# Neue Indexingverfahren für die Ähnlichkeitssuche in metrischen Räumen über großen Datenmengen

**Dissertation (Kurzfassung)**
zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Wirtsch.Inf. Dipl.-Softwaretechn. Steffen Guhlemann

Betreuender Hochschullehrer Sen.-Prof. Dr.habil. Uwe Petersohn

Dresden, Juli 2015

# 1 Einleitung

Der Umgang mit *Ähnlichkeit* ist Kernbestandteil des Bewusstseins und Denkens. [28] Daher wird es auch in der Informatik zu einem zunehmend wichtigen Thema mit Anwendung in einer großen Anzahl unterschiedlicher Domänen, wie z.B.

- Analogieschließen zur Lösung eines Problems auf Basis *ähnlicher*, bereits bekannter Fälle (bspw. für Case-based Reasoning),
- Wiederfinden verrauschter Daten bspw. bei der Gesichts- oder Fingerabdruckerkennung,
- Ähnlichkeitssuche bspw. zum Auffinden von Gensequenzen in Gendatenbanken, sowie
- Query-by-Example in Multimedia-Datenbanken.

In allen diesen Anwendungen wird eine Daten- und Infrastruktur benötigt, die große Mengen von Objekten dauerhaft speichern und ähnliche Objekte zu einem Anfrageobjekt effizient auffinden kann. Gegenüber klassischen strukturierten Datenbankabfragen existieren eine Reihe fundamentaler Unterschiede.

- Im Fall der Ähnlichkeitsindizierung muss eine große Bandbreite sehr komplexer Datentypen mit sehr heterogenen Distanzfunktionen unterstützt werden. Eine bestimmte Struktur der Daten kann mithin nicht zur Indizierung herangezogen werden.
- Weiterhin gelten implizite Annahmen eindimensionaler Indices nicht mehr. Bspw.
  - ist die Balance eines Suchbaumes kein Garant eines effizienten Zugriffs mehr, und
  - der Zeitaufwand einer Abfrage wird eher durch die Anzahl an Distanzberechnungen denn durch die notwendigen Externspeicherzugriffe bestimmt.

Derzeit existiert keine einheitliche, universell verfügbare und verwendbare Infrastruktur für die Ähnlichkeitssuche in großen Datenmengen. Eine Reihe von Datenstrukturen erfüllt Teilanforderungen eines solchen Index oder ist nur in bestimmten Domänen einsetzbar. Aktuelle Anwendungen der Ähnlichkeitssuche behelfen sich entweder damit, eine ad hoc Lösung selbst zu entwickeln oder existierende, einfach verfügbare Datenstrukturen zu verwenden und dabei bewusst oder unbewusst Nachteile in Kauf zu nehmen. Bspw. wird manchmal auf einfach verfügbare Indices wie den kd-Baum [3] zurückgegriffen. Damit bestimmt der Index die Repräsentation der Daten (kardinaler Vektor) und die Distanzfunktion (euklidischer Abstand), obwohl in den meisten Domänen eine andere Metrik und Datenrepräsentation geeigneter ist.

**Ziel.** In dieser Arbeit wird die Grundlage für eine universelle Infrastruktur zur Indizierung der Ähnlichkeitssuche großer Datenbestände geschaffen, die nahezu keine Einschränkungen an die unterliegenden Daten oder die verwendete Distanzfunktion stellt. Eine solche Infrastruktur kann analog zum R-Baum [15] als zusätzliche Indexoption in Datenbankmanagementsystemen angeboten oder in Logiksysteme eingebettet [7] werden. Dafür wird nach ausführlicher Analyse eine existierende Indexstruktur – der M-Baum [6] – als Basis gewählt. Dieser wird in dieser Arbeit in vielfältiger Hinsicht zum EM-Baum verbessert, um den Anforderungen einer universell einsetzbaren, effizienten Infrastruktur gerecht zu werden.

**Anforderungen.** Funktionale Anforderungen an ein solches Systems sind:

- dauerhafte Speicherung von teil- oder unstrukturierten Daten beliebiger Domänen,
- inkrementelle Änderung der gespeicherten Datenmenge sowie
- Beantwortung von Ähnlichkeitsanfragen wie
  - Bereichssuchen ("Alle gespeicherten Objekte, die von Anfrageobjekt nicht weiter als eine Grenzentfernung entfernt sind.") und
  - ($k$) Nächsten Nachbar-Suchen ("die $k$ nächsten gespeicherten Objekte zum Anfrageobjekt").

Nicht-Funktionale Anforderungen an die Performance eines solchen Systems sind:

- Lineare Speicherkomplexität $O(N)$: Systeme mit superlinearer Speicherkomplexität wie bspw. *AESA* [27] mit $O(N^2)$ sind für eine universelle Indexstruktur sind sie ungeeignet.
- Sublineare Komplexität von Änderungsoperationen.
- Sublineare Komplexität der Beantwortung von Ähnlichkeitsanfragen.
- Domänenabhängige Optimierbarkeit des Aufwandes von Änderungs- und Anfrageoperationen.[1]
- Erreichen der Performance, die durch Beschränkung auf die intrinsische Dimensionalität möglich wäre.[2]

Die zu indizierenden Daten können beliebigen Domänen entstammen – es bestehen keine Anforderungen an deren Struktur. Das einzige Merkmal, welches zur Beantwortung der Anfragen und der Indizierung herangezogen werden kann, ist eine für eine Instanz der Indexstruktur unveränderliche Distanzfunktion.[3]

# 2 Der M-Baum als Basis einer universellen Indexstruktur für die Ähnlichkeitssuche

## 2.1 Eignung als universelle Indexstruktur

Für die Indizierung der Ähnlichkeitssuche in metrischer Räumen existiert eine Vielzahl von Verfahren. Die meisten davon besitzen Einschränkungen, die es bspw. nicht erlauben, diese Verfahren für beliebige metrische Räume einzusetzen. Wie in der Arbeit gezeigt wird, erscheint ein Indizierungsprinzip in jeder Hinsicht am vielversprechendsten: In einer Baumstruktur werden Daten naheliegenden Pivot-Elementen zugeordnet. Die Baumknoten umfassen einen hyperkugelförmigen Raumbereich. Aus diesem Grund wird im Gegensatz zu den meisten ein- und mehrdimensionalen Verfahren der Raum weder vollständig noch überlappungsfrei unterteilt.

Der M-Baum [6] folgt diesem Prinzip, um Ähnlichkeitssuchen in metrischen Räumen mit kontinuierlicher Distanzfunktion zu indizieren. Damit ist eine sehr diskriminative, rekursive Unterteilung der Datenelemente möglich. Es werden nicht nur der Aufwand für Entfernungsberechnungen, sondern. auch sonstige Zugriffszeiten und der I/O-Aufwand minimiert. Der Baum ermöglicht sowohl den sehr effizienten initialen Aufbau bei a priori bekannten Daten als auch effiziente inkrementelle Änderungen. Aus diesen Gründen eignet sich der M-Baum gut als Basis einer universellen Indexstruktur für die Ähnlichkeitssuche in metrischen Räumen.

## 2.2 Prinzip

Der M-Baum ist ein paginierter Mehrweg-Blattbaum. Die Knoten entsprechen hierarchisch geschachtelten Hyperkugeln. Ein Knoten $n$ wird somit geometrisch durch sein Zentrumsobjekt $c_n$ sowie einen Hüllradius $r_n$ beschrieben. Auf den oberen Baumebenen enthält dieser kleinere M-Baum-Knoten (Kindknoten), auf der Blattebene Datenelemente. Sowohl Kindknoten als auch Datenelemente müssen vollständig innerhalb der Knotenhyperkugel liegen. D.h. unterhalb eines Knotens $n$ dürfen nur Datensätze gespeichert werden, die zum Knotenzentrum $c_n$ nicht weiter als $r_n$ entfernt sind. Das Prinzip verdeutlicht Abbildung 1.

Ein Blattknoten enthält dabei den eigentlichen Schlüsselwert, die Entfernung zu seinem direkten Elternknoten sowie einen Zeiger auf den vollständigen Datensatz. Ein innerer Knoten enthält keinen Datensatz, dafür jedoch Verweise auf alle direkten Kinderknoten sowie den Radius des Teilbaums.

[1] Bei multimedialen Objekten übersteigt der Aufwand einer Entfernungsberechnung teilweise den Aufwand, das entsprechende Objekt von der Festplatte zu laden. In derartigen Anwendungen muss der Fokus der Ähnlichkeitsindizierung auf der strikten Vermeidung von Entfernungsberechnungen liegen. Dies ist bspw. möglich, indem wichtige Entfernungen vorberechnet und gespeichert werden. Diese Optimierung erhöht jedoch den Speicher- und damit auch den I/O-Aufwand. Je nach Domäne dominiert ein anderer Aufwand die notwendige Zeit zur Beantwortung einer Anfrage. Entsprechend muss es möglich sein, mit einfachen konfigurativen Mitteln entweder den I/O-Aufwand oder die Anzahl der Entfernungsberechnungen zu minimieren.

[2] Daten in Vektorräumen unterliegen dem sogenannten "Curse of Dimensionality". In hochdimensionalen Vektorräumen ist eine effiziente Ähnlichkeitsindizierung gleichverteilter Daten unabhängig vom verwendeten Indexprinzip nahezu unmöglich. In realen Anwendungen entsprechen die Dimensionen des Vektorraums kardinalen Eigenschaften der Daten, die oft stark korreliert sind. Damit ist die tatsächliche (intrinsische) Dimensionalität der Daten, die bspw. durch eine Principal Component Analyse [19] gefunden werden kann, deutlich niedriger als die nominale. Gute Indexverfahren sollten hier bei direkter Indizierung der Daten die höhere Effizienz erreichen, die nach vorheriger Reduktion auf die intrinsische Dimensionalität möglich gewesen wäre.

[3] Es existieren auch Indexstrukturen (bspw. das $A_0$-Verfahren [10, 9], der $M^2$-Baum [5] oder ein logikbasierter Ansatz für gewichtete euklidische Distanzfunktionen [31]), die in begrenztem Rahmen mit variablen Distanzfunktionen umgehen können. Diese bauen jedoch auf einfachen Indexstrukturen mit konstanten Distanzfunktionen auf.

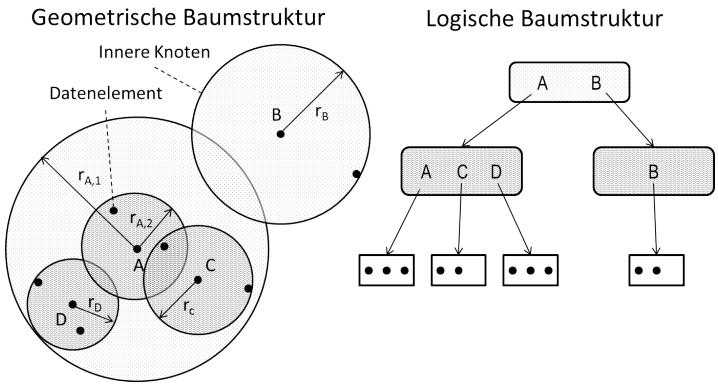


Abbildung 1: Prinzip M-Baum: Auf Wurzelebene existieren zwei Koten (Zentren $A$ bzw. $B$). $A$ enthält drei kleinere Kindknoten ($A$, $C$ und $D$) – einer davon hat das selbe Zentrum ($A$). Diese Kindknoten enthalten die eigentlichen Datenelemente.

