. Wir nehmen also an, es ist B. Da B auf einer Seite liegt (siehe Abbildung 4.4, rechts) nimmt f auf einer der Seitenenden, also einer Ecke ihr Maximum an.

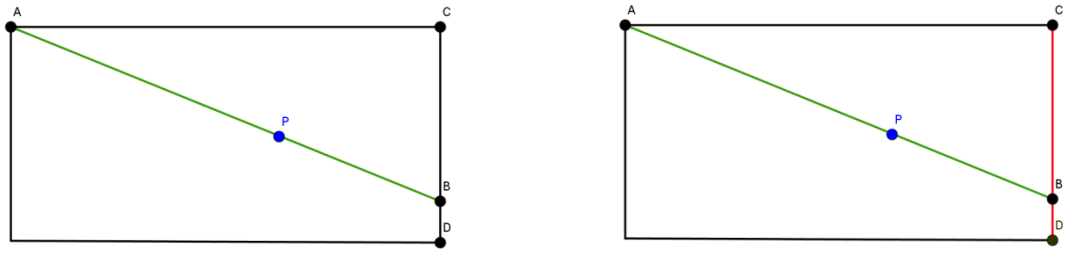


Abbildung 4.4: Maximum auf der Ecke eines Rechtecks

Induktionsschritt $(n-1) \to n$:

Wir nehmen an, dass die Aussage für $n-1$ stimmt, und müssen zeigen, dass sie dann für n gilt. Wir nehmen wieder an, dass das n-dimensionale Hyperrechteck im inneren Punkt M (wie oben) maximal ist. Wir ziehen wieder eine Verbindungslinie einer beliebigen Ecke A durch P, die das Hyperrechteck in einer seiner $2n$ $(n-1)$-dimensionalen Seitenflächen in B schneidet, für die die Induktionsannahme gilt. ■

Der Beweis zeigt deutlich, dass Abgeschlossenheit und Beschränktheit des Hyperrechteckes genügen, um das Maximumprinzip ableiten zu können. Die Stetigkeit der Funktion ist dazu nicht erforderlich. Dennoch wird man Stetigkeit der Funktion in der praktischen Anwendung voraussetzen wollen.

Quasi-konvexe Funktionen verallgemeinern das Maximumprinzip monotoner Funktionen auf MBRs in dem Sinne, dass sie auf (mindestens) einer beliebigen, nicht ausgezeichneten Ecke eines MBRs maximal sind.

Damit lässt sich der Pseudocode zur Ermittlung des Maximums einer quasi-konvexen Funktion gemäß Abbildung 4.5 darstellen.

```
    Algorithmus getMaxScoreQC (E_H = (e_1, ..., e_2^n), f)
    /* E_H ist die Menge der 2^n Ecken eines n-dimensionalen MBRs */
1   Return Max{f(e_1), ..., f(e_2^n)}
    End getMaxScoreQC
```

Abbildung 4.5: Pseudocode getMaxScoreQC für MBR

Bemerkenswert ist, dass die Eigenschaft auf den Ecken eines jeden allgemeinen Hyperrechtecks maximal zu sein, die Quasi-Konvexität bereits hinreichend beschreibt. Den Beweis dazu, also die Rückrichtung von Theorem 2 (Teil 1), wollen wir im nun folgenden Kapitel führen.

4.2.3 Charakterisierung quasi-konvexer Funktionen auf Hyperrechtecken

Theorem 2 (Teil 2)

Es sei $f: D \to \mathbb{R}$ eine Funktion, $D \subset \mathbb{R}^n$ eine konvexe Menge, dann gilt:

f ist quasi-konvex $\Leftrightarrow$ f ist auf mindestens einer der Ecken eines jeden allgemeinen Hyperrechteckes $H \subset D$ maximal.

Beweis: Mit Teil 1 von Theorem 2 wurde „⇒“ gezeigt. Es bleibt noch die Rückrichtung zu beweisen.

Man betrachte zwei beliebige Punkte x, y des Definitionsbereiches der Funktion und muss zeigen, dass $f(\lambda x + (1-\lambda)y) \leq max\{f(x), f(y)\}$ gilt, sie also auf den Endpunkten der Verbindungsstrecke maximal ist. Da f aber nach Voraussetzung auf den Ecken eines *jeden* allgemeinen Hyperrechteckes maximal ist, so insbesondere auch auf Hyperechtecken H der Dimension $n = 1$, also den Strecken. Damit ist f auch maximal auf demjenigen H, das durch x und y aufgespannt wird. ■

Damit ist die Klasse der quasi-konvexen Funktionen diejenige Funktionenklasse, die auf mindestens einer beliebigen Ecke eines jeden allgemeinen Hyperrechteckes maximal ist. Sie stellen damit die Verallgemeinerung der Klasse der monotonen Funktionen dar, die stets auf einer ausgezeichneten Ecke eines achsenparallelen Hyperechteckes maximal sind. Da eine quasi-konvexe Funktion im Allgemeinen weder monoton noch komponentenweise monoton sein muss (Siehe Abbildung 6.1), lässt sich der in [3] vorgestellte getMaxScore-Algorithmus zur Bestimmung der dominierenden Ecke nicht auf quasi-konvexe Funktionen anwenden. Somit muss man zur Bestimmung der maximalen Ecke tatsächlich alle 2^n Funktionswerte berechnen und den größten all dieser Werte auswählen.

4.2.4 Bewertung und Anwendung

Wie für alle Typen von Funktionen, so gilt auch für quasi-konvexe, dass man sie im Allgemeinen anhand ihrer Funktionsgleichung nicht direkt als quasi-konvex identifizieren kann. Umgekehrt zeigt der Versuch, ad-hoc eine Gleichung für eine quasi-konvexe Funktion anzugeben, wie schwierig die umgekehrte Richtung sein kann (siehe [47]). Die Charakterisierung quasi-konvexer Funktionen auf Hyperrechtecken bietet dem Anwender jedoch ein schönes und in vielen Fällen geeignetes Mittel, geometrisch „entscheiden“ zu können, ob er es mit einer quasi-konvexen Funktion zu tun hat, oder nicht. Die Höhenlinien des Graphen lassen erkennen, wo die Maxima der Funktion auf einem Hyperrechteck lokalisiert sind. Die „einfache“ Struktur der Niveaulinien der Funktion in Abbildung 4.1 lässt vermuten, dass die Bewertungsfunktion auf jedem Hyperrechteck ihr Maximum auf einer Ecke annimmt und also quasi-konvex sein muss. Es obliegt dem Anwender, diese Annahme aus mathematischer Sicht dann noch zu beweisen. Hierzu kann man wie folgt vorgehen:

Eine mathematische Bestimmung von Extremstellen kann, wie auch schon in [3] angedeutet, mit Hilfe der partiellen Ableitungen der Funktion erfolgen. Wir wollen abschließend ein aus der multivariaten Analysis bekanntes Verfahren zur Ermittlung von Extremwerten multivariater (stetig-differenzierbarer) Funktionen angeben. Für die Definition der verwendeten Begriffe, wie etwa den der partiellen Ableitung, sei auf einschlägige Literatur zum Thema verwiesen, wie z.B. [19] und [48].

Die Idee der partiellen Ableitung ist es, die Änderung des Funktionswertes zu untersuchen, wenn man eine Koordinate x_i variiert und alle anderen konstant lässt. Sei f eine multivariate Funktion in n Variablen, dann erhält man die (erste) *partielle Anleitung von f nach x_i* geschrieben als $\frac{\partial f}{\partial x_i}$, wenn man alle anderen Variablen als Konstante auffasst und f nach x_i nach den bekannten Regeln für Funktionen in einer Variable ableitet. Dadurch entstehen die partiellen Ableitungen erster Ordnung, die man als Vektor im sogenannten *Gradienten* von f zusammenfasst, also

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial n}\right).$$

Damit spielt der Gradient die gleiche Rolle, wie die „einfache“ erste Ableitung einer Funktion in einer Variablen. Notwendig für die Existenz eines lokalen wie globalen Extrempunktes x_0 (Maximum und Minimum) ist, dass $\nabla f(x_0) = 0$ ist, der Gradient also in diesem Punkt verschwindet. Ein solcher Punkt wird auch *stationärer* Punkt der Funktion bezeichnet.

Die hinreichende Bedingung, mit der entschieden wird, ob ein stationärer Punkt ein Maximum oder Minimum ist, erfolgt wieder in Analogie zum eindimensionalen Fall. Indem man nämlich die zweiten partiellen Ableitungen bildet, die dann in einer Matrix, der sogenannten *Hesse-Matrix* der Funktion zusammengefasst werden, also

$$H_f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1 x_1}(x) & \cdots & \frac{\partial f}{\partial x_1 x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n x_1}(x) & \cdots & \frac{\partial f}{\partial x_n x_n}(x) \end{pmatrix}.$$

Sie spielt die Rolle der zweiten Ableitung im Falle einer Funktion in einer Variablen, d.h. mit ihr wird die hinreichende Bedingung wie folgt ermittelt. Man bestimmt die einzelnen *Hauptminoren* der Matrix und bildet deren Determinante. Die Determinante der oberen k-ten Untermatrix mit $k \in \{1, \dots, n\}$ von $H_f(x)$ heißt *k-ter führender Hauptminor*, er ist also

$$M_k = det \begin{pmatrix} \frac{\partial f}{\partial x_1 x_1}(x) & \cdots & \frac{\partial f}{\partial x_1 x_k}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_k x_1}(x) & \cdots & \frac{\partial f}{\partial x_k x_k}(x) \end{pmatrix}.$$

Dann gelten die folgenden hinreichenden Bedingungen für die Existenz der Extremstellen:

Sei x_0 ein stationärer Punkt von f und M_k der k-te Hauptminor von $H_f(x_0)$, dann gilt:

i) Sind alle Hauptminoren $M_k > 0$, dann ist x_0 ein lokales Minimum von f.
ii) Wenn für alle Hauptminoren gilt, dass $(-1)^k M_k > 0$, dann ist x_0 ein lokales Maximum von f.

Fazit

Das oben dargestellte Verfahren zur Berechnung von Extremstellen einer multivariaten, stetig-differenzierbaren Funktion macht zwei Punkte deutlich. Erstens, es wird deutlich, wie aufwendig die Bestimmung der (lokalen) Extrema einer Funktion sein kann, sofern man keinerlei Anhaltspunkte darüber hat, im welchem Bereich der Definitionsmenge sie liegen könnten, man also nicht mehr über die Funktion weiß, außer dass sie stetig ist. Und zweitens wird klar, wie bedeutsam das in diesem Kapitel vorgestellte Ergebnis über quasi-konvexe Funktionen ist, für die man lediglich die Ecken eines MBRs betrachten muss, um das (lokale) Maximum der Funktion bestimmen zu können.

Effizienz des Algorithmus zur Bestimmung der maximalen Ecke

Der Algorithmus getMaxScoreQC (Abbildung 4.5) zur Bestimmung des maximalen Funktionswertes $f_{max}(H)$ von f auf einem n-dimensionalen MBR, also die Anweisung

$$Max\{f(e_1), \dots, f(e_{2^n})\},$$

benötigt 2^n Rechenschritte zur Bestimmung der oberen Schranke. Die Dimension n entspricht der Anzahl der Variablen der Bewertungsfunktion und kann, da wir Datenkomplexität und keine Abfragekomplexität betrachten wollen, als konstant bzw. klein $n \ll N$ angenommen werden. Damit ist getMaxScoreQC *linear* in der Anzahl der Punkte und damit effizient. Der Algorithmus ist ein Spezialfall zur Lösung eines parametrisierbaren Problems. Allgemein nennt man ein Problem *parametrisierbar*, (engl. fixed parameter tractable, kurz FPT), wenn ein Algorithmus existiert, der es mit einer Laufzeit von $f(k) \cdot p(n)$ löst, wobei f eine berechenbare Funktion, k der Parameter, p ein beliebiges Polynom und n die Eingabelänge ist. Der BRS+-Algorithmus (Abbildung 3.27) zusammen mit dem in dieser Arbeit entwickelten Algorithmus getMaxScoreQC ergeben eine effiziente Anfragebeantwortungsstrategie zur Lösung des Maximierungsproblem von Top-k-Anfragen mit

quasi-konvexen Bewertungsfunktionen. Für eine solche Funktion f, einen R-Baum $RTree$ und eine gewünschte Anzahl $k \in \mathbb{N}$ stellt sich das BRS-Verfahren wie folgt dar:

```
    Algorithm BRS+ (RTree, f, k)
 1  Liste PQ
 2  Lade Wurzel von RTree in die PQ
 3  While die top-k Elemente der PQ nicht nur Punkte sind Do
 4        For Each MBR H von PQ mit maximaler Schranke Do
 5            If H ist aus der letzten innerer Ebene Then
 6                For Each top-k Punkt von H.Punkte Do
 7                   Insert(Punkt, Punkt.f(Punkt), PQ)
 8                End For
 9            Else
10                For Each Kind von H.Kindknoten Do
11                   Kind.maxscore = getMaxScoreQC(Kind = (e_1, ..., e_2^n), f)
12                   Insert(Kind, Kind.maxscore, PQ)
13                End For
14            End If
15            Delete H
16        End For
17  WEnd
18  Return top-k Elemente der PQ
    End BRS+
```

Abbildung 4.6: Pseudocode des BRS+-Algorithmus für quasi-konvexe Funktionen

4.3 Quadratische Bewertungsfunktionen

Eine *Quadrik* im n-dimensionalen euklidischen Raum, auch *Hyperfläche zweiten Grades* genannt, ist die Nullstellenmenge $Q = \{(x_1, \dots, x_n) \in \mathbb{R}^n \mid f(x_1, \dots, x_n) = 0\}$ eines Polynoms f vom Grade 2. Quadriken lassen sich mit Hilfe sogenannter Koordinatentransformation auf eine einfache Form, die sogenannten *Normalform* reduzieren und dadurch klassifizieren. Zwei Quadriken sind äquivalent und gehören zur selben Äquivalenzklasse, wenn sie in die gleiche Normalform überführt werden können (siehe [49]).

Wir werden jetzt zeigen, dass zwei dieser Äquivalenzklassen im Kontext von Top-k-Anfragen besonders interessant sind, nämlich die der *elliptischen* und die der *hyperbolischen Paraboloide*. Dies soll anhand eines Beispiels aus der praktischen Anwendung demonstriert werden. Sie eignen sich zur Lösung von Minimumproblemen.

4.3.1 Motivation

Dem Beispiel 2.1 der Immobiliendatenbank folgend, betrachten wir einen Anwender, der nach einem Apartment sucht und eine genaue Vorstellung darüber hat, wie groß es sein soll und wieviel es kosten darf. Er sucht nach den drei passendsten Apartments in Berlin, mit einer Wohnfläche von 80 m² und einem Kaufpreis von 100 TEuro. Damit beide Attribute in ihren Ausprägungen nicht zu weit auseinander liegen, sei die Einheit im DBS für den Kaufpreis als TEuro angenommen.

Desweiteren sei angenommen, dass der Anfragepunkt $q = (80; 100) \in \mathbb{R}^2$ nach Anwendung einer Würfelprojektion der beiden Domänen auf den Punkt $q' = (0{,}4; 0{,}5) \in \mathbb{R}^2$ abgebildet worden sei.

Wir müssen auch hier in Analogie zum quasi-konvexen Fall zeigen, dass quadratische Funktionen invariant unter affinen Abbildungen sind, d.h. das die Projektion einer gegebenen Relation auf den Einheitswürfel die „ursprüngliche“ Problemstellung nicht verändert. Dieser Beweis erfolgt in Lemma 4 im direkten Anschluss an die Definition der quadratischen Funktionen.

Zur Ermittlung der drei Punkte des Datenraumes, die möglichst nahe bei q' liegen, kann die Top-k-Anfrage wie folgt formuliert werden:

Beispiel 4.3: Gleichgewichtete Anfrage

SELECT $Top\ 3$ *
FROM $Immobilien$
WHERE $Typ = 'Apartment'$ AND $Standort = 'Berlin'$
ORDER BY $(Wohnraum - 0{,}4)^2 + (Kaufpreis - 0{,}5)^2$ ASC

Die Anfrage so zu formulieren impliziert, dass dem Anwender die beiden Eigenschaften Wohnraum und Kaufpreis gleichermaßen wichtig sind. Sehr häufig möchte man, wenn die Situation es erfordert, ein Attribut einem anderen vorziehen, was sich durch eine entsprechende Gewichtung modellieren lässt. Eine Gewichtung kann im Allgemeinen auch negativ sein, so dass man zu einem Anfragepunkt $q = (q_1, q_2)$ und $\omega_1, \omega_2 \in \mathbb{R}$, sowie zwei beliebigen Attributen a_1, a_2 die folgende Anfrage erhält:

Beispiel 4.4: Anfrage mit möglicher unterschiedlicher Gewichtung

SELECT $Top\ k$ *
FROM R
[WHERE…]
ORDER BY $\omega_1(a_1 - q_1)^2 + \omega_2(a_2 - q_2)^2$ ASC

Für $\omega_1 = \omega_2 = 0{,}5$, erhält man wieder eine Gleichgewichtung wie in Beispiel 4.3.

Die Ergebnismenge wird absteigend sortiert, d.h. die Bewertungsfunktion muss minimiert werden. Die Anfrage von Beispiel 4.4 lässt vermuten, welche Bewertungsfunktionen wir in diesem Kapitel untersuchen wollen. Die Verallgemeinerung der Funktion dieses Beispiels auf n Attribute führt, wie bereits in der Einleitung beschrieben, zu der aus der Algebra bekannten Klasse von quadratischen Funktionen, die jetzt eingeführt werden.

Definition 4.3 Quadratische Funktion

Eine Abbildung $f: D \to \mathbb{R}, D \subseteq \mathbb{R}^n$ der Form

$$(x_1, \dots, x_n) \mapsto \sum_{i=1}^{n} \omega_i \, (x_i - q_i)^2$$

mit $\omega_i \in \mathbb{R}\backslash\{0\}, q_i \in \mathbb{R}$ für jedes $i \in \{1, \dots, n\}$ heißt (multivariate) *quadratische* Funktion.

