Fallbasiertes Schließen – Case-Based Reasoning

Grundlagen und Anwendung für Konstruktions- und Entwurfsaufgaben

Benno Stein
Michael Suermann
Hans Kleine Büning

Hans-Werner Kelbassa
Andreas Reckmann
Rainer Tellmann
Carsten Thurau
Hans-Gerd Wiegard

Universität Paderborn
Fakultät EIM
Institut für Informatik
D-33100 Paderborn

Zusammenfassung

Fallbasierte Techniken (Case-Based Reasoning, CBR) bei der rechnergestützten Lösung von Problemen haben sich bewährt. Das gilt insbesondere für Aufgaben, bei denen die Vorgehensweise des Menschen nur unzureichend verstanden ist oder nur mit hohem Aufwand nachgebildet werden kann.
Fallbasiertes Schließen ist ein dem Menschen ureigenes Prinzip. Seine Übertragung auf den Rechner bedeutet die Umsetzung von Konzepten wie *Fallähnlichkeit*, *Fallspeicherung* oder *Fallanpassung*.
Dieser Forschungsbericht gibt eine Einführung in die Grundlagen des fallbasierten Schließens. Besonderer Schwerpunkt ist die Anwendung dieses Prinzips für Aufgaben aus dem Bereich der Konstruktion und des Entwurfs.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Wissensrepräsentation in CBR-Systemen

1.1 Einführung

Beim Fallbasierten Schließen handelt es sich um einen Ansatz zur Modellierung des menschlichen Denkens sowie zum Bau intelligenter Systeme. Die Grundidee besteht darin, daß Erfahrungen in Form von Fällen gespeichert und zur Lösung neuer Probleme wiederverwendet werden. Dazu werden ähnliche Erfahrungen aus dem Speicher abgerufen und im Kontext der neuen Situation verarbeitet. Hierbei neu gewonnene Erfahrung kann wieder gespeichert werden.

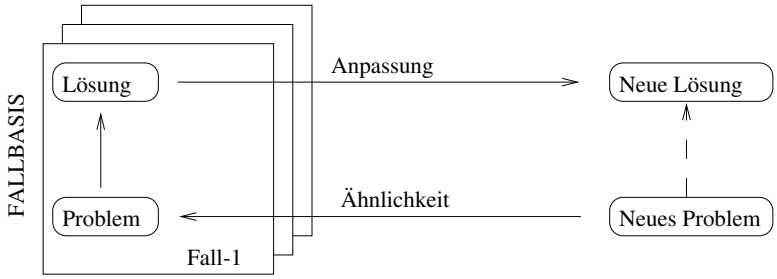


Abbildung 1.1: Prinzipielles Vorgehen beim fallbasierten Schließen

Analoges Vorgehen läßt sich beim Menschen beobachten. Ein Arzt erinnert sich z.B. an die Krankengeschichte eines anderen Patienten und versucht so, eine Diagnose für einen neuen Patienten zu stellen. Ein weiteres Beispiel ist ein Jurist, der mit einem ähnlichen Präzedenzfall argumentiert.

Definition 1.1 (Fallbasiertes Schließen) *Fallbasiertes Schließen ist Problemlösen unter Verwendung von gespeicherten Fällen.*

Warum benutzt man fallbasierte Systeme? Es ergeben sich die folgenden Vorteile:

- Vermeidung eines hohen Wissensakquisitionsaufwandes
  Der Graphik kann man entnehmen, daß in einem wissensbasierten System sehr viel allgemeines Wissen (z.B. in Form von Regeln) einfließt. Im Unterschied dazu repräsentieren bei einem fallbasierten System die Fälle den größten Teil des Wissens (Fallbasis), ausgenommen das Ähnlichkeitsmaß. Da Fallwissen oft einfacher zu erhalten ist (z.B. in Form von Konstruktionszeichnungen), gestaltet sich hier die Wissensakquisation leichter.
- Einfache Wartung des Wissens
  Gewöhnlich ist es sehr schwer, große Wissensbasen, in denen das Wissen in Form von Regeln vorliegt, konsistent zu halten. Dies liegt daran, daß es viele Abhängigkeiten zwischen den Regeln gibt, die der Benutzer nicht überschaut. Bei Änderungen einer Regel innerhalb großer Regelbasen ist es selbst für Experten ohne Systemunterstützung kaum möglich, die Auswirkungen einer Regeländerung zu übersehen. Durch die Verwendung von Fallbasen lassen sich einige der oben genannten Nachteile beseitigen. Fälle sind oft unabhängig voneinander und auch für Laien regelmäßig leichter zu verstehen. Eine Wartung der Fallbasis kann dementsprechend relativ

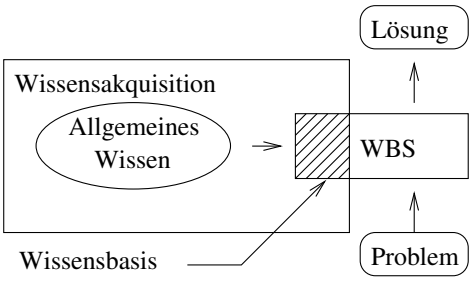


Abbildung 1.2: Aufbau eines klassischen wissensbasierten Systems (WBS)

einfach durch Hinzufügen oder Löschen von Fällen geschehen, ohne daß das implementierte Ähnlichkeitsmaß geändert werden muß [16].

- Höhere Qualität der entstehenden Lösung
  Da die Problembereiche oft nicht vollkommen verstanden werden, ist es selbst für Experten nicht immer möglich, Regeln anzugeben, die zu einer guten Lösung führen. Falls man in einer solchen – nicht vollständig verstandenen – Domäne jedoch eine gute Lösung vorliegen hat (z.B. einen bewährten Konstruktionsplan), gelingt es dem CBR-System möglicherweise, diese Lösung zu finden und an die neue Situation anzupassen. Man muß dabei voraussetzen, daß geringe Anpassungen keine großen Auswirkungen auf die Qualität der neuen Lösung haben.

- Höhere Effizienz beim Problemlösen
  In klassischen Systemen kann es vorkommen, daß Lösungen von Grund auf neu generiert werden müssen. Dies wirkt sich gerade bei komplexen Problemen auf die Effizienz aus. Da das fallbasierte Schließen die Wiederverwendung von Erfahrungen ermöglicht, muß man Lösungen häufig „nur noch“ abändern.

- Ausnutzung bestehender Datenbestände
  Häufig kommt es vor, daß Datenbestände schon vorhanden sind, diese von Datenbanken nur begrenzt ausgenutzt werden können. Fallbasiertes Schließen ermöglicht dagegen die Weiterverarbeitung (Anpassung) der gefundenen Daten.

- Benutzerakzeptanz
  Da ein fallbasiertes System dem Benutzer den ausgewählten Fall sowie die Lösungsanpassungen mitteilen kann, ist es für diesen leichter, das Ergebnis nachzuvollziehen. Bei neuronalen Netzen ist die Situation anders. Dort kann es vorkommen, daß der Inferenzprozeß kompliziert und unübersichtlich ist. Eine benutzerfreundliche Darstellung eines solchen komplexen Prozesses ist häufig sehr schwer zu realisieren.

1.2 Dynamic Memory

1.2.1 Kognitionswissenschaftlicher Hintergrund

Die Anfänge des fallbasierten Schließens gehen auf Arbeiten zurück, die an der *Yale University* durchgeführt wurden [7]. Dort beschäftigte sich eine Gruppe um Roger Schank mit den Problemen des Sprachverständnisses. Wesentlich für das fallbasierte Schließen war dabei vor allem die Entwicklung der Scripts als Wissensrepräsentationsformalismus. Ein Script ist ein Gedächtnisschema, das bestimmte Gedächtnisinhalte strukturiert. Es hilft bei der Analyse von Ereignissen oder Handlungen, indem es die jeweils spezifischen Aktionen vorhersagt, die allgemein in einer bestimmten Situation auftreten. Einen Restaurantbesuch würde man zum Beispiel durch daß Script der Abbildung 1.3 beschreiben. Man kann mit Hilfe dieses Scripts vorhersagen, welche Aktion nach dem Essen geschieht (hier: Bezahlen). Ein Problem beim Benutzen von Scripts liegt darin, daß nur allgemeines Wissen, z.B. über Restaurants, vorliegt. Spezifisches Wissen, z.B. daß es auch Restaurants gibt, in denen man zuerst bezahlen muß, liegt nicht vor. Aufgrund der Nachteile der Script-Theorie wurde die für das fallbasierte Schließen wichtige Dynamic-Memory-Theorie als eine Erweiterung der Script-Theorie entwickelt. Die Entwicklung der dynamischen Gedächtnisstrukturen basierte auf folgender These:

Die Vorgänge des Verstehens, Erinnerns und Lernens sind eng miteinander verknüpft. Sie basieren auf den gleichen Gedächtnisschemata. Erlebte Episoden werden vorzugsweise dann gespeichert, wenn sie in irgendeiner Art und Weise

Abbildung 1.3: Script eines Restaurantbesuches

nicht den ursprünglichen Erwartungen entsprochen haben (z.B. Restaurant, in dem man zuerst bezahlt). Durch den Versuch, die falsche Erwartung zu erklären, wird der Prozeß des Lernens ausgelöst [4].

1.2.2 Dynamic Memory

Speicherstruktur

Der Kernpunkt der Theorie des dynamischen Gedächtnisses ist die Existenz von speziellen Strukturen, deren Aufgabe es ist, Speicherinhalte gezielt miteinander zu verbinden. Diese Speicherstrukturen, Memory Organization Packets oder auch kurz MOPs genannt, verwalten Informationen darüber, wie Elemente des Speichers zueinander in Verbindung stehen. Einen MOP entspricht einer Art Script, wobei man mit Hilfe der Indizes auf spezielleres Wissen, d.h. die Ausnahmen, zugreifen kann [4]. Die folgende Graphik soll dies veranschaulichen.

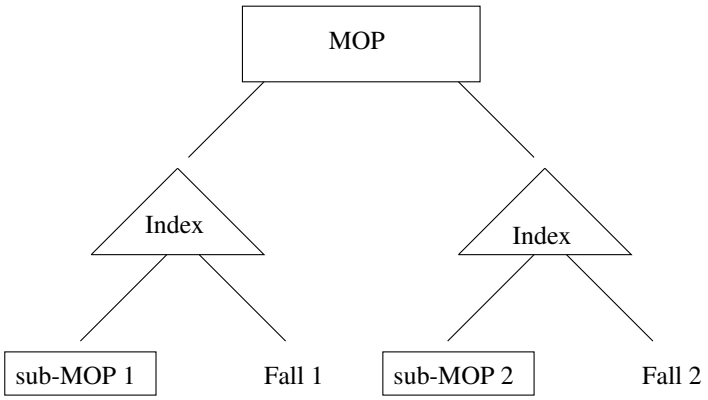


Abbildung 1.4: Graphische Darstellung eines Memory Organization Packets (MOP)

Die MOPs sind dabei in einer Spezialisierungshierarchie angeordnet. Je *tiefer* ein MOP in der Struktur liegt, um so spezieller ist er. Als Beispiel möge eine Speicherstruktur dienen, die unterschiedliche Vogelarten verwalten soll. Angenommen ein MOP besitzt die Norm[1] : Nahrung = Fisch, ein sub-MOP die Norm: Nahrung = Insekten. Dies besagt folgendes: Alle Vogelarten (das sind die Fälle), die vom sub-MOP verwaltet werden, d.h. unter ihm liegen, essen gewöhnlich Insekten. Die Norm *Nahrung = Fisch* wurde also überschrieben.

Der *Dynamic Memory*-Ansatz organisiert also sowohl allgemeines Wissen (z.B. Ablauf eines Restaurantbesuchs) als auch spezielles Wissen (Fälle, wie z.B.: In einem *bestimmten* Restaurant muß man zuerst bezahlen).

Reorganisation

Der *Dynamic Memory* ändert sich durch Erfahrungen, d. h. durch das Speichern neuer Fälle, das Einfügen neuer und das Ändern bestehender Indizes, sowie Erzeugen neuer Generalisierungen (MOPs) aus Fällen. Die Indizierung ist

[1] Unter Normen versteht man Merkmale, die das *normale Verhalten* des MOPs beschreiben

dabei für das Erinnern, d.h. das Auffinden des Speichereintrages, der der aktuellen Situation am nächsten kommt, von zentraler Bedeutung.

Falls der Speicherstruktur ein neuer Fall vorgelegt wird, muß entschieden werden, an welchen Positionen im Netz dieser neue Fall angelegt werden soll. Die Indizes, die im neuen Fall angegeben sind, definieren dabei eine Menge von Pfaden durch die Speicherstruktur. An jedem Punkt der Pfade können folgende Möglichkeiten auftreten:

- Wenn ein Fall an diesem Punkt gespeichert ist, dann wird dieser mit dem neuen Fall verglichen und die Ähnlichkeiten werden in einem neuen MOP untergebracht. Zusätzlich werden die beiden Fälle mittels ihrer Unterschiede unterhalb des neuen MOPs indiziert.

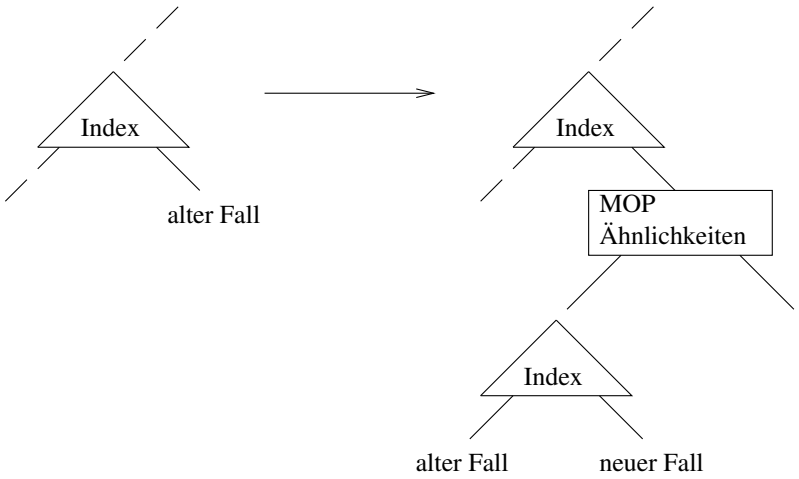


Abbildung 1.5: Speicherstruktur vor und nach dem Einfügen eines neuen Falles

- Wenn an einem Punkt ein MOP vorliegt, dann wird der neue Fall unterhalb dieses MOPs indiziert.

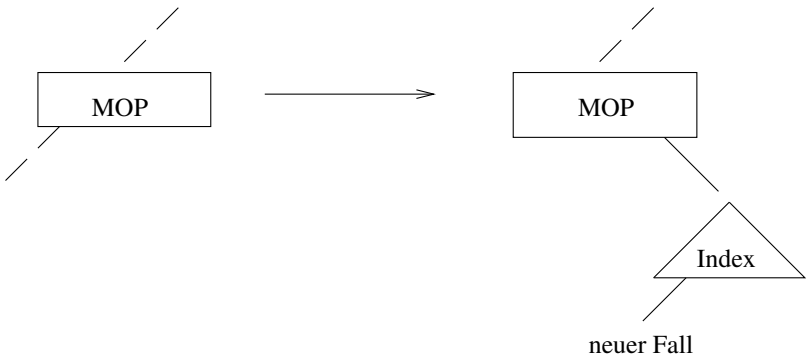


Abbildung 1.6: Speicherstruktur nach dem Einfügen des neuen Falles

Schlußfolgerungen ziehen

Das Ziehen von Schlußfolgerungen geschieht folgendermaßen: Wenn man einen neuen Fall bekommt, besteht die Aufgabe darin, die Fälle in der Speicherstruktur zu finden, die dem neuen Fall am ähnlichsten sind. Daraufhin wird die Struktur des Speichers reorganisiert, um dem neuen Fallwissen Rechnung zu tragen.

Wie im vorherigen Abschnitt beschrieben, definiert die Menge der Merkmale (aus dem neuen Fall) eine Menge von Pfaden in der Speicherstruktur. $\mathcal{N}$ soll die Menge der Schlußfolgerungen sein, die an einem beliebigen Punkt P in der Struktur ableitbar sind. Diese Menge läßt sich so bestimmen:

Seien k Pfade, deren Länge l_k ist, von der Wurzel nach P gegeben, wobei l_k den Index des MOPj definiert. Dieser Index j ergibt sich aus dem MOPj, zu welchem der Pfad l_k führt bzw. bei welchem der Pfad l_k endet. Ergibt sich für P beispielsweise der Pfad MOP1 – MOP2, so ist $l_k = l_1 = 2$. Für den Pfad MOP1 – MOP3 dagegen ist $l_k = l_1 = 3$. Analoges gilt, wenn der Pfad bei einem Fall endet, d. h. wenn P ein Fall ist. Für die Schlußfolgerungsmenge $\mathcal{N}$ gilt:

$$\mathcal{N} = \bigcup_{i=1}^{k} \mathcal{N}_i$$
$$\mathcal{N}_i = \bigsqcup_{j=1}^{l_k} \vartheta_j,$$

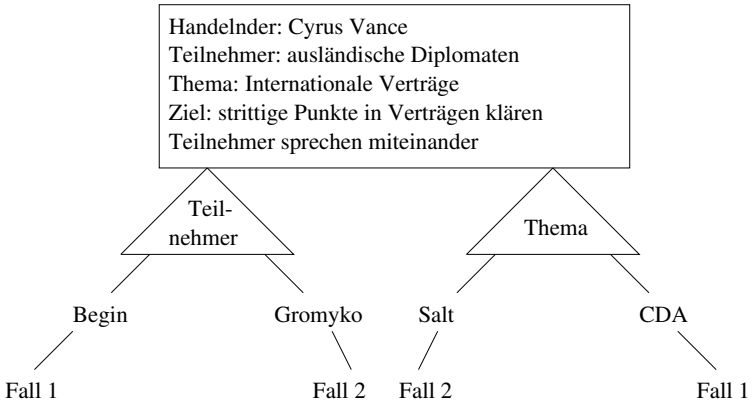
wobei ϑ_j die Normen des MOPs j sind und $\bigsqcup$ folgendermaßen definiert ist:

> Für jede allgemeine Norm $x \in \mathcal{N}_i$ gilt:
> WENN zu einer Norm x eine speziellere Norm $y \in \mathcal{N}_{i+1}$ existiert,
> DANN ist $\mathcal{N}_{i+1} = (\mathcal{N}_i \setminus x) \cup y$ die nächste Norm;
> SONST ist $\mathcal{N}_{i+1} = \mathcal{N}_i \cup x$ die nächste Norm (Vererbung).

Dies bedeutet also, daß bei jedem Schritt entlang eines Pfades, die neuen, spezielleren Werte die allgemeineren Normen ersetzen.

Das System CYRUS

Das CBR-System CYRUS basiert auf einer Erweiterung des Dynamic Memory Ansatzes. Es wurde von *Kolodner* zur Speicherung und Erinnerung von Ereignissen aus der beruflichen Tätigkeit des ehemaligen US–Außenministers Cyrus Vance entwickelt [7]. Eine Ausgangssituation zeigt Abbildung 1.7. Das Einfügen zweier neuer Fälle 3 und 4 in diese Struktur führt zu dem Ergebnis in Abbildung 1.8.



Fall 1: Cyrus verhandelt
mit Begin über CDA

Fall 2: Cyrus verhandelt
mit Gromyko über Salt

Abbildung 1.7: MOP 1 „Diplomatisches Treffen“ als Ausgangssituation in CYRUS

Man kann Anfragen der folgenden Form stellen: *Hat Cyrus Vance jemals mit Dayan über das Camp-David-Abkommen (CDA) gesprochen?* Das CBR-System würde hier als Antwort zwei Ableitungen liefern:

- MOP 1 - Index: Teilnehmer (Dayan) → Fall 4
- MOP 1 - Index: Thema (CDA) → MOP 3 → Index: Teilnehmer (Dayan) → Fall 4

Das System CYRUS zählt in der CBR-Literatur zu den bekanntesten Systemen, obwohl es eigentlich kein klassischer Problemlöser ist, denn im engeren Sinne wird ja hiermit kein Problem gelöst. Klassische Systeme zum fallbasierten Schließen gliedern die Falldefinition in eine Problembeschreibung und in eine dazu gehörende Problemlösung.

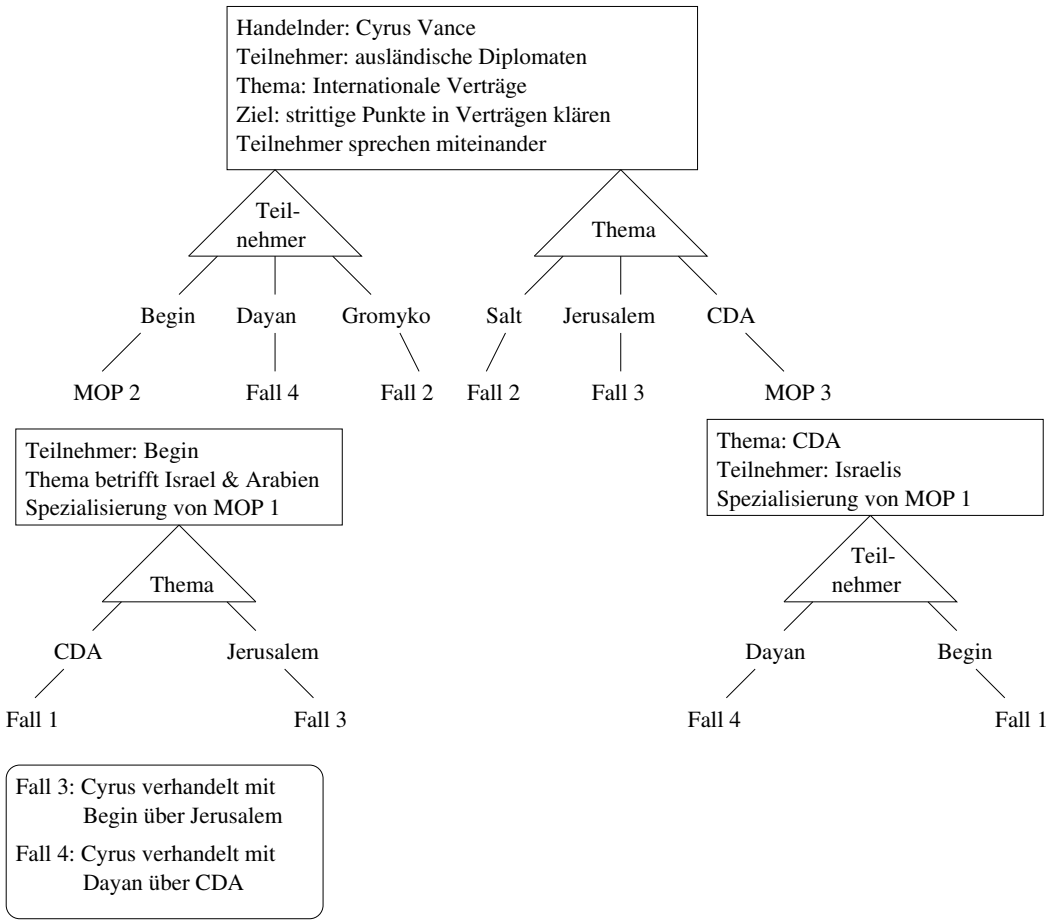


Abbildung 1.8: MOP 1 „Diplomatisches Treffen“ nach dem Einfügen der Fälle 3 und 4

1.3 Formale Grundlagen

Reiter und *Etherington* zeigten, daß es möglich ist, bestimmte Speicherstrukturen in Ausdrücke der *Default-Logik* umzuwandeln. Der Grund hierfür ist folgender: Man möchte eine formale Grundlage für die Schlußfolgerungen haben, die mit Hilfe derartiger Strukturen erzeugbar sind. In diesem Abschnitt wird deshalb ein Zusammenhang zwischen der *Default-Logik* und dem Fallspeicher hergestellt.

1.3.1 *Default-Logik*

Obwohl das Wissen über die Welt meistens unvollständig ist, und sich häufig nicht alle gewünschten Fakten ableiten lassen, möchte man dennoch Entscheidungen treffen. Vielfach verwendet man deshalb Regeln mit Ausnahmen, um so das fehlende Wissen zu ergänzen. Solche Regeln drücken aus, was typischerweise der Fall ist, und können dazu benutzt werden, um plausible Konklusionen abzuleiten, sofern keine der jeweiligen Konklusion widersprechende Information vorliegt. Diese Regeln bezeichnet man deshalb auch als *Default-Regeln*.

Ein zweiter wichtiger Punkt ist der Umgang mit inkonsistenter Information. Es kommt ständig vor, daß sich die vorliegende Information widerspricht, z. B. weil sie aus unterschiedlichen Quellen stammt. Man muß also einen Weg finden, der es erlaubt, mit inkonsistenten Informationen umzugehen.

Für die beiden gerade genannten Probleme bietet die klassische Prädikatenlogik keine gute Lösung. Sicherlich könnte man eine Regel wie *„Vögel fliegen typischerweise“* folgendermaßen darstellen:

$$\forall x : Vogel(x) \wedge \neg Ausnahme(x) \Rightarrow Fliegt(x)$$
$$\forall x : Ausnahme(x) \Leftrightarrow Pinguin(x) \vee Strauss(x) \vee \neg HatFlügel(x) \ldots$$

Diese Darstellung erfordert aber eine vollständige Auflistung aller möglichen Ausnahmen. Dies ist aber so gut wie nie

möglich. Aber selbst wenn wir eine solche Auflistung hätten, wäre die Darstellung sehr unpraktisch. Um beispielsweise für einen bestimmten Vogel, sagen wir Tweety, abzuleiten, daß er fliegt, ist es notwendig zu beweisen, daß kein Ausnahmefall vorliegt, d.h. daß Tweety kein Pinguin ist, kein Strauß, usw. Das möchte man aber gerade nicht: Man möchte *„Tweety fliegt"* ableiten können, wenn nicht gezeigt werden kann, daß Tweety eine Ausnahme ist.

Worin liegt die Schwierigkeit? Die klassische Logik besitzt folgende Monotonieeigenschaft: Für alle Mengen von Prämissen A und Formeln p, q gilt: $A \vdash q \Rightarrow A \cup \{p\} \vdash q$; dies bedeutet, daß zusätzliche Information nie alte Konklusionen ungültig machen kann. Wie oben im Beispiel zu sehen war, möchte man eine *Default-Regel* dazu benutzen, um eine plausible Konklusion abzuleiten, sofern nichts auf einen Ausnahmefall hindeutet. Wenn man später zusätzliche Information erhält, die besagt, daß entgegen den Erwartungen doch eine Ausnahme vorliegt, dann muß diese Konklusion zurückgenommen werden. Jedes Schließen, das auf Regeln mit Ausnahmen basiert, muß deshalb in diesem Sinne nichtmonoton sein. Dasselbe gilt konsequenterweise auch für eine Logik, die dieses Schließen formalisiert.

Ein nichtmonotones System ermöglicht die Verarbeitung von Klassenhierarchien (Speicherstrukturen), wie sie oben dargestellt wurden. Die grundlegende Idee ist die, daß im Falle von Widersprüchen die spezifischen Informationen, die sich in den sub-MOPs befinden, den allgemeineren Informationen vorgezogen werden.

Formal gesehen handelt es sich bei der *Default-Theorie* um ein Paar (D, W), wobei W eine Menge von Formeln darstellt, die sicheres Wissen repräsentiert. Bei D handelt es sich hingegen um eine Menge von *Defaults* der Form $\frac{A:B_1,\ldots,B_n}{C}$. Diese *Default-Regel* ist folgendermaßen aufzufassen: Wenn A ableitbar ist und für alle $i(1 \leq i \leq n), \neg B_i$ nicht abgeleitet werden kann, dann leite C ab. Die Menge der Schlußfolgerungen, die in der *Default-Theorie* ableitbar sind, werden *extensions* (Erweiterungen) genannt [5].

1.3.2 Umformulierung von CBR in *Default-Logik*

Um diese Umformulierung verständlich zu machen, sei ein Beispiel erläutert. Gegeben sei der in Abbildung 1.9 dargestellte Ausschnitt aus einer Speicherstruktur, die verschiedene Vogelarten unterscheidet.

Jeder Knoten besitzt eine Menge von Merkmalen. Jedes Merkmal hat einen Attributnamen und einen dazugehörigen Wert. Wenn der Knoten ein *Fall* ist, so repräsentieren diese Merkmale Eigenschaften des Falles. Wenn der Knoten ein *MOP* ist, repräsentieren dessen Merkmale Eigenschaften, die von den meisten Fällen, die durch das MOP organisiert werden, geteilt werden. Jeder Unterschied ist eine Verbindung von einem MOP zu einem anderen MOP oder zu einem Fall. Jeder Unterschied ist durch mindestens ein Merkmal gekennzeichnet und differenziert zwischen den spezielleren Knoten und den Verallgemeinerungen.

Ausgehend von einer derartigen Speicherstruktur möchte man eine Umwandlung in die *Default-Logik* vornehmen. Die Merkmale der Knoten, d. h. die Normen der MOPs und die Merkmale der Fälle, sowie die Verbindungen der Knoten untereinander liefern die benötigten Inferenzregeln:

1. Merkmal: Ein Knoten $N(x)$ hat das Merkmal $f(x, a)$ und es gibt keine Spezialisierung zu einem Unterknoten, der mit diesem Merkmal korrespondiert. Dies hieße nämlich, daß dieses Merkmal überschrieben werden würde. Eine solche Situation wird interpretiert als: Normalerweise haben N's den Wert a für das Merkmal f. Identifiziert mit:
$$\frac{N(x) : f(x, a)}{f(x, a)}$$

2. Verbindung: Ein MOP x ist mit einem Knoten $N_*(x)$ durch den Unterschied $f(x, b)$ verbunden. Dies bedeutet: N_*'s unterscheiden sich vom MOP dadurch, daß sie den Wert b für das Merkmal f besitzen. Identifiziert mit:
$$\frac{N(x) \wedge f(x, b) : N_*(x)}{N_*(x)}$$

3. Unterscheidungsnorm: Ein MOP x hat eine *Norm* $f(x, a)$ sowie Unterschiede $f(x, b_1), \ldots, f(x, b_n)$. Dies bedeutet: N's haben gewöhnlich einen Wert a für das Merkmal f, solange es keine Ausnahmen gibt. Identifiziert mit:
$$\frac{MOP(x) \wedge \neg f(x, b_1) \wedge \ldots \wedge \neg f(x, b_n) : f(x, a)}{f(x, a)}$$

4. Restricted Closed World (RCW): Ein MOP x hat eine *Norm* $f(x, a)$ und ist mit einem Knoten $N(x)$ durch den Unterschied $f(x, b)$ verbunden. Zusätzlich existieren weitere Verbindungen zu $N(x)$, $f_1(x, b_1), \ldots, f_n(x, b_n)$. Dies bedeutet: Wenn das MOP nicht durch die Unterschiede $f_1(x, b_1), \ldots, f_n(x, b_n)$ zu N spezialisiert werden

kann, dann soll man nicht annehmen, daß das MOP den Wert b für das Merkmal f annimmt (siehe nachfolgendes Beispiel; setze in Punkt 4: $MOP =$ MOP1, $N =$ MOP3 und $b =$ Fisch). Identifiziert mit:

$$\frac{MOP(x) \wedge \neg f_1(x, b_1) \wedge \ldots \wedge \neg f_n(x, b_n) : \neg f(x, b)}{\neg f(x, b)}$$

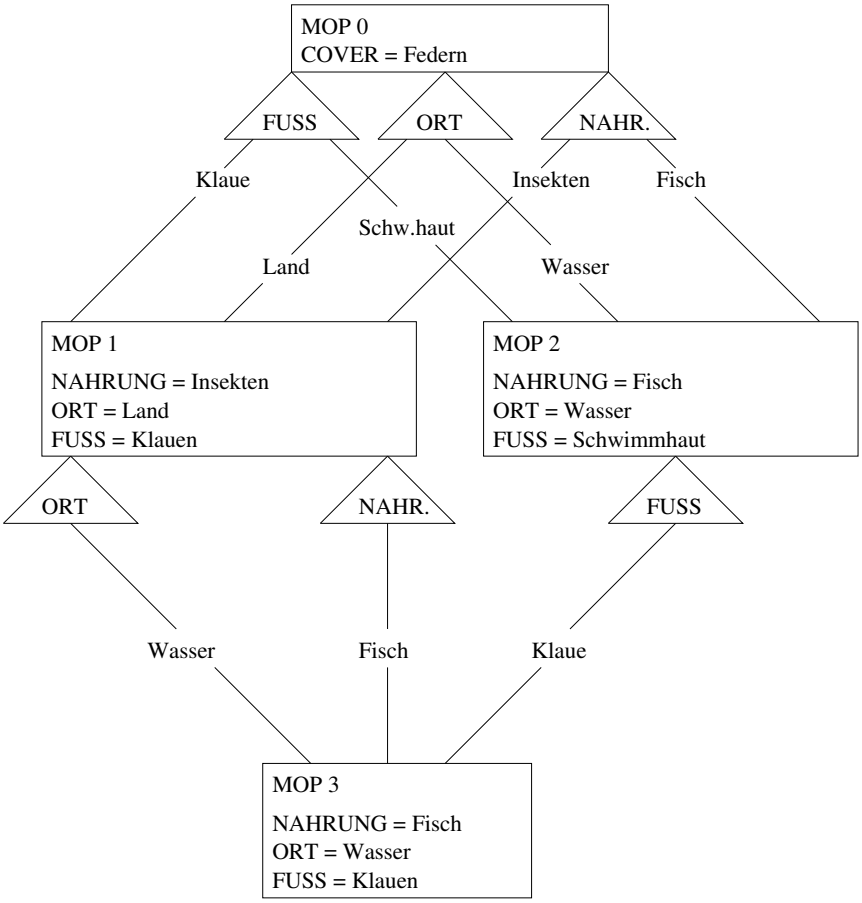


Abbildung 1.9: Ausschnitt aus einer Speicherstruktur zur Unterschiedung von Vogelarten

Es fällt auf, daß vier *Default-Regeln* benötigt werden, um die Beziehung zwischen einem MOP und einer Spezialisierung davon darzustellen. Beispiel: MOP3 unterscheidet sich von MOP1 dadurch, daß MOP3 das Merkmal Nahrung = Fisch hat. Daraufhin werden vier Regeln erzeugt:

1. Eine Regel, die beschreibt, daß MOP3 das Merkmal *Nahrung = Fisch* besitzt.
2. Eine Regel, die beschreibt, daß MOP3 sich von MOP1 durch den Unterschied des Merkmals *Nahrung = Fisch* auszeichnet.
3. Eine Regel, die explizit die Annahme macht, daß MOP1 gewöhnlich das Merkmal *Nahrung = Insekten* hat, sofern nicht *Nahrung = Fisch* gilt.
4. Eine Regel, die explizit die Annahme macht, daß MOP1 typischerweise nicht das Merkmal *Nahrung = Fisch* hat.

Wendet man dieses Verfahren auf die oben angegebene Speicherstruktur an, erhält man folgende

Default-Theorie:

$$W = \{\forall x : MOP0(x)\}$$

$$D = \left\{ \begin{array}{ll} \frac{MOP0(x):cover(x,Federn)}{cover(x,Federn)} & \frac{MOP0(x) \wedge Fuss(x,Klauen):MOP1(x)}{MOP1(x)} \\ \frac{MOP0(x) \wedge Fuss(x,Schwimmhaut):MOP2(x)}{MOP2(x)} & \frac{MOP0(x) \wedge Ort(x,Land):MOP1(x)}{MOP1(x)} \\ \frac{MOP0(x) \wedge Ort(x,Wasser):MOP2(x)}{MOP2(x)} & \frac{MOP0(x) \wedge Nahrung(x,Insekt):MOP1(x)}{MOP1(x)} \\ \frac{MOP0(x) \wedge Nahrung(x,Fisch):MOP2(x)}{MOP2(x)} & \frac{MOP1(x) \wedge Ort(x,Wasser):MOP3(x)}{MOP3(x)} \\ \frac{MOP1(x) \wedge Nahrung(x,Fisch):MOP3(x)}{MOP3(x)} & \frac{MOP2(x) \wedge Fuss(x,Klauen):MOP3(x)}{MOP3(x)} \\ \frac{MOP1(x) \wedge \neg Ort(x,Wasser):Ort(x,Land)}{Ort(x,Land)} & \frac{MOP1(x) \wedge \neg Nahr.(x,Fisch):Nahrung(x,Insekt)}{Nahrung(x,Insekt)} \\ \frac{MOP1(x) \wedge \neg Nahr.(x,Fisch):\neg Ort(x,Wasser)}{\neg Ort(x,Wasser)} & \frac{MOP1(x) \wedge \neg Ort(x,Wasser):\neg Nahr.(x,Fisch)}{\neg Nahrung(x,Fisch)} \\ \frac{MOP2(x) \wedge \neg Fuss(x,Klauen):Fuss(x,Schw.haut)}{Fuss(x,Schwimmhaut)} & \frac{MOP2(x):\neg Fuss(x,Klauen)}{\neg Fuss(x,Klauen)} \\ \frac{MOP3(x):Ort(x,Wasser)}{Ort(x,Wasser)} & \frac{MOP3(x):Nahrung(x,Fisch)}{Nahrung(x,Fisch)} \\ \frac{MOP3(x):Fuss(x,Klauen)}{Fuss(x,Klauen)} & \end{array} \right\}$$

1.3.3 Beispiel mit einer Erweiterung

Wir gehen davon aus, daß wir folgendes wissen: $cover(Tweety, Federn)$, $Fuss(Tweety, Klauen)$ und $Ort(Tweety, Wasser)$. Aus diesen Aussagen können wir folgende *Erweiterungen* ziehen: Tweety ist eine Instanz vom MOP3, MOP2, MOP1 und MOP0; sowie: Tweety ißt Fisch, lebt beim Wasser, seine Füße besitzen Klauen und er ist mit Federn bedeckt. Der Grund, warum es nur eine Erweiterung gibt, wurde oben schon erwähnt. Dies liegt daran, daß speziellere Werte die weniger speziellen Verallgemeinerungen (Normen) überschreiben können. Formal ausgedrückt:

$$\begin{array}{ll} MOP3(Tweety) & MOP2(Tweety) \\ MOP1(Tweety) & MOP0(Tweety) \\ Nahrung(Tweety, Fisch) & Ort(Tweety, Wasser) \\ Fuss(Tweety, Klauen) & cover(Tweety, Federn). \end{array}$$

Dabei verhindert die Regel

$$\frac{MOP2(x) \wedge \neg Fuss(x, Klaue) : Fuss(x, Schwimmhaut)}{Fuss(x, Schwimmhaut)}$$

zusammen mit dem Merkmal *Fuss(Tweety, Klauen)* das Vererben der Eigenschaft *Fuss(x,Schwimmhaut)* vom MOP2.

Die Regel

$$\frac{MOP1(x) \wedge \neg Ort(x, Wasser) : Ort(x, Land)}{Ort(x, Land)},$$

zusammen mit dem Merkmal $Ort(x, Wasser)$ verhindert hier im Beispiel die Vererbung von $Ort(x, Land)$. Das Merkmal $Ort(Tweety, Wasser)$, in Verbindung mit den Regeln

$$\frac{MOP1 \wedge \neg Ort(x,Wasser):\neg Nahrung(x,Fisch)}{\neg Nahrung(x,Fisch)},$$

$$\frac{MOP1(x) \wedge \neg Nahrung(x,Fisch):Nahrung(x,Insekten)}{Nahrung(x,Insekten)}$$

verhindert die Vererbung von $Nahrung(x, Insekten)$.

1.3.4 Beispiel mit mehreren Erweiterungen

Angenommen es ist uns bekannt, daß Dick Quaker ist, es gelte also $party(Dick, republican)$. Wir können auf folgende Erweiterungen schließen:

$$E_1 = G_1(Dick), G_0(Dick), religion(Dick, quaker), party(Dick, republican), pacifist(Dick, yes)$$
$$E_2 = G_2(Dick), G_0(Dick), party(Dick, republican), religion(Dick, quaker), pacifist(Dick, no)$$

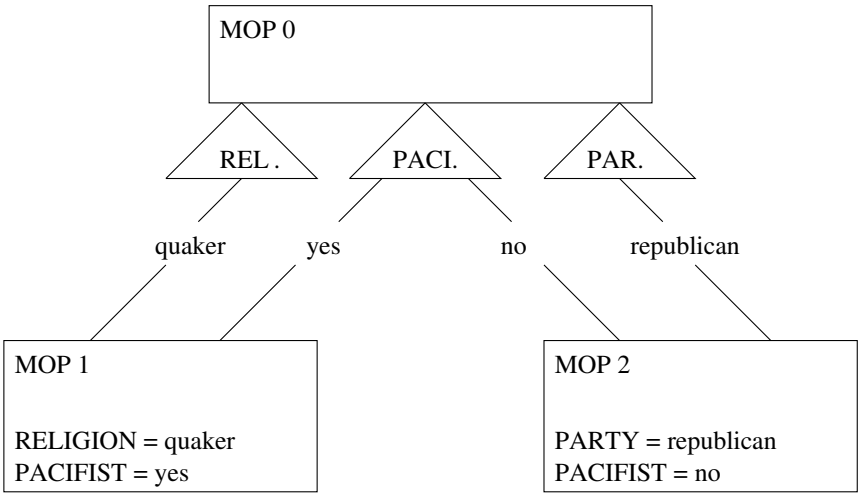


Abbildung 1.10: Speicherstruktur mit mehreren Erweiterungen

Das Quaker-Republican Beispiel zeigt, daß es möglich ist, aus der zum Fallspeicher gehörenden *Default-Theorie* mehrfache Erweiterungen zu ziehen. Dies ist in komplexen Speicherstrukturen sogar der Normalfall. Typischerweise besitzen fallbasierte Systeme einen Mechanismus, um bestimmte Merkmale anderen vorzuziehen.

1.4 Fallwissen

1.4.1 Fallwissen und dessen Strukturierung

Man kann drei Kategorien von Fallwissen unterscheiden:

- Problem-/Situationsinformation
  Die Situationsbeschreibung muß alle problemrelevanten Informationen enthalten, damit man entscheiden kann, ob der Fall in einer neuen Situation anwendbar ist. Welche Informationen man dazu benötigt, das hängt von der jeweiligen Lösung ab. In einer Situationsbeschreibung könnten beispielsweise das Ziel der Problemlösung oder Einschränkungen zur Lösung (Constraints) angegeben werden.
- Lösungsinformation
  Hier werden alle Informationen angegeben, die die Lösung des Problems hinreichend genau beschreibt. Zusätzlich können Daten angegeben werden, die bei der Lösungsanpassung (Adaption) nützlich sind. Mögliche Komponenten der Lösungsinformation:
  - Lösung selbst (z.B. Klassenbeschreibung);
  - Lösungsweg (Folge von Schritten, mit denen das Problem gelöst wurde);
  - Rechtfertigung für getroffene Entscheidungen;
  - Alternative Lösungsschritte, die ebenfalls erfolgreich wären;
  - Lösungsschritte, die versuchsweise ausgeführt wurden und fehlgeschlagen sind.
- Güteinformation
  Unter einer Güteinformation versteht man die „Rückmeldung“ aus der realen Welt, z. B. ob die Lösung erfolgreich war oder fehlschlug (Validierungsangaben). Weiterhin läßt sich der Aufwand (Kosten, Zeit) für die Realisierung der Lösung angeben.

Die Abbildung 1.11 veranschaulicht diese drei Kategorien.

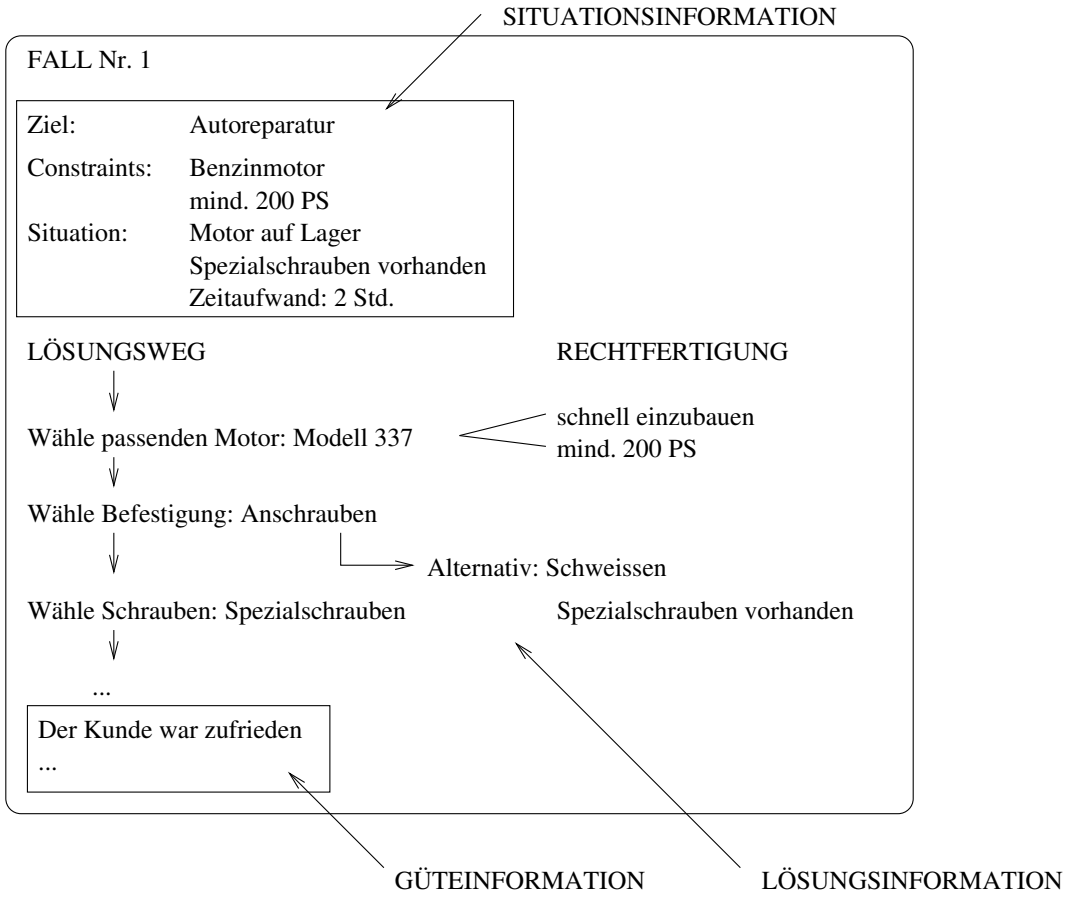


Abbildung 1.11: Fallwissen

1.4.2 Wissensumfang im Fall

Nachdem im zweiten Abschnitt zum Thema Dynamic Memory eine Möglichkeit erläutert wurde, Fälle mittels einer geeigneten Speicherstruktur zu repräsentieren, z. B. um ein effektives *Retrieval* durchzuführen, interessiert, ob es Möglichkeiten gibt, aufgrund einer *besseren* Fallrepräsentation die Effektivität einer solchen Struktur zu steigern. Eine derartige Effektivitätssteigerung kann z. B. durch eine einfachere Repräsentation der Fälle oder durch eine verbesserte Grundlage für eine spätere Ähnlichkeitsbestimmung erzielt werden. Im folgenden werden hierzu zwei Möglichkeiten, nämlich die Abstraktion sowie die Generalisierung von Fällen, erläutert. Diese sollen zur Verbesserung des *Dynamic Memory*-Ansatzes beitragen.

Abstrakte Fälle

Zum besseren Verständnis dient die Abbildung 1.12. Dort werden drei verschiedene Arten von Fällen unterschieden. Hierbei handelt es sich um:

- konkrete Fälle (Fall auf der niedrigsten Abstraktionsebene)
- abstrakte Fälle (Fall auf einer einzigen höheren Abstraktionsebene)
- hierarchische Fälle (Fall, der mehrere Abstraktionsebenen umfaßt)

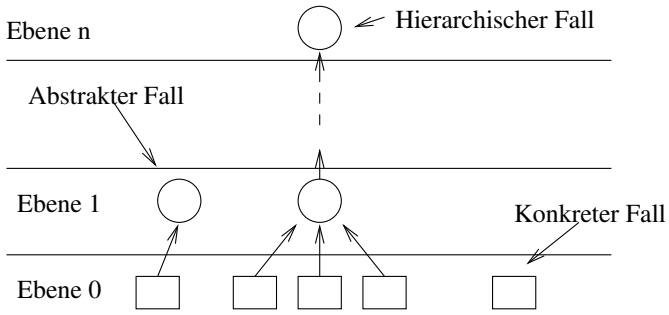


Abbildung 1.12: Unterscheidung von Fallarten

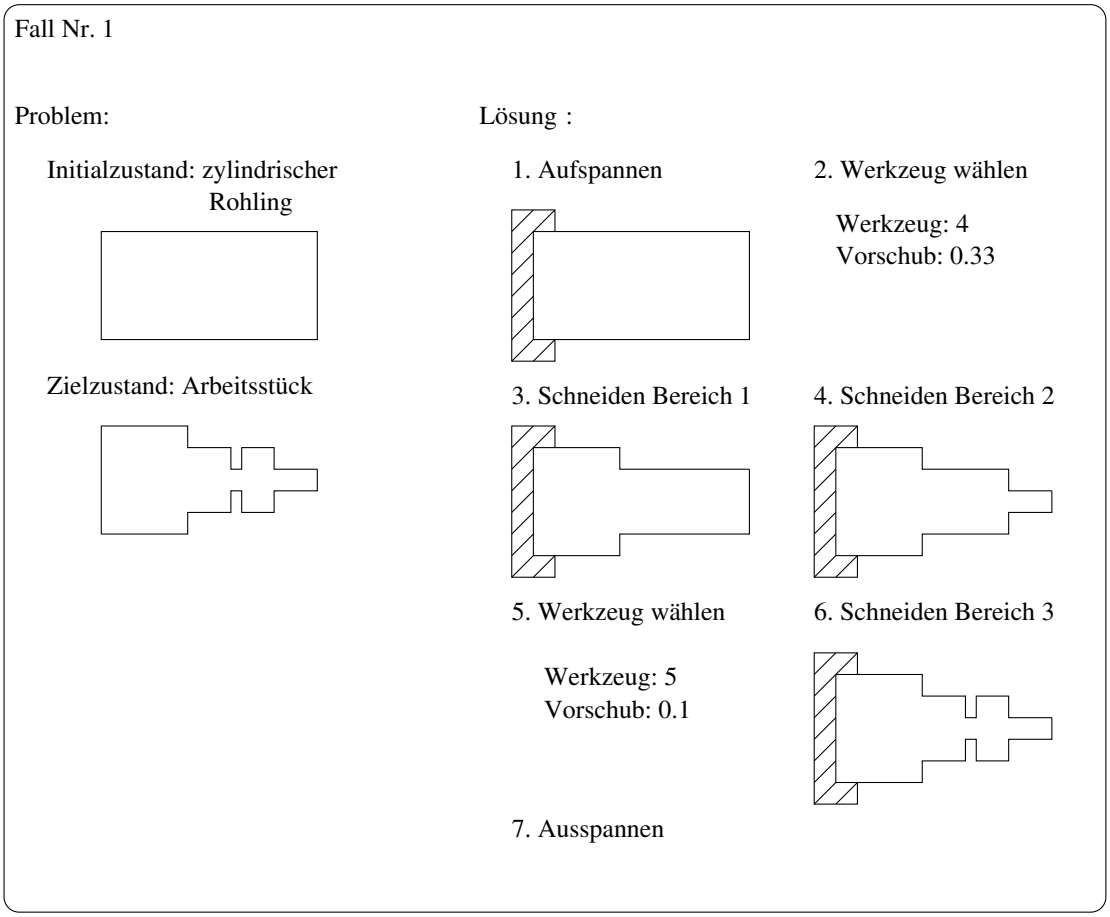


Abbildung 1.13: Beispiel für einen konkreten Fall „Drehen“

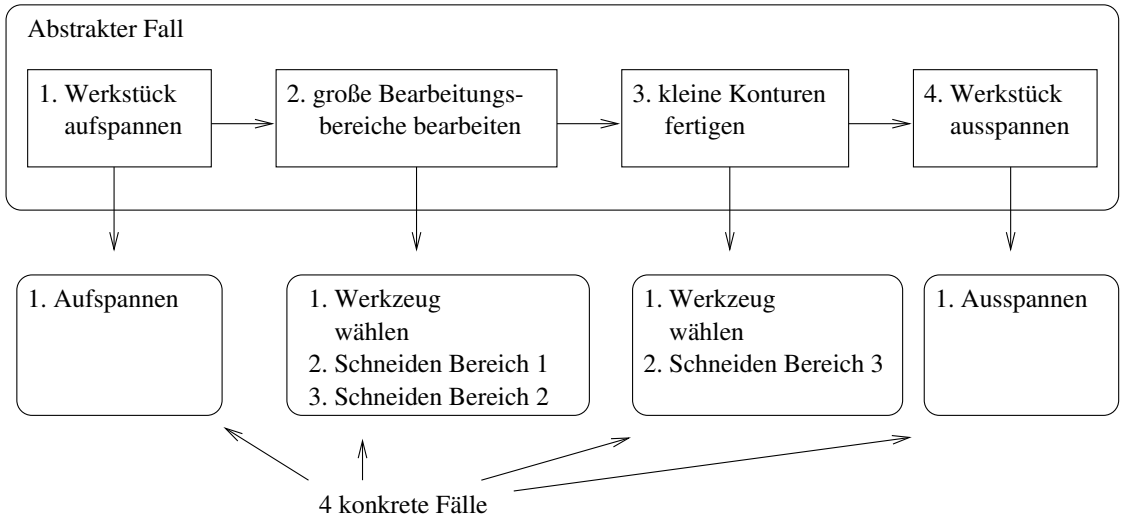


Abbildung 1.14: Beispiel für einen abstrakten Fall

Eine abstraktere Ebene hat die Eigenschaft, daß sie weniger Details repräsentiert als die darunterliegende. Sie besitzt somit weniger Beschreibungselemente, d.h. weniger Merkmale oder auch Relationen. Eine Folge davon ist natürlich, daß die Erfahrung, die man letztendlich darstellen möchte, auf dieser abstrakten Ebene nicht sehr präzise beschrieben wird. Dennoch sollte man darauf achten, daß wichtige Eigenschaften auch hier beibehalten werden.

Die Abbildung 1.13 und 1.14 gibt ein Beispiel für eine konkrete und eine abstrakte Fallbeschreibung. Man sieht hier, daß ein Fall (eine Situation) durch eine Menge von Fällen auf verschiedenen Abstraktionsebenen gespeichert ist. Nur der Fall auf der höchsten Abstraktionsebene beschreibt die Situation vollständig (wenn auch nur abstrakt). Die Fälle auf den niedrigeren Ebenen enthalten nur Ausschnitte der gesamten Situation.

Generalisierte Fälle

Die Grundidee besteht darin, daß ein generalisierter Fall für eine Menge *normaler*, einander sehr ähnlicher Fälle steht. Dies wird realisiert, indem man anstelle eines Attributwertes eine Menge von Attributwerten zuläßt (mehrwertige Attribute). Man könnte dies als Verwendung einer ODER-Beziehung auffassen (siehe Beispiel: i52 ODER i57 ODER i59). Wie man sich leicht denken kann, führt eine solche Generalisierung zu einer Verkleinerung der Fallbasis. Außerdem trägt sie zu einer Vereinfachung der Lösungsanpassung bei.

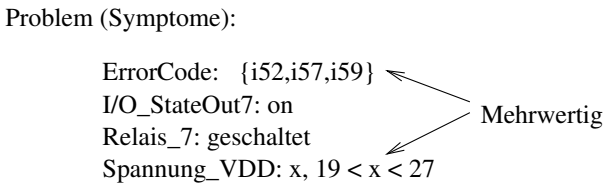


Abbildung 1.15: Generalisierter Fall

1.4.3 Wissensrepräsentationsformalismus

Attribut-Wert-Repräsentation

Ein Fall wird durch Attribut-Wert-Paare repräsentiert wie z.B. das Attribut $Preis$ mit $Wert$ 10 DM. Die Menge der Attribute (A_i) wird entweder für alle Fälle fest vorgegeben oder variiert von Fall zu Fall. Jedem Attribut ist ein Wertebereich (Typ) zugeordnet, z.B. Integer, Real, Symbol oder Text. Falls die Attributmenge fest vorgegeben ist, entspricht ein Fall einem n-stelligen Vektor $F = (a_1, \ldots, a_n) \in T_1 \times \ldots \times T_n$, wobei $a_i \in T_i$ (a_i sind die Werte und T_i die Typen/Wertebereiche). Unbekannte Attributwerte können durch die Einführung eines speziellen Symbols *unknown* dargestellt werden. Daraus folgt: $a_i \in T_i \cup \{unknown\}$.

