Request-Prediction and Hyperlink-Proposals
Methodologies and Mathematics behind Web-Applications

Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften
am Fachbereich IV der Universität Trier

vorgelegt von

Diplom-Informatiker
Ernst-Georg Haffner

November 2000

Gutachter: Prof. Dr. sc. Christoph Meinel (Universität Trier)

Prof. Dr. Michael M. Richter (Universität Kaiserslautern)

Datum der Disputation: 10. April 2001

Als ich mir vorgenommen hatte zu erkennen,
was Wissen wirklich ist, und zu beobachten,
welches Geschäft eigentlich auf der Erde getätigt wird,
da sah ich ein, dass der Mensch,
selbst wenn er seinen Augen bei Tag und Nacht keinen Schlaf gönnt,
das Tun Gottes in seiner Ganzheit nicht wiederfinden kann,
das Tun, das unter der Sonne getan wurde.
Deshalb strengt der Mensch, danach suchend, sich an
und findet es doch nicht wieder.

Kohelet 8,16-17a
Die Bibel – Einheitsübersetzung der Heiligen Schrift
(ökumenischer Text)

Danksagung

Sehr viele Menschen haben dazu beigetragen, dass diese Arbeit in der vorliegenden Form erstellt werden konnte.

Mein Dank gilt in erster Linie Prof. Dr. Christoph Meinel, dem Direktor des Instituts für Telematik, für die wissenschaftliche Begleitung der Dissertation sowie dem gesamten Stab an Mitarbeitern, die durch vielfältige Anregungen, Diskussionen und Hinweise das Promotionsvorhaben erst möglich gemacht haben.

Die langjährige Zusammenarbeit mit Dipl.-Inform. Uwe Roth und Dipl.-Phys. Andreas Heuer in den projektbezogenen Anwendungsbereichen "Smart Data Server" (SDS) und "Hyperlink-Management Systeme" (HLM) erwies sich hierbei als besonders fruchtbar und resultierte darüber hinaus in einigen gemeinsamen - thematisch relevanten - wissenschaftlichen Publikationen.

Darüber hinaus möchte ich insbesondere Dipl.-Math. oec. Torsten Becker danken für zahlreiche Anmerkungen, Kommentare und Korrekturvorschläge formaler und inhaltlicher Natur im Rahmen des Reviewing. Die Qualität der Arbeit konnte so deutlich gesteigert werden.

Für das sprachliche Korrekturlesen wurde ich auf der Suche nach einer Expertin auf diesem Gebiet sehr schnell fündig. Spontan sagte mir Ide Düro ihre Hilfe zu und fand zahlreiche grammatikalische und stilistische Verbesserungen. Dafür herzlichen Dank!

Mein aufrichtiger Dank gilt auch den vielen Menschen, die indirekt zur Fertigstellung der Arbeit beigetragen haben. Dr. Thomas Engel als stellvertretender Institutsleiter wäre hier ebenso zu nennen wie etwa Rüdiger Schlegel und Giulia Bäuerlein, die mich über viele Monate hinweg tatkräftig als wissenschaftliche Hilfskräfte unterstützt haben. Sehr wichtig ist und war mir stets die Motivation und der Rückhalt meiner Familie, ohne die das gesamte Promotionsvorhaben sicherlich undenkbar gewesen wäre.

Schließlich danke ich Prof. Dr. Michael M. Richter, dem Betreuer meiner Diplomarbeit, für seine Zusage, dieser Dissertation als Gutachter zur Verfügung zu stehen.

Abstract

Due to the breath-taking growth of the World Wide Web (WWW), the need for fast and efficient web applications becomes more and more urgent. In this doctoral thesis, the emphasis will be on two concrete tasks for improving Internet applications.

On the one hand, a major problem of many of today's Internet applications may be described as the performance of the Client/Server-communication: servers often take a long time to respond to a client's request. There are several strategies to overcome this problem of high user-perceived latencies; one of them is to predict future user-requests. This way, time-consuming calculations on the server's side can be performed even before the corresponding request is being made. Furthermore, in certain situations, also the pre-fetching or the pre-sending of data might be appropriate. Those ideas will be discussed in detail in the second part of this work.

On the other hand, a focus will be placed on the problem of proposing hyperlinks to improve the quality of rapid written texts, at first glance, an entirely different problem to predicting client requests. Ultra-modern online authoring systems that provide possibilities to check link-consistencies and administrate link management should also propose links in order to improve the usefulness of the produced HTML-documents. In the third part of this elaboration, we will describe a possibility to build a hyperlink-proposal module based on statistical information retrieval from hypertexts.

These two problem categories do not seem to have much in common. It is one aim of this work to show that there are certain, similar solution strategies to look after both problems. A closer comparison and an abstraction of both methodologies will lead to interesting synergetic effects. For example, advanced strategies to foresee future user-requests by modeling time and document aging can be used to improve the quality of hyperlink-proposals too.

Zusammenfassung

Aufgrund des atemberaubenden Wachstums des Internets seit der Einführung des World-Wide-Web Dienstes (WWW) wird auch der Bedarf an effizienten Webanwendungen zunehmend dringlicher. Die vorliegende Arbeit konzentriert sich hierbei auf zwei spezielle Aspekte moderner Internetanwendungen.

So erfordert die Client/Sever-Kommunikation im "Netz der Netze" eine besondere Aufmerksamkeit aufgrund des potentiell kritischen Antwortzeitverhaltens. Zur Reduktion der Wartezeiten für die Benutzer existieren dabei eine Reihe von Strategien, darunter die "Request-Prediction", die Vorhersage künftiger Benutzeranforderungen, die eines der beiden Hauptthemen dieser Arbeit darstellt. Korrekte Vorhersagen können dabei helfen, aufwendige Rechenarbeit auf Serverseite oder gar das Übertragen von Daten zu initiieren, bevor die entsprechende Anfrage durch den Anwender überhaupt gestellt wird.

Das zweite Schwerpunktthema der Arbeit befasst sich mit der automatisierten Generierung von Vorschlägen für Hyperlinks, den sogenannten "Hyperlink-Proposals". Dabei werden Wege aufgezeigt, wie man existierende oder künftige Redaktionssysteme für Websites im Internet oder Intranet durch die Ergänzung um ein "Hyperlink-Proposal Modul" (HPM) aufwerten könnte. Das vorgestellte Verfahren basiert auf der statistischen Informations-Extraktion aus den Online-Texten und lehnt sich dabei an die Technik des "fallbasierten Schließens", des "Case-Based-Reasoning" (CBR), an.

Zunächst scheinen diese beiden Aspekte von Webanwendungen kaum Gemeinsamkeiten aufzuweisen. Im Gegensatz dazu versucht die vorliegende Arbeit gerade nachzuweisen, dass ein tieferes Verständnis beider Anwendungsfelder nicht nur ähnliche methodologische Lösungsstrategien aufdeckt, sondern darüber hinaus auch Wege aufzeigt, wie mögliche Synergieeffekte genutzt werden können.

Kurzfassung

Das rasante Wachstum des Internets seit der Einführung des World-Wide-Web Dienstes (WWW) erfordert eine performante Infrastruktur und effiziente Softwarelösungen, um die informationstechnischen Herausforderungen angemessen bewältigen zu können. Hierbei müssen verschiedene Paradigmen moderner Webapplikationen berücksichtigt werden.

So ist die Client/Server-Kommunikation im Internet grundsätzlich durch variierende und unzuverlässige Bandbreiten und Auslastungen der Netzwerke geprägt. Die zugrundeliegenden Technologien weisen naturgemäß aufgrund der globalen Verbreitung sehr unterschiedliche Entwicklungsstadien auf. Mannigfaltige Informationen unterschiedlichster Qualität finden sich in allzu chaotischer Verteilung im "Netz der Netze" auf diversen Systemen, Datenbanken, Plattformen. Allein die Festlegung auf gemeinsame Kommunikationsprotokolle macht den Datenaustausch erst möglich.

Dem steht - im Allgemeinen - eine relativ hohe Erwartungshaltung auch und gerade des ungeübten Anwenders entgegen: Informationen sollen einfach, präzise, mit beachtlicher Qualität und insbesondere schnell auf Anfrage verfügbar sein.

Ausgehend von dieser Basis behandelt die vorliegende Arbeit primär zwei spezielle Themen aus bedeutsamen Bereichen der Webanwendungen: *Request-Prediction*, die Vorhersage von Benutzeranforderungen, sowie *Hyperlink-Proposals*, das Vorschlagen von Hyperlinks für die Texterstellung von Online-Autoren.

Der Prediction-Ansatz verfolgt hierbei unterschiedliche Ziele. Zum einen sollte sich das vom Einzelbenutzer wahrgenommene Antwortzeitverhalten des Servers merklich verbessern ohne dabei in der Gesamtbilanz Leistungseinbußen für andere Teilnehmer zu erzeugen. Zum anderen kann eine Modellierung des Benutzerprofils selbst Erkenntnisse über die Natur des Anforderungsverhaltens a posteriori erbringen, die als Grundlage für angemessenes Design von Webinhalten dienen mag.

Die Idee zur Generierung von Link-Vorschlägen für Hypertexts beruht auf dem Urgedanken des WWW, nämlich dass verwandte Informationsinhalte durch einfache Mausklicks miteinander in Beziehung gebracht werden können und sollten. Der zentrale und sehr einfache Mechanismus basiert hierbei auf den sogenannten *Hyperlinks*, Referenzen auf weitere relevante Adressen, die das Navigieren im WWW erst zum "Surfen" machen.[1] Das in dieser Arbeit vorgestellte automatische Erzeugen von derartigen Links als Vorschläge zur Ergänzung von Hypertexts sollte einen kleinen Beitrag

[1]Für die vorliegende Arbeit sind nur Links, deren Zieladressen weitere Hypertexts darstellen, von besonderem Interesse. Referenzen zu graphischen, akustischen oder anderen multimedialen Zielobjekten werden dabei nicht gesondert berücksichtigt, obgleich ihre Bedeutung im WWW zunehmend steigt.

dazu leisten, die Vernetzungsdichte von Dokumenten zu erhöhen und damit ebenso die Brauchbarkeit der Inhalte zu steigern.

Die vorliegende Arbeit gliedert sich in 4 Hauptteile. Im ersten Teil wird neben einer Einführung in die Thematik Basiswissen vermittelt, das sowohl die methodischen Grundsätze als auch das Umfeld und die konkreten Einsatzgebiete der später vorgestellten Entwicklungen und Anwendungen verdeutlichen soll.

Der zweite Teil ist dem Thema "Request-Prediction" gewidmet. Hier werden in mehreren Kapiteln Grundlagen, eine exemplarische Implementation sowie die Auswertung eines einfachen Modells zur Vorhersage von Benutzeranfragen im Internet geliefert. Darüber hinaus werden fortgeschrittene Modellerweiterungen diskutiert. Eine Zusammenfassung der wichtigsten Verfahren und Erkenntnisse findet sich im folgenden Abschnitt *Request-Prediction*.

Der Bereich des "Hyperlink-Proposals" wird im dritten Teil der Arbeit behandelt. Als methodische Ausgangsbasis zur Entwicklung eines Moduls zur Generierung von Linkvorschlägen dient der Forschungsbereich des fallbasierten Schließens, oder *Case-Based-Reasoning* (Grundlagen hierüber sind in Abschnitt 3.1 des Haupttextes zu finden). Ein kompaktes Resümee der wichtigsten Erkenntnisse wird im übernächsten Abschnitt *Hyperlink-Proposals* innerhalb dieser Zusammenfassung thematisiert.

Das überraschende Ergebnis der Gegenüberstellung beider Anwendungsgebiete, die auf den ersten Blick nur geringe Gemeinsamkeiten aufweisen, findet sich im vierten Teil der vorliegenden Dissertation. Ein Extrakt der bedeutendsten Analyseergebnisse und mögliche Synergieeffekte werden im letzten Abschnitt *Synthese* vorgestellt.

Aufgrund der formal unterschiedlichen Anwendungsgebiete werden die bedeutsamsten Referenzen für Request-Prediction und Hyperlink-Proposals getrennt in den jeweiligen Hauptabschnitten behandelt. Das gemeinsame Literaturverzeichnis findet sich am Ende des vierten Teils.

Die beiden Anhänge enthalten kommentierten Quellcode für mögliche Implementationen eines Testszenarios für die Benutzervorhersage und bedeutsame Extrakte eines Moduls zur Generierung von Hyperlink-Vorschlägen. Eine Besprechung der jeweiligen Algorithmik findet sich für die Request-Prediction in den Kapiteln 5 und 6 sowie für den Bereich des Hyperlink-Proposals in Kapitel 8.

Request-Prediction

Die zum Teil erheblichen Wartezeiten bei Webanwendungen auf eine Antwort des Servers nach Absendung einer Anforderung von der Client-Seite können auf den Anwender sehr störend wirken. Eine Möglichkeit zur Verbesserung des Antwortzeitverhaltens kann darin bestehen, künftige Benutzeranforderungen vorherzusehen. Aufwendige Rechenoperationen etwa können so bereits auf der Serverseite stattfinden, bevor eine entsprechende Anfrage von Seiten des Anwenders stattgefunden hat. Einen noch stärkeren Effekt kann das Client-Programm durch Prefetching von Daten erzielen. Ein analoger Mechanismus auf der Server-Seite kann ebenfalls Daten im Voraus an den Klienten senden, allerdings nur als Antwort auf eine (initiale) Anfrage des Anwendersystems.

Insbesondere das Übertragen von noch nicht angeforderten Informationen birgt jedoch enorme Risiken mit negativen Effekten auf die Netzwerk- und Systemlast für

den Fall, dass die Vorhersage unzutreffend ist.

Ausgangspunkt der Überlegungen in Kapitel 4 sind deshalb generelle Voraussetzungen, die als Basis für alle hierauf aufbauenden Verfahren dienen können. Eine besondere Rolle für die Herleitung konkreter Vorhersage-Formeln spielen dabei ebenso verschiedene Kostenbetrachtungen. Die voraussichtlich verursachten Kosten für die Bereitstellung und Übertragung von Datensätzen mit und ohne den Einsatz von Prediction-Techniken werden einander gegenübergestellt und verglichen (4.3). Hieraus ergeben sich - je nach gewähltem Ansatz - unterschiedliche Schwellwerte für die Wahrscheinlichkeit, dass ein bestimmter Datensatz innerhalb einer Session[2] angefordert werden wird. Nur bei Überschreiten dieses Wertes kann die Vorhersage durch frühzeitige Berechnungen bzw. Datenübertragungen kostenmindernd eingesetzt werden.

Die Anforderungswahrscheinlichkeiten selbst ergeben sich hierbei unmittelbar aus den bedingten Wahrscheinlichkeiten für künftige Requests unter Berücksichtigung des bisherigen Nutzerverhaltens (4.4.2.1). Dieser Ansatz ist etwas allgemeiner als das üblicherweise eingesetzte *Markov-Modell* (3.2), damit es sich nicht allein für die Vorhersage von HTTP[3]-Anforderungen eignet, sondern allgemeiner für unverlinkte Datensätze einsetzbar ist. Als Beispiel hierfür sei der Smart-Data-Server (SDS) angeführt, der in 2.1 beschrieben und charakterisiert wird.

Ein besonderes Problem im Umfeld von Prediction-Algorithmen ergibt sich aus der Schwierigkeit einer angemessenen Bewertung der Praxistauglichkeit des jeweiligen Ansatzes. Aufgrund der hohen Risiken bei inkorrekter Vorhersage ist die Probeeinführung eines derartigen Systems in der späteren Einsatzumgebung oft unangebracht.

Anstatt dessen modellieren wir ein Testszenario, in dem unterschiedliches Benutzerverhalten simuliert und über diverse Parameter gesteuert und kontrolliert werden kann. Dabei gehen wir davon aus, dass bestimmte Elemente einer Session teilweise vorhersehbar ("semi-random") und andere so sehr vom Zufall abhängig sind, dass sie keine signifikant höheren a priori Wahrscheinlichkeiten gegenüber den verbleibenden Elementen aufweisen ("random"). Demgemäß wirken sich die systematisch eingestellten Parameter wie der *Random Factor* und die *Density* unterschiedlich auf verschiedene Requests innerhalb einer Session aus. Mit ersterem lässt sich die Bedeutung des Zufalls steuern, während letzterer für die mittlere Anforderungshäufigkeit innerhalb einer Benutzersitzung verantwortlich zeichnet. Zusammen mit den Einstellungen für Umfang und Anteil an teilweise vorherzusagenden Elementen sowie den sich aus den Kostenbetrachtungen ergebenden Wahrscheinlichkeitsschwellwerten erhalten wir so eine naheliegende und relativ natürliche Modellierung des Nutzerverhaltens. Details hierzu sind im Abschnitt 5.1 zu finden.

Aus den verschiedenen Wertezuweisungen der beschriebenen Parameter ergibt sich eine unterschiedliche Qualität des Prediction-Ansatzes. Zur Quantifizierung dieser Ergebnisse dient dabei die *Prediction Quality* (*PQ*), die primär als Quotient aus korrekten Vorhersagen und der Anzahl aller Versuche betrachtet werden kann. Hierbei zeigt sich deutlich, dass der Zufälligkeitsanteil in den Benutzeranforderungen nicht zu groß werden darf, damit der Einsatz von Vorhersage-Algorithmen sinnvoll bleibt. Eine detaillierte Diskussion dieser und weiterer Ergebnisse ist in Abschnitt 5.3 aufgeführt.

[2] Zur Diskussion des Session-Begriffs sei auf 4.3.1 verwiesen.

[3] **H**yper**t**ext **T**ransfer **P**rotocol, das Kommunikationsprotokoll des WWW [HTTP]

Obgleich die Vorhersage-Qualitäten des beschriebenen Ansatzes bereits ermutigend sind, lässt sich das Benutzerverhalten mit einem fortgeschritten Modell noch angemessener abbilden. Hierbei spielt die *Zeit* und das *Altern von Datensätzen* eine enorme Rolle. Es erscheint adäquat, dass sich die Bedeutung gewisser Informationen und damit ebenso das Anforderungsverhalten der Benutzer von Serverdaten mit der Zeit verändert. Durch die Einführung einer *Request-Change Probabilty* (RCP) in die Generierungsalgorithmik wird dem Rechnung getragen. Die temporären Verhaltensschwankungen lassen sich somit in Grundzügen modellieren. Eine Diskussion dieser Aspekte findet sich in 6.1. Hier werden ebenfalls die verbesserten Formeln für die Kostenbetrachtung hergeleitet und der *Time Factor* eingeführt, der als variabler Kontrollpunkt zur praktischen Umsetzung dieses Ansatzes dient. Die Prediction Ergebnisse werden dabei auf die gleiche Weise wie im Standard-Fall bewertet.

Darüber hinaus dienen Webserver-Logs a posteriori zur objektiven Bestätigung der theoretischen Ergebnisse mittels geeigneter Parametereinstellungen. Ein außergewöhnliches Resultat ergibt sich hierbei aus der Rückermittlung möglicher Parameter für das Benutzermodell. Aus den Vorhersage-Ergebnissen lassen sich somit quantifizierbare Rückschlüsse auf das Anforderungsverhalten ziehen. Für das künftige Informationsdesign von Websites können derartige Charakterisierungen von sehr großem Wert sein. Eine Gesamtevaluation des zeitmodellierenden Ansatzes findet sich unter 6.4, der den zweiten Teil der Arbeit abschließt.

Hyperlink-Proposals

Es lassen sich zahlreiche Möglichkeiten ersinnen, wie die Erzeugung von Linkvorschlägen automatisiert werden könnte. In dieser Arbeit wählen wir einen relativ ungewöhnlichen Weg, indem wie diese Aufgabenstellung als eine mögliche Anwendung des bereits erwähnten *fallbasierten Schließens* (CBR) verstehen. Die Texte von Online-Autoren werden dabei als Problemstellung modelliert, während die hierin aufscheinenden Hyperlinks als Lösungen dieser Situation begriffen werden. Das Expertenwissen, repräsentiert durch eine Fallbasis, dient dazu, eine neue Problemkonstellation dadurch zu klassifizieren, dass die Lösungsansätze ähnlicher Fälle übernommen oder transferiert werden, um das aktuelle Problem lösen.

Diese klassische Ausgangssituation des CBR wird jedoch nur in den Grundzügen angewendet, weil zahlreiche besondere Aspekte im Umfeld von Hyperlink-Proposals eine andere Vorgehensweise nahe legen. So muss etwa beachtet werden, dass Links in Hypertexts sehr unterschiedliche Funktionalität besitzen können. Reine Navigationsverweise werden üblicherweise von Online-Redaktionssystemen automatisch erzeugt (vgl. 2.2.2), andererseits kann Metawissen über externe Seiten bei der Erzeugung diesbezüglicher Hyperlinks kaum verlässlich Verwendung finden.[4]

Im Gegensatz zu Diagnosesystemen, einem klassischen Anwendungsgebiet des CBR, spielt die Vermittlung des Prozesses an den Benutzer, *wie* das Programm, ausgehend von den Symptomen bis hin zur endgültigen Diagnose gelangt, eine untergeordnete

[4]Der Begriff "extern" impliziert hierbei Zieladressen, deren Domainname außerhalb der Kontrolle des aktiven Redaktionssystems liegt und bei denen möglicherweise nicht einmal die Verfügbarkeit für die nahe Zukunft gewährleistet ist, geschweige denn, dass umfangreiches Metawissen über diese Seiten zur besseren Klassifikation ausgenutzt werden könnte.

Rolle. Außerdem verursachen fehlerhafte oder nur unzureichend qualifizierte Linkvorschläge keine Kosten, die mit unzutreffenden Diagnosen vergleichbar wären. Dem steht eine rasche Verfügbarkeit von Vorschlägen gegenüber, die ohne großen Pflegeaufwand im Bereich der Wissensbasis mit akzeptablen Trefferquoten erreicht werden muss. Dieser Aspekt erwächst aus dem Paradigma von Webanwendungen, dass Inhalte sehr zeitnah bereitzustellen sind und die verwendeten Systeme nur unter Einsatz vergleichsweise geringer personeller Ressourcen betrieben werden können.

Die Modellierung eines *Hyperlink-Proposal Moduls* (HPM) auf der Basis von CBR weicht dementsprechend auch sehr stark von den traditionellen Vorgehensweisen ab und lässt sich kaum gewinnbringend in das Umfeld von Diagnosesystemen transferieren.

Im Abschnitt 7.2 wird ein Ansatz zum Design eines HPM hergeleitet, der im wesentlichen aus zwei Phasen besteht. In einer *Klassifikationsphase* werden die Merkmale von Texten in Form von Attributsvektoren mit einer Relevanzmatrix multipliziert, die die Bedeutung der einzelnen Attribute für die Zuordnung zu den im System bekannten Links beinhaltet. Als Ergebnis ergibt sich ein Vektor, dessen Elemente als Wahrscheinlichkeiten dafür interpretiert werden können, dass die zugehörigen Links auch brauchbar für den zu klassifizierenden Text sind. Das HPM wird nur dann eine gewisse Anzahl dieser Proposals absteigend sortiert für den Online-Autor anbieten, wenn gewisse Schwellwerte überschritten sind, die die Verwendung der Hyperlinks nahe legen. Die *Lernphase* neuer Hypertexts zusammen mit ihren Links beinhaltet zunächst einen Klassifikationsprozess des reinen Textes, von dessen Ergebnis das weitere Vorgehen abhängt. Nur solche Verweise müssen "gelernt" werden, die das System nicht ohnehin bereits korrekt klassifiziert. Das Lernen selbst bedeutet dabei eine Veränderung der Gewichte innerhalb der Relevanzmatrix, so dass im Ergebnis der zu lernende Link gerade den Klassifikationsschwellwert erreicht. Eine unmittelbar nachfolgende (erneute) Klassifikationsphase würde somit das gewünschte Resultat zeitigen. Es ist wichtig anzumerken, dass hierbei die Vorschläge zusätzlicher Links, die sich nicht im Ausgangstext finden, zunächst keine Anpassung der Gewichte nach sich ziehen. Vielmehr werden wir auf diesen Aspekt im folgenden Abschnitt *Synthese* genauer eingehen.

In der Praxis treten Lernen und Klassifizieren gemeinsam in Erscheinung. Der Autor von Hypertexts erfährt die Klassifikation seines Elaborates in Form von Linkvorschlägen, die er verwerfen oder in seine Arbeit einbauen kann, was zugleich einen Lernschritt nach sich zieht. Dies gilt ebenso für neu zu erstellende Links, die bis zu diesem Zeitpunkt nicht vorgeschlagen werden konnten. Weiter unten werden wir auf diesen dynamischen Aspekt erneut und detaillierter eingehen.

Ein Punkt, der bisher unerwähnt geblieben ist, betrifft das Informations-Retrieval von Hypertexts. Auf welcher Basis wird der Attributsvektor, der einen Text repräsentiert, erzeugt? Die genauer Beantwortung dieser Frage ist Aufgabe des Abschnitts 8.1. Zwei grundlegend verschiedene Ansätze könnten hier gewählt werden. Entweder findet diese Informationsextraktion auf der Basis einer komplexen semantischen Repräsentation des Textes statt (z.B. mittels "semantischer Netzwerke"), oder aber anhand statistischer Kenndaten auf syntaktischer Ebene.[5]

[5]Natürlich ist auch eine Kombination denkbar und de facto beinhaltet eine semantische Repräsentation in der Regel auch die entsprechenden syntaktischen Analysemethoden.

Aufgrund des bereits erwähnten Paradigmas haben wir uns dazu entschlossen, ein schnelles, vollautomatisiertes Informations-Retrieval einem höher qualifizierten semantischen Verfahren vorzuziehen, das zwar im Endeffekt Vorschläge von höherer Qualität liefern könnte, doch zu einem für Webanwendungen kaum zu bezahlenden Preis: einem hohen, zeitaufwendigen Anteil an Benutzer-Interaktion.[6]

Im Wesentlichen wird ein Hypertext demgemäß durch Informationen über die Autorenschaft, den zugehörigen Bereich innerhalb der Website, den Gültigkeitszeitraum sowie weiteren leicht zu ermittelnden Kenndaten, die das betreibende Redaktionssystem zur Verfügung stellt, repräsentiert. Der größte Anteil an Elementen des Attributsvektors resultiert jedoch aus der Schlüsselwort-Extraktion. Hierbei wird das Aufscheinen von Wörtern innerhalb von bestimmten Hypertags (z.B. <TITLE> ... </TITEL>) gewichtet und aufsummiert, wobei auch Stoppwortlisten Anwendung finden. Der so entstandene Vektor modelliert automatisch einen Text auf einer statistischen Basis, die zwar im allgemeinen eine gute Repräsentation liefert, im Einzelfall jedoch auch erheblich von der semantischen Ausrichtung abweichen kann.

Eine Besonderheit des vorgestellten HPM - die Implementierung wird in Kapitel 8 beschrieben - ergibt sich aus den dynamischen Größen des Attributsvektors und des möglichen Klassifikationsvektors: Innerhalb der Lernphase können beide Dimensionen wachsen und diesem Umstand müssen ebenso die verwendeten Datenstrukturen Rechnung tragen. Für überschaubare Websites stellt dies kein allzu gravierendes Problem dar, während umfangreiche Dokumentensammlungen hier an die Grenzen von Systemressourcen stoßen. Im folgenden Abschnitt werden wir hierauf kurz eingehen und einen Lösungsweg skizzieren, der sich - unter anderem - aus gewissen Synergieeffekten mit dem Feld der Request-Prediction ergibt.

Synthese

Bei näherer Betrachtung lässt sich der erste Eindruck, nämlich dass die beiden Forschungsfelder von Request-Prediction und Hyperlink-Proposals kaum Gemeinsamkeiten aufweisen, nicht weiter aufrecht erhalten. Tatsächlich sind nicht nur sehr abstrakte Ansätze wie Lernprozesse und Klassifikationsphasen einander ähnlich, sondern de facto weisen auch implementatorische Details bis hin zu nahezu identischen Datenstrukturen wie z.B. der Relevanz- bzw. Memory Matrix große Gemeinsamkeiten auf. Hieraus ergeben sich unmittelbar zwei Fragestellungen, die in 10.1 genauer untersucht werden: Wodurch entsteht diese Gemeinsamkeit und wie lässt sie sich in Form von Synergie ausnutzen?

Die erste Antwort ergibt sich aus der Betrachtung der Problemkonstellation von einer höheren Abstraktionsebene aus. Hier erweisen sich Request-Prediction und Hyperlink-Proposals als verschiedene Objekte einer einzigen Klasse von Webanwendungsgebieten, die sich mit *kognitiven Algorithmen* lösen lassen. Unter 10.2 sind weitere Beispiele dieser Gruppe aufgelistet.

Die Frage nach möglichen Synergieeffekten lässt sich anhand eines Beispiels unmittelbar plausibel machen. Wenn sich der fortgeschrittene Ansatz der Request-

[6] Im Einzelfall mag sogar diese Rechnung aufgehen: Wenn das System nicht allein Linkvorschläge generieren, sondern umfangreiches, unternehmensweites Informationsmanagement betreiben muss, kann sich unter Umständen selbst enormer Pflegeaufwand wieder rechtfertigen lassen.

Prediction, bei dem Zeit und Dokumentalterung modelliert werden, als erfolgversprechend für die Vorhersage von Benutzeranforderungen zeigt, so kann eine Übertragung auf das Gebiet der Linkvorschläge ebenfalls eine Verbesserung erwarten lassen. De facto bedeutet die Einführung des Zeitfaktors für die Prediction eine Anpassung von Gewichten für Elemente, die gerade *nicht* angefordert worden sind. Der Transfer dieser Idee auf den HPM-Ansatz führt somit zu einer Betrachtung auch von Linkvorschlägen, die ebenfalls *nicht* zu klassifizieren sind. Als Term aus dem Bereich der Kognition könnte hier das *"Vergessen"* angeführt werden. Eine Übertragung der Zeitmodellierung auf Hyperlink-Proposals führt also zu einer Erweiterung des Lernalgorithmus, bei dem nicht zu klassifizierende Links zur Reduktion von Gewichten in der Relevanzmatrix führen. Obgleich dieses Konzept eine geradlinige Verbesserung des Grundalgorithmus darstellt, so mag dennoch verwundern, dass es ebenso als eine Modellierung von Zeit und Dokumentenalterung begriffen werden kann.

Es darf erwartet werden, dass sich in Zukunft weitere Synergieeffekte in dieser Form auch für andere Bereiche von Webanwendungen ergeben, die nicht allein zu einer Verbesserung in den Ergebnissen der jeweiligen Algorithmen führen, sondern darüber hinaus das Verständnis vom Wesen der entsprechenden Methodologien erweitern und vertiefen.

Contents

List of Figures

List of Tables

Table of Symbols

Symbol	Description	Page
$\mathcal{A}$	Length of the artificial (generated) session vector	32
$\mathcal{B}$	Number of predictable elements of a generated session vector	32
$\mathcal{C}$	Set of contradictory attributes	21
$\mathcal{D}$	*Density* as global controling parameter for session generation	40
$\mathcal{E}$	Abbrevation variable for a term concerning fulfilled attributes defined in (3.8)	23
$\mathcal{F}$	Set of fulfilled attributes	21
$\mathcal{H}$	Hypertext representated as a case	66
$\mathcal{I}$	Number of initial training sessions within the test runs	43
$\mathcal{L}$	Hyperlink-vector representing the solutions as part of a case	66
$\mathcal{M}$	Number of possible solutions of a classification (CBR)	21
$\mathcal{N}$	Number of different attribute classes (CBR)	21
$\mathcal{P}$	Number of prediction sessions within the test runs	43
$\mathcal{R}$	*Random-factor* as global controling parameter for session generation	40
$\mathcal{S}$	Vector representing probabilities of possible solutions for a classification step	67
$\mathcal{T}$	Taw text represented as attribute (problem) vector	66
$\mathcal{W}$	Abbrevation variable for a term concerning contradictory attributes defined in (3.9)	23
$\vec{s}$	User session as binary vector	31
α	Factor for common attributes to calculate similarity	20
β	Factor for contradictory attributes to calculate similarity	20
Γ, γ	Cost functions for sessions and single requests, overview table (4.1)	32
δ	Threshold value; for attributes denoted as δ^j, for problem classes denoted as δ_i	21
ϵ	Small value to guarantee a classification below a corresponding threshold	100
ζ	Attribute separation function	69
η	Attribute comparison function	69
κ	Abbrevation term used for the derivation of distribution functions	69
λ	Limitation value for reasonable request prediction calcualtion	33
ξ	Request change probability used for advanced request prediction	48
ρ	Abbrevation term used for the derivation of distribution functions	69
τ	Time tactor used for advanced request prediction	48
ϕ	Probability for generated session vector elements to become 1	40
φ	Memory Matrix as core of the request prediction module	36
χ	Abbrevation term used for the derivation of distribution functions	69
ψ	Relevance Matrix as core of an Hyperlink Proposal Module (HPM)	66
ω	Relevance Matrix used for CBR	21
Δ	Value for chaning the relevance weights within a learning phase	70
∇	Abbrevation term used for the derivation of advanced prediciton formulas	48

Part I

Basics

Chapter 1

Introduction

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.

Thomas Stearns Eliot, Little Gidding
from "Four Quartets" (1943)

Due to the breath-taking growth of the World Wide Web (WWW), the need for high quality, fast and efficient applications has become more urgent. On the one hand, user-perceived latency has to be reduced in order to provide multimedia information. On the other, the hyperlink-structure of the web is very important and the diversity of data sources can only be achieved by appropriate links between web-pages.

Basically, the overall aim of this work is to provide practical usable concepts and methodologies to improve web-applications. Concretely, two special areas of remarkable importance are being analyzed and examined: *Request-Prediction* and *Hyperlink-Proposals*. At first glance, they have hardly anything in common. We will show in the fourth part of this work that in fact, the prediction and the proposal modules follow the same rules and can be modeled with a similar mathematical approach and a common methodology.

1.1 Request-Prediction

Let us first have a closer look at Request-Prediction (RP). This is a very exiting topic. We try to foresee things that might happen in the future. People often have the problem that they could behave more efficiently if they were able to know what the future would bring. For instance, it would make a man rich, if he could foresee the numbers of the upcoming Saturday-lottery. Or let us take the weather forecast: it is very important for certain areas to know the weather of the future, but due to the

chaotic behavior of the climate only a few days can be computed with enormous effort.

In part II of this work we try to answer the question: which documents will the user request in the near future? Or, more specific: what are the probabilities of future user requests?

To answer these questions, we do need a mathematical model. This is a typical means of proceeding. Most of the interesting[1] problems that the Computer Science is able to solve have to be transferred into formal representations.

For the RP we were looking for a rather simple model as a result of the following consideration: if the model is simple it is also - hopefully - easy, fast and efficiently to implement and therefore, may soon be tested. So we decided to start with the evaluation of a simple model and refine it stepwise instead of constructing a rather complex model in the beginning. This serves only the overall aim: we want to find algorithms for practical improvements of web-applications. Proceeding this manner, we followed the famous saying (cited from [CB95]):

Keep it simple! If it's complex, it's probably wrong.

There is another very important requirement for our model: it should not only be simple but also easy extensible and, thus, very flexible. It is clear that we used a rather straightforward mathematical model without considering possible implementation difficulties. For example, the central data-structure of our RP-model is - firstly - a symmetric matrix. Certainly, nobody would ever implement a symmetric matrix without sparing nearly half the memory by changing the representation so that double entries were only stored once.[2]

The core idea of the model is based on the consideration: RP is possible because the request-behavior of the users is not simply random but - at least partially - computable from former request-behavior. In chapter 4 we will find additional requirements for RP to become appropriate.

The simplicity of the model forced us to use a rather ordinary statistical approach. One the one hand, we calculate conditional probabilities from the relative frequencies of the former requests. On the other, we compute the individual cost thresholds depending on network and system load and the costs of the prediction-algorithm itself. Finally, we compare the results from both sides to decide whether prediction is - currently or in general - for the server a good idea or not.

But to decide this question correctly, it is not sufficient at all to have an adequate model. The next step is to find an appropriate algorithm derived from the model. Next, this algorithm has to be tested. In the area of prediction the testing is very hard. To be able to measure the quality of the algorithm we could simply run RP on a test-server, but the results would not be representative. Either the request-numbers would be too low or not randomly chosen. Then, the evaluation results would not be true for the general case. Therefore, we modeled the user request-behavior in an additional test-scenario. Presumably, this modeling is much more interesting than the modeling of the prediction-algorithm itself. Within our own test-scenario with

[1] In the sense that these problems are not too simple.

[2] Of course, there could be other constraints to consider so that it might make sense to store most of the matrix elements twice.

generated (virtual-) user-requests, it is possible to start millions of test-runs and the general usability of the result depends only on the universality of the request-modeling.

How can one imagine the modeling of those requests? In fact, there are two extreme positions: either the requests are quite random or completely foreseeable.[3] So it is important that the parameters of the request-generation allows both extreme request-models and - smoothly - everything in between. The general usability of the RP, thus, can be fixed depending on the randomness-degree of the requests. In section 5.1 we will speak here of *semi-randomness*. In practice, the algorithm must be able to provide indicators that help to find out if the use of prediction does make sense in a specific context. Request-behavior depends on several factors, e.g. the kind of provided data on the server-side, the clientele of users that poses requests to the server, or external events. For instance, company messages can influence stock-data values and thus lead to higher system-loads on stock-data servers.

To be honest, it is not sufficient to have a single parameter to model user-request behavior. In fact, we do need several additional parameters. There is, for instance, the *density*. This factor describes the average frequency of requests within a session. Furthermore, for the advanced modeling of RP (section 6.1), we have to provide a *request-change-probability* to find evaluation results for the modeling of time and document aging.

However, it is not enough to model the RP and test it within a test-scenario. How can we be sure that the modeling has been correct? Certainly, the model of the user-behavior is straightforward and clear, but an insecurity remains. Therefore, we decided to test the algorithm with aid of real request-logs *a posteriori*. Even though the RP is modeled for use in the Smart Data Server (described in section 2.1) - a universally usable server for web-applications that has been designed originally as stock-data server - it can also be applied to classical web-servers.[4] Our RP-modeling does not consider possibly existing hyperlinks that change the a priori probabilities of subsequent requests but works for general data-requests. By the way, this is the reason why we did not model RP as kind of path-profiling (more information on that topic can be found in section 4.2).

