DIPLOMARBEIT

Optimiertes Retrieval von Fallbeispielen mit k-d-Bäumen

Frank Goebel

November 1993

Betreuer:
Dipl.-Inf. Stefan Weß

Universität Kaiserslautern
Fachbereich Informatik
AG Künstliche Intelligenz – Expertensysteme

Prof. Dr. Michael M. Richter

SFB 314, Künstliche Intelligenz und wissensbasierte Systeme

Zusammenfassung

In fallbasierten Systemen ist es notwendig, ein über die normalen Datenbank-Suchaufgaben hinausgehendes Retrieval bereitzustellen. Hier müssen die n zu einem Anfragefall ähnlichsten Fälle aus einer Fallbasis gesucht werden.

In dieser Diplomarbeit wird ein solches System zum ähnlichkeitsbasierten Retrieval von Fällen entwickelt. Dieses System übernimmt die Verwaltung der Fälle unter Verwendung der Datenstruktur des k-d-Baumes, hierbei werden die k-d-Bäume so aufgebaut, daß sie in optimaler Weise die sogenannte Best-Match- bzw. Nearest-Neighbour-Suche ermöglichen. Hierbei stand bereits ein existierendes System zur Verfügung, welches diese Suche zwar schon unterstützt, aber noch eine unbefriedigende Performance aufwies.

Die verschiedenen Parameter, mit denen man auf den k-d-Baum Einluß nehmen kann, wurden untersucht und einige Konzepte zur Bestimmung der optimalen Parametereinstellungen entwickelt und implementiert. Es stellte sich heraus, daß die veränderte Berechnung dieser Einstellungen in der Generierungsprozedur für die k-d-Bäume deutlich bessere Ergebnisse als das bisher bestehende System liefert. Die angewendete Vorgehensweise bei der Baumgenerierung nähert sich dem von *Conceptual Clustering*-Systemen an, der entstehende k-d-Baum repräsentiert schon eine gewisse Klassifizierung der Fälle.

Da die veränderte Baumgenerierung mit einem höheren Aufwand verbunden ist, wurde ein *inkrementelles Insert* vorgestellt, mit dem ein effizientes Einfügen von neuen Fällen ermöglicht wird, ohne das der k-d-Baum neu generiert werden muß, um seine Optimalität zu garantieren.

Auf der Retrievalseite wurde das neues Konzept der *virtuellen Bounds* entwickelt, die die Suche wesentlich beschleunigen, schließlich wurden zwei Retrievalvarianten für einen häufig benötigten Spezialfall des Retrievals, das *inkrementelle Retrieval*, vorgestellt. Hierbei werden in besonderer Weise Anfragen unterstützt, die nach und nach verfeinert, d. h. genauer spezifiziert werden. So konnte erreicht werden, daß das erworbene Wissen von den vorherigen Retrievalanfragen genutzt werden kann.

Es wird schließlich gezeigt, daß der k-d-Baum in Verbindung mit geeigneten Generierungs- und Retrieval-Algorithmen eine geeignete und effiziente Unterstützung des ähnlichkeitsbasierten Fallretrievals ermöglicht. Die Anzahl der zu durchsuchenden Fälle kann stark eingeschränkt werden, was sich im Besonderen bei großen Fallbasen bezahlt macht.

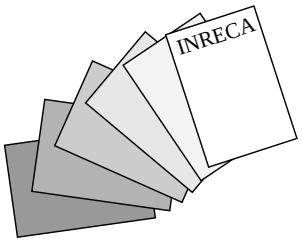
Inhaltsverzeichnis

Kapitel 1

Einführung und Motivation

1.1 Das INRECA-Projekt

Diese Diplomarbeit ist Teil des INRECA-Projektes, welches durch die europäische Gemeinschaft als ESPRIT Projekt gefördert wird. INRECA liefert Werkzeuge und Methoden für die Entwick-



lung, Validierung und Wartung von entscheidungsunterstützenden Systemen. Der INRECA-Ansatz ist intuitiv und leicht verständlich für den Anwender, da er keine Regeln oder ähnliche Ausdrücke einer formalen Sprache benötigt, vielmehr ist die Vorgehensweise der eines Ingenieurs ähnlich: „Hatte ich ein ähnliches Problem nicht schon einmal? Ja! Wie habe ich es denn damals gelöst?“.

INRECA's Basistechnologien sind induktives Schließen und fallbasiertes Schließen. Induktion extrahiert Entscheidungswissen aus Falldaten und hilft, Strukturen und Trends in Fällen zu erkennen. Fallbasiertes Schließen stellt Beziehungen her zwischen dem aktuellen Problem des Anwenders und Erfahrungen aus der Vergangenheit.

INRECA soll beispielsweise für computergestützte Hilfesysteme, Wartungssysyteme und die Qualitätskontrolle in der Produktion eingesetzt werden.

Die Universität Kaiserslautern arbeitet neben den Firmen AcknoSoft (Frankreich, USA), Irish Multimedia Systems (Irland) und tecInno (Deutschland) im INRECA-Projekt mit.

Die vorgestellten Konzepte wurden in Smalltalk implementiert und getestet, und sind im Rahmen der kommerziellen INRECA-Software an der Universität Kaiserslautern reimplementiert worden.

1.2 Aufgaben der Diplomarbeit

Diese Diplomarbeit beschäftigt sich mit dem Teil von INRECA, der mit den Methoden des fallbasierten Schließens (*Case-Based Reasoning*) arbeitet. Es soll ein Modul für INRECA entwickelt werden, welches es ermöglicht, aus einem durch Fälle repräsentierten Erfahrungswissen einen möglichst ähnlichen Fall zu derjenigen Situation aufzufinden, die gelöst werden soll.

In einer früheren Diplomarbeit ([Öch92]) wurde bereits ein Basismodul für das System CAbPlan entwickelt, welches mit dem Ansatz des *fallbasierten Schließens* die Verwaltung der bereits bekannten Fälle und das Retrieval innerhalb dieser Fallbasis unterstützt.

Allerdings wies dieses Modul noch Schwächen in der Performance auf, in dieser Arbeit sollten die Konzepte des vorhandenen Moduls geprüft und verbessert werden.

So wird in dieser Diplomarbeit ein System zum ähnlichkeitsbasierten Retrieval von Fällen entwickelt. Dieses System übernimmt die Verwaltung der Fälle unter Verwendung der Datenstruktur des k-d-Baumes, hierbei werden die k-d-Bäume so aufgebaut, daß sie in optimaler Weise die sogenannte Best-Match- bzw. Nearest-Neighbour-Suche unterstützt.

Die verschiedenen Parameter, mit denen man auf den k-d-Baum Einluß nehmen kann wurden untersucht und verschiedene Konzepte zur Bestimmung der optimalen Parametereinstellungen entwickelt und implementiert. Es stellte sich heraus, daß neue Berechnung dieser Einstellungen in der Generierungsprozedur für die k-d-Bäume deutlich bessere Ergebnisse als das bisher bestehende System liefert. Diese Vorgehensweise nähert sich dem von *Conceptual Clustering*-Systemen an, der entstehende k-d-Baum repräsentiert schon eine gewisse Klassifizierung der Fälle.

Da die neue Baumgenerierung mit einem höheren Aufwand verbunden ist, wurde ein *inkrementelles Insert* vorgestellt, mit dem ein effizientes Einfügen von neuen Fällen ermöglicht wird, ohne das der k-d-Baum neu generiert werden muß, um seine Optimalität zu garantieren.

Auf der Retrievalseite wurde das neues Konzept der *virtuellen Bounds* entwickelt, die die Suche wesentlich beschleunigen, schließlich wurden zwei Retrievalvarianten für einen häufig benötigten Spezialfall des Retrievals, das *inkrementelle Retrieval*, vorgestellt. Hierbei werden in besonderer Weise Anfragen unterstützt, die nach und nach verfeinert, d. h. genauer spezifiziert werden. So konnte erreicht werden, daß das erworbene Wissen von den vorherigen Retrievalanfragen genutzt werden kann.

Das vorgestellte System kann als Erweiterung des Patdex-Systems ([Wes93]) angesehen werden, einem an der Universität Kaiserslautern entwickelten Case-Based Reasoning System für Diagnose-Anwendungen. Patdex ist Teil der Moltke-Werkbank ([AMWT92]). Eine Einschränkung bei Patdex ist allerdings, daß die dort verwendete Indexierung nur symbolische Attribute verarbeitet werden können und die Verarbeitung großer Fallbasen mit vielen Attributen problematisch ist.

Mit der multidimiensionale Zugriffsstruktur des k-d-Baumes konnten diese Probleme bewältigt werden.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in fünf Kapitel. In diesem Kapitel wird eine kurze Einordnung der Diplomarbeit vorgenommen, im nachfolgenden Kapitel wird eine Einführung in das Case-Based-Reasoning gegeben, dann wird aufgezeigt, wie das System prinzipiell aufgebaut ist und man diskutiert einige Möglichkeiten zur Verwaltung der Fälle. Die nachfolgend verwendete Datenstruktur des k-d-Baums wird mit anderen Verwaltungsstrukturen verglichen.

Das dritte Kapitel beschäftigt sich mit der Generierung von k-d-Bäumen, hierbei wird zunächst das bestehende System vorgestellt, im Anschluß sind die Optimierungsmöglichkeiten aufgezeigt. Außerdem wird hier ein Algorithmus zum inkrementellen Aufbau der k-d-Bäume entwickelt.

Im vierten Kapitel wird das Retrieval in k-d-Bäumen erläuert, dann soll dargestellt werden, wie mit Veränderungen des Retrievals eine Leistungssteigerung erreicht werden kann, des weiteren werden Möglichkeiten vorgestellt, wie eine oft benötigte spezielle Suchaufgabe, das inkrementelle Retrieval, unterstützt werden kann.

Im abschließenden Kapitel wird eine Bewertung der vorgestellten neuen Algorithmen und Datenstrukturen vorgenommen, das Kapitel schließt mit einem kurzen Ausblick ab.

Kapitel 2

Grundlagen

2.1 Fallbasiertes Schließen

Regeln, Klauseln und andere Konstrukte bilden die Basis vieler Formalismen zur Wissensrepräsentation in Expertensystemen. Die Verwendung dieser Formalismen zur Wissensrepräsentation liegt nahe, da Expertenwissen oft in dieser Form vorzuliegen scheint. Aber auch die anhand bereits bearbeiteter und gelöster Probleme gewonnene Erfahrung ist ein wichtiger Bestandteil des Expertenwissens. Durch die Erinnerung an einen bereits erlebten Fall kann man Lösungen für neu auftretende Probleme finden oder den Lösungsprozeß beschleunigen.

Dieses sogenannte episodische Wissen kann allerdings nicht ohne Verlust in Regeln oder anderen Formalismen repräsentiert werden. Mit dem fallbasierten Schließen (*Case-Based Reasoning,* CBR) soll es Expertensystemen ermöglicht werden, frühere Fälle zur Lösung neuer Aufgaben zu nutzen und aus den gemachten Erfahrungen für zukünftige Situationen zu lernen.

Vereinfachend kann unter fallbasiertem Schießen das Lösen von Problemen anhand von bereits bekannten Fällen verstanden werden. Dazu werden Erfahrungen in Form von Falbeispielen gesammelt und in das bereits vorhandene Erfahrungswissen eingeordnet. Ein neues Problem wird dann gelöst, indem die Lösung eines ähnlichen, bereits bekannten Problems komlett bzw. teilweise auf die neue Situation übertragen und entsprechend den aktuellen Anforderungen modifiziert wird. (aus [AWBJMV92]).

Prozeßmodell

Für das Fallbasierte Schließen wurden bereits einige Prozeßmodelle vorgeschlagen, hier jedoch das Modell von Rissland et al. [RKW89] vorgestellt werden, welches sich durch eine genaue Spezifikation der einzelnen Phasen auszeichnet (aus [AWBJMV92]).

Eingabe: Aktuelle Problembeschreibung P, Fallbasis FB mit den Fällen F_i.
Ausgabe: Lösung S für das aktuelle Problem.

1. **Bereitstellung:** Ziel des ersten Schrittes ist es, geeignete Fälle aus der Fallbasis zu wählen. Die Auswahl erfolgt meistens über Eigenschaften der aktuellen Situation, welche sich in der Vergangenheit für die Lösung des Problems als relevant erwiesen haben. Das Retrieval soll zunächst die Anzahl der in den folgenden Schritten zu betrachtenden Fälle einschränken.

2. **Auswahl:** Im zweiten Schritt muß aus der unter Umständen großen Anzahl von gefundenen Fällen der für die aktuelle Problemsituation am besten geeignete Fall ausgewählt werden. Eine gute Auswahl erleichtert die nachfolgenden Adaptionsschritte.

3. **Anpassung und Interpretation:** In diesem Schritt erfolgt, falls nötig, die Anpassung des gefundenen Falles an die aktuelle Situation. Hier gibt es jedoch keine allgemeingültige Vorgehensweise, dieser Schritt erfolgs meist prolemspezifisch.

4. **Test und Kritik:** Hier wird die Ausgabe des letzten Schrittes vom System selbst nochmals überprüft.

5. **Überprüfung der Ergebnisse:** In diesem Schritt wir das Ergebnis der Problemlösung bzw. die Entscheidung des Systems in der realen Welt überprüft. Das System erhält dann ein positives oder negatives Feedback vom Benutzer.

6. **Lernphase:** In der Lernphase wird das im vorhergehenden Schritt erhaltene Feedback analysiert. Das Ergebnis der Analyse ist eine Änderung im System. Hier kann beispielsweise der durch die Problemlösung neu erzeugte Fall in die Fallbais aufgenommen werden oder eine Adaption der verwendeten Ähnlichkeitsmaße vorgenommen werden (z. B. beim System PATDEX/2, [Wes90]).

Ein einfacher Ansatz zur Bereitstellung der Fälle wäre es, einfach die Ähnlichkeit aller Fälle der Falbasis mit dem aktuellen Problem zu berechnen (siehe Abschnitt 2.2). Viele CBR-Systeme arbeiten mit diesem Ansatz, allerdings ist die Komplexität hier $O(n)$ (Es sei n die Anzahl der Fälle in der Fallbasis).

Die Steigerung der Effizienz des Fall-Retrievals ist das Ziel von verschiedenen Forschungsprojekten. Man kann diese Projekte in zwei Ansätze unterteilen:

1. „Brute-force"-Methoden, die sich paralleler Architekturen bedienen, um somit das einfache lineare Durchsuchen der Fallbasis zu beschleunigen ([Kol88], [SW86]).

2. Methoden, die Indexstrukturen berechnen, mit denen auf die Fälle schnell zugegriffen werden kann ([AX91], [BM88], [SHK89]).

Der erste Ansatz benötigt eine aufwendige Hardware-Unterstützung um den Retrievalprozeß zu beschleunigen, beim zweiten Ansatz ist es schwierig, die Vollständigkeit des Retrievals bezüglich der benutzten Ähnlichkeitsfunktion sicherzustellen.

Das in dieser Diplomarbeit entwickelte System zum ähnlichkeitsbasierten Retrieval von Fällen kann dem zweiten Ansatz zugeordnet werden. Es arbeitet in den Phasen „Bereitstellung", „Auswahl" sowie „Lernphase". Unter der Bereitstellung versteht man hier die Auswahl der zu einem Anfragefall ähnlichsten Fälle der Fallbasis. Der Anfragefall definiert hier die Problemstellung.

Da es das Ähnlichkeitsmaß bereits ermöglicht, poetentiell brauchbare Fälle zu erkennen, indem wichtige Faktoren des Falles im besonderen Maße zu der Ähnlichkeitsberechnung beitragen, kann diese ähnlichkeitsbasierte Suche in der Fallbasis auch bereits Aufgaben der „Auswahl-Phase" übernehmen.

Schließlich ist das Einfügen neuer oder modifizierter Fälle in die Fallbasis der Lernphase zuzuordnen.

2.2 Einfacher Ansatz: Die Lineare Suche

Die einfachste Lösung für unser Problem wäre natürlich, alle Fälle der Fallbasis mit dem Anfragefall bezüglich des Ähnlichkeitsmaßes zu vergleichen und anschließend die angeforderten n ähnlichsten Fälle auszugeben.

```
PROCEDURE SearchLin(CaseBase,A,n)
VAR i, Queue[1..n]
BEGIN
  FOR i:=1 TO SIZE(CaseBase) DO
    IF (Sim(A,CaseBase[i]) > Queue[n])
       THEN [Aktualisiere Queue mit CaseBase[i] ].
  RETURN(Queue).
  END. (* FOR *)
END. (* SearchLin *)
```

Hierbei bezeiche `CaseBase` die vorliegende Fallbasis, `A` den Anfragefall, `n` die Anzahl der zu suchenden zum Anfragefall ähnlichsten Fälle und `Queue` ein Array der Größe `n`, in dem die gefundenen Fälle gespeichert werden.

Bei diesem linearen Algorithmus ist es allerdings immer notwendig, **alle** Fälle der Fallbasis zu durchsuchen und jeweils die Ähnlichkeit zum Anfragefall zu berechnen, ein Vorteil ist es, daß quasi kein Overhead zur Verwaltung der Fälle in Kauf genommen werden muß.

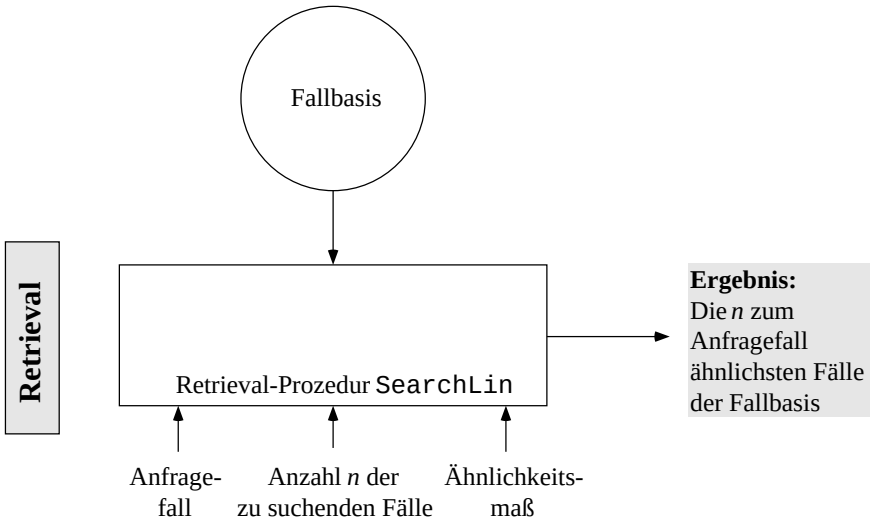


Abbildung 2.1: Einfacher Ansatz

In dieser Arbeit wird ein System vorgestellt, welches durch geschickte Verwaltung der Fälle die Suche in der Fallbasis unterstützt, so daß eine komplette Durchsuchung der Fallbasis nicht mehr notwendig ist. Zu diesem Zweck wir die Bestimmung der ähnlichsten Fälle in zwei Schritte zerlegt:

1. Eine Generierungsphase, in man die Fallbasis so organisiert, daß die folgende Retrieval-Phase nicht mehr alle Fälle durchsuchen muß und somit eine schnelle Suche ermöglicht wird.

2. Eine Retrieval-Phase, in der auf der strukturierten Fallbasis die n zu einem jetzt definierten Anfragefall ähnlichsten Fälle gesucht werden.

Der oben vorgestellte lineare Algorithmus verzichtet ganz auf die Generierungsphase und führt das Retrieval auf der unbearbeiteten Fallbasis durch. Im folgenden werden nun einige Verwaltungskonzepte vorgestellt, mit denen die Fallbasis strukturiert werden soll.

2.3 Verwaltung der Fälle

Das in dieser Diplomarbeit betrachtete Retrieval von Fällen, die über Paare von Attributen und Werten beschrieben werden, ist keine einfache Suche über nur einen einzigen Suchschlüssel (Primärschlüssel). Hier sind bei der Suche mehrere Schlüssel relevant, Anfragen an das System werden durch Bedingungen für die zu suchenden Daten über Attributwerte spezifiziert. Diese Anforderung muß natürlich in der Verwaltung der Fälle berücksichtigt werden um sicherzustellen, daß nicht bei jeder Anfrage der gesamte Datenbestand zu durchsuchen ist.

Die oben angesprochene Problematik wird in einem Teilbebiet der Datenbanksysteme, den *mehrdimensionalen Zugriffspfaden*, behandelt.

In unserem Fall benötigen wir die Unterstützung für einen bestimmten Anfragetyp, den *Best-Match-Queries*. Bei diesen Anfragen muß der in der Anfrage beschriebene Datensatz nicht in der Datenbank enthalten sein, es sollen jedoch möglichst ähnliche Datensätze gefunden werden.

Im folgenden werden drei mehrdimensionale Zugriffspfadstrukturen vorgestellt, die im Rahmen dieser Diplomarbeit auf die Tauglichkeit für unseren Einsatzzweck untersucht wurden:

- das Grid-File Konzept
- Voronoi-Diagramme
- der k-d Baum

2.3.1 Das Grid-File

Die *Grid-Files*, die von J. Nievergelt, H. Hinterberger und H. J. Sevcik [NHS84] vorgestellt wurden, versuchen mehrere gleichwertige Zugriffs-Attribute in die Primärorganisation einzubeziehen. Gleichzeitig werden dabei Bereichsabfragen für mehr als ein Attribut unterstützt.

In diesem Konzept wird der k-dimensionale Datenraum durch ein orthogonales Raster partitioniert, wobei immer nur in einer Dimension gesplittet werden kann. Die Verwaltung dieses Rasters wird über sogenannte *Skalierungsvektoren* $S_i, i \in (1, \ldots, k)$ vorgenommen, die angeben, an welcher Stelle der Datenraum aufgeteilt wurde (siehe Abbildung 2.2). Die entstehenden Teilräume werden als *Gridblocks* bezeichnet.
Die Speicherung der Daten erfolgt in externen Buckets, die Verbindung zwischen den Skalierungsverktoren und den Daten erfolgt durch das *Grid-Directory*. Das Grid-Directory ist ein k-dimensionales Array mit Vektoren, wobei jedem Gridblock eine Komponente (und somit ein Vektor) des Arrays zugeordnet wird. Die Zeiger verweisen nun auf den entsprechenden Bucket, in dem die Daten des entspechenden Gridblocks zu finden sind.

Da eine weitere Partitionierung des Datenraums oft auch solche Gridblocks weiter zerteilt, die eigentlich nicht weiter partitioniert werden müßten, besteht die Möglichkeit, sogenannte *regions* zu bilden. In einer *region* sind mehrere Gridblocks zusammengefaßt, deren Daten sich im selben Bucket befinden. Dies kann einfach implementiert werden, da hier nur mehrere Zeiger des Grid-Directories auf den selbem Bucket verweisen müssen. Zur Verdeutlichung ist ein partitionierter Datenraum und die entstehenden Datenstrukturen des Grid-File Konzepts in Abbildung 2.3 dargestellt.

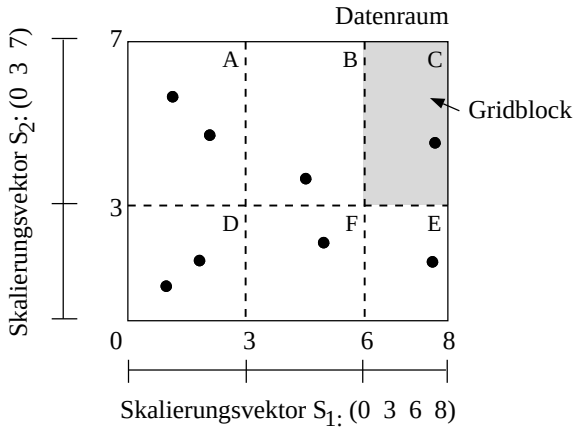


Abbildung 2.2: Grid-File Konzept

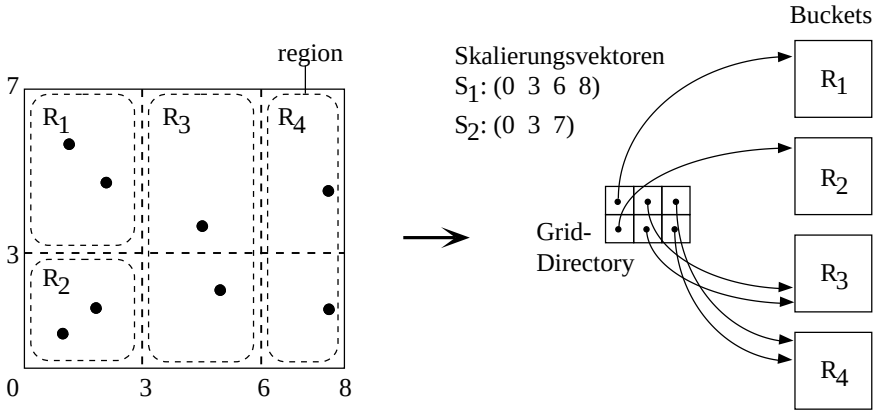


Abbildung 2.3: Beispiel für das Grid-File Konzept

2.3.2 Voronoi-Diagramme

Ein geometrisches Modell, das eine besonders elegante Lösung des „nearest-neighbour“-Problems erlaubt, ist die Konstruktion des sogenannten *Voronoi-Diagramms*. In diesen Diagrammen wird jedem Element aus einer gegebenen Menge von Punkten in einem n-dimensionalen Raum eine Region zugeordnet, in der genau alle Punte liegen, für die das jeweilige Element der nächste Nachbarpunkt ist.

Hierbei werden die Fälle als Punkte im n-dimensionalen Raum dargestellt, die zu jedem Fall berechnete Region definiert dann den Teil des Datenraumes, in dem der zu der Region gehörende Fall der Ähnlichste ist.

So ist die Nearest-Neighbour Suche leicht lösbar, man muß nur die Region bestimmen, in welcher der Anfragefall liegt; der zum Anfragefall nächste Fall ist durch die Region definiert. Ein Voronoi-Diagramm für einen zweidimensionalen Datenraum ist in Abbildung 2.4 dargestellt.

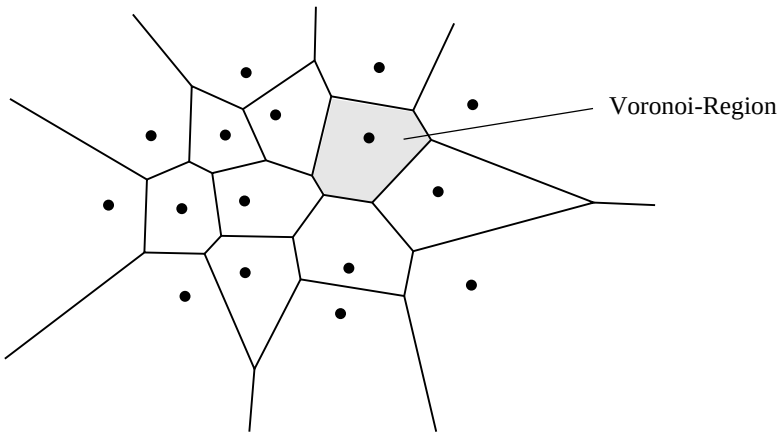


Abbildung 2.4: Voronoi-Diagramm

Definition

Betrachtet man eine Menge S von Punkten in der Ebene E, so kann man für jeden Punkt p aus S die Menge $R(p)$ derjenigen Punte aus E bestimmen, die *näher* zu p liegen als zu jedem anderen Punkt aus S. So müssen alle Punkte bestimmt werden, deren Ähnlichkeit zum keinem Punkt p' aus S größer ist als ihre Ähnlichkeit zu p:

$$R(p) = \{x \in E | sim(x,p) \geq sim(x,p'), p' \in S\}$$

Diese Menge $R(p)$ wird als *Voronoi-Region* bezeichnet, die Menge aller von S erzeugten Voronoi-Regionen heißt *Voronoi-Diagramm*. Die Verwaltung der Voronoi-Diagramme erfolgt über die Beschreibung der Voronoi-Regionen mit n-dimensionalen Polyedern.

Eine ausführliche Beschreibung der Voronoi-Diagramme ist in [Aur84] und [Aur88] zu finden.

2.3.3 Der k-d Baum

Der *k-d-Baum* ist eine von J. Bentley entwickelte Zugriffsstruktur, die mehrdimensionalen Zugriff unterstützt. Der k-d-Baum ist eine Erweiterung des Binärbaums, im k-d-Baum werden k Schlüssel berücksichtigt. Beim Binärbaum werden die Daten durch einen Schlüsselwert, den *Primärschlüssel* eindeutig identifiziert, hier wird während des Retrievals nur der Primärschlüssel überprüft. Bei der Suche im k-d-Baum kann eine Suchanfrage über mehrere Attribute spezifiziert werden.

Definition

Gegeben seien k Wertebereiche der Attribute A_j als geordnete Mengen $\{W_j\}_{j=1}^{k}$ und ein entsprechender Datenraum $W = W_1 \times \ldots \times W_k$.

Definition (k-d-Baum). Es sei $B \subseteq W$ eine Menge von n Fällen mit $B = \{X | X = (x_1, x_2, \ldots, x_k)\}$, $|B| = n$. Weiter sei $b \in N^+$. Der Wert b legt fest, wieviele Fälle maximal in einem Blattknoten gespeichert werden und wird als *Bucketgröße* bezeichnet. Ein k-d-Baum $T^{i_1}(B)$ für die Menge B ist definiert durch:

1. Ist $n \leq b$, so ist $T^{i_1}(B)$ ein Blattknoten, der alle $X \in B$ enthält ($i_1 := 0$).

2. Ist $n > b$, dann bezeichnet $i_1 \in \{1, \ldots, k\}$ die Dimension des Datenraums W, in der die Menge B am Wurzelknoten des k-d-Baumes partitioniert wird. i_1 bezeichnet man als das

Diskriminatorattribut A_{i_1}. Die Wurzel von $T^{i_1}(B)$ enthält einen Wert $p_{i_1} \in W_{i_1}$, den man als *Partitionswert* bezeichnet und je einen Zeiger auf weitere k-d-Bäume $T^{i_2}_{\leq}(B_{\leq})$ und $T^{i_2}_{>}(B_{>})$, wobei $B_{\leq} := \{X \in B | x_{i_1} \leq p_{i_1}\}$ und $B_{>} := \{X \in B | x_{i_1} > p_{i_1}\}$.