Bei variabler Attributmenge ist ein Fall eine Menge $F = \{A_1 = a_1, \ldots, A_n = a_n\}$ mit $a_i \in T_i$. Attribute, die nicht vorkommen, sind unbekannt. Ein formales Beispiel zeigt die Abbildung 1.16.

Problem (Symptome):

ErrorCode: i59	Symbol {i51,i52,i58,i59,i60}
I/O_StateOut7: on	Symbol {on,off}
Relais_7: geschaltet	Symbol {geschaltet, nicht-geschaltet}
Spannung_VDD: 23.8 V	Real [0..50]

Lösung (Diagnose):

Schaltschrank: III	Symbol{I,II,III}
Defekt: ''Magnetschalter''	String

Abbildung 1.16: Beispiel zu einem Diagnosefall

Bewertung der Attribut-Wert Repräsentation:

- Vorteile
  - Einfache Repräsentation, leicht zu verstehen
  - Einfache und effiziente Ähnlichkeitsbestimmung
  - Einfache Speicherung (z.B. in Datenbanken)
  - Effizientes Retrieval möglich
- Nachteile
  - Keine strukturelle und relationale Information repräsentierbar
  - Keine Reihenfolgeinformation (Aktionsfolge) repräsentierbar
- Geeignet für
  - Analytische Aufgaben, insbesondere Klassifikation
  - Große Fallbasen, wenige Merkmale
- Ungeeignet für
  - Synthetische Aufgaben

Objektorientierte Modelle

Zusammengehörige Attribute werden zu Objekten zusammengefaßt. Ein Fall besteht somit aus einer Menge von Objekten. Zwischen diesen Objekten bestehen Beziehungen, die auch Relationen genannt werden. Die wichtigsten Beziehungen sind:

- Taxonomische Relationen: Eine *a-kind-of*-Relation drückt Abstraktions- und Verfeinerungsbeziehungen zwischen Objekten der Domäne aus. Beispiel: Auto *a-kind-of* Transportmittel.

- Kompositionelle Relationen: Eine *is-part-of*-Relation drückt die Zusammensetzung von Objekten aus Teilobjekten aus. Beispiel: Fenster *is-part-of* Auto.

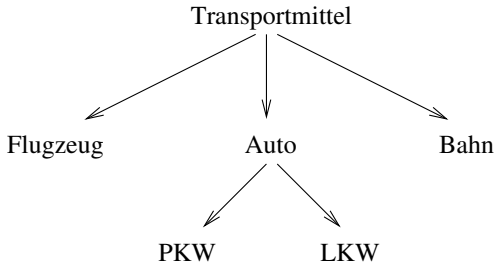


Abbildung 1.17: Taxonomische Relation

Bewertung objektorientierter Repräsentation:

- Vorteile
  - Strukturierte und „natürliche“ Fallrepräsentation
  - Strukturelle und relationale Information ist repräsentierbar
  - Kompaktere Speicherung als mit Attribut-Wert-Repräsentation
  - Strukturinformation kann für Ähnlichkeitsberechnung und Lösungsanpassung genutzt werden
- Nachteile
  - aufwendigere Ähnlichkeitsberechnung und Retrieval
- Geeignet für
  - analytische Aufgaben, insbesondere bei komplexen Fallstrukturen
  - synthetische Aufgaben im Bereich Konfiguration und Design

Bäume und Graphen

Eine Anwendung von Graphen in der Fallrepräsentation wäre beispielsweise die Beschreibung des Lösungsweges (siehe Abschnitt 1.4.1). Die Knoten würden dabei aus den durchgeführten, alternativen oder fehlgeschlagenen Problemlöseschritten, sowie den Merkmalen der Problembeschreibung bestehen. Ferner könnte man auch die Merkmale der Lösungsbeschreibung als Knoten darstellen. Die Kanten wären die jeweiligen Verbindungslinien, z.B. zum nächsten Problemlöseschritt oder zu alternativen, sowie fehlgeschlagenen Schritten.

Bewertung von Graphenrepräsentation:

- Vorteile
  - Einfache Strukturen sind darstellbar
  - Algorithmen aus der Graphentheorie (z.B. Isomorphie) können für die Ähnlichkeitsberechnung genutzt werden.
- Nachteile
  - Begrenzte Strukturierungsmöglichkeiten im Vergleich zu objektorientierter Repräsentation
- Geeignet für
  - analytische und synthetische Aufgaben in Bereichen mit Graphenstruktur
- Ungeeignet für
  - detaillierte Repräsentation komplexer, stark strukturierter Aufgaben

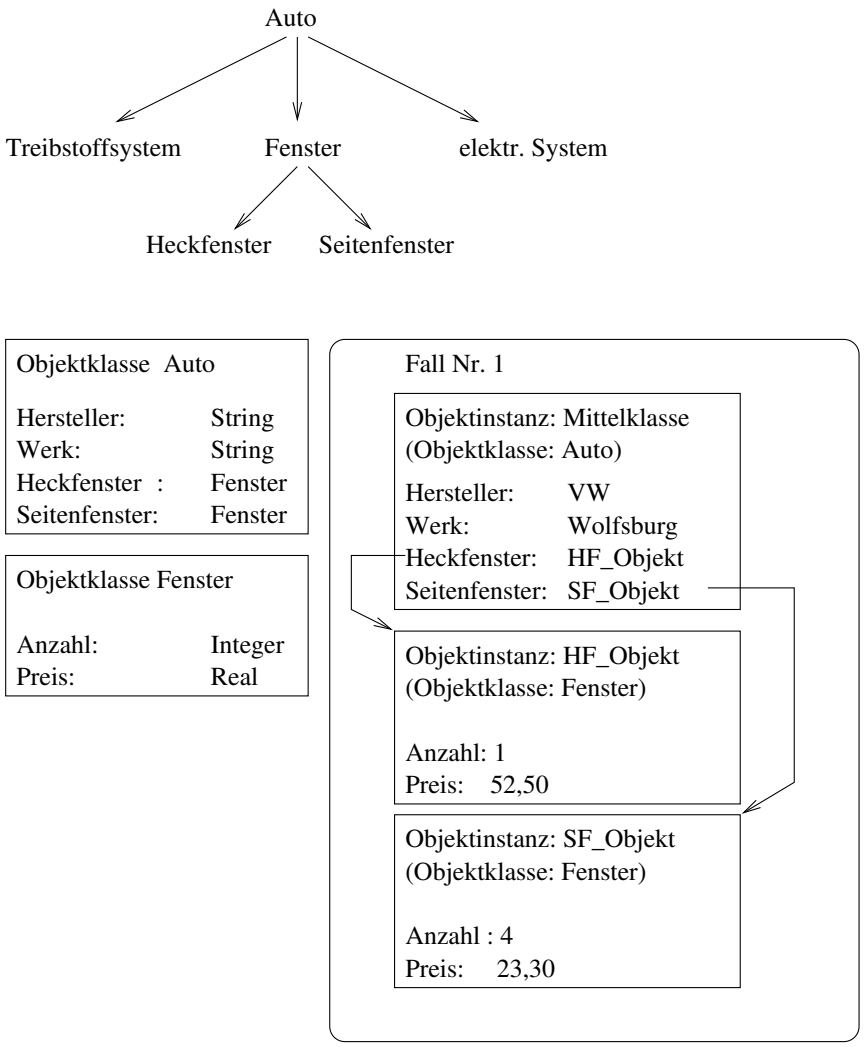


Abbildung 1.18: Kompositionelle Relation zur Objektklasse Auto

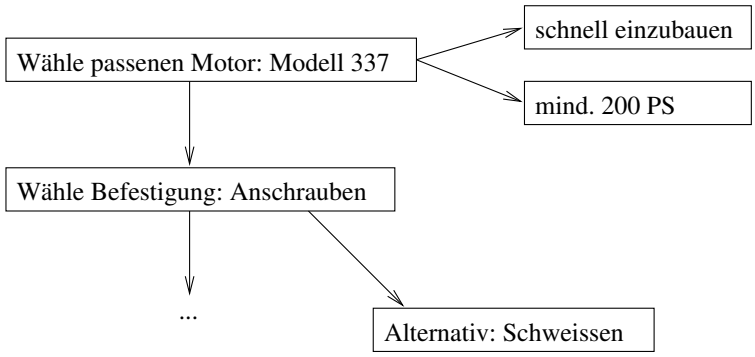


Abbildung 1.19: Fallrepräsentation als Graph

1.5 Vergleich zweier Fallbasen

Nachdem in den obigen Abschnitten die Struktur einer Fallbasis erkläutert wurde, stellt sich die Frage, welche Fälle man in diese Fallbasis aufnehmen oder gegebenenfalls später wieder entfernen sollte. Ausgangspunkt sei eine Fallbasis mit $n+1$ Fällen. Welchen Fall sollte man entfernen, um die *beste* Fallbasis mit n Fällen zu erhalten? Hierzu muß man klären, was eine *beste* Fallbasis der Größe n ist.

In der Einleitung war davon die Rede, daß Erfahrungen in Form von Fällen gespeichert werden, um sie zur Lösung neuer Probleme wiederzuverwenden. Dazu werden *ähnliche* Erfahrungen aus dem Speicher abgerufen und im Kontext der neuen Situation verarbeitet. Einen zum vorliegenden Problem ähnlichen Fall in der Fallbasis zu finden ist die Aufgabe eines Ähnlichkeitsmaßes. Hier wird angenommen, daß ein Ähnlichkeitsmaß gegeben ist, d. h. man weiß, wie ähnlich sich zwei Fälle sind. Da sich zwei Fälle aber meistens nur ähnlich – nicht identisch – sind, ist noch Arbeit zu investieren, um den mittels Ähnlichkeitsmaß gefundenen Fall in der neuen Situation auch verwerten zu können. Diese Arbeit nennt man Adaptionsleistung Φ. Allgemein geht man davon aus, daß der ähnlichste Fall sich für den Adaptionsprozess am besten eignet, da dieser a-Priori die geringste Adaptionsleistung beansprucht.

Hinsichtlich des Ausgangsproblems gilt somit, daß eine Fallbasis $\mathcal{B}_1$ besser als eine Fallbasis $\mathcal{B}_2$ ist, sofern $\Phi(\mathcal{B}_1) \leq \Phi(\mathcal{B}_2)$. Aus dieser Aussage läßt sich auch folgern, welchen Fall man aus der oben erwähnten Fallbasis der Größe $n+1$ entfernen sollte. Es wird der Fall entfernt, der $\Phi(\mathcal{B})$ minimiert. Man muß das genannte Kriterium auf die mittlere Adaptionsleistung über alle Fälle beziehen, weil zwei Fallbasen insgesamt verglichen werden sollen.

Es stellt sich weiterhin die Frage, wie die Adaptionsleistung Φ zu definieren ist. Hierzu sind noch folgende Vereinbarungen notwendig. Wir gehen davon aus, daß ein Fall als ein Paar (Problem, Lösung) vorliegt. Sei $\mathcal{P}$ die Menge aller Probleme und $\mathcal{S}$ die Menge aller Lösungen. Eine Fallbasis $\mathcal{B}$ kann dementsprechend als eine endliche Menge $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ aufgefaßt werden. Sei ferner eine Menge von Relationen $\mathcal{A} \in \mathcal{P}^2$ gegeben, wobei die Gleichheitsrelation (Δ) in $\mathcal{A}$ enthalten sei. Diese Relationenmenge $\mathcal{A}$ wird benötigt, um bei der Ähnlichkeitsbestimmung (im Retrieval) alle Probleme zu finden, die in einer Beziehung zum neuen Problem stehen[2]. Aus der Menge dieser Problemfälle wird dann ein Fall ausgewählt, der ex ante die geringste Anpassungsarbeit benötigt. Zu diesem Zweck wird allerdings noch eine Abbildung M benötigt. Sie bildet ein $\alpha \in \mathcal{A} \rightarrow M(\alpha) \in I\!R$ ab, wobei gilt: $M(\alpha) = 0$ wenn $\alpha = \Delta$ und $\alpha_1 \subseteq \alpha_2 \Rightarrow M(\alpha_1) \leq M(\alpha_2)$. Die Semantik der Abbildung M liefert eine Schätzung der Adaptionsleistung. Je kleiner der Wert von $M(\alpha)$ ist, umso weniger Arbeit muß in die Adaption gesteckt werden. Sollte also das Ergebnis $M(\alpha) = 0$ lauten, bedeutet dies, daß keine Adaption notwendig ist, weil die Probleme identisch sind.

Sind zwei Probleme $p, p_k \in \mathcal{P}$ gegeben, so hängt deren Ähnlichkeit $p\ \alpha\ p_k$ natürlich von der Semantik der Relation α ab.

Mit Hilfe dieser Vereinbarungen läßt sich die Distanz d zweier Probleme $p_1, p_2 \in \mathcal{P}$ definieren: $d(p_1, p_2) = \{M(\alpha) \mid \alpha \in \mathcal{A} \wedge p_1\ \alpha\ p_2\}$. Wenn ein $p \in \mathcal{P}$ und ein Fall $(p_k, s_k) \in \mathcal{B}$ gegeben ist, so kann man mit Hilfe der oben angegebenen Semantik von M eine problembezogene Schätzung der erforderlichen Adaptionsleistung angeben. Diese lautet $d(p, p_k)$. Gesucht wird die Distanz $d(p, p_k)$, welche bezüglich der gegebenen Fallbasis $\mathcal{B}$ minimal ist [14].

1.5.1 Beispiel

Ein Schlosser möchte Schlüssel für seine Kunden herstellen. Dazu besitzt er n verschiedene Gußformen. Wenn ein Kunde einen Schlüssel einer bestimmten Größe bestellt, stellt der Schlosser eventuell direkt diesen her, sofern er die passende Gußform hierzu besitzt, oder er gießt einen etwas größeren Schlüssel, bei dem er dann natürlich etwas Material abschleifen muß, um die richtige Schlüsselgröße zu erhalten.

Das Problem besteht darin, herauszubekommen, welche n Gußformen am günstigsten sind, so daß der Schlosser möglichst wenig abschleifen muß.

Ein Schlüssel wird hier in diesem Beispiel wie folgt dargestellt: Die Tiefe der Gußform wird der Einfachheit halber als konstant angenommen. Jeder Schlüssel γ kann also über seine Breite und seine Länge beschrieben und somit als Paar (Breite, Länge) = $(x, y) \in [1, 5] \times [1, 10]$ (z.B. in Millimeter) angesehen werden. Der Schlosser gibt also den gewünschten Schlüssel γ ein, und der *Retrieval*-Prozess liefert eine Gußform γ_k, für welche der erforderliche Adaptionsprozess, also das Abschleifen, minimal ausfällt. Wenn keine passende Gußform vorhanden ist, dann kann

[2] Dieses wird klarer, wenn man sich die Definition der Menge $\mathcal{A}$ im Beispiel 1.5.1 ansieht

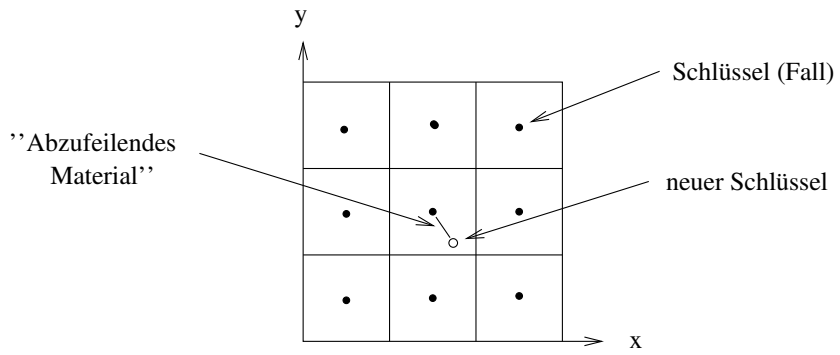


Abbildung 1.20: Definition des Ähnlichkeitsmaßes

man den Anpassungsprozess folgendermaßen darstellen:

$$d(\gamma,\gamma_k) = \begin{cases} \text{Abzufeilendes Material,} & falls\ \gamma \subseteq \gamma_k,\ k \in \{1,2,\ldots,n\} \\ \infty & \text{sonst} \end{cases}$$

Mit Hilfe der Theorie des vorherigen Abschnitts stehen jetzt folgende Aussagen zur Verfügung:

$$\begin{aligned} \mathcal{P} = \mathcal{S} &= [1,5] \times [1,10] \\ \mathcal{A} &= \{\alpha \mid \alpha = \{(\gamma,\gamma_k)\},\ (\gamma,\gamma_k) \in \mathcal{P}^2,\ \gamma \subseteq \gamma_k,\ k \in \{1,2,\ldots,n\}\} \\ \forall \alpha \in \mathcal{A} : M(\alpha) &= d(\gamma,\gamma_k) \text{ für } \alpha = \{(\gamma,\gamma_k)\} \end{aligned}$$

Der durch das Retrieval gefundene ähnlichste Fall $\gamma_k \in \mathcal{B}$ genügt der folgenden Bedingung: $d(\gamma,\gamma_k) \leq d(\gamma,\gamma_j)$ $\forall \gamma_j \in \mathcal{B}$, wobei $\forall\ j : \gamma \subseteq \gamma_j$. Die minimale fallspezifische Anpassungsarbeit wird also $d(\gamma,\gamma_k)$ sein.[3]

Sei $\gamma \in \mathcal{P}$ ein beliebiger Schlüssel, der herzustellen ist. Wenn man berücksichtigen will, daß jeder Schlüssel γ nicht gleich häufig bestellt wird, so kann man eine Zufallsvarible Γ, definiert durch $p : \mathcal{P} \rightarrow [0,1]$, einführen, welche die Wahrscheinlichkeit $p(\gamma)$ dafür angibt, daß der nächste bestellte Schlüssel γ sein wird.

Hat man eine Fallbasis $\mathcal{B} = \{(p_1,s_1),(p_2,s_2),\ldots,(p_n,s_n)\}$ und eine definierte Zufallsfunktion $d(\Gamma,\mathcal{B})$, berechnet sich der Erwartungswert E[$d(\Gamma,\mathcal{B})$] für die Anpassungsarbeit $\Phi(\mathcal{B})$ bezogen auf eine beliebige Fallbasis der Kardinalität |$\mathcal{B}$| = n wie folgt:

$$\Phi(\mathcal{B}) = E[d(\Gamma,\mathcal{B})] = \sum_{\gamma \in \mathcal{P}} d(\gamma,\mathcal{B}) * p(\gamma).$$

Zum besseren Verständnis möge die obige Graphik dienen. Dort ist die beste Fallbasis der Größe |$\mathcal{B}$| = 9 abgebildet. Bei diesem Beispiel geht man allerdings davon aus, daß die Wahrscheinlichkeit $p(\gamma)$ für alle Schlüssel gleich ist.

Das Schlosserproblem besteht also darin, eine Fallbasis $\mathcal{B}_n^*$ der Größe n zu bestimmen, so daß gilt:

$$|\mathcal{B}_n^*| = n \wedge \forall \mathcal{B} \subseteq \mathcal{P} : |\mathcal{B}| = n \Rightarrow \Phi(\mathcal{B}_n^*) \leq \Phi(\mathcal{B}).$$

Die beste Fallbasis $\mathcal{B}_n^*$ ermöglicht eine Minimierung der erforderlichen Adaption Φ.

1.5.2 Kriterium für den Vergleich zweier Fallbasen

Zusammenfassend sei gesagt, daß folgendes gilt: Ein fallbasiertes System erhält den ähnlichsten Fall mit Hilfe des Distanzmaßes d. Weiterhin ist davon auszugehen, daß für ein Problem $p \in \mathcal{P}$ und für einen Fall $(p_k,s_k) \in \mathcal{P} \times \mathcal{S}$ das Distanzmaß $d(p,p_k)$ ein Maß für die nötige Adaptionsleistung ist, d. h. je kleiner der Wert von $d(p,p_k)$ ist, umso weniger Anpassungsarbeit ist erforderlich.

Schließlich wird angenommen, daß man die Realität hinsichtlich der Schlüsselbestellungen sowie deren Häufigkeit durch eine bekannte Zufallsvariable Γ simulieren kann. Aus diesen Angaben erhält man für eine konkrete Fallbasis $\mathcal{B}$ die Adaptionleistung

$$\Phi(\mathcal{B}) = E[d(\Gamma,\mathcal{B})].$$

[3]Es gilt, daß die Gußform γ_k regelmäßig nicht der erwünschten Problemlösung s entspricht: $\gamma_k \subseteq s$, denn zur Lösung gehört regelmäßig nicht allein die Auswahl der Gußform, sondern auch das Abschleifen, worüber schließlich die erwünschten Endmaße des Schlüssels zustande kommen.

Das Kriterium für den Vergleich zweier Fallbasen $\mathcal{B}_1$ und $\mathcal{B}_2$ lautet somit:

$$\Phi(\mathcal{B}_1) \leq \Phi(\mathcal{B}_2).$$

Die beste Fallbasis $\mathcal{B}_1$ erfordert c. p. ein Minimum an Adaption.

1.6 Schlußbetrachtungen

Ziel dieses Kapitels ist es, unterschiedliche Aspekte der Wissensrepräsentation in fallbasierten Systemen zu erläutern. Dazu gehören sowohl praktische Themen, wie die *Dynamic Memory*-Theorie, Repräsentationsmöglichkeiten eines Falles sowie der Vergleich zweier Fallbasen und auch theoretische Aspekte, also z. B. die Darstellung formaler Grundlagen mit Hilfe der *Default-Logik*.

Es sei auch erwähnt, daß die beschriebenen Verfahren teilweise schon von der Forschung in Frage gestellt werden. So konnten Wissenschaftler nachweisen, daß die Dynamic-Memory-Theorie nicht mit den tatsächlichen Vorgängen im Gehirn übereinstimmt. Neue Ergebnisse in der Hirnforschung deuten darauf hin, daß es in den nächsten Jahren schon möglich sein wird, leistungsfähigere und dem menschlichen Gehirn *ähnlichere* Speicherstrukturen zu entwerfen [6].

Kostenverantwortung und Kostenfestlegung Der vorliegende Bericht widmet sich auch der Automatisierung von technischen Konstruktionsprozessen (vgl. Kapitel 4, 5), weshalb es hier angebracht ist, auf die Problematik der Kostenverantwortung sowie der Kostenfestlegung im Produktentstehungsprozeß kurz einzugehen.

Jede Produktentwicklung verursacht Kosten, die auch dann, wenn der ausgearbeitete Entwicklungsauftrag nicht zur Fertigung freigegeben wird, ganz erheblich sein können. Aus diesem Grund haben Rationalisierungsfachleute schon in den 70er Jahren die Kostenverteilung im Produktentstehungsprozeß analysiert. In einer vielzitierten Veröffentlichung skizzierte *Opitz* im Jahre 1970 ein Diagramm, welches die von den einzelnen Produktionsbereichen eines Unternehmens (Konstruktion, Arbeitsvorbereitung, Einkauf, Fertigung, Verwaltung, Vertrieb) verursachten Kosten (qualitativ) den Kosten gegenüberstellt, die von diesen Produktionsbereichen festgelegt werden [8]. Das Opitz-Diagramm wurde im Prinzip noch in den 80er Jahren von mehreren Autoren bestätigt. Es zeigt deutlich, daß die vom Konstruktionsbereich verursachten Kosten in krassem Verhältnis zu den (Produktions-)Kosten stehen, welche diese in den Konstruktionsunterlagen festlegt [8]. Die genauen Zahlen variieren je nach Branche. Man ging aber seinerzeit global davon aus, daß die Konstruktionsbereiche nur etwa 10-15% der Produktentstehungskosten verursachen, jedoch etwa 70% der Herstellkosten über das Konstruktionsergebnis festlegen. Die geringsten Kosten fallen im Maschinenbau regelmäßig im Konstruktionsbereich, die höchsten Kosten im Fertigungsbereich an. Hierüber entwickelte die Ingenieurwissenschaft seinerzeit die These von der Kostenverantwortung der Konstruktion: Da die Konstruktion die Produktionskosten in großem Umfang determiniert, ist es nicht sinnvoll, im Fertigungsbereich rigorose Rationalisierungsanstrengungen einzuleiten, ohne zumindest vergleichbare Maßnahmen im Konstruktionsbereich durchzuführen [10]. Der Entwicklungs- und Konstruktionsbereich wurde seinerzeit zum Rationalisierungsbereich Nummer 1 erklärt. Die Unternehmensleitungen haben inzwischen insbesondere auch die strategische Bedeutung der Konstruktionsautomatisierung (CAE)[4] erkannt und entsprechend gehandelt. Mittlerweile sind weltweit CIM-Systeme[5] im industriellen Einsatz, die mit Systemen im Konstruktionsbereich (CAD/CAM)[6] gekoppelt sind [11] [12].

Mit der Kostenverantwortungsthese ist die Konstruktionsautomatisierung zu einem fruchtbaren Gebiet der Informatikforschung geworden, und es ist erkennbar, daß wissensbasierte Systeme zu einem unverzichtbaren Bestandteil der Konstruktionspraxis geworden sind [15]. Der vorliegende Bericht über fallbasiertes Schließen soll vor allem auch ein Beitrag zur Automatisierung des technischen Konstruktionsprozesses sein.

Unter einem technischen Konstruktionsprozeß versteht man jene Aktivitätenkette, die entweder direkt zur stofflichen Verwirklichung eines Konstruktionsobjektes – technischen Gebildes – führt, oder zu einer Bereitstellung von Informationsträgern, welche die jeweiligen konstruktiven Funktionsträger für produktionstechnische Zwecke sowie zur Inbetriebnahme und Nutzung hinreichend gut beschreiben [13]. Die Begriffswahl Konstruktion und Entwurf orientiert sich im vorliegenden Bericht an dem klassischen Phasenmodell der VDI-Richtlinie 2222 [9].

[4] CAE steht für Computer Aided Engineering.
[5] CIM steht für Computer Integrated Manufacturing.
[6] CAD steht für Computer Aided Design; CAM steht für Computer Aided Manufacturing.

Kapitel 2

Auswahl und Klassifikation von Fällen

2.1 Phasenmodell

Dieses Kapitel widmet sich den Themen *Auswahl und Klassifikation von Fällen* aus dem Bereich des fallbasierten Schließens. In dieser Einleitung werden beide Themen in die allgemeine Funktionsweise fallbasierter Systeme eingeordnet und dann erläutert.

Das fallbasierte Schließen lehnt sich stark an eine Vorgehensweise an, die dem Menschen wohlbekannt ist. Wenn jemand ein Problem zu lösen hat, erinnert er sich an eine vergleichbare (ähnliche) Situation und versucht, diese Erfahrungen auf das neue Problem anzuwenden. Zum Beispiel löst ein Student eine Aufgabe in einer Klausur, indem er sich an die Lösung einer ähnlichen Aufgabe auf einem Übungszettel erinnert und dieses Wissen auf die aktuelle Aufgabe überträgt. Für den fallbasierten Ansatz bilden Erfahrungen die Fälle und die Menge aller verfügbaren Erfahrungen die Fallbasis. Erfahrungen sind für einen Menschen nur dann nützlich, wenn er sich auch an diese erinnern, und diese auf ein neues Problem anwenden kann. Ein fallbasiertes System erinnert sich an gespeicherte Fälle durch die Verwendung eines Ähnlichkeitsmaßes. Dabei sollte das Ähnlichkeitsmaß so formalisiert worden sein, daß die Lösung eines gefundenen Falles auch für ein neues Problem nützlich ist. Im folgenden wird eine allgemeine Vorgehensweise des fallbasierten Schließens vorgestellt. Dieses klassische Prozeßmodell[1] hat vier Phasen [37]:

Retrieve	Bereitstellung bzw. Suche mindestens eines geeigneten Falles.
Reuse	Wiederverwendung von gespeichertem Problemlösungswissen.
Revise	Test und Modifikation der gefundenen Lösung.
Retain	Speichern der neu gewonnenen Erfahrungen.

Beispiel Ein Arzt untersucht einen Patienten.

Retrieve	Der Arzt erinnert sich an die Krankengeschichte eines früheren Patienten.
Reuse	Der Arzt wendet die Behandlung des früheren Patienten auch beim neuen Patienten an.
Revise	Der Test bezieht sich auf den Erfolg der Behandlung.
Retain	Der Arzt merkt sich die Krankengeschichte und Behandlung des Patienten.

Das Thema *Auswahl von geeigneten Fällen* ist dem ersten Schritt *Retrieve* zuzuordnen. Die *Klassifikation* von Fällen durchläuft alle vier Schritte, wobei die gefundene Lösung nicht modifiziert wird, d. h. im Schritt *Revise* wird nur ein Test durchgeführt. Im folgenden werden beide Themenbereiche beschrieben.

Eines der Hauptprobleme bei der Realisierung von fallbasierten Systemen ist die Bestimmung der Ähnlichkeit von Fällen. In Kapitel 2.2 wird diesbezüglich auf die Beziehung zwischen Ähnlichkeit von Problembeschreibungen, Nützlichkeit von Problemlösungen und Unsicherheit eingegangen. Danach werden Modellierungsmöglichkeiten des Ähnlichkeitsbegriffes, d. h. Möglichkeiten, die Ähnlichkeit zwischen einzelnen Fällen formal auszudrücken, behandelt. So kann man z. B. versuchen, ähnliche Fälle zu einem Problem in einer Reihenfolge bezüglich der Ähnlichkeitausprägung anzuordnen. Es ist wünschenswert, ein Verfahren angeben zu können, welches für ein konkretes Problem eine

[1] Dieses Prozeßmodell von Aamodt und Plaza (1994) ist in jüngster Zeit von Roth-Berghofer und Iglezakis (2001) um die beiden Phasen Restore and Review erweitert worden [34].

Ähnlichkeitsbestimmung liefert. Dieses Problem ist allerdings noch Forschungsgegenstand. In diesem Kapitel werden unterschiedliche Klassen von Modellen zur Ähnlichkeitsbestimmung beschrieben, die als Anhaltspunkte für die Vorgehensweise bei der Suche nach einem Ähnlichkeitsbegriff dienen. Um ein vorteilhaftes Modell auszumachen, müssen die einzelnen Modelle vergleichbar sein. Deshalb werden unten auch Möglichkeiten zum Vergleich von Ähnlichkeitsbewertungen angegeben.

Im Abschnitt 2.3 wird eine einfache Realisierung des fallbasierten Ansatzes zur Klassifikation beschrieben. Bei der Klassifikation muß das Prozeßmodell nur teilweise realisiert werden, d. h. die Anpassung der gefundenen Lösung (Adaption) ist nicht notwendig. Aufgabe der Klassifikation ist die Einteilung von Objekten in Klassen bzw. die Zuordnung zu Begriffen. Zum Beispiel werden Tiere den verschiedenen Tierarten zugeordnet. Typische Anwendungsgebiete der Klassifikation sind die Diagnose und die Entscheidungsunterstützung. Zunächst wird ein allgemein gehaltener Basisalgorithmus für die fallbasierte Klassifikation angegeben, der die Grundlage für die weiteren Abschnitte bildet. Es werden folgende Fragestellungen untersucht:

- Wie wird Wissen in fallbasierten Systemen repräsentiert?
- Welche Beziehungen bestehen zwischen Fallbasis und dem verwendeten (Ähnlichkeits-/Distanz-)Maß?
- Welcher Zusammenhang besteht zwischen fallbasiertem Schließen und Lernen?

Im dritten Abschnitt wird die Klassifikation als ein geometrisches Problem interpretiert. Ein Ergebnis des ersten Abschnitts ist, daß man einem fallbasierten System nicht direkt ansehen kann, welche Begriffe es repräsentiert. Die geometrische Sichtweise der Klassifikation bietet die Möglichkeit, die Begriffe explizit darzustellen. Weitere Vorteile sind eine größere Anschaulichkeit der betrachteten Problemstellungen und eine Vielzahl neuer Algorithmen. Der Basisalgorithmus aus dem ersten Abschnitt ist sehr allgemein gehalten und läßt deshalb noch viele Gestaltungsmöglichkeiten offen. Im dritten Abschnitt werden daher Ansätze vorgestellt, die Möglichkeiten für eine Einteilung fallbasierter Algorithmen in Klassen bieten. Ein Ansatz ist z. B. die Einteilung bezüglich verschiedener Aufnahmestrategien von Fällen oder hinsichtlich der Änderungsstrategien des Maßes.

2.2 Auswahl von geeigneten Fällen

Unter fallbasiertem Schließen kann vereinfacht das Lösen von Problemen anhand bereits bekannter Fälle (Fallbeispiele) verstanden werden. Beispielsweise

- diagnostiziert ein Arzt die Krankheit eines Patienten, indem er sich an die Krankengeschichte anderer Patienten erinnert oder
- ein Rechtsanwalt argumentiert vor Gericht mit der Entscheidung einer höheren Instanz in einem ähnlich gelagerten Fall.

Die Auswahl von geeigneten Fällen ist ein zentrales Problem des Case-based Reasoning (CBR). Es soll regelmäßig ein Fall gefunden werden, der für die Lösung des aktuellen Problems *nützlich* ist. Das Kriterium der *Nützlichkeit* von Fällen für die *Problemlösung* muß jedoch auf die Bestimmung der *Ähnlichkeit* von *Problemstellungen* reduziert werden. Dabei hat man die Hoffnung, daß die *Ähnlichkeit* der *Problemstellung* die *Nützlichkeit* für die *Problemlösung* impliziert. Zum Beispiel führt die gleiche Behandlung, die schon bei einem Patienten mit ähnlichen Symptomen (Ähnlichkeit der Problemstellung) durchgeführt wurde, auch bei einem anderen Patienten zum Erfolg (Nützlichkeit der Problemlösung). In diesem Kapitel werden Modellierungsmöglichkeiten des Ähnlichkeitsbegriffes anhand mathematischer Modelle sowie entsprechender Formalisierungen vorgestellt.

2.2.1 Zusammenhang von Ähnlichkeit, Nützlichkeit und Unsicherheit

Ein Fall ist zur Lösung eines aktuellen Problems umso nützlicher je weniger die bekannte Lösung dieses Falles für die aktuelle Problemstellung modifiziert werden muß. Kann die Nützlichkeit der Lösung eines Falles für die aktuelle Problemstellung exakt bestimmt werden, so erhält man gleichzeitig eine optimale Auswahlstrategie für Fälle. Dies ist möglich, wenn zuerst die Lösung des aktuellen Problems bestimmt wird, und anhand dieser Lösung ein Fall gewählt wird. Nur: wozu sollte es sinnvoll sein, einen Fall zu bestimmen, wenn die Lösung schon bekannt ist? Wozu sollte

ein Arzt einen Patienten mit ähnlichen Symptomen bestimmen, wenn der Kranke schon geheilt wurde? Beim fallbasierten Schließen soll eine Lösung für eine im allgemeinen noch nicht behandelte Problemstellung gefunden werden. Das Kriterium Nützlichkeit von Problemlösungen wird daher auf den Begriff der Ähnlichkeit von Problemstellungen reduziert. Diese Vorgehensweise liegt in der Hoffnung begründet, daß die Ähnlichkeit von Problemstellungen die Nützlichkeit für die Problemlösung impliziert [25]. Da auf der Basis von unsicheren und unvollständigen Informationen auf noch nicht bekannte Problemlösungen geschlossen werden soll, besteht eine Unsicherheit hinsichtlich der folgenden Fragen:

- Wann sind Probleme ähnlich?
- Wann sind Lösungen eines Falls für das aktuelle Problem nützlich?
- Impliziert die Ähnlichkeit von Problemstellungen die Nützlichkeit für die Problemlösung?

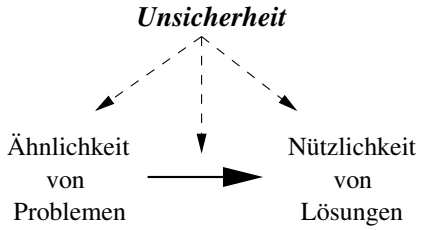


Abbildung 2.1: Nützlichkeit, Ähnlichkeit und Unsicherheit (Wess 1995)

Aus dem Alltag ist bekannt, daß man ein und dieselbe Situation verschieden interpretieren kann. So kommt es beispielsweise in einem Strafprozeß vor, daß der Staatsanwalt und der Verteidiger in Kenntnis derselben Fakten völlig unterschiedlich plädieren. Es kann also keineswegs aus der realen Ähnlichkeit zweier Situationen oder Probleme stets auf eine bestimmte Klassifikation oder Problemlösung geschlossen werden. Dies gilt auch in der Konstruktionswissenschaft, wo infolge des technischen Fortschrittes neue Problemlösungen zutage treten. Wenn man also beispielsweise im Automobilbau für eine bestimmte Problemlösung eine Vergangenheitslösung identifiziert, also z. B. einen VW-Käfer der 50er Jahre, so ist es regelmäßig unrealistisch zu glauben, diese Vergangenheitslösung sei einer Gegenwartslösung, also etwa dem Nachfolgemodell des VW-Käfers mit Baujahr 2001, vorzuziehen. Andererseits ist aber auch erkennbar, daß die Bauweise des VW-Käfers im Prinzip nicht veraltet ist, sonst gäbe es heute kein Nachfolgemodell.[2] In diesem Sinne läßt sich beobachten, daß in der Vergangenheit bekannte und erprobte Konstruktionsprinzipien eine Renaissance erleben. Dies gilt hierzulande derzeit auch für die Windenergie. Während Windmühlen im Mittelalter typischerweise auf dem Lande arbeiteten, sind in der jüngsten Zeit nicht nur Propeller für die Energiegewinnung auf dem Lande konstruiert worden. Vielmehr entstehen derzeit in der deutschen Nordsee wie auch in der deutschen Ostsee relativ große Windparks, die die Energie von mehreren hundert Rotoren zur Stromerzeugung bündeln.

Die Problematik der Nützlichkeitsvermutung bei ähnlicher Problemstellung hat an der Universität Kaiserslautern dazu geführt, daß man ein neues CBR-Modell verfolgt. Diesbezüglich zielen aktuelle Forschungsaktivitäten darauf ab, einen direkten Vergleich zwischen Anfragen und den dazu gesuchten Lösungen der Fallbasis zu etablieren [32]. Diese Intention bedeutet, daß die Dichotomie, wonach ein Fall eine Problembeschreibung und eine Problemlösung hat, aufgehoben wird.[3]

Zu erwähnen ist noch, daß bei falladaptierenden CBR-Systemen ein Ähnlichkeitsmaß regelmäßig nicht die Adaptionsproblematik berücksichtigt, weshalb hier beim Retrieval *Adaptionswissen* verarbeitet werden sollte (Adaptation Guided Retrieval, AGR) [33].

[2] Daran erkennt man, wie wichtig es ist, daß ein geeignetes Ähnlichkeitsmodell gefunden wird.

[3] Bergmann et al. (2001) erläutern ihre Konzeption mit den folgenden Worten: „In other words, one needs a similarity function $sim(p, s_i)$ which compares queries p and solutions s_i ... This model overcomes the assumption that cases consist of a problem and a solution description. In our extended view, only a solution description is requiered. From this solution description s_i we compute the so-called utility-description d_i, which means some kind of $utility\ reasoning$ has to be performed. Also, the description of the new problem p is transformed into a utility description d. During retrieval, the utility description d is compared to the utility descriptions d_i of the cases from the case base. The purpose of this similarity assessment, which is now on the level of utility reasoning, is to approximate the utility of a given solution for a new problem. In this model, the utility descriptions d_i of the individual cases can be part of the case description if they are static and do not depend on the problem itself, but they can also be computed during retrieval. In the traditional CBR view the utility description is the problem description of the case and the supplementary computation step is saved due to the availability of experiences.“ Bergmann et al. [32], p. 268f.

2.2.2 Formalisierungen des Ähnlichkeitsbegriffes

In diesem Abschnitt werden drei Möglichkeiten zur Formalisierung des Ähnlichkeitsbegriffes, d. h. wie eine Ähnlichkeit zwischen Objekten ausgedrückt werden kann, vorgestellt. Dabei sei das Universum $\mathcal{U}$ eine Menge aus Objekten, Situationen oder Problemen.

Ähnlichkeit als Prädikat

Der Ähnlichkeitsbegriff wird als ein Prädikat $SIM \subseteq \mathcal{U}^2$ wie folgt definiert:

$$SIM(x,y) = \begin{cases} 1 & \text{y ist x ähnlich} \\ 0 & \text{sonst} \end{cases}$$

Diese Formalisierung gibt an, ob eine Ähnlichkeit zwischen zwei Objekten vorhanden ist oder nicht. Dadurch kann eine Aussage, wie z. B. *Patient Reckmann und Tellmann haben ähnliche Symptome* (Formal: $SIM(\ Reckmann, Tellmann) = 1$), charakterisiert werden. Für $SIM(x,y)$ wird Reflexivität (Objekte sind zu sich selbst ähnlich) und Symmetrie (Wenn Objekt x zu y ähnlich ist, so ist auch Objekt y zu x ähnlich) gefordert.

Auswahl von Fällen Falls $\exists\ y \in CB : SIM(x,y)$ gilt, d. h. falls in der Fallbasis CB (Case Base) ein zu Objekt x ähnliches Objekt y gefunden werden kann, erhält man somit eine übertragbare Lösung für ein Objekt $x \in \mathcal{U}$. Dabei können mehrere Objekte y zu x ähnlich sein. In diesem Fall muß eine zusätzliche Konfliktlösungsstrategie die Auswahl eines geeigneten Falls y zur Lösung von x gewährleisten.

Ähnlichkeit als Präferenzrelation

Durch die Angabe einer Präferenz- oder Ordnungsrelation kann ein mehr oder weniger an Ähnlichkeit ausgedrückt werden [35]. Dadurch kann eine Aussage, wie etwa *Die Symptome des Patienten Reckmann stimmen mehr mit denen von Patient Rumpke als mit denen von Patienten Tellmann überein* (Formal: $Rumpke \succeq_{Reckmann} Tellmann$), bestimmt werden. Hierzu definiert man eine Basisrelation $Prä \subseteq \mathcal{U}^4$ für die gilt:

$$Prä(x,y,u,v) \Longleftrightarrow x \text{ ist mindestens so ähnlich zu } y \text{ wie } u \text{ zu } v.$$

Forderungen

$Prä(x,x,u,v)$	maximales Element
$Prä(x,y,u,v) \Leftrightarrow Prä(y,x,u,v) \Leftrightarrow Prä(x,y,v,u)$	Symmetrie in den Argumenten
$Prä(x,y,x,y)$	Reflexivität
$Prä(x,y,u,v) \wedge Prä(u,v,s,t) \Rightarrow Prä(x,y,s,t)$	Transitivität
$Prä(x,y,u,v) \wedge Prä(u,v,x,y) \Rightarrow (x,y) \sim (u,v)$	Antisymmetrie

Das Symbol $\sim$ bezeichnet hier eine Äquivalenzrelation.

Die Relation $Prä$ induziert eine Partialordung $\succeq$ über den Tupeln aus $\mathcal{U}$:

$$Prä(x,y,u,v) \Longleftrightarrow (x,y) \succeq (u,v).$$

Zur Vereinfachung wird eine weitere dreistellige Ordnungsrelation $S \subseteq \mathcal{U}^3$ definiert:

$$S(x,y,z) \Longleftrightarrow Prä(x,y,x,z) \Longleftrightarrow y \succeq_x z.$$

Die von S induzierte Ordnung bezieht sich im Gegensatz zu $Prä$ nicht auf Tupeln von $\mathcal{U}^2$, sondern auf die Elemente von $\mathcal{U}$.

Auswahl des ähnlichsten Falles Für ein Objekt $x \in \mathcal{U}$ kann eine geeignete Lösung durch Verwendung der maximalen Elemente von $\succeq_x$ bzgl. $\mathcal{U}$ bestimmt werden. Es wird also ein Objekt y gesucht, das zu x ähnlicher ist als alle anderen Objekte.

Definition 2.1 (Maximales Element) *Das Element y ist ein maximales Element einer beliebigen Menge $\mathcal{M}$ bezüglich der Relation $\succeq_x$, falls eine Teilmenge $M_i \subseteq \mathcal{M}$ existiert, so daß $\forall z \in M_i : y \succeq_x z$ gilt und kein weiteres Element $u \in \mathcal{M}$ existiert, welches bezüglich Relation $\succeq_x$ größer ist als Element y: $\nexists u \in M : u \succeq_x y$.*

Die zur Lösung von $x \in \mathcal{U}$ geeigneten Objekte aus der Fallbasis CB bezüglich der Relation $\succeq_x$ werden durch folgende Menge angegeben:

$$L_x = \{y \in CB \mid y \text{ ist maximal in } CB \text{ bezüglich } \succeq_x\}.$$

Enthält die Menge mehr als ein Element, also mehrere Lösungen für x, so ist wie bei SIM eine separate Konfliktlösungsstrategie anzuwenden.

Beispiel Seien die Objekte a, b, c,d, $e \in CB$ gegeben. Es soll eine Lösung für ein Objekt $x \in \mathcal{U}$ gefunden werden. Dabei wurden folgende Ähnlichkeiten zwischen dem Objekt x und den Objekten aus der Fallbasis CB festgestellt:

(i) $a \succeq_x b \succeq_x c$

(ii) $d \succeq_x e$

Da (i) und (ii) voneinander unabhängig sind, enthält die Menge L_x mehrere Elemente: $L_x = \{a, d\}$. Auf L_x ist eine separate Konfliktlösungsstrategie anzuwenden.

Ähnlichkeit als Maß

Ein Ähnlichkeitsmaß wird definiert als eine Funktion $sim(x, y) : \mathcal{U}^2 \rightarrow [0, 1]$. Dadurch kann eine Aussage, wie z.B. *Die Symptome der Patienten Reckmann und Tellmann stimmen zu 80 Prozent* (Formal: $sim(Reckmann, Tellmann) = 0.8$) *überein*, abgeleitet werden. Dieses Maß liefert neben der Ordnungsinformation auch eine Information über das Ausmaß der entsprechenden Ähnlichkeit. Für $sim(x, y)$ wird Reflexivität und Symmetrie gefordert.

Fallauswahl Für eine Lösung von $x \in \mathcal{U}$ wird ein $y \in \mathcal{U}$ mit maximaler Ähnlichkeit gesucht: $\exists y \in CB : \forall z \in CB : sim(x, y) \geq sim(x, z)$? Das Ähnlichkeitsmaß kann in einem geometrisch motivierten Modell auch als ein äquivalentes Distanzmaß formuliert werden.

Definition 2.2 (Distanzmaß) *Sei $\mathcal{U}$ die Menge aller Punkte des $\mathbb{R}^n$. Ein Distanzmaß ist eine Funktion $d(x, y) : \mathcal{U}^2 \rightarrow \mathbb{R}_0^+$ die folgende Axiome erfüllt:*

(i) Reflexivität: $d(x, x) = 0$;

(ii) Symmetrie: $d(x, y) = d(y, x)$.

Das Distanzmaß kann gleichwertig zum Ähnlichkeitsmaß verwendet werden, falls beide Maße kompatibel sind.

Definition 2.3 (Kompatibilität) *Das Ähnlichkeitsmaß $sim(x, y)$ und das Distanzmaß $d(x, y)$ heißen kompatibel, falls gilt:*

$$d(x, y) \leq d(u, v) \Longleftrightarrow sim(x, y) \geq sim(u, v).$$

Dieser Zusammenhang bedeutet umgangsprachlich: Wenn zwei Objekte x und y ähnlicher zueinander sind als zwei Objekte u und v, so liegen x und y im geometrischen Sinn näher beieinander als u und v. Auch die Umkehrung gilt, d. h. wenn die Entfernung zwischen x und y geringer ist als die von u und v, so sind sich x und y ähnlicher als u und v [29].

2.2.3 Zusammenhänge zwischen den Ansätzen

Aus einem gegebenen Ähnlichkeitsmaß kann die Präferenzinformation $\succeq_x$ und ein Prädikat SIM, z.B. mit Hilfe eines Schwellenwertes, extrahiert werden. Weiterhin ist die Umwandlung einer gegebenen Präferenzrelation S in ein entsprechendes Prädikat SIM möglich. Bei diesen Transformationen kommt es allerdings zu Informationsverlusten bezüglich der Semantik der Ähnlichkeit und / oder der vorhandenen Ordnungsinformation.

Beispiel Zur Lösung des Problems x liegen folgende kardinale Ähnlichkeiten (Ähnlichkeiten als Maß) vor: $sim(x, r) = 0.4$, $sim(x, c) = 0.8$, $sim(x, b) = 0.7$.

- Ähnlichkeit als Maß → Ähnlichkeit als Präferenzrelation
  Aus obigen Informationen läßt sich folgende Ordnung ableiten beziehungsweise extrahieren: $c \succ_x b \succ_x r$, d. h. in der Reihenfolge c, b, r nimmt die Ähnlichkeit zu x ab.
- Ähnlichkeit als Maß → Ähnlichkeit als Prädikat
  Weiterhin läßt sich aus den kardinalen Informationen ein binäre Ähnlichkeit (Ähnlichkeit als Prädikat) bestimmen. Dabei sei ein Schwellenwert von $\delta = 0.65$ angenommen. Für die binäre Entscheidung kommen also die Objekte c und b in Frage, wobei das ähnlichste Objekt c gewählt wird. Falls der Schwellenwert $\delta > 0.8$ wäre, würde kein ähnlicher Fall gefunden werden.
- Ähnlichkeit als Präferenzrelation → Ähnlichkeit als Prädikat
  Mit Hilfe der Präferenzrelation $(c \succ_x b \succ_x r)$ wird für die binäre Ähnlichkeit das zu x ähnlichste Objekt, also c, gewählt.

Binäre, ordinale und kardinale Ähnlichkeitsinformation

Im vorherigen Abschnitt wurden unterschiedliche Ähnlichkeitsrelationen vorgestellt. Für die Realisierung des fallbasierten Ansatzes wird der ähnlichste Fall benötigt:

$$SIM(x, y) \Longleftrightarrow \forall z \in CB : \; y \succeq_x z : y \text{ ist am ähnlichsten zu } x.$$

Der ähnlichste Fall kann mit Hilfe der anderen Relationen bestimmt werden. Im weiteren wird der Informationsgehalt der verschiedenen Ansätze diskutiert.

- **Binäre Ähnlichkeitsskala:** y ist ähnlich zu x
  Die vorhandene Information zur Bestimmung des ähnlichsten Falles muß zu einer binären Entscheidung $SIM(x, y)$ komprimiert werden, d. h. nach Beendigung der Suche nach ähnlichen Fällen wird nur das ähnlichste Beispiel benötigt. Die Semantik von SIM wird durch die Semantik des entsprechenden Problemlöseprozesses bzw. durch die entsprechende Definition von SIM bestimmt. SIM könnte beispielsweise zur Einteilung (Klassifikation) von Fallbeispielen einer Fallbasis in ähnliche und nicht ähnliche Fallbeispiele bezüglich eines bestimmten Objektes dienen.
- **Ordinale Ähnlichkeitsskala:** y ist ähnlicher zu x als z zu x
  Hier sind die Fälle bezüglich ihrer Ähnlichkeit geordnet. Die ordinale Bewertung ist der Kern von fallbasierten Ansätzen, da ein mehr oder weniger an Ähnlichkeit von entscheidender Bedeutung ist. Die Semantik, d. h. die Aussage von $y \succeq_x z$, hängt von den konkret verfolgten Zielen ab; z. B. könnten dies Kostengrößen oder auch leichter zur Lösung zu modifizierende Fälle sein. Da bei der Verwendung des Prädikates SIM zur Auswahl von Fallbeispielen meistens mehrere Objekte zu einem Objekt $x \in \mathcal{U}$ ähnlich sind, ist die Präferenzrelation $\succeq_x$ eine separate Konfliktlösungsstrategie, um einen geeigneten Fall zur Lösung zu finden.
- **Kardinale Ähnlichkeitsskala:** Die Ähnlichkeit von x und y hat den Wert Z
  Die kardinale Bewertung ist eine Erweiterung der ordinalen Ähnlichkeitsskala um Informationen über die Eignung von Fällen zur Lösung des aktuellen Problems. Um diese zusätzlichen Informationen zu nutzen, muß die Semantik für die kardinale Information genau für die speziell zu realisierende Anwendung festgelegt werden. Eine Interpretation von sim kann z. B. anfallender Modifikationsaufwand oder geringe Produktionskosten sein. Der nach einer Auswertung von $\succeq_x$ offensichtlich am besten geeignete Fall zur Lösung von x kann jedoch in Wirklichkeit vollkommen ungeeignet sein. Eine derartige Fehlauswahl kann durch die Angabe eines Schwellenwertes mit $sim(x, y) \geq \delta$ vermieden werden.

2.2.4 Modelle zur Ähnlichkeitsbestimmung

In diesem Abschnitt geht es um die Frage, wie für eine konkrete Anwendung ein geeignetes Verfahren zur Auswahl von Fällen bestimmt werden kann. Dieses Problem ist allerdings noch als Forschungsgegenstand zu betrachten. Im folgenden werden einige unterschiedliche Klassen von Modellen, die als Ansätze zur Ähnlichkeitsbestimmung dienen, vorgestellt.

Geometrischer Abstand In diesem Ansatz werden die Fälle als Punkte in einem mehrdimensionalen Raum interpretiert. Dabei wird dann die Ähnlichkeit zwischen Fällen durch die räumliche Nähe zweier Punkte bestimmt. Die Ähnlichkeitsbestimmung von Fällen reduziert sich auf das geometrische Nearest-Neighbor Problem.

Vergleich von Feature Sets Ein Objekt wird über verschiedene Merkmale (Features) beschrieben, die bezüglich ihrer Bedeutung für ein Problem gewichtet werden. Die Ähnlichkeit zwischen zwei Objekten wird durch eine Gegenüberstellung der gewichteten Merkmale ausgedrückt. Dabei erhöhen gleiche und verringern unterschiedliche Merksmalwerte die Ähnlichkeit von Objekten.

Beispiel Ein Arzt vergleicht die Krankheitsbilder zweier Patienten A und B, bezüglich der Diagnose „Erkältung“ bei Patient A. Die Gewichte der berücksichtigten Merkmale hat der Arzt in Abhängigkeit von dem betrachteten

Merkmale	Gewicht	Patient A	Patient B	Ähnlichkeit
Husten	0.6	stark	sehr stark	0.7
Fieber	0.3	leicht	leicht	1
Bauchschmerzen	0.1	leicht	keine	0

Abbildung 2.2: Krankheitsbilder zweier Patienten

Problem (Erkältung) festgelegt. Für die Bestimmung der Ähnlichkeit lagen für jedes Merkmal entsprechend der Tabelle verschiedene Ähnlichkeitsmaße zugrunde. Die Krankheitsbilder der Patienten stimmen zu $sim(A, B) = 0.6 * 0.7 + 0.3 * 1 + 0.1 * 0 = 0.72$ überein.

Transformation von relaxierten Repräsentationen Bei adaptierenden CBR-Systemen wird ein Fall für die Lösung eines neuen Problems gesucht, dessen Lösung möglichst wenig für die aktuelle Problemlösung modifiziert werden muß, d. h. der Modifikationsaufwand für das Übertragen der Lösung des ähnlichsten Falles auf das aktuelle Problem soll möglichst gering sein. Für die exakte Bestimmung des minimalen Modifikations- bzw. Transformationsaufwands müßte daher eigentlich eine Lösung des neuen Problems a-Priori schon bekannt sein. Dies ist aus Sicht des fallbasierten Schließens aber zumeist nicht annehmbar, da ja gerade eine Lösung für ein neues Problem konstruiert werden soll. Mit Hilfe eines relaxierten Modells versucht man dieses Dilemma zu umgehen, indem die Ähnlichkeitsbewertung durch die Bestimmung einer geeigneten Transformation vorgenommen wird, ohne daß dabei die volle Komplexität des Problems erreicht wird.