To additionally verify the RP-modeling, we took real web-server logs, transferred the content into an abstract representation and fed the test-scenario with the real logs as if they were generated artificially! The evaluation results were very promising, but we got supplementary information: we could extract the virtual parameter setting of the real-logs, the randomness, the density and so forth. We will analyze those details in chapter 6.

1.2 Hyperlink-Proposals

The second focus of this works lays on proposing hyperlinks as a tool for online authors to enhance their texts. Hyperlinks are the key structure of the WWW and one main reason for the enormous success of the Internet. They improve substantially the quality of hypertexts. With hyperlinks it is easy to explore the diverse information sources of

[3] In the sense that always the same requests during the sessions take place.
[4] Providing [HTTP]

the web.

Hyperlinks provide a kind of *coherence* between two hypertexts.[5] In reality, a lot of coherence or similarities might exist between texts because there are potentially manifold relations between the (possibly abstract) objects described by them.

In general, hyperlinks suggest a similarity between two documents depending on - at least - one property. But the world consists of several different categories of properties so that hyperlinks form a multi-dimensional network between HTML-pages.

Let us consider the following example. An HTML-page about number theory might contain links to sites describing the generation of prime numbers. This would be a kind of "topic-coherence". Additionally, other links might point to further pages of the same author (with different content). Such hyperlinks would denote a "personal-coherence". This is one reason why it is very hard to provide high-quality proposals for hyperlinks of a given text.

In general, we distinguish between two different approaches for suggesting links, the *semantic model approach* and the *statistical approach*. The former one tries to model the real world by building object-networks with semantic data while the latter one reduces the content of a text to a few statistical values.

We chose the second approach because it is much easier to retrieve information on statistical base and this process can be achieved automatically by computer programs. Certainly, the proposal-quality of the semantic model approach is much better, but this kind of processing requires a good deal more user interaction.

We live in a world - especially concerning web-applications - where everything has to run very fast. It is almost impractical to administer the semantic network in time. Even a company can presumably not manage to keep its semantic model up to date. Information flows too fast and in too large an amount. If it takes too long to generate the semantic model it might not be quite appropriate when it is finished or - even worse - the company might not exist anymore!

For the statistical approach, the step of retrieving relevant information from the texts is very important. Usually, the knowledge is abbreviated to weighted keywords, authors names, expiration date and other kind of meta-data. HTML-files provide the possibility to distinguish between title, headline, body and meta-tags so that the existing keywords can be weighted according to their position within the document. Section 8.1 will discuss the information retrieval in detail.

Starting from the (statistical) "information-essence" of the hypertexts the question remains how to propose acceptable hyperlinks for those texts? It would have been possible to model this process with the aid of *Neural Nets*. We decided to go another way by regarding the process to find hyperlinks as a kind of *Case-Based Reasoning* (CBR)[6].

In CBR, expert-knowledge is represented in form of a case-base, where selected *cases* are stored. Cases are represented as problems together with their solutions. A new problem can thus be solved by finding the most similar case in the case-base and transferring its solution to the solution of the new problem. This is a kind of *classification*. The process of storing new cases into the case-base can be regarded as *learning*.

[5] We do not focus here on the diverse links to CGI-scripts, images or other targets.

[6] CBR will be explained in section 3.1.

From our point of view, CBR represents the expert knowledge clearer and in a form which is better to verify than neural networks do. Furthermore, the basic structures and elements of CBR - at least for our modified version - are fast and efficient to implement.

The problem of CBR is that the similarity between cases is very important. A reasonable maintenance of the case-base is hard to provide in the area of proposing hyperlinks due to the fast growing and strongly varying contents of the data sources. The amount of information is hard to handle and for the most part there is not enough time to select the best representing cases to store them within the case-base.

Therefore, we decided to only rudimentarily transfer the concepts of CBR to model our hyperlink-proposal module (HPM).[7]

This seems - perhaps - to be appropriate because we are not diagnosing human diseases[8] but we only want to provide link proposals ordered by their probability of usability for a concrete context.

The results might not be as precise as classical CBR could yield, but our approach is - as we argued for the RP-model - very flexible and easy to extend. Furthermore, due to the concept of dynamically increasing matrices we can avoid some of the CBR-difficulties of our simplification.[9] Nevertheless, especially for rather small websites, one can expect quite convenient proposal results using the methodology that will be presented later on.

The idea to regard the main tasks of the HPM as learning and classifying problems resembles the view of the RP approach. In chapter 10, we will describe some synergy-effects between the RP and the HPM. Furthermore, those concepts are also transferable to other web-applications.

In the following chapter 3, we present some standard web-applications that are the basis for the discussed research areas of RP and HPM.

Part II of this work presents the derivation, implementation and improvements of the RP approach. Part III deals with the aspects all around the HPM. Part IV contains the synergy-results and the final summary and outlook chapter.

Some concrete programming code is listed in the appendices of part V (Appendix-A for the RP and Appendix-B for the HPM, respectively).

[7] The design and the key elements of this HPM can be found in section 7.2, while 7.3 outlines the main differences between our model and CBR.

[8] Diagnose-systems are a famous operational area of CBR.

[9] See 8.2 for details.

Chapter 2

Advanced Internet Applications

Measure what is measurable,
and make measurable what is not so.

Galileo Galilei, Quoted by I. Gordonand and S. Sorkin,
"The Armchair Science Reader" New York (1959)

In this chapter, we will highlight briefly some general concepts of advanced Internet applications as a platform for the prediction-theory (part II) and the hyperlink-proposal techniques (part III).

At first, we will describe the architecture of a complex data server to improve the information flow for the Internet. Basically, the derived prediction formulas of chapter 4 are evolved for server types like the one described in 2.1.

The second section, then, places the emphasis on an online authoring system developed at the Institute of Telematics to provide an appropriate methodology for supporting web authors. It is necessary to describe the functionality of such systems to comprehend the handling of hyperlink-proposal tools (section 2.2).

Finally, we will present in section 2.3 the working environment with hyperlinks and we will sketch a concrete hyperlink-management system as basis for the hyperlink-proposal module of section 7.2.

2.1 The Smart Data Server

2.1.1 Properties of the SDS

Due to the very fast growing of the Internet, especially the World Wide Web (WWW), there is also an increasing amount of data and information flow between several servers and clients. Therefore, also the need for central systems to collect and distribute information from heterogeneous data sources is growing.

The "Smart Data Server" (SDS) as a general framework for distributed functionality has been constructed to fulfill these needs. The SDS is able to connect different

data sources and improve information exchange. The system serves as middle tier in a three tier architecture [RHE99b]. It can work together with several C/S components and structures and is a pure Java implementation. The overall idea is to put intelligence on the server-side to increase the efficiency of the communication with the clients.

Similar approaches can be found in Satoshi Hirona's HORB, an object-oriented request-broker [Dua96]. Also the idea of the common request broker architecture (CORBA) is related [OMG]. In this approach, several resources can be distributed over the network. DCOM [Bro97] and Java's RMI [SUN] also resemble the basic ideas of the SDS.

In general, the main concepts of the SDS can be summarized as:

- *Modularity*

  The SDS consists of three different classes of modules. *Action-layer modules* and *service-layer modules* belong to the core of the SDS. The *function-layer modules* carry out the different application tasks of the system. They work independently from each other and can be exchanged easily.

  If the SDS operates as a portfolio-management system[1] or as a medical server is decided by the different function-layer modules.

- *Flexibility*

  The SDS is designed as a very flexible system to connect several different databases. It can be regarded as a kind of *universal adapter* where the different data structures come together with minimal effort. Instead of designing end-to-end adapters for each data-format, it is sufficient to build one adapter for the SDS of each of the corresponding databases.

  Furthermore, the SDS comes with its own database service-layer module, so that it is - basically - not restricted to relational databases.

- *Scalability*

  A very exiting property of the SDS is its scalability. If the system load reaches critical values, one possibility is to clone the SDS and thus provide additional computation power. While, normally, servers are only scalable by providing machines of higher performance or other kind of hardware or operation system architectures the SDS provides additional possibilities.

  Its communication language allows to simply transfer requests to other SDS (e.g. clones of itself) so that their computing power can be used to reduce the latency for the server response.

  Additionally, this flexibility helps to overcome problems of geographic or topologic distances between databases or even security problems.

More information about this topic can also be found in [RHE99a].

2.1.2 Optimizing Communication Performance

The SDS as a platform for distributed functionality provides several possibilities to reduce user perceived latency.

We will briefly discuss three general techniques to improve communication and data exchange between clients and servers as conceptually designed for the SDS.

[1] Originally, the SDS has been developed as a controlling system in the financial sector. Later on, its general conception could be extended to a multitude of other application areas.

- *Prediction*

  There are several possibilities to overcome the problem of high user perceived latencies after requesting a document. One idea is to predict future requests. Thus, time-consuming calculations on the server-side can be done before a request is made. If the server is very "sure" that certain documents will be requested in the near future, the corresponding data can be sent to the client (or pre-fetched by the client) even though the user does not recognize it.

- *Minimization*

  To overcome the disadvantages of wrongly predicted user requests the SDS provides a scheme to reduce the transferred data by investigating superfluous or redundant data.

  This idea is reversed to a general trend of Client/Server-communication. While normally two- and three-tier architectures [Dic95] in opposition to central computing often transfer a maximum of data from the server to the client for future manipulating and working on it, it can be sufficient - especially for the Internet usage - to do the opposite. Sending only exactly the data the client can use should improve C/S-communication and reduce network load and - as prediction should do - user perceived latency.

  Therefore, this method is based on an idea opposite to prediction: instead of sending more data than requested, it could be useful - in some cases - to send less!

  For instance, the display resolution of the client hardware might be a strong limitation for the amount of data the server should return. If there is no further use of the information it would be senseless to send more data than the client can present. Certainly, the two methods are equivalent: the client tells the server either how much data can be evaluated or what resolution can be found on its side, in both cases the "smart" server has to work on the data to return a minimal amount of values.

- *Request Optimization*

  Request Optimization is a method to improve C/S-communication that depends highly on the type and structure of the transferred data. The implementation as module of the SDS, therefore, is explained here only exemplary as part of a stock-data services application.

  To be able to fulfill very different client requirements, the SDS provides several functions with optional features. For instance, moving average trend indicators, a "Relative Strength Index" (RSI), a "Moving Average Convergence/Divergence" system (MACD) and "Bollinger Bands" can all be calculated on server-side or on client-side, respectively.

  Request Optimization in this context means that it should be decided dynamically whether the server or the client calculates the trend indicators. On the one hand, the well-known data lack of the first 38 days of the 38-day moving average is a result of the client-side calculation while the server-side can response with all requested data. On the other hand, it would be superfluous to start an SDS-request, for instance an MACD trend indicator, that could simply be calculated from the stock-data values already available on the client-side.

For more information on the two latter communication optimization strategies for the SDS we refer to [HRE00b].

Certainly, the prediction aspects will be discussed in detail in part II of this work. In chapter 4, the used prediction theory is derived. Chapter 5 provides the concrete implementation of the concepts and its evaluation in general. Finally, in chapter 6 advanced models and concepts with modeling of time and document aging are discussed.

2.2 Online Authoring Systems

2.2.1 General Requirements

With the growing of the number of web-pages on the Internet also distributed online authoring and publishing, especially in enterprises, is gaining more and more importance. In particular, the WWW has provided the protocol and thus pushed the infrastructure that is needed for realizing the ambitious visions. Now, it is possible for users to perform online collaborative authoring directly from their workplace (see for instance [LZO98] or [MPS98]).

Besides the use of editing web-pages, online authoring systems can also be applied to intranets of companies. Not seldom, they can play the role of a document management system.

The critical aspects of such systems come to the surface depending on the (large) amount of diverse (documentation) information sources and on complex and manifold role structures of users.

Another important aspect of online authoring systems is the separation of content and layout. Several authors should provide the relevant information to the system and by using templates or style sheets the system must produce acceptable HTML-results, statically or dynamically.

Furthermore, a strict role-based access control is very important to guarantee high quality results and at the same time information security is granted [Wan99].

2.2.2 DAPHNE

2.2.2.1 General Characteristics

At the Institute of Telematics a web-based online authoring and publishing system, called DAPHNE (*Distributed Authoring and Publishing in a Hypertext and Networked Environment*), has been developed as a modern tool to provide information based on Internet-technology. DAPHNE is a distributed collaborative authoring and publishing tool [HZE99]. The system offers the following operating features:

- DAPHNE differs from other approaches or other web authoring publishing systems by - under most circumstances - employing strictly standardized and open hypertext technology [OW96]. Components that incorporate standard applications such as word processors or Internet services into DAPHNE have been developed
- Authoring and publishing of multilingual documents is supported by the online authoring system
- DAPHNE allows the structuring of various information sources by employing a structural element, i.e. departmental information
- It offers a workflow with access control: various roles are supported by DAPHNE and each role is assigned with a fixed set of actions; for instance, content examiners can examine and approve documents. By means of assigning actions and departmental information to roles an efficient workflow system with access control mechanism has been built (see also [ZHH99])
- DAPHNE can assure a common layout for all documents of a web site
- The system is "open", i.e. DAPHNE is designed to work with documents in any formats and for multiple purposes (web publishing, document management)
- It is very flexible, i.e. users can use any software they prefer to generate the content for documents that are processed by DAPHNE

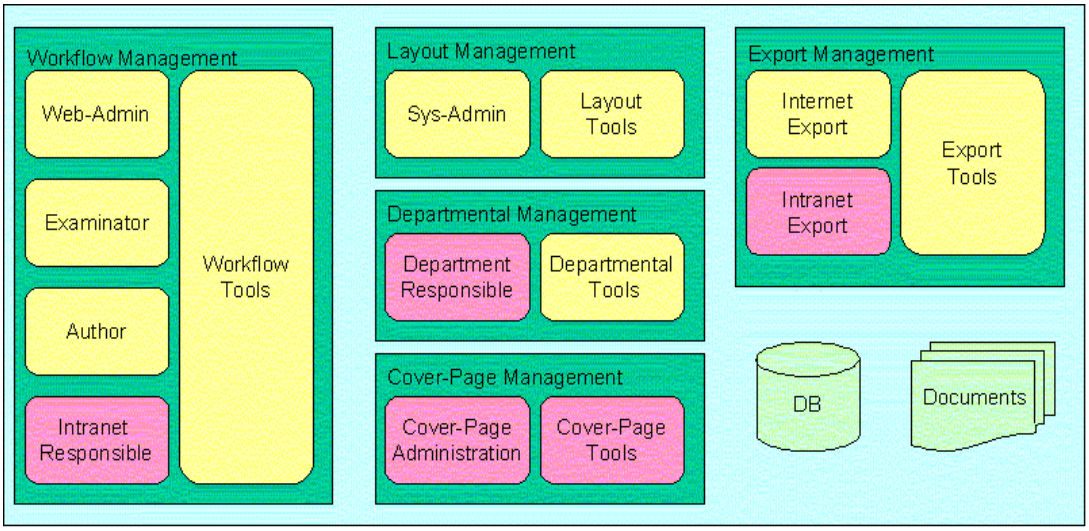

Figure 2.1: System architecture of DAPHNE

2.2.2.2 System Architecture

DAPHNE is a web-based system with Client-Server structure. On the server-side, DAPHNE employs a database to manage dynamical text and image documents with several status values. Basically, web browsers are used as clients.

For document processing (e.g., writing, modifying, etc.), authors can choose any commercial software they prefer by configuring the application MIME/TYPE in the web browsers. The communication within DAPHNE between server and client is based on HTTP[2]. An authentication mechanism restricts login into the system and work with it to authorized users only. The facilities of the system depend on the role of the user. The system architecture of DAPHNE is illustrated by Figure 2.1.

DAPHNE server programs are mostly CGI-programs. The system stores meta data of documents, user data, and further information in a relational database, while documents themselves are managed by the file system. The main functions of DAPHNE's server programs include user management and access control, data parsing/metadata collecting, file uploading, editing, substitution, delete, content examination, interface generating, layout management and HTML-generation.

Based on its system architecture and the basic functions, DAPHNE is capable of supporting distributed web authoring and publishing.

Detailed information about the online authoring tool DAPHNE can be found in [HZE99]. Further information about the role-based access control is provided in [ZHH99].

2.3 Hyperlink Management

2.3.1 General Considerations

Since the introduction of the WWW-service to the Internet in the early 90's, the popularity of the "Net of nets" is growing rapidely. A main reason for this growth is the user-friendly

[2] **H**yper**t**ext **T**ransfer **P**rotocol, [HTTP]

possibility to explore the World Wide Web by simply clicking on hyperlinks within the HTML-pages presented by the browsers.

Therefore, the role of the links is very important. The term *Hypertext* itself can be defined as follows:

> *Hypertext is a database that has active cross-references and allows the reader to 'jump' to other parts of the database as desired.*

This definition originates from Shneidermann. Further information about his view can be found in [Shn89].

It is necessary to provide practical usable tools for generating hypertext. An early example of such a hypertext-system is *Xanadu* from Ted Nelson [XAN].

Besides the well-known standard of hypertexts, also other approaches are possible. For instance *HyTime*[3], a norm (ISO/IEC 10744) that defines a technical standard for open Hypermedia [HyT].

2.3.2 Concepts of ^{M}HLM

The Institute of Telematics developed a system for managing hyperlinks as part of a project supported by the "Stiftung für Innovation, Rheinland-Pfalz".

It was the aim of the project to provide a system that allows the flexible administration of hyperlinks for multi-lingual documents. We called this system the *Multi-lingual Hyperlink-Management System*, ^{M}HLM.

Below, we present some essential features and concepts of ^{M}HLM for providing hyperlink administration. Detailed information about this topic can be found in a technical report of the Institute of Telematics [RHH99].

The main data-structure of the ^{M}HML is the *PURI*. The core and central identifier of a document is its *Uniform Resource Identifier*, short URI. We extended this sight by adding specific properties that we called *peculiarities*. Thus, a document is identified by the URI and several peculiarities. The common data-structure is, therefore, the PURI.

Examples of kinds of peculiarities are listed below.

- *Language*

  A document can be provided in several languages. To be able to identify two files that provide the same content in different languages, we decided to model the language as a peculiarity of the document.

- *Status*

  As described in 2.2.2, the reviewing-status of a document can contain several steps. An author may generate a file that is examined by the department manager. Only if he accepts the document, it is transferred to the web-administrator of the company. Finally, it is exported to the Internet.

  To model the different states of a text we regard the status as a peculiarity of the document.

- *Version*

  Certainly, the content of a document might become - partially or in the whole - obsolete. The concept of the PURIs allows to model the version of a document as peculiarity, so that different versions of the same text have the URI in common, but differs - at least - in their version peculiarity.

[3] **Hy**permedia/**Time**-based structuring language, [HyT]

The ^{M}HLM is a pure Java implementation where the PURIs are defined as vectors that contain the URI as one attribute.

To manage hyperlinks it is necessary to model relationships between PURIs. So we defined a link as a data-structure that provides a source and a target PURI. Additionally, a default-description (the *link-label*) and some further technical elements are attached, for instance the specific position of the link within a document or a field to denote whether the link is broken, i.e. the target PURI is not valid any more.

Three important tasks of a hyperlink-management system in general and of the ^{M}HLM in specific are listed below.

- *Link-Consistency*

  To provide consistency of hyperlinks within a website is a very straining job. Every move of a document or deletion assumes that the corresponding links within all other pages and within the moved page are changed so that they remain consistent.

  The PURIs of the ^{M}HLM provide excellent prerequisites to change documents and verify the consistency of the links.

  We have to distinguish between *internal* and *external* links[4]. For the former ones, a very specific consistency check can be made and both, target and source, might be changed. For the latter ones, the link verification consists in checking the existence of the external documents.

  Additional difficulties arise for the multi-lingual case: a link-change in a document entails that all documents with same content in different languages, i.e. documents with the same URI and common peculiarities besides the language-peculiarity, are changed according to the first one. Here, it is very important to check whether the target-link points to a document of the same language or not. In the first case, for different link-changes the target-document has to be transferred to the corresponding document of another language to provide multi-lingual consistency.

- *Generation of Navigational Links*

  The automatic generation of navigational links within a website is mostly the task of an online authoring system (described in 2.2).

  Again, changes concerning the hyperlinks of a document or the moving of a file to a different location can cause several link adaptations of other documents. Therefore, ^{M}HLM provides a multitude of tools to provide navigational link generation on the base of additional data, e.g. departmental information.

- *Link-Proposals*

  Perhaps the most difficult task of a hyperlink-management system consists of proposing high quality links for the web-author of a text. Mostly, an author does not know the complete website or appropriate external links so that it would improve the quality of the produced hypertexts enormously if a hyperlink-proposal module (HPM) would generate suggestions.

  We will analyze this task in detail in part III of this work where we also present a solution strategy and a concrete implementation of an HPM together with its evaluation.

Basically, we generated the hyperlink-proposal module described in chapter 8 as part of the ^{M}HLM. Certainly, also the usage for the online authoring system DAPHNE (2.2.2) is appropriate.

[4]Internal links point to documents within the same domain while external links point to different domains not under "one's own" control.

Chapter 3

Theoretical Basics and Concepts

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

Alan Turing, from his paper on the "Turing test"
The Bell Labs Cafeteria, New York (1943)

3.1 Case-Based Reasoning

3.1.1 CBR Terminology

Research in the area of Case-Based Reasoning[1] (CBR) began in the early years of the last decade [Aha91]. CBR-systems are well known means of representing knowledge in form of *cases*. Each case can be regarded as a *problem* together with its *solution*. A problem consists of its description in form of *attributes* and one or more solutions which refer to it. A typical environment of CBR is the area of *diagnostics*. Here, the attributes are the *symptoms* and the solution is the *diagnosis* [AW92b] (see also [Ric92] for further information and [RW91] for a detailed analysis of the PATDEX/2 CBR-system).

In general, CBR-systems store their case in a knowledge database called *case- base*. To solve a new problem, CBR-systems try to find the most similar cases in the case-base. Next, the solutions of this result are transferred to the new problem or are simply regarded as solutions of it. The former systems are called *case-adaptation systems* while the latter ones are denoted as *case-matching systems*.

If we consider the set of problems $P = \{P_1, P_2, \ldots P_n\}$ as *problem-base* and the set of solutions $S = \{S_1, S_2, \ldots S_m\}$ as *solution classes*, then the process to associate one or more solutions $S_{i_1}, S_{i_2}, \ldots$ to a given problem P_i can be regarded as a *classification process*.

In general, a case is a 2-tuple $C = (P, \vec{s})$ of a problem description and a (mostly binary) vector of solutions.

Remarkable attempts have been made to find out how to store only really usable cases (to avoid storage overflow) and how to learn to adapt the rules to compare cases for calculating their similarities [Joh97].

[1] Certain methods and algorithms are also known as *Case-Based Learning* (CBL). For simplicity, we will always speak of CBR in this elaboration.

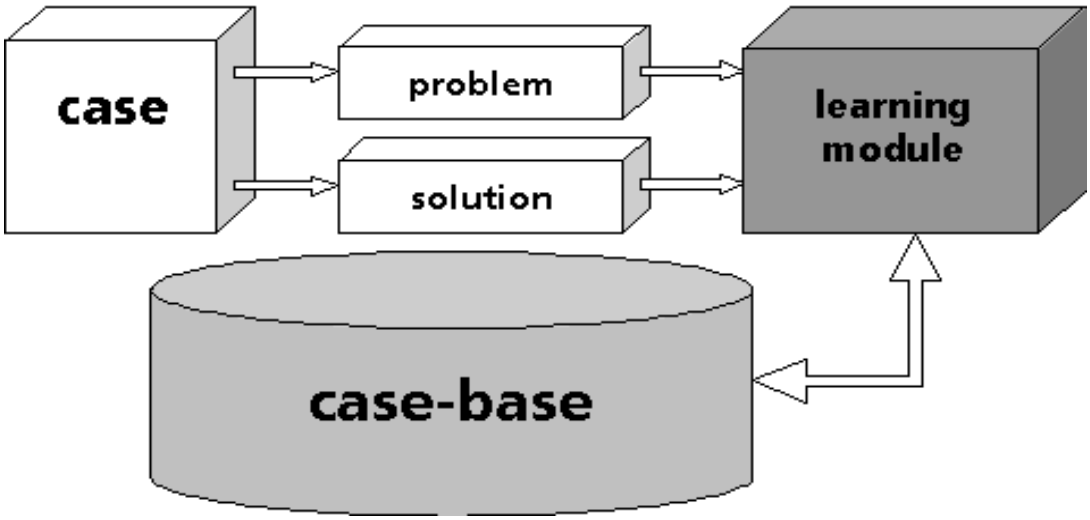

Figure 3.1: Learning phase of CBR-systems

CBR work can be divided into two different phases. The first process, the *learning phase*, builds up the case-base with reasonable cases, e.g. problems together with their solutions (see Figure 3.1). The quality of the resulting case-base is better particularly after the learning phase if the corresponding cases cover the scope of the problem.

The second process, the *classification phase*, compares a new problem with the existing problems of the cases in the case-base. The solution of the most similar case found is a good proposition for a solution of the new problem (illustrated by Figure 3.2).

In practice, both phases are combined. The learning of new cases (new *knowledge*) will continue as long as the (real) solutions of formerly posed problems are being recognized. A good overview on CBR-principles can be found in [KL95].

3.1.2 The CBR Algorithm and Some Refinements

The simplest method to represent expert-knowledge with CBR can be achieved by the following processing, the *Simple-CBR*[2]:

1. Storing cases for training into an empty case-base CB
2. Classify a new problem P_i by simply finding the most similar problem of a case C within the case-base CB and
3. Take the solutions of the corresponding case C as solutions for P_i

Therefore, it is necessary to find and learn good training-cases first. It is seldom a good idea to put every new case into the case-base. On the one hand it is difficult and inefficient to store too much cases. On the other hand the learning effect might be critical: a classification of a new case could probably change the classification of formerly presented cases and thus change the learning process.[3]

The disadvantages of the Simple-CBR algorithm are summarized below:

[2]The Simple-CBR is also called *Case-Based Learning Algorithm 1 (CBL1)*. For details see [Haf93]

[3]... and who knows if the new classification might be better or worse than the old one?

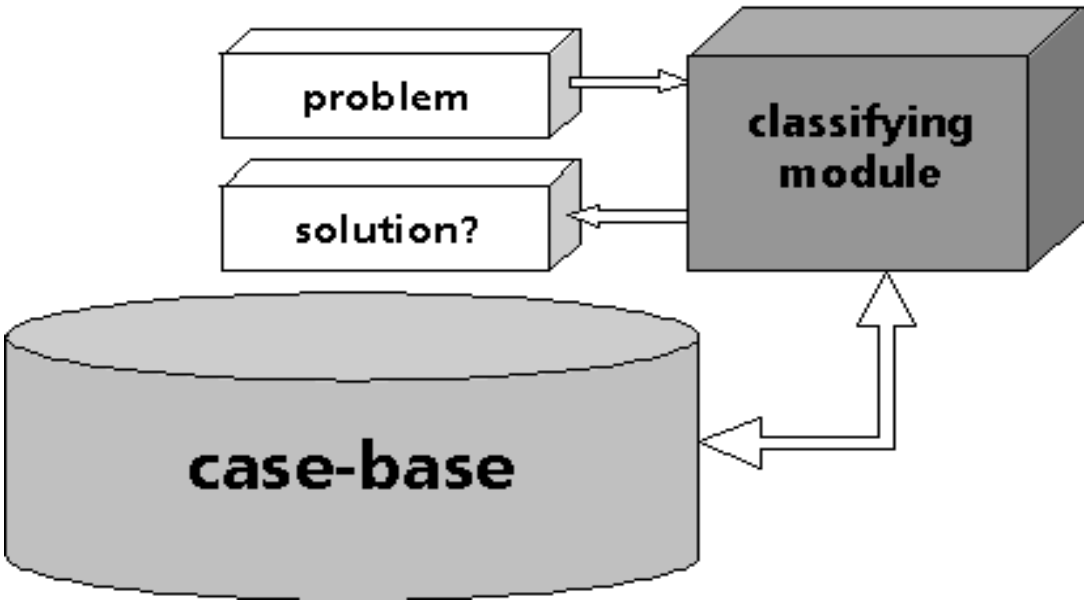

Figure 3.2: Classifying phase of CBR-systems

- The Simple-CBR is storage-intensive and time-consuming due to the storage of all training-cases
- The algorithm is not able to treat erroneous data appropriately: every new case can change the whole classification process
- Simple-CBR can not distinguish between important and irrelevant attributes of problems
- The quality of the algorithm depends on the quality of the problem-comparison mechanism, the *similarity measurements*

Furthermore, David Aha showed that non-numeric attributes are hard to model and that the expert-knowledge stored in form of cases in the case-base is not represented appropriately [Aha91].

Several improvements have been made to overcome the difficulties discussed above. The most important changes are the heuristics to store only the "usable" cases in the case-base within the training-phase. Also the introduction of the *Relevance Matrix* to represent the importance of certain problem attributes for the classification to the concerning solutions was a critical change in the CBR algorithm (see also [Sal88]).

We will come back to the Relevance Matrix in section 3.1.3.2.

3.1.3 The Ratio Model

3.1.3.1 Discussion of similarity

In section 3.1.1 we mentioned the importance of the similarity-measurement between different problems. But what is *similarity*?

In the late 70's the classical mathematical similarity-measurements have been criticized. The canonical Cartesian comprehension of similarity as a kind of geometrical distance between

objects in space[4] posed many problems concerning the psychological view on similarity.

Amos Tverski showed that the three fundamental properties of the geometrical understanding of similarity could not be fulfilled psychologically [Tve77]:

- *Reflexivity*: The similarity between an object and itself is not always "1" if the probability of re-identification of the object is not 100 percent. In the real world, an object can hardly ever been identified without any doubt.
- *Symmetry*: A son may be similar to his father but not vice versa! The psychological similarity is seldom symmetric.
- *Transitivity*: Even though the playing power of a chess-player 1 may be similar to another player 2 and his/her strength may be similar to a third one, it is very improbable that also player 1 and player 3 mostly draw their games. Similarity is too complex to be simply transitive, at least from the viewpoint of psychology.

To overcome these troubles, Tverski proposes *Feature-Matching* as an adequate method of comparing objects.

Every object O_x can be represented by a set of *attributes* A_x. The class of similarity functions to measure the similarity between two objects O_x and O_y must fulfill the two conditions below (whereas the function F simply weights the cardinality of its three parameter sets and A_x, A_y, A_z denote the attribute sets of the corresponding objects O_x, O_y, O_z):

1. **Matching**: $sim(O_x, O_y) = F(A_x \cap A_y, A_x \setminus A_y, A_y \setminus A_x)$
2. **Monotony**: $sim(O_x, O_y) \geq sim(O_x, O_z) \iff$
   - $A_x \cap A_y \supseteq A_x \cap A_z$
   - $A_x \setminus A_y \subseteq A_x \setminus A_z$
   - $A_y \setminus A_x \subseteq A_z \setminus A_x$

One advantage of this modeling is a general usability of the similarity function: Not only numerical representations of object attributes can be treated but also every attribute description as set of elements can be modeled (i.e. a *symbolic representation*).

The following function sim_R would be a good candidate to model similarity adequately and thus fulfilling the requirements above:

$$sim_R(O_x, O_y) = \frac{f(A_x \cap A_y)}{f(A_x \cap A_y) + \alpha f(A_x \setminus A_y) + \beta(A_y \setminus A_x)} \quad \alpha, \beta \geq 0, \alpha + \beta > 0 \tag{3.1}$$

for all classes of "natural" functions f. According to Amos Tverski equation (3.1) is called the *ratio model* [Tve77].

In the following, we will use a slightly simplified version of sim_R (3.1):

$$sim_S(O_x, O_y) = \frac{\alpha f(A_x \cap A_y)}{\alpha f(A_x \cap A_y) + \beta f(A_x \setminus A_y \cup A_y \setminus A_x)} \quad \alpha > 0, \beta \geq 0 \tag{3.2}$$

To be able to decide whether an attribute $O^j \in A_x \cap A_y$ or not for numeric valued attribute representations it would not be appropriate to use simply the mathematical *equality*.

If the attribute might be "heartbeat-frequency" than the values "98.7" and "99.1" might be rather similar and thus belong to the set of common attributes $A_x \cap A_y$. But this would not be true if these values denoted the date of income for the tax declaration.

[4] Mathematicians call this structure a *metric*.

Nevertheless, as a possible attribute-comparison function η to calculate the equality of two numeric attribute-values O_x^j and O_y^j where N denotes the number of different attribute-classes:

$$\forall j \in \{1, 2, \ldots \mathcal{N}\} : O_x^j \in A_x \wedge O_y^j \in A_y$$

we can choose the straightforward settings of equation (3.3). All numeric modeled attributes ranges from zero to an attribute dependent maximum M^j.

$$O^j \in [0 \ldots M^j] \quad \text{or} \quad O^j \in \{0, 1, \ldots, M^j\} :$$

$$\eta(O_x^j, O_y^j) := 1 - \frac{|O_x^j - O_y^j|}{M^j + 1} \tag{3.3}$$

If O_x^j and O_y^j are equal then $\eta(O_x^j, O_y^j) = 1$ and $\eta(O_x^j, O_y^j)$ becomes rather small for more different attribute values.

Additionally, a threshold δ^j is needed for every attribute class to decide whether the similarity between the settings of O_x^j and O_y^j belongs to the set of common attributes of O_x and O_y, i.e. $O_x^j, O_y^j \in A_x \cap A_y$ or not.

This is extremely important. Otherwise, distinguishing between equal attributes and others in the ratio model (3.2) can not be accomplished.

Only similarity values exceeding δ^j are counted as "equal", the others as "different". We use the term $\mathcal{F}$ for *fulfilled attributes* and $\mathcal{C}$ for *contradictory attributes* as shown in (3.4):

$$\forall j \in \{1, 2, \ldots \mathcal{N}\} : \begin{array}{lll} O_x^j, O_y^j \in \mathcal{F} := A_x \cap A_y & \Longleftrightarrow & \eta(O_x^j, O_y^j) \geq \delta^j \\ O_x^j, O_y^j \in \mathcal{C} := (A_x \cup A_y) \setminus \mathcal{F} & \Longleftrightarrow & \eta(O_x^j, O_y^j) < \delta^j \end{array} \tag{3.4}$$

3.1.3.2 Advanced CBR-algorithm

In the case of numerical representation of attributes it is also appropriate to weight the relevance of every attribute for the classification to all of the solutions. This leads us to the Relevance Matrix ω already mentioned in 3.1.2.

The $\mathcal{M}$ rows belong to the several solutions known within the case-base and the $\mathcal{N}$ columns are corresponding to the different attribute properties.

$$\omega = \begin{pmatrix} \omega_{11} & \omega_{12} & \cdots & \omega_{1\mathcal{N}} \\ \omega_{21} & \omega_{22} & \cdots & \omega_{2\mathcal{N}} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{\mathcal{M}1} & \omega_{\mathcal{M}2} & \cdots & \omega_{\mathcal{M}\mathcal{N}} \end{pmatrix}$$

ω fulfills the condition:[5]

$$\forall i \in \{1, 2, \ldots, \mathcal{M}\} : \sum_{j=1}^{\mathcal{N}} \omega_{ij} = 1 \quad \wedge \quad \omega_{ij} \geq 0 \tag{3.5}$$

Every ω_{ij} represents the importance of attribute j for the classification to the solution i.

With the relevance matrix ω we can refine the Simple-CBR approach mentioned in 3.1.2 to a usable CBR-algorithm, the *Advanced-CBR*:

[5] In 3.2.1 we will call this property *stochastic*.

1. (a) Initializing the relevance weights of ω to:

$$\forall i \in \{1, 2, \ldots \mathcal{M}\}, \forall j \in \{1, 2, \ldots \mathcal{N}\} : \omega_{ij} = \frac{1}{M^j + 1}$$

   (b) Storing the training-cases into the empty case-base CB and adapting the weights of ω so that all training-cases are classified correctly. This step must be repeated for all training-cases![a]

2. Classify a new problem P_i by finding the most similar problem of a case C within the case-base CB considering the attribute similarities computed by the function η and the relevance weights of the attributes

3. Take the solutions of the corresponding case C as solutions for P_i

[a] This is a very critical process. In general it works pretty well but for several application areas the training steps must be limited to avoid running into endless loops; in [Haf93] a detailed description of such problems and possibilities to overcome them are provided.

Certainly, the relevance weights together with the case-base as used in the refined approach are modeling expert knowledge much more appropriate than the Simple-CBR approach of section 3.1.2 did.

3.1.4 Learning Parameters of CBR

According to the ratio model presented in section 3.1.3 and the description of the numeric attribute calculation and weighting by the relevance matrix ω we can see several parameters that can be learnt during the learning phase of the refined CBR-algorithm.

First of all, the relevance adaptations themselves can be regarded as a learning step. During the training-phase at the very beginning of Case-Based Reasoning there are several weight adaptations. In section 3.1.5 we will come to a general derivation of the adaptation formulas for the relevance weights.

Then, the parameters α and β of the (slightly simplified) ratio model (3.2) are subject of learning considerations.

Furthermore, the thresholds δ^i to decide whether the differences between two numerical attributes are small enough to be regarded as belonging to the set of common attributes of two objects are changeable.

It is even more important than the thresholds δ^j to determine the similarity between two attribute values of the same attribute class j are the thresholds δ_i for every solution class. These values represent the importance of the similarity comparison to accept the solution proposal of the corresponding classes $i \in \{1, 2, \ldots \mathcal{M}\}$. Only similarity-comparisons that exceeds the threshold can be regarded as "learnt". In section 3.1.5 we will derive formulas to determine acceptable weight changes of ω depending on δ_i.

An adaptation of the threshold values δ_i themselves has crucial effects on the learning speed and the learning quality. A detailed analysis of this kind of impact can be found in [Haf93].

The most important rule of the CBR-learning step in our context is a result of common sense and longtime practical experience:

CBR-Rule	:	Within the learning phase the adapted parameters, especially the relevance weights, should be only *minimally* changed, i.e. the change must be so small that the classification of the corresponding cases just fits exactly and the threshold δ_i is exceeded minimally.