Bei der Suche wird rekursiv im Baum abgestiegen. Bei jeder Teilbaumwurzel wird mit Hilfe der Dreiecksungleichung überprüft, ob die Suchhyperkugel die Hyperkugel des Teilbaums überlappt. Falls dies nicht der Fall ist, kann der Teilbaum verworfen werden. Weitere Einsparungen von Entfernungsberechnungen werden dadurch erreicht, dass jeder Knoten die Entfernung zu seinem Elternknoten speichert. Auf Basis dieser Information kann unter Umständen bereits ohne explizite Entfernungsberechnung entschieden werden, dass die Teilbaum-Hyperkugel von der Anfrage nicht betroffen ist.

Beim Einfügen eines neuen Datensatzes lehnt sich das Verfahren am B-Baum an. Neue Daten werden in ein Blatt eingefügt. Ist dessen Kapazität überschritten, wird es geteilt und die beiden neuen Teilbaumwurzeln werden in den Elternknoten des Blattes eingefügt. So wächst der Baum von unten.

### 2.3 Kritik

Die meisten Indexverfahren unterteilen den Raum vollständig und überlappungsfrei. Um eine bessere Diskriminativität bzgl. hyperkugelförmiger Anfragen zu erreichen, wählt der M-Baum ein anderes Vorgehen. Analog zum R-Baum [15] für ausgedehnte Objekte in euklidischen Vektorräumen können sich Baumknoten sowohl überlappen als auch freien Raum dazwischen lassen. Dieses Vorgehen invalidiert die sonst üblichen auf dem eindimensionalen B-Baum [22] und der exakten Suche beruhenden Effizienzanalysen. Auch erhöht sich die Anzahl möglicher Knotenunterteilungen stark, da im Prinzip jede Auswahl von Datenelementen als Pivot-Element dienen und theoretisch jedes Datenelement jedem Pivot-Element zugeordnet werden kann.

Insgesamt ist die Struktur eines konkreten M-Baums also deutlich komplexer zu bewerten als etwa die eines B- oder kd-Baums. Zusätzlich gibt es einen weit größeren Gestaltungsspielraum durch Variationen im Einfügealgorithmus. Leider wird das original von Ciaccia et al. [6] vorgestellte Verfahren zum Baumaufbau der Komplexität der notwendigen Entscheidungen nicht gerecht. An vielen Stellen werden Heuristiken in Anlehnung an Eigenschaften des B-Baums (bspw. Balance) verwendet, um den Raum möglicher Knotenunterteilungen zu reduzieren. Auch die in den Knoten gespeicherten Entfernungen zum jeweiligen Elternknoten werden nur teilweise verwendet, um bei Anfragen Entfernungsberechnungen einzusparen.

## 3 Erweiterung des M-Baums zum EM-Baum.

Wie in Kapitel 2 dargestellt wurde, eignet sich der M-Baum [6] gut als Basis einer universell einsetzbaren Infrastruktur zur Indizierung der Ähnlichkeitssuche in metrische Räumen. Es wurde jedoch auch herausgearbeitet, dass der originale M-Baum noch sehr großes Potential für Effizienz-Verbesserungen besitzt, da die ursprünglich vorgestellten Einfüge- und Suchalgorithmen der hohen Entscheidungskomplexität des M-Baums nicht gerecht werden. Aus diesen Gründen wird in dieser Arbeit der Baum zunächst theoretisch analysiert. Auf dieser Basis werden neue Einfüge- und Suchalgorithmen entwickelt, so dass der dann deutlich effizientere, verbesserte EM-Baum als eigentliche Basis der Indizierung der Ähnlichkeitssuche dienen kann.

Für den M-Baum existieren eine große Zahl von Erweiterungen (bspw. [30, 5, 4, 1, 2, 29, 24, 23, 25, 18]). Damit ist es u.a. möglich, viele Daten auf einmal effizient in den Baum einzufügen (Bulk Loading) oder

verschiedene Distanzfunktionen in einer Indexstruktur zu speichern. Um das Potential dieser Erweiterungen zu erhalten, wurde der M-Baum im Rahmen dieser Arbeit strukturkompatibel zum EM-Baum erweitert. D.h. alle Algorithmen für den M-Baum können unverändert für den EM-Baum verwendet werden.[4]

In dieser Arbeit konnte der M-Baum in mehreren Aspekten verbessert werden.

1. Verbesserung der Suchalgorithmen, so dass eine effizientere Anfragebeantwortung bei gleichbleibender Baumstruktur möglich ist. [14]
   (a) Verbesserte Bereichssuche
   (b) Verbesserte (k)-Nächste-Nachbarn-Suche
   (c) Neue domänenabhängige Entfernungsschranke zur optionalen Verwendung in Suchalgorithmen.
2. Verbesserung des Struktur des M-Baums (unter Erhaltung der Strukturkompatibilität), so dass existierende Suchalgorithmen über den selben gespeicherten Daten effizienter beantwortet werden können. [13]
   (a) Mathematisch fundierte Analyse der Anfrageeffizienz in Abhängigkeit von der Baumstruktur
   (b) Allgemeine Strukturverbesserungen
   (c) Verbessertes Verfahren zum Baumaufbau aus a priori bekannten Daten (Bulk Loading)
   (d) Verbessertes Verfahren für das inkrementelle Einfügen neuer Daten
      i. Optimierte Blattauswahl
      ii. Optimierte Knotenteilung (Auswahl der Teilbaumwurzeln und Aufteilung der Kindknoten)

## 3.1 Verbesserung der Suchalgorithmen

### 3.1.1 Grundlagen

Die Effizienz jeder Baumsuche basiert darauf, dass an Knoten komplette Äste von der weiteren Suche ausgeschlossen werden können. Hierzu muss aufgrund der grundsätzlichen Baumstruktur gesichert sein, dass der Teilbaum keine suchrelevanten Datenelemente enthält. Beim M-Baum kann ein Teilbaum von der Suche ausgeschlossen werden, wenn dessen Hyperkugel die Suchhyperkugel nicht schneidet. I.d.R. ist dazu eine Berechnung der Entfernung der Zentren von Knoten- und Anfragehyperkugel notwendig.

Bei klassischen Suchbäumen (wie dem B- [22] oder kd-Baum [3]) wird implizit davon ausgegangen, dass Operationen der Baumtraverse gleichwertig mit dem Aufwand einer Knotenanfrageprüfung bzgl. Rechenaufwand sind. Entsprechend existieren für klassische Suchbäume keine Optimierungen, die über Baumknoten iterieren, ohne eine Knotenanfrageprüfung auszuführen. Im Fall metrischer Bäume gilt diese Annahme nicht mehr. Eine Knotenanfrageprüfung involviert i.d.R. eine verglichen mit der Baumiteration sehr aufwendige Entfernungsberechnung. Entsprechend sind hier andere Optimierungen denkbar. Im Rahmen dieser Arbeit wurde bspw. eine Optimierung vorgestellt, die nicht nur prüft, ob Knoten- und Anfragehyperkugel sich schneiden, sondern auch, ob die Anfragehyperkugel die Knotenhyperkugel umschließt. In diesem Fall sind alle Elemente des Teilbaums dieses Baumknotens Teil der Ergebnismenge, d.h. alle Datenelemente können ohne Entfernungsberechnung in die Ergebnismenge übernommen werden.

### 3.1.2 Typen von Anfragen

Es werden zwei grundlegende Typen von Suchanfragen unterschieden – komplexere Anfragen lassen sich daraus konstruieren:

- Bei der Bereichssuche werden Elemente gesucht, die von einem Anfrageelement $q$ nicht weiter als ein Maximalabstand $r_q$ entfernt sind. Damit entspricht die Anfrage einer exakt definierten Hyperkugel (Zentrum $q$, Radius $r_q$). Grundprinzip der Bereichssuche ist ein hierarchischer Baumabstieg, bei dem an jedem Knoten $n$ entschieden wird, ob dessen Hyperkugel von der Anfragehyperkugel überlappt wird. Die Reihenfolge des Baumabstiegs ist irrelevant, so dass aus Gründen des Hauptspeicherbedarfs die Tiefensuche präferiert werden sollte.

[4] Beim A0-Verfahren [10] (Suche mit variabler Ähnlichkeitsfunktion) muss die Basis-Indexstruktur (bspw. M-Baum) alle Elemente nach Ähnlichkeit sortiert (lazy evaluiert) zurückgeben. Ein für den klassischen M-Baum entwickelter Algorithmus kann unverändert auf den EM-Baum angewendet werden und läuft dort aufgrund der verbesserten Baumstruktur deutlich schneller.

- Bei der $k$-Nächsten-Nachbarn-Suche werden die $k$ nächsten Elemente zu einem Anfrageobjekt $q$ gesucht. Dies entspricht einer Bereichssuche um $q$, deren Radius gerade so groß ist, dass genau $k$ Elemente Teil der Ergebnismenge sind. Bei diesem Suchtyp wird der Suchradius schrittweise mit jedem untersuchten Datenelement weiter beschränkt.[5] Entsprechend kann für ein Datenelement initial nicht endgültig entschieden werden, ob dieses Teil der Ergebnismenge ist. Die Effizienz dieser Suche ist abhängig von der Reihenfolge des Baumabstiegs. Ein Teilbaum, der bei initial großer Suchradiusschranke noch untersucht werden müsste, könnte unter Umständen im späteren Suchverlauf ausgeschlossen werden. Bei der Optimierung dieser Suche muss also neben den für die Bereichssuche wichtigen Optimierungen eine Reihenfolge des Baumabstiegs so gewählt werden, dass

  - der Suchradius schnell auf einen kleinen Wert beschränkt werden kann sowie
  - Knoten, die potentiell ausgeschlossen werden können, erst spät untersucht werden.

### 3.1.3 Existierende Optimierungen

Der Suchaufwand wird vom Aufwand für Entfernungsberechnungen im Rahmen von Knotenanfrageprüfungen dominiert. Eine Grundidee verschiedener Optimierungen der Bereichssuche ist es, ohne tatsächliche (aufwändige) Entfernungsberechnung, die Entfernung zwischen Knotenzentrum und Anfrageobjekt nach unten zu beschränken. Wenn dies im Vergleich mit einer Entfernungsberechnung sehr effizient möglich ist, kann ein Teilbaum unter Umständen bereits auf Basis der Entfernungsschranke von der Suche ausgeschlossen werden.

