Flexible Indexierung für Ähnlichkeitssuche mit logikbasierten Multi-Feature-Anfragen

Marcel Zierenberg
Brandenburgische Technische Universität Cottbus
Institut für Informatik, Informations- und Medientechnik
Lehrstuhl Datenbank- und Informationssysteme
Postfach 10 13 44, 03013 Cottbus, Deutschland
zieremar@tu-cottbus.de

KURZFASSUNG

Ähnlichkeitssuche beschäftigt sich mit dem Auffinden ähnlicher Objekte zu einem vorgegebenen Anfrageobjekt. Die logische Kombination verschiedener Features des Anfrageobjekts erhöht dabei die Ausdruckskraft von Anfragen und führt zu besseren Anfrageergebnissen. Um eine effiziente Suche zu ermöglichen ist eine Indexierung der Datenbankobjekte nötig. Neben einer möglichst hohen Sucheffizienz spielt die Flexibilität des Indexierungsverfahrens eine entscheidende Rolle. Das in dieser Arbeit vorgestellte Indexierungsverfahren ermöglicht eine effiziente Verarbeitung von Multi-Feature-Anfragen mit beliebigen logischen Kombinationen und Gewichtungen anhand eines einzigen Index. Die Verwendung metrischer Indexierungsmethoden garantiert die Anwendbarkeit des Konzepts für ein großes Spektrum von Features und Distanzfunktionen.

Kategorien und Themenbeschreibung

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing — Indexing methods

Allgemeine Begriffe

Algorithms, Performance

Schlüsselwörter

similarity search, retrieval, nearest neighbor search, metric indexing, complex query

1. EINLEITUNG

Ähnlichkeitssuche [13] verfolgt das Ziel, in einer Menge von Objekten genau die Objekte zu finden, die einem Anfrageobjekt am ähnlichsten sind. Die (Un-)Ähnlichkeit der Objekte wird mithilfe von *Distanz-* oder *Ähnlichkeitsmaßen*[1] anhand der aus den Objekten extrahierten *Features* bestimmt. Bei einem Feature handelt es sich dabei um eine Menge von Werten, die bestimmte Eigenschaften eines Objekts charakterisieren. Die Nutzung von Features, welche die gewünschte Semantik geeignet beschreiben, ist dabei entscheidend für eine effektive Ähnlichkeitssuche.

Die Verwendung mehrerer Features und einer geeigneten Kombination dieser, erhöht die Ausdruckskraft von Anfragen und führt somit zu besseren Anfrageergebnissen [20]. *Multi-Feature-Anfragen* nutzen daher eine Vielzahl von Features für die Anfrageformulierung.

Logikbasierte Anfragemodelle, wie die *Commuting Quantum Query Language* (*CQQL* [14]) oder die *Fuzzy-Logik* [18], erlauben die Kombination mehrerer Features mithilfe boolescher Junktoren. Neben der verbesserten Ausdruckskraft von Anfragen wird hierdurch eine Verbindung von (unscharfen) Ähnlichkeitsbedingungen und (scharfen) relationalen Datenbankbedingungen ermöglicht [13]. Eine *Gewichtung* der einzelnen Anfrageatome erlaubt weiterhin eine dynamische Anpassung der Anfragen. Das Lernen dieser Gewichte anhand von Nutzerpräferenzen [19] ermöglicht eine schrittweise Verfeinerung von Anfragen in Form von *Relevance Feedback*.

Der naive Ansatz zur Umsetzung einer Ähnlichkeitssuche ist der *lineare Scan* der Datenbank, bei dem alle Distanzen zwischen Anfrageobjekt und Datenbankobjekten ermittelt werden. Für Multi-Feature-Anfragen bedeutet dies, dass für jedes einzelne Feature die Distanz zwischen Anfrage- und jedem Datenbankobjekt ermittelt wird. Anschließend werden diese *Teildistanzen* für jedes Objekt zu einer *Gesamtdistanz* aggregiert. Da die Evaluierung von Distanzfunktionen mitunter hohe CPU- und das Lesen der zugehörigen Features hohe I/O-Kosten verursachen, ist der lineare Scan für große Datenbanken mit einer Vielzahl von Objekten und Features nicht praktikabel. Stattdessen sollten *Indexierungsverfahren* genutzt werden, um eine effizientere Suche zu realisieren. Sie ermöglichen einen frühzeitigen Ausschluss von Objekten, die nicht zur Ergebnismenge gehören können, und verringern somit die Anzahl der nötigen Distanzberechnungen und I/O-Operationen.

Eine besondere Anforderung an die logikbasierte Indexierung stellt die *Flexibilität* dar. Die logische Kombination der Anfrage und auch die Anfragegewichte können sich im Rahmen von Relevance Feedback dynamisch verändern, dennoch sollte nur ein einziger Index zur Verarbeitung beliebiger Anfragen benötigt werden. Weiterhin sollte das Indexierungsverfahren unabhängig von Art und Struktur der verwendeten Features und Distanzfunktionen sein.

Hauptbeitrag dieser Arbeit ist die Entwicklung eines effizienten Indexierungsverfahrens zur Verarbeitung logikbasierter Multi-Feature-Anfragen am Beispiel von CQQL. Dazu werden spezifische Anforderungen an die logikbasierte Indexierung definiert und anhand dieser ein Indexierungsverfahren entworfen, das eine effizi-

[1] Im Folgenden gehen wir von Distanzmaßen aus.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

Tabelle 1: Umwandlung boolescher Ausdrücke in arithmetische Formeln in CQQL

Ausdruck	arithmetische Formel
$\neg a$	$1 - a$
$a \wedge b$	$a * b$
$a \vee b$	$a + b - a * b$
$(c \wedge a) \vee (\neg c \wedge b)$	$a + b$

ente und gleichzeitig flexible Verarbeitung beliebiger logikbasierter Multi-Feature-Anfragen und eine dynamische Gewichtung dieser ermöglicht.