Again, the impact to the learning process of breaking the "CBR-Rule" can be found in [Haf93].

3.1.5 CBR Formula Derivation

If we take the learning rule of section 3.1.4 as major strategy to evolve a CBR-approach usable for the area of hyperlink-proposals (described in detail in chapter 7 and the subsequent chapters) we can derive the formulas for the changes in the relevance matrix ω.

Let us consider the similarity between two (numerical modeled) problems P_x and P_y. How can this similarity be determined?

The similarity calculation have to be done between all cases in the case-base CB and a new problem must be classified according to the refined CBR-algorithm mentioned on page 22.

At first, one have to decide whether the attributes of P_x and P_y belong to $A_x \cap A_y$ or not. According to the definition (3.3) we can use the function η:

$$\begin{aligned} \forall j \in \{1,2,\ldots\mathcal{N}\}: \quad & \eta(P_x^j, P_y^j) > \delta^j \quad \rightsquigarrow P_x^j, P_y^j \in A_x \cap A_y \rightsquigarrow P_x^j, P_y^j \in \mathcal{F} \\ & \eta(P_x^j, P_y^j) \leq \delta^j \quad \rightsquigarrow P_x^j, P_y^j \notin A_x \cap A_y \rightsquigarrow P_x^j, P_y^j \in \mathcal{C} \end{aligned} \tag{3.6}$$

For simplicity, we will write $j \in A^{\mathcal{F}}$ or $j \in A^{\mathcal{C}}$ to denote that the corresponding problem descriptions $P_x^j, P_y^j \in \mathcal{F}$ or $P_x^j, P_y^j \in \mathcal{C}$ respectively. These values have to be multiplied with the relevance matrix ω to judge whether the similarity of certain attributes is important for the corresponding solutions or not.

Thus, according to the refined ratio model (3.2) we get:

$$sim(P_x, P_y) = \frac{\alpha \sum\limits_{j \in A^{\mathcal{F}}} \omega_{ij} \cdot \eta(P_x^i, P_y^i)}{\alpha \sum\limits_{j \in A^{\mathcal{F}}} \omega_{ij} \cdot \eta(P_x^i, P_y^i) + \beta \sum\limits_{j \in A^{\mathcal{C}}} \omega_{ij}(1 - \eta(P_x^i, P_y^i))} \tag{3.7}$$

With formula (3.7) it is easy to determine the similarities between two problems and thus calculate the most similar case in the case-base CB during the classification phase.

For the learning phase we need to adapt at least the weights of ω. A change of the parameters α, β or the threshold values δ_i leads to different approaches (details about this point can be found in [Haf93]).

Within the learning phase let us consider a case $C = (P_x, \vec{s})$.

The storage of this case into CB only make sense if the similarities between P_x and all the problems P_y of cases within CB are smaller than a certain threshold δ_i where $i \in \{1,2,\ldots\mathcal{M}\}$. If this is not the case the relevance values have to be adapted, i.e.:

$$sim(P_x, P_y) = \delta_i + \Delta \quad \text{with} \quad \Delta > 0$$

For simplicity and clearness, we should use the following abbreviations:

$$\mathcal{E} = \sum_{j \in A^{\mathcal{F}}} \omega_{ij} \cdot \eta(P_x^j, P_y^j) \tag{3.8}$$

$$\mathcal{W} = \sum_{j \in A^{\mathcal{C}}} \omega_{ij} \cdot (1 - \eta(P_x^j, P_y^j)) \tag{3.9}$$

As a result of the learning rule described on page 22 we want to change the relevance weights so that the corresponding classification results in exactly the threshold δ_i for the solution class i, i.e.:

$$sim(P_x, P_y) = \frac{\alpha\mathcal{E}}{\alpha\mathcal{E} + \beta\mathcal{W}} = \delta_i + \Delta \tag{3.10}$$

It is important to understand the differences between δ_i and δ^j. The former describes the probability of a classification to be associated with the corresponding solution while the latter one means a similarity between two values of the same attribute for two different problem descriptions.

Equation (3.10) can be achieved by changing the importance of $\mathcal{E}$- and $\mathcal{W}$-values:

$$\delta_i = \frac{\alpha(\mathcal{E} - \Delta_\mathcal{F})}{\alpha(\mathcal{E} - \Delta_\mathcal{F}) + \beta(\mathcal{W} + \Delta_\mathcal{C})} \tag{3.11}$$

where $\Delta_\mathcal{F}$ denotes that weights of attributes within $\mathcal{F}$ (the *fulfilled attributes*) must be reduced in order to fulfill the condition above and $\Delta_\mathcal{C}$ means that contradicting values (within $\mathcal{C}$) have to be increased due to condition (3.10).

Therefore, the term *learning* can be seen as a kind of "error-correction" for a wrong classification.

The resolution of (3.11) towards $\Delta_\mathcal{C}$ results in:

$$\Delta_\mathcal{C} = \frac{1 - \delta_i}{\delta_i} \cdot \frac{\alpha}{\beta} \cdot (\mathcal{E} - \Delta_\mathcal{F}) - \mathcal{W} \tag{3.12}$$

This means, that the absolute values of α and β are unimportant, only their relationship counts.[6]

In the next step we present a possibility to determine the new weights of the relevance matrix ω in dependence of $\Delta_\mathcal{F}$. Finally, we derive the calculation of the value.

A typical setting for the new weights ω'_{ij} with a *proportional distribution* is shown in (3.13). For other distribution strategies like *constant, polynomial* or *exponential* we refer to [Haf93].

$$\omega'_{ij} = \begin{cases} \omega_{ij} \cdot \frac{\mathcal{E} - \Delta_\mathcal{F}}{\mathcal{E}} & : \quad P_x^i, P_y^i \in \mathcal{F} \\ \omega_{ij} \cdot \frac{\mathcal{W} + \Delta_\mathcal{C}}{\mathcal{W}} & : \quad P_x^i, P_y^i \in \mathcal{C} \end{cases} \tag{3.13}$$

The derivation of the value $\Delta_\mathcal{F}$ is remaining. Due to the *stochastic-property* of ω (see equation (3.5)) we can derive:

$$\sum_{j \in A^\mathcal{F}} \omega_{ij} \cdot \frac{\mathcal{E} - \Delta_\mathcal{F}}{\mathcal{E}} + \sum_{j \in A^\mathcal{C}} \omega_{ij} \cdot \frac{\mathcal{W} + \Delta_\mathcal{C}}{\mathcal{W}} = 1 \tag{3.14}$$

The substitution of $\Delta_\mathcal{C}$ with equation (3.12) results in:

$$\sum_{j \in A^\mathcal{F}} \omega_{ij} \cdot \frac{\mathcal{E} - \Delta_\mathcal{F}}{\mathcal{E}} + \sum_{j \in A^\mathcal{C}} \omega_{ij} \cdot \frac{\mathcal{W} + \frac{1-\delta_j}{\delta_j} \cdot \frac{\alpha}{\beta} \cdot (\mathcal{E} - \Delta_\mathcal{F}) - \mathcal{W}}{\mathcal{W}} = 1 \tag{3.15}$$

After a simplification we get:

$$\frac{\mathcal{E} - \Delta_\mathcal{F}}{\mathcal{E}} \cdot \sum_{j \in A^\mathcal{F}} \omega_{ij} + \frac{1 - \delta_j}{\delta_j} \cdot \frac{\alpha}{\beta} \cdot \frac{\mathcal{E} - \Delta_\mathcal{F}}{\mathcal{W}} \cdot \sum_{j \in A^\mathcal{C}} \omega_{ij} = 1 \tag{3.16}$$

[6] Certainly, this becomes clear if we divide both numerator and denominator of the right side of equation (3.2) by α.

And the transformation towards $\Delta_{\mathcal{F}}$ yields to:

$$\Delta_{\mathcal{F}} = \mathcal{E} - \frac{1}{\frac{1}{\mathcal{E}} \cdot \sum_{j \in A^{\mathcal{F}}} \omega_{ij} + \frac{1-\delta_j}{\delta_j} \cdot \frac{\alpha}{\beta} \cdot \frac{1}{\mathcal{W}} \cdot \sum_{j \in A^{\mathcal{C}}} \omega_{ij}} \tag{3.17}$$

With (3.17) a straightforward learning process of CBR is completely described.

In chapter 7 we will derive a slightly different set of formulas because we modeled the hyperlink-proposal systems not exactly as a kind of Case-Based Reasoning system.

3.2 Markov Chains

3.2.1 Terminology of Markov-Chains

In this section we will briefly present some theory of *Stochastic Modeling*, especially the idea of *Markov-Chains*, because this is very important for the area of predicting future user events. Furthermore, we will outline the main differences between our prediction approach (presented in chapter 4) and Markov-Chain modeling.

A short summary of attempts that have already been made to use these ideas for predicting web requests can be found in section 4.2.

A *stochastic process* can be regarded as a - mostly infinite - collection of random variables. This collection is indexed as shown in the following:

Stochastic process: $\{X(t) : t \in T\}$

$X(t)$ describes a random variable for each t belonging to the index set T. In general, $X(t)$ may be a vector of random variables. The set T can be for instance:

$$T = [0, \infty) \quad \text{or} \quad T = \{0, 1, \ldots\}$$

Mostly, the index t means "time" and therefore, the sets above stand for *continuous-time* or *discrete-time* modeling. But also other meanings of t are thinkable.

Possible values of $X(t)$ will be called *states* and the term $\{X(t) = x\}$ means, that the process $X(t)$ is in state x at time, or *epoch* t. With other words: the event $\{X(t) = x\}$ has occurred.

In the following, we will always consider the case of discrete-time modeling. For further details there is a lot of literature, see e.g. [Wol89].

3.2.2 Discrete Markov-Chains

A stochastic process as described in 3.2.1 is called a *Markov-Process* if it has the following *Markov-Property*:

Markov-Property	Given the present state of a stochastic process the future evolution of the process is independent of the past evolution of the process.

This is the fundamental axiom of the Markov-Chain theory. As we will see in subsection 3.2.3, it is possible to efficiently calculate the probabilities of future events when they can be modeled as Markov-Processes.

Admittedly, we think that in the case of predicting user-requests of general data-sets (not only HTML-pages) a modeling as Markov-Process would not be appropriate even though the calculations could be simplified (compare also the general considerations of 1.1).

In general, a sequence of random values of the form $\{X(t) : t \in \{1, 2, \ldots\}\}$ is called a *Discrete Markov-Chain* if it has the Markov-Property:

$$P(X(t+1) = i \mid X(t), X(t-1), \ldots, X(0)) = P(X(t+1) = i \mid X(t)) \quad \forall t \in \{0, 1, \ldots\} \quad \forall i \in \{0, 1, \ldots\}$$

The probabilities P are also called the *one-step transition probabilities*. When we think of independent probabilities, i.e. independent of the time t then we can construct the *one-step transition probability matrix* P:

$P = (P_{ij})$ with: $P_{ij} = P(X(t+1) = j \mid X(t) = i) \quad \forall i, j \in \{0, 1, \ldots\}$

Because the entries of P are probabilities, it is easy to see that the following property holds:

$$\forall i \in \{1, 2, \ldots\} : \sum_{j \in \{0,1,\ldots\}} P_{ij} = 1 \tag{3.18}$$

Matrices with the property (3.18) are called *stochastic*. In section 4.4.2.1 we will see that the matrix used for our prediction scenario is similar to the transition probability matrix[7] and can easily be transformed into a stochastic matrix, even though our model can not be described as discrete Markov-Chain.

With the Markov-Property, it can be proved that the n-step transition matrix $P^{(n)}$ that is the further guessing of probabilities for future events $X(t+n)$ is equal to the n-th power of P:

$$\forall n \in \{1, 2, \ldots\} : P^{(n)} = P^n \tag{3.19}$$

We will use this property to simplify the calculations of example 3.2.3.

3.2.3 Example of Efficient Markov-Chain Calculations

Let us consider the one-step transition matrix P to calculate the future weather forecast (for example "rainy" and "sunny"):

$$P = \begin{pmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{pmatrix}$$

For instance, $P_{11} = 0.3$ means that the probability for rainy weather tomorrow is 0.3, assumed that today is rainy weather.

What about the weather the day after tomorrow? Equation (3.19) guarantees that we just have to multiply the one-step transition matrix P to P^2:

$$P^2 = \begin{pmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{pmatrix} = \begin{pmatrix} 0.44 & 0.56 \\ 0.4 & 0.6 \end{pmatrix}$$

Certainly, this generates the same probability as if we would have taken the result of $P^{(1)}$ as input for the calculation of $P^{(2)}$. The probability, that the day after tomorrow will be a rainy day, under the condition that it is raining today, is 0.44.

As stated earlier, our prediction modeling is based on the idea that all former requests could possibly change the probability for future ideas. Therefore, our approach does not fulfill the Markov-Property. Nevertheless, there are in general several similarities in modeling future user-behavior.

[7] We will call this matrix *Memory Matrix*.

Part II

Request-Prediction

Chapter 4

Request-Prediction Theory

When you have eliminated the impossible,
whatever remains, however improbable,
must be the truth.

Sir Arthur Conan Doyle, The sign of four
from "The Adventures of Sherlock Holmes" (1887)

4.1 Basic Preferences of Request-Prediction

The *Request-Prediction Theory* - that is, the laws of Request-Prediction - is founded on some fundamental principles. These principles, or *axioms*[1] as we will call them, are usually not mentioned in scientific literature but they are most important.

Below, we will use the expression of requesting *data-sets*. Otherwise, the use of the term "data record" or "document" might imply that we are talking about a data-server or a web server, respectively.

The basic idea of prediction theory is embodied in the first - and most important - of the axioms:

Axiom 1	:	The probability for a request of a data-set in the future can be computed by previous request behavior.

This statement is not trivial at all. It is the fundament of all further request-prediction efforts. We want to stress explicitly that there is not only a coherence between past and future user-behavior but that this behavior can be predicted accurately. The axiom does not say that the request-probability for a specific user action must be calculated only on base of former request behavior of the same user. All past requests can be input data for future request behavior calculations.

Humans mostly tend to retain their request behavior, but this psychological argument depends on the quality of the presented data and the personal interests. If one of these

[1] We use the term "axiom" here not in the strict mathematical sense but rather to emphasize on the importance of the corresponding requirements for predicting user-requests.

parameters change, the request behavior might also change. Furthermore, it will become boring to request pages, whose content does not change. We will come back to this problem in section 6.1.

We do not aim to determine here whether such kind of prediction should actually be happening on the client-side or on the server-side. A brief discussion of this topic can be found in section 5.4.3.

The following axiom is nearly as essential as the first axiom. Here, a main requirement for predictions is declared:

Axiom 2	:	Results of predicted requests can be calculated beforehand.

If we can not take advantage of predictions by pre-calculations or some kinds of pre-actions they are senseless for our theory even though they could be predicted correctly. A simple example for this problem is the request of real-time stock data: even though one can foresee that certain stock values will be requested in the (near) future it would be senseless to make any pre-calculations because the data must be up-to-date at the moment of request[2]. Some problems concerning Axiom 2 with regard to proxy systems are implicitly discussed in [CDF98] and [KLM97] (see also [CI98]).

The third axiom serves as a connection between the first two axioms. It restricts the applicability of prediction technology considerably.

Axiom 3	:	Prediction of user-requests and pre-calculation of requested data increases the subjective performance of the client-server communication.

The main focus of request-prediction research is the impression on the user - thus we use the term "subjective" - who is made to feel that his waiting time has been reduced. Such improvements for the individual user may not be achieved at the expense of the rest of the Internet community, though. Therefore, is it not enough to speak of *reducing user-perceived latency*.

Prediction can only be recommended where all three axioms are valid. A wrong use of prediction algorithms can lead to a deterioration of network and system load. So we will learn (e.g. in section 5.4.2) that predictive algorithms should be used in a very *defensive* way and that the overall gain is as important as the user specific improvements.

Unfortunately, only few parts of the Internet communication fit the requirements of the described axioms. But prediction algorithms will improve only those services where all three axioms are fulfilled. The possible advantages of predicting future user-requests are enormous, though. Beside early calculation of complex functions also large retrievals of different databases in advance are possible. Furthermore, the results of those computations can be sent from the server to the client before they are requested. The user will perceive the response of the data-requests as if they were stored locally (pre-fetching or pre-sending). Right where the potential chances are the best, also the risks are the most dangerous ones: incorrectly predicted requests that lead to pre-sending or pre-fetching cause higher network and system load and can worsen the perceived latency for all users.

4.2 Prediction Research

The notion of predicting future events originates from compiler construction (*branch-prediction*) and is now being applied to Internet applications [JK98]. Many attempts have already been

[2]Certainly, other kinds of pre-actions might be appropriate, e.g. preparation of server resources and network bandwidth.

made to find the best and most efficient algorithms for fulfilling prediction needs (e.g. [Bes96]). *Discrete Markov-Chain* models were used as a basis for one of the Web's first algorithms of prediction [KW97]. Their concept is to store the frequency of user-requests and to apply the adequate statistical model. Later, these ideas were extended to a *Continuous Markov-Chain* approach [KW98] and to *Path Profiling* [SKS98] which focuses on the order of document demands and the resulting request path. By using predictions, average latency and system load may be reduced but several risks may result from inaccurate data prediction. The negative effects of incorrect predictions are discussed in [CB98] and [CDF98]. Performance modeling in general is described in [EY99].

Our first prediction-approach (mentioned in [HRE99a]) is based on an idea of Padmanabhan and Mogul [PM96]. We have improved this straightforward approach to model time and document aging [HRE00a].

As mentioned in section 4.1, the prediction of user-requests in general aims at reducing user-perceived latency. A central term in most of the existing prediction algorithms is the *session*. Even though the session cannot be justified on the basis of the standard WWW protocol HTTP[3], it is very important to group several requests together. In general, a session is regarded as a time period of about 30 minutes [SKS98]. During such a session, a user can request several documents (or other kinds of data-packets). The main goal of prediction aims at foreseeing some of the upcoming requests during the same session on the base of the requests that have been made already. Therefore, standard prediction-algorithms generate relative probabilities for future document requests on the basis of the relative frequencies of requests in the past.

Certainly, there are also some other approaches, but the perhaps most straightforward one simply stores the actual user-requests at first and then calculates probabilities for future request wishes. The former process is a (rather simple) kind of *learning phase*, while the latter one can be called a *classification phase*. We will come back to this kind of modeling in section 10.2.

If we do not consider maintaining the correct request order, a straightforward mathematical conception of a session will lead to a vector $\vec{s}$. Every request of a document corresponds to an element of $\vec{s}$. The size of the vector corresponds to the number of documents that should be part of the prediction algorithm (predictable documents). Details about finding the appropriate documents and criteria whether prediction should be made will be discussed in the following section.

4.3 Derivation of Prediction Thresholds

There are two different aspects of the derivation of prediction formulas. On the one hand we have to calculate the probabilities for requests of data-sets. We will highlight this topic in section 4.4.

On the other hand, we need a threshold value depending on the request-probability to find out whether the *prediction-action* (e.g. "pre-calculation" or "pre-fetching") can economize the cost function for the requests. We emphasize this point in the following subsections.

4.3.1 Cost Functions and Thresholds in General

To calculate the profit of prediction we will focus on the aspect of *pre-calculation* (of time intensive operations), on *pre-fetching* (of data from server to client) and on *piggyback transmission* (of meta data together with requested data)[4]. The early transfer of data to the client initiated by the server (if possible), also called *pre-sending*, is analogous to the pre-fetching process.

[3] See [HTTP] for definition details.

[4] Further details on this kind of transmission can be found in [CKR98] and [BC98].

Function	Description
γ^R	Resource costs that can not be reduced by prediction methods (e.g. retrieving real-time stock-data values)
γ^A	Avoidable costs that can be spared by correctly predicting the corresponding requests (e.g. calculating moving average values)
γ^T	Transmission costs (divided into γ^C and γ^S)
γ^C	Part of γ^T, the costs to connect to a remote host (might be a server or a client computer)
γ^S	Part of γ^T, the costs to transmit data that depends on the size of the transferred data-sets (assuming the use of the Internet-Protocol (IP))
γ^B	Book-keeping costs of the prediction algorithm itself (e.g. probability calculations)

Table 4.1: Ingredients of cost function γ

The following method for calculating the cost functions and the thresholds corresponds to many known approaches (e.g. [KLM97], [JK98]). A central term in the area of request-prediction is the *session* $\vec{s}$ as a sequence of user-requests within a fixed time span T. A customary value would be $T = 30$ minutes (e.g. [SKS98]). The function γ investigating the cost of a session $\vec{s}$ consists of several parts. For our approach we need the specifications described in Table 4.1.

Furthermore, we think of a server S as a pool of $\mathcal{A}$ data-sets d_i as mentioned in 4.1.

$$S = \{d_1, d_2, \ldots, d_{\mathcal{A}}\} \qquad \mathcal{A} \in N$$

Certainly, only potentially predictable data-sets are counted. To be able to derive the probability threshold for the prediction algorithms for a specific session $\vec{s}$ we have to divide the data-sets of S into two parts: the ones that could be predicted and those that could not. Without loss of generality, the first $\mathcal{B}$ data-sets $d_1, d_2, \ldots, d_{\mathcal{B}}$ could be predicted, where $\mathcal{B} \leq \mathcal{A}$.

In general, the regular[5] cost function Γ_r for a session $\vec{s}$ can be described as:

$$\Gamma_r(\vec{s}) = \sum_{i=1}^{\mathcal{A}} \gamma_r(d_i) \cdot p_i \tag{4.1}$$

where the data-set d_i will be requested with a probability of p_i during the session $\vec{s}$ and thus produces a regular cost of γ_r.

The use of prediction algorithms to reduce the request costs changes the equation (4.1) to (4.2). We call this function Γ_p (and γ_p evaluates to the corresponding costs of a single data-set d_i, respectively):

$$\Gamma_p(\vec{s}) = \sum_{i=1}^{\mathcal{B}} \gamma_p(d_i) + \sum_{i=1}^{\mathcal{A}} \gamma^B(d_i) + \sum_{i=\mathcal{B}+1}^{\mathcal{A}} \gamma_p(d_i) \cdot p_i \tag{4.2}$$

The cost calculations for sessions with use of prediction algorithms consist of three parts. For $d_i \in \{1, 2, \ldots, \mathcal{B}\}$ the probability part (p_i) disappears because those data-sets could be predicted and therefore their request-probability is 1. The second term models the book-keeping prediction costs for all data-sets independent of a successful prediction. The third term is the same term as before (for $d_i \in \{\mathcal{B}+1, \mathcal{B}+2, \ldots, \mathcal{A}\}$).

[5] without using prediction methods

To get a threshold for prediction-actions depending on the request-probabilities p_i we have to compare the costs of every data-set d_i from equation (4.1) to the costs calculated by equation (4.2). On basis of this consideration we get two conditions:

$$\gamma_p(d_i) < \gamma_r(d_i) \quad \forall i \in \{1, 2, \ldots, \mathcal{B}\} \tag{4.3}$$

what means that for correctly predicted data-requests prediction must always be an advantage. Additionally, for the remaining data-sets a certain threshold λ must not be exceeded:

$$\sum_{i=\mathcal{B}+1}^{\mathcal{A}} (\gamma_p(d_i) - \gamma_r(d_i)) < \lambda \tag{4.4}$$

This term results from the requirements of Axiom 3 (defined on page 30). Not only the predicted requests must be kept in mind, but also the cost considerations for the not-predicted elements play an important role. The condition (4.4) can only be verified *a posteriori* because it is impossible to determine the value without knowing exactly which data-sets were requested. Prediction must be deactivated if condition (4.4) can not be guaranteed.[6]

In the following, we will derive concrete equations for the different possibilities of prediction-actions (pre-fetching, pre-calculation etc.)

4.3.2 Pre-Calculation

In the case of *pre-calculation* as prediction-action the cost function Γ consists of resource costs γ^R and avoidable costs γ^A (see Table 4.1 for a description of the cost terms). Then, the equation (4.1) looks as follows:

$$\Gamma_r(\vec{s}) = \sum_{i=1}^{\mathcal{A}} (\gamma^R(d_i) + \gamma^A(d_i)) \cdot p_i \tag{4.5}$$

The arising costs are simply the sum of resource costs and avoidable costs (from the perspective of prediction algorithms). For instance, sorting of data takes a certain amount of time, thus producing costs (γ^R). Deteriorating effects for the user arise when system load is high. If a server could manage to sort data when system load is low, the corresponding avoidable costs (γ^A) would depend on the moment of computation. Therefore, only γ^A can be reduced by means of prediction.

Making use of prediction the cost function Γ_p (derived from equation (4.2)) looks as follows:

$$\Gamma_p(\vec{s}) = \sum_{i=1}^{\mathcal{B}} \gamma^R(d_i) + \sum_{i=1}^{\mathcal{A}} \gamma^B(d_i) + \sum_{i=\mathcal{B}+1}^{\mathcal{A}} (\gamma^R(d_i) + \gamma^A(d_i)) \cdot p_i \tag{4.6}$$

The share of γ^B arises as a result of prediction calculations which have to be made, regardless of whether they will be needed later on or not. The avoidable costs of $d_{\mathcal{B}+1}, \ldots, d_{\mathcal{A}}$, however, can then be ignored ("correctly predicted"[7]).

Analogous to the considerations in 4.3.1, prediction only makes sense if (4.6) is smaller than (4.5) for any data-set d_i with $i \in \{1, \ldots, \mathcal{B}\}$. Thus, the threshold for the probability value can be calculated as:

[6] Certainly, the concrete setting of λ depends on several prediction system aspects and can not be evaluated in general.

[7] For the derivation of the request-probability threshold the importance of the share of correctly predicted requests is not too high due to the consideration of the book-keeping costs γ^B. Nevertheless, the condition (4.4) is an appropriate de facto limit for the reasonable usage of prediction algorithms.

$$\gamma_p(d_i) < \gamma_r(d_i) \quad \forall i \in \{1, 2, \ldots, \mathcal{B}\} \quad \Rightarrow \tag{4.7}$$

$$\gamma^R(d_i) + \gamma^B(d_i) < (\gamma^R(d_i) + \gamma^A(d_i)) \cdot p_i \quad \forall i \in \{1, 2, \ldots, \mathcal{B}\} \quad \Rightarrow \tag{4.8}$$

$$p_i > \frac{\gamma^R(d_i) + \gamma^B(d_i)}{\gamma^R(d_i) + \gamma^A(d_i)} = \frac{1 + \frac{\gamma^B(d_i)}{\gamma^R(d_i)}}{1 + \frac{\gamma^A(d_i)}{\gamma^R(d_i)}} = \frac{1 + S_o(d_i)}{1 + S_f(d_i)} \quad \forall i \in \{1, 2, \ldots, \mathcal{B}\} \tag{4.9}$$

S_o describes the *Prediction-Overhead* that has to be small in comparison to S_f, the *System Flexibility*, for a low threshold.

The equation (4.9) describes a threshold (later also called δ[8]) for the request-probability of a data-set d_i for prediction to make sense. This means, if the request-probability exceeds the value of (4.9) the (successful!) usage of prediction can economize costs and otherwise presumably not!

Analogous to (4.4) we find the additional a posteriori condition:

$$\sum_{i=\mathcal{B}+1}^{\mathcal{A}} \gamma^B(d_i) < \lambda \tag{4.10}$$

We will come back to the practical consequences of the condition (4.10) when describing the prediction algorithm in 4.4.2.

4.3.3 Pre-Fetching (and Pre-Sending)

The cost equation for a session $\vec{s}$ in case of the prediction-action *pre-fetching* on the client-side (or *pre-sending* on the server-side, respectively) looks like this:

$$\Gamma_r(\vec{s}) = \sum_{i=1}^{\mathcal{A}} (\gamma^R(d_i) + (\gamma^C(d_i) + \gamma^S(d_i)) + \gamma^A(d_i)) \cdot p_i \tag{4.11}$$

where the change towards equation (4.5) consists of the additional transmission costs, the size-independent connection costs γ^C (also called *Routing-Overhead*) and the transferring costs γ^S. For simplicity, in our model each data-set d_i is requested (and sent) for itself. If more than one data-set is transferred at once, the modeling of the data-sets d_i has to be adapted so that the calculations will be similar to the case of single data-set sendings. For detailed analyses of transmission costs see also [JK98].

With the usage of pre-fetching, that is transmission of data not yet requested, the cost function can be calculated as:

$$\Gamma_p(\vec{s}) = \sum_{i=1}^{\mathcal{B}} \gamma^R(d_i) + \sum_{i=1}^{\mathcal{A}} \gamma^B(d_i) + \sum_{i=\mathcal{B}+1}^{\mathcal{A}} (\gamma^R(d_i) + (\gamma^C(d_i) + \gamma^S(d_i)) + \gamma^A(d_i)) \cdot p_i \tag{4.12}$$

where $d_1, d_2, \ldots, d_{\mathcal{B}}$ could be predicted (analogous to (4.2)). It is assumed here that correctly pre-fetched data produces no transmission costs.

[8] Certainly, the threshold δ depends on the cost calculation of data-set d_i. But for simplicity we will leave the index δ_i to signal that this threshold can also be set individually (e.g. for pessimistic strategies $\delta > p_i \quad \forall i \in \{1, \ldots, \mathcal{A}\}$).

Again, we can easily investigate the threshold value of (4.11) and (4.12):

$$p_i > \frac{\gamma^R(d_i) + \gamma^B(d_i)}{\gamma^R(d_i) + \gamma^C(d_i) + \gamma^S(d_i) + \gamma^A(d_i)} \quad \forall i \in \{1, 2, \ldots, \mathcal{B}\} \tag{4.13}$$

We see that the request-probability has to be higher if the system resources are much more expensive than the transmission costs. On the other side, the gain of correctly pre-fetched (or pre-sent) data-sets is much higher than in case of pre-calculation (4.3.2).

4.3.4 Piggyback Transmission

While sending requested information additional data can be piggybacked upon the transferred ones. But as we will see in section 5.4, it is not advisable to send very large packets of data due to serious network problems. Only if the volume of piggybacked information is reduced to a very small part, very interesting mechanisms are possible (see [CKR98] and [BC98] for details).

In this essay, we will not focus on the piggyback transmission because our main interest for predicting future user-requests is for larger amount of raw data (often not even HTML-pages[9]) where the piggybacking approach is not appropriate.

4.3.5 Multiple Preprocessing

It would be most effective to combine the advantages of the prediction-actions presented in 4.3.2 and 4.3.3. This is possible on the server side, yet a persistent connection between client and server must be granted. The server can then send not yet requested data in the idle time of the client so the information is local available when needed. Unfortunately, this solution causes problems with proxy-cache-servers between client and data-server. Due to this impact on practical implementations of *multiple preprocessing* we do not go into details for that topic. General considerations on this point can also be found in [CKR98].

4.4 Determination of Request Probability

4.4.1 The Mathematical Model

There are many known ways to determine the request-probabilities for HTML-pages described in the literature, for instance *Path-Profiling* [SKS98]. The tree-like connections between documents via hyperlinks make a *Markov-Model* approach adequate [KW97]. Besides a modeling of stochastic processes as *discrete Markov-Chains* ([Bes96] and [Bes95]) there are also examinations for a *continuous Markov-Chain* approach [KW98]. In the general case where there are no hyperlinks between requested data-sets we have to work with *Conditional Probabilities*.

4.4.2 The Basic Algorithm

4.4.2.1 Formula Derivation

The requests of a user session $\vec{s}$ is modeled - without considering the order or repeating requests - as a binary $\mathcal{A}$-vector of data-set indices:

$$\vec{s} = (s_i) \text{ with: } s_i = \begin{cases} 1 & : \text{ data-set } d_i \text{ was requested by the client} \\ 0 & : \text{ data-set } d_i \text{ was not requested} \end{cases}$$

To store the relative frequencies of requests, e.g. the number of requests for every document depending on the requests of all the other documents, we need - at least - a quadratic matrix

[9] Compare also the explanations of section 1.1

with the dimension of the number of predictable documents. We call this matrix the *Memory Matrix* φ. In the case where the request order is irrelevant, the matrix φ is also symmetric.[10] For efficiency, the relative probabilities are not stored directly in φ, but they are calculated from the relative frequencies at the moment when they are needed (some details about the implementation of the matrix can be found in 5.2).

The predicted data-sets are not treated in the same way as real requests, so that the Memory Matrix tends to "forget" the coherence between the data-sets.

$\varphi = (\varphi_{ij})$ with $1 \leq i, j \leq \mathcal{A}$

At first, φ has to be initialized as 0-matrix:

$$\varphi_0 = (\varphi_{ij}) \quad \text{where} \quad \varphi_{ij} = 0 \qquad \forall \ 1 \leq i, j \leq \mathcal{A} \tag{4.14}$$

Within every session $\vec{s}$ the matrix φ is calculated to φ', so that the elements of φ represent the frequencies of data requests, depending on requests from other data-sets (rows and columns of the matrix).

$$\varphi' = (\varphi'_{ij}) \quad \text{where} \quad \varphi'_{ij} = \begin{cases} \varphi_{ij} + 1 & : \quad s_i = 1 \wedge s_j = 1 \\ \varphi_{ij} & : \quad s_i \neq 1 \vee s_j \neq 1 \end{cases} \tag{4.15}$$

The conditional probability p_i, stating that d_i will be requested, if d_j has been requested during the same session $\vec{s}$ can then been calculated as:

$$p_i(d_i|d_j) = \frac{\varphi_{ij}}{\varphi_{jj}} \tag{4.16}$$

This formula expresses that the probability for a data request can be derived from the request frequencies so far (according to Axiom 1 defined in section 29).

However, we do not focus on the efficiency of that algorithm in this stage but use it as a (general) basis to model prediction in a test scenario. An interesting aspect is the modeling of user vectors in such a scenario.

We propose here an additional requirement (analogous to 4.4) to be able to fulfill Axiom 3 (of page 30) potentially:

$$\sum_{i=1}^{\mathcal{A}} \varphi_{ii} \ll \sum_{i=1}^{\mathcal{A}} \sum_{j=i+1}^{\mathcal{A}} \varphi_{ij} \tag{4.17}$$

The sum of values on the main axis, that represents the frequencies of all requests, should be much smaller than the sum of "connected" data requests. If this can not be guaranteed prediction-algorithms will presumably not work due to missing affiliation between the requested data.

But condition (4.17) only helps to find out specific reasons for low quality prediction results. It is not a strict constraint. Furthermore, equation (4.10) gives a condition for rating the performance of the prediction algorithm a posteriori. Several limits (denoted as λ) are possible. While comparing the same prediction algorithm for different sets of data, the quotient $\frac{\lambda}{\mathcal{A}-\mathcal{B}}$ can be regarded as a kind of measuring element, the lower the better. But λ does not say anything about the quality of the prediction process itself.

[10] This is true only for the most simple case. Later on, we will model time and document aging and the resulting matrix will not be symmetric any longer (see 6.1)

4.4.2.2 An example for clarification

To demonstrate the work of the basic algorithm we give the following simple example. Let us presume $\mathcal{A}$ and the Memory Matrix φ to be:

$$\mathcal{A} = 4 \quad \wedge \quad \varphi = \begin{pmatrix} 5 & 3 & 3 & 2 \\ 3 & 6 & 2 & 1 \\ 3 & 2 & 4 & 4 \\ 2 & 1 & 4 & 5 \end{pmatrix}$$

Here, according to (4.16), the conditional probability for data-set d_4 to be requested, if data-set d_1 has already been requested during the same session $\vec{s}$ amounts to:

$$p_i(d_4|d_1) = \frac{\varphi_{41}}{\varphi_{11}} = \frac{2}{5}$$

Furthermore, the session $\vec{s}$ should look as follows:

$$\vec{s} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

After the complete learning step (4.15) the new matrix φ' will be:

$$\varphi' = \begin{pmatrix} 6 & 3 & 3 & 3 \\ 3 & 6 & 2 & 1 \\ 3 & 2 & 4 & 4 \\ 3 & 1 & 4 & 6 \end{pmatrix}$$

We can see, that the relative request-probability has changed now:

$$p_i(d_4|d_1) = \frac{3}{6} = \frac{1}{2}$$

This means, that presumably in 50 percent of all cases data-set d_4 will be requested, if data-set d_1 has already been requested within $\vec{s}$.

More advanced and complex modeling of prediction will follow in section 6.1.

Chapter 5

A Request-Prediction Scenario

Alice came to a fork in the road. 'Which road do I take?' she asked.
'Where do you want to go?', responded the Cheshire cat.
'I don't know.' Alice answered. 'Then,' said the cat, 'it doesn't matter.'

Lewis Caroll, Alice in Wonderland (1865)

To be able to derive prediction algorithms adequately it is very important to describe in detail the underlying prediction model that we will call here the *Prediction Scenario*.

In the first step, the meaning of this scenario helps to explain the reasons for special assumptions and derivation of formulas (as in chapter 4).

Later on, the prediction scenario itself can be implemented to sustain an appropriate platform for testing one's prediction algorithms. This is much better than "real-life" tests where the critical aspects and errors in the prediction consideration lead directly to a deterioration of system and network performance. Obviously, the final step has to be the implementation of the prediction modules in existing server architectures to verify the testing results and drawing advantage of the prediction theory. Then, eventually, testing under real-life conditions is necessary.

In our case, this server architecture is the *Smart-Data-Server* introduced in section 2.1.

5.1 Modeling Randomness and Semi-Randomness

5.1.1 Introduction to the Idea of Semi-Randomness

Requests of data can be divided into two classes: *predictable requests* and *random requests*. The reason for that classification lies in the different a priori request-probability. Predictable requests are often based on measurable personal facts of certain importance for a user so that the request-probability is not purely random, but semi-random.

Let us consider the case of financial data - especially of stock-data values - as a first example. It is obvious that the customer of a bank wants to be informed about the values of shares that are part of her/his personal portfolio. Perhaps all stock-data values are of a certain interest but those of the personal portfolio are of a *special interest*. For security reasons, it might not be possible to access portfolio data. But when the proposed method is used, accessing the portfolio is not even necessary! It is sufficient to take into account that there is potential *semi-random* data to provide.

