Persistente Speicherung von Geo-Daten auf mobilen Endgeräten

CARSTEN KLEINER[1] & ALEXANDER OLBRICHT[1]

Zusammenfassung: Für anspruchsvolle, effiziente ortsbasierte Anwendungen auf mobilen Endgeräten wie Smartphones oder Tablets bietet es sich an, die erforderlichen Geo-Daten lokal auf dem Gerät vorzuhalten, um einerseits eine Offline-Nutzung der Anwendung zu ermöglichen und andererseits die Bandbreite zu schonen sowie die Antwortzeit zu optimieren. In diesem Artikel wird daher untersucht, welche konzeptuellen und technologischen Erweiterungen erforderlich sind, um eine Verwaltung von Geo-Daten auf aktuellen mobilen Plattformen zu ermöglichen. Für verschiedene Realisierungsvarianten werden aus theoretischer Sicht Funktionen zur Berechnung der Antwortzeit für räumliche Selektionen hergeleitet. Ferner werden dann einige dieser Varianten auf den Plattformen Android und Windows Phone auf Basis eines realisierten Prototyps evaluiert. Es zeigt sich, dass sich für Liniengeometrien auf Android mit spatialite räumliche Selektionen in akzeptabler Zeit auch für recht große Datenmengen realisieren lassen. Dazu ist die Verwendung eines räumlichen Index unerlässlich. Auch für Windows Phone konnte die prinzipielle Machbarkeit nachgewiesen werden. Die Einbindung entsprechender Funktionalität in eigene Apps ist bei beiden sehr einfach möglich. Insgesamt zeigt sich, dass bereits heute die lokale Speicherung und Abfrage von Geo-Daten auf mobilen Geräten mit vertretbarem Aufwand möglich ist, sofern die Datenmengen innerhalb bestimmter Grenzen (bspw. alle Straßen Bayerns) bleiben.

1 Einleitung und Motivation

Mobile Endgeräte wie z.B. Smartphones haben in den letzten Jahren zunehmend alle privaten Lebensbereiche durchdrungen, immer öfter werden diese auch im Unternehmensbereich verwendet. Dabei werden technologisch im Bereich der mobilen Endgeräte bisher häufig für jede Plattform spezifisch entwickelte Anwendungen (so genannte Apps) eingesetzt; ein Übergang zu reinen Web-Anwendungen, die auf allen Plattformen lauffähig wären, findet eher langsam statt. Das liegt sicher auch darin begründet, dass diese meist nur mit funktionierender Internet-Verbindung sinnvoll nutzbar sind. Dies ist jedoch zurzeit noch nicht immer gegeben.

Für anspruchsvolle, effiziente ortsbasierte Anwendungen auf mobilen Endgeräten wie Smartphones oder Tablets bietet es sich daher an, die erforderlichen Geo-Daten (oder zumindest einen relevanten Ausschnitt davon) lokal auf dem Gerät vorzuhalten, um einerseits eine Offline-Nutzung der Anwendung zu ermöglichen und andererseits die Bandbreite zu schonen sowie die Antwortzeit zu optimieren. Aktuelle Plattformen mobiler Endgeräte bieten oft bereits abgespeckte Versionen SQL-basierter Datenbanksysteme für eine lokale, persistente Speicherung von Daten an. Allerdings fehlt diesen Systemen zumeist die Möglichkeit zur

[1] Carsten Kleiner, Alexander Olbricht, Hochschule Hannover, Fakultät IV (Abt. Informatik), Ricklinger Stadtweg 120, 30459 Hannover; E-Mail: carsten.kleiner@hs-hannover.de