Für ein festes i heißt die eindimensionale Funktion $x_i \mapsto \omega_i(x_i - q_i)^2$ die i-te Komponentenfunktion von f. Ist für mindestens ein $i \in \{1, \dots, n\}$ das zugehörige Gewicht $\omega_i < 0$, so ist der Graph der Funktion ein *hyperbolisches Paraboloid*. Sind alle $\omega_i > 0$, so ist er ein *elliptisches Paraboloid*.

Der n-dimensionale Datenpunkt $q = (q_1, \dots, q_n)$ wird meist als „Zentrum" oder auch als der „Mittelpunkt" des Graphen der Funktion bezeichnet. $\omega = (\omega_1, \dots, \omega_n)$ ist der Gewichtungsvektor.

Die folgende Abbildung zeigt den typischen Verlauf zweier Paraboloide im dreidimensionalen Raum. Links $f(x, y) = -x^2 + y^2$ und rechts $g(x, y) = x^2 + y^2$.

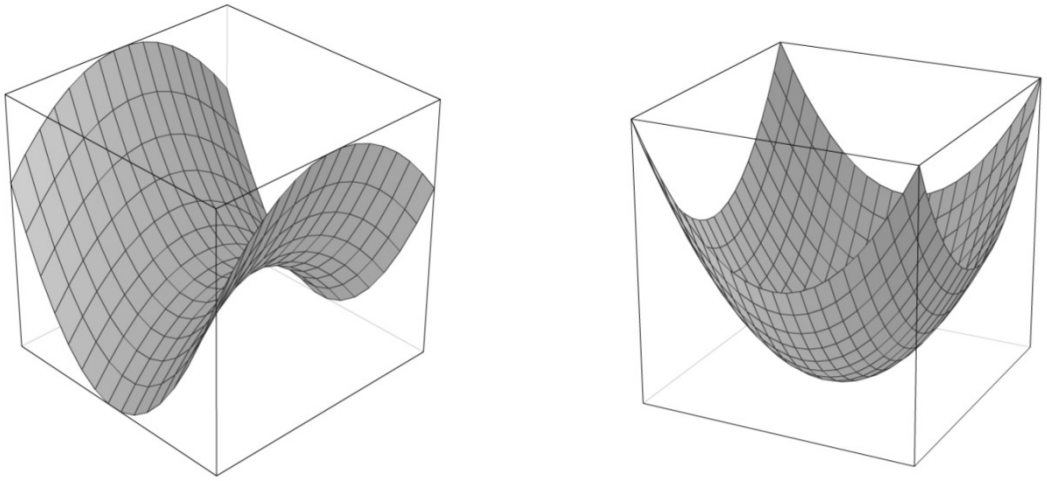


Abbildung 4.7: Hyperbolisches und elliptisches Paraboloid

Zum Abschluss dieses Kapitels beweisen wir in Analogie zum quasi-konvexen Fall, dass quadratische Funktionen invariant unter affinen Abbildungen sind. Das heißt, dass man auch für solche Funktionen den Einheitswürfel als Datenmenge voraussetzen darf.

Lemma 4 Quadratische Funktionen sind invariant unter affinen Abbildungen

Beweis: In Analogie zu Lemma 3 ist zu zeigen, dass für die ursprünglich gegebene quadratische Funktion f auch $\bar{f}$ mit $f = \bar{f} \circ \phi$ quadratisch ist. Es also eine quadratische Funktion $\bar{f}$ gibt, die nach Anwendung einer linearen Transformation ϕ das gleiche tut wie f, dass also gilt:

(A) $$f = \bar{f} \circ \phi.$$

Da im Gegensatz zu quasi-konvexen Funktionen eine quadratische Funktion explizit durch Angabe ihrer Funktionsgleichung gegeben ist, lässt sich auch die Funktionsgleichung für $\bar{f}$ konkret angeben. Eine Würfelprojektion ϕ von Beispiel 2.6 ist eine Transformation, die einen Punkt $x = (x_1, \dots, x_n)^T$ überführt in

$$\begin{pmatrix} \frac{1}{b_1 - a_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{b_n - a_n} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} \frac{1}{b_1 - a_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{b_n - a_n} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \frac{x_1 - a_1}{b_1 - a_1} \\ \vdots \\ \frac{x_n - a_n}{b_n - a_n} \end{pmatrix}$$

Die Darstellung zeigt, dass sich die Transformation jeder Komponenten $x_j \in [a_j, b_j]$ direkt aus den Intervallgrenzen angeben lässt, so dass man den Beweis koordinatenweise führen kann. Es sei also $f_j(x_j) = \omega_j(x_j - q_j)^2$ mit $x_j \in I_j = [a_i, b_i]$ die j-te Komponentenfunktion einer quadratischen Funktion.

Die affine Abbildung (Würfelprojektion) ist in der j-ten Komponente gegeben durch $\phi_j(x) = \frac{x_j - a_j}{b_j - a_j}$.
Wir müssen zeigen, dass mit f_j auch die Funktion $\overline{f_j}$, mit $\overline{f_j} \circ \phi_j = f_j$ eine quadratische Funktion ist, also $\overline{f_j}$ sich schreiben lässt als:

(B)
$$\overline{f_j} = \overline{\omega_j}(x_j - \overline{q_j})^2.$$

Der Beweis erfolgt durch direkten Vergleich der Komponenten:

$$\overline{f_j}(x') = f_j(x)$$
$$\Leftrightarrow \overline{\omega_j}\left(\frac{x_j - a_j}{b_j - a_j} - \overline{q_j}\right)^2 = \omega_j(x_j - q_j)^2$$

Wir wählen $\overline{q_j} = \phi_j(q_j) = \frac{q_j - a_j}{b_j - a_j}$, also das Bild der affinen Transformation und erhalten weiter:

$$\Leftrightarrow \overline{\omega_j}\left(\frac{x_j - a_j}{b_j - a_j} - \frac{q_j - a_j}{b_j - a_j}\right)^2 = \omega_j(x_j - q_j)^2$$

$$\Leftrightarrow \overline{\omega_j}\left(\frac{x_j - q_j}{b_j - a_j}\right)^2 = \omega_j(x_j - q_j)^2$$

$$\Leftrightarrow \frac{\overline{\omega_j}}{(b_j - a_j)^2}(x_j - q_j)^2 = \omega_j(x_j - q_j)^2$$

(C)
$$\Leftrightarrow \overline{\omega_j} = \omega_j(b_j - q_j)^2 \blacksquare$$

Das bedeutet, dass sich der gesuchte Anfragepunkt und die Gewichtung koordinatenweise, also aus den jeweilig zugehörigen Intervallgrenzen berechnen lassen, indem man den Anfragepunkt linear transformiert und die Gewichtung gemäß (C) bestimmt.

Aus der quadratischen Funktion $f: D \to \mathbb{R}$ mit

$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} \omega_i (x_i - q_i)^2,$$

und $D = I_1 \times \ldots \times I_n, I_j = [a_j, b_j]$ als Hyperrechteck, wobei a_j die kleinste und b_j die größte Ausprägungen des Attributes A_j der Relation R ist, erhält man dann die Abbildung $\bar{f}: I^n \to \mathbb{R}$

$$(x_1', \dots, x_n') \mapsto \sum_{i=1}^{n} \omega_i (b_i - q_i)^2 \left(x_i' - \frac{q_j - a_j}{b_j - a_j} \right)^2 .$$

4.3.2 Einordnung in den Kontext von Top-k-Anfragen

Wir wollen im Folgenden kurz erläutern, mit welchen Verfahren Top-k-Anfragen mit quadratischen Bewertungsfunktionen gelöst werden können und werden zusätzlich einige Gründe dafür angeben, warum die Lösung des Minimierungsproblems für nicht-monotone Funktionen seine Daseinsberechtigung besitzt und man es nicht auf ein Maximierungsproblem reduzieren kann.

Top-k-Anfragen mit quadratischen Funktionen sind nicht monoton

Wie im Falle der quasi-konvexen Funktionen, so ist auch die Bewertungsfunktion einer Top-k-Anfrage mit einer quadratischen Funktion nicht eindeutig durch das gegebene Problem bestimmt. So kann man statt der in Beispiel 4.4 angegebenen Funktion auch die Funktion $f(x) = \sum_{i=1}^{2} \omega_i |x_i - q_i|$ verwenden. Da man es aber immer mit Abstandsfunktionen zu tun hat, kann das Problem nicht mit monotonen Funktionen dargestellt und gelöst werden (siehe [50] und [51]).

f und seine Negation -f gehören im nicht-monotonen Fall zu unterschiedlichen Funktionsklassen

Werden für eine gegebene Funktion f die Datenpunkte mit den k größten Funktionswerten bestimmt, so lassen sich umgekehrt die k kleinsten Punkte dieser Funktion durch Maximierung der negativen Funktion -f ermitteln. Durch einen Vorzeichenwechsel kann man grundsätzlich ein Minimierungsproblem in ein Maximierungsproblem überführen.

Da für jede monotone Funktion auch ihre Negation monoton ist, stellt sich für die Klasse monotoner Funktionen die Minimierungsproblematik nicht, denn jedes monotone Verfahren kann, sofern keine weiteren Bedingungen gestellt werden, sowohl auf f als auch auf -f angewendet werden.

Für beliebige nicht-monotone Funktionen gilt das nicht. So ist beispielweise die Negation einer konvexen Funktion konkav und damit das in Kapitel 4.2 vorgestellte Verfahren zur Lösung des Maximierungsproblem für eine (quasi-)konvexe Funktion f nicht auch auf -f anwendbar, da das Verfahren explizit die Eigenschaften der Quasi-Konvexität verwendet. Zur Ermittlung der top-k kleinsten Objekte nicht-monotoner Bewertungsunktionen müssen Verfahren entwickelt werden, welche die Eigenschaften der Funktionsklasse und der ihr zugehörigen Funktionen berücksichtigen.

Betrachten wir eine k-Nächste-Nachbarn- oder eine Top-k-Anfrage?

Sind alle Gewichte positiv, so entspricht die quadratische Funktion einer Metrik. Das bedeutet, dass die entsprechende Top-k-Anfrage in diesem Fall eine spezielle k-NN-Anfragen ist, da sich k-NN-Anfragen auch als Top-k-Anfragen mit einer Metrik als Bewertungsfunktion interpretieren lassen (siehe [52]). Im „elliptischen Fall“ kann das Anfrageproblem als k-NN-Anfrage, oder mit dem BRS+-Algorithmus gelöst werden.

Da wir in dieser Arbeit aber allgemein auch negative Gewichtungen der Komponentenfunktionen quadratischer Funktionen zulassen wollen, so ist außer dem BRS-Verfahren kein anderes der hier vorgestellten Verfahren zur Beantwortung einer Top-k-Anfrage mit quadratischen Funktionen und negativer Gewichtung verwendbar. Wir werden jetzt zeigen, wie sich die für den R-Baum und das BRS-Verfahren erforderlichen unteren Schranken einer quadratischen Funktion bestimmen lassen.

4.3.3 Minimumprinzip quadratischer Funktionen

Der Satz vom Minimum und Maximum (Satz 2.2) sichert uns die Existenz eines Minimums einer stetigen Funktion auf einer kompakten Menge. Quadratische Funktionen sind im Gegensatz zu quasi-konvexen immer stetig, so dass die Existenz des Minimums, sofern der Definitonsbereich kompakt ist, gegeben ist.

Proposition 4.1 Stetigkeit quadratischer Funktionen

Jede quadratische Funktion

$$f: \mathbb{R}^n \to \mathbb{R};\ (x_1, \dots, x_n) \mapsto \sum_{i=1}^{n} \omega_i\,(x_i - q_i)^2$$

ist stetig auf ganz $\mathbb{R}^n$.

Beweis: Aus der Analysis wissen wir, dass Funktionen in einer Variablen der Form $x_i \mapsto \omega_i(x_i - q_i)^2$ stetig sind. Da unter Addition solcher Funktionen die Stetigkeit erhalten bleibt, ist eine n-dimensionale quadratische Funktion ebenfalls stetig. ■

Quadratische Funktionen sind konvex und damit auch quasi-konvex. Leider lässt sich aus der Quasi-Konvexität kein zu Theorem 2 analoges Minimumprinzip auf Hyperrechtecken ableiten. Dennoch erlauben die speziellen Eigenschaften quadratischer Funktionen, wie wir nun zeigen werden, die explizite Berechnung eines Minimums auf Hyperrechtecken.

Theorem 3 Minimumprinzip quadratischer Funktionen auf Hyperechtecken

Es sei $f: H \to \mathbb{R}$ quadratisch, H ein achsenparalleles Hyperrechteck und $x_{min} \in H$ ein Punkt auf dem f minimal auf H ist.

Dann liegt x_{min} auf dem Rand des Hyperrechteckes, oder $f(x_{min}) = 0$.

Beweis: Jedes Hyperechteck ist eine kompakte Teilmenge des $\mathbb{R}^n$, so dass zusammen mit Satz 2.2 aus Kapitel 2 die Existenz des Minimums von f auf $H = I_1 \times I_2 \times \dots \times I_n$ gegeben ist. Des Weiteren ist aus der Theorie multivariater Funktionen bekannt, dass wenn $f(x) = \sum_{i=1}^{n} f_i(x)$ die Summe aus n Komponentenfunktionen ist, sich ihr Minimum aus der Summe der Einzelkomponenten ermitteln lässt. Genauer gesagt, ist die folgende Gleichung erfüllt:

$$\min_{(x_1,\dots,x_n)\in H} \sum_{i=1}^{n} \omega_i\,(x_i - q_i)^2 = \sum_{i=1}^{n} \min_{x_i \in I_i} (\omega_i (x_i - q_i)^2)\,.$$

Damit lässt sich die Berechnung des Minimums einer quadratischen Funktion auf einem Hyperrechteck stets auf den eindimensionalen Fall, nämlich seine Komponentenfunktionen reduzieren. Im $\mathbb{R}^1$ lassen sich, wie in Abbildung 4.8 dargestellt, genau zwei Typen solcher Komponentenfunktionen unterscheiden.

Ist die Gewichtung *positiv* (Abbildung 4.8, linke Seite), so ist das Minimum Null, wenn das Intervall das globale Minimum der Funktion enthält (siehe I_2). Ist dieses nicht im Intervall enthalten, wie im Falle I_1 oder I_3, so liegt das Minimum entweder auf der linken, oder der rechten Intervallgrenze.

Ist die Gewichtung *negativ* (Abbildung 4.8, rechte Seite), so entfällt eine solche Fallunterscheidung. In diesem Fall liegt das Minimum entweder auf der linken, oder der rechten Intervallgrenze (I_1 oder I_2).

Wenn jede Komponentenfunktion f_i der quadratischen Funktion f im Inneren des Intervalls I_i minimal ist, dann ist auch f im Inneren des Hyperrechteckes $H = I_1 \times \ldots \times I_n$ minimal und der Funktionswert ist Null. Wenn jede Komponentenfunktion auf einem Endpunkt des Intervalls I_i minimal ist, dann ist auch f auf einer Ecke von H minimal. In allen anderen Fällen ist f auf einer der $(n-1)$-dimensionalen Seitenflächen des Hyperrechteckes minimal. ■

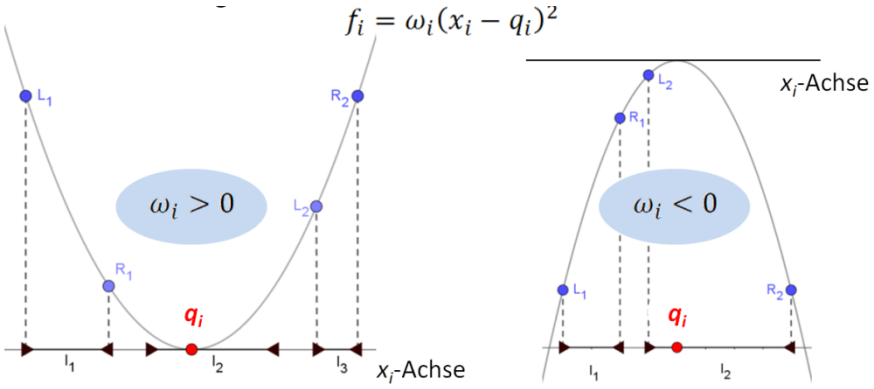


Abbildung 4.8: Eindimensionale Paraboloide

Zu beachten ist, dass für den Beweis die Stetigkeit der Funktion implizit vorausgesetzt und verwendet wird. Bei Nicht-Stetigkeit der Funktion ist es möglich, dass sie an den Intervallenden (Abbildung 4.8) und damit an möglichen minimalen Punkten Sprungstellen besitzt und somit dort kein Minimum existiert.

Damit lässt sich der Pseudocode zur Ermittlung des Minimums einer eindimensionalen quadratischen Funktion gemäß Abbildung 4.9 darstellen. getMin ist der eindimensionale Fall, der bei Übergabe der Gewichtung ω und dem Parameter q die Parabel erzeugt und das gesuchte Minimum auf dem übergebenen Intervall $[a, b]$ gemäß oben dargestellter Fallunterscheidung bestimmt. Die Berechnung des Minimums einer n-dimensionalen quadratischen Funktion wird dann durch getMinScoreQ bestimmt.

```
    Algorithmus getMin (a, b, ω, q)
    /* a untere, b obere Intervallgrenze, ω ∈ ℝ\{0}, q ∈ ℝ */
1   f = ω(x − q)²
2   If ω < 0 Then
3         Min = Min{f(a), f(b)}
4   Else /* wenn Gewichtung positiv */
5         If q ∉ [a, b] Then
6               Min = Min{f(a), f(b)}
7         Else /* wenn q ∈ [a, b] */
8               Min = f(q) = 0
9         End If
10  End If
11  Return Min
    End getMin
```

```
    Algorithmus getMinScoreQ ((a_1, b_1, …, a_n, b_n), (ω_1, …, ω_n), (q_1, …, q_n), getMin)
1   Return Σ_{i=1}^{n} getMin(a_i, b_i, ω_i, q_i)
    End getMinScoreQ
```

Abbildung 4.9: getMinScore-Algorithmus

Es lassen sich, wie oben beschrieben, drei verschiedene „Minimumtypen“ quadratischer Funktionen unterscheiden. Solche die innerhalb eines Hyperrechteckes liegen, solche die auf dem Rand, aber nicht auf einer Ecke liegen und zuletzt diejenigen, die auf einer Ecke liegen. Quadratische Funktionen erfüllen also ein Minimumprinzip auf achsenparallelen Hyperrechtecken. Einzige Ausnahme: Das Minimum liegt im Inneren und ist Null. Abbildung 4.10 veranschaulicht die verschiedenen Minima durch Darstellung einiger Niveaulinien am Beispiel der Funktion $f(x_1, x_2) = (x_1 - 0{,}4)^2 + (x_2 - 0{,}4)^2$ auf dem Einheitsrechteck mit einer regelmäßigen Gitterpartition. Das globale Minimum von f auf dem roten Rechteck liegt auf dem roten Punkt, so dass die Funktion dort das Minimum Null annimmt. Auf den blauen Rechtecken ist die Funktion auf den jeweiligen zugehörigen blauen Punkten, also auf den Kanten minimal. Die übrigen schwarz dargestellten Punkte entsprechen den Minimalstellen der übrigen Rechtecke und liegen auf einer der Ecken derselben.