For another example we should have a look at great sports events, for instance the "Football World Cup". The preliminary rounds are mostly more interesting for people that have the same nationality as the football players on the playground. Here too, pure randomness in the television-quotes can not be found.

There are also examples with no objective semi-randomness. Weather forecasts can be of extremely high interest for someone who plans to renew the roof of her/his house but there are no measurable facts that can be used by algorithms to predict the high interest in such data. Of course, the distinction between *random* and *semi-random* requests is not very sharp. There can be practical cases where no real difference between both types of a priori probability exist. Nevertheless, it is still important to model random and semi-random requests in a different way.

5.1.2 Modeling Semi-Randomness and Randomness

To test our prediction algorithms of chapter 4, we build a test scenario where the session vectors are generated with a *Randomizer-Module*. As in section 4.3.1, we can say that the first $\mathcal{B}$ elements of the session $\mathcal{A}$-vectors $(d_1, \ldots, d_{\mathcal{B}})$ are considered as *semi-random* or *predictable*, the others $(d_{\mathcal{B}+1}, \ldots, d_{\mathcal{A}})$ as *random* or *casual*.

$$\vec{s} = \begin{pmatrix} d_1 \\ \vdots \\ d_{\mathcal{B}} \\ d_{\mathcal{B}+1} \\ \vdots \\ d_{\mathcal{A}} \end{pmatrix}$$

Within the prediction scenario, we can identify the request for a data-set with the corresponding element of the session vector $\vec{s}$ and thus get:

$$d_i \in \{0, 1\} \quad \text{and} \quad d_i = 1 \iff d_i \text{ has been requested during session } \vec{s}$$

We are controlling the generation of these binary values with two global parameters: the random factor $\mathcal{R}$ and the density $\mathcal{D}$ with:

$$\mathcal{R} \in [0, 1] \quad \text{and} \quad \mathcal{D} \in \{1, \ldots, \mathcal{A}\}$$

The random factor $\mathcal{R}$ controls the probabilities of all values of the generated session vector $\vec{s}$. $\mathcal{R} = 1$ means *pure randomness* with a constant probability (of $\frac{1}{2}$) for all elements of $\vec{s}$. $\mathcal{R} = 0$ means *no randomness* where the first $\mathcal{B}$ elements of $\vec{s}$ are always set to 1, all other values are set to 0.

The density $\mathcal{D}$ describes the average share of values set to 1 within the session vectors for the highest randomness ($\mathcal{R} = 1$). $\mathcal{D}$ plays no role if $\mathcal{R}$ equals to 0. Thus, the importance of $\mathcal{D}$ increases with an increasing value of $\mathcal{R}$.

In general, the probability ϕ of the elements of the generated session vectors to become 1 follow the condition below:

$$\phi(d_i = 1) = \begin{cases} 1 + \mathcal{R} \cdot (\frac{\mathcal{D}}{\mathcal{A}} - 1) & : \quad 1 \leq i \leq \mathcal{B} \\ \\ \mathcal{R} \cdot \frac{\mathcal{D}}{\mathcal{A}} & : \quad \mathcal{B} + 1 \leq i \leq \mathcal{A} \end{cases} \tag{5.1}$$

Therefore, a random factor of $\mathcal{R} = 1$ generates the probabilities:

$$\phi(d_i = 1) = \frac{\mathcal{D}}{\mathcal{A}} \quad : \quad \forall i \in \{1, 2, \ldots, \mathcal{A}\} \tag{5.2}$$

whereas a random factor of $\mathcal{R} = 0$ leads to:

$$\phi(d_i = 1) = \begin{cases} 1 & : \quad 1 \leq i \leq \mathcal{B} \\ \\ 0 & : \quad \mathcal{B} + 1 \leq i \leq \mathcal{A} \end{cases} \tag{5.3}$$

With both controlling parameters $\mathcal{R}$ and $\mathcal{D}$, we will prove to model our prediction scenario adequately so that the testing results of the algorithm are applicable to the real-life usage of the prediction-actions.

5.2 Module Implementation

5.2.1 The Session Generator Module

To verify the quality of the derived prediction formulas (see section 4.3.1), we implemented the *Randomizer-Module* (also called the *Session Generator Module*) to generate artificial session vectors.

The program can be called with the following parameters (in parenthesis the variable name within the source code is appended):

- $\mathcal{A}$ (slen)

  Length of the sessions to be generated

- $\mathcal{B}$ (nump)

  Number of predictable elements of a session

- $\mathcal{R}$ (rf)

  Random factor

- $\mathcal{D}$

  Density, the number of settings within the session vector for $\mathcal{R} = 1.0$

The program produces its results to the standard output of the operating system as input for subsequent processing steps.

Within the session generating loop, random numbers are produced and compared to a calculated boundary. For predictable elements (first $\mathcal{B}$ elements of the session vector), this boundary is called *boundary_pred* and calculated by formula (5.1). In the same formula, the computing of *boundary_rand* for the semi-random elements is described.

All values exceeding the boundary lead to a "1" within the generated session vector. All others result in a "0" and mean "no request of the corresponding data set".

For reasons of run-time efficiency, the generation of the predictable and the semi-random elements happen in two different "for-loops". As result the session vectors are given as one line of output for each of them.

An exemplary main procedure of a C-programming language source code can be found in appendix A.1.

5.2.2 The Prediction Module

The implementation of the predicting module within the test scenario is more complex than the generating module. In the following, we describe briefly the procedures and the parameters to start the program. A coding example with C-programming language syntax can be found in appendix A.2.

Both programming modules, the generator and the predictioner, can be combined on the shell by piping the output of the generator as input for the predictioner. Certainly, also real logs can be processed by exchanging the generator.

The predicting program consists of the following procedures (for the detailed description of the input and output specification we refer to the source code in the appendix A.2).

- *init_phi*

  Initializes the matrix φ for subsequent processing steps.

- *evaluate_prediction*

  To verify the prediction quality this function compares the predicted probabilities with the real settings within the presented session vector. The evaluation is measured according to four aspects: The number of correctly predicted requests *grp*, requests that were predicted but did not happen *gwp*, those that were not predicted but occurred *gwn*, and finally, requests with a low probability that did not occur *grn*. Certainly, also the overall number of predictions *gtotal* is very important. The most critical value is *gwp*, because wrongly predicted requests lead to a higher system load and - in case of pre-fetching or pre-sending of data - also to a higher network load.

- *learn_request* and *learn_session*

  These are the procedures that change the memory matrix φ by adding the relative frequencies of the data sets. The derivation of the used formulas can be found in section 4.4.2.

- *predict_with* and *predict_session*

  The classification algorithms calculate a probability for the requests. Only those are predicted that exceed the threshold δ. The setting of this threshold is discussed at the end of this section.

- *main*

  The main procedure accepts three parameters. Beside the *session vector length* and the *threshold* δ also the *number of training sessions* is possible. The evaluation of the model showed that this distinction is not needed. At the beginning, the probabilities seldom exceed the threshold and thus produce no predictions. Special training sessions are, therefore, hardly needed.

  After checking the parameters and initializing the variables, the main procedure runs into a loop reading new sessions (e.g. generated by the randomizer module). For non-training sessions the classification routine is called by *predict_session*. At the end of each loop-run, the session is "learnt". Finally, the calculated evaluation results are given to the standard output.

As part of the Smart Data Server architecture, the implementation of the prediction module leads to an *SDS-workflow module* (2.1).

As mentioned earlier, a very critical point in generating request-prediction is to take care of the different costs. Not only the system costs for the prediction itself must be taken into account, but also the network load and the server load for wrongly predicted documents. These considerations are very important especially for making pre-fetches.

Within the testing scenario, the costs are given in advance. In the real life, those cost calculations are not very easy to obtain. As part of a project supported by the "Stiftung Innovation" of Rhineland-Palatinate, the Institute of Telematics is working on efficient extensions of the Smart Data Server with increased performance towards intelligent data routing. Here, load balancing plays an important role. With such a module, also the costs of calculations can be obtained by measuring their running time.

MV	Description
OP	Overall sum of predictions (of whole sessions)
CP	sum of correctly predicted settings ("1" in the prediction vector corresponded to "1" in the session vector)
WP	sum of wrongly predicted settings ("1" in the prediction vector corresponded to "0" in the session vector)
CN	sum of correctly prediction of non-settings ("0" in the prediction vector corresponded to "0" in the session vector)
WN	sum of wrongly prediction of non-settings ("0" in the prediction vector corresponded to "1" in the session vector)

Table 5.1: Measuring values (MV) for the prediction analysis

The advanced implementation of the prediction module to map time and aging is given in section 6.2.2.

5.3 Evaluation of the Model

5.3.1 Test Volume and Training Phase

Our results are based on thousands of test-runs with different settings of the global parameters $\mathcal{A}, \mathcal{B}, \mathcal{D}, \mathcal{R}$, and of the number of initial training sessions $\mathcal{I}$ terminating with $\mathcal{P}$ prediction sessions. Each run generated a session vector $\vec{s}$. Elements of $\vec{s}$ that are set to 1 represent a data-request. In our scenario the order of those requests is not modeled. Thus, every $d_i = 1$ of $\vec{s}$ could potentially be the first request of $\vec{s}$ as the base for a prediction of the rest. For this reason, we generated several prediction calculations, one for each "1" in $\vec{s}$ without using information about the settings of the other elements. Certainly, the session vector $\vec{s}$ changes the Memory Matrix φ only once!

The prediction algorithm as presented in section 4.4.2 produces a probability for every element of $\vec{s}$. Any calculated value exceeding the system dependent threshold p_i has to be considered as "predicted". In our test scenario we used the (constant) thresholds $\delta := 0.95$ and $\delta := 0.85$ as reasonable settings for p_i. The complete prediction vector has been compared afterwards with the "real" session vector. After several test-runs we evaluated the values presented in Table 5.1.

High values of WP must be viewed as *critical* because they can cause enormous negative side effects whereas high values of WN are not critical. However, they are a signal for inefficient prediction mechanisms. The ratio $PQ = \frac{CP}{WP}$ is called the *Prediction Quality*.

The first results during the test-runs were rather surprising. We had been thinking that it would be a good idea to settle a training phase at the beginning of the tests to fill the Memory Matrix with reasonable values. Astonishingly, the predictive correctness in the training phase had no significant effect on the results of the predicting phase. The main reason for this is that prediction will not occur if the corresponding line of the Memory Matrix φ is 0. This is always the case at the beginning of the training phase.

5.3.2 Impact of Random Factor $\mathcal{R}$ and Density $\mathcal{D}$

The following tables show some test results with different global parameter settings. We begin with variable values of $\mathcal{B}$ and two different settings of δ (Table 5.2).

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{D}$	$\mathcal{P}$	δ	**PQ**	**OP**	**CP**	**WP**	**CN**	**WN**
10^3	$\mathbf{10^0}$	10^2	10^3	**0.95**	0.5	9917	5505	10386	$9.8 \cdot 10^6$	101956
10^3	$\mathbf{10^1}$	10^2	10^3	**0.95**	3.6	18045	52986	14896	$1.7 \cdot 10^7$	282609
10^3	$\mathbf{10^2}$	10^2	10^3	**0.95**	8.9	99084	657088	74200	$8.9 \cdot 10^7$	9178672
10^3	$\mathbf{10^0}$	10^2	10^3	**0.85**	0.6	9921	6591	10499	$9.7 \cdot 10^6$	100870
10^3	$\mathbf{10^1}$	10^2	10^3	**0.85**	5.9	18066	136187	22934	$1.8 \cdot 10^7$	199408
10^3	$\mathbf{10^2}$	10^2	10^3	**0.85**	10.1	98997	$8.6 \cdot 10^6$	858014	$8.8 \cdot 10^7$	1195560

Table 5.2: Prediction results with different share of predictable values (for $\mathcal{R} = 0.1$)

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{D}$	$\mathcal{R}$	$\mathcal{P}$	**PQ**	**OP**	**CP**	**WP**	**CN**	**WN**
10^2	10^1	10^1	**0.0**	10^5	∞	99990	899910	0	$9.0 \cdot 10^6$	0
10^2	10^1	10^1	**0.1**	10^5	10.3	99942	22170	2146	$8.9 \cdot 10^6$	894584
10^2	10^1	10^1	**0.2**	10^5	4.7	100081	5174	1108	$8.9 \cdot 10^6$	929835
10^2	10^1	10^1	**0.3**	10^5	2.5	99912	2498	1003	$8.9 \cdot 10^6$	942867
10^2	10^1	10^1	**0.4**	10^5	1.3	100058	1246	977	$8.9 \cdot 10^6$	958507
10^2	10^1	10^1	**0.5**	10^5	0.7	100039	658	894	$8.9 \cdot 10^6$	968370
10^2	10^1	10^1	**0.6**	10^5	0.4	99891	339	911	$8.9 \cdot 10^6$	972688

Table 5.3: Influence of the random factor upon prediction quality (for $\delta = 0.95$)

We can see here that the prediction quality PQ increases with the share of predictable elements $\mathcal{B}$. The threshold value of $\delta = 0.95$ seems to be too pessimistic, especially for high values of $\mathcal{B}$. Nevertheless, the absolute value of wrongly predicted requests is rather low. It depends not only on the system cost conditions (as defined in 4.3.1) but on the "long time request-behavior" of the users. Certainly, therefore the kind of presented data on the server-side plays an important part in the decision if a rather pessimistic or an optimistic prediction strategy is the best.

The importance of the random factor $\mathcal{R}$ is presented in Table 5.3. It is obvious that low values of $\mathcal{R}$ lead to very good prediction conditions, but even a random factor of $\mathcal{R} = 0.4$ is not acceptable because there are almost as many wrongly predicted requests (WP) as correctly predicted ones (CP). Higher values of $\mathcal{R}$ (e.g. $\mathcal{R} > 0.4$) do not make sense for prediction algorithms any more. Requests are then too casual and prediction modules do not have the chance to predict future requests correctly and should be turned off.

The density $\mathcal{D}$ is variable in the Table (5.4), again with different settings of the threshold δ. The density factor $\mathcal{D}$ shows a very interesting impact on the prediction quality PQ. For very low values of $\mathcal{D}$, PQ is very good. This becomes clearer if we regard $\mathcal{D}$ as a kind of *unpredictable part* of a session vector. But when $\mathcal{D}$ reaches very high values, nearly as high as $\mathcal{A}$, the part of "randomly" correctly predicted (CP) requests increases as well. Thus, $\mathcal{D}$ can be viewed as a key parameter of the whole prediction process.

5.3.3 Summarized Results

The main and most evident test results are summarized below.

- There is a strong relationship between the randomness of session vectors and therefore of user-request behavior and the possibility to predict upcoming events. Prediction itself

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{D}$	δ	**PQ**	**OP**	**CP**	**WP**	**CN**	**WN**
10^3	10^1	$\mathbf{10^0}$	**0.95**	8.7	9003	1429	165	$8.9 \cdot 10^6$	72473
10^3	10^1	$\mathbf{10^1}$	**0.95**	5.8	9383	4421	760	$9.3 \cdot 10^6$	81480
10^3	10^1	$\mathbf{10^2}$	**0.95**	3.6	18045	52986	14896	$1.8 \cdot 10^7$	282609
10^3	10^1	$\mathbf{10^3}$	**0.95**	10.6	107887	$1.1 \cdot 10^6$	101682	$9.6 \cdot 10^7$	$1.1 \cdot 10^7$
10^3	10^1	$\mathbf{10^0}$	**0.85**	9.1	8997	71477	7871	$8.9 \cdot 10^6$	2425
10^3	10^1	$\mathbf{10^1}$	**0.85**	8.9	9401	74515	8418	$9.3 \cdot 10^6$	11386
10^3	10^1	$\mathbf{10^2}$	**0.85**	5.9	18053	136187	22934	$1.8 \cdot 10^7$	199408
10^3	10^1	$\mathbf{10^3}$	**0.85**	10.6	106939	$1.1 \cdot 10^6$	101687	$9.6 \cdot 10^7$	$1.1 \cdot 10^7$

Table 5.4: Importance of the density factor (for $\mathcal{R} = 0.1$ and $\mathcal{P} = 10^3$)

should be turned off if the amount of semi-random data ($\mathcal{B}$, Table 5.2) is becoming too small due to the negative side effects of wrong prediction.

- Even a moderate value of the random factor $\mathcal{R}$ leads to a catastrophic prediction profit (Table 5.3).
- Increasing the amounts of semi-random elements ($\mathcal{B}$) and decreasing the density ($\mathcal{D}$) to a very low value has a desirable effect upon the prediction results[1].
- As we stated above, the number of training sessions is of no significant importance for the correctness of the ensuing prediction. In addition, we found that even the amount of session vectors in the prediction phase plays no important role for the quality of the prediction.
- The threshold δ that is determined by the calculated value p_i of equation (4.9) controls the prediction results in a decisive manner. But lowering the threshold values does not only lead to a better result of correct hits (CP) but also accounts for the increasing amounts of the very negative wrong prediction (WP). Therefore, a pessimistic strategy requires probabilities that are higher than the calculated threshold p_i.

To evaluate the quality of the prediction values we must have a closer look at the prediction quality PQ-quotient. All values with $PQ < 1.0$ are evidently not preferable. We found that in practice, $PQ > 5.0$ is a good signal to improve user perceived performance with prediction methods.

5.4 Critical Aspects of Request Prediction

5.4.1 Identification of Similar Requests

The main idea of prediction is based upon the fact that future requests can be derived from the user behavior so far (Axiom 1 defined on page 29). This idea can only be applied if similar requests are grouped together. Two extreme positions should be avoided:

- Two requests of the same data-set are not identified due to different function parameters. For instance, *cookies* complicate the cacheability of documents in proxy-cache servers [Lau98]. The argument is applicable for functional parametric calls (CGI-scripts). A

[1] In "real-life", the density $\mathcal{D}$ depends on the average session length - that is the number of user-requests per session - and the number of presented data-sets $\mathcal{A}$ that define the length of the session vector $\vec{s}$.

prediction algorithm must take into account that there are possibly some irrelevant[2] parameters in function requests.

- Two seemingly identical requests can lead to very different server answers. Information about the newest weather forecast, about stock quotations or train departures can be very different even though all parameters are identical.

Nevertheless, the identification of similar requests is most important for every implementation of request-prediction algorithms. Much effort has to be made in order to check whether user-requests are really the same or not.

In most cases, the answer to this question can only be found by analyzing intensively the possibilities to request (the same) server-data on the one hand and the kind of offered data on the other.

5.4.2 Network Contradictions

It seems to be evident that the transmission of a few but big TCP-packets is much more efficient than the transfer of several small packets of the same volume in the sum (for TCP protocol definition see [Ste94]). But this is not true. Big packets can cause troubles with router buffers, so that they are transferred more slowly than many small ones [CB98]. This means while prefetching data from a server the whole set should be transmitted in "equal proportions". We called this strategy *defensive*.

The defensive use of pre-sending or pre-fetching data decreases the contradictory network problems and finally leads to higher performance.

The request strategy can be part of the client application or can be controlled from the server-side, respectively. Further details can be found in [CB98].

5.4.3 Server-Side versus Client-Side

We have not mentioned so far (in chapter 4) where the prediction should happen, on the client-side or on the server-side. In (4.3.5) we showed that a combination of both might be the best thing. The server can use as base of its prediction not only the requests of one user, but of all users ever connected to it. Especially at the beginning of the learning process, it is very helpful to have a large database as foundation of the prediction [SKS98].

One can argue against the rule "only clients can connect to servers but not vice versa" that the use of *persistent connections* blurs the sharpness of this aspect.

This kind of connections cause many other difficulties to the network environment. Therefore, their discussion is not within the scope of this work. For some details on this topic see [CDF98].

5.4.4 Use of Additional Information

Axiom 1 (defined on page 29) proclaims coherence between user-requests in the past and in the future. It is psychologically obvious that (human) users tend to request from the same data-servers several times (personal habits) if they are content with the presentation of the information. Those who are interested in stock quotations are looking for any information available about shares of their personal portfolio. It would be very effective to use that additional information while predicting the next requests. An online-authoring systems like DAPHNE (as presented in 2.2.2) could provide several information categories. But there are a lot of legal and security questions to be answered before such data can be used.

Furthermore, such a situation for prediction algorithms can not be discussed in general but depends strongly on the kind of predicted data.

[2] If we only take into consideration the prediction algorithms.

Chapter 6

Advanced Prediction Approaches

I returned, and saw under the sun, that the race is not to the swift,
nor the battle to the strong, neither yet bread to the wise,
nor yet riches to men of understanding, nor yet favor to men of skill;
but time and chance happen to them all.

KOHELET; 9,11
The Hebrew Names Version of the World English Bible (HNV),
based on the American Standard Version

6.1 Modeling Time and Document Aging

With regard to our former prediction scenario (chapter 5), we will focus on document requests without considering either the order of requests or multiple requests per session. Thus, a user-session can be regarded as a binary vector $\vec{s}$. The whole test scenario, therefore, consists of two parts, a *Randomizer-Module* that was able to generate session vectors randomly and a *Prediction-Module* that needs uncompleted session vectors (at the beginning of a session) and that is able to predict the missing requests of the currently running user session.

The most interesting part of the Randomizer-Module (also called the *Session Generator Module* is the setting of the parameters for controlling the structure of the session vectors. It must be possible to model both a very stringent user-behavior and a very random one. Also, the average number of requests per session must be variable to receive general and useful results. But the most important idea for the refinement of the prediction scenario of chapter 5 that we present here is an explicitly *modeling of time and of document aging.*

It is not only the generator module that has to be adapted to implement time modeling. The prediction module has to be refined as well. Therefore, the prediction module works in two different ways:

- An incoming (incomplete) session vector describing the beginning of a session is mathematically multiplied with a *Memory Matrix* φ (described in detail in section 4.4.2). From the result of this multiplication a vector which contains the probability for every document to be requested during the current session can be derived easily. This step can also be called *classification phase*[1].

[1]For a detailed discussion of regarding this step as a "classification step" we refer to section 10.2.

- In the *learning phase*, the prediction module stores a (complete) session vector with the real document requests in the matrix φ in order to improve future speculations.

The elements of the Memory Matrix φ represent frequencies of former requests and weights for future requests. In order to model time and document aging only the learning phase must be refined. In section 6.1.1, we derive a method to do this by "forgetting" former requests step by step. The longer the time span since the last request is, the higher is the degree of forgetting.

6.1.1 Derivation of the Time Formulas

To extend the scenario of 4.3.1 as described on page 31, we will first repeat the main ideas up to the following aspect.

Every user session $\vec{s}$ is described as a binary $\mathcal{A}$-vector of requests with:

$$\vec{s} = \begin{pmatrix} d_1 \\ \vdots \\ d_{\mathcal{B}} \\ d_{\mathcal{B}+1} \\ \vdots \\ d_{\mathcal{A}} \end{pmatrix}$$

where the first $\mathcal{B}$ elements of $\vec{s}$ are *predictable* or *semi-random*, the other elements are random requests and are not predictable. To control the generation of session vectors we use two important parameters.

- $\mathcal{R}$ is the random factor (or indicator): $\mathcal{R} \in [0, 1]$ where $R = 1.0$ indicates totally random requests and $\mathcal{R} = 0.0$ indicates no random.
- The density $\mathcal{D} \in \{1, \ldots, \mathcal{A}\}$ describes the average number of single document requests per session (for details see 5.1.2).

Accordingly, the probability ϕ for elements of $\vec{s}$ to become 1 (=requested) has been described already in section 5.1.2. For simplicity of reading this section, we repeat the equation (5.1) here as formula (6.1).

$$\phi(d_i = 1) = \begin{cases} 1 + \mathcal{R} \cdot (\frac{\mathcal{D}}{\mathcal{A}} - 1) & : \quad 1 \leq i \leq \mathcal{B} \\ \\ \mathcal{R} \cdot \frac{\mathcal{D}}{\mathcal{A}} & : \quad \mathcal{B} + 1 \leq i \leq \mathcal{A} \end{cases} \tag{6.1}$$

This formula contains no time modeling factor. To implement a time factor τ in a prediction algorithm that is supposed to make any sense for an automatically generated number of user-requests in a testing scenario, we need an additional factor ξ, called the *Request-Change-Probability* for the generation of data.

$\xi \in [0, 0.5]$ simulates the different possibilities for user-behavior. To express that there is no constant pattern in the request demands, ξ should be set to $\xi := 0.5$. This means that the conditional probability to request a data-set d_i during a session is independent from any requests that have taken place before. If ξ is set to 0.0 then a change in the request behavior is out of the question and if a data-set d_i is requested in one session, it must be requested in all subsequent sessions too.

A straightforward integration of the new factor ξ into the formula (6.1) is shown in the following. As abbreviation we set

$$\nabla := 1 + \mathcal{R} \cdot \left(\frac{\mathcal{D}}{\mathcal{A}} - 1 \right) \tag{6.2}$$

The new probabilities ϕ^j for session $\vec{s}_j$ now depend on the values of the previous session $\vec{s}_{j-1}$:

$$\phi^j(d_i^j = 1) = \begin{cases} 1 + 2 \cdot \xi \cdot (\nabla - 1) & : \quad 1 \leq i \leq \mathcal{B} \quad \wedge \quad d_i^{j-1} = 1 \\ \nabla \cdot 2 \cdot \xi & : \quad 1 \leq i \leq \mathcal{B} \quad \wedge \quad d_i^{j-1} = 0 \\ \mathcal{R} \cdot \frac{D}{A} & : \quad \mathcal{B} + 1 \leq i \leq A \end{cases} \tag{6.3}$$

The random part of the probability setting ($\mathcal{B} + 1 \leq i \leq \mathcal{A}$) has not changed. To derive the formula for the first $\mathcal{B}$ probability values ϕ^j of $\vec{s}_j$ we consider the case where the former value shows a requests, that is $d_i^{j-1} = 1$. The new probability must become greater with an increasing value of ξ. On the one hand, for $\xi = 0.0$ the probability of $d_i^j = 1$ is thus equal to 1. On the other hand the probability value has to be the same as before if $\xi = 0.5$.

To understand the concrete use of ξ in formula (6.3) we can also take into account the following consideration for the data-sets $\{d_1, \ldots, d_{\mathcal{B}}\}$. The probability ϕ^j for a new data-request in session $\vec{s}_j$ under the condition that the same request in session $\vec{s}_{j-1}$ has taken place already should be equal to the current probability to become 1 in the session vector added to the probability to become 0 multiplied with a factor depending on ξ. This can be written for every data-set $d \in \{d_1, \ldots, d_{\mathcal{B}}\}$ as:

$$\phi(d^j = 1 | d^{j-1} = 1) = \nabla + (1 - \nabla) \cdot (1 - 2 \cdot \xi) = 1 + 2 \cdot \xi \cdot (\nabla - 1) \tag{6.4}$$

Thus we get:

$$\phi(d^j = 1 | d^{j-1} = 1) \in [\nabla, 1] \tag{6.5}$$

Analogous we find:

$$\phi(d^j = 1 | d^{j-1} = 0) = 2 \cdot \xi \cdot \nabla \tag{6.6}$$

$$\phi(d^j = 1 | d^{j-1} = 1) \in [0, \nabla] \tag{6.7}$$

The overall sum of the probabilities must be the same as before, thus it is easy to see that the following equation (6.8) is true:

$$\nabla \cdot \phi(d^j = 1 | d^{j-1} = 1) + (1 - \nabla) \cdot \phi(d^j = 1 | d^{j-1} = 0) = \nabla \tag{6.8}$$

If we keep in mind that $\nabla = \phi(d_i = 1)$ for $d_i \in \{1, \ldots, \mathcal{B}\}$ (see formulas (6.1) and (6.2) for details) we can rewrite equation (6.8) to (6.9):

$$\phi(d^{j-1} = 1) \cdot \phi(d^j = 1 | d^{j-1} = 1) + \phi(d^{j-1} = 0) \cdot \phi(d^j = 1 | d^{j-1} = 0) = \phi(d^j = 1) \tag{6.9}$$

Formula (6.9) resembles then the well-known *Theorem of the total probability* (see [GKH86] for further information on that topic).

With the new probability values we can generate time dependent binary vectors for the prediction test scenario.

6.1.2 Extensions of the Basic Algorithm

To describe the extensions for the prediction algorithms of 4.4.2, we compare the components of the program with the new extensions. The core of the prediction algorithm is a symmetric Memory Matrix φ. Every row and column corresponds to an available data-set on server-side, for instance a document on a web-server.

The classification of an (incomplete) session vector $\vec{s}$ can be obtained by multiplying the matrix φ with $\vec{s}$ resulting in a vector that represents the probabilities of the corresponding documents to be requested. Only values exceeding a system-dependent threshold δ (derived from the probability p_i of equation (4.9)) will lead to a prediction of the corresponding document request (details to determine this threshold can be found in 4.3.1).

A new session vector $\vec{s} = (d_i)$ changes the memory of the prediction module by increasing the corresponding values ("weights") in $\varphi = (\varphi_{ij})$:

$$\varphi' = (\varphi'_{ij}) \quad \text{where} \quad \varphi'_{ij} = \begin{cases} \varphi_{ij} + 1 & : \quad d_i = 1 \wedge d_j = 1 \\ \varphi_{ij} & : \quad d_i \neq 1 \vee d_j \neq 1 \end{cases} \tag{6.10}$$

In order to model time and aging of documents we extend the algorithm by introducing the time factor $\tau \in [0, 1]$ (first mentioned in 6.1.1). This number describes the absolute decrease of the probability values not requested for every time unit since the last request was made. We denote the time unit since the last request of a data-set d_i as ε_i. For web applications, a typical time unit consists of one day. Other time units are possible, but to model time in our proposed scenario it is not important to know the exact duration of one unit. The value of a document not requested is decreased by the time factor τ multiplied with the number of time units since the last request was made. Thus we get:

$$\varphi' = (\varphi'_{ij}) \quad \text{where} \quad \varphi'_{ij} = \begin{cases} \varphi_{ij} + 1 & : \quad d_i = 1 \wedge d_j = 1 \\ \max\{\varphi_{ij} - \tau \cdot \varepsilon_j, 0\} & : \quad d_i = 1 \wedge d_j = 0 \\ \varphi_{ij} & : \quad d_i \neq 1 \end{cases} \tag{6.11}$$

The resulting Memory Matrix φ' is not symmetric any more. According to 4.4, φ serves as base for the prediction of documents. When φ is multiplied with a session vector we will obtain the conditional probabilities of all documents available. Usually, one will start with a session vector with only one entry which does not equal to zero. After that, the next and most probable documents may be predicted. Even though φ is not symmetric any more, the prediction algorithm works quite the same way as before (as described in 4.4.2).

6.1.3 Example of the Extended Algorithm

The learning phase and formula (6.11) should be illustrated by a simple example. Let us consider the following settings: $\mathcal{A} = 3, \tau = 0.1, \varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 1$ *day* (e.g. the last request on all documents is one day old).

Let us suppose the current Memory Matrix φ to be:

$$\varphi = \begin{pmatrix} 33.0 & 11.1 & 21.8 \\ 74.1 & 409.0 & 50.5 \\ 10.3 & 0.1 & 16.0 \end{pmatrix}$$

The session vector is given as $\vec{s} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$. In that case, the resulting matrix after a learning step would look like:

$$\varphi' = \begin{pmatrix} 34.0 & 11.0 & 22.8 \\ 74.1 & 409.0 & 50.5 \\ 11.3 & 0.0 & 17.0 \end{pmatrix}$$

Only the first and the last row have changed. Every element in the first and the last row that corresponds to a "1" in the session vector has been increased by 1. Every element that

corresponds to a "0" has been decreased by $\tau \cdot \epsilon_i = 0.1$. If such a value were to drop below "0", the new value would remain "0". The reason for this lies in the fact that the relevance of a document which has not yet been requested should not influence the prediction of other requests in the classification phase, since such a request could still take place later during the same session.

6.2 Implementation of the Advanced Model

6.2.1 The Extended Generator Module

The session generator module for the advanced modeling is more complex than the standard approach (described in 5.2). The additional parameters require a kind of memory, while the simple version generates new values without considering old ones.

The possible parameters are (the first four of them are the same as before):

- $\mathcal{A}$ (slen)

  Length of the sessions to be generated

- $\mathcal{B}$ (nump)

  Number of predictable elements of a session

- $\mathcal{R}$ (rf)

  Random factor

- $\mathcal{D}$

  Density, the number of settings within the session vector for $\mathcal{R} = 1.0$

- *Length of a time session* (lent)

  This is needed only for the subsequent prediction operation. After *lent* sessions, a mark is set to the output which can be understood as a time section break, e.g. a new day has begun. The time factor of the predictioner requires such time sections to be modeled adequately.

- ξ (RCP)

  The Request-Change-Probability presented in 6.1.1.

- *Number of test runs* (runs)

  This is required for modeling the user behavior, because the probabilities of sessions now depend on former settings.

In contrast to the standard implementation of the generator described in section 5.2.1, two different boundaries for the calculation of the session vector entries in the case of predictable elements are needed. So, instead of using *boundary_pred*, the extended implementation work with *boundary_set_1* and *boundary_set_0* respectively, depending on the setting of the preceding session. Therefore, also an additional memory is needed, stored in the array *lastvalues*.

For the semi-random elements, the modeling of time does not influence the generation of the session vectors and thus remains the same as in the simple version of the generator.

The documented source code for the implementation of the advanced model can be found in the appendix A.3.

6.2.2 The Extended Prediction Module

Also the predictioner module has to be extended in order to adequately model time and aging of data sets. As we did in section 5.2.2, we give in the following a short description of the used functions.

- *init_phi*

  Does not only initializes the matrix φ but also the array $lasv[]$ that is needed to store the preceding session vector in order to apply the extended learning procedures (described in 6.1.1).

- *evaluate_prediction*, *predict_with* and *predict_session*

  Those procedures are the same as for the standard modeling of the predictioner module.

- *learn_timeline* and *learn_timesession*

  The learning routines had to be extended to take care of different time sections. The recognition of a time mark (that corresponds for instance to a new day) leads to an adaptation of the matrix elements. According to the time factor, the entries of φ within the same row of a requested data set are decreased if they are not part of the current session vector.

- *main*

  The main procedure is called with an additional parameter: the *time factor*. This value can control the resulting prediction quality, as we will see in 6.3. The main procedure reads the standard input, makes a prediction for every element set to 1 of the session vector, evaluates the prediction quality and finally uses the session vector as input for the learning process.

It is important to see that φ is not symmetric anymore and that the entries of φ are now floating point numbers instead of integers. Within this modeling, the matrix must not become too large because we need to store the elements as *doubles*[2].

As result of the predicting process, the module denotes the overall prediction quality of the testing runs together with the global evaluation of the measuring variables *grp*, *gwp*, *grn* and *gwn* respectively, as mentioned in 5.2.2.

The interesting parts of the C-programming language code can be found in the appendix A.4.

6.3 Results of the Prediction Scenario

6.3.1 Introduction to the Test Scenario

We initiated millions of test-runs with different settings as for session generation and prediction parameters. To be able to compare the results, we will, in this section, present only settings with a session vector length of $\mathcal{A} = 125$, a predictable part of $\mathcal{B} = 50$ and a density value of $\mathcal{D} = 25$ while the other parameters $\mathcal{R}$, ξ, the threshold δ and the time factor τ were variable.

In the next section (6.3.2), we will work with real logs and additional test settings to reflect the proposed ideas. Due to the time modeling factors τ and ξ it has been necessary to re-start every test setting several times. The reason for this becomes clear when we consider a value of $\xi = 0.0$. Evidently, the first $\mathcal{B}$ values of all session vectors within one test-run are the same. We will only be able to achieve the overall statistical distribution of session values adequately, if the same test settings are run several times.

[2]Those are floating point numbers with double precision.

As a means of measuring the accuracy of the prediction algorithm we proposed the prediction quality PQ as quotient of correctly predicted and wrongly predicted values (analogous to 5.3.1).

Again, $PQ > 5$ is a supposition for applying prediction as a possibility to improve the performance of the Client/Server-communication.

6.3.2 Analysis of the Results

6.3.2.1 The standard parameters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{D}$ and $\mathcal{R}$

In section 5.3, the settings of $\mathcal{A}, \mathcal{B}, \mathcal{D}$ and $\mathcal{R}$ were being analyzed. In general, the absolute value of $\mathcal{A}$ is not important for testing the relative effect of the other parameters, especially with regard to the PQ-factor. Naturally, a high value of $\mathcal{B}$ leads to better prediction results, since the share of random values decreases.

In our tests, we analyzed different settings of $\mathcal{B}$ and quotients of $\frac{\mathcal{B}}{\mathcal{A}}$, but to realize the relevant points of our results we restrict ourselves to $\frac{\mathcal{B}}{\mathcal{A}} = 0.2$.

The threshold value δ for probabilities that lead to a prediction of the corresponding document is calculated on the basis of system-dependent factors and the kind of prediction strategy (see section 5.3.2). For our simulation, we have tested several settings. If not described otherwise, the results presented in this section are based on a threshold value of $\delta = 0.95$.

It is clear that for high values of the random factor $\mathcal{R}$ a high $\frac{\mathcal{B}}{\mathcal{A}}$ value will obtain much better results than for lower ones. It is also important to understand the influence of the density $\mathcal{D}$ on the PQ-factor. In general, $\mathcal{D}$ controls the absolute number of correctly and of wrongly predicted requests because prediction guesses are made only for the "1"s in the session vector and $\mathcal{D}$ determines the share of such settings. Furthermore, we can confirm the observation that the prediction quality decrease is more than linear with the value of the random factor $\mathcal{R}$ growing.

6.3.2.2 The time factor τ

The modeling of time and document aging by using the time factor τ resulted in an improved performance of the prediction qualities. Figure 6.1 shows a strongly increasing PQ with an increasing time factor for low values of the request change probability ξ. For high values of ξ, the influence of τ disappears. The figure is based on a fixed random value of $\mathcal{R} = 0.2$.

Nevertheless, it should be mentioned that the absolute number of predictions is much lower with high values of τ. This can be explained by improved prediction accuracy: even though there are less prediction tries, the resulting quality is much better. This means that the time factor is able to control the prediction strategy. On the one hand, similar to the threshold δ, a high setting of τ leads to a pessimistic algorithm with only a few but very good predictions. On the other hand, a low value of τ or of the threshold δ allows much more prediction proposals which are then of a low quality. We will come back to this problem in 6.3.2.4.