effizienten Speicherung und Abfrage von Geo-Daten, etwa zweidimensionaler Daten nach dem Simple Feature Modell. Ferner ergibt sich auf einem mobilen Endgerät eine besondere Situation, da die Datenbank zumeist ohne ein vollwertiges Datenbankmanagementsystem zum Einsatz kommt. Dies hat einen wesentlichen Einfluss auf die Speicherung der Geo-Daten, da bekannte Verfahren nur in abgewandelter Form eingesetzt werden können.
Rein-relationale Datenbanksysteme sind aufgrund der nur elementaren Datentypen bekanntermaßen nicht für die Verwaltung von Geo-Daten geeignet. Daher ist eine Erweiterung der Standardfunktionalitäten erforderlich. In diesem Artikel wird untersucht, welche konzeptuellen und technologischen Erweiterungen erforderlich sind, um eine Verwaltung von Geo-Daten auf aktuellen mobilen Plattformen zu ermöglichen. Dabei wird eine Teilmenge des zwei-dimensionalen Simple Feature-Modells für die Datentypen und zugehörigen Operationen verwendet. Besondere Beachtung bei der Auswertung wird der effizienten Bearbeitung anspruchsvoller geometrischer Operationen (bspw. räumliche Selektion) auf den Geo-Daten auf dem mobilen Endgerät gewidmet. Konkret wurde zum einen der Einsatz der Bibliothek spatialite für Android (eine Erweiterung der sqlite-Datenbank für Android um geometrische Datentypen und Operationen) untersucht. Zum Vergleich wurde ferner eine Eigenimplementierung für geometrische Typen und Operationen auf Basis von LINQ to SQL für Windows Phone herangezogen.
Die Evaluation zeigt, dass bei Verwendung einer räumlichen Datenbank mit räumlichen Indexen durchaus auf aktuellen Android-Geräten Datenmengen bis zu etwa 500.000 Liniengeomtrien noch effizient genutzt werden können.
Nach einer kurzen Übersicht über verwandte Arbeiten in Kapitel 2 wird die Speicherung von Geo-Daten auf mobilen Endgeräten konzeptionell untersucht. Es werden in Kapitel 3 Unterschiede zu herkömmlichen Geräten herausgearbeitet und mögliche Realisierungsvarianten vorgestellt. Für diese werden dann Laufzeitfunktionen für räumliche Selektionen theoretisch hergeleitet. In Kapitel 4 werden darauf Erfahrungen bei der Realisierung zweier dieser Varianten auf den Mobilplattformen Android und Windows Phone präsentiert, bevor der Artikel in Kapitel 5 mit einem Fazit und Ausblick auf weitere interessante Fragestellungen schließt.

2 Verwandte Arbeiten

Die lokale Speicherung von Daten auf mobilen Endgeräten war Thema mehrerer älterer Publikationen aus der Zeit, als mobile Geräte nur sehr schwache Internetverbindungen aufwiesen. Auch heute in Zeiten besserer Verbindungen wird jedoch für einige Anwendungsbereiche eine lokale Speicherung vorgeschlagen, etwa in COELHO, P. & AGUIAR, A. & LOPES, J.C. Mobile Anwendungen für räumliche Daten werden ebenfalls in etlichen Veröffentlichungen beschrieben; die meisten allerdings ohne lokale Datenspeicherung bzw. Anfrageverarbeitung auf dem Endgerät; eine Ausnahme bildet hier etwa GARDINER, K. & YIN, J. & CARSWELL, J. D.
Arbeiten, die als Kernthema die effiziente, lokale Speicherung von Geo-Daten auf mobilen Geräten haben, sind bisher rar. Die einzige bekannte Veröffentlichung dazu ist JACOB, R. & SMITHERS, S. & WINSTANLEY, A. C. Der wesentliche Unterschied zwischen dieser Arbeit und

dem vorliegenden Artikel ist, dass dort nur punktförmige Geometrien und die „N nächste Nachbarn“-Anfrage behandelt werden. Dies hat zur Folge, dass auf Punkten alle geometrischen Operationen in O(1) realisiert werden können und damit die Zahl der Kandidatenobjekte keine so große Rolle spielt wie in unserem Fall. Die in Abschnitt 3 hergeleiteten Funktionen vereinfachen sich dann entsprechend. Als technologische Grundlage wurde dort wie auch hier in Abschnitt 4.1 Android mit Spatialite verwendet, so dass unsere Arbeit die Erkenntnisse des Artikels um die Behandlung linienförmiger Geometrien mit räumlichen Selektionen ergänzt.

Räumliche Zugriffsstrukturen wie der R-Baum (GUTTMAN, A.) wurden auf stationären IT-Systemen bereits seit langem vorgeschlagen und werden für spezielle Anwendungen immer weiter optimiert. Eine Übersicht über die wichtigsten Varianten bis zum Veröffentlichungstermin bietet GAEDE, V. & GÜNTHER, O. Dort werden auch unterschiedliche Strukturen für Haupt- und Sekundärspeicherzugriff vorgeschlagen, die in diesem Artikel noch referenziert werden.

Der Einfluss der gewählten Speichertechnologie auf mobilen Endgeräten auf die Anfrageperformanz wird auch in KIM, H. & AGRAWAL, N. & UNGUREANU, C. thematisiert. Die Bedeutung einer spezifischen an die Speicherhardware angepassten Implementierung räumlicher Indexe wurde auch in WU, C.-H. & CHANG, L.-P. & KUO, T.-W. beschrieben. Dies sollte im Bereich der künftigen Arbeiten auch für die Speicherung mit Spatialite geprüft werden. Die Bearbeitung von Anfragen im Kontext der ortsbasierten Dienste, insbesondere bei sich bewegenden Objekten wird in zahlreichen Veröffentlichungen besprochen, eine Übersicht findet sich z. B. in ILARRI, S. & MENA, E. & ILLARRAMENDI, A. Allerdings werden hier zumeist nur punktförmige Geometrien benötigt.