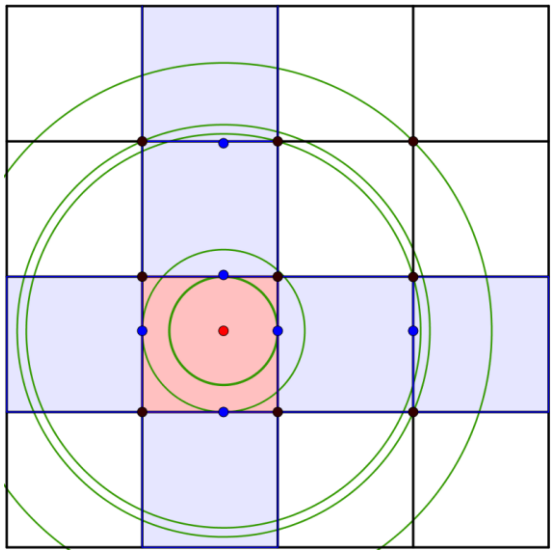


Abbildung 4.10: Minimalstellen einer quadratischen Funktion in einem regelmäßigen Gitter

Fazit

Auch die Bestimmung des Minimums einer Funktion auf einem MBR erfordert, wenn man nicht mehr über sie weiß als das sie stetig ist, dass man aufwendige Verfahren wie etwa das zur Bestimmung der Hesse-Matrix und ihrer Minoren anwenden muss. Hinzu kommt, dass im Gegensatz zum Maximumprinzip, welches in der Analysis für verschiedene Funktionsklassen bekannt ist, es im Allgemeinen für diese Klassen kein entsprechendes Minimumprinzip gibt. Mit Stetigkeit alleine lässt sich also das Minimierungsproblem einer Bewertungsfunktion nicht effizient lösen. Zu dieser Erkenntnis haben auch unsere Untersuchungen für (quasi-)konvexe Funktionen geführt. Um dennoch eine Minimumprinzip herleiten zu können, haben wir uns auf die spezielle Subklasse der quadratischen Funktionen beschränkt. Wie im quasi-konvexen Fall lässt sich auch für quadratische Funktionen das Problem der Bestimmung von Schranken auf MBRs effizient lösen. Die quadratischen Funktionen sind, soweit uns bekannt ist, die erste Funktionsklasse nicht-monotoner Bewertungsfunktionen, für die ein konkretes Verfahren zur Bestimmung unterer Schranken für das BRS-Verfahren entwickelt worden ist.

Effizienz des Algorithmus

Um die untere Schranke auf einem n-dimensionalen MBR $H = [a_1, b_1] \times \ldots \times [a_n, b_n]$ zu berechnen, benötigt der Algorithmus getMinScore (ohne Fallunterscheidungen) genau n Berechnungsschritte, indem er, wie oben beschrieben, auf jedem Intervall $I_i = [a_i, b_i]$ für $i = 1, \ldots, n$ das Minimum bestimmt und alle so ermittelten i Werte aufaddiert. Damit ist er linear und effizient. Das BRS-Verfahren unter Anwendung des getMinScore-Algorithmus löst in effizienter Weise das Minimierungsproblem von Top-k-Anfragen mit quadratischen Bewertungsfunktionen. Für eine quadratische Funktion f, einen R-Baum $RTree$ und eine gewünschte Anzahl $k \in \mathbb{N}$ erhält man das in der folgenden Ergebnis (Abbildung 4.11):

```
    Algorithm BRS+ (RTree, f, k)
 1  Liste PQ
 2  Lade Wurzel von RTree in die PQ
 3  While die top-k Elemente der PQ nicht nur Punkte sind Do
 4     For Each MBR H von PQ mit minimaler Schranke Do
 5        If H ist aus der letzten innerer Ebene Then
 6           For Each top-k Punkt von H.Punkte Do
 7                 Insert(Punkt, Punkt.f(Punkt), PQ)
 8           End For
 9        Else
10           For Each Kind von H.Kindknoten Do
11              Kind.minscore =getMinScoreQ ((a1, b1, …, an, bn), (ω1, …, ωn), (q1, …, qn), getMin)
12              Insert(Kind, Kind.minscore, PQ)
13           End For
14        End If
15        Delete H
16     End For
17  WEnd
18  Return top-k Elemente der PQ
    End BRS+
```

Abbildung 4.11: Pseudocode des BRS+-Algorithmus für quadratische Funktionen

5 BRS für spezielle Klassen von Bewertungsfunktionen

Im nun folgenden Kapitel wird eine konkrete Implementierung des BRS-Verfahrens für quasi-konvexe und quadratische Funktionen in einem relationalen Datenbankmanagementsystem (RDBMS), dem SQL-Server angegeben. Alle zur Evaluierung erforderlichen Objekte, wie Tabellen, Prozeduren und Funktionen, sind mit Transact-SQL (T-SQL), einer Erweiterung des SQL-Standards von Sybase und Microsoft entwickelt (siehe [53]), können aber mit wenigen Anpassungen auf jedes andere RDBMS portiert werden.

Die in diesem Kapitel aufgeführten Beispiele und Funktionen sind der besseren Anschauung wegen zweidimensional. Sie lassen sich aber ohne Weiteres auf höhere Dimensionen verallgemeinern.

Zunächst wollen wir in Abhängigkeit einer gegebenen Verteilung der Datenunkte im Raum, drei verschiedene Verfahren angeben, die eine Gleichverteilung der Datenpunkte in den MBRs anstreben. Eine solche Gleichverteilung hat den Vorteil, dass hinreichend viele Top-k-Kandidaten in den MBRs enthalten sind, was für die Laufzeit des Verfahrens vorteilhaft sein wird.

5.1 Konstruktion von R-Bäumen

Die Konstruktion von Hyperrechtecken (MBRs) im Kontext von R-Bäumen hat ihren Ursprung (siehe Kapitel 3.1) aus dem Bereich der Geodaten und ist naturgemäß allein durch die Lage der Geoobjekte im Raum nicht eindeutig bestimmt. So gibt es auch zur Beantwortung von Top-k-Anfragen mit R-Bäumen ebenfalls keine eindeutige „Lösung“ zur Konstruktion dieser MBRs. Als Grundlage für den in [3] vorgestellten BRS-Algorithmus (vgl. Kapitel 3.4) wurde ein beliebiger R-Baum ohne spezielle Angabe seiner Konstruktion vorausgesetzt.

Ein wesentlicher Vorteil der Anwendung eines R-Baumes für Top-k-Anfragen ist die Vermeidung von Suchen in leeren Datenraumpartitionen. Um möglichst wenig Leerraum zu indizieren, sollte jedes MBR eine „hinreichend“ große Menge an Datenpunkten einschließen. Wir werden im Folgenden drei Partitionsverfahren angeben, die dieser Forderung in Abhängigkeit von der gegebenen Verteilung der Datenpunkte im Raum gerecht werden.

Konstruktion 1: Organisation der Datenpunkte durch Gleichverteilung

Möchte man bei einer unregelmäßig vorgegebenen verteilten Punktemenge im Raum (Datenraum mit Punktkonzentrationen), durch Organisation der Punkte eine Gleichverteilung in den MBRs erreichen, so kann man das Median-Split-Verfahren auf die jeweiligen Punktkoordinaten anwenden (siehe [54] und [55]), wie es auch beim k-d-B-Baum Verwendung findet.

Da der Median von Punkten in der Literatur nicht eindeutig festgelegt ist, wollen wir die folgende Definition für das weitere Vorgehen verwenden.

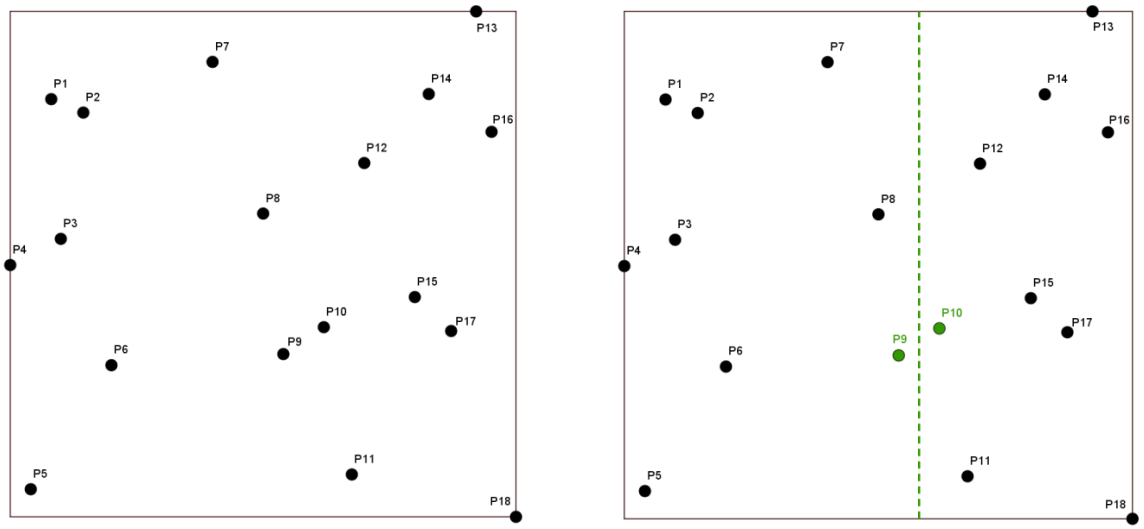
Definition 5.1 Median Split

Der Median teilt eine endliche Liste reeller Zahlen in zwei Teilmengen und soll wie folgt bestimmt werden:

Die Werte der Liste werden aufsteigend sortiert. Ist ihre Anzahl ungerade, also $2n + 1, n \in \mathbb{N}$, und $a_1 \leq a_2 \leq \cdots \leq a_{2n+1}$, so ist die mittlere Zahl a_{n+1} der Median. Ist sie gerade, und $a_1 \leq a_2 \leq \cdots \leq a_{2n}$, so ist der Median das arithmetische Mittel $\frac{a_{n+1}-a_n}{2}$ der beiden mittleren Zahlen a_n, a_{n+1}, die man auch Unter- und Obermedian nennt. Im letzten Fall erhält man die beiden Teillisten $\{a_1, \ldots, a_n\}$ und $\{a_{n+1}, \ldots, a_{2n}\}$, im ersteren soll der Median zur Teilliste der kleineren Werte gehören. Man erhält demnach die Einteilung in $\{a_1, \ldots, a_{n+1}\}$ und $\{a_{n+2}, \ldots, a_{2n+1}\}$.

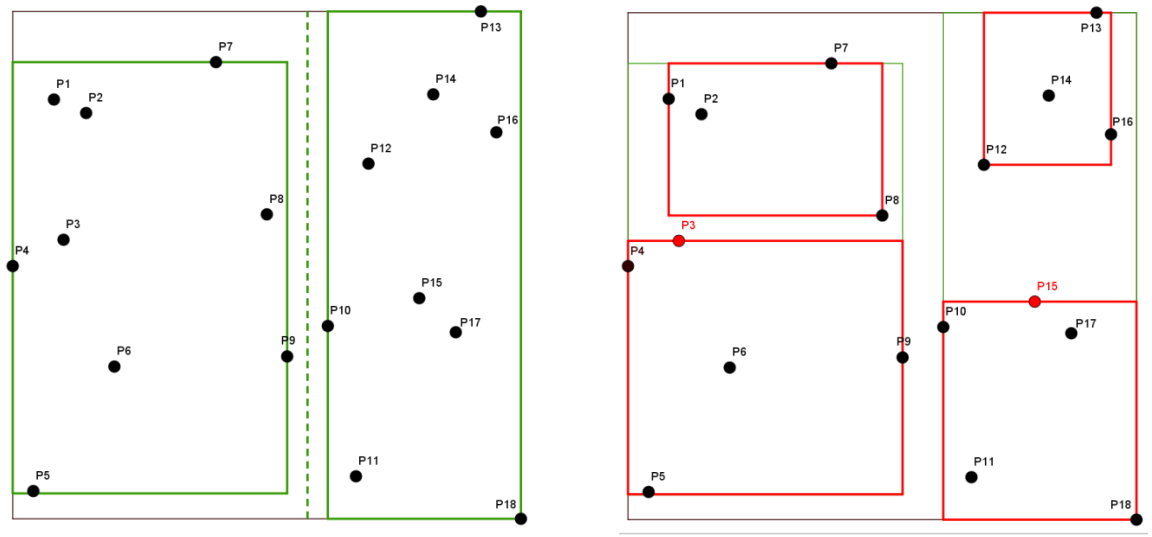
Zu beachten ist, dass für diese Definition eine Multimenge und keine einfache Menge verwendet wird, da man im Allgemeinen zulassen muss, dass Werte mehrfach vorkommen. Ein entsprechender Algorithmus ist in [56] auf Seite 263 angegeben.

Das Verfahren teilt zuerst die Punktmenge in einer Koordinate und bildet im Anschluss die jeweiligen MBRs der entstandenen Mengen. Es bricht ab, sobald eine zuvor festgesetzte Mindestanzahl von Punkten je MBR erreicht ist. Abbildung 5.1 und Abbildung 5.2 veranschaulichen das Verfahren für 16 gegebene Punkte und einer Mindestanzahl von vier Punkten je MBR. Die farbig markierten Datenpunkte zeigen die für den jeweiligen Teilschritt relevante Mediane an.



Erzeugen des MBRs des gesamten Raumes | Median Split durch P_9 und P_{10}

Abbildung 5.1: Binärer R-Baum mit disjunkten MBRs



Erzeugen der ersten beiden MBRs | Median Split und Bildung der MBRs

Abbildung 5.2: Binärer R-Baum mit disjunkten MBRs

Offensichtlich führt diese Form der Aufteilung genau dann dazu, dass sich die Kardinalität der MBRs auf unterster innerer Nicht-Blattebene paarweise um nur einen Punkt unterscheiden, wenn die Koordinaten aller Datenpunkte komponentenweise ebenfalls paarweise verschieden sind, also für zwei Datenpunkte mit $P_1 = (x_1, \dots x_n)$ und $P_2 = (y_1, \dots y_n)$ gilt, dass $x_i \neq y_i$ für $i = 1, \dots, n$. Genau dieser Fall ist in Abbildung 5.2 dargestellt. Als Ergebnis erhält man einen binären R-Baum mit einer nahezu gleichen Anzahl von Datenpunkten in den jeweiligen MBRs.

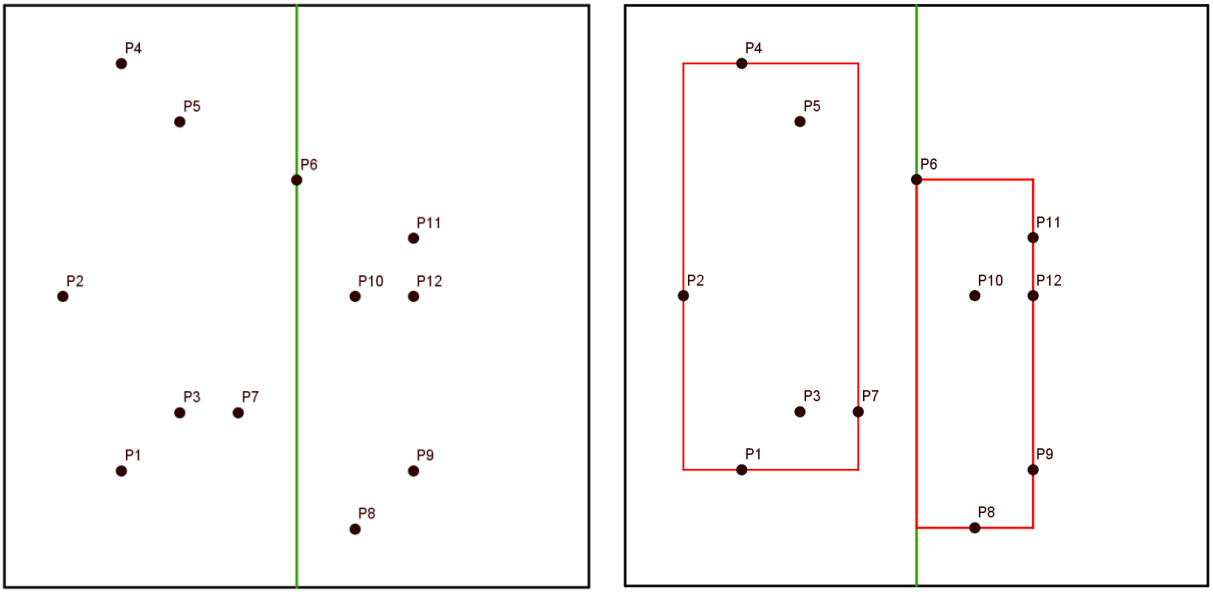
Dieses Verfahren wird man dann anwenden, wenn Datenpunkte ungleichmäßig im Raum verteilt sind, man aber dennoch jedem MBR nahezu gleich viele Datenpunkte zuordnen möchte.

