Erweiterungen des R-Baums für räumliche Datenbankanfragen

Der R*-Baum

Patrick Schulz & Simon Hötten

Seminar Geodatenbanken
Dozent: Prof. Dr.-Ing. Jan-Henrik Haunert
Institut für Geoinformatik und Fernerkundung
Universität Osnabrück
Sommersemester 2015

Schlüsselwörter: Geodatenbanken, R*, Spatial Access, Index, R-Tree

1 Motivation

Herkömmliche eindimensionale Indexstrukturen bieten keine Möglichkeiten, mehrdimensionale räumliche Daten effizient zu durchsuchen. Die Reduzierung auf Punkte, um Objekte mit Point access methods (PAM) abzufragen, ist mit gewissen Einbußen möglich, aber insbesondere für komplexere Anfragen unzureichend. Der 1984 von Guttman entwickelte R-Baum (Guttman, 1984) versucht dieses Problem zu lösen, in dem der Index direkt auf den räumlichen Eigenschaften basiert. Mittlerweile existieren unzählige Varianten und Verwandte des R-Baums, dessen Einsatzgebiet weit über die klassische Geoinformatik hinaus geht.

Eine dieser Varianten ist der R*-Baum, welcher die (teils unbegründeten) Annahmen in der ursprünglichen Veröffentlichung hinterfragt und so die Datenstruktur weiter optimiert. Im Folgenden gehen wir auf die Verfahren und Eigenheiten des regulären R-, als auch des R*-Baums ein, stellen allgemeine Optimierungskriterien auf und schließen mit einem Vergleich.

Im Prinzip ist der R*-Baum für n-dimensionale Daten geeignet. Diese Arbeit beschränkt sich allerdings auf Geodaten, insbesondere sind alle Beispiele im zwei dimensionalen Raum. Hier liegt auch das Haupteinsatzgebiet von R*-Bäumen. Für höher-dimensionale Daten sind andere Indexstrukturen, wie der X-Baum, zu bevorzugen (vgl. Berchtold u. a., 1996, S. 28-29).

2 Prinzipien eines R-Baums

Der R-Baum ist eine räumliche Indexstruktur für die effiziente Bereichsabfragen von Geometrien im zweidimensionalen Raum. Der R-Baum teilt in jeder Ebene seiner Struktur die beinhalteten Geometrien in Partitionen auf. Die Gesamtheit der Geometrien einer Partitionen werden durch minimal umschließende Rechtecke (kurz MBR) repräsentiert. Innerhalb einer Partition werden die Geoemtrien in

weitere Partitionen unterteilt, bis die Anzahl der Geometrien den Schwellwert M innerhalb einer Partition nicht mehr überschreitet. Der Vorteil von dieser Strukturierung sorgt dafür, dass bei Bereichabfragen - zB. Was befindet sich in der Umgebung von Polygon p? - nicht zwingend alle Geometrien in der Ebene betrachtet werden müssen: Die Abbildung 3 stellt einen solchen R-Baum dar, wo bei einer Bereichsabfrage im besten Fall nach der ersten Entscheidungsebene die Hälfte des Geometriebestandes nicht mehr betrachtet werden muss. Die Verwendung von achsparallelen MBR sorgt für effiziente Verschneidungs-Abfragen. Durch das alleinige Vergleichen der Koordinaten kann ermittelt werden, ob zwei MBR sich schneiden oder nicht. Die Geometrien selbst werden ebenfalls durch MBR repräsentiert, sind allerdings zur Bewahrung der Übersichtlichkeit in der Abbildung 3 nicht dargestellt.

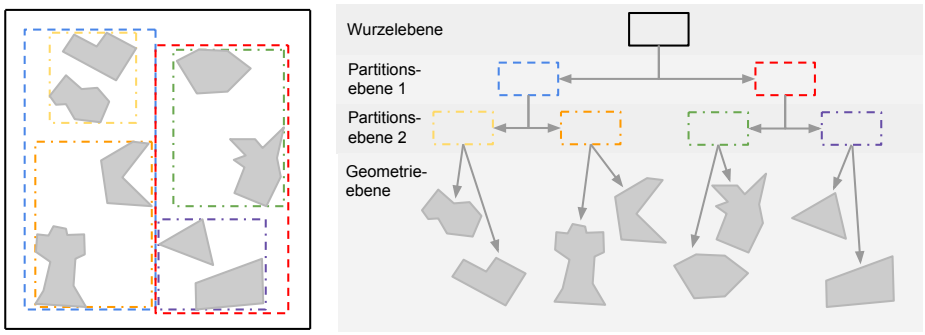

Abb. 1. Beispiel für einen R-Baum. Links: Verteilung der Polygone und Partitionen im zweidimensionalen Raum. Rechts: Die dazugehörige Baumstruktur. (M=3, m=2)