The advantage of the time factor τ in contrast to the threshold value δ is evident: The former can be set individually for every data-set available while the latter must be calculated from system-dependent resources (see 4.3.1 for a discussion of threshold dependencies).

6.3.2.3 The request change probability ξ

It was clear from the very beginning that the role of the request change probability ξ is as important as the random factor $\mathcal{R}$ is. Both parameters control the structure and entropy of the generated session vectors. A low value of ξ can even improve the prediction quality much more. While decreasing values of $\mathcal{R}$ increase the PQ factor polynomial, ξ can improve the quality exponentially. Figure 6.2 shows the prediction qualities of some settings of ξ for a random value of $\mathcal{R} = 0.3$.

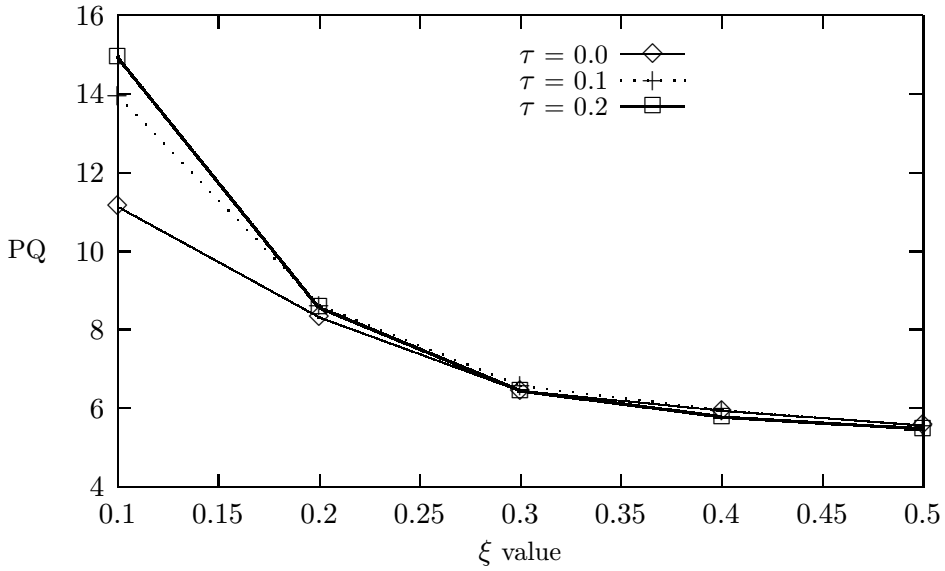

Figure 6.1: Influence of ξ and τ towards PQ

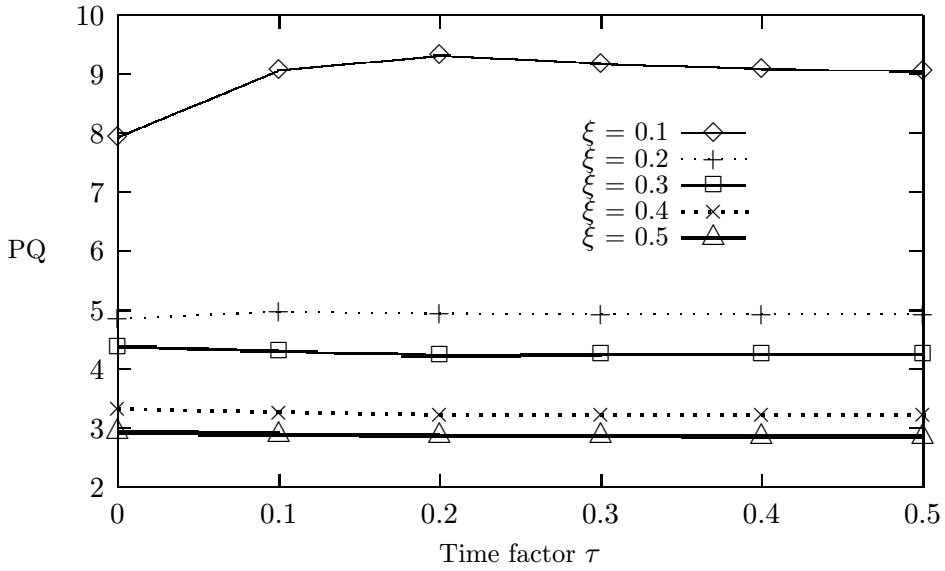

Figure 6.2: Results of different ξ and τ values

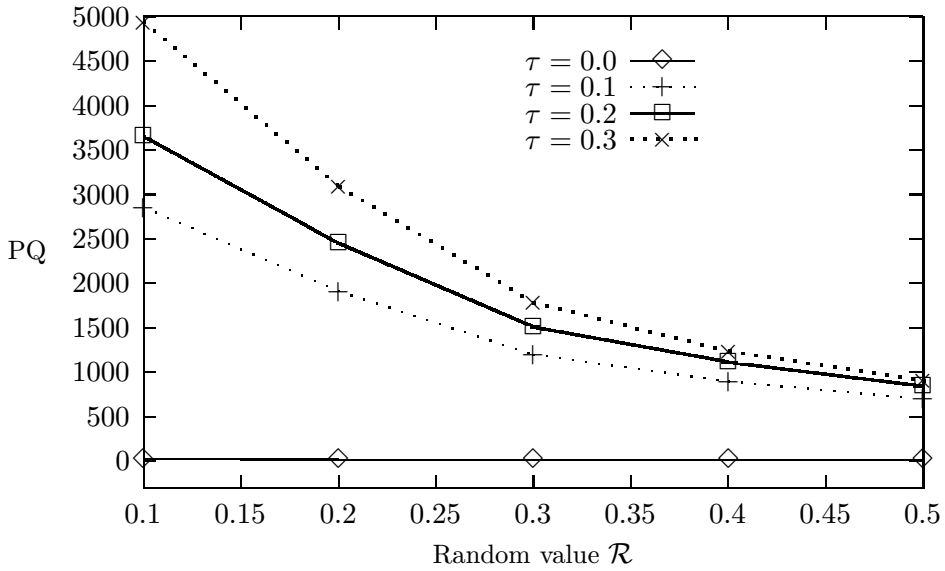

Figure 6.3: Prediction Quality for $\xi = 0.0$

In both figures (6.1 and 6.2) we have omitted the case where $\xi = 0.0$ because the prediction quality is naturally highly increasing for such a value and even a logarithmic integration of the values is not sufficient to obtain an acceptable graph.

Figure 6.3 shows several time factor and random settings for a vanishing value of ξ. Here again, the influence of the time factor decreases with high values of the random factor.

6.3.2.4 The threshold value δ

Every prediction algorithm must deal with a sort of threshold: What is a minimum value of the probability for future user-requests to process a pre-calculation or a pre- fetching?[3] We proposed a model where thresholds result directly from system- and document-dependent parameters in 4.3.1. This means that the threshold can not be set deliberately. Nevertheless, we will see that the influence of this variable is enormous. In the next paragraph, we will present a possibility for overcoming the problem of fixed threshold-settings. Figure 6.4 signals prediction qualities in dependence of time factors and threshold settings (for $\mathcal{R} = 0.1$).

In Figure 6.4 the absolute PQ-values are shown and it is interesting to see that high thresholds lead to better prediction results, but only for time factors of $\tau > 0.0$. What the graphics do not show is the absolute number of prediction tries (illustrated by Figure 6.5, the variable parameters are the same as in Figure 6.4, 10^5 test-runs for each setting).

Here, the graphical results are inverse to those of Figure 6.4. High threshold values prevent the prediction algorithm from making good - or bad - guesses for future requests. Thus, the costs[4] for good prediction quality consist in comparatively low numbers of correct guesses. This aspect must be kept in mind when proposing certain settings for prediction algorithms. If the costs of wrongly predicted pre-fetches are too high, even good prediction qualities would not be acceptable because of the worrying number of wrong guesses.

[3] A brief analysis of the third possibility, *piggybacking information* can be found in 4.3.4. More details on this topic can be found in [CKR98].

[4] A further discussion of prediction costs can also be found in [CI98].

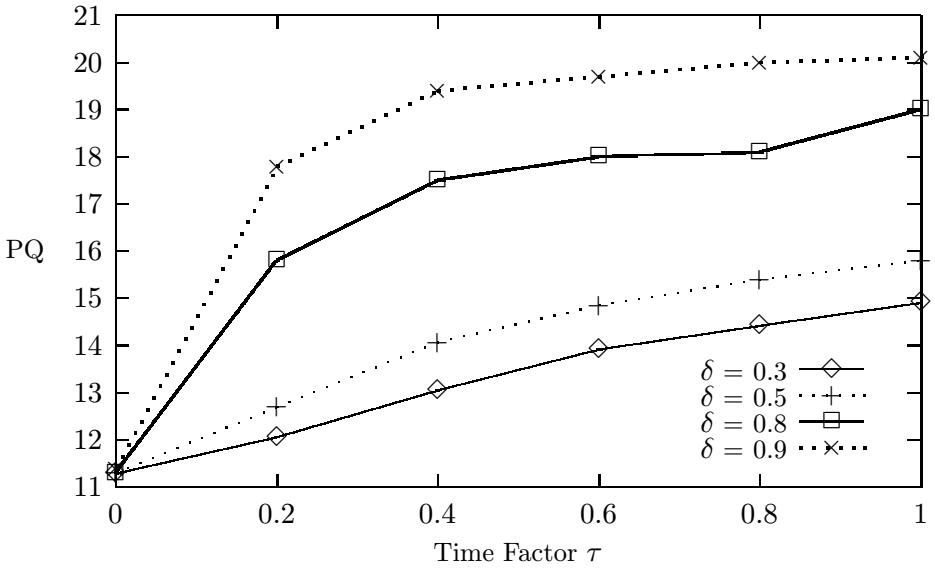

Figure 6.4: Prediction Quality with different threshold settings and time factors

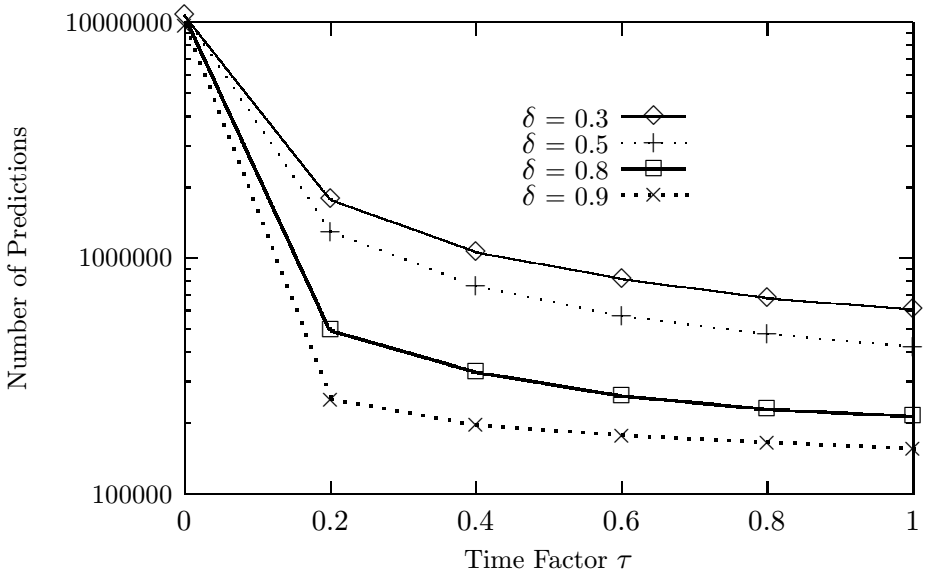

Figure 6.5: Total number of prediction tries

6.3.2.5 Interference of the controlling parameters $\mathcal{R}$, τ, ξ

There are some interesting observations beyond the isolated analysis of the single parameters. One aspect is the growing importance of τ and $\mathcal{R}$ while ξ decreases near zero (as illustrated by Figure 6.1 on page 54).

The question is: What can be done with such information? Our idea was to re-investigate those parameters from prediction results of real user logs. In 6.4, we will present those results. It is easier to find the best settings for τ if an exact or even only a vague impression of factors like $\mathcal{R}$ or like ξ can be assumed.

This leads us directly to a kind of hybrid approach: While trying to predict user-requests as accurately as possible the same requests are analyzed a posteriori to find the best parameter-settings of τ and ξ. Anticipating that user-behavior will be, in general, statistically constant - if the server information offer has not changed fundamentally - these parameter-settings can be used to predict the upcoming sessions. Otherwise, if the parameter results are too bad and do not show any solid, constant user-behavior, the prediction can be stopped completely.

6.4 Verification of the Model

6.4.1 Evaluation of Real User Logs

We extracted user-requests from some log files of a web-server[5] to verify our model. The first thing we did with the logs was a transformation to session[6] vectors without considering the request order. We randomly selected $\mathcal{A} = 125$ server documents in relation to the simulations of section 6.3 with an unknown share of potentially predictable data, but we set $\mathcal{B} = 125$ to receive results with high importance of ξ.

As average density we found $\mathcal{D} = 6$ with a high variance. To simulate different system-environment characteristics we applied several settings of the threshold δ in the prediction model. Figure 6.6 shows a selection of the *a posteriori* prediction quality PQ in dependence of the threshold.

The most obvious observation is the irregular behavior of the curves. For some time factors (e.g. $\tau = 0.5$) a lower threshold resulted in a better prediction quality, while normally high thresholds obtain the best results. For all these threshold settings the condition of $PQ > 5$ was not fulfilled.

It is interesting to look at the relationship between PQ and τ for a fixed threshold of $\delta = 0.6$. Figure 6.7 can thus be compared to Figure 6.2 on page 54 with an unknown implicit ξ value in the data.

The progression of the curve in Figure 6.7 corresponds to a low ξ value. If the implicit request change probability ξ was higher, the curve would be more similar to a straight line. In the real logs, under the conditions described, the best prediction results can be achieved with a time factor of $\tau = 0.1$.

6.4.2 Investigation of unknown parameters

The combination of $\mathcal{R}$ and ξ makes it very difficult to determine their values directly from the log files. Both parameters influence the prediction quality enormously, though not in the same way. Even though it is easy to determine the average number of requests in a log file, the standard deviation or the empirical variance, there is no easy or evident way to determine the settings of $\mathcal{R}$ or ξ without conducting test-runs in the prediction scenario.

[5] We used log files of selected homepage-requests on the web-server of the Computer Science Department of the University of Trier.

[6] Again, we used as time span of a session 30 minutes proposed by [SKS98].

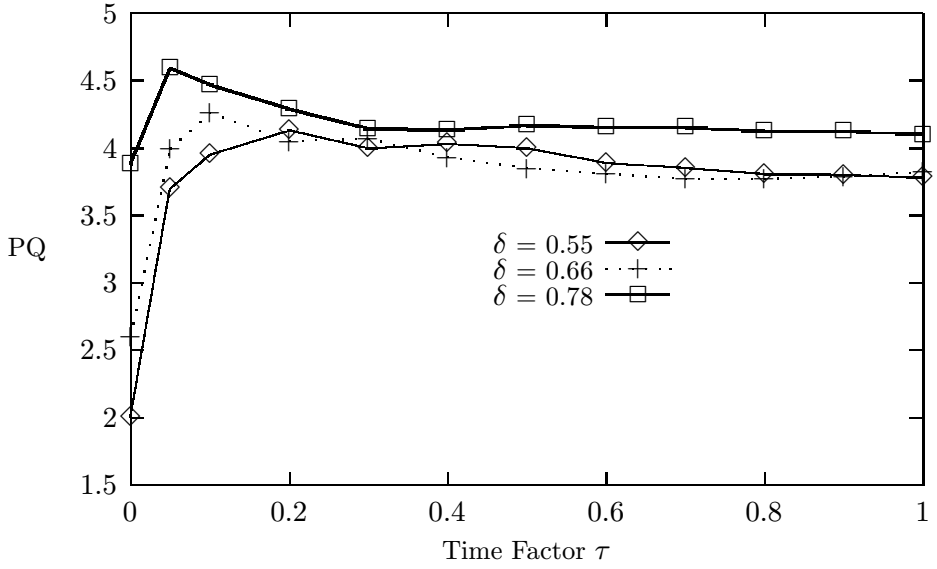

Figure 6.6: Prediction Quality of user-request for different threshold values

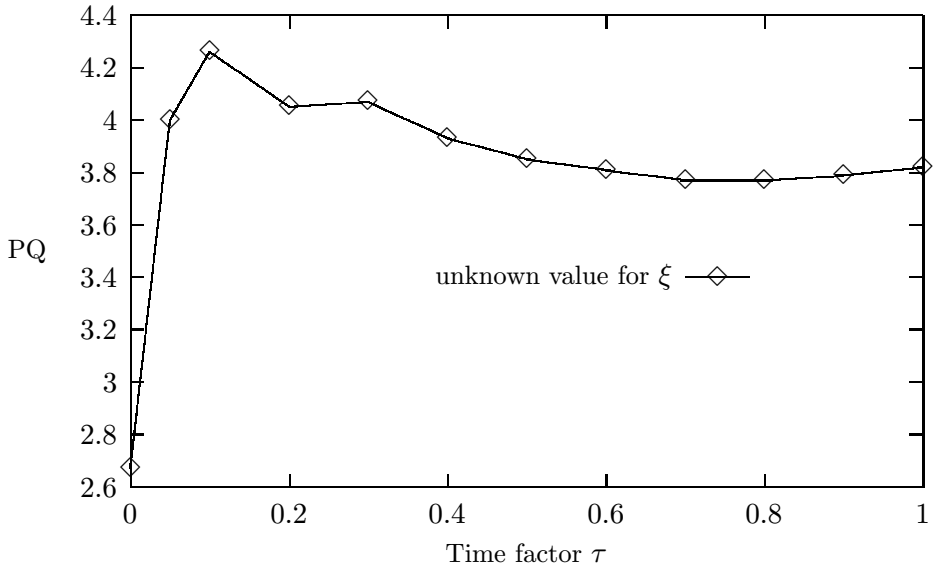

Figure 6.7: Time factor τ dependent results for real user logs with unknown ξ

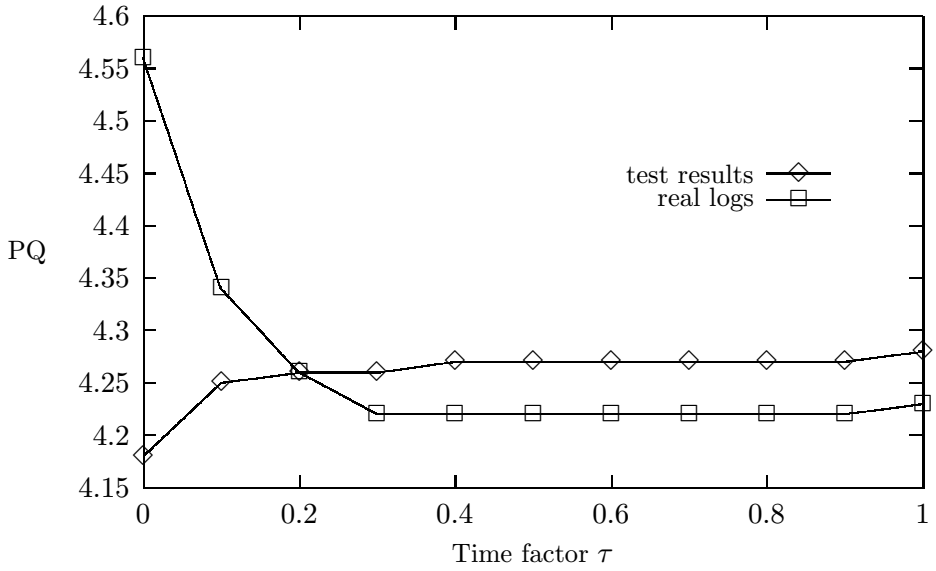

Figure 6.8: Comparison between test results and real log results

We know from section 6.3.2.4 that high thresholds show the most significant quality differences corresponding to the setting of τ. Therefore, we used a threshold value of $\delta = 0.95$ to determine ξ and $\mathcal{R}$, without even considering system-dependent costs. Certainly, this works only a posteriori and not while the prediction algorithm is actually running.

We started our test-runs with the settings described in 6.4.1 to find the best correspondence to the results found for the real logs. As a rule of thumb one can say that for every fixed value of ξ there can be found a corresponding value of $\mathcal{R}$ to obtain a predefined prediction quality PQ - and vice versa. But if we try to match also the absolute number of right (or wrong) guesses, only very few settings of $\mathcal{R}$ and ξ can be used.

Beginning with a great grid of values ($\mathcal{R} \in \{0.0, 0.1, \ldots, 1.0\}$, $\xi \in \{0.0, 0.1, \ldots, 0.5\}$) it becomes clear very fast that the best parameter settings to meet our conditions lay in the area of $\mathcal{R} \in [0.8, 0.9]$ and $\xi < 0.1$. The next step is the exact determination of both factors. Here, the really obtained prediction quality is the decisive factor. As a result we found $\xi = 0.0703$ and $\mathcal{R} = 0.885$. The time factor dependent curve we found in the test scenario and the corresponding values of the real server logs is shown in Figure 6.8.

The resulting difference between the test setting and the real server logs was very small ($< 0.5\ PQ$). For the use of prediction algorithm in every environment we propose to determine the unknown parameters as we did in the section. Herewith, we come one step further to one aim of all prediction ideas: reducing user perceived latency.

Part III

Hyperlink-Proposals

Chapter 7

Hyperlink-Proposal Theory

The history of human knowledge has so uninterruptedly shown that to collateral, or incidental, or accidental events we are indebted for the most numerous and most valuable discoveries, that it has at length become necessary, in any prospective view of improvement, to make not only large, but the largest allowances for inventions that shall arise by chance, and quite out of the range of ordinary expectation.

Edgar Allan Poe, The Mystery of Marie Roget
A sequel to "The Murders in the Rue Morgue" (1841)

After the general considerations about proposing hyperlinks in the introducing section (1.2) and the brief discussion of the special requirements of Hyperlink-Management Systems (in section 2.3) we come in this chapter to a derivation of a concrete *Hyperlink-Proposal Module* (HPM) based on some fundamental ideas of Case-Based-Reasoning (CBR, described in detail in section 3.1).

At first, we will highlight the research that have already been done on that area (described in section 7.1). Then, we describe a concrete, straightforward modeling of a Hyperlink-Proposal Module based on CBR in 7.2. Finally, it is necessary to delimit our approach to a direct CBR modeling and to discuss some of the differences in detail (7.3).

7.1 Hyperlink-Proposal Research

It is very important yet difficult to provide high-quality hyperlinks for a HTML-document. Hyperlinks dramatically improve content quality by presenting related work, contradictory positions, further information or simply by the continuation of the next page or by giving similar navigational information [Ric98]. The question of how a web author can easily find such information remains, though.

Research on the area of hyperlinks has been carried out since the introduction of the World Wide Web service to the Internet. Kaindl et. al. present a compact history of the progress made so far [KKD99]. Link retrieval research aims at generating hyperlinks if not completely automatically, at least with as little user interaction as possible. Very serious problems arise, though, when trying to retrieve hyperlinks of texts on a statistical base without any semantic knowledge. The results are of low quality [Glu89].

Allan classified link types into three major groups: *manual*, *automatic* and *pattern-matching* [All96]. The idea is to retrieve at least the easy-to-find links of the two latter groups and leave

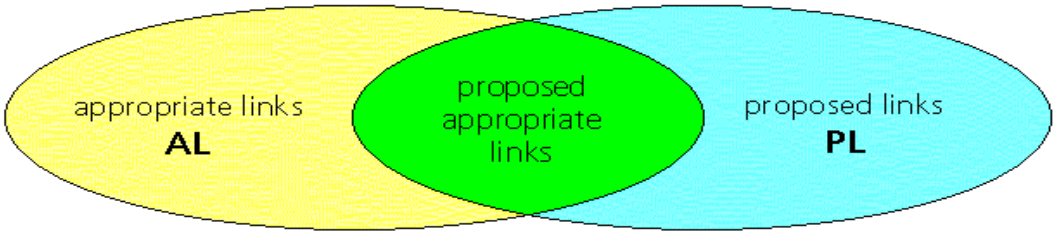

Figure 7.1: Measuring the quality of proposals: "recall" and "precision"

most of the former one to the user. This consideration is very useful even though it contains a disadvantage: the classification only works *a posteriori*.

We will focus on retrieving "automatic" links on a statistical base with approved methods (e.g. [Teb98] or [Ret99]). A different yet very promising approach is Chang's *HieNet*. A description of this system can be found in [Cha93].

In the following, we will describe the generation of hyperlink-proposals on base of some CBR-similar techniques and we will focus on the mechanisms to retrieve the corresponding knowledge from the hypertexts. Good examples for classical solutions without using CBR can be found in [MS93] or [ANR89].

It is a very complex task to measure the quality of hyperlink-proposal algorithms. Cleary and Bareiss mention as important factors the *recall*, the share of acceptable proposals of all good (or appropriate) links (AL) and the *precision*, the share of acceptable proposals of all proposed links (PL) [CB96]. Therefore, the recall R can be described with:

$$R = \frac{\sharp\{AL \cap PL\}}{\sharp\{AL\}}$$

and the precision P with:

$$P = \frac{\sharp\{AL \cap PL\}}{\sharp\{PL\}}$$

Certainly, those shares do only make sense if both sets AL and PL are not empty. Otherwise, recall and precision are not defined. Figure 7.1 should clarify the meaning of recall and precision.

Often, the quality of proposals in detail is only measurable by human experts. In general, the dilemma of hyperlink-retrieval is that a fully automated generation of links on a statistical base (see also [GZ93]) leads to relatively bad results in terms of precision and recall, while semantic approaches with very good results require a high degree of user interaction [PMH97]. If hyperlink retrieval is to be used as a tool for supporting web authors in easily adding up links, it would not be appropriate to require the time consuming formulation of a complete model of the semantic dependencies of a text. Web authors and users (readers) of hypertexts can also be supported without a generation of links. Zellweger et. al. introduce the concept of *Fluid Links* as a convenient way to deal with temporarily visible information [ZCM98].

The hyperlink-proposal module presented here can easily be adapted as part of a web authoring system like DAPHNE (section 2.2.2, further information can be found in [HZE99]). It is also appropriate to extend hyperlink management systems such as *Microcosm* [HDH96] with CBR-methods. Another possibility would be the use of CBR in combination with *Distributed Link Services* (DLS) presented by Carr et. al. in [CHH98].

7.2 Modeling of a Hyperlink-Proposal Module

7.2.1 Methodological Approach for Generating Links

In this chapter, we present a hyperlink-proposal system where all links that are part of any hypertext can be proposed no matter whether the target document is part of the system or not. Storing the possible hyperlinks is not enough, though. Of major importance is the fact that the system must store the relationship between the text and the associated links. This step is called a *learning process*. Therefore, the database can also be called a *knowledge-base*. The quality of this knowledge-base depends on the quality of the learning process: How can the information of the texts be combined with appropriate hyperlinks?

One possibility would be the human teacher. A person or a group of persons could derive the important information of the document manually. This process is very cost- and time-consuming and is not feasible in practice. Most of today's learning algorithms conquer the problem of high quality knowledge retrieval by extracting some information automatically on the basis of advanced heuristics. The learning process itself evaluates the relevance of the extracted information.

At first, the learning algorithm treats a text with hyperlinks as if it does not contain any link and derives the relevant information. Next, it proposes one or more hyperlinks and compares the result to the hyperlinks that the document in fact contains. By using this method, the learning process can be carried out without any human teacher. A disadvantage arises from the strong relationship between the quality of the learning process and the quality of the initial documents. Furthermore, only those hyperlinks can be proposed that are already known to the system and at the very beginning of the learning process there are no links to be proposed at all.

Our idea is to model the hyperlink generation problem as a kind of CBR-system (as described in section 3.1) and to use some experiences of CBR-research to retrieve high quality links as proposals for the web author. The written texts of the web authors are regarded as the *problems* and the hyperlinks within are the *solutions*. A complete hypertext can thus be viewed as a *case*. In the learning process, (statistical) text attributes are stored together with their attached hyperlinks into the knowledge-base, in the CBR-environment also called the *case-base*. In the classifying step, raw texts are presented to the system which proposes hyperlinks for the text as solutions. A possible *a posteriori* classification to evaluate the quality of the proposals is illustrated in Figure 7.2. We will come back to this idea in 9.1.

Furthermore, every new hypertext generates an additional solution - namely the link that refers to itself. We designed the CBR-model for the hyperlink environment with further differences to the classical approach. The cases are not stored *explicitly* into case-base but more *implicitly* without a strong relation between problem and solution. Only the importance of certain attributes for the affiliated solutions are represented in a weighted relevance matrix. The differences are discussed in section 7.3.

7.2.2 Representing Hypertexts as Cases

Cases are the most important data-structures of CBR-systems. Our approach provides a modeling of hypertexts as cases, where several textual attributes form the problem vector and the hyperlinks represent the corresponding solutions.

The raw text (without hyperlinks) as a whole is regarded as an information source to propose useful hyperlinks. At the current stage, there is no possibility to provide link proposals exactly for a concrete sentence of the text but all proposals together belong to the whole text.

In a further version of the HPM we plan to propose links not only by direct request but also continuously: Then, the proposals belong with a higher probability to the current paragraph and sentence the editor is actually working on.

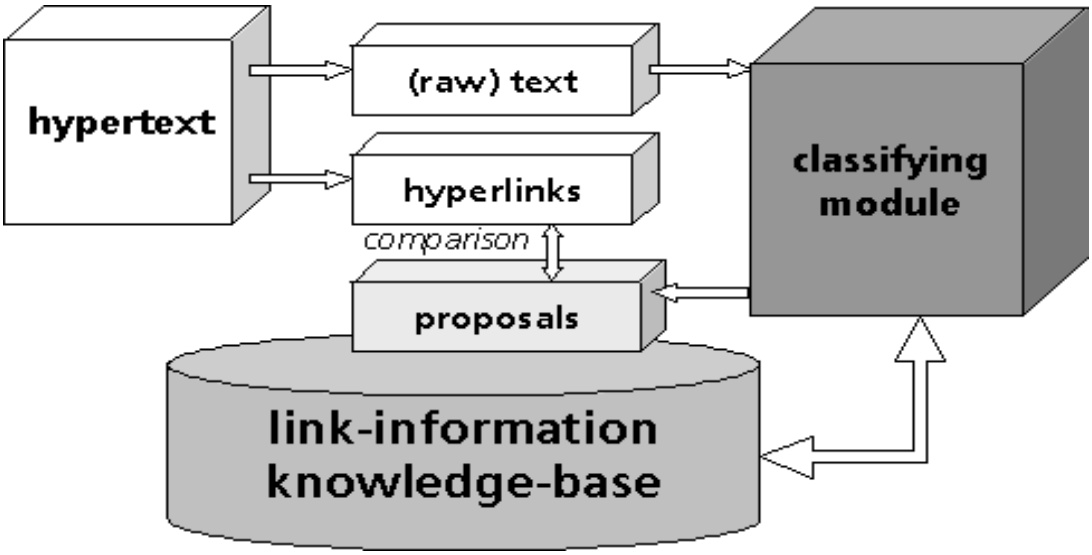

Figure 7.2: Learning and classifying applied to the hyperlink context

If we denote the main properties or *attributes* of the raw text within a hypertext $\mathcal{H}$ with $\mathcal{T}$, and the links with $\mathcal{L}$, then $\mathcal{H}$ can be regarded as the case: $\mathcal{H} = (\mathcal{T}, \mathcal{L})$. $\mathcal{T}$ represents the text in form of an attribute vector where the elements correspond to certain properties of the corresponding text, e.g. keywords, author information and so on. This data-structure will be the result of the knowledge-retrieval process described in detail in section 8.1. The hyperlink representation in form of the binary vector $\mathcal{L}$ consists of links known to the system. Every element set to "1" in $\mathcal{L}$ means that the corresponding hyperlink can be found within $\mathcal{T}$. Every element of $\mathcal{L}$ represents a target document (within the set of local documents of the own web-server or not)[1] and the corresponding default description. This description can be derived from the used online authoring system (if generated with such a tool) or from the concrete labeling of the first occurrence of the link (otherwise). Certainly, the user of the HPM can change the default labeling of the hyperlinks.

If the system currently provides a pool of $L_{\max}$ links, then the actual size of $\mathcal{L}$ is $L_{\max}$. The hyperlinks known to the HPM are increasing steadily, therefore also $\mathcal{L}$ has to be adapted dynamically (the consequences of this concept for the implementation of HPM are explained in detail in chapter 8).

Analogously, also the size of the vector $\mathcal{T}$ is changing dynamically, but as we will explain in section 8.1, $\mathcal{T}$ is not a binary vector. Attributes can also be fulfilled partially. We denote the current size of $\mathcal{T}$ with $T_{\max}$.

The core of our HPM consists of a relevance matrix ψ where the number of rows $L_{\max}$ and the number of columns $T_{\max}$ have to be increased dynamically. Every entry ψ_{ij} corresponds to the importance of the attribute j for the appearance of the hyperlink i, the "solution". Both sets, the attributes and the solutions, can take up additional elements any time (e.g. keywords in new HTML-texts and the hyperlinks within). In the sections 7.2.3 and 7.2.4 we will see how to calculate with these data-structures. In chapter 10 we will come back to the described modeling and we will compare the methodologies to the prediction concepts (described in chapter 4).

[1]Links that refer to documents within the own web site are also called *internal links*, otherwise they are *external*.

The sum of all relevance values of ψ according to the derived hyperlink-proposals must always fulfill the following normalizing condition:

$$\forall i, 1 \leq i \leq L_{\max} : \sum_{j=1}^{T_{\max}} \psi_{ij} = 1 \tag{7.1}$$

Matrices with this property are also called *stochastic* (see 3.2.1 for details). With (7.1) it will be possible to use elementary linear algebra to calculate the probabilities to be suitable to a presented text for all provided hyperlinks within the knowledge-base.

7.2.3 The Classification Phase

As we have seen in section 7.2.2, a case $\mathcal{H}$ consists of a pair of vectors, the problem $\mathcal{T}$ and the solution $\mathcal{L}$: $\mathcal{H} = (\mathcal{T}, \mathcal{L})$. The aim of the *classification phase* is to find the (unknown) solutions of a presented problem, i.e. to investigate the corresponding link-vector $\mathcal{L}$ for the presented text attributes $\mathcal{T}$ to form a hypertext $\mathcal{H}$. Thus, the classification phase should supply a resulting $L_{\max}$-vector $\mathcal{S}$ where the elements of $\mathcal{S}$ contain the probabilities that the corresponding hyperlink is applicable to the text $\mathcal{T}$. These proposed links are presented to the user in descending order of their request-probability.

To classify a problem $\mathcal{T}$ we first have to *separate* the attribute values of $\mathcal{T}$ with the threshold vector $\delta = (\delta^j)$, thus using the *attribute separation* function ζ. In the style of section 3.4 we want to speak of *fulfilled attributes* and *contradictory attributes* and thus use again the sets $\mathcal{F}$ and $\mathcal{C}$:

$$\zeta(\mathcal{T}) = (\zeta^j(\mathcal{T}^j)) \quad \text{where} \quad \zeta^j(\mathcal{T}^j) := \begin{cases} \mathcal{T}^j & : \quad \mathcal{T}^j \geq \delta^j \rightsquigarrow \mathcal{T}^j \in \mathcal{F} \\ 0 & : \quad \mathcal{T}^j < \delta^j \rightsquigarrow \mathcal{T}^j \in \mathcal{C} \end{cases} \qquad \forall\, 1 \leq j \leq T_{\max} \tag{7.2}$$

The threshold-elements δ^j control whether an attribute is fulfilled or not. It is not possible to abandon the separation step: The whole learning algorithm presented in 7.2.4 could then possibly not determine the new relevance results. The meaning of the sets $\mathcal{F}$ and $\mathcal{C}$ has slightly changed: In our modeling high values within the attribute vector $\mathcal{T}$ are regarded as "fulfilled" themselves while the classical CBR-approach only measures the similarity between two different attribute settings.

Analogously to (3.7) we will use the term $j \in \mathcal{T}^{\mathcal{F}}$ to denote that $\mathcal{T}^j \in \mathcal{F}$ and $j \in \mathcal{T}^{\mathcal{C}}$ respectively.

Finally, $\zeta(\mathcal{T})$ simply has to be multiplied with the relevance matrix ψ to obtain the proposal vector $\mathcal{S}$ indicating whether and in what degree the corresponding links $\mathcal{L}_i$ are useful to improve the quality of $\mathcal{T}$ or not.

$$\mathcal{S} := \psi \cdot \zeta(\mathcal{T}) \tag{7.3}$$

An additional threshold value[2] δ can control whether the probability values of $\mathcal{S}$ should be regarded as corresponding solutions (proposals) or not. Nevertheless, our HPM presents the proposals exceeding δ in descending order of their probability. We thought of a maximum of 20 proposals as useful in practice. This proceeding implies an additional (implicit) threshold if more than 20 hyperlinks exceed the threshold value.

[2]Also a threshold vector $\delta = (\delta_i)$ would be a good idea, where the different link-proposals could be qualified individually. We left this modeling to a further development phase of our HPM.

$\mathcal{S}_i$	$\mathcal{L}_i$	**Actions to be done**
0	0	nothing
0	1	relevance adaptation (*learning*)
1	0	nothing (adaptation in later version, see 10.3 for details)
1	1	nothing

Table 7.1: Relevance adaptation

7.2.4 The Learning Phase

In the *learning phase*, the relevance-indicating values of ψ have to be adapted so that the classification (as described in section 7.2.3) of a raw text (the "problem") $\mathcal{T}$ for a given case $\mathcal{H} = (\mathcal{T}, \mathcal{L})$ would result in a probability vector $\mathcal{S}$ with the property that all elements of $\mathcal{S}$ where the corresponding values of $\mathcal{L}$ are set to "1", are really exceeding the threshold value δ.

For this learning step, it could be necessary to increase both the number of rows $L_{\max}$ and the number of columns $T_{\max}$ within ψ dynamically. All new (dynamically generated) relevance values are then initialized with the values $\frac{1}{T_{\max}}$ to fulfill condition (7.1).