Konstruktion 2: Gitterpartitionierung mit MBRs

Setzt man eine „Nahezu-Gleichverteilung“ der Datenpunkte im Raum voraus, so macht es Sinn, den Raum statt die Datenpunkte mit dem Median-Split-Verfahren aufzuteilen. Also bei jeder Teilung das entstandene Rechteck wieder in der nächsten Koordinate zu halbieren, so dass ein gleichmäßiges Gitter entsteht. Über die Anzahl der Teilungen je Koordinate lässt sich zwar keine genaue Anzahl gewünschter Datenpunkte je MBR erreichen, aber annähern.

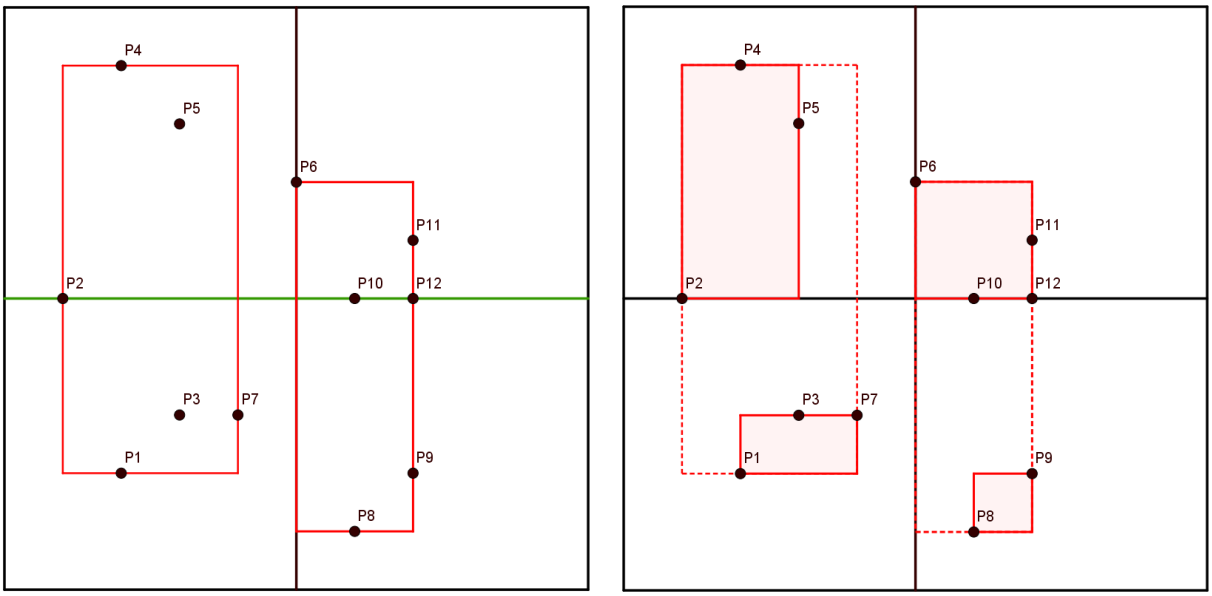
Nach jeder Teilung werden die Punkte in den einzelnen Gitterpartitionen zu MBRs zusammengefasst. Gemäß Lemma 1 bildet man dazu die Minima bzw. die Maxima über alle Koordinaten aller im jeweiligen MBR liegenden Datenpunkte. Bei diesem Verfahren wird im Vorfeld eine feste Anzahl von Teilungen je Koordinate angegeben, die das Gitter festlegt.

Die folgende Abbildung zeigt das Verfahren für jeweils eine Teilung in jeder Koordinate. Die grünen Linien sind die jeweils durchgeführten Teilungsschritte.



Teilung des Datenraumes in Koordinate 1 Erzeugung der MBRs je Partition

Abbildung 5.3: Mediane Teilung entlang der erste Koordinate



Teilung des Datenraumes in Koordinate 2 Erzeugung der MBRs je Partition

Abbildung 5.4: Mediane Teilung entlang der zweiten Koordinate

Die Wurzel und jeder Folgeknoten besteht aus zwei MBRs. Die rechte Figur von Abbildung 5.4 zeigt die Kinder der Wurzelknoten und macht deutlich, dass auch die MBRs ineinander verschachtelt sind. Um eine eindeutige Zuordnung der Punkte zu einem MBR zu gewährleisten, wird vorausgesetzt, dass die Punktemenge eines MBRs aus seinen rechts halboffenen Intervallen besteht (siehe Abbildung 5.5).

Dieses Verfahren erfordert deutlich weniger Rechenaufwand als das erste, da ein Median-Split des Raumes keine Sortierung von Punktkoordinaten erforderlich macht. Allerdings können Partitionen

(MBRs) mit wenigen, oder auch gar keinem Punkt entstehen, wenn die Datenpunkte ungleichmäßig verteilt sind. Man wird dieses Vorgehen also nur dann anwenden, wenn vorausgesetzt werden kann, dass die Datenpunkte im Raum nahezu gleichverteilt sind.

Konstruktion 3: Gitterpartitionierung mit BRs

Das letzte Verfahren unterscheidet sich vom zweiten lediglich darin, dass man von der Erzeugung der MBRs nach jeder Teilung in der Koordinate in den jeweiligen Gitterpartitionen absieht und damit das Gitter die Rechtecke selbst bilden (siehe Abbildung 5.5). Liegt ein Punkt $P = (P_1, P_2)$ auf der Kante mehrerer MBRs, so verfährt man wie oben angegeben. So gehört Punkt P6 aus Abbildung 5.5 zum rechten oberen MBR und P2 wird bei der zweiten Teilung dem oberen linken MBR zugeordnet. Gäbe es einen Punkt mit den Koordinaten $P = (0{,}5; 0{,}5)$, so würde dieser, wie P6 dem MBR rechts oben zugeordnet.

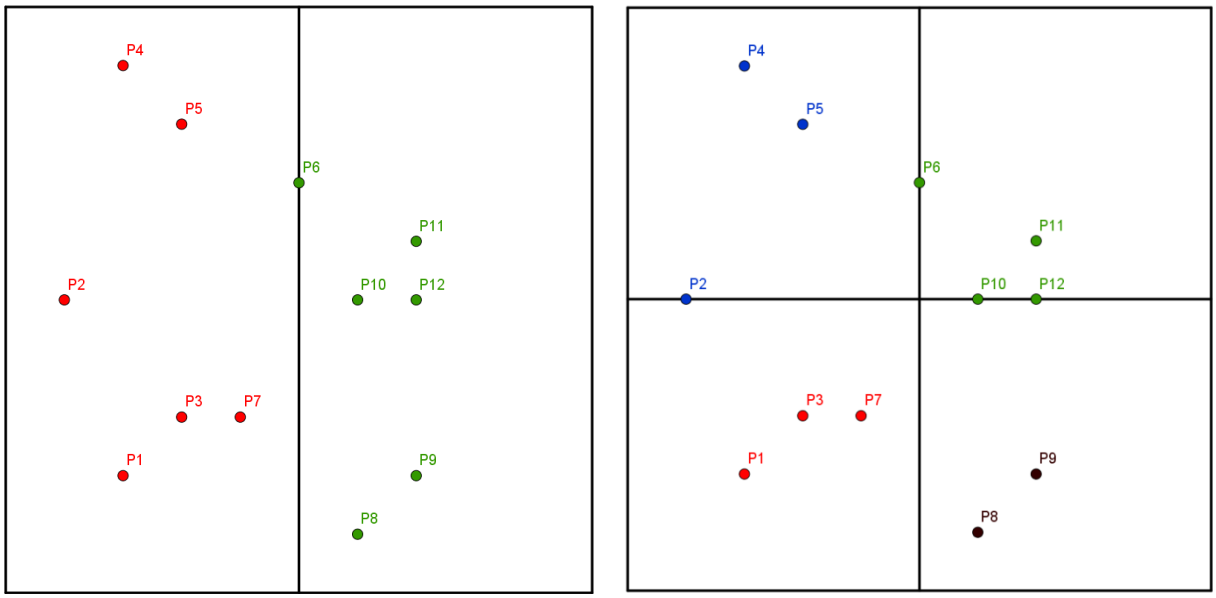


Abbildung 5.5: Zuordnung der Datenpunkte

Die Anwendung von Konstruktion 3 ist dann sinnvoll und dem zweiten Verfahren vorzuziehen, wenn eine Gleichverteilung der Punkte vorliegt *und* gleichzeitig die Anzahl von Punkten so groß ist, dass die Bildung eines MBR die ihn umgebende Gitterpartition nur marginal verkleinern würde.

In diesem Fall steht der Aufwand einer Minimierung und Maximierung der Koordinaten aller Datenpunkte zur Vermeidung von indiziertem Leerraum einer möglichen Indizierung von Leerraum entgegen, die wie gesagt, bei hinreichend großer Anzahl von Punkten vernachlässigbar ist. Der so erzeugte Baum ist kein R-Baum in dem Sinne, dass man minimal begrenzende Rechtecke der Datenpunkte in den Knoten hat, sondern nur begrenzende Rechtecke (bounding rectangles). Wir bezeichnen die so erzeugten Rechtecke im Folgenden entsprechend als BRs statt MBRs.

Da in unseren Evaluierungen der Fokus auf den zugrundeliegenden Bewertungsfunktionen mit Maximum- bzw. Minimumprinzip auf MBRs liegt, wollen wir eine Gleichverteilung mit großer Punktmenge erzeugen und Konstruktion 3 zur Anwendung bringen. Dieses Verfahren hat neben einer

einfachen Implementierung aus der Sicht der Evaluierung zudem den Vorteil, dass sich an einem gleichmäßigen Gitter und seinen MBRs, die wesentlichen Funktionseigenschaften demonstrieren und besser nachvollziehen lassen.

5.2 Relationales BRS-Modell

Aufgrund der Ausführungen zur Projektion einer Relation auf den Einheitswürfel aus Kapitel 2.2.4, dürfen wir voraussetzen, dass die Datenpunkte im zweidimensionalen Einheitswürfel I^2 enthalten sind. In dem nun folgenden Kapitel wird gezeigt, wie sich das BRS-Verfahren (R-Baum und Funktionen) zur Beantwortung von Top-k-Anfragen als relationales Modell abbilden lässt. Als Prioritätswarteschlange implementieren wir eine unsortierte Liste, gemäß Abbildung 3.27: Pseudocode des BRS+-Algorithmus, da sich die Operationen (PQ1)-(PQ4) dann mit Standard-SQL darstellen lassen und keine zusätzlichen Entwicklungen oder gar Anbindung externer Funktionsbibliotheken erforderlich sind.

Das im folgenden dargestellte auf SQL-Server basierende Modell hat zum einen den Vorteil, dass es sich auf jedes andere RDBS portieren lässt, zum anderen werden mit einer solchen Implementierung realistische und vergleichbare Szenarien zur naiven sequentiellen Suche im DBS möglich und messbar gemacht. Zur besseren Nachvollziehbarkeit werden wir die einzelnen Schritte und Zwischenergebnisse in Relationen speichern und die dazu notwendigen Routinen angeben und erläutern. Alle übrigen für unser Modell verwendeten Objekte, wie Tabellen, Prozeduren und Funktionen, sind im Anhang als DDL-Scripte abgebildet. Das Modell ist vollständig und lässt sich mit den angegebenen Scripten in einer SQL-Server-Datenbank ab Version 2012 implementieren.

Der besseren Anschauung wegen ist das gewählte Modell zweidimensional. Die Methoden lassen sich jedoch auch auf höhere Dimensionen übertragen. Das Verfahren wird unter Anwendung der Relation des Beispiels (Abbildung 2.12) erläutert. Auch wenn die Datenpunkte dieses Beispiels nicht gleichverteilt sind, so lassen sich unter seiner Verwendung einige Besonderheiten verdeutlichen.

5.2.1 Erzeugung der Relation mit Datenpunkten

Die N zugrundeliegenden Datenpunkte des Datenraumes werden in der Tabelle *R* gespeichert. Für die spätere Evaluation wird eine Gleichverteilung einer hinreichend großen Anzahl an Datenpunkten angestrebt, die sich im SQL-Server mit Hilfe der Zufallsfunktion $RAND()$ je Attribut erzeugen lassen.

Die entsprechende Routine ist die Folgende:

```
1 DECLARE @i INT = 1, @N INT = 1000, @ID varchar(10)

2 WHILE @i <= @N
3 BEGIN
4         SELECT @ID = 'P' + CONVERT(varchar(10),@i)
5         INSERT INTO R (PID, X1, X2) VALUES (@ID, RAND(), RAND());
6         SET @i = @i + 1;
7 END;
```

Abbildung 5.6: Erzeugung einer Gleichverteilung von Punkten

5.2.2 Gitter und BRs

Wir wenden das Konstruktionsverfahren 3 aus Kapitel 5.1 an. Dabei wird ein regelmäßiges Gitter unter Angabe der Anzahl von Teilungen t je Koordinate erzeugt und das Einheitsquadrat mit dem Median-Split-Verfahren in jeder Koordinate t-mal geteilt. Es entstehen Rechtecke, die wie für MBRs in einem R-Baum gefordert, von Ebene zu Ebene ineinander enthalten sind.

Um die Position eines MBRs im R-Baum, also seine Ebene und die Zugehörigkeit zum jeweiligen Vaterknoten identifizieren zu können, erhält jedes MBR eine NodeID, die aus einer Bit-Zeichenfolge besteht. Die Länge der NodeID entspricht der Tiefe des Knotens, also der Ebene, auf dem das MBR lokalisiert ist. Die Reihenfolge der Bits (0 und 1) zeigt die genaue Position des MBR innerhalb der Ebene an. Abbildung 5.7 und Abbildung 5.8 veranschaulichen das Verfahren für jeweils zwei Teilungen je Koordinate $(t = 2)$, also einer Baumhöhe von 4.

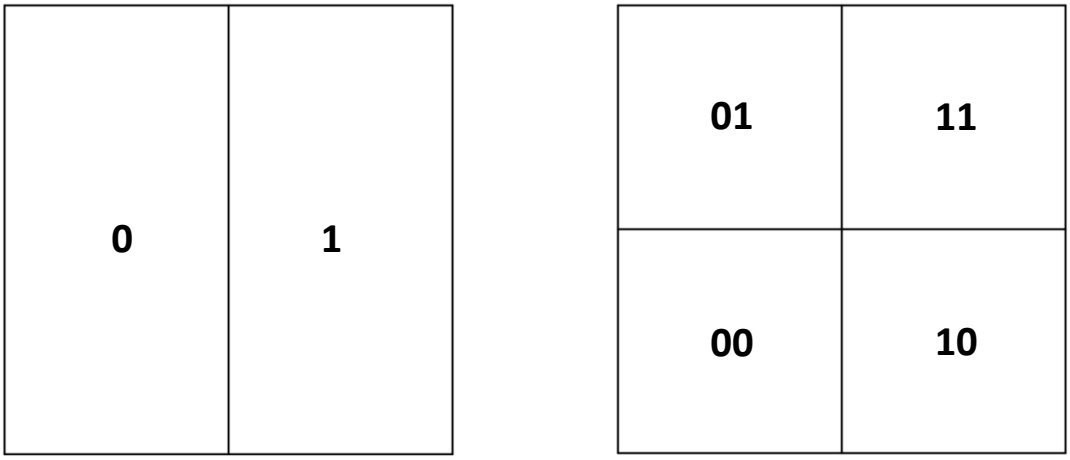


Abbildung 5.7: Teilung nach Koordinate 1 (Knoten und MBRs der zweiten Ebene)

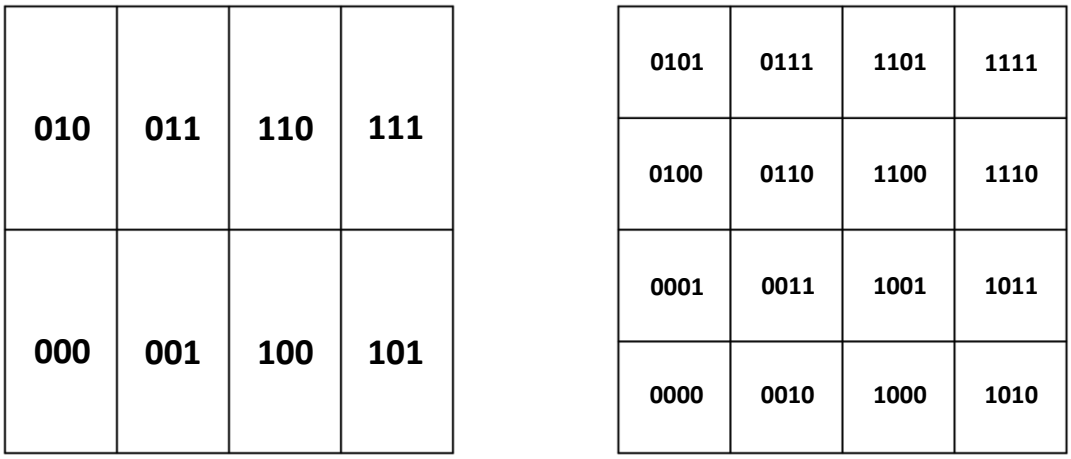


Abbildung 5.8: Teilung nach Koordinate 2 (MBRs der dritten und vierten Ebene)

Bei der ersten Teilung nach der x_1-Koordinate wird die Wurzel erzeugt. Man erhält durch Teilung des Einheitsquadrates das MBR mit der kleinsten Ecke und bezeichnet es als „0“. Das MBR mit der größten Ecke heißt „1“.

Im nächsten Schritt werden die Kinder von „0“ und „1“ durch Teilung entlang der x_2-Koordinate erzeugt. Bei der Teilung des MBRs „0“ erhält das Kind-MBR mit der kleinsten unteren Ecke die Bezeichnung „00“, das mit der größten Ecke den Namen „01“. Entsprechend entstehen bei der Teilung von „1“ die beiden Kinder „10“ und „11“, usw. Das Verfahren wird solange fortgesetzt, bis die Länge der NodeIDs = 4 ist. Bei der letzten Teilung wird das MBR „111“ in die beiden Kinder „1110“ und „1111“ zerlegt.