Das Adaptionsproblem ist zum einen darin zu sehen, daß gewöhnliche Ähnlichkeitsmaße mitunter schlechte Schätzungen für die erforderliche Adaption liefern, und zum anderen mitunter Fälle auswählen, die sehr schwierig zu adaptieren sind, obwohl die Fallbasis bezüglich der aktuellen Anfrage mindestens einen Fall aufweist, der eine einfache Adaption ermöglicht. Man will also vermeiden, daß ein CBR-System keine Lösung findet, weil es für einen zur Anfrage ähnlichen Fall keine Adaption vornehmen kann, und daß ein falladaptierendes System geeignete Fälle übersieht [36].

Beispiel Ein fallbasiertes System soll für ein vorgegebenes Werkstück einen Fertigungsplan generieren. Dabei ist eine Fallbasis mit unterschiedlichen Werkstücken und den entsprechenden Fertigungsplänen gegeben. Ein Werkstück wird durch eine relaxierte Baumdarstellung repräsentiert, d. h. es werden z. B. aus Komplexitätsgründen nicht alle Merkmale des Werkstückes berücksichtigt. Zur Ähnlichkeitsbestimmung wird der relaxierte Baum durch eine minimale Anzahl von Transformationsschritten in einen Fall übergeführt. Als Lösung des aktuellen Problems wird schließlich der Fall ausgewählt, bei dem der geringste Transformationsaufwand angefallen ist. Nach dieser Methodik arbeitet das System CAPLAN [17].

Erklärungsprozeß Bei einem Erklärungsprozeß wird, wie beim vorherigen Modell, eine Transformation eines Falles auf die aktuelle Problemstellung vorgenommen, um so den Modifikationsaufwand abzuschätzen. Während das Modell aus dem vorhergehenden Abschnitt einfache Repräsentationen wie Strings, Bäume oder Graphen zum Fallvergleich nutzt, stehen bei den erklärungsorientierten Ansätzen entsprechende wissensbasierte Methoden, also etwa Regelsysteme, zur Verfügung.

Abstraktionsprozeß Die Ähnlichkeit von Fällen wird in einem abstraktionsorientierten Ansatz ausschließlich durch die Wahl einer Abstraktionsabbildung bestimmt. Falls zwei Probleme auf der durch die Abstraktionsabbildung induzierten Abstraktionsebene eine gemeinsame abstrakte Problemstellung bzw. Lösung besitzen, so werden diese als ähnlich bezeichnet. Sind die wesentlichen Punkte der Lösung des Falles auf dieser abstrakten Ebene bekannt, so erhält man eine gute Abschätzung der bei einer Lösungstransformation zu erwartenden Modifikationsschritte. Die Ähnlichkeitsbestimmung durch einen Abstraktionsprozeß kommt häufig in technischen Domänen zum Einsatz, da dort beim Aufbau und der Konstruktion von Systemen am Anfang primitive Strukturen stehen, die in komplexe Strukturen überführt werden.

Beispiel Problem: Vergleich zweier elektrischer Schaltpläne auf Ähnlichkeit. Ein Schaltplan besteht aus Grundbauteilen wie Widerständen, Kondensatoren, Spulen usw. Mit Hilfe einer Abstraktionsabbildung werden diese Grundelemente auf eine höhere Ebene abgebildet, d. h. die einzelnen Bauteile werden zu Komponenten wie Dioden, Transistoren, Verstärker usw. zusammengefaßt. Sind die Schaltpläne auf dieser Ebene der Komponenten gleich, so werden diese als ähnlich bezeichnet.

Exkurs: Begriffliche Distanz zwischen Merkmalen

Im vielen praktischen Anwendungen sind oft keine keine Zahlen als Merkmalwerte gegeben, sondern symbolische Werte wie z. B. „leicht“ oder „sehr stark“. Weiterhin ist es möglich, daß Merkmale unterschiedliche Typen (polymorphe Merkmale) haben, z. B. könnte das Merkmal *Fieber* durch Werte wie *39 Grad*, *leicht* oder *ungefähr 40 Grad* beschrieben werden. In diesem Exkurs wird eine Möglichkeit vorgestellt, solche Werte zu verarbeiten.

Wenn ein Fall mit den Fällen in der Fallbasis verglichen werden soll, werden die einzelnen Merkmale miteinander verglichen und es wird eine begriffliche Distanz zwischen ihnen bestimmt. Wenn man diese einzelnen Vergleiche heranzieht, um sie zu einem Gesamtwert Δ zusammenzufassen, geschieht es zwangsläufig, daß man Werte addieren muß, die nicht miteinander vereinbar sind, z. B. $\Delta = f(d_{Blutdruck} = 35 + d_{Alter} = 12)$. Die erste Möglichkeit, dieses Problem zu beseitigen wäre, die Merkmale zu normalisieren, sie also ohne Angabe von Einhaiten auf ein Intervall $[0, 1]$ abzubilden. Bei der Normalisierung benötigt man jedoch sämtliche Merkmalswerte aller Fälle. Dies bedeutet, daß die begriffliche Distanz zweier Merkmale vom jeweiligen Stand der Fallbasis abhängt. Dieser ändert sich jedoch. Da eine solche Situation unpraktisch ist, versucht man hier einen anderen Weg zu gehen.

C4-Architektur Bei der C4-Wissensarchitektur handelt es sich um eine Fallspeicherorganisation sowie ein Klassifikationssystem, das die begrifflich verwandten Fälle in *fuzzy* (unscharfe) Kategorien einteilt.

- Fuzzy-Logik
  Ein betrachtetes Element eines Grundbereichs X kann einer Menge A ganz oder auch nur zu einem gewissen Grad angehören. Die Zugehörigkeit eines Elementes $x \in X$ zur Menge A kann durch eine charakteristische Funktion μ_A auf folgende Weise beschrieben werden:

  $$\mu_A : X \rightarrow \{0, 1\}$$
  $$\mu_A(x) = \begin{cases} 1 & x \in A \\ 0 & \text{sonst} \end{cases}$$

  Diesen klassischen Mengen (scharfe Mengen) sind also solche charakteristischen Funktionen zugeordnet, die nur zwei Werte annehmen können. Eben diese Zweiwertigkeit soll aufgehoben werden, indem der Wertevorrat der charakteristischen Funktion μ_A auf alle reellen Zahlen im Intervall [0,1] erweitert wird. Die charakteristische Funktion μ_A wird dann als Zugehörigkeitsfunktion bezeichnet und als quantitatives Maß interpretiert, ein Maß dafür, inwieweit ein Element die Eigenschaften einer nun unscharfen Menge A erfüllt [24].

- Polymorphe Merkmale
  Die Fuzzymengen-Theorie stellt einen Rahmen bereit, der eine numerische sowie auch eine symbolische Verarbeitung ermöglicht. Ein Merkmal kann also reelle Zahlen, normalisierte Fuzzywerte und linguistische Ausdrücke als Werte annehmen. Ein Beispiel hierzu wäre, daß ein Merkmal *Alter* die Werte 73, *ungefähr 70* und *alt* annehmen kann. Jedes polymorphe Merkmal wird jetzt in Verbindung zu einem Konzeptrahmen gebracht.

  Ein Konzeptrahmen entsprechend Abbildung 2.3 wird durch eine Menge von Konzepten dargestellt, wobei jedes Konzept durch einen linguistischen Ausdruck und eine Fuzzymenge beschrieben ist. In der obigen Graphik

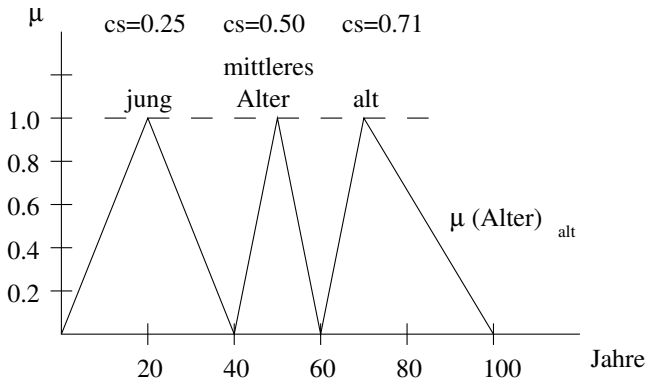


Abbildung 2.3: Beispiel eines Konzeptrahmens *Alter*

bezieht sich der linguistische Ausdruck auf den Namen des Konzepts (alt), und die Fuzzymenge, die durch eine Funktion $\mu(Alter)_{alt}$ dargestellt wird, repräsentiert die Bedeutung des Konzepts. Zusätzlich existiert noch zu jedem Konzept ein *crisp score (cs)*, an dem man ablesen kann, wie nah ein Konzept an einem anderen liegt. In obiger Abbildung 2.3 liegt z. B. das Konzept *alt* näher am Konzept *mittleres Alter* ($\Delta cs = 0.21$) als am Konzept *jung* ($\Delta cs = 0.25$). Die Konzeptrahmen erlauben bestimmte Transformationen, unter anderem Abbildungen eines Merkmalformats in ein anderes, z. B. *ungefähr 45* $\rightarrow$ mittleres Alter [28].

Berechnung der *Crisp Scores* Für die Berechnung der *Crisp Scores* gibt es mehrere Verfahren, die in [19] vorgestellt werden. Nachfolgend wird das Verfahren von S.-J. Chen, C.-L. Hwang und F. P. Hwang beschrieben. Zuerst sind zwei Funktionen, das unscharfe Maximum und und das unscharfe Minimum, zu definieren:

$$\mu_{max}(x) = \begin{cases} x, & 0 \leq x \leq 1 \\ 0, & \text{sonst} \end{cases}$$
$$\mu_{min}(x) = \begin{cases} 1-x, & 0 \leq x \leq 1 \\ 0, & \text{sonst} \end{cases}$$

Danach wird für ein Konzept M ein *right score* berechnet:

$$\mu_R(M) = \sup_x \min[\mu_M(x), \mu_{max}(x)].$$

Diese Formel berechnet den maximalen Wert des Schnitts zwischen dem Konzept M und dem des unscharfen Maximums. Auf ganz ähnliche Weise wird der *left score* berechnet:

$$\mu_L(M) = \sup_x \min[\mu_M(x), \mu_{min}(x)].$$

Diese Formel berechnet den maximalen Wert des Schnitts zwischen dem Konzept M und dem des unscharfen Minimums. Mit diesen beiden Werten kann der cs von M berechnet werden:

$$cs(M) = \frac{\mu_R(M) + 1 - \mu_L(M)}{2}.$$

Beispiel Seien zwei Konzepte M_1 und M_2 entsprechend Abbildung 2.4 gegeben. Dann sind die *Crisp Scores* wie folgt berechnen. Das unscharfe Maximun und das unscharfe Minimun sind wie folgt definiert:

$$\mu_{max}(x) = \begin{cases} x, & 0 \leq x \leq 1 \\ 0, & \text{sonst} \end{cases}$$
$$\mu_{min}(x) = \begin{cases} 1-x, & 0 \leq x \leq 1 \\ 0, & \text{sonst} \end{cases}$$

Außerdem sind die charakteristischen Funktionen von M_1 und M_2 bekannt:

$$\mu_{M_1}(x) = \begin{cases} \frac{x-0.4}{0.2}, & 0.4 \leq x \leq 0.6 \\ \frac{0.8-x}{0.2}, & 0.6 \leq x \leq 0.8 \end{cases}$$
$$\mu_{M_2}(x) = \begin{cases} \frac{x-0.6}{0.2}, & 0.6 \leq x \leq 0.8 \\ \frac{1-x}{0.2}, & 0.8 \leq x \leq 1 \end{cases}$$

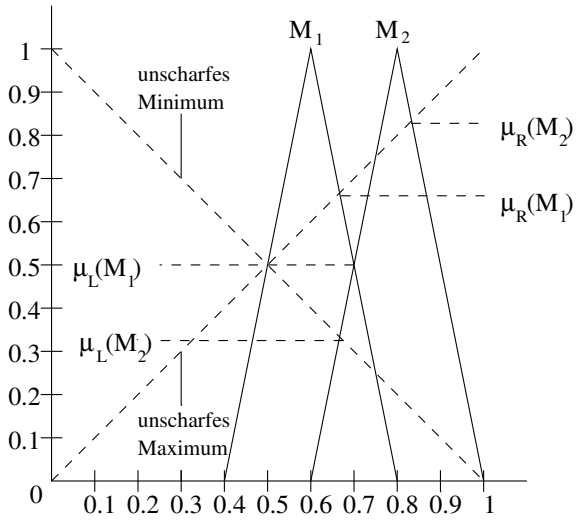


Abbildung 2.4: Der *left score* und *right score* von M_1 und M_2

i	$\mu_R(M_i)$	$\mu_L(M_i)$	$cs(M_i)$
1	0.667	0.500	0.584
2	0.833	0.333	0.750

Abbildung 2.5: Wertetabelle der Scores von M_1 und M_2

Für M_1 und M_2 sind der *right score*, *left score* und *crisp score* in obiger Wertetabelle dargestellt: Beispielsweise wird der cs von M_1 folgendermaßen berechnet:

$$\begin{aligned} \mu_R(M_1) &= \sup_x \min[\mu_{max}(x), \mu_{M_1}(x)] = 0.667, \\ \mu_L(M_1) &= \sup_x \min[\mu_{min}(x), \mu_{M_1}(x)] = 0.500, \\ cs(M_1) &= \tfrac{[\mu_R(M_1)+1-\mu_L(M_1)]}{2} = 0.584. \end{aligned}$$

- Vergleich polymorpher Merkmale
  Um die begriffliche Distanz zwischen zwei Fällen zu bestimmen, sammeln komplexe[4] polymorphe Merkmale die einzelnen Distanzwerte der wichtigsten Merkmale. Hierbei tritt jedoch das Problem auf, daß man unterschiedlich kodierte Untermerkmale, d.h. z.B. einen linguistischen Wert und eine Zahl, systematisch zu einem vernünftigen Wert verbinden muß.

  Beispiel Ein komplexes abstraktes Merkmal A ist durch die Merkmale

  $$B = \{Cholesterin, Blutdruck, Alter\}$$

  gegeben.

  Anfrage: $q =$ (hoch, sehr niedrig, 63)

  Fallbasis: $b_1 =$ (6.3, ungefähr 130, mittleres Alter)
  $b_2 =$ (niedrig, niedrig, 71)

  Um festzustellen, welches b näher am q liegt, müssen die wichtigsten Merkmale aus B verglichen und die resultierenden begrifflichen Distanzen zu einem Gesamtmaß verbunden werden. Dabei können 6 verschiedene Kombinationen auftreten: (RN,RN), (LT,LT), (NFN,NFN), (RN,LT), (RN,NFN) und (LT,NFN)[5]. Im Beispiel bezieht sich dies auf die Vergleiche (hoch, 6.3), (sehr niedrig, (ungefähr 130), (63, mittleres Alter), (hoch, niedrig), (sehr niedrig, niedrig) und (63, 71). Bis auf (RN,RN) werden alle Kombinationen in *crisp scores* umgewandelt

[4] Zusammenfassung mehrerer Merkmale
[5] RN = reelle Zahl, LT = linguistischer Term, NFN = normalisierte Fuzzy Zahl

und dann verglichen. Der Vorteil der *crisp scores* liegt darin, daß man auf konventionelle Vergleichsmethoden zurückgreifen kann. Nur im Fall eines Vergleichs zweier reeller Zahlen benutzt der C4 Algorithmus die Normalisierung.

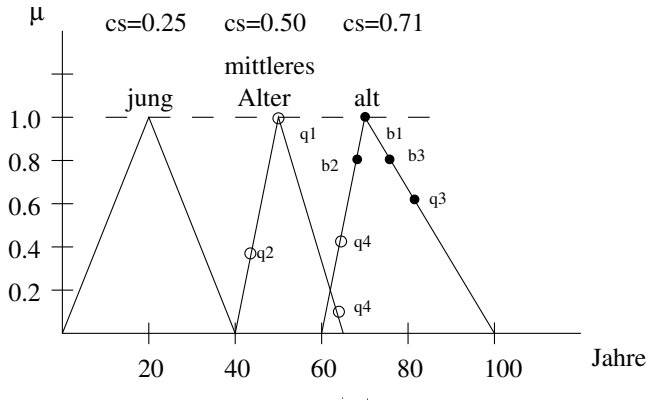


Abbildung 2.6: Konzepte des Merkmals *Alter*

Begriffliche Distanz zwischen Merkmalen In Abbildung 2.6 wird die Bedeutung der Konzepte *jung*, *mittleres Alter* und *alt* durch die dazugehörigen Fuzzymengen und die Zugehörigkeitsfunktion μ ausgedrückt. Einen grundlegenden Unterschied zwischen diesen Konzepten kann man durch eine minimale begriffliche Distanz δ, definiert durch $\delta(a,b)_{min} = |cs_a - cs_b|$, beschreiben. Die minimale begriffliche Distanz ist hierbei also ein Resultat der Form der jeweiligen Zugehörigkeitsfunktion und deren Anordnung im Rahmen. Dies liegt daran, daß die *crisp scores* aus der Zugehörigkeitsfunktion berechnet werden. Basierend auf der minimalen begrifflichen Distanz wird eine Strategie entwickelt, um zwei abstrakte Merkmale begrifflich miteinander zu vergleichen. b_1 und b_2 listen die einzelnen Merkmale der Fälle, die in der Fallbasis enthalten sind. Die q's sind Merkmale der Anfrage (query). Möchte man die begriffliche Distanz zwischen einem Merkmal q_j der Anfrage und einem b_i bestimmen, können vier verschiedene Möglichkeiten auftreten:

- b_i und q_j werden eindeutig auf unterschiedliche Zugehörigkeitsfunktionen abgebildet:
  Nehmen wir als Beispiel die Werte $q_1 = 50$ und $b_1 = 70$. Da beide dem jeweiligen Konzept mit einem Grad 1 angehören, läßt sich die begriffliche Distanz leicht durch die Berechnung δ(mittleres Alter, alt) = 0.21) bestimmen. Was ist jedoch, wenn man die Werte $q_2 = 40$ und $b_2 = 68$ betrachtet? Es ist logisch, daß die Distanz größer werden muß, da die Zugehörigkeit zum jeweiligen Konzept gesunken ist. Dies läßt sich sehr leicht einsehen, wenn man den Extremfall betrachtet, in dem der Zugehörigkeitswert zu beiden Konzepten gleich 0 ist. Dann ist die Distanz zwischen diesen beiden Konzepten am höchsten, d. h. $\delta = 1$. Es wird definiert:

$$\begin{aligned} \delta &= (\delta(a,b)_{min} - 1) * \Delta p + 1 \\ \Delta p &= (\mu(x_1)_a + \mu(x_2)_b)/2 \end{aligned}$$

- b_i und q_j werden eindeutig auf dieselbe Zugehörigkeitsfunktion abgebildet:
  Dies ist ein Spezialfall der ersten Möglichkeit. Da die begriffliche Distanz $\delta(a,b)_{min} = 0$ ist, erhalten wir: $\delta = 1 - \Delta p$

- Ein Wert wird eindeutig auf eine Zugehörigkeitsfunktion abgebildet, während der andere auf zwei oder mehrere Zugehörigkeitsfunktionen abgebildet wird:
  Die partiellen begrifflichen Distanzen erhält man hier aus den Paaren $A \times B$. Also

$$\phi_1 = \delta(a_1, b_1), \ldots, \phi_n = \delta(a_{n_1}, b_{n_1}).$$

  Die begriffliche Distanz ergibt sich dann folgendermaßen:

$$\delta = (1/n) * \sum_{i=1,\ldots,n} \phi_i$$

- Beide Werte werden auf eine oder mehrere Zugehörigkeitsfunktionen abgebildet:
  Dies ist ein Spezialfall der dritten Möglichkeit.

2.2.5 Vergleich von unterschiedlichen Ansätzen

Im letzten Abschnitt wurden unterschiedliche Modelle vorgestellt, die ein Verfahren angeben, wie formale Ähnlichkeitsbegriffe realisiert bzw. implementiert werden können. Zur Auswahl eines geeigneten Modells müssen diese Modelle miteinander verglichen werden. In diesem Abschnitt werden Methoden zum Vergleich unterschiedlicher Ansätze vorgestellt.

Formaler Vergleich von Ähnlichkeitsbewertungen

In diesem Abschnitt sollen die Zusammenhänge zwischen dem in der Anwendung tatsächlich existierenden und dem im System repräsentierten Ähnlichkeitsbegriff aufgezeigt werden. Dabei werden zwei Ebenen von Ähnlichkeitsbewertungen unterschieden:

Ähnlichkeit auf der Repräsentationsebene Ähnlichkeit auf der Repräsentationsebene Sim_{Rep} bezeichnet den auf der Systemebene realisierten Ähnlichkeitsbegriff. In der Auswahlphase wird ausschließlich dieser Ähnlichkeitsbegriff verwendet.

Ähnlichkeit auf der Anwendungsebene Ähnlichkeit auf der Anwendungsebene Sim_{App} bezeichnet die in der Anwendung tatsächlich vorhandene Ähnlichkeit, d. h. diese gibt an, inwieweit gefundene Fälle a-Priori nützlich für das aktuelle Problem sind. In einem fallbasierten System muß versucht werden, einen möglichst nahe an der tatsächlichen Ähnlichkeit liegenden repräsentierten Ähnlichkeitsbegriff zu finden. Diese Annäherung ist auch ein Teil des Lernprozesses in fallbasierten Systemen. Im folgenden werden formale Kriterien vorgestellt, die bestimmen, inwieweit der im System repräsentierte und der in der Anwendung vorhandene Ähnlichkeitsbegriff übereinstimmen.

Definition 2.4 (Übereinstimmung) *Um die Übereinstimmung zweier Ähnlichkeitsbegriffe Sim_1 und Sim_2 zu bewerten, werden die von den entsprechenden Relationen $Prä_1$ und $Prä_2$ induzierten Ordnungen $\succ^1$, $\succ^2$ auf $\mathcal{U}$ verglichen:*

- *Sim_1 und Sim_2* stimmen partiell überein*, falls $\forall a, b, c \in \mathcal{U} : a \succ_c^1 b \Rightarrow a \succ_c^2 b$.*
- *Sim_1 und Sim_2 stimmen überein, falls $\forall a, b, c \in \mathcal{U} : a \succ_c^1 b \Leftrightarrow a \succ_c^2 b$.*

Mit Hilfe dieser Begriffsbildungen können die Begriffe Korrektheit *und* Vollständigkeit *des repräsentierten Ähnlichkeitsbegriffes definiert werden.*

Definition 2.5 (Korrektheit) *Der repräsentierte Ähnlichkeitsbegriff Sim_{Rep} heißt* korrekt *bezüglich des Ähnlichkeitsbegriffes der Anwendung Sim_{App}, falls Sim_{Rep} und Sim_{App}* partiell übereinstimmen*, d. h. $\forall a, b, c \in \mathcal{U} : a \succ_c^{Rep} b \Rightarrow a \succ_c^{App} b$. Dies bedeutet, daß eine mit dem repräsentierten Ähnlichkeitsbegriff erfaßte Ähnlichkeit auch durch den tatsächlichen Ähnlichkeitsbegriff erfaßt wurde.*

Definition 2.6 (Vollständigkeit) *Der repräsentierte Ähnlichkeitsbegriff Sim_{Rep} heißt* vollständig *bezüglich des Ähnlichkeitsbegriffes der Anwendung Sim_{App}, falls Sim_{App} und Sim_{Rep}* partiell übereinstimmen*, d. h. $\forall a, b, c \in \mathcal{U} : a \succ_c^{App} b \Rightarrow a \succ_c^{Rep} b$. Dies bedeutet, daß eine mit dem tatsächlichen Ähnlichkeitsbegriff erfaßte Ähnlichkeit auch durch den repräsentierten Ähnlichkeitsbegriff erfaßt wurde.*

Definition 2.7 (Korrektheit und Vollständigkeit) *Der repräsentierte Ähnlichkeitsbegriff Sim_{Rep} heißt* korrekt *und* vollständig *bezüglich des Ähnlichkeitsbegriffes der Anwendung Sim_{App}, falls Sim_{Rep} und Sim_{App} übereinstimmen, d. h. $\forall a, b, c \in \mathcal{U} : a \succ_c^{Rep} b \Leftrightarrow a \succ_c^{App} b$.*

Beispiel Zwischen den Objekten a, b, c, x bestehen in der Realität (Sim_{App}) folgende Beziehungen:

(1) a ist ähnlicher zu x als b ($a \succeq_x b$)

(2) a ist ähnlicher zu x als c ($a \succeq_x c$)

(3) b ist ähnlicher zu x als c ($b \succeq_x c$)

Der Ähnlichkeitsbegriff Sim_{Rep} ist

- Korrekt, aber nicht vollständig, falls z.B. (1), (2), (1) und (2), oder (1) und (3) ausgewählt wurde.
- Vollständig, aber nicht korrekt, falls z.B. (1) und (2) und (3) und *c ist ähnlicher zu x als a* ausgewählt wurde.
- Korrekt und vollständig, falls (1) und (2) und (3) ausgewählt wurde.

Das Ziel des Auswahlprozesses in fallbasierten Systemen ist es, daß der repräsentierte Ähnlichkeitsbegriff im Sinne der Anwendung nach Möglichkeit vollständig und korrekt ist. Dies ist in der Praxis im allgemeinen nicht gesichert oder formal zu beweisen. Die Begriffe ermöglichen es aber, die Zielsetzungen der Lernphase klar zu definieren.

Empirischer Vergleich von Ähnlichkeitsbewertungen

Hinsichtlich der Interpretation der Begriffe Ähnlichkeit und Korrektheit ist man auf eine empirische Bewertung des verwendeten Auswahlverfahrens angewiesen. Ein vergleichbares Problem besteht im Bereich des Information Retrieval. Dort ist eine große Anzahl von Dokumenten gegeben, aus der die für den Anwender relevanten Informationen bestimmt werden müssen. Dafür gibt es verschiedene Retrievalansätze und Systeme, die empirisch miteinander verglichen werden müssen. Hierzu wird eine Leistungskennzahl (Retrievaleffektivität) von Retrieval Systemen ermittelt. Dabei können die Begriffe *Precision* und *Recall* aus dem Information Retrieval auf den Auswahlprozeß geeigneter Fälle übertragen werden. Später wird man sehen, daß diese Begriffe im engen Zusammenhang mit den Definitionen *Korrektheit* und *Vollständigkeit* stehen. Es werden folgende Bezeichnungen eingeführt:

Rel_{sel} :	Anzahl der geeigneten und von Sim_{Rep} ausgewählten Fälle
Rel_{rej} :	Anzahl der geeigneten und von Sim_{Rep} nicht ausgewählten Fälle
$Irel_{sel}$:	Anzahl der nicht geeigneten und von Sim_{Rep} ausgewählten Fälle

Im Information Retrieval bezeichnet der Begriff *Precision* die Fähigkeit eines Systems, in einem Anwendungskontext zwischen relevanten und irrelevanten Informationen zu unterscheiden.

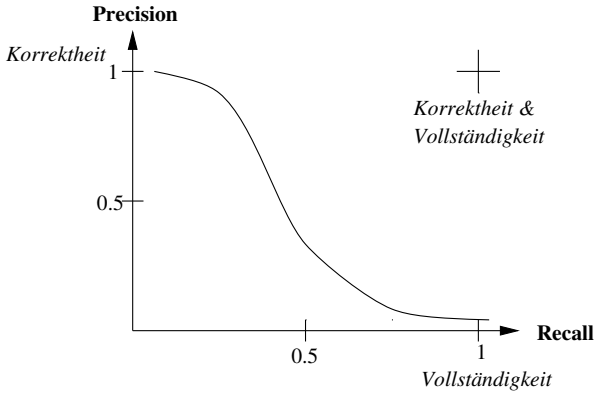


Abbildung 2.7: Empirisches Precision/Recall-Diagramm

Definition 2.8 (Precision)

$$Precision : P = \frac{\textit{Ausgewählte geeignete Fälle}}{\textit{Alle ausgewählten Fälle}} = \frac{Rel_{sel}}{Rel_{sel} + Irel_{sel}}$$

Falls nur relevante Fälle ausgewählt werden $(Precision = 1)$*, ist der repräsentierte Ähnlichkeitsbegriff* korrekt. *Im Information Retrieval bezeichnet der Begriff* Recall *die Fähigkeit eines Systems, in einem Anwendungskontext die relevanten Informationen zu bestimmen.*

Definition 2.9 (Recall)

$$Recall : R = \frac{\textit{Ausgewählte geeignete Fälle}}{\textit{Alle bekannten geeigneten Fälle}} = \frac{Rel_{sel}}{Rel_{sel} + Rel_{rej}}$$

Falls alle geeigneten Fälle bestimmt wurden ($Recall = 1$), ist der repräsentierte Ähnlichkeitsbegriff *vollständig*. In empirischen Untersuchungen stellte man einem System eine Vielzahl von unterschiedlichen Anfragen. Die Ergebnisse, d. h. die einzelnen Precision-/Recall-Werte für jede Anfrage, wurden in einem Diagramm aufgetragen – siehe Abbildung 2.7.

Die Untersuchung lieferte folgende Resultate:

- ein hoher Precision-Wert und ein hoher Recall-Wert stellen widersprüchliche Anforderungen dar.
- Hohe Precision-Werte ergeben sich zumeist bei spezifischen Suchanfragen, während hohe Recall-Werte eher bei allgemeinen Anfragen zu erwarten sind.

2.3 Fallbasierte Klassifikation

Für einige Anwendungen des fallbasierten Schließens, z. B. Klassifikation, Entscheidungsunterstützung oder Diagnose, kann oftmals auf die Adaption der gefundenen Lösung verzichtet werden. Beim fallbasierten Schließen wird zumeist diese Klasse von fallbasierten Ansätzen in der Praxis eingesetzt. Für fallbasierte Ansätze kann ein relativ einfaches Modell angegeben werden, mit dem dennoch eine Vielzahl von unterschiedlichen Arten fallbasierter Systeme realisiert werden können.

2.3.1 Grundlagen der fallbasierten Klassifikation

Die Aufgabe der Klassifikation ist die Zuordnung von Objekten oder Situationen zu vorgegebenen Klassen; hierfür seien zwei Beispiele angeführt:

- Psychologen versuchen Menschen bezüglich ihrer Persönlichkeit als introvertiert oder extravertiert einzustufen.
- Ein Mitarbeiter eines Versicherungsbüros möchte die versicherungstechnische Risikoeinstufung eines bestimmten Kraftfahrzeugs bestimmen.

Sei U ein Universum von Situationen und Objekten x und bezeichne $\mathcal{P}(\mathcal{U})$ die Potenzmenge von $\mathcal{U}$. Das Ziel der Klassifikation $U \rightarrow \mathcal{P}(\mathcal{U})$ ist die Zuordnung der Objekte $x \in \mathcal{U}$ zu bestimmten Begriffen $C_i \subseteq \mathcal{U}$, d. h. zu Teilmengen des Universums.

Die Klassifikation hat zum Beispiel im Bereich der ärztlichen Diagnose von Krankheiten das Ziel, einer Krankheitssituation die Art einer Krankheit zuzuordnen. Hierzu faßt man das Universum $\mathcal{U}$ als Menge aller Krankheitssituationen auf. Dementsprechend repräsentiert ein $x \in \mathcal{U}$ eine bestimmte Menge von Symptomen. Der Klassifikator nimmt also eine konkrete Zuordnung einer durch Symptome beschreibbaren Krankheitssituation zu einer Krankheit vor. In dieser Weise wird er beim Krankheitbild mit den Symptomen (rote Flecken im Gesicht, Juckreiz) die Kinderkrankheit *Masern* diagnostizieren.

Definition 2.10 (Klassifikator) *Ein Klassifikator $class$ für eine Menge $\mathcal{U}$ und eine vorgegebene Menge von Begriffen $\mathcal{C} = \{C_1, \ldots, C_n\}$ ist eine Abbildung $class : \mathcal{U} \rightarrow \mathcal{C}$, d. h. der Klassifikator $class$ ordnet jedem Element $x \in \mathcal{U}$ einen entsprechenden Begriff $C_i \in \mathcal{C}$ mit $1 \leq i \leq n$ zu.*

In diesem Abschnitt wird von einem sehr einfachen Modell der fallbasierten Klassifikation ausgegangen. Dazu wird folgendes definiert:

Definition 2.11 (Fall, Anfrage) *Ein Fall $F \in \mathcal{U} \times \mathcal{C}$ ist ein Tupel $(x, class(x)) = (x, C_i)$ mit $1 \leq i \leq n \in \mathbb{N}$ bestehend aus einem Element $x \in U$ zur Problembeschreibung und einer entsprechenden Klassifikation $class(x) = C_i \in \mathcal{C}$ als Problemlösung. Ist die Klassifikation eines (Problem-)Falles F unbekannt, so wird dies durch $class(x) = ?$ oder durch $F = (x, ?)$ beschrieben und als Anfrage bezeichnet.*

Diese Definition trennt dichotom klar zwischen einer Problembeschreibung und dem Teil des Falles, der die Problemlösung – Klassifikation – repräsentiert.[6] Wenn nur eine Problembeschreibung ohne Problemlösung vorliegt, so handelt

[6] M. Lenz (1999) definiert Fälle ohne die Dichotomie Problem + Lösung allein über sogenannte Information Entities: „Definition (Case and Query): A case consists of a unique case descriptor and a set of Information Entities. Similarly, a query is just a set of Information Entities." [31], p. 29. Lenz arbeitet auch mit dynamischen Gewichtungen für die einzelnen Attribute; [31], p. 56f.

es sich um eine Anfrage. Diese Anfragedefinition bezieht keine Gewichtung der Attribute mit ein, ist aber entsprechend erweiterbar. Die obige Falldefinition entspricht nicht dem jüngst an der Universität Kaiserslautern gefundenem CBR-Modell, wonach eben eine Lösungsbeschreibung das Fallwissen konstituiert; allerdings ist denkbar, daß Fälle hier auch noch eine Nutzenbeschreibung (Utility Description) enthalten [32].

Es ist aus Korrektheitsgründen sinnvoll, die obige Falldefinition einzuführen. Jedoch trennt die wissenschaftliche Literatur nicht immer strikt zwischen einem Fall und einer Anfrage ohne Lösungsbeschreibung. So spricht Wess beispielsweise in seiner Dissertation hüfig von fallvergleichenden Systemen. Es erwies sich bei der Realisierung des Systems *FluidCase*, die im zweiten Teil dieses Berichtes erläutert wird, als zweckmäßig, die strikte Trennung zwischen Anfrage und Fall nicht beizubehalten.

Definition 2.12 (Fallbasis) *Die Fallbasis $CB \subseteq \mathcal{U} \times \mathcal{C}$ ist eine* endliche *Menge von bekannten, bereits klassifizierten Fällen $F_j = (x_j, class(x_j));\ 1 \leq j \leq |CB| \in I\!N$.*

Ein Beispiel für einen (Krankheits-)Fall wäre also ((*rote Flecken im Gesicht*, *Juckreiz*), Masern). Eine Fallbasis besteht dementsprechend aus vielen Symptomen-Krankheitspaaren. Die Ähnlichkeit von Fällen wird hier zunächst als eine Präferenzordnung $y \succ_x z$ über der Fallbasis CB definiert, die durch ein entsprechendes Maß $sim(x, y)$ induziert wird.

Als nächstes wird ein einfacher Basisalgorithmus vorgestellt, der auch die Grundlage für die folgenden Abschnitte bildet [17].

Basisalgorithmus für die fallbasierte Klassifikation

Eingabe: sim_0, Anfrage / Fall F
Ausgabe: Klassifikation $C_i \in \mathcal{C},\ 1 \leq i \leq n$

1. Setze Fallbasis $CB = \emptyset$ und initialisiere das Ähnlichkeitsmaß sim mit einem initialen Maß sim_0.
2. Lese einen neuen Fall $F = (y, class(y))$ ein.
3. Bestimme ein $F_j = (x, class(x)) \in CB, j \in 1, \ldots, n$, mit dem Maximum $sim(x, y)$.
4. Ist die Klassifikation der Anfrage F noch unbekannt $(F = (y, ?))$, dann:
   (a) Gebe die bekannte Klassifikation $class(x)$ von F_j als Klassifikation von $F = (y, ?)$ aus: $F = (y, class(x))$.
   (b) Überprüfe die gefundene Klassifikation $F = (y, class(x))$ in der realen Welt. Sei $class(y)$ die korrekte Klassifikation der Anfrage $F = (y, ?)$.
5. Wenn $class(x) = class(y)$, dann setze $classification = correct$, sonst $classification = incorrect$.
6. Modifiziere die Fallbasis CB und / oder das Maß sim in Abhängigkeit von $classification$.
7. Gehe zu Schritt 2.

In diesem Algorithmus ist durch Schritt 6 eine Lernphase integriert. Dabei wird eine Modifikation der Fallbasis CB und / oder des Maßes sim gespeichert und auf die Lösung eines neuen Problems sofort angewandt. Die Entscheidung über die Aufnahme eines neuen Falles oder die Veränderung des Maßes könnte ein Experte oder das System selbst vornehmen. Die Abbildung 2.8 veranschaulicht den Ablauf des Algorithmus graphisch.

Fallbasiertes Lernen

Der Prozeß des fallbasierten Schließens ist eng mit dem Vorgang des Lernens verbunden. So kann beispielsweise ein neu klassifizierter Fall in die Fallbasis aufgenommen werden. Dabei sind Lernen und die Anwendung des gelernten Wissens miteinander verzahnt (Basisalgorithmus Schritt 6). Die gelernten Begriffe des fallbasierten Systems werden *implizit* als ein Tupel (CB, sim) repräsentiert.

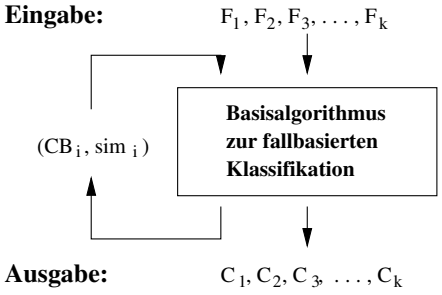


Abbildung 2.8: Basisalgorithmus zur fallbasierten Klassifikation

Ein *fallbasierter* Klassifikator beschreibt eine gelernte Begriffsmenge $\mathcal{C}$ *implizit* durch die Angabe eines Tupels (CB, sim), bestehend aus der Fallbasis CB und einem entsprechenden Maß sim [26]. Bei *falladaptierenden* Systemen muß dieses Tupel um ein Symbol T für die Transformation der gefundenen Lösung (Adaption) ergänzt werden; hier hat man also das Tripel (CB, sim, T). Dies läßt erkennen, daß der gelernte Begriff nicht direkt aus dem Klassifikator abzulesen ist, sondern sich erst aus dem Zusammenspiel von Fallbasis und Maß ergibt. Dadurch ergeben sich unterschiedliche Möglichkeiten, einen Begriff fallbasiert zu repräsentieren, d. h. es gibt eventuell mehrere Tupel $(CB_1, sim_1), \ldots, (CB_k, sim_k)$ für eine Begriffsmenge $\mathcal{C}$.

In der fallbasierten Klassifikation gibt es verschiedene Arten zu lernen:

- durch eine Änderung der Fallbasis CB;
- durch eine Änderung des verwendeten Maßes sim;
- durch eine gleichzeitige Änderung von Fallbasis CB und Maß sim.

Da dem fallbasierten System ständig neue Fälle $F_1, F_2, \ldots, F_k$ präsentiert werden, kann man auch von einem *Lernprozeß* sprechen. Dabei wird eine Folge von Tupeln $(CB_1, sim_1), \ldots, (CB_n, sim_n)$ mit $CB_i \subseteq \{F_1, F_2, \ldots, F_n\}$ und $i \in I\!N$ erzeugt. Ein Begriff $C_i \in \mathcal{C}$ wurde vom fallbasierten Klassifikator gelernt, wenn keine weitere Eingabe eines neuen Falles F_j den Klassifikator (CB_n, sim_n) ändert.

Repräsentation von Begriffen in fallbasierten Systemen

Ein fallbasiertes System besitzt als Ansatz zur Begriffsbildung ein wie auch immer geartetes Wissen über die Begriffsstruktur des zugrundeliegenden Universums. Aus dem vorherigen Abschnitt ist bekannt, daß man dem Tupel (CB, sim) nicht direkt ansehen kann, welche Begriffe es repräsentiert. Das Wissen über den Begriff ist auf die Fallbasis CB und die Ähnlichkeitsfunktion sim verteilt, d. h. man kann den durch (CB, sim) repräsentierten Begriff nicht allein durch die Kenntnis der Fallbasis oder des Ähnlichkeitsmaßes bestimmen. Erst der *Prozeß* der Anwendung des Maßes auf die gespeicherten Fälle ergibt die Bedeutung des repräsentierten Begriffes. Dies wird durch folgendes Beispiel veranschaulicht:

In einem Textverarbeitungssystem soll die Rechtschreibung eines Textes überprüft und ggf. korrigiert werden. Dabei wird für ein zu überprüfendes Wort wie folgt vorgegangen:

1. Überprüfe, ob das Wort im vorgegebenen Wörterbuch vorhanden ist.
2. Falls ja, ist das Wort korrekt geschrieben.
3. Falls nein, stellt das System dem Benutzer *ähnliche* Wörter vor, aus dem er ein geeignetes Wort auswählen kann.
4. Falls der Benutzer kein Wort ausgewählt hat, kann er das Wörterbuch durch die Eingabe eines neuen Wortes erweitern.

Diese Vorgehensweise stellt einen primitiven Ansatz des fallbasierten Schließens dar, wobei sich das Wörterbuch als Fallbasis CB interpretierten läßt. Dabei könnte man sich die Repräsentation des Wissens durch folgende Möglichkeiten vorstellen:

- *Jedes mögliche Wort wird in das Wörterbuch aufgenommen*. Diese Vorgehensweise scheitert normalerweise an der großen Anzahl zu speichernder Wörter.
- *Es werden nur Grundwörter (Wortstamm) und Ausnahmen gespeichert*. Zusätzlich wird eine Grammatik in Form eines Regelsystems angegeben, mit dem sich weitere Wörter (z. B. Pluralformen) ableiten lassen.

Bei der zweiten Möglichkeit ist das Wissen über korrekt geschriebene Wörter verteilt auf eine *Fallbasis* (Wörterbuch) und ein *Ähnlichkeitsmaß* (teilweise die Grammatik), d. h. erst durch die Anwendung der Grammatik auf das Wörterbuch ist eine vollständige Korrektur des Textes möglich. Dies zeigt auch, daß es in der Praxis aus Gründen der Effizienz sinnvoll ist, Wissen auf beide Komponenten zu verteilen.

2.3.2 Geometrisches Modell der Klassifikation

In diesem Abschnitt wird das fallbasierte Schließen als ein geometrisches Problem dargestellt. Dies ist durch eine größere Anschaulichkeit der Problemstellung und eine Vielzahl alternativer Verfahren (z. B. bekannte geometrische Algorithmen) und neuen Einsichten motiviert. Dazu werden die Einträge einer Fallbasis CB in einen mehrdimensionalen Vektorraum $\mathbb{R}^n$ transformiert. Die Nähe zweier Punkte spiegelt dann die Ähnlichkeit zwischen zwei Fällen wieder. Also geht es um die Frage, welcher Punkt aus einer Menge von gegebenen Punkten einem weiteren neuen Punkt – Anfrage – am nächsten liegt. Dieses Problem ist auch als *Nearest-Neighbor-Problem* bekannt.

Im nächsten Abschnitt wird die Nearest-Neighbor Klassifikation, die eng mit der Lösung des Nearest-Neighbor Problems verbunden ist, erläutert.

Nearest-Neighbor-Klassifikation

Die *Nearest-Neighbor-Klassifikation* (NN-Klassifikation) geht davon aus, daß die vorhandenen Punkte im geometrischen Raum gewisse Ballungen (Cluster) bilden und somit nahe beieinander liegende Punkte (Fälle) auch zur selben Klasse gehören. Ein neues Objekt wird klassifiziert, indem die Klassifikation des nächsten Nachbarn ausgegeben wird. Hierzu betrachte man für den zweidimensionalen Fall die Abbildung 2.9.

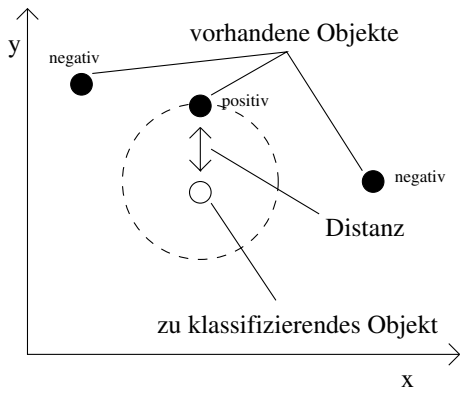


Abbildung 2.9: Beispiel für eine Nearest-Neighbor Klassifikation. Das zu klassifizierende Objekt wird *positiv* klassifiziert.

Als nächstes werden zwei mit dem Nearest-Neighbor Problem verwandten Suchprobleme vorgestellt:

- Das *Fixed-Radius-Nearest-Neighbor-Problem* bestimmt alle Punkte aus dem Universum U, die zu einem Anfragepunkt $q \in \mathbb{R}^n$ des zu klassifizierenden Objektes einen Abstand von höchstens $d_{max} > 0$ haben.
- Das *k-Nearest-Neighbor-Problem* bestimmt zu einem Anfragepunkt $q \in \mathbb{R}^n$ die k nächsten Nachbarn.

Letzteres Problem bildet die Grundlage für die k-Nearest-Neighbor Klassifikation. Die einfache NN-Klassifikation besitzt den Nachteil, daß in den interessanten Randbereichen bestimmte Punkte, durch die räumliche Nähe bedingt, eventuell als bereits zum nächsten Cluster gehörend klassifiziert werden. Dies veranschaulicht die Abbildung 2.10.

Diese Schwäche, besonders gegenüber verrauschten Daten, kann durch die Bestimmung der $k > 1$ Nachbarn aufgehoben werden. Dies gründet darin, daß in den Randbereichen die räumliche Nähe zu einem anderen Cluster durch die starke Präsenz von Punkten der eigenen Klasse ausgeglichen werden kann. Der Erfolg der k-NN-Klassifikation hängt jedoch stark von der Wahl des Parameters k ab.

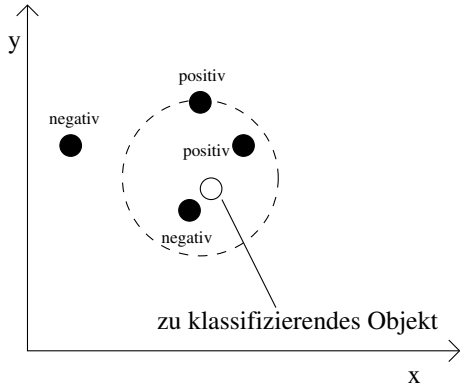


Abbildung 2.10: Beispiel für eine 3-Nearest-Neighbor Klassifikation. Das zu klassifizierende Objekt wird *positiv* klassifiziert, obwohl der nächste Nachbar *negativ* klassifiziert wurde. Die Begründung dafür ist, daß zwei der drei Nachbarn *positiv* klassifiziert sind.

Voronoi-Diagramme

Zur Lösung des Nearest-Neighbor Problems werden auch sogenannte Voronoi-Diagramme verwendet. Dabei ist eine Voronoi-Region die Menge aller Punkte, die von einem gegebenen Punkt einen geringeren Abstand haben als von allen anderen Punkten aus einer Punktemenge.

Definition 2.13 (Voronoi-Region) *Betrachtet man eine Menge $P \subseteq U$ von Punkten, so läßt sich für jeden beliebigen Punkt $p \in P$ die Menge $V(p)$ derjenigen Punkte aus U bestimmen, die näher bei p liegen als bei jedem anderen Punkt $q \neq p \in P$:*

$$V(p) = \{x \in U \mid d(x,p) \leq d(x,q),\ q \in P \backslash \{p\}\}.$$

Diese Menge $V(p)$ nennt man Voronoi-Region von p.

Ein Voronoi-Diagramm *ist die Menge $VD(P)$ aller von $p \in P$ erzeugten Voronoi-Regionen. Die Begrenzung einer Voronoi-Region $V(p)$ heißt* Voronoi-Polygon. *Für den zweidimensionalen Fall ist in Abbildung 2.11 hierzu ein Beispiel angegeben [30].*

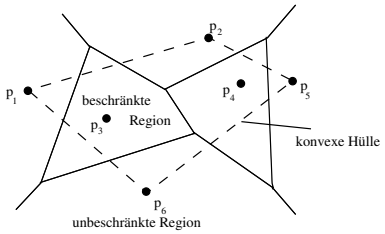


Abbildung 2.11: Beispiel eines Voronoi-Diagramms VD(P) für den zweidimensionalen Raum mit konvexer Hülle

Konstruktion eines Voronoi-Diagramms Zunächst sind einige Eigenschaften eines Voronoi-Diagramms zu nennen – siehe Abbildung 2.12:

- Jede Kante eines Voronoi-Diagramms trennt zwei Voronoi-Regionen.
- Ein Voronoi-Diagramm besteht aus beschränkten und unbeschränkten Regionen.

- Die Punkte der unbeschränkten Regionen bilden die konvexe Hülle von P.

Ein Voronoi-Diagramm wird als ein Graph gespeichert. Dabei wird zu jeder Kante des Voronoi-Diagramms vermerkt, welche beiden Regionen diese trennt. Im folgenden wird eine Konstruktion eines Voronoi-Diagramms für den zweidimensionalen Fall (euklidischer Abstand) erläutert [18].

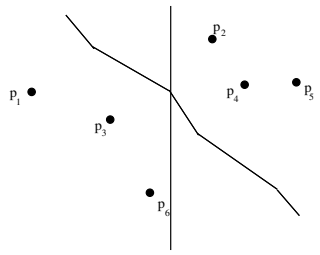


Abbildung 2.12: Ein Kantenzug zwischen Voronoi-Regionen

Dieses Verfahren verwendet das Divide-and-Conquer Prinzip [30]:

1. Teile P in zwei gleich große Teilmengen P_1 und P_2.
2. Berechne die Voronoi-Diagramme für P_1 und P_2 rekursiv.
3. Verschmelze die beiden Voronoi-Diagramme für P_1 und P_2 zum Voronoi-Diagramm $VD(P)$.

Das Rekursionsende ist bei der Betrachtung der Menge eines Punktes erreicht, da hierfür das Voronoi-Diagramm gerade die ganze Ebene ist. In Schritt 3 ist es wichtig, daß die Teil-Diagramme effizient zu verschmelzen sind. Dabei hilft die Beobachtung, daß, wenn man das Voronoi-Diagramm durch eine vertikale Linie in zwei Teile aufspaltet, ein Kantenzug zwischen den beiden Teil-Diagrammen existiert, der sowohl an Punkte aus P_1 als auch an Punkte aus P_2 grenzt – siehe Abbildung 2.13. Das Voronoi-Diagramm $VD(P)$ setzt sich dann aus dem links vom Kantenzug liegenden Voronoi-Diagramm von P_1 und dem rechts vom Kantenzug liegenden Voronoi-Diagramm von P_2 zusammen. Jetzt kann ein genauerer Algorithmus angegeben werden:

Algorithmus *Voronoi-Diagramm*

Gegeben: Eine Menge P aus n Punkten
Gesucht: Das Voronoi-Diagramm $VD(P)$

1. {Divide}
   Teile P durch eine vertikale Trennlinie T in zwei etwa gleich große Teilmengen P_1 (links von T) und P_2 (rechts von T), falls |P| > 1 ; sonst ist $VD(P)$ die gesamte Ebene.
2. {Conquer}
   Berechne $VD(P_1)$ und $VD(P_2)$ rekursiv.
3. {Merge}
   (a) Berechne den P_1 und P_2 trennenden Kantenzug K, der Teil von $VD(P)$ ist.
   (b) Schneide den rechts vom Kantenzug K liegenden Teil von $VD(P_1)$ ab, und schneide den links von K liegenden Teil von $VD(P_2)$ ab.
   (c) Vereinige $VD(P_1)$, $VD(P_2)$ und K; das ergibt $VD(P)$

Das Aufteilen der Menge P in Schritt 1 kann durch die Bestimmung des Medians der x-Koordinaten aller Punkte erfolgen. Das Abschneiden der Kanten in Schritt 3b kann durch das Durchlaufen der jeweiligen Kanten erfolgen. Beides kann in linearer Zeit ausgeführt werden. Die Berechnung von K ist der kritische Punkt. Im folgenden wird gezeigt, daß dies auch in linearer Zeit möglich ist und sich somit eine gesamte Laufzeit $O(n \log n)$ ergibt.

Die beiden Voronoi-Diagramme $VD(P_1)$ und $VD(P_2)$ können als gegeben angenommen werden, da diese schon rekursiv berechnet wurden. Der trennende Kantenzug K beider Diagramme muß in $VD(P)$ so verlaufen, daß alle

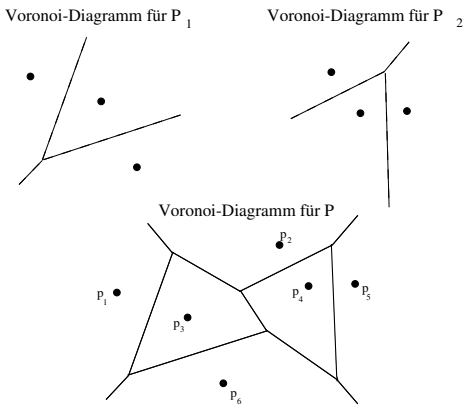


Abbildung 2.13: Rekursive Berechnung des Voronoi-Diagramms

Punkte links von K näher bei einem Punkt aus P_1 als bei einem Punkt aus P_2 liegen – dies gilt analog für Punkte rechts von K [30]. Also besteht K aus Geradenstücken, die jeweils die Mittelsenkrechten eines Punktes aus P_1 und P_2 sind. Das Verfahren wird anhand des Beispiels aus Abbildung 2.14 erläutert. Der Kantenzug K wird schrittweise,

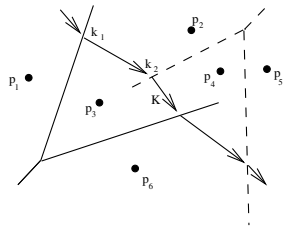


Abbildung 2.14: Konstruktion des Kantenzuges K

also ein Geradenstück nach dem anderen berechnet. Ausgangspunkt ist die Bildung der Mittelsenkrechten der beiden oberen Punkte der konvexen Hülle der Punktmenge P. Diese Punkte werden dadurch gefunden, daß eine Gerade von oben an die konvexen Hüllen von P_1 und P_2 gelegt wird. Im Beispiel sind dies die Punkte p_1 und p_2. Auf dieser Mittelsenkrechten läßt man einen Punkt k von oben nach unten wandern, bis er eine Kante des Teil-Diagramms $VD(P_1)$ oder $VD(P_2)$ schneidet. Im Beispiel ist dies der Punkt k_1, dort ist die Grenze zwischen $V(p_1)$ und $V(p_3)$, also der Voronoi-Region von p_1 und p_3, erreicht. An dieser Stelle muß der Kantenzug K von der aktuellen Mittelsenkrechten abweichen, weil sich der nächstliegende Punkt in P_1 für K geändert hat. Da der Kantenzug K in vertikaler Richtung monoton fällt, wird p_3 zum K nächstliegenden Punkt in P_1. Das nächste Geradenstück für K ergibt sich damit aus der Mittelsenkrechten der Verbindungsstrecke von p_3 nach p_2. Das Herabwandern auf dieser Mittelsenkrechten endet beim Schnittpunkt k_2. Das nächste Geradenstück von K wird dann durch die Mittelsenkrechte zwischen p_3 und p_4 gebildet. Dieser Prozeß wird solange fortgesetzt, bis K der Mittelsenkrechten der unteren Geraden an die beiden konvexen Hüllen von P_1 und P_2 folgt. Ein genauer Algorithmus und eine Laufzeitabschätzung ist in [18] sowie [30] angegeben.