To be able to learn the new case $\mathcal{H} = (\mathcal{T}, \mathcal{L})$ its attribute-separated problem $\zeta(\mathcal{T})$ has to be classified. The resulting proposal vector $\mathcal{S}$ is then compared, element for element, to the real solution vector $\mathcal{L}$ of $\mathcal{H}$. Four cases have to be distinguished in each comparison of the corresponding vector elements i (Table 7.1).

The process of learning involves a change in the relevance values of ψ. If a suggested probability is too low so that the corresponding link will not appear on the proposal list of the HLM-system (second case of Table 7.1), the weight of the corresponding relevance entries in ψ must be increased. To fulfill condition (7.1), the remaining weights have to be decreased by the same amount.[3]

In the basic version of our HPM implementation, an element "0" in $\mathcal{L}$ did not lead to an adaptation of entries in ψ.

Let the i-th element of $\mathcal{L}$ correspond to a hyperlink that should be classified within the learning phase and that is not classified with the current weights.[4]

Then, the result c_i of the multiplication of the (attribute-separated) raw text $\zeta(\mathcal{T})$ would not exceed the threshold δ:

$$\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} \cdot \zeta^j(\mathcal{T}^j) =: c_i \leq \delta \tag{7.4}$$

Due to the learning rule of CBR (described on page 22) ψ must be adapted to ψ' so that we obtain:

$$\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi'_{ij} \cdot \zeta^j(\mathcal{T}^j) = \delta \tag{7.5}$$

Now, there a several possibilities to change the weights of ψ to achieve the result of equation (7.5).

[3] In the classical CBR-approach at this step a conflict is possible: What can be done if all elements (or too much) of the presented problem-vector expect an increasing of weights? In our implementation due to the dynamically increasing attributes this conflict can be omitted (see also the example described in section 7.3.2).

[4] Otherwise, the learning process would be superfluous, or - it has already been done!

The most simple and straightforward ones are the *proportional distribution* and the *constant distribution*, both presented in the following subsequent sections. For further distribution ideas see [Haf93].

7.2.4.1 The proportional distribution

If the missing values of the weights are distributed according to the height of their current value and the current weights are *natural*, i.e. $c_i > 0$ and $\chi := \sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij} > 0$, then the new values of the corresponding link i relevance weights should be set to:

$$\forall j \in \mathcal{T}^{\mathcal{F}} : \psi'_{ij} = \psi_{ij} \cdot \frac{\delta}{c_i} \tag{7.6}$$

Otherwise, if the weights are not natural, other kinds of distribution might be possible, e.g. the "constant distribution" discussed in 7.2.4.2.

At first, we have to verify that the classification leads exactly to the threshold value δ for the settings of (7.6):

$$\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi'_{ij} \cdot \zeta^j(\mathcal{T}^j) = \sum_{j \in \mathcal{T}^{\mathcal{F}}} \frac{\delta}{c_i} \cdot \psi_{ij} \cdot \zeta^j(\mathcal{T}^j) = \frac{\delta}{c_i} \cdot c_i = \delta \quad \square \tag{7.7}$$

Due to the stochastic-property (7.1) of ψ we find:

$$\psi'_{ij} = \begin{cases} \psi_{ij} \cdot \frac{\delta}{c_i} & : \quad \mathcal{T}^j \in \mathcal{F} \\ \psi_{ij} \cdot \frac{\chi + \rho}{\chi} & : \quad \mathcal{T}^j \in \mathcal{C} \end{cases} \tag{7.8}$$

where $\rho = (1 - \frac{\delta}{c_i}) \cdot \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij}$

The proof of the stochastic property is rather clear:

$$\begin{aligned} \forall i \in \{1, 2, \ldots L_{\max}\} : \quad \sum_{j=1}^{T_{\max}} \psi'_{ij} &= \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi'_{ij} + \sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi'_{ij} \\ &= \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} \cdot \frac{\delta}{c_i} + \sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij} \cdot \frac{\chi + \rho}{\chi} \\ &= \frac{\delta}{c_i} \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} + \frac{\sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij} + (1 - \frac{\delta}{c_i}) \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij}}{\sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij}} \sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij} \\ &= \frac{\delta}{c_i} \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} + \sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij} + (1 - \frac{\delta}{c_i}) \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} \\ &= \sum_{j \in \mathcal{T}^{\mathcal{C}}} \psi_{ij} + \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} \\ &= 1 \qquad \square \end{aligned} \tag{7.9}$$

7.2.4.2 The constant distribution

Let us denote $\kappa = \sum_{j \in \mathcal{T}^{\mathcal{F}}} \zeta^j(\mathcal{T}^j)$ as abbreviation. It is always true that $\kappa > 0$, because attributes are not fulfilled, i.e. do not belong to $\mathcal{F}$, if their value is "0". Furthermore, the set $\mathcal{C}$ must not

be empty. If this is the case, there is no possible weight distribution. See also 7.3 for general limitations of the presented approach.

Here, we are deriving the constant value Δ that describes the changes of the relevance weights ψ_{ij} for $j \in \mathcal{F}$ to "learn" the classification of the link corresponding to i:

$$\begin{aligned}
& \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} \psi'_{ij} \cdot \zeta^j(\mathcal{T}^j) = \delta \\
\iff \quad & \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} (\psi_{ij} + \Delta) \cdot \zeta^j(\mathcal{T}^j) = \delta \\
\iff \quad & \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} \psi_{ij} \cdot \zeta^j(\mathcal{T}^j) + \sum_{j\in\mathcal{T}^{\mathcal{F}}} \Delta \cdot \zeta^j(\mathcal{T}^j) = \delta \\
\iff \quad & c_i + \Delta \cdot \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} \zeta^j(\mathcal{T}^j) = \delta \\
\iff \quad & c_i + \Delta \cdot \kappa = \delta \\
\rightsquigarrow \quad & \Delta = \tfrac{\delta - c_i}{\kappa}
\end{aligned} \tag{7.10}$$

Therefore, for the new settings of ψ' we get (analogously to (7.8)):

$$\psi'_{ij} = \begin{cases} \psi_{ij} + \frac{\delta - c_i}{\kappa} & : \quad \mathcal{T}^j \in \mathcal{F} \\ \psi_{ij} - \frac{\sum_{j\in\mathcal{T}^{\mathcal{F}}} \frac{\delta - c_i}{\kappa}}{\sum_{j\in\mathcal{T}^{\mathcal{C}}} 1} & : \quad \mathcal{T}^j \in \mathcal{C} \end{cases} \tag{7.11}$$

We still need to provide the proof that the stochastic property is also for ψ' true:

$$\begin{aligned}
\forall i \in \{1,\ldots,L_{\max}\} : \textstyle\sum_{j=1}^{T_{\max}} \psi'_{ij} &= \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} \psi'_{ij} + \sum_{j\in\mathcal{T}^{\mathcal{C}}} \psi'_{ij} \\
&= \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} \left(\psi_{ij} + \frac{\delta - c_i}{\kappa}\right) + \sum_{j\in\mathcal{T}^{\mathcal{C}}} \left(\psi_{ij} - \frac{\sum_{j\in\mathcal{T}^{\mathcal{F}}} \frac{\delta - c_i}{\kappa}}{\sum_{j\in\mathcal{T}^{\mathcal{C}}} 1}\right) \\
&= \textstyle\sum_{j\in\mathcal{T}^{\mathcal{F}}} \psi_{ij} + \sum_{j\in\mathcal{T}^{\mathcal{F}}} \frac{\delta - c_i}{\kappa} + \sum_{j\in\mathcal{T}^{\mathcal{C}}} \psi_{ij} - \sum_{j\in\mathcal{T}^{\mathcal{F}}} \frac{\delta - c_i}{\kappa} \cdot \frac{\sum_{j\in\mathcal{T}^{\mathcal{C}}} 1}{\sum_{j\in\mathcal{T}^{\mathcal{C}}} 1} \\
&= \textstyle\sum_{j\in\mathcal{T}^{\mathcal{C}}} \psi_{ij} + \sum_{j\in\mathcal{T}^{\mathcal{F}}} \psi_{ij} \\
&= 1
\end{aligned} \tag{7.12}$$

due to the stochastic property of ψ $\square$

After such a learning step has been taken, hyperlinks for new documents can be proposed by using the knowledge-base in the classification phase (see 7.2.3).

In general, the user of the hyperlink-management system (HLM, 2.3) accepts or rejects a proposal of the classification algorithm and thus plays the role of a human teacher. The learning algorithm must compare the proposal of the system to the reaction of the human user and thus adapt the relevance values according to the proposed algorithm. Figure 7.3 shows the described ideas in graphic form.

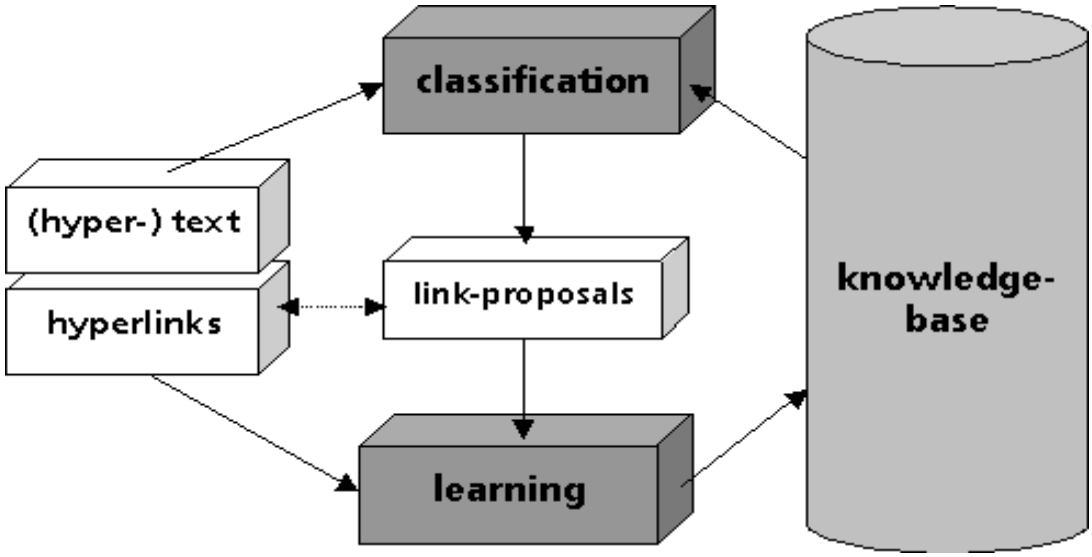

Figure 7.3: Principle structure of a statistical-based hyperlink-proposal algorithm

In practice, the learning and the classification steps are combined so that the system is able to make proposals even though the knowledge-base is still rather small.

A description of an implementation and an evaluation of the described principle can be found in the chapters 8 and 9.

In chapter 10, we will come back to the main ideas of this approach and we will show that these algorithms can be improved by drawing parallels from the - at first glance - quite different field of request-prediction.

7.2.4.3 Examples

Starting from the relevance matrix:

$$\psi = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.7 & 0.3 & 0.0 \end{pmatrix}$$

we consider the attribute vector:

$$\mathcal{T} = \begin{pmatrix} 0.9 \\ 0.2 \\ 0.8 \end{pmatrix}$$

and the attribute thresholds $\delta^1 = 0.7, \delta^2 = 0.75$ and $\delta^3 = 0.8$.

Then, according to (7.3), the classification of the attribute-separated $\mathcal{T}$ leads to:

$$\mathcal{S} := \psi \cdot \zeta(\mathcal{T}) = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.7 & 0.3 & 0.0 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.0 \\ 0.8 \end{pmatrix} = \begin{pmatrix} 0.51 \\ 0.74 \\ 0.63 \end{pmatrix}$$

Let the classifying threshold be: $\delta = 0.7$ and the link-vector $\mathcal{L}$ that is to be "learnt":

$$\mathcal{L} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

The classification of link 1 corresponds to the first line of Table 7.1 (on page 68) because of: $\mathcal{S}^1 = 0.51 < 0.7 = \delta$. For the link 2 we find: $\mathcal{S}^2 = 0.74 > 0.7 = \delta$ and $\mathcal{L}^2 = 1$, therefore, we have case four of Table 7.1. Only the link 3 leads to an adaptation of ψ:

$\mathcal{S}^3 = 0.63 < 0.7 = \delta$, but $\mathcal{L}^3 = 1$. We show in the following both learning strategies, the proportional and the constant one.

7.2.4.3.1 Weight adaptation with proportional distribution According to equation (7.8) we find for the example settings:

$$\psi'_{ij} = \begin{cases} \psi_{ij} \cdot \frac{\delta}{c_i} = \psi_{ij} \cdot \frac{0.7}{0.63} & : \quad j \in \mathcal{T}^{\mathcal{F}} = \{1,3\} \\ \\ \psi_{ij} \cdot \frac{\chi+\rho}{\chi} = \psi_{ij} \cdot \frac{0.3-\frac{1}{9}\cdot 0.7}{0.3} & : \quad j \in \mathcal{T}^{\mathcal{C}} = \{2\} \end{cases} \quad \text{for } i = 3$$

This leads to the new matrix:

$$\psi' = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.778 & 0.222 & 0.0 \end{pmatrix}$$

In case of the proportional distribution the "old" value ψ_{ij} plays an important role. We can see in this example, that the value of 0.0 can not be increased with the proportional distribution strategy. Nevertheless, it is easy to see that the resulting matrix ψ'_{ij} fulfills the stochastic property.

7.2.4.3.2 Weight adaptation with constant distribution For the constant distribution of the additional relevance weights we come to the following results. The starting point is equation (7.11):

$$\psi'_{ij} = \begin{cases} \psi_{ij} + \frac{\delta - c_i}{\kappa} = \psi_{ij} + \frac{0.7-0.63}{1.7} & : \quad j \in \mathcal{T}^{\mathcal{F}} = \{1,3\} \\ \\ \psi_{ij} - \frac{\sum_{j \in \mathcal{T}^{\mathcal{F}}} \frac{\delta - c_i}{\kappa}}{\sum_{j \in \mathcal{T}^{\mathcal{C}}} 1} = \psi_{ij} - \frac{2 \cdot \frac{0.07}{1.7}}{1} & : \quad j \in \mathcal{T}^{\mathcal{C}} = \{2\} \end{cases}$$

which leads to the resulting matrix:

$$\psi' = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.741 & 0.218 & 0.041 \end{pmatrix}$$

Again, it is easy to see that the stochastic property can still be provided.

In contrast to the proportional distribution the result seems to be more natural. For our implementation of the HPM we found that the proportional distribution poses more problems due to the manifold 0-values in ψ. Therefore, we placed the emphasis in an early step on the constant distribution. In section 7.3.2 we present an example where the constant distribution strategy leads to extreme weights and how to overcome such problems. Furthermore, the overall proposal results evaluated in chapter 9 are rather promising.

7.3 Delimitation of CBR and the presented HPM

7.3.1 General Comparison Aspects

It is very important to outline the differences between the proposed HPM and the classical Case-Based-Reasoning systems (already described in 3.1). One major difference has been mentioned in 7.2.1 already. Usually, a new problem has to be compared with the cases of the case-base to find the most similar one. Then, its solution is transferred to the new problem.

We are not comparing explicitly all cases of the knowledge-base with the new problem. Instead, we transfer directly the importance of attributes for the corresponding solutions of the stored cases to the new problem by multiplying the relevance matrix ψ with the new problem vector $\mathcal{T}$.

Therefore, we lose degrees of freedom concerning the learning algorithm. Detailed analyses of dynamical learning algorithms for Case-Based Reasoning systems can be found in [Haf93]. One modeling problem, for instance, consists of the possibility for relevance weights to become smaller than 0.0. This situation is illustrated by the example of section 7.3.2.

In general, our proposed approach seems to be simpler and faster and the results are not too poor (see the usability evaluation of the algorithms in chapter 9). But we left the possibility to extend the system later to implement more sophisticated algorithms with more complex learning rules.

Another difference is the understanding of cases themselves. Normally, a case consists of a problem together with its solution. In the HPM, also the new document (as part of the problem) generates a new hyperlink that points to itself (as an additional solution). This is very important for future classification steps but it has no meaning for the current problem solving.

Some of the main advantages and disadvantages of CBR for hyperlink-proposal use are listed below. On the one hand, there are the following advantages of using CBR to built a HPM:

- CBR research proves serviceable for extended use (several years) and for use in many areas
- It requires no special user interaction
- The learning process takes place implicitly (i.e. while the user accepts or rejects a link proposal)
- Core functions of CBR are fast and easy to implement
- CBR-systems “learn” to adapt personalized link favorites
- Due to the case model all kinds of (typed) links can be found - not only those that point to documents on the local web side
- The link proposal system can be applied to existing web sites by filling the case-base with hypertexts
- CBR can be used in conjunction with other methods (e.g. the concept model of [CB96])
- It is not restricted to language characteristics as described in [KKD99][5]
- Link proposals of CBR do not determine non-ambiguous sources of the hyperlinks so that the same keyword can (implicitly) generate more than one link for the hypertext[6]

[5]We based our studies on English and German documents. For the latter, we have an additional restriction: only words with beginning capitalized letter (beside the first word of a sentence) are regarded as potential keywords. In German nouns always start with a capitalized letter.

[6]The number of keywords varied between five and hundred.

On the other hand, there are also some disadvantages of the CBR-similar approach:

- The proposed links do not belong to a small fragment of text but to the whole page so that special link positions must be adapted manually
- CBR generates (many!) link proposals ordered by the probability of their usefulness. Therefore, the classical measurements of recall and precision cannot simply be applied
- The quality of the proposals depends on the structure of the case-base. If it is empty, the system cannot make any proposals. If it overflows, some cases will be "forgotten"

7.3.2 Example of a Critical Learning Phase

In paragraph 7.2.4.3.2 we mentioned already that there are potentially problems for certain weight distribution strategies. Especially, when there are very few attributes the process to distribute the weights becomes rather unexpected.

Let us consider, for instance, a case where the attribute vector looks like this:

$$\mathcal{T} = \begin{pmatrix} 0.93 \\ 0.84 \\ 0.88 \\ 0.84 \\ 0.96 \\ 0.99 \\ 0.89 \end{pmatrix}$$

For the given attribute thresholds $\forall\ 1 \leq j \leq T_{\max} : \delta^j = 0.8$ all existing attribute would be classified as "fulfilled" and thus $\forall\ 1 \leq j \leq T_{\max} : \mathcal{T}^j \in \mathcal{F}$.

Such a situation poses problems to the learning algorithm: if there are any links that are not proposed already it would not be possible to change any weights of ψ_{ij} to increase the resulting value. To fulfill the stochastic property it would be necessary to decrease the weights of contradictory attributes, but in the given vector $\mathcal{T}$ there are no such attributes! In [Haf93] these weight changes are called *impossible requirements*.

Nevertheless, it is very improbable that $\mathcal{T}$ does not propose all of the existing links already. Furthermore, in our modeling of the hyperlink-proposing scenario only few links can be proposed (for instance the 20 links with the highest classification results). It would be an appropriate solution for the described problem to limit the number of links to be classified. Then, the weights of the remaining solutions could serve to be decreased for fulfilling the stochastic property.

There is another critical situation, although very improbable, it could happen that relevance weights become negative.

Let the relevance matrix be:

$$\psi = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.1 & 0.1 & 0.8 \\ 0.4 & 0.4 & 0.2 \end{pmatrix}$$

Furthermore, let the separated attribute vector be:

$$\zeta(\mathcal{T}) = \begin{pmatrix} 0.7 \\ 0.7 \\ 0.0 \end{pmatrix}$$

The classification results, herewith, in:

$$\mathcal{S} = \psi \cdot \zeta(\mathcal{T}) = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.1 & 0.1 & 0.8 \\ 0.4 & 0.4 & 0.2 \end{pmatrix} \cdot \begin{pmatrix} 0.7 \\ 0.7 \\ 0.0 \end{pmatrix} = \begin{pmatrix} 0.49 \\ 0.14 \\ 0.56 \end{pmatrix}$$

The link 2 should be classified with a threshold $\delta = 0.77$.

The constant weight distribution strategy, thus, leads to the new matrix:

$$\psi' = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.55 & 0.55 & -0.1 \\ 0.4 & 0.4 & 0.2 \end{pmatrix}$$

The value $\psi'_{23} = -0.1$ has become negative due to the high threshold value δ and the rather low attribute values. For $\delta = 0.7$, for instance, the resulting value of ψ' would be: $\psi'_{23} = 0.0$.

What could be the meaning of such a relevance weight? The existence of a certain attribute preserves the corresponding link from being proposed. Even though this kind of modeling is possibly manageable we can omit this critical situation within our scenario. As stated earlier, for our HPM we increase both the number of rows (corresponding to the number of known links $L_{\max}$) and the number of columns (corresponding to the number of textual attributes $T_{\max}$) dynamically.

Additional rows are necessary to treat hyperlinks that were extracted from the texts to model the learning phase adequately.

New columns are the result of the classification phase: if new important keywords are recognized they lead to additional columns (see section 8.1.2 for details).

With such dynamically increasing matrices it is easy to overcome the problem of negative weights. Every additional column decreases the probability for weights to become very small. Furthermore, an additional column can be modeled as part of the learning phase. If the new column corresponds to a keyword attribute it is very simple to add an appropriate keyword (for $\mathcal{T}^j$ that is not part of the current text and thus belong to $\mathcal{T}^j \in \mathcal{C}$. If necessary, more than one additional column can be added.

The practice of our HPM shows that negative weights are very improbable and they did not occur outside of our testing laboratory.

Chapter 8

Implementation of the Proposal-Module

Whole and unity; thing or entity or being.
Every whole is a unity and every unity that is divisible is a whole.
For example, the primitive concepts, the monads, the empty set,
and the unit sets are unities but not wholes.
Every unity is something and not nothing.
Any unity is a thing or an entity or a being.
Objects and concepts are unities and beings.

KURT GÖDEL, recorded by Hao Wang
in "A Logical Journey" MIT Press (1996)

After the derivation of the formulas to provide both a classifying phase and a learning phase in chapter 7 we come in this chapter to a description of a concrete implementation.

A very important part of the whole process is the transformation of hypertexts into an attribute vector representation. Section 8.1, thus, describes the process of retrieving knowledge from texts. Details of the module implementation are discussed in section 8.2. The evaluation and the refining of the measurement terms are provided in chapter 9.

8.1 Knowledge Retrieval of Hypertexts

A very crucial question in the context of our CBR-similar Hyperlink-Proposal Module (HPM) is the transformation of the problem into certain attributes that represent it. Therefore, it is necessary to retrieve the relevant information of the corresponding hypertexts. In the following subsections, we present a technique for knowledge retrieval that is based on statistical and syntactical considerations. As described in 7.2, a problem is modeled as a $T_{\max}$-vector $\mathcal{T}$ where $T_{\max}$ denotes the number of attributes used to describe the problem. Every element of $\mathcal{T}$ must be normalized into the interval [0,1]. The solutions of a case are also represented that way. Here, we speak of an $L_{\max}$-vector $\mathcal{L}$, which is mostly a binary vector with the i-th element set to 1 if and only if the solution corresponding to i solves the Problem $\mathcal{T}$, and 0 otherwise. The variable $L_{\max}$ is the number of all solutions available from the relevant case-base. Obviously, a problem $\mathcal{T}$ can have up to $L_{\max}$ solutions. In the presented concept, the

solutions are hyperlinks within the (problem-)files. Therefore, a text $\mathcal{T}$ is represented with $T_{\max}$ attributes and can contain up to $L_{\max}$ hyperlinks $\mathcal{L}$.

To specify the attributes of hypertexts we chose the following settings (if available):

- Every important (weighted) keyword of the document is regarded as an attribute
- Every author of the document forms an attribute
- The creation date and the expiration date of a document are subsumed to one attribute *validation*
- The publishing state[1] and the version are combined to form the attribute *availability*
- The department information is one attribute *structure*, but we set the restriction that each document must not belong to more than one department

Thus, we made a statistical approach to apply our HPM scenario. Semantic methods could have been modeled at this point too. An evaluation of our settings will be given in chapter 9.

8.1.1 Modeling of the HPM in General

Our idea was the straightforward, relatively simple adaptation of the CBR-concept to the context of hyperlink management systems (HLM) and the evaluation of its potentials. Our HPM should only be one possibility to propose links and it must collaborate with other methods. As the environment to implement a prototype version we selected an HLM-system as presented in [HHR99] where problems arising from different supported languages were also modeled.

As already mentioned in the beginning of this section, we choose the keywords of the documents to specify the attributes of hypertexts. Other types of meta-data can also be used here (e.g. *author information, creation date, expiration date*) if the document has been generated with an online authoring tool, e.g. DAPHNE (see description in section 2.2.2).

The following subsections describe in detail the methods to derive information from texts, especially from hypertexts.

8.1.2 Keyword Extraction

A very difficult problem is the extraction of keywords from a document on the basis of statistical distribution [CB93]. We decided to carry out a full text analysis with a special treatment of HTML-tags. All words beside HTML tags, comments and the stopwords (e.g. a multilingual list from CD-ISIS [CDI99]) were treated as potential keywords.

Beside the classical stopwords we regard in the context of hyperlink management also terms like “homepage” and the company’s name as unusable for classification of whole web pages by keywords. A “word” in this context is a sequence of letters without special characters (e.g. hyphens). The following Table 8.1 shows the - arbitrary chosen - weights we attached to every word in a text depending on its relative position between tags. These settings reflect that keywords in titles or headlines are more important than those in the body. In the next version of our HPM, the weights of the keywords should also be part of the learning process.

The number of occurrences of a word in a document multiplied with the settings of Table 8.1 results in an absolute weight. Words within the anchor-tag for hyperlink references (HREF) are not considered because their information results already in a concrete link.

Only the words that exceed a minimum threshold δ_D (depending upon the document length) are treated as keyword attributes. In addition - if there are too many keywords - only the ones with the highest weights are selected. At the end all weights are proportionally transformed into the interval [0,1]. Thus, all weights are divided by the maximum value among them.

[1] Allowed states are for instance: *generation in progress, reviewed, exported to the Internet, published.* For further details see also 2.2.2.

Position within tag	Weight
<TITLE>	100
<META> (description)	50
<H1>	5
<H2>	4
<H3>	3
<H4>	2
<BODY>	1
<A HREF>	0

Table 8.1: Distribution of keyword weights

Some essential points of the keyword extraction are:

- Keyword extraction does not consider ambiguities in the meaning of the words that are spelled the same
- Abridgments and acronyms can be defined in the text itself and will thus be treated like stopwords
- Even if two texts only have few keywords in common, they can share their solutions in the HPM
- The use of full form lexicons for treating different kinds of word-flexion [Spr92] should be applied in the future

8.1.3 Author Information

If the author of an HTML document is known, this information will form an additional attribute for the corresponding text representation. If there is more than one author, the system is able to take care of the varying relevance of the different authors (e.g. the first author is weighted by $\frac{1}{1}$, the second by $\frac{1}{2}$, the third by $\frac{1}{3}$ and so on; or all authors are weighted by 1 in case of alphabetically sorted authors).

This information is retrieved from the corresponding online authoring system (for instance the system "DAPHNE" described in 2.2.2).

8.1.4 Document Validation

The idea to consider the *age* of an HTML-file to form an attribute arises from the perception that the relevance of the content depends on its creation and expiration time. This is also true for the links contained in these documents. To get a linear value between 0.0 and 1.0 for the validation of a file, we calculate the *distance in time* between the current time ("now") and the lifetime of the document. There are three possibilities as described in Figure 8.1.

1. If the *creation time* or the *publishing time* of a document represented as $\mathcal{T}$ is in the past and the expiration time is in the future (T2 = "now"), the *validation* value ν of $\mathcal{T}$ will result in:

$$\nu = 1.0 \tag{8.1}$$

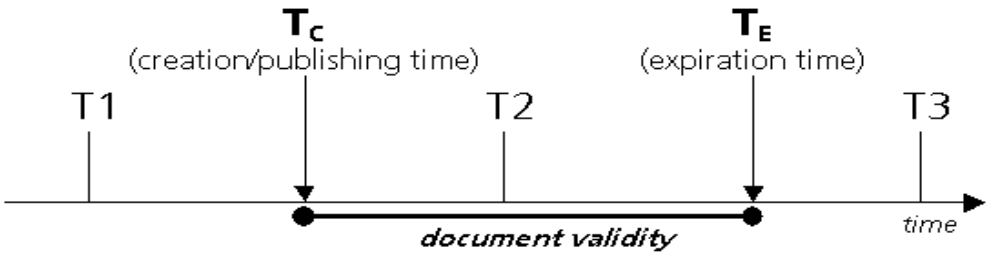

Figure 8.1: Timeline to calculate validation

2. If the publishing time of $\mathcal{T}$ is in the future, e.g. $\mathcal{T}$ is not yet visible in the Internet/Intranet (T1 = "now") the validation ν is calculated as:

$$\nu = \frac{T_E - T_1}{T_C - T_1} \tag{8.2}$$

3. If the document is already obsolete, the validation attribute of the corresponding documents will obtain the following value (T3 = "now"):

$$\nu = \frac{T_3 - T_C}{T_3 - T_E} \tag{8.3}$$

Further information about this topic can also be found in [HHR00].

8.1.5 Departmental Information

If possible, additional information of the document *structure* is also used as an attribute for the problem representation. Here, the idea is that those documents that are positioned "deeper" in the (tree) structure of a website obtain a lower value as those on the top level. The usability of links with regard to the structure depends on how general the contents of the concerned web pages are. It is more probable that links on the top level are not as specific as those in other positions, even though this is not always true. Very often, files all over the website refer to the document-root of the tree (the "home-link").

Again, the departmental information can mostly not be retrieved from the HTML-files and the directory structures themselves, but only with the aid of the corresponding online authoring system that provided the departmental information (see also [ZHH99]).

8.2 Hyperlink-Proposal Module Implementation

We implemented a first version of the algorithm as described in section 7.2 to propose links on the basis of an existing HLM-system written in Java [HHR99]. Here, the links are represented as objects with source document and target document as variables. Every link has its own description and refers to a default label. For simplicity, the generated link proposals supply only one (default) description.

8.2.1 General Concepts and Ideas

From the very beginning it was clear that we should provide and implement the HPM as part of a universal hyperlink-management system (such as the HLM described in 2.3). A standalone

version would not be able to treat the manifold additional information such as author or departmental information adequately even though this might be a good idea in general.

Nevertheless, we needed in an early step already useful results to see if our ideas were practical. Therefore, we decided to scan existing websites. The HLM-import-functions should read several international known websites (e.g. "www.acm.org") and treat them as if they were part of the HLM internal structure. Certainly, most of the additional information was not available.

The next step was very exiting. We extracted the hyperlinks from the scanned hypertext and run the classification step from the raw texts. Afterwards, in the learning step, we compared the proposed links to the really existing hyperlinks within the original pages and the HPM "learnt" the corrects solutions. The results of these evaluation steps are provided in chapter 9.

The whole HLM-packet is a pure Java implementation. Our HPM is one module of the packet. It is based on the data structures such as "Meta" and "Link" that we sketch below briefly. We omit here a detailed description of the implementation of these concepts and refer to [RHH99]. A short introduction to this topic can also be found in 2.3.

Furthermore, we needed to program some additional features in order to treat the imported website adequately. For instance, we associated a "default-author" to the new HTML-files.

A rather unusual concept of our modeling are the dynamically increasing rows and columns of the matrix ψ. Even though they help to manage difficulties in the learning step (as described in 7.3.2), they lead to a slightly more complicated implementation.

One the one hand, we can not use static data structures because the number of rows and columns are not limited. On the other hand, the system load increases dramatically if the growth of the matrix ψ can not be controlled.

In a further step of the implementation, we will not use the array structure any more, but we will decide heuristically which text attributes (that form the columns) and which hyperlinks (that form the rows) are appropriate and which are not. Furthermore, we sketch briefly the idea of decreasing the matrix rows and columns again in the outlook.

8.2.2 An Exemplary Java Implementation

To illustrate the derived concepts of the HPM we present our shortened implementation in form of Java source code in the appendix B.1. This is a simplified version of the main programming code for use in the HLM-project as mentioned above. Some important parts of the class definition with variables and methods are discussed below.

The matrix *psi* (ψ) is used a central data structure with dynamically increasing number of rows and columns. In Java, the implementation for such a kind of array is rather simple. The only difficulty consists in taking into consideration the different sizes of this structure in subsequent programming steps. For instance, the number of rows and columns of the matrix between learning and classifying could have been changed.

A text (with or without hyperlinks) is represented as its meta-object ("Meta"), not only containing a reference to the content of the text itself, but also to some meta information, e.g. the author(s), a description, creation and expiration date and so on. For simplicity and clearness of studying the sources, only two special attribute treatments - besides the most important *keyword attributes* - are left in the code of the appendix: the document validation (8.1.4 *validity* and the departmental information (8.1.5) *departmental structure*.

An object class "Link" represents a hyperlink that points to a document. A link consists of its description (label) and certain properties, for instance the language or the version. The link can receive its label as the description of the first occurrence of the hyperlink within a document or directly by the author.

The most interesting methods of the HPM class are "classify" and "learn":

- *classify*

  This method takes a meta-object (text) as input and results in a link list, where all elements exceed the class threshold probability. The list is sorted in descending order by the value of the corresponding probabilities of the links.

- *learn*

  The "learn"-method takes a meta-object and a list of links as input parameters. The aim of the learning process is to adapt the weights within ψ so that a subsequent classification process would result in - at least - the given link list. Additional proposals are acceptable at this point.[2]

The concrete weight adaptation of ψ is realized within the method *changeWeights*. It is called for a complete row of ψ that corresponds to a single link. The sum of all weights within a row must remain exactly 1 after each learning step in order to fulfill the stochastic property of (7.1).

The method *zeta* performs the function ζ defined in formula (7.2). It separates the attribute values into fulfilled and contradictory elements. Only the former values remain while the latter ones are reset to 0.

The core of the information retrieval task is done by the method *retrieveAttributeVector*. The keywords of the given text are weighted according to their position within HTML-tags[3]. Those weights build up the resulting vector together with some additional attributes (as mentioned above).

Further information especially on the basic data structures of the hyperlink-management objects can be found in [RHH99].

[2]Even though the overall number of proposals must be limited in order to be practically usable for online authors.

[3]This step is performed by the subroutine *getKeywordWeight*. The basis of this function is the weighting task described in section 8.1.

Chapter 9

Hyperlink-Proposal Evaluation

There are problems to whose solution I would attach
an infinitely greater importance than to those of mathematics,
for example touching ethics, or our relation to God,
or concerning our destiny and our future;
but their solution lies wholly beyond us
and completely outside the province of science.

Johann Carl Friedrich Gauss, quoted by J. R. Newman
in "The World of Mathematics", New York (1956)

In chapter 7 we derived the theory to build a Hyperlink-Proposal Module (HPM) and in chapter 8 we discussed a concrete development of such an HPM.

In this chapter, we try to measure the usability of the proposed links. At first, we discuss some general aspects of the evaluation in section 9.1. Then, we refine the classical measurement terms in section 9.2 and finally, the evaluation results are presented in section 9.3.

9.1 General Evaluation Aspects

For an advanced evaluation of the presented model on base of a large amount of data we tested the system "a posteriori" on existing web pages.[1] The advantage of this processing is the "real-life" usability of the approach. The disadvantage arises from missing data: world wide well-known web-pages that we scanned from the net do not provide helpful information as *author, validity, structure* and others as presented in section 8.1. In section 9.3 we will compare the results of those websites with our own web-presence ("www.ti.fhg.de") where we could use some of the additional information.

To evaluate the quality of the HPM, we extracted the links within several HTML-files, classified the raw texts using our model, and finally compared the classification results to the existing hyperlinks according to the learning and classifying steps demonstrated in 7.2.2.

Certainly, the HPM produces many proposals, arranged according to the probability of their usefulness. The user should be able to scroll in the proposal list. In terms of the classical measurements of *recall* and *precision* this is rather problematic. What are the "appropriate" hits? A web author can, for instance, select several links from the proposal list and can create

[1] We mentioned this idea already in section 8.2.

additional links herself/himself as well. It would be neither correct to count all proposals made nor to ignore the additional ones. We will highlight this problem in 9.2.

Due to these difficulties, we decided to split the results of our web scans into several parts. There are no unequivocal recall and precision values. Nevertheless - compared to some results so far (e.g. Cleary et al. [CB96]) - we think that our approach may be used highly advantageously and may obtain implicitly a very good recall and a high precision.

For a general evaluation of the proposal-quality we provide some important aspects of the presented HPM that should be kept in mind as a summary.

- The HPM is easy to use because it provides proposals without any prior user interaction (i.e. no construction of semantic models etc.)
- All link proposals belong to the whole document. Therefore, the web author has to replace the links if she/he wants to have it at a specific position within the text. This is an inconvenience of the current implementation that will be reduced in further versions (see also 7.3 for some ideas to overcome this problem)
- The HPM can only propose links as accurate as the corresponding data in the case-base. It can never propose a hyperlink which has not already been learnt
- The system makes many proposals, ordered by the probability of their usefulness. An evaluation in terms of recall and precision is, therefore, rather problematic

Next, we will highlight the measurement methods to quantify the proposal-quality of the HPM.

9.2 Measurement Refinements

Due to the difficulties in determining the quality of link suggestions we introduced new terms on base of the probabilities of the link proposals.

The classical terms to measure the proposal-quality are the *recall*, the share of appropriate proposals of all good links, and the *precision*, the share of appropriate proposals of all proposals. A detailed explanation of these terms has been provided already in section 7.1. Beside these, Cleary and Bareiss mention *ease of use* and *thoroughness* as important factors of link proposals [CB96].[2]

9.2.1 Quantified Cumulating Recall

The *Quantified Cumulating Recall* (QCR) is derived from the term *recall*, that denotes the share of appropriate links among all "good links" of the hypertext. The term is explained in detail in section 7.1.

The idea is to extend the meaning of the "recall" as a dynamically growing factor. If applied not only to every link in a given hypertext but to several documents, the QCR becomes an increasing curve. The gradient of that curve signals the recall-share of the proposals as one measurement of the quality of the HPM results.