Anhand der NodeIDs lassen sich die Vaterknoten rekursiv zurückverfolgen. So gilt beispielsweise für den Knoten „1001“ die Inklusion (siehe Abbildung 5.9, blauer Pfad):

$$1001 \subseteq 100 \subseteq 10 \subseteq 1.$$

Der zugehörige R-Baum hat das folgende Aussehen:

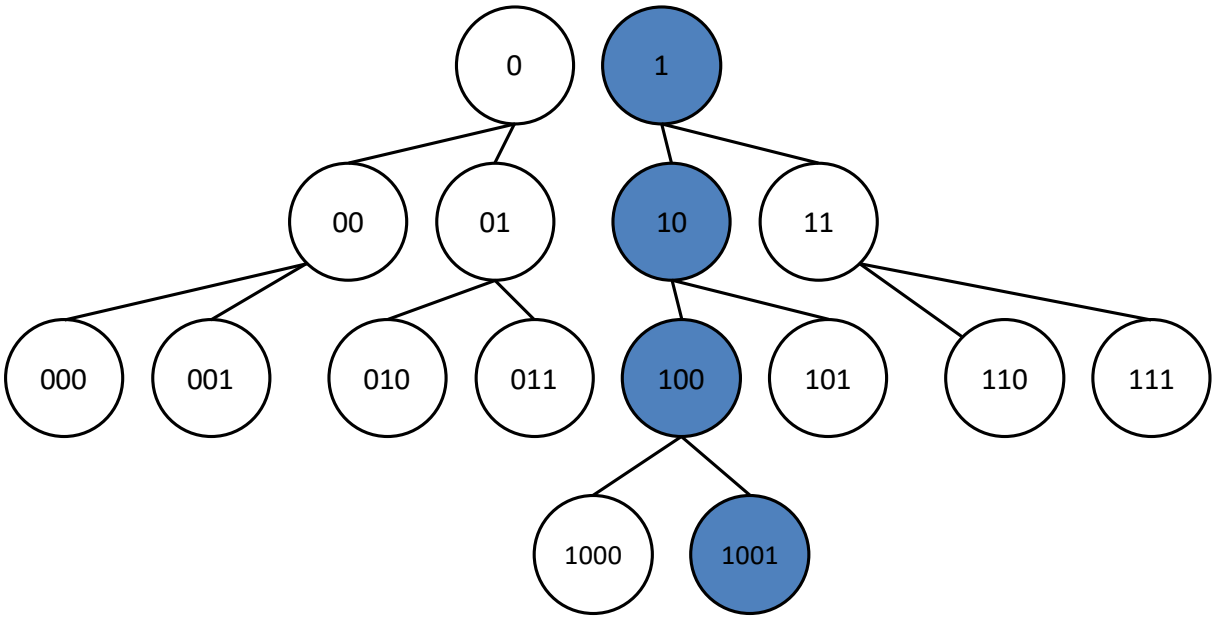


Abbildung 5.9: Beispiel eines binären R-Baums

Da wir wissen, dass jedes MBR durch die untere und obere Ecke aufgespannt wird, lässt sich der oben erzeugte R-Baum als fünfstellige Relation mit den Attributen: MBRName, P_1, P_2, Q_1 und Q_2 darstellen, wobei $P = (P_1, P_2)$ die untere, $Q = (Q_1, Q_2)$ die obere Ecke des jeweiligen MBRs ist.

In Abbildung 5.10 (unten) sind die BRs des R-Baumes mit den jeweiligen beiden Koordinatenpaaren der Punkte, die sie aufspannen abgebildet. Das zugehörige T-SQL-Script ist im oberen Teil von Abbildung 5.10 dargestellt. Als Ergebnis erhält man einen R-Baum der Höhe 4. Die jeweiligen Punkte sind mit Koordinaten als $P = (P_1, P_2), Q = (Q_1, Q_2)$, dargestellt.

Die ersten beiden Zeilen des Scripts erzeugen initial die beiden MBRs „0“ und „1“ des Wurzelknotens. Da die Prüfung der WHILE-Schleife zu Beginn erfolgt, führt die Angabe der Anzahl von Teilungen, hier exemplarisch durch $@Teilungen = 3$ zu 3+1, also 4 Teilungen, je zwei in jeder Koordinate. In Zeile

20 wird die Prozedur $BuildMBR$ mit Übergabe der unteren und oberen Ecke eines MBR aufgerufen, die die mediale Teilung eines MBRs durchführt. $BuildMBR$ befindet sich im Anhang.

Das Gitter muss für eine weitere Evaluation nur dann neu erzeugt werden, wenn man eine feinere Partitionierung zugrunde legen möchte. Von Änderungen der Bewertungsfunktion ist es nicht betroffen. Wir verstehen im Folgenden unter der *Granularität* einer Partitionierung die Anzahl der Gitterpartitionen, also die Anzahl der MBRs, die auf der letzten Nicht-Blattebene des R-Baumes lokalisiert sind.

```
INSERT INTO RBaum (MBR,P1,P2,Q1,Q2) Values ('0',0,0,0.5,1)
INSERT INTO RBaum (MBR,P1,P2,Q1,Q2) Values ('1',0.5,0,1,1)
GO

DECLARE @p1 real, @p2 real, @q1 real, @q2 real, @mbrVater varchar(10), @tk int, @len int, @mod int, @Teilungen int
SELECT @len = 1
/** EINGABE **/
SELECT @Teilungen = 1
WHILE @len <= @Teilungen
BEGIN
                SELECT @mod = @len % 2
                IF (@mod = 0) select @tk = 1 ELSE SELECT @tk = 2
                DECLARE db_cursor CURSOR FOR
                                SELECT DISTINCT P1, P2, Q1, Q2, MBR FROM RBaum WHERE len(MBR) = @len
                OPEN db_cursor
                FETCH NEXT FROM db_cursor INTO @p1, @p2, @q1, @q2, @mbrVater

                WHILE @@FETCH_STATUS = 0
                BEGIN
                                EXEC BuildMBR @p1, @p2, @q1, @q2, @mbrVater, @tk
                                FETCH NEXT FROM db_cursor INTO @p1, @p2, @q1, @q2, @mbrVater
                END
                CLOSE db_cursor
                DEALLOCATE db_cursor
SELECT @len = @len + 1
END
```

T-SQL Script zur Erzeugung des Gitters (MBRs)

MBR	P1	P2	Q1	Q2
0	0	0	0,5	1
1	0,5	0	1	1
00	0	0	0,5	0,5
01	0	0,5	0,5	1
10	0,5	0	1	0,5
11	0,5	0,5	1	1

Abbildung 5.10: Scripts zur Partitionierung (oben), MBRs mit Punktkoordinaten (unten)

5.2.3 Erzeugen der Blattebene

Im nächsten Schritt wird die noch fehlende Blattebene des R-Baumes gebildet. Dazu müssen die Datenpunkte der Relation R, gemäß ihrer Koordinaten, eindeutig den MBRs der letzten Nicht-Blattebene zugewiesen werden. Eine solche Zuordnung erfolgt durch Vergleich der Koordinaten jedes Datenpunktes mit den Einträgen aus Tabelle R.

Die entsprechende Prozedur und das Ergebnis der Zuordnung ist in Abbildung 5.11 und Abbildung 5.12 dargestellt. Damit die Routine funktioniert, müssen initial die beiden MBRs (0 und 1) der Wurzel in die Tabelle *RBaum* geladen werden (siehe Zeile 1 und 2, Abbildung 5.10, oben). Das Ergebnis der Zuordnungen wird in der Tabelle *RBaum_Blaetter* gespeichert. Wichtig dabei ist, dass MBRs und Datenpunkte verschiedene (eindeutige) Bezeichnungen besitzen.

```
 1 DECLARE @p1 real, @p2 real, @id varchar(10), @mbr varchar(10), @baumtiefe int
 2 /** EINGABE **/
 3 SELECT @baumtiefe = 2
 4 DECLARE db_cursor CURSOR FOR
 5         SELECT DISTINCT PID, X1, X2 FROM R
 6 OPEN db_cursor
 7 FETCH NEXT FROM db_cursor INTO @id, @p1, @p2
 8 WHILE @@FETCH_STATUS = 0
 9 BEGIN
10         SELECT @mbr = MBR FROM RBaum
11         WHERE @p1 >= P1 AND @p1 < Q1 AND @p2 >= P2 AND @p2 < Q2 AND Len(MBR) = @baumtiefe
12         INSERT INTO RBaum_Blaetter (PID, MBR, X1, X2) VALUES (@id, @mbr, @p1, @p2)
13         FETCH NEXT FROM db_cursor INTO @id, @p1, @p2
14 END
15 CLOSE db_cursor
16 DEALLOCATE db_cursor
```

Abbildung 5.11: Erzeugung der Blattebene

PID	MBR	X1	X2
P1	00	0,2	0,2
P2	01	0,1	0,5
P3	00	0,3	0,3
P4	01	0,2	0,9
P5	01	0,3	0,8
P6	11	0,5	0,7
P7	00	0,4	0,3
P8	10	0,6	0,1
P9	10	0,7	0,2
P10	11	0,6	0,5
P11	11	0,7	0,6
P12	11	0,7	0,5

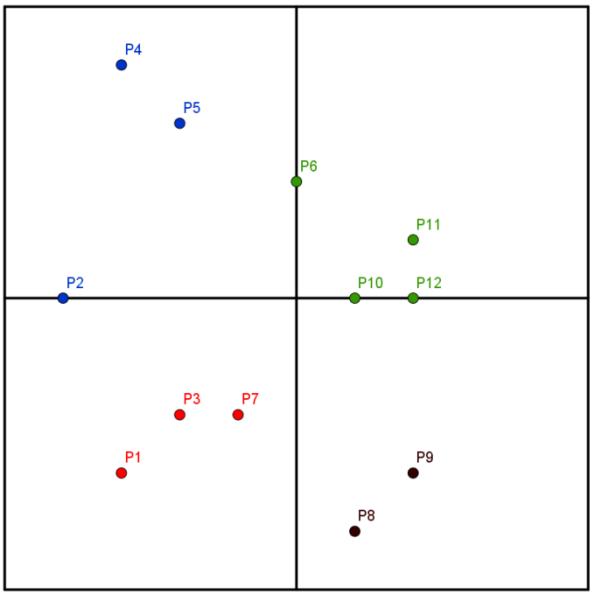


Abbildung 5.12: Tabelle: RBaum_Blaetter (links), Punktezuordnung graphisch (rechts)

5.2.4 Schranken der MBRs

Schritt 4 berechnet die oberen (unteren) Grenzen der einzelnen MBRs für eine gegebene Bewertungsfunktion und speichert die Werte zusammen mit den jeweiligen MBRs in der Tabelle *MBR_Grenzen*. Für die Bestimmung der maximalen Grenze (quasi-konvexe Funktionen) wird das Maximum der Funktionswerte der Ecken bestimmt. Zur Minimierung wird das in Kap. 4.3.3 dargestellte Verfahren für quadratische Funktionen verwendet.

Obergrenzen

Ist ein MBR durch seine untere Ecke $P = (P_1, P_2)$ und die obere $Q = (Q_1, Q_2)$ bestimmt, so lässt sich die maximale Ecke durch Übergabe der vier möglichen Paare der Koordinaten mit anschließender Maximierung derselben berechnen (siehe Abbildung 5.13, rechts).

MBR	Wert
0	1,25
00	0,75
01	1,25
1	1,25
10	0,75
11	1,25

```
WITH ALLEMaxima AS
(
SELECT MBR, dbo.RF(P1,P2) AS FWert
FROM RBaum
UNION
SELECT MBR, dbo.RF(Q1,Q2) AS FWert
FROM RBaum
UNION
SELECT MBR, dbo.RF(P1,Q2) AS FWert
FROM RBaum
UNION
SELECT MBR, dbo.RF(Q1,P2) AS FWert
FROM RBaum
)
INSERT INTO MBR_Grenzen (MBR, Wert)
SELECT MBR, MAX(FWert)
FROM ALLEMaxima
GROUP BY MBR
```

Abbildung 5.13: Obere Schranken für MBRs. Tabelle MBR_Grenzen (links), Script (rechts)

Untergrenzen

Zur Minimierung sind für die von den Punkten P und Q aufgespannten Seiten (Intervalle) $[P_1, Q_1]$ und $[P_2, Q_2]$ Fallunterscheidungen mit der Funktion getMinScore (siehe Abbildung 4.8) durchzuführen. Das Scripts mit Übergabe entsprechender Parameter und Aufruf von getMinScore ist in der folgenden Abbildung 5.14 dargestellt. Die Funktion getMinScore führt die Ermittlung des Minimums für jede Komponentenfunktion $\omega_i \cdot (x_i - q_i)^2$ durch. getMinScore befindet sich im Anhang.

MBR	Wert
0	0
00	0,009999
01	0
1	0,009999
10	0,009999
11	0,02

```
DECLARE @w1 real, @w2 real, @q1 real, @q2 real
SELECT @w1 = 1, @w2 = 1
SELECT @q1 = 0.4, @q2 = 0.4
INSERT INTO MBR_Grenzen (MBR, Wert)
SELECT MBR, dbo.getMinScore(P1,Q1,@w1,@q1)
           + dbo.getMinScore(P2,Q2,@w2,@q2)
FROM RBaum
```

Abbildung 5.14: Untere Schranken und MBRs in Tabelle MBR_Grenzen (links), Scripts (rechts)

5.2.5 BRS+-Algorithmus

Nach diesen Vorbereitungen kann für den in Abbildung 3.27 dargestellten BRS+-Algorithmus ein relationales Modell angegeben werden. Wir erläutern zunächst die Methode zur Maximierung, also für quasi-konvexe Funktionen. Zur Minimierung sind lediglich zwei kleine Änderungen am T-SQL-Script erforderlich, die im Anschluss aufgeführt werden. Der Ablauf ist ansonsten identisch.

Initial werden die ersten beiden MBRs, also die Knotenebene, aus der Relation *MBR_Grenzen* in die Prioritätswarteschlange (Tabelle *PQ*) geladen (siehe Abbildung 5.15, Zeile 1-3).

Die WHILE-Schleife (Zeile 8) wird solange ausgeführt, bis in der Prioritätswarteschlange k Datenobjekte vom $Typ = P$ gefunden werden, also Datenpunkte und keine MBRs ($Typ = M$). Die IF-ELSE-Unterscheidung (Zeile 16 und Zeile 20) ist erforderlich, weil die MBRs samt (oberen) Grenzen in der Tabelle *MBR_Grenzen*, die Datenpunkte der Blätter in der Tabelle *RBaum_Blaetter* gespeichert sind, was zwei verschiedene SELECT-Anweisungen erforderlich macht (siehe Zeile 17-19 und Zeile 21-25). Der Einsatz eines Cursors (Zeile 10 und 11) bildet ein temporäres Recordset (vgl. [57]) bestehend aus MBRs, da im Allgemeinen mehr als nur ein MBR mit demselben maximalen Wert in der Prioritätswarteschlange enthalten sein kann.

Enthält die Top-k-Anfrage eine WHERE-Bedingung, so die INSERT-Anweisung in Zeile 21-25, welche die Datenpunkte in die Prioritätswarteschlange überträgt, durch diese entsprechende Bedingung zu erweitern, um lediglich diejenigen Datenpunkte in die PQ zu übertragen, die diese Bedingung erfüllen. Der Aufbau des R-Baumes und seiner MBRs ist von einer solchen WHERE-Bedingung nicht betroffen.

```
   Algorithmus BRS+
 1 INSERT INTO PQ (Datenobjekt, Wert, Typ) SELECT MBR, Wert, 'M'
 2 FROM MBR_Grenzen
 3 WHERE LEN(MBR) = 1
 4 DECLARE @check bit, @k int, @ebene int, @MBR varchar(10), @maxLen int, @anzahl_topk int
 5 SELECT @check = 0, @anzahl_topk = 0, @maxLen = MAX(LEN(MBR)) FROM MBR_Grenzen
 6 /** Eingabe **/
 7 SELECT @k = 3
 8 WHILE @anzahl_topk < @k
 9 BEGIN
10        DECLARE db_cursor CURSOR FOR
11               SELECT Datenobjekt FROM PQ WHERE Wert IN (SELECT DISTINCT MAX(WERT) FROM PQ WHERE Typ = 'M')
12        OPEN db_cursor
13        FETCH NEXT FROM db_cursor INTO @MBR
14        WHILE @@FETCH_STATUS = 0
15        BEGIN
16               IF LEN(@MBR) <> @maxLen
17                        INSERT INTO PQ (Datenobjekt, Wert, Typ)
18                        SELECT MBR, Wert, 'M' AS Typ FROM MBR_Grenzen
19                        WHERE LEN(MBR) = LEN(@MBR) + 1 AND LEFT(MBR,LEN(@MBR)) = @MBR
20               ELSE
21                        INSERT INTO PQ (Datenobjekt, Wert, Typ)
22                        SELECT TOP (@k) PID AS MBR, [dbo].[RF] (X1,X2) AS FWert, 'P' AS Typ
23                        FROM RBaum_Blaetter
24                        WHERE MBR = @MBR
25                        ORDER BY [dbo].[RF] (X1,X2) DESC
26               DELETE FROM PQ WHERE Datenobjekt = @MBR
27        FETCH NEXT FROM db_cursor INTO @MBR
28        END
29        CLOSE db_cursor
30        DEALLOCATE db_cursor
31        SELECT @anzahl_topk = COUNT(Datenobjekt) FROM PQ WHERE Wert IN
32        (SELECT DISTINCT TOP (@k) Wert FROM PQ ORDER BY Wert DESC) AND Typ = 'P'
   END
```

Abbildung 5.15: BRS+-Algorithmus (T-SQL)

Die Top-3-Kandidaten der zwölf Datenpunkte des durchgängigen Beispiels sind graphisch und als Output des obigen BRS+-Algorithmus-Scriptes in Abbildung 5.16 dargestellt. Die obere Niveauline der graphischen Darstellung rechts „beweist“ die Korrektheit des Ergebnisses geometrisch.