Der Aufbau eines R-Baums verläuft iterativ, wo jedes einzelne Polygon separat nach einem Schema eingefügt wird. Der komplette Prozess des Einfügens lässt sich in zwei Methoden beschreiben, der Author bezeichnet diese Methoden als ChooseSubtree() und QuadraticSplit(). Beide Methoden werden in den nächsten Absätzen beschrieben. Abhängig in welcher Reihenfolge die Polygone in den R-Baum eingesetzt werden kommt es zu verschiedenen Verteilungen innerhalb des R-Baumes. Diese unterschiedlichen Verteilungsmöglichkeiten sind nicht immer ideal und sorgen für weniger effizientere Abfragen, wo durch eine andere Verteilung eine bessere Performance erzielt werden könnte. Die Polygone in der Abbildung 3 sind im R-Baum zur Veranschaulichung ideal verteilt. Eine Lösung für dieses Problem wird in Kapitel ?(ForcedReInsert)? behandelt und ist Teil der Verbesserungen im R*-Baum.(Beispiel?)

2.1 ChooseSubTree()

ChooseSubTree() ist eine rekursive Methode, die für das einzufügende Polygon die paasende Partition in der aktuellen Baum-Ebene zufinden, welches das Polygon daraufhin zugeordnet werden soll. Die Prozedur wird bis zur untersten Baum-Ebene fortgesetzt. Abbildung 2 zeigt eine Situation in der untersten Baumebene mit zwei Partitionen.

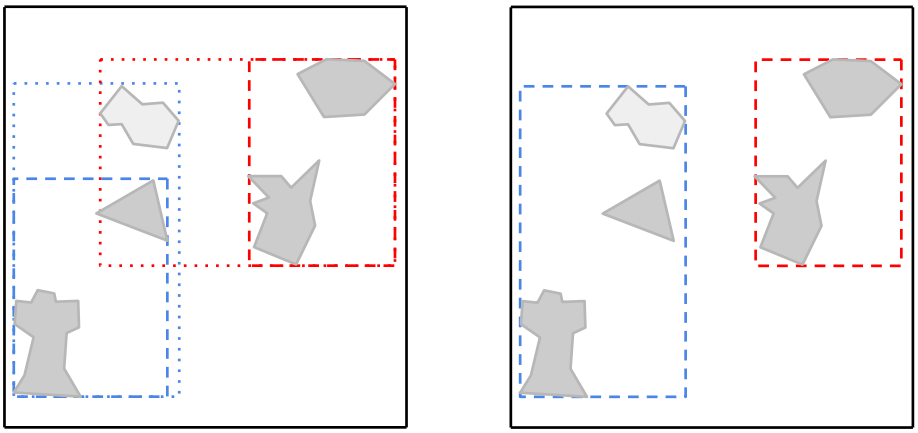

Abb. 2. ChooseSubTree()-Szenario. Links: Vergleich des Flächenzuwachses der Partitionen bei Aufnahme des Polygons. Rechts: Polygon wird der Partition zugeordnet, die den geringen Flächenzuwachs erfährt. (M=4, m=2)

2.2 QuadraticSplit()

QuadraticSplit() wird aufgerufen, sobald innerhalb einer Partition die Anzahl der Geometrien den Wert M überschreitet. Wenn der Fall eintritt, muss der Inhalt der überfüllten Partition in zwei neue Partition aufgeteilt werden.

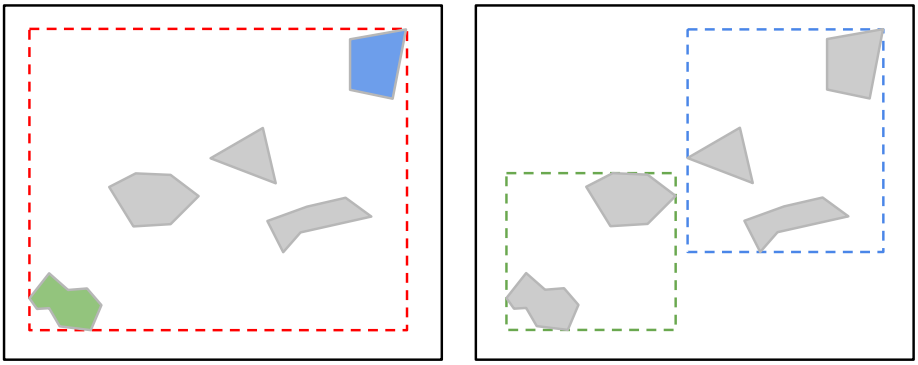

Abb. 3. Split()-Szenario. Links: Überfüllte Partition und gewähltes Polygonpaar, welche eine neue Partition repräsentieren. Rechts: Aufteilung der Polygone in die zwei neuen Partitionen. (M=4, m=2)

Verfahren...