Frühere Erweiterungen des M-Baums stellen hier drei verschiedene Entfernungsschranken vor:

- *Elternentfernungsheuristik*: Eine Idee von [6] ist, die während des Einfügens ohnehin bestimmte Entfernung $d_{n,p}$ eines Knotenzentrums $n$ zum Zentrum seines Elternknotens $p$ in $n$ zu speichern. Nachdem bei einer Anfrage mit Zentrum $q$ der Elternknoten $p$ expandiert (und somit die Entfernung $d_{q,p}$bestimmt) wurde, kann über die Dreiecksungleichung die Entfernung $d_{q,n}$ beschränkt werden.

- *AESA-Schranken*: Analog zu AESA [27] schlagen [1] vor, in jedem Knoten alle bilateralen Entfernungen der Kindknoten zu speichern. Damit kann bei sequentieller Untersuchung der Kindknoten eines Knotens die Entfernung zu später untersuchten Kindknoten immer besser eingeschränkt werden. Der verbesserten Sucheffizienz stehen jedoch auch einige Nachteile gegenüber. So verschlechtern sich Speicherkomplexität und Einfügeaufwand, die Entfernung zum Elternknoten selbst ist nicht unbedingt gespeichert und der Baum ist nicht mehr kompatibel zum originalen M-Baum.

- *Domänenabhängige Schranken*: Für manche metrische Räume existieren effizient bestimmbare Entfernungsschranken. [2] schlagen für die Domäne der Editierdistanz nach Levenshtein [16] die *Bag-Heuristik* vor. Damit kann eine Entfernungsschranke zweier Texte (Länge $M$ bzw. $N$) in $O(N+M)$ bestimmen, während für eine tatsächliche Entfernungsberechnung $O(N \cdot M)$ benötigt werden. Da die Heuristik jedoch nicht in $O(1)$ bestimmt werden kann, verursacht sie einen nicht unerheblichen Aufwand, so dass abhängig von der üblichen Länge indizierter Texte der Zeitaufwand für die Suche sogar steigen kann.

### 3.1.4 Allgemeine Optimierung der Suchalgorithmen

Im Rahmen dieser Arbeit wurden zunächst gemeinsame Aspekte verschiedener Suchalgorithmen optimiert:

- Bislang wurden nur ausgewählte Entfernungsschranken fest im Algorithmus integriert. Neu ist die generische Nutzung einer beliebigen Zahl domänenabhängiger und domänenunabhängiger Entfernungsschranken. Im Algorithmus wird das Minimum verschiedener unterer Entfernungsschranken unter Beachtung ihrer Bestimmungsaufwandes genutzt. Alle in $O(1)$ bestimmbaren Schranken werden direkt verwendet. Aufwändigere Schranken wie die Bag-Heuristik werden abhängig von Domäne und konkreter Datenverteilung konfigurativ eingesetzt, nachdem die effizient bestimmbaren Schranken keinen Ausschluss eines Knotens ermöglichen.

- Neben unteren (Symbol $d^{\perp}$) können auch obere ($d^{\top}$) Entfernungsschranken eingesetzt werden. Es ist möglich, Suchschritte ohne tatsächliche Entfernungsberechnung durchzuführen:

[5] Wenn $k$ Datenelemente untersucht wurden, muss kein Element berücksichtigt werden, welches weiter entfernt liegt als diese $k$ Elemente. Darum kann der Suchradius auf die Entfernung zum derzeit bekannten $k$t nächsten Element beschränkt werden.

  - Mit Hilfe von $d^{\top}$ kann festgestellt werden, dass ein Knoten von der Suchhyperkugel umschlossen wird. (Alle Datenelemente werden ungeprüft in die Ergebnismenge übernommen.)
  - Durch Kombination von $d^{\perp}$ und $d^{\top}$kann festgestellt werden, dass ein Knoten von der Suchhyperkugel geschnitten wird. (Die Kindknoten werden untersucht.)
  - Falls $d^{\perp} = d^{\top}$, ist auch die tatsächliche Entfernung (ohne Berechnung) zu den beiden Schranken identisch. (Die tatsächliche Entfernung liegt zwischen beiden Schranken: $d_n^{\perp} \leq d_{n,q} \leq d_n^{\top}$.)

- In dieser Arbeit wurde die *Längen-Heuristik* für die Domäne der Levenshtein-Editierdistanz entworfen. Sie ist etwas ungenauer als die Bag-Heuristik, jedoch in $O(1)$ bestimmbar. Die *Längen-Heuristik* erwies sich in eigenen Experimenten als beste Heuristik für die Indizierung der Domäne.

- Da der M-Baum [6] wie ein B-Baum [22] von unten nach oben wächst, ohne wie dieser balanciert geteilt zu werden, neigt er zu "Luftwurzeln". Dies sind Ketten von Knoten mit jeweils einem Kind, um zu erzwingen, dass sich alle Blattknoten auf einer Ebene befinden. Hierfür wurde eine Suchoptimierung vorgestellt, die bei derartigen Ketten nur die Entfernung zum letzten Knoten der Kette bestimmt.

### 3.1.5 Spezielle Optimierung der Bereichssuche

Bei der Bereichssuche werden am klassischen Algorithmus folgende Veränderungen vorgenommen:

- Verwendung mehrerer Schranken:
  - Kombination aller effizient berechenbaren Heuristiken zu einer oberen und unteren Schranke.
  - Ist damit keine Entscheidung möglich, potentielle Nutzung ineffizient berechenbarer Heuristiken.
  - Ist immer noch keine Entscheidung gefallen, Bestimmung der tatsächlichen Entfernung.

- Upper Bound Enclosure: Auf Basis der oberen Entfernungsschranke oder auch der Entfernung kann festgestellt werden, dass ein M-Baum-Knoten vollständig innerhalb der Suchhyperkugel liegt. In diesem Fall werden alle Datenelemente unter diesem Knoten ungeprüft in die Ergebnismenge übernommen.

- Zero Intervall: Sind obere und untere Schranke gleich, wird die Entfernungsberechnung eingespart.[6]

- Upper Bound Intersection: Nur mit Hilfe der oberen und unteren Entfernungsschranke ist es manchmal möglich festzustellen, dass ein innerer Baumknoten geschnitten wird. In diesem Fall werden die Kindknoten untersucht ohne die Entfernung zum Elternknoten zu bestimmen. Diese Optimierung wird jedoch nicht in jeder Domäne angewandt, da die Abschätzung der Elternentfernungsheuristik zu Kindknoten weniger genau möglich ist, wenn die exakte Entfernung zum Elternknoten unbekannt ist.[7]

- One Child Cut: Bei Untersuchung eines "Luftwurzel"-Knotens wird ohne Entfernungsberechnung direkt der eine Kindknoten anstelle des Elternknotens untersucht.

In eigenen Experimenten wurde eine Verringerung der Anzahl an Entfernungsberechnungen durch die vorgestellten Optimierungen um 50% festgestellt, wie in Abbildung 2 dargestellt ist.[8]

[6] In diesem Fall entspricht die tatsächliche Entfernung $d$ dem Wert der beiden Schranken $d^{\perp}$ und $d^{\top}$.

[7] Bei ungenauer Elternentfernungsheuristik können manchmal Kindknoten nicht direkt von der Suche ausgeschlossen werden, die andernfalls sofort verworfen worden wären. D.h. es wird zwar eine Entfernungsberechnung (zum Elternknoten) eingespart, dafür sind jedoch viele zusätzliche Entfernungsberechnungen zu Kindknoten notwendig. Die negativen Auswirkungen dieses Effektes werden durch die Existenz anderer guter Heuristiken neben der Elternentfernungsheuristik abgemildert. Aus diesem Grund sollte diese Optimierung vorzugsweise in Domänen mit domänenspezifischen Heuristiken angewandt werden.

[8] In unterschiedlichen metrischen Räumen ist aufgrund des Curse of Dimensionality eine sehr unterschiedliche Anzahl an Entfernungsberechnungen notwendig. Um in allen metrischen Räumen einen guten Überblick über den Vergleich verschiedener Strategien zu ermöglichen, wurde die Anzahl an Entfernungsberechnungen in einem metrischen Raum jeweils an der notwendigen Anzahl für eine Basisstrategie relativiert (Wert 1). Eine Strategie, die in einem metrischen Raum 20% weniger Entfernungsberechnungen benötigt als die Basisstrategie, wird mit 0,8 dargestellt. Dies gilt für alle folgenden Diagramme.

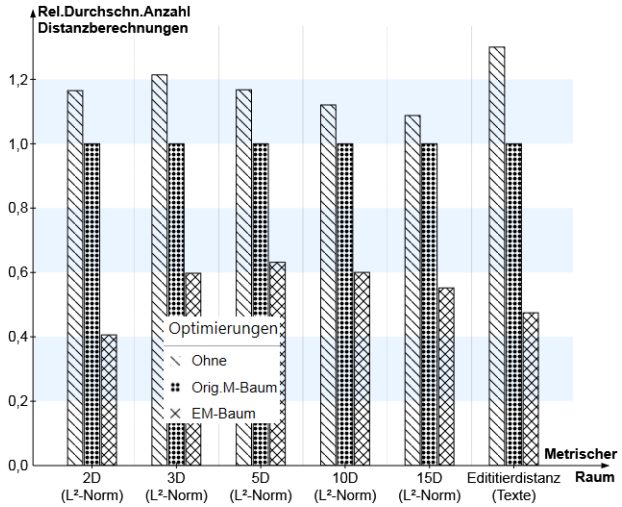


Abbildung 2: Bereichssuche: Auswirkung unterschiedlicher Optimierungen in verschiedenen metrischen Räumen. (Durchschnittliche Anzahl Distanzberechnungen pro Anfrage, relativiert an der Anzahl der originalen M-Baum-Strategie, 10.000 gespeicherte Elementen.)

### 3.1.6 Spezielle Optimierung der k-Nächste-Nachbarn-Suche

Die $k$-Nächste-Nachbarn-Suche (kNN) entspricht einer Bereichssuche mit initial unbekanntem Radius. Dementsprechend unterscheidet sich der Suchalgorithmus in drei Aspekten von der Bereichssuche:

- Im Suchverlauf ist anstelle des finalen Suchradius nur eine obere Schranke dieses Radius bekannt. Entsprechend sind Datenelemente innerhalb dieses Radius keine endgültigen Ergebnisse, sondern nur Ergebniskandidaten. Die Ergebniskandidatenmenge wird bei jedem Suchfortschritt aktualisiert.
- Die Suche muss fortlaufend die aktuelle Beschränkung des Suchradius bestimmen.
- Die Effizienz der Suche ist abhängig von der Reihenfolge, in der die Knoten untersucht werden.