Datenobjekt	Wert	Typ
P4	0,99	P
P5	0,84	P
P6	0,7	P

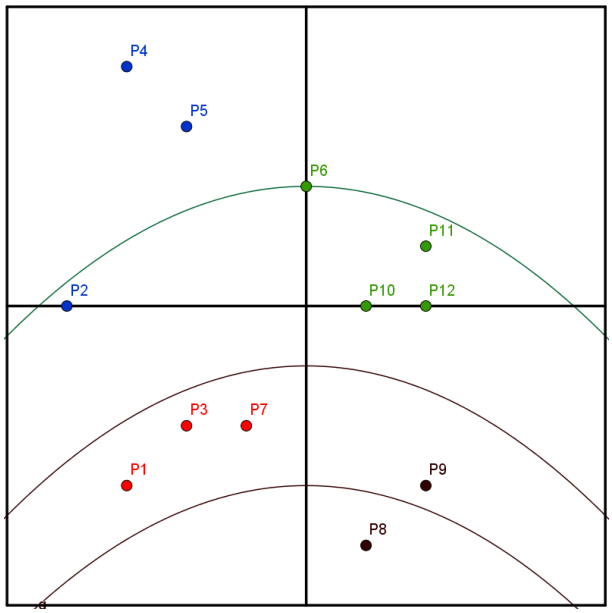


Abbildung 5.16: Top-3-Datenpunkte

Zur Ermittlung der top-k kleinsten Datenpunkte, also im Falle einer quadratischen Bewertungsfunktion, müssen im BRS+-Algorithmus lediglich die Zeilen 11 und 32 durch die folgenden beiden Zeilen ersetzt werden:

```
11 SELECT Datenobjekt FROM PQ WHERE Wert IN (SELECT DISTINCT MIN(WERT) FROM PQ WHERE Typ = 'M')

32 (SELECT DISTINCT TOP (@k) Wert FROM PQ ORDER BY Wert ASC) AND Typ = 'P'
```

Abbildung 5.17: BRS+-Algorithmus Anpassungen zur Minimierung

Sei N die Anzahl der Datenpunkte, g die Granularität der Partitionierung und damit die Anzahl der MBRs auf der untersten Ebene des R-Baumes, von denen c besucht werden müssen, so ergeben sich die folgenden Schritte für den BRS+ Algorithmus:

BRS+1: Entferne den MBR mit der höchsten Priorität, also der minimalen (maximalen) Obergrenze aus der unsortierten Liste (Prioritätswarteschlange).

BRS+2: Ermittle die Top-k-Elemente der einem MBR mit $\frac{N}{g}$ zugeordneten Datenpunkten. Hier setzen wir vereinfacht eine Gleichverteilung hinsichtlich der Anzahl von Datenpunkten in den MBRs voraus.

BRS+3: Vereinige die Menge aus bereits vorhanden (untersuchten) k Datenpunkten mit den k neu ermittelten.

BRS+4: Bestimme die Top-k-Elemente aus einer Liste von $2k$ Punkten.

Damit ergibt sich:

$$T_{BRS+} \in \mathcal{O}\left(c\left[T_{deleteMin}(g) + T_{Topk}\left(\frac{N}{g}\right) + T_{Merge}(2k) + T_{Topk}(2k)\right]\right).$$

Da wir die Prioritätswarteschlange als unsortierte Liste implementiert haben, so ergibt sich gemäß Tabelle Abbildung 3.6, dass

$$T_{BRS+} \in \mathcal{O}\left(c\left[g + \left(\frac{kN}{g}\right) + 1 + 2k^2\right]\right),$$

also eine Laufzeit des BRS+ von

(L2) $$\approx \mathcal{O}\left(c \cdot \frac{kN}{g}\right).$$

Wenn alle MBRs der untersten Ebene besucht werden müssen (Traversierung), also $c = g$ ist, so erhält man (L1), also die Laufzeit des naiven sequentiellen Suchens. Die Variable c ist nicht abhängig von der Anzahl der Datenpunkte N, sondern lediglich abhängig von der Bewertungsfunktion und der (geometrischen) Lage der Datenpunkte (Anfragekomplexität). Hinsichtlich der Datenkomplexität ist der BRS+ demnach linear in der Anzahl N der Datenpunkte mit festen, aber kleinem k. Das Verfahren wird dominiert von der Suche nach Top-k-Kandidaten in den Datenpunkten der MBRs auf der untersten inneren Ebene des R-Baumes.

5.3 Evaluation

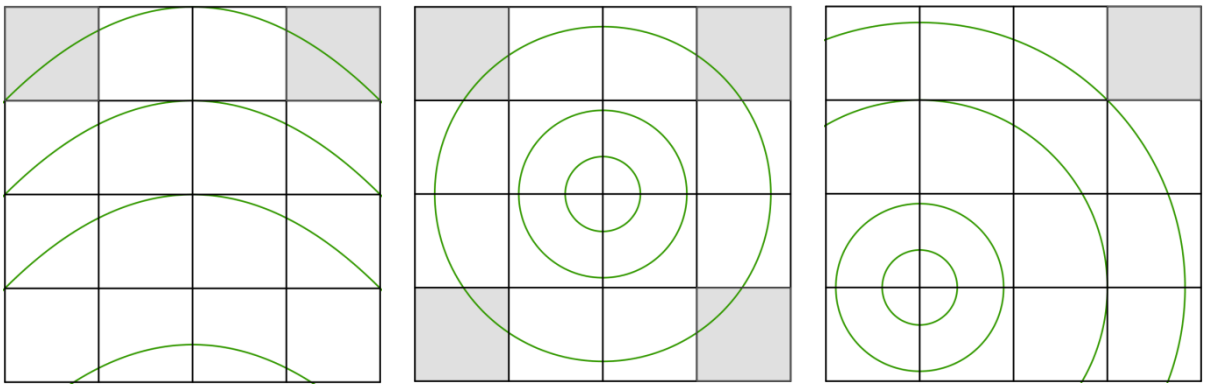
Im nun folgenden Kapitel wird der BRS+-Algorithmus auf quasi-konvexe und quadratische Funktionen angewendet und die Laufzeit-Effizienz für drei Vertreter je Klasse untersucht. Wir werden einen R-Baum gemäß Konstruktionsverfahren 3 (siehe Kapitel 5.2) aufbauen, also eine hinreichend große, nahezu gleichverteilte Anzahl Datenpunkte im Raum erzeugen und mit einem gleichmäßigen Gitter den Datenraum partitionieren. Die Partitionen entsprechen den MBRs des R-Baumes. Anhand einer solchen Gitterpartition lassen sich die wesentlichen Eigenschaften der gewählten Funktionen herausstellen und die Evaluationsergebnisse leichter interpretieren, als das bei beliebigen MBRs der Fall ist. In allen aufgeführten Szenarien werden die Top-3-Kandidaten gesucht. Die Laufzeiten beider Verfahren (Sequentielles Lesen und BRS+) ist in Millisekunden (ms) gemessen.

5.3.1 Quasi-konvexe Bewertungsfunktionen

Es werden drei Beispiele untersucht und die entsprechende Bewertungsfunktion maximiert.

(1) $f_1(x_1, x_2) = (x_1 - 0{,}5)^2 + x_2,$
(2) $f_2(x_1, x_2) = (x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2,$
(3) $f_3(x_1, x_2) = (x_1 - 0{,}25)^2 + (x_2 - 0{,}25)^2.$

Die folgende Abbildung zeigt die Niveaulinien der Funktionen und deutet die Partitionen an, in denen bei angenommener Gleichverteilung und einer hinreichend großen Anzahl von Datenpunkten, die Top-k-Kandidaten erwartungsgemäß zu finden sind (graue Quadranten).



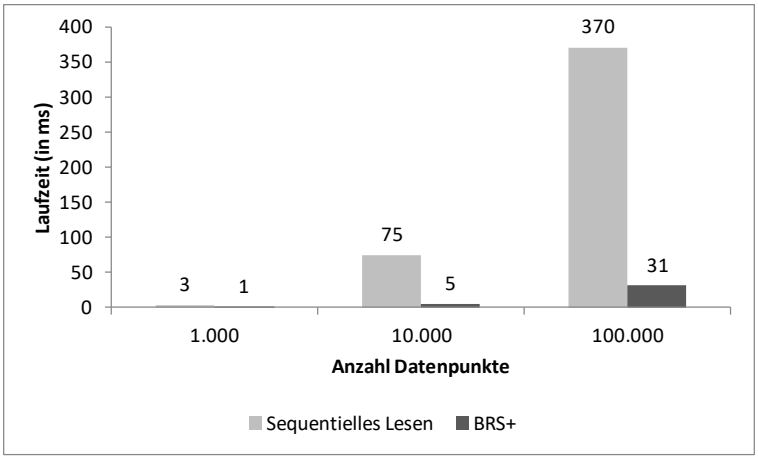
(1) (2) (3)

Abbildung 5.18: Beispiele quasi-konvexer Funktionen

Szenario 1: Variation Anzahl Punkte

Im ersten Szenario wird für Funktion (1) und einer Gittergranularität von 64, also einer Baumhöhe von 6, untersucht, wie sich eine Änderung der Anzahl der Datenpunkte auf das Laufzeitverhalten auswirkt.

Die in Abbildung 5.19 dargestellten Ergebnisse bestätigen die Erwartung, dass eine Erhöhung der Datenpunkte grundsätzlich die Laufzeit eines jeden Verfahrens erhöht. Es zeigt sich, dass das BRS-Verfahren das sequentielle Lesen für eine in einer Achse symmetrischen Funktion um Faktor 3 bis 15 dominiert.



Funktion	k	Granularität	Anzahl Datenpunkte	Sequentielles Lesen	BRS+	Faktor
(1)	3	64	1.000	3	1	3,0
(1)	3	64	10.000	75	5	15,0
(1)	3	64	100.000	370	31	11,9

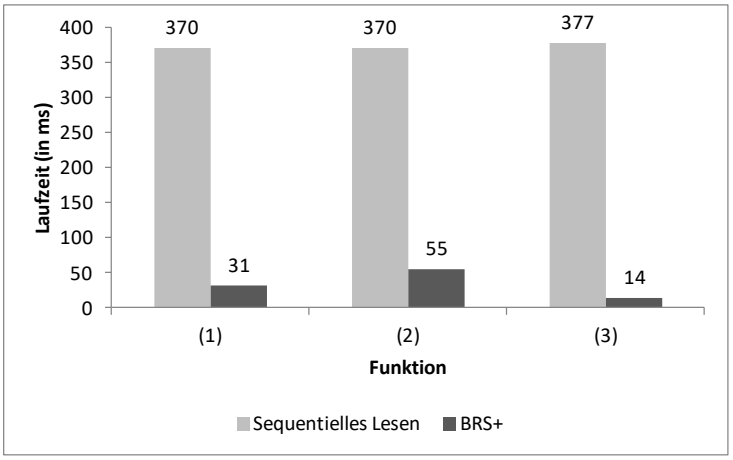
Abbildung 5.19: Szenario 1

Szenario 2: Variation der Bewertungsfunktion

Im zweiten Szenario wird bei fester Granularität und einer Anzahl von 100K Datenpunkten die Bewertungsfunktion variiert. Abbildung 5.18, (1) zeigt, dass der BRS+-Algorithmus mit Funktion (1) in einer regelmäßigen Gitterpartition immer zwei Pfade bis zu den Blattknoten parallel durchlaufen wird, bevor er die ersten Datenpunkte in die Prioritätswartschlange einfügen kann. Grund dafür ist, dass alle Knoten der beiden äußeren Pfade (vgl. Abbildung 5.9) denselben maximalen Wert (obere Schranke) besitzen.

Alle Funktionen, deren Niveaulinien symmetrisch zur x_1-Achse, oder zur x_2-Achse sind, haben diese Eigenschaft, die sich natürlich negativ auf die Laufzeit des BRS+-Algorithmus auswirkt. Funktion (2), deren Komponentenfunktionen der Form $(x_i - 0{,}5)^2$ sind, hat ihr Minimum im Zentrum des Hyper-Einheitsrechtecks und damit ihre Maxima auf *allen* Ecken desselben. Das bedeutet, dass der BRS+-Algorithmus mit Funktion (2) und der gewählten Gitterpartition immer vier Pfade bis zu den Blattknoten parallel durchläuft, bevor er die ersten Datenpunkte in die Prioritätswartschlange einfügen kann. Funktion (2) stellt damit den „Worst-Case“ hinsichtlich einer quasi-konvexen Funktion dar. Wir erwarten die beste Laufzeit für Funktion (3), gefolgt von Funktion (1), sowie die schlechteste für Funktion (2).

Die in Abbildung 5.20 aufgeführten Ergebnisse bestätigen die Erwartungen. Für alle quasi-konvexen Funktionen, die nicht symmetrisch sind, wie Funktion (3), ist gemäß Experiment eine etwa 27-fache schnellere Laufzeit des BRS+ gegenüber der sequentiellen Suche zu erwarten.



Funktion	k	Granularität	Anzahl Datenpunkte	Sequentielles Lesen	BRS+	Faktor
(1)	3	64	100.000	370	31	11,9
(2)	3	64	100.000	370	55	6,7
(3)	3	64	100.000	377	14	26,9

Abbildung 5.20: Szenario 2

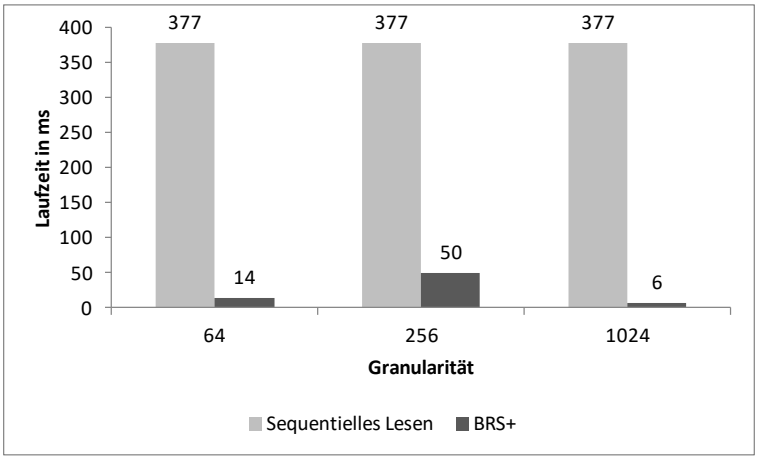
Szenario 3: Variation der Granularität

Im letzten Szenario verändern wir die Granularität des Gitters und wählen Funktion (3) als Bewertungsfunktion mit jeweils 100K Datenpunkten. Eine „feinere“ Partitionierung sollte grundsätzlich vorteilhaft für den BRS+-Algorithmus sein, da weniger Datenpunkte den einzelnen Partitionen zugeordnet werden und damit, nach besuchtem Blatt, weniger Funktionswerte berechnet und Datenpunkte in die Warteschlange geladen werden müssen. Diesem Vorteil steht allerdings eine größere Baumhöhe und die Möglichkeit zu wenig „günstig“ gelegene Datenpunkte in einem MBR zu haben, entgegen. Ein Datenpunkt liegt *günstig* in einem (besuchten) MBR, wenn sein Funktionswert oberhalb jeder Grenze eines noch nicht besuchten MBRs liegt. Solange genügend günstig gelegene Datenpunkte in den einzelnen Partitionen (BRs) vorhanden sind, wird die erhöhte Granularität den BRS+-Algorithmus beschleunigen.

Eine Granularität von 64 ergibt eine Gesamtzahl von 126 MBRs auf 6 Ebenen des R-Baumes, eine von 256 erzeugt 512 MBRs, während eine Granularität von 1024 bereits 10 Ebenen mit insgesamt 2046 verschiedenen MBRs erzeugt. Im letzen Fall würden bei gegebener Gleichverteilung der Datenpunkte ca. 48 Punkte auf jeden MBR verteilt.

Liegen mindestens drei der Datenpunkte günstig, also oberhalb der angrenzenden dominierenden Ecken der MBRs mit kleinerer dominierender Ecke, so wird der BRS+ seine Top-3-Kandidaten im ersten Blatt finden und muss keinen weiteren Blattknoten in die Suche einbeziehen. Diese Konstellation ist in Szenario 3 (Abbildung 5.21) im ersten und dritten Fall gegeben. Im zweiten Fall müssen aufgrund ungünstig gegebener Punkteverteilungen zusätzlich die Datenpunkte zweier MBRs der letzten inneren Ebene in die Wartschlange überführt werden, so dass der BRS-Algorithmus „nur“ etwa 7,5 mal schneller ist, als die sequentielle Suche.

Grundsätzlich lässt sich einer ungünstigen Punkteverteilung mit einer Partitionierungsstrategie im Vorfeld der Ausführung des BRS+-Algorithmus leider nicht entgegenwirken, da der Rang neben der (geometrischen) Lage eines Punktes natürlich auch von der Bewertungsfunktion abhängig ist.



Funktion	k	Granularität	Anzahl Datenpunkte	Sequentielles Lesen	BRS+	Faktor
(3)	3	64	100.000	377	14	26,9
(3)	3	256	100.000	377	50	7,5
(3)	3	1024	100.000	377	6	62,8

Abbildung 5.21: Szenario 3

5.3.2 Quadratische Bewertungsfunktionen

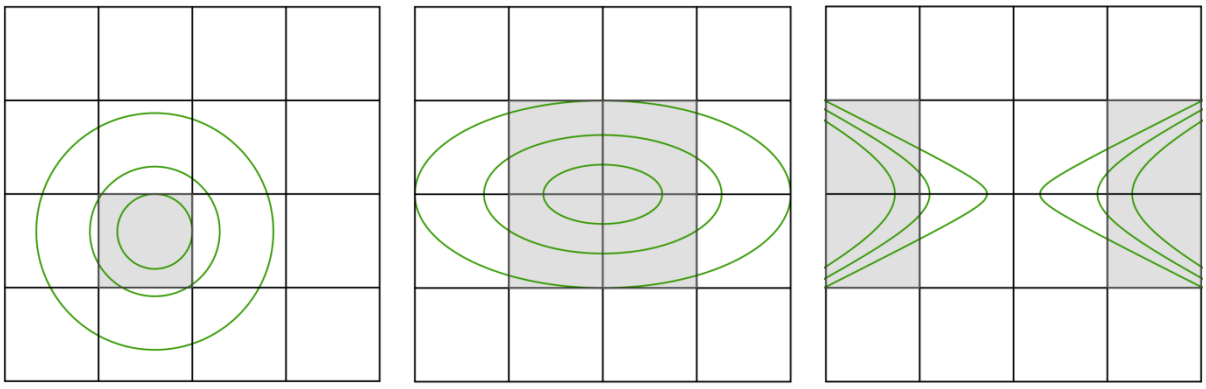
Es werden drei Beispiele untersucht und die entsprechende Bewertungsfunktion minimiert.