Arbeiten zum Vergleich der SW-Entwicklung auf verschiedenen mobilen Plattformen sind aufgrund der großen Dynamik des Marktes sehr starken Veränderungen unterworfen. Diese finden sich daher zumeist eher in nicht-wissenschaftlichen Quellen im Internet. Eine Ausnahme bildet GRØNLI, T. M. & HANSEN, J. & GHINEA, G.; dieser Artikel ist aber heute bereits in großen Teilen als veraltet anzusehen, da einige der Plattformen in der Praxis kaum noch eine Rolle spielen.

3 Konzepte zur Speicherung von Geo-Daten auf mobilen Geräten

Bei der klassischen Verwaltung von Geo-Daten in Softwaresystemen kommt zumeist ein Geo-Datenbankserver mit Datenbankmanagementsystem zum Einsatz. Hier werden die Geo-Daten mit zugehörigen räumlichen Indexstrukturen abgespeichert. Im Falle einer Anfrage mit geometrischen Operationen (räumliche Selektion oder räumlicher Verbund) wird dabei eine zweistufige Anfragebearbeitung realisiert: zunächst werden basierend auf Objektapproximationen (mit eher geringem Platzbedarf) Ergebniskandidaten ermittelt. Die Prüfung, ob ein Datensatz ein Kandidat ist, benötigt dabei nur O(1) Rechenzeit und kann daher wie für herkömmliche Datentypen erfolgen. Für die ermittelten Kandidaten wird dann der zumeist aufwändige geometrische Algorithmus verwendet, um die tatsächliche Treffermenge zu ermitteln. Dazu müssen die Kandidatenobjekte vollständig in den Hauptspeicher des Servers geladen werden; dies ist aufgrund der Größe der Datensätze sowie des langsamen Zugriffs auf Sekundärspeicher eine teure Operation. Die gesamte Anfragebearbeitung findet dabei auf dem

Server statt, nur die tatsächlichen Ergebnisse werden an den Anfrager übermittelt. Insgesamt ergibt sich damit die folgende Laufzeit für eine räumliche Selektion:

(1) T_{stat} = n*load(approx)*O(1) + k*load(geoobj)*O(geoalgo) + e*trans(geoobj)

Hierbei bezeichne n die Anzahl der Geoobjekte in der Datenbank, k die Anzahl der Ergebniskandidaten und e die Anzahl der Ergebnisse. Der erste Term kann dabei durch den Einsatz eines räumlichen Index im Filterschritt deutlich reduziert werden auf:

(2) $T_{statIdx}$ = k*log(n)*load(id)*O(1) + k*load(geoobj)*O(geoalgo) + e*trans(geoobj)

Im Falle der persistenten Speicherung der Geo-Daten auf einem mobilen Gerät ergeben sich selbst beim Einsatz eines lokalen Geo-Datenbanksystems einige Änderungen. Zum einen liegt die Datenbank lokal im Filesystem vor (zumeist als eine einzige Datei). Insofern finden die geometrischen Berechnungen zur Ermittlung der Ergebnisse im Hauptspeicher des Gerätes statt und die Zeit für die Übermittlung der Ergebnisse (dritter Term in (1)) entfällt. Eine zweistufige Ergebnisberechnung mit räumlichem Index ist dennoch sinnvoll, um die aufwändige geometrische Berechnung nur für möglichst wenige Kandidatenobjekte durchführen zu müssen (insbesondere bei der im Vergleich zu einem DB-Server geringen Rechenleistung eines Mobilprozessors). Auch das Laden der Objekte ist signifikant, da die Datenbank auf einem mobilen Gerät zumeist auf einer internen oder externen Speicherkarte abgelegt ist, die relativ lange Zugriffszeiten erfordert. Man erhält also:

(3) T_{mobDB} = n*load(approx)*O(1) + k*load(geoobj)*O(geoalgo)

bzw. mit Verwendung eines räumlichen Index:

(4) $T_{mobDbIdx}$ = k*log(n)*load(id)*O(1) + k*load(geoobj)*O(geoalgo)

Bei besonders einfach aufgebauten mobilen Plattformen kann es sogar vorkommen, dass beim Programmstart die gesamte Datenbankdatei in den Hauptspeicher des mobilen Geräts geladen werden muss und die gesamte Anfragebearbeitung dann basierend auf Daten im Hauptspeicher stattfindet. Selbst dann macht aber eine Vorfilterung der Kandidaten basierend auf Approximationen noch Sinn, denn die eher leistungsschwachen Mobilprozessoren sollten für möglichst wenige Objekte die aufwändigen geometrischen Operationen berechnen. Man erhält:

(5) T_{mobHS} = n*load(geoobj)*O(1) + k*O(geoalgo)

Dies dürfte zumeist deutlich unter der Zeit n*load(geoobj) + n*O(geoalgo) liegen, da die geometrischen Algorithmen oft eine Laufzeit von O(m log m) erfordern, wobei m die Zahl der Stützpunkte des zu prüfenden Objekts ist.