**Einfluss der Reihenfolge der Baumtraverse.** Bei der Optimierung der kNN-Suche werden nicht nur isoliert einzelne Abstandsbestimmungen vermieden, sondern auch die Knotenexpansionsreihenfolge verbessert. Die Schranke des Suchradius sinkt im Suchverlauf mit den untersuchten Datenelementen. Es existieren also Baumknoten, die im Fall einer frühen Untersuchung noch expandiert werden müssten, aber bei später Untersuchung verworfen werden könnten. Solche Knoten sollten spät im Suchverlauf untersucht werden.

Zwei Aspekte führen dabei zu einem geringeren Suchaufwand:

1. Geschwindigkeit der Einschränkung des Suchradius: Je früher nahe Datenelemente gefunden werden, desto schneller sinkt der Suchradius und um so mehr Knoten können ausgeschlossen werden.
2. Effektivität der Einschränkung: Wenn entfernt liegende Knoten spät untersucht werden, steigt mit der engeren Schranke die Wahrscheinlichkeit, sie ohne Expansion ausschließen zu können.

Für beide Aspekte ist es optimal, jeweils die räumlich vielversprechenden Knoten zuerst zu expandieren, d.h. Knoten mit minimalen potentiellen Abstand enthaltener Datenelemente zum Anfrageobjekt. In der Arbeit wird bewiesen, dass bei Anwendung dieser Strategie die selbe Anzahl an Knoten expandiert wird, wie auch bei einer äquivalenten Bereichssuche hätte expandiert werden müssen. Damit ist diese Strategie optimal.

**Frühzeitige Reduktion des Suchradius.** Da die Anzahl der expandierten Knoten nur von Expansionsreihenfolge abhängt, können andere Optimierungen zur beschleunigten Reduktion des Suchradius[9] keine Entfernungsberechnungen vermeiden. Derartige Optimierungen können unter Umständen dennoch angewendet werden, da sie die Größe der Expansionswarteschlange im Suchverlauf reduzieren. Dies führt zu einem verringerten Hauptspeicher- sowie Sortieraufwand (sonstiger CPU-Aufwand).

[9] Auf Basis des größtmöglichen Abstandes von Datenelementen unter einem Knoten kann postuliert werden, dass mindestens ein solches Datenelement existiert. Werden $k$ derartige Aussagen kombiniert, kann der finale Suchradius beschränkt werden.

**Nutzung von Schranken im klassischen kNN-Algorithmus.** Im klassischen Algorithmus [6] werden bei Expansion eines Knotens die Entfernungen $d_{q,c}$ seiner Kindknoten zum Anfrageobjekt bestimmt. Damit werden untere Entfernungsschranken $\perp_c$ zu Datenelementen unter den Kindknoten berechnet. Auf Basis dieser $\perp_c$ werden die Kindknoten in die Expansionswarteschlange eingefügt, sofern $\perp_c$ nicht den derzeitigen Suchradius $r_q$ übersteigt. Weiterhin wird vor der tatsächlichen Bestimmung einer Entfernung $d_{q,c}$ mit Hilfe der Elternentfernungsheuristik eine untere Entfernungsschranke und somit auch eine ungenaue untere Schranke $\perp'_c$ bestimmt. In seltenen Fällen ist $\perp'_c$ größer als $r_q$ und der Kindknoten kann direkt verworfen werden.

**Optimierung durch bessere Schrankennutzung.** Dieser Ablauf besitzt Optimierungspotential. Für einen zu untersuchenden Knoten $c$ wird zunächst eine ungenaue untere Schranke $\perp'_c \leq \perp_c$ ohne Entfernungsberechnung bestimmt (Nutzung der effizient bestimmbaren unteren Entfernungsschranken) und dieser in die Warteschlange eingefügt. Wird $c$ später von der Warteschlange entnommen, so wird die genaue Schranke $\perp_c$ (Entfernungsberechnung) bestimmt, um $c$ erneut einzufügen. Bei nochmaliger Entnahme wird $c$ expandiert.

**Weitere Optimierungen.** Weitere neue Optimierungen dieser Arbeit sind:

- Wird die generelle M-Baum-Optimierung "Verkleinerte Hüllradien" (Abschnitt 3.3) angewendet, so können Kindknoten aus dem Elternknoten herausragen. Aufgrund der Konsistenzeigenschaft können in dem herausragenden Bereich keine Datenelemente liegen. Die minimal mögliche Entfernung zu Datenelementen $\perp_c$ unter einem Kindknoten $c$ kann deshalb genauer spezifiziert werden, wenn auch die $\perp_p$ unter allen Vorfahrenknoten $p$ dieses Knotens berücksichtigt werden.

- Bei der "Optimierung durch bessere Schrankennutzung" werden immer noch überflüssige Entfernungen bestimmt. Dies kann vermieden werden, wenn ein Knoten nach der Erstentnahme nicht erneut auf der Warteschlange abgelegt, sondern direkt nach Entfernungsberechnung expandiert wird. Diese Optimierung sollte jedoch nur optional abhängig von der Domäne eingesetzt werden. Sie minimiert zwar die Anzahl an Entfernungsberechnungen, expandiert jedoch dafür überflüssige Knoten. In Domänen mit relativ effizienter Entfernungsberechnung kann durch den Expansionsaufwand und die größere Warteschlange der gesamte Zeitaufwand steigen.

- Zur Reduktion der Größe der Expansionswarteschlange (nicht jedoch der Anzahl an Knotenexpansionen und Entfernungsberechnungen) kann auf Basis der oberen Entfernungsschranke die maximale Entfernung zu Datenelementen unter einem Knoten abgeschätzt werden (Frühzeitige Suchradiusreduktion).

- Weiterhin können die meisten Optimierungen der Bereichssuche angewandt werden: One Child Cut, Zero Intervall, Verwendung mehrerer Schranken.

In eigenen Experimenten konnte validiert werden, dass durch die neuen Optimierungen ein Großteil der notwendigen Entfernungsberechnungen eingespart wird (siehe Abbildung 3).

## 3.2 Analyse der Anfrageeffizienz in Abhängigkeit von der Baumstruktur

Für den M-Baum existiert bislang keine mathematisch fundierte Analyse des Einflusses der Baumstruktur auf die Anfrageeffizienz. Diese ist jedoch notwendig, um bei Einfügeentscheidungen die bessere Struktur präferieren zu können. Die Anfrageeffizienz wird als erwartete Anzahl an Entfernungsberechnungen bei der Bereichssuche quantifiziert, da andere Suchtypen als Spezialfall der Bereichssuche aufgefasst werden können.

**Erwartete Suchkosten der Bereichssuche.** Ähnlichkeitssuchen steigen im Baum hierarchisch ab. Bei jedem untersuchten Teilbaum sind potentiell einige Entfernungsberechnungen zu Kindknoten notwendig, um entscheiden zu können, welche Kindknoten weiter verfolgt werden müssen. Die erwarteten Suchkosten $E[sc]$ entsprechen somit der Summe der an jedem Teilbaum $k$ erwarteten Anzahl an Entfernungsberechnungen unabhängig von der Einordnung des Teilbaums in den Gesamtbaum:

$$E[sc_{Mtree}] = \sum_k E[sc_k].$$

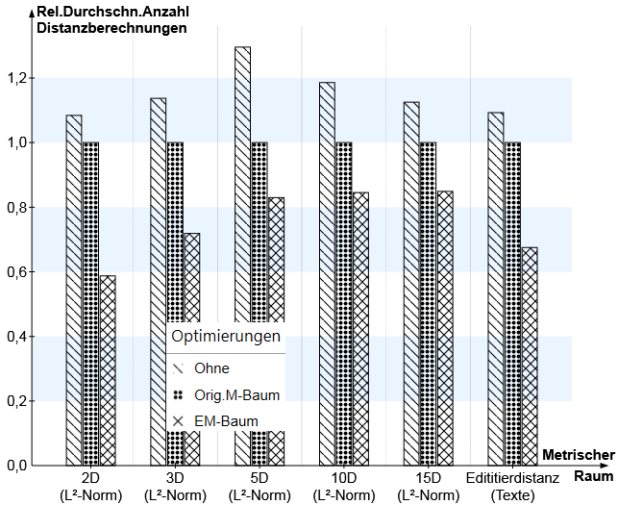


Abbildung 3: 3-Nächste-Nachbarn-Suche: Auswirkung unterschiedlicher Optimierungen in verschiedenen metrischen Räumen. (Durchschnittliche Anzahl Distanzberechnungen pro Anfrage, relativiert an der Anzahl der originalen M-Baum-Strategie, 10.000 gespeicherte Elemente)

**Erwartete Suchkosten der Bereichssuche für einen Knoten $k$.** Bei der Untersuchung einer Teilbaum-Hyperkugel $k$ wird geprüft, ob $k$ von einer Suchanfrage-Hyperkugel geschnitten wird. Dazu muss die Entfernung des Knotenzentrums von $k$ zum Anfrageelement $q$ bestimmt werden.

- Besteht ein Schnitt der beiden Hyperkugeln, so kann $k$ Ergebnisse der Suchanfrage enthalten. Diese können in jedem Kinderknoten von $k$ enthalten sein. Deshalb werden alle Kindknoten untersucht – dazu muss die Entfernung von $q$ zu jedem Kindknotenzentrum bestimmt werden.
- Gibt es keine Überlappung der Hyperkugeln, so enthält $k$ keine Suchergebnisse. Der Teilbaum $k$ kann mit allen Kindknoten ohne weitere Entfernungsberechnungen für diese Suche komplett verworfen werden.

Die potentiellen Entfernungsbestimmungen zu Kindknoten treten nur auf, falls $k$ selbst untersucht wird. (Dies ist abhängig von der Lage von $k$ im Gesamtbaum.) Damit sind vier Fälle zu unterscheiden:

1. $k$ wird untersucht und nicht überlappt. Es müssen keine Entfernungen zu Kindknoten, aber eine zu $k$ bestimmt werden. Diese Entfernungsberechnung ist jedoch dem Elternknoten von $k$ zuzuschreiben.
2. $k$ wird untersucht und überlappt. Es müssen Entfernungen zu allen $card_k$ Kindknoten sowie zu $k$ selbst bestimmt werden. Die letzte Entfernungsberechnung ist dem Elternknoten von $k$ zuzuschreiben.
3. $k$ wird weder untersucht noch überlappt. Es sind keinerlei Entfernungsberechnungen notwendig.
4. $k$ wird nicht untersucht, aber überlappt. Somit werden potentiell Ergebnisse unter $k$ nicht gefunden. Baumaufbaualgorithmen stellen sicher, dass eine solche inkonsistente Struktur nicht auftreten kann.