Jeder Baumknoten repräsentiert eine Teilmenge der im k-d-Baum gespeicherten Fälle.

- Jeder Blattknoten enthält alle in ihm gespeicherten Fälle. Ein Blattknoten wird auch als *Bucket* bezeichnet.
- Jeder innere Knoten repräsentiert alle Fälle, die in den Blattknoten der durch diesen inneren Knoten aufgespannten Teilbaum enthalten sind. Hierbei teilt ein innerer Knoten die unter ihm gespeicherten Fälle in zwei Teilmengen auf, wobei ein Sohn alle Fälle, deren Attributsausprägungen im Diskriminatorattribut kleiner oder gleich dem Partitionswert sind und der andere Sohn alle übrigen Fälle enthält.

Beispiel

Die Abbildung 2.5 zeigt einen 2-dimensionalen Datenraum, der neun Fälle enthält. Hierzu soll ein 2-d-Baum mit einer Bucketgröße von 2 aufgebaut werden. Zunächst wird der Datenraum in der

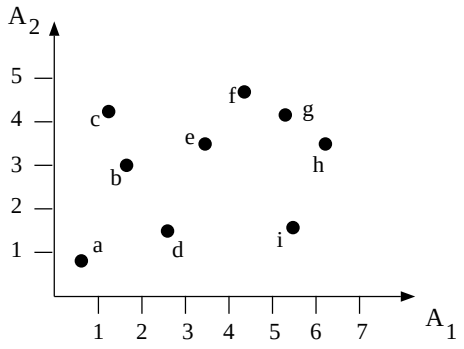


Abbildung 2.5: 2-dimensionaler Datenraum

Dimension 1 aufgeteilt, also ist in der Wurzel des k-d-Baums A_1 das Diskriminatorattribut. Beim weiteren Aufbau wird der Datenraum in immer feinere Partitionen zerteilt. Den so entstandenen k-d-Baum und den jetzt aufgeteilten Datenraum zeigt Abbildung 2.6.

2.3.4 Bewertung der Verwaltungskonzepte

Auf den ersten Blick scheint das Grid-File-Konzept für unseren Einsatzzweck durchaus geeignet zu sein. Beim Grid-File wird die topologische Struktur des Datenraumes erhalten, dies vereinfacht es, *benachbarte* und somit ähnliche Datensätze zu einem bestimmten Punkt im Datenraum aufzufinden.

Allerdings tritt beim Grid-File ein großes Problem auf: Es ist zwar möglich, einen Bucket isoliert weiter zu zerteilen, dies hat jedoch meist eine starke Vergrößerung des Grid-Directories zur Folge (siehe Abbildung 2.7). Da die Anzahl der *regions* und Buckets jedoch nicht so stark anwachsen, ist dieser Nachteil für die üblichen Datenbank-Aufgaben nicht derart relevant. In unserer Anwendung treten jedoch Fälle auf, die fünfzig oder mehr Attribute spezifiziert sind, dies hätte zur Folge, daß das Grid-Directory enorm anwächst.

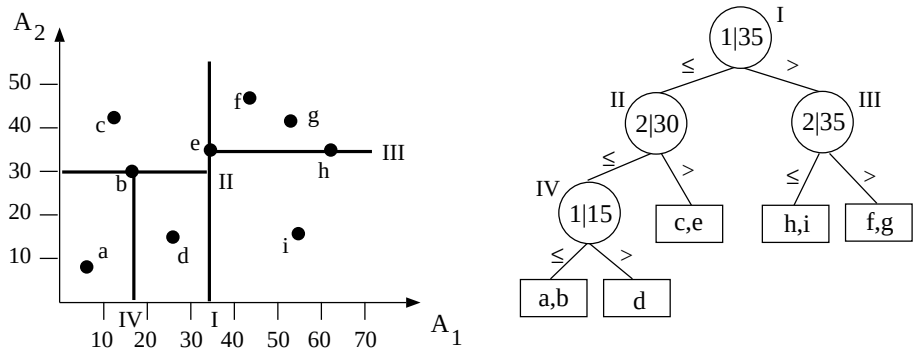


Abbildung 2.6: Beispiel für einen k-d-Baum

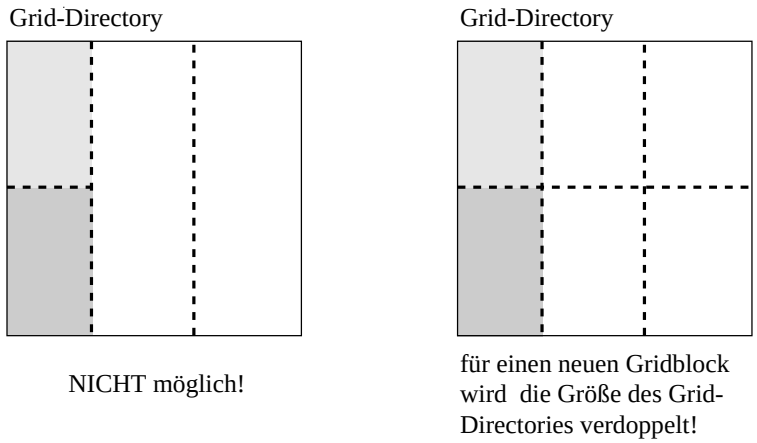


Abbildung 2.7: Aufteilung des Datenraums im Grid-Directory

Zahlenbeispiel

Müßte man eine mehrdimensionale Zugriffsstruktur für Fälle mit fünfzig Attributen aufbauen und würde man in jeder der so entstehenden fünfzig Dimensionen nur ein mal splitten, so würde dies bereits ein Grid-Directory der Größe $2^{50} = 1,123 * 10^{15}$ erzwingen. So wäre allein für die Zeiger auf die Buckets ein Speicherplatz von über 7 Gigabyte (bei einem 50 Bit-Pointer, mit dem man gerade 2^{50} Gridblocks adressieren könnte) bereitzustellen. Das Beispiel zeigt, daß das Grid-File nur für Probleme von geringeren Dimensionen geeignet ist.

In einer Projektarbeit an der Universität Kaiserslautern ([Sch93]) wurden die Voronoi-Diagramme auf Ihre Einsatzmöglichkeiten in fallbasierten Systemen untersucht.

Die Voronoi-Diagramme unterstützen zwar in idealer Weise das ähnlichkeitsbasierte Retrieval, jedoch zeigte sich, daß höherdimensionale Diagramme (bereits mehr als 3 Dimensionen) der Rechen- und Verwaltungsaufwand stark anstieg. Außerdem wurde deutlich, daß die geometrische Repräsentation von Fallbeispielen sehr strenge Anforderungen an die verwendeten Ähnlichkeitsfunktionen stellt. Auch die Repräsentation von symbolischen Attributen erwies sich als kritisch.

Aus diesem Grund wurde der k-d-Baum als Zugriffspfadstruktur ausgewählt, hier ist die Topologieerhaltung zwar nicht so stark ausgeprägt, benachbarte Datenräume sind etwas schwieriger zu erkennen. Die entsprechenden Verfahren werden in den folgenden Kapiteln vorgestellt.

2.4 Struktur des Systems

Das System ist so aufgebaut, daß Generierungs- und Retrieval-Prozeduren unabhängig voneinander arbeiten. So ist es möglich, bei der Baumgenierung verschiedene Verfahren anzuwenden, ohne das hiervon die Retrieval-Prozeduren betroffen sind; Änderungen auf Retrieval-Seite sind kompatibel mit k-d-Bäumen aller Generierungsverfahren. Der k-d-Baum ist die einzige Schnittstelle zwischen Generierung und Retrieval.Die Prozeduren selbst greifen wieder auf Module zu, die auch austauschbar sind (Abbildung 2.8).

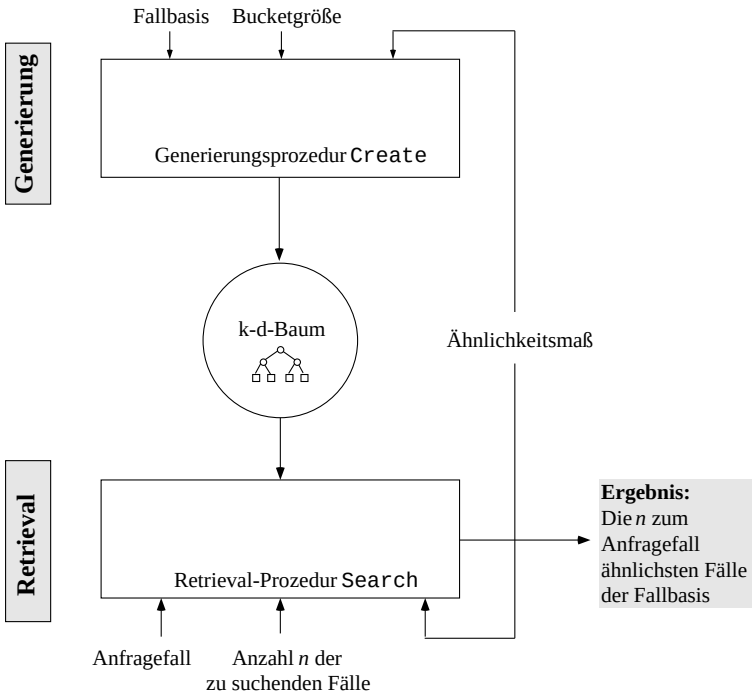


Abbildung 2.8: Systemstruktur

Die Generierungsprozedur `Create` erhält als Eingabe die Fallbasis $\mathcal{S}$ und erzeugt hiermit einen k-d-Baum, der von den Retrievalprozeduren genutzt wird.

```
PROCEDURE Create(S)
BEGIN
...
RETURN (kdTree).
END.
```

Die Retrievalprozedur `Search` wird mit dem generierten k-d-Baum, der Anzahl n der zu suchenden ähnlichsten Fällen und dem Anfragefall A aufgerufen und ermittelt dann die n ähnlichsten Fälle zum Anfragefall.

```
PROCEDURE Search(kdTree,n,A)
```

```
BEGIN
...
RETURN (Cases[n]).
END.
```

Kapitel 3

Generierung von k-d-Bäumen

In diesem Kapitel wird zunächst die Generierung von k-d-Bäumen beschrieben, wie es bei [Öch92] entwickelt und implementiert wurde. Es soll als Grundlage zur Beschreibung der im Rahmen dieser Diplomarbeit erarbeiteten Verbesserungen dienen, die im Anschluß vorgestellt und bewertet werden.

In der Systemübersicht (Abbildung 3.1) sind die für die Generierung relevanten Teile des Systems markiert.

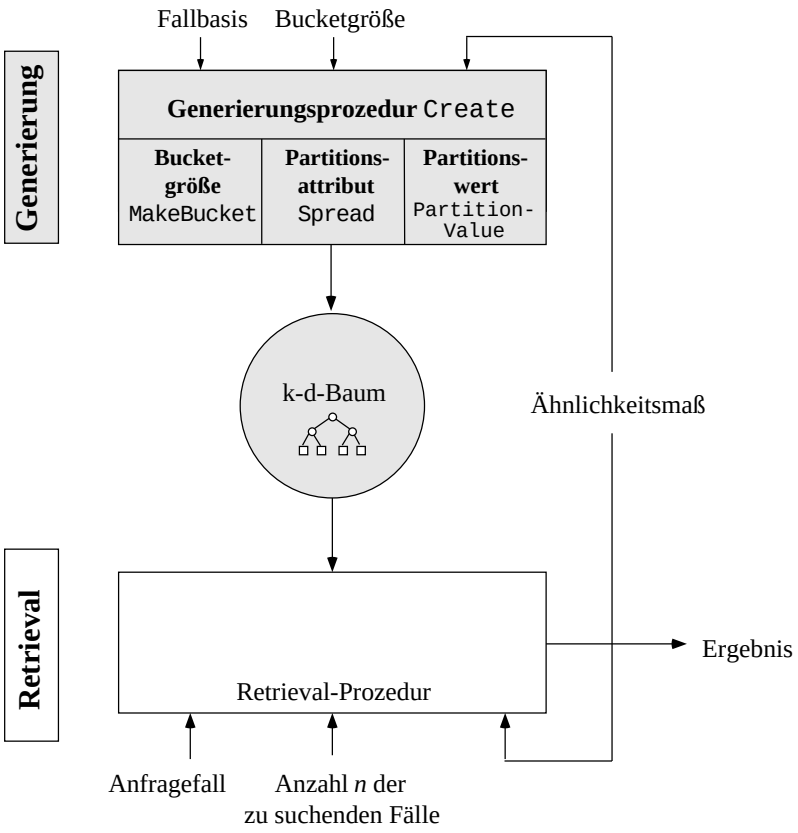


Abbildung 3.1: System-Übersicht

Ziel ist es, einen k-d-Baum als mehrdimensionale Zugriffsstruktur auf eine Fallbasis aufzubauen, der eine *nearest-neighbour*-Suche erlaubt, es soll also möglich sein, bei einer Anfrage die bezüglich

des vorgegebenen Ähnlichkeitsmaßes am dichtesten zum Anfragefall liegenden n Fälle der Fallbasis zu finden.

3.1 Aufbau des k-d-Baumes

Aufgabe der Methoden, die den k-d-Baum aufbauen, ist es, die gegebene Menge von Fällen dermaßen aufzuteilen und eine Baumstruktur darauf zu erstellen, so daß im späteren Retrieval die relevanten Fälle schnell und effizient gefunden werden können. Hierbei werden die Fälle immer wieder in zwei disjunkte Teilmengen partitioniert, jeder Partitionierungsvorgang wird durch einen inneren Baumknoten repräsentiert. In jedem inneren Knoten wird das Diskriminatorattribut und der Partitionswert abgespeichert.

Die Attributsausprägungen aller Fälle der einen Teilmenge sind im Diskriminatorattribut kleiner als der Partitionswert, die Attributsausprägungen der Fälle in der anderen Teilmenge sind im Diskriminatorattribut größer als der Partitionswert. Mit diesen Teilmengen werden weitere Unterbäume gebildet.

So hat man bei der Partitionierung der entspechenden Menge von Fällen zwei Freiheitsgrade:

- Wahl des Partitionswertes
- Wahl des Diskriminator-Attributes

3.1.1 Die Wahl des Partitionswertes

Der Partitionswert ist die Attributsausprägung des Diskriminator-Attributs, bei der die Fälle aufgeteilt werden. Um einen balancierten k-d-Baum zu erhalten, sollten in etwa gleich große Partitionen entstehen.

Nun tritt natürlich die Frage auf, womit man die balancierten Bäume „bezahlen“ muß. Die Balancierung garantiert möglichst kleine Bäume und ermöglicht somit ein schnelles Aufsuchen der Buckets aufgrund der geringen Baumtiefe. Allerdings hat eine strenge Balancierung den Nachteil, daß nicht optimal auf die Struktur in den Daten eingegangen werden kann, da die zu partitionierende Fallmenge immer in zwei gleiche Teile aufgesplittet wird. In der Implementierung von [Öch92] wurde jedoch eine Balancierung der Bäume als vorrangig erachtet. Um dies zu gewährleisten, verwendet man als Partitionswert den Median der Werte, die im Diskriminatorattribut der zu zerteilenden Fälle auftreten.

Die Berechnung des Medians übernimmt die Prozedur `GetPartitionValue(A[i],S)`, sie gibt den Medianwert der Attributsausprägungen des Attributes A_i in der Menge $\mathcal{S}$ zurück und wird von der Generierungsprozedur `Create` bei Bedarf aufgerufen.

3.1.2 Die Wahl des Diskriminator-Attributes

Bei der Wahl des Diskriminator-Attributes verfolgt man das Ziel, ein Attribut auszuwählen, welches die Fälle möglichst gut voneinander trennt, so daß beim späteren Retrieval die Wahrscheinlichkeit groß ist, direkt in den richtigen Teilbaum zu laufen.

Hier benötigt man ein Maß, welches die Streuung der Attributsausprägungen erfassen kann. Da sowohl qualitative als auch quantitative Merkmale auftreten, kann hier beispielsweise nicht die Varianz (oder ähnliche statistische Maße) angewendet werden. Deshalb wird hier zur Abschätzung der Streuung das statistische Maß des *Interquartilsabstandes* [Koo87] verwendet, welches sowohl die Streuung qualitativer als auch quantitativer Mermale beschreiben kann. So wird für jedes Attribut der Interquartilsabstand errechnet, das Attribut mit der größten Streuung wird zum Diskriminatorattribut.

Der Interquartilsabstand

Zur Berechnung dieses Maßes werden die zu untersuchenden Werte mit Hilfe des Medians erst in zwei Teile zerlegt, anschließend werden diese Hälften nochmals mit dem Medium halbiert. So erhält man vier *Quartile* (siehe Abbildung 3.2). Das erste Quartil q_1 teilt die untere Hälfte der Verteilung in zwei gleich große Bereiche und das dritte Quartil q_3 entsprechend die obere Hälfte. Der Interquartilsabstand *iqr* errechnet sich aus dem Abstand zwischen dem ersten und dem dritten Quartil.

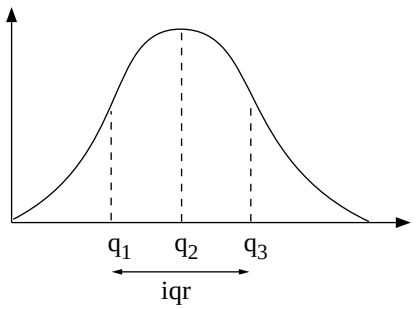


Abbildung 3.2: Interquartilsabstand

Da wir jedoch nicht mit Distanzmaßen, sondern mit einem Ähnlichkeitsmaß arbeiten, wird nicht die Distanz, sondern die Ähnlichkeit zwischen dem ersten und dritten Quartil berechnet. Da wir das Attribut mit der größten Streuung als Diskriminatorattribut wählen wollen, müssen wir das Attribut bestimmen, dessen Quartile zueinander die *geringste* Ähnlichkeit $sim(q_1, q_3)$ haben.

Die Ermittlung des Interquartilsabstandes wird in der Prozedur `Spread(A[i],S)` vorgenommen, die für die Menge $\mathcal{S}$ die Streuung der Werte des Attributs A_i mit Hilfe des Interquartilsabstandes berechnet.

3.1.3 Der Algorithmus zum Aufbau des k-d-Baumes

In diesem Abschnitt wird die Prozedur angegeben, die aus einer Menge $\mathcal{S}$ von Fallbeispielen einen k-d-Baum mit Bucketgröße b aufbaut. Ist $|\mathcal{S}| \leq b$, so wird ein Blattknoten mit allen Fällen in $\mathcal{S}$ erzeugt, andernfalls wird die Fallmenge $\mathcal{S}$ partitioniert und ein innerer Baumknoten generiert.

```
PROCEDURE Create(S)
VAR Discriminator,PartitionValue,minSimilarity,i
BEGIN
   IF MakeBucket(S) THEN RETURN(MakeTerminalNode(S)).
   minSimilarity := infinity.
   FOR ALL Attributes A[i] DO
      IF Spread(A[i],S) < minSimilarity THEN
         minSimilarity := Spread(A[i],S).
         Discriminator := A[i].
      END (* IF *)
   END (* FOR *)

   PartitionValue := GetPartitionValue(Diskriminator,S).
   RETURN(MakeInternalNode(Diskriminator,PartitionValue,
```

```
    Create(LowerPartition(Diskriminator,PartitionValue,S)),
    Create(UpperPartition(Discriminator,PartitionValue,S))
  ).
END (* Create *)
```

Die Prozedur `MakeBucket(S)` liefert `true`, wenn die Anzahl der in $\mathcal{S}$ enthaltenen Werten kleiner als ein vorher definierten Wert b ist, wenn also $|\mathcal{S}| \leq b$ gilt. Schließlich teilen die Prozeduren `LowerPartition(A[i],Value,S)` und `UpperPartition(A[i],Value,S)` die Menge $\mathcal{S}$ in zwei Partitionen auf, wobei `LowerPartition` alle Fälle beinhaltet, deren Attributsausprägung im Attribut A_i kleiner als `Value` ist, `UpperPartition` enthält die jeweils größeren Fälle. `MakeTerminalNode(S)` erzeugt einen Blattknoten mit den Fällen $\mathcal{S}$, `MakeInternalNode(A[i],Value,lowerSon,upperSon)` generiert einen inneren Baumknoten mit dem Diskriminatorattribut A_i, dem Partitionswert `Value` und zwei Pointern auf Blattknoten oder innere Knoten von Teilbäumen `lowerSon` und `upperSon`.

3.1.4 Beispiel

Die Vorgehensweise des Algorithmus' soll nun an einem Beispiel verdeutlicht werden. Es sollen Fälle (in diesem Beispiel repräsentieren sie Personen) verarbeitet werden, die mit jeweils drei Attributen beschrieben werden: Geschlecht, Haarfarbe und Alter.

- *Geschlecht* ist ein symbolisches Attribut mit dem Wertebereich $W_G = \{m, w\}$. Es gelte die Ordnung m < f, das Ähnlichkeitsmaß sei gegeben durch

$$sim_G(x, y) := \begin{cases} 0 & \text{falls } x \neq y \\ 1 & \text{falls } x = y \end{cases}$$

- *Haarfarbe* ist ein symbolisches Attribut mit dem Wertebereich $W_H = \{$braun,schwarz,grau$\}$. Es gelte die Ordnung braun < schwarz < grau, daß Ähnlichkeitsmaß sei gegeben durch

$$sim_H(x, y) := \begin{cases} 0 & \text{falls } x \neq y \\ 1 & \text{falls } x = y \end{cases}$$

- *Alter* ist ein numerisches Attribut mit dem Wertebereich $W_A = \mathcal{N}$, es gelte die übliche Ordnung auf $\mathcal{N}$. Das Ähnlichkeitsmaß sei gegeben durch

$$sim_A(x, y) := \frac{1}{1 + |x - y|}$$

Die Fallbasis $\mathcal{S}$ enthält 8 Fälle:

Person	Geschlecht	Haarfarbe	Alter
A	m	grau	65
B	w	grau	50
C	w	grau	65
D	w	schwarz	15
E	m	schwarz	33
F	m	grau	70
G	w	braun	25
H	m	braun	10

Aufbau des Baumes

Nun soll mit den gegeben Daten ein k-d-Baum mit der Bucketgröße 2 aufgebaut werden.

Zunächst wird die gesamte Fallbasis betrachtet, um den Wurzelknoten des k-d-Baumes zu generieren. Zu diesem Zweck wird für jedes Attribut der Median, q_1, q_2 und der Interquartilsabstand iqr berechnet:

Innererer Knoten I

Attribut	Werte	Median	q_1	q_3	iqr
Geschlecht	m, m, m, m, w, w, w	m	m	w	0 •
Haarfarbe	br, br, sw, sw, gr, gr, gr, gr, gr	sw	br	gr	0
Alter	10, 15, 25, 33, 50, 65, 65, 70	33	15	65	$\frac{1}{51}$

Haben mehrere Attribute den gleichen Interquartilsabstand, so wählt das vorhandene Verfahren das erste Attribut mit minimalem Interquartilsabstand aus. Also wird ein neuer Baumknoten mit

Diskriminator	Partitionswert	$S_{\leq}$	$S_{>}$
Geschlecht	m	$\{A, E, F, H\}$	$\{B, C, D, G\}$

erzeugt.

Da die Teilmengen $S_{\leq}$ und $S_{>}$ noch mehr als zwei Elemente besitzen, müssen sie weiter geteilt werden. So entstehen zwei weitere innere Baumknoten, bevor die Fälle in Blattknoten gespeichert werden:

Innererer Knoten II

Attribut	Werte	Median	q_1	q_3	iqr
Geschlecht	m, m, m, m	m	m	m	1
Haarfarbe	br, sw, gr, gr	sw	br	gr	0 •
Alter	10, 33, 65, 70	33	10	65	$\frac{1}{56}$

Nun wird also im Attribut *Haarfarbe* gesplittet:

Diskriminator	Partitionswert	$S_{\leq}$	$S_{>}$
Haarfarbe	sw	$\{E, H\}$	$\{A, F\}$

Innererer Knoten III

Attribut	Werte	Median	q_1	q_3	iqr
Geschlecht	w, w, w, w	w	w	w	1
Haarfarbe	br, sw, gr, gr	sw	br	gr	0 •
Alter	15, 25, 50, 65	25	15	50	$\frac{1}{36}$

Auch hier hat das Attribut *Haarfarbe* die größte Streuung:

Diskriminator	Partitionswert	$S_{\leq}$	$S_{>}$
Haarfarbe	sw	$\{E, H\}$	$\{A, F\}$

Der so erzeugte Baum ist in Abbildung 3.3 dargestellt.

Bei diesem Beispiel werden bereits Nachteile des bei [Öch92] beschriebenen Aufbauverfahrens deutlich:

- Da beim Aufbau die einzelnen Attribute nur isoliert betrachtet wurden, sind Abhängigkeiten zwischen den Attributen nicht erkannt worden. So ist die Haarfarbe in einen Zusammenhang mit dem Alter zu bringen, die Haarfarbe grau tritt bei älteren Personen natürlich oft auf. So sollten Attribute, von denen wiederum viele andere Attribute abhängen, im Wurzelbereich des Baumes angesiedelt werden. So kann weit oben im Baum bereits eine gute Trenneng der Fälle erreicht werden. Dies leistet das vorhandene Verfahren nicht.

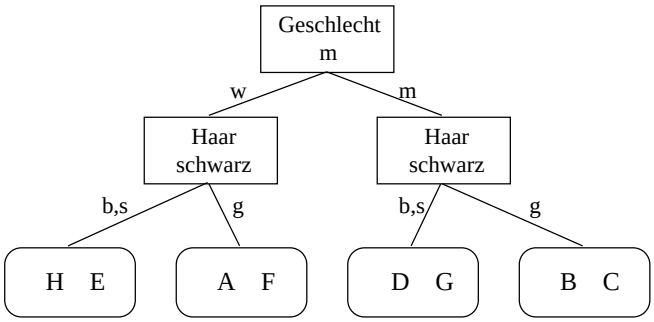


Abbildung 3.3: k-d-Baum

- Beim Aufbau wird nur die Ähnlichkeitsfunktion für die einzelnen Attribute betrachtet. Bei dem Gesamt-Ähnlichkeitsmaß werden jedoch diese einzelnen Werte miteinander verrechnet, dabei können Wichtungsfaktoren verwendet werden. Diese Information über die Relevanz der einzelnen Attribute geht bisher verwendete Prozedur nicht ein.
- Ergibt sich während des Baumaufbaus für meherere Attribute der gleiche Interquartilsabstand, erfolgt die Auswahl eher zufällig. Dies tritt bei symbolischen Attributen relativ häufig auf, da dort meistens die obige *0 oder 1 -Ähnlichkeitsfunktion* verwendet wird. So ergibt sich in vielen Attributen der Interquarilsabstand 0. Eine Verfeinerung dieses Bewertungsmaßes wäre wünschenswert.

3.2 Optimierungsmöglichkeiten bei der k-d-Baum Generierung

Bei der Vorstellung der vorhandenen Generierungs-Algorithmen im vorangegangenen Abschnitt wurde bereits deutlich, daß dort noch divererse Schwächen auftreten. Das System wurde jedoch derart gestaltet, daß einzelne Prozeduren isoliert austauschbar sind).

Beim Aufbau des k-d-Baumes kann man Einfluß auf folgende Parameter nehmen:

- Auswahl des Partitionswertes in den inneren Knoten
- Auswahl des Diskriminatoratributs in den inneren Knoten
- Festlegung der Bucketgröße für die Blattknoten

Mit Veränderung dieser Parameter muß nun versucht werden, einen k-d-Baum aufzubauen, der sich besser an die Struktur der Fallbasis anpaßt. Hierbei ensteht jedoch immer ein korrekter k-d-Baum, auf dem bestehende Retrieval-Prozeduren ohne Modifikation arbeiten können.

3.2.1 Wahl des Partitionswertes

Bisher wurde als Partitionswert immer der Median der Werte ausgewählt, die im Diskriminatoratribut der zu zerteilenden Fälle auftreten. Diese Vorgehensweise hat sicherlich seine Berechtigung bei nominal skalierten Daten. Bei Diskriminatoratributen, die ordinal oder metrisch skaliert sind, ist jedoch eine solche Einteilung nicht sinnvoll (siehe Abbildung 3.4). Die Abbildung verdeutlicht, daß bei einer metrischen Skala zusammengehörige Werte durch das Median-Splitting zerteilt werde, die Struktur in den Daten wurde nicht erkannt. So wäre es wünschenswert auch die Abstände

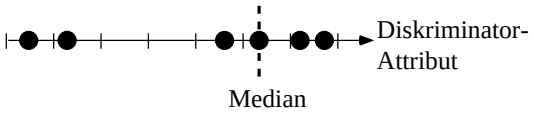


Abbildung 3.4: Median-Splitting

zwischen den Attributsausprägunegen zu berücksichtigen, um so eine besser zu den Daten passende Einteilung zu finden.

So führt hier führt eine neue Vorgehensweise zu besseren Ergebnissen: Man splittet nicht mehr im Median, sondern sucht die *größte Lücke* zwischen den einzelnen Attributsausprägungen. Die Daten werden nach der auf ihnen definierten Ordnung durchsucht, an der Stelle, wo die Distanz zwischen zwei Attributsausprägungen am größten ist (d. h. die Ähnlichkeit am geringsten ist), wird gesplittet (Abbildung 3.5).

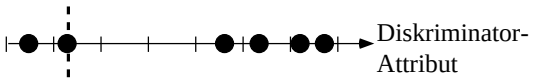


Abbildung 3.5: Abstands-Splitting

Dieses neue Vorgehen hat außerdem den Vorteil, daß hier weiteres Wissen, repräsentiert durch das Ähnlichkeitsmaß, in die Wahl des Partitionswertes mit einfließt. Bei der bisherigen Auswahlvariante war dies nicht der Fall.