Der Grund, wieso die Methode QuadraticSplit() heißt und nicht einfach Split(), liegt daran, dass verschiedene Umsetzungen für die Aufteilung existieren, die eine Unterschiedliche Performance aufweisen. QuadraticSplit() deutet darauf hin, dass die Methode abhängig von der Anzahl Polygone eine quadratische Laufzeit besitzt. Neben dem QuadraticSplit() existieren noch der LinearSplit(), QubicSplit() und der GreenesSplit(). Der LinearSplit() unterscheidet sich vom QuadraticSplit() nur daran, dass dieser die Sortierung der ausstehenden Polygone nur einmal am Anfang durchführt. Dies sorgt zwar für eine bessere Laufzeit, die daraus resultierenden Aufteilungen werden im Vergleich zum QuadraticSplit() allerdings nie besser sein. Der CubicSplit hingegen verspricht, dass nach dem Flächenkriterium die besten Aufteilungen generiert wird auf Kosten der längeren Bearbeitungszeit. Der QuadraticSplit ist somit ein Kompromiss zwischen Effizienz und Qualität.

2.3 GreenesSplit()

In einem Paper von "Greenes"wird ein alternatives Verfahren für die bisher bekannten Split-Methoden vorgestellt. Die Idee besteht darin, die Polygone nach der jeweiligen Achse zu sortieren, in der die größte Ausbreitung ermittelt wird. Auf Basis dieser Sortierung hat man nun eine Reihe Splits, aus dem man dann den Favoriten zu ermitteln hat und übernimmt.

3 Optimierungskriterien

Bei dem herkömmlichen R-Baum wird, sowohl beim Hinzufügen neuer Elemente als auch beim Split, lediglich die Fläche der umschließenden Rechtecke minimiert

(vgl. Guttman, 1984, S. 50-51). Einige der daraus resultierenden Probleme wurden bereits im vorherigen Abschnitt dargelegt. Im Folgenden werden weitere mögliche Optimierungen und ihre Wechselwirkungen aufgeführt. Ein R-Baum muss mit unterschiedlichen Geometrien und Anfragen umgehen können, daher wirken sich einige Kriterien in einigen Situationen stärker aus als andere.

Flächenausnutzung maximieren

Die Fläche, welche von dem umschließenden Rechteck, aber nicht von den in ihm enthaltenen Rechtecken, überdeckt wird, soll minimiert werden. Es soll also möglichst wenig Platz „verschwendet“ werden. (vgl. Beckmann u. a., 1990, S. 323)

Überlappung minimieren

Die Überlappung der umschließenden Rechtecke soll minimiert werden. Dadurch müssen ebenfalls weniger Pfade im Baum traversiert werden. Liegt ein angefragter Punkt beispielsweise in einer Region, in der sich viele Rechtecke Überschneiden, müssen alle Möglichkeiten weiter verfolgt werden, was zu erhöhtem Rechenaufwand führt.

Summe der Kantenlänge minimieren

Die Summe der Kantenlänge der Verzeichnisrechtecke (*„margin“*) soll möglichst klein sein. Quadrate werden also bevorzugt. Da Quadrate auf den jeweils höheren Ebenen im Baum besser zusammengefasst werden können, reduziert sich so die benötigte Fläche. Außerdem profitieren Anfragen mit großen, quadratischen Elementen von dieser Optimierung. (vgl. ebd., S. 323)

Speichernutzung maximieren

Eine geringe Höhe des Baumes wirkt sich positiv auf die Kosten einer Abfrage aus. Das kann durch eine möglichst gleichmäßige Verteilung der Blattknoten erreicht werden. Insbesondere für große Abfragerechtecke ist dies relevant, da hier, auch abgesehen von den ersten drei genannten Optimierungen, mehrere Pfade traversiert werden müssen. (vgl. ebd., S. 323-324)

Wechselwirkungen

Um die Flächenausnutzung zu maximieren und die Überlappung zu minimieren, bedarf es einer größeren Freiheit bei der Wahl der Formen und der Anzahl an Rechtecken pro Knoten. Die Kriterien stehen also in Konkurrenz mit einer geringen Kantenlänge und hohen Speicherausnutzung. Auf der anderen Seite können quadratischere Rechtecke besser zusammengefasst werden, was sich wiederum positiv auf die Speichernutzung auswirkt. (vgl. ebd., S. 323-324)

4 Der R*-Baum

5 Fazit

Zunächst lässt sich festhalten, dass der R*-Baum alle vorgestellten Optimierungskriterien berücksichtigt. Das hat einen etwas erhöhten Implementierungsaufwand gegenüber herkömmlichen R-Bumen zur Folge.