Wenn die Entfernungsberechnung zu $k$ selbst dessen Elternknoten zugeschrieben wird, verursacht $k$ also nur im zweiten Fall $card_k$ Entfernungsberechnungen. Unabhängig von der Lage eines Knotens $k$ im Gesamtbaum sind also die von $k$ verursachten erwarteten Kosten nur von der Knotenkardinalität $card_k$ und der Wahrscheinlichkeit abhängig, dass $k$ von einer beliebigen Such-Hyperkugel geschnitten wird:

$$E[sc_k] = P(geschnitten(k)) \cdot card_k.$$

Die Wahrscheinlichkeit von Überlappungen einer Knoten-Hyperkugel $k$ mit beliebigen Such-Hyperkugeln wächst mit dem Knotenradius $r_k$. Somit können die durch $k$ verursachten, erwarteten Suchkosten wie folgt in Form eines $Perf$-Wertes quantifiziert werden:[10]

$$E[sc_k] \sim Perf_k := card_k \cdot (r_k + a).$$

[10] $a$ ist eine domänenabhängige Konstante, die in der Arbeit ausführlich erläutert wird.

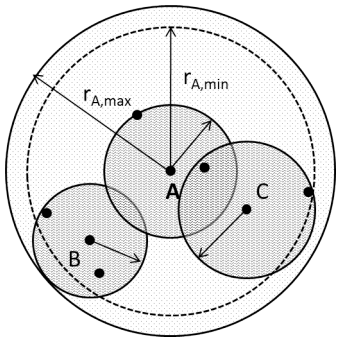


Abbildung 4: Verkleinerte Hüllradien: M-Baum-Knoten mit drei Kindknoten ($A$, $B$ und $C$) und Datenelementen. Im klassischen M-Baum [6] werden Knotenhyperkugeln umschlossen. Sollen nur die Datenelemente umschlossen werden, so ist es möglich, den Radius zu verkleinern (gestrichelt). In diesem Fall ragen manche innere Knoten aufgrund ihrer stärkeren Krümmung aus der Elternhyperkugel heraus.

Der erwartete Gesamtsuchaufwand wird durch die Summe der $Perf$-Werte der Einzelknoten ausgedrückt:

$$E[sc_{Mtree}] \sim Perf_{Mtree} := \sum_k Perf_k = \sum_k card_k \cdot (r_k + a).$$

Mit $Perf$ ist eine nicht-heuristische Steuerung des Einfügeprozesses möglich. Unter verschiedenen Optionen wird jeweils die gewählt, die den $Perf$-Wert des entstehenden Baumes minimiert.

## 3.3 Verkleinerte Hüllradien

Da der $E[sc_k]$ vom $r_k$ abhängt (siehe Abschnitt 3.2), kann eine Reduktion von Knotenradien bei gleichbleibender Baumstruktur den durchschnittlichen Suchaufwand eines M-Baums reduzieren.

**Prinzip.** Basis der effizienten Suche im EM-Baum ist der Ausschluss kompletter Teilbäume aufgrund einer Nicht-Überlappung mit der Anfragehyperkugel. Dies ist nur dann sicher möglich, wenn sich alle Datenelemente unterhalb eines Knotens vollständig in dessen Hyperkugel befinden.

Im klassischen M-Baum [6] wird dies sichergestellt, indem jeder Knoten $k$ rekursiv alle Kindknoten umschließt. Damit ist der Knotenradius $r_k$ jedoch größer als notwendig – er kann soweit reduziert werden, dass nur noch alle enthaltenen Datenelemente umschlossen werden (siehe Abbildung 4). Mit diesem verringertem Knotenradius bei ansonsten identischer Baumstruktur können einige Suchanfragen bereits an Knoten $k$ abgebrochen werden, der erwartete Anfrageaufwand wird reduziert.

**Schwierigkeiten der Umsetzung.** Im klassischen M-Baum ist die Bestimmung von Hüllradien während des Einfügens stets ohne Entfernungsberechnung möglich (Aggregation der Radien zum Umschließen der Kindknotenhyperkugeln). Für die Bestimmung des verkleinerten Knotenradius eines Knotens $k$ müssen dagegen die Entfernungen von $k$ zu allen Datenelementen unter $k$ aggregiert werden. Deren Anzahl ist i.d.R. nicht beschränkt. Weiterhin werden Entfernungen aggregiert, die im Baum nicht gespeichert sind. Aus diesen Gründen würde eine ständige naive Neuberechnung des Hüllradius wie im klassischen M-Baum zu einer starken Zunahme von Entfernungsberechnungen während des Einfügens führen.

**Verkleinerte Hüllradien im Kontext von Bulk-Loading.** Im Rahmen eines Bulk-Loading-Verfahrens [4] (siehe Abschnitt 3.4) werden ohnehin alle Entfernungen aller unter einem Knoten gespeicherten Datenelemente zum Knotenzentrum bestimmt. Damit ist implizit der minimal mögliche Hüllradius gegeben.

**Verkleinerte Hüllradien im Kontext inkrementeller Einfügevorgänge.** Im Fall inkrementeller Änderungsvorgänge sind die notwendigen Entfernungen meist nicht verfügbar. Dennoch konnte im Rahmen dieser Arbeit die Verkleinerung der Hüllradien auf diesen Anwendungsfall ausgeweitet werden. Dabei wird mit einer Kombination verschiedener Strategien sichergestellt, dass die notwendige Anzahl an Entfernungsberechnungen pro Einfügevorgang nicht oder nur wenig ansteigt:

- Zunächst wird analytisch festgestellt, welche Knotenradien durch den Änderungsvorgang betroffen sind. In den meisten Fällen betrifft dies die Knoten auf dem Pfad vom gewählten Blattknoten zur Wurzel.
- Wenn möglich, wird der minimale Knotenradius inkrementell fortgeschrieben.
  - Während des Einfügens eines Datenelementes $e$ wird durch eine hierarchisch absteigende Suche ein Blattknoten zur Speicherung von $e$ gesucht. Im Rahmen dieser Suche werden zu allen Knoten $k$ auf dem Pfad zur Wurzel Entfernungen $d_{k,e}$ bestimmt. Diese können direkt genutzt werden, um die Radien dieser Knoten inkrementell anzupassen: $r_{k,neu} = \max\{r, d_{k,e}\}$.
  - Verschiedene Operationen verändern den minimalen Knotenradius nicht (bspw. löschen eines nicht maximal entfernten Datenelementes, Knotenteilung oder oberhalb eines Knotens, ...).
- Manchmal ist eine inkrementelle Fortschreibung des Knotenradius nicht möglich. Ein Beispiel hierfür ist der Promote-Schritt der Knotenteilung. Dabei wird ein Knoten $k$ neues Zentrum eines Teilbaums. Als dessen Elemente in den Baum eingefügt wurden, lag $k$ noch nicht auf dem Pfad zur Wurzel, so dass die Entfernungen zu $k$ nie bestimmt wurden. Zur Minimierung des Radius von $k$ müssen diese Entfernungen zusätzlich bestimmt werden – ein verringerter Abfrage- muss also einem erhöhten Einfügeaufwand gegenübergestellt werden. Da je nach Domäne der optimale Kompromiss anders ausfällt, kann konfigurativ eine von zwei Strategien verwendet werden:
  - *Stark reduzierte Radien* (Optimaler Abfrage-, aber domänenabhängig höherer Einfügeaufwand): In der Arbeit wurde ein Algorithmus zur effizienten Suche des entferntesten Elementes (invers zur Nächsten-Nachbarn-Suche) vorgestellt. Für einen Knoten, dessen minimaler Radius nach einer Umstrukturierung unbekannt ist, kann damit effizient der minimale Hüllradius bestimmt werden.
  - *Effizient reduzierte Radien* (Optimaler Einfüge-, aber suboptimaler Abfrageaufwand): Der genannte Algorithmus ist zwar effizient, verursacht aber dennoch zusätzliche Entfernungsberechnungen. Um diese zu vermeiden kann anstelle eines perfekt minimierten Radius auch die Umhüllung der Kindknotenhyperkugeln genutzt werden. Durch diesen Mischansatz entsteht ein Baum dessen Knotenradien zwischen dem klassischen M-Baum und minimalen Radien liegen.[11]

In eigenen Experimenten konnte validiert werden, dass die Baumgüte durch Minimierung der Knotenradien deutlich verbessert werden kann. Wie in Abbildung 5 ersichtlich, trifft dies nicht nur auf den Abfrageaufwand zu (wie analytisch vorhergesagt werden kann), sondern auch für den Einfügeaufwand. Dabei treten zwei widersprüchliche Effekte auf. Einerseits verursacht die Bestimmung stark reduzierter Knotenradien zusätzliche Entfernungsberechnungen. Andererseits benötigen suchende Einfügeoperationen (bspw. Bestimmung des Blattknotens) weniger Entfernungsberechnungen.

## 3.4 Bulk Loading

Während in CBR-Systemen wie iSuite[12] ein steter Wechsel zwischen Änderungs- und Anfrageoperationen besteht, sind bei einem Klassifizierungsystem[13] alle Trainingsdaten von Anfang an bekannt. In einem solchen Fall können anstelle inkrementeller Einfügeverfahren auch Spezialverfahren (Bulk Loading) verwendet werden, die einen deutlich geringeren Einfüge- bei gleichzeitig sehr gutem Abfrageaufwand ermöglichen.

**Klassisches Verfahren.** Mit Bulk Loading [4] wird aus a priori bekannten Daten ein kompatibler[14] M-Baum aufgebaut. Gegenüber inkrementellen Einfügeverfahren bietet Bulk Loading drei Vorteile:

- Für den Baumaufbau sind deutlich weniger Entfernungsberechnungen notwendig.

[11] Einige Knotenradien werden inkrementell fortgeschrieben und sind somit minimal. Andere Radien umschließen analog zum klassischen M-Baum ihre Kindknoten gerade. Dadurch dass der Baum auch minimale Radien enthält, sind auch die umhüllenden Radien gegenüber dem klassischen M-Baum reduziert. Dies setzt sich rekursiv durch alle Baumebenen fort. Später können durch weitere Einfügevorgänge umschließende Radien wieder minimal werden.

[12] Das iSuite-System [17, 12] schlägt auf Basis einer Kombination von generischem Domänenwissen, konzept- und fallbasiertem Schließen [8, 20, 21] über dem episodischen Wissen einem Arzt Behandlungsschritte vor. Eine Teilaufgabe ist dabei die Verwaltung und schnelle Suche ähnlicher bereits bekannter Fälle zu einem neuen Patientenfall.

[13] Bei der Klassifizierung mittels der $k$ nächsten Nachbarn wird zur Klassifizierung eines neuen Beispiels die Mehrheitsklasse ähnlicher Trainingsbeispiele herangezogen. Die Suche nach ähnlichen Trainingsbeispielen kann mit einer Datenstruktur wie dem EM-Baum beschleunigt werden. Da sich die indizierten Daten nicht ändern, können sie en bloc eingefügt werden.

[14] Ein kompatibler M-Baum ermöglicht die Nutzung von Standardalgorithmen zum Suchen und inkrementellen Einfügen.