Allerdings ist bei der neuen Variante nicht mehr gesichert, einen balancierten k-d-Baum zu erhalten, da die Fälle nicht mehr in zwei gleich große Partitionen aufgeteilt werden. Die Balancierung hat den Vorteil, daß die Wege im Baum kürzer werden, man erreicht die Buckets schneller. Dieser Vorteil hat sich in der Praxis aber nicht als entscheidend erwiesen, es ist viel wichtiger, die Fälle optimal voneinander zu trennen.

3.2.2 Wahl des Diskriminatorattributs

Bei der Vorstellung der bisher verwendeten Generierungsprozedur wurden bereits einige Probleme und Nachteile deutlich: Bei der Auswahl des Diskriminatorattributs werden die Attribute nur isoliert betrachtet, dies hat zur Folge, daß Abhängigkeiten zwischen den einzelnen Attributen nicht erkannt werden. Die Auswahl erfolge bisher nur mit dem statistischen Maß des Interquartilsabstandes. Diese isoliere Betrachtungsweise führte oft zu unbefriedigenden Ergebnissen (siehe Abbildung 3.6).

Wünschenswert wäre aber ein Vorgehen, das alle Dimensionen betrachtet und so eine gute Partitionierung der Fälle erreicht (Abbildung 3.7).

Diese bisherige *a-priori*-Abschätzung mit dem Interquartilsabstand wird nun durch eine *a-posteriori*-Abschätzung ersetzt, bei der nach einem probeweisen Splitting die Qualität der Partitionierung unter Berücksichtigung aller Dimensionen bewertet wird.

So wird für jedes potentielle Splitting-Attribut eine Partitionierung am bereits berechneten Partitionswert vorgenommen, anschließend wird die Güte der Partitionierung mit einem Maß bewertet.

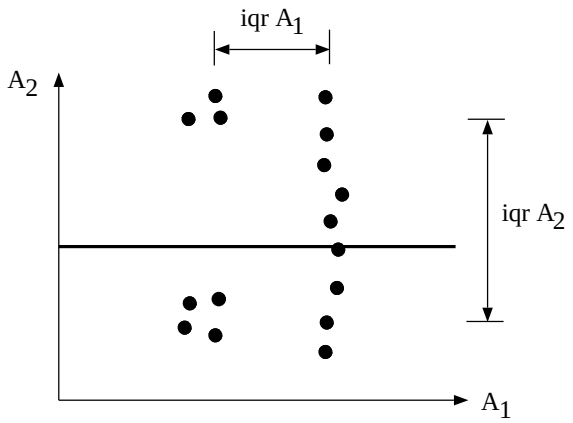


Abbildung 3.6: Schlechte Wahl des Diskriminatoratributes

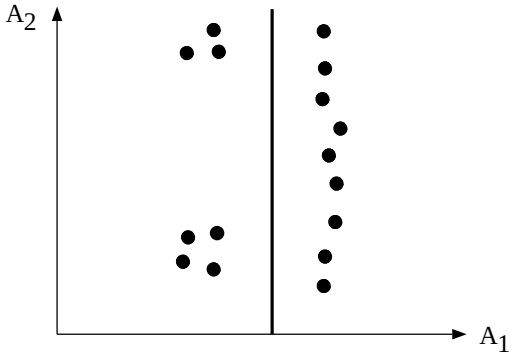


Abbildung 3.7: Gute Wahl des Diskriminatoratributes

Im folgenden wird die Generierungsprozedur vorgestellt, die mit der *a-posteriori*-Abschätzung arbeitet.

```
PROCEDURE CreateNew(S)
VAR Discriminator,PartitionValue,PartitionQuality,
    bestPartitionQuality,i
BEGIN
   IF MakeBucket(S) THEN RETURN(MakeTerminalNode(S)).
   bestPartitionQuality := -infinity.
   FOR ALL Attributes A[i] DO
      PartitionValue := GetPartitionValue(A[i],S).
      PartitionQuality :=
         QualityMeasure( LowerPartition(A[i],PartitionValue,S),
                         UpperPartition(A[i],PartitionValue,S)).
      IF (PartitionQuality > bestPartitionQuality)
      THEN
         bestPartitionQuality := PartitionQuality.
         Discriminator := A[i].
```

```
      END (* IF *)
   END (* FOR *)

   PartitionValue := GetPartitionValue(Diskriminator,S);
   RETURN(MakeInternalNode(Diskriminator,PartitionValue,
     CreateNew(LowerPartition(Diskriminator,PartitionValue,S)),
     CreateNew(UpperPartition(Discriminator,PartitionValue,S))
   ).
END (* CreateNew *)
```

Zunächst wird mit der Prozedur **MakeBucket** überprüft, ob ein neuer Bucket erzeugt werden soll. Bisher geschah dies einfach durch eine Konstante b, die beim Aufruf der Generierungsprozedur festgelegt wurde. Beeinhaltet die Fallmenge $\mathcal{S}$ nicht mehr als b Elemente, ist also $\mid \mathcal{S} \mid \leq b$, wird ein Bucket erzeugt. Eine alternative Bedingung wird in Abschnitt 3.3 vorgestellt.

Anschließend wird in einer Schleife mit der Prozedur **GetPartitionValue** für alle Attribute ein Partitionswert (Median oder größter Abstand, siehe Abschnitt 3.2.1) errechnet.

Für diesen Partitionswert wird nun ein Splitting durchgeführt, die bei dem Splitting entstandenen zwei Teilmengen von $\mathcal{S}$ werden mit der Prozedur **QualityMeasure** bewertet. Diese Prozedur entscheidet somit, welches Attribut zum Diskriminatorattribut wird.

Für die Prozedur **QualityMeasure** sollen jetzt in den folgenden Abschnitten drei alternative Gütemaße untersucht werden:

- Das *Category-Utility* aus dem System COBWEB
- Verwendung eines Entropie-Maßes
- Das Maß der durchschnittlichen Ähnlichkeit

3.2.3 Das Category-Utility

Bei dem von Douglas Fisher 1987 vorgestellten System COBWEB handelt es sich um ein *conceptual clustering*-System ([Fis87.1], [Fis87.2]). Auf die Repräsentation und Organisation in COBWEB soll hier nicht eingegangen werden, jedoch bedient sich das System einer für uns interessanten Bewertungsfunktion, die in COBWEB die Klassifikation und das Lernverhalten kontrolliert.

Diese Bewertungsfunktion arbeitet mit dem *Category-Utility*, einer von Gluck und Corter 1985 vorgestellten Funktion ([GC85]).

Mit dieser Funktion versucht man, die Ähnlichkeit **in** den Klassen (in unserem Fall Partitionen) und die Unterschiede **zwischen** den Klassen zu maximieren. Diese Ähnlichkeit in den Klassen wird als *intra-class-similarity*, die Unterschiede zwischen den Klassen als *inter-class-dissimilarity* bezeichnet.

Intra-class-similarity wird durch die bedingte Wahrscheinlichkeit $P(A_i = V_{ij} \mid C_k)$ ausgedrückt, wobei C_k eine Klasse, A_i ein Attribut und V_{ij} eine Attributsausrägung bezeichnet. $P(A_i = V_{ij} \mid C_k)$ bezeichnet also die Wahrscheinlichkeit, daß wenn man sich in der Klasse C_k befindet, das Attribut A_i den Wert V_{ij} hat. Ist diese Wahrscheinlichkeit groß, so ist die Zahl der Fälle in dieser Klasse groß, die sich in dem Attribut A_i den Wert V_{ij} teilen.

Die *inter-class-dissimilarity* wird durch die bedingte Wahrscheinlichkeit $P(C_k \mid A_i = V_{ij})$ beschrieben. $P(C_k \mid A_i = V_{ij})$ beschreibt die Wahrscheinlichkeit, daß man sich in der Klasse C_k befindet, falls das Attribut A_i die Ausprägung V_{ij} hat. Ein hoher Wert bedeutet also hier, das wenige Fälle in unterschiedlichen Klassen sich den Wert V_{ij} im Attribut A_i teilen.

Nun werden diese Wahrscheinlichkeiten jede Klasse C_k und alle Attribut-Wert Paare $(A_i = V_{ij})$ aufsummiert, mit $P(A_i = V_{ij})$ wird eine Gewichtung der einzelnen Werte vorgenommen,

da Attributsausprägungen, die öfter vorkommen, auch einen größeren Einfluß auf die Endsumme haben müssen.

$$\sum_{k=1}^{n} \sum_{i} \sum_{j} P(A_i = V_{ij}) \quad \underbrace{P(C_k \mid A_i = V_{ij})}_{\text{inter-class dissimilarity}} \quad \underbrace{P(A_i = V_{ij} \mid C_k)}_{\text{intra-class similarity}} \tag{3.1}$$

Um die Berechnung zu vereinfachen, wird nun der Satz von Bayes angewendet, nach dem gilt:

$$P(A_i = V_{ij})P(C_k \mid A_i = V_{ij}) = P(C_k)P(A_i = V_{ij} \mid C_k) \tag{3.2}$$

Also berechnet sich unser Maß nun so:

$$\sum_{k=1}^{n} P(C_k) \sum_{i} \sum_{j} P(A_i = V_{ij} \mid C_k)^2 \tag{3.3}$$

In unserer Formel bezeichnet $\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2$ die erwartete Anzahl von Attributsausprägungen, die für einen beliebigen Fall in der Klasse C_k richtig erraten werden kann.

Schließlich definieren Gluck und Corter das *Category-Utility* als Anstieg der korrekt zu erratenden Attributsausprägungen nach der Partitionierung ($\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2$) gegenüber den zu erratenden Ausprägungen ohne Splitting ($\sum_i \sum_j P(A_i = V_{ij})^2$). Das *Category-Utility* errechnet also den Informationsgewinn, der durch die Partitionierung ereicht wurde. Der Wert n bezeichne die Anzahl der Partitionen.

$$\frac{\sum_{k=1}^{n} P(C_k)(\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2)}{n} \tag{3.4}$$

Bei der Anwendung des *Category-Utilities* in unserem System wird die Fallmenge immer in zwei Partitionen zerteilt, wir arbeiten also nur mit den Klassen C_1 und C_2.

Beispiel

Hier soll an einem Beispiel gezeigt werden, wie mit dem *Category-Utility* ein Diskriminatorattribut ausgewählt wird. Eine Menge mit acht Fällen, im Beispiel beschreiben sie Personen, soll gespittet werden.

Person	Größe	Haarfarbe	Augenfarbe
A	klein	blond	blau
B	groß	rot	blau
C	groß	blond	blau
D	groß	blond	braun
E	klein	dunkel	blau
F	groß	dunkel	blau
G	groß	dunkel	braun
H	klein	blond	braun

Nun soll überprüft werden, ob die Fallmenge nach der Körpergröße, der Haarfarbe oder der Augenfarbe getrennt werden soll. Zunächst wird der Faktor $\sum_i \sum_j P(A_i = V_{ij})^2$ ausgerechnet:

$$\sum_{i} \sum_{j} P(A_i = V_{ij})^2 \quad = \quad \underbrace{((\frac{3}{8})^2 + (\frac{5}{8})^2)}_{\sum_j P(A_{\text{Größe}} = V_{\text{Größe}_j})^2} \quad + \quad \underbrace{((\frac{4}{8})^2 + (\frac{1}{8})^2 + (\frac{3}{8})^2)}_{\sum_j P(A_{\text{Haar}} = V_{\text{Haarfarbe}_j})^2}$$

$$+ \quad \underbrace{((\frac{5}{8})^2 + (\frac{3}{8})^2)}_{\sum_j P(A_{\text{Augen}}=V_{\text{Augenfarbe}_j})^2}$$

$$= \frac{94}{64}$$

Anschließend wird $\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2$ für die einzelnen Partitionierungsvarianten bestimmt:

1. **Partitionierung nach** *Größe* in die Partitionen {klein} und {groß}.

$$\sum_i \sum_j P(A_i = V_{ij} \mid C_{klein})^2 = \underbrace{(1^2 + 0^2)}_{\sum_j P(A_{Größe}=V_{Größe_j} \mid C_{klein})^2}$$
$$+ \quad \underbrace{((\frac{2}{3})^2 + 0^2 + (\frac{1}{3})^2))}_{\sum_j P(A_{\text{Haar}}=V_{Haarfarbe_j} \mid C_{klein})^2}$$
$$+ \quad \underbrace{((\frac{2}{3})^2 + (\frac{1}{3})^2))}_{\sum_j P(A_{Augen}=V_{Augenfarbe_j} \mid C_{klein})^2}$$
$$= \frac{19}{9}$$

und

$$\sum_i \sum_j P(A_i = V_{ij} \mid C_{groß})^2 = \underbrace{(0^2 + 1^2)}_{\sum_j P(A_{\text{Größe}}=V_{\text{Größe}_j} \mid C_{\text{groß}})^2}$$
$$+ \quad \underbrace{((\frac{2}{5})^2 + (\frac{2}{5})^2 + (\frac{1}{5})^2))}_{\sum_j P(A_{\text{Haar}}=V_{\text{Haarfarbe}_j} \mid C_{\text{groß}})^2}$$
$$+ \quad \underbrace{((\frac{2}{5})^2 + (\frac{3}{5})^2))}_{\sum_j P(A_{\text{Augen}}=V_{\text{Augenfarbe}_j} \mid C_{\text{groß}})^2}$$
$$= \frac{47}{25}$$

So errechnet sich der Informationsgewinn durch das Splitting der Menge in die zwei Teilmengen $C_{klein} = \{A, E, H\}$ und $C_{groß} = \{B, C, D, F, G\}$ mit der Formel 3.4 zu

$$\frac{\frac{3}{8}(\frac{19}{9} - \frac{94}{64}) + \frac{5}{8}(\frac{47}{25} - \frac{94}{64})}{2} \approx \mathbf{0,249}$$

2. **Partitionierung nach** *Haarfarbe* in die Partitionen {blond, rot} und {dunkel}.

$$\sum_i \sum_j P(A_i = V_{ij} \mid C_{\text{blond,rot}})^2 = ((\frac{2}{5})^2 + (\frac{3}{5})^2) + (1^2 + 0^2)$$
$$+((\frac{3}{5})^2 + (\frac{2}{5})^2)$$
$$= \frac{51}{25}$$

$$\sum_i \sum_j P(A_i = V_{ij} \mid C_{\text{dunkel}})^2 = ((\frac{1}{3})^2 + (\frac{2}{3})^2) + (0^2 + 1^2)$$

$$+((\frac{3}{3})^2 + (\frac{1}{3})^2)$$
$$= \frac{19}{9}$$

So errechnet sich der Informationsgewinn durch das Splitting der Menge in die zwei Teilmengen $C_{blond,rot} = \{A, B, C, D, H\}$ und $C_{dunkel} = \{E, F, G\}$ mit der Formel 3.4 zu

$$\frac{\frac{5}{8}(\frac{51}{25} - \frac{94}{64}) + \frac{3}{8}(\frac{19}{9} - \frac{94}{64})}{2} \approx \mathbf{0,3}$$

3. **Partitionierung nach** *Augenfarbe* in die Partitionen {blau} und {braun}.

$$\sum_i \sum_j P(A_i = V_{ij} \mid C_{blau})^2 = ((\frac{2}{5})^2 + (\frac{3}{5})^2)) + ((\frac{2}{5})^2 + (\frac{1}{5})^2) + (\frac{2}{5})^2)) + (1^2 + 0^2)$$
$$= \frac{47}{25}$$

$$\sum_i \sum_j P(A_i = V_{ij} \mid C_{braun})^2 = ((\frac{1}{3})^2 + (\frac{2}{3})^2)) + ((\frac{1}{3})^2 + 0^2 + (\frac{2}{3})^2)) + (0^2 + 1^2)$$
$$= \frac{19}{9}$$

So errechnet sich der Informationsgewinn durch das Splitting der Menge in die zwei Teilmengen $C_{blau} = \{A, B, C, E, F\}$ und $C_{braun} = \{D, G, H\}$ mit der Formel 3.4 zu

$$\frac{\frac{5}{8}(\frac{47}{25} - \frac{94}{64}) + \frac{3}{8}(\frac{19}{9} - \frac{94}{64})}{2} \approx \mathbf{0,249}$$

Die errechneten Werte zeigen, daß man die Menge nach der Haarfarbe partitionieren soll. Hier ist der Informationsgewinn durch die Partitionierung mit 0,3 höher als bei den beiden anderen Partitionierungsvarianten, die den Wert 0,249 ergaben.

Anschaulich bedeutet dies, daß mit dem Attribut „Haarfarbe“ die anderen Attribute verknüpft sind, so scheint beispielsweise die Augenfarbe einer Person mit deren Haarfarbe zusammenzuhängen. Gerade dieses Verhalten wollen wir von unserem System, es soll zusammenhängende Fälle erkennen und diese gemeinsam in einen Teilbaum abspeichern. So ereicht man weit oben im Baum bereits eine gute Trennung der Fälle.

Abschließend ist zu bemerken, daß das *Category-Utility* nur für nominal skalierte (symbolische) Daten entwickelt wurde. Um auch ordinal oder metrisch skalierte Daten handhaben zu können, muß man diese Werte in Intervalle aufteilen und arbeitet dann mit diesen Intervallen weiter.

Ein weiter Ansatz für die Behandlung nicht-symbolischer Attribute ergibt sich aus dem Ansatz des Systems CLASSIT, welches 1989 von John H. Gennari, Pat Langley und Doug Fisher vorstellt wurde ([GLF89]). Hier wird das *Category-Utility* für reele Werte modifiziert. In CLASSIT wird der Informationsgewinn folgendermaßen errechnet:

$$\frac{\sum_{k=1}^{n} P(C_k)(\sum_i \frac{1}{\sigma_{ik} - \sum_i \frac{1}{\sigma_i}}}{n} \tag{3.5}$$

Hierbei bezeichnet σ_{ik} die Standardabweichung der Attributsausprägungen des Attributs A_i der Fälle in der Klasse C_k und σ_i die Standardabweichung der Attributsausprägungen des Attributs A_i aller zu partitionierenden Fälle.

Baut man mit dem Category-Utility einen k-d-Baum mit maximaler Bucketgröße 2 für die Fallbasis auf, entsteht nachfolgender Baum (Abbildung 3.8).

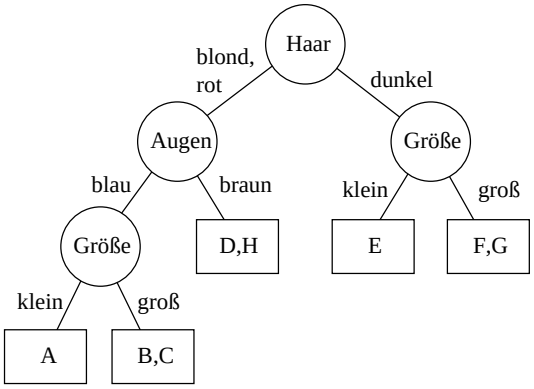


Abbildung 3.8: entstehender k-d-Baum mit Category-Utility

3.2.4 Entropie-Maß

Hier soll nun ein Maß vorgestellt werden, welches aus dem Bereich der Entscheidungsbäume kommt und zu den TDIDT-Algorithmen (*Top-Down Induction of Decision Trees*) gehört. Einen echten Entscheidungsbaum können wir natürlich nicht erzeugen, da unser CBR-System die ähnlichsten Fälle zurückliefern soll und nicht immer direkt eine korrekte Diagnose für den Anfragefall ausgeben kann. Dieser Ansatz ist nur für Fallbasen anwendbar, bei denen bereits eine Diagnose vorliegt. Sind die eingespeicherten Fälle beispielsweise Krankheitsdaten, so würde man als Suchattribute die auftretenden Symptome, als Diagnose die entsprechende Krankheit wählen. Ziel dieses Ansatzes ist es, einen Baum aufzubauen, der in Wurzelnähe diejenigen Attribute enthält, die besonders wichtig für die zu erstellende Diagnose sind. Somit sollen weit oben im Baum nicht relevate Fälle bei der Suche ausgeklammert werden.

Als *bedingte Entropie* bei einem bereits erfolgten Versuch $\mathcal{B}$ für einen weiteren Versuch $\mathcal{A}$ bezeichnet man

$$H(\mathcal{A} \mid B_k) = -\sum_i P(A_i \mid B_k) \log_2(P(A_i \mid B_k)) \tag{3.6}$$

wobei unter B_k ein Ereignis zu verstehen ist, welches beim Versuch $\mathcal{B}$ eingetreten ist. A_i bezeichnet einen Ausgang des Versuches $\mathcal{A}$.

Die *Rückschlußentropie* ist definiert als:

$$H(\mathcal{A} \mid \mathcal{B}) = \sum_k P(B_k) H(P(\mathcal{A} \mid B_k)) \tag{3.7}$$

Diese eher allgemein gehaltenen Formeln sollen nun für unseren Einsatzzweck spezifiziert werden. Die Rückschlußentropie soll uns dazu dienen, den Informationsgewinn einzelner Partitionierungsvarianten zu beurteilen. Ist die Rückschlußentropie klein, so ist der Informationsgewinn, den wir

durch die Partitionierung erreichen, groß. So lautet die bedingte Entropie für unsere untere Partitionsmenge $S_{A_i,lower}$, wobei die gesamte Fallmenge $\mathcal{S}$ nach dem Attribut A_i gesplittet wurde:

$$H(\mathcal{A} \mid S_{A_i,\text{lower}}) = -\sum_{d=1}^{\#\text{ Diagnosen}} \left(\frac{\#\text{ Fälle mit Diagnose}_d \text{in } S_{A_i,\text{lower}}}{\mid S_{A_i,\text{lower}} \mid} \log_2\left(\frac{\#\text{ Fälle mit Diagnose}_d \text{in } S_{A_i,\text{lower}}}{\mid S_{A_i,\text{lower}} \mid}\right)\right) \tag{3.8}$$

sowie für die obere Partitionsmenge $S_{A_i,upper}$:

$$H(\mathcal{A} \mid S_{A_i,\text{upper}}) = -\sum_{d=1}^{\#\text{ Diagnosen}} \left(\frac{\#\text{Fälle mit Diagnose}_d \text{in } S_{A_i,\text{upper}}}{\mid S_{A_i,\text{upper}} \mid} \log_2\left(\frac{\#\text{ Fälle mit Diagnose}_d \text{in } S_{A_i,\text{upper}}}{\mid S_{A_i,\text{upper}} \mid}\right)\right) \tag{3.9}$$

So ergibt sich die Rückschlußentropie zu:

$$H(\mathcal{A} \mid \text{Part. Att}_i) = \frac{\mid S_{A_i,\text{lower}} \mid}{\mid S \mid} \cdot H(\mathcal{A} \mid S_{A_i,\text{lower}}) + \frac{\mid S_{A_i,\text{upper}} \mid}{\mid S \mid} \cdot H(\mathcal{A} \mid S_{A_i,\text{upper}}) \tag{3.10}$$

Beispiel

Nun wird wieder an einem Beispiel verdeutlicht, wie die Wahl des Diskriminatorattributes mit den Entropie-Maß vor sich geht. Im wesentlichen wird das Beispiel aus Abschnitt 3.2.3 verwendet, allerdings wird hier ein weiteres Attribut, welches die Diagnose enthält, hinzugefügt.

Person	Diagnose	Größe	Haarfarbe	Augenfarbe
A	+	klein	blond	blau
B	+	groß	rot	blau
C	+	groß	blond	blau
D	-	groß	blond	braun
E	-	klein	dunkel	blau
F	-	groß	dunkel	blau
G	-	groß	dunkel	braun
H	-	klein	blond	braun

Nun soll die Rückschlußentropie für die einzelnen Partitionierungsvarianten Größe, Haar und Augen berechnet werden:

1. **Partitionierung nach** *Größe* in die Partitionsmengen $S_{A_{\text{Größe}}=\{klein\}} = \{A, E, H\}$ und $S_{A_{\text{Größe}}=\{groß\}=\{B,C,D,F,G\}}$.

$$H(\mathcal{A} \mid S_{A_{\text{Größe}}=\{klein\}}) = -\frac{1}{3} \cdot \log_2 \frac{1}{3} - \frac{2}{3} \cdot \log_2 \frac{2}{3} = 0,971$$

$$H(\mathcal{A} \mid S_{A_{\text{Größe}}=\{groß\}}) = -\frac{2}{5} \cdot \log_2 \frac{2}{5} - \frac{3}{5} \cdot \log_2 \frac{3}{5} = 0,981$$

So ergibt sich eine Rückschlußentropie für die Partitionierung nach Größe:

$$H(\mathcal{A} \mid \text{Part. Att}_{Größe}) = \frac{3}{8} \cdot 0,971 + \frac{5}{8} \cdot 0,981 = \mathbf{0,951}$$

2. **Partitionierung nach** *Haar* in die Partitionsmengen $S_{A_{\text{Haar}}=\{blond,rot\}} = \{A, B, C, D, H\}$ und $S_{A_{\text{Haar}}=\{dunkel\}} = \{E, F, G\}$.

$$H(\mathcal{A} \mid S_{A_{\text{Haar}}=\{blond,rot\}}) = -\frac{3}{5} \cdot \log_2 \frac{3}{5} - \frac{2}{5} \cdot \log_2 \frac{2}{5} = 0,676$$

$H(\mathcal{A} \mid S_{A_{Haar}=\{dunkel\}}) = 0$ (da hier alle Personen mit dunklem Haar die selbe Diagnose haben)

So ergibt sich eine Rückschlußentropie für die Partitionierung nach Haarfarbe:

$$H(\mathcal{A} \mid \text{Part. Att}_{Haar}) = \frac{5}{8} \cdot 0,971 + \frac{3}{8} \cdot 0 = \mathbf{0,486}$$

3. **Partitionierung nach** *Augen* in die Partitionsmengen $S_{A_{\text{Augen}}=\{blau\}} = \{A, B, C, E, F\}$ und $S_{A_{Augen}=\{braun\}} = \{D, G, H\}$.

$$H(\mathcal{A} \mid S_{A_{Augen}=\{blau\}}) = -\frac{3}{5} \cdot \log_2 \frac{3}{5} - \frac{2}{5} \cdot \log_2 \frac{2}{5} = 0,971$$

$H(\mathcal{A} \mid S_{A_{Augen}=\{braun\}}) = 0$ (da hier alle Personen die selbe Diagnose haben)

So ergibt sich eine Rückschlußentropie für die Partitionierung nach Augenfarbe:

$$H(\mathcal{A} \mid \text{Part. Att}_{Augen}) = \frac{5}{8} \cdot 0,971 + \frac{3}{8} \cdot 0 = \mathbf{0,486}$$

Da die Rückschlußentropie der Attribute „Haar“ und „Augen“ am niedrigsten ist, sollte man die Fallmenge nach einem dieser Attribute teilen.

Am Beispiel erkennt man, daß diejenigen Partitionierungsvarianten bevorzugt werden, die auch eine gute Teilung des Diagnoseattributs bewirken. Aus diesem Grund ist es sehr wichtig, ein korrektes Diagnoseattribut zu wählen. Fälle, die aufgrund des Ähnlichkeitsmaßes dicht zusammen liegen, sollten auch die selbe Diagnose haben.

Baut man mit dem Entropiemaß einen k-d-Baum mit maximaler Bucketgröße 2 für die Fallbasis auf, entsteht nachfolgender Baum (Abbildung 3.9).

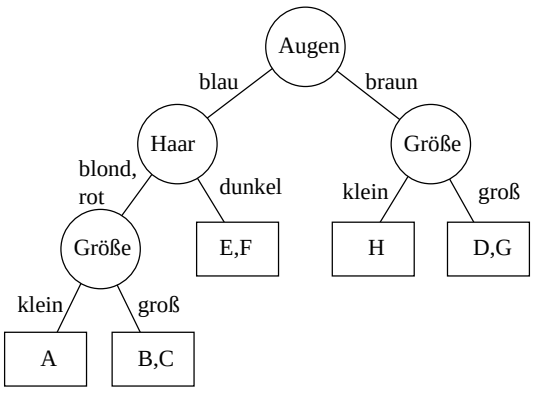


Abbildung 3.9: entehender k-d-Baum mit Entropiemaß

3.2.5 Maß der durchschnittlichen Ähnlichkeit

Als drittes Bewertungsmaß wurde das *Maß der durchschnittlichen Ähnlichkeit* untersucht und implementiert. Dieses Bewertungsmaß wurde aufgrund eines Nachteils der beiden anderen vorgestellten Varianten entwickelt, denn bei diesen Maßen wurde das Ähnlichkeitsmaß bei der Auswahl des Diskriminatorattributes nicht verwendet. Da unserer Ähnlichkeitsmaß aber auch eine wesentliche Komponente des vorhandenen Wissens darstellt, wurde bisher zu wenig Wissen, nämlich nur die Fälle, genutzt.

Hier werden für die vorgenommene zu untersuchende Partitionierung in jeder Partitionsmenge die Ähnlichkeit unter den in einer Partition befindlichen Fällen errechnet, anschließend wird der Mittelwert gebildet:

$$\text{AverageSim}_{\text{Part}} = \frac{\sum_{i=1}^{|\text{Part}|} \sum_{j=i}^{|\text{Part}|} Sim(\text{Case}_i, \text{Case}_j)}{\sum_{k=1}^{|\text{Part}|} k} \tag{3.11}$$

Hierbei sei | Part | die Anzahl der sich in der Partition *Part* befindlichen Fälle.

Die obige Formel gilt nur für symmetrische Ähnlichkeitmaße, ist jedoch $Sim(\text{Case}_A, \text{Case}_B) \neq Sim(\text{Case}_B, \text{Case}_A)$, dann errechnet sich $\text{AverageSim}_{\text{Part}}$:

$$\text{AverageSim}_{\text{Part}} = \frac{\sum_{i=1}^{|\text{Part}|} \sum_{j=1}^{|\text{Part}|} Sim(\text{Case}_i, \text{Case}_j)}{(\mid \text{Part} \mid)^2} \tag{3.12}$$

Die Abbildung 3.10 zeigt die zu berechnenden Abstände bei einer Partitionierung für das Attribut A_1.