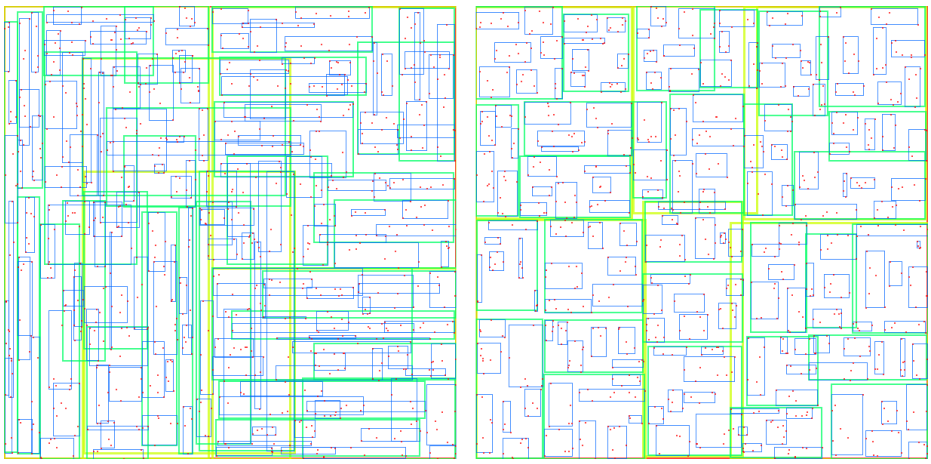

Abb. 4. Quadratischer- (links) und R*-Split (rechts) im Vergleich. (Bildquelle: `https://github.com/davidmoten/rtree`)

Der R*-Baum betreibt sehr hohen Aufwand beim Hinzufügen und Löschen von Elementen, um eine gute Struktur zu bewahren. Abbildung 4 zeigt den gleichen Datensatz (mit gleicher Einfügereihenfolge), aber unterschiedlichen Split-Verfahren. Das Beispiel zeigt, wie positiv sich der Mehraufwand auswirkt. Mit R*-Split Überschneiden sich die Verzeichnisrechtecke deutlich weniger, was schnellere Abfragen ermöglicht.

Benchmarks

Weiterentwicklungen

Die Effizienz des R*-Baums nimmt ab fünf Dimensionen rapide ab (vgl Kriegel u. a., 2008, S. 29). Um auch höher dimensionalen Daten gerecht zu werden, existieren daher zahlreiche Weiterentwicklung. Dazu gehört, wie eingangs erwähnt, der X-Baum, welcher darauf ausgelegt ist auch in höheren Dimensionen Überlappungen zu vermeiden (Berchtold u. a., 1996, vgl.). Andere Indizes bilden Näherungen der tatsächlichen Daten und führen Anfragen zunächst auf diesen aus (*„Vector Approximation“*, siehe Gibas und Ferhatosmanoglu, 2008 oder Daoudi u. a., 2008).

Anhang

Abkürzungsverzeichnis

m	Variable: minimale Anzahl der Geometrien in einer Partition
M	Variable: maximale Anzahl der Geometrien in einer Partition
MBR	Minimum bounding Rectangle
SAM	Spatial access methods
PAM	Point access methods

Literatur

Beckmann, Norbert, Hans-Peter Kriegel, Ralf Schneider und Bernhard Seeger (1990). „The R*-tree: An Efficient and Robust Access Method for Points and Rectangles“. In: *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. SIGMOD '90. Atlantic City, New Jersey, USA: ACM, S. 322–331. DOI: 10.1145/93597.98741 (siehe S. 5).

Berchtold, Stefan, Daniel A. Keim und Hans-Peter Kriegel (1996). „The X-tree: An Index Structure for High-Dimensional Data“. In: *Proceedings of the 22nd VLDB Conference*. Mumbai, India, S. 28–39 (siehe S. 1, 6).

Daoudi, I., S.E. Ouatik, A. El Kharraz, K. Idrissi und D. Aboutajdine (2008). „Vector Approximation based Indexing for High-Dimensional Multimedia Databases“. In: *Engineering Letters* 16.2 (siehe S. 6).

Gibas, MICHAEL und Hakan Ferhatosmanoglu (2008). „High Dimensional Indexing“. In: *Encyclopedia of GIS*. Springer US, S. 502–507. DOI: 10.1007/978-0-387-35973-1_1148 (siehe S. 6).

Guttman, Antonin (1984). „R-trees: a dynamic index structure for spatial searching“. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. SIGMOD '84. New York, NY, USA: ACM, S. 47–57. DOI: 10.1145/602259.602266 (siehe S. 1, 5).

Kriegel, Hans-Peter, Peter Kunath und Matthias Renz (2008). „R*-Tree“. In: *Encyclopedia of GIS*. Springer US, S. 987–992. DOI: 10.1007/978-0-387-35973-1_1148 (siehe S. 6).