Usually, the QCP is presented as a two-dimensional curve in a Cartesian coordinate-system. The x-axis denotes the number of documents classified. The corresponding value on the y-axis is summarizing all recall-values of the documents classified so far, where, certainly, the recall of each document belongs to the interval $[0, 1]$. Therefore, we can speak of the "quantified cumulating recall". For the calculation of the recall we do not count those links that are not yet known to the system and therefore in principle not suggestible. Such a modeling would not be "fair" to the HPM. Nevertheless, each new document generates a virtual hyperlink to

[2] We will focus mainly on the refined recall and precision measurements as described in chapter 7.

itself so that the HPM could propose this link even though it might not be contained in any hypertext known to the system.

At the beginning, the corresponding knowledge-base must not contain any specific information about the documents of the website that is to be "learnt". This would generate extremely good proposal results that are unobtainable in practice. Normally, the QCR of the first document results in the value "zero".[3] Evidently, the main-diagonal would be the maximum-result of the proposal-quality.

Further information can also be found in the following description of the QCP (9.2.2).

9.2.2 Quantified Cumulating Precision

Analogous to the QCR (9.2.1), we also refined the term "precision". As described in 7.1, the precision signals the usefulness of the proposed links.

Again, we see that the process of learning has an elementary influence on the quality of the precision and therefore, we summarize the precision of all documents classified so far.

It would also be possible to denote the single precision of every new document in graphic form, but the resulting curve could veil the real changes in the classifying process. In fact, it is very important to clarify the influence of the learning steps on the quality of the proposal results. For practical use, it might be most important to realize how fast the learning of new topics works.

For documents, where no proposals could be generated, we treat the recall and the precision as 0, even though this situation is really better than proposing only wrong links. Our real-world evaluation shows that it is very improbable - except for the very beginning - to receive no link proposals from the HPM. Therefore, the setting of precision and recall in the case of no proposals is statistically insignificant.

Section 9.3 presents the evaluation of our HPM implementation and demonstrates the QCR and the QCP by means of several examples.

9.3 Evaluation Results

9.3.1 General Evaluation Concepts

We decided to choose the web-pages of some well-known institutions to evaluate the proposal-qualities of the HPM. Additionally, we compared those results where no special information of online authoring systems were available for our testing to our own web-presence.[4]

In the graphical representation of our testing results, we show the link proposal results in terms of QCR and QCP for the following websites, respectively:

- Association for Computing Machinery (ACM)
  http://www.acm.org
- The World Wide Web Consortium (W3C)
  http://www.w3c.org
- Association for the Advancement of Computing in Education (AACE)
  http://www.aace.org

[3] Even though this is not a strict rule. If the knowledge-base already contains general information, it is possible that the classifying steps return link proposals may be appropriate.

[4] Even for our own web-pages we used only some additional information and not all the features available from DAPHNE (2.2.2). Therefore, those testing results could certainly be reconstructed with other online authoring systems.

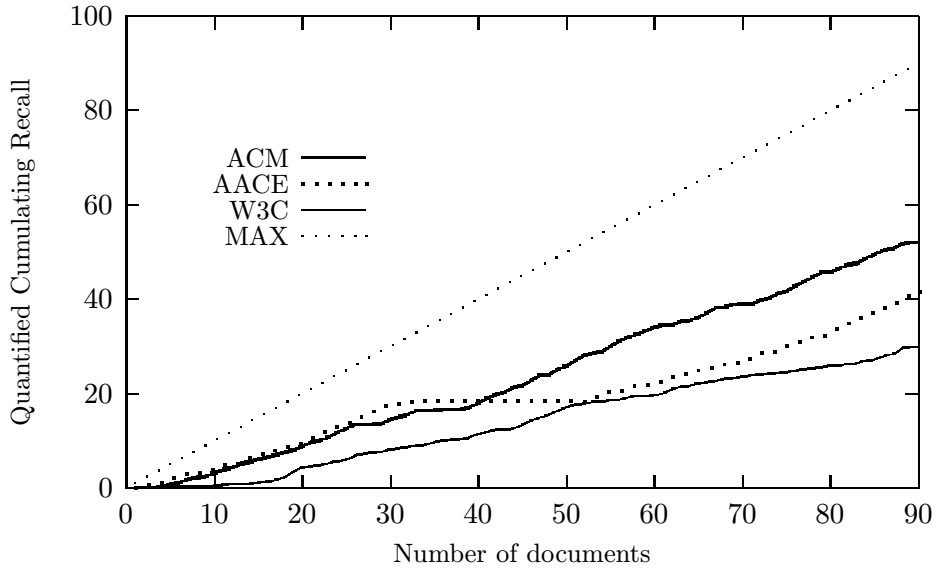

Figure 9.1: Quantified Cumulating Recall (ACM, AACE, W3C)

- Institute of Telematics (TI-FHG)
  http://www.ti.fhg.de

We decided to analyze the first ninety pages of each of the websites above beginning with the document root and scanned in breadth-first-search manner. Only HTML-files were considered and for the external web scans only the keyword attributes were available.

For the scan of (TI-FHG) we did not begin with the document root, but with the "no frames" root page to get comparable results to the external web scans.

Furthermore, we only followed the hyperlink references within the same domain, but we stored all of the links within the pages as potential solutions for the HPM so that even external links could be proposed.

All of the subsequent web-scans took place in the last quarter of 1999.

9.3.2 Graphical Results of the Evaluation-Domains

The results in terms of QCR and QCP (as defined in 9.2) of our implementation of the HPM are illustrated in Figure 9.1.

Here, the external QCR evaluation of ACM, AACE and W3C is described. Evidently, the proposal-quality is not constant. In the beginning, where only a few documents are classified and learnt, the AACE pages perform the best proposals.

From the fortieth document on, the ACM QCR is much better and AACE recalls are very poor up to the fiftieth document. The W3C-pages are not classified as good as the other ones due to the manifold links. Furthermore, the site-index pages of W3C are part of the set of HTML-files to be classified. Those pages disturb the algorithm because the content of the keywords can not lead to all the hyperlinks within the pages. Therefore, also the subsequent pages are classified less accurately.

In the next graphics, we compare the QCR of the best-performing external website (ACM) to the scan of TI-FHG with usage of some additional information (Figure 9.2).

We can see that the resulting QCR-curves are rather similar. The classifying steps of the TI-FHG pages are more efficient than the ACM ones from the very beginning, but this picture

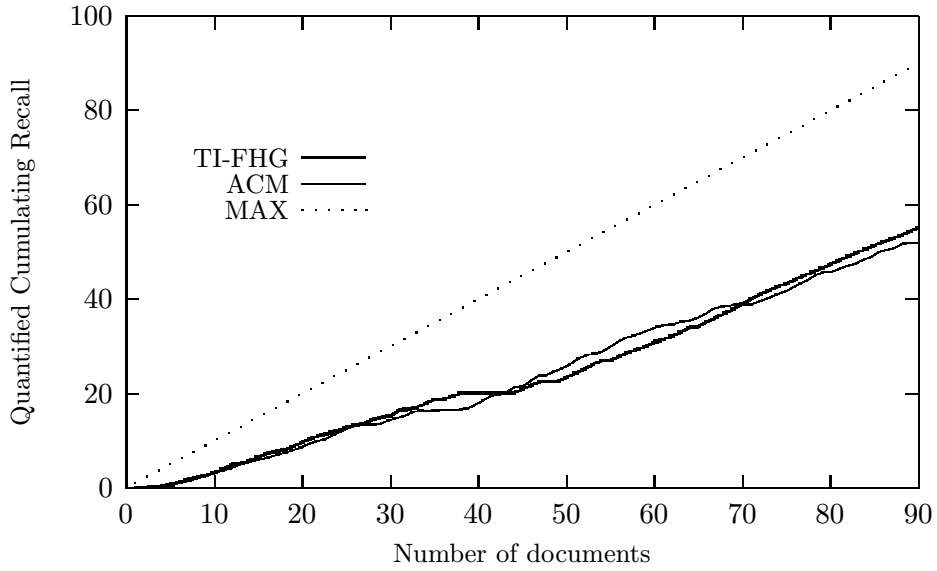

Figure 9.2: Quantified Cumulating Recall (TI-FHG, ACM)

changes between the 45th and the 70th document. Here, the ACM learning and classifying steps are very good and the TI-FHG pages contain summary-pages and overview-files, where lots of hyperlinks can be found that are not always derivable from the textual context.

In general, textual pages with many keyword-attributes that are set (i.e. many high values in the attribute vector) and rather few hyperlinks perform much better HPM results than those with only a few keywords but manifold links.

In the next two figures, we show the corresponding QCP results of the presented pages. We start again with a comparison of the external websites (Figure 9.3).

In general, most of the QCP-curves are not as high as the QCR ones. A system like the HPM proposes several hyperlinks according to the probability of the usefulness for the given text represented by the attribute vector.

For the precision of the proposals it is necessary that most of the presented links are appropriate for the given text. For instance, if we considered more than 20 proposals per document the recall would extremely increase while the precision would decrease.

In Figure 9.4 we compare the precision of the best-performer W3C according to QCP to the TI-FHG proposals.

It is obvious that the additional information from the online authoring system leads to higher QCP-values, while the QCR-values are rather high anyway. Even at the beginning of the classification process the TI-FHG pages perform very good.

To be able to analyze the overall quality of the proposals it is necessary to compare the QCR and the QCP-curves in a common graph. The result for TI-FHG is presented in Figure 9.5.

It is easy to see here that the QCP-values are higher than the QCR-values. In general, we can say that the usage of additional information leads to much better precision than recall. The recall-values themselves are rather high, but not necessarily much better than the results of web-scans without additional data.

The QCR- and the QCP-curves for the ACM web-scans (illustrated by Figure 9.6) demonstrate a typical result of the HPM for keyword-attributes.

The ACM pages contain relatively much information per link so that the QCR (and thus the single recall-values) are acceptable. But due to the missing additional information the

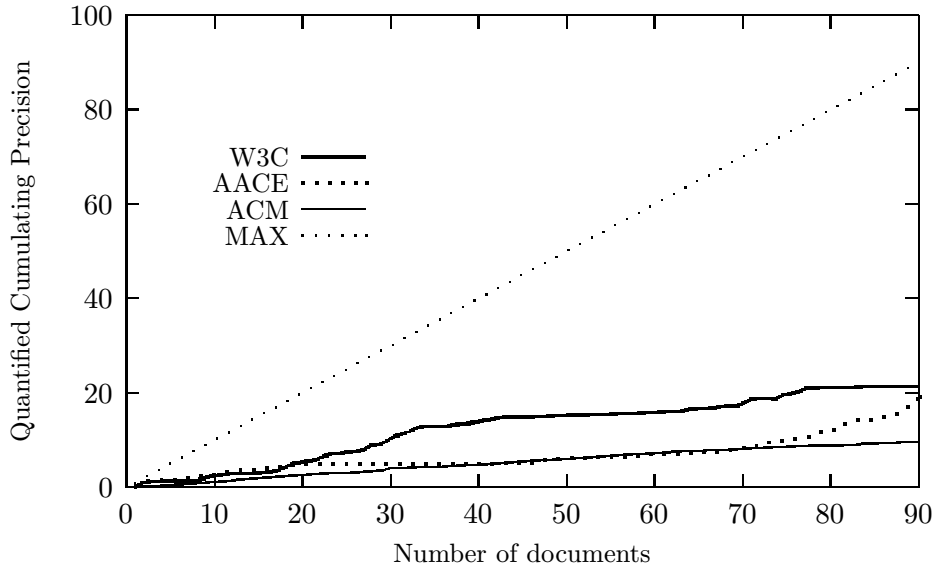

Figure 9.3: Quantified Cumulating Precision (ACM, AACE, W3C)

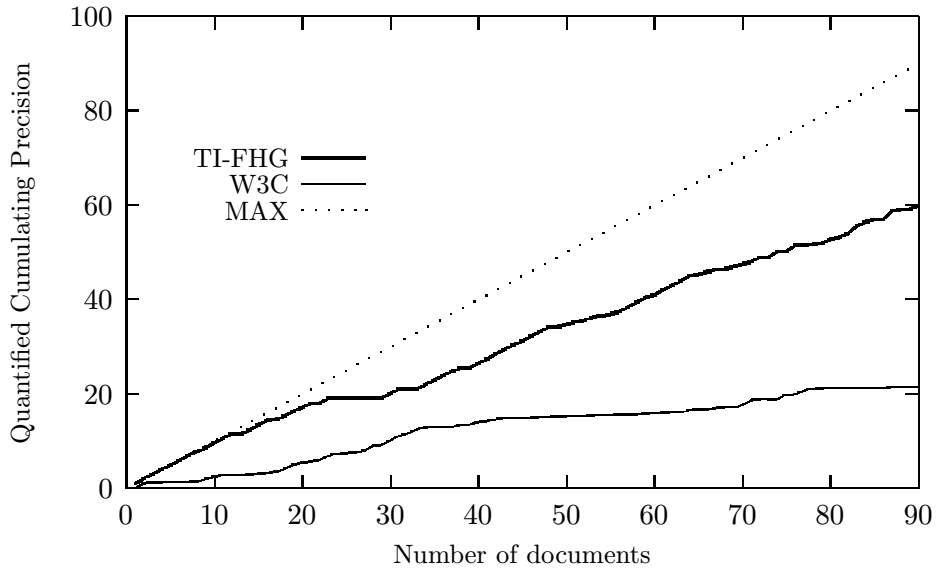

Figure 9.4: Quantified Cumulating Precision (TI-FHG, W3C)

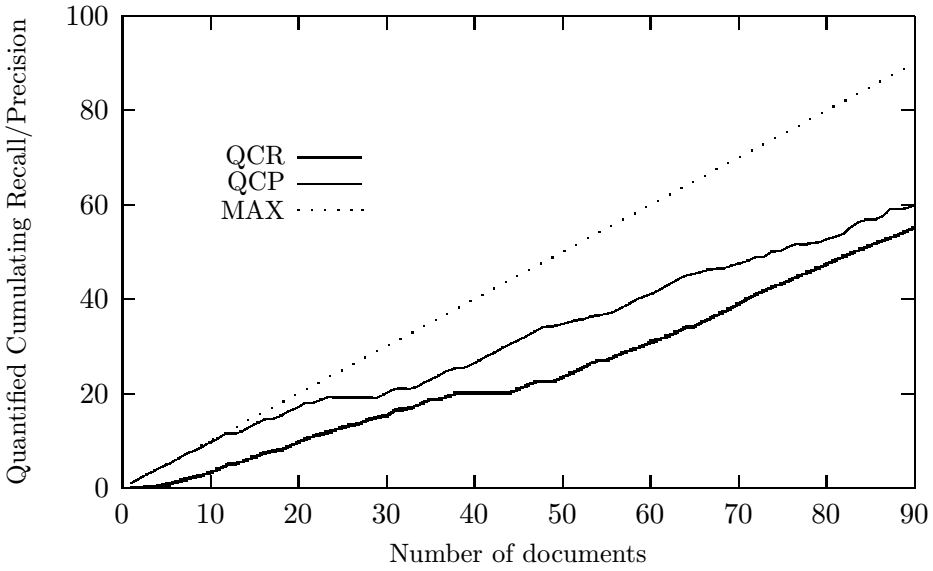

Figure 9.5: Proposal Qualities of TI-FHG

precision performs not as well. The HPM proposes most of the good links, but too much additional links that might not be appropriate.[5]

Furthermore, Figure 9.6 shows that the discrepancy between recall and precision is growing with the time. The reason for this is laying in the architecture of the HPM. At the very beginning, only few links are proposed while with the growing case-base also the number of proposals increases. Certainly, for the current version of the HPM we count only the first 20 proposals as acceptable. But the possibility to scroll in the proposal list make this processing obsolete. It is not feasible in practice that the user has to search in a proposal-list that contains too much links. Therefore, the QCP is so important that it qualifies the QCR.

In Figure 9.7, another possibility is shown. The QCP- and the QCR-curves are rather similar. Thus, the proposal-qualities of the W3C-pages are not too good. Due to many overview-pages and index-documents the quality of the proposals in terms of QCR becomes rather poor. The HPM is not able to determine the correct, good links.

But here another point is very important. Even though HPM provides hyperlink proposals, the probabilities are not too high. Therefore, the number of relevant proposals that exceed the probability threshold is rather low.

This is a good combination of circumstances for the precision value. The relative number of good proposals within all proposals is much better than in the case of ACM. In other words, the HPM "recognizes" that there are difficulties in classifying the hypertexts and therefore proposes only few hyperlinks.

In general, there are several other interesting relations found in our testing results. A change of the threshold δ can lead to very high-quality proposals but also to a faster "forgetting" of former cases. If we re-classify cases that have been learnt before, the probability amount does not reach the number of real links, however. A great number of attributes (e.g. *keywords*) help to find very subtle proposals, but the handling of the relevance matrix becomes inconvenient.

Furthermore, we also compared the highest proposal probabilities of links that were part of a hypertext to those that were not. All in all, we found a high correlation between the

[5]There is another possibility: a hyperlink might be appropriate even though it can not be found within an existing page.

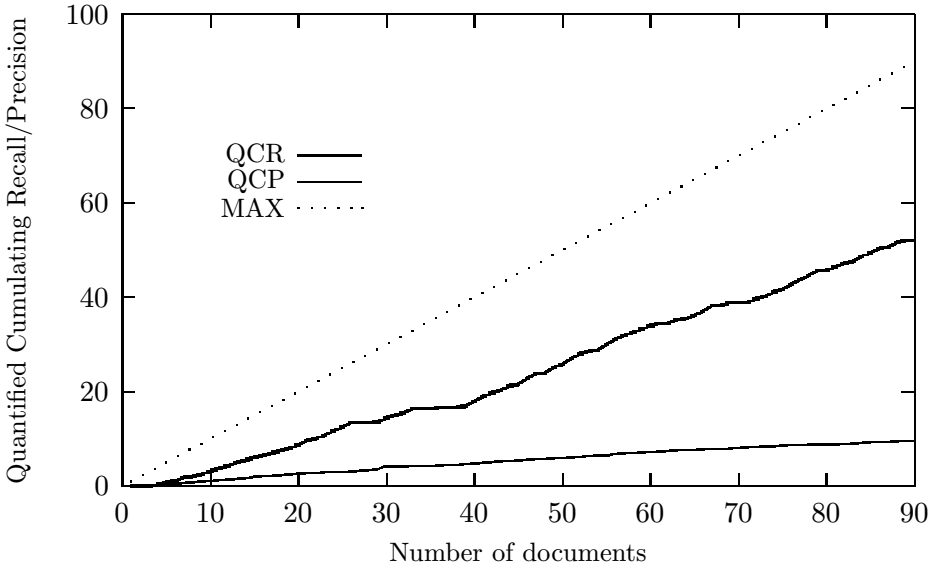

Figure 9.6: Proposal Qualities of ACM

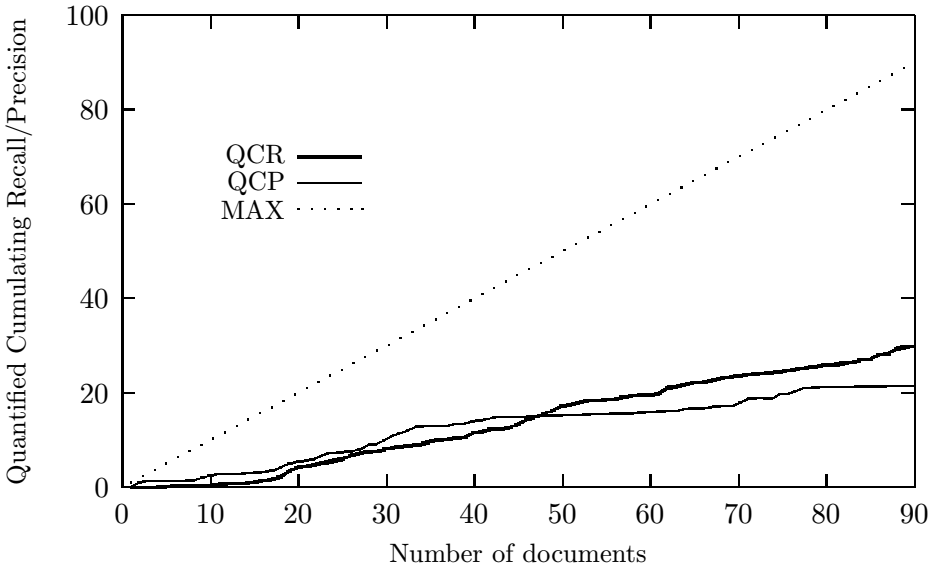

Figure 9.7: Proposal Qualities of W3C

complexity and the length of a text and the quality of the suggested links. In some cases, a proposed link that was no part of the hypertext might have been an appropriate supplement.

Even though the results of the HPM are rather promising, there is still much ground to be covered to become perfect.

Part IV

Similarities and Synergy

Chapter 10

Structural Interdependencies

Now here, you see, it takes all the running you can do, to keep in the same place.
If you want to get somewhere else, you must run at least twice as fast as that!

Lewis Caroll, Alice and the White Queen
in "Through the Looking Glass" (1872)

While the preceding chapters analyzed and discussed the concepts and implementation strategies for Request-Prediction (part II) and Hyperlink-Proposals (part III) in detail, we will emphasize in this chapter the common methodological and mathematical background of both areas.

Section 10.1, therefore, leads to an abstraction and a generalization of the presented RP-model and the HPM to highlight the similarities on a higher level of view.

Furthermore, we will see (in section 10.2) that well-known methodologies[1] embed both kinds of web-applications. From this point of view, it is easy to imagine manifold areas to apply similar concepts with promising concepts.

Finally, the section on synergy (10.3) returns to the concrete researches and sketches how to use the knowledge of the common conceptual background to derive implementation advantages. As an adequate example, the idea of "modeling time and document aging" of RP is transferred into terms of proposing-hyperlinks and the improvement possibilities of the HPM are discussed.

10.1 Abstraction and Generalization

A closer analysis of the tasks of proposing hyperlinks and predicting user-requests leads to astonishingly similar results. Abstractly spoken, the core function of both algorithms is the storage and structuring of information and the completion of partially given data in a foreseeing-step.

In the area of RP, the continuation of a starting user-session is to be predicted while the HPM helps an online author in completing a hypertext by proposing appropriate links. With other words, HPM tries to foresee the necessary user actions of generating hyperlinks. Certainly, the foreseeing-step pre-requests appropriate and well-structured databases in both cases to compute high-quality results.

[1] Originally derived from concepts of Artificial Intelligence.

Another more mathematical or algorithmic generalization of RP and HPM brings to light further similarities. Both solution strategies achieve their results by calculating probabilities which are based on statistically retrieved data. Moreover, existing information can be stored in a matrix that fulfills the stochastic-property[2] (described in 3.2.1).

As we will discuss in section 10.2, both, RP and HPM, execute the classification task by multiplying a vector with that matrix. The elements of the resulting vector lead (directly or after a few computation steps) to values that can be interpreted as *probabilities* which is very important.

Certainly, differences arise where the matrix and the vector elements are modeled. When searching for hyperlinks, this is a very difficult task: extracting appropriate keywords to retrieve important text attributes for later analyzing steps is rather delicate. Too many keywords will result in a very large matrix and thus in a long duration of the calculation process. Too few or even false keywords, though, may lead to a wrong classification and inappropriate link-proposals. In the case of predicting user-requests, the modeling of a session-vector is a rather simple and straightforward task. The decision and selection of potentially predictable data on the server-side is often made beforehand.

Nevertheless, the calculation of costs for incorrectly predicted or pre-fetched data is highly complex.[3] And even though wrongly proposed-hyperlinks are somehow disturbing, wrongly predicted information-packets of RP can cause enormous network and system load and thus pose a much higher danger than the HPM results.

Table 10.1 shows a short and abstract characterization of the main tasks of predicting user-requests and proposing links for hypertexts. The major differences arise in modeling the complexity of the corresponding problem in order to achieve appropriate vectors.

Yet it is very important to see that we could speak in both cases of learning and classifying tasks. We will come back to this aspect in section 10.2.

Naturally, the RP concept of chapter 4 could have been modeled in a different way where the similarities to the HPM would not have been as evident as discussed in this section. With the same argument, also the modeling of the HPM could be different. On the base of a semantic network the finding of appropriate links would mainly be a kind of searching in the net and retrieving semantic coherence.

Nevertheless, every straightforward concept for both algorithms provides - finally - statistical results with measurable probabilities. The concrete implementation does not contradict this statement. In the following section (10.2), we will go even an abstraction-step further and find out that both, RP and HPM belong to a larger class of web-applications[4] with a general problem-modeling and the corresponding solution-strategies in common. A summary of the comparison between RP and the HPM can be found in [HRH00a].

10.2 Cognitive Algorithms

As mentioned in the preceding section, the algorithms of RP and HPM can both be described in terms of *learning* and *classifying*. In Computer-Science, there are several different basic concepts and methods for the design of algorithms. A well known technique is for instance *divide-and-conquer*. The general idea of this methodology is to divide a complex problem into several at least slightly simpler sub-problems.

Another strategy related to divide-and-conquer is *recursion*. Here, an algorithm calls itself with a simplified set of parameters. Certainly, the design of these programs must be carefully

[2] In the strict sense only the basic version of the RP matrix fulfills the stochastic-property. The advanced solution leads to a more complex structure of the matrix.

[3] See 5.2.2 for a possible solution strategy of this problem.

[4] The described ideas are not limited to web-programs, but the focus of this work places the emphasis on this class of applications.

Phase	Step	Hyperlink-Proposals	Request-Prediction
Learning phase	Retrieval of knowledge	Keyword extraction of hypertexts and attached links	Different document requests during the same session (with or without taking care of the order)
	Storing of knowledge	Vectorization of keyword attributes and storage of values in a relevance matrix	Transforming sessions into (mostly binary) vectors and storing their values in a memory matrix
Classification phase	Recognition of information	Generating attributes (keywords) from texts (without considering link information)	Interpreting the first client request as ignition for calculation of relative frequencies
	Calculation of (likely) candidates	Multiplying the relevance matrix with the new attribute vector and thus getting link-relevance-probabilities → generating link-proposals	Multiplying the memory matrix with the current session-vector and thus getting relative probabilities for other documents to be requested soon → generating request-prediction

Table 10.1: Similarities between hyperlink-proposal and request-prediction processing

prepared. Endless loops has to be avoided by adding exact ending-conditions.

The class of *greedy-algorithms* operates as set-increasing strategies where a good solution can be approximated. Furthermore, there are the manifold solution strategies where the problems are modeled as trees, e.g. *tree search- and traversal methods*. Those concepts are explained in detail in [Knu88]. A good overview can also be found in [Mei91].

Moreover, the research area of *Artificial Intelligence* (AI) also categorizes algorithms by terms of *cognitive science*. A central aspect hereof is *knowledge*. Knowledge retrieval and storage in knowledge bases is regarded as a kind of *learning*.

Certainly, not every algorithm that stores a "0" in a database "learns". Learning can happen by acquiring knowledge, by recognizing legalities or rules, or by analogous deduction where the similarity plays an important part.

In fact, the term of "learning" should only be applied to systems that are able to transform complex, unstructured, incomplete, inconsistent or incorrect input into simple, well structured, complete, consistent or error free output based on a dynamically changing knowledge base. This transformation can be called *classification*[5]. Expert systems, for instance, fulfill most of the described requirements.

Expert systems can be *analytic* or *synthetic*. A good example of the former is a *diagnose system* that works on the base of existing data while a *planing system* that is able to deduct new data corresponds to the latter.

Mostly, expert systems are very large, complex programs that have to be attended and maintained during a long time span in order to become absolutely reliable. They can store the knowledge of human experts either explicitly or implicitly. Detailed information on the whole topic of AI-principles in general and especially on expert systems can be found in [Ric89].

[5]The term originates from pattern-recognition research. See for instance [HB90].

The idea to propose hyperlinks with expert systems might be appropriate. A synthetic modeling would be able to generate new links on base of the textual information while an analytical modeling would propose links that were learnt before. Even though our HPM described in part III of this work can not be regarded as an expert system due to its simplicity and lack of some needed properties, it is an analytic tool that performs learning and classification tasks.

Just as the HPM, also the RP can be described in terms of learning and classifying. An incomplete session is to be completed on base of former requests. The learning consists in storing the data requests from the client. From that point of view, proposing links and predicting requests belong to the same class of algorithms.

We will call algorithms that can be described adequately in terms of learning and classifying as *cognitive algorithms*. There are several further web-applications which might be appropriate for this kind of modeling. Certainly, the main ideas and concepts that were used to implement straightforward solutions for RP and HPM can also be applied to the following examples:

- *Intelligent mail filters*

  Here, the learning consists of recognizing user action: Which mails belong to which folders? The classification step would result in automatically structuring incoming mails into the corresponding folders.

- *Web searching algorithms*

  An area where *Intelligent Agents* or *Mobile Agents* are well suited is web searching. They too can belong to the class of cognitive algorithms. Learning means understanding the meaning of the user requests while the classification results in consistent search outcomes.[6]

- *Name services*

  An interesting idea is the use of predictive methods for naming services on the Internet. Analytic cognitive algorithms can help to improve the response times by pre-checking the consistence of domain names. Actually, the caching methods of these servers are not too complex.

- *Intelligent data routing*

  The Institute of Telematics is currently working on an advanced Smart Data Server (2.1) project with intelligent data transfer routing for the web[7]. The idea of considering the actual server loads to re-route certain client requests can be modeled with cognitive algorithms. The learning consists in measuring current response-times and deducing the reasons. The classification would result in an efficient request routing.

The strategy of modeling cognitive algorithms leads to potential improvements of manifold web-applications due to synergy-aspects. As we will see in the following section (10.3), this synergy does not only consist of sharing algorithms for data manipulation and complex objects with general usable methods, but also of higher modeling levels. Thus, ideas to improve one cognitive algorithm can - potentially - be an advantage for most of the others too.

10.3 Synergy

10.3.1 General Considerations

Even though the conceptions of RP and HPM are rather similar on an abstract level of view due to the common methodology of cognitive algorithms the question remains: Are there

[6] Further information on mobile or intelligent agents can be found in [EK98].

[7] Certainly, also intranet solutions or even general C/S-applications are also possible.

any further concrete advantages of recognizing similarities between link-proposals and request-prediction?

Obviously, additional synergy effects should be the result of such a situation. We will illustrate this by applying an advanced strategy of RP to improve the HPM. In section 6.1 we described some elementary ideas to model time and document aging for Request-Prediction. Here, the entries of the memory matrix were adapted so that not recently used data-sets led to a decreasing value of the corresponding elements.

What could this mean in the area of proposing hyperlinks? Time also influences the meaning of links. While writing a text about big sports events an HPM could propose a hyperlink that points to a page of the Olympic Games 2000 in Sydney. But what about proposals in the next year? Other interesting links might become more appropriate due to the actual focus of interest.

In general, even though the content behind certain links might still be a good reason to propose an old hyperlink too, a newer one is probably better. In other words, the modeling of time for hyperlink proposals leads to a decreasing relevance of attributes for links that have not been chosen for a longer time duration. At the same time this would have been the result of directly transferring the technical implementation of RP time modeling to the area of hyperlink proposals: matrix elements that are not part of fulfilled attributes for recent classification tasks have to be decreased.

While considering document aging, the RP modeling became more appropriate and the prediction results improved. Below, we want to show how the idea of modeling time could be applied to the concrete HPM derivation and we will briefly sketch some further improvement possibilities.

10.3.2 Advanced HPM modeling

In general, there are two main concept improvements while modeling time and aging for the HPM. At first, we will discuss an extension of the dynamically growing matrix ψ. Then, we will go into more details to adapt the learning algorithm of the HPM to specifically also forget the relevance of certain links.

In chapter 8, we saw that the increasing number of rows and columns does not only solve some problems concerning the CBR-like modeling but also poses new difficulties. Nevertheless, the data structure of ψ has to be limited. To do this, we mentioned some encouraging concepts dealing with priorities of attributes, for instance a limited number of keywords.

The idea to model time can here be applied by also shrinking the matrix ψ. If a link becomes "too old", its corresponding row within the matrix ψ can be deleted. The critical age of link depends on its recently selected proposal and on the systems storage limits. If a hyperlink is of no use anymore, it should be deleted from the "memory", and thus be "forgotten".

Additionally, also the number of columns could be decreased depending on the deleted rows. If certain attributes mainly serve to classify links that are not available anymore, it would be quite obvious to also delete the corresponding attribute columns of the matrix ψ. Naturally, only keyword attributes should be deleted. Here, further experiences have to be made to adequately shrink the matrix in order to "forget" former relevance weights for hyperlink proposals.

The other concept to model time for the HPM consists in extending the learning procedure described in section 7.2.4. During the learning phase the relevance weights of ψ are changed to ψ' so that the threshold δ is reached for all links that have to be "learnt" (see also equation (7.4)). In the case of additionally considering superfluous proposals we want that a link corresponding to the i-th element of the link vector $\mathcal{L}$ should *not* reach the threshold δ:

$$\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi'_{ij} \cdot \zeta^j(\mathcal{T}^j) < \delta \tag{10.1}$$

In this equation, ζ is the function that separates the attribute values so that a distinction between fulfilled attributes $\mathcal{F}$ and contradictory attributes $\mathcal{C}$ is possible (ζ is defined by (7.2)). Actually, according to Table 7.1 on page 68 (row 2) this learning task is only accomplished for links that should be proposed but which were not classified yet.

It would be an adequate transfer of time modeling to also learn links of the third row of Table 7.1. This means that links which are proposed but not necessarily useful should be treated too by fulfilling the learning equation (10.1), i.e. the classification actually results in a value too high. In that case, the weights of the fulfilled attributes $\mathcal{F}$ must be decreased and those of the contradictory ones $\mathcal{C}$ must be increased in order to reduce the classification values.

This step seems to be interesting but the time modeling component is not easy to see. A straightforward transforming of the RP idea for HPM usage results in a adaptation of matrix weights for links that are not learnt. The aim is to reduce the relevance of attributes for the proposal of links that are not currently useful. This situation corresponds to the rows 1 and 3 of Table 7.1. Due to an immense calculation overhead it is not feasible to change all those relevance weights for the dynamically increasing matrix ψ. Instead, only those of the third row are adapted because here the relevance values are too high.

To derive the corresponding weight adaptations we choose again a constant distribution for the dispersal of weights. As defined in section 7.2.4.2, we abbreviate:

$$\kappa = \sum_{j \in \mathcal{T}^{\mathcal{F}}} \zeta^j(\mathcal{T}^j)$$

Now, we are deriving the constant value Δ that describes the changes of the relevance weights ψ_{ij} for $j \in \mathcal{F}$ to "forget" the classification of link corresponding to i:

$$\begin{aligned}
& \textstyle\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi'_{ij} \cdot \zeta^j(\mathcal{T}^j) < \delta \\
\Longrightarrow \quad & \textstyle\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi'_{ij} \cdot \zeta^j(\mathcal{T}^j) = \delta - \epsilon \\
\Longleftrightarrow \quad & \textstyle\sum_{j \in \mathcal{T}^{\mathcal{F}}} (\psi_{ij} - \Delta) \cdot \zeta^j(\mathcal{T}^j) = \delta - \epsilon \\
\Longleftrightarrow \quad & \textstyle\sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{ij} \cdot \zeta^j(\mathcal{T}^j) - \sum_{j \in \mathcal{T}^{\mathcal{F}}} \Delta \cdot \zeta^j(\mathcal{T}^j) = \delta - \epsilon \qquad (10.2) \\
\Longleftrightarrow \quad & c_i - \Delta \cdot \textstyle\sum_{j \in \mathcal{T}^{\mathcal{F}}} \zeta^j(\mathcal{T}^j) = \delta - \epsilon \\
\Longleftrightarrow \quad & c_i - \Delta \cdot \kappa = \delta - \epsilon \\
\rightsquigarrow \quad & \Delta = \tfrac{c_i - \delta + \epsilon}{\kappa}
\end{aligned}$$

With the small value $\epsilon > 0$ the influence of the "forgetting" can be controlled. ϵ must not be zero because the corresponding link would still be proposed.[8] In general, ϵ could have been applied for the derivation in section 7.2.4.2 too and thus it could have influenced the learning *speed*. Details on the idea to control the learning speed can be found in [Haf93].

Using formula (10.2) we find for the new settings of ψ':

$$\psi'_{ij} = \begin{cases} \psi_{ij} - \frac{c_i - \delta + \epsilon}{\kappa} & : \quad \mathcal{T}^j \in \mathcal{F} \\ \psi_{ij} + \frac{\sum_{j \in \mathcal{T}^{\mathcal{F}}} \frac{c_i - \delta + \epsilon}{\kappa}}{\sum_{j \in \mathcal{T}^{\mathcal{C}}} 1} & : \quad \mathcal{T}^j \in \mathcal{C} \end{cases} \qquad (10.3)$$

[8] Certainly, also a modeling where $\epsilon < 0$ might make sense. We will come soon back to this idea.

10.3.3 Example

As an example, we review the settings of section 7.2.4.3. Here, the starting relevance matrix consisted of the following elements:

$$\psi = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.7 & 0.3 & 0.0 \end{pmatrix}$$

Let us take the following attribute vector to be classified:

$$\mathcal{T} = \begin{pmatrix} 0.9 \\ 0.9 \\ 0.2 \end{pmatrix}$$

and the attribute thresholds $\delta^1 = \delta^2 = \delta^3 = 0.8$.
Then, according to (7.3), the classification would lead to:

$$\mathcal{S} := \psi \cdot \zeta(\mathcal{T}) = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.7 & 0.3 & 0.0 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.9 \\ 0.0 \end{pmatrix} = \begin{pmatrix} 0.63 \\ 0.27 \\ 0.90 \end{pmatrix}$$

In addition, the classifying threshold might be $\delta = 0.75$ and the link-vector $\mathcal{L}$:

$$\mathcal{L} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

A comparison of $\mathcal{S}$ and $\mathcal{L}$ shows:

- $\mathcal{S}^1$ must be learnt according to the classical method described in section 7.2.4.
- $\mathcal{S}^2$ is not proposed and does not need to be classified ($\mathcal{L}^2 = 0$). This situation corresponds to the first row of Table 7.1.
- $\mathcal{S}^3$ is proposed ($0.9 > \delta$) but the corresponding entry of $\mathcal{L}^3 = 0$. Here, the modeling of time grips. This link should be "forgotten" according to (10.3).