Die Arbeit ist wie folgt aufgebaut. Abschnitt 2 definiert die grundlegenden Begriffe und die Notationen dieser Arbeit. In Abschnitt 3 wird ein theoretisches Anwendungsbeispiel aus dem Bereich der Bildähnlichkeitssuche mithilfe logikbasierter Multi-Feature-Anfragen präsentiert. Auf die speziellen Anforderungen an Indexierungsverfahren für derartige Anfragen wird in Abschnitt 4 detailliert eingegangen. Abschnitt 5 beschäftigt sich mit dem Stand der Technik zur Indexierung. Abschnitt 6 zeigt das Konzept eines Indexierungsverfahrens und Kriterien zur Indexauswahl. Abschließend wird in Abschnitt 7 eine Zusammenfassung der Arbeit gegeben.

2. GRUNDLAGEN

Der folgende Abschnitt definiert die grundlegenden Begriffe und die Notationen, die in dieser Arbeit verwendet werden.

2.1 Nächste-Nachbarn-Suche

Ähnlichkeitsanfragen anhand von Distanzmaßen werden auch als *k-Nächste-Nachbarn-Suche* (*kNN*) bezeichnet und geben die k Objekte aus einer Datenbank zurück, deren Distanz zum Anfrageobjekt am geringsten ist.

Eine Ähnlichkeitsanfrage $\mathrm{kNN}(q)$ besteht aus einem Anfrageobjekt q aus dem Universum $\mathbb{U}$ und bezieht sich auf eine Datenbank $D = \{o_1, o_2, \ldots, o_n\} \subseteq \mathbb{U}$ von Objekten o_i. Die Distanz (Unähnlichkeit) von Objekten wird mithilfe einer Distanzfunktion $\delta : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}_{\geq 0}$ anhand der aus den Objekten extrahierten Features q' und o_i' bestimmt. Das Ergebnis der Anfrage $\mathrm{kNN}(q)$ ist dann eine (nichtdeterministische) Menge $K \subseteq D$, für die gilt: $|K| = k$ und $\forall o_i \in K, o_j \in D \setminus K : \delta(q, o_i) \leq \delta(q, o_j)$.

Bei einer kNN-Anfrage bestehend aus m Features werden die Features eines Objekts mit $q' = (q^1, q^2, \ldots, q^m)$ beziehungsweise $o_i' = (o_i^1, o_i^2, \ldots, o_i^m)$ bezeichnet. Jedem Feature wird eine eigene Distanzfunktion δ^j zugeordnet. Die Teildistanzen d_i^j aller Features werden durch eine *Aggregationsfunktion* $\mathrm{agg} : \mathbb{R}_{\geq 0}^m \mapsto \mathbb{R}_{\geq 0}$ zu einer Gesamtdistanz d_i^{agg} vereinigt. Die k nächsten Nachbarn werden dann anhand dieser Gesamtdistanz bestimmt.

2.2 CQQL

Für die Evaluation einer auf CQQL basierenden Anfrage werden boolesche Ausdrücke nach ihrer DNF-Normalisierung anhand von festgelegten Transformationsregeln in arithmetische Formeln umgewandelt [14]. Tabelle 1 zeigt die in CQQL verwendeten Transformationsregeln. Ebenso ist eine Umsetzung der Gewichtung in CQQL direkt innerhalb der booleschen Logik möglich. Dazu werden Gewichte anhand der in Tabelle 2 gezeigten Transformationsregeln in die Logik eingebettet.

2.3 Logikbasierte kNN

Im Folgenden werden die in Abschnitt 2.2 beschriebenen arithmetischen Formeln als Aggregationsfunktionen betrachtet, deren Definitions- und Wertebereich auf Ähnlichkeitswerte im Intervall [0,1] beschränkt ist: $\mathrm{agg} : [0,1]^m \mapsto [0,1]$. Hierbei steht ein Ähnlichkeitswert von 1 für die maximale Ähnlichkeit (Identität), während ein Wert von 0 die größtmögliche Unähnlichkeit darstellt.

Tabelle 2: Einbettung von Gewichten in die Logik

Ausdruck	Einbettung
$a \wedge_{\theta_1,\theta_2} b$	$(a \vee \neg\theta_1) \wedge (b \vee \neg\theta_2)$
$a \vee_{\theta_1,\theta_2} b$	$(a \wedge \theta_1) \vee (b \wedge \theta_2)$

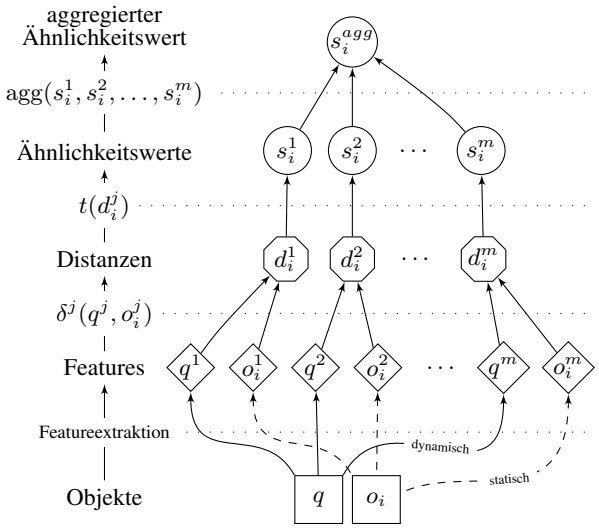

Abbildung 1: Ablauf einer logikbasierten Multi-Feature-Anfrage

Um eine Nächste-Nachbarn-Suche anhand logikbasierter Anfragen zu realisieren, müssen alle Teildistanzen d_i^j vor ihrer Aggregation in Ähnlichkeitswerte s_i^j umgewandelt werden. Die nächsten Nachbarn sind nach der Aggregation dieser Ähnlichkeitswerte dann genau die Objekte, deren aggregierte Ähnlichkeitswerte s_i^{agg} bezüglich dem Anfrageobjekt am größten sind. Abbildung 1 verdeutlicht den Suchablauf.