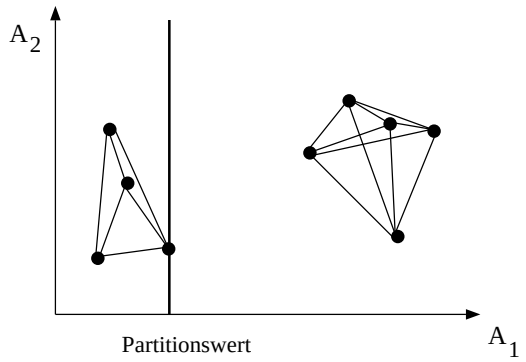


Abbildung 3.10: Maß der durchschnittlichen Ähnlichkeit

Dieser Mittelwert wird für die Partitionen *lower* und *upper* berechnet, die alle Fälle beeinhalten, die im zu untersuchenden Diskriminatorattribut kleiner bzw. größer als der Partitionswert sind. Anschließend werden diese beiden Werte addiert, nachdem sie mit einem Wichtungsfaktor multipliziert wurden, der die Größe der Partitionen angibt:

$$\text{AverageSim} := \frac{\mid lower \mid}{\mid S \mid}\text{AverageSim}_{\text{lower}} + \frac{\mid upper \mid}{\mid S \mid}\text{AverageSim}_{\text{upper}} \tag{3.13}$$

Es ist klar, daß hier sehr viele Ähnlicheiten berechnet werden müssen, eine solche Berechnung kann bei vielen Attributen sehr aufwendig werden.

Um nicht bei jedem inneren Baumknoten des k-d-Baumes immer die benötigten Ähnlicheiten erneut berechnen zu müssen, wurde bei der Implementierung dieses Maßes zunächst eine Matrix

mit allen Fällen und deren Ähnlichkeiten zueinander erzeugt und abgespeichert. So muß bei der obigen Berechnung nur der entsprechende Ähnlichkeitswert der Matrix entnommen werden.

Beispiel

Nun wird wieder an einem Beispiel die Funktionsweise des Maßes der durchschnittlichen Ähnlichkeit erläutert, eingesetzt wird wieder die selbe Fallbasis:

Person	Größe	Haarfarbe	Augenfarbe
A	klein	blond	blau
B	groß	rot	blau
C	groß	blond	blau
D	groß	blond	braun
E	klein	dunkel	blau
F	groß	dunkel	blau
G	groß	dunkel	braun
H	klein	blond	braun

Das Ähnlichkeitsmaß wird folgendermaßen definiert:

$$sim_{Größe}(x,y) := \begin{cases} 0 & \text{falls } x \neq y \\ 1 & \text{falls } x = y \end{cases}$$

$$sim_{Haar}(x,y) := \begin{cases} 0,5 & \text{falls } (x = \text{blond} \wedge y = \text{rot}) \vee (x = \text{rot} \wedge y = \text{blond}) \\ 1 & \text{falls } x = y \\ 0 & \text{sonst} \end{cases}$$

$$sim_{Augen}(x,y) := \begin{cases} 0 & \text{falls } x \neq y \\ 1 & \text{falls } x = y \end{cases}$$

Das Gesamt-Ähnlichkeitsmaß sei definiert als:

$$Sim(X,Y) := \frac{sim_{\text{Größe}}(X_{\text{Größe}}, Y_{\text{Größe}}) + sim_{\text{Haar}}(X_{\text{Haar}}, Y_{\text{Haar}}) + \frac{1}{2} sim_{\text{Augen}}(X_{\text{Augen}}, Y_{\text{Augen}})}{2,5}$$

Hier erkennt man bereits, daß durch das Ähnlichkeitsmaß weiteres Wissen definiert ist: Eine blonde Haarfarbe ist einer zu roten Haarfarbe ähnlicher als zu dunklem Haar, außerdem ist die Augenfarbe bei einem Vergleich zwischen Personen nicht so relevant wie Größe und Haarfarbe. Diese Information wurde bisher nicht genutzt.

Also berechnet sich die Ähnlichkeitsmatrix zu:

	A	B	C	D	E	F	G	H
A	1,0	0,4	0,6	0,4	0,6	0,2	0	0,8
B		1,0	0,8	0,6	0,2	0,6	0,4	0,2
C			1,0	0,8	0,2	0,6	0,4	0,4
D				1,0	0	0,4	0,4	0,2
E					1,0	0,6	0,4	0,4
F						1,0	0,8	0
G							1,0	0,2
H								1,0

Nun soll unser Maß für die einzelnen Partitionierungsvarianten Größe, Haar und Augen berechnet werden:

1. Partitionierung nach *Größe* in die Partitionsmengen $S_{A_{Größe}=\{klein\}} = \{A, E, H\}$ und $S_{A_{Größe}=\{groß\}} = \{B, C, D, F, G\}$.

$$\text{AverageSim}_{klein} = \frac{1,0 + 0,6 + 0,8 + 1,0 + 0,4 + 1,0}{6}$$

$$
\begin{aligned}
&= 0,8 \\
\text{AverageSim}_{groß} &= 0,72 \\
\text{AverageSim} &= \frac{3}{8} \cdot 0,8 + \frac{5}{8} \cdot 0,72 \\
&= 0,75
\end{aligned}
$$

2. Partitionierung nach *Haar* in die Partitionsmengen $S_{A_{Haar}=\{blond,rot\}} = \{A, B, C, D, H\}$ und $S_{A_{Haar}=\{dunkel\}} = \{E, F, G\}$.

$$
\begin{aligned}
\text{AverageSim}_{blond,rot} &= 0,6 \\
\text{AverageSim}_{dunkel} &= 0,8 \\
\text{AverageSim} &= 0,675
\end{aligned}
$$

3. Partitionierung nach *Augen* in die Partitionsmengen $S_{A_{Augen}=\{blau\}} = \{A, B, C, E, F\}$ und $S_{A_{Augen}=\{braun\}} = \{D, G, H\}$.

$$
\begin{aligned}
\text{AverageSim}_{blau} &= 0,653 \\
\text{AverageSim}_{braun} &= 0,633 \\
\text{AverageSim} &= 0,646
\end{aligned}
$$

In unserem Beispiel hätten wir also bei der Partitionierungsvariante „Größe“ die größte durchschnittliche Ähnlichkeit in unseren Partitionsmengen, man würde dieses Attribut zum Diskriminatorattribut machen.

Baut man mit dem Maß der durchschnittlichen Ähnlichkeit einen k-d-Baum mit maximaler Bucketgröße 2 für die Fallbasis auf, entsteht nachfolgender Baum (Abbildung 3.11).

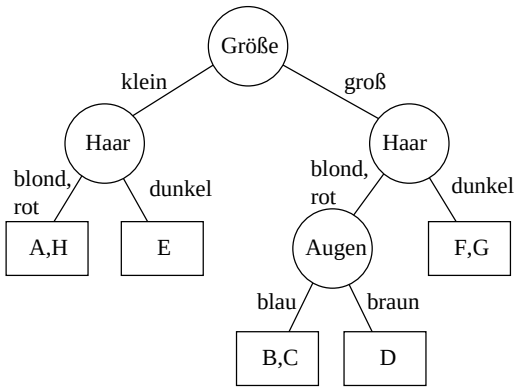


Abbildung 3.11: entstehender k-d-Baum mit durchschnittlicher Ähnlichkeit

3.2.6 Performance-Vergleich und Bewertung

Nun sollen die vorgestellten Konzepte verglichen und bewertet werden. Zu diesem Zweck wurde von einem Rechner der *UCI Repository Of Machine Learning Databases and Domain Theories* in den USA einige Datenbanken über das Internet kopiert, um so mit echten Daten arbeiten zu

können. Auf diesem Rechner befinden sich mehr als 20 Megabyte Datenmaterial, daß von jedem genutzt werden kann.

Für unsere Messungen wurde eine Datenbank ausgewählt, die eine Aufstellung von PKW's beinhaltet. Sie beschreibt 205 verschiedene Autos über 25 Attribute wie Hersteller, technische Daten, Preis und eine Risikoeinstufung für Versicherungen. In der benutzten Ähnlichkeitsfunktion werden die Attribute mit Wichtungsfaktoren klassifiziert. Es gibt 8 symbolische (Hersteller, Kraftstofftyp, Motortyp ...) und 17 numerische Attribute (Länge, Hubraum, Leistung, Preis ...). Eine genaue Beschreibung der verwendeten Datenbank befindet sich im Anhang.

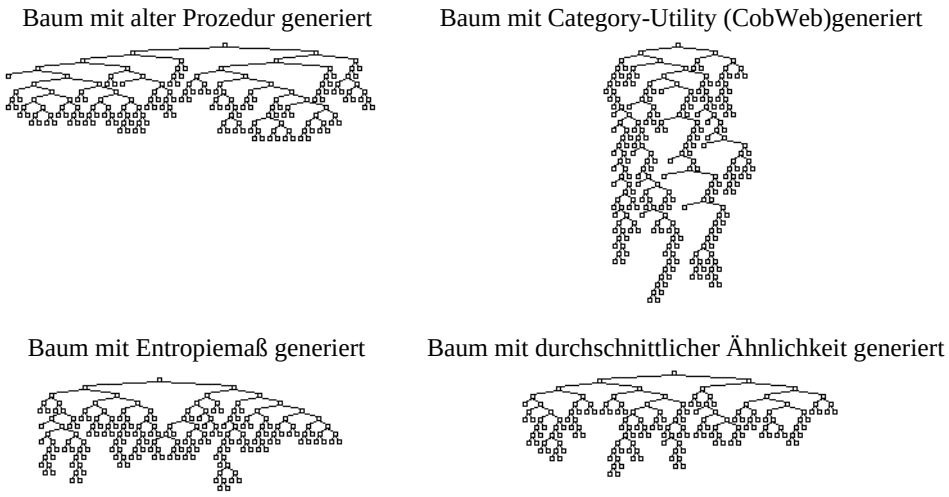


Abbildung 3.12: Die 4 aufgebauten Bäume

Nun wurden mit den verschiedenen Generierungsverfahren k-d-Bäume aufgebaut und auf diesen Bäumen ein Retrieval durchgeführt, um die Qualität der k-d-Bäume und somit der Generierungsprozedur zu bewerten. Ein „guter" Baum sollte also ein schnelles Retrieval erlauben. Die Abbildung 3.12 zeigt die vom System aufgebauten Bäume.

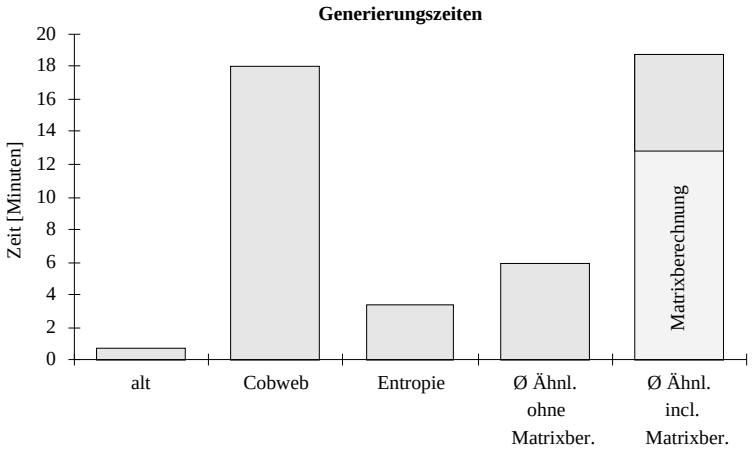


Abbildung 3.13: Generierungszeiten

Die benötigten Generierungszeiten sind in der Grafik 3.13 dargestellt, man erkennt, daß die

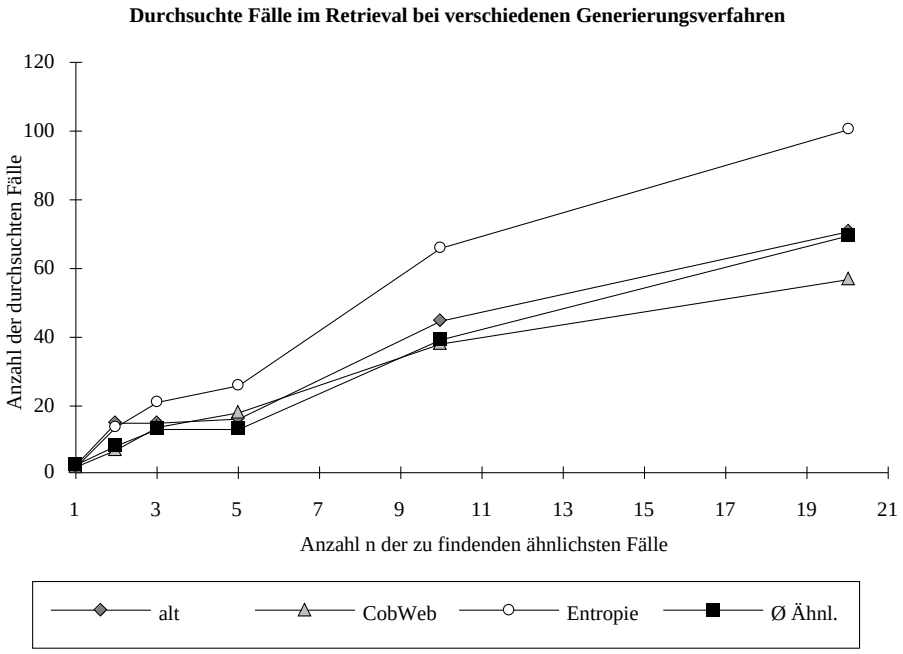


Abbildung 3.14: Durchsuchte Fälle

alte Generierungsprozedur am schnellten arbeitet. Dies ist aber nicht verwunderlich, da hier noch die a-priori-Abschätzung eingesetzt wird. Die anderen Generierungsvarianten Cobweb, Entropie und durchschnittliche Ähnlicheit arbeiten mit der a-posteriori-Abschätzung, es wird also für alle Attribute gesplittet und dann erst eine Bewertung vorgenommen.

In der Grafik wird die Variante, die sich des Maßes der durchschnittlichen Ähnlichkeit bedient, mit und ohne Berechnung der Ähnlichkeitsmatrix gezeigt. Da es möglich ist, bereits bei der Erstellung der Fallbasis diese Matrix zu berechnen und bei neuen Fällen gegebenfalls zu erweitern, kann für die reine Baumgenerierung die Zeit ohne Matrixaufbau angenommen werden.

Die dargestellten Zeiten sollen aber nicht wesentliches Bewertungsmerkmal sein, da der Baum nur einmalig aufgebaut werden muß, wichtigeres Kriterium sind die Retrievalzeiten.

Zur Abschätzung des Retrievalaufwandes wurde die Anzahl der durchsuchten Fälle, also diejenigen Fälle, mit denen ein Ähnlichkeitsvergleich in den Buckets durchgeführt werden mußte und die Anzahl der beim Retrieval durchgeführten Bounds-Tests gemessen. Die Funktion der Bounds-Tests wird im nächsten Kapitel genau erläutert, hier ist jedoch von Interesse, daß die Anzahl der Bounds-Tests den durch das Retrieval entstehenden Overhead zur Navigation innerhalb des k-d-Baumes angibt.

Die Grafik 3.14 zeigt die Anzahl der durchsuchten Fälle für verschiedene Retrievalanfragen, es wurden die jeweils 1, 2, 5, 10 und 20 ähnlichsten Fälle zum Anfragefall gesucht, der voll spezifiziert ist und sich selbst in der Fallbasis befindet. Die Grafik zeigt, daß die Generierungsprozedur, die sich des Maßes der durchschnittlichen Ähnlichkeit bedient, bessere Ergebnisse erzielt als die anderen Varianten, nur bei einer Anfrage für die 10 ähnlichsten Fällen muß die Variante mit dem Category-Utility (CobWeb) noch weniger Fälle untersuchen.

Bei der Anzahl der benötigten Ball-Tests (Grafik 3.15) wird der Unterschied noch deutlicher: Man erkennt, daß die Generierung mit der durchschnittlichen Ähnlichkeit auch bei der Navigation im Baum, die durch die Bounds-Tests gesteuert wird, am effizientesten arbeitet.

Die Anzahl der durchsuchten Fälle und der benötigten Bounds-Tests bei den einzelnen Vari-

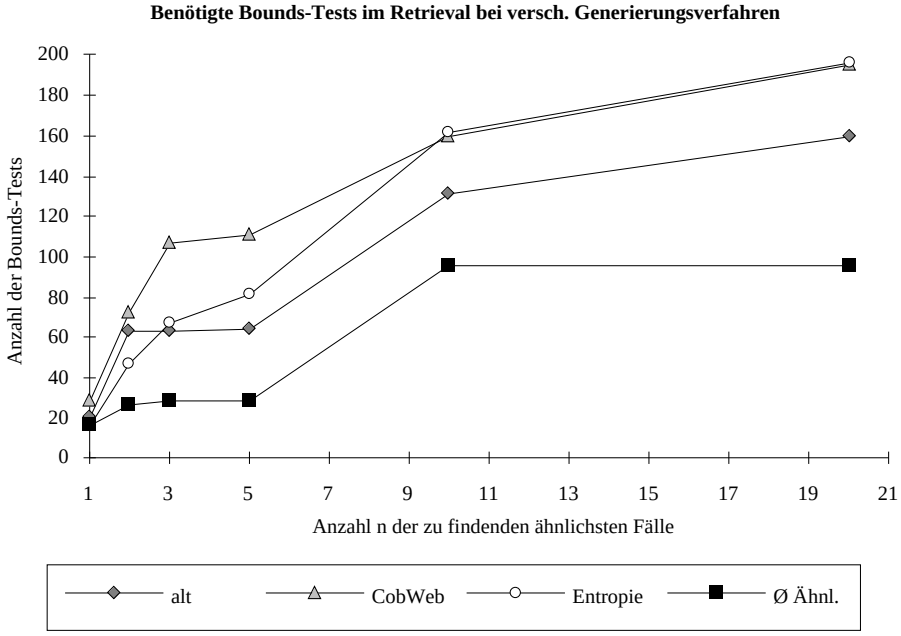


Abbildung 3.15: Bounds-Tests

anten werden durch Zeitmessungen für das Retrieval bestätigt (siehe Grafik 3.16). Zum Vergleich ist der Zeitaufwand der einfachen linearen Suche in die Grafik eingetragen, sie ist unabhängig von der Anzahl der zu suchenden Fälle, da ohnehin die gesamte Fällbasis durchsucht werden muß.

Bewertung

Im folgenden soll nun eine Bewertung der drei neu vorgestellten Generierungsvarianten vorgenommen werden. Sie arbeiten alle mit der *a-posteriori*-Abschätzung, bei der in allen Attributen partitioniert und anschließend die Güte der Partitionierung untersucht wird.

- **Generierung mit dem Category-Utility aus CobWeb**
  Der hinter dem Category-Utility stehende Grundgedanke Attribute zu bevorzugen, die einen großen Einfluß auf andere Attribute haben, ist sicherlich gut. Problematisch ist jedoch die starke Einschränkung auf symbolische Attribute, selbst die CLASSIT-Variante mit der Varianz stellt hier keine befriedigende Lösung dar, den für die Berechnung der Varianz wird der Mittelwert benötigt, der bei bestimmten Attributen nicht ermittelt werden kann. Da sich das Category-Utility nicht die gegeben Ähnlichkeitsfunktion nutzt, kann sie diese zusätzlich Information auch nicht beim Aufbau verwenden.

- **Generierung mit dem Entropie-Maß**
  Auch die Variante mit Nutzung des Entropie-Maßes hat bereits prinzipielle Nachteile. Auch hier bleibt das Ähnlichkeitsmaß ungenutzt, es wird mit Wahrscheinlichkeiten gearbeitet. Ein weiteres Problem dieser Generierung besteht in der Nutzung der Diagnose. Zunächst muß nicht bei jeder Fallbasis eine geeignete Diagnose zur Verfügung stehen, auch die Auswahl der richtigen Diagnose ist hier ausgesprochen kritisch, da die komplette Generierung von diesem Faktor abhängt.

- **Generierung mit der durchschnittlichen Ähnlichkeit**
  Diese Generierungsvariante wurde konsequent für unser System entwickelt, sie bedient sich

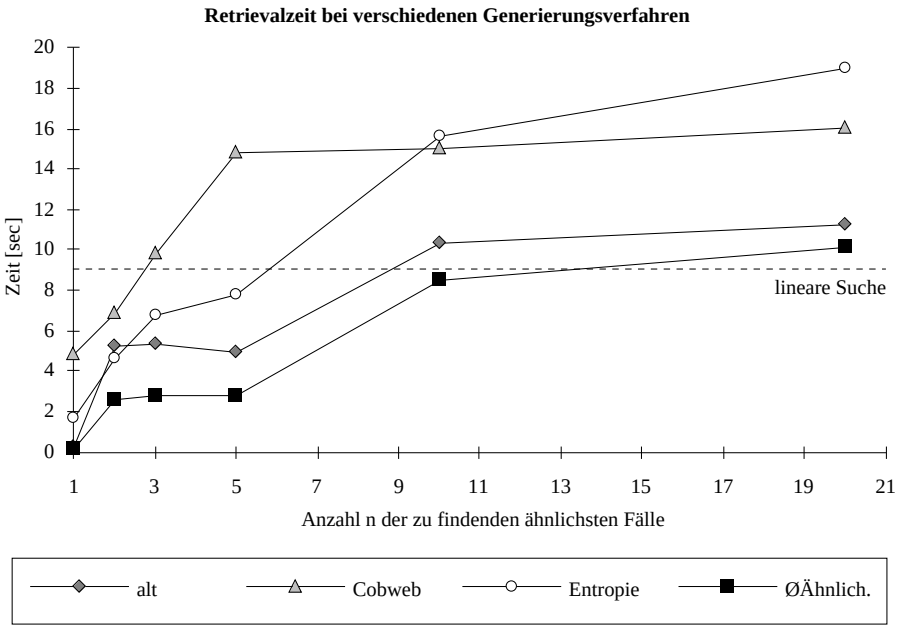


Abbildung 3.16: Retrievalzeiten

bei allen Entscheidungen des Ähnlichkeitsmaßes. Der Gedanke, Teilräume und Buckets zu bilden, deren Fälle möglichst ähnlich zueinander sind, unterstützt optimal die Anforderungen des Retrievals, außerdem wurde hier eine große Flexibilität erreicht, man benötigt keine Diagnose und ist nicht auf bestimmte Attributtypen eingeschränkt, da alles benötigte Wissen bereits durch die Fälle und das Ähnlichkeitsmaß repräsentiert ist.

Diese theoretischen Überlegungen werden von den praktischen Messungen unterstrichen, es wurde eine deutliche Performancesteigerung erreicht.

3.3 Wahl der Bucketgröße

Bisher wurde vor dem Aufbau des k-d Baumes festgelegt, wie groß die Buckets sein sollten, d.h. wie viele Fälle sie maximal aufnehmen können. Nun ist es aber nicht sinnvoll, Fälle die zueinander sehr ähnlich sind, nochmals zu partitionieren und in verschiedene Teilbäume zu zerteilen (siehe Abbildung 3.17). Dies würde den Generierungs- und Retrieval-Aufwand nochmals steigern, ohne Vorteile zu bringen.

So wurde versucht, beim Aufbau dynamisch die Bucketgröße zu variieren. Für die zu partitionierende Fallmenge wird die durchschnittliche Ähnlichkeit berechnet (siehe Abschnitt 3.2.5), liegt diese Ähnlichkeit über einem bestimmten Grenzwert, wird ein Bucket erzeugt, sonst wird partitioniert und ein innerer Knoten generiert.

Allerdings soll hier nicht verschwiegen werden, daß die Wahl dieses Grenzwertes kritisch ist, hier ist eine Anpassung an die zu verwaltenden Fälle notwendig. In manchen Fallbasen liegen die Fälle relativ dicht zusammen, in anderen ist der Datenraum dünn besetzt. Auch die Retrieval-Anfragen spielen hier eine Rolle, werden viele zum Anfragefall ähnliche Fälle gesucht, sind beispielsweise große Buckets von Vorteil.

Die Abbildung 3.18 zeigt einen konventionell aufgebauten k-d-Baum mit einer festen Bucketgröße. Teilbäume, deren enthaltene Fälle einen Schwellwert für die durchschnittliche Ähnlichkeit

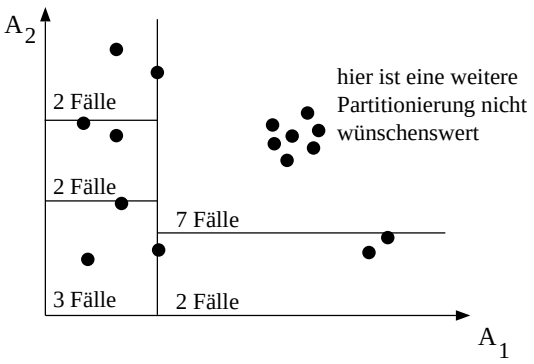


Abbildung 3.17: Dynamische Wahl der Bucketgröße

Abbildung 3.18: k-d-Baum mit fester Bucketgröße

überschritten haben, sind schwarz markiert. In der Abbildung 3.19 ist nun ein k-d-Baum mit dyna-

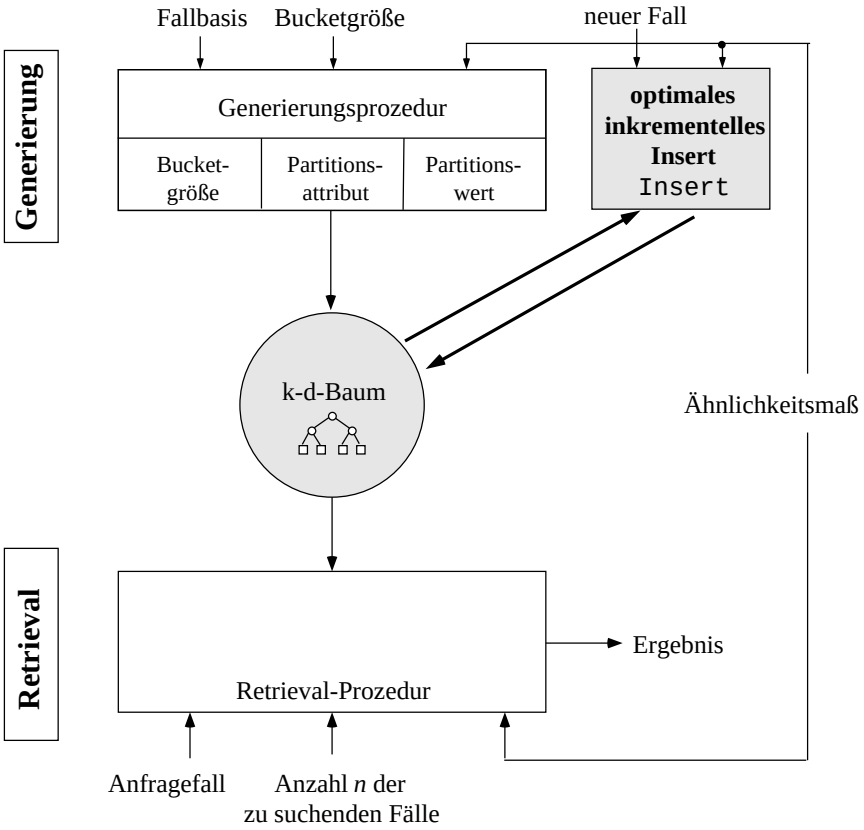
Abbildung 3.19: k-d-Baum mit dynamischer Bucketgröße

mischer Bucketgröße dargestellt. Die Fälle, deren durchschnittliche Ähnlichkeit einen definierten Schwellwert überschritten haben, wurden in einem Bucket abgespeichert.

3.4 Inkrementeller Aufbau von k-d-Bäumen

Da sich die Generierungsprozedur mit dem Maß „maximale durchschnittliche Ähnlichkeit“ als beste Variante erwiesen hat, zeigt sich nun das Problem, daß der Aufbau hierbei doch aufwendiger und somit langsamer ist als die bisherige Lösung mit einer *a-priori*-Abschätzung. Im bestehenden System gab es keine Möglichkeit, neue Fälle optimiert einzufügen, d.h. nach Eintrag eines neuen Falles war der k-d Baum nicht im selben Zustand als wären alle Fälle in einem Zug durch die Generierungsprozedur eingefügt worden. Bisher kam eine Optimierungsprozedur zum Einsatz, die den Baum quasi neu aufbaute. Da die Generierung und Optimierung mit dem neuen Maß nun deutlich aufwendiger geworden ist, wäre ein solches Vorgehen nicht mehr tragbar.

Im folgenden wird nun eine Einfügeprozedur *Insert* entwickelt, die in einen bestehenden k-d Baum einen neuen Fall einfügt. Nach dem Einfügen soll es gesichert sein, daß dennoch in jedem inneren Knoten jeweils das optimale Diskriminatorattribut (Partitionen mit maximaler durch-



schnittlicher Ähnlichkeit) und der optimale Partitionswert (Median oder größter Abstand) gewählt ist.

Der Hauptaufwand in der bestehenden Generierungsprozedur besteht darin, die Ähnlichkeiten innerhalb der Partitionen zu ermitteln und zu addieren. Der Grundgedanke des Einfügealgorithmus ist es, die bisherigen Ähnlichkeitssummen innerhalb der Partitionen $\sum_{i=1}^{|\mathrm{Part}|}\sum_{j=i}^{|\mathrm{Part}|} Sim(\mathrm{Case}_i, \mathrm{Case}_j)$ bei der Generierung im betreffenden inneren Baumknoten abzuspeichern. Beim Einfügen wird nun für alle Atribute überprüft, in welchen Sohn (Partition) des betreffenden Knotens der neue Fall gehört. Nun kann die neue durchschnittliche Ähnlichkeit leicht errechnet werden, die Ähnlichkeitssumme des anderen Sohns bleibt unberührt, zu der Ähnlicheitssumme des Sohnes, in den eingefügt wurde, werden nur die Ähnlichkeiten des neuen Falles zu den bisher in dieser Partition befindlichen Fällen errechnet und addiert. Nun werden die beiden Ähnlichkeitssummen durch die Anzahl der enthaltenen Ähnlicheiten $\sum_{k=1}^{|\mathrm{Part}|} k$ dividiert, anschließend mit den Wichtungsfaktoren $\frac{|\mathrm{Part}|}{|S|}$ multipliziert und dann addiert. So erhält man die aktuelle durchschnittliche Ähnlichkeit.