(4) $g_1(x_1, x_2) = (x_1 - 0{,}4)^2 + (x_2 - 0{,}4)^2,$

(5) $g_2(x_1, x_2) = 0{,}2 \cdot (x_1 - 0{,}5)^2 + 0{,}8 \cdot (x_2 - 0{,}5)^2,$

(6) $g_3(x_1, x_2) = 0{,}2 \cdot (x_1 - 0{,}5)^2 - 0{,}8 \cdot (x_2 - 0{,}5)^2.$

Die folgende Abbildung zeigt die Niveaulinien der Funktionen und deutet die Partitionen an, in denen bei angenommener Gleichverteilung und einer hinreichend großen Anzahl von Datenpunkten, die Top-k-Kandidaten zu erwarten sind (graue Quadranten).



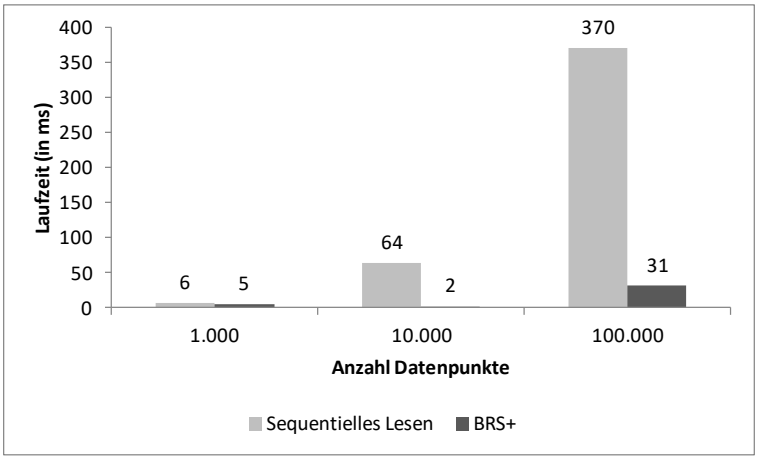
(4) (5) (6)

Abbildung 5.22: Beispiele quadratischer Funktionen

Szenario 4: Variation Anzahl Punkte

Die nun folgenden Szenarien sind in Analogie zu denen der quasi-konvexen Funktionen aufgebaut. Im vierten Szenario wird für Funktion (4) und einer Gittergranularität von 64 untersucht, wie sich die Variation der Anzahl der Datenpunkte auf das Laufzeitverhalten auswirkt.

Wie im quasi-konvexen Fall führt eine Erhöhung der Anzahl der Datenpunkte grundsätzlich zu einer Verlangsamung beider Verfahren. Solange hinreichend viele Datenpunkte in den MBRs eine günstige Lage haben und der BRS+-Algorithmus nicht zu viele Blätter besuchen muss, ist er der sequentiellen Suche überlegen. Müssen aber zu viele Blattknoten besucht und deren Datenpunkte in die Prioritätswarteschlange (ungeordnete Liste) übernommen werden, kann der BRS+ der sequentiellen Suche auch unterlegen oder nahezu gleichwertig sein (siehe Abbildung 5.23, erster Fall).



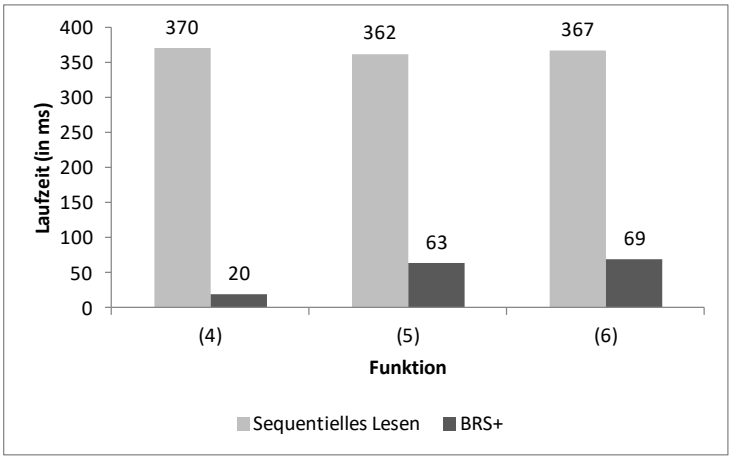
Funktion	k	Granularität	Anzahl Datenpunkte	Sequentielles Lesen	BRS+	Faktor
(4)	3	64	1.000	6	5	1,2
(4)	3	64	10.000	64	2	32,0
(4)	3	64	100.000	370	31	11,9

Abbildung 5.23: Szenario 4

Szenario 5: Variation der Bewertungsfunktion

Wir untersuchen jetzt das Laufzeitverhalten beider Verfahren bei Variation der Bewertungsfunktion. Abbildung 5.22 lässt vermuten, dass Funktion (4) den beiden anderen überlegen sein wird. Für Funktion (5) und (6) ist grundsätzlich, also bei einer hohen Anzahl gleichverteilter Datenpunkte eine ähnliche Laufzeit des BRS+-Algorithmus zu erwarten.

Die Ergebnisse aus Abbildung 5.24 bestätigen die Erwartungen. Das BRS-Verfahren erzielt das beste Ergebnis mit einer Funktion, die ihr Minimum im Inneren eines MBRs besitzt (Funktion 4, Faktor 18,5). Es ist deutlich schneller als das mit Funktionen der Fall ist, die auf dem Rand oder auf Ecken von MBRs minimal sind, wie Funktion (5) und (6).

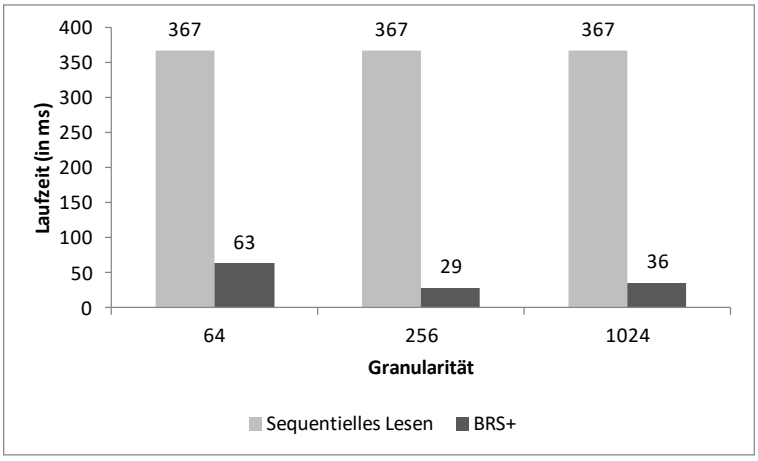


Funktion	k	Granularität	Anzahl Datenpunkte	Sequentielles Lesen	BRS+	Faktor
(4)	3	64	100.000	370	20	18,5
(5)	3	64	100.000	362	63	5,7
(6)	3	64	100.000	367	69	5,3

Abbildung 5.24: Szenario 5

Szenario 6: Variation der Granularität

Eine Verfeinerung der Partitionierung, also eine Erhöhung der Granularität des Gitters, hat dieselben Konsequenzen auf das Laufzeitverhalten des BRS+-Algorithmus wie im quasi-konvexen Fall. Optimal ist es, wenn der erste Besuch der Blattebene zum erfolgreichen Abbruch des Verfahrens führt. Dies konnte in allen drei Fällen von Szenario 6 erreicht werden.



Funktion	k	Granularität	Anzahl Datenpunkte	Sequentielles Lesen	BRS+	Faktor
(6)	3	64	100.000	367	63	5,8
(6)	3	256	100.000	367	29	12,7
(6)	3	1024	100.000	367	36	10,2

Abbildung 5.25: Szenario 6

Fazit

Die in der Evaluation durchgeführten Berechnungsszenarien zeigen zum einen, dass der BRS+-Algorithmus unter Verwendung von quasi-konvexen und quadratischen Bewertungsfunktionen der (naiven) sequentiellen Suche überlegen ist.

Zum anderen wird deutlich, dass sich das Laufzeitverhalten des BRS-Verfahrens, unabhängig von der gewählten Bewertungsfunktion, nur durch eine „hinreichend" große Anzahl von Datenpunkten in den MBRs positiv beeinflussen lässt. Vor Anwendung des Suchalgorithmus sollte daher, wenn möglich, als erstes die Art der Verteilung der Datenpunkte geprüft werden, um sich für ein geeignetes Konstruktionsverfahren des R-Baumes entscheiden zu können.

Die Anzahl zu besuchender MBRs, insbesondere die der untersten Knotenebene, beeinflusst in entscheidendem Maße das Laufzeitverhalten des BRS-Verfahrens, da durch den Besuch eines Blattes die entsprechenden Top-k-Datenpunkte ermittelt und in die Warteschlange gelangen werden.

Diese Anzahl steigt, wie oben bereits erwähnt, wenn bei der Verarbeitung der Warteschlange wiederholt weniger als k Datenpunkte in den besuchten MBRs ungünstig liegen, also für mindestens einer der k vorläufigen Kandidaten, der entsprechende Funktionswert im Falle des Maximierungs-problems unterhalb der oberen Schranke eines MBRs der Warteschlange liegt. Da man im Vorfeld der Anwendung des BRS+-Algorithmus aber weder die Bewertungsfunktion, noch die genau Lage und damit die Zuordnung der Datenpunkte zu den MBRs vorhersagen kann, kann man nur mit einer hinreichend großen Anzahl zugeordneter Punkte zu den MBRs zu vielen (notwendigen) Besuchen von Blättern entgegenwirken.

In Kapitel 5.2 wurde gezeigt, dass die Laufzeit zur Ermittlung der Top-k-Objekte einer Menge von N Datenpunkten mit dem BRS+-Verfahren beschrieben wird in einer Zeitfunktion von $\mathcal{O}\left(c \cdot \frac{kN}{g}\right)$. Der BRS+-Algorithmus wird demnach von der Suche nach den Top-k-Kandidaten in den Mengen der Datenpunkte dominiert, die den (besuchten) MBRs der letzten inneren Knotenebene des R-Baumes zugeordnet sind.

Dieses Ergebnis konnte durch die oben aufgeführten Evaluationsergebnisse der verschiedenen Szenarien bestätigt werden. Wie groß die Anzahl c besuchter MBRs auf unterster Nicht-Blattebene sein wird, hängt von der Bewertungsfunktion (vgl. Abbildung 5.18 und Abbildung 5.22) ab und davon, ob die Datenpunkte in diesen MBRs günstig liegen oder eben nicht.

6 Zusammenfassung und Ausblick

Monotone Funktionen wurden in einer Vielzahl von Anfragebeantwortungsstrategien untersucht und einige Methoden zur Beantwortung von Top-k-Anfragen vorgeschlagen. Beispiele dafür sind die Arbeiten [58], [59] und [60]. Geometrisch zeichnen sich monoton steigende (fallende) Funktionen insbesondere dadurch aus, dass sie auf einer ausgezeichneten Ecke eines achsenparallelen Hyperrechtecks maximal (minimal) sind, was sich günstig auf die Entwicklung von Verarbeitungsstrategien von Algorithmen auswirkt, die einen R-Baum zur Indizierung verwenden. Für nichtmonotone Funktionen lässt sich, wie schon die Arbeiten [61] deutlich macht, allgemein keine eindeutige Aussage über den geometrischen Ort solcher Extremstellen angeben. Man wird daher spezielle Funktionsklassen untersuchen (siehe [62] und [63]) und nach Eigenschaften suchen, die mit der gewählten Partitionierungsstrategie der Datenpunkte kompatibel sind. In dieser Arbeit wurden die quasi-konvexen Funktionen als diejenigen identifiziert, die auf den Ecken eines jeden allgemeinen Hyperrechteckes maximal sind, wodurch sie sich als Verallgemeinerung monotoner Funktionen im Kontext der Beantwortung von Top-k-Anfragen auszeichnen. Quadratische Funktionen erfüllen ein Minimumprinzip auf achsenparallelen Hyperrechtecken, oder das Minimum liegt innerhalb und ist Null. In beiden Fällen lassen sich die Extremstellen explizit bestimmen und obere, wie untere Schranken, für die MBRs eines gewählten R-Baumes effizient berechnen. Beide Funktionsklassen sind damit gute Kandidaten für die Anwendung des Branch-and-Bound-Prinzips. Die hier erzielten Ergebnisse und natürlich auch die Vorgehensweise wie sie erreicht wurden, geben Anlass zur Untersuchung weiterer Fragestellungen und Forschungsthemen, die sich in (mindestens) vier Kategorien einteilen lassen.

Anwendung (quasi-)konvexer Funktionen auf weitere Verfahren

Die Verallgemeinerung monotoner Funktionen im Kontext des BRS-Verfahrens ist ein erster Schritt, der gezeigt hat, wie wichtig die Klasse der konvexen und quasi-konvexen Funktionen ist. So, wie sich monotone Funktionen für viele andere Verfahren eignen, gilt es zu untersuchen, inwieweit sich die Eigenschaft der Quasi-Konvexität auch für andere, schon bekannte effiziente Verfahren nutzen lässt.

Unterklassen (quasi-)konvexer Funktionen für das Minimierungsproblem

Quadratische Funktionen sind eine spezielle Unterklasse quasi-konvexer Funktionen, für die wir zeigen konnten, dass sie ein Minimumprinzip auf Hyperechtecken erfüllen. Aufgrund ihrer „einfachen“ Struktur lassen sich ihre minimalen Punkte explizit berechnen und damit ein Algorithmus zur Bestimmung unterer Schranken für das BRS-Verfahren angeben. Es stellt sich die Frage, ob es weitere Unterklassen quasi-konvexer Funktionen gibt, für die sich das Minimumproblem mit dem Branch-and-Bound-Prinzip effizient lösen lässt. Natürlich sollten etwaige in Frage kommende Klassen nicht nur von theoretischem Interesse sein, sondern natürlich immer mit einer aus der Praxis verbundenen Problemstellung in Verbindung gebracht werden können.

Verallgemeinerung des MBR-Prinzips

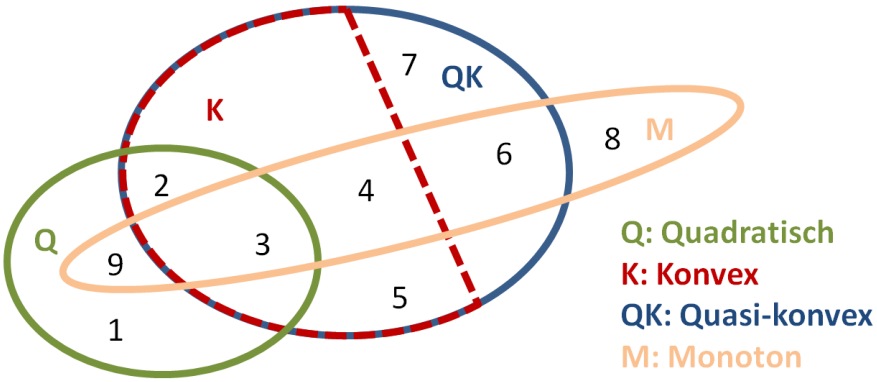
R-Bäume verwenden zur Ermittlung einer Bestenliste von Objekten die Struktur der minimal begrenzenden Rechtecke, um Datenpunkte zu umgeben. Das Branch-and-Bound-Prinzip in seiner

allgemeinen Form, sowie die in dieser Arbeit vorgestellten Ergebnisse zu quasi-konvexen Funktionen, sind aber auch für andere kompakte und konvexe Strukturen geeignet. So lässt sich über andere Formen der Partitionierung nachdenken, wie zum Beispiel der konvexen Hülle von Datenpunkten als zu begrenzende Struktur, die in anderen Teilbereichen, wie z.B. der Geodaten schon Verwendung findet. Wie wir gesehen haben, ist es nicht selten, dass eine spezielle Klasse von Definitionsmengen auch zu einer speziellen Klasse von Funktionen in dem Sinne „passt“, als das sich daraus geeignete Verfahren ableiten lassen.

Weitere Funktionsklassen

Zuletzt stellt sich natürlich grundsätzlich die Frage ob sich, abgesehen von den hier untersuchten Funktionsklassen, noch weitere Klassen finden lassen, die ähnlich schöne und zugleich nützliche Eigenschaften besitzen, um das Branch-and-Bound-Prinzip anwenden zu können. Voraussetzung für eine solche Klasse muss es sein, dass sie auf höchstens endlich vielen Punkten eines Hyperrechteckes maximal (minimal) ist, da sich ansonsten kein effizientes Verfahren zur Berechnung der oberen (unteren) Schranke angeben lässt.

Zum Abschluss sind die drei in dieser Arbeit vorgestellten Funktionsklassen in einem Schema dargestellt, welches zeigt, dass lediglich die Menge der konvexen Funktionen eine echte Teilmenge der quasi-konvexen Funktionen ist und das eine monotone Funktion nicht notwendigerweise quasi-konvex ist (Fall 8 und 9).



	Funktion	Einschränkung	Kommentar
1	$-(x^2+y^2)$	keine	konkav
2	x^2+y^2	keine	
3	x^2+y^2	$x,y > 0$	
4	$x+y$	keine	
5	x^4+y^4	keine	
6	sqrt(max(x,y))	$x,y > 0$	konkav
7	sqrt(max(x,y))	keine	
8	$-(x^4+y^4)$	$x,y < 0$	konkav
9	$-(x^2+y^2)$	$x,y < 0$	konkav

Abbildung 6.1: Quadratisch, Konvex, Quasi-konvex und Monoton

Anhang

Im Folgenden sind die zur Evaluation verwendeten Objekte des relationalen BRS-Modells, die nicht in Kapitel 5 dargestellt wurden, als DDL-Script aufgeführt.