Beispiele für geeignete Funktionen zur Umwandlung von Distanzen in Ähnlichkeitswerte sind $t(x) = e^{-x}$ oder $t(x) = 1 - \frac{x}{\delta_{max}}$, wobei δ_{max} der maximale Distanzwert der genutzen Distanzfunktion darstellt [13].

2.4 Monotonie

Eine Aggregationsfunktion ist *monoton steigend im i-ten Argument*[2], wenn gilt:

$$x^i \leq y^i \implies \mathrm{agg}(x^1, \ldots, x^i, \ldots, x^m) \leq \mathrm{agg}(x^1, \ldots, y^i, \ldots, x^m) \quad (1)$$

Das bedeutet, dass wenn alle Parameter außer x^i festgelegt sind und x^i vergrößert wird, kann das Ergebnis der Aggregationfunktion nicht kleiner werden. Eine Aggregationsfunktion ist *global monoton steigend*, wenn sie in allen m Argumenten monoton steigend ist. Ein Beispiel ist $\mathrm{agg}(x^1, x^2) = x^1 + x^2$.

Eine Aggregationsfunktion ist *lokal monoton*, wenn sie in einem Teil ihrer m Argumente monoton steigend und in allen anderen Argumenten monoton fallend ist. Ein Beispiel ist $\mathrm{agg}(x^1, x^2) = x^1 - x^2$.

[2] *Monoton fallend im i-ten Argument* ist analog anhand des $\geq$-Operators definiert.

3. ANWENDUNGSBEISPIEL

Das folgende Anwendungsbeispiel illustriert eine logikbasierte Ähnlichkeitsanfrage anhand mehrerer Features.

Gegeben sei eine Datenbank mit Bildern verschiedener Pilze und Informationen über deren Art und Giftigkeit. Ziel sei es nun anhand eines Anfragebildes eines gesammelten Pilzes, die Art des Pilzes zu ermitteln und so zu bestimmen, ob der Pilz essbar ist. Die aus den Bildern extrahierten Features umfassen dabei aus den Pixeldaten erzeugte Farb- ($color$) und Formfeatures ($shape$) sowie aus den Metadaten stammende Informationen, wie das Datum der Bildaufnahme ($date$) oder die GPS-Koordinaten des Aufnahmeortes (gps). Zur Ermittlung der (Un-)Ähnlichkeit von Objekten werden jeweils für das Feature geeignete Distanzfunktionen genutzt, beispielsweise die *Earth Mover's Distanzfunktion* [11] für Farbsignaturen oder verschiedene *Minkowski-Distanzfunktion* (L_p*-Distanzfunktion*).

Eine Anfrage, bestehend aus nur einem Feature, genügt nicht, um eine korrekte Zuordnung der Pilzarten vorzunehmen. So kann zum Beispiel allein anhand der Farbfeatures eines Pilzes nicht immer ein Rückschluss auf dessen Art gezogen werden, wenn sich etwa die Form stark von der des Anfragebildes unterscheidet. In diesem Fall ist eine UND-Verknüpfung der Features nötig: $shape_{\approx} \wedge color_{\approx}$. Für den Fall, dass der Pilz des Anfragebildes eine für seine Art sehr untypische Form aufweist ($\neg shape_{\approx}$) und daher anhand dieser nicht zugeordnet werden kann, soll stattdessen allein anhand der GPS-Koordinaten des Bildes ($gps_{\approx}$) auf dessen Art geschlossen werden. Als zusätzliche relationale (scharfe) Bedingung sollen nur Pilze betrachtet werden, die im selben Monat gesammelt wurden, wie der Pilz des Anfragebildes ($date_{=}$). Eine logische Verknüpfung der Features eines Anfragebildes sieht dann wie folgt aus, wobei θ_1, θ_2 für Parameter stehen, die eine unterschiedliche Gewichtung der Teilbedingungen ermöglichen.

$$date_{=} \wedge ((shape_{\approx} \wedge color_{\approx}) \vee_{\theta_1,\theta_2} (\neg shape_{\approx} \wedge gps_{\approx})) \quad (2)$$

Nach einer DNF-Normalisierung und Transformation anhand der in Abschnitt 2.2 beschriebenen Transformationsregeln, ergibt sich aus dem booleschen Ausdruck (2) folgende arithmetische Formel:

$$\begin{aligned}&(\theta_1 * date_{=} * shape_{\approx} * color_{\approx}) + \\ &(\theta_2 * date_{=} * (1 - shape_{\approx}) * gps_{\approx})\end{aligned} \quad (3)$$

4. ANFORDERUNGEN

In [13] werden allgemeine Anforderungen an Indexierungsverfahren im Bereich des Multimedia Retrievals definiert. Auf Grundlage dieser werden im Folgenden die spezifischen Anforderungen für Indexierungsverfahren zur effizienten Verarbeitung von logikbasierten Multi-Feature-Anfragen vorgestellt.

Flexibilität: Multi-Feature-Anfragen setzen sich aus unterschiedlichen Features und Distanzfunktionen zusammen, das Indexierungsverfahren muss daher unabhängig von Art und Struktur der Features sein und ein breites Spektrum an Distanzfunktionen unterstützen. Die Anzahl der unterschiedlichen zur Verfügung stehenden Features ist potentiell hoch. Das Indexierungsverfahren muss daher mit einer großen Menge von Features umgehen können, auch wenn nur eine Teilmenge dieser für eine Anfrage genutzt wird. Je nach Anfrage können sich die genutzten Features unterscheiden. Ebenso sind unterschiedliche logische Kombinationen und unterschiedliche Gewichtungen der selben Features möglich. Das Indexierungsverfahren darf daher nicht nur auf eine logische Kombination zugeschnitten werden, sondern muss mit beliebigen logischen Kombinationen und Gewichtungen umgehen können.