Die folgenden Formeln zeigen die Ermittlung der neuen durchschnittlichen Ähnlichkeit für die Variante, daß der neue Fall in die Partition *lower* eingefügt wurde, die Berechnung bei Einfügen in die *upper*-Partition erfolgt analog. Die neue Ähnlichkeitssumme der *lower*-Partition ist also:

$$\mathrm{AverageSim}_{\mathrm{lower,NEW}} := \frac{\sum_{i=1}^{|\mathrm{Part}|} Sim(\mathrm{Case}_{\mathrm{new}}, \mathrm{Case}_j)}{\sum_{k=1}^{|\mathrm{Part}|} k} + \mathrm{AverageSim}_{\mathrm{lower,OLD}} \tag{3.14}$$

Die neue durchschnittliche Ähnlichkeit für den Knoten, in dem partitioniert wird:

$$\mathrm{AverageSim} := \frac{|\ lower_{\mathrm{old}}\ | +1}{|\ S_{\mathrm{old}}\ | +1} \cdot (\mathrm{AverageSim}_{\mathrm{lower,OLD}} + \mathrm{AverageSim}_{\mathrm{lower,NEW}})$$

$$+\frac{\mid upper_{\text{old}} \mid +1}{\mid S_{\text{old}} \mid +1}\text{AverageSim}_{\text{upper,OLD}} \tag{3.15}$$

Ergibt der Test, daß trotz des eingefügten neuen Falles Diskriminatorattribut und Partitionswert gleich bleiben, muß hier nicht neu partitioniert werden, der Sohn, in den nicht eingefügt wurde, ist bereits korrekt. Nun wird *Insert* solange mit den entsprechenden Teilbäumen aufgerufen, in die der neue Fall eingefügt wird, bis ein Bucket erreicht wird oder ein Neuaufbau notwendig wird.

Muß laut Test ein anderer Partitionswert oder Diskriminatorattribut gewählt werden, ist mit den entsprechenden Fällen die Generierungsprozedur aufzurufen, ein neuer Baum wird für die verbliebenen Fälle erzeugt. Weiter oben im Baum muß aber nichts geändert werden, wenn dort Partitionswert und Diskriminatorattribut noch korrekt waren.

Nach der Implementierung zeigte es sich, daß ein kompletter Neuaufbau des Baumes nur selten notwendig wurde, meist mußten nur kleine Teilbäume neu erzeugt werden. Je größer die durch einen inneren Knoten repräsentierte Fallmenge, desto geringer ist die Wahrscheinlichkeit, daß sich das Diskriminatorattribut oder der Partitionswert durch nur einen Fall verändert, große Bäume sind also sehr stabil.

Bei dieser neuen Einfügeprozedur hat sich die Wahl des Partitionswertes beim größten Abstand zwischen den einzelnen Attributsausprägungen nicht-symbolischer Attribute bewährt, denn hierdurch ist der Partitionswert beim Neueinfügen stabiler geworden. Sind beispielsweise die Attributsausprägungen eines Attributes (1 2 2 5 7 8) und man fügt einen Fall mit der Ausprägung (6) in diesem Attribut ein, so wechselt der Median von 2 auf 6, der größte Abstand liegt aber unverändert zwischen 2 und 5.

Beispiel

Hier soll ein neuer Fall X in einen bestehenden Baum eingefügt werden. Zunächt wird der Fall in die Wurzel (1) eingefügt, die Tests ergeben, daß sich das Diskriminatorattribut und der Partitionswert nicht ändern. So wird der neue Fall in den linken Teilbaum (2) übergeben, wo erneut eine Überprüfung des Partitionswertes und des Diskriminatorattributes vorgenommen wird, auch hier kann der Knoten unverändert bleiben.

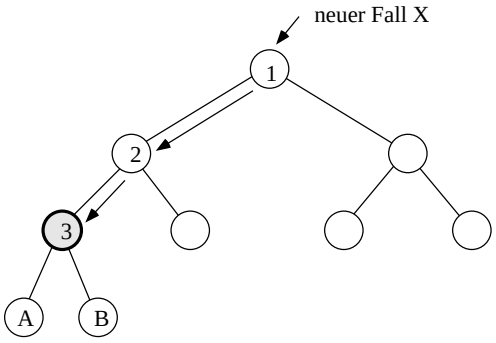


Abbildung 3.20: Inkrementelles Insert

Beim Einfügen in den Teilbaum 3 ergibt die Überprüfung, daß das bisherige Diskriminatorattribut nicht mehr optimal ist, mit den verbleibenden Fällen A und B sowie dem neu einzufügenden Fall X wird ein neuer Baum aufgebaut und damit Teilbaum 3 ersetzt (Abbildung 3.21).

Das Beispiel zeigt, daß ein großer Teil des alten Baumes übernommen werden konnte, ein totaler Neuaufbau ist nicht notwendig gewesen.

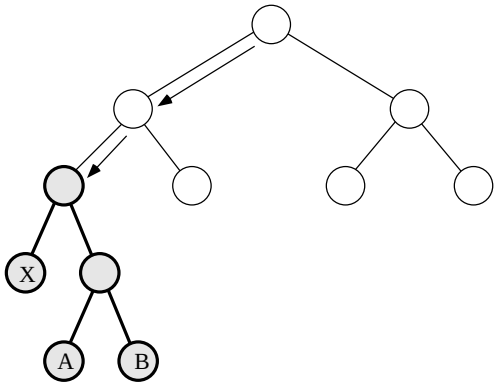


Abbildung 3.21: Einfügen eines neuen Teilbaumes

Kapitel 4

Retrieval in k-d-Bäumen

Im vorangegangen Kapitel wurde erläutern, wie ein möglichst günstiger k-d-Baum aufgebaut wird, der die Fälle bereits klassifiziert und somit eine schnelle Suche erlauben soll, wird jetzt das Retrieval der Fälle innerhalb dieser Bäume erläuert. Zu diesem Zweck wird eine Retrieval-Prozedur *Search* entwickelt, die auf den Bäumen arbeitet (siehe Abbildung 4.1).

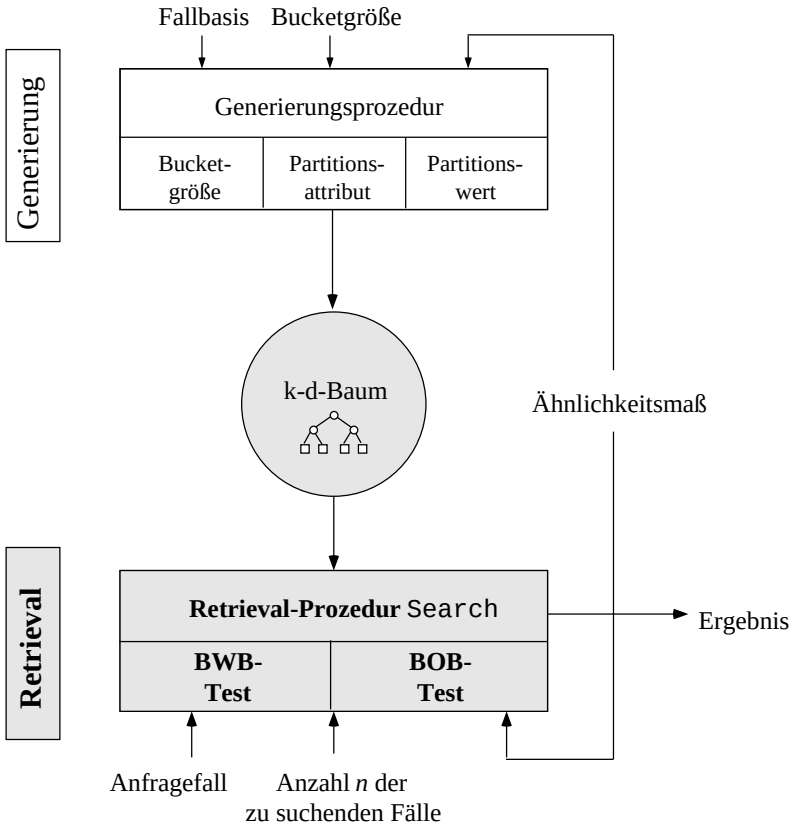


Abbildung 4.1: Systemstruktur

4.1 Die Retrieval-Prozedur

In den aufgebauten k-d-Bäumen soll eine *Nearest-Neighbour*-Suche implementiert werden, zu einem in der Suchanfrage spezifizierten Fall sind die n bezüglich des vorgegeben Ähnlichkeitsmaßes am nächsten gelegen Fälle zu finden (siehe Abbildung 4.2). Gegeben sind hierbei:

- die in der Anfrage zu spezifizierenden k Schlüsselwerte
- die Anzahl m der zu bestimmenden Nachbarn (Falls die in der Anfrage spezifizierten Schlüsselwerte genau in dieser Ausprägung in der Fallbasis enthalten sind, die Ähnlichkeit des gefundenen Falles zu dem Anfragefall also 1 ist, so zählt dieser gefundene Fall auch als ein Nachbar)
- der k-d-Baum, der die Zugriffsstruktur auf die Fallbasis darstellt

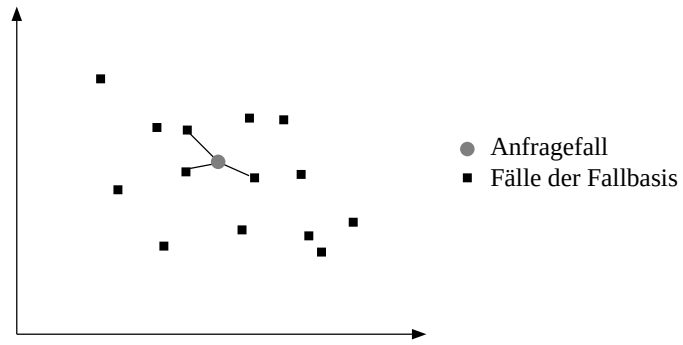


Abbildung 4.2: Nearest-Neighbour-Suche

Der in der Anfrage angebene Fall muß also nicht in der tatsächlichen Fallmenge enthalten sein.

Nun kann man mit Hilfe der k-d-Baum Struktur schnell auf den Bucket zugreifen, in dem der Anfragepunkt liegt, indem man einfach ein mal in dem Baum von der Wurzel zu den Blättern läuft und an jedem inneren Knoten in denjenigen Teilbaum eintritt, der die Attributsausprägungen den Anfragefalles enthält. Jetzt kann jedoch das Problem auftreten (und dies passiert auch meist), daß es weitere Fälle gibt, die zwar dicht am Anfragepunkt, aber dennoch in einem anderen Bucket liegen (siehe Abbildung 4.3). Um diese Situation zu erkennen, wird der sogenannte *Ball-Within-Bounds-Test* (siehe Abschnitt 4.1.1) angewendet. Sind die nächsten Fälle alle im aktuellen Bucket, so ist die Suche beendet.

Müssen noch weitere Buckets untersucht werden, so wird überprüft, in welchen anderen Buckets und Teilbäumen noch Fälle liegen könnten, die ähnlicher zum Anfragefall als die bisher gefunden Fälle sein könnten. Diese Überprüfung wird durch den *Ball-Overlap-Bounds*-Test vorgenommen (Abschnitt 4.1.1).

Die Retrieval-Prozedur arbeitet mit einigen globalen Variablen:

- $A[1 \ldots k]$ enthält eine Spezifikation der k Suchattribute, repräsentiert also den Anfragefall.
- $Queue[1 \ldots m].Case$ eine Queue, die während der Suche die bisher gefundenen Fälle enthält. Die Queueelemente sind absteigend nach ihren Ähnlichkeiten zum Anfragefall sortiert, das heißt der ähnlichste gefundene Fall befindet sich in $Queue[1].Case$.
- $Qeuue[1 \ldots m].Sim$ eine Queue, die während der Suche die Ähnlichkeiten der bisher gefundenen Fälle zum Anfragefall enthält ($Qeuue[i].Sim = Sim(Queue[i].Case, A)$)

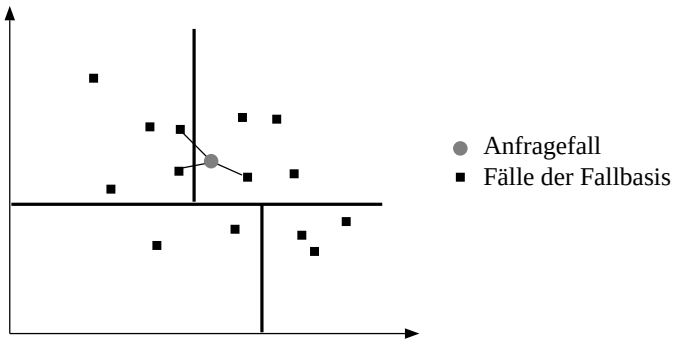


Abbildung 4.3: Nearest-Neighbour-Suche

- $Bounds[1 \ldots k].Upper$ enthält die oberen Koordinatenschranken des aktuellen Baumknotens.
- $Bounds[1 \ldots k].Lower$ enthält die unteren Koordinatenschranken des aktuellen Baumknotens.

und sieht also folgendermaßen aus:

```
PROCEDURE Search (treeNode)
VAR
BEGIN
   IF (TerminalNode(treeNode)) THEN
      [Aktualisiere queue mit dem Inhalt von treeNode] .
      IF BallWithinBounds THEN [Fertig] ELSE RETURN.   BWB1
   END.
   Disc:=treeNode.discriminator.
   Part:=treeNode.partitionValue.
   IF A[Disc]<=Part THEN
      temp:=Bounds.Upper[Disc].
      Bounds.Upper[Disc]:=Part.
      Search(treeNode.lowerSon).
      Bounds.Upper[Disc]:=temp.
   END
   ELSE
      temp:=Bounds.Lower[Disc].
      Bounds.Lower[Disc]:=Part.
      Search(treeNode.upperSon).
      Bounds.Lower[Disc]:=temp.
   END.

   IF A[Disc]<=Part THEN
      temp:=Bounds.Lower[Disc].
      Bounds.Lower[Disc]:=Part.
      IF BoundsOverlapBall THEN Search(treeNode.upperSon).   BOB
      Bounds.Lower[Disc]:=temp.
      END
      ELSE
```

```
      temp:=Bounds.Upper[Disc].
      Bounds.Upper[Disc]:=Part.
      IF BoundsOverlapBall THEN Search(treeNode.lowerSon).   BOB
      Bounds.Upper[Disc]:=temp.
      END.

   IF BallWithinBounds THEN [Fertig] ELSE RETURN.   BWB2
END.
```

Ein Suchaufruf läuft folgendermaßen ab:

1. Belegung des globalen Arrays A mit dem Anfragefall
2. Initialisierung der globalen Variablen:
   - $Queue[1 \ldots m].Case := nil$
   - $Queue[1 \ldots m].Sim := -\infty$
   - $Bounds[1 \ldots k].Upper := \infty$
   - $Bounds[1 \ldots k].Lower := -\infty$
3. Aufruf von `Search(root)`, wobei `root` ein Zeiger auf die Wurzel des k-d-Baums ist.

Ablauf der Prozedur Search

Zunächst wird überprüft, ob der Eingabeknoten ein Blattknoten ist, in diesem Fall wird die Queue, welche die bisherigen Suchergebnisse enthält, mit den Fällen des Blattknotens aktualisiert. Zeigt ein Test an, daß an keiner anderen Stelle bessere Fälle gefunden werden können, kann die Suche ganz beendet werden, ansonsten wird mit `RETURN` eine Rekursionsstufe zurück gesprungen. Diese Überprüfung erledigt der *Ball-Within-Bounds*-Test.

Ist der aktuelle Eingabeknoten jedoch ein innerer Baumknoten, so wird `Search` in dem Wurzelknoten desjenigen Teilbaums erneut aufgerufen, in dem sich der Anfragefall befinden muß. Dies geschieht, indem der Partitionswert des inneren Baumknotens mit dem Wert des Anfragefalles im Diskriminatorattribut verglichen wird. Zuvor werden jedoch die Koordinatenschranken geändert, da der Suchraum mit dem Eintreten in einen Teilbaum genauer spezifiziert ist. Die alten Koordinatenschranken werden in der Variable `temp` zwischengespeichert, um sie später wiederherstellen zu können.

Ist jetzt der vorhergehende `Search`-Aufruf abgearbeitet, ist man sicher, daß die besten Fälle im bereits abgearbeiteten Teilbaum gefunden wurden. Nun wird überprüft, ob sich in dem Teilbaum, der im vorhergenden Schritt nicht abgearbeitet wurde auch noch Fälle befinden könnten, die bezüglich des Ähnlichkeitsmaßes näher am Anfragefall liegen als die bisher gefunden Fälle. Diese Aufgabe übernimmt der *Ball-Overlap-Bounds-Test* (Abschnitt 4.1.1). Gibt der *Ball-Overlap-Bounds-Test* an, daß in dem anderen Teilbaum auch noch gesucht werden muß, wird `Search` mit der Wurzel dieses Teilbaums aufgerufen.

Zum Abschluß wird überprüft, ob die jetzt gefunden Fälle ausreichen oder ob noch weitere Fälle durchsucht werden müssen. Dies führt der *Ball-Within-Bounds-Test* (Abschnitt 4.1.1) mit den aktuellen Koordinatenschranken durch. Sind die betrachteten Fälle ausreichend, kann das Retrieval ganz beendet werden, ansonsten muß die Suche eine Rekursionsstufe höher weiterlaufen.

4.1.1 Die Bounds-Tests

Die prinzipielle Vorgehensweise der Bounds-Tests soll an der Abbildung 4.4 klargemacht werden. Im Beispiel soll der zu dem Anfragefall am ähnlichten Fall gefunden werden, der Fall $\mathcal{A}$ sei bereits lokalisiert. Nun legt der Bounds-Test eine k-dimensionale Kugel (in unserem Beispiel also 2-dimensional) so um den Anfragefall. Der Mittelpunkt der Kugel sei der Anfragefall, der bisher gefundene Fall $\mathcal{A}$ liege auf der Oberfläche der Kugel.

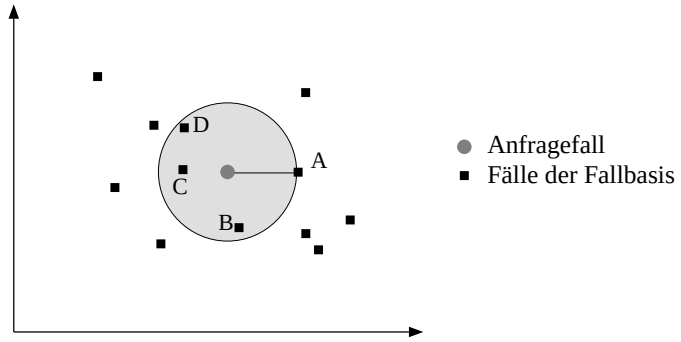


Abbildung 4.4: Ein Bounds-Test

Alle weiteren Fälle $\mathcal{B}$, $\mathcal{C}$ und $\mathcal{D}$ die sich nun innerhalb der Kugel befinden, sind also mindenstens ebenso ähnlich zu dem Anfragefall als der bisher aufgefundene Fall $\mathcal{A}$.

Da wir bei der Ähnlichkeitsberechnung die einzelnen Dimensionen auch unterschiedlich gewichten können, entsteht natürlich nur eine Kugel, wenn alle Dimensionen gleich gewichtet sind. Bei einer unterschiedlichen Gewichtung der einzelnen Dimensionen können auch andere Formen entstehen (siehe Abbildung 4.5), hier wird in der Ähnlichkeitsfunktion das in der horizontalen Achse dargestellte Attribut stärker gewichtet.

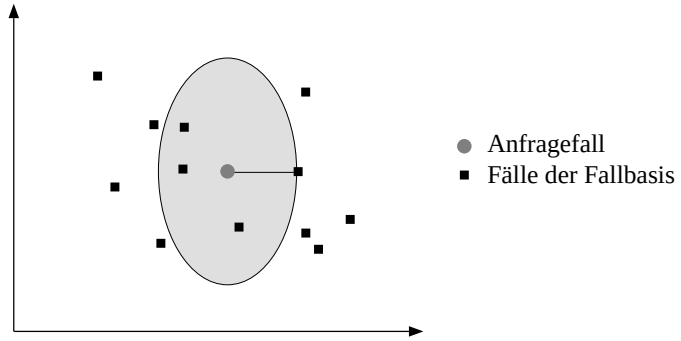


Abbildung 4.5: Ein Bounds-Test mit unterschiedlichen Wichtungsfaktoren

Zur vereinfachten Darstellung und Erklärbarkeit seien nun alle Dimensionen gleich gewichtet.

Der Ball-Overlap-Bounds-Test

Die Aufgabe des Ball-Overlap-Bounds-Tests (im folgenden als *BOB-Test* bezeichnet) in der `Search`-Prozedur ist es festzustellen, ob auch noch der andere Teilbaum des aktuellen Knotens abgearbeitet werden muß. Hierzu erhält der BOB-Test als Eingabe:

- den Anfragefall, spezifiziert über $A[1 \ldots k]$
- den Wert $Queue[m].Sim$, also die Ähnlichkeit des bisher gefundenen m-ten Falls zum Anfragefall ($= Sim(Queue[m].Case, A)$)
- die Koordinatenschranken des Teilbaums, für den überprüft werden soll, ob er noch abgearbeitet werden muß, spezifiziert über die Strukturen $Bounds[1 \ldots k].Upper$ und $Bounds[1 \ldots k].Lower$

Nun muß der BOB-Test überprüfen, ob die Kugel mit dem Mittelpunkt $A[1 \ldots k]$ und dem Radius $Queue[m].Sim$ die Koordinatenschranken des eventuell noch zu untersuchenden Teilbaums überlappt, das heißt, ob auch im anderen Teilbaum noch Fälle liegen können, die besser als die bisher gefundenen Fälle sind. Diese Situation ist in Abbildung 4.6 dargestellt. Hier befindet sich der Anfragefall $\mathcal{A}$ im Teilbaum I, der BOB-Test soll nun testen, ob noch in Teilbaum II gesucht werden muß.

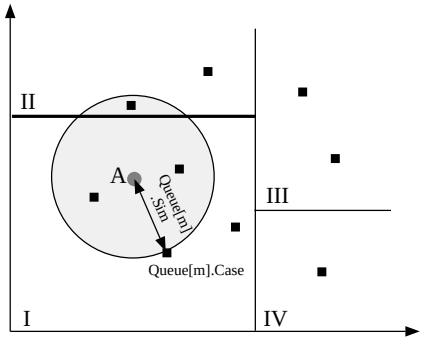


Abbildung 4.6: BOB-Test

Zu diesem Zweck konstruiert der BOB-Test einen Punkt X_{min}, der am nächsten zum Anfragefall $\mathcal{A}$ liegt, sich aber dennoch im den Koordinatenschranken des Teilbaums II befindet (siehe Abbildung 4.7).

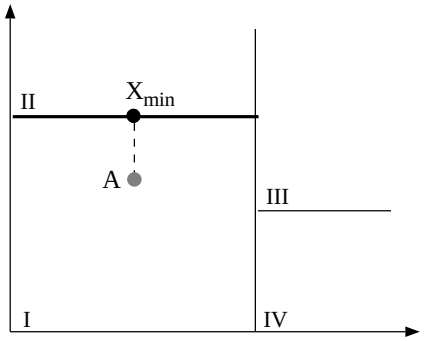


Abbildung 4.7: BOB-Test: Konstruktion von X_{min}

Der Punkt $X_{min}[1 \ldots k]$ berechnet sich dann folgendermaßen:

$$X_{min}[i] := \begin{cases} Bounds[i].Upper & \text{falls } A[i] > Bounds[i].Upper \\ Bounds[i].Lower & \text{falls } A[i] < Bounds[i].Lower \\ A[i] & \text{sonst} \end{cases} \tag{4.1}$$

X_{min} ist also ein Punkt auf der Oberfläche eines n-dimensionalen Würfels, der die Koordinaten eines bestimmten Suchraums (nämlich den betreffenden Teilbaum) beschreibt. Um jetzt zu überprüfen, ob dieser Punkt innerhalb unserer n-dimensionalen Kugel liegt, wird einfach die Ähnlichkeit zwischen dem Mittelpunkt der Kugel ($A[1 \ldots k]$) und X_{min} errechnet. Ist diese Ähnlichkeit größer (d.h. die Distanz kleiner) als der Radius der Kugel (gegeben durch $Queue[m].Sim$), so liegt X_{min} in der Kugel. Also gilt:

$$\boxed{BOB = true \quad \text{gdw} \quad Sim(A, X_{min}) \geq Queue[m].Sim} \tag{4.2}$$

Der Ball-Within-Bounds-Test

Während der Ball-Overlap-Bounds-Test bestimmt, ob ein bestimmter Teilbaum noch Fälle enthalten kann, die näher zum Anfragefall liegen als die bisher gefundenen Fälle, ist es Aufgabe des Ball-Within-Bounds-Tests (*BWB-Test*) zu entscheiden, ob alle zu betrachtenden Fälle auch tatsächlich berücksichtigt wurden. Hierzu erhält der BWB-Test als Eingabe:

- den Anfragefall, spezifiziert über $A[1 \ldots k]$
- den Wert $Queue[m].Sim$, also die Ähnlichkeit des bisher gefundenen m-ten Falls zum Anfragefall ($= Sim(Queue[m].Case, A)$)
- die Koordinatenschranken des aktuellen Teilbaums, spezifiziert durch $Bounds[1 \ldots k].Upper$ und $Bounds[1 \ldots k].Lower$

Liegt die Kugel mit dem Mittelpunkt $A[1 \ldots k]$ und dem Radius $Queue[m].Sim$ ganz in den Koordinatenschranken des aktuellen Teilbaums (beschrieben durch $Bounds[1 \ldots k].Upper$ und $Bounds[1 \ldots k].Lower$), so können keine ähnlicheren Fälle mehr in anderen Teilbäumen gefunden werden, da die Suchräume der Teilbäume disjunkt sind (siehe Abbildung 4.8).

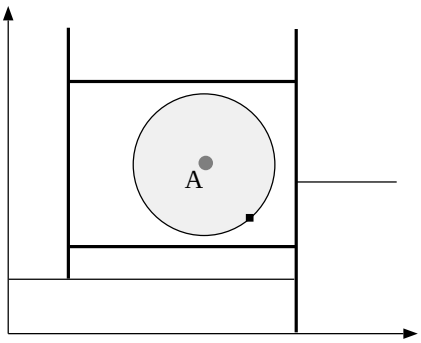


Abbildung 4.8: BWB-Test

Wie beim BOB-Test werden jetzt wieder Punkte auf der Oberfläche des durch die Koordinatenschranken beschriebenen n-dimensinalen Würfels konstruiert, zu denen dann die Ähnlichkeit berechnet wird. Da nun aber gewährleistet werden muß, daß die Kugel *keine* Koordinatenschranke überschneidet, sind für jede Dimension j 2 Punkte X^j_{lower} und X^j_{upper} zu betrachten (siehe Abbildung 4.9).

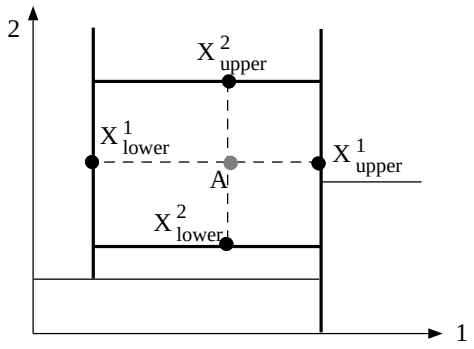


Abbildung 4.9: BWB-Test: Kontstruktion der Testpunkte

Die jeweiligen Punkte müssen für alle Dimensionen $j \in \{1, \ldots, k\}$ folgendermaßen berechnet werden:

$$X^k_{lower}[j] := \begin{cases} Bounds[j].Lower & \text{falls } j = k \\ A[j] & \text{falls } j \neq k \end{cases}$$

sowie

$$X^k_{upper}[j] := \begin{cases} Bounds[j].Upper & \text{falls } j = k \\ A[j] & \text{falls } i \neq k \end{cases} \tag{4.3}$$

Nun wird die Ähnlichkeit der Anfragefalles zu den konstruierten Punkten X^k_{lower} und X^k_{upper} errechnet, ist eine dieser Ähnlichkeiten größer als $Queue[m].Sim$, so liegt die Kugel nicht vollkommen in den untersuchten Koordinatenschranken.

Um einen Ähnlichkeitstest zu sparen, wird zunächst geprüft, ob der Punkt X^k_{lower} oder der Punkt X^k_{upper} näher am Anfragefall liegt. Dieser Test ist einfach zu realisieren, da sich der Anfragefall und die konstruierten Punkte nur in der Dimension k unterscheiden, so muß auch nur in dieser Dimension getestet werden. Anschließend wird die Ähnlichkeit des näher liegenden Punkts zum Anfragefall errechnet und mit $Queue[m].Sim$ verglichen.
Also gilt:

$$\boxed{BWB = true \quad \text{gdw} \quad \forall j = 1, \ldots, k : Sim(A, X^j) < Queue[m].Sim} \tag{4.4}$$

wobei

$$X^j := \begin{cases} X^k_{lower} & \text{falls } sim(X^k_{lower}[j], A[j]) \leq sim(X^k_{upper}[j], A[j]) \\ X^k_{upper} & \text{sonst} \end{cases}$$

4.1.2 Beispiel

Nun soll an einem Beispiel das Retrieval verdeutlicht werden. Zur einfachen Anschauung sollen die Dimensionen auf 2 beschränkt werden, ein k-d-Baum mit der Bucketgröße 2 sei bereits aufgebaut. Die Retrieval-Prozedur soll die zwei nächsten Nachbarn zum Fall $(A_1 = 5, A_2 = 2.5)$ finden. In der Beschreibung der Vorgehensweise wird auf die Anmerkungen an der Retrieval-Prozedur `Search` hingewiesen, die auf der Seite 44 dargestellt ist. Bei Berechnung der Ball-Tests soll hier nicht genau die Konstruktion der Testpunkte dargestellt werden, die Ball-Tests sind hier in den Zeichnungen grafisch durchgeführt. Die in ein Koordinatensystem eingetragenen Fälle und er entsprechende k-d-Baum ist in Abbildung 4.10 dargestellt.