Natürlich ist im letzten Fall die maximale Größe der lokalen Geo-Datenbank deutlich geringer, da sie durch den zur Laufzeit zur Verfügung stehenden Hauptspeicher und nicht die Größe einer Speicherkarte begrenzt ist.

Für die im folgenden Kapitel beschriebenen prototypischen Implementierungen lokaler persistenter Speicherung von Geo-Daten auf mobilen Endgeräten ergeben sich also die folgenden prinzipiellen Optionen:

A. Verwendung einer lokalen räumlichen Datenbank mit räumlichen Indexen und zweistufiger Anfragebearbeitung unter Nutzung des Index (vgl. (4))
B. Verwendung einer lokalen räumlichen Datenbank ohne räumliche Indexe und zweistufiger Anfragebearbeitung (vgl. (3))

C. Verwendung einer lokalen nicht-räumlichen Datenbank mit Eigenimplementierung der zweistufigen Anfragebearbeitung und der räumlichen Operationen (hierfür kann ggfs. eine Bibliothek verwendet werden) (vgl. (3))
D. Verwendung einer lokalen Hauptspeicherdatenbank mit Eigenimplementierung der zweistufigen Anfragebearbeitung im Hauptspeicher (vgl. (5))
E. Verwendung einer lokalen räumlichen Datenbank ohne räumliche Indexe und mit direktem Aufruf der geometrischen Operationen (für nicht-punktförmige Geometrien ist mit sehr langen Laufzeiten zu rechnen; T = n*load(geoobj) + n*O(geoalgo))

Aus konzeptioneller Sicht ist offenbar die Variante A zu präferieren; diese bedingt allerdings die Verfügbarkeit eines entsprechenden Datenbanksystems auf der zu verwendenden Plattform.

4 Implementierungsaspekte

In diesem Abschnitt sollen aus den zuvor herausgearbeiteten Realisierungsmodellen zur lokalen Speicherung von Geo-Daten auf mobilen Endgeräten für zwei konkrete Plattformen (Android, Windows Phone) die zur Zeit möglichen ausgewählt und in einem Prototyp untersucht werden.

4.1 Geo-Daten auf Android-Geräten

Die aktuell am weitesten verbreitete mobile Plattform ist Android. Diese Plattform bringt für die persistente Speicherung von Daten auf dem mobilen Gerät von Hause aus eine herkömmliche relationale Datenbank mit (sqlite[2]). Diese bietet zunächst jedoch keine räumlichen Datentypen an. Allerdings existiert eine Erweiterung zu sqlite (das auch auf nicht-mobilen Plattformen zur Verfügung steht), die dieses um räumliche Datentypen erweitert: spatialite[3]. Zu beachten ist, dass diese Erweiterung von einem anderen Anbieter und unter anderer (ebenfalls kostenfreier) Lizenz angeboten wird. Schließlich gibt es für diese Erweiterung eine Portierung auf Android, die wiederum von einem anderen Anbieter und unter anderer (ebenfalls kostenfreier) Lizenz angeboten wird[4]. Diese nutzt zur Beschleunigung in C implementierte Funktionen, die über das Java Native Interface (JNI) aufgerufen werden. Damit sind allerdings für unterschiedliche Prozessorarchitekturen unterschiedliche Installationspakete zu bauen. Insgesamt ist es damit möglich, eine um räumliche Datentypen erweiterte eingebettete SQL-Datenbank unter Android zu nutzen. Somit eignet sich die Android-Plattform mit der spatialite-Erweiterung aus theoretischer Sicht für die Umsetzung der Implementierungen gemäß der Modelle A, B und E aus Abschnitt 3.

Um diese theoretische Machbarkeit praktisch zu überprüfen, wurde eine Beispielanwendung realisiert. Diese Anwendung soll zu einer zuvor erstellten und auf das Gerät übertragenen Datenbank mit räumlichen Informationen (hier aus den OpenStreetMap Datensätzen entnommene Straßengeometrien für je ein Bundesland, also Geometrien vom Typ LINESTRING) elementare räumliche Operationen ausführen. Da es sich bei diesen Daten um einen feststehenden Datensatz handelt, der in der Prototyp-Anwendung nur abgefragt, jedoch

[2] http://sqlite.org/
[3] http://www.gaia-gis.it/gaia-sins/
[4] http://code.google.com/p/spatialite-android/

nicht geändert wird, bietet es sich zu Testzwecken an, mithilfe des Desktop-Tools spatialitegui die Datenbank zunächst auf einem herkömmlichen Rechner zu erzeugen und dann als fertige spatialite-Datendatei auf das mobile Endgerät zu übertragen. Mit der genannten GUI ist es auch möglich, räumliche Indexe anzulegen, die dann in die Datendatei integriert sind. Für künftige sinnvolle Anwendungen wird die Einschränkung des nur lesenden Zugriffs auf dem mobilen Gerät nicht mehr gelten, so dass dort dann ggfs. ein anderes Setup zu wählen ist.