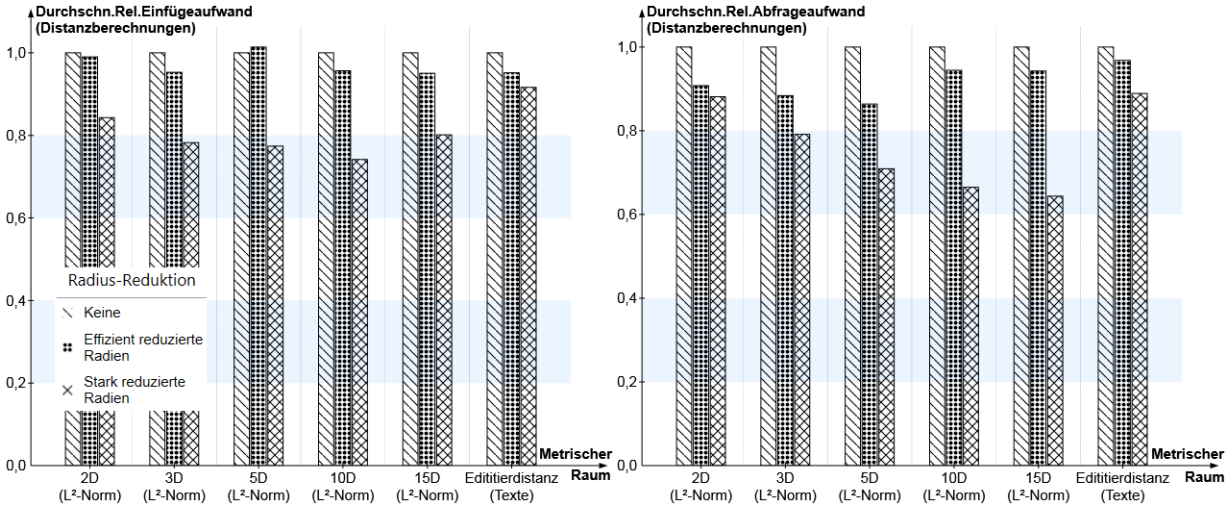


Abbildung 5: Level der Radienreduktion (Einfügestrategie *"Min.Cost"*, verschiedene metrische Räume, jeweils 10.000 gespeicherte Elemente). Links Einfügeaufwand pro Element, rechts Abfrageaufwand pro Abfrage. (Durchschnittliche Anzahl Entfernungsberechnungen relativiert an Strategie "*Stark konsistente Radien*".)

- Ohne zusätzliche Entfernungsberechnungen besitzt der Baum verkleinerte Hüllradien (Abschnitt 3.3).
- Es wird eine relativ gut geclusterte Baumstruktur erreicht.[15]

Grundprinzip ist die zufällige Auswahl von Teilbaumwurzeln und die anschließende Zuordnung der restlichen Datenelemente zur nächstliegenden Teilbaumwurzel. Ist die Knotenkapazität in einer Teilbaumwurzel überschritten, wird die Aufteilung rekursiv fortgesetzt. Zuletzt schließen sich zwei Umstrukturierungen an:

- Entfernung unterausgelasteter Knoten sowie
- Erzwingen gleicher Blatttiefen.

Im Rahmen dieser Arbeit wurde das Verfahren zunächst analysiert. Dabei wurde festgestellt, dass

- die zwei Umstrukturierungen unabhängig voneinander optional sind,
- eine davon (Erzwingen gleicher Blatttiefen) ausschließlich negative Effekte besitzt und
- der Hauptschritt in Teilen dem Clusterverfahrens PARTITIONING AROUND MEDOIDS entspricht.

**Erweiterung im EM-Baum.** Eine deutliche Verbesserung der Baumstruktur wird dadurch erreicht, dass einer der beiden optionalen Schritte nicht durchgeführt wird.

Weiterhin entspricht das Bulk-Loading in Teilen PARTITIONING AROUND MEDOIDS, bei dem abwechselnd eine Verbesserung der Aufteilung der Clustermitglieder und -zentren bis hin zur Konvergenz durchgeführt wird. Beim klassischen Bulk-Loading wird zwar einmalig die Verbesserung der Aufteilung der Knoten/Cluster-Mitglieder durchgeführt, nicht jedoch die Optimierung der Clusterzentren und die Iteration bis zur Konvergenz. Beim EM-Baum-Bulk-Loading wird zumindest einmalig die Optimierung der Cluster/Knoten-Zentren durchgeführt, wodurch die Knotenradien minimiert werden ohne die sonstige Baumstruktur zu ändern. Im Detail muss dabei entschieden werden, welche Datensätze als potentielles Knotenzentrum untersucht werden. Die domänenabhängige Abwägung zwischen Einfüge- und Abfrageaufwand findet sich in den zwei vorgestellten EM-Baum-Bulk-Loading-Verfahren "Lokale Optimierung" (Testen weniger Datensätze) und "Globale Optimierung" (Testen vieler Datensätze) wieder. In jedem Fall wird die Baumstruktur deutlich verbessert, während der Einfügeaufwand nur moderat ansteigt.[16] Abbildung 6 zeigt die experimentellen Ergebnisse.

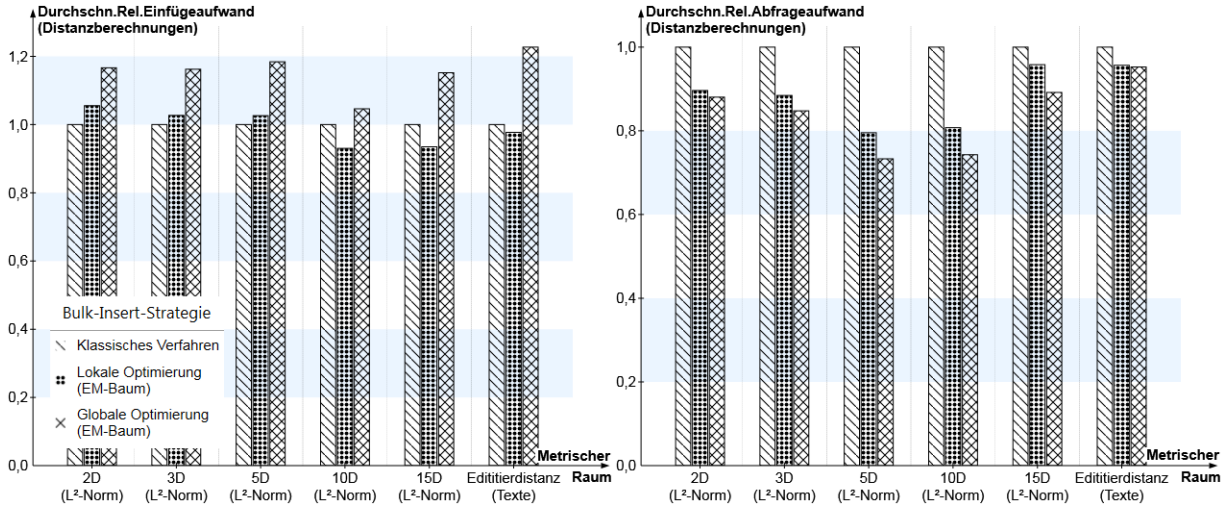


Abbildung 6: Optimierungen beim Bulk-Insert (verschiedene metrische Räume, jeweils 10.000 gespeicherte Elemente) Links Einfügeaufwand pro Element, rechts Abfrageaufwand pro Abfrage (Durchschnittliche Anzahl Distanzberechnungen, relativiert an Strategie "*Klassisches Verfahren*").

## 3.5 Inkrementeller Baumaufbau

Beim inkrementellen Einfügen werden nacheinander einzelne Datensätze eingefügt, so dass der Baum immer in einem konsistenten Zustand ist. Das Grundprinzip des Einfügens ist analog zum B-Baums [22] ein Wachstum von den Blättern zur Wurzel und die gleiche Tiefe aller Blattknoten:

- Zunächst wird zunächst ein Blattknoten gesucht, in welchem der Datensatz gespeichert werden soll.
- Falls die Kapazität des Blattknotens nicht ausreicht, werden der Blattknoten und daraus resultierend potentiell Knoten auf dem Pfad zur Wurzel geteilt. Die Teilung besteht dabei aus den zwei Teilschritten "Auswahl der Schnittposition" (PROMOTE) und "Aufteilung der Unterknoten" (PARTITION).

**Unterschiede zum B-Baum.** Gegenüber dem B-Baum sind jedoch einige Unterschiede zu beachten:

1. Mit dem Einfügen eines Datensatzes müssen (potentiell aufwändig) auch die Radien aller Knoten auf dem Pfad zur Wurzel angepasst werden.
2. Der B-Baum teilt den Domänenraum auf jeder Baumebene vollständig und überlappungsfrei. Weil jeweils genau ein Knoten existiert, der den neuen Datensatz aufnehmen kann, erfolgt die Blattauswahl effizient durch rekursiven Baumabstieg. Beim EM-Baum ist die Raumteilung weder vollständig noch überlappungsfrei. Da den neuen Datensatz auf jeder Baumebene kein, ein oder mehrere Knoten aufnehmen können, ist die Blattauswahl eine nicht-triviale Entscheidung (siehe Abschnitt 3.5.1).
3. Beim B-Baum sind durch die vollständige und überlappungsfreie Raumteilung die Einzelschritte der Knotenteilung trivial. Da für einen solchen Suchbaum eine balancierte Teilung optimal ist, wird am effizient bestimmbaren Median geschnitten. Im Fall des EM-Baum sind beide Teilschritte nicht-trivial (siehe Abschnitte 3.5.2 und 3.5.3):
   (a) Im PROMOTE-Schritt werden zwei neue Teilbaumwurzeln anstelle des alten Elternknotens gewählt. Bei einer Knotenkapazität von $C$ gibt es $O(C^2)$ Möglichkeiten, wobei weder ein eindeutiges Optimalitätskriterium, noch ein Algorithmus ein solches zu bestimmen, existiert.

[15] Die durch BULK LOADING erzeugte Baumstruktur erreicht die Güte der besten klassische inkrementellen Einfügeverfahren, nicht jedoch die Güte der im Rahmen dieser Arbeit vorgestellten Verfahren.

[16] Der Anstieg des Einfügeaufwandes relativiert sich weiter, wenn man ihn mit inkrementellen Einfügeverfahren vergleicht. Bei allen Bulk-Loading-Verfahren ist er verglichen mit inkrementellen Verfahren vernachlässigbar gering (siehe Abbildung 8).

(b) Im PARTITION-Schritt werden Datenelement/Unterknoten auf die beiden neuen Teilbaumwurzeln aufgeteilt. Dafür existieren $O(2^C)$ Möglichkeiten ohne eindeutiges Optimalitätskriterium.

### 3.5.1 Verbesserte Blattauswahl

Im M-Baum kann ein Datensatz in jeden Blattknoten eingefügt werden, wobei manche dafür stark erweitert und andere geteilt werden müssen. Durch die weder vollständige noch überlappungsfreie Teilung lassen sich auf einer hohen Baumebene nur sehr begrenzt Aussagen über die unter einem Teilbaum verfügbaren Blattknoten treffen. [6] führen dennoch eine suboptimale, heuristische Einwege-Suche nach dem Blattknoten durch.