Zunächst muß noch ein Ähnlichkeitsmaß für für die 2-dimensionalen Fälle definiert werden. Um das Beispiel im Koordinatensystem besser verdeutlichen zu können, wählt man die euklidische

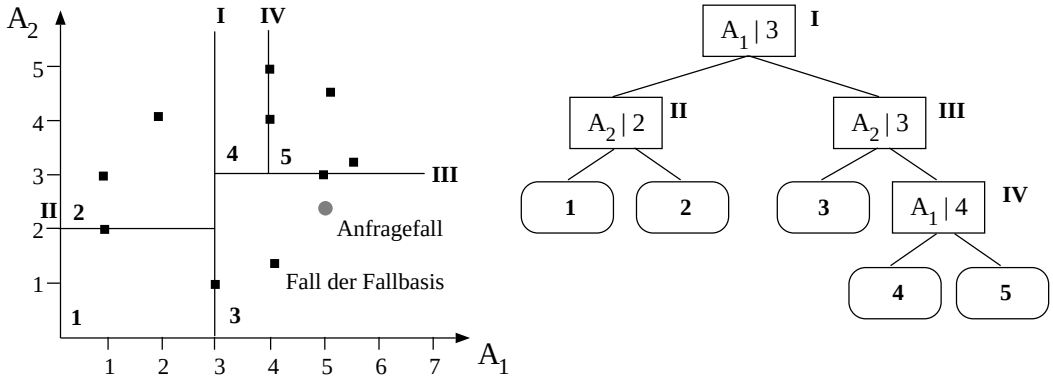


Abbildung 4.10: Beispiel

Distanz:

$$Sim(X,Y) := \sqrt{sim_1(x_1,y_1)^2 sim_2(x_2,y_2)^2}$$

wobei

$$sim_1(x,y) = sim_2(x,y) := |x-y|$$

Die globalen Variablen *Queue* und *Bounds* werden wie in Abschnitt 4.1 initialisiert, der Anfragefall wird in A abgelegt. Anschließend wird die Retrieval-Prozedur gestartet:

1. `Search` wird mit dem Wurzelknoten I des k-d-Baumes aufgerufen; Vergleich des Anfragefalles mit dem Partitionswert 3 im Diskriminatorattribut A_1
2. `Search` wird mit dem rechten Teilbaum III aufgerufen, der die Fälle mit $A_1 > 3$ enthält; Vergleich des Anfragefalles mit dem Partitionswert 3 im Diskriminatorattribut A_2

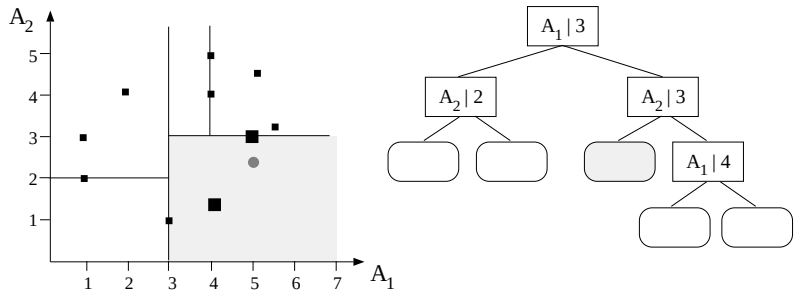


Abbildung 4.11: Eintragen der ersten Fälle

3. `Search` wird mit linken Sohn, dem Blattknoten 3 aufgerufen. Jetzt werden in die Queue die zwei besten Fälle dieses Buckets eingetragen (siehe Abbildung 4.11). Anschließend wird mit einen BWB-Test („BWB1“) überprüft, ob die Suche bereits abgebrochen werden kann. Da die konstruierte Kugel jedoch die Koordinatenschranken des Teilbaumes überschneidet, muß die Suche fortgesetzt werden (Abbildung 4.12).
4. Rückkehr in dem darüberliegenden inneren Knoten III, in dem mit einem BOB-Test („BOB“) geklärt wird, ob der andere Teilbaum auch noch zu durchsuchen ist. Da die Kugel die be-

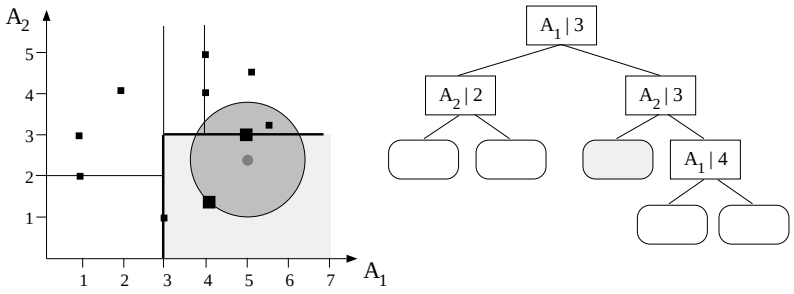


Abbildung 4.12: BWB-Test im Bucket

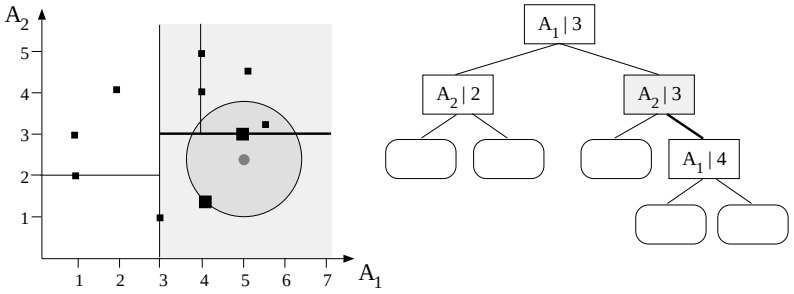


Abbildung 4.13: BOB-Test im inneren Knoten

treffende Koordinatenschranke überschneidet, wird auch noch dieser Zeilbaum untersucht (Abbildung 4.13).

5. `Search` wird mit dem inneren Knoten IV aufgerufen; Vergleich des Anfragefalles mit dem Partitionswert 4 im Diskriminatorattribut A_1

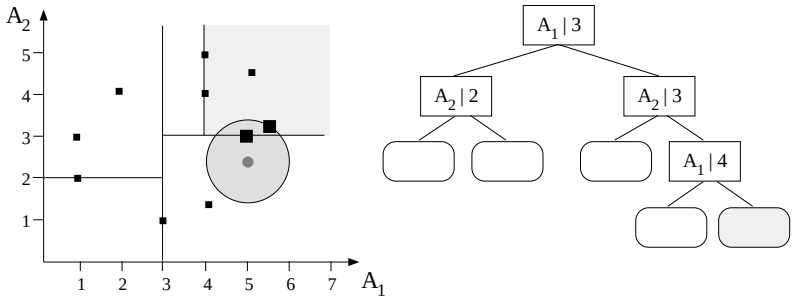


Abbildung 4.14: BWB-Test im Bucket

6. `Search` wird mit rechten Sohn, dem Blattknoten 5 aufgerufen, dort wird auch tatsächlich ein Fall gefunden, der näher am Anfragepunkt liegt als die bisher aufgefundenen Fälle. Der folgende BWB-Test („BWB1“) schlägt natürlich fehl, da der Anfragefall nicht im betreffenden Bucket liegt (Abbildung 4.14).

7. Beim Aufstieg im Baum wird nun im darüberliegenden inneren Knoten IV ein BOB-Test ausgeführt („BOB“). Da die Kugel die betreffende Koordinatenschranke jedoch nicht über-

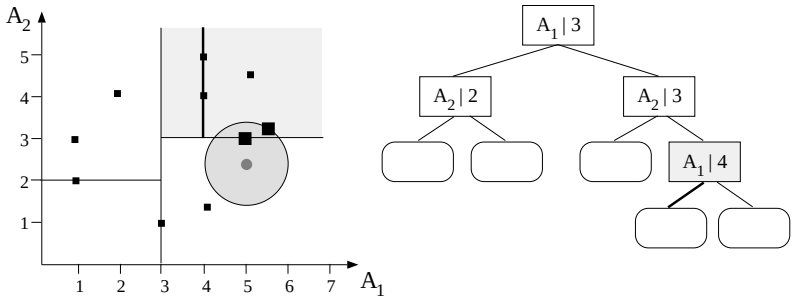


Abbildung 4.15: BOB-Test im inneren Knoten

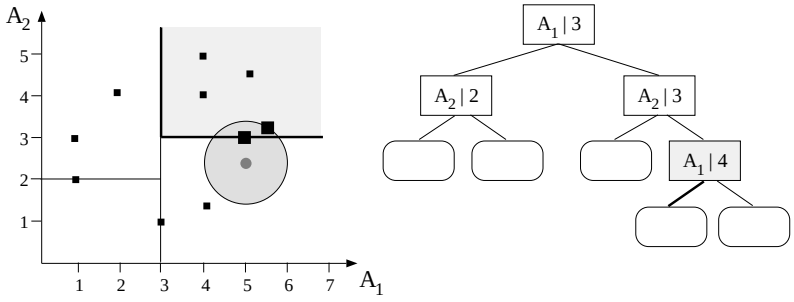


Abbildung 4.16: BWB-Test im inneren Knoten

schneidet, muß der andere Teilbaum des inneren Knotens nicht durchsucht werden (Abbildung 4.15). Der nun angefertigte BWB-Test („BWB2“) läßt jedoch noch keinen Abbruch der Suche zu, da sich der Anfragefall nicht im aktuellen Teilbaum befindet (Abbildung 4.16).

8. Nun befinden wir uns wieder in dem inneren Knoten III, in dem vorher durch den BOB-Test entschieden wurde, daß noch im zweiten Teilbaum gesucht werden muß. Der jetzt durchführte BWB-Test („BWB2“) zeigt an, daß die Suche abgebrochen werden kann, da die Kugel völlig im aktuellen Teilbaum liegt (Abbildung 4.17).

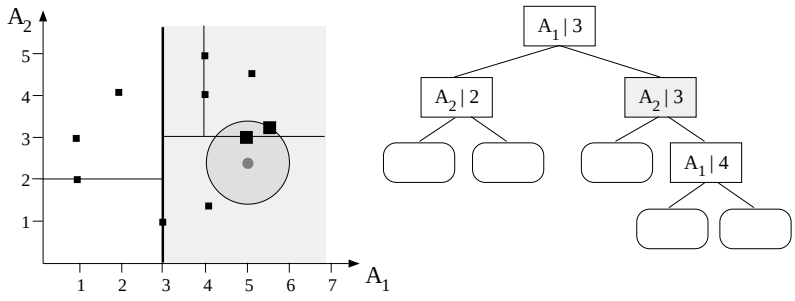


Abbildung 4.17: BWB-Test im inneren Knoten

Dieses Beispiel zeigt, daß nur zwei der fünf Buckets durchsucht werden mußten.

4.2 Optimierungsmöglichkeiten beim Retrieval in k-d-Bäumen

In diesem Kapitel werden nun neue Konzepte beim Retrieval in k-d Bäumen vorgestellt, die im Rahmen dieser Diplomarbeit entwickelt und implementiert wurden.

Eine starke Beschleunigung des Retrievals wurde durch das neue Konzept der *virtuellen Bounds* erreicht, hierdurch führen die Bounds-Tests zu weniger pessimistischen Ergebnissen.

Im Anschluß wird eine Modifikation an der Retrieval-Prozedur vorgenommen, schließlich sollen noch Prozeduren für eine besondere Retrieval-Aufgabe, die *inkrementelle Suche*, erläutert werden.

4.2.1 Virtuelle Bounds

Die wichtigte Komponente beim Retrieval sind die Bounds-Tests, sie entscheiden, ob weitere Teilbäume zu durchsuchen sind und ob die Suche beendet werden kann (siehe Abschnitt 4.1.1).

Minimale virtuelle Bounds

Der Ball-Overlap-Bounds-Test überprüft, ob eine Kugel in einen bestimmten Teil des Datenraumes, repräsentiert durch seine Außengrenzen, den Bounds, überlappt. Der Test gibt eine Überlappung an, falls der die Bounds überschneidet. Der BOB-Test bewirkt innerhalb der Retrieval-Prozedur das noch andere Teilbäume, die die Fälle des überprüften Datenraumes enthalten, durchsucht werden müssen. Werden aber in diesem Teilbaum keine ähnlicheren Fälle aufgefunden, war die Suche umsonst. Im bisherigen Algorithmus trat diese Situation sehr häufig auf. Untersuchungen zeigten, daß der Grund hierfür meist in der spärlich besetzten Teilräumen lag. Die Abbildung 4.18 zeigt eine solche Situation, der BOB-Test zeigt hier eine Überschneidung der Bounds des Datenraums II an, aber es befinden sich keine neuen Fälle des Teilbaumes II innerhalb der Kugel. Ein Durchsuchen dieses Datenraumes erbrächte also keine ähnlicheren Fälle.

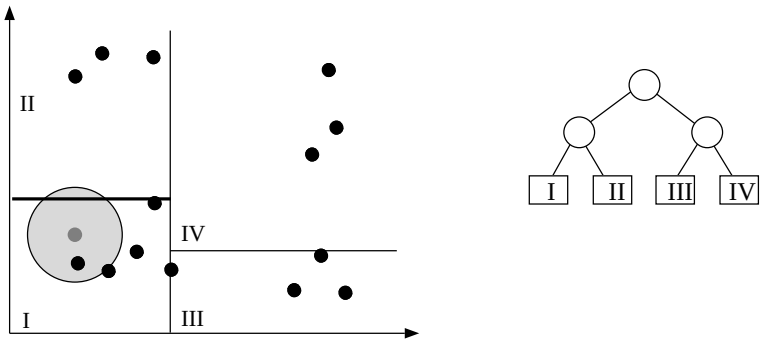


Abbildung 4.18: BOB-Test mit bisherigen Bounds

So wurde versucht, mehr Wissen über die Fälle in die Bounds-Tests mit einzubringen. Der Raum, im dem sich tatsächlich Fälle befinden, sollte genauer beschrieben werden. Es wären Bounds wünschenswert, die sich *dichter* an die tatsächlichen Fälle anpassen würden. Zu diesem Zweck werden alle Fälle innerhalb eines Datenraumes untersucht und jeweils die minimale und die maximale Ausprägung jedes Attributes gespeichert. So entstanden die *minimalen virtuellen Bounds* (siehe Abbildung 4.19). Man erkennt an den weißen Flächen, wieviel des Datenraums nun ausgeklammert werden kann. Ein BOB-Test, der nun mit minimalen virtuellen Bounds arbeitet, erkennt, daß im Bucket II keine besseren Fälle enthalten sind (siehe Abbildung 4.20).

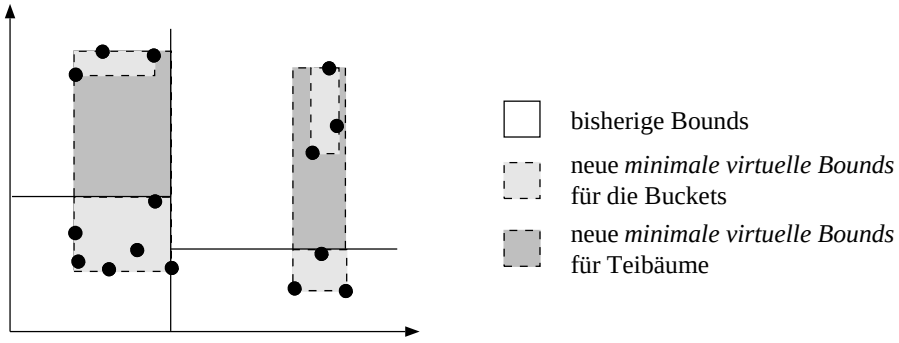


Abbildung 4.19: Minimale virtuelle Bounds

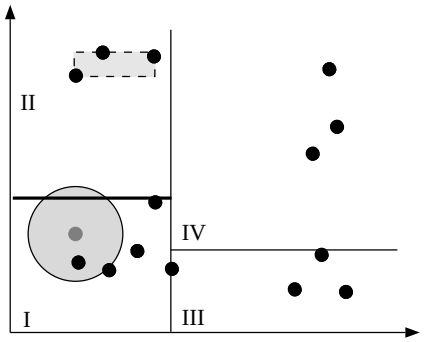


Abbildung 4.20: BOB-Test mit minimalen virtuellen Bounds

Die minimalen virtuellen Bounds eines Baumknotens für die Dimension k sind also definiert als:

$$\begin{aligned} minBounds[k].Upper &:= max(\{Case_i[k]\}) \\ minBounds[k].Lower &:= min(\{Case_i[k]\}) \end{aligned} \tag{4.5}$$

Hierbei sei $\{Case_i[k]\}$ die Menge der Attributsausprägungen im Attribut k aller durch den Baumknoten repräsentierten Fälle $Case_i$.

Die Einführung der neuen Bounds zieht weder eine Änderung der Retrieval-Prozedur noch der Bounds-Tests nach sich, den den Tests werden einfach die neuen virtuellen Bounds als Koordinatenschranken übergeben.

Nun wäre es nicht sinnvoll, während des Retrievals die virtuellen Bounds zu berechnen, da es dort auf Geschwindigkeit ankommt, die gesuchten Fälle sollen so schnell wie möglich zurückgeliefert werden. Also werden die Bounds bereits bei der Generierung berechnet und in den Baumknoten abgespeichert. Dies hat den Vorteil, daß in der Generierungsphase die einzelnen Fälle bereits sortiert nach ihren Attributsausprägungen vorliegen, diese sortierten Listen werden für die Partitionierung benötigt. So muß man nur aus den ensprechenden Listen für jeden Attribut den minimalen und maximalen Wert entnehmen, ein Durchsuchen nach Minimum und Maximum ist nicht mehr notwendig.

Maximale virtuelle Bounds

So wie die minimalen virtuellen Bounds für den Ball-Overlap-Bounds-Test entwickelt wurde, wird nun das Konzept der virtuellen Bounds für den Ball-Within-Bounds-Test erweitert.

Der BWB-Test entscheidet, ob innerhalb eines Datenraumes alle ähnlichsten Fälle liegen, ist dies der Fall, kann die Suche abgebrochen werden. Zu diesem Zweck wird wieder eine Kugel um den Anfragefall aufgebaut, liegt diese Kugel völlig innerhalb des zu überprüfenden Datenraums, sind eine weiteren Fälle zu untersuchen. Abbildung 4.21 verdeutlicht einen solchen Test. Hier

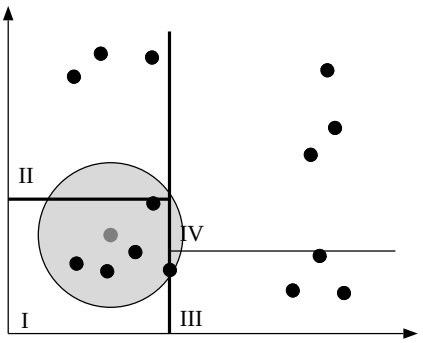


Abbildung 4.21: BWB-Test mit bisherigen Bounds

würde der Test keinen Abbruch der Suche erlauben, obwohl wiederum innerhalb der Kugel keine besseren Fälle der benachbarten Teilräume liegen. Die Kugel überschneidet sich jedoch mit den Koordinatenschranken des Buckets II sowie des Teilbaums, der die Buckets III und IV enthält.

Wiederum bietet sich hier eine Lösung mit virtuellen Bounds an. Beim BOB-Test war es das Ziel, möglichst dichte Grenzen um den Datenraum zu legen, zu dem eine Überschneidung festgestellt werden sollte. Nun ist es für den BWB-Test sinnvoll, die Grenzen des zu überprüfenden Datenraumes so weit wie möglich zu fassen, so daß die Kugel vielleicht doch in die Grenzen hineinpaßt.

Zu diesem Zweck werden die *maximalen virtuellen Bounds* eingeführt, sie sind in Abbildung 4.22 dargestellt.

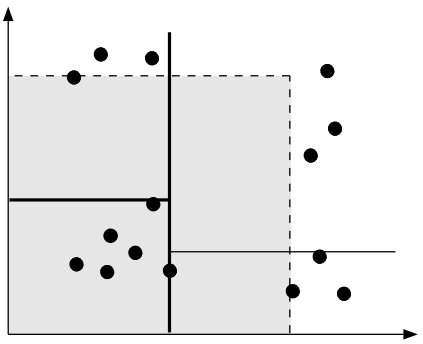


Abbildung 4.22: Maximale virtuelle Bounds

Innerhalb dieser maximalen virtuellen Bounds ist es garantiert, daß sich dort nur Fälle des Datenraumes liegen, zu dem die maximalen virtuellen Bounds gehören. Es ist also ausgeschlossen,

innerhalb der Bounds bessere Fälle zu finden, die Suche kann beendet werden (Abbildung 4.23).

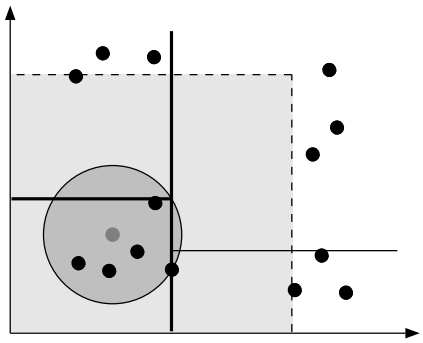


Abbildung 4.23: BWB-Test mit maximalen virtuellen Bounds

Die Berechnung der maximalen virtuellen Bounds ist etwas aufwendiger als die der minimalen virtuellen Bounds. Hierzu müssen nicht die Fälle des Datenraums untersucht werden, für den die maximalen virtuellen Bounds zu berechnen sind, sondern alle umliegenden Datenräume. Auch hier bietet sich wieder die Berechnung bei der Generierung des k-d-Baumes an.

Bei einem Partitionierungsvorgang während der Baumgenerierung werden zunächst die minimalen virtuellen Bounds wie in Abschnitt 4.2.1 dargestellt, berechnet. Anschließend können in der Dimension des Diskriminatorattributes die maximalen virtuellen Bounds für die Söhne eingetragen werden, in den anderen Dimensionen werden die maximalen virtuellen Bounds des Elternknotens übernommen.

Die maximalen virtuellen Bounds der Wurzel werden initialisiert:

$$\begin{aligned} \text{Wurzelknoten.maxBounds[k].Upper} &:= \infty \\ \text{Wurzelknoten.maxBounds[k].Lower} &:= -\infty \end{aligned} \tag{4.6}$$

Die Berechnung der maximalen virtuellen Bounds des linken Sohns *leftSon*, der alle Fälle enthält, deren Attributsausprägung im Diskriminatorattribut des aktuellen Knotens *parent* kleiner als der Partitionswert sind:

$$\begin{aligned} \text{leftSon.maxBounds[k].Upper} &:= \begin{cases} \text{rightSon.minBounds[k].Lower} & \text{falls k Disk.-Attr.} \\ \text{parent.maxBounds[k].Upper} & \text{sonst} \end{cases} \\ \text{leftSon.maxBounds[k].Lower} &:= \text{parent.maxBounds.Lower} \end{aligned} \tag{4.7}$$

und des rechten Sohns *rightSon*:

$$\begin{aligned} \text{rightSon.maxBounds[k].Lower} &:= \begin{cases} \text{leftSon.minBounds[k].Upper} & \text{falls k Disk.-Attr.} \\ \text{parent.maxBounds[k].Lower} & \text{sonst} \end{cases} \\ \text{rightSon.maxBounds[k].Upper} &:= \text{parent.maxBounds.Upper} \end{aligned} \tag{4.8}$$

Die maximalen virtuellen Bounds ergeben sich also immer aus der beim Splitting gegenüberliegenden minimalen virtuellen Bounds. Dieser Zusammenhang der maximalen und minimalen virtuellen Bounds wird in Abbildung 4.24 verdeutlicht.

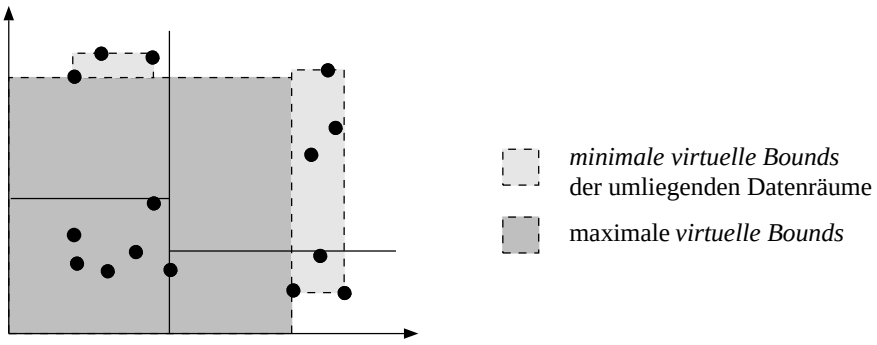


Abbildung 4.24: Zusammenhang maximale & minimale virtuelle Bounds

Performance-Vergleich und Bewertung

Um eine Bewertung des Konzeptes der virtuellen Bounds zu ermöglichen, wurde ein k-d-Baum mit unserer PKW-Datenbank aus Abschnitt 3.2.6 aufgebaut. Bei der Generierung wurden bereits

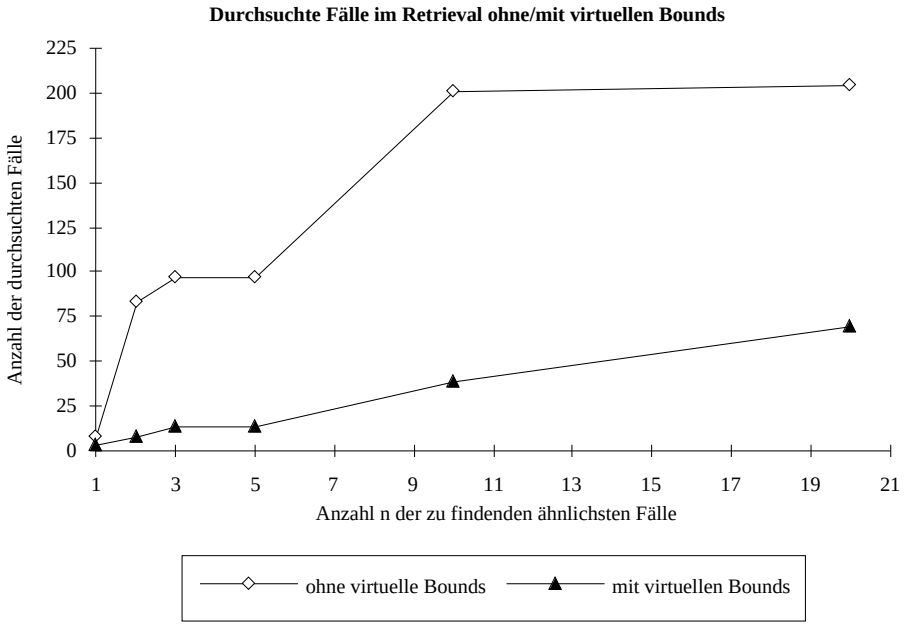


Abbildung 4.25: Durchsuchte Fälle

die maximalen und minimalen virtuellen Bounds ermittelt und im Baum mit abgespeichert. Da die Ermittlung der virtuellen Bounds bei der Generierung recht einfach ist, konnte hier keine wesentliche Verlangsamung des Baumaufbaus festgestellt werden. Auf diesem Baum wurde mit der bisherigen Retrievalprozedur, die noch mit konventionellen Bounds arbeitet, und der Retrievalprozedur, die sich der virtuellen Bounds bedient, jeweils 1, 2, 5, 10 und 20 ähnlichste Fälle gesucht. Die Grafik 4.25 zeigt die Anzahl der durchsuchten Fälle für diese beiden Varianten. Die Variante mit virtuellen Bounds muß deutlich weniger Fälle durchsuchen als bisher, die durchsuchten Fälle werden auf einen Bruchteil der bisher benötigten Vergleiche beschränkt.

Auch bei den Bounds-Tests kann eine große Einsparung beobachtet werden (Grafik 4.26). Da

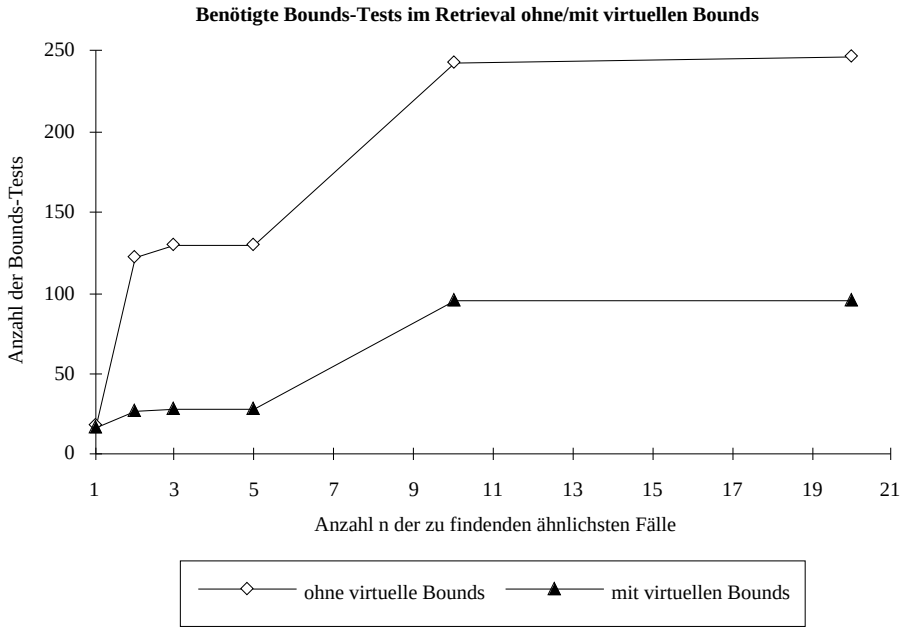


Abbildung 4.26: Bounds-Tests

bereits in wenigen Buckets die benötigten Fälle gefunden wurden, ist auch die Zahl der Bounds-Tests geringer, da nur noch kleine Teile des k-d-Baums durchsucht werden müssen.