Sucheffizienz: Die Anzahl der nötigen Berechnungen der Distanzfunktion und die Anzahl von I/O-Operationen (Seitenzugriffe) dienen als Effizienzmaß für Indexierungsverfahren. Die Grundlage für die Bewertung der Sucheffizienz bildet der Vergleich mit dem Suchaufwand des linearen Scans der Datenbank. Ein effizientes Indexierungsverfahren sollte diesen linearen Aufwand stets unterbieten. Es existieren zwar Verfahren, welche eine sehr hohe Sucheffizienz bieten, dabei aber einen nicht realisierbar hohen Speicherverbrauch verursachen (vgl. [16]). Ein geeignetes Indexierungsverfahren sollte daher einen möglichst geringen Speicherverbrauch bei gleichzeitig möglichst hoher Sucheffizienz aufweisen.

Skalierbarkeit: Ein inhärentes Problem der Ähnlichkeitssuche ist der *Fluch der hohen Dimensionen* (*FdhD* [13, 5]). Dieser bewirkt, dass die Performanz von Indexierungsverfahren mit steigender (intrinsischer) Dimensionalität[3] eines Features abnimmt. Indexierungsverfahren können den Suchaufwand des linearen Scans dann nicht mehr signifikant unterbieten oder übersteigen ihn sogar. Analog zur Erhöhung der Dimensionsanzahl bei einem Feature lässt sich der FdhD auch bei Multi-Feature-Anfragen beobachten. Die Kombination von Features bewirkt hier ebenfalls eine Erhöhung der (intrinsischen) Dimensionalität. Ein geeignetes Indexierungsverfahren für Multi-Feature-Anfragen muss daher Möglichkeiten bieten, mit dem FdhD umzugehen und möglichst ohne Effizienzeinbußen skalierbar bezüglich der (intrinsischen) Dimensionalität einzelner Features und der Kombination mehrerer Features sein.

5. STAND DER TECHNIK

Der folgende Abschnitt geht auf den Stand der Technik auf dem Gebiet der effizienten Verarbeitung von Multi-Feature-Anfragen ein. Die existierenden Verfahren werden kurz vorgestellt und ausschnittsweise hinsichtlich der in Abschnitt 4 definierten Anforderungen bewertet. Wir beschränken uns dabei auf Verfahren für die *exakte Suche* der nächsten Nachbarn und gehen nicht auf Spezialfälle wie die *approximative Suche* oder zusätzliche Anfragearten wie *getNext* (*Ranking-Anfrage*) ein.

5.1 Combiner-Algorithmen

Combiner-Algorithmen [8] kombinieren die Ergebnisse mehrerer Ähnlichkeitsanfragen zu einem aggregierten Ergebnis. Für Multi-Feature-Anfragen existiert dazu je Feature eine nach Distanz[4] zum Anfrageobjekt sortierte Liste der Datenbankobjekte.

Combiner-Algorithmen sind keine Indexierungsverfahren, sondern arbeiten auf einer darüber liegenden Ebene. Sie legen nicht fest, wie die sortierten Listen bereitgestellt werden. Um eine effiziente Suche zu ermöglichen, sollte der Zugriff über Indexierungsverfahren mit Unterstützung von getNext-Anfragen umgesetzt werden.

Combiner-Algorithmen erlauben eine dynamische Auswahl der verwendeten Features sowie unterschiedliche Aggregationsfunktionen und Gewichtungen. Aufgrund der Forderung nach globaler Monotonie kommen sie jedoch nicht für alle logikbasierten Multi-Feature-Anfragen in Frage. Formel 3 ist beispielsweise nicht global monoton steigend, da eine Erhöhung des Ähnlichkeitswerts $shape_{\approx}$ nicht in jedem Fall zu einer Erhöhung des aggregierten Ähnlichkeitswerts führt. Ein weiterer Nachteil ist die geforderte Bereitstellung der sortierten Listen, da Indexstrukturen mit effizienten getNext-Anfragen nicht immer zur Verfügung stehen und sich durch die getrennte Verwaltung jedes einzelnen Index ein Mehraufwand ergibt.

[3]Die Dimensionalität eines Features ergibt sich aus der Anzahl seiner Featurewerte. Die intrisische Dimensionalität lässt sich beispielsweise anhand der paarweisen Distanzen zwischen den Featurewerten der Datenbankobjekte abschätzen und ist dann definiert als $\rho = \mu^2/2 * \sigma^2$ [1].

[4]Combiner-Algorithmen sind ohne größere Anpassungen auch für Ähnlichkeitswerte nutzbar.

5.2 Räumliche Indexierung

Räumliche Indexierungsverfahren gehen davon aus, dass die Features in Form von Vektoren vorliegen und die euklidsche Distanzfunktion (L_2-Distanzfunktion) zur Berechnung der Unähnlichkeit zwischen den Featurevektoren verwendet wird. Da diese Beschränkung im Widerspruch zur Flexibilität steht, scheidet die Verwendung räumlicher Indexierungsverfahren aus. Dennoch soll im Folgenden auf sie eingegangen werden, da ihre Konzepte teilweise übertragbar sind.

Hierarchische Verfahren, wie zum Beispiel der *R-Baum* [9], beschreiben Mengen von Objekten durch geometrische Regionen (*Cluster*). Aufgrund des Fluchs der hohen Dimensionen sinkt die Sucheffizienz dieser Verfahren jedoch bereits ab einer Dimensionsanzahl der Featurevektoren von 10-20 unter die des linearen Scans [13]. Im Folgenden stellen wir daher lediglich den nicht-hierarchischen Ansatz der VA-Datei vor.