**Optimalität eines Blattknotens.** Der Beitrag eines Blattknotens zu $Perf$ ist durch dessen Kardinalität und Radius bedingt (siehe Abschnitt 3.2). Durch das Einfügen eines Elementes verändert sich dieser Beitrag:

1. Der Abstand zum neuen Datenelement muss bestimmt werden, wenn eine Anfrage den Blattknoten schneidet. Die Häufigkeit dieses Falles steigt mit dem finalen Blattknoten-Radius $r_b$. Es sind also Blattknoten mit kleinem $r_b$ zu bevorzugen.

2. Im Fall einer Radiuserweiterung (potentiell aller Knoten auf dem Weg zur Wurzel) werden die betroffenen Knoten häufiger von Anfragen geschnitten, wodurch Entfernungen zu allen Kindknoten bestimmt werden. Radiuserweiterungen gewichtet mit der Knotenkardinalität sollten also minimiert werden.

**Suche des optimalen Blattknotens.** Im EM-Baum wird mit einer $A^*$-Suche ein Blattknoten gesucht, der $\Delta Perf$ minimiert. Weiterhin wurden andere Möglichkeiten evaluiert, wie bspw.

- das Einfügen in den Blattknoten, der den nächsten Nachbarn des neuen Datensatzes enthält oder
- eine vorgeschaltete Suche nach einem Blattknoten auf dessen Pfad kein Knoten erweitert werden muss.

In eigenen Experimenten wurden Blattauswahlstrategien verglichen (siehe Abbildung 7). Die EM-Baum-Strategien verringern zumeist sowohl den Einfüge- als auch vor allem den Abfrageaufwand deutlich.

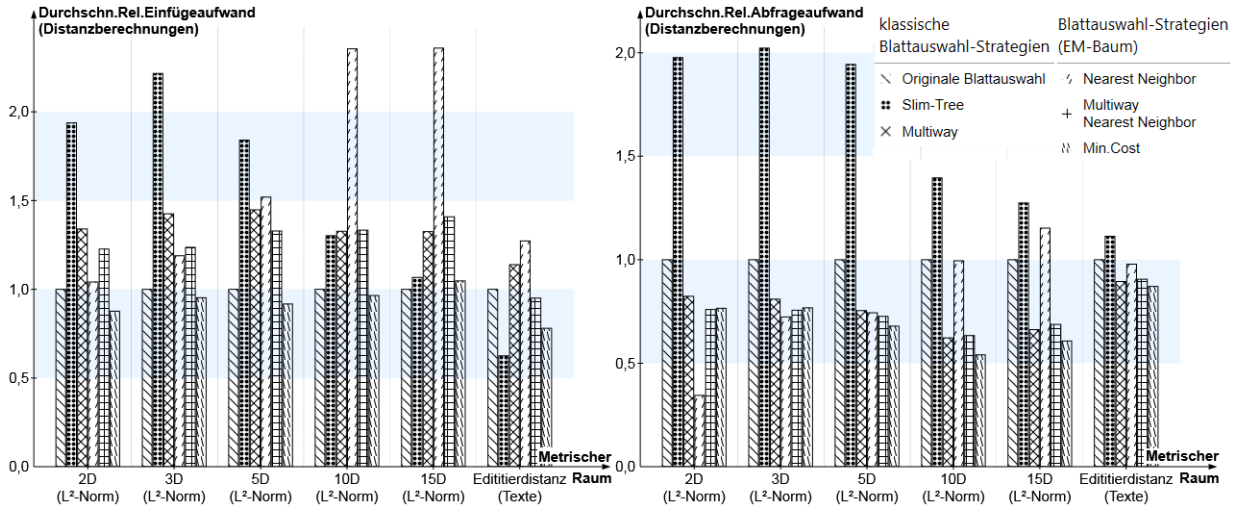


Abbildung 7: Blattauswahl-Strategien (unterschiedliche metrische Räume, jeweils 10.000 gespeicherte Elemente, Knotenteilungsstrategie *Min.Cost,* Reduktion auf minimal konsistente Radien). Links Einfügeaufwand pro Element, rechts Abfrageaufwand pro Abfrage. (Durchschnittliche Anzahl Entfernungsberechnungen, relativiert an Strategie *Originale Blattauswahl.*)

### 3.5.2 Knotenteilung im EM-Baum – Grundlagen

Für die Knotenteilung existieren $O(C^2 \cdot 2^C)$ Möglichkeiten bei einer Knotenkapazität von $C$. Zur Suche des Optimums müssen alle $O(C^2)$ bilateralen Abstände der Kindknotenzentren bestimmt werden. Deshalb werden Heuristiken auf verschiedene Weise eingesetzt, um die Suche zu beschleunigen:

- Vollständig heuristische bzw. zufällige Teilungsalgorithmen.
  - Random [6]: Zufällige Wahl der Teilbaumwurzeln (PROMOTE) und heuristischen PARTITION.
  - M_LB_Dist [6] (ähnlich: M_UB_Dist [18]): PROMOTE: alter Elternknoten und entferntester Kindknoten, heuristische PARTITION. Es werden nur $O(C)$ Entfernungsberechnungen benötigt.
- Gruppe von Strategien mM_XXX_Rad [18]: Beibehaltung des alten Elternknoten. Suche nur im Raum der PROMOTE-Möglichkeiten, dabei Abschätzung der Güte der resultierenden PARTITION ohne Entfernungsberechnung. Final heuristische Partition. Es werden nur $O(C)$ Entfernungsberechnungen benötigt.
- Minimum-Spanning-Tree [25]: Heuristische PARTITION an der längsten Kante eines minimal spannenden Baumes. PROMOTE: Knotenzentrum, welches den Teilbaumradius minimiert.
- Minimierung Gütekriterium (bspw. Minimierung der Radiensumme [6], Minimierung des größeren der beiden neuen Radien [6], Minimierung des $Perf$-Wertes): Es werden alle $O(C^2)$ PROMOTE-Möglichkeiten durchsucht. Für jede PROMOTE-Möglichkeit wird heuristisch eine PARTITION durchgeführt, deren Ergebnis nach dem Gütekriterium bewertet wird.

**Partition-Schritt** Aufgrund des extrem großen Suchraums von $O(2^N)$ PARTITION-Möglichkeiten pro PROMOTE-Aufteilung existieren keine Strategien, die vollständig alle PARTITION-Möglichkeiten durchsuchen. Statt dessen werden heuristische Strategien angewandt:

- Aufteilung der Datenelemente/Unterknoten an einer generalisierten Hyperebene, welche mittig zwischen den beiden neuen Elternknotenzentren liegt. (GHP) [6]
- Balancierte Aufteilung der Datenelemente auf die beiden neuen Elternknoten.[6]
- Nach GHP-PARTITION lokale Suche (Hillclimbing). (potentiell im EM-Baum)

GHP ist sowohl analytisch als auch experimentell der balancierten Teilung weit überlegen. Bei weder vollständiger noch überlappungsfreier Raumteilung sind die Vorteile eines balancierten Baumes marginal.

### 3.5.3 Verbesserte Knotenteilung im EM-Baum

Das EM-Knotenteilungsverfahren besteht aus unabhängig anwendbaren Teilkomponenten:

- Slim-Down-Schritt zur Vermeidung von Knotenteilungen,
- Schrumpfung auf minimal konsistente Radien und Berücksichtigung bei der Teilungsbewertung,
- Minimierung von $Perf$ bei der Suche im Raum der PROMOTE-Möglichkeiten sowie
- Erweiterung der GHP-PARTITION-Strategie um die Suche nach einem lokalen Optimum.

Die Komponenten können beliebig miteinander und mit existierenden Strategien kombiniert werden.

**Slim-Down.** In [25, 26, 24] wurde u.a. ein Verfahren zur Verbesserung der Baumgüte vorgestellt. Der entfernteste Kindknoten $c_e$ eines Knotens $n$ spannt den Radius $r_n$ von $n$ auf. Kann $c_e$ in einen anderen Knoten $b$ umverlagert werden, dessen Radius nicht erweitert und der nicht geteilt werden muss (Slim-Down-Schritt), so kann $r_n$ sinken. [25, 26] diskutieren dabei nur am Rande, wann und für welche Knoten ein Slim-Down-Schritt erwogen wird und welche Knoten als Verschiebeziel herangezogen werden.

Im EM-Baum wird Slim-Down in Anlehnung an ein Verfahren von [11] für R-Bäume [15] zur Teilungsvermeidung eingesetzt. Muss ein Knoten geteilt werden, wird untersucht, ob statt dessen mit einem Slim-Down-Schritt ein Kindknoten in einen Bruderknoten verschoben werden kann. Ist dies möglich, sinkt die Radiensumme, die Knotenteilung wird hinausgezögert und die Balance des Baumes erhöht.

Slim-Down kann positive und negative Effekte auf $Perf$ haben.[17] Eigene Experimente zeigen, dass in manchen metrischen Räumen der Abfrageaufwand etwas reduziert wird, in anderen aber negative Effekte überwiegen. Slim-Down sollte also nur in ausgewählten Domänen eingesetzt werden.

[17] Der positive Effekt beruht auf der Annahme, dass $r_n$ sinkt. Zunächst ist dies nicht sicher, da ein anderes Element genauso weit entfernt sein kann wie $c_e$. Im Fall innerer Knoten (speziell bei verkleinerten Knotenradien) ist es noch schwieriger dies sicherzustellen, da nicht klar ist, in welchem Kindknoten das Datenelement liegt, welches $r_n$ aufspannt. Weiterhin sinkt zwar so die Radiensumme des M-Baums, nicht jedoch zwingend der $Perf$-Wert, da dieser eine gewichtete Radiensumme darstellt.

**Minimal konsistente Knotenradien.** Die Verwendung minimal konsistenter Knotenradien hat erheblichen Einfluss auf die Laufzeit und den Algorithmus zur Durchsuchung der PROMOTE-Möglichkeiten. Bei der Bewertung einer PROMOTE-Möglichkeit müssen die Knotenradien der neuen Elternknoten bestimmt werden. Bei klassischen Knotenradien ist dies einfach und effizient auf Basis der bilateralen Entfernungen der Kindknotenzentren und der Kindknotenradien möglich. Im Fall verkleinerter Knotenradien muss für jeden Elternkandidaten aufwändig das entfernteste Datenelement bestimmt werden. Im Rahmen der Arbeit wurde ein Such- und Cache-Verfahren vorgestellt, welches die zusätzlichen Kosten dieser Bewertung erheblich verringert. Eigene Experimente zeigen, dass damit der Abfrageaufwand erheblich reduziert werden kann.

**Minimierung von $Perf$.** Im Gegensatz zu relativ willkürlich gewählten Zielgrößen wie der Minimierung des größeren der beiden Radien wird der analytisch bestimmte $Perf$-Wert minimiert. In eigenen Experimenten (siehe Arbeit) konnte gezeigt werden, dass diese PROMOTE-Strategie sowohl allen klassischen PROMOTE-Strategien als auch der Minimum-Spanning-Tree-Strategie deutlich überlegen ist.