Die Einführung der virtuellen Bounds hat zu einem großen Performance-Anstieg geführt, dabei ist der zu treibende Aufwand beim Retrieval nicht angewachsen, da hier nur andere Bounds, die bereits bei der Generierung einfach erzeugt werden können, an die Bounds-Tests übergeben werden. Die eingesparten Retrievalzeiten werden in der Grafik 4.27 deutlich.

Zum Vergleich ist hier die benötigte Zeit zum linearen Durchsuchen der Fallbasis angegeben. Bei der alten Variante ohne virtuelle Bounds war bereits ab 2 zu suchenden Fällen die lineare Variante schneller als das Retrieval im Baum. Diese Ergebnis hat sich durch die Verwendung der virtuellen Bounds sehr verbessert, die Anforderung von 10 ähnlichsten Fällen ist noch schneller als die lineare Suche.

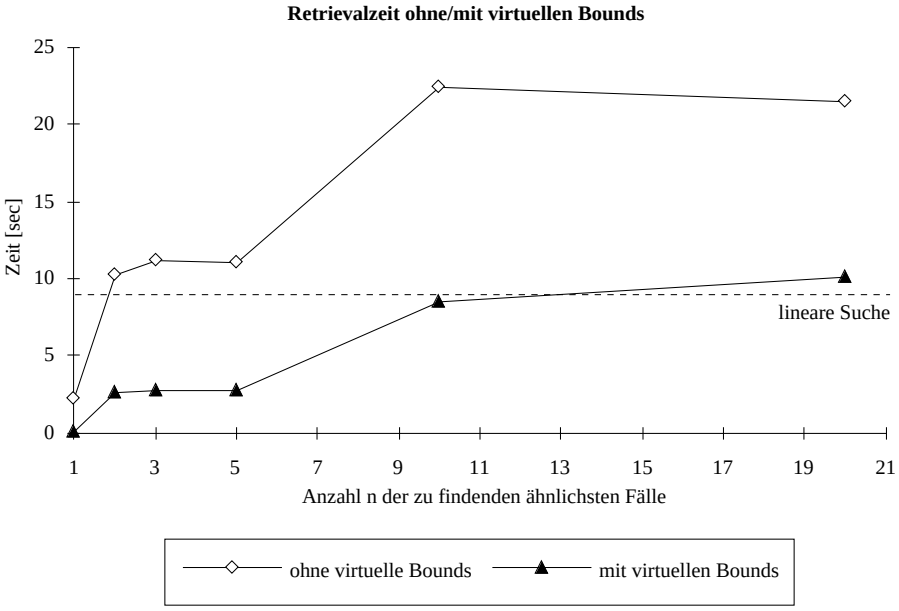


Abbildung 4.27: Retrieval-Zeiten

4.2.2 Die Retrieval-Prozedur

Die Einführung der virtuellen Bounds hat zur Folge, daß die Bounds-Tests deutlich bessere Entscheidungen erbrachten. Dennoch war bei Nutzung der bisherigen Retrieval-Prozedur zu beobachten, daß noch viele Buckets durchsucht wurden, ohne bessere Fälle zu finden.

Der Grund hierzu liegt in der Struktur der Retrieval-Prozedur: Nachdem beim Wiederaufstieg nach Duchsuchen des ersten Buckets ein Ball-Overlap-Bounds-Test angibt, daß noch ein weiterer Teilbaum durchsucht werden muß, wird in diesem Teilbaum direkt bis in den Bucket gelaufen. Ein typischen Suchablauf inerhalb eines k-d-Baumes zeigt Abbildung 4.28. Zunächst wird bis in

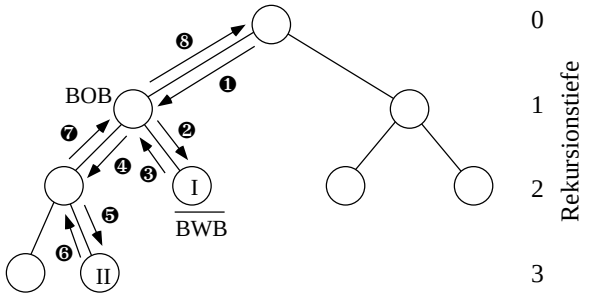


Abbildung 4.28: Typischer Retrieval-Ablauf mit bisheriger Prozedur

den Bucket I hinabgestiegen, der BWB-Test schlägt dann fehl, es muß also noch weiter gesucht werden. So wird wieder in den darüberliegenden Elternknoten aufgestiegen, hier wird mit einem BOB-Test überprüft, ob der andere Teilbaum auch untersucht werden soll. Der BOB-Test verlangt dann diese Untersuchung, der andere Teilbaum wird mit Search durchsucht. Nun wird aber direkt wieder bis in Bucket II hinabgelaufen, obwohl dort nichts gefunden wird.

Das Beispiel zeigt, daß hier ein zu hoher Suchaufwand betrieben wurde. Deshalb wird jetzt

nicht mehr direkt bis zu einem Bucket gelaufen, sondern es wird in jeder Baumebene überprüft, ob ein weiters Hinabsteigen noch Sinn macht. Dies ist möglich, da die Bounds-Tests immer „besser“ werden, wenn man sich tiefer im Baum befindet, denn dort sind die Bounds enger definiert (siehe Abbildung 4.29).

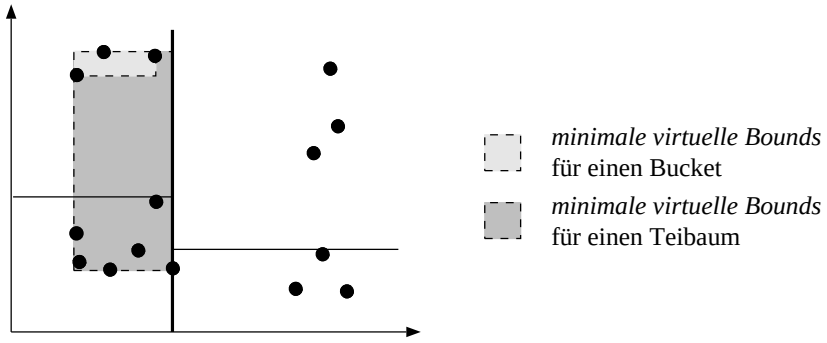


Abbildung 4.29: virtuelle Bounds

Deshalb wird ein weiterer BOB-Test vor dem Absteigen in den Baum eingefügt:

```
PROCEDURE Search (treeNode)
VAR Disc,Part,temp.
BEGIN
   IF (TerminalNode(treeNode)) THEN
      [Aktualisiere queue mit dem Inhalt von treeNode] .
      IF BallWithinBounds THEN [Fertig] ELSE RETURN.
   END.
   Disc:=treeNode.discriminator.
   Part:=treeNode.partitionValue.
   IF A[Disc]<=Part THEN
      temp:=Bounds.Upper[Disc].
      Bounds.Upper[Disc]:=Part.
      IF BoundsOverlapBall THEN        neuer BOB-Test
         Search(treeNode.lowerSon).
      END.
      Bounds.Upper[Disc]:=temp.
   END
   ELSE
      temp:=Bounds.Lower[Disc].
      Bounds.Lower[Disc]:=Part.
      IF BoundsOverlapBall THEN        neuer BOB-Test
         Search(treeNode.upperSon).
      END.
      Bounds.Lower[Disc]:=temp.
   END.

   IF A[Disc]<=Part THEN
      temp:=Bounds.Lower[Disc].
      Bounds.Lower[Disc]:=Part.
```

```
      IF BoundsOverlapBall THEN Search(treeNode.upperSon).
      Bounds.Lower[Disc]:=temp.
      END
      ELSE
      temp:=Bounds.Upper[Disc].
      Bounds.Upper[Disc]:=Part.
      IF BoundsOverlapBall THEN Search(treeNode.lowerSon).
      Bounds.Upper[Disc]:=temp.
      END.

   IF BallWithinBounds THEN [Fertig] ELSE RETURN.
END.
```

Mit dieser Prozedur arbeitete das Retrieval jetzt effizienter (siehe Abbildung 4.30).

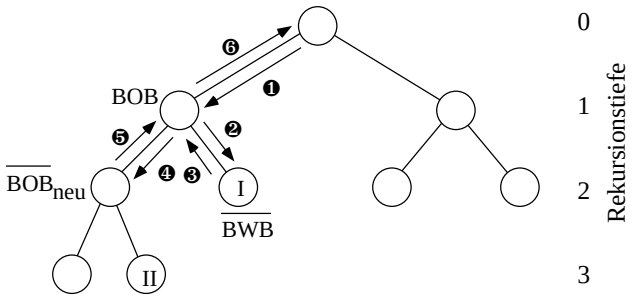


Abbildung 4.30: Retrieval-Ablauf mit neuer Prozedur

Der BOB-Test beim Wiederaufstieg in den Eltenknoten nach Durchsuchen des Buckets I verlangt zwar noch wie zuvor einen Einstieg in den anderen Teilbaum, aber in der nächsten Baumebene wird erkannt, daß ein weiteres Absteigen in den Bucket II nicht sinnvoll ist. So wurden die Ähnlichkeitsberechnungen im Bucket gespart.

Performance-Vergleich und Bewertung

Nun soll anhand von Messungen die Qualität der neuen Retrieval-Prozedur bewertet werden. Die beiden Varianten der Retrieval-Prozedur, alt und neu mit zusätzlichen Bounds-Tests, arbeiten auf einem k-d-Baum, der wiederum aus unserer PKW-Datenbank (Beschreibung Abschnitt 3.2.6) generiert wurde.

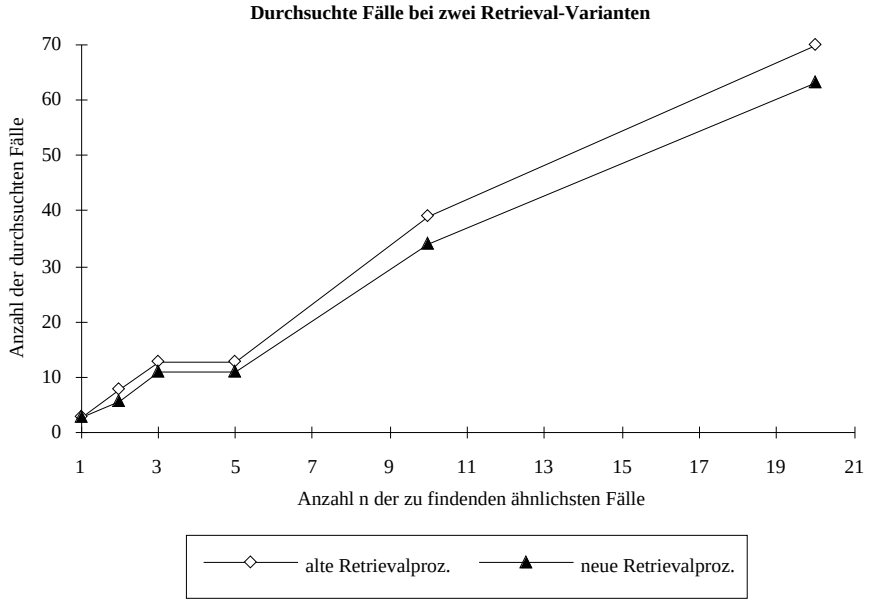


Abbildung 4.31: Durchsuchte Fälle

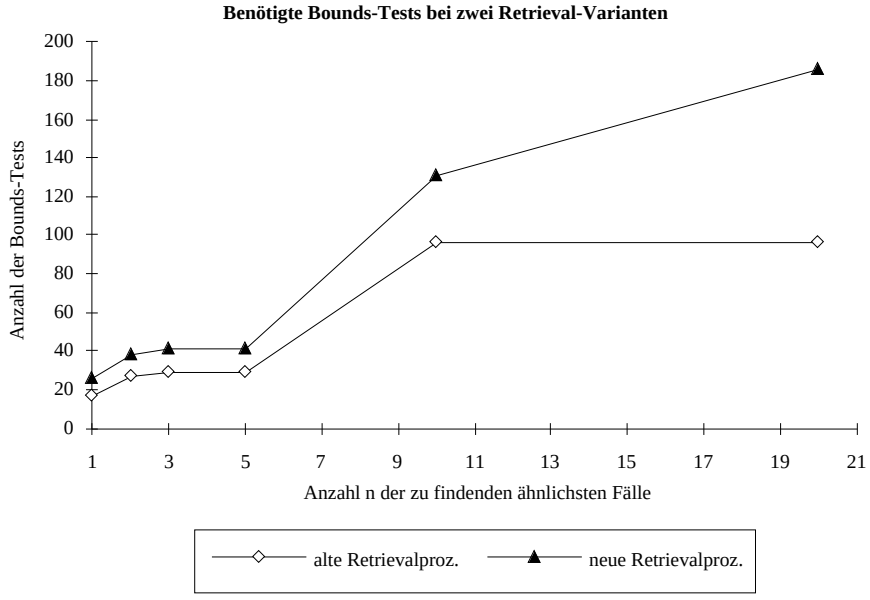


Abbildung 4.32: Bounds-Tests

Die Grafik 4.31 zeigt die Anzahl der durchsuchten Fälle in Abhängigkeit von der Anzahl zu

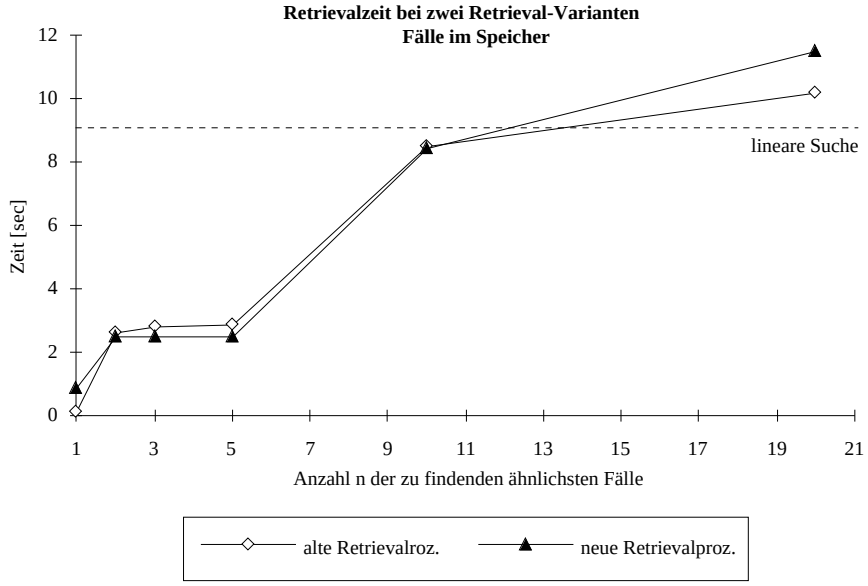


Abbildung 4.33: Retrievalzeiten

suchenden ähnlichsten Fälle. Hier muß die neue Retrieval-Variante weniger Fälle durchsuchen, da einige Buckets aufgrund des vorgeschalteten BOB-Tests nicht aufgesucht wurden.

Die in der Grafik 4.32 dargestellten benötigten Bounds-Tests verdeutlichen jedoch den Preis, den man für die weiger durchsuchten Buckets zahlen muß: Die Anzahl der Bounds-Tests, verursacht durch die zusätzlich eingesetzten BOB-Tests, steigt deutlich an.

Da mit jedem Bounds-Test im Vergleich mit dem Durchsuchen eines Falles ein ähnlichen Aufwand zu treiben ist, lohnt sich der Einsatz der neuen Retrieval-Prozedur in den meisten Fällen nicht. Dies gilt aber nur, wenn die Fälle im Hauptspeicher liegen. Ist es jedoch notwendig, die Fallbasis auf einem langsameren Sekundärspeicher zu halten, kann sich die geringe Einsparung der durchsuchten Fälle jedoch gegenüber der gestiegenen Anzahl der Bounds-Tests bezahlt machen, da der k-d-Baum und der Anfragefall im Hauptspeicher liegen und somit schnellere Tests zulassen.

Genau dieses Verhalten konnte auch bei Zeitmessungen beobachtet werden, die neue Variante ist bei Speicherung der Fälle im Hauptspeicher (siehe Grafik 4.33) noch unterlegen, liegen die Fälle jedoch auf Platte, ermöglicht sie ein schnelleres Retrieval (Grafik 4.34).

Da beide Retrieval-Varianten jedoch nebeneinander bestehen können und mit den identischen k-d-Bäumen arbeiten, kann man beide Varianten im System anbieten.

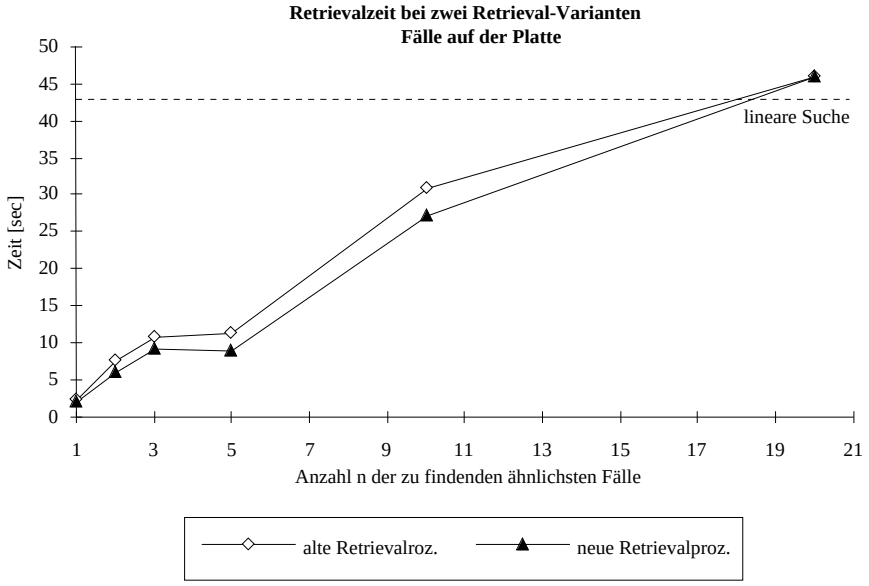


Abbildung 4.34: Retrievalzeiten

4.3 Inkrementelles Retrieval

Beim Einsatz des Systems hat sich herausgestellt, daß der Anfragefall oft nicht genau spezifiziert ist. Ein typisches Vorgehen ist es, zunächst nur einige Attribute des Anfragefalls anzugeben, die anderen Attribute erhalten den Wert „unbekannt“. Ist das Ergebnis noch nicht befriedigend, werden weitere Attribute spezifiziert und das Retrieval erneut gestartet. Diese Vorgehensweise bezeichnen wir als *inkrementelles Retrieval*.

Ein typisches Einsatzgebiet wäre das Erstellen einer medizinischen Diagnose. Zunächst werden nur preiswerte und schnell durchzuführende Untersuchungen wie Messung der Körpertemperatur, des Blutdrucks u.s.w. durchgeführt. Wären die vom System gefundenen ähnlichsten Fälle nun aber nicht zufriedenstellend, würde man weitere Untersuchungen anordnen und mit deren Ergebnissen erneut einen besser spezifizierten Anfragefall erhalten.

Nun wäre es wünschenswert, die alten Ergebnisse beim erneuten Retrieval weiter verwenden zu können. Mit dieser Aufgabe beschäftigen sich die jetzt vorgestellten Verfahren.

4.3.1 Verwendung der alten Ergebniss-Queue

Grundgedanke dieses Algorithmus‘ ist es, daß bei einer weiter spezifizierten Anfrage die Qualität der vom Retrieval zurückgelieferten Fälle immer besser wird oder gleich bleibt, eine Verschlechterung ist ausgeschlossen.

Dies soll in einem Beispiel verdeutlicht werden, es soll wieder unsere Standard-Ähnlichkeitsfunktion zur Anwendung kommen. Die erste Anfrage mit (A_1=a,A_2=?,A_3=?) liefere den Fall (A_1=a,A_2=b,A_3=c), die Ähnlichkeit zum Anfragefall ist also

$$\begin{aligned} Sim(\text{Anfragefall,gefundener Fall}) &= \frac{sim(a,a)+sim(?,b)+sim(?,c)}{3} \\ &= \frac{1+0+0}{3} \\ &= \frac{1}{3} \end{aligned}$$

Bei einer verfeinerten Anfrage (A_1=a,A_2=d,A_3=c) wäre die Qualität des gefundenen Falls mindestens auch $\frac{1}{3}$, da die unbestimmten Attribute in der vorangehenden Anfrage eine *worst-case*-Annahme waren. Nun würde der Fall (A_1=a,A_2=b,A_3=c) nochmals gefunden werden, allerdings wäre die Ähnlichkeit zum Anfragefall jetzt $\frac{2}{3}$.

Das Beispiel zeigt, daß es sinnvoll ist, die bereits gefundenen Fälle in der verfeinerten Anfrage wieder zu verwenden. Zu diesem Zweck können allerdings die Ähnlichkeiten der Fälle nicht übernommen werden, sie müssen bezogen auf den veränderten Anfragefall neu berechnet werden. Gerade diese Ähnlichkeiten fließen in das Retrieval wesentlich ein, die schlechteste Ähnlichkeit der alten Ergebnis-Queue bestimmt gerade den Kugelradius bei den Bounds-Tests (siehe Abschnitt 4.1.1). Jetzt wird die Suche erneut gestartet, durch die von Beginn belegte Ergebnis-Queue werden jetzt nur noch Teilbäume und Buckets durchsucht, die bessere Fälle als die der schlechteste Fall in der Ergebnis-Queue enthalten können. Es wird also folgendermaßen vorgegangen:

1. Belegen des globalen Arrays A mit dem Anfragefall.
2. Initialisieren der globalen Variablen, die die Ergebnisfälle verwalten:
   - $Queue[1 \ldots m].Case := QueueALT[1 \ldots m].Case$
   - $Queue[1 \ldots m].Sim := Sim(QueueALT[1 \ldots m].Case, A)$
3. Absteigendes Sortieren von $Queue[1 \ldots m]$ nach $Queue[1 \ldots m].Sim$. Dies ist notwendig, da sich die Reihenfolge der Ergebnissfälle durch die neue Ähnlichkeitsberechnung geändert haben kann.
4. Aufruf von Search(root).

4.3.2 PATDEX-Algorithmus für symbolische Attribute

Eine zweite Variante zur Unterstützung inkrementellen Retrievals bedient sich einer Zugriffsstruktur aus dem System PATDEX („Pattern Directed Expert System“), ein fallbasiertes Expertensystem zur Fehlerdiagnose in CNC-Bearbeitungsmaschinen ([Wes93]).

Hier wird eine Verwaltungsstruktur verwendet, die einen direkten Zugriff auf Fälle ermöglicht, die in einem symbolischen Attribut eine bestimmte Ausprägung haben. Hierzu wird für jeden symbolische Attribut eine Liste von dessen Atributsausprägungen geführt, diese Attributsausprägungen sind wiederum mit den entsprechenden Fällen verzeigert (siehe Abbildung 4.35).

Hiermit ist es möglich, direkt auf alle Fälle zuzugreifen, die beispielsweise im Attribut *Augen* die Attributsausprägung *braun* haben.

Genau diese Eigenschaft ist auch für unser inkrementelles Retrieval interessant. Bei einer genaueren Spezifizierung eines Attributes kann man direkt alle Fälle untersuchen, die in dem neu spezifizierten Attribut die korrekte Attributsausprägung haben.

Das Vorgehen wird wieder an einem Beispiel gezeigt:
Die erste Anfrage mit ($Größe$ =?, $Haar$ =?, $Augen = braun$) liefere die Personen D, G und H, deren Ähnlicheit zum Anfragefall $\frac{1}{3}$ ist. Bei einer verfeinerten Anfrage ($Größe = klein, Haar =$?, $Augen = braun$) sucht man über die neue Zugriffsstruktur alle Fälle auf, die im Attribut *Größe* die Ausprägung *klein* haben, also die Personen A, E und H. Nun werden für alle in Betracht kommenden Fälle, d.h. die bei der alten Anfrage aufgefundenen und die neuen Fälle, die Ähnlichkeit zum Anfragefall berechnet, anschließend kann die Ergebnisqueue ausgegeben werden.

4.3.3 Bewertung

Die inkrementelle Suche mit der PATDEX-Zugriffsstruktur erwies sich zwar bei einigen Anfragen als sehr schnell, weist aber Schwächen auf:

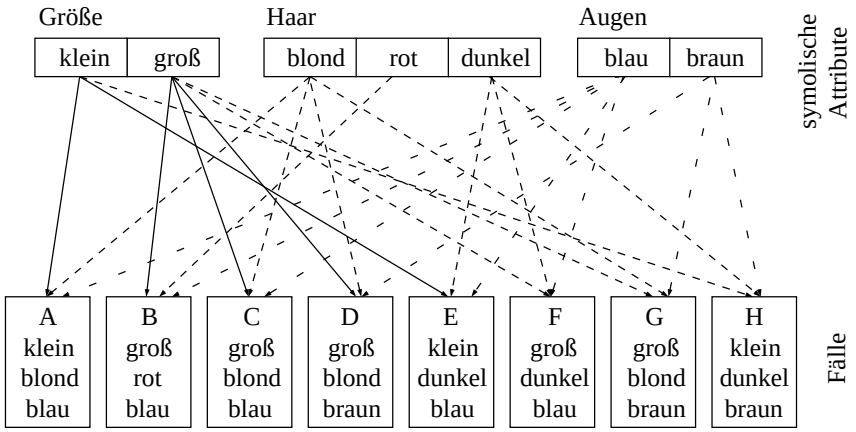


Abbildung 4.35: Zugriffsstruktur PATDEX

- Es werden nur symbolische Atribute unterstützt.
- Das Vorgehen funktioniert nur mit der Standard-Ähnlichkeitsfunktion

$$sim(x, y) := \begin{cases} 0 & \text{falls } x \neq y \\ 1 & \text{falls } x = y \end{cases}$$

  Eine andere Ähnlichkeitsfunktion, die Werte über Null für ungleiche Symbole liefert, arbeitet nicht mit dem Verfahren, da sonst bei der Suche keine Einschränkung auf nur eine Attributsauspägung vorgenommen werden kann.

- Werden mehrere Attribute neu spezifiziert, muß man bei die Vereinigungsmenge aller Fälle durchsuchen, die in nur einem der neu spezifizierten Attribute mit dem neuen Anfragefall übereinstimmen. Hier wird schnell ein Durchsuchen eines Großteils der Fallbasis notwendig.

Abschließend ist zu bemerken, daß für spezielle Anwendungen (symbolische Attribute, viele verschiedene Ausprägungen pro Attribut) die Anwendung der PATDEX-Zugriffsstruktur sinnvoll sein kann. In einem allgemein gehaltenen System mit vielen Anwendungsmöglicheiten sollte die Variante des inkrementellen Retrievals „*Verwendung der alten Ergebniss-Queue*“ unbedingt vorgezogen werden. Diese Vorgehensweise beschneidet in keiner Weise die Funktionalität des Gesamtsystems, sie arbeitet mit allen Attributsskalierungen und allen Ähnlichkeitsfunktionen.

Kapitel 5

Zusammenfassung

In dieser Diplomarbeit wurde ein System zum ähnlichkeitsbasierten Retrieval von Fällen entwickelt. Dieses System übernimmt die Verwaltung der Fälle unter Verwendung der Datenstruktur des k-d-Baumes, hierbei werden die k-d-Bäume so aufgebaut, daß sie in optimaler Weise die Best-Match- bzw. Nearest-Neighbour-Suche unterstützt.

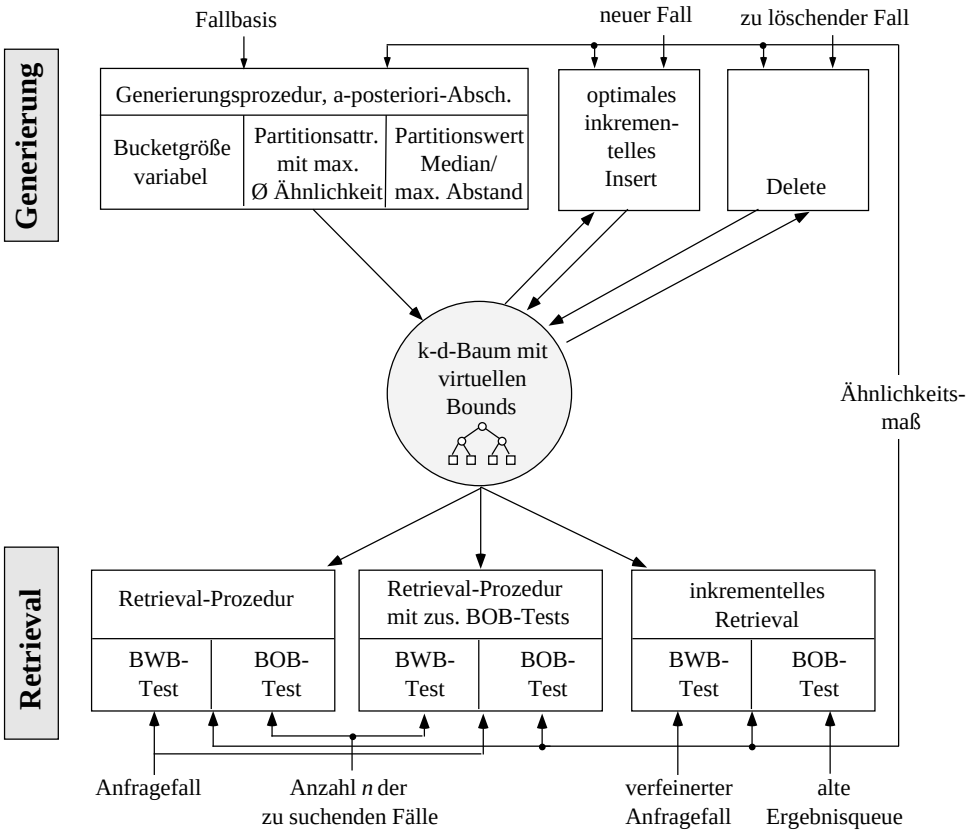


Abbildung 5.1: neue Systemstruktur

Die verschiedenen Parameter, mit denen man auf den k-d-Baum Einfluß nehmen kann (Partitionswert, Diskriminatorattribut und Bucketgröße) wurden untersucht und verschiedene Konzepte zur Bestimmung der optimalen Parametereinstellungen entwickelt und implementiert.