Die übertragene Datenbankdatei wird dann von der Anwendung auf dem mobilen Gerät angesprochen. In der Anwendung sollen dabei räumliche Selektionen durchgeführt werden, die mit einer der Operationen INTERSECTS, TOUCHES oder CROSSES alle Geometrien aus der Datenbank findet, die mit einer gegebenen Geometrie in der entsprechenden räumlichen Beziehung stehen. Dazu wurde eine einfache GUI entwickelt, die es dem Benutzer erlaubt, eine Straße anhand des Namens aus der Datenbank auszuwählen und dann alle anderen Straßen in der Datenbank findet, die in der gewählten räumlichen Beziehung zu dieser Straße stehen. Dabei werden die entsprechenden SQL-Anfragen von der Anwendung erzeugt, also etwa:

- SELECT r1.name, r2.name FROM roads r1, roads r2 WHERE r1.name = <NAME> AND
  ST_INTERSECTS(r1.geometry, r2.geometry).[5]

Die Anwendung wurde für die Android-Plattform erstellt, Details sind in OLBRICHT, A. beschrieben. Mithilfe der funktionierenden Anwendung kann einerseits die praktische Realisierbarkeit der Verfahren A und E auf Android unter Beweis gestellt werden. Ferner kann damit eine quantitative Analyse durchgeführt werden, bis zu welcher Datenbankgröße noch akzeptable Antwortzeiten zu erreichen sind bzw. wie groß die Geschwindigkeitsvorteile durch Nutzung eines räumlichen Index sind (also wie deutlich die Variante A besser als E ist).

Einen kurzen Überblick über die verwendeten Datensätze für die Evaluation gibt Tab. 1; es wurden jeweils die Straßendaten (Linienzüge) des Datensatzes des jeweiligen Bundeslandes verwendet.

Aus Tab.1 lässt sich erkennen, dass die Verwendung eines räumlichen Index mit einer Vergrößerung der Datenbankdatei von gut 20% einhergeht. Dies ist auf einem mobilen Endgerät mit begrenzter Speicherkapazität (auch auf einer zusätzlichen SD-Karte) durchaus zu bedenken. Insgesamt erkennt man, dass alleine die Straßendaten des Bundeslandes Bayern bereits fast 0,5 GB Platz in der Datenbank belegen, so dass mit aktuellen mobilen Geräten die Größe der Datenbank durchaus zu einem limitierenden Faktor werden kann. Auf jeden Fall kann man erkennen, dass eine reine Hauptspeicherdatenbank für in der Praxis relevante Datensatzgrößen nur für sehr eng begrenzte Bereiche in Frage kommt.

Tab. 1: Für die Evaluation verwendete Datensätze aus OSM

Datensatz	**DB-Größe netto (KB)**	**DB-Größe brutto (KB)**	**Anzahl Linienzüge**	**Index-Overhead**
Bremen	9236	11413	41163	23,6%
Sachsen	100603	122400	421626	21,7%

[5] Für die Nutzung eines räumlichen Index in Spatialite sind komplexere Anfragen zu stellen; Details dazu finden sich in der Dokumentation unter http://www.gaia-gis.it/gaia-sins/spatialite-tutorial-2.3.1.html