Die Relation *R* stellt die Datenpunkte des Einheitsrechteckes zur Verfügung. Jeder Punkt erhält eine eindeutige Bezeichnung, die *PID*, die zu seiner Identifikation in der Prioritätswarteschlange erforderlich ist.

```
CREATE TABLE [dbo].[R](
	[PID] [varchar](10) NOT NULL,
	[X1] [real] NOT NULL,
	[X2] [real] NOT NULL
) ON [PRIMARY]

GO
```

Abbildung A1: Relation R

Die Relation *RBaum* hält die MBRs des R-Baumes vor. Jedes MBR erhält eine eindeutige Bezeichnung, den Namen (*MBR*), welcher zu seiner Identifikation in der Prioritätswarteschlange erforderlich ist und der von der *PID* verschieden sein muss. Ein MBR wird durch die Koordinaten seiner unteren und seiner obere Ecke aufgespannt.

```
CREATE TABLE [dbo].[RBaum](
	[MBR] [varchar](10) NOT NULL,
	[P1] [real] NOT NULL,
	[P2] [real] NOT NULL,
	[Q1] [real] NOT NULL,
	[Q2] [real] NOT NULL
) ON [PRIMARY]

GO
```

Abbildung A2: Relation RBaum

Die Relation *RBaum_Blaetter* speichert die Blattebene, also die eindeutige Zuweisung jedes Datenpunktes zu einem MBR der letzten inneren Knotenebene. Die Zuordnung erfolgt durch die Paarbildung *PID* zu *MBR*.

```
CREATE TABLE [dbo].[RBaum_Blaetter](
	[PID] [varchar](10) NOT NULL,
	[MBR] [varchar](10) NOT NULL,
	[X1] [real] NOT NULL,
	[X2] [real] NOT NULL
) ON [PRIMARY]

GO
```

Abbildung A3: Relation RBaum_Blaetter

In der Relation *MBR_Grenzen* wird zu jedem MBR der Tabelle *RBaum* die jeweilige obere (untere) Grenze (Wert) gespeichert.

```
CREATE TABLE [dbo].[MBR_Grenzen](
	[MBR] [varchar](10) NOT NULL,
	[Wert] [real] NOT NULL
) ON [PRIMARY]

GO
```

Abbildung A4: Relation MBR_Grenzen

Die Relation *PQ* ist die Prioritätswarteschlange. Das Feld *Datenobjekt* ist entweder die *PID* aus Tabelle *R*, oder der *MBR* aus Tabelle *RBaum*. Im Falle eines Punktes ist *Typ = P*, im Falle eines MBRs ist *Typ = M*.

```
CREATE TABLE [dbo].[PQ](
	[Datenobjekt] [varchar](10) NOT NULL,
	[Wert] [real] NOT NULL,
	[Typ] [char](1) NOT NULL
) ON [PRIMARY]

GO
```

Abbildung A5: Relation PQ

Beim Zugriff auf die Blattebene verwendet der BRS-Algorithmus die Spalte *MBR* der Relation *RBaum_Blaetter*. Da in dieser Relation alle Datenpunkte der Relation *R* vorkommen, sollte diese Spalte bei einer hohen Anzahl von Punkten indiziert werden.

```
CREATE NONCLUSTERED INDEX [NC_RBaum_Blatt] ON [dbo].[RBaum_Blaetter]
(
	[MBR] ASC
)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]

GO
```

Abbildung A6: Index NC_RBaum_Blatt

Die Funktion *getMinScore* ermittelt das Minimum des eindimensionalen Paraboloiden $\omega_i \cdot (x_i - q_i)^2$ auf einem Intervall $[@intervall\_L, @intervall\_R] \subset \mathbb{R}^2$.

```
CREATE FUNCTION [dbo].[getMinScore](@intervall_L real, @intervall_R real, @wi real,
@qi real)
RETURNS real
AS
BEGIN
	DECLARE @rückgabe real, @Wert_intervall_L real, @Wert_intervall_R real
	IF @wi > 0
	BEGIN
		IF @intervall_L <= @qi AND @intervall_R >= @qi
			SET @rückgabe = 0
		ELSE
		BEGIN
```

```
                    SELECT @Wert_intervall_L = @wi * SQUARE(@intervall_L - @qi)
                    SELECT @Wert_intervall_R = @wi * SQUARE(@intervall_R - @q
                    IF @Wert_intervall_L < @Wert_intervall_R
                           SET @rückgabe = @Wert_intervall_L
                    ELSE
                           SET @rückgabe = @Wert_intervall_R
             END
      END
      IF @wi < 0
      BEGIN
             SELECT @Wert_intervall_L = @wi * SQUARE(@intervall_L - @qi)
             SELECT @Wert_intervall_R = @wi * SQUARE(@intervall_R - @qi)
             IF @Wert_intervall_L < @Wert_intervall_R
                           SET @rückgabe = @Wert_intervall_L
                    ELSE
                           SET @rückgabe = @Wert_intervall_R
      END
      RETURN @rückgabe
END
```

Abbildung A7: Funktion zur Bestimmung des Minimums

Die nachfolgende Abbildung zeigt die Bewertungsfunktion(en). Die grün dargestellten Zeilen sind deaktiviert.

```
Create FUNCTION [dbo].[RF](@x1 real, @x2 real)

RETURNS real
AS
BEGIN
      DECLARE @agg real;
      --Quasi-konvexe Funktionen
      --Szenario 1
      SET @agg = SQUARE(@x1-0.5) + @x2
      --Szenario 2
      --SET @agg = SQUARE(@x1-0.5) + SQUARE(@x2-0.5)
      --Szenario 3
      --SET @agg = SQUARE(@x1-0.25) + SQUARE(@x2-0.25)

      --Quadratische Funktionen
      --Szenario 4
      --SET @agg = SQUARE(@x1-0.4) + SQUARE(@x2-0.4)
      --Szenario 5
      --SET @agg = 0.2 * SQUARE(@x1-0.5) + 0.8 * SQUARE(@x2-0.5)
      --Szenario 6
      --SET @agg = 0.2 * SQUARE(@x1-0.5) - 0.8 * SQUARE(@x2-0.5)
      RETURN @agg
END
```

Abbildung A8: Bewertungsfunktion

Die letzte Prozedur *BuildMBR* führt den Median-Splitt für ein MBR aus.

```
CREATE Procedure [dbo].[BuildMBR] (@p1 real, @p2 real, @q1 real, @q2 real, @mbrVater
varchar(10), @tk integer)
AS
BEGIN
```

```
DECLARE @t real, @mbr1 varchar(10), @mbr2 varchar(10), @tiefe as bit
DECLARE @p1neu real, @p2neu real, @q1neu real, @q2neu real

	IF @tk = 1
	BEGIN
		SELECT @t = (@q1 - @p1)/2
		SELECT @q1neu = @p1 + @t
		SELECT @q2neu = @q2
		SELECT @mbr1 = @mbrVater + '0'
		SELECT @p1neu = @p1 + @t
		SELECT @p2neu = @p2
		SELECT @mbr2 = @mbrVater + '1'

		INSERT INTO dbo.[RBaum] (MBR, P1, P2, Q1, Q2)
		VALUES (@mbr1, @p1, @p2, @q1neu, @q2neu)
		INSERT INTO dbo.[RBaum] (MBR, P1, P2, Q1, Q2)
		VALUES (@mbr2, @p1neu, @p2neu, @q1, @q2)
	END

	IF @tk = 2
	BEGIN
		SELECT @t = (@q2 - @p2)/2
		SELECT @q1neu = @q1
		SELECT @q2neu = @p2 + @t
		SELECT @mbr1 = @mbrVater + '0'
		SELECT @p1neu = @p1
		SELECT @p2neu = @p2 + @t
		SELECT @mbr2 = @mbrVater + '1'

		INSERT INTO dbo.[RBaum] (MBR, P1, P2, Q1, Q2)
		VALUES (@mbr1, @p1, @p2, @q1neu, @q2neu)
		INSERT INTO dbo.[RBaum] (MBR, P1, P2, Q1, Q2)
		VALUES (@mbr2, @p1neu, @p2neu, @q1, @q2)
	END
END
```

Abbildung A9: Median-Split-Verfahren

Literaturverzeichnis

[1] T. Härder und E. Rahm, Datenbanksysteme - Konzepte und Techniken der Implementierung, Berlin Heidelberg New York: Springer, 2001.

[2] A. Drozdek, Data Structures and Algorithms in C++, Boston: Cengage Learning, 2012.

[3] Y. Tao, V. Hristidis, D. Papadias and Y. Papakonstantinou, "Branch-and-Bound Processing of Ranked Queries," *Information Systems, 32(3),* pp. 424-445, 2007.

[4] A. Friedman, Partial Differential Equitations of Parabolic Type, New York: Dover Publications, Inc., Mineola, 2008.

[5] R. Sperb, Maximum Principles and their Appications, New York: Academic Press, Inc., 1981.

[6] G. Vossen, Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, Münster: Oldenbourg Verlag, 2008.

[7] C. J. Date, An Introduction to Database System, Boston: Addison Wesley, 2003.

[8] R. Elmasri und S. B. Navathe, Fundamentals of Database Systems, Boston: Prentice Hall, 2016.

[9] H. Garcia-Molina, J. Ullman und J. Widom, Database Systems: The Complete Book, Prentice Hall, 2008.

[10] I. F. Ilyas, G. Beskales and M. A. Soliman, "A Survey of Top-k Query Processing Techniques in Relational Database Systems," *ACM Computing Surveys 40(4), Article 11,* 2008.

[11] G. Das, D. Gunopulos and N. Koudas, "Answering Top-k Queries Using Views," *Proc. of the 32nd international conference on Very Large Data Bases,* pp. 451-462, September 2006.

[12] R. Courant, Differential- und Integralrechnung, Berlin-Heidelberg-New York: Springer-Verlag, 2013.

[13] J. Dieudonné, Grundzüge der modernen Analysis, Bd. 1, Braunschweig: Vieweg, 1985.

[14] R. Dyer and D. Edmunds, From Real to Complex Analysis, Switzerland: Springer, 2014.

[15] T. S. Madhulatha, "An Overview of Clustering Methods," *IOSR Journal of Engineering, Vol. 2(4),* pp. 719-725, April 2012.

[16] R. Engelkind, General Topology, Revised and completed edition, Berlin: Heldermann Verlag, 1989.

[17] H. S. M. Coxeter, Regular Polytopes, New York: Dover Publications, Inc., 1973.

[18] J. L. Taylor, Foundations of Analysis, Rhode Island: American Mathematical Soc, 2012.

[19] E. Brieskorn, Lineare Algebra und Analytische Geometrie, Bd. 1, Braunschweig/Wiesbaden: Vieweg, 1983.

[20] J. Bendoraityte, „R-Baum und seine Spezialisierungen: R*- und R+-Baum," Abteilung Datenbanken der Universität Leipzig, Institut f. Informatik, 2004.

[21] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Conference Boston,* pp. 47-57, 1984.

[22] J. T. Robinson, "The K-D-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes," *Proc. ACM SIGMOD Conference Ann Arbor,* pp. 10-18, 1981.

[23] T. Sellis, R. Roussopoulos and C. Faloutsos, "The R+-tree: A dynamic index for multidimensional objects," *In Proceedings of Very Large Data Bases,* 1987.

[24] S. Timos, N. Roussopoulos und F. Chistos, „The R+-tree: A dynamic index for multidimensional objects," *International Conference of VLDB,* pp. 507-518, 1987.

[25] K.-U. Sattler, G. Saake und V. Köppen, Data Warehouse Technologien, Heidelberg: mitp-Verlag, 2014.

[26] A. Land and A. Doig, "An Automatic Method of Solving Discrete Programming Problems," *Econometrica,* pp. 497-520, July 1960.

[27] R. Darkin, „A tree-search algorithm for mixed integer programming problems," *The Computer Journal 8(3),* pp. 250-255, 1965.

[28] U. Schöning, Algorithmik, Heidelberg: Spektrum Akademischer Verlag, 2001.

[29] M. L. Fredman und R. E. Tarjan, „Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM,* Bd. 34, Nr. 3, pp. 596-615, 1987.

[30] M. L. Fredman, „A Priority Queue Transform," *WAE: International Workshop on Algorithm Engineering LNCS,* Bd. 1668 , pp. 243-257, 1999.

[31] M. L. Fredman, „On the Efficiency of Pairing Heaps and Related Data Structures," *Journal of the ACM,* Bd. 46, pp. 473-501, 1999.

[32] M. L. Fredman und R. E. Tarjan, „Fibonacci Heaps and Their Uses in Improved Optimization Algorithms," *Journal of the ACM,* Bd. 34, pp. 596-615, 1987.

[33] M. L. Fredman, R. Sedgewick, D. D. Sleator und R. E. Tarjan, „The Pairing Heap: A new Form of Self-Adjusting Heap," *Algorithmica,* pp. 111-129, 1986.

[34] N. Roussopoulos, S. Kelly and F. Vincent, "Nearest Neigbor Queries," *Proc. ACM SIGMOD Conference New York,* pp. 71-79, 1995.

[35] I. Schmitt, Ähnlichkeitssuche in Multimedia-Datenbanken: Retrieval, Suchalgorithmen und Anfragebehandlung, München: Oldenbourg Wissenschaftsverlag, 2005.

[36] G. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM TODS, 24(2),* pp. 265-318, Juni 1999.

[37] S. Berchtold, C. Böhm, D. Keim und H. Kriegel, „Cost Model for Nearest Neighbor Searchin High-Dimensional Data Spaces," *Proc. ACM PODS,* pp. 78-86, 1997.

[38] B. Börzsönyi, D. Kossmann and K. Stocker, "The Skyline Operator," *International Conference of Data Engineering,* pp. 241-430, 2001.

[39] D. Papadias, Y. Tao, K. Mouratidis and C. K. Hui, "Aggregate Nearest Neighbor Queries in Spatial Databases," *ACM TODS 30(2),* pp. 529-576, 2005.

[40] D. Papadias, Y. Tao, G. Fu und B. Seeger, „An Optimal and Progressive Algorithm for Skyline Queries," *ACM SIGMOD,* pp. 467-478, 2003.

[41] V. Hristidis and Y. Papakonstantinou, "Algorithm and Application for answering Ranked Queries using Ranked Views," *The VLDB Journal, 13(1),* pp. 49-70, 2004.

[42] Y. Chang, L. Bergmann, V. Castelli, C. Li, M. Lo and J. Smith, "The Union Technique: Indexing for Linear Optimization Queries," *Proc. ACM SIGMOD Conference Dallas,* pp. 391-402, 2000.

[43] P. M. Gruber and J. M. Wills, Handbook of Convex Geometry, Vol.A+B, North Holland: Elsevier, 1993.

[44] K. Leichtweiß, Konvexe Mengen, Berlin, Heidelberg, New York: Springer, 1980.

[45] S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge: Cambridge University Press, 2004.

[46] B. Amitabh and T. Oertel, "Centerpoints: A Link between Optimization and Convex Geometry," *SIAM Jornal of Optimization 27.2,* pp. 866-889, 2017.

[47] V. Sverák, "New Examples of Quasiconvex Functions," *Archive for rational mechanics and analysis 119.4,* pp. 293-300, 1992.

[48] O. Forster, Differenzialrechnung im n-dimensionalen Euklidischen Raum, gewöhnliche Differenzialgleichungen, Bd. 8, Wiesbaden: Vieweg+Teubner Verlag, 2008.

[49] E. Brieskorn und H. Knörrer, Ebene algebraische Kurven, Basel - Boston - Stuttgart: Birkhäuser, 1981.

[50] S. Ranu and A. Singh, "Answering Top-k Queries Over a Mixture of Attractive and Repulsive Dimensions," *Proc. of the VLDB Endowment 5(3),* pp. 169-180, 2011.

[51] D. Xin, J. Han, H. Chang and L. X., "Answering Top-k Queries with Multi-Dimensional Selections: The Ranking Cube Approach," *Proc. of the VLDB Endowment 5(3),* 2006.

[52] N. Bruno, S. Chaudhuri and L. Gravano, "Top-k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation," *ACM Transactions on Database Systems 27(2),* pp. 153-187, June 2002.

[53] B. Nevarez, Microsoft Sql Server 2014 Query Tuning & Optimization, New York: McGraw-Hill Education, 2014.

[54] A. K. Mann and N. Kaur, "Survey Paper on Clustering Techniques," *International Journal of Science, Engineering and Technology Research (IJSETR), 2(4),* April 2013.

[55] K. Mehlhorn and P. Sanders, Algorithms and Data Structures, Karlsruhe, Saarbrücken: Springer, 2007.

[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introductions to Algorithms, Third Edition, Cambridge, London: The MIT Press, 2009.

[57] D. Mertins, J. Neumann und A. Kühnel, SQL Server 2016, Bonn: Rheinwerk Computing, 2017.

[58] R. Fagin, A. Lotem and M. Naor, "Optimal Aggregation Algorithms for Middleware," *Proc. of the 20th ACM SIGMOD Symposium on Principles of Database Systems,* pp. 102-113, 2001.

[59] A. Marian, N. Bruno und L. Gravano, „Evaluating top-k queries over web-accessible databases," *ACM Transactions on Database Systems,* Nr. 29, pp. 319-362, 2004.

[60] U. Güntzer, W. Balke und W. Kiessling, „Towards efficient multi-feature queries in heterogeneous environments," *In Proc. of the International Conference on Information Technology: Coding and Computing,* 2001.

[61] Z. Zhang, S. Hwang, K. Chang, M. Wang, C. A. Lang and Y. Chang, "Boolean + Ranking: Querying a Database by K-Constrained Optimization," *Proc. ACM SIGMOD International Conference on Management of Data,* pp. 359-370, 2006.

[62] D. Xin, J. Han and K. Chang, “Progressive and Selective Merge: Computing Top-k with Ad-hoc Ranking Functions,” *Proc. ACM SIGMOD International Conference on Management of Data,* pp. 103-114, June 2007.

[63] N. Madrid and U. Straccia, “On Top-k Retrieval for a Family of Non-monotonic Ranking Funktions,” *Flexible Query Answering Systems,* pp. 507-518, 2013.

Definitionsverzeichnis

Abbildungsverzeichnis