Es stellte sich heraus, daß das neue Auswahlkonzept für das Diskriminatorattribut *maximale*

Ähnlichkeit in den Partitionen und die damit verbundene a-posteriori-Abschätzung in der Generierungsprozedur deutlich bessere Ergebnisse als das bisher bestehende System liefert. Diese Vorgehensweise nähert sich dem von *Conceptual Clustering*-Systemen an, der entstehende k-d-Baum repräsentiert schon eine gewisse Klassifizierung der Fälle.

Die veränderte Bestimmung des Partitionswertes (größter Abstand statt Median) unterstützt diesen Klassifizierungsvorgang, da zusammenhängende Attributsausprägungen nicht mehr zerteilt werden.

Für die Einstellung des Parameters „Bucketgröße“ wurde ein neues Konzept vorgestellt, welches eine dynamische Anpassung der Bucketgröße an die einzuspeichernden Fälle erlaubt.

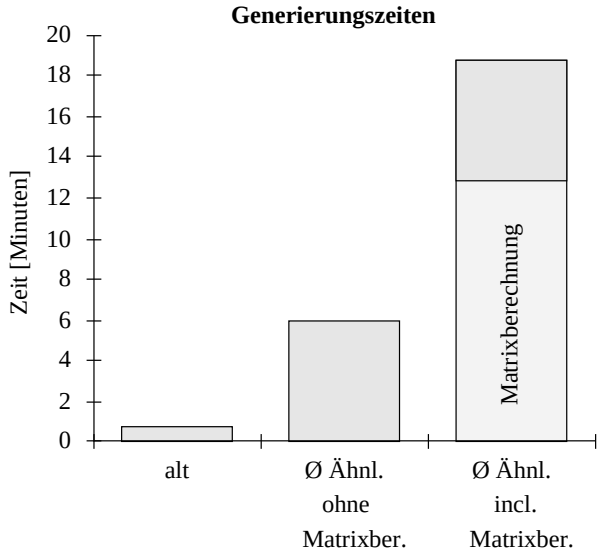


Abbildung 5.2: Generierungszeiten

Da die neue Baumgenerierung mit einem höheren Aufwand verbunden ist (siehe Grafik 5.2), wurde ein *inkrementelles Insert* vorgestellt, mit dem ein effizientes Einfügen von Fällen ermöglicht wird, ohne das der k-d-Baum neu generiert werden muß um seine Optimalität zu garantieren.

Auf der Retrievalseite wurde das neues Konzept der *virtuellen Bounds* entwickelt, die die Suche wesentlich beschleunigen. In gewisser Weise stellt dieses Konzept auch eine Verfeinerung der Zugriffstruktur k-d-Baum dar, da in den Baumknoten nun auch die virtuellen Bounds gespeichert werden und somit eine genauere Aussage über die durch den Baumknoten repräsentierten Fälle erlaub wird.

Schließlich wurden zwei Retrievalvarianten für einen häufig benötigten Spezialfall des Retrievals, das *inkrementelle Retrieval*, vorgestellt. Hierbei werden in besonderer Weise Anfragen unterstützt, die nach und nach verfeinert, d. h. genauer spezifiziert werden. So konnte erreicht werden, daß das erworbene Wissen von den vorherigen Retrievalanfragen genutzt werden kann.

Die beobachteten Gesamtleistungssteigerungen der durch die in dieser Diplomarbeit entwickelten Konzepte liegen bei dem Faktor 5–10. Es ist natürlich nicht möglich, hier genaue Werte anzugeben, da die Ergebnisse sehr stark von der Fallbasis, dem Anfragefall und dem verwendeten Ähnlichkeitsmaß abhängen.

Die Grafik 5.3 zeigt die Anzahl der durchsuchten Fälle bei Suchanfragen. Im Vergleich zum alten System muß nur noch ein Bruchteil der Fälle in der Fallbasis durchsucht werden.

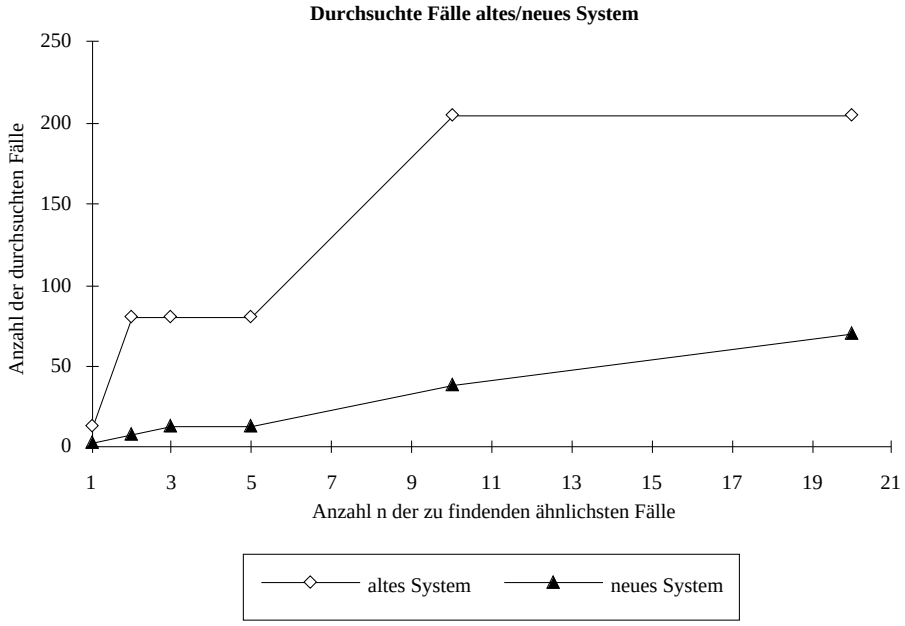


Abbildung 5.3: Durchsuchte Fälle

Auch die Anzahl der Bounds-Tests, ein Maß für den entstehenden Suchaufwand im k-d-Baum, konnte entscheidend verringert werden (Grafik 5.4).

Insgesamt wurde eine deutliche Beschleunigung der Suchzeiten erreicht, im Besonderen beim Vergleich zur linearen Suche wird die Leistungssteigerung klar. Bei den für die Grafik 5.5 vorgenommenen Messungen zeigte sich, daß bei einer Suche nach 10 ähnlichsten Fällen (5% der gesamten Fallbasis) die Suche im k-d-Baum noch schneller als die lineare Suche ist. Hierbei lagen die Fälle im Hauptspeicher. Bei der Verwaltung der Fälle auf der Festplatte sind die Ergebnisse noch besser, selbst bei der Suche nach 20 ähnlichsten Fällen (10% der gesamten Fallbasis) entspricht der Zeitaufwand noch dem der linearen Suche (Grafik 5.6).

Zusammenfassend lässt sich festhalten, daß der k-d-Baum in Verbindung mit geeigneten Generierungs- und Retrieval-Algorithmen eine geeignete und effiziente Unterstützung des ähnlichkeitsbasierten Fallretrievals ermöglicht. Die Anzahl der zu durchsuchenden Fälle kann stark eingeschränkt werden, was sich im Besonderen bei sehr großen Fallbasen bezahlt macht.

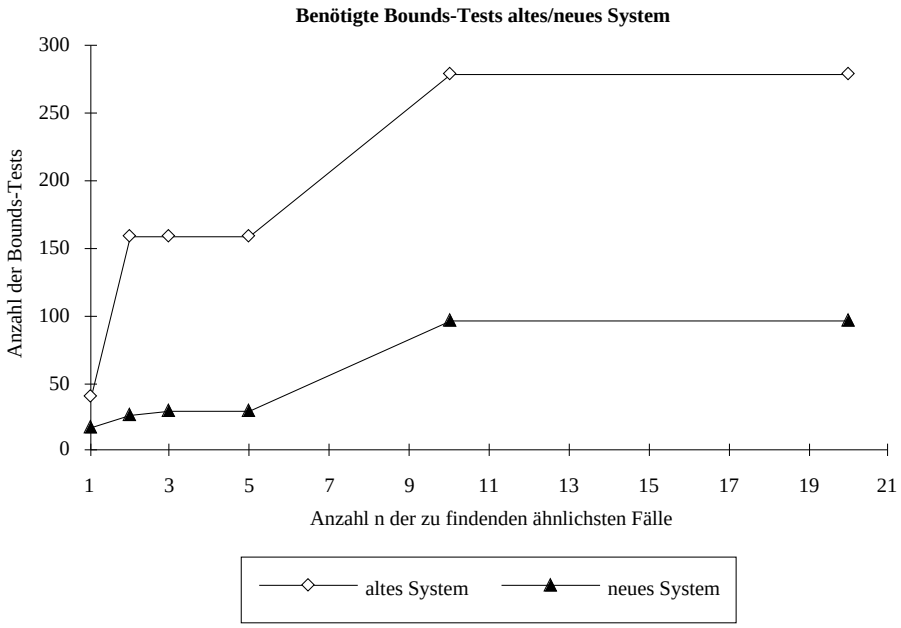


Abbildung 5.4: Bounds-Tests

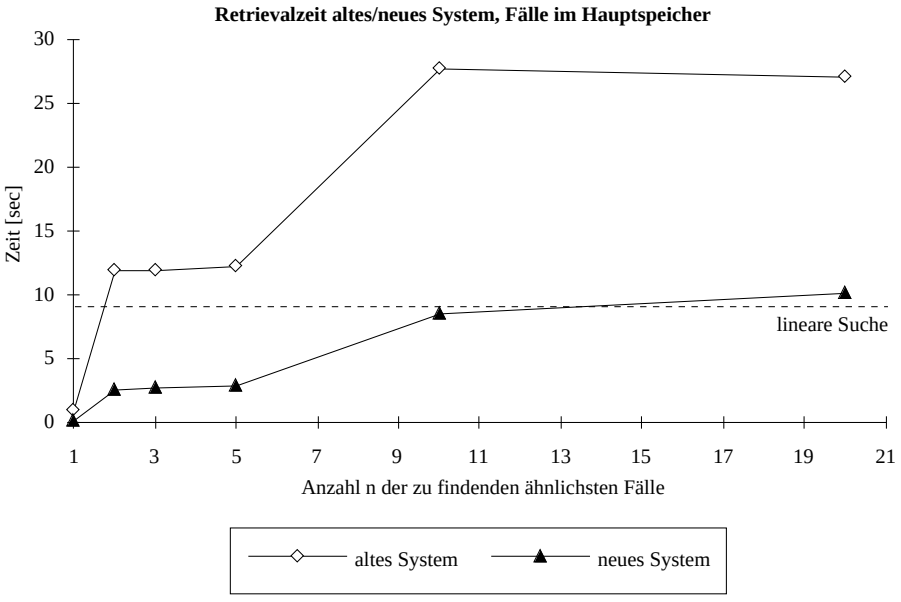


Abbildung 5.5: Retrievalzeiten

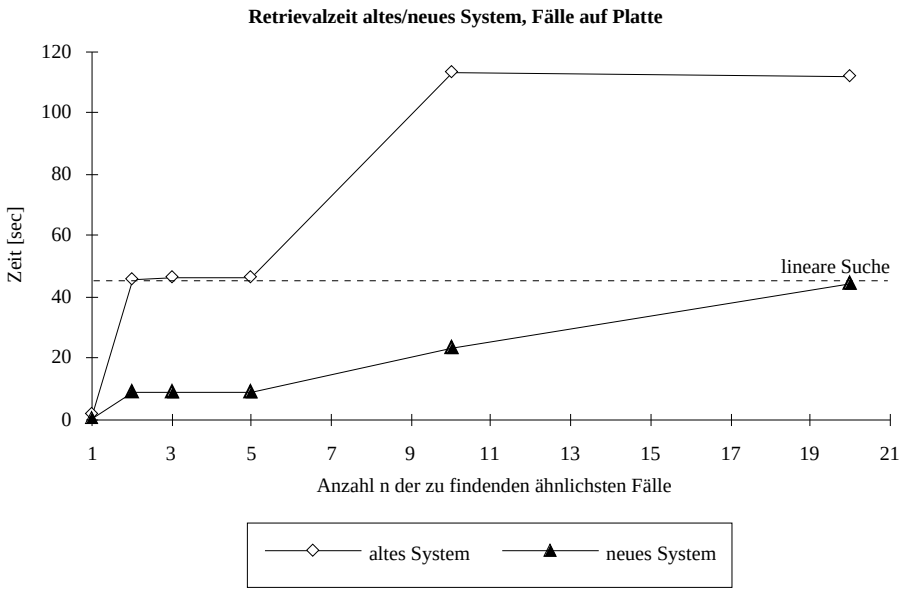


Abbildung 5.6: Retrievalzeiten

Anhang A

Die PKW-Datenbank

Für die Performance-Messungen wurde von einem Rechner der *UCI Repository Of Machine Learning Databases and Domain Theories* in den USA einige Datenbanken über das Internet kopiert, um so mit echten Daten arbeiten zu können. Auf diesem Rechner befinden sich mehr als 20 Megabyte Datenmaterial, daß von jedem genutzt werden kann. Man erreicht diesen Rechner über das Internet an der Adresse `ics.uci.edu`.

Die PKW-Datenbank besteht aus zwei Teilen: die Datei `imports-85.data` mit den eigentlichen Nutzdaten sowie die Datei `imports-85.names`, die eine Beschreibung der Attribute und weitere Informationen enthält. Beide Dateien sind unter dem Pfad `pub/machine-learning-databases/autos/` zu finden.

Es wurden folgende Attribute verwendet:

1. **symboling**, -3, -2, -1, 0, 1, 2, 3 (Versicherungseinstufung).
2. **make**, alfa-romeo, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota, volkswagen, volvo.
3. **fuel-type**, diesel, gas.
4. **aspiration**, std, turbo.
5. **num-of-doors**, 2,4.
6. **body-style**, hardtop, wagon, sedan, hatchback, convertible.
7. **drive-wheels**, 4wd, fwd, rwd.
8. **engine-location**, front, rear.
9. **wheel-base**, continuous from 86.6 120.9.
10. **length**, continuous from 141.1 to 208.1.
11. **width**, continuous from 60.3 to 72.3.
12. **height**, continuous from 47.8 to 59.8.
13. **curb-weight**, continuous from 1488 to 4066.
14. **engine-type**, dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
15. **num-of-cylinders**, 2,3,4,5,6,8,12.
16. **engine-size**, continuous from 61 to 326.
17. **fuel-system**, 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
18. **bore**, continuous from 2.54 to 3.94.

19. **stroke**, continuous from 2.07 to 4.17.
20. **compression-ratio**, continuous from 7 to 23.
21. **horsepower**, continuous from 48 to 288.
22. **peak-rpm**, continuous from 4150 to 6600.
23. **city-mpg**, continuous from 13 to 49.
24. **highway-mpg**, continuous from 16 to 54.
25. **price**, continuous from 5118 to 45400.

Anhang B

Meßwerte

Meßwerte für die Grafik 3.13.

Generierung	Zeit [min]
alt	0,8
Cobweb	18
Entropie	3,375
d. Ähnl. ohne Matrixber.	6
d. Ähnl. mit Matrixber.	18,75

Meßwerte für die Grafiken 3.14, 3.15 und 3.16.

Generierung	n	#Examined	#BOB	#BWB	#BOB+#BWB	Zeit [sec]
alt	1	3	10	11	21	0,27
alt	2	15	28	35	63	5,30
alt	3	15	28	35	63	5,31
alt	5	16	28	36	64	4,94
alt	10	45	54	77	131	10,31
alt	20	71	63	97	160	11,29
CobWeb	1	2	14	15	29	4,87
CobWeb	2	7	34	39	73	6,89
CobWeb	3	14	49	58	107	9,89
CobWeb	5	18	50	61	111	14,84
CobWeb	10	38	68	92	160	15,07
CobWeb	20	57	79	116	195	16,01
Entropie	1	2	8	9	17	1,65
Entropie	2	14	20	27	47	4,69
Entropie	3	21	29	39	68	6,79
Entropie	5	26	35	47	82	7,77
Entropie	10	66	66	96	162	15,62
Entropie	20	101	76	120	196	19,05
d. Ähnl.	1	3	8	9	17	0,14
d. Ähnl.	2	8	12	15	27	2,59
d. Ähnl.	3	13	12	17	29	2,77
d. Ähnl.	5	13	12	17	29	2,84
d. Ähnl.	10	39	40	56	96	8,48
d. Ähnl.	20	70	40	56	96	10,18
linear	$*$	205	0	0	0	9

Meßwerte für die Grafiken 4.25, 4.26 und 4.27.

Retrieval	n	#Examined	#BOB	#BWB	#BOB+#BWB	Zeit [sec]
ohne virtuelle B.	1	8	8	11	19	2,19
ohne virtuelle B.	2	83	44	78	122	10,25
ohne virtuelle B.	3	97	45	85	130	11,18
ohne virtuelle B.	5	97	45	85	130	11,03
ohne virtuelle B.	10	201	81	162	243	22,37
ohne virtuelle B.	20	205	82	165	247	21,49
mit virtuelle B.	1	3	8	9	17	0,14
mit virtuelle B.	2	8	12	15	27	2,59
mit virtuelle B.	3	13	12	17	29	2,77
mit virtuelle B.	5	13	12	17	29	2,84
mit virtuelle B.	10	39	40	56	96	8,48
mit virtuelle B.	20	70	40	56	96	10,18

Meßwerte für die Grafiken 4.31 und 4.32.

Retrieval	n	#Examined	#BOB	#BWB	#BOB+#BWB
ohne zus. BOB	1	3	8	9	17
ohne zus. BOB	2	8	12	15	27
ohne zus. BOB	3	13	12	17	29
ohne zus. BOB	5	13	12	17	29
ohne zus. BOB	10	39	40	56	96
ohne zus. BOB	20	70	40	56	96
mit zus. BOB	1	3	9	17	26
mit zus. BOB	2	6	13	25	38
mit zus. BOB	3	11	15	26	41
mit zus. BOB	5	11	15	26	41
mit zus. BOB	10	34	50	81	131
mit zus. BOB	20	63	73	113	186

Meßwerte für die Grafiken 4.33 und 4.34.

Retrieval	n	Zeit (Fälle im HS)	Zeit (Fälle auf Platte)
ohne zus. BOB	1	0,1	2,3
ohne zus. BOB	2	2,6	7,7
ohne zus. BOB	3	2,8	10,8
ohne zus. BOB	5	2,8	11,3
ohne zus. BOB	10	8,5	30,9
ohne zus. BOB	20	10,2	46
mit zus. BOB	1	0,9	2,1
mit zus. BOB	2	2,5	5,9
mit zus. BOB	3	2,5	9,3
mit zus. BOB	5	2,5	8,9
mit zus. BOB	10	8,5	27,13
mit zus. BOB	20	11,5	46,1
linear	$*$	9	45

Meßwerte für die Grafiken 5.3, 5.4 und 5.5.

Generierung	Retrieval	n	#Examined	#BOB	#BWB	#BOB+#BWB
alt	ohne virtuelle B.	1	13	17	23	40
alt	ohne virtuelle B.	2	81	60	99	159
alt	ohne virtuelle B.	3	81	60	99	159
alt	ohne virtuelle B.	5	81	60	99	159
alt	ohne virtuelle B.	10	205	93	187	280
alt	ohne virtuelle B.	20	205	93	187	280
d. Ähnl.	mit virtuelle B.	1	3	8	9	17
d. Ähnl.	mit virtuelle B.	2	8	12	15	27
d. Ähnl.	mit virtuelle B.	3	13	12	17	29
d. Ähnl.	mit virtuelle B.	5	13	12	17	29
d. Ähnl.	mit virtuelle B.	10	39	40	56	96
d. Ähnl.	mit virtuelle B.	20	70	40	56	96

Meßwerte für die Grafiken 5.5 und 5.6.

Generierung	Retrieval	n	Zeit [sec] (Fälle im HS)	Zeit (Fälle auf Platte)
alt	ohne virtuelle B.	1	1,0	1,7
alt	ohne virtuelle B.	2	12,0	45,7
alt	ohne virtuelle B.	3	12,0	46,2
alt	ohne virtuelle B.	5	12,3	46,5
alt	ohne virtuelle B.	10	27,8	113,3
alt	ohne virtuelle B.	20	27,1	111,8
d. Ähnl.	mit virtuelle B.	1	0,1	0,6
d. Ähnl.	mit virtuelle B.	2	2,6	9,0
d. Ähnl.	mit virtuelle B.	3	2,8	9,0
d. Ähnl.	mit virtuelle B.	5	2,8	9,1
d. Ähnl.	mit virtuelle B.	10	8,5	23,4
d. Ähnl.	mit virtuelle B.	20	10,2	44,6
-	linear	$*$	9	45

Literaturverzeichnis

[AX91] Klaus-Dieter Althoff und Stefan Wess. Case-Based Knowledge Acquisition, Learning and Problem Solving for Diagnostic Real World Tasks. In *Proceedings EKAW-91* (1991)

[AMWT92] K.-D. Althoff, F. Maurer, S. Wess und R. Traphöner. MOLTKE – an integrated workbench for fault diagnosis in engeneering systems. In S. Hashemi, J. P. Marciano and G. Guarderes, editors, *Proc. 4th international conference Artificial Intelligence & Expert Systems Aplications (EXPERSYS-92)*, Paris, October 92. i.i.t.t international. (1992)

[AWBJMV92] Klaus-Dieter Althoff, Stefan Wess, Brigitte Bartsch-Spörl, Dietmar Janetzko, Frank Maurer, Angi Voß. Fallbasiertes Schließen in Expertensystemen: Welche Rolle spielen Fälle für wissensbasierte Systeme? In *Künstliche Intelligenz* KI(4/92), 14–21, FBO-Verlag, Baden-Baden (1992)

[Aur84] F. Aurenhammer. *Gewichtete Voronoi-Diagramme – Geometrische Deutung und Konstruktions-Algorithmen.* Intitut für Informationsverarbeitung, Technische Universität Graz (1984)

[Aur88] F. Aurenhammer. *Voronoi-Diagrams – A Survey.* Intitut für Informationsverarbeitung, Technische Universität Graz (1988)

[BM88] R. Barletta and W. Mark. Explanation-Based Indexing of Cases. In *Proceedings of the Case-Based Reasoning Workshop*, pages 50–61, Morgan Kaufmann Publishers. Clearwater Beach, Florida (1988)

[Car93] Claire Cardie. Using Decision Trees to Improve Case-Based Learning. In Derek Sleeman, Peter Edwards (Eds.), *Machine Learning, Proceedings of the 10th International Workshop (ML93)*, Morgan Kaufmann Publishers, San Mateo, CA (1993)

[Cre93] Ulf Cremer. *ID5R: Inkrementeller Aufbau von Entscheidungsbäumen.* Projektarbeit Universität Kaiserslautern (1993)

[Fis87.1] Douglas H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning 2, 139–172*, Kluwer Academic Publishers, Boston (1987)

[Fis87.2] Douglas H. Fisher. *Knowledge Acquisition Via Incremental Conceptual Clustering.* Tech. Rept. No. 87-22 (Doctoral Dissertation), Department of Information and Computer Science, University of California, Irvine, CA (1987)

[Fro91] Ingo Frobenius. *Verfahren der Clusteranalyse, der künstlichen Intelligenz und der neuronalen Netze zum unsupervised Learning.* Diplomarbeit Universität Kaiserslautern (1991)

[GC85] M. A. Gluck, J. E. Corter. Information, uncertainty, and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* 283–287, Lawrence Erlbaum Associates, Irvine, CA (1985)

[GLF89] John H. Gennari, Pat Langley and Dough Fisher. Models of Incremental Concept Formation. *Artificial Intelligence* **40** *(1989) 11–91*, Elsevier Science Publishers B.V. (North-Holland)

[Kol88] Janet Kolodner. Retrieving Events from a Case Memory: A Parallel Implementation. In *Proceedings of the Case-Based Reasoning Workshop*, pages 233–249, Morgan Kaufmann Publishers. Clearwater Beach, Florida (1988)

[Koo87] L. H. Koopmans. *Introduction to Contemporary Statistical Methods.* Second Edition, Duxbury Press, Bostion (1987)

[Leb86] Michael Lebowitz. Concept Learning in a Rich Input Domain: Generalisation-Based Memory. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach* (Vol. 2), Los Altos, CA: Morgam Kaufmann (1986)

[MAT93] Michel Manago, Eric Auriol, Ralph Traphöner, Stefan Wess and Klaus Dieter Althoff. *Integration between Induction and Case-Based Reasoning in INRECA project.* ESPRIT project 6322 (1993)

[NHS84] J. Nievergelt, H. Hinterberger und H. J. Sevcik. The grid file: an adaptable, symmetric multikey file structure. In *ACM Trans. Database System*, 9:1, 38–71 (1984)

[Öch92] Hannes Öchsner. *Mehrdimensionale Zugriffspfadstrukturen für das ähnlichkeitsbasierte Retrieval von Fällen.* Diplomarbeit Universität Kaiserslautern (1992)

[ÖW92] Hannes Öchsner und Stefan Wess. Ähnlichkeitsbasiertes Retrieval von Fällen durch assoziative Suche in einem mehrdimensionalen Datenraum. In: K.-D. Althoff, S. Wess, B. Bartsch Spörl und D. Janetzko (Hrsg.), Proc. *Ähnlichkeit von Fällen beim fallbasierten Schließen*, SEKI-Report, Universität Kaiserslautern, Deutschland (1992)

[PPW92] J. Paulokat, R. Präger und S. Wess. CAPPLAN - fallbasierte Arbeitsplanung. In T. Messner und A. Winkelhofer (Hrsg.), *Beiträge zum 6. Workshop Planen und Konfigurieren*, Nr. 166 in FR-1992-001, S. 169, Deutschland (1992)

[RiW90] Michael M. Richter und Oliver Wendel. *Lernende Systeme, Teil I: Symbolische Methoden.* Manuskript zur Vorlesung „Lernende Systeme“ im Wintersemester 1990/91, Universität Kaiserslautern (1990)

[RKW89] Edwina L. Rissland, Janet L. Kolodner und David Waltz. Cased-Based Reasoning from darpa: Machine Learning program plan. In J. Hammond *Proceedings: Case-Based Reasoning Workshop, San Mateo, California, 1989*, 1–13. Pensacola Beach, Florida, USA, May 31 - June 1 (1989)

[Sch92] Hans Schwinn. *Relationale Datenbanksysteme.* Hanser Studienbücher der Informatik (1992)

[Sch93] Werner Schirp. *Einsatz von Voronoi-Diagrammen für das Ähnlichkeits-Retrieval in fallbasierten Systemen.* Projektarbeit Universität Kaiserslautern (1993)

[SHK89] R. H. Stottler, A. L. Henke and J. A. King. Rapid Retrieval Algorithms for Case-Based Reasoning. In *Proceedings of the 11th International Conference of Artificial Intelligence IJCAI-89*, pages 233–237. IJCAI, 1989. Detroit, Michigan, USA (1989)

[SW86] Craig Stanfill and David Waltz. Toward Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1229 (1986)

[TS93] Toshikazu Tanaka, Naomichi Sueda. Combining Strict Matching and Similarity Assessment for Retrieving Appropriate Cases Efficiently. In *Working Notes AAAI SPring Symposium: Case-Based Reasoning and Information Retrieval*, pages 112–119 (1993)

[UB90] Paul E. Utgof, Carla E. Brodley. An Incremental Method for Finding Multivariate Splits for Decision Trees. In *Proceedings of the 7th International Conference of Machine Learning*, 58–65, Morgan Kaufmann Publishers, San Mateo, CA

[Utg88] Paul E. Utgof. ID5. An Incremental ID3. In *Proceedings of the 5th International Conference of Machine Learning*, 107–120, Morgan Kaufmann Publishers, San Mateo, CA

[Utg90] Paul E. Utgof, Carla E. Brodley. Incremental Learning of Decision Trees. In *Machine Learning 4*, 161–186

[WAD93] Stefan Wess, Klaus-Dieter Althoff and Guido Derwand. Improving the Retrieval Step in Case-Based Reasoning. In M. M. Richter, S. Wess, K.-D. Althoff, F. Maurer (Eds.). First European Workshop for Case-Based Reasoning – EWCBR-93, Posters and Presentations, Volume I, pages 83–88. Kaiserslautern, Germany (1993)

[Wes90] Stefan Wess. PATDEX/2 – *Ein System zum fallfokusierenden Lernen in technischen Diagnosesituationen.* Diplomarbeit Universität Kaiserslautern (1990) und SEKI-Working Paper SWP-91-01 (1991)

[Wes93] Stefan Wess. PATDEX – Inkrementelle und wissenbasierte Verbesserung von Ähnlichkeitsurteilen in der falbasierten Diagnostik. In *Tagungsband 2. deutsche Expertensystemtagung XPS-93*. Springer Verlag, Hamburg (1993)