Fallbasierte Klassifikation und Voronoi-Diagramme

Bisher wurde ein gelernter Begriff $C_i \in \mathcal{C}$ nur implizit durch die Angabe des Tupels (CB, sim) beschrieben. Die Verwendung von Voronoi-Diagrammen VD bietet die Möglichkeit, die vom System (CB, sim) repräsentierten Begriffe *explizit* darzustellen. Bei der fallbasierten Klassifikation entsprechen die problembeschreibenden Teile aller Fälle der Fallbasis den Punkten der Punktemenge P. Die Voronoi-Region $V(F)$ zu einem Fall $F \in CB$ ist genau der Bereich, in dem alle Fälle F_i $(i \in \{1, \ldots, n\},\ i \in I\!N, n = |CB|)$ des Universums $\mathcal{U}$ wie F klassifiziert werden. Die Klassifikation der Fälle $F_i \in \mathcal{U}$ wird durch die Zugehörigkeit zu einer bestimmten Voronoi-Region bestimmt. Der vom fallbasierten Klassifikator (CB, sim) repräsentierte Begriff entspricht der Vereinigung aller Voronoi-Regionen $V(p)$, die gleichklassifizierte Fälle enthalten.

Geometrische Interpretation des Lernvorgangs

Die Aufnahme eines Falles F_i in die Fallbasis CB entspricht geometrisch der Konstruktion einer weiteren Voronoi-Region $V(F_i)$ und eine Anpassung der bisherigen Voronoi-Regionen $V(p)$ im Voronoi-Diagramm $VD(CB)$. Die Anzahl der Voronoi-Regionen wächst damit um eins. Eine Änderung des verwendeten Maßes kann geometrisch als Anwachsen bzw. Schrumpfen der Fläche der einzelnen Voronoi-Regionen $V(p)$ verstanden werden. Die Anzahl der Voronoi-Regionen bleibt gleich.

2.3.3 Klassen fallbasierter Algorithmen

Der in Abschnitt 2.3.1 beschriebene Basisalgorithmus für die fallbasierte Klassifikation ist sehr allgemein gehalten und läßt deshalb noch viele Gestaltungsmöglichkeiten offen. Fallbasierte Algorithmen könnten beispielsweise durch die Verteilung des Wissens auf Fallbasis und Maß, durch verschiedene Aufnahmestrategien für Fälle oder Änderungsstrategien für das Maß charakterisiert werden.

Verhältnis von Maß und Fallbasis in fallbasierten Systemen

Im folgenden werden Möglichkeiten, qualitative Aussagen über den Informationsgehalt einer Fallbasis oder eines Maßes zu erhalten, erläutert. Die Vergleiche werden dabei mit Hilfe der von einem System korrekt klassifizierten Fälle vorgenommen.

Definition 2.14 (Klassifikationsgüte) *Ein System $S_1 = (CB_1, sim_1)$ klassifiziert* besser *als ein System $S_2 = (CB_2, sim_2)$, geschrieben als $S_1 \sqsupset S_2$, falls mit dem System S_1* mehr Fälle korrekt klassifiziert *werden können als mit dem System S_2.*

Man könnte meinen, daß das Wissen einer Fallbasis leicht über die Anzahl und die Art der darin enthaltenen Fälle quantifizierbar ist. Da sich aber der Beitrag eines einzelnen Falles zur Klassifikationsgüte des Gesamtsystems nur schwer bestimmen läßt, erweist sich dies als sehr schwierig. Daher bezieht sich die nächste Definition wieder auf die vom System korrekt klassifizierten Fälle [29].

Definition 2.15 (Relative Informiertheit von Fallbasen) *Die Fallbasis CB_1 eines fallbasierten Systems $S_1 = (CB_1, sim)$ heißt bezüglich des Maßes sim* besser informiert *als die Fallbasis CB_2 des Systems $S_2 = (CB_2, sim)$, geschrieben als $CB_1 \sqsupset CB_2$, falls mit dem System S_1* mehr Fälle korrekt klassifiziert *werden können als mit dem System S_2: $S_1 \sqsupset S_2$.*

Die Informiertheit von Fallbasen ist ein sehr relativer Begriff, da Aussagen nur bezüglich eines bestimmen Maßes gemacht werden können. Gleiches gilt auch für das in einem Maß implizit kodierte Wissen.

Definition 2.16 (Relative Informiertheit von Maßen) *Das in einem fallbasierten System $S_1 = (CB, sim_1)$ verwendete Maß sim_1 heißt bezüglich der Fallbasis CB* besser informiert *als das Maß sim_2 eines fallbasierten Systems $S_2 = (CB, sim_2)$, geschrieben $sim_1 \sqsupset sim_2$, falls mit dem System S_1* mehr Fälle *des Universums U* korrekt klassifiziert *werden können als mit dem System S_2 : $S_1 \sqsupset S_2$.*

Bei der Konstruktion eines fallbasierten Systems wird in das System durch die Angabe eines Maßes Wissen eingebracht, das sich nicht unbedingt auf eine konkrete Fallbasis bezieht. Daher ist es sinnvoll, das kodierte Wissen in einem Maß absolut, d. h. ohne Bezug auf eine Fallbasis, beurteilen zu können. Die Gleichung Begriff ≡ Fallbasis + Maß zeigt dazu einen indirekten Weg auf. Ist der zu lernende Begriff bekannt, so ist in einem Maß umso mehr Wissen kodiert, je weniger Wissen in Form von Fällen dem System zugeführt werden muß.

Definition 2.17 (Minimale Fallbasis) *Die Fallbasis CB eines fallbasierten Systems (CB, sim) heißt* minimal *für ein Maß sim, falls keine Fallbasis $CB' \subset CB$ existiert mit: (CB', sim) klassifiziert mehr Fälle korrekt als (CB, sim).*

Definition 2.18 (Informiertheit von Maßen) *Das in einem System $S_1 = (CB_1, sim_1)$ verwendete Maß sim_1 heißt* besser informiert *als das Maß sim_2 eines fallbasierten Systems $S_2 = (CB_2, sim_2)$, geschrieben als $sim_1 \lhd sim_2$, falls für die entsprechenden* minimalen *Fallbasen CB_1^{min} und CB_2^{min} bei gleicher Klassifikationsgüte gilt: $\mid CB_1^{min} \mid < \mid CB_2^{min} \mid$.*

Minimalität und Universalität von fallbasierten Systemen In diesem Abschnitt werden fallbasierte Systeme anhand der Dimensionen *Minimalität* und *Universalität* verglichen. Bei der *Minimalität* soll bereits eine geringe Anzahl von Fällen F ausreichen, um das System sinnvoll nutzen zu können. Hierfür muß das verwendete Maß entsprechend gut informiert sein. Bei der *Universalität* soll das fallbasierte System für eine Vielzahl von unterschiedlichen Begriffen (Anwendungsdomänen) einsetzbar sein. Dabei bezieht sich die Universalität direkt auf die Menge der mit dem Maß sim erlernbaren Begriffe C.

Definition 2.19 (Universalität von Maßen) *Ein Maß sim_1 heißt* universeller *als ein Maß sim_2, falls die Menge der mit sim_1 erlernbaren Begriffe eine echte Obermenge der mit sim_2 ableitbaren Begriffe ist.*

Definition 2.20 (Universalität) *Ein fallbasiertes System (CB_1, sim_1) heißt* universeller *als ein System (CB_2, sim_2), falls das Maß sim_1* universeller *als das Maß sim_2 ist.*

Die *Universalität* und die *Minimalität* eines fallbasierten Systems stehen in einer konfliktären Zielbeziehung zueinander, d. h. in dem Maße, in dem das Ziel *Universalität* erreicht wird, geht dies zu Lasten des Ziels *Minimalität*. Dieser Zusammenhang wird durch die Abbildung 2.15 veranschaulicht.

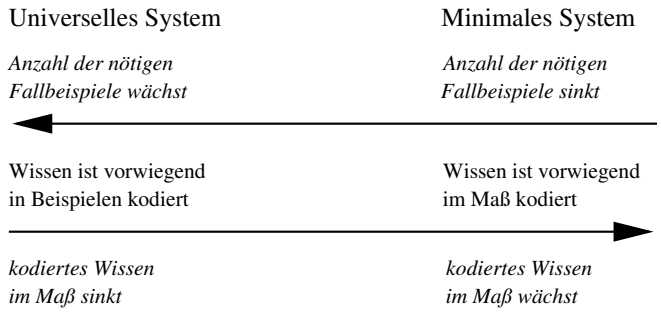


Abbildung 2.15: Minimalität gegen Universalität in fallbasierten Systemen (Wess 1995)

Aufnahme- und Änderungsstrategien

Der Lernprozeß in einem fallbasierten System findet auf der Ebene der Fallbasis (Aufnahmestrategie) und / oder des verwendeten Maßes (Änderungsstrategie) statt.

Begriffsbestimmung 1 (Aufnahmestrategie) *Die* Aufnahmestrategie *eines fallbasierten Klassifikators (CB_k, sim_k) bestimmt nach der Präsentation eines weiteren Falles den Übergang der Fallbasis von CB_k nach CB_{k+1}; $k \in I\!N$.*

Begriffsbestimmung 2 (Änderungsstrategie) *Die* Änderungsstrategie *eines fallbasierten Klassifikators (CB_k, sim_k) bestimmt nach der Präsentation eines weiteren Falles den Übergang des Maßes von sim_k nach sim_{k+1}; $k \in I\!N$.*

Bei beiden Strategien muß der mögliche Vorteil einer verbesserten Klassifikationsfähigkeit, d. h. unter Umständen können nach der Aufnahme eines neuen Falles mehr Fälle korrekt klassifiziert werden, gegen die Nachteile einer sich verschlechternden Performanz, wie z. B. hohe Laufzeiten und steigenden Speicheranforderungen, abgewogen werden. Dieses Problem wird als Utility-Problem bezeichnet.

Begriffsbestimmung 3 (Utility-Problem) *Beim fallbasierten Lernen muß grundsätzlich die durch einen Lernschritt zu erzielende* Klassifikationsfähigkeit *gegen die Auswirkungen auf die daraus resultierende* Systemperformanz *abgewogen werden. Für den allgemeinen Fall ist das Utility-Problem bisher ungelöst.*

Aufnahmestrategien In der Literatur lassen sich drei Klassen von Aufnahmestrategien (Case-Based Learning) identifizieren, wobei jede Strategie eine spezifische Folge von Tupeln $(CB_1, sim), \ldots, (CB_k, sim), (CB_{k+1}, sim), \ldots$ erzeugt.

CBL1: Alle Fälle werden in die Fallbasis aufgenommen.
$CB_k = \{F_1, \ldots, F_k\}$ wird zu $CB_{k+1} = \{F_1, \ldots, F_k, F_{k+1}\},\ k \in \mathbb{N}$.

CBL2: Nur vom System falsch klassifizierte Fälle werden aufgenommen.
$CB_k \subseteq \{F_1, \ldots, F_k\}$ wird zu $CB_{k+1} \subseteq CB_k \cup \{F_{k+1}\}$ mit $CB_k \subseteq CB_{k+1},\ k \in \mathbb{N}$.

CBL3: Zur Klassifikation nicht beitragende Fälle werden wieder entfernt.
$CB_k \subseteq \{F_1, \ldots, F_k\}$ wird zu $CB_{k+1} \subseteq CB_k \cup \{F_{k+1}\},\ k \in \mathbb{N}$.

Während Strategie CBL1 das Ziel der Klassifikationsverbesserung verfolgt, zielt die Strategie CBL3 auf eine Performanzsteigerung ab; Strategie CBL2 berücksichtigt beide Ziele.

Empirischer Vergleich verschiedener Aufnahmestrategien Zunächst wurden dem CBL1- dowie dem CBL2-Algorithmus 100 Fälle zum Training präsentiert. Anschließend wurden diese 100 Fälle erneut zur Performanzbewertung präsentiert. Der CBL2-Algorithmus verringerte die Menge der Beispiele auf 49 Fälle. Anschließend testete man die Klassifikationsgenauigkeit beider Algorithmen, wobei die Anzahl der pro Fall bekannten Symptome schrittweise vermindert wurde. Die Ergebnisse sind in Abbildung 2.16 zusammengefaßt. Es zeigte sich, daß durch die Anwendung

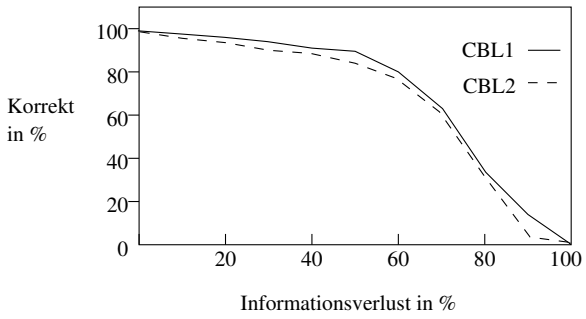


Abbildung 2.16: Klassifikationsgüte der Aufnahmestrategie CBL1 und CBL2 für eine Beispielfallbasis nach Wess (1995)

einer relativ einfachen Aufnahmestrategie – wie etwa CBL2 – die Fallbasis erheblich reduziert werden kann, ohne daß die Klassifikationsfähigkeit des Systems zu stark vermindert wird.

Änderungsstrategien für das Maß In diesem Abschnitt wird ein allgemein gehaltener Algorithmus zur Änderung des Ähnlichkeitsmaßes in fallbasierten Systemen betrachtet.

CBL4: Die Relevanz der Merkmale für einen bestimmten Begriff wird berücksichtigt.

Soll beispielsweise ein Motorschaden diagnostiziert werden, so ist die Bedeutung des Symptoms Öltemperatur größer als die des Reifenluftdrucks. Bei der Diagnose geplatzter Reifen verhält es sich genau umgekehrt.

Empirischer Vergleich von verschiedenen Änderungsstrategien Für das PATDEX[7] System, dessen Funktionsweise in diesem Zusammenhang nicht wichtig ist, wurden unterschiedliche Änderungsstrategien empirisch untersucht [23]. Dabei wurde die Klassifikationsgüte des Klassifikators nach einer bestimmten Anzahl von präsentierten Fällen und die Konvergenzgeschwindigkeit, d. h. die Anzahl der präsentierten Fälle, die zur Erreichung einer bestimmten Klassifikationsgüte notwendig sind, betrachtet.

[7] PATtern Directed EXpert Systems. Ein an der Universität Kaiserslautern entwickelter fallbasierter Problemlöser [22].

Klassifikationsgüte Für die Klassifikationsgüte verschiedener Strategien ergab sich im statistischen Mittel eine Übereinstimmung der Klassifikationsergebnisse. Ist der zu lernende Begriff C oder eine Klasse von Begriffen vor Beginn des Lernvorgangs bekannt, so lassen sich durchaus geeignete und weniger geeignete Strategien identifizieren. Aber über alle möglichen zu lernenden Begriffe heben sich die Vorteile bestimmter Strategien allerdings weitgehend auf.

Konvergenzgeschwindigkeit Für die Konvergenzgeschwindigkeit konnten, im Gegensatz zur Klassifikationsgüte, starke Unterschiede festgestellt werden, d. h. bestimmte Änderungsstrategien führten bei kleineren Fallbasen zu besseren Klassifikationsergebnissen. Bei zunehmender Anzahl von präsentierten Fällen glichen sich die verschiedenen Strategien allerdings an. Zusammenfassend konnte für die verschiedenen Änderungsstrategien nicht festgestellt werden, ob eine Änderungsstrategie besser ist als eine andere, d. h. es konnte keine Präferenzordnung bestimmt werden. Dies liegt an der starken Abhängigkeit der Änderungsstrategie von der Anwendungsdomäne. Also ist der Erfolg der verschiedenen Strategien sehr stark vom zu lernenden Begriff abhängig.

Veränderte Rahmenbedingungen

In diesem Kapitel ist bisher von folgendem Idealzustand ausgegangen worden:

- Endliche Domänen
- Vollständige und korrekte Information
- Boolesche Begriffsdefinition
- Metrische Räume

Diese Voraussetzungen haben aber keine praktische Relevanz, da man es meistens mit unvollständigen, verrauschten oder falschen Informationen, multiplen / ambivalenten Begriffsdefinitionen und nicht nur mit metrischen Räumen zu tun hat. Deshalb werden in diesem Abschnitt die Rahmenbedingungen erweitert.

Endliche Domänen Die Anzahl der gespeicherten Fälle |CB| sowie die Anzahl der verarbeiteten Merkmale sind endlich, aber die Anfrage Q kann einen Wert enthalten, der nicht durch ein definiertes Merkmal repräsentiert ist. Dies gilt beispielsweise für das Attribut $Preis$. Denn der Benutzer kann in einer Anfrage Q einen (beliebigen) Preis eingeben, der im CBR-System nicht repräsentiert ist, so daß die Grenzen der Wissensbasis überschritten werden. Das Intervallproblem ist hier also für nicht scharf begrenzte Wertebereiche zu lösen [31].[8]

Unvollständige Information Beispielsweise stehen im Bereich der technischen Diagnostik nicht immer alle erwünschten Meßgrößen eines technischen Systems zu Verfügung. In vielen Entscheidungs- und Auswahlsituationen möchte der Benutzer nur die für ihn relevanten Fragestellungen betrachten. Es werden zwei Arten von unvollständiger Information unterschieden:

- Eine Information kann fehlen, weil der Benutzer die Irrelevanz (don't care) der Information im gegebenen Kontext erkannt hat. Dieser Fall ist eher unproblematisch, da die entsprechenden Attribute bei der Klassifikation einfach ignoriert werden können.
- Es handelt sich um eine wirklich unvollständige Information (unknown). Dabei kann das Fehlen zumindest eines Attributwertes für die Klassifikation relevant sein. In diesem Fall muß der fehlende Attributwert, entweder explizit oder implizit, ergänzt werden.
  - explizit: Man kann die fehlenden Attributwerte ergänzen, z. B. durch die Akquisition von Domänenwissen.
  - implizit: Man kann die Behandlung von fehlenden Attributwerten in das verwendete Maß integrieren.

[8] Lenz berichtet über seine Erfahrungen bei der Entwicklung eines CBR-Systems zum Auffinden von sogenannten *know how* Dokumenten: „ ... this domain is not finite a priori. Rather in each new document and in each query new terms may occur that are not contained in the domain model." [31], p. 134

Unsichere Information Beispielsweise kann die Unsicherheit über Informationen an technischen Gegebenheiten (z. B. unsichere Meßdaten), an den am Entscheidungsprozeß beteiligten Benutzer selbst oder auch an unscharfen Begriffen (z. B. niedrige Temperatur) liegen. Es werden drei Formen von Unsicherheit unterschieden:

- Unsicherheit in den Attributwerten;
- Unsicherheit in den Klassifikationen;
- Unsicherheit in den Begriffsdefinitionen.

Es stellt sich natürlich die Frage, wie sich solche Unsicherheiten auf das Verhalten fallbasierter Verfahren auswirkt. Ist unter verrauschten Daten eine Begriffsbestimmung oder eine Klassifikation von geeigneten Fällen überhaupt noch gewährleistet? In empirischen Untersuchungen zeigte sich, daß, um Störungen zu vermeiden, spezifisches Wissen über die unsichere Information im Ähnlichkeitsmaß integriert werden muß. Außerdem sind gut informierte Maße rauschtoleranter als universelle Maße.

2.4 Zusammenfassung und Ausblick

In Kapitel 2.2 wurde die für das fallbasierte Schließen wichtige Auswahl von geeigneten Fällen behandelt. Dabei muß aus Komplexitätsgründen die *a-Posteriori Nützlichkeit* von Fällen für die *Problemlösung* auf die *a-Priori Ähnlichkeit* von *Problemstellungen* reduziert werden. Grundsätzlich besteht beim Auswahlprozeß im fallbasierten Schließen immer das Problem, von unvollständigen und unsicheren Informationen auf noch nicht bekannte Problemlösungen zu schließen, d. h. die Begriffe *Ähnlichkeit*, *Nützlichkeit* und *Unsicherheit* sind eng miteinander verbunden. Aufgrund solcher Unsicherheiten kann die Auswahl von Fällen im allgemeinen nicht als *richtig* oder *falsch*, sondern nur als *besser* oder *schlechter* beurteilt werden. Für den Ähnlichkeitsbegriff wurden drei Formalisierungen angegeben:

- als Prädikat $SIM(x, y)$;
- als Präferenzrelation $y \succeq_x z$;
- als Distanzmaß $d(x, y)$ sowie das Ähnlichkeitsmaß $sim(x, y)$.

Dabei wurde festgestellt, daß die Semantik der einzelnen Ansätze stark von den Zielsetzungen des Auswahlprozesses abhängig ist. Am Ende des Auswahlprozesses steht immer die Interpretation des Ähnlichkeitsbegriffes als Prädikat, da nur der *nützlichste* Fall für die Lösung eines Problems benötigt wird. Durch die binäre Ähnlichkeitsbetrachtung kann allerdings bei mehreren ähnlichen Fällen nicht zwischen diesen unterschieden werden. Für Anordnungen von Fällen wird also eine Präferenzrelation benötigt, die dabei die Rolle einer Konfliktlösungsstrategie spielt. Die Frage nach einem geeigneten Ähnlichkeitsmodell für eine gegebene Situation konnte nicht geklärt werden. Die Bestimmung einer systematischen Vorgehensweise für dieses Problem wurde z. B. in [25] behandelt. In diesem Kapitel wurden Modelle des Ähnlichkeitsbegriffes angegeben, die für bestimmte Anwendungsbereiche geeignet sind. Beispielsweise liegt in technischen Domänen eine Betrachtung der Ähnlichkeit mit Hilfe einer Abstraktion nahe, da dort eine Komposition von primitiven zu komplexen Strukturen stattfindet. Bei der Entwicklung eines fallbasierten Systems für eine konkrete Anwendung müssen die folgenden drei Phasen durchlaufen werden:

1. Bestimmung eines adäquaten Modells zur Ähnlichkeitsbestimmung;
2. Bestimmung einer entsprechenden Präferenzrelation;
3. Entwicklung von Algorithmen zu effizienten Bestimmung von $y \succeq_x z$ zur Laufzeit.

In Kapitel 2.3 wurden die Grundlagen der fallbasierten Klassifikation hinsichtlich der Entwicklung von wissensbasierten Systemen dargestellt. Viele Ergebnisse dieses Kapitels können auf fallbasierte Systeme im allgemeinen übertragen werden. Für die fallbasierte Klassifikation wurden die Beziehungen zwischen dem zu lernenden Begriff, dem verwendeten Maß und der Fallbasis untersucht. Dabei erwiesen sich die Parameter Fallbasis und Maß als stark zusammenhängend, d. h. die vom fallbasierten System repräsentierten Begriffe ergeben sich aus dem Zusammenspiel von Fallbasis und Maß. Ein fallbasierter Klassifikator kann als ein Tupel (CB, sim) aufgefaßt werden. Weiterführende Untersuchungen theoretischer Eigenschaften fallbasierter Klassifikatoren wurden z. B. in [26] angegeben. Da dem fallbasierten Klassifikator nicht direkt anzusehen ist, welche Begriffe dieser repräsentiert, wurde ein geometrisches

Modell der Klassifikation vorgestellt. Voronoi-Diagramme liefern dabei ein gutes theoretisches Modell, daß für praktische Anwendungen eher ungeeignet ist, da in realen Anwendungen viele Dimensionen benötigt werden. Effiziente Algorithmen zur Berechnung von Voronoi-Diagrammen sind aber nur für zwei bzw. drei Dimensionen bekannt.

Zum Schluß wurden Lernfunktionen in Form von Aufnahmestrategien und Änderungsstrategien vorgestellt. Für die verschiedenen Strategien wurden Ergebnisse empirischer Untersuchungen angeführt. Genauere Darstellungen findet man z. B. in [27]. Die dort erzielten Ergebnisse sind eher theoretischer Art. Der praktische Erfolg eines fallbasierten Ansatzes in einer konkreten Anwendungsdomäne ist stark abhängig von:

- den Möglichkeiten, für das in einer Anwendungsdomäne bereits vorhandene Wissen formal ein adäquates Maß anzugeben, um so einen effizienten Lernprozeß zu ermöglichen und
- dem relativ hohen Berechnungsaufwand fallbasierter Verfahren beim Retrieval von ähnlichen Fällen – dieser ist auf ein in der Praxis akzeptables Ausmaß zu minimieren.

In der Dissertation von S. Wess [17], mit deren Kapiteln 4 und 5 sich dieses Kapitel beschäftigt hat, wird durch das dort beschriebene INRECA-System eine praktische Umsetzung der hier erwähnten Ergebnisse aufgeführt. Außerdem behandeln zwei weitere Kapitel derselben die Beschleunigung des Retrievalvorganges.

Kapitel 3

Retrieval

3.1 Definition

Bei der Betrachtung von Ähnlichkeiten auf Fallbasen[1] unterscheidet man folgende Ebenen:

- Anwendungsebene: SIM_{App} beschreibt die in der Anwendung existierende Ähnlichkeit. In einer Fallbasis, in der Autos gespeichert sind, nennt man z. B. zwei Autos ähnlich, falls sie die gleiche Farbe, Form und Motorleistung haben. Dabei spielt die Repräsentation der Attribute Farbe, Form und Motorleistung keine Rolle.
- Repräsentationsebene: SIM_{Rep} beschreibt die Repräsentation der Ähnlichkeiten in dem entsprechenden fallbasierten System. Es findet eine Modellierung der Ähnlichkeiten der Ebene SIM_{App} statt, um diese innerhalb des fallbasierten Systems zu verwenden. Bezüglich des Autobeispiels muß man also eine passende Beschreibung für die Farbe, Form und Motorleistung von Autos finden, damit die jeweiligen Fälle in einem fallbasierten System verwendet werden können.
- Implementierungsebene: SIM_{Imp} beschreibt die operationale Umsetzung der Ähnlichkeiten der Repräsentationsebene. Hier wird eine konkrete Formel entwickelt, mit der die Ähnlichkeit zweier Autos ermittelt werden kann. Die Darstellungsform der Repräsentationsebene ist dabei so zu wählen, daß eine operationale Umsetzung der Ähnlichkeit überhaupt möglich ist.

Definition 3.1 (Retrieval, Abfragen) *Beim Retrieval (Abfragen) wird zu einem konkreten Problem(-fall) Q auf der Basis der Ähnlichkeit der Repräsentationsebene SIM_{Rep} die Menge der m $(m \geq 1)$ ähnlichsten Fälle aus der Fallbasis (Case Base) gesucht. Eine Retrievalprozedur ist ein Algorithmus der mit Hilfe einer Ähnlichkeitsbeschreibung auf der Ebene SIM_{Imp} zu einer Anfrage Q die $m \in \mathbb{N}$ ähnlichsten Fälle heraussucht.*

Man kann Retrieval mit Abfragen übersetzen. Oftmals ist in der Literatur davon die Rede, daß zu einem konkreten Fall der ähnlichste Fall gesucht wird. Dies ist aber regelmäßig nicht korrekt, weil ein Fall definitionsgemäß aus einer Problembeschreibung und einer dazu gehörenden Lösung (Klassifikation) besteht: Fall = Problem + Lösung. Da die Anfrage Q, zu der ein maximales Element gesucht wird, aber regelmäßig keine Lösungsbeschreibung enthält, ist es besser, hier von einer Anfrage (Query) oder von einem Problem(-fall) zu sprechen. Der Wert m für die Zahl der gesuchten ähnlichen Fälle ist entweder explizit vorgegeben (z. B. finde die 50 Autos deren Motorleistung dem Auto Q am ähnlichsten ist) oder durch die Form der Anfrage bestimmt (z. B. finde alle Autos, deren Motorleistung maximal um 20% von der Motorleistung des Autos Q abweicht).
Dabei müssen folgende Voraussetzungen gelten:

- Für die Anwendungsdomäne existiert bereits eine Repräsentation eines geeigneten Ähnlichkeitsbegriffes SIM_{Rep}, der durch die Retrievalprozedur implementiert wird.
- Für den Ähnlichkeitsbegriff SIM_{Rep} existiert bereits ein Verfahren, um aus drei Fällen $F1$, $F2$ und Q die Relation $SIM_{Rep}(F1, Q) > SIM_{Rep}(F2, Q)$ bzw. $F1 >_Q F2$ effizient zu bestimmen.

[1] M. Lenz (1999) unterscheidet zwischen „case base“ und „case memory“: „The case base is merely a set of cases. A case memory, in contrast, is a structured case base, i. e. a case base which, for retrieval purposes, is organized according to a particular memory structure.“ [38], p. 18

Das Retrieval ist durchaus mit Suchproblemen oder ähnlichen Aufgabenstellungen aus dem Bereich der Datenbanken zu vergleichen. Die effiziente Implementierung des Retrievals ist elementar für die Laufzeit und die Funktionalität des Gesamtsystems.

An eine Retrievalprozedur werden folgende Anforderungen gestellt:

- Korrektheit: Ein Verfahren heißt korrekt, falls die durch die Implementierung beschriebene Ähnlichkeitsrelation mit der in der Repräsentation festgelegten Ähnlichkeitsrelation übereinstimmt. Formal: Für alle a, b, c aus einer Fallbasis U gilt: Falls $SIM_{Imp}(a,c) > SIM_{Imp}(b,c) \Rightarrow SIM_{Rep}(a,c) > SIM_{Rep}(b,c)$. Bei einem korrekten Verfahren wird durch die Implementierung die gleiche Ordnung definiert wie durch die Repräsentation.
- Vollständigkeit: Ein Verfahren heißt vollständig, falls alle Ähnlichkeiten aus der Repräsentationsphase auch in der Implementierung gelten. Formal: Für alle a, b, c aus einer Fallbasis U gilt: Falls $SIM_{Rep}(a,c) > SIM_{Rep}(b,c) \Rightarrow SIM_{Imp}(a,c) > SIM_{Imp}(b,c)$. Bei einem vollständigen Verfahren werden alle Relationen der Repräsentationsebene durch die Implementierung umgesetzt.

In bestimmten Anwendungen kann es möglich sein, daß Korrektheit und Vollständigkeit zwingend erforderlich sind. Auf der anderen Seite kann es aber auch sinnvoll sein, die Vollständigkeit für eine effiziente Heuristik zu opfern. Die Begriffe Korrektheit und Vollständigkeit können auch auf mehr als zwei Fälle übertragen werden.

3.2 Elementare Retrievalverfahren

Folgende Retrievalverfahren sind von elementarer Bedeutung und können universell eingesetzt werden:

- Sequentielles Retrieval: jeder Fall der Fallbasis wird einzeln mit der Anfrage verglichen.
- Relationales Retrieval: Vorauswahl und Retrieval auf der Vorauswahl.
- Indexorientiertes Retrieval: Retrieval mit Hilfe einer Indexstruktur.

In den folgenden Abschnitten werden diese Verfahren im einzelnen beschrieben und bewertet.

3.2.1 Sequentielles Retrieval

Beim Sequentiellen Retrieval wird die gesamte Fallbasis sequentiell mit der Anfrage verglichen und alle Fälle, die in Frage kommen, werden ihrer Ähnlichkeit entsprechend, in einer Liste abgespeichert. Dieses Verfahren dient als das grundlegende Basisverfahren und damit als Referenz für die Bewertung der anderen Verfahren.

Datenstruktur

```
TYPE
      CaseType         = ...
      SimilarityType   = ...
      SimCase          = RECORD
            case       = CaseType
            similarity = SimilarityType
      END

VAR
      SimCaseQueue     = ARRAY [1..m] of SimCase
      QueryCase        = CaseType
      Case             = CaseType
      CaseBase         = ARRAY of Case
```

Die genaue Definition von CaseType und SimilarityType hängt von der konkreten Anwendung ab und soll hier nicht näher betrachtet werden. Die Datenstruktur SimCase besteht aus einem konkretem Fall und einer Ähnlichkeit zu

einem anderen Fall (i.d.R. der Anfrage Q). Diese Datenstruktur wird für SimCaseQueue verwendet, in dem die m ähnlichsten Fälle und die Ähnlichkeit zur Anfrage Q gespeichert werden. Die ähnlichsten Fälle werden, entsprechend ihrer Ähnlichkeit zur Anfrage Q, in SimCaseQueue gespeichert.

Algorithmus:

```
PROCEDURE SearchSeq(CaseBase, QueryCase, m):SimCaseQueue
VAR i INT
BEGIN
      SimCaseQueue[1..m].similarity := 0
      FOR i := 1 TO n DO
      IF SIMImp(QueryCase,CaseBase[i]) > SimCaseQueue[m].similarity
SimCaseQueue := SimCaseQueue + CaseBase[i]
      END
      RETURN (SimCaseQueue)

END
```

Der Algorithmus betrachtet jeden Fall in der Fallbasis und vergleicht diesen mit der Anfrage Q (=QueryCase). Falls dieser Fall ähnlicher ist als der bisher am wenigsten ähnliche Fall (SimCaseQueue[m]) in SimCaseQueue, wird der Fall in das Array eingefügt und der letzte Fall aus dem Array geschoben. Aus Gründen der Einfachheit geht man hier davon aus, daß die Operation + automatisch für eine Einordnung der Fälle in das Array der bisher gefundenen Fälle sorgt. Für die Einordnung der Fälle können beliebige Sortier- oder Einfügealgorithmen verwendet werden, die hier nicht näher betrachtet werden.

Bewertung

Das Sequentielle Retrieval kann universell eingesetzt werden und ist einfach zu realisieren. Die Performanz hängt stark von der Funktion SIM_{Imp} ab, da jeder Fall der Fallbasis mit der Anfrage verglichen und ggf. in die Queue eingeordnet werden muß.

Vorteile

- Das Verfahren ist vollständig und korrekt, da alle Fälle überprüft werden.
- Einfache Implementierung, weil das gesamte Verfahren auf einer Schleife basiert, mit der die gesamte Fallbasis durchlaufen wird.
- Keine Anforderungen an das Ähnlichkeitsmaß, da weder eine Ordnung der Fälle noch eine Klassifizierung für das Retrieval nötig ist.
- Spontane Anfragen sind möglich, da kein Vorlauf für die Anfrage nötig ist. Dies ermöglicht zu jedem Zeitpunkt beliebige Anfragen.

Nachteile

- Schlechte Performanz bei komplizierter Ähnlichkeitsfunktion, da jeder Fall mit der Anfrage verglichen werden muß und ggf. in die Queue der bereits gefundenen ähnlichsten Fälle einzuordnen ist.
- Konstanter Aufwand unabhängig von der Menge der gewünschten Ergebnisse, weil immer alle Fälle mit der Anfrage verglichen werden, auch wenn nur wenige Fälle als Ergebnis benötigt werden.
- Keine Beschränkung des Aufwands möglich, da vorhandenes Wissen über die Struktur der Fallbasis, wie z. B. Häufungen, nicht zur Minimierung der Vergleiche eingesetzt werden kann.

Trotz der einfachen Implementierung und der linearen Laufzeit ist dieses Verfahren für die meisten praktischen Anwendungen nicht geeignet, da die Suche in großen Fallbasen mit komplizierten Ähnlichkeitsmaßen zu viel Aufwand erfordert.

3.2.2 Relationales Retrieval

Das Relationale Retrieval kann als natürliche Erweiterung des Sequentiellen Retrievals verstanden werden. Hierbei wird in einem Vorlauf eine geeignete Menge von Fällen ausgewählt, die als Basis für das eigentliche Retrieval dienen. Diese Vorauswahl funktioniert wie ein binärer Filter der erheblich effektiver als die Funktion SIM_{Imp} arbeitet, da nicht eine Ähnlichkeit berechnet wird, sondern nur eine abstrakte Ähnlichkeit geprüft wird oder ganze Klassen von Fällen ausgewählt werden. Im Idealfall arbeitet die Vorauswahl so gut, daß genau m Fälle ausgewählt werden, die dann im zweiten Schritt durch die Änlichkeitsfunktion SIM_{Imp} bewertet und angeordnet werden. Die optimale Vorauswahl ist in der Praxis nicht realisierbar, aber eine Einschränkung der möglichen Fälle kann erreicht werden. Die Vorauswahl ist entscheidend für die Korrektheit und die Vollständigkeit des Relationalen Retrievals. Falls die Vorauswahl zuwenig Fälle aussucht, ist das spätere Retrieval nicht vollständig, weil keine weiteren Fälle hinzugefügt werden. Auf der anderen Seite muß durch die Vorauswahl die Menge der Fälle, die mit der Anfrage verglichen werden, erheblich eingeschränkt werden, damit ein Performanzvorteil gegenüber dem Sequentiellen Retrieval vorliegt.

Mögliche Strategien für die Vorauswahl

Folgende grundsätzliche Strategien sind Ansätze für eine Vorauswahlstrategie. Die Anfrage für alle Beispiele sei: Limousine, grün, 2000 ccm Hubraum, jünger als 6 Jahre .

- **Gleichheit:** Ein Fallbeispiel stimmt in allen Merkmalsausprägungen mit der Anfrage überein. Es werden nur grüne Limousinen mit genau 2000 ccm Hubraum, die jünger als 6 Jahre sind, ausgewählt. Dies ist eine triviale Vorauswahl, die nur dann in Frage kommt, falls viele gleichartige Fälle in der Fallbasis vorliegen.
- **Partielle Gleichheit:** Ein Fallbeispiel stimmt in einer oder mehreren Ausprägungen mit der Anfrage überein. Hier werden z.B. alle Limousinen ausgewählt, die 2000 ccm Hubraum haben aber auch älter als 6 Jahre sein können und nicht unbedingt grün sind. Dabei muß darauf geachtet werden, daß nur wichtige Attribute für die Vorauswahl genommen werden.
- **Lokale Ähnlichkeit:** Lokale Ähnlichkeit ist die Ähnlichkeit zwischen einzelnen Merkmalen zweier Fälle. Man sagt, ein Fall ist lokal ähnlich zur Anfrage, falls dessen einzelne Merkmale sehr ähnlich zu den einzelnen Merkmalen der Anfrage sind. Im Beispiel kommen alle Limousinen und Caravans mit grünlichem Farbton, 1800 ccm - 2200 ccm Hubraum in Frage, die zudem jünger als 7 Jahre sind.
- **Partielle lokale Ähnlichkeit:** Ein Fallbeispiel ist in einer oder mehreren Merkmalsausprägungen sehr ähnlich zu den einzelnen Merkmalen der Anfrage. In diesem Fall werden alle Limousinen und Caravans mit einem Hubraum zwischen 1800 ccm und 2200 ccm ausgewählt. Die Farbe und das Alter spielen für die Vorauswahl keine Rolle.

Die partielle Ähnlichkeit unterscheidet sich von der allgemeinen Ähnlichkeit durch die Betrachtung der Merkmale im einzelnen. Bei der lokalen Ähnlichkeit werden die Merkmale direkt miteinander verglichen; bei der allgemeinen Ähnlichkeit ist es möglich, einen abstrakten Wert aus allen Merkmalen zu bilden, der die Grundlage für die Änlichkeitsbewertung ist. Durch eine Gewichtung einzelner Merkmale kann erreicht werden, daß die Vorauswahl bessere Ergebnisse liefert. Es ist z. B. möglich, daß für eine Anfrage die Farbe des Autos eine untergeordnete Rolle spielt. Diese Vorauswahlstrategien können auch direkt mit Hilfe einer Datenbank realisiert werden, falls die Fallbasis in einer entsprechenden Form vorliegt. Andere Vorauswahlstrategien sind denkbar (z.B. wähle alle Fälle aus, die zu einer bestimmten Klasse von Fällen gehören), falls zusätzliches Wissen über die Struktur der Fallbasis zur Verfügung steht. Dieses zusätzliche Wissen ist entweder Expertenwissen mindestens einer Person oder aber eine Klassifizierung, die durch einen Clusteringalgorithmus gewonnen wurde. Dabei werden als Vorauswahl diejenigen Klassen von Fällen, deren symbolische Beschreibung der Anfrage ähnlich sind, ausgewählt. Einige Methoden für Klassenbildung werden in dem Abschnitt „Clustering“ beschrieben.

Bewertung

Das Relationale Retrieval ist eine Erweiterung des sequentiellen Retrievals, die nur dann einen Performanzvorteil aufweisen kann, wenn die Vorauswahl effektiv arbeitet. Vorhandenes Expertenwissen oder Ergebnisse eines Clusteringalgorithmus können die Vorauswahl verbessern.

Vorteile

- Performanzvorteil bei guter Vorauswahl.
- Retrievalaufwand ist abhängig von der Größe m der Anfrage, weil eine kleine Anzahl von Ergebnissen auch nur eine kleine Vorauswahl ermöglicht.
- Vorhandenes Wissen über die Struktur der Fallbasis kann für die Vorauswahl genutzt werden, um z. B. Klassen von Fällen oder Häufungen in der Fallbasis für die Vorauswahl zu nutzen.
- Spontananfragen beschränkt möglich, da die Vorauswahl einen geringen Aufwand darstellt.

Nachteile

- Retrievalfehler durch eine schlecht organisierte Vorauswahl sind möglich.
- Der Performanzvorteil ist nicht in jedem Fall gesichert, da unter Umständen die Vorauswahl so aufwendig ist, daß das Sequentielle Retrieval schneller ein Ergebnis liefert.
- Die Bestimmung der Vorauswahl ist kompliziert und hängt von der Fallbasis ab. Eine optimale Vorauswahlstrategie für alle Fallbasen liegt nicht vor, und ist auch nicht zu erwarten.

3.2.3 Indexorientiertes Retrieval

Das Indexorientierte Retrieval ist eine weitere Möglichkeit, unnötige Zugriffe auf Fälle, die als Ergebnis nicht in Frage kommen, zu verhindern. Anders als beim Relationalen Retrieval wird allerdings nicht mit einer Vorauswahl gearbeitet, sondern es wird eine Zugriffsstruktur für alle Fälle erstellt, mit deren Hilfe gezielt auf einzelne Fälle zugegriffen werden kann. Dieser Index muß allerdings ausreichend viel Informationen über die Fälle enthalten, damit die Retrievalprozedur über den Index auf die Fälle zugreifen kann.

Algorithmus

```
PROCEDURE SearchIDX (CaseBase,QueryCase,m) : SimCaseQueue;
BEGIN
      RETURN (INDEX(SearchSeq(IDXBase,QueryCase,m), QueryCase,m));
END
```

Die Retrievalprozedur durchsucht die Fallbasis mit Hilfe der Indexstruktur und liefert eine Liste von gefundenen Fällen zurück. Als Rückgabewert werden die konkreten Fälle ausgegeben, die durch die Funktion INDEX ermittelt wurden.

Bewertung

Das Indexorienterte Retrieval ist besonders interessant, wenn die Fallbasis in Form einer Datenbank vorliegt. Durch ein Datenbanksystem kann eine Indexstruktur in geeigneter Form generiert werden, und die Retrievalprozedur findet mit diesem Datenbanksystem die m ähnlichsten Fälle.

Vorteile

- Retrieval mit einem Datenbanksystem möglich.
- Beschleunigung der Retrievalprozedur durch schnelleren Zugriff.
- Minimierung der Fallvergleiche.

Nachteile

- Indexstruktur nicht immer trivial.
- Spontanabfragen nur begrenzt möglich.
- Einfügen neuer Fälle verursacht Neugenerierung der Indexstruktur.

3.3 Clustering

In vielen Fällen ist es erforderlich, die Fallbasis in Klassen zu unterteilen (z. B. für die Vorauswahl beim Relationalen Retrieval). Falls eine Klassifizierung der Daten durch Wissen über die Struktur der Fälle nicht vorhanden ist, kann mit Hilfe des Clustering eine Klassifizierung gefunden werden. Ziel des Clustering ist die Klassenbildung, d. h. eine sinnvolle Zuordnung der Fälle zu Klassen und eine symbolische Beschreibung der einzelnen Klassen, die möglichst präzise die Fälle einer Klasse abbildet. Die Klassen sind Teilmengen der Fallbasis, deren Fälle besonders ähnlich zueinander sind. Im einfachsten Fall des Clusterings haben alle Fälle einer Klasse für ein Attribut den gleichen Wert. In diesem Fall werden die Klassen durch direkten Vergleich ermittelt, und die Beschreibung einer Klasse ist der gemeinsame Attributwert dieser Klasse. Wenn die Klassifizierung auf Basis der Ähnlichkeit SIM_{Imp} ermittelt wird, sind kompliziertere Algorithmen nötig. Als Verfahren, die auf der Funktion SIM_{Imp} basieren, soll das Clustering auf der Basis des minimalen Spannbaums als Beispiel für graphorientiertes Clustering und dynamisches Clustering vorgestellt werden. Für die Beschreibung einer Klasse können entweder Repräsentanten der Klasse angegeben werden oder aber eine symbolische Beschreibung, die aus allen Fällen einer Klasse ermittelt wird. Die symbolische Beschreibung ist in diesem Fall nicht immer trivial. Denkbar ist die Angabe eines Intervalls, falls die Klassen so einfach aufgebaut sind, oder die Angabe von Mittelpunkt und Radius. Der Mittelpunkt einer Klasse ist ein abstrakter Fall, der als Attributwert den Mittelwert aller Attributwerte der Fälle dieser Klasse hat (sofern ein Mittelwert berechenbar ist). Der Radius ist die Ähnlichkeit, berechnet mit der Funktion SIM_{Imp}, zwischen dem Mittelpunkt – Anfrage – und dem Fall, der dem Mittelpunkt am wenigsten ähnlich ist.

3.3.1 Clustering auf der Basis eines Attributwertes

Falls es ein Attribut gibt, das signifikant für die Beschreibung eines Falles ist, kann das Clustering auf dieses Attribut beschränkt werden. Im einfachsten Fall ist die Menge der möglichen Werte begrenzt und abzählbar. Für jeden möglichen Attributwert wird eine Klasse gebildet, die durch ihren Attributwert eindeutig beschrieben werden kann. In diese Klasse werden alle Fälle eingeordnet, die den entsprechenden Attributwert haben. Es entsteht für jeden möglichen Attributwert eine Klasse, wodurch dieses Verfahren auf Fallbasen beschränkt ist, die in einem wichtigen Attribut eine begrenzte und abzählbare Menge von Werten annimmt.

Eine Variante diese Verfahrens ist möglich, falls der Wertebereich eines Attributes in Intervalle aufgeteilt werden kann. Die Anzahl der gewünschten Klassen dient als Grundlage für die Größe des einzelnen Intervalls. Jedes Intervall wird durch seine Grenzen beschrieben und besteht aus den Fällen, deren Attributwerte in dem Intervall liegen. Diese Form des Clustering ist immer dann möglich, wenn das signifikante Attribut auf ein numerisches Intervall abgebildet werden kann.

Vorteile

- einfach zu implementieren
- lineare Laufzeit

Nachteile

- Ein Attribut muß die Fälle genau beschreiben, da die anderen Attribute das Clustering nicht beeinflussen können.
- Attributwerte müssen endlich und abzählbar bzw. teilbar sein.

3.3.2 Clustering mit der Funktion SIM_{Imp}

Bei dieser Form des Clustering werden keine besonderen Voraussetzungen an die Fallbasis gestellt. Es wird nur verlangt, daß die Funktion SIM_{Imp} eine Ordnung der Fallbasis definiert, damit bestimmt werden kann, ob für drei Fälle A, B, X gilt: $SIM_{Imp}(a, x) > SIM_{Imp}(b, x)$. Ziel ist die Bildung von Teilmengen mit Fällen, die bezüglich SIM_{Imp} zueinander ähnlich sind und eine symbolische Beschreibung dieser Teilmengen, die für alle Fälle dieser Teilmenge treffend ist. Folgende Grundformen des Clustering sollen in diesem Abschnitt kurz beschrieben werden:

- Clustering durch einen minimalen Spannbaum:
  Bei dieser Form des Clustering wird ein minimaler Spannbaum über alle Fälle erzeugt und hieraus bestimmte Kanten entfernt. Die zusammenhängenden Bäume, die übrig bleiben, bilden die einzelnen Cluster.
- Dynamisches Clustering:
  Beim dynamischen Clustering werden in einer initialen Prozedur Klassen aus einzelnen Elementen gebildet und schrittweise, durch Vereinigung mit anderen Klassen oder einzelnen Fällen, vergrößert.

Für beide Verfahren kann eine Klasse durch den Mittelpunkt und ihren Radius beschrieben werden, falls durch externes Wissen keine bessere Beschreibung möglich ist.

Clustering mit minimalem Spannbaum

Grundlage für diese Form des Clustering ist ein minimaler Spannbaum über der Fallbasis. Eine Kante zwischen zwei Fällen wird mit einer Ähnlichkeit, die mit SIM_{Imp} ermittelt wird, gewichtet. Ziel ist ein Graph, der alle Fälle enthält, und dessen Kanten möglichst hohe Ähnlichkeit haben. Die Bezeichnung minimaler Spannbaum ist als minimaler Abstand bezüglich der Funktion SIM_{Imp} der Fälle zu verstehen. Dazu bietet sich der folgende Greedy-Algorithmus an:

1. Wähle einen beliebigen Fall aus und betrachte diesen als Graph.
2. Ermittle die Ähnlichkeit zwischen den Fällen innerhalb des Graphen und außerhalb des Graphen. Füge den Fall mit der größten Ähnlichkeit in den Graphen ein.
3. Wiederhole Schritt 2 bis keine Fälle mehr übrig sind.

Die Klassen werden gebildet, indem bestimmte Kanten aus diesem minimalen Spannbaum entfernt werden. Die Kanten, die entfernt werden, zeichnen sich entweder durch besondere Länge oder durch die Knoten aus, die durch die Kanten verbunden werden. Dadurch entstehen einzelne zusammenhängende Bäume, die dann die Cluster bilden. Als Beschreibung der Klassen dienen Mittelpunkt und Radius.

Bei der einfachsten Form des Clustering mit einem minimalen Spannbaum werden Kanten mit besonders geringer Ähnlichkeit entfernt. Diese Kanten zeichnen sich durch besondere Länge aus und können ermittelt werden, indem man die durchschnittliche Länge aller Kanten betrachtet und alle Kanten entfernt, die z.B. doppelt so lang wie die durchschnittliche Länge sind. Die Teilbäume enthalten nur noch Fälle, die eine gewisse Mindestähnlichkeit zueinander haben.

In der nachfolgenden Abbildung sind die Punkte die einzelnen Fälle und die Länge der Kante steht für die Ähnlichkeit. Kurze Kanten stehen für hohe Ähnlichkeit. Durch die Trennung entstehen zwei neue Bäume, die zwei neue Klassen symbolisieren.

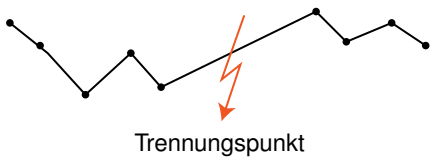


Abbildung 3.1: Beispiel für einen minimalen Spannbaum

Diese Form des Retrievals funktioniert allerdings nur, wenn Häufungen von Fällen weit genug voneinander getrennt sind. Bei fast gleichverteilten Fällen oder bei Häufungen, die dicht nebeneinander liegen, versagt diese Form des

Clustering, weil keine besonders lange Kanten vorliegen. Die Wahl der Kantenlänge, die als Mindestmaß für das Entfernen dient, ist ein weiteres Problem. Falls kein sinnvolles Mindestmaß für die Fallbasis vorliegt, ist in vielen Fällen nur Ausprobieren und anschließendes Bewerten der Klassifizierung möglich. Für die Bewertung kann die Anzahl der Klassen und die Verteilung der Fälle auf die Klassen dienen.

Für Häufungen, die dicht nebeneinander liegen, müssen die einzelnen Fälle (Knoten im Graph) bewertet werden. Bei dieser Bewertung wird die Anzahl der Fälle, die eine gewisse Mindestähnlichkeit haben, als Wert für einen Fall genommen. Als Mindestähnlichkeit kommt z. B. die durchschnittliche Ähnlichkeit aller Fälle in Frage. Der Fall, der die wenigsten Nachbarfälle hat, ist der Trennungspunkt, an dem der minimale Spannbaum aufgeteilt wird und neue Klassen entstehen. Da auch Randpunkte nur wenige Fälle in unmittelbarere Nachbarschaft haben, ist eine Bewertung der Verteilung der Fälle auf die neuen Klassen notwendig.

In der folgenden Abbildung symbolisieren die Punkte einzelne Fälle. Würde man für diese Punkte einen minimalen Spannbaum berechnen, so wären alle Kanten fast gleich lang. Es ist nicht möglich, nur mit Hilfe der Kantenlänge die beiden Häufungen zu erkennen. Nur durch die Bewertung der einzelnen Fälle können mögliche Trennungspunkte gefunden werden, die anschließend bewertet werden müssen. Randpunkte ergeben Klassen mit sehr wenigen und sehr vielen Fällen und können auf diese Weise identifiziert werden. Klassen mit vielen Fällen sind z.B. Klassen, die mehr als die durchschnittliche Menge an Fällen enthalten. Diese Form des Clustering liefert sehr gute Ergebnisse, falls

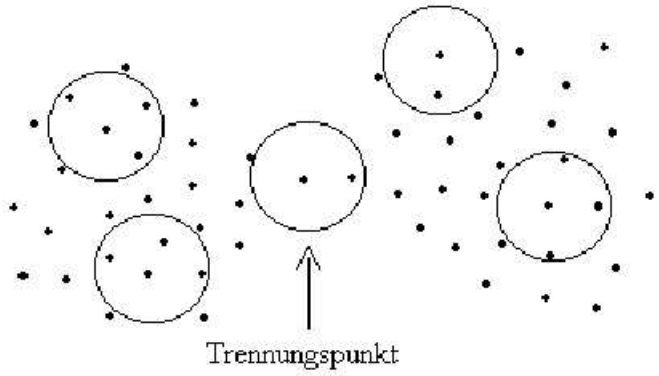


Abbildung 3.2: Beispiel für nebeneinander liegende Häufungen

in der Fallbasis Häufungen sind und diese zudem weit getrennt sind. Falls der Spannbaum keine besonders langen Kanten aufweist, ist eine Bewertung der einzelnen Fälle notwendig. Bei gleichverteilten Fällen versagt diese Form des Clustering, da keine Strukturinformationen durch den minimalen Spannbaum und die Bewertung der Fälle ermittelt werden kann.

Dynamisches Clustering

Beim dynamischen Clustering wird eine initiale Klassifizierung angenommen, die schrittweise, durch Vereinigung der Cluster oder hinzufügen einzelner Fälle zu den Clustern, verbessert wird. Bei der einfachsten Form der initialen Klassifizierung wird jeder Fall oder eine zufällige Auswahl einzelner Fälle eine Klasse.

Bei der Klassenvereinigung gibt es verschiedene Kriterien, nach denen die zu vereinigenden Klassen ausgewählt werden können. Entweder man vereinigt zwei Klassen deren Mittelpunkte am ähnlichsten zueinander sind oder man vereinigt zwei Klassen, falls zwei Fälle aus diesen Klassen ähnlich zueinander sind. Der Mittelpunkt einer Klasse ist ein Fall, der aus einem Mittelwert aller Attributswerte berechnet wird. Die Ähnlichkeit zweier Mittelpunkte kann mit der Funktion SIM_{Imp} bestimmt werden, da der Mittelpunkt ein abstrakter Fall ist. Falls einzelne Fälle als Grundlage für die Vereinigung von Klassen dienen, muß von allen Fällen einer Klasse die Ähnlichkeit zu den Fällen der anderen Klassen betrachtet werden. Es werden die beiden Klassen miteinander vereinigt, bei denen zwei Fälle einander am ähnlichsten sind. Dazu ist es sinnvoll die Ähnlichkeit aller Fälle zueinander in einer Dreiecksmatrix darzustellen, damit diese nicht in jedem Schritt neu berechnet werden muß.

Eine Variante dieses Verfahrens wäre, nicht nur zwei Klassen in einem Schritt zu vereinen, sondern alle Klassen in einem Schritt zu betrachten. In einem Schritt wird dann jede Klasse mit der für sie ähnlichsten Klasse vereinigt. Dies führt zu gleich großen Klassen und ist daher besonders für gleichverteilte Fallbasen vorteilhaft. Der Nachteil ist, daß

Häufungen nicht berücksichtigt und in vielen Fällen sogar zerstört werden.

Eine genauere Aufteilung erhält man, wenn nicht ganze Klassen miteinander vereinigt, sondern einzelne Fälle zu einer Klasse hinzufügt werden. Grundlage ist wieder eine initiale Klassenaufteilung – wie z. B. jeder Fall ist eine Klasse – oder eine zufällige Auswahl von n Fällen, die die Klassen bilden. In jedem Schritt werden die Mittelpunkte der Klassen ermittelt und die Ähnlichkeit zwischen dem Mittelpunkt einer Klasse und allen Fällen aus den anderen Klassen und den nicht zugeordneten Fällen mit SIM_{Imp} berechnet. Der Fall, der dem Mittelpunkt einer anderen Klasse am nächsten ist, wird in diese Klasse übernommen. Um eine Terminierung des Algorithmus zu garantieren, werden grundsätzlich kleine Klassen zugunsten der größeren aufgelöst. Falls zwei Klassen gleichviele Elemente haben, wird die Klasse aufgelöst, deren Summe aller Ähnlichkeiten der Fälle der Klasse mit dem Mittelpunkt am kleinsten ist[2]. Wenn dies nicht der Fall ist, muß eine andere Möglichkeit gefunden werden, um die Ähnlichkeit aller Fälle einer Klasse bewerten zu können. Der Vorteil dieses Verfahrens ist die genauere Klassifizierung der Fallbasis. Es kann allerdings zu einer entarteten Klassifizierung kommen, mit einigen sehr großen Klassen und vielen sehr kleinen.