**Lokal optimierte PARTITION.** Ausgehend von der relativ guten GHP-PARTITION kann diese durch Hillclimbing-Suche weiter verbessert werden. Hierzu wird nach Umverteilungen gesucht, die $Perf$ verbessern, bis ein lokales Optimum erreicht wurde. Für diese Optimierung sind keine zusätzlichen Entfernungsberechnungen, jedoch erheblicher sonstiger Rechenaufwand notwendig. Die potentiell erreichbare verbesserte Baumstruktur muss somit dem erhöhten sonstigen CPU-Aufwand für das Einfügen gegenübergestellt werden.

## 3.6 Gesamtvergleich

Im experimentellen Vergleich der besten klassischen Baumaufbau-Verfahren *m_LB_Rad2* und *Slim-Tree* mit den EM-Verfahren für Bulk Loading (*Bulk*) und inkrementellen Baumaufbau (*Min.Cost*) konnte eine deutliche Verbesserung der Abfrageperformance festgestellt werden (siehe Abbildung 8). Bulk Loading besaß dabei mit Abstand den geringsten Einfügeaufwand. Die resultierende Baumgüte war besser als die der meisten klassischen Verfahren, erreichte jedoch nicht die von *Min.Cost*. Letzteres besitzt einen relativ geringen Aufwand zum Baumaufbau und erzeugt mit Abstand die beste Baumgüte.

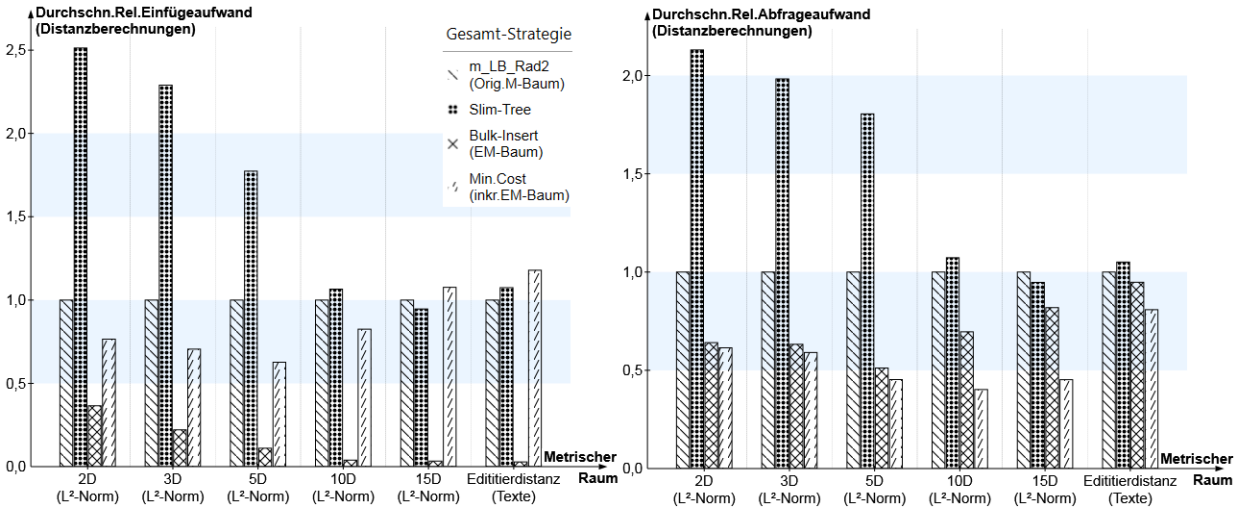


Abbildung 8: Gesamt-Einfügestrategien M-Baum-Erweiterungen und EM-Baum (unterschiedliche metrische Räume, jeweils 10.000 gespeicherte Elemente). Links Einfügeaufwand pro Element, rechts Abfrageaufwand pro Abfrage. (Durchschnittliche Anzahl Entfernungsberechnungen, relativiert an Strategie *m_LB_Rad2*.)

# 4 Zusammenfassung

Thema dieser Arbeit ist die Indizierung der Ähnlichkeitssuche in beliebigen metrischen Räumen. Da dies ein zunehmend wichtiges Thema mit hohen Anforderungen ist, wäre hierzu eine ähnlich universelle und datenunabhängige Infrastruktur notwendig und möglich, wie sie relationale Datenbankmanagementsysteme

für strukturierte Daten bieten. Analog zu anderen Datenbankerweiterungen wie R-Baum-Indices oder Volltextsuche kann die in dieser Arbeit entwickelte Infrastruktur in existierende Datenbankmanagementsysteme integriert und so universell verfügbar gemacht werden.

Anforderungen an eine solche Infrastruktur sind Daten- und Domänenunabhängigkeit, lineare Speicherkomplexität sowie effiziente Änderungs- und Abfrageoperationen. Trotz Datenunabhängigkeit sollte sich die Infrastruktur automatisch und konfigurativ an die Besonderheiten einer Domäne anpassen, um in dieser Domäne die Effizienz speziell dafür entwickelter Indexstrukturen zu erreichen.

Hierfür wurde nach einer Analyse verfügbarer Indexprinzipien der M- zum EM-Baum erweitert und vielfach verbessert. Zunächst wurde dessen Struktur mathematisch analysiert, so dass bei statischer Betrachtung eines Baumes Aussagen über den resultierenden Anfrageaufwand möglich sind. Anschließend wurden der M-Baum und seine Varianten in unabhängig rekombinierbare Strategiemodule zerlegt, um diese theoretisch und experimentell einzeln zu bewerten. Auf Basis der Analyse wurden neue Strategiebausteine entworfen, die eine abfrageeffiziente Baumstruktur erzeugen. Weiterhin wurden verbesserte Algorithmen für Basis-Suchanfragen vorgestellt. Alle Verbesserungen wurden einzeln experimentell validiert.

Damit ist es gelungen, die Grundlagen für die benötigte universelle, datenunabhängige Infrastruktur zur Indizierung der Ähnlichkeitssuche in metrischen Räumen zu legen. Die relevanten Algorithmen sind sowohl im Text als auch als Pseudocode in der Arbeit enthalten.

# Literatur


[1] Lior Aronovich and Israel Spiegler. CM-tree: A dynamic clustered index for similarity search in metric databases. *Data & Knowledge Engineering*, 63(3):919–946, 2007.

[2] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. String matching with metric trees using an approximate distance. In *String Processing and Information Retrieval*, pages 423–431. Springer, 2002.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[4] P. Ciaccia and M. Patella. Bulk loading the M-tree. In *Proceedings of the 9th Australasian Database Conference (ADC 98)*, pages 15–26. Citeseer, 1998.

[5] P. Ciaccia and M. Patella. The M2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, 2000.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES*, pages 426–435, 1997.

[7] Conrad Drescher, Hongkai Liu, Franz Baader, Steffen Guhlemann, Uwe Petersohn, Peter Steinke, and Michael Thielscher. Putting ABox Updates into Action. In *Proceedings of the Seventh International Symposium on Frontiers of Combining Systems (FroCoS 2009)*, Trento, Italy, 2009.

[8] B. Drzymajllo, St. Guhlemann, and L. Iwer. A Knowledge Based Model for Evaluation and Recommendation of Medical Therapies. In *Proceedings of the Russian-German Workshop „Innovation Information Technologies: Theory and Practice“*, Ufa, Russia, July 2009.

[9] R. Fagin. Fuzzy queries in multimedia database systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 1–10. ACM, 1998.

[10] Ronald Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 216–226. ACM, 1996.

[11] Yván J García, Mario A Lopez, and Scott T Leutenegger. On optimal node splitting for R-trees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 334–344. Morgan Kaufmann Publishers Inc., 1998.

[12] K. Gastmeier, U. Petersohn, and St. Guhlemann. iSuite „Chronischer Schmerz“ – Ein Wissensbanksystem zur Dokumentation und Qualitätssicherung. In *Proceedings of Dt. Schmerzkongress*, Berlin, Germany, Oktober 2009.

[13] Steffen Guhlemann. An optimized Insertion Algorithm for the M-Tree based on a mathematical Analysis of the Influence of the Tree-Structure on the resulting Query Performance. To appear.

[14] Steffen Guhlemann. Optimizing Similarity Search in the M-Tree. To appear.

[15] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, 1984.

[16] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics - Doklady 10*, 10:707–710, 1966.

[17] Wolfgang Oertel and Uwe Petersohn. Managing Episodes, Cases, Concepts, and Rules – an Integration Approach for Medical Problem Solving. In D. Aha and J. Daniels, editors, *Case-Based Reasoning Integrations. Papers from the 1998 Workshop*, Madison, Wisconsin, USA, 1998.

[18] Marco Patella. Similarity search in multimedia databases. *Dipartmento di Elettronica Informatica e Sistemistica, Bologna*, 1999.

[19] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[20] U. Petersohn, St. Guhlemann, L. Iwer, and R. Balzer. Concept-based and Case-based Inferences for Medicinal Therapies. In *Proceedings of the 11th international workshop on computer science and information technologies CSIT 2009*, Crete, Greece, 2009.

[21] U. Petersohn, St. Guhlemann, L. Iwer, and R. Balzer. Problem solving with Concept and Case Based Reasoning. In T. Burczyński, W. Cholewa, and W. Moczulski, editors, *Proceedings of the Conference „Recent Developments in Artificial Intelligence Methods“*, AI-METH Series on Artificial Intelligence Methods, pages 259–270, Gliwice, Poland, 2009.

[22] R.Bayer and E.M.McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3), September 1972.

[23] T. Skopal. Pivoting M-tree: A metric access method for efficient similarity search. In *Proceedings of the Dateso 2004 Annual International Workshop on Databases, Texts, Specifications and Objects, Desna, Czech Republic, April 14-16*, pages 27–37. Citeseer, 2004.

[24] T. Skopal, J. Pokornỳ, M. Krátkỳ, and V. Snášel. Revisiting M-tree building principles. In *Advances in Databases and Information Systems*, pages 148–162. Springer, 2003.

[25] C. Traina, A. Traina, B. Seeger, and C. Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. *Advances in Database Technology, EDBT 2000*, pages 51–65, 2000.

[26] C. Traina Jr, A. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization of metric data sets using slim-trees. *Knowledge and Data Engineering, IEEE Transactions on*, 14(2):244–260, 2002.

[27] E. Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986.

[28] Robert Andrew Wilson and Frank C Keil. *The MIT encyclopedia of the cognitive sciences*. MIT press, 2001.

[29] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*. Springer, Berlin, 2006.

[30] Xiangmin Zhou, Guoren Wang, Jeffrey Xu Yu, and Ge Yu. M+-tree: a new dynamical multidimensional index for metric spaces. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 161–168, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[31] M. Zierenberg. Flexible Indexierung für Ähnlichkeitssuche mit logikbasierten Multi-Feature-Anfragen.