Niedersachsen	176326	216381	779659	22,7%
Bayern	390329	472452	1532962	21,0%

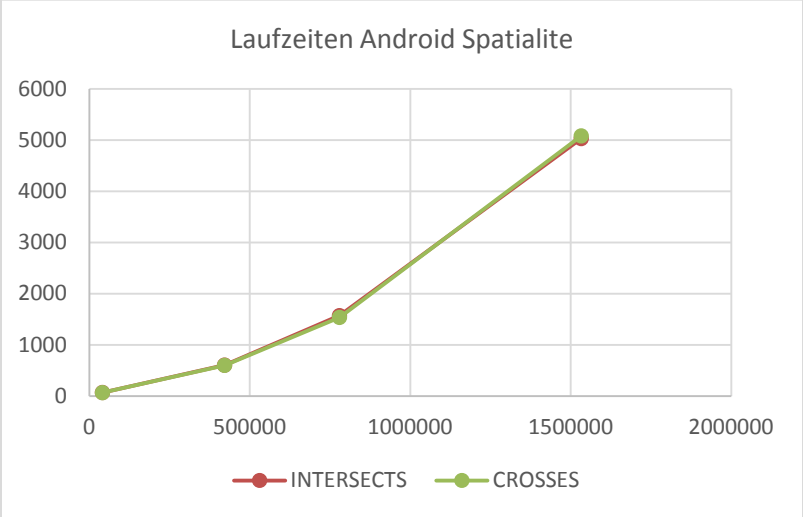
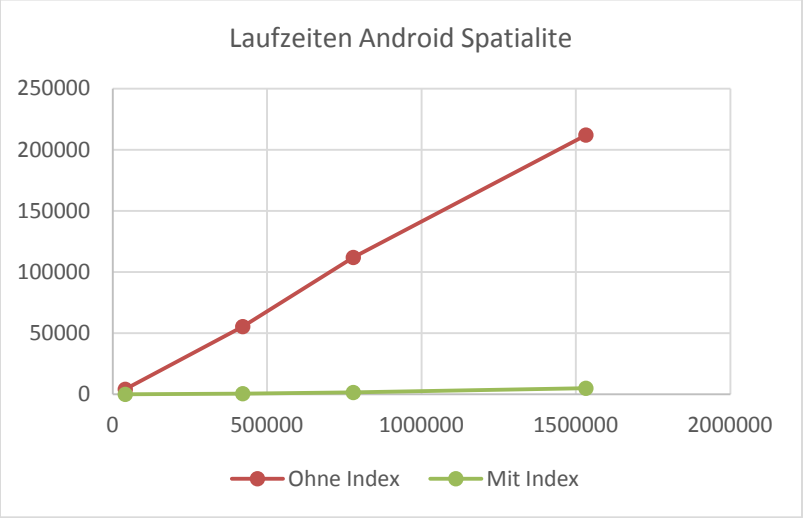


Abb. 1: Laufzeitvergleich Operationen INTERSECTS und CROSSES

In Abb. 1 sind die Laufzeiten (Mittelwerte über verschiedene Anfragen ohne Initialaufwand) für INTERSECTS und CROSSES-Anfragen auf den verschiedenen Datenbeständen dargestellt. Wie man erkennen kann, nimmt die Laufzeit mit zunehmender Zahl an Datensätzen superlinear zu. Hierbei ist zu beachten, dass die Standardabweichung der Zeiten für größere Datensätze stark ansteigt; dies ist vermutlich auf die bekannten Probleme des R*-Baums bei großen Überlappungen zurückzuführen, die bei großen Datensätzen wahrscheinlicher werden. Der Unterschied zwischen den verschiedenen geometrischen Operationen ist sehr gering, allerdings ist eine CROSSES-Anfrage immer geringfügig schneller als eine INTERSECTS-Anfrage. Ferner kann man deutlich erkennen, dass die absoluten Laufzeiten[6] vollkommen ausreichend sind, um eine interaktive Anwendung auch bei Datensätzen der maximalen hier getesteten Größe zu ermöglichen (kleiner als 5 Sekunden). Für die erste Anfrage zu einer neu geöffneten Datenbank muss man noch einen Aufschlag für die zu füllenden Datenbank-Caches einrechnen, der zwischen 10 und 50% der o.g. Laufzeiten betrug (prozentual geringer je größer die Datenbank). Nach einer einzigen Anfrage sind danach jedoch alle anderen Anfragen (egal welcher Operation) beschleunigt. Evtl. wäre in der Praxis also eine einmalige Anfrage schon beim Starten der Anwendung angeraten, damit der Nutzer bei seiner ersten Anfrage schon von den gefüllten Caches profitieren kann.

In Abb. 2 wird illustriert, wie sich die Anfragezeiten mit und ohne räumlichen Index verhalten. Es ist deutlich zu erkennen, dass sich die gut 20% Speicher-Overhead für den räumlichen Index hier stark bezahlt machen. Bereits beim Datensatz Sachsen ist die Anfragezeit ohne Index völlig inakzeptabel, selbst beim (sehr kleinen) Datensatz Bremen ist die Zeit mit über 4 Sekunden

[6] Es ist zu beachten, dass die absoluten Laufzeiten als obere Schranken zu betrachten sind, da spezifische weitere Optimierungen der Datenbank (außer räumlichen Indexen) nicht eingesetzt wurden. Ebenso wurden weder die Parameter der VM optimiert noch die aktuellste Android-Version verwendet.



bereits an der Grenze des zumutbaren für eine interaktive Anwendung. Die Laufzeit ohne Index steigt linear mir der Anzahl der vorhandenen Geometrien, da der geometrische Algorithmus dann für entsprechend mehr Objekte ausgeführt werden muss. Der Einfluss der Datendichte in der Nähe des Anfragebereichs ist hier dementsprechend nicht mehr vorhanden. Schließlich ist der Einfluss

Abb. 2: Laufzeitvergleich Operationen mit und ohne räumlichen Index

des DB-Cache kaum nachweisbar, da die ersten Anfragen kaum länger als Folgeanfragen benötigten. Trotz des super-linearen Anstiegs der Zeiten mit Index, sind die absoluten Werte noch extrem niedriger als bei Anfragen ohne Index, so dass sich der Einsatz des Index bei allen realistisch denkbaren Datenmengen lohnen wird.