Diese Verfahren terminieren, falls eine bestimmte Anzahl von Klassen bestimmt ist und alle Fälle einer Klasse zugeordnet sind oder eine bestimmte Anzahl von Schritten durchlaufen ist. Falls die Klassenanzahl oder die maximale Schrittanzahl nicht vom Benutzer angegeben werden oder durch externes Wissen bestimmt werden, können diese auch anhand der Anzahl der Fälle berechnet werden wie z.B. 10 % der Anzahl der Fälle ist die Obergrenze für die Anzahl der Klassen und die Anzahl der Fälle ist die Obergrenze für die Anzahl der Vereinigungsschritte. Die oben genannten Verfahren liefern, besonders bei Verteilungen ohne besondere Häufungen, bessere Ergebnisse als das Clustering mit minimalen Spannbäumen.

Clustering stellt in jedem Fall einen hohen Aufwand dar, der besonders bei Fallbasen, die ständig geändert werden oder eine komplizierte Ähnlichkeitsfunktion erfordern, berücksichtigt werden muß. Änderungen der Fallbasis erfordern eine Änderung der Klassen und im Einzelfall ein vollständig neues Clustering. Andererseits liefert das Clustering eine automatische Aufteilung der Fallbasis, die z. B. für die Vorauswahl beim Relationalen Retrieval vorteilhaft sein kann.

3.4 Retrieval mit kd-Bäumen

Der k-dimensionale Suchbaum ist eine natürliche Erweiterung von binären Suchbäumen auf k Dimensionen. Beim binären Suchbaum wird jeder Schlüssel durch genau einen Wert charakterisiert. In jedem Knoten wird der Wertebereich in zwei Teile aufgeteilt und die Suche ist beendet, falls entweder ein Blatt erreicht wird oder der gesuchte Wert in einem Knoten gefunden wird.

Der kd-Baum bietet eine passende Baumstruktur für Schlüssel, die durch k ($k > 1$) Attribute charakterisiert werden. Auf einer Ebene des kd-Baumes werden nur Attributwerte einer bestimmten Dimension $i \in I\!N$ des Diskriminator betrachtet. Das bedeutet, daß in einem Knoten der Wertebereich nach einem Attributwert p_i in der Dimension $i \in I\!N$, dem Partitionswert, aufgeteilt wird. Die Dimension i wird auch als Diskriminatorattribut bezeichnet. Bei der Wahl des Diskriminatorattributes muß darauf geachtet werden, daß alle Dimensionen gleichberechtigt behandelt werden und daß der Datenraum optimal aufgeteilt wird. Hierzu werden unten später einige grundlegende Verfahren vorgestellt. Im Gegensatz zum binären Suchbaum stehen beim kd-Baum die Werte nicht in den Knoten, sondern nur in den Blättern, den sogenannten Buckets des kd-Baumes. Die Suche in einem kd-Baum endet grundsätzlich in einem Bucket, unabhängig davon, ob der gesuchte Wert gefunden wurde oder nicht.

Definition 3.2 (kd-Baum) *Gegeben seien k geordnete Wertebereiche W_j der Attribute $A_j, j = 1, \ldots, k$ und ein Datenraum $D = W_1 \times \ldots \times W_k$ mit $j, k \in I\!N$. Es sei $CB \subseteq D$ eine Menge von Fällen $F_i = (v_1, v_2, \ldots, v_k)$. Weiter sei ein Parameter $b \geq 1$ für die Bucketgröße fest gewählt ($b \in I\!N$). Ein kd-Baum $T^i(CB)$ für eine Menge CB ist definiert durch:*

- *Gilt $n \leq b$, so ist $T^i(CB)$ ein Blattknoten, der alle $F \in CB$ enthält ($i := \emptyset$).*
- *Gilt $n > b$, dann bezeichne $i \in \{1, \ldots, k\}$ die Dimension, in der die Fälle aus CB partitioniert werden sollen. Die Wurzel von $T^i(CB)$ enthält dabei einen Wert $p_i \in W_i$ und einen Zeiger auf die beiden Teilbäume $T^{i_1}_{\leq}(CB_{\leq})$ und $T^{i_2}_{>}(CB_{>})$, wobei*

$$CB_{\leq} := \{F \in CB | v_i \leq p_i\};$$
$$CB_{>} := \{F \in CB | v_i > p_i\}.$$

[2]Da die Ähnlichkeit in der Regel auf numerische Werte abgebildbar ist, darf man hier davon ausgehen, daß eine Summe gebildet werden kann.

Der Wert b legt fest, wann der Partitionierungsprozeß terminiert, d. h. wieviele Fälle maximal in einem Blattknoten gespeichert werden können. Der Parameter b wird daher auch als Bucketgröße bezeichnet. Die gewählte Dimension i bezeichnet man als das Diskriminatorattribut A_i, und der Wert $p_i \in W_i$ wird als Partitionswert bezeichnet. Jeder Baumknoten $T^i(K)$ repräsentiert dabei eine Teilmenge $K \subseteq CB$ der im kd-Baum gespeicherten Fälle. Dabei gilt:

- *Die Wurzel $T^{R^*}(CB)$ des kd-Baumes repräsentiert die gesamte Fallbasis CB.*
- *Jeder Blattknoten $T^b(K)$ – Bucket – repräsentiert alle in ihm gespeicherten Fälle $F \in CB$, d. h. $K = \{F_1, \ldots, F_j\} \subseteq CB$ mit $j \leq b$.*
- *Jeder innere Knoten $T^i(K)$ repräsentiert alle Fälle F_i der entsprechenden Blattknoten, die in den durch den inneren Knoten aufgespannten Teilbäumen enthalten sind. Hierbei teilt ein innerer Knoten die Menge der jeweils betrachteten Gälle in zwei disjunkte Teilmengen $K_>$ und $K_\leq$ auf, wobei ein Sohn alle Fälle, deren Attributausprägungen kleiner oder gleich dem Partitionswert sind, und der andere Sohn alle übrigen Fälle erhält.*

Gespeichert werden Fälle aber nur in den Blättern T^b des kd-Baumes. Die inneren Knoten T^i dienen daher nur zur Patitionierung des betrachteten Datenraumes.

Beispiel Ein Beispiel eines 2d-Baumes sieht man in der Abbildung 3.3. Der Wertebereich dieses Beispiels ist zwei-

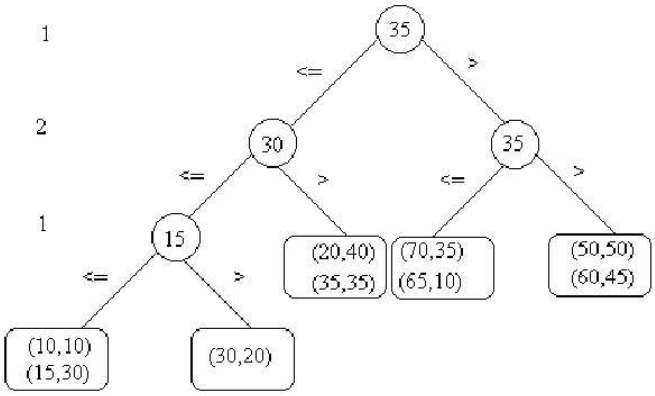


Abbildung 3.3: Beispiel für einen kd-Baum

dimensional und umfasst die Menge

$$\{(10,10),(15,30),(20,40),(35,35),(65,10),(70,35),(50,50),(60,45),(30,20)\}.$$

Die Diskriminatorattribute werden der Reihe nach ausgewählt. Der erste Diskriminatorattributwert a_{*1} beträgt 35. In dem linkem Teilbaum unterhalb der Wurzel werden alle Werte der Fälle eingefügt, die im ersten Attribut einen Wert $a_{i1} \leq 35$ haben, und in dem rechten Teilbaum unterhalb der Wuzel stehen alle Werte derjenigen Fälle, die im ersten Attribut einen Attributwert $a_{i1} > 35$ haben. In der zweiten Ebene des kd-Baumes gilt das zweite Diskriminatorattribut: $a_{*2} = 30$ für den linken Sohn des Wurzelknotens und $a_{*2} = 35$ für den rechten Sohn des Wurzelknotens. In beiden Knoten auf der zweiten Ebene werden die Teilbäume somit nach den Werten des zweiten Attributes gebildet. Auf der dritten Ebene gilt wieder der Diskriminatorattributindex 1, und in den Buckets stehen hier die entsprechenden Werte.

Durch diese Aufteilung repräsentiert jeder Knoten des Baumes eine Teilmenge des Wertebereiches, der im kd-Baum gespeichert ist:

- Die Wurzel repräsentiert den gesamten Wertebereich.
- Jedes Bucket repräsentiert die in ihm gespeicherten Werte.
- Jeder innere Knoten repräsentiert die Werte, die in den Buckets gespeichert sind, die zu dem durch den Knoten aufgespannten Teilbaum gehören.

Der kd-Baum liefert eine Datenstruktur für fallbasierte Systeme, auf der effiziente Verfahren zum Retrieval implementierbar sind. Bei fallbasierten Systemen ist die Fallbasis in der Regel ein Wertebereich, bei dem die Fälle die Werte repräsentieren und die einzelnen Eigenschaften der Fälle durch die Attribute repräsentiert werden. In den folgenden Abschnitten werden zuerst Einzelheiten zur Generierung beschrieben, da die Generierung des Baumes grundsätzlich unabhängig vom eigentlichen Retrieval ist.

3.4.1 Generierung von kd-Bäumen

Der Aufbau eines kd-Baums für eine bestimmte Fallbasis wird durch folgende Eckdaten eindeutig festgelegt:

- Die Wahl des Diskriminatorattributes legt fest in welcher Dimension auf einer Ebene des Baumes die Fallbasis aufgeteilt wird. Der triviale Ansatz ist einfach alle Dimensionen der Reihe nach zu betrachten. Dabei wird allerdings die Struktur der Daten nicht berücksichtigt. Mit Hilfe des Interquartilsabstands kann ein Diskriminatorattribut gefunden werden, das die Verteilung der Werte besser berücksichtigt.
- Mit dem Partitionswert wird festgelegt, welcher Attributwert zum Aufteilen auf einer bestimmten Ebene des Baumes ausgewählt wird. Auf der rechten Seite des Knotens werden alle Fälle eingeordnet, für die gilt: Attributwert $>$ Partitionswert; auf der linken Seite alle Fälle für welche gilt: Attributwert $\leq$ Partionswert. Der Wert muß so gewählt werden, daß ähnliche Fälle in den gleichen Teilbäumen gespeichert werden. Dabei können auf der einen Seite Ballungen von Werten berücksichtigt werden, auf der anderen Seite kann, bei gleichverteilten Daten, der Median einen guten Partitionswert liefern.
- Die Wahl der Bucketgröße legt fest, wann ein Blattknoten gebildet wird und dadurch die Generierung für diesen Teilbaum terminiert. Der triviale Ansatz bildet ein Bucket, falls nur noch eine bestimmte Menge von Fällen übrig ist. Falls weitere Informationen über die Fallbasis verfügbar sind, kann auch nach anderen Kriterien wie Klassenzugehörigkeit, die durch Clustering oder zusätzliches Wissen über die Fallbasis ermittelt wurde, die Generierung von Teilbäumen beendet werden.

Wahl des Diskriminatorattributes

Durch die Wahl des aktuellen Diskriminatorattributes wird jeweils festgelegt, in welcher Dimension die Fallbasis wie aufgeteilt wird. Am einfachsten ist es, alle Merkmale der Reihe nach als Diskriminatorattribut zu wählen. Dies hat den Vorteil, sehr einfach anwendbar zu sein, allerdings bleibt dabei die Struktur der Daten, wie z. B. die Verteilung der Werte in den einzelnen Dimensionen, unberücksichtigt. Die Wahl des Diskriminatorattributes soll ähnliche Fälle in einer möglichst frühen Phase der Generierung in gleiche Teilbäume und weniger ähnliche Fälle in verschieden Teilbäumen abspeichern. Dies hat den Vorteil, daß beim späteren Retrieval möglichst schnell in den richtigen Teilbaum verzweigt wird und so die Suche in benachbarten Teilbäumen minimiert wird.

Um die Streuung eines Wertebereiches abzuschätzen, bietet sich der Interquartilsabstand an. Dieses statistische Maß beruht auf dem Median und ist daher auf jeden Datentyp anwendbar, auf den durch die Funktion SIM_{Imp} eine Ordnung definiert ist. Der Median teilt einen Wertebereich in zwei Teile, indem das mittlere Element der geordneten Aufzählung aller Werte als Teilungspunkt genommen wird. Teilt man die beiden Teile wieder mit dem Median als Zentralwert, so erhält man vier Intervalle, die durch 3 Teilungspunkte markiert sind. Diese Teilungspunkte werden Quartile genannt. Zwischen den Quartilen kann mit Hilfe der Ähnlichkeitsfunktion SIM_{Imp} der Abstand bestimmt werden, da ein Quartil mindestens ein bestimmter Fall der Fallbasis ist. Der Interquartilsabstand ist der Abstand zwischen dem ersten und dem dritten Quartil. Bei fallbasierten Systemen kann dieser Abstand mit der Funktion SIM_{Imp} ermittelt werden. Dieses Maß kann für die Ermittlung des Attributes, nach dem der Wertebereich aufgeteilt werden

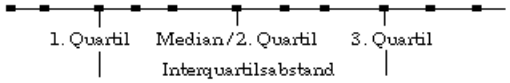


Abbildung 3.4: Ein Beispiel zum Interquartilsabstand

soll, verwendet werden. Dazu nimmt man von allen Fällen die Attributswerte einer Dimension und berechnet für diesen Wertebereich den Interquartilsabstand. Dies macht man mit allen Dimensionen und wählt die Dimension, deren Interquartilabstand am größten war, als Diskriminatorattribut. Da der Interquartilabstand etwas über die Streuung

der Werte aussagt, wird die Dimension mit der größten Streuung ausgewählt. Dadurch werden die Fälle möglichst gleichmäßig über den kd-Baum verteilt.

Bestimmung des Partitionswertes

Durch die Wahl des Diskriminatorattributes wird festgelegt, nach welchem Attribut in dieser Ebene der kd-Baum die Fallbasis teilt. Da die meisten Mittelwertberechnungen nicht für alle Formen von Daten anwendbar sind, bieten sich folgende Alternativen für alle Daten, auf denen eine Ordnung definiert ist:

- Median Splitting: Durch den Median wird die Datenmenge in zwei fast gleich große Teile aufgeteilt [39]. Durch den Median wird der kd-Baum bei gleich verteilten Werten also fast balanciert, wodurch die Suche nach exakten Werten beschleunigt wird. Allerdings können dadurch Ballungen von sehr ähnlichen Fällen zerstört werden und beim späteren Retrieval müssen weitere Teilbäume in die Suche nach ähnlichen Fällen einbezogen werden.
- Maximum Splitting: Um Ballungen von ähnlichen Fällen zu erhalten, ordnet man den Wertebereich und teilt die Wertemenge an der Stelle, an der die Lücke zwischen zwei Werten am größten ist. Dadurch bleibt die innere Struktur des Suchraumes erhalten, und bei der Suche nach m ähnlichen Fällen sind die Fälle in einem Teilbaum zu finden. In dem Beispiel der Abbildung 3.5 ergeben sich durch die Ballung der Punkte zwei mögliche Werte, um den Wertebereich zu splitten.

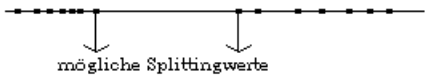


Abbildung 3.5: Ein Beispiel zum Maximum Splitting

Wenn man von gleich verteilten Daten ausgehen kann und fast ausschließlich wenige genau definierte Fälle sucht, dann ist das Median Splitting eine gute Strategie, da durch die ausgewogenen Suchbäume der Suchaufwand minimiert wird. Typische Fallbasen weisen in der Regel Ballungen auf und im allgemeinen interessieren den Anwender mehrere Fälle oder die Menge der Fälle, die eine bestimmte Nähe zur Anfrage aufweisen. Für diese Anwendungen ist das Maximum Splitting deutlich besser geeignet, weil ähnliche Fälle in einem Teilbaum zusammen bleiben und dadurch weniger Teilbäume durchsucht werden müssen.

Generierung der Blätter

Da bei einem kd-Baum die Fälle nur in den Blättern abgespeichert werden, ist die Wahl der Methode, nach der die Blätter gebildet werden, wichtig für die Terminierung der Generierung und für das spätere Retrieval. Folgende Methoden stehen zur Verfügung, um zu bestimmen, ob die Werte weiter aufgeteilt werden, oder ob ein Blatt mit Fällen (Bucket) gebildet wird:

- Bucketgröße: Der triviale Ansatz ist der, ein Bucket zu bilden, wenn bei der Aufteilung nur noch eine bestimmte Menge Fälle übrig geblieben ist. Diese Methode ist einfach zu implementieren und läßt sich auf alle Formen von Daten anwenden, allerdings bleibt die Struktur der Daten völlig unberücksichtigt. Sinnvoll ist diese Variante, falls eine bestimmte Bucketgröße durch das System sinnvoll ist wie z.B. bei einem externen Speicher mit fester Seitengröße, bei dem die Bucketgröße durch die Seitengröße festgelegt ist.
- Klassenzugehörigkeit: Um die Struktur der Daten besser zu berücksichtigen, kann man, unabhängig von der Anzahl der Fälle, auch ein Bucket bilden, falls die verbleibenden Fälle zu einer Klasse gehören. Dazu ist es erforderlich, daß vorher die Fallbasis durch ein Clustering in Klassen aufgeteilt wurde und alle Fälle einer Klasse zugeordnet sind.

Empirische Untersuchungen zeigen, daß die Wahl einer festen Bucketgröße und die Klassenzugehörigkeit geeignete Methoden für die Terminierung der Baumgenerierung sind. Die Orientierung an der Klassenzugehörigkeit ist allerdings nur dann möglich, wenn die Datenbasis eine Klassifizierung der Daten zuläßt.

3.4.2 Retrieval von Fällen

Bei der Suche nach einem Wert in einem binären Suchbaum hat man einen Wert, der mit jedem Knotenwert verglichen wird, bis der entsprechende Wert in einem Knoten gefunden oder ein Blatt erreicht wird. Bei kd-Bäumen muß ein k Werte langer Schlüssel verarbeitet werden, wobei jeder Knoten mit einem Attributwert dieses Schlüssels verglichen wird. Dabei wird mit jedem Knoten der Wertebereich nach oben oder nach unten in einer bestimmten Dimension eingeschränkt. Also kann für jeden Knoten ein k-Tupel $[lower_i, upper_i],\ 1 \leq i \leq k \in I\!N$ mit den Bereichsgrenzen für alle Dimensionen angegeben werden. Durch diese k-dimensionale Intervallschachtelung werden die Teilräume beim Retrieval durch das Traversieren des Baumes eingeschränkt, und jeder Knoten repräsentiert einen k-dimensionalen Hyperquader in dem alle Fälle liegen, die durch das Intervall des k-Tupels $[lower_i, upper_i],\ 1 \leq i \leq k$, eingegrenzt werden.

Falls der ähnlichste Fall zur Anfrage Q gesucht wird, so durchläuft man beim Retrieval den Baum von oben nach unten und vergleicht jeden Knotenwert der Diskriminatoren mit dem entsprechen Attributwert der aktuellen Anfrage, und verzweigt solange in Teilbäume, bis ein Bucket erreicht wird. Anschließend werden alle Fälle dieses Buckets mit der Anfrage verglichen. Bevor der ähnlichste Fall ausgegeben werden kann, müssen noch zwei Tests gemacht werden, auf die unten näher eingegangen wird [43].

Bei den meisten fallbasierten Anwendungssystemen ist es notwendig, zu einer Anfrage Q die Menge der m ähnlichsten Fälle zu finden. Dazu sind vorweg einige Überlegungen notwendig, die völlig unabhängig von den Strukturen und Algorithmen sind, die im Zusammenhang von kd-Bäumen eingeführt wurden. In dem folgenden Abschnitt wird Ähnlichkeit als geometrische Nähe interpretiert und die Anfrage Q, zu der die m ähnlichsten Fälle gefunden werden sollen, als der Mittelpunkt des geometrischen Raumes. Die Menge der m ähnlichsten Fälle kann daher als Hyperkugel um die Anfrage verstanden werden, bei der die ähnlichsten Fälle im Inneren der Kugel liegen. Je kürzer der Abstand eines Falles zum Mittelpunkt ist, um so ähnlicher ist der Fall zur Anfrage.

Auf der Basis dieser Hyperkugel kann ein einfacher Algorithmus zum Finden der m ähnlichsten Fälle angegeben werden. Dabei wird eine initiale Hyperkugel willkürlich festlegt und immer mehr verkleinert, bis genau m Fälle übrig bleiben. Sei *dist* eine Funktion, die den Abstand zwischen zwei Fällen oder zwischen der Anfrage und einem Vergleichsfall angibt, Q die Anfrage, für die m ähnliche Fälle gesucht werden und $SCQ[i]$ mit $i \in I\!N$ der i-te Fall in der Hyperkugel. Dabei soll für alle Fälle $1 \leq i \leq m$ in der Hyperkugel gelten, daß $dist(SCQ[i], Q) \leq dist(SCQ[i+1], Q)$ ist, d. h. die ähnlichsten Fälle sind bezüglich ihrer Ähnlichkeit zur Anfrage Q in der Hyperkugel angeordnet. Es gelte der folgende Basisalgorithmus:

Eingabe: Anfrage Q, Fallbasis CB, Parameter $m \in I\!N$.
Ausgabe: Liste $SCQ[1, \ldots, m]$ der m zur Anfrage Q ähnlichsten Fälle aus der Fallbasis CB.

1. Bestimme m beliebige Fälle und ordne sie in die Liste $SCQ[1], \ldots, SCQ[m]$ ein. Als Datenstruktur kommt eine lineare Liste oder ein Array in Frage.

2. Konstruiere die Hyperkugel $H^{CB}(Q)$ mit dem Radius $dist(SCQ[m], Q)$. Dies bedeutet, daß alle Fälle aus der Fallbasis CB gesucht werden, die ähnlicher sind als der Fall $SCQ[m]$.

3. Falls $|H^{CB}_m(Q)| = m$, so gebe $SCQ[1, \ldots, m]$ aus. In dieser Situation sind die m ähnlichsten Fälle für die Anfrage Q gefunden und können als Ergebnis ausgegeben werden.

4. Falls $|H^{CB}(Q)| > m$, dann durchsuche $H^{CB}(Q)$ nach einem Fall $F \in CB$ mit $dist(F, Q) < dist($ $SCQ[m], Q)$, ordne diesen in die Liste SCQ[1, …,m] ein und gehe zu Schritt 2. Dabei fällt der alte Fall $SCQ[m]$ aus der Liste heraus; die Hyperkugel $H^{CB}(Q)$ wird solange verkleinert, bis genau m Fälle übrig bleiben.

Für die konkrete und effektive Implementierung dieses Algorithmus sind folgende Probleme zu lösen:

- Die initiale Hyperkugel muß effizient ermittelt werden können und schon fast mit der Hyperkugel der tatsächlichen nächsten Nachbarn übereinstimmen.
- Die Suche innerhalb der Hyperkugel muß effektiv sein.
- Das Terminierungskriterium $|H^{CB}(Q)| = m$ ist aufwendig.

Realisierung der Hyperkugel durch kd-Bäume

Die Grundidee des Retrievals mit einem kd-Baum ist eine schrittweise Approximation der Hyperkugel durch Hyperquader, die beim Durchlauf durch den kd-Baum entstehen. Dazu wird der kd-Baum durchlaufen, bis ein Bucket gefunden wird und anschließend der Raum der gefundenen Lösungen erweitert, indem man schrittweise rückwärts durch den Baum läuft. Dabei wird in jedem Knoten der Hyperquader in einer Dimension erweitert und die Grenzen, die durch den Knoten festgelegt werden, können mit den Grenzen der Hyperkugel verglichen werden. Der Algorithmus für das Retrieval von m Fällen sieht folgendermaßen aus:

Eingabe: Anfrage Q, Fallbasis CB, kd-Baum, Anzahl der zu findenden Fälle $m \in I\!N,\ m \leq |CB|$.
Ausgabe: Liste $SCQ[1, \ldots, m]$ der m ähnlichsten Fälle aus der Fallbasis CB.

1. Durchlaufe den kd-Baum, bis ein Bucket erreicht wird.
2. Bilde eine geordnete Liste $SCQ[1], \ldots, SCQ[m]$ aus den Bucketfällen, so daß gilt: Für alle $1 \leq i \leq m:\ SIM_{Imp}(Q, SCQ[i]) \geq SIM_{Imp}(Q, SCQ[i+1])$. Falls weniger als m Fälle in dem Bucket sind, wird der Eltern-Knoten des Buckets genommen und die Ähnlichkeitsliste mit den Fällen gebildet, die durch diesen Knoten repräsentiert wird. Falls immer noch zuwenig Fälle gefunden werden, wird der Baum weiter nach oben durchlaufen, bis mindestens m Fälle gefunden werden. Falls mehr als m Fälle in einem Bucket oder Teilbaum gefunden werden, müssen die m Fälle, die der Anfrage Q am ähnlichsten sind, ausgewählt werden.
3. Bilde eine Hyperkugel $H^{CB}(Q)$ um die Anfrage Q, so daß der Radius r der Abstand vom Mittelpunkt Q zum Fall $SCQ[m]$ ist: $r = dist(Q, SCQ[m])$.
4. Liegt die Hyperkugel $H^{CB}(Q)$ vollständig im betrachteten Teilraum, so ist das Retrieval beendet. Ansonsten muß noch festgestellt werden, welcher Teilraum von der Hyperkugel überlappt wird, damit dieser in einem Backtracking-Schritt rekursiv durchsucht wird. Es muß also festgestellt werden, ob im *benachbarten* Teilbaum nach Fällen zu suchen ist, die noch innerhalb der Hyperkugel liegen.

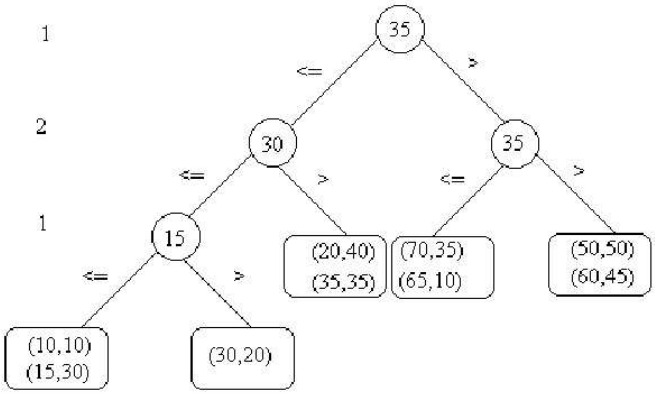


Abbildung 3.6: Ein Beispiel für einen kd-Baum

Beispiel Betrachten wir als Beispiel den obigen kd-Baum. Sei $Q = (60, 40)$ die Anfrage. Es sollen die $m = 2$ ähnlichsten Fälle gefunden werden, d. h. es müßten die Werte $(60, 45)$ und $(70, 35)$ ausgewählt werden. Der erste Durchlauf durch den kd-Baum endet in dem rechten Bucket $[(50, 50), (60, 45)]$. Es wird also die Liste $SCQ[1, 2] = \{(60, 45), (50, 50)\}$ erstellt. Um die Anfrage Q wird die Hyperkugel $H_2^{CB}(Q)$ mit dem Radius $r = dist((60, 40), (50, 50))$ definiert. Angenommen, es gelte hier der euklidische Abstand im zweidimensionalem Raum, also: $dist((60, 40), (50, 50)) = \sqrt{(60-50)^2 + (40-50)^2)} = 14, 1$.

Für die anderen Fälle in dem Teilbaum gilt:

$$dist((60, 40), (60, 45)) = 5 < r!$$
$$dist((60, 40), (70, 35)) = 11, 2 < r!$$
$$dist((60, 40), (65, 10)) = 30, 4$$

Die Hyperkugel enthält die Fälle $H^{CB}(Q) = \{(60,45),(70,35),(50,50)\}$ – und überlappt das linke benachbarte Bucket. Deswegen wird als Teilraum der Elternknoten dieser beiden Buckets genommen und die Ausgabe ist: $SCQ[1,2] = \{(60,45),(70,35)\}$.

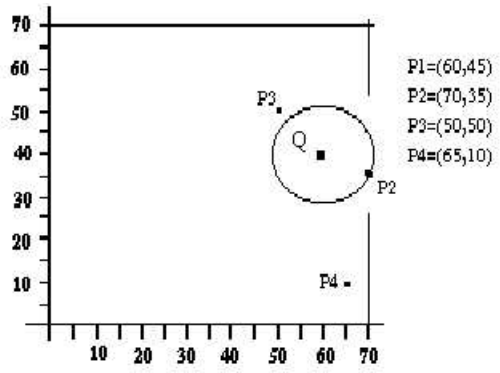


Abbildung 3.7: Eine Hyperkugel HCB(Q) zum kd-Baum

Durch diesen Ansatz können die oben genannten Probleme gelöst werden:

- Durch die Fälle in dem Bucket bzw. die durch einen Knoten repräsentiert werden, kann eine gute initiale Lösung gefunden werden
- Der kd-Baum liefert eine Zugriffsstruktur mit deren Hilfe die Hyperkugel effizient nach weiteren Fällen, die ähnlicher als der m-te gefundene Fall sind, durchsucht werden kann
- Eine Terminierung ist möglich, indem die Hyperkugel mit dem konstruierten Hyperquader verglichen wird.

Da durch die einfache Suche im kd-Baum nur auf Un-/Gleichheit geprüft werden kann, sind Tests nötig, die die Approximation der Hyperkugel durch die Hyperquader bewerten. Es muß geprüft werden, ob die Hyperkugel vollständig im Hyperquader liegt (Ball-Within-Bounds Test) oder ob die Hyperkugel andere Teilräume schneidet (Ball-Overlaps-Bounds Test). Durch diesen Test wird das Terminierungskriterium festgelegt und es werden die Teilbäume bestimmt, in denen nach weiteren Fällen gesucht werden muß.

Der Ball-Within-Bounds Test Beim Ball-Within-Bounds (BWB) Test werden die Grenzen des aktuellen Teilbaums, die durch das k-Tupel der Intervalle $B_T = ([l_1,u_1],\ldots,[l_k,u_k])$ gegeben sind – wobei $l = lower$ die untere Grenze und $u = upper$ die obere Grenze des jeweiligen Intervalls darstellt – in allen Dimensionen mit den Grenzen der Hyperkugel verglichen.

Definition 3.3 (Ball-Within-Bounds-Test, BWB-Test) *Seien m ähnliche Fälle zu finden und T der aktuell betrachtete Teilbaum, der durch die Dimensionsgrenzen $B_T = ([l_1,u_1],\ldots,[l_k,u_k])$ beschrieben werden kann, wobei l_i die untere und u_i die obere Schranke bezeichnet. Sei weiter $Q = (q_1,\ldots,q_k)$ die Anfrage und r der Radius der Hyperkugel; dann gilt:*

$$r = dist(Q, SCQ[m]), \quad \textit{falls } SCQ[m] \textit{ der } m\textit{-ähnlichste der bisher gefundenen Fälle ist;}$$
$$r = -\infty, \quad \textit{falls noch keine } m \textit{ ähnliche Fälle gefunden wurden.}$$

Für den Ball-Within-Bounds-Test sind zwei Hilfsfälle

$$F^{(i)}_{min} = (h^{(i)}_1,\ldots,h^{(i)}_k) \textit{ und } F^{(i)}_{max} = (v^{(i)}_1,\ldots,v^{(i)}_k)$$

für $i,j \in \{1,\ldots,k\}$ wie folgt zu definieren:

$$h^{(i)}_j = \begin{cases} l_j & : \textit{falls } i = j \\ q_j & : \textit{sonst} \end{cases}$$
$$v^{(i)}_j = \begin{cases} u_j & : \textit{falls i = j} \\ q_j & : \textit{sonst} \end{cases}$$

Damit läßt sich das Prädikat $BWB(Q, r, B_T)$ formal wie folgt beschreiben:

$$BWB(Q, r, B_T) \Leftrightarrow \forall i \in \{1, \ldots, k\} : dist(Q, F_{min}^{(i)}) < r \wedge dist(Q, F_{max}^{(i)}) < r \wedge l_i \leq q_i \leq u_i.$$

Beispiel Betrachten wir den BWB-Test für das obige Beispiel zu dem Zeitpunkt des ersten Backtracking-schrittes. Der betrachtete Teilraum ist der Teilbaum, der durch den Knoten mit dem Diskriminatorattributwert 35 im rechten Teilbaum aufgespannt wird. Für den kd-baum ergeben sich folgende Mengen B_T:

1. Wurzel: $B_T = ([l_1 = 10, u_1 = 70], [l_2 = 10, u_2 = 50])$
2. linker Teilbaum, Knoten mit Attributwert 30: $B_T = ([(10, 35], [10, 40])$
3. rechter Teilbaum, Knoten mit Attributwert 35: $B_T = ([(50, 70], [10, 50])$
4. linker Teilbaum, Knoten mit Attributwert 15: $B_T = ([10, 30], [10, 30])$

Die Anfrage sei $Q = (60, 40)$ und $SCQ[1, 2] = ((60, 45), (70, 35))$. Damit lassen sich jetzt die Werte für B_T, $F_{min}^{(1,2)}$ und $F_{max}^{(1,2)}$ berechnen.

$$\begin{aligned}
B_T &= ([60, 70], (35, 45]) = ([l_1, u_1), (l_2, u_2]) \\
F_{min}^{(1)} &= (h_1^{(1)} = l_1 = 60, h_2^{(1)} = q_2 = 40) \\
F_{min}^{(2)} &= (h_1^{(2)} = q_1 = 60, h_2^{(2)} = l_2 = 35) \\
F_{max}^{(1)} &= (v_1^{(1)} = u_1 = 70, v_2^{(1)} = q_2 = 40) \\
F_{max}^{(2)} &= (q_1 = 60, u_2 = 45)
\end{aligned}$$

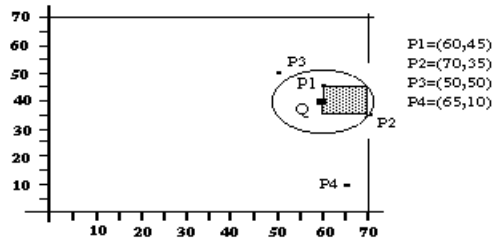


Abbildung 3.8: Der BWB-Test für das Beispiel

Mit $SCQ[1, 2] = ((60, 45), (70, 35))$ und dem Radius der Hyperkugel $r = dist((60, 40), (70, 35)) = 11, 2$ ergeben sich folgende Abstände zur Anfrage Q:

$$\begin{aligned}
dist(F_{min}^{(1)}, Q) &= dist((60, 40), (65, 40)) = 5 \\
dist(F_{min}^{(2)}, Q) &= dist((60, 40), (60, 35)) = 5 \\
dist(F_{max}^{(1)}, Q) &= dist((60, 40), (70, 40)) = 10 \\
dist(F_{max}^{(2)}, Q) &= dist((60, 40), (60, 45)) = 5
\end{aligned}$$

Der BWB-Test ist erfüllt. Die grau unterlegte Fläche in der obigen Abbildung ist der Teil, der durch F_{max} und F_{min} festgelegt wird.

Der Ball-Overlaps-Bounds Test Das Gegenstück des BWB-Tests ist der Ball-Overlaps-Bounds (BOB) Test, bei dem entschieden wird, ob die Hyperkugel größer ist als der Hyperwürfel, der durch den aktuellen Teilbaum T repräsentiert wird. Falls der BOB-Test ein positives Ergebnis liefert, muß der Teilbaum T des kd-Baums, der außerhalb des aktuellen Teilbaumes ist, nach Fällen durchsucht werden, die mindestens genauso ähnlich sind wie der ähnlichste m-te Fall $SCQ[m]$.

Definition 3.4 (Bounds-Overlap-Ball-Test, BOB-Test) *Sei T der zu untersuchende Teilbaum mit den Dimensionsgrenzen $B_T = ([l_1, u_1], \ldots, [l_k, u_k])$ sowie Q und r entsprechend obiger Definition. Es ist ein Hilfsfall F_{min} zu konstruieren, der am nahesten bei Q, aber noch innerhalb von T liegt. Liegt dieser Fall innerhalb der Hyperkugel, so überlappt diese in den Teilraum T. Dieser Hilfsfall $F_{min} = (v_1, \ldots, v_k)$ wird wie folgt konstruiert:*

$$v_i = \begin{cases} q_i & : \textit{falls } l_i \leq q_i \leq u_i \\ l_i & : \textit{falls } q_i \leq l_i \\ u_i & : \textit{falls } u_i \leq q_i \end{cases}$$

Damit läßt sich das Prädikat BOB formal wie folgt beschreiben:

$$BOB(Q, r, B_T) \Leftrightarrow dist(Q, F_{min}) \leq r.$$

Beispiel Betrachten wir den BOB-Test zum obigen Beispiel zu dem Zeitpunkt, in dem der aktuelle Teilbaum das Bucket mit den Werten $(50, 50)$ und $(60, 45)$ ist. Die Anfrage ist $Q = (60, 40)$. Damit ergibt sich für die Hyperkugel ein Radius $r = dist((60, 40), (50, 50)) = 14, 1$. Die Dimensionsgrenzen des *benachbarten* Buckets mit den Werten $((70, 35), (65, 10))$ sind durch folgende Bereichsgrenzen festgelegt:

$$B_T = ([l_1, u_1], [l_2, u_2])$$

mit l_1 = niedriegster Wert der Dimension $i = 1$ und u_1 = höchster Wert in der Dimension $i = 1$, also erhält man $B_T = ([65, 70], [10, 35])$. Damit kann der Hilfsfall $F_{min} = ((v_1 = l_1 = 65), (v_2 = u_2 = 35))$ konstruiert werden. Der Abstand zwischen Q und F_{min} ist $dist((60, 40), (65, 35)) = 7, 1 < 14, 1$! Damit ist klar, daß dieses Bucket von der Hyperkugel überlappt wird und daher nach ähnlicheren Fällen als dem Fall $(50, 50)$ durchsucht werden muß.

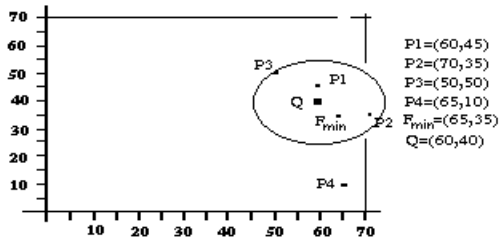


Abbildung 3.9: Der BOB-Test für das Beispiel

Diese beiden Tests sind elementar für den Retrievalprozess, da sie auf der einen Seite bei der Terminierung des Retrievals helfen (BWB), aber auch auf der anderen Seite ein Kriterium für das Durchsuchen von weiteren Teilbäumen liefern (BOB).

Dynamische Bounds Die Bounds-Tests haben einen großen Anteil an dem Aufwand für das Retrieval und werden doch in vielen Fällen überflüssig ausgeführt, weil nur lokale Information für die Ausführung der Tests zur Verfügung stehen. Die Grundidee von *dynamischen* Bounds ist, die Anzahl der Tests zu verringern, indem bei der Generierung zusätzliche Informationen über die Verteilung der Fälle bereitgestellt werden. Dazu werden zusätzlich zu dem Intervall der Bereichsgrenzen $B_T = ([l_1, u_1], \ldots, [l_k, u_k])$ Intervalle mit maximaler Ausdehnung und minimaler Einschränkung erzeugt, die dann als neue Grenzen für die beiden Tests BWB und BOB dienen.

Minimale dynamische Bounds für den BOB Test Für den BOB Test ist es sinnvoll, die Fälle eines Buckets oder eines Teilbaumes in möglichst enge Grenzen zu schließen, um die Entscheidung, ob ein weiterer Teil des Baumes durchsucht werden muß, früh treffen zu können. Es kommt oft vor, daß ein Teilbaum zwar von der Hyperkugel geschnitten wird, aber in der Schnittmenge von Teilbaum und Hyperkugel *keine* Fälle liegen.

Definition 3.5 (Minimale Dynamische Bounds) *Minimale dynamische Bounds geben die minimalen Grenzen von Teilräumen an, in denen Fälle enthalten sind. Sei $VAL_j(S)$ die Menge aller für ein Attribut A_j in einer Menge von Fällen $S \subseteq CB$ existierenden Attributausprägungen, d. h.*

$$VAL_j(S) = \{v | case[j] = v \wedge Case \in S\}.$$

Die minimalen dynamischen Bounds für ein Bucket B in der Dimension $j \in \{1, \ldots, k\}$ sind dann definiert als ein Intervall, bestehend aus den minimalen und maximalen Attributwerten im entsprechenden Bucket:

$$minBounds.Upper(B)[j] = \max(VAL_j(B))$$
$$minBounds.Lower(B)[j] = \min(VAL_j(B))$$

Analog werden minimale dynamische Bounds für innere Knoten mit linkem Teilbaum und rechtem Teilbaum R definiert.

Definition 3.6 (Minimale Dynamische Bounds für innere Knoten) *Die minimalen dynamischen Bounds für einen beliebigen inneren Knoten P in der Dimension $j \in \{1, \ldots, k\}$ ergeben sich rekursiv aus den minimalen und maximalen Attributwerten der durch den linken Sohn L und den rechten Sohn R repräsentierten Mengen von Fällen, d. h.*

$$minBounds.Upper(P)[j] = \max(VAL_j(L), VAL_j(R))$$
$$minBounds.Lower(P)[j] = \min(VAL_j(L), VAL_j(R))$$

Die minimalen dynamischen Bounds werden von den Buckets ausgehend nach oben berechnet werden. Diese Berechnung findet während der Generierung des kd-Baums statt, wobei die Werte der dynamischen Bounds in den Knoten gespeichert werden.

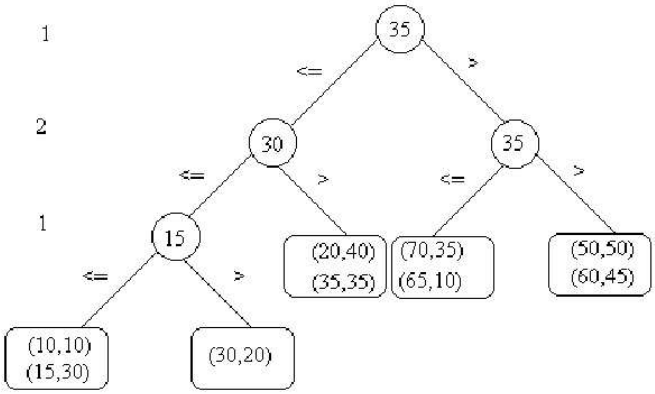


Abbildung 3.10: kd-Baum

Beispiel Zuerst werden die minimalen dynamischen Bounds für die Buckets berechnet - und zwar von rechts nach links. In diesem Beispiel werden für den rechten Teilbaum, der durch den Knoten P mit dem Diskriminatorattributwert 35 aufgespannt wird, die minimalen dynamischen Bounds berechnet:

1. Rechter Teilbaum, linkes Bucket:

$$\text{Bucket } B = \{(70, 35), (65, 10)\}$$

$$VAL_1 = \{70, 65\}$$
$$VAL_2 = \{35, 10\}$$

$$minBounds.Upper(B)[j] = \max(VAL_j(B)) \Rightarrow minBounds.Upper(B)[1] = 70,\ minBounds.Upper(B)[2] = 35$$

$$minBounds.Lower(B)[j] = \min(Val_j(B)) \Rightarrow minBounds.Lower(B)[1] = 65,\ minBounds.Lower(B)[2] = 10$$

2. Rechter Teilbaum, rechtes Bucket:

$$\text{Bucket } B = \{(50, 50), (60, 45)\}$$

$$minBounds.Upper(B)[1] = 60$$
$$minBounds.Lower(B)[1] = 50$$
$$minBounds.Upper(B)[2] = 50$$
$$minBounds.Lower(B)[2] = 45$$

3. Damit gilt für den Knoten P der beiden Buckets:

$$minBounds.Upper(P)[1] = \max\{70, 60\} = 70$$
$$minBounds.Upper(P)[2] = \max\{35, 50\} = 50$$
$$minBounds.Lower(P)[1] = \min\{65, 50\} = 50$$
$$minBounds.Lower(P)[2] = \min\{10, 45\} = 10$$

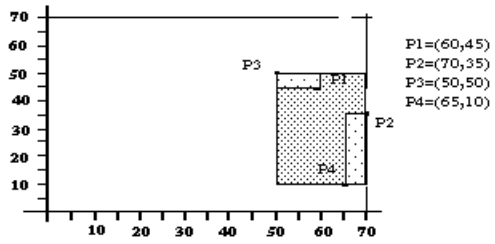


Abbildung 3.11: Berechnung dynamischer Bounds für das Beispiel: BOB Test

Die beiden kleinen hellen Flächen in der obigen Abbildung sind die dynamischen Bounds für die beiden Buckets. Daraus ergibt sich der dynamische Bound für den Knoten. Hier sieht man, daß für den Knoten die obere Grenze durch den Punkt $(70, 50)$ und die untere Grenze durch den Punkt $(50, 10)$ markiert ist. Erst ab diesen Grenzen sind Punkte anzutreffen. Falls dieser Teilbaum beim BOB Test von der Hyperkugel geschnitten wird, aber die Hyperkugel außerhalb dieser Grenzen bleibt, so braucht dieser Teilbaum nicht weiter durchsucht werden.

Maximale dynamische Bounds für den BWB Test Beim BWB Test wird entschieden, ob die Hyperkugel vollständig in einem besuchten Teilbaum liegt. Dieser Teilbaum wird durch ein k-Tupel von Intervallgrenzen beschrieben. Für den dynamischen BWB Test werden diese Grenzen soweit verschoben, daß man gerade an die Fälle in den benachbarten Teilbäumen heranreicht, d. h. man versucht den Raum, der durch diesen Teilbaum beschrieben wird, soweit wie möglich zu erweitern, ohne daß neue Fälle aus anderen Teilbäumen in diesen Raum gelangen. Dadurch ist der BWB-Test besser informiert und der Retrievalprozess kann schneller terminieren. Für den BWB Test werden *maximale dynamische Bounds* definiert. Die Berechnung dieser dynamischen Bounds ist allerdings kompliziert, da im Gegensatz zu den minimalen dynamischen Bounds nicht nur Knoten des aktuellen Teilbaums untersucht werden müssen, sondern auch Knoten aus *benachbarten* Teilbäumen. Daher ist es sinnvoll und nötig, zuerst die minimalen Bounds in den einzelnen Teilbäumen zu berechnen und anschließend mit deren Hilfe die maximalen dynamischen Bounds zu bestimmen.

Definition 3.7 (Maximale dynamische Bounds) *Die maximalen dynamischen Bounds beschreiben die Grenzen eines Teilraumes, bis zu denen der Teilraum maximal ausgedehnt werden kann, ohne dabei auf Fälle zu stoßen. Die maximalen Bounds des Wurzelknotens R^* werden initialisiert durch:*

$$maxBounds.Upper(R^*)[j] = +\infty$$
$$maxBounds.Lower(R^*)[j] = -\infty$$

Die Berechnung der maximalen dynamischen Bounds der Söhne eines Knotens P mit dem Diskriminatorattribut A_d erfolgt rekursiv:

Für den linken Sohn L:

$$maxBounds.Upper(L)[j] = \begin{cases} minBounds.Lower(R)[j] : & \textit{falls } j = d \ (Diskriminator) \\ maxBounds.Upper(P)[j] : & \textit{sonst} \end{cases}$$

$$maxBounds.Lower(L)[j] = maxBounds.Lower(P)[j]$$

Für den rechten Sohn R:

$$maxBounds.Lower(R)[j] = \begin{cases} minBounds.Upper(L)[j] : & \textit{falls } j = d \ (Diskriminator) \\ maxBounds.Lower(P)[j] : & \textit{sonst} \end{cases}$$

$$maxBounds.Upper(R)[j] = maxBounds.Upper(P)[j]$$

Beispiel Das Beispiel beschränkt sich hier wieder auf den Knoten T im rechten Teilbaum mit dem Diskriminatorattributwert 35.
1. Initialisierung des Wurzelknotens R^*:

$$\begin{aligned} maxBounds.Upper(R^*)[1] &= +\infty \\ maxBounds.Upper(R^*)[2] &= +\infty \\ maxBounds.Lower(R^*)[1] &= -\infty \\ maxBounds.Lower(R^*)[2] &= -\infty \end{aligned}$$

2. Berechnung der maximalen dynamischen Bounds für den rechten Sohn des Wurzelknotens R^* mit dem Diskriminatorindex d=1:

$$\begin{aligned} maxBounds.Lower(R)[1] = minBounds.Upper(L)[1] &= max(VAL_1(L)) \\ maxBounds.Lower(R)[1] = max\{10,15,30,20,35\} &= 35 \\ maxBounds.Lower(R)[2] = maxBounds.Lower(L)[2] &= -\infty \\ maxBounds.Upper(R)[1] = maxBounds.Upper(R^*)[1] &= +\infty \\ maxBounds.Upper(R)[2] = maxBounds.Upper(R^*)[2] &= +\infty \end{aligned}$$

Damit hat man also für den Knoten T folgende Werte erhalten:

$$\begin{aligned} maxBounds.Lower(T)[1] &= 35 \\ maxBounds.Lower(T)[2] &= -\infty \\ maxBounds.Upper(T)[1] &= +\infty \\ maxBounds.Upper(T)[2] &= +\infty \end{aligned}$$

In diesem Beispiel sieht man, daß die Grenzen für die Hyperkugel nur durch die minimalen Bounds des linken Teilbaums, der durch den Knoten mit dem Partitionswert 35 begrenzt ist, festgelegt werden. Man braucht also beim Retrieval nur dann einen Teilbaum aus der linken Hälfte des kd-Baumes zu betrachten, wenn die minimalen Bounds des ersten Knotens unterschritten werden.

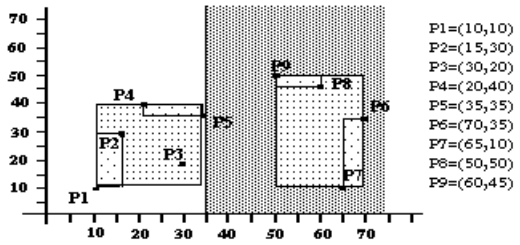


Abbildung 3.12: Maximale dynamische Bounds für das Beispiel: BWB Test

Die etwas helleren Flächen in der obigen Abbildung stellen die minimalen dynamischen Bounds der Buckets bzw. der Knoten dar. Die große dunkle (graue) Fläche ist der maximale dynamische Bound für den rechten Knoten mit dem Diskriminatorattributwert 35. Man erkennt an dem Häufungspunkt, das rechts bzw. oben und unten vom rechten Knoten keine Fälle mehr auftreten; das Koordinatensystem weist in Richtung der positiven Abzisse keine Punkte mehr auf, dies gilt analog auch für maximalen Bounds der Ordinate.

Die Werte für die dynamischen Bounds werden während der Generierungsphase berechnet und mit den Attributswerten in dem kd-Baum gespeichert. Der Mehraufwand in der Generierungsphase wird bei jedem der beiden dynamischen Bounds durch eine effektivere Retrievalphase ausgeglichen.

Ungeordnete Attributwerte

Bei der bisherigen Betrachtung von kd-Bäumen spielte die Ordnung der einzelnen Attribute eines Falles sowohl für die Generierung als auch für das spätere Retrieval eine elementare Rolle. In der Regel ist nicht davon ausgehen, daß alle Attribute eines Falles eine Ordnung aufweisen. Für eine Erweiterung des Konzepts von kd-Bäumen auf ungeordnete Attribute wird hinsichtlich der Generierungsphase sowie hinsichtlich Retrievalphase unterschieden.

Generierungsphase Bei der Generierung wird nach geordneten und ungeordneten Knoten unterschieden, je nachdem, ob die Diskriminatorattribute geordnete oder ungeordnete Wertebereiche haben. Bei geordneten Wertebereichen kann der Baum, je nach Ordnung, weiter aufgegliedert werden. Bei ungeordneten Knoten muß für jeden Fall ein Teilbaum mit dem Wert des Attributes der jeweiligen Knotendimension angelegt werden. Hat ein ungeordnetes Attribut A_i also m Werte, so wird der Raum dieses Diskriminators in m Teilräume aufgeteilt; hierbei entspricht jeder erzeugte Teilraum genau einem Wert des Diskriminatorattributes.

Retrievalphase Für die Retrievalphase ändert sich das Absteigen im Suchbaum und das Backtracking, da jetzt zwischen vielen verschiedenen Werten ausgewählt werden muß. Für das Absteigen im Suchbaum bedeutet dies, daß man sequentiell alle Werte in dem Knoten auf Gleichheit überprüft. Beim Backtracking muß entschieden werden, welcher Teilbaum als nächster zu untersuchen ist. Dazu ist die lokale Ähnlichkeit der Attribute eine gute Entscheidungshilfe.

3.4.3 Bewertung der Verwendung von kd-Bäumen

Durch die Verwendung von kd-Bäumen im allgemeinen, und durch dynamische Bounds im besonderen, ist es möglich, die Anzahl der Fallvergleiche so zu reduzieren, daß Retrieval effizient implementiert wird. Dies macht sich insbesondere bei großen Datenbeständen, die nicht einer ständigen Änderung unterworfen sind, nicht vollständig in den Hauptspeicher passen, oder auf verteilten Systemen gespeichert sind, bemerkbar. Die Verwendung von dynamischen Bounds erfordert zwar einen Mehraufwand an Speicher- und Generierungszeit, dieser zahlt sich aber beim Retrieval durch die Einsparung von Bounds-Tests und Fallvergleichen aus.

Vorteile

- Korrektheit und Vollständigkeit: Das Verfahren ist korrekt, da der Ähnlichkeitsbegriff SIM_{Rep} auf alle Fallbeispiele eines Buckets angewendet wird. Die Vollständigkeit wird durch die konstruierte Hyperkugel gesichert, weil diese die Suche nach Fallbeispielen steuert und damit auch das Ende der Suche festlegt.
- Performanz: Bei einer geeigneten Partionierung des Datenraums ist das System insbesondere bei externen und verteilten Datenbeständen sehr effektiv.
- Gut geeignet für große Fallbasen
- Wartungsfreundlich: Ein kd-Baum ist eine wartungsfreundliche Datenstruktur weil durch eine geeignete Organisation der Buckets nicht jede Einfüge- und Löschoperation eine Neugenerierung des kd-Baums bedeuten muß.
- Konfigurierbarkeit: Insbesondere durch anwendungsspezifische Erweiterungen ist das System an viele mögliche Situationen anpassbar.

Nachteile

- Keine Spontanabfragen: Für eine Anfrage muß erst der kd-Baum generiert werden. Wenn noch kein Baum generiert ist, ist auch keine Anfrage möglich.
- Parameterwahl: Die Performanz des Systems hängt ab von vielen Parametern wie z.B. der Bucketgröße oder auch der Bestimmung des Partitionswertes ab.
- Verwaltungsaufwand: Die Pflege der Baumstruktur verlangt zusätzlichen Verwaltungsaufwand.
- Beschränkung der Ähnlichkeitsmaße: Voraussetzung für die Anwendung ist Monotonie bezüglich der Funktion SIM_{Imp}.
- Komplexe Implementierung: Verglichen mit dem Sequentiellen und dem Relationellen Retrieval ist die Implementierung dieses Verfahrens komplex.