5.2.1 VA-Datei

Die *Vektor-Approximations-Datei* (*VA-Datei*) [17] ist ein nicht-hierarchisches Verfahren und akzeptiert den FdhD in dem Sinne, dass sie statt Cluster zu bilden, direkt einen linearen Scan der Datenbank durchführt. Sie setzt dazu einen *Filter-Refinement*-Ansatz auf Basis kompakter Bitsignaturen ein. Ziel dieses Ansatzes ist es in der Filterphase, anhand der durch Bitsignaturen approximierten Objekte, Distanzgrenzen zu ermitteln und mithilfe dieser möglichst viele Objekte von der weiteren Suche auszuschließen. Die exakte Distanz muss in der Verfeinerungsphase dann lediglich für die Objekte berechnet werden, die in der Filterphase nicht ausgeschlossen werden konnten.

Die Sucheffizienz des Verfahrens ergibt sich daraus, dass die Bitsignaturen in der Filterphase sequentiell aus *Signaturdateien* gelesen werden und durch den Ausschluss von Objekten die Anzahl teurer, wahlfreier Zugriffe in der Verfeinerungsphase verringert wird. Für Multi-Feature-Anfragen ist eine Anpassung der VA-Datei nötig.

5.2.2 GeVAS

GeVAS [2] ist eine Erweiterung der VA-Datei für Multi-Feature-Anfragen und erlaubt eine dynamische Auswahl der in der Anfrage verwendeten Features aus einer großen Menge vorhandener Features. Für jedes Feature wird dazu eine separate VA-Datei erzeugt, wobei die Reihenfolge der Objekte in allen Signaturdateien gleich ist. Bei einer Multi-Feature-Anfrage werden nun nur die Signaturdateien der Features parallel abgearbeitet, die tatsächlich in der Anfrage eingesetzt werden. Für jedes einzelne Feature eines Objekts werden Distanzgrenzen ermittelt und dynamisch zu aggregierten Distanzgrenzen zusammengefasst. Der Ausschluss von Objekten geschieht in der Filterphase anhand dieser aggregierten Distanzgrenzen. Voraussetzung für die korrekte Aggregation von Distanzgrenzen ist, dass die Aggregationsfunktion global monoton steigend ist. Diese Forderung lässt sich jedoch so abschwächen, dass beliebige, logikbasierte Multi-Feature-Anfragen ermöglicht werden (siehe Abschnitt 6.1).

Liegen die einzelnen VA-Dateien auf der gleichen Festplatte (HDD) ergeben sich für GeVAS Effizienzprobleme beim Lesen der Daten vom Sekundärspeicher, da statt jede VA-Datei einzeln sequentiell zu lesen, der Lesekopf der Festplatte zwischen den verschiedenen VA-Dateien hin und her springen muss [13].

5.3 Metrische Indexierung

Metrische Indexierungsverfahren stellen keine Anforderungen an die Art der Featuredaten. Im Gegensatz zu räumlichen Indexierungsverfahren erlauben sie daher auch die Indexierung von Features bei denen es sich nicht um Vektoren handelt (zum Beispiel textuelle Daten) oder die nicht die euklidsche Distanzfunktion verwenden. Sie erfordern lediglich das Vorliegen einer Metrik.

Eine *Metrik* δ ist eine Distanzfunktion, für die folgenden Eigenschaften für alle $o_i, o_j, o_k \in \mathbb{U}$ gelten: $\delta(o_i, o_j) > 0$ für $o_i \neq o_j$ (Positivität), $\delta(o_i, o_i) = 0$ (Selbstidentität), $\delta(o_i, o_j) = \delta(o_j, o_i)$ (Symmetrie) und $\delta(o_i, o_k) \leq \delta(o_i, o_j) + \delta(o_j, o_k)$ (Dreiecksungleichung).

Der Ausschluss von Objekten wird mithilfe der Dreiecksungleichung erreicht, die es ermöglicht, untere und obere Grenzen bezüglich der Distanz von Anfrageobjekt und Datenbankobjekten zu bestimmen. Die Grenzen können effizient anhand von vorberechneten Distanzen zu einem oder mehreren Referenzobjekten[5] ermittelt werden. Die untere und die obere Grenze für die Distanz $\delta(q, o_i)$ und ein Referenzobjekt p sind wie folgt definiert:

$$\underbrace{|\delta(q,p) - \delta(p,o_i)|}_{\delta_{lb}(q,o_i)} \leq \delta(q,o_i) \leq \underbrace{\delta(q,p) + \delta(p,o_i)}_{\delta_{ub}(q,o_i)} \quad (4)$$

Analog zu räumlichen Indexierungsverfahren lässt sich zwischen hierarchischen (*M-Baum* [7]) und nicht-hierarchischen (*AESA* [16]) metrischen Indexierungsverfahren unterscheiden. Eine umfassende Übersicht metrischer Indexierungsverfahren bietet zum Beispiel das Lehrbuch von Samet [12].

Der Großteil der existierenden metrischen Indexierungsverfahren ist auf die Indexierung anhand einer einzigen Distanzfunktion ausgelegt. Die Forderung nach der Unterstützung beliebiger logischer Kombinationen kann daher nicht erfüllt werden. *Multi-Metrische Indexierungsverfahren* [4] ermöglichen die Indexierung auf Grundlage einer dynamischen Kombination mehrerer Distanzfunktionen zu einer Aggregationsfunktion. Sie unterstützen dadurch mit einem einzigen Index unterschiedliche Gewichtungen der gleichen Aggregationsfunktion. Da es sich bei der Aggregationsfunktion jedoch um eine Metrik handeln muss, ist diese auf die gewichtete Summe metrischer Distanzfunktion beschränkt. Für logikbasierte Multi-Feature-Anfragen sind diese Verfahren daher nur eingeschränkt anwendbar.

5.3.1 M²-Baum

Der *M²-Baum* [6] ist eine Erweiterung des M-Baums für Multi-Feature-Anfragen. Statt die Distanzen bezüglich aller Features bereits bei der Indexierung zu aggregierten Distanzen zusammenzufassen, werden die Distanzgrenzen bei der Anfrage dynamisch für jedes einzelne Feature abgeschätzt. Diese Grenzen werden anschließend in Ähnlichkeitswerte umgewandelt und zu aggregierten Ähnlichkeitsgrenzen kombiniert (vergleiche aggregierte Distanzgrenzen bei GeVAS). Der Vorteil bei diesem Vorgehen ist, dass die metrischen Eigenschaften in diesem Fall nicht für die Aggregationsfunktion gelten müssen, sondern nur für jede zugrundeliegende Distanzfunktionen.