Insgesamt zeigt diese Evaluation, dass die Variante A aus Abschnitt 3 auf aktuellen Smartphones unter Android mit Spatialite eine gute Option darstellt, um auch mit relativ großen Datensätzen interaktive räumliche Anwendungen zu realisieren. Eine Verwendung ohne räumliche Indexe (und nur einstufige Anfrageverarbeitung) ist allenfalls für sehr kleine Datensätze eine mögliche Alternative. Dabei sind diese Aussagen hier für Liniengeometrien nachgewiesen worden; die Art der räumlichen Operation spielt keine große Rolle.

4.2 Geo-Daten auf Windows Phone-Geräten

Die Plattform Windows Phone bringt ebenfalls von Hause aus eine lokale eingebettete relationale SQL-Datenbank mit. Diese wird aus den Anwendungen mithilfe des LINQ to SQL-Frameworks angesprochen. Diese Datenbank bietet jedoch keine räumlichen Erweiterungen an, so dass die eigentlichen geometrischen Operationen in der Anwendung ausgeführt werden müssen. Auf dieser Plattform ist also aus theoretischer Sicht zurzeit nur eine Realisierung nach Verfahren C aus Kapitel 3 möglich.

Auch für Windows Phone soll die praktische Machbarkeit einer Implementierung durch eine prototypische Anwendung mit der gleichen Funktionalität wie im Abschnitt 4.1 für Android beschrieben unter Beweis gestellt werden. Diese Implementierung erfordert allerdings einen deutlich höheren Aufwand als die Implementierung für Android, da die folgenden Komponenten zu entwickeln sind, die auf der Android-Plattform durch das räumliche Datenbanksystem bereitgestellt wurden:

- Ein Klassenmodell für die zu verwendenden Geometrieklassen muss entwickelt werden (alternativ ist die Verwendung einer Geometriebibliothek denkbar)
- Für die Nutzung von LINQ to SQL müssen Abbildungsregeln mithilfe von Annotationen definiert werden, die die Objektklassen der Anwendung auf relationale Tabellen in SQL abbilden. Auch wenn der Aufwand der Erstellung der Annotationen eher gering ist, so bleibt auf jeden Fall das Problem der potentiell ineffizienten Abbildung von Geometrieobjekten auf das relationale Modell.
- Die zweistufige Anfragebearbeitung und die Verwendung von Approximationen der Geometrieobjekte muss in der Anwendung implementiert werden.
- Die eigentlichen geometrischen Operationen auf den Kandidatenobjekten müssen ebenfalls in der Anwendung implementiert werden. Hier ist prinzipiell die Verwendung einer Geometriebibliothek (passend zu den oben definierten Geometrieklassen) möglich, so dass kein zusätzlicher Aufwand anfallen muss.

Im Rahmen der Arbeit in OLBRICHT, A. wurde eine Anwendung analog zur Android-Anwendung realisiert, die keine Geometriebibliothek verwendet, so dass die zu nutzenden Geometrieoperationen selbst implementiert wurden. Die Anwendung dokumentierte die prinzipielle Machbarkeit einer solchen persistenten Speicherung von Geo-Daten auf der Windows Phone-Plattform. Allerdings ist die Performanz dieser Anwendung bisher noch nicht ausreichend, um Datenbestände in aus praktischer Sicht nützlicher Größe verwalten zu können. Ob die Effizienz durch eine optimierte Implementierung der Anwendung hinreichend verbessert werden kann, muss sich in künftigen Arbeiten an der Verbesserung der Anwendung zeigen. Denkbar ist auch, dass die stets erforderliche Abbildung der Geometrien auf Relationen den eigentlichen Flaschenhals darstellt, der auch mit einer verbesserten Realisierung nicht zu vermeiden ist.

5 Fazit und Ausblick

In diesem Artikel wurden in Abschnitt 3 verschiedene Modelle zur lokalen Verwaltung räumlicher Daten auf aktuellen Smartphone-Plattformen diskutiert und die jeweiligen Antwortzeiten für räumliche Selektionen theoretisch dargestellt. Im Abschnitt 4.1 wurden dann zwei dieser fünf Varianten auf der Plattform Android mithilfe der räumlichen Datenbank spatialite in einer prototypischen Anwendung realisiert und evaluiert. Es zeigte sich, dass eine effiziente Verwaltung räumlicher Daten lokal auf einem Android-Gerät mit der Variante A möglich ist. Die Antwortzeiten für räumliche Selektionen lagen dabei in einem Bereich, der bis zu etwa 500.000 Liniengeometrien noch interaktive Anwendungen erlaubt. Allerdings ist bei einer zunehmenden Größe der Datenbank eine große Abweichung in einzelnen Fällen möglich. Im Abschnitt 4.2 wurde dann eine prototypische Realisierung auf der Plattform Windows Phone nach Variante C beschrieben. Diese hat jedoch bisher noch keine Performanz erreicht, die einen Test mit realistischen Datenmengen ermöglichen würde.
Die Erweiterung der Prototypen stellt somit eines der naheliegenden Ziele für künftige Arbeiten dar. Dabei sind verschiedene Erweiterungen denkbar:

- Weitere Optimierung der Windows Phone Anwendung, um Messungen mit realistischen Datenmengen durchführen zu können.

- Implementierung und Auswertung der Varianten B und D aus Abschnitt 3 auf passenden Plattformen.
- Evaluierung, in wie weit weitere Optimierungen an der bestehenden Android-Anwendung Vorteile bringen (z. B. Nutzung des spatialite MBR-Cache)

Auch aus konzeptioneller Sicht sind noch Erweiterungen wünschenswert. So sollten die Zeitfunktionen der Varianten aus Abschnitt 3 auch für die (wesentlich aufwändigeren) räumlichen Verbunde (spatial join) ausgedehnt werden. Dazu sollten dann umfangreiche Auswertungen der vorhandenen Prototypen erfolgen. Gleiches gilt für Anfragen an komplexere Geometrien als Linien, also etwa Polygone. Hier sind die geometrischen Operationen noch komplexer, so dass mit entsprechend größerem Aufwand bei der Anfragebearbeitung zu rechnen ist. Schließlich sollten die Evaluationen auf verschiedenen Android-Plattformen und Gerätekonfigurationen getestet werden, um den Einfluss von Android-Version und Geräteleistung quantifizieren zu können. Zudem könnten weitere mobile Plattformen wie z. B. iOS untersucht werden. Diese Plattform verhält sich konzeptionell ähnlich wie Windows Phone, so dass dort vermutlich die gleichen Varianten realisierbar sind.

Abschließend soll noch erwähnt werden, dass die Verwendung einer räumlichen Datenbank für flexible Abfragen und auch DML-Operationen besonders dann gerechtfertigt ist, wenn auch auf dem mobilen Gerät mit Modifikationen/Ergänzungen an den Daten zu rechnen ist. Geht es um reine Abfragen vorher feststehender Geometrien (wie etwa bei statischer Routenplanung), so gibt es bessere Speicher- und Abfragemodelle für räumliche Daten als R-Bäume (vgl. GAEDE, V. & GÜNTHER, O.).

6 Literaturverzeichnis

COELHO, P. & AGUIAR, A. & LOPES, J.C., 2011: OLBS: Offline Location Based Services. Proceedings of the Fifth International Conference on Next Generation Mobile Applications, Services and Technologies, S. 70-75.

GAEDE, V. & GÜNTHER, O., 1998: Multidimensional access methods. ACM Computing Surveys **30** (2, June 1998), S. 170-231.

GARDINER, K. & YIN, J. & CARSWELL, J. D., 2009: EgoViz - A Mobile Based Spatial Interaction System. Proceedings of the 9th Int. Symposium on Web and Wireless GIS. Lecture Notes in Computer Science **5886**, Springer-Verlag, S. 135-152.

GRØNLI, T. M. & HANSEN, J. & GHINEA, G., 2010: Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments. Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments (PETRA '10), ACM, New York, NY, USA, Article 45.

GUTTMAN, A., 1984: R-trees: a dynamic index structure for spatial searching. SIGMOD Record **14** (2), S. 47-57.

ILARRI, S. & MENA, E. & ILLARRAMENDI, A., 2010: Location-dependent query processing: Where we are and where we are heading. ACM Computing Surveys 42 (3), Article 12.

JACOB, R. & SMITHERS, S. & WINSTANLEY, A. C., 2012: Performance evaluation of storing and querying spatial data on mobile devices for offline location based services. Proceedings of the Signals and Systems Conference (ISSC 2012), Maynooth, S. 1-6.

KIM, H. & AGRAWAL, N. & UNGUREANU, C., 2012: Revisiting storage for smartphones. ACM Transactions on Storage **8** (4), Article 14.

OLBRICHT, A., 2013: Persistente Speicherung von Geo-Daten auf verschiedenen mobilen Plattformen. Hochschule Hannover, Fakultät IV, Abt. Informatik, Masterarbeit.

WU, C.-H. & CHANG, L.-P. & KUO, T.-W., 2003: An efficient R-tree implementation over flash-memory storage systems. Proceedings of the 11th ACM international symposium on Advances in geographic information systems (GIS '03). ACM, New York, USA, S. 17-24.