3.5 Case Retrieval Nets

Bei den *Case Retrieval Nets (CRN)* handelt es sich um ein jüngst von Burkhard und Lenz an der Humboldt-Universität Berlin entwickeltes Retrieval-Konzept [45]. Es unterscheidet sich in mehrerlei Hinsicht vom Konzept zum Retrieval mit kd-Bäumen. Während bei der Suche in einem kd-Baum die Menge der Fälle, in dem die zur Anfrage ähnlichsten Fälle enthalten sind, im Sinne einer top-down Strategie schrittweise verkleinert wird, handelt es sich beim Case Retrieval Net um einen bottom-up Ansatz. Im Unterschied zum Retrieval mit kd-Bäumen wird für das Arbeiten mit einem Case Retrieval Net anfangs nicht die vollständige Eingabe der Anfrage verlangt; es müssen also nicht alle zum Anfragewissen gehörenden Attribut-Wert-Tupel in der ersten Anfrage angegeben werden. Vielmehr wird eine interaktive Suche mit dem CRN als *Information Completion* gestaltet, also als ein Prozeß, bei dem die erste Anfrage im weiteren vervollständigt werden kann. Während eine klassische Fallbeschreibung aus einer Problembeschreibung und einer dazu gehörenden Angabe der Problemlösung besteht, gibt es in der Falldefinition für ein CRN keine Lösungsbeschreibung. Die Fälle der Fallbasis werden durch sogenannte *Information Entities (IE)* konstituiert, und zwar völlig unstrukturiert. Bei diesen Information Entities handelt es sich um unteilbare Informationseinheiten, also gewissermaßen um Informationsatome oder eben um kleinste, Informationselemente, die nicht weiter reduzierbar sind und als Indices für das Retrieval dienen. Dementsprechend weist der von einem CRN gefundene ähnlichste Fall im Unterschied zu einem klassischen CBR-Fall keine separate Lösungsbeschreibung auf. CRNs wurden für ein virtuelles Reisebüro im World Wide Web entwickelt, über welches ein Kunde online eine Reise buchen kann; hierbei handelt es sich um eine Anwendung mit einer großen Fallbasis von 250 000 Fällen, die keine Adaption erfordert, und für die Lenz eine Antwortzeit von 1,3 Sekunden angibt [38].

3.5.1 Komplexitätsbetrachtungen

Im ungünstigsten Fall wächst der Retrievalaufwand für ein CRN *linear* mit der Größe der Fallbasis |C|. Eine theoretische Bewertung der CRN-Effizienz muß mehrere Parameter einbeziehen; dazu gehören

- die Größe oder der Umfang der Anfrage: je mehr Information Entities (IE) anfänglich aktiviert werden müssen, umso größer ist der Propagierungsaufwand.
- der Grad der Verbundenheit der einzelnen Information Entities (connectivity): je mehr positive Ähnlichkeitsrelationen zwischen den einzelnen IEs existieren, umso mehr Aufwand ist für die die sogenannte *Propagation* erforderlich. Man unterscheidet innerhalb eines CRN zwischen der Ähnlichkeitsfortschreibung und der Relevanzfortschreibung (*Relevance Propagation*). Innerhalb der IE-Ebene, also zwischen den einzelnen IE, existiert nur Ähnlichkeitsfortschreibung (*Similarity Propagation*). Zwischen den jeweiligen IEs und den einzelnen Fallknoten eines CRN gibt es keine Ähnlichkeitsrelationen, vielmehr spricht man hier von Relevanzwertfortschreibung; wenn ein IE zu einem Fall gehört, so hat es einen Relevanzwert.
- der Spezifität der IE: je mehr Fälle zu den einzelnen IE gehören, umso größer ist der Aufwand für die Relevanzfortschreibung.
- der Verteilung der Fälle: wenn viele ähnliche Fälle existieren, dann werden auch viele von diesen in den Retrievalfocus kommen. Wenn wenige ähnliche Fälle abrufbar sind, so muß der Retrievalfocus ggf. erweitert werden, bis genügend Fälle gefunden werden.
- der Anzahl der gesuchten Fälle: CRN definieren für die Anfrage eine Präferenzordnung. Je größer die Zahl der gesuchten ähnlichen Fälle ist, umso größer ist der Retrievalaufwand.

Für eine abschätzende Berechnung des Retrievalaufwandes mögen die folgenden Angaben gelten:

- Sei SIM ein globales Ähnlichkeitsmaß, welches $k \in \mathbb{N}$ Attribute berücksichtigt:
  $E = E_{A_1} \cup E_{A_2} \cup \ldots \cup E_{A_k}$ mit E als Gesamtmenge aller IE.
- Die Anfrage bestehe aus k IE.
- Es gibt einen Ähnlichkeits-fan out out_{EE}, es wird also angenommen, daß jedes IE $e \in E$ eine positive Ähnlichkeitsrelation zu den IE in out_{EE}, einschließlich zu sich selbst, hat.
- Es gibt einen Relevanz-fan out out_{EC}, d. h. es wird berücksichtigt, daß jedes IE mit einer konkreten Beziehung zu einem Fall aus der Fallbasis C, also $e \in C$, eine Relevanzrelation zu diesem out_{EC} Fallknoten hat.

Auf dieser Basis kann schrittweise der Retrievalaufwand kalkuliert werden.

Schritt 1: Initialisierung
Jedes Element einer Anfrage wird zu einem IE im CRN: k

Schritt 2: Ähnlichkeitspropagierung
Für die mit der Anfrage aktivierten IE werden entlang der Ähnlichkeitsrelationen numerische Werte propagiert: $k * out_{EE}$

Schritt 3: Relevanzpropagierung
Für jedes aktivierte IE wird entlang der Relevanzrelationen ein numerischer Wert propagiert: $(k * out_{EE}) * out_{EC}$

Es gibt also in diesem Kalkül |E| verschiedene Information Entities $e \in E$, eine Fallbasis mit insgesamt |C| Fällen und k Attribute. Der Leser sieht, daß nicht nur Ähnlichkeitswerte zwischen den IEs, sondern auch Relevanzwerte propagiert werden, welche die Bedeutung der einzelnen IEs für die Fälle angeben.

Der Relevanz-fan out ist: $out_{EC} = \frac{|C| * k}{|E|}$

Insgesamt erhält man dann über alle drei Schritte für den CRN-Aufwand:

O(CRN Retrieval) = $k + k * out_{EE} + (k * out_{EE}) * out_{EC}$
O(CRN Retrieval) = $k + k * out_{EE} + (k * out_{EE}) * \frac{|C| * k}{|E|}$
O(CRN Retrieval) = $k + k * out_{EE} + (k^2 * out_{EE}) * \frac{|C|}{|E|}$
O(CRN Retrieval) = $O(\frac{out_{EE}}{|E|} * |C|)$

Für die meisten Anwendungen muß eine sehr kleine Zahl $\frac{out_{EE}}{|E|}$ angenommen werden. Hierüber ergibt sich also eine lineare Komplexität: der Retrievalaufwand steigt linear mit der Größe der Fallbasis |C| an [38].

3.5.2 Basic Case Retrieval Net

In diesem Abschnitt wird das Basis-CRN erläutert. Ein Basic Case Retrieval Net (BCRN) ist ein Netzwerk mit zwei Arten von Knoten: den IE-Knoten und den Fallknoten. Zwischen den IE-Knoten existieren Ähnlichkeitsrelationen, die – aufsetzend auf der Anfrage – im Laufe des Retrievals propagiert werden. Die entsprechenden Funktionen werden unten erläutert. Des weiteren gibt es in einem CRN noch Relevanzrelationen, die zwischen den einzelnen IEs und den Fallknoten bestehen und ebenfalls im Laufe des Retrievals propagiert werden. Ein positiver Relevanzwert zeigt an, daß ein IE zu einem bestimmten Fall gehört. Wenn ein IE e nicht zu einem Fall $c \in C$ gehört, so ist der Relevanzwert hierfür Null: $\rho(e, c) = 0$.

Definition 3.8 (Basic Case Retrieval Net) *Ein Basic Case Retrieval Net (BCRN) ist definiert als eine Struktur* $N = [E, C, \sigma, \rho, \Pi]$ *mit*

- *E als endliche Menge von IE-Knoten;*
- *C als endliche Menge von Fallknoten;*
- *σ als Ähnlichkeitsfunktion* $\sigma : E \times E \rightarrow \mathbb{R}$, *welche die Ähnlichkeit* $\sigma(e_i, e_j)$ *zwischen den IEs* $e_i, e_j \in E$ *beschreibt;*
- *ρ als Relevanzfunktion* $\rho : E \times C \rightarrow \mathbb{R}$, *welche die numerische Relevanz* $\rho(e, c)$ *eines IE* $e \in E$ *für einen Fallknoten* $c \in C$ *angibt;*
- Π *als Menge von Propagierungsfunktionen* $\pi_n : \mathbb{R}^E \rightarrow \mathbb{R}$ *für jeden Knoten* $n \in E \cup C$.

Die Entwickler von CRN betrachten fallbasiertes Schließen als eine Assimilation von Fallwissen im Sinne von *Case Completion*. Der mit der Anfrage q eingeleitete Assimilationsprozeß führt in jedem Arbeitsschritt dazu, daß das Anfragewissen um ein IE erweitert wird, bis der bearbeitete Fall schließlich komplettiert worden ist und der Benutzer terminiert. Die Aktivierung des BCRN geschieht mit einer Propagierungsfunktion.

Definition 3.9 (Aktivierung eines BCRN) *Die Aktivierung eines Basic Case Retrieval Net mit der Struktur N = $[E, C, \sigma, \rho, \Pi]$ ist eine Funktion $\alpha : E \cup C \rightarrow I\!R$.*

Es gibt sowohl Funktionen $\alpha(e)$ für die Aktivierung der IE, wie auch Funktionen $\alpha(c)$, welche die Fallknoten $c \in C$ aktivieren. Die Aktivierung $\alpha(e)$ bringt die Bedeutung der jeweiligen IE $e \in E$ für die aktuelle Anfrage zum Ausdruck. Formal ist der Propagierungsprozeß wie folgt definiert:

Definition 3.10 (Propagierungsprozeß in einem BCRN) *Gegeben sei ein Basic Case Retrieval Net mit der Struktur N = $[E, C, \sigma, \rho, \Pi]$ mit der Menge E =$\{e_1, \ldots, e_s\}$. Die Abbildung $\alpha_t : E \cup C \rightarrow I\!R$ heißt Aktivierung zur Zeit t. Die Aktivierung von IE-Knoten $e \in E$ zur Zeit t+1 ist gegeben durch*

$$\alpha_{t+1}(e) = \pi_e(\sigma(e_1, e) * \alpha_t(e_1), \ldots, \sigma(e_s, e) * \alpha_t(e_s))$$

und die Aktivierung von Fallknoten $c \in C$ zur Zeit t+1 ist gegeben durch

$$\alpha_{t+1}(c) = \pi_c(\rho(e_1, c) * \alpha_t(e_1), \ldots, \rho(e_s, c) * \alpha_t(e_s)) \; .$$

Die Initialaktivierung der IE $e \in q$ setzt auf den Inhalten der Anfrage auf und erfolgt mit einer binären Funktion:

$$\alpha_0(e) = \begin{cases} 1 & : \text{ falls } e \in q \\ 0 & : \text{ sonst} \end{cases}$$

Es werden also nur die IE-Knoten aktiviert, die Bestandteil der Anfrage sind, die anderen werden ausgeblendet. Damit ist $\alpha_0(e)$ bekannt und man kann die nächste Aktivierung α_1 ausrechnen, wenn die Ähnlichkeitsfunktion σ definiert ist. Es sei noch erwähnt, daß man mit $\alpha_0(e)$ auch Merkmalsgewichtungen verarbeiten kann, wenn man anstelle der 1 die jeweiligen Gewichtswerte zuweist.

Das Retrieval eines Falles mittels Aktivierungspropagierung ist ein Prozeß mit drei Schritten:

Schritt 1: Initialaktivierung
Ist eine Anfrage q gegeben, so ist α_0 für alle IE bestimmt.

Schritt 2: Ähnlichkeitspropagierung
Die Aktivierung α_0 wird zu allen IE-Knoten $e \in E$ propagiert:

$$\alpha_1(e) = \pi_e(\sigma(e_1, e) * \alpha_0(e_1), \ldots, \sigma(e_s, e) * \alpha_0(e_s))$$

Schritt 3: Relevanzpropagierung
Das Ergebnis von Schritt 2 wird zu den Fallknoten $c \in C$ propagiert:

$$\alpha_2(c) = \pi_c(\rho(e_1, c) * \alpha_1(e_1), \ldots, \rho(e_s, c) * \alpha_1(e_s))$$

Das Ergebnis des Retrievals mit dem CRN ist eine Präferenzordnung der Fälle.

Theorem 3.1 (Präferenzordnung) Gegeben sei ein Basic Case Retrieval Net mit der Struktur
N = $[E, C, \sigma, \rho, \Pi]$ und der oben definierten Aktivierungsfunktion α_t. Das Retrievalergebnis für eine gegebene Anfragenaktivierung α_0 ist eine Präferenzordnung der Fälle mit fallenden Aktivierungen $\alpha_2(c)$ für die Fallknoten $c \in C$.

In einem klassischen CBR-System gibt es keinen Unterschied zwischen Fällen und Fallknoten. Dies ist bei einem BCRN anders. Hier unterscheidet man eingehender zwischen den konkreten Fällen $\hat{c} \in \hat{C}$ und den sogenannten Fallknoten $c \in C$. Die Fallknoten sind die Repräsentanten der konkreten Fälle während des Retrievals.

Definition 3.11 (Implementierungsähnlichkeit eines BCRN) *Für eine gegebene Fallbasis $\hat{C}$ wird mit einem Basic Case Retrieval Net mit der Struktur N = $[E, C, \sigma, \rho, \Pi]$ die Berechnung einer Ähnlichkeitsfunktion SIM implementiert, wenn für eine Initialaktivierung auf Basis der Anfrage q gilt:*

$$\forall q \in P(E) \; \forall \hat{c} \in \hat{C} \; : \; \alpha_2(c) = SIM(q, \hat{c}) \; .$$

In dieser Definition ist P(E) die Potenzmenge von E. Bevor diese Voraussetzung verifiziert wird, ist die Ähnlichkeitsfunktion zu definieren.

Definition 3.12 (Globales Ähnlichkeitsmaß) *Ein Ähnlichkeitsmaß SIM ist globaler Natur, wenn es mit einer Funktion $\phi : \mathbb{R}^k \rightarrow \mathbb{R}$ aus lokalen Merkmalsähnlichkeitsfunktionen der Art*

$$sim^i \ : \ Definitionsbereich^i \times Definitionsbereich^i \rightarrow [0,1]$$

mit dem Merkmalindex $i \in \mathbb{N}(1 \leq i \leq k)$ kombiniert wurde, so daß gilt:

$$SIM(x,y) = SIM([x^1,\ldots,x^k],[y^1,\ldots,y^k])$$
$$SIM(x,y) = \phi(sim^1(x^1,y^1),\ldots,sim^k(x^k,\ldots,y^k)).$$

Zu den bekanntesten globalen Ähnlichkeitsfunktionen gehört die gewichtete Summe der Merkmalsähnlichkeiten:

$$SIM([x^1,\ldots,x^k],[y^1,\ldots,y^k]) = \sum_{i=1}^{k} w_i * sim^i(x^i,y^i)$$

.

Es wird gefordert, daß globale Ähnlichkeitsfunktionen *monoton wachsen* [45].

Für jedes Paar $e_i.e_j \in E$ ist die lokale Ähnlichkeit nur dann ungleich Null, wenn beide IE zum selben Attribut gehören. Wenn man also beispielsweise das Attribut Farbe definiert, so hat man etwa $E_{Farbe} = \{rot, schwarz, blau, grün\}$. Ist das IE $e_i = blau \in E_{Farbe}$, so ist nur dann eine Ähnlichkeit definiert, wenn auch gilt: $e_j \in E_{Farbe}$, also etwa $e_j = schwarz$. Es gilt für *lokale* Ähnlichkeiten die folgende Wertezuordnung:

$$\sigma(e_i,e_j) = \begin{cases} sim^i(e_i,e_j) & : \text{ falls } type(e_i) = A_l \wedge type(e_j) = A_l \\ 0 & : \text{ sonst} \end{cases}$$

Der Index i ist der lokale Merkmalindex.

Burkhard hat vorgeschlagen, lokale Ähnlichkeitsfunktionen σ mit positiven wie auch mit negativen Wertebereichen einzuführen, und positive Werte $\sigma > 0$ als Akzeptanzwerte aufzufassen, die negativen Werte $\sigma < 0$ dagegen als Ablehnungsindikatoren zu betrachten. Er berichtet allerdings auch, daß die mehrfache Verarbeitung negativer Werte, also etwa für $\sigma,\ \alpha(e)\ oder\ \rho$ auf offene Fragen verweist.[3]

Für jedes IE $e \in E$ und für jeden Fallknoten $c \in C$ ist die Relevanz ρ nur dann ungleich Null, wenn IE e im Fall $\hat{c}$ enthalten ist:

$$\rho(e,c) = \begin{cases} 1 & : \text{ falls } type(e) = A_l \wedge \hat{c} = [\hat{c}^1,\ldots,\hat{c}^k] \wedge e = \hat{c}^l \\ 0 & : \text{ sonst} \end{cases}$$

Für jedes IE $e \in E$ berechnet die Propagierungsfunktion den *maximalen* Input:

$$\alpha_1(e) = \pi_e(\sigma(e_1,e) * \alpha_0(e_1),\ldots,\sigma(e_s,e) * \alpha_0(e_s))$$
$$\alpha_1(e) = max(\sigma(e_1,e) * \alpha_0(e_1),\ldots,\sigma(e_s,e) * \alpha_0(e_s))$$

Man erhält also für jedes IE $e \in E$ einen Maximalwert $\alpha_1(e_1,\ldots,\alpha_1(e_s)$, der als Input in α_2 eingeht.

Für jeden Fallknoten $c \in C$ berücksichtigt die Propagierungsfunktion nur die Aktivierungen derjenigen IE, die im Fall $\hat{c}$ enthalten sind:

[3] „However, negative values with different sources may cause problems, which need another treatment (e.g. by inhibiting strategy). Further investigations are necessary.“ Burkhard [45], p. 50

$$\alpha_2(c) = \pi_c(\rho(e_1, c) * \alpha_1(e_1), \ldots, \rho(e_s, c) * \alpha_1(e_s))$$
$$\alpha_2(c) = \pi_c(r_1, \ldots, r_s) = \phi(r_{i_1}, \ldots, r_{i_k}) \quad - \; wobei\; \rho(e_{i_j}, c) \neq 0.$$

In der obigen Definition ist gefordert worden, daß $\alpha_2(c) = SIM(q, \hat{c})$ gelten soll.

Es läßt sich über einfache Umformungen zeigen, daß diese Forderung erfüllt ist:

$$\alpha_2(c) = \pi_c(\rho(e_1, c) * \alpha_1(e_1), \ldots, \rho(e_s, c) * \alpha_1(e_s))$$
$$\alpha_2(c) = \pi_c(0, \ldots, 0, \rho(e_{c^1}, c) * \alpha_1(e_{c^1}), 0, \ldots, 0, \rho(e_{c^k}, c) * \alpha_1(e_{c^k}), 0, \ldots)$$
$$\alpha_2(c) = \pi_c(0, \ldots, 0, \alpha_1(e_{c^1}), 0, \ldots, 0, \alpha_1(e_{c^k}), 0, \ldots)$$
$$\alpha_2(c) = \pi_c(0, \ldots, 0, sim(q^1, \hat{c}^1), 0, \ldots, 0, sim(q^k, \hat{c}^k), 0, \ldots)$$
$$\alpha_2(c) = \phi(sim(q^1, \hat{c}^1), \ldots, sim(q^k, \hat{c}^k))$$
$$\alpha_2(c) = SIM([q^1, \ldots, q^k], [\hat{c}^1, \ldots, \hat{c}^k])$$
$$\alpha_2(c) = SIM(q, \hat{c}).$$

Die Implementierungsähnlichkeit entspricht also der Repräsentationsähnlichkeit:

$$SIM_{Imp} = SIM_{Rep} = SIM_{BCRN}.$$

Dementsprechend gilt Theorem 3.2.

Theorem 3.2 (Mächtigkeit eines BCRN) Für jeden endlichen Wertebereich einer beliebigen globalen Ähnlichkeitsfunktion kann ein Basic Case Retrieval Net konstruiert werden, welches diese Funktion implementiert.

Es läßt sich leicht beweisen, daß ein Basic Case Retrieval Net hinsichtlich seiner Implementierungsähnlichkeit einen Allgemeinheitscharakter hat [38].

Theorem 3.3 (Allgemeinheitscharakter) Für jeden beliebigen Wertebereich einer beliebigen globalen Ähnlichkeitsfunktion kann ein Basic Case Retrieval Net konstruiert werden, welches diese Ähnlichkeitsfunktion implementiert.

3.5.3 Bewertung

Die prototypischen Implementierungen haben gezeigt, daß CRNs für große Fallbasen im World Wide Web geeignet sind. CRN wurden auch erfolgreich für das sogenannte Textuelle Fallbasierte Schließen, also für das Retrieval von *know how* Dokumenten, erprobt. Charakteristikum dieses Ansatzes sind die Interaktivität sowie die Flexibilität, denn es ist nicht erforderlich, daß für die Initialaktivierung eine komplette Anfrage eingegeben wird, und darüber hinaus gibt es keine erwähnenswerten Anforderungen an die Fallrepräsentation: $\forall \hat{c} \in \hat{C} : \hat{c} \subset P(E)$. Allerdings liefert das Retrieval keine klassische Lösungsbeschreibung für die Anfrage, weil definitionsgemäß keine expliziten Lösungsbeschreibungen im Fallwissen enthalten sind.

Für Case Retrieval Nets ergeben sich folgende Bewertungen.

Vorteile

- Korrektheit und Vollständigkeit: Für CRNs gilt Korrektheit und Vollständigkeit, weil die repräsentierte Ähnlichkeit der implementierten Ähnlichkeit entspricht: $SIM_{Rep} = SIM_{Imp} = SIM_{BCRN}$.
- Effizienz: Der Retrievalaufwand für ein beliebiges BCRN wächst im ungünstigsten Fall *linear* mit der Größe der Fallbasis $|\hat{C}|$.[4]
- Wartung: CRNs sind wartungsfreundlich.
- Allgemeiner Ähnlichkeitsansatz: Das BCRN ermöglicht die Implementierung beliebiger globaler Ähnlichkeitsfunktionen.

[4] „In the worst case, the effort for retrieval in Case Retrieval Nets will growe linearly ... However, due to the distributed representation being used the effort will be considerably lower than for simple linear search. In most real world situations, the similarity model will further partition the set of IEs in that only certain subsets have in fact non-zero similarity. This further reduces the retrieval effort greatly.“ Lenz [38], p. 151

- Benutzerinteraktion: Im Unterschied zu klassischen fallbasierten Systemen erfolgt die Komplettierung der Anfrage interaktiv durch den Benutzer, der auch über die Terminierung entscheidet.

Nachteile

- Monotonie: Voraussetzung für die Implementierung eines BCRN ist eine Beschränkung des globalen Ähnlichkeitsmaßes auf monoton wachsende Ähnlichkeitsfunktionen SIM_{Rep}.
- Strukturneutral: Die Fallbeschreibungen eines CRN bestehen aus unstrukturierten IE und unterstützen keine strukturelle Ähnlichkeitsbeschreibung. Allerdings hat Lenz gezeigt, daß man Object-oriented CRNs (OCRN) erfolgreich für Diagnosezwecke einsetzen kann.
- Adaptionsneutral: Die Adaption wird von CRNs nicht direkt unterstützt. BCRN unterstützen nicht direkt die Integration von Adaptionswissen in der Retrievalphase. Effizientes Retrieval ohne Adaptionswissen kann dazu führen, daß der gefundene ähnlichste Fall suboptimal ist, weil er nicht direkt adaptierbar ist [48]. Wie der Name schon sagt, dient ein CRN primär dem Retrieval.
- Entwicklungslücke: Zur direkten Implementierung sind bisher nicht alle Fragen bezüglich negativer Werte wissenschaftlich gelöst.

Kapitel 4

Fallbasiertes Konstruieren – Case-Based Design

4.1 Kapitelinhalt

Dieses Kapitel befaßt sich zunächst mit der Motivation und der Geschichte des Case-Based Design (CBD).

In Kapitel 4.2 folgt eine Einführung zu diesem Thema, d. h. es wird eine generelle Vorgehensweise des Case-Based Design erläutert. Diese besteht aus den Phasen

- Fallsuche: Suche von zur Anforderung möglichst passenden Fällen;
- Transformation: Anpassung dieser Fälle an die Anforderung zum Zweck der Lösungsfindung;
- Fallspeicherung: Speichern der gefundenen Lösung.

Die Möglichkeiten der Benutzereinbindung in das System werden dazu aufgezeigt.

Die Abschnitte 4.3-4.5 beschäftigen sich mit verschiedenen CBD-Ansätzen. Einzelne Vorgehensweisen werden aufgezeigt und mit Hilfe von Beispielen erläutert. Hierbei wird in Kapitel 4.3 der genetische Ansatz betrachtet, der Vorteile von Case-Based Design-Methoden mit den Vorteilen von genetischen Algorithmen verbindet. Man erhält ein flexibleres, weniger an Regeln und Expertenwissen gekoppeltes System als bei vielen anderen Ansätzen. Genetische Case-Based Design-Methoden zeichnen sich durch Lernfähigkeit aus. Dafür sind sie allerdings nicht deterministisch. Dieses Kapitel orientiert sich stark an den Arbeiten von Hunt [56] und Tanaka [67].

Ein Grund für die Betrachtung der strukturellen Ähnlichkeit im Kapitel 4.4 liegt an der häufig komplexen Struktur eines Konstruktionsfalles. In vielen Bereichen lassen sich die vorliegenden Entwürfe nicht durch einfache Attribut-Wert-Paare repräsentieren. Für die Erkennung und Verarbeitung von Strukturen sind spezielle Methoden nötig. Im Mittelpunkt des Interesses steht die Arbeit von Börner [50].

Hierarchische Ansätze werden im Kapitel 4.5 einbezogen. Hier werden die gesuchten Gesamtkonstruktionen aus mehreren Teillösungen zusammengesetzt, die sukzessive gefunden werden. Dabei kann es zu dem Problem kommen, daß eine neue Teillösung eine schon eingebaute Teillösung untauglich macht. Dieser Abschnitt beruht zu allererst auf der Arbeit von Kumar und Krishnamoorthy [57].

Das darauf folgende Kapitel 4.6 widmet sich der Adaptation. Es wird kurz auf lernfähige Adaptions-Verfahren eingegangen, und zwar mit Bezug auf die Arbeit von Hanney [53]. Dazu wird ein Indizierungsverfahren für die Fallbasis vorgestellt, welches für vorhandene CAD-Pläne benutzt werden kann. Die Indizierung mit Hilfe von *Gestalts* wird in der Arbeit von Schaaf näher erklärt [66].

In Kapitel 4.7 werden die vorgestellten Ansätze mit anderen Ansätzen verglichen, die im gleichen Problembereich benutzt werden. Es folgt der Vergleich der einzelnen Ansätze untereinander. Dabei werden neben den Unterschieden auch Verknüpfungsmöglichkeiten aufgezeigt.

Engineering Design im Sinne dieses Kapitels ist ein ausgedehnter Bereich. Es sind alle Gebiete eingeschlossen, in denen etwas im Rahmen der Konstruktionstätigkeit oder am Reißbrett entworfen wird, also beispielsweise in der

Architektur, im Maschinenbau, in der Elektrotechnik oder im Straßenbau. In diesen und anderen Bereichen läßt sich durch schnelle und vernünftige Planung viel Geld einsparen und der Konstruktionsprozeß beschleunigen. Es handelt sich um Problembereiche, deren Bearbeitung mit meist viel Erfahrung (Hydrauliksysteme, Häuser, Autobahnbrücken) erfordert, und durch eine schwierige Modellbildung gekennzeichnet ist. Bei der Konstruktion eines neuen technischen Objektes greift der erfahrene Konstrukteur (Architekt, Maschineningenieur, etc.) regelmäßig auf sein Wissen über frühere Objekte und die Art und Weise ihrer Konkretisierung und Planung zurück. Das neue Konstruktionsobjekt wird also mit Hilfe dieser Erfahrung konzipiert. Dieses Erfahrungswissen ist jedoch für den Ingenieur nur schwer als Regelmenge oder Bedingungskonstrukt (englisch: constraints) zu formulieren und zu formalisieren. Im Unterschied zur Diagnose hat sich die Automatisierung von Konstruktionsprozessen mit Hilfe von großen Regelbasen derzeit in der industriellen Praxis noch nicht durchgesetzt [70]. Es ist aber sinnvoll, Erfahrungen über alte, schon geplante und realisierte Objekte erneut ins Gedächtnis zu rufen, und sie den Bedürfnissen der neuen Problemstellung entsprechend zu modifizieren bzw. zu überarbeiten.

Hier setzt das Case-Based Design (deutsch: *fallbasiertes Entwerfen, fallbasiertes Konstruieren*) an, das auf eben diesem Prinzip des *Case-Based Reasoning* (deutsch: *fallbasiertes Schließen*) aufsetzt. Alte Entwürfe werden für aktuelle Konstruktionsprobleme wiederverwendet und kombiniert sowie modifiziert. Man muß also die Lösungen nicht von Grund auf neu konzipieren; anstelle der Neukonstruktion wird eine Anpassungskonstruktion oder – im ähnlichsten Fall – eine Variantenkonstruktion ausführbar.

Die Entwickler von Case-Based Design-Systemen benötigen regelmäßig kein vollständiges Hintergrundwissen über das Problemgebiet, die sogenannte *Domäne* (englisch: *domain*). Theoretisch reichen bei nicht adaptierenden CBD-Systemen sogar kleine Änderungen aus, um das System für andere Domänen zu nutzen. Allerdings sind in der industriellen Praxis viele Case-Based Design-Systeme für spezielle Anwendungen entwickelt worden.

Durch das Aufnehmen der vom System neu geschaffenen Fälle in die Menge der alten Lösungen – Erweiterung der Fallbasis – kommt es zu einem Lerneffekt: das CBD-System kann nicht nur von außen verbessert werden, es erhält auch neues Fallwissen durch die selbst geschaffenen und validierten Entwürfe und Problemlösungen. Eine derartige Verbesserung von außen durch Experten ist recht einfach: neues Wissen wird einfach in Form eines neuen Falles eingebracht.

Die CBR-Techniken werden seit Ende der achtziger Jahre vermehrt im Konstruktionsbereich eingesetzt. Die ältesten in der Literatur namentlich bekannten Systeme sind STRUPLE für den Anwendungsbereich Bauwesen und CYCLOPS für Landschaftsdesign aus dem Jahre 1988 sowie das System KRITIK / KRITIK2, welches unter anderem auch für den Entwurf elektrischer Schaltkreise entwickelt worden ist [68], [69], [75]. Seitdem wurden eine kaum überschaubare Vielzahl von Systemen entwickelt [58]. In Deutschland sind vor allem die Forschungen im FABEL-Projekt[1] erwähnenswert, deren Ergebnisse auch hier eingeflossen sind (4.4).

Gegenwärtig läuft das bis 2003 geplante europäische Projekt WINDS (Web-based INtelligent Design tutoring System), an dem 20 Universitäten aus 10 europäischen Ländern beteiligt sind. Ziel dieses ehrgeizigen Forschungsvorhabens ist die Entwicklung eines Web-basierten intelligenten Tutoring-Systems für Studenten der Ingenieurwissenschaft [72].

4.2 Aufbau von Case-Based Design Systemen

Der generelle Ablauf des Case-Based Design wird hier mit Hilfe der Abbildung 4.1 veranschaulicht. Zur Wissensrepräsentation ist zu erwähnen, daß in einigen jüngeren Systemen die Trennung zwischen Problem und Lösung aufgehoben wurde: der Fall wird eher als Kette von Informationseinheiten oder *snippets* (deutsch: Schnipsel) gesehen. Die Anforderungen geben einige dieser Einheiten als Lösungsfundament vor. Dieses Fundament wird sukzessive mit weiteren Einheiten angereichert (Adaptation), bis die Informationsmenge ausreichend, und der Fall damit gelöst ist.

4.2.1 Die drei großen Schritte des Case-Based Design

Fallsuche

In der 1. Phase werden aus der Fallbasis mit den „alten“ Entwürfe diejenigen Fälle heraus gesucht, die den Anforderungen am nächsten kommen. In dem Ähnlichkeitmaß zur Fallsuche können neben den normalen quantitativen – meist numerischen – oder qualitativen Werten (Verhalten oder funktionelle Aspekte) auch die Erfahrungen bezüg-

[1] Siehe etwa [50], [51], [52] oder [60], [61]

lich der Praxistauglichkeit (Bewährung) verarbeitet werden, man kann also evaluieren, wie gut die alten Lösungen in der industriellen Praxis funktionieren. Die von Experten bemessene Qualität, dies ist meist ein numerischer oder linguistischer Wert, wird als *Kritik* oder *Gütemaß* in die Fallrepräsentation aufgenommen.

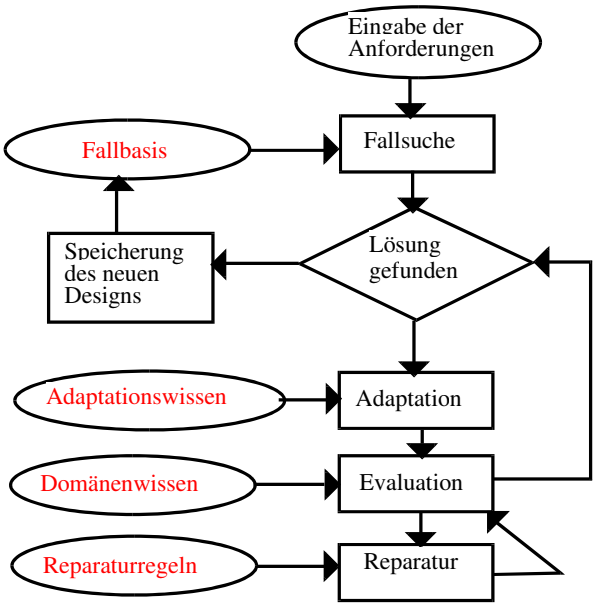


Abbildung 4.1: Die Schritte des Case-Based Design nach Hunt (1995)

Falltransformation

Die Falltransformation unterteilt sich in vier Bereiche:

- Vergleich (im Bild: *Lösung gefunden*)
- Adaptation
- Evaluation
- Reparatur

Nur selten sind die Anforderungen der gefundenen Fälle im Vergleich so ähnlich zu den aktuellen Anforderungen, daß einer dieser Fälle direkt als Lösung genommen werden kann. Deswegen müssen die Lösungen modifiziert werden. Für die Adaptation der Fälle gibt es drei grundlegende Techniken, die auch gemischt verwendet werden können:

1. Modifikationen an einem oder mehreren ähnlichen Fällen, um die erwünschte Lösung zu erhalten. Hierauf wird in Abschnitt 4.2.2 eingegangen; es werden drei Möglichkeiten angegeben.
2. Lösungsfindung, indem die gleiche Entwurfssstrategie angewendet wird, wie bei einem ähnlichen Fall. Dazu ist es notwendig, neben den Anforderungen und der Lösung auch die Schritte von den Anforderungen zur Lösung in der Fallbasis abzulegen.
3. Schema- oder Strukturbasierter Lösungsübertrag: Dieses Thema wird in Abschnitt 4.4 ausführlich behandelt. Hierbei versucht man, die Problembeschreibungen einiger alter Fälle in eine abstrakte Form zu bringen, und die Lösungen dann auf diesen abstrakten Versionen zu finden. Zuletzt muß dann die abstrakte Lösung wieder in eine konkrete Lösung verwandelt werden.

Die Adaptation ist oftmals in starkem Maße domänenabhängig. Häufig werden hier Regeln verwendet, die man natürlich erst mit Hilfe von Domänenwissen finden muß.

Eine andere, ebenso Domänenwissen erfordernde Methode besteht darin, den Fall immer weiter zu verändern, bis gewisse Bedingungen (constraints) erfüllt sind. Häufig wird dabei zwischen harten (auf jeden Fall zu erfüllenden) und weichen (möglichst zu erfüllenden) constraints unterschieden. Neben der Möglichkeit, diese Regeln und constraints *per Hand* in das System einzufügen, gibt es auch Ansätze, die Regeln selbst mit Hilfe von Case-Based Reasoning oder durch unabhängige, lernfähige Algorithmen zu bestimmen. Für die domänenunabhängige Nutzbarkeit von Case-Based Design ist es wichtig, daß dieses *hart kodierte*, domänenabhängige Wissen durch domänenunabhängiges ersetzt wird. In *GenCad* z. B. versuchen Forscher, die Vorteile von neuronalen Netzwerken und genetischen Algorithmen zu diesem Zweck zu koppeln [58].

Nach der Adaptation muß evaluiert werden, ob der modifizierte Fall tatsächlich den Anforderungen entspricht. Dabei können wiederum verschiedene Techniken genutzt werden. Unter anderem bietet sich hier auch eine Zusammenarbeit mit dem Benutzer an. Andere Möglichkeiten sind die Simulation, vor allem wieder die Prüfung auf constraints.

Sollte sich herausstellen, daß der modifizierte Fall keine Lösung darstellt, erweist es sich als günstig, wenn man den Grund dafür finden kann. Dann kann unter Umständen die Lösung repariert und danach doch genutzt werden. Die Reparatur wird häufig mit Hilfe von heuristischen Regeln vorgenommen[2].

Als Beispiel soll hier eine Autobahnbrücke dienen. Der modifizierte Fall ist eine Holzbrücke mit 6 Spuren. Allerdings dürfen derartige Brücken aus Sicherheitsgründen nicht gebaut werden. In diesem Kontext liegt folgende Reparaturregel vor: *Wenn die Lösung eine Holzbrücke mit 6 Spuren ist, dann ersetze das Holz durch Beton*. Daraus resultiert dann eine Betonbrücke mit 6 Spuren, also eine mögliche Lösung – sofern keine anderen Bedingungen dagegen sprechen. Ungünstiger wäre die Regel: *Wenn die Lösung eine Holzbrücke mit 6 Spuren ist, dann verwerfe die Lösung*. Dies ist gleichbedeutend mit der Tatsache, daß keine Reparatur (Adaption) möglich ist. Dann muß ein neuer Anlauf zur Lösungsfindung stattfinden. Hierbei kann es auch dazu kommen, daß der gleiche Fall wie zuvor genommen wird, auf den dann aber andere Adaptionsregeln angewendet werden.

Oftmals gibt es für ein Problem mehrere Adaptationsalternativen. Es kann sinnvoll sein, diese nacheinander auszuprobieren und schließlich die erfolgreichste davon zu nehmen, statt nur eine auszuwählen. Dafür steigt natürlich die Laufzeit.

Fallspeicherung

Einfache Systeme speichern sämtliche gefundenen Lösungen. Einerseits kann das dazu führen, daß die Fallbasis unpraktisch groß und die Fallsuche unnötig zeitintensiv wird. Dieses Verhalten ist vor allem dann ärgerlich, wenn viele sehr ähnliche Fälle gespeichert werden, die eigentlich schnell voneinander abgeleitet werden könnten. Andererseits kann die Qualität der Fallmenge leiden, wenn eine Lösung aufgenommen wird, die die Anforderungen in gerade ausreichendem Maß erfüllt. Neue Probleme werden dann unter Umständen ebenfalls sehr schlecht gelöst. Es kann durchaus eine Fallbasis mit weniger ähnlichen Fällen geben, die Anforderungen weitaus besser erfüllt, und im Endeffekt für die aktuellen Anforderungen zu besseren Lösungen führt.

In manchen Domänen ist es auch nicht empfehlenswert, zu unähnliche Fälle aufzunehmen, weil damit weitaus mehr Modifikationen und Reparaturen ausgeführt werden müssen. Hieraus resultiert ein höherer Laufzeitaufwand und ein größerer Reparaturregel-Bedarf. Aus diesen Gründen werden bei vielen Systemen nicht automatisch sämtliche Fälle abgespeichert; statt dessen wird eine vernünftige Auswahl getroffen. Häufig geschieht dieses nach dem Prinzip der Mindestunähnlichkeit. Das bedeutet, daß der gefundene neue Fall sich um einen gewissen Wert von dem ähnlichsten in der Fallbasis vorhandenen Fall unterscheiden muß. Außerdem kann die Güteklasse des Falles, also die oben erwähnte Experten-Kritik, in die Entscheidung einfließen. Das Aufnehmen von externen Lösungen verbessert häufig die Qualität der Fallbasis. Diese Lösungen sollten sinnvollerweise eine Innovation oder eine besondere Entwurfsidee einbringen.

4.2.2 Transformationsarten

Hier sind drei typische, generelle Vorgehensweisen aufgelistet, die Aufschluß darüber geben, wie die alten Fälle in der Adaptationsphase verändert werden.[3]

- Bei den konstruktiven oder kollaborativen Transformationstechniken werden mehrere Fälle ausgewählt. Diese

[2] Eine andere Möglichkeit der Reparatur besteht darin, daß der Benutzer die Reparatur selbst durchführt. Das kommt vor allem bei Assistenzsystemen vor.

[3] Siehe zum Vergleich vor allem Kumar und Krishnamoorthy [57], S. 163

werden als *Puzzle* zur Lösung zusammengesetzt. Der genetische Ansatz aus Kapitel 4.3 ist ein gutes Beispiel dafür.

- Rahmentransformationstechniken gehen wie folgt vor: Der ähnlichste Fall wird ausgesucht, um als genereller Rahmen für die Lösung zu dienen. Nicht vorhandene oder unpassende Daten werden durch Teillösungen oder Daten von anderen Fällen ausgefüllt oder ersetzt. Beispielsweise kann der ähnlichste Fall bei einem Hausbauproblem ein vierstöckigen Haus sein. Die einzelnen Stockwerke (Teillösungen) werden aus einem drei- und einem zweistöckigen Haus genommen. Ein Beispiel findet man in Kapitel 4.5.
- Hybride Transformationstechniken: Wenn kein passender Fall für den Rahmen gefunden wird, dann wird dieser mit anderen Methoden gebaut, z. B. mit Heuristiken. Dieser *künstliche* Lösungsrahmen wird wiederum durch Teillösungen und Daten mehrerer Fälle gefüllt.

4.2.3 Möglichkeiten der Benutzeranbindung

Der Traum eines Benutzers ist ein Programm, bei dem er nur die Anforderungen eingeben muß und den Rest erledigt das Programm. Möglichst sollte dieses für alle möglichen Domänen funktionieren, und das in kurzer Zeit mit wenig Materialaufwand (Speicher und Prozessor). Wie schon erwähnt, sind die meisten Case-Based Design-Systeme jedoch für spezielle Probleme konzipiert.

In einigen, nur *assistierenden Systemen* fällt auch ein Großteil der Arbeit auf den Benutzer zurück. Entscheidungsunterstützende, sehr einfache Systeme beschränken sich darauf, nur die ähnlichsten Fälle zu suchen. Weitergehende Systeme, z.B. im Bereich Hausplanung, erlauben es zusätzlich, für Teilbereiche (Zimmer, Korridore) ebenfalls ähnliche Fälle zu suchen, so daß der Benutzer nacheinander kleine Lösungen finden, bearbeiten und zusammenbauen kann.

Die Adaptation jedoch bleibt die Aufgabe des Benutzers. Um dessen Arbeit zu vereinfachen, werden hier häufig graphisch ausgefeilte Darstellungen des bearbeiteten Objektes präsentiert (z. B. *CaseCad* [58]).

Der Übergang zwischen diesen einfachen und den vollautomatischen Systemen ist fließend. So gibt es Implementierungen, die die Adaptation teilweise oder ganz übernehmen, die Beurteilung und Reparatur aber dem Benutzer überlassen; und andere, die nur die durch den Benutzer erfolgende Adaptation überwachen und unter Umständen Verbesserungsvorschläge machen.

Bei nicht wenigen Implementierungen handelt es sich um *assoziierte Systeme* : die Adaptation geschieht nicht direkt im System, sondern durch sogenannte *co-reasoner*, also durch Zusatzprogramme. Bei größeren Projekten ist Case-Based Reasoning nur ein Weg, um zum Erfolg zu kommen. Andere wissensbasierte Methoden werden in derartigen Projekten gleichfalls benutzt.

4.3 Der genetische Ansatz

Genetische Algorithmen, erfunden 1975 von John Holland, stellen die Übernahme von biologischem Wissen (Genetik, Evolution) in die Informatik dar [55]. In diesen Algorithmen ist die natürliche Evolution (Kreuzung, Mutation und Auslese) nachempfunden. Anstelle von Lebewesen werden Daten betrachtet und verändert. Case-Based Design-Systeme zeichnen sich durch *Fortpflanzung* aus: aus bestehenden Fällen werden neue konstruiert und in die Fallbasis eingegliedert. Der genetische Algorithmus geht noch weiter in diese Richtung, indem für die Adaptation wie in der Natur Kreuzungen und Mutationen verwendet werden [73].

4.3.1 Die Vorgehensweise

Der genetische Algorithmus in Bezug auf Case-Based Reasoning funktioniert ähnlich wie die im Kapitel 4.1 erklärten Schemata, allerdings mit einem gravierenden Unterschied: *Herkömmliche* Case-Based Design-Systeme unternehmen einen neuen Anlauf mit einem Fall aus der Fallbasis, wenn durch die Adaptation (und evtl. Reparatur) keine Lösung gefunden wird. Bei evolutionärem Case-Based Design wird hingegen mit den modifizierten Fällen weitergearbeitet.

Zunächst werden mehrere Fälle mit Hilfe des Ähnlichkeitsmaßes aus der Fallbasis herausgesucht (es sollte sinnvollerweise mehr als einer sein; in den meisten einfachen Beispielen werden 4 Fälle genommen). Wenn in den gefundenen

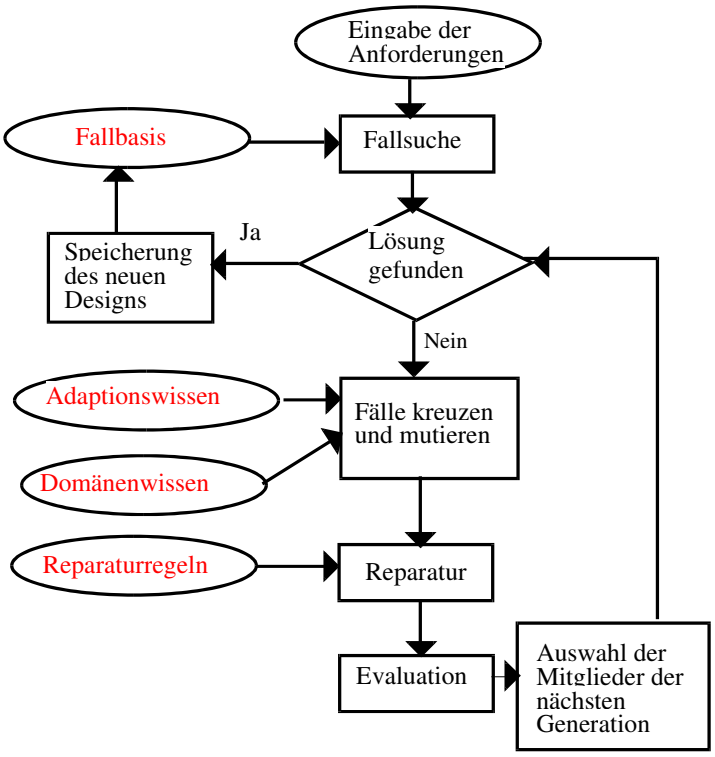


Abbildung 4.2: Das evolutionäre Case-Based Design-System nach Hunt (1995)

Fällen keine sofort die Anforderung erfüllende Lösung gefunden werden kann, beginnt die Adaptation. Hier geschehen zwei Dinge: vorhandene Fälle werden miteinander gekreuzt, und sie werden mutiert. Für Kreuzung und Mutation können mehrere Methoden verwendet werden, auch in ein und demselben Programm.

Kreuzung

Es gibt mehrere Möglichkeiten für die Kreuzung, die mehr oder weniger abhängig von der Art der beschreibenden Werte sind. Bei allen Kreuzungen werden 2 Fälle genommen, aus deren Werten dann ein neuer Fall erzeugt wird. Die beiden ursprünglichen Fälle nennt man auch die *Eltern*, den neuen Fall dementsprechend den *Nachkommen*.

Vier der gebräuchlichsten Möglichkeiten sind hier aufgeführt:

1. Die der natürlichen Kreuzung ähnlichste Methode besteht darin, daß man jede (meist durch Werte beschriebene) Eigenschaft des Falls als *Chromosom* auffaßt. Zum Beispiel soll in einer bestimmten Domäne ein Fall durch drei numerische Werte gekennzeichnet sein, vielleicht Höhe, Breite und Länge. Für zwei Fälle, die jetzt miteinander gekreuzt werden sollen, liegen diese drei Werte vor: Fall 1 hat die Werte (also Chromosomen) x, y und z, Fall 2 die zu denselben Attributen gehörenden Werte a, b und c. Der Nachkomme (also neue Fall) wird gebildet, indem man einige Chromosomen von 1 nimmt und einige von 2. Das mag zum Beispiel die Kombination x, b und z ergeben, also tatsächlich einen neuen Fall. Geschickterweise werden bei dieser Methode meistens sogar zwei Nachkommen erzeugt, die zusammen alle Merkmale der Eltern abdecken: der zweite Nachkomme besteht dann aus a, y und c.

2. Als Verfeinerung der ersten Methode kann man die Kreuzung von der Häufigkeit der einzelnen Chromosomen in der Fallbasis abhängig machen. Wenn das Chromosom des 1. Elternteils häufiger vorkommt als das entsprechende Chromosom des Partners, also *dominant* ist, nimmt man dieses. Man kann die Dominanz auch als Wahrscheinlichkeit ausdrücken (z.B. wenn Chromosom A 5 mal vorkommt, Chromosom B nur 1 mal, dann wird Chromosom A mit 5 mal höherer Wahrscheinlichkeit genommen). Bei großen Fallbasen oder bei nur geringem Vorkommen vieler Chromosomen ist diese Methode zu umständlich.

3. Wenn man einzelne Chromosomen als Bitkette betrachtet, läßt sich eine wirkungsvolle Kreuzung erzielen, indem man ein zufälliges Bit in den Chromosomen auswählt. Ab dieser Stelle werden die Bitwerte in den beiden

Chromosomen ausgetauscht. Einfache Algorithmen begnügen sich mit diesem Austausch, andere tauschen nur bis zu einer vorgegebenen Länge oder einer weiteren zufälligen Stelle.

4. Eine weitere Möglichkeit gibt es für numerische Werte: diese kann man mit Hilfe von mathematischen Funktionen kreuzen. Die einfachste Möglichkeit ist hier natürlich das Mittel. Beim Beispiel zur 1. Möglichkeit würde man also $\frac{(a+x)}{2}$, $\frac{(b+y)}{2}$ und $\frac{(c+z)}{2}$ errechnen. Andere zu nutzende Funktionen sind je nach Anliegen etwa die Multiplikation oder das geometrische Mittel.

Andere Kreuzungsmethoden können größtenteils mit Domänenwissen konzipiert werden.

Mutation

Die Mutation von vorhandenen Fällen geschieht dadurch, daß bei einem vorhandenen Fall ein zufälliges Chromosom – also der Wert eines bestimmten Attributes – verändert wird.

Bei numerischen Werten kann dieses in der Art geschehen, daß man einen kleinen Betrag zum Wert addiert, vom Wert subtrahiert, oder den Wert mittels Multiplikation verändert.

Bei anderen Datentypen empfiehlt es sich wiederum, die Bit-Repräsentation der Chromosomen zu betrachten. Die Werte eines zufälliges Bit oder sämtlicher Bits einer Bit-Kette mit zufälligem Anfang und zufälliger Länge werden invertiert (also von 0 auf 1 bzw. von 1 auf 0).

Bei der Invertierung wählt man sich ebenfalls eine längere Bit-Kette in einem Chromosom, und invertiert diese (d. h. das erste Bit der Kette wird mit dem letzten getauscht, etc.). Die beiden letztgenannten Techniken sind natürlich auch für numerische Werte anwendbar.

Evaluation, Reparatur und die Auswahl der nächsten Generation

Die Evaluation und Reparatur ist bei genetischen Case-Based Design-Systemen äußerst wichtig, da es hier auch zu sehr unwahrscheinlichen und teilweise absurden Lösungen kommen kann. Für die Reparatur gibt es keine spezielle *evolutionäre* Vorgehensweise, es werden die Techniken aus Abschnitt 4.2 benutzt (constraint-Erfüllung, Regeln).

Im Adaptationsteil wurden mit Hilfe von Mutationen und Kreuzungen mehrere Fälle erschaffen. Bei der Evaluation werden diese ebenso wie ihre Eltern mit Hilfe des *Fitneßmaßes* auf ihre Tauglichkeit geprüft. Dieses Fitneßmaß ist häufig dem *Ähnlichkeitsmaß* gleich.

Aus den Fällen werden dann die nach dem Fitneßmaß am besten bewerteten herausgesucht. Meistens nimmt man die gleiche Anzahl wie schon bei der Fallsuche. Diese ausgesuchten Fälle werden als *neue Generation* zusammengefaßt, und als Eltern für weitere Durchläufe genommen. Dieses Prinzip nennt man *Survival of the fittest* (*Überleben der Bestgeeigneten*).

Diese Prozedur (Kreuzung/Mutation der aktuellen Generation, Reparatur, Auswahl der Besten) wird solange durchlaufen, bis endlich ein Fall gefunden wird, der den Anforderungen entspricht. Geschickterweise sollte außerdem ein Abbruch nach einer gewissen Anzahl von Generation möglich sein: es besteht sonst die Gefahr einer Endlosschleife, zumindest einer zu langen Laufzeit. Es kann durchaus vorkommen, daß eine Generation mit ihrer Vorgängergeneration vollkommen übereinstimmt; genauso aber, daß diese vollständig divergieren.

4.3.2 Entwurfsbeispiel aus der Robotik

Ein fallbasiertes System soll Roboter für bestimmte Aufgaben entwerfen. Der Roboter wird durch die Werte Material, Gewicht (in kg), Größe (in cm), Tragkraft und Techgrad (Qualität der Technik) beschrieben, sowie durch die abhängigen Werte

$$Druckstabilität = 10000 * Gewicht/Grösse^2 \quad \text{(Annäherung)}$$

und

$$Preis = Gewicht * e * f.$$

Dabei ist e eine Materialkonstante und f eine Techgrad-abhängige Konstante. Die Anforderungen, die der Benutzer jetzt stellt, sind folgende[4]:

[4]Preise werden im folgenden in der Einheit TDM (1000 DM) angegeben

Probleme	Größe (G_v)	Druckstabilität (D_v)	Preis (P_v)
Problem	155 cm	170	100 TDM

Der Index v bedeutet „verlangt“.

Es soll ein Roboter entworfen werden, der diesen Anforderungen entspricht. Dazu wird das folgende Fitneßmaß entworfen:

$$F = \sqrt{(D_i - D_v)^2 + (G_i - G_v)^2 + p}$$

Es werden also füer zwei Kriterien (Größe G, Druckstabilität D) die Differenzen zwischen dem Anforderungs- und dem Fallwert gebildet und quadriert. Dazu wird der Wert p addiert, der wie folgt definiert wird:

$$p = \begin{cases} P_i - P_v * 100, & \text{wenn } P_i > P_v \\ 0, & \text{sonst} \end{cases}$$

Aus der Summe der beiden Differenzen und p wird dann die Wurzel gezogen. Man sieht, daß in diesem Beispiel höhere Kosten schnell zu einem sehr hohen Fitneßwert führen[5].