Der M^2-Baum verlangt die lokale Monotonie der Aggregationsfunktion. Die arithmetische Formel 3 erfüllt jedoch auch diese Eigenschaft nicht. Analog zu GeVAS lässt sich die Monotonie-Eigenschaft aber auch hier so abschwächen, dass beliebige, logikbasierte Multi-Feature-Anfragen möglich werden (siehe Abschnitt 6.1). Der M^2-Baum erlaubt somit beliebige, logische Kombinationen und eine dynamische Auswahl der tatsächlich genutzten Features. Da es sich beim M^2-Baum um ein hierarchisches Verfahren handelt, nimmt die Sucheffizienz jedoch aufgrund des Fluchs der hohen Dimensionen mit steigender intrinsischer Dimensionalität der Features stärker ab, als bei nicht-hierarchischen Verfahren [13].

[5] Für Referenzobjekte existieren unterschiedliche Benennungen in der Literatur, wie zum Beispiel *Routing*-, *Focus*-, *Vantage*- oder *Pivot*-Objekt, die das gleiche Konzept widerspiegeln.

6. KONZEPT

Dieser Abschnitt beschreibt das Konzept eines Indexierungsverfahrens zur effizienten Verarbeitung logikbasierter Multi-Feature-Anfragen. Dazu wird zuerst auf die Berechnung aggregierter Distanzgrenzen eingegangen. Die Übertragung des GeVAS-Ansatzes auf den metrischen Raum stellt den Kern des Konzepts dar. Wir wählen GeVAS aufgrund seiner Flexibilität im Bezug auf Featureanzahl und -auswahl sowie aufgrund seiner besseren Skalierbarkeit im Bezug auf die Dimensionalität als hierarchische Verfahren. Zusätzliche Anpassungen, wie die direkte Berechnung exakter Distanzen für eine Teilmenge der Features, dienen dazu, die Sucheffizienz des entworfenen Verfahrens bei steigender Featureanzahl zu verbessern.

6.1 Aggregierte Distanzgrenzen

Die korrekte Berechnung aggregierter Distanzgrenzen[6] hängt von der Monotonie der Aggregationsfunktion ab. Zur Berechnung der oberen Distanzgrenze einer global monoton steigenden Aggregationsfunktion müssen die oberen Grenzen aller Teildistanzen in die Aggregationsfunktion eingesetzt werden [2].

Für die obere Distanzgrenze lokal monotoner Aggregationsfunktionen kommt es auf die Monotonie der einzelnen Argumente an. Bei monoton steigenden Argumenten muss die obere Distanzgrenze und bei monoton fallenden Argumenten die untere Distanzgrenze eingesetzt werden [6].

Die arithmetische Formel (3) ist nicht lokal monoton. Jedoch liegt eine Monotonie vor, für die wir den Begriff *fixe Monotonie* einführen. Hierbei hängt die Monotonie eines Arguments der Aggregationsfunktion von den Wertebelegungen der anderen Argumente ab. Eine fix monotone Funktion kann also in einem Argument für bestimmte Wertebelegungen monoton fallend und für alle andere Wertebelegungen monoton steigend sein. In Formel (3) ergibt sich die fixe Monotonie daraus, dass das Argument $shape_{\approx}$ in einem Teil der Formel negiert und in einem anderen Teil nicht-negiert auftritt.

Die aggregierte obere Distanzgrenze für fix monotone Funktionen ergibt sich durch das Einsetzen aller möglichen Kombinationen von oberen und unteren Grenzen in die Aggregationsfunktion und einer Auswahl des maximalen Ergebnisses.

$$C = \left\{d_{lb}^1, d_{ub}^1\right\} \times \cdots \times \left\{d_{lb}^m, d_{ub}^m\right\} \tag{5}$$

$$d_{ub}^{agg} = \max_{c \in C} \mathrm{agg}(c) \tag{6}$$

Wie zuvor erwähnt, lässt sich die Berechnung der aggregierten Grenzen in GeVAS und M^2-Baum entsprechend anpassen.

Es kann gezeigt werden, dass alle mithilfe von CQQL erzeugten arithmetischen Formeln eine der beschriebenen Monotonien erfüllen. Die Monotonie kann dabei allein anhand der Syntax der Anfrage bestimmt werden. Auf die Darstellung der Beweise dieser Eigenschaften wird an dieser Stelle aus Platzgründen verzichtet.

6.2 Metrisches Filter-Refinement

Im Gegensatz zu GeVAS werden die Signaturen beim *metrischen Filter-Refinement* nicht anhand der Featuredaten der Datenbankobjekte ermittelt, sondern ergeben sich aus den Distanzen der Datenbankobjekte zu Referenzobjekten. Jede Signaturdatei besteht daher aus den Bitsignaturen der Distanzen jedes Datenbankobjekts zu einer Menge von Referenzobjekten. Die Reihenfolge der Datenbankobjekte ist dabei in allen Signaturdateien gleich. Die kNN-Suche verläuft analog zu GeVAS, wobei der Ausschluss von Objekten jedoch anhand aggregierter Ähnlichkeitsgrenzen stattfindet.

[6]Die Beschreibungen lassen sich analog auf Ähnlichkeitsgrenzen anwenden.

Für die Auswahl geeigneter Referenzobjekte stehen verschiedene Verfahren zur Verfügung, darunter die zufällige Auswahl oder die inkrementelle Auswahl entfernter Objekte [3]. Um möglichst enge Grenzen zu garantieren, nutzt jedes Datenbankobjekt eine dynamisch ausgewählte Teilmenge seiner nächsten Referenzobjekte.