Das Fitneßmaß wird auch als Ähnlichkeitsmaß für die Fallsuche eingesetzt[6]. Wir nehmen an, daß folgende vier Fälle gefunden werden:

Nr.	Material	Gewicht	Höhe	Stabilität	Tragkaft	Techgrad	Preis (P_i)	Fitneß
1	010 Eisen	300 kg	140 cm	153	500 kg	0001 zu alt	80 TDM	22.67
2	011 Stahl	400 kg	140 cm	204	500 kg	0100 normal	100 TDM	37.16
3	011 Stahl	500 kg	180 cm	154	600 kg	0101 gut	120 TDM	53.67
4	011 Stahl	450 kg	160 cm	176	550 kg	1100 sehr gut	110 TDM	32.57

Fall 1 kommt sogar beinahe als Lösung in Frage, die Lösung soll aber eine Fitneß von 5 oder weniger haben[7]. Zunächst werden 4 neue Fälle durch zufällige Kreuzung jeweils zweier Fälle erschaffen. Dabei werden die numerischen Werte gemittelt, Techgrad und Material zufällig aus den beiden Möglichkeiten ausgewählt. Zwei Beispiele sind der aus Fall 2 und 3 (also den Fällen mit schlechteren Fitneßwerten) entstandene Fall 5 und der aus den besseren Fällen 1 und 4 errechnete Fall 6:

Nr.	Material	Gewicht	Höhe	Stabilität	Tragkraft	Techgrad	Preis	Fitneß
5	011 Stahl	450 kg	160 cm	176	550 kg	0101 gut	108 TDM	29.34
6	011 Stahl	375 kg	150 cm	167	525 kg	0001 zu alt	90 TDM	5.83

Beide Fälle haben einen besseren Fitneßwert als ihre Eltern. Jetzt wird für alle 8 Fälle (also die 4 ursprünglichen und die 4 gekreuzten) ein Mutationsfall erzeugt, indem ein beliebiger Wert verändert wird. Mit ein bißchen Glück wird der Fall 6’ erzeugt, bei dem die Höhe von Fall 6 um 3 cm vergrößert wird (dann resultiert daraus eine Fitneß von 3.6, und der Algorithmus kann beendet werden). Dies wäre ein schnelles Ende. Normalerweise müssen einige Generationen berechnet werden, um zu einem akzeptablen Ergebnis zu kommen. Dazu wird die Fitneß der 16 Fälle (4 ursprüngliche, 4 gekreuzte, 8 mutierte) verglichen. Die 4 besten werden für *die nächste Runde* genommen. Nicht beachtet wurde bei diesem Beispiel die Tatsache, daß tatsächlich noch Reparaturregeln nötig sein können. Zum Beispiel könnte eine Bedingung lauten, daß das Gewicht höchstens das 2.8fache der Größe sein kann. Dadurch müßte der Fall 5 verworfen oder repariert werden.

4.3.3 Bemerkungen und Beurteilung

Evolutionäres Case-Based Design profitiert aus der Vereinigung der Vorteile des Case-Based Design und des evolutionären Ansatzes. Im Gegensatz zu und vorteilhaft gegenüber vielen anderen evolutionären Algorithmen startet man

[5] Ein Fall mit zu hohen Kosten wird also schnell für untauglich befunden.

[6] Es ist nicht notwendig, daß Ähnlichkeit und Fitneßmaß identisch sind. Außerdem sollte beachtet werden, daß in diesem Beispiel die Fitneß umso kleiner ist, je mehr sich die Fälle ähneln. In der Praxis wird häufig der umgekehrte und sprachlich einleuchtende Weg gewählt: je höher die Fitneß, desto besser. Ein Fall ist dann ähnlich genug, wenn ein gewisser Schwellenwert überschritten wird. Im hier gegebenen Beispiel ist es genau umgekehrt: der Schwellenwert muß unterschritten werden.

[7] Der Schwellenwert 5 muß sorgfältig und nach vielen Probeläufen ausgewählt werden: einerseits darf kein zu unähnlicher Fall durch einen zu hohen Wert ermittelt werden, andererseits führt ein zu niedriger Wert zu wachsendem Rechenaufwand.

nicht *von Null*, sondern mit früheren Lösungen. Das Case-Based Design-System wiederum wird flexibler, denn mehrere Alternativen werden gleichzeitig betrachtet. Es muß keine Entscheidung für einen Fall beziehungsweise die beste Methode zur Modifizierung getroffen werden, die Modifizierungen sind nicht in enge Regeln eingeschlossen. Außerdem hängen Case-Based Design-Systeme sehr von den früheren Fällen ab, evolutionäre Case-Based Design-Systeme hingegen sind nicht so sehr von der Vergangenheit abhängig, da ja ein Großteil des Algorithmus auf zufälligen Veränderungen beruht. Durch die Arbeit mit mehreren Fällen gleichzeitig erhöht sich natürlich die Laufzeit. Nachteilig wirkt sich außerdem aus, daß ein genetisches Case-Based Design-System nicht deterministisch sein kann; die Lösungen kommen auf *zufällige* Weise zustande. Somit kann man auch nicht die Lösungsfindung erklären, den Weg vom Problem zur Lösung nachzeichnen. Letzteres ist für evolutionäres Case-Based Design allerdings nicht nötig.

4.4 Strukturelle Ähnlichkeit als Leitmotiv

Der in diesem Kapitel vorgestellte Ansatz geht von zwei grundlegenden Überlegungen aus:

1. Viele Probleme lassen sich nicht durch die Veränderung von numerischen Werten lösen.
2. Es ist sinnvoll, Fallsuche, Vergleich und Adaptation nicht separat voneinander zu betrachten.

Wie schon der Titel dieses Abschnittes besagt, werden nicht (oder nicht nur) die Werte, sondern die grundsätzliche Struktur des Problems und der Lösung betrachtet.

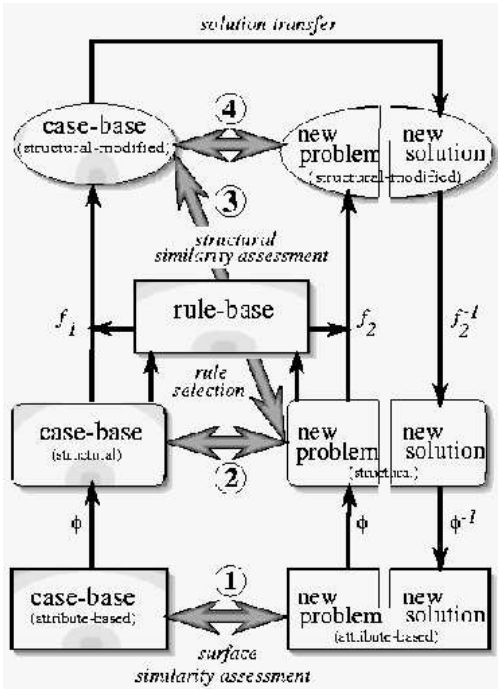


Abbildung 4.3: Strukturelle Ähnlichkeit nach Börner (1994).

4.4.1 Eine Vorgehensweise

Die Fälle sind häufig in attributbasierter Form abgespeichert. Wie gewöhnlich werden zunächst ähnliche Fälle aus der Fallbasis herausgesucht. Es handelt sich allerdings eher um ein Aussieben, eine Vorauswahl: zu unähnliche Fälle werden verworfen. Im Gegensatz zu anderen Systemen wird eine größere Zahl von Fällen als für die Lösungsfindung geeignet akzeptiert, nur *augenscheinlich* ungeeignete Fälle werden nicht in Betracht gezogen.

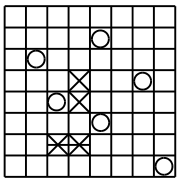


Abbildung 4.4: Spielsituation beim Schiffe versenken

Als Beispiel soll in diesem Kapitel das Spiel *Schiffe versenken*[8] dienen. Das fiktive Programm soll als Anforderung einen Spielstand erhalten und für diesen den nächsten Zug ermitteln. Nebenstehend ist ein Spielstand, ergo eine Anforderung angegeben. Dabei bedeutet O einen Fehlschuß ins Wasser, X einen Treffer auf ein Schiff. Die beiden eingekästelten X besagen, daß die Treffer an diesen Stellen reichten, ein Schiff zu versenken. Die Frage ist also: Auf welche Stelle soll als nächstes geschossen werden?

Jeder Fall aus der Fallbasis besteht aus einem Spielstand und der dazu gefundenen Lösung, also dem nächsten Schußziel. Ein Spielstand wird zu Anfang als Matrix dargestellt mit den Eintragsmöglichkeiten *Treffer*, *Treffer in gesunkenem Schiff*, *kein Treffer* und *noch nicht versucht*. Aus den Fällen der Fallbasis wird wie gesagt eine Vorauswahl getroffen.

Als nächstes werden die Anforderungen des aktuellen Problems und der herausgesiebten alten Fälle in eine strukturelle Form gebracht, in der sie durch abstrakte Ausdrücke (wie z.B. Richtungsangaben, *Reihe*, *regulär*) beschrieben werden. Diese Abbildung vollführt die Transformationsfunktion Φ. Die Daten des Beispiels können durch Anwendung von Φ abstrakter als *abgeschossenes Schiff in Koordinate* $(4, 4)$*, 2 Felder, nach unten* und *angeschossenes Schiff in Koordinate* $(3, 7)$*, 2 Felder, nach rechts* abgelegt werden (dazu kommen dann die Koordinaten der Fehlschüsse wie z.B. *Koordinate* $(8, 8)$*, kein Treffer*).

Nun liegen die alten Fälle und die Anforderungen in abstrakter Form vor. Als nächstes müssen sie aneinander angepaßt werden. Dafür nimmt man die Funktionen f_1 und f_2. Diese Funktionen beinhalten Modifikationsregeln, die auf die Anforderung (f_2) bzw. die alten Fälle (f_1) so lange angewendet werden, bis endlich ein Fall zur Anforderung paßt. Zu beachten ist dabei, daß die Modifikationen in f_1 und f_2 wirklich paarweise zusammengehören: für jede Modifikation der Anforderung werden dementsprechend die alten Fälle verändert. Die Funktionen müssen also vernünftig aufeinander abgestimmt sein. Manchmal ist es sinnvoll, daß eine der beiden Modifikationen eines Paares nichts tut. Zum Beispiel kann eine Modifikation aus f_1 die Halbierung eines numerischen Wertes bedeuten, um Anforderung und Altfall in die gleiche Größenordnung zu bringen. Dann ist es sinnvoll, wenn die dazu gehörige Modifikation aus f_2 nichts tut. Verschiedene Arten der Modifikation sind:

- **Vereinfachungen**: die genauen Werte sind häufig für die Lösungsstruktur unerheblich. Deswegen können die Werte durch Variablen ersetzt werden. Im Beispiel ist es unwichtig, ob das angeschossene Schiff zwei oder drei Felder lang ist (im Endeffekt besteht die Lösung darin, auf das nächste Feld in der Reihe zu schießen).
- **Geometrische Transformationen**: ebenso ist es für den Vergleich von zwei Fällen häufig nicht wichtig, welche Richtung vorliegt. Im Beispiel ist es unerheblich, ob die Richtung des angeschossenen Schiffes unten oder rechts ist. Neben einfachen Drehungen kommen je nach Art der Domäne Spiegelungen, Streckungen und Verzerrungen in Frage.
- **Vergleich von Abstraktionen**: damit ist die Einordnung des Problems in Strukturen gemeint, wie im Beispiel ja schon die Einordnung Reihe. Andere Einordnungen wären Kreis, Quadrat (also geometrische Strukturen), genauso aber die Einordnung nach der Größe oder Beschaffenheit einer maximalen Clique bei einem Graphen. Je mehr Einordnungen gleich sind, desto ähnlicher sind sich die Fälle. Auch hier können je nach Domäne sehr unterschiedliche Modifikationen auftreten: es sind alle vorstellbaren Arten von Strukturen möglich, wie z.B. die Struktur Reihenfolge nach Alter bei Menschen. Teilweise ist es sehr schwierig, diese Strukturen zu erkennen.

Für das genannte Beispiel wird also die Länge des angeschossenen Schiffes durch eine Variable ersetzt. Außerdem wird das angeschossene Schiff geometrisch transformiert: *nach rechts* wird durch *nach unten* ersetzt. Wenn jetzt ein

[8] Jeder der beiden Spieler verteilt zu Anfang *Schiffe* auf einem 10×10 Felder umfassenden Spielfeld, auf das nur er Einsicht hat. Dem Gegenspieler bleiben diese Stellungen also verborgen. *Schiffe* sind senkrechte oder waagerechte Reihen bestimmter Längen. In der bekanntesten Variante muß jeder Spieler je ein Schiff der Länge drei, vier und fünf und zwei Schiffe der Länge zwei unterbringen. Dabei ist es egal, ob Schiffe waagerecht oder senkrecht gesetzt werden, diagonal ist allerdings nicht erlaubt. Außerdem dürfen sich die Schiffe nicht kreuzen. In jeder Spielrunde darf jeder Spieler einen Schuß auf das gegnerische Gebiet abgeben, indem er die entsprechende Koordinate nennt. Der andere muß ihm daraufhin mitteilen, ob der Schuss ein Treffer war, und bei einem Treffer zusätzlich, ob ein Schiff versenkt wurde. Ein Schiff gilt als versenkt, wenn sämtliche Felder dieses Schiffes getroffen wurden. Gewonnen hat derjenige Spieler, der als erster sämtliche Gegnerschiffe versenkt hat.

alter Fall vorliegt, bei dem ein angeschossenes Schiff drei Felder lang nach unten liegt (auch hier wird die drei durch eine Variable ersetzt), ist dieser strukturell ähnlich (beide liegen nach unten, beide sind x Felder lang). Zu beachten ist dabei, daß die Modifikationen hintereinander gemacht werden, nach jeder Modifikation wird verglichen. Da dieses bei einigen Fällen gleichzeitig geschieht, ist der Laufzeitaufwand nicht unerheblich.

Nachdem die Lösung gefunden wurde, müssen die auf den Anforderungen gemachten Modifikationen für die Lösung rückgängig gemacht werden. So muß im Beispiel *nach unten* wieder durch *nach rechts* ersetzt werden. Dazu dient die Umkehrfunktion f_2^{-1}, in der für jede der Modifikationen aus f_2 eine umgekehrte Modifikation liegen muß. Die zunächst gefundene Lösung im Beispiel wird sein: *schieße auf das 1. Feld unter (oder über) der Reihe*. Also heißt die Lösung nach Anwendung der Umkehrfunktion *schieße auf das 1. Feld rechts von der Reihe*.

Abschließend wird die Funktion Φ^{-1} verwendet, die aus der abstrakten Lösung die Endlösung macht. Im Beispiel wird die Lösung in attribut-basierte Form gebracht. Sie besteht dann einfach aus der Koordinate $(5, 7)$.

4.4.2 Bewertung und Bemerkungen

Auch wenn das Beispiel vergleichsweise einfach war, sollte es nicht über die Komplexität dieses Vorgehens täuschen. Zunächst braucht man ein (wenngleich einfaches) Ähnlichkeitsmaß, um die Vorauswahl zu treffen. Jedoch sollte man bedenken, daß: die weiteren Schritte sehr aufwendig sind, jeder vorher entfernte Fall erleichtert und beschleunigt die Arbeit.

Die Funktionen Φ und f sowie ihre Umkehrfunktionen können nur mit Hilfe von Experten gefunden werden. Sie sind von diesen häufig nicht einfach zu finden, und meistens ergänzungsbedürftig. In [50] wird ausdrücklich darauf hingewiesen, daß bei der Implementation des Prototyps SynTerm/SynGraph im FABEL-Projekt an diesen Punkten (Regelergänzung; Vervollständigungen der Strukturerkennung; vor allen Dingen die Suche nach generellen, domänenübergreifenden Techniken) in Zukunft noch stark gearbeitet werden muß. Beachtet werden sollte auch die Komplexität der Suche nach einem ähnlichen Fall: im Normalfall wird nach jeder Modifikation ein Vergleich zwischen Problem und sämtlichen Fällen, die die anfängliche Ähnlichkeitsüberprüfung überstehen, durchgeführt. Die Laufzeitkosten sind hier immens. Trotzdem ist das Einbringen der strukturellen Ähnlichkeit ein vielversprechender Ansatz, da hier einerseits über den Rand der numerischen Werte hinweggeschaut wird, andererseits die in vielen Systemen eher strengen Grenzen zwischen Fallsuche, Vergleich, Adaptation, Evaluation und Reparatur beiseite fallen, und ineinander integriert werden.

4.5 Case-Based Design mit hierarchischer Repräsentation und Rahmentransformationstechnik

In diesem Kapitel wird hauptsächlich gezeigt, wie man die Rahmentransformationstechnik verwendet. Das heißt: zunächst wird ein geeigneter Fall gesucht, aus dem die generelle Struktur der Lösung entnommen wird. Dann werden die einzelnen Werte mit Hilfe anderer Fälle gesucht. Die zu zeigende Vorgehensweise arbeitet außerdem mit einer hierarchischen Repräsentation. Das in dieser Art programmierte *CaseTool* wurde Anfang der 90er Jahre als Teil einer größeren wissensbasierten System-Entwicklungsumgebung namens *DEKBASE*[9] implementiert. Interessant ist die Tatsache, daß in diesem Programm auch die Bewertung des Ergebnisses durch Experten einfließt.

4.5.1 Vorgehensweise mit Beispiel

Als Beispiel soll hier der Entwurf (Design) eines *idealen* Fahrzeugs dienen: der Benutzer gibt einige Daten ein, und erhält ein (relativ) perfektes Fahrzeug entsprechend seinen Anforderungen (Bedingungen). Das Problem soll die folgenden Anforderungen haben[10]:

[9] DEKBASE wird beschrieben in [57].

[10] andere Möglichkeiten für den Antrieb wären Eigenkraft mit Hilfsmotor, oder Motor

Antrieb:	Eigenkraft
Sitzzahl:	1
Größe des längsten Mitfahrers:	1.87 m
ungefähre Höchstgeschwindigkeit:	60 km/h
Höchstgewicht des Gepäcks:	20 kg

Fallsuche

Die Fallsuche geschieht in CaseTool dreigeteilt. Der erste Teil der Fallsuche, eine Vorauswahl, verläuft Index-basiert. Im Beispiel bietet sich an, die Antriebsart und die Sitzzahl als Index zu benutzen. Gewählt wird die Gruppe 1E (1 Sitz, Eigenantrieb) [11]. Nur die Fälle aus dieser Gruppe sind für die Lösungsfindung interessant. Die folgenden Fälle sollen unter diesem Index zu finden sein:

Nr.	Größe	Höchstgeschwindigkeit	Höchstlast	Güte
1	1.89 m	58 km/h	22 kg	81
2	1.65 m	39 km/h	15 kg	84
3	1.86 m	62 km/h	15 kg	74

Abbildung 4.5: Index 1E

Die Güte ist dabei der von Experten bestimmte Wert, der die Qualität des Produktes darstellen soll. Gegeben ist dieser Wert als Zahl zwischen 0 und 100 (100 bedeutet optimale Güte). In CaseTool sind auch die Unterlösungen mit einer Güte versehen.

Im nächsten Abschnitt der Fallsuche werden die Fälle weiter eingeschränkt, indem nur noch die Fälle betrachtet werden, bei denen sämtliche Werte in der Nähe der äquivalenten Werte der Anforderung liegen. Dazu werden die Werte in Gruppen eingeteilt, z.B. die Größe in:

Gruppe	Einteilung
1	1.50 m bis 1.65 m
2	1.65 m bis 1.75 m
3	1.75 m bis 1.85 m
4	1.85 m bis 1.95 m
5	1.95 m bis 2.10 m

Die Größe 1.87 m bedeutet also eine Einordnung in die Klasse 4. Eine Größe von 1.55 m würde demnach eine Einordnung in Gruppe 1 bedeuten, und eine Suchreichweite von 1.50 m bis 1.65 m. Aus der Beschreibung des Problems fallen Sitzzahl und Antrieb für die Betrachtungen weg, da diese schon als Index genommen wurden. Außerdem wird das Gewicht herausgenommen, das nur für Teillösungen relevant sein soll. Somit werden nur folgende Grenzen für das Suchen gesetzt:

Größe des längsten Mitfahrers	1.85 m bis 1.95 m
ungefähre Höchstgeschwindigkeit	55 km/h bis 65 km/h

Damit fällt Fall 2 weg. In CaseTool werden diese Suchkriterien durch Heuristiken festgelegt, die im sogenannten *Search Condition Generator* eingebunden sind.

Als letztes folgt die tatsächliche Ähnlichkeitsbestimmung. Dafür wird eine Ähnlichkeitsfunktion mit unterschiedlicher Gewichtung der relevanten Werte gewählt (Gewichtung 3.20 für Größe und 1.22 für Geschwindigkeit). Die Ähnlichkeit, hier auch Relevanz[12] genannt, wird wie folgt berechnet:

$$RN = \frac{1}{n} * \sqrt{\sum_{i=1}^{n} w_i * (\frac{v_i - a_i}{m_i})^2}$$

[11] Es ist denkbar (allerdings nicht in CaseTool), eine der beiden für den Index wichtigen Angaben wegzulassen, oder mehrere Möglichkeiten offenzulassen, z.B. 1 bis 2 Sitze zuzulassen. Dann könnte man alle in Frage kommenden Indizes betrachten.

[12] Englisch: Relevancy. Dieser Ausdruck wurde direkt aus [57] übernommen. Ein niedriger Relevanzwert bedeutet einen relevanteren Fall. Sprachlich wäre es deswegen sicherlich besser, von der Irrelevanz zu sprechen.

Dabei ist n die Anzahl der untersuchten Attribute (im Beispiel also $n = 2$), w_i die Gewichtung des Attributes i, v_i der Wert des Attributes i beim Problem, a_i der entsprechende Wert beim alten Fall, und m_i die Höchstgrenze, die das jeweilige Attribut erreichen darf.

Neben den Relevanzen wird auch noch die maximale Relevanz bestimmt. Dabei wird die gleiche Formel benutzt, allerdings steht im Zähler jeweils die größere Differenz vom Wert der Anforderung zu einer der beiden Grenzen. Im Beispiel ergibt sich folgende maximale Relevanz:

$$RN_{max} = \frac{1}{2} * \sqrt{3.20 * (\frac{1.95 - 1.87}{1.95})^2 + 1.22 * (\frac{55 - 60}{65})^2} = 0.0622$$

Die Fälle werden jetzt eingeordnet: sämtliche Fälle mit einer Relevanz zwischen 0 und $\frac{1}{3} * RN_{max} = 0.0207$ werden als *perfekt* eingestuft, alle Fälle zwischen 0.0207 und $RN_{max} = 0.0622$ als *nahe*.[13]

Die Fälle und ihre Einordnung:

Name	Relevanz	Relevanzklasse	Güte	Güteklasse
Fall 1	0.0340	nahe	81	sehr gut
Fall 3	0.0206	perfekt	74	gut

Die Güte wurde hier auch schon linguistisch ausgedrückt ($0 - 30$ schlecht, $30 - 50$ durchschnittlich, $50 - 75$ gut, $75 - 100$ sehr gut). Denkbar für ähnliche Programme ist der Wegfall des mittleren Teils. Dann würden im letzten Abschnitt sämtliche Fälle, deren Relevanz über der maximalen Relevanz liegen, herausgesiebt.[14] Daraus erwächst der Nachteil, daß man für mehr (unter Umständen sehr viele) Fälle die Ähnlichkeit berechnen muß. Bei einer komplizierten Ähnlichkeitsfunktion ist diese Vorgehensweise weitaus weniger effektiv als der bloße Vergleich mit den Schranken. Außerdem würden einige durch die Schrankenmethode entfernte Fälle eine ausreichende Relevanz aufweisen, so daß die Anzahl der in der Adaptation zu betrachtenden Fälle ansteigt. Eine weitere Möglichkeit besteht in der Festlegung variabler Grenzen (durch Bestimmungsregeln wie 5 *cm vom Ausgangswert entfernt* oder *bis zu 1.2 facher Ausgangswert*).

Adaptation

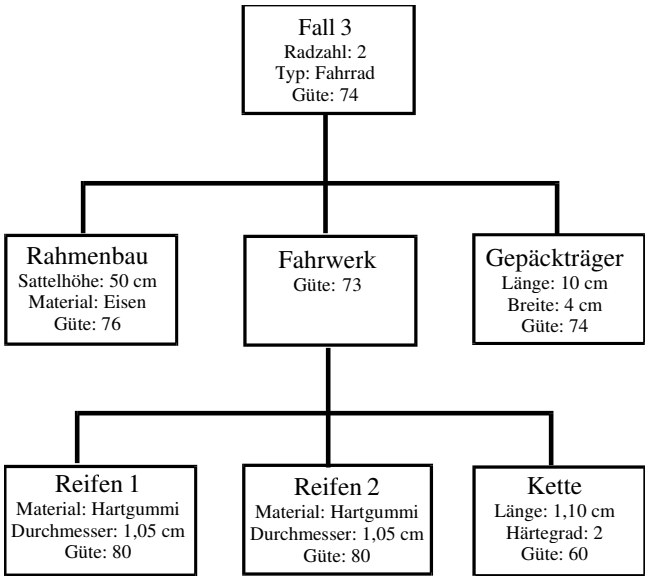


Abbildung 4.6: Darstellung des Falls 3

[13] Es wird in [57] nicht erläutert, warum gerade 1/3 als Grenze genommen wird. Vermutlich hat sich dieser Wert in Versuchen als der günstigste erwiesen.

[14] In der dreigeteilten Variante ist es unmöglich, daß ein nach dem zweiten Teil nicht verworfener Fall über der maximalen Relevanz liegt!

Für die Adaptation ist es wichtig, wie die Fälle abgelegt sind. Die Lösung wird durch einen Suchbaum gefunden, der von den alten Lösungen abhängt. Dabei kann es auch zu Abhängigkeiten zwischen den Teillösungen kommen (so hängt die Länge des Gepäckträgers vom Durchmesser der Reifen ab). Ein Netzwerk aus Abhängigkeiten und Backtracking-Techniken wird benötigt, um die Lösung konsistent zu halten. Für das vorliegende Problem ist der Suchbaum vergleichsweise einfach. Als Beispiel ist die Darstellung des Falls 3 in der Abbildung 4.6 zu sehen, aus der sich der Suchbaum für die gegebene Anforderung ableiten läßt. Für Autos würde dieser Baum komplexer sein: der Antrieb müßte als weiteres Unterproblem gefunden werden, die Reifenzahl steigt an, etc. In der Abbildung sind die Werte für den Fall 3 zu sehen, die nicht direkt in die Lösung übernommen werden.

Für jede Komponente werden zunächst die Auswahlmöglichkeiten beschränkt durch die Festlegung eines Suchraums für die Komponenten-relevanten Attribute, die noch nicht in der Fallsuche benutzt wurden. Dabei geht man ähnlich wie im 2. Teil der Fallsuche vor. So wird im vorliegenden Beispiel, wenn es um die Komponente Gepäckträger geht, das Gewicht betrachtet. Wenn das Gewicht der Anforderung zu dem Suchraum $18 - 22\,\mathrm{kg}$ führt, darf man für diese Komponente nicht den andernfalls passenden Fall 3 nehmen. Die Reihenfolge, in welcher die noch verbleibenden Fälle für diese Komponente ausprobiert werden, wird nacheinander durch folgende drei Werte festgelegt:

1. Relevanzklasse
2. Güteklasse
3. Güte der einzelnen Komponente.

Wenn also Relevanzklasse und Güteklasse gleich sind, entscheidet die erwähnte Güte der Komponente. Im Beispiel landet allerdings Fall 3 immer vor Fall 1.

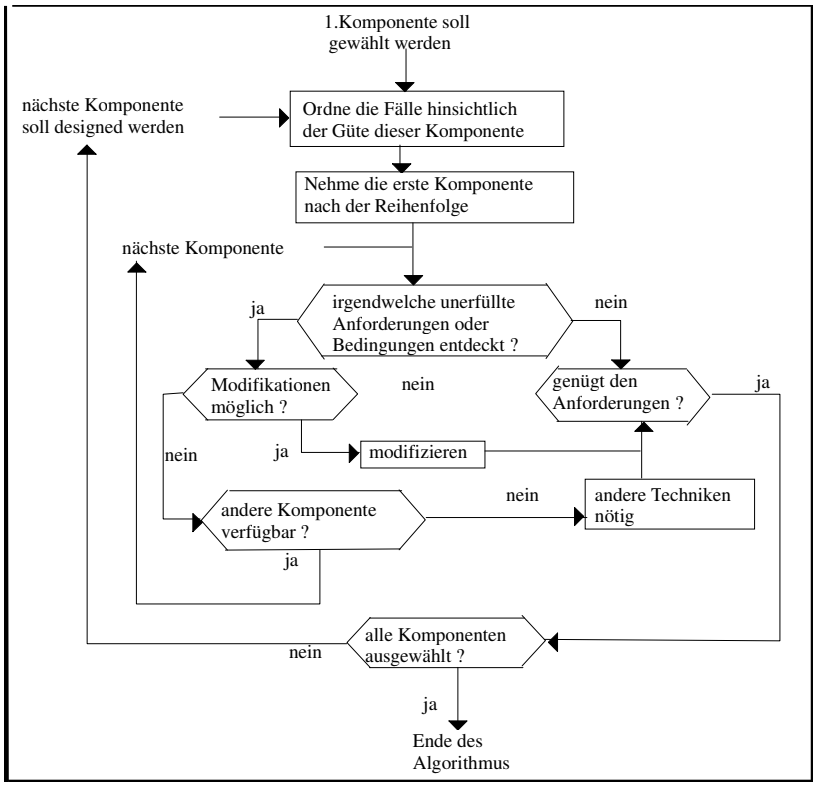


Abbildung 4.7: Vorgehensweise im Adaptationsbereich nach Kumar und Krishnamoorthy

Die weitere Vorgehensweise ist in der Abbildung 4.7 verdeutlicht. Die Komponenten werden der festgelegten Reihenfolge (also dem Suchbaum) folgend ausprobiert. Sollten irgendwelche Anforderungen durch diese Komponente nicht erfüllt sein, wird sie mit Hilfe von Modifikations-Regeln verändert. Ist es nicht möglich, mit Hilfe dieser Regeln die Komponente nutzbar zu machen, oder werden nach der Modifikation Verletzungen von Regeln oder Anforderungen erkannt, wird die nächste Möglichkeit ausprobiert. Wenn keiner der Fälle für die Komponente eine Lösung bringt,

müssen andere Mechanismen greifen, also zum Beispiel regelbasierte Methoden oder die Einsetzung einer vorher festgelegten Standardkomponente; im ungünstigsten Fall kann für die definierte Anforderung keine Lösung gefunden werden.

4.5.2 Beispiel

Bei der Betrachtung der Komponente *Reifen 1* spricht nichts gegen die Wahl der Komponente aus dem Fall 3. Anders sieht es vielleicht aus, wenn der Fall 3 nicht existieren würde, und statt dessen diese Komponente aus Fall 1 genommen wird. Angenommen, der Reifen wäre aus Weichgummi mit Durchmesser 1 m, und eine Bedingung läge vor in Form der Regel *Reifen aus Weichgummi mit einem Durchmesser unter* 1.05 *m führen dazu, daß die Höchstgeschwindigkeit höchstens* 59 *km/h ist*. Die Kombination aus Anforderung und Teillösung verletzt dann diese Bedingung. Es gibt drei mögliche Schritte:

1. Eine Modifikation bringt die Lösung: im Beispiel kann vielleicht der Durchmesser erhöht werden, um die Geschwindigkeit zu erhöhen.
2. Die Komponente wird verworfen, die entsprechende Komponente des nächsten möglichen Falls wird ausgesucht. Leider gibt es im Beispiel keinen weiteren Fall.
3. Die Komponente muß auf andere Art ermittelt werden.

Glücklicherweise gibt es Fall 3, so daß die Reifen eine Größe von 1.05 m erreichen. Wie man sieht, sind auch bei dieser Art des Case-Based Design verschiedene Regeln notwendig. Einerseits müssen bestimmte Bedingungen in die Lösungsfindung einfließen (wie im gerade erwähnten Beispiel: *WENN Art = Weichgummi UND Durchmesser* < 1.05 *m DANN Geschwindigkeit* < 60 *km/h*). Andererseits benötigt man Modifikationsregeln. Außerdem müssen noch die weiteren Techniken zur Lösungsfindung zur Verfügung gestellt werden.

Eine Komplikation soll sich beim Rahmenbau ergeben: hier wird der Rahmen des Falls 3 genommen. Es soll jetzt folgende Bedingung gelten: *die Größe des Menschen muß im Bereich von Sattelhöhe + Durchmesser +* 27 *cm und Sattelhöhe + Durchmesser +* 31 *cm liegen*. Bei der vorliegenden Sattelhöhe $(50\,\text{cm})$ und dem gefundenen Reifendurchmesser $(1.05\,\text{m})$ ergibt dieses eine Spanne von 1.82 m und 1.86 m. Damit ist diese Bedingung nicht erfüllt. Mit Hilfe einer Modifikationsregel kann vielleicht die Sattelhöhe auf 51 cm erhöht werden. Aber: das wiederum kann Auswirkungen haben auf das Material der Reifen, wenn z.B. eine Sattelhöhe von 51 cm für Hartgummireifen aus Sicherheitsgründen nicht möglich ist. Wie man also sieht, muß dann wiederum die Komponente Reifen modifiziert werden. Die Lösung für die Komponenten sind also nicht voneinander unabhängig.

4.5.3 Bewertung der Lösung und Speicherung

Wenn die Lösung endlich gefunden wurde, wird sie hinsichtlich der Güte klassifiziert. Bei *CaseTool* wird dabei nicht ein Experte hinzugezogen, vielmehr werden spezielle Bewertungsregeln auf den Fall angewendet. Abgespeichert wird ein Fall, wenn folgende Bedingungen erfüllt werden:

- Die Güte liegt über der Durchschnittsgüte der perfekten Fälle
- Auf mehr als 25% der Komponenten mußten Modifikationen angewendet werden.
- Über 75% der herausgesuchten Fälle waren nahe.

Damit würde die Beispielslösung nicht gespeichert werden (da 50% der Fälle perfekt sind).

4.5.4 Bewertung und Bemerkungen

Die hierarchische Darstellung ist sehr gut geeignet für Probleme, die sich in viele kleine Probleme teilen lassen. So kann man z.B. ein Gebäude in Stockwerke unterteilen, die Stockwerke in Räume, etc. Man löst die kleinen Probleme, und damit löst man sukzessive auch die größeren Probleme, bis schließlich das eigentliche Problem gelöst ist. Man darf allerdings nicht vergessen, daß das Zusammenwirken einzelner ausgewählter Komponenten unmöglich sein kann.

Dazu ist ein guter Evaluationsalgorithmus notwendig, der das als Lösung oder Teillösung vorgesehene Konstrukt überprüft.

Der Einfluß der Güte ist ein äußerst interessanter Ansatz. Hier wird sie durch Regeln ermittelt – je schlechter und ungenauer die Bewertungsregeln, desto schlechter natürlich der Einfluß der Güte. Viel besser wäre natürlich die Bewertung durch einen oder sogar mehrere Experten, nachdem das Objekt tatsächlich konstruiert wurde (und vielleicht erst einige Zeit später, so daß Mängel zu Tage kommen). Es sollte bei *CaseTool*-ähnlichen Systemen zumindest möglich sein, daß die durch das Programm ermittelte Güte durch einen Experten verändert werden kann, so daß die Praxiserfahrungen tatsächlich einfließen.

4.6 Weitere Aspekte von Case-Based Design

Nachdem oben drei Ansätze für Case-Based Design dargestellt wurden, sollen hier noch zwei Aspekte genauer betrachtet werden: einerseits die Durchführung der Adaptation, andererseits die Verwendung von *Gestalts* als Indizes bei der Fallsuche.

4.6.1 Adaptation

Ein eher domänenspezifisches Problem ist das Finden von Adaptationsregeln. In diesem Punkt ähneln Case-Based Design-Systeme herkömmlichen Implementierungen mit Hilfe von Regeln oder Heuristiken. Wie in Kapitel 4.2.1 erwähnt, wird die Adaptation häufig durch Regeln – Heurisman – oder Constraint-Mechanismen, die in das Programm integriert werden, verwirklicht. Beide Methoden bedeuten die Einbeziehung eines Bereichsexperten, der dieses Wissen in für Computer verarbeitbare Form bringen muß. Außerdem kann dieses Adaptationswissen nur schwerlich übertragen werden auf andere Domänen. Das in Kapitel 4.3 vorgestellte evolutionäre Case-Based Design beruht auf einer domänenunabhängigen Adaptation. Dafür muß allerdings der Evaluationsteil gut genug sein, um untaugliche Fälle zu entfernen.

Wichtig ist die Einbeziehung des Anwenders, vor allem eines Experten, in die Adaptation, weil dieser mit der Domäne gut vertraut ist – wie z. B. der Architekt bei einem Bauplanungs-Programm. Dieser kann oftmals sagen, ob noch Anforderungen unerfüllt sind. Außerdem ist es ihm vielleicht möglich, Ideen für die Lösung zu liefern. Allerdings wünscht sich der Anwender häufig eine schnelle und vor allem automatische Lösung. Außerdem wird es bei steigender Größe und Komplexität für den Anwender immer schwieriger, eine Systemtransparenz zu behalten.

Ein sinnvoller Ansatz ist das *doppelte Case-Based Reasoning* oder *Adaptation durch Case-Based Reasoning*, vorgeschlagen von Kathleen Hanney und Mark T. Keane [53]. Hierbei werden die Adaptationsregeln aus der Fallbasis selbst herausgezogen. Fälle in der Fallbasis werden paarweise darauf untersucht, wie sich ihre Anforderungen und Lösungen voneinander unterscheiden. Daraus werden einfache Adaptationsregeln abgeleitet. Man merkt sich in der Regel

- die in den beiden Fällen identischen Parameter;
- die in den beiden Fällen unterschiedlichen Parameter;
- die Werte der sich unterscheidenden Parameter.

Beispielsweise könnten die beiden Anforderungen einfach Vierecke sein: das eine ist $2\,\mathrm{m}$ lang, und $3\,\mathrm{m}$ breit, das andere ist ein Quadrat mit $2\,\mathrm{m}$ Seitenlänge. Für beide soll ermittelt werden, wie groß die Fläche ist. Für die erste Anforderung ist die Lösung $6\,\mathrm{m}^2$, für die zweite $4\,\mathrm{m}^2$. Man merkt sich: die Länge ist gleich; Höhe und Fläche sind unterschiedlich; die Breite ist $3\,\mathrm{m}$ bzw. $2\,\mathrm{m}$, die Flächen sind $6\,\mathrm{m}^2$ und $4\,\mathrm{m}^2$.

Es ist klar, daß man sehr schnell auf eine sehr große Anzahl von Regeln kommt, die dazu nicht gerade spezifisch sind. Deswegen versucht man, die Regeln zu generalisieren. Wenn zwei weitere Fälle in jeder Richtung doppelt so groß sind wie die ersten beiden ($4 * 6\,\mathrm{m}^2$ bzw. $4 * 4\,\mathrm{m}^2$ mit den Flächen $24\,\mathrm{m}^2$ und $16\,\mathrm{m}^2$), läßt sich daraus eine ähnliche Regel wie die obrige ableiten. Die beiden Regeln faßt man dann zusammen. Die zusammengefaßte Regel kann dann so aussehen: *WENN die Breite von Fall A größer sind als die von Fall B, DANN ist die Fläche für Fall B zu errechnen, indem man die Fläche von A multipliziert mit dem Quotienten Breiten von B und A.*

Man muß natürlich versuchen, die Anzahl der Regeln möglichst klein zu halten, d.h.: möglichst viele Regeln vernünftig zusammenzufassen, so daß wenige ungeneralisierte Regeln übrig bleiben. Außerdem sollten wenig eingesetzte Regeln

verschwinden. Dabei muß man allerdings darauf achten, daß eine wenig angewandte Regel, die allerdings die einzige für einen bestimmten Kontext ist, doch sehr sinnvoll sein kann.

Adaptation ist ein Bereich des Case-Based Design und allgemein des Case-Based Reasoning, indem noch Forschungsarbeit geleistet werden kann und muß. Wichtig ist dabei die Suche nach domänenunabhängigen Adaptationsmethoden, da ansonsten einige Nachteile der regelbasierten Methoden zutrage kommen (vor allen Dingen die Tatsache der Domänenabhängig des Programms).

4.6.2 *Gestalts* als Indizes

In der Architektur wird häufig mit CAD-Plänen (Grundrißzeichnungen) von Gebäuden gearbeitet. In diesen Plänen findet man eine Vielzahl von Teilen und Strukturen. Primitive Ansätze zählen die Anzahl der Elemente und nutzen diese Informationen (z.B. Anzahl der Fenster, der Türen, der Aufzüge, etc.). Diese Ansätze sind einfach, schnell und wenig präzise. Andere Ansätze, die sich auf die Topologie beziehen, sind zu langsam für große Fallbasen.

Der in Schaaf [66] vorgestellte Ansatz beschäftigt sich mit den Strukturen, die den Architekten wirklich an den konkreten Fall erinnern, und somit gut als Indizes dienen können. Dabei handelt es sich weniger um konkrete Strukturen, sondern eher um Gedankenmodelle im Kopf des Architekten, sogenannte *Gestalts*. In Bild 4.8 sieht man Beispiele dafür.

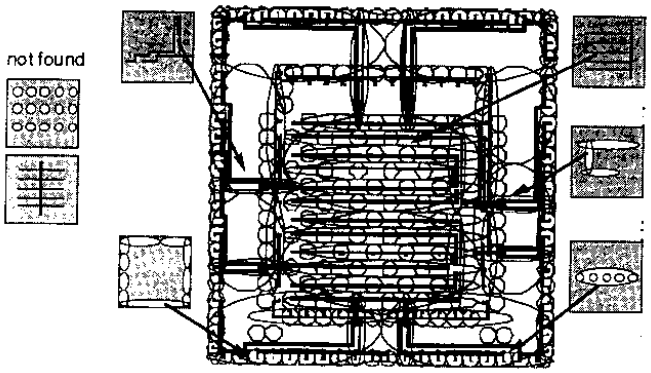


Abbildung 4.8: Beispiel für das Herauslesen von *Gestalts* nach Schaaf (1994)

Ein Algorithmus, der nacheinander für jede *Gestalt* schaut, ob und wo diese im Plan zu finden ist, braucht sehr lange (schlimmstenfalls das Produkt aus der Anzahl der möglichen *Gestalts* der Anzahl der möglichen Erscheinungsformen der *Gestalt*, und der Anzahl der Objekte). Deswegen wird eine abstrakte Darstellungsform für jede Objektgruppe im Plan und jede *Gestalt* gewählt, die einen effizienten Vergleich erlaubt.

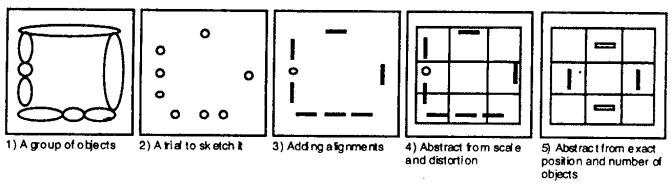


Abbildung 4.9: Von der Objektgruppe zur Skizze; abstrakte Darstellung nach Schaaf (1994)

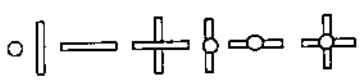


Abbildung 4.10: Das Skizzen-Alphabet (Schaaf 1994)

Abbildung 4.9 zeigt, wie aus einer Objektgruppe eine solche Abstraktion entwickelt wird. Im ersten Bild ist die Objektgruppe zu sehen, im zweiten sind die Objekte, im dritten die Ausrichtung dieser Objekte angedeutet. Danach wird versucht, den Maßstab und die Verzerrung herauszunehmen, und die Objektgruppe auf einer 3×3-Matrix darzustellen. Im letzten Bild wird die exakte Position und die Anzahl der Objekte aus der Abstraktion entfernt. Damit soll eine den Architekten nachahmende *Gestalt*suche möglich sein. Um ein Rechteck zu entdecken, wenn man ein Beispiel dafür vor Augen hat, muß der Mensch vor allem die Größe, Verzerrung und Drehung vernachlässigen (die Drehung wird in der vorliegenden Implementation jedoch nicht berücksichtigt).

Die *Gestalt* liegt jetzt also in abstrakter Form vor. In Abbildung 4.10 erkennt man das im Beispiel benutzte Alphabet der Elemente, die in diesen abstrakten Skizzen vorkommen können. In ähnlicher Art und Weise werden auch die mit Hilfe von Suchalgorithmen gefundenen Objektgruppen des eingelesenen Planes abstrahiert. Danach werden diese mit den *Gestalts* verglichen. Somit können die Fälle mit Hilfe von *Gestalts* indiziert werden.

Ein kleines Problem ist noch zu erwähnen: wenn zwei *Gestalts* die gleiche Repräsentation haben, kommt es zu einer *Kollision*. Dieses kann u.U. verhindert werden, in dem man

- von einer 3×3- zu einer 5×5-Matrix übergeht,
- die Regeln verändert, mit deren Hilfe Objektgruppen im Plan zusammengefaßt werden, oder
- das Alphabet erweitert.

Allerdings sollte man dabei bedenken, daß eine Objektgruppe auch einen Menschen an zwei verschiedene *Gestalts* erinnern kann.

4.7 Vorteile, Nachteile und Unterschiede

4.7.1 Vergleich mit anderen wissensbasierten Systemen

Case-Based Design kommt der Vorgehensweise eines menschlichen Experten entgegen:

> „Human experts are not systems of rules, they are libraries of experiences.“[15]

Die reine Umsetzung des Wissens mit Heuristiken oder Constraints ist für Experten häufig schwierig. Einfacher hingegen ist die Bereitstellung von alten Fällen. Allerdings gibt es dabei eine Einschränkung: bei vielen CBD-Ansätzen ist die Umsetzung von Expertenwissen nicht unerheblich, und drückt sich in einem Bedarf an Adaptions- und weiteren Regeln oder etwa weiteren Constraints aus. Das gezeigte Vorgehen ist auch besser geeignet für Domänen, die nicht vollständig verstanden sind, und bei denen die Modellbildung große Probleme bereitet (diffuses Gebiet). Die Regeln oder definitive Bedingungen sind weitaus schwieriger zu finden als Fälle. Außerdem gibt es Anwendungsbereiche, in denen Ausnahmen von Regeln in großer Anzahl vorliegen können.

Die Voraussetzung für Case-Based Design ist das Vorhandensein einer hinreichend großen Fallbasis aus Vergangenheitsbeispielen. Ein großer Vorteil ist diesbezüglich die Tatsache, daß das System durch neue Fälle lernt, daß neue Fälle weitaus besser eingefügt werden. Es ist schwieriger, neue Regeln und Bedingungen einzubauen. Durch das Einfügen neuer Fälle in die Fallbasis wird das System außerdem automatisch verbessert (dabei erweist es sich als günstig, wenn die Qualität der Lösungen berücksichtigt wird). Auch können leichter Erfahrungen verschiedener Experten eingebaut werden, denn bei der Umsetzung ihres Wissens in Regeln oder Bedingungen können leicht Unterschiede oder gar Gegensätze bedingt durch die Erfahrungen der Experten auftreten. Beim Case-Based Design können beide Expertenmeinungen als Fälle eingebaut werden. Insgesamt sind regel- oder constraint-basierte Systeme größtenteils weniger flexibel und lernfähig. Allerdings gibt es auch Schwierigkeiten, die teilweise in der Tatsache begründet liegen, daß Case-Based Reasoning erst seit relativ kurzer Zeit Forschungsgebiet ist. Hauptprobleme liegen in der Schaffung eines Ähnlichkeitsmaßes und in der Adaptation. Bei beiden Problemen wären selbst lernende und sich ändernde Maße und Algorithmen sinnvoll, auch hinsichtlich der von vielen erhofften relativen Domänenunabhängigkeit eines Case-Based Design-Systems. Man sollte natürlich nicht vergessen, daß wie so häufig das Beharren auf nur einem Ansatz unklug sein kann. Es gibt keinen Grund dagegen, zum Beispiel ein System mit Regelbasiertem und Fallbasiertem Teil zu erstellen. Systeme mit verschiedenen Lösungselementen sind schon existent, beispielsweise *FABEL*. Außerdem fließen wie gezeigt Regeln oder *constraints* in den Adaptationsteil ein.

[15] Sinngemäß übersetzt: *Menschliche Experten sind keine Regelsysteme, sondern Bibliotheken von Erfahrungen. Siehe hierzu Riesbeck [64]*

4.7.2 Vergleich der vorgestellten Möglichkeiten

Es ist weder nötig noch sinnvoll, die vorgestellten Ansätze hinsichtlich ihrer Qualität einzuordnen: keiner ist *besser* als ein anderer – sondern einfach nur anders. Der genetische Ansatz ist sehr domänenunabhängig und zeichnet sich durch besondere Lernfähigkeit aus. Allerdings können komplizierte Strukturen nur schwierig in Chromosomen codiert werden.

Ein hierarchischer Ansatz eignet sich vor allen Dingen für zu entwerfende Objekte, die als Summe von Einzelteilen angesehen werden können, und bei denen die Struktur relativ fest ist. Denkbar ist im übrigen eine Kopplung mit einem genetischen Algorithmus, der für die Erstellung von Subkomponenten benutzt werden könnte.

Für komplexe Strukturen sind strukturbedingte Ähnlichkeitsmaße und Adaptation nötig. Hier braucht man Domänenwissen, um die Abstraktionen und vor allen Dingen die Regelbasis zu konstruieren.

Die vorgestellten Ansätze sind bei weitem nicht erschöpfend, doch beispielhaft für die sehr unterschiedliche Umsetzungsweise des Case-Based Design. Ein großes Ziel ist die Entwicklung eines Systems oder zumindest grundlegender Methoden, die es erlauben, in beliebigen Anwendungsbereichen zu arbeiten. Der evolutionäre Ansatz kommt dieser Idee in der Transformationsphase sehr entgegen, ist aber dennoch nicht überall geeignet. Man sollte nicht vergessen, daß die Anwendungsanforderungen doch äußerst unterschiedlich sein können, und daß die Probleme bei einer übergreifenden Implementierung auf eine einheitliche Problemstruktur abgebildet werden müssen. Je nach Art der Anwendung bieten sich eher verschiedene Methoden an, und zwar abhängig zum Beispiel davon, ob es sich eher um strukturelle Probleme handelt, oder um Probleme, bei denen numerische Werte den Konstruktionsfall durchaus ausreichend beschreiben. Im letzteren Fall sind einfachere Algorithmen angebracht, als sie bei komplexen Strukturen benötigt werden.

Auffällig ist ganz allgemein, daß bei vielen der vorliegenden Veröffentlichungen der jeweilige Autor darauf hinweist, daß die Forschung in diesem Bereich noch nicht so weit fortgeschritten ist, als daß Case-Based Design bei wirklich relevanten Problemen eingesetzt werden könnte. Gleichzeitig wurde aber die Auffassung geäußert, daß Case-Based Design viel für die Zukunft verspricht.

4.8 Resümee

Mit Case-Based Design wurde ein interessanter Ansatz für das Konstruieren und das Entwerfen vorgestellt, bei dem auf frühere Entwürfe zurückgegriffen wird. Der Leser sollte einen Überblick bekommen haben, wie ein Case-Based Design-System generell aufgebaut ist. Einige Ansätze zur Implementation wurden beschrieben. Eine gute Einführung in den aktuellen Forschungsstand vermittelt das Buch von Maher und Pu [74]; eine anwendungsbezogene Darstellung des Case-Based Reasoning beschreibt das Buch von Bergmann et al. [76].

Kapitel 5

Expertensysteme für die Anpassungskonstruktion (CAE)

5.1 Einführung

Ein unbewältigtes wissenschaftliches Problem in der Konstruktionstheorie ist die Einbeziehung von Komplexitätsmaßen bei der Wissensverarbeitung. In der Konstruktionswissenswissenschaft fordern Ingenieure hierzulande regelmäßig eine integrale Funktionsausnutzung[1] oder eine Funktionsintegration sowie eine Integralbauweise[2] im Sinne einer Kompaktkonstruktion [85] [100]. Hierüber lassen sich oftmals die Herstellkosten reduzieren [97]. Die integrale Funktionsausnutzung hat große Bedeutung für den ingenieurmäßigen Entwurf und unseren Lebensalltag, weil technische Neuheiten und Innovationen häufig auf einer integralen Funktionsausnutzung - Funktionsintegration - basieren [86] [110].

In der Forschungsdisziplin Künstliche Intelligenz strebt man an, Ingenieurwissen über entsprechende Fallwissensbasen für aktuelle oder neuartige Entwurfsprobleme verfügbar zu machen. So bestimmt man etwa für ein vorliegendes Problem zur Hydraulikkonstruktion denjenigen Fall, der von der Problemstruktur her am ähnlichsten ist, und überprüft dessen Tauglichkeit für das zu lösende Problem. Wenn die identifizierte Vergangenheitslösung nicht voll den Anforderungen entspricht, so wird der betreffende Fall überarbeitet. Diese Überarbeitung kann auch mit sogenannten Skalierungsregeln erfolgen, die eine neue Dimensionierung vornehmen [87]. In der Informatik spricht man in diesem Zusammenhang von Adaptation; in der Konstruktionswissenschaft von Varianten- und Anpassungskonstruktion. Das Charakteristikum der Anpassungskonstruktion ist ein bereits vorhandener Lösungsansatz (z.B. Konstruktionszeichnung), so daß nur Teilprobleme von Grund auf neu zu lösen sind [81].

5.2 Entwicklungsstand

Die meisten CAE-Systeme[3] für die mechanische Konstruktion im allgemeinen und für die Hydraulik im Besonderen ermitteln sogenannte Varianten [98]. Bei dieser Variantenkonstruktion steht das Lösungsprinzip fest, und daß Schwergewicht liegt auf der Gestaltung. Leistungsfähige Expertensysteme, die in der Hydraulik eine komplexe Anpassungskonstruktion ausführen, sind bisher nicht bekannt geworden[4].

Dies ist eigentlich auch nicht verwunderlich, weil bei der Entwicklung von CAE-Systemen in der Vergangenheit ei-

[1] Der Begriff integrale Funktionsausnutzung kommt aus dem Gerätebau und ist im Maschinenbau weniger geläufig. Frölich und Schlottmann (1982) sprechen diesbezüglich von einer Funktionsintegration; ebenso Roth (1994):„Die Erhöhung der Eigenschaften eines Einzelteils mit dem Ziel, mehrere Funktionen mit ihm erfüllen zu können, wird Funktionsintegration ... genannt.“ Siehe Seite 236.

[2] Koller (1998) unterscheidet eingehender zwischen der Integralbauweise und einer Multifunktionalbauweise: „Die Zusammenfassung mehrerer Bauteile gleicher oder unterschiedlicher Funktion(en) zu einem Bauteil mit der gleichen Zahl von Funktionen, welche den Einzelteilen gemeinsam waren, soll als Integriertes Bauteil bzw. als Integral-Bauweise bezeichnet werden.“ Siehe Seite 311.

„Bauteile so zu konstruieren, daß diese ohne nennenswerten Fertigungs- und Kostenaufwand weitere Funktionen erfüllen können (kostenlose Realisierung von Funktionen) soll als Multifunktionalbauweise bezeichnet werden.“ Siehe Seite 312.

[3] CAE bedeutet Computer Aided Engineering.

[4] Zur Kritik siehe Hoffmann [87], S. 92 ff.

ne algorithmisch-analytische Vorgehensweise betont worden ist. Diese stößt allerdings bei fluidischen Systemen[5] auf Grenzen [119]. Viele technischen Strömungsvorgänge lassen sich analytisch nur sehr schwer oder überhaupt nicht berechnen, weshalb man auf Ähnlichkeitsgesetze zurückgreift [102] [121]. Selbst bei der Untersuchung relativ einfach strukturierter hydrostatischer Antriebe müssen Ingenieure nichttriviale Annahmen machen und berechnete Werte durch Messungen verifizieren [84]. Darüber hinaus gibt es im Konstruktionsbereich Entwicklungsdefizite bezüglich der Kostenfrüherkennung und Vorkalkulation. Beispielsweise diskutiert Velten explizit ein Anforderungsmodell, ein Funktionsmodell, ein Prinzipmodell sowie ein Gestaltmodell – aber nicht in expliziter Weise ein *Kostenmodell*, obwohl die Fertigungs- und Montageplanung integriert wird [118].

Zum Stand der Rechnerunterstützung für die frühen Phasen des Konstruktionsprozesses – Aufgabenklärung, Konzipieren – siehe die Dissertationen von R. J. Huber (1994) sowie R. Kläger (1993) von der TH Karlsruhe.

Mit dem Paradigmenwechsel hin zum fallbasierten Entwurf sind neue Lösungsmöglichkeiten in Praxisnähe gerückt [88]. Es ist absehbar, daß zukünftige Expertensysteme für die Anpassungskonstruktion das Fachwissen von einem oder von mehreren Bereichsexperten – Spezialisten – primär fallbasiert verarbeiten werden. Während man in den 80er Jahren auch stark an Datenbankanwendungen (und Informationssystemen) für Wiederholteile arbeitete, konzentriert sich die jüngste Informatikforschung auf Konfigurationsprobleme und auf die Automatisierung der Anpassungskonstruktion. Von zunehmender Beceutung ist hierbei die Entwicklung von integrierten Produktmodellen [118].

5.3 Anpassungskonstruktion

Die Anpassungskonstruktion für fluidische Systeme ist Gegenstand reger Automatisierungsanstrengungen [119]. Bei seiner Analyse zur Problematik der Anpassungskonstruktion in fallbasierten Systemen für die Fluidtechnik kommt Hoffmann zu folgendem Schluß: „In der Praxis kann das Zusammenspiel verschiedener Unterfunktionen so komplex sein, daß sie sich nicht aus verschiedenen Fällen kombinieren lassen. In der Fluidtechnik tritt diese Situation auf, wenn zwei fluidische Achsen (Unterfunktionen) eine Reihe von Komponenten teilen. Auf der anderen Seite lassen sich komplexe Unterfunktionen auch nicht aus solchen Fällen extrahieren. In diesen Situationen muß ein Fall aus der Fallbasis gesucht werden, der alle geforderten Unterfunktionen enthält.“[6]

Für die rechnergestützte Anpassungskonstruktion ergeben sich somit im Prinzip drei Fallunterscheidungen:
(1) das CBR-System findet einen Fall, der alle geforderten Funktionen enthält;
(2) der ähnlichste Fall hat mehr Funktionen, als im Pflichtenheft verlangt;
(3) der gefundene Fall weist nicht alle verlangten Unterfunktionen auf.

Auch wenn das CBR-System einen interessanten Fall identifiziert, welcher alle eingeforderten Funktionen enthält, so kann ex ante nicht behauptet werden, daß dieser Konstruktionsfall mit Sicherheit als Entwurfsgrundlage für die angestrebte Systemlösung anzusehen ist. Es ist durchaus denkbar, daß der zuständige Konstrukteur diesen Konstruktionsfall, der der aktuellen Problemstellung insgesamt am ähnlichsten ist, doch zurückweist, weil beispielsweise die verlangte Präzision nicht gegeben ist [115]. Insofern ist einzubeziehen, daß die rechnergestützte Anpassungskonstruktion sich auch fruchtbar auf (zu modifizierende) Systemlösungen erstrecken kann und soll, die mehr oder weniger Funktionen als verlangt haben. Grundsätzlich muß ein Ähnlichkeitsmaß alle drei der oben erwähnten Kategorien erfassen, die maßgeblich für die fallbasierte Anpassungskonstruktion sind.

Die Informatikforschung hat gezeigt, daß es schneller gelingt, einen relevanten Fall zu überarbeiten und diesen an die vorliegenden Anforderungen anzupassen, als von Grund auf neu eine Einzellösung herzuleiten [77]. Wenn man dem an einem fluidischen System arbeitenden Konstrukteur keinen zur vorliegenden Problemstellung ähnlichen Fall heraussucht, so muß er unter anderem auch die komplexe Koppelung der hydraulischen Achsen vollständig neu auslegen. Wenn er einen praxisrelevanten Fall vorgelegt bekommt, so kann er technisch abschätzen, wie das Zielsystem in etwa arbeitet (Interpolation, Extrapolation), weil er Anhaltspunkte hat (z.B. für die Ventilsteuerung).

Eines der wissenschaftlichen Probleme, die in diesem Zusammenhang auftreten, ist die definitive Bewertung der Systemkomplexität.[7] Ist es in der Hydraulikkonstruktion aus heuristischer Sicht sinnvoller, Anpassungssysteme (z.B.

[5] Das die Fluiddynamik für Überraschungen sorgen kann, dies zeigt die Entdeckung eines neuartigen Materiezustandes zum Bose-Einstein-Kondensat durch den deutschen Physiker Wolfgang Ketterle (MIT, USA) und zwei US-Physiker, denen hierfür der Physik-Nobelpreis 2001 zugesprochen wurde [78].

[6] Hoffmann [87], S. 65 f.

[7] Ropohl [107] hat zwischen Kompliziertheit und Komplexität unterschieden: „Von *komplizierten* Systemen spricht man, wenn sie eine größere Anzahl verschiedenartiger Subsysteme enthalten; das Maß der Kompliziertheit, die *Varietät*, wird durch die absolute Zahl unterscheidbarer Subsysteme oder durch den dualen Logarithmus dieser Zahl angegeben. Demgegenüber zeichnen sich *komplexe* Systeme durch eine Vielzahl unterschiedlicher Relationen aus; analog zur Varietät gibt man die Komplexität durch die Zahl unterscheidbarer Relationen oder durch den dualen

Konfigurationsregeln, Skalierungsregeln) zu entwickeln, die eine 4-Achsen-Lösung an die verlangte 5-Achsen-Lösung anpassen (Scale-up), oder sollte ein Expertensystem besser eine gefundene 6-Achsen-Lösung (Scale-down) auswählen, um hieraus fallbasiert die gewünschte 5-Achsen-Lösung zu entwickeln? Bisher sind CAE-Systeme, die Komplexitätsmaße für die Anpassungskonstruktion von hydraulischen Systemen verarbeiten, in der Literatur nicht beschrieben worden, obschon das sogenannte Teilevielfaltsproblem (Standardisierung) in der Ingenieurwissenschaft recht bekannt ist [100].

Man mag darüber streiten, ob ein Scale-up oder ein Scale-down im Einzelfall besser ist. Dies wird in der Praxis auch von dem Bewährungsgrad einer Systemlösung abhängen – und von innovativem Ehrgeiz.

5.4 Strukturgraphen und integrale Funktionsausnutzung

Wenn man Strukturgraphen von alten und diesbezüglich weiterentwickelten Konstruktionen miteinander vergleicht, so kann man oftmals feststellen, daß die neue Systemlösung einen höheren Grad an integraler Funktionsausnutzung[8] aufweist. Die Einsparung[9] von Funktionsträgern bedeutet, daß der Strukturgraph der neuen (integrierten) Systemlösung weniger Kanten und auch weniger Knoten hat.

Eine interessante Analyse hierzu ist die klassische Arbeit von Kettner und Klingenschmitt [93], in welcher am Beispiel einer Umformmaschine gezeigt wird, daß die neue Konstruktion weniger Funktionsträger hat. Nach Kenntnis des Verfassers gibt es diesbezüglich leider keine empirischen Studien über fluidische Systeme.

Man kann am Beispiel von Rodenackers Kronenkorkenproblem[10] theoretisch analysieren, inwiefern es gelingen müßte, inkrementell eine höhere integrale Funktionsausnutzung zu entwickeln, indem man z. B. einzelne Systemelemente eliminiert, und für die somit sabotierte Effektträgerstruktur eine neue physikalisch-technische Systemlösung definiert. [11] Technische Expertensysteme, die Konstruktionsobjekte in Richtung auf eine stärkere integrale Funktionsausnutzung hin entwickeln können, sind bisher nicht bekannt geworden. Für das Problem der Integralbauweise sowie für die Funktionsintegration existiert hierzulande praktisch keine erwähnenswerte Rechnerunterstützung.

W. G. Rodenacker hat in seinem Buch zum methodischen Konstruieren ein Kronenkorkenproblem skizziert, für welches von Konstrukteuren keine Ideallösung mit nur einem integrierten Bauteil gefunden wurde; vielmehr dokumentierten diese Kurslösungen mit relativ komplexer Systemstruktur [105]. Aus theoretischer Sicht darf man allerdings annehmen, daß sich eine derartige suboptimale Systemlösung inkrementell durch Herausnehmen von einzelnen Funktionsträgern vereinfachen und transformieren läßt, so daß man schließlich die einelementige Ideallösung erhält. Zukünftige Expertensysteme sollten diesbezüglich eine offensichtlich existente menschliche Leistungslücke schließen und hinsichtlich einer integralen Funktionsausnutzung in der Lage sein, komplexe Konstruktionen zu reduzieren [99].

Zunächst einmal sollten fallbasierte Systeme aber auch mit einer leistungsfähigen Ähnlichkeitsfunktion arbeiten, die Systemlösungen unterschiedlicher Komplexität und Kompliziertheit miteinander vergleichbar macht. Als Maß für die Ähnlichkeit kann man diesbezüglich beispielsweise den reziproken Wert der Anzahl der Systemkomponenten ansetzen. In diesem Sinne läßt sich etwa ein technisches Ventil (z. B. ein 4/2-Wegeventil) in einem Hydraulikschaltkreis als Systemkomponente definieren.

Die Automatisierung der Anpassungskonstruktion erfordert ein leistungsfähiges Konfigurationssystem. B. Stein hat ein inkrementell arbeitendes Konfigurationssystem beschrieben, welches mit einem sogenannten Additionsoperator arbeitet, um einzelne Systemkomponenten – Funktionsträger – nacheinander zu integrieren. [12]

Es ist gewiß ein Fortschritt in der Konstruktionstheorie für fluidische Systeme, wenn in Analogie zum Additionsoperator theoretisch ein sogenannter Minusoperator definiert wird, über welchen eine integrale Funktionsausnutzung

Logarithmus dieser Zahl an.“ Siehe Seite 71.

[8] Der Begriff integrale Funktionsausnutzung ist in der Konstruktionswissenschaft bislang nicht einschlägig definiert worden, obwohl man hier häufig von Integralbauweise und mitunter auch von einem Multifunktionsteil spricht. Daher sei ein Beispiel zur begrifflichen Präzisierung angegeben: In einem modernen Taxi hat sich das Fahrpreis-Display verändert. Früher befand sich die für den Fahrgast gut ablesbare Fahrpreisanzeige in der Nähe des Handschuhkastens oder in der Mittelkonsole; hierfür wurde also eigens ein kleiner rechteckiger Displaykasten eingebaut. Diesen Displaykasten gibt es in den neueren Taxis nicht mehr. Hier kann der Fahrgast den aktuellen Fahrpreis vom Innenspiegel ablesen; dort steht dann etwa in gut lesbaren roten Buchstaben „Fahrpreis: 12,80 DM“. Man hat also eine Symbiose zwischen der Rückspiegelfunktion und der Displayfunktion gefunden und hierfür eine innovative Lösung entwickelt, mit der man ein elektronisches Bauteil einspart.

[9] Diese Einsparungsthese für neue Systemlösungen gilt nicht immer. Hierbei handelt es sich somit um eine heuristische Bewertung.

[10] Rodenacker [105], S. 64 ff.

[11] Hierbei handelt es sich um eine Heuristik, also um ein Verfahren, welches nicht immer, aber meistens zu guten Ergebnissen führt.

[12] Stein [113], S. 49: „An addition operator specifies how two values of a functionality will be composed to a new value, if a new object is added to a given collection of objects, which themselves describe a part of the system to be configured.“

approximierbar ist. Hierbei muß man auch entscheiden, in welchem Umfang das Expertensystem eine Anpassungskonstruktion ausführt. Es kann durchaus sinnvoll sein, bestimmte Konstruktionsheurismen anzuwenden, die dem Konstrukteur eine abstrakte (Teil-)Lösung präsentieren, und ihm Gestaltungsfreiheiten belassen.

Die Idee des Minusoperators korrespondiert hier mit einem von dem Aachener Konstruktionstheoretiker R. Koller beschriebenen Heurismus: „Vollkommenheit entsteht offensichtlich nicht dann, wenn man nichts mehr hinzuzufügen hat, sondern wenn man nichts mehr wegnehmen kann.“[13]

Der wissenschaftliche Fortschritt ist in der Konstruktionsautomatisierung weniger in der Identifikation von „intelligenten“ Ableitungsmechanismen zu erwarten, sondern primär in einer kognitiven Durchdringung des fluidtechnischen Bereichs im Verbund mit einer fallbezogenen Assimilation von Konstruktionswissen. Leistungsfähige Entwurfsassistenten werden vor allem effiziente Wissensverarbeitungssysteme sein.

5.5 Der Funktionsbegriff

Wenn man die Entwicklung des wissenschaftlichen Fortschritts beobachtet, so stellt man häufig fest, daß wissenschaftlich-technischer Fortschritt mit einer zunehmenden Ausdifferenzierung des Erkenntnisobjektes einher geht. Dies läßt sich recht anschaulich am Atommodell der Kernphysik erläutern [106]. Der Begriff Atom kommt aus dem Griechischen und heißt unteilbar. Während der Atomphysiker Niels Bohr sein Atommodell noch mehr aus einer holistischen Sicht begründete, etablierte sich wenige Dekaden später die (experimentelle) Kernphysik, die sich insbesondere für die Kernkräfte interessiert. Obwohl sich das Atom und die Nukleonen somit als teilbar erwiesen, ist der Atombegriff geblieben und durch die Elementarteilchentabelle der Kernphysik ergänzt worden [89].

So ähnlich verhält es sich mit dem Funktionsbegriff in der Informatik und in der Konstruktionswissenschaft. Der ingenieurwissenschaftliche Funktionsbegriff wurde hierzulande insbesondere von Koller [97] [100], Rodenacker [105] und Roth [109] [110] geprägt. Parallel dazu entwickelte sich mit der industriellen Einführung des rechnergestützten Konstruierens sowie der Expertensystemtechnologie der Funktionsbegriff in der Informatik. Der anwendungsorientierte Beitrag von Stein [113] arbeitet mit einem anwendungsneutralen Funktionskalkül, welches sich somit prinzipiell auf den Elementarfunktionsbegriff von Koller übertragen läßt.

Es zeichnet sich ab, daß die rechnergestützte Anpassungskonstruktion vermutlich nicht ohne ein fundiertes Funktionskalkül leistungsfähig wird. Yoshikawa [120] hat den theoretischen Unterbau der Konstruktionswissenschaft heftig kritisiert, und hierbei auch das Funktionskalkül nicht verschont. Bislang gibt es keine einheitliche konstruktionswissenschaftliche Funktionstheorie. Koller [100] arbeitet mit sogenannten Elementarfunktionen und hat ein anderes Funktionskonzept als Roth [110], der mit Allgemeinen Funktionen arbeitet. So definiert Roth beispielsweise Speichern als Allgemeine Funktion, wogegen Koller diese Funktion nicht als Elementarfunktion betrachtet.

In dieser Situation ohne einheitliche Funktionstheorie ist es für die Entwickler von CAE-Systemen und konstruktionstechnischen Expertensystemen der beste Weg, die Systementwicklung auf anerkannte und bekannte Industrienormen – wie etwa DIN ISO 1219 – zu stützen. Der Stand der konstruktionswissenschaftlichen Funktionstheorie rechtfertigt diesen pragmatischen Ansatz.

5.6 Die Komplexitätsschranke

Wenn man von Komplexität und von Kompliziertheit spricht, so muß man sich vor Augen halten, daß dieser Begriff schwer faßbar ist. Es gibt Gestalterkennungsprobleme, da ist die menschliche Erkenntnisfähigkeit der Interpretationsleistung eines Expertensystems in zeitlicher und auch in qualitativer Hinsicht deutlich überlegen. Umgekehrt existieren Probleme, die dem Experten einige Schwierigkeiten bereiten, die jedoch für den Rechner trivial sind. Wenn man beispielsweise von einem Ingenieur eine einfache Multiplikation, also z. B. 7 mal 7 verlangt, so wird er sich nicht die Mühe machen, diese beiden Zahlen in den Taschenrechner einzugeben, sondern das Ergebnis im Kopf ausrechnen. Verlangt man aber die Multiplikation zweier neunstelliger Zahlen von ihm, so wird er sofort zum Taschenrechner greifen. Was für den Menschen schwierig wird, kann für den Rechner trivial sein; auch das Umgekehrte ist der Fall. Schwierige kognitive Probleme, z. B. aus dem sprachlichen Bereich, löst der menschliche Experte oftmals mühelos, während diese von der Expertensystemtechnologie nur unzulänglich bewältigt werden.

Man kann also nicht einfach ein Kriterium im absoluten Sinne für den Leistungsvergleich zwischen menschlicher

[13] Koller [99], S. 343

Tabelle 5.1: Case-Based Reasoning-Evaluierungsergebnisse zum Anwendungsbereich fluidische Systeme (Dissertation M. Hoffmann 1999).

Achsen-zahl	Retrieval-zeit	Reuse - Adaptation -	$sim \geq 0,8$	$sim \geq 0,9$	Simulations-test	Experten-urteil
1	« 1 s	10	17	13	10	10 x (+) 6 x (0) 1 x (-)
2	« 1 s	1,6	16	11	9	8 x (+) 7 x (0) 1 x (-)
3	« 1 s	1,1	17	10	7	7 x (+) 8 x (0) 2 x (-)
4	« 1 s	0,7	15	8	5	3 x (+) 10 x (0) 2 x (-)
5	« 1 s	0,5	18	6	1	1 x (+) 15 x (0) 2 x (-)

und künstlicher Intelligenz heranziehen, sondern muß differenzierter betrachten.[14] Obwohl in der Expertensysteminformatik oftmals davon die Rede ist, daß ein wissensbasiertes System die Fähigkeiten eines Experten nachbilden soll, indem es dessen Fachwissen verarbeitet, geht es in Wahrheit um eine gelungene Symbiose von menschlicher und künstlicher Intelligenz: Während Beschäftigte in Industriebetrieben oftmals in Schichten arbeiten, läuft der betreffende Prozeßrechner (z. B. für den CIM-Bereich) Tag und Nacht ermüdungsfrei. Wenn der Mensch angesichts der intensiven Regelverarbeitung die Übersicht verliert, weil das wissensbasierte System fallbezogen über 150 Regeln feuert, gilt diese Komplexitätsschranke keineswegs für den Regelinterpreter.

In der Forschungsdisziplin Künstliche Intelligenz ist es möglich, die Leistungsfähigkeit eines CBR-Systems über einen Turing-Test zu bewerten [96]. Wenn man die Kompetenz eines CBR-Systems für die Konzeption fluidischer Systeme evaluieren will, so müßte man mit einem (erfahrenen) Ingenieur und einem CBR-System einen Turing-Test machen. Bislang liegen diesbezüglich jedoch leider keine wissenschaftlichen Ergebnisse vor.

Verfügbar sind allerdings Evaluierungsresultate aus der Dissertation von M. Hoffmann [87], die in der Tabelle 5.1 wiedergegeben sind.

Die Evaluierungstabelle faßt fünf Testreihen mit je 20 Anfragen, also insgesamt 100 Systemresultate zusammen. Die Bezeichnungen der Spalten besitzen die folgenden Bedeutungen:

- *Achsenzahl*
  Die Testreihen wurden mit 1, 2, 3, 4 und 5 hydraulischen Achsen durchgeführt.
- *Retrievalzeit*
  Diese Spalte enthält die durchschnittliche Dauer des Retrieval-Schrittes in Sekunden.
- *Reuse, Adaptation*
  Diese Spalte enthält die Zahl der pro Sekunde angepaßten Fälle, also die Adaptionsrate. Im Schritt *Reuse* können die n ähnlichsten Fälle erzeugt werden bzw. adaptiert werden. Die Spalte *Reuse* gibt an, wieviele Gesamtentwürfe pro Sekunde aus einzelnen hydraulischen Achsen kombinierbar sind.
- $sim \geq 0.8$
  Diese Spalte enthält die Anzahl der vom CBR-System erzeugten Entwürfe, die eine Ähnlichkeit größer gleich 0,8 besitzen.

[14] Es ist interessant zu sehen, daß die These der Kybernetik [114] von Psychologen gestützt wird; dies vor allem auch von D. Dörner [80], S. 8: „Man kann die einzelnen Schritte einer geistigen Operation so genau beschreiben, daß man sie, wenn man das will, auch in einem ganz anderen Medium als im Gehirn stattfinden lassen könnte.“ Diese Auffassung wurde in jüngster Zeit von dem Neurologen Roth bestätigt [108].

- $sim \geq 0.9$
  Diese Spalte enthält die Anzahl der vom CBR-System erzeugten Entwürfe, die eine Ähnlichkeit größer gleich 0,9 besitzen.
- *Simulationstest*
  Diese Spalte enthält die Anzahl vom CBR-System der erzeugten Entwürfe, deren Simulationsergebnisse die Anforderungen erfüllen, die also praxistauglich sind.
- *Expertenurteil*
  Diese Spalte enthält die Bewertungsergebnisse eines Experten auf dem Gebiet fluidischer Systeme, der die Ergebnisse des CBR-Systems (Entwurfsassistenten) entsprechend den Urteilen geringer Änderungsaufwand (+), vertretbarer Änderungsaufwand (0) und großer Änderungsaufwand (-) bewertet hat.

Natürlich sind 100 Anfragen statistisch gesehen keine große Zahl, jedoch belegt dieses Evaluierungsergebnis, daß CBR-Systeme *praxistauglich* sind. Hoffmann vertritt den Standpunkt, daß Konstrutkionsfälle mit mehr als 5 hydraulischen Achsen selten vorkommen: „In der Praxis ist in der Regel die Anzahl der Unterfunktionen dieser Fälle kleiner gleich vier. Fälle mit mehr als vier Unterfunktionen bilden die Ausnahme, müssen aber der Völlständigkeit wegen auch behandelt werden können.“ [15]

Hinsichtlich der obigen Evaluierungstabelle ist festzuhalten:

- Die Retrievaldauer kann vernachlässigt werden.
- Bei Anfragen mit einer einzigen hydraulischen Achse können im Reuseschritt relativ viele Gesamtentwürfe generiert werden. Dies hängt damit zusammen, daß lediglich Skalierungen erforderlich sind und die Kombination der hydraulischen Achsen entfällt.
- Mit steigender Achsenzahl nimmt die Anzahl der Systementwürfe, die im Simulationstest den Anforderungen genügen, deutlich ab, d.h. die Entwurfsqualität des CBR-Systems verschlechtert sich mit steigender Achsenzahl. Der Grund hierfür ist, daß das von M. Hoffmann evaluierte CBR-System keine komplexe Anpassungskonstruktion ausführen kann, d.h. die Anpassungskonstruktion erstreckt sich hier nicht auf die Koppelung der hydraulischen Achsen. Zwar sind diesbezüglich Anwendungserfolge mit regelbasierten Systemen vorhersehbar, jedoch gibt es diesbezüglich für fluidische Systeme noch keine entsprechenden Expertensysteme.
- Die angegebenen Zahlen für die gefundenen Lösungsvorschläge charakterisieren Wahrscheinlichkeiten für gute Entwürfe. Obwohl die vom CBR-System gefundenen Entwurfslösungen regelmäßig nicht genau den Anforderungen entsprechen, lassen sich diese Systemlösungen meistens mit vertretbarem Konstruktionsaufwand erfolgreich ändern.

Das von Hoffmann evaluierte CBR-System automatisiert schwerpunktsmäßig die *Funktionelle Phase* sowie die *Prinzipielle Phase*. Für die Automatisierung der *Gestaltenden Phase* sind schon vor Jahren anwendungsbezogen Lösungen von Koller et al. [101] sowie von Roth und Bohle [111] beschrieben worden.

Eine Ähnlichkeitsfunktion für fluidische Systeme wurde jüngst von Stein und Niggemann beschrieben [115]. Hierbei handelt es sich um eine *allgemeine Ähnlichkeitsfunktion*, deren Anwendbarkeit noch verbessert werden kann, wenn man diese auf bestimmte Klassen, also z.B. für Hochdrucksysteme oder für Hochgeschwindigkeitssysteme, bezieht.

Zur Problematik des Lernens von Ähnlichkeitsfunktionen wurde in jüngster Zeit von Stahl auf der International Conference on Case-Based Reasoning 2001 ein Beitrag [112] veröffentlicht. Dessen Verfahren lernt die optimalen Gewichte einer globalen Ähnlichkeitsfunktion mit einem sogenannten *Similarity Teacher*. Die Aufgabe dieses Lehrers – Experten – ist es, die richtige Reihenfolge für ein Retrievalergebnis zu einer Anfrage q zu beurteilen. Hat das Case-Based Reasoning-System etwa die vier Fälle $< c_1, c_{16}, c_4, c_{42} >$ ausgegeben, so muß der Similarity Teacher bei einer falschen Reihenfolge eine Korrektur vornehmen. Kommt dieser etwa zu dem Schluß, daß die richtige Reihenfolge zu einer Anfrage q die Folge $< c_1, c_{42}, c_4, c_{16} >$ ist, so registriert das Verfahren von Stahl hierzu einen durchschnittlichen Ähnlichkeitsfehler, der mit einem Hill Climbing Algorithmus minimiert wird (lokales Minimum), d.h. das Lernverfahren generiert eine neue Ähnlichkeitsfunktion.

Die Methode von Stein und Niggemann [115] zur Konstruktion von Ähnlichkeitsfunktionen und das Lernverfahren von Stahl sind komplementär, d.h. die mit dem graphischen Cluster-Verfahren erzeugte Ähnlichkeitsfunktion kann mit dem überwachten Lernverfahren von Stahl verbessert werden.

[15] Hoffmann [87], S. 68

Ein bislang meistens unter der Bezeichnung *Adaptation Guided Retrieval (AGR)* wissenschaftlich diskutiertes Problem ist die Berücksichtigung von Adaptationsmerkmalen beim Retrieval [116]. Diesbezüglich ist festzuhalten, daß die unter dem Begriff AGR subsumierten Schemata regelmäßig auf die Berücksichtigung von Anpassungsmerkmalen in der Ähnlichkeitsfunktion zielen, was auch sinnvoll ist. Eine Verbesserung der Ähnlichkeitsfunktion hinsichtlich der Adaptionsmerkmale ist jedoch nicht gleichbedeutend mit einer Verbesserung der Adaptionsleistung, die etwa durch ein regelbasiertes Expertensystem erbracht wird. Es reicht bei unzulänglichen Retrievalergebnissen nicht aus, lediglich die Ähnlichkeitsfunktion zu sensitivieren, vielmehr müssen auch die Adaptionsleistungen im Sinne einer verbesserten Anpassungskonstruktion weiterentwickelt werden. Hierzu sind vor allem auch Verfahren zur Validierung und Verfeinerung von regelbasiertem Expertenswissen von Bedeutung. Ein Ansatz zur Validierung und Verfeinerung von Regelwissen, der mit einem ähnlichen *expert feedback* wie das Verfahren von Stahl arbeitet, wurde seinerzeit von Kelbassa [91] beschrieben. Die Weiterentwicklung dieses Verfahrens im Hinblick auf leistungsfähigere Verfeinerungsheurismen wird in [92] erläutert.

5.7 Zusammenfassung

Das vorliegende Kapitel analysiert den technischen Entwicklungsstand von CAE-Systemen für die Anpassungskonstruktion, insbesondere hinsichtlich der Anwendung für fluidische Systeme.

- Klassische Konfigurationssysteme wenden Kompositionsverfahren an: man konfiguriert ein System aufwärts von n auf n+1 Komponenten. Bislang gibt es leider keine erprobten Dekompositionsverfahren, also solche, die ein technisches System von n Komponenten auf n-1 oder n-x Komponenten reduzieren (Komponentenkonfiguration mit Exclude).
- Es sind Entwicklungsdefizite hinsichtlich einer Kostenfrüherkennung sowie einer Vorkalkulation von Konstruktionsobjekten zu registrieren. Bislang wurde für fluidische Systeme kein Kostenmodell erprobt. Die Konstruktionswissenschaft Karlsruher Prägung konzentriert sich zwar explizit auf ein umfassendes Produktmodell, jedoch wurde hier kein explizites Kostenmodell verfolgt, obschon die Kostenfrage stets betont wird.
- Unstrittige konstruktionswissenschaftliche Ideale sind Funktionsintegration und Integralbauweise. Für diesen konstruktionstechnischen Anspruch gibt es nicht nur hierzulande, sondern weltweit für den Ingenieur keine leistungsfähige Rechnerunterstützung. Es gibt keine CAE-Systeme, die eine Integralbauweise oder eine Funktionsintegration methodisch unterstützen.
- Der wirkungsvolle Einsatz von CBR-Systemen im Konstruktionsbereich wird stark von den Adaptionsleistungen des Anwendersystems abhängen. Die Adaption kann als Einsatzhürde (oder als Entwicklungsengpaß) aufgefaßt werden, die derzeit von vielen prototypischen Systemen nicht zufriedenstellend bewältigt wird.
- Von tragender Bedeutung für die Konstruktionsautomatisierung ist, inwieweit ein anerkanntes Funktionskalkül zu Grunde gelegt werden kann. Dies ist derzeit leider nicht der Fall. Während die Aachener Konstruktionstheorie sogenannte Elementarfunktionen favorisiert, arbeitet die Braunschweiger Konstruktionswissenschaft mit Allgemeinen Funktionen. In dieser Situation ist es für die Systementwickler der beste Weg, sich auf allgemein anerkannte und auch entsprechend bekannte Normen wie etwa DIN ISO 1219 zu stützen, weil die konstruktionswissenschaftliche Funktionstheorie noch nicht vereinheitlicht worden ist.
- Es gibt bisher hierzulande kein Turing-Testergebnis für konstruktionstechnische Expertensysteme, welches die Komplexitätsschranken von Ingenieur und CAE-System erfaßt und offenlegt.
- Hinsichtlich einer natürlichsprachlichen Benutzerschnittstelle sind Entwicklungsdefizite zu registrieren. Es gibt in Deutschland und vermutlich auch in Europa kein ausgereiftes CAD/CAE-System, welches dem Benutzer eine natürlichsprachliche Interaktion anbietet [90].

Literaturverzeichnis

[1] R. Bergmann (1999): Skriptum zur Vorlesung ‚Fallbasiertes Schließen'. Universität Kaiserslautern, Fachbereich Informatik

[2] P. Koten, M. P. Chase: Knowledge Representation in a Case-Based Reasoning System: Defaults and Exceptions, The MITRE Corporation, Bedford

[3] J. Lieber: A Criterion of Comparison between two Case Bases. RFIA group (France)

[4] S. Wess (1995): Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunter-stützung und Diagnostik. Dissertation. Universität Kaiserslautern, Fachbereich Informatik. Infix St. Augustin

[5] G. Görz (Hrsg.): Einführung in die künstliche Intelligenz. Addison-Wesley Bonn 1995

[6] Spektrum der Wissenschaft, Dossier 4/97: Kopf oder Computer. Heidelberg: September 1997

[7] J. Kolodner (1993): Case-based Reasoning. Morgan Kaufmann Publishers San Mateo

[8] W. Eversheim (1982): Organisation in der Produktionstechnik. Band 2: Konstruktion. VDI-Verlag Düsseldorf

[9] W. Eversheim (1996): Organisation in der Produktionstechnik. Band 1: Grundlagen. VDI-Verlag Düsseldorf

[10] W. Eversheim (1998): Organisation in der Produktionstechnik. Band 2: Konstruktion. 3. Auflage. VDI-Verlag Düsseldorf

[11] R. Koller, A. Ludwig, H.-P. Mannweiler (1982): Ein Programm zum Automatisieren von Hydrauliksteuerblöcken. Maschinenmarkt 88 (37), S. 745 - 748

[12] K. Roth, D. Bohle (1982): Rechnerunterstütztes methodisches Konstruieren von Hydraulik-Steuerplatten. Konstruktion 34, Heft 4, S. 125 -131

[13] F. Puppe, S. Ziegler, U. Martin, J. Hupp (Hrsg.): Wissensbasierte Diagnosesysteme im Service-Support. Springer-Verlag Berlin 2001

[14] D. R. Wilson, T. R. Martinez (1997): Improved Heterogeneous Distance Functions. Journal of Artificial Intelligence Research 6, pp. 1 - 34

[15] M. Hoffmann (1999): Zur Automatisierung des Designprozesses fluidischer Systeme. Dissertation. Universität Paderborn, Fachbereich Mathematik und Informatik

[16] T. Roth-Berghofer, I. Iglezakis (2001): Six Steps in Case-Based Reasoning: Towards a maintenance methodology for case-based systems. H. P. Schurr, S. Raab, G. Stumme, Y. Sure (Hrsg.): Professionelles Wissensmanagement - Erfahrungen und Visionen. Proceedings German Workshop on Case-Based Reasoning 2001 (GWCBR 2001), S. 198 - 208. Shaker Verlag Aachen

[17] S. Wess(1995): Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen. Dissertation. Universität Kaiserslautern, Fachbereich Informatik. Sankt Augustin: Infix

[18] Th. Ottmann, P. Widmayer (1993): Algorithmen und Datenstrukturen. 2. Auflage. BI-Wisschaftsverlag Mannheim Leipzig

[19] S.-J. Chen, C.-L. Hwang, F. P. Hwang (1992): Fuzzy Multiple Attribute Decision Making: Methods and Applications. Springer-Verlag Berlin New York

[20] R. Sedgewick (1992): Algorithmen in C. Addison-Wesley Bonn

[21] W. Dubitzky, A. Schuster, J. G. Hughes, D. A. Bell: Conceptual Distance of Numerically Specified Case Features.

[22] S. Wess (1993): PATDEX - Ein Ansatz zur wissensbasierten und inkrementellen Verbesserung von Ähnlichkeitsbewertungen in der fallbasierten Diagnostik. F. Puppe, A. Günter (Hrsg.): Expertensysteme XPS-93, S. 42 - 55. Springer-Verlag Berlin Heidelberg

[23] S. Wess:. PATDEX/2 - Ein fallbasiertes System zur Diagnose. Universität Kaiserslautern. Fachbereich Informatik

[24] H. Kleine Büning (1999): Wissensbasierte Systeme. Vorlesungsskript. Universität Paderborn, Fachbereich Mathematik und Informatik

[25] R. Bareiss, J. A. King (1989): Similarity Assessment in Case-Based Reasoning. K. J. Hammond (Hrsg). Proceedings: Case-Based Reasoning Workshop. Morgan Kaufmann Publishers San Mateo 1989

[26] Ch. Globig (1995): Fallbasiertes Repräsentieren und Lernen von Begriffsmengen. B. Bartsch-Spörl, D. Janetzko, S. Wess (Hrsg.): Proceedings Fallbasiertes Schließen - Grundlagen und Anwendungen. Workshop auf der 3. deutschen Expertensystemtagung (XPS-95), S. 37 - 43.

[27] E. G. Haffner (1993): Analyse Dynamischer Lernregeln für Case-Based Learning Systeme. Diplomarbeit. Universität Kaiserslautern, Fachbereich Informatik

[28] R. Rojas (1993): Theorie der neuronalen Netze. Eine systematische Einführung. Springer-Verlag Berlin Heidelberg

[29] S. Wess, Ch. Globig (1994): Case-based and Symbolic Classification - A Case Study - S. Wess, K.-D. Althoff, M. M. Richter (Eds.): Topics in Case-Based Reasoning. Proceedings: 1st European Workshop on Case-Based Reasoning (EWCBR-93), pp. 77 - 91. Springer-Verlag Berlin Heidelberg

[30] Th. Ottmann, P. Widmayer (1996): Algorithmen und Datenstrukturen. 3. Auflage. Spektrum Akademischer Verlag Heidelberg Berlin

[31] M. Lenz (1999): Case Retrieval Nets as a Model for Building Flexible Information Systems. Dissertation. Humboldt-Universität Berlin, Mathematisch-Naturwissenschaftliche Fakultät. Akademische Verlags-gesellschaft Aka GmbH. Infix (DISKI 236) Berlin

[32] R. Bergmann, M. M. Richter, S. Schmitt, A. Stahl, I. Vollrath (2001): Utility-Oriented Matching: A New Research Direction for Case-Based Reasoning. H.-P. Schnurr, S. Raab, R. Studer, G. Stumme, Y. Sure (Hrsg.): Profesionelles Wissensmanagement - Erfahrungen und Visionen. Proceedings German Workshop on Case-Based Reasoning 2001 (GWCBR 2001), S. 264 - 274. Shaker Verlag Aachen

[33] B. Smyth, M. T. Keane (1994): Retrieving adaptable cases: The role of adaptation knowledge in case retrieval. S. Wess, K.-D. Althoff, M. M. Richter (Eds.): Topics in Case-Based Reasoning. Proceedings: 1st European Workshop on Case-Based Reasoning (EWCBR-93), pp. 209 - 220. Springer-Verlag Berlin Heidelberg

[34] T. Roth-Berghofer, I. Iglezakis (2001): Six Steps in Case-Based Reasoning: Towards a maintenance methodology for case-based reasoning systems. H. P. Schnurr, S. Staab, R. Studer, G. Stumme, Y. Sure (Hrsg.): Professionelles Wissensmanagement - Erfahrungen und Visionen. Proceedings German Workshop on Case-Based Reasoning 2001 (GWCBR 2001), S. 198 - 208. Shaker Verlag Aachen

[35] I. N. Bronstein, K. A. Semendjajew, G. Musiol, H. Mühlig (1995): Taschenbuch der Mathematik. 2. Auflage. Verlag Harri Deutsch Thun und Frankfurt/Main

[36] D. B. Leake, A. Kinley, D. Wilson (1997): Case-Based Similarity Assessment: Estimating Adaptability from Experience. Proceedings 14th National Conference on Artificial Intelligence (AAAI-97), pp. 674 - 679. AAAI Press / The MIT Press Menlo Park, California

[37] A. Aamodt, E. Plaza (1994): Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AICOM 7 (1), pp. 39 - 57

[38] M. Lenz (1999): Case Retrieval Nets as a Model for Building Flexible Information Systems. Dissertation. Humboldt-Universität Berlin, Mathematisch-Naturwissenschaftliche Fakultät. Akademische Verlags-gesellschaft Aka GmbH. Berlin: Infix (DISKI 236)

[39] I. N. Bronstein, K. A. Semendjajew, G. Musiol, H. Mühlig (1995): Taschenbuch der Mathematik. 2. Auflage. Verlag Harri Deutsch Thun und Frankfurt/Main

[40] T. Ottmann, P. Widmayer (1996): Algorithmen und Datenstrukturen. 3. Auflage. Spektrum Akademischer Verlag Heidelberg Berlin

[41] S. Wess, D. Althoff, G. Derwald (1994): Using k-d Trees to Improve the Retrieval Step in Case-based Reasoning. S. Wess, K.-D. Althoff, M. M. Richter (Eds.): Topics in Case-based Reasoning, pp. 167 - 181. Proceedings First European Workshop (EWCBR-93), Kaiserslautern: November 1993. Lecture Notes in Artificial Intelligence 837. Springer-Verlag Berlin Heidelberg

[42] S. Wess (1995): Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen. Dissertation. Universität Kaiserslautern, Fachbereich Informatik. Infix St. Augustin

[43] J. H. Friedman, J. L. Bentley; R. A. Finkel (1977): An algorithm for finding best matches in logarithmic expected time. Transactions on Mathematical Software 3 (3), pp. 209 - 226

[44] O. Niggemann (2001): Visual Data Mining of Graph-Based Data. Dissertation. Universität Paderborn, Fachbereich Mathematik und Informatik

[45] H.-D. Burkhard (1998): Extending some Concepts of CBR-Foundations of Case Retrieval Nets. M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, S. Wess (Eds.): Case-Based Reasoning Technology. From Foundations to Applications. Lecture Notes in Artificial Intelligence 1400, pp. 17 - 50. Springer Verlag Berlin New York

[46] M. Lenz, E. Auriol, M. Manago (1998): Diagnosis and Decision Support. M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, S. Wess (Eds.): Case-Based Reasoning Technology. From Foundations to Applications. Lecture Notes in Artificial Intelligence 1400, pp. 51 - 90. Springer-Verlag Berlin New York

[47] T. M. Mitchell (1997): Machine Learning. MacGraw-Hill Boston New York

[48] B. Smyth, M. T. Keane (1994): Retrieving Adaptable Cases. S. Wess, K.-D. Althoff, M. M. Richter (Eds.): Topics in Case-Based Reasoning, pp. 209 - 220. First European Workshop (EWCBR '93), Kaiserslautern: November 1993. Springer-Verlag Berlin Heidelberg

[49] J. Altmeyer, S. Ohnsorge, B. Schürmann (1994): Reuse of Design Objects in CAD Frameworks. Proceedings of the International Conference on Computer Aided Design, pp. 754 - 761 (San Jose, CA)

[50] K. Börner (1994): Structural Similarity as Guidance in Case-Based Design. S. Wess, K.-D. Althoff, M. M. Richter (Eds.): Topics in Case- Based Reasoning. Proceedings First European Workshop on Case-Based Reasoning (EWCBR-93). Lecture Notes on Artificial Intelligence 837, S. 197 - 208. Springer-Verlag Berlin Heidelberg New York

[51] K. Börner, E. Pippig, E.-Ch. Tammer, C.-H. Coulon (1996): Structural Similarity and Adaptation. I. Smith (Ed.): Advances in case-based reasoning. Proceedings 3rd European Workshop on Case-Based Reasoning 1996 (EWCBR '96). Lecture Notes in Artificial Intelligence 1168, pp. 58 - 74. Verlag Berlin Heidelberg New York

[52] Börner, Katy (Ed.): Fabel: Modules for Design Support. Fabel-Report Nr.35. Universität Freiburg, Zentrum für kognitive Wissensschaft, 1995. Daraus:
Kapitel 4, S. 33 - 45: H. Coulon (1995): Generell Geometric and Topological Retrieval and Adaptation (TOPO).
Kapitel 6, S. 52 - 56: R. R. Bhat (1995): Case Adaptation using Agents (AgentEx).

[53] K. Hanney, M. T. Keane (1996): Learning Adaptation Rules from a Case-Base. I. Smith (Ed.): Advances in case-based reasoning. Proceedings 3rd European Workshop on Case-Based Reasoning 1996 (EWCBR '96). Lecture Notes in Artificial Intelligence 1168, pp. 179 - 192. Springer-Verlag Berlin Heidelberg New York

[54] T. R. Hinrichs, J. L. Kolodner (1991): The Roles of Adaptation in Case Based Design. Proceedings DARPA Case Based Reasoning Workshop, pp. 121 - 132

[55] J. H. Holland (1975): Adaptation in Natural and Artificial Systems. MIT Press Cambridge Mass.

[56] J. Hunt (1995): Evolutionary Case Based Design. Progress in Case-Based Reasoning. Proceedings First United Kingdom Workshop, pp. 17 - 31. Salford, UK

[57] B. Shiva Kumar, C. S. Krishnamoorthy: A framework for case-based reasoning in engineering design. Artificial Intelligence for Engineering Design Analsysis and Manufacturing (9) 3, pp. 161 - 182

[58] M. L. Maher, A. G. de Silva Garza (1996): Developing Case-Based Reasoning for Structural Design. IEEE Expert (11) 3, pp. 42 - 52

[59] Y. Nakatani, M. Tsukiyama, T. Fukuda (1992): Engineering Design Support Framework by Case-Based Reasoning. ISA 1992, S. 165 - 180

[60] R. Oxman, A. Voß (1996): CBR in Design. AI Communications 9, pp. 117 - 127

[61] R. Oxman, A. Voß (1996): Fallgestütztes Entwerfen. Künstliche Intelligenz 1/96, S. 29 - 34

[62] S. Perera, I. Watson, M. Alshawi: Case-Based Design Approach for Integration of Design and Estimating. B. H. V. Topping (Ed.): Developments in Computer Aided Design and Modelling for Civil Engineering, pp. 29 - 41. Civil-Comp Press, Edinburgh, UK

[63] B. Raphael, B. Kumar (1993): Representing Design Cases. B. H. V. Topping (Ed.): Knowledge Based Systems for Civil and Structural Engineering, pp. 259 - 264. Civil-Comp Press Edinburgh, UK

[64] C. Riesbeck, R. C. Schank (1989): Inside Case-Based Reasoning. Lawrence Erlbaum Hillsdale, N.J. (ISBN 0-89859-767-6)

[65] R. Rong, D. A. Lowther (1996): Adaptating Design Using Dimensional Models of Electromagnetic Devices. IEEE Transactions on Magnetics (32) 3, pp. 1437 - 1440

[66] J. W. Schaaf (1994): Detecting Gestalts in CAD-Plans to be Used as Indices for Case-Retrieval in Architecture. B. Nebel, L. Dreschler-Fischer (Eds.): KI-94: Advances in Artificial Intelligence. 18th Annual German Conference on Artificial Intelligence. Saarbruecken: September 18 - 23, 1994. Lecture Notes in Computer Science 861, S. 154 - 165. Springer-Verlag Berlin Heidelberg New York

[67] M. Tanaka, T. Hira (1995): Genetic Case-Base for Conceptual Structural Design in architecture. Systems Control and Information (33) 9, S. 1337 - 1340

[68] F. Zhao, M. L. Maher (1988): Using analogical reasoning to design buildings. Engineering with Computers 4, pp. 107 - 119

[69] D. Navin-Chandra (1988): Case Based Reasoning in CYCLOPS, a design problem solver. J. L. Kolodner (Ed.): Proceedings of the DARPA Workshop on Case-based Reasoning, pp. 286 - 301. Morgan Kaufman San Mateo CA

[70] F. Puppe, S. Ziegler, U. Martin, J. Hupp (Hrsg.): Wissensbasierte Systeme im Service-Support. Springer-Verlag Berlin 2001

[71] B. Smyth, M. T. Keane (1996): Using adaptation knowledge to retrieve and adapt design cases. Knowledge-Based Systems 9, pp. 127 - 135

[72] A. Voss, A. Giretti, M. de Grassi (2001): Cases, Concepts and Relationales for Intelligent Coaching in Design. H.-P. Schnurr, S. Staab, R. Studer, G. Stumme, Y. Sure (Hrsg.): Professionelles Wissensmanagement - Erfahrungen und Visionen. Proceedings German Workshop on Case-Based Reasoning 2001 (GWCBR '01), pp. 179 - 188. Shaker-Verlag Aachen

[73] T. M. Mitchell (1997): Machine Learning. WCB/McGraw-Hill Boston New York

[74] M. L. Maher, P. Pu (1997): Issues and Applications of Case-Based Reasoning in Design. L. Erlbaum Associates Publishers Nahwah, New Jersey London

[75] A. K. Goel, S. R. Bhatta, E. Stroula (1997): KRITIK: An Early Case-Based Design System. M. L. Maher, P. Pu (Eds.): Issues and Applications of Case-Based Reasoning in Design, pp. 87 - 132. Lawrence Erlbaum Associates, Publishers Mahwah, New Jersey London

[76] R. Bergmann, S. Breen, M. Göker, M. Manago, S. Wess (1998): Developing Industrial Case-Based Reasoning Applications - The INRECA-Methodology. Lecture Notes in Artificial Intelligence 1612. Springer-Verlag Berlin Heidelberg

[77] R. Bergmann; H. Munoz-Avila; M. Veloso; E. Melis (1998): *CBR Applied to Planning.* M. Lenz; B. Bartsch-Spörl; H.-D. Burkhard; S. Wess (Eds.): Case-based Reasoning Technology. From Foundations to Applications. Lecture Notes in Artificial Intelligence 1400, pp. 169 - 199. Springer-Verlag Berlin Heidelberg

[78] E. A. Cornell, C. E. Wiemann (1998): Die Bose-Einstein-Kondensation.
Spektrum der Wissenschaft 5/1998, S. 44 - 48

[79] J. C. da Silva; D. Dawson (1997): The Development of an Expert System for Hydraulic Systems Design Focussing on Concurrent Engineering Aspects.
Proceedings International Conference on Engineering Design (ICED 97), pp. 271 - 276

[80] D. Dörner (1979): *Problemlösen als Informationsverarbeitung.*
Verlag W. Kohlhammer Stuttgart Berlin

[81] W. Eversheim (1982): *Organisation in der Produktionstechnik.*
Band 2: Konstruktion. VDI-Verlag Düsseldorf

[82] W. Eversheim (1996): *Organisation in der Produktionstechnik.*
Band 1: Grundlagen. VDI-Verlag Düsseldorf

[83] W. Eversheim (1998): *Organisation in der Produktionstechnik.*
Band 2: Konstruktion. 3. Auflage. VDI-Verlag Düsseldorf

[84] D. G. Feldmann (1971): *Untersuchung des dynamischen Verhaltens hydrostatischer Antriebe.* Konstruktion 23, Heft 11, S. 420 - 428

[85] M. Fritsch (1967): *Zur integralen Funktionsausnutzung von Bauelementen.*
Feingerätetechnik 16 Jg., Heft 9, S. 402 - 404

[86] B. Frölich; K. Schlottmann (1982): *Systematische Konstruktion elektronischer Geräte.* VDI-Berichte Nr. 460, S. 19 - 27

[87] M. Hoffmann (1999): *Zur Automatisierung des Designprozesses fluidischer Systeme.* Dissertation. Universität Paderborn, Fachbereich Mathematik und Informatik

[88] R. J. Huber (1994): Wissensbasierte Funktionsmodellierung als Grundlage zur Gestaltsfindung in Konstruktionssystemen. Dissertation. Universität Karlsruhe (TH), Fakultät für Maschinenbau. Shaker Verlag Aachen

[89] B. M. Jaworski; A. A. Detlaf (1986): *Physik-Handbuch.*
Verlag Harri Deutsch Thun und Frankfurt/Main

[90] B. Jung, S. Kopp, M. E. Latoschik, T. Sowa, I. Wachsmuth (2000): Virtuelles Konstruieren mit Gestik und Sprache. Künstliche Intelligenz 2/00, S. 5 - 11

[91] H.-W. Kelbassa (1990): Fallbezogene Revision und Validierung von regelbasiertem Expertenwissen für die Altlastenbeurteilung. W. Pillmann und A. Jaeschke (Hrsg.): *Informatik für den Umweltschutz.*
Informatik-Fachberichte 256, S. 276 - 285. Springer-Verlag Berlin Heidelberg

[92] H.-W. Kelbassa (2002): Higher Order Refinement Heuristics for Rule Validation. Proceedings 15th Florida Artificial Intelligence Research Society Conference 2002 (FLAIRS-2002). Pensacola, Florida: May 14 - 16, 2002

[93] H. Kettner; V. Klingenschmitt (1971/73): *Die morphologische Methode und das Lösen konstruktiver Aufgaben.*
Erster Teil: wt - Z. ind. Fertigung 61, Heft 12, S. 737 - 741
Zweiter Teil: wt - Z. ind. Fertigung 63, Heft 6, S. 357 - 363

[94] R. Kläger (1993): Modellierung von Produktanforderungen als Basis für Problemlösungsprozesse in intelligenten Konstruktionssystemen. Dissertation. Universität Karlsruhe (TH), Fakultät für Maschinenbau. Verlag Shaker Aachen

[95] H. Kleine Büning (2001): Wissensbasierte Systeme III. Vorlesungsmanuskript. Universität Paderborn, Fachbereich Mathematik und Informatik

[96] R. Knauf (2000): *Validating Rule-Based Systems - A Complete Methodology.* Habilitationsschrift, Technische UniversitätIlmenau. Shaker-Verlag Aachen

[97] R. Koller (1979): *Konstruktionsmethode für den Maschinen-, Geräte- und Apparatebau.* Springer-Verlag Berlin Heidelberg

[98] R. Koller (1989): *CAD - Automatisiertes Zeichnen, Darstellen und Konstruieren.* Springer-Verlag Berlin Heidelberg

[99] R. Koller (1991): *Expertensysteme der Konstruktion.* Konstruktion 43, S. 339 - 343

[100] R. Koller (1998): *Konstruktionslehre für den Maschinenbau. Grundlagen zur Neu- und Weiterentwicklung technischer Produkte mit Beispielen.* Springer-Verlag Berlin Heidelberg

[101] R. Koller; A. Ludwig; H.-P. Mannweiler (1982): *Programm zum Automatisieren der Konstruktion von Hydraulikventilblöcken.* Maschinenmarkt 88 (37), S. 745 - 748

[102] I. Prigogine (1985): *Vom Sein zum Werden. Zeit und Komplexität in den Naturwissenschaften.* R. Piper & Co. Verlag München

[103] F. Puppe; S. Ziegler; U. Martin; J. Hupp (Hrsg.): *Wissensbasierte Diagnose-systeme im Service-Support.* Springer-Verlag Berlin Heidelberg 2001

[104] O. Niggemann (2001): *Visual Data Mining of Graph-Based Data.* Dissertation. Universität Paderborn, Fachbereich Mathematik und Informatik

[105] W. G. Rodenacker (1984): *Methodisches Konstruieren.*
Konstruktionsbücher: Band 27. Springer-Verlag Berlin Heidelberg

[106] U. Röseberg (1992): *Niels Bohr. Leben und Werk eines Atomphysikers.*
Spektrum Akademischer Verlag Heidelberg Berlin

[107] G. Ropohl (1979): *Eine Systemtheorie der Technik. Zur Grundlegung der Allgemeinen Technologie.* Carl Hanser Verlag München

[108] G. Roth (2001): Ist das menschliche Gehirn in seinen Leistungen nachbaubar?
Künstliche Intelligenz 2/01, S. 50 - 51

[109] K. Roth (1982): *Konstruieren mit Konstruktionskatalogen.*
Springer-Verlag Berlin Heidelberg

[110] K. Roth (1994): *Konstruieren mit Konstruktionskatalogen.*
Band 1: Konstruktionslehre.
Band 2: Konstruktionskataloge.
Springer-Verlag Berlin Heidelberg

[111] K. Roth; D. Bohle (1982): *Rechnerunterstütztes methodisches Konstruieren von Hydraulik-Steuerplatten.*
Konstruktion 34, Heft 4, S. 125 - 131

[112] A. Stahl (2001): Learning Feature Weights from Case Order Feedback. D. W. Aha und I. Watson (Eds.): *Case-Based Reasoning Research and Development.* Proceedings 4th International Conference on Case-Based Reasoning 2001 (ICCBR 2001), pp. 502 - 516. Springer-Verlag Berlin New York

[113] B. Stein (1995): *Functional Models in Configuration Systems.* Dissertation. Universität Paderborn, Fachbereich Mathematik und Informatik

[114] K. Steinbuch (1971): *Automat und Mensch. Auf dem Weg zu einer kybernetischen Anthropologie.*
Springer-Verlag Berlin Heidelberg

[115] B. Stein; O. Niggemann (2000): *Generation of similarity measures from different sources.* Proceedings 14th Internat. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2000)

[116] B. Smyth, M. T. Keane (1996): Using adaptation knowledge to retrieve and adapt design cases. Knowledge-Based Systems 9, pp. 127 - 135

[117] K. A. Tschörtner (1978): Entwicklung von Konstruktionsalgorithmen zur rechnerunterstützten Konstruktion von Hydraulik-Steuerblöcken. Dissertation. RWTH Aachen, Fakultät für Maschinenbau

[118] V. Velten (1999): Integration von Strömungsberechnungen in den rechnerunterstützten methodischen Konstruktionsprozeß. Dissertation. Universität Karlsruhe (TH), Fakultät für Maschinenbau. Shaker Verlag Aachen

[119] E. Vier (1999): *Automatisierter Entwurf geregelter hydrostatischer Systeme.* Fortschritt-Berichte VDI, Reihe 8, Nr. 795. VDI-Verlag Düsseldorf

[120] H. Yoshikawa (1983): *Automation of Thinking in Design.* E. A. Warman, K. Four (Eds.): Computer Applications in Production and Engineering. Proceedings of the First International IFIP Conference on Computer Applications in Production and Engineering (CAPE '83), Preprints: Part 1: pp. 405 - 417. North-Holland Publishing Company Amsterdam

[121] J. Zierep (1991): *Ähnlichkeitsgesetze und Modellregeln der Strömungslehre.* Verlag G. Braun Karlsruhe

[122] E. Vier, B. Stein, M. Hoffmann (1996): Strukturelle Formulierung von Anforderungen an hydrostatische Antriebe. Forschungsbericht Nr. 8/97. Universität Paderborn, Fachbereich Mathematik und Informatik

[123] S. Wess (1995): Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen. Dissertation. Universität Kaiserslautern, Fachbereich Informatik. Infix Sankt Augustin

[124] CASUEL: A Common Case Representation Language, ESPRIT project 6322, University of Kaiserslautern, Fachbereich Informatik 1994

[125] A. Schulz (1997): Graphenanalyse hydraulischer Schaltkreise zur Erkennung von hydraulischen Achsen und deren Kopplung. Diplomarbeit. Universität Paderborn, Fachbereich Mathematik und Informatik

[126] M. Hoffmann (1999): Zur Automatisierung des Designprozesses fluidischer Systeme. Dissertation. Universität Paderborn, Fachbereich Mathematik und Informatik