Bei der Indexerzeugung werden für einen repräsentativen Ausschnitt der Datenbank die paarweisen Distanzen je Feature bestimmt. Ein *Equi-Height-Histogramm* [10] dieser Distanzen wird genutzt um Distanzintervalle für jedes Feature zu berechnen. Diese Distanzintervalle dienen dann der Quantisierung der Distanzen zwischen Referenzobjekten und Datenbankobjekten. Statt also zur Indexierung exakte Distanzen zu speichern, werden die exakten Distanzen durch nummerierte Distanzintervalle repräsentiert (vgl. [4]). Die kompakte Darstellung dieser Nummern durch Bitsignaturen verringert den Speicherverbrauch und erhöht gleichzeitig die Sucheffizienz in der Filterphase, da weniger Daten von der Festplatte gelesen werden müssen. Der Approximationsfehler der Distanzgrenzen steigt jedoch durch die Verwendung von Distanzintervallen. Die Festlegung der Anzahl an Bits pro Signatur entspricht daher der Steuerung der Genauigkeit der Distanzintervalle.

6.3 Lesefenster

Dem in Abschnitt 5.2.2 erläuterten GeVAS-Problem der sinkenden Sucheffizienz bei der Ablage aller Signaturedateien auf einer Festplatte begegnen wir mit der Einführung eines *Lesefensters*. Statt für jedes Objekt parallel auf alle genutzten Signaturdateien zuzugreifen, wird jede Signaturdatei einzeln in Größe des Lesefensters ausgelesen. Der Vorteil dabei ist, dass längere sequentielle Lesephasen entstehen und weniger Sprünge zwischen den Signaturdateien stattfinden. Allerdings müssen die gelesenen Daten jeweils solange im Hauptspeicher gehalten werden, bis alle genutzten Signaturdateien in Größe des Lesefensters abgearbeitet wurden.

6.4 Begrenzter Hauptspeicher

Ein Problem des Filter-Refinements ist, dass alle Objekte, die in der Filterphase nicht ausgeschlossen werden können, bis zur Verfeinerungsphase in einer nach unterer Distanzgrenze sortierten Kandidatenliste gehalten werden müssen. Für große Datenbanken kann es jedoch vorkommen, dass der vorhandene Hauptspeicher dazu nicht ausreicht. Ein weiteres Lesefenster ermöglicht daher die Einhaltung von festen Grenzen für die Hauptspeichernutzung.

Nach einem in [15] vorgeschlagenen Prinzip werden Filter- und Verfeinerungsphase verschränkt. Die Filterphase wird gestoppt, sobald eine festgelegte Speichergrenze erreicht ist. In der nun folgenden Verfeinerungsphase wird die Kandidatenliste abgearbeitet, bis k vorläufige nächste Nachbarn ermittelt wurden. Damit diese k Objekte nicht verloren gehen, werden sie abschließend in die zuvor geleerte Kandidatenliste eingefügt, bevor die Filterphase wieder an der abgebrochenen Stelle fortgesetzt wird. Der Effizienznachteil dieses Vorgehens ist, dass das sequentielle Lesen der Filterphase regelmäßig unterbrochen wird und durch einen wahlfreien Zugriff wieder fortgesetzt werden muss.

6.5 Steigende Featureanzahl

Steigt die Anzahl der in der Anfrage genutzten Features, steigt der Approximationsfehler bei der Abschätzung der aggregierten Ähnlichkeitsgrenzen, da alle Teildistanzen in Form von Distanzgrenzen eingehen. Für Anfragen mit vielen Features bedeutet dies, dass sich die Anzahl der Objekte, die anhand dieser Grenzen ausgeschlossen werden können, verringert und die Sucheffizienz des Verfahrens sinkt.

Um diesem Problem zu begegnen, werden bei steigender Featureanzahl, statt Distanzgrenzen für jedes einzelne Feature zu nutzen, nur noch für eine Teilmenge der verwendeten Features Di-

stanzgrenzen abgeschätzt. Für alle anderen Features werden direkt die exakten Distanzen berechnet. Aus der exakten Berechnung von Teildistanzen ergibt sich ein Mehraufwand gegenüber der Nutzung von Distanzgrenzen. Dieser Mehraufwand kann jedoch ausgeglichen werden, wenn durch diese exakte Berechnung genügend zusätzliche Objekte ausgeschlossen werden können.

Dieses Prinzip lässt sich in der Filterphase anwenden um mehr Objekte auszuschließen. Analog zum sequentiellen Lesen der Signaturdateien müssen die Features in diesem Fall ebenfalls sequentiell gelesen werden, um die Sucheffizienz in der Filterphase zu garantieren.

Das gleiche Prinzip kann in der Verfeinerungsphase genutzt werden, um die Suche früher und mit weniger exakten Distanzberechnungen abbrechen zu können. Statt für ein Objekt in der Verfeinerungsphase direkt alle Teildistanzen auf einmal exakt zu berechnen und zu aggregieren, wird schrittweise vorgegangen. Jedes Mal, wenn ein Objekt in der Verfeinerungsphase am Anfang der Kandidatenliste steht, für das noch nicht alle Teildistanzen berechnet wurden, wird eine Teildistanz berechnet und das Objekt anhand der neuen (genaueren) aggregierten Ähnlichkeitsgrenze wieder in die Kandidatenliste einsortiert. Ein Vorteil ergibt sich dann, wenn der Ausschluss eines Objekts von nur wenigen Teildistanzen abhängt. Bei einer geeigneten Reihenfolge der berechneten Teildistanzen müssen dann nur diese diskriminierenden Distanzen exakt berechnet werden, die Berechnung aller weiteren Teildistanzen kann gespart werden.

Die Auswahl der Features, für die exakte Distanzen berechnet werden, erfolgt statisch (zur Indexierungszeit) oder dynamisch (zur Anfragezeit) nach unterschiedlichen Kriterien, wie der (intrinsischen) Dimensionalität der Features oder der Berechnungsdauer der zugehörigen Distanzfunktion.

6.6 Indexauswahl

Die Sucheffizienz des beschriebenen Verfahrens hängt besonders von der Anzahl der verwendeten Features und der daraus resultierenden (intrinsischen) Dimensionalität ab. Hierarchische Verfahren können bei niedriger (intrinsischer) Dimensionalität, aufgrund der hohen Lokalität der Daten, größere Mengen von Objekten auf einmal ausschließen und dadurch effizienter als nicht-hierarchische Verfahren sein. Ein Vergleich des entworfenen Konzepts mit dem angepassten M^2-Baum dient daher zur Bestimmung der Schnittpunkte bezüglich der Sucheffizienz beider Ansätze. Eine Selektion des effizienteren Index kann dann entweder bereits bei der Indexerzeugung oder erst zur Anfragezeit anhand der in der Anfrage genutzten Features und ihrer (intrinsischen) Dimensionalität stattfinden. Überschreitet die (intrinsische) Dimensionalität eine bestimmte Grenze, ist unter Umständen ein Rückfall auf den linearen Scan sinnvoll.

7. ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Konzept zur flexiblen Indexierung für Ähnlichkeitssuche mit logikbasierten Multi-Feature-Anfragen vorgestellt. Dazu wurden die spezifischen Anforderungen an geeignete Indexierungsverfahren definiert und der Stand der Technik im Bezug auf diese Anforderungen analysiert. Das entwickelte Konzept zur Indexierung basiert auf einer Übertragung und Anpassung des GeVAS-Ansatzes auf den metrischen Raum. Der Index ist dadurch unabhängig von der genutzten logischen Kombination, der Art und Struktur der verwendeten Features und unterstützt eine Vielzahl unterschiedlicher Features und Distanzfunktionen zur Berechnung der Unähnlichkeit.

Als zukünftige Arbeiten verbleiben Teile der Implementierung, die Bestimmung der optimalen Indexparameter und die Evaluation des Konzeptes anhand von synthetischen und realen Daten. Die Unterstützung von Multi-Objekt-Anfragen sowie von Distanzfunktionen die keine Metriken sind, stellen weitere Herausforderungen dar.

Literatur

[1] Stanislav Barton u. a. “Estimating the indexability of multimedia descriptors for similarity searching”. In: *Adaptivity, Personalization and Fusion of Heterogeneous Information*. RIAO ’10. 2010, S. 84–87.

[2] Klemens Böhm u. a. “Fast Evaluation Techniques for Complex Similarity Queries”. In: *Proceedings of the 27th International Conference on Very Large Data Bases*. VLDB ’01. 2001, S. 211–220.

[3] Benjamin Bustos, Gonzalo Navarro und Edgar Chávez. “Pivot selection techniques for proximity searching in metric spaces”. In: *Pattern Recogn. Lett.* 24 (14 2003), S. 2357–2366.

[4] Benjamin Bustos und Tomáš Skopal. “Dynamic similarity search in multi-metric spaces”. In: *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*. MIR ’06. 2006, S. 137–146.

[5] Edgar Chávez u. a. “Searching in metric spaces”. In: *ACM Comput. Surv.* 33 (3 2001), S. 273–321.

[6] Paolo Ciaccia und Marco Patella. “The M^2-tree: Processing Complex Multi-Feature Queries with Just One Index”. In: *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*. 2000.

[7] Paolo Ciaccia, Marco Patella und Pavel Zezula. “M-tree: An Efficient Access Method for Similarity Search in Metric Spaces”. In: *VLDB’97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*. Hrsg. von Matthias Jarke u. a. 1997, S. 426–435.

[8] Ronald Fagin, Amnon Lotem und Moni Naor. “Optimal aggregation algorithms for middleware”. In: *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS ’01. 2001, S. 102–113.

[9] Antonin Guttman. “R-trees: a dynamic index structure for spatial searching”. In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. SIGMOD ’84. 1984, S. 47–57.

[10] Yannis Ioannidis. “The history of histograms (abridged)”. In: *Proceedings of the 29th international conference on Very large data bases - Volume 29*. VLDB ’03. 2003, S. 19–30.

[11] Yossi Rubner, Carlo Tomasi und Leonidas J. Guibas. “The Earth Mover’s Distance as a Metric for Image Retrieval”. In: *Int. J. Comput. Vision* 40 (2 2000), S. 99–121.

[12] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. 2005.

[13] Ingo Schmitt. *Ähnlichkeitssuche in Multimedia-Datenbanken - Retrieval, Suchalgorithmen und Anfragebehandlung*. 2005.

[14] Ingo Schmitt. “QQL: A DB&IR Query Language”. In: *The VLDB Journal* 17 (1 2008), S. 39–56.

[15] Ingo Schmitt und Sören Balko. “Filter ranking in high-dimensional space”. In: *Data Knowl. Eng.* 56 (3 2006), S. 245–286.

[16] Enrique Vidal. “An algorithm for finding nearest neighbours in (approximately) constant average time”. In: *Pattern Recognition Letters* 4.3 (1986), S. 145 –157.

[17] Roger Weber, Hans-Jörg Schek und Stephen Blott. “A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces”. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB ’98. 1998, S. 194–205.

[18] Lofti A. Zadeh. “Fuzzy Logic”. In: *Computer* 21 (1988), S. 83–93.

[19] David Zellhöfer und Ingo Schmitt. “A preference-based approach for interactive weight learning: learning weights within a logic-based query language”. In: *Distributed and Parallel Databases* 27 (1 2010), S. 31–51.

[20] David Zellhöfer und Ingo Schmitt. “Approaching Multimedia Retrieval from a Polyrepresentative Perspective”. In: *Adaptive Multimedia Retrieval. Context, Exploration, and Fusion*. Hrsg. von Marcin Detyniecki u. a. Bd. 6817. Lecture Notes in Computer Science. 2011, S. 46–60.