As calculated before $c_3 = \sum_{j \in \mathcal{T}^{\mathcal{F}}} \psi_{3j} \cdot \zeta^j(\mathcal{T}^j) = 0.9$.
The value of κ results in: $\kappa = \sum_{j \in \mathcal{T}^{\mathcal{F}}} \zeta^j(\mathcal{T}^j) = 1.8$.
ϵ should be a small value, for instance $\epsilon := 0.03$.
Thus, the third row ($i = 3$) of the matrix ψ has to be changed to:

$$\psi'_{3j} = \begin{cases} \psi_{3j} - \frac{c_3 - \delta + \epsilon}{\kappa} = \psi_{3j} - \frac{0.9 - 0.75 + 0.03}{1.8} = \psi_{3j} - 0.1 & : \; j \in \mathcal{T}^{\mathcal{F}} = \{1, 3\} \\ \psi_{3j} + \frac{\sum_{j \in \mathcal{T}^{\mathcal{F}}} \frac{c_3 - \delta + \epsilon}{\kappa}}{\sum_{j \in \mathcal{T}^{\mathcal{C}}} 1} = \psi_{3j} + \frac{0.2}{1} = \psi_{3j} + 0.2 & : \; j \in \mathcal{T}^{\mathcal{C}} = \{2\} \end{cases}$$

Thus, ψ' would result in:[9]

$$\psi' = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ \mathbf{0.6} & \mathbf{0.2} & \mathbf{0.2} \end{pmatrix}$$

[9] Without taking care of the classical learning step for the first row.

A new classification would yield to:

$$\mathcal{S} := \psi \cdot \zeta(\mathcal{T}) = \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.7 \\ 0.6 & 0.2 & 0.2 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.9 \\ 0.0 \end{pmatrix} = \begin{pmatrix} 0.63 \\ 0.27 \\ \mathbf{0.72} \end{pmatrix}$$

Therefore, the former classification of $\mathcal{L}^3$ has been "forgotten" ($0.72 < \delta = 0.75$). Naturally, also an approach where the relevance values are more slightly changed so that is would take several "forgetting-steps" to no longer classify the link, might be appropriate. Here, a lot of research has still to be done.

The first evaluation steps of the advanced approach were very promising, even though long-term studies have to validate the advantages of the extended HPM.

Especially the setting of ϵ is rather important. It can be compared to the time factor τ of section 6.1.1, where the advanced prediction formulas were derived and in the subsequent sections evaluated.

We think that several further concept-improvements in the area of cognitive algorithms will lead to more efficient and faster web applications (as sketched in 10.2) in the future.

Chapter 11

Summary and Outlook

It is not the fruits of scientific research that elevate man and enrich his nature, but the urge to understand, the intellectual work, creative or receptive.

Albert Einstein, from "Ideas and Opinions" (1954)

The overall aim of this work is to present conceptual ideas and techniques for improving web applications. We placed the emphasis mainly on two concrete tasks, namely predicting future user requests and proposing links for hypertexts.

In the first part, we gave a general introduction to the main aspects of both research fields and presented fundamental basics for the solution of these problems. To support the endeavors of this doctoral thesis, we described concrete web applications that can be improved by prediction of user requests or proposals of hyperlinks. Therefore, chapter 2 placed the emphasis on online authoring tools and a smart server application.

Markov Chains are a well known mathematical conception for modeling request-prediction. We sketched briefly the main ideas and outlined the differences to our modeling in part II of the onhand work.

A useful strategy for hyperlink-proposals is given by the technique of Case-Based Reasoning (CBR). We focused on the principal ideas of CBR and transferred some of them to our modeling of a hyperlink-proposal module (HPM) presented in the third part of this work.

The second part of this elaboration dealt with a detailed discussion of predicting future client requests to reduce user perceived latency without producing too heavy a network or system load.

To do this, we first derived concrete formulas to develop an algorithm for performing prediction tasks. From the very beginning, we emphasized the problem of evaluating such an implementation.

Therefore, we modeled a testing scenario too, where user sessions could be generated automatically. The idea was to have thousands of test runs for improving our approach without actually risking network or system overload. Furthermore, we wanted to model different situations: By changing the controlling parameters such as the random factor we could study the different behavior of our Request-Prediction (RP) implementation.

The test runs resulted in the recognition that the usefulness of prediction strongly depends on the kind of presented data on the server side and the request behavior of the user clientele.

Even though the developed RP-approach has been quite useful already, we presented in chapter 6 an extended version where time and document aging has been modeled. The idea was that the importance of relations between data sets on server side and the meaning of the content itself should lose importance with the progression of time.

Here, we adapted additional parameters for the testing scenario such as the request change probability to model time more adequately. Also the RP-algorithm had to be changed in order to consider document aging. After this, the advanced approach could also be controlled by a time factor.

It was in by no means enough to validate the approach by evaluating the results of the test runs within our scenario. But rather we took real user logs to verify the usefulness of the RP modeling a posteriori. Those results were quite encouraging.

The other main focus of this work is on hyperlink-proposals. Part III discussed all aspects of our HPM modeling. Chapter 7 presented a derivation of the formulas for the HPM based on the concept of CBR. It was important to highlight also the differences between our approach and the established techniques successfully used in several areas, for instance to run diagnose systems.

The view of learning and classifying led us later on to an abstraction of the concepts. The implementation of the HPM, as part of a hyperlink managing system was the main topic of chapter 8. We found several critical aspects of our modeling that could be solved by a dynamically increasing matrix. Certainly, also the risks of that concept had to be considered.

Even though the statistical knowledge retrieval of hypertexts poses several disadvantages, we chose it for reasons of feasibility. It is not appropriate to leave the user with maintaining semantic networks for retrieving highest quality link proposals. Instead, we found as a practical solution that our HPM proposes links of rather good quality by completely automatically retrieving the information of texts. Probably today's web applications should require as little user interaction as possible.

After the presentation of RP in part II and the HPM in part III, chapter 10 started the fourth and last part with a very exiting idea: Having a closer look at both research areas - even though very different at first glance - they could be proved as similar on a higher abstraction level.

Therefore, we briefly sketched the opinion that algorithms (for web applications) with appropriate modeling of learning and classifying belong to a common class of cognitive algorithms. As an example, we presented some web applications and sketched their possible modeling in terms of learning and classifying.

It should be possible and adequate to transfer conceptual and - probably - technical improvements of one approach to another of the same class of algorithms. To underline and verify this thought we took the HPM of part III and discussed its improvement by modeling of time and document aging - derived from the RP-approach.

Even though not too simple, we found very promising conceptual improvements of the HPM extending the learning algorithm itself and the technical aspect of the dynamically growing matrix. Hopefully, those synergy effects can also help to shift other existing web applications to a higher level of quality or to develop new ones.

Naturally, we have to keep in mind that even though the comparison of different topics of modern Internet applications is very useful and advantageous, certain differences always remain and that some difficult parts of their problem aspects are, as a rule, quite unique.

For the future we plan not only to find more synergy effects between members of the same class of web applications, but also to solve concrete problem aspects of the existing implementations.

For instance, as part of a project supported by the "Stiftung für Innovation" of Rhineland-Palatinate we are constructing efficient and high-performance extensions to the Smart Data Server (SDS) presented in section 2.1.

One idea here is intelligent data routing. To do this, we need to retrieve reliable information for the development of a load balancing module. With such a tool also the concept of the prediction module could be improved. The calculation of costs for certain operations could then happen not only statically, but dynamically with every request the user performs. Thus, the cost calculation on base of results of an efficient load balancing module might become much better than the guesses so far.

In the area of the HPM an improvement can be the dynamical proposal of hyperlinks while the user is editing a text. These proposals would reflect more accurately the concrete paragraph or even the sentence the author is actually working on. Furthermore, the concept of a dynamically growing and shrinking matrix has to be elaborated on, in order to become quit efficient.

The observant reader may think that for the described improvement ideas of both research areas, RP and HPM, the concept of *dynamics* plays an important role. This might be - even though only slightly - another synergy effect of the similarity between both web applications. May the future bring more of such effects.

Bibliography

[Aha91] David W. Aha. *Case-Based Learning Algorithms.* Proceedings of the DARPA Workshop on Case Based Reasoning, Morgan Kaufmann, 1991. 147-157

[AKA91] David W. Aha, Dennis Kibler, Marc K. Albert. *Instance-Based Learning Algorithms.* Machine Learning 6, 1991. 37-66

[All96] James Allan. *Automatic Hypertext Link Typing.* Proceedings of the Seventh ACM Conference on Hypertext, Hypertext '96, 1996. 42-52

[And89] John Robert Anderson. *Kognitive Psychologie.* Spektrum der Wissenschaft Verlagsgesellschaft, 1989

[ANR89] Michael H. Andersen, Jakob Nielsen, and Henrik Rasmussen. *A Similarity-Based Hypertext Browser for Reading the UNIX Network News.* Hypermedia, 1(3), 1989. 255-265

[AS83] Dana Angluin, Carl H. Smith. *A Survey of Inductive Inference: Theory and Methods.* Computing Surveys 15, 1983. 237-269

[ASS87] Harold Abelson, Gerald Jay Sussman, Julie Sussman. *Structure and Interpretation of Computer Programs.* Fifth reprinting. MIT Press, 1987

[AW91a] Klaus-Dieter Althoff, Stefan Weß. *Case-Based Knowledge Acquisition, Learning and Problem Solving for Diagnostic Real World Tasks.* Proceedings of the European Knowledge Acquisition Workshop, EKAW-91, Crieff, Scottland, 1991

[AW91b] Klaus-Dieter Althoff, Stefan Weß. *Fallbasiertes Problemlösen in Expertensystemen - begriffliche und inhaltliche Betrachtungen*, SEKI Working Paper SWP-91-03, Universität Kaiserslautern, 1991

[AW92a] Klaus-Dieter Althoff, Stefan Weß. *Ähnlichkeit in PATDEX.* Proceedings of the Workshop: Ähnlichkeit von Fällen beim fallbasierten Schließen, SEKI Working Paper SWP-92-11, Universität Kaiserslautern, 1992

[AW92b] Klaus-Dieter Althoff, Stefan Weß. *Case-Based Reasoning and Expert System Development.* Schmalhofer, Strube, Wetter (Hrsg.), Contemporary Knowledge Engineering and Cognition, Springer Verlag, 1992

[AWB92] Klaus-Dieter Althoff, Stefan Weß, Brigitte Bartsch-Spörl, Dietmar Janetzko, Frank Maurer, Angi Voß. *Fallbasiertes Schließen in Expertensystemen: Welche Rolle spielen Fälle für wissensbasierte Systeme?* KI-Künstliche Intelligenz, 4, FBO-Verlag, 1992

[BC98] Paul Barford, Mark Crovella. *Generating Representative Web Workloads for Network and Server Performance Evaluation.* Proceedings of the International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS98, ACM, 1998. 151-160

[Ber90] Mark Bernstein. *An apprentice that Discovers Hypertext Links.* Proceedings of the First European Conference on Hypertext, ECHT-90, 1990

[Bes95] Azer Bestavros. *Using Speculation to Reduce Server Load and Service Time on the WWW.* Proceedings of the International Conference on Information and Knowledge Management, CIKM95, ACM, 1995. 403-410

[Bes96] Azer Bestavros. *Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems.* Proceedings of the World Conference on Open Learning and Distance Education, ICDE'96, New Orleans, Louisiana, 1996. 180-189

[Boe92] Katy Börner. *Ein allgemeines Modell zur Speicherung episodischen Wissens und separat lernbare Ähnlichkeitsmaße.* Positionspapier, 1992

[Bro97] Nat Brown. *Distributed Component Object Model Protocol - DCOM/1.0.* Microsoft Coperation, 1997

[CB93] R. J. Chitashvili, R. H. Baayen. *Word Frequency Distributions of Texts and Corpora as Large Number of Rare Event Distributions.* QL. Hrebicek, G. Altmann. Quantitative text analysis, Quantitative linguistics, Vol. 52, WVT Trier, 1993

[CB95] William R. Cheswick, Steven M. Bellovin. *Firewalls and Internet Security, Repelling the Wily Hacker.* Addison-Wesley, Massachusetts, 1995. ISBN 0-201-63357-4

[CB96] Chip Cleary, Ray Bareiss. *Practical Methods for Automatically Generating Typed Links.* Proceedings of the Seventh ACM Conference on Hypertext, Hypertext '96, ACM, 1996. 31-41

[CB98] Mark Crovella, Paul Barford. *The Network Effects of Prefetching.* Conference on Computer Communications, IEEE Infocom, 1998. 1232-1240

[CDF98] Ramon Caceres, Fred Douglis, Anja Feldmann, Gideon Glass, Michael Rabinovich. *Web Proxy Caching: The Devil is in the Details.* Workshop on Internet Server Performance, Madison, WI, 1998. 11-15

[CDI99] CD/ISIS. *Wageningen Agricultural University Library.* http://www.bib.wau.nl/isis/docum.html (multilingual), 1999

[CF82] Paul R. Cohen, Edward A. Feigenbaum. *The Handbook of Artificial Intelligence.* Volume 3. Pitman, 1982

[Cha93] Daniel T. Chang. *HieNet: A User-Centered Approach for Automatic Link Generation.* Proceedings of the Fifth ACM Conference on Hypertext, Hypertext '93, ACM, 1993. 145-158

[CHH98] L. A. Carr, W. Hall, S. Hitchcock. *Link Services or Link Agents?* Proceedings of the Ninth ACM Conference on Hypertext, Hypertext '98, ACM, 1998. 113-122

[CI98] Pei Cao, Sandy Irani. *Cost-Aware WWW Proxy Caching Algorithms.* Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1998. 193-206

[CKR98] Edith Cohen, Balanchander Krishnamurthy, J. Rexford. *Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters.* Proceedings of the International Conference on Applications, Technologies, Architectures and Protocols for Computer Communication, SIGCOMM98, ACM. 1998. 241-253

[Dep96] J. Depp. *Developing CGI Applications with Perl.* WILEY-VCH, 1996

[Dic95] A. Dickmann. *Two-Tier versus Three-Tier Applications.* Informationweek 553, 13/95. 74-80

[Dua96] Nick N. Duan. *Distributed Database Access in a Corporate Environment Using Java.* Computer Network and ISDN Systems 28, 1996. 1149-1156

[EK98] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag Pathak, C. B. Suresh. *Patterns of Intelligent and Mobile Agents.* Proceedings of the Second International Conference on Autonomous agents, ACM, 1998. 92-99

[EY99] Stephen Erickson, Danying Yi. *Modelling the Performance of a Large Multi-Tiered Application.* American Management Systems, AMS Center For Advanced Technology, 1999

[GA96] Jim Griffioen, Randy Appleton. *The Design, Implementation, and Evaluation of a Predictive Caching File System.* CS-Department University of Kentucky, CS-264-96, 1996

[GK79] Leo A. Goodman, William H. Kruskal. *Measures of Association for Cross Classifications.* Springer New-York, 1979

[GKH86] Gottwald, Künster, Hellwich, Kästner (Hrsg.). *Handbuch der Mathematik.* Veb Bibliographisches Institut, Leipzig, 1986. ISBN: 3-8166-0015-8

[Glu89] Robert J. Glushko. *Design Issues for Multi-Document Hypertexts.* Proceedings of the 2nd ACM Conference on Hypertext, Hypertext'89, ACM, Pittsburgh, PA, 1989. 51-60

[GZ93] J. Gordesch, A. Zapf. *Computer-Aided Foramtion of Concepts.* L. Hrebicek, G. Altmann (eds.). Quantitative text analysis, Quantitative linguistics, Vol. 52, WVT Trier, 1993

[Haf93] Ernst-Georg Haffner. *Analyse dynamischer Lernregeln für Case-Based Learning Systeme.* Diploma Thesis, AG Expertensysteme, Universität Kaiserslautern, 1993

[HB90] F. Hönes, R. Bleisinger, A. Dengel. *Intelligent Word-based Text Recognition.* Proceedings of the Symposium on Advances in Intelligent Systems, Machine Vision and System Integration, Boston, MA, 1990

[HDH96] W. Hall, H. Davis, G. Hutchings. *Rethinking Hypermedia: The Microcoms Approach.* Kluwer Academic Publishers, 1996

[HHR00] Ernst-Georg Haffner, Andreas Heuer, Uwe Roth, Thomas Engel, Christoph Meinel. *Advanced Studies on Link- Proposals and Knowledge-Retrieval of Hypertexts with CBR.* Proceedings of the International EC-Web Conference, ECWeb2000, Greenwich, United Kingdom, Springer-Verlag LNCS 1875, 2000. 378-396

[HHR99] Andreas Heuer, Ernst-Georg Haffner, Uwe Roth, Zhongdong Zhang, Thomas Engel, Christoph Meinel. *Hyperlink Management System for Multilingual Websites.* Proceedings of the Asia Pacific Web Conference, APWEB '99, 1999. http://www2.comp.polyu.edu.hk/ apweb99/

[HRE00a] Ernst-Georg Haffner, Uwe Roth, Thomas Engel, Christoph Meinel. *Modeling Time and Document Aging for Request Prediction - One Step Further.* Symposium on Applied Computing, ACM, SAC2000, Como, Italy, 2000. 984-990

[HRE00b] Ernst-Georg Haffner, Uwe Roth, Thomas Engel, Christoph Meinel. *Optimizing Requests for the Smart Data Server.* Proceedings of the International Conference on Applied Informatics, IASTED, AI2000, Innsbruck, Austria, 2000. 481-486

[HRE99a] Ernst-Georg Haffner, Uwe Roth, Thomas Engel, Christoph Meinel. *A Semi-Random Prediction Scenario for User Requests.* Proceedings of the Asia Pacific Web Conference, APWEB99, 1999. 11-18

[HRE99b] Ernst-Georg Haffner, Uwe Roth, Thomas Engel, Christoph Meinel. *Vorhersage von Benutzeranforderungen im WWW.* 7. GI-Workshop Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen, ABIS'99, Magdeburg, 1999. 283-294

[HRH00a] Ernst-Georg Haffner, Uwe Roth, Andreas Heuer, Thomas Engel, Christoph Meinel. *What do Hyperlink-Proposals and Request-Prediction have in Common?* Proceedings of the International Conference on Advances in Information Systems, ADVIS2000, Izmir, Turkey, SPringer-Verlag LNCS 1909, 2000. 285-293

[HRH00b] Ernst-Georg Haffner, Uwe Roth, Andreas Heuer, Thomas Engel, Christoph Meinel. *Link Proposals with Case-Based Reasoning Techniques.* World Conference on the WWW and Internet, AACE, WebNet'00, San Antonio, Texas, USA, 2000. 233-239

[HTML] HTML - HyperText MarkUp Language. http://www.w3.org/MarkUp/

[HTTP] HTTP - HyperText Transfer Protocol. http://www.w3.org/Protocols/

[HyT] International Organization for Standardization. *HyTime Standard Materials.* 1992-1997. http://www.hytime.org

[HZE99] Andreas Heuer, Zhongdong Zhang, Thomas Engel and Christoph Meinel. *DAPHNE - Distributed Authoring and Publishing in a Hypertext and Networked Environment.* Proceedings of the "Initiative Information und Kommunikation der wissenschaftlichen Fachgesellschaften in Deutschland", IuK99 - Dynamic Documents, Jena, 1999.

[Jan91] Klaus P. Jantke. *Monotonic and Non-monotonic Inductive Inference.* New Generation Computing 8, 1991. 349-360

[Jan92] Klaus P. Jantke. *Case-Based Learning in Inductive Inference.* Proceedings of the 5th ACM Workshop on Computational Learning Theory, COLT-92, 1992. 218-223

[Java] Java. http://www.javasoft.com/

[JB81] Klaus P. Jantke, Hans-Rainer Beick. *Combining Postulates of Naturalness in Inductive Inference.* ELK 17, 1981 8/9. 465-484

[JK98] Zhimei Jiang and Leonard Kleinrock. *An Adaptive Network Prefetch Scheme.* IEEE Journal on Selected Areas in Communications, 16(3), 1998. 358-368

[Joh97] Daeyeon Joh. *CBR in a Changing Environment.* Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97, LNAI 1266, Springer-Verlag, 1997. 53-62

[JWM93] Dietmar Janetzko, Stefan Weß, Erica Melis. *Goal Driven Similarity Assessment.* Hans-Jürgen Ohlbach (Hrsg.), Proceedings of the German Workshop on Artificial Intelligence, GWAI-1992, LNAI 671, Springer-Verlag, 1993

[KKD99] Hermann Kaindl, Stefan Kramer, Papa Samba Niang Diallo. *Semiautomatic Generation of Glossary Links: A Practical Solution.* Proceedings of the Tenth ACM Conference on Hypertext, Hypertext '99, ACM, 1999. 3-12

[KLM97] Thomas Kroeger, Darrell Long, Jeffrey Mogul. *Exploring the Bounds of Web Latency Reduction from Caching and Prefetching.* Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1997. 13-22

[KL95] J. Kolodner, D. Leake. *A Tutorial Introduction to Case-Based Reasoning.* Case-Based Reasoning. AAAI Press, the MIT Press, 1995

[KM79] Albert K. Kurtz, Samuel T. Mayo. *Statistical Methods in Education and Psychology.* Springer-Verlag, 1979

[Knu88] Donald E. Knuth. *The Art of Computer Programming*. Vol 1. Fundamental Algorithms. Addison-Wesley, Reading, Mass. 2nd ed. 1998. ISBN 0-201-03809-9

[Kol83] Janet L. Kolodner. *Reconstructive Memory: A Computer Model*. Cognitive Science, 7, 1983. 281-328

[Kop91] Helmut Kopka. *LaTex - Eine Einführung*. Addison-Wesley, 1991

[KW97] Achim Kraiss, Gerhard Weikum. *Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions*. Proceedings of the 23rd International Conference on Very Large Databases, VLDB, Athens, Greece, 1997. 246-255

[KW98] Achim Kraiss, Gerhard Weikum. *Integrated Document Caching and Prefetching in Storage Hierarchies Based on Markov-Chain Predictions*. The VLDB Journal, Springer-Verlag, 7(3), 1998. 141-162

[Lau98] Simon St. Laurent. *Cookies*. McGraw-Hill, 1998

[LZO98] C. Liu, X. Zhou, M. Orlowska. *Issues in Workflow and Web-Based Workflow Systems*. Proceedings of the Asia Pacific Web Conference, APWeb98, World Wide Web: Technologies and Applications, Beijing, China, 1998

[Mei91] Christoph Meinel. *Effiziente Algorithmen: Entwurf und Analyse*. Leipzig: Fachbuchverlag, 1991

[MM96] Jose M. Martinez, Francisco Moran. *Catalog: a WWW Gateway for DBMSs*. World Conference on the WWW and Internet, AACE WebNet'96, San Francisco, California, USA, 1996. http://aace.virginia.edu/aace/conf/webnet/proc96index.html

[MPS98] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, H. Singh. *Webwork: Meteors Web-Based Workflow Management System*. Journal of Intelligent Information Systems, 10(2), 1998. 1-30

[MS93] C. Marshall, F. Shipman. *Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext*. Proceedings of the Fifth ACM Conference on Hypertext, Hypertext '93, ACM, 1993. 217-230

[OMG] Object Management Group. http://www.omg.org

[OW96] K. Osterbye, U. K. Wiil. *The Flag Taxonomy of Open Hypermedia Systems*. Proceedings of the Seventh ACM Conference on Hypertext, Hypertext '96, ACM, 1996. 129-139

[PM96] V.N. Padmanabhan, J.C. Mogul. *Using Predictive Prefetching to Improve World Wide Web Latency*. Proceedings of the International Conference on Applications, Technologies, Architectures and Protocols for Computer Communication, SIGCOMM96, ACM, 1996. 26-36

[PMH97] C. Petrou, D. Martakos, S. Hadjiefthymiades. *Adding Semantics to Hypermedia Towards Link's Enhancement and Dynamic Linking*. Hypertext - Information Retrieval - Multimedia '97, HIM 1997, Universitaetsverlag Konstanz, 1997

[Ret99] Jean-Hugues Rety. *Structure Analysis for Hypertext with Conditional Linkage*. Proceedings of the Tenth ACM Conference on Hypertext, Hypertext '99, ACM, 1999. 135-136

[Ric88] Elain Rich. *KI-Einführung und Anwendungen*. MC Graw Hill, 1988

[Ric89] Michael M. Richter. *Prinzipien der Künstlichen Intelligenz*. B. G. Teubner Stuttgart, 1989

[Ric90] Michael M. Richter. *Konnektionismus*. Vorlesungsskriptum Universität Kaiserslautern, 1990

[Ric91] Michael M. Richter. *Lernende Systeme*. Vorlesungsskriptum Universität Kaiserslautern, 1991

[Ric92] Michael M. Richter. *Classification and Learning of Similarity Measures*. Proceedings der Jahrestagung der Gesellschaft für Klassifikation. Opitz, Lassen, Klar (Hrsg.), Studies in Classification, Data Analysis and Knowledge Organisation, Springer-Verlag, 1992

[Ric98] F. J. Ricardo. *Stalking the Paratext: Speculations on Hypertext Links as Second Order Text*. Proceedings of the Ninth ACM Conference on Hypertext, Hypertext '98, ACM, 1998. 142-151

[RHE99a] Uwe Roth, Ernst-Georg Haffner, Thomas Engel, Christoph Meinel. *An Approach to Distributed Functionality - the Smart Data Server*. World Conference on the WWW and Internet, AACE WebNet'99, Honolulu, Hawaii, USA, 1999. 931-936

[RHE99b] Uwe Roth, Ernst-Georg Haffner, Thomas Engel, Christoph Meinel. *The Smart Data Server: A New Kind of Middle-Tier*. Internet and Multimedia Systems and Applications, IASTED IMSA'99, Nassau, Bahamas, 1999. 361-365

[RHH99] Uwe Roth, Ernst-Georg Haffner, Andreas Heuer, Thomas Engel, Christoph Meinel. *Hyperlink Management System - HLM*. Technical Report 99-05, Institute of Telematics, 1999

[RW91] Michael M. Richter, Stefan Weß. *Similarity, Uncertainty and Case-Based Reasoning in PATDEX*. Boyer, R.S. (Ed.), Automated Reasoning, Essays in Honor of Woody Bledsoe, Kluwer Academic Publishers, 1991

[Sal88] Steven Salzberg. *Exemplar-Based Learning: Theory and Implementation*. Technical Report, TR 10-88, Cambridge, MA, Harvard University, Center for Research in Computing Technology, 1988

[Sch82] Roger C. Schank. *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press, New York, 1982

[Shn89] Ben Shneiderman. *Reflections on Authoring, Editing, and Managing Hypertext*. In E. Barret, The Society of Text, 1989. 115-131

[SKS98] S. Schechter, M. Krishnan, and M. D. Smith. *Using Path Profiles to Predict HTTP Requests*. 7th International World Wide Web Conference, Brisbane, Qld., Australia, 1998. 457-467

[Spr92] Richard Sproat. *Literary and Linguistic Computing*. Vol.8, No. 3, Morphology and Computation, The MIT Press, 1992. 113-130

[Sri97] Prashant Sridharan. *Advanced Java Networking*. Prentice-Hall, 1997

[Sta91] Michael Stadler. *Vergleich von Fallbasierten, Induktiven und Statistischen Lernverfahren für die Klassifikation*. Diploma Thesis, Universität Kaiserslautern, 1991

[Ste94] W. Richard Stevens. *TCP/IP Illustrates*. Volume 1: The Protocol, Addison-Wesley, 1994

[SUN] Sun Microsystems Inc. 901 San Antonio Road, Palo Alto, CA 94303 USA, 1994-2000, http://www.sun.com

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

[Teb98] John Tebbutt. *Finding links.* Proceedings of the Ninth ACM Conference on Hypertext, Hypertext '98, ACM, 1998. 299-300

[Tve77] Amos Tverski. *Features of Similarity.* Psychological Review, 1977. 327-352

[Wan99] Weigang Wang. *Team-and-Role-based Organizational Context and Access Control for Cooperative Hypermedia Environments.* Proceedings of the Tenth ACM Conference on Hypertext, Hypertext '99, ACM, 1999. 37-46

[Wes91] Stefan Weß. *PATDEX/2 - ein System zum Adaptiven, Fallfokussierenden Lernen in Technischen Diagnosesituationen.* SEKI Working Paper, SWP91/01, Universität Kaiserslautern, 1991

[Wes93] Stefan Weß. *PATDEX - Ein Ansatz zur Wissensbasierten und Inkrementellen Verbesserung von Ähnlichkeitsurteilen in der Fallbasierten Diagnostik.* Proceedings der 2. Deutschen Tagung Expertensysteme, XPS-93, Springer-Verlag, 1993

[WPA92] Stefan Weß, J. Paulokat, K.-D. Althoff. *Fallbasiertes Schließen - ein Überblick.* Technical Report, Fachbereich Informatik, Universität Kaiserslauten, 1992

[Wol89] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1989

[XAN] PROJECT XANADU. *The Original Hypertext Project.* http.//www.xanadu.net

[ZCM98] P. Zellweger, B.-W. Chang, J. Mackinlay. *Fluid Links for Informed and Incremental Link Transitions.* Proceedings of the Ninth ACM Conference on Hypertext, Hypertext '98, ACM, 1998. 50-57

[ZHH99] Zhongdong Zhang, Ernst-Georg Haffner, Andreas Heuer, Thomas Engel and Christoph Meinel. *Role-based Access Control in Online Authoring and Publishing Systems vs. Documentation Hierarchy.* Proceedings of the International Conference on Documentation, SIGDOC '99, ACM, 1999. 193-198

Part V

Appendix

Appendix A

Prediction-Modules Sourcecode

This appendix presents an exemplary implementation of the prediction scenario of part II of this work. The commented code segments are formulated with the C-programming language syntax. For the hyperlink-proposals, we present in appendix B an implementation model with use of the Java syntax.

A.1 C-Sourcecode of the Generator Module

In the following, the commented C-programming language source code of the *Session Generator Module*, also called the *Randomizer-Module*, is presented. The extended version with modeling of time and document aging can be found in section A.3.

```
// Session Generator
// Randomizer-Module

// part of the prediction testing scenario
// standard modeling

// debugging code excluded

#include <stdio.h>
#include <stdlib.h>

main (int argc, char **argv) {
// generator program

  int slen, nump, density, value;
  // slen   : length of the generated session vectors
  // nump   : number of predictable values of the session
  //          slen - nump is the number of (semi-)random values
  // density: number of settings for total random sessions
  // value  : concrete session setting (0 or 1)

  double rf, boundary_pred, boundary_rand;
  // rf     : random factor (ranges from 0.0, no randomness to 1.0
  //          , total randomness)
  // boundary_pred, boundary_rand
  //        : variables to calculate the concrete setting (0 or 1)
  //          from the random values, boundary_pred is for the
  //          predictable one (first nump values)
  //          and boundary_rand for the random ones (rest of the session)
```

```
long num_vec, i, j;
// num_vec : number of session vectors to be generated
// i, j    : inner and outer loop variables

// verify parameters
if (argc != 6) {
  fprintf(stderr, "Usage: %s <slen> <nump> <rf> <density> <# of vectors>\n",
      argv[0]);
  exit(1);
}

// slen = length of session vectors
if (sscanf(argv[1], "%d", &slen) != 1) {
  fprintf(stderr, "Error: could not read \"slen\"\n");
  exit(2);
}
if (slen < 0) {
  fprintf(stderr, "Error: \"slen\" must not be negative (%d)\n", slen);
  exit(3);
}

// nump = number of predefined settings for rf = 0.0
if (sscanf(argv[2], "%d", &nump) != 1) {
  fprintf(stderr, "Error: could not read \"nump\"\n");
  exit(4);
}
if (nump < 0 || nump > slen) {
  fprintf(stderr, "Error: \"nump\" not in range {0,..,%d} (%d)\n",
          slen, nump);
  exit(5);
}

// rf = random factor range [0.0,1.0]
if (sscanf(argv[3], "%lf", &rf) != 1) {
  fprintf(stderr, "Error: could not read \"rf\"\n");
  exit(6);
}
if (rf < 0.0 || rf > 1.0) {
  fprintf(stderr, "Error: \"rf\" not in range [0.0,1.0] (%.2lf)\n", rf);
  exit(7);
}

// density = number of settings for rf = 1.0
if (sscanf(argv[4], "%d", &density) != 1) {
  fprintf(stderr, "Error: could not read \"density\"\n");
  exit(8);
}
if (density < 0 || density > slen) {
  fprintf(stderr, "Error: \"density\" not in range {0,..,%d} (%d)\n",
      slen, density);
  exit(9);
}
```

```
  // num_vec = number of vectors to be calculated
  if (sscanf(argv[5], "%ld", &num_vec) != 1) {
    fprintf(stderr, "Error: could not read \"number of vectors\"\n");
    exit(10);
  }
  if (num_vec < 0) {
    fprintf(stderr,
        "Error: \"number of vectors\" must not be negative (%ld)\n",
        num_vec);
    exit(11);
  }


  // calculating the boundaries for settings (0,1) depending on the
  // random values for predictable and random parts of the session vector
  boundary_pred = 1.0 + rf*((double)density/(double)slen - 1.0);
  boundary_rand = (double)density/(double)slen*rf;


  // generating session vectors and printing the result on stdout
  // for postprocessing
  for (i = 0; i < num_vec; i++) {
    // generation of the first nump values


    for (j = 0; j < nump; j++) {
      // checking random value
      if (boundary_pred > rand()/(RAND_MAX+1.0)) {
        value = 1;
      } else {
        value = 0;
      }
      printf("%d", value);
    }


    // second loop for performance reasons (no comparisons)
    for (j = nump; j < slen; j++) {
      // checking boundary for random value
      if (boundary_rand > rand()/(RAND_MAX+1.0)) {
        value = 1;
      } else {
        value = 0;
      }
      printf("%d", value);
    }
    printf("\n");
  }


  exit(0);
}
```

A.2 C-Sourcecode of the Prediction-Module

The standard version of the prediction module is illustrated in the following. The advanced version can be found in A.4.

```
// Prediction-Module
// predicts and evaluate the test sessions

// part of the prediction testing scenario
// standard modeling

// debugging code excluded

#include <stdio.h>
#include <stdlib.h>

// ***********************************************
// fixed testing scenario settings
// ***********************************************

// MAX_VAL describes the length of the session vector
#define MAX_VAL 10000

// fixed value for positive requests
#define SETTING '1'

// define double "zero" for comparisons
#define DZERO 0.0000001

// caution: the max value of unsigned short is system dependend
// this is also a limit for the number of requests
#define INIT_VAL 0

// definition of the memory matrix phi
unsigned short phi[MAX_VAL][MAX_VAL];

// actual session vector length
int global_session_length;

// fixed threshold delta
double global_threshold;

// number of traing-vectors
int global_trainings;

// global counters
long grp = 0, grn = 0, gwp = 0, gwn = 0;
// grp : number of correctly predicted requests
// grn : number of data sets that were not requested and not predicted
// gwp : number of incorrectly predicted requests
// gwn : number of requests that were not predicted

// global number of request-predictions
long gtotal = 0;
```

```
// ***********************************************
// helping sub-routines
// ***********************************************

init_phi () {
// initialization of the matrix phi
  int i,j;
  for (i = 0; i < global_session_length; i++) {
    for (j = 0; j < global_session_length; j++) {
      phi[i][j] = INIT_VAL;
    }
  }
  return 0;
}

// ***********************************************
// procedures for prediction tasks
// ***********************************************

evaluate_prediction (char vector[], double result[], int position) {
// compare prediction with real vector to calcualte the prediction quality
// result is the output of the prediction, vector the "real" session vector
  char c;
  int i, rp = 0, wp = 0, rn = 0, wn = 0;

  gtotal++;

  for (i = 0; i < global_session_length; i++) {
    if (i == position) continue;

    // prediction check
    if (result[i] >= global_threshold) {
      if (vector[i] == SETTING) {
        // right positive prediction
        rp++;
      } else {
        // wrong positive prediction
        wp++;
      }
    } else {
      if (vector[i] == SETTING) {
        // wrong negative prediction
        wn++;
      } else {
        // right negative prediction
        rn++;
      }
    }
  }

  // update global counters
  gwp += wp;
  gwn += wn;
  grp += rp;
  grn += rn;
```

```
  return 0;
}

learn_request (char vector[], int number) {
// the session vector is "added" to the matrix phi at line "number"
// this routine is needed for learn_session

  int i = 0;
  char c;

  while ((c = vector[i++]) != '\0' && c != '\n') {
    if (c == SETTING) phi[number][i-1] ++;
  }
}

predict_with(int value, double result[]) {
// make a prediction with predefined setting on value
// the array (vector) result is changed
// this function is needed for predict_session

  int i;
  double main_diag = (double)phi[value][value];

  // no prediction possible
  if (main_diag < DZERO) return 1;

  for (i = 0; i < global_session_length; i++) {
    result[i] = (double)phi[value][i] / main_diag;
  }

  return 0;
}

predict_session(char vector[], double result[]) {
// make a prediction with elements set to 1 of vector
   int i = 0;
   char c;

   while (c = vector[i++]) {
     if (c == SETTING) { // only a 1 within the s-vector leads to a prediction
       if (!predict_with(i-1, result))
                  evaluate_prediction (vector, result, i-1);
     }
   }
   return 0;
}

learn_session(char vector[]) {
// store vector into the memory matrix phi
   int i = 0;
   char c;
```

```
   while (c = vector[i++]) {
     if (c == SETTING) learn_request(vector, i-1);
   }
   return 0;
}


// ***********************************************
// main prediction program
// ***********************************************


main (int argc, char **argv) {
// predictioner program


  char vector [MAX_VAL+1];
  // storage for the current session vector


  double result [MAX_VAL+1];
  // storage for the probability vector result from the classification step


  char c;
  // variable for the current vector elements


  int actual_vector = 0;
  // counter for the number of vectors


  // ******************
  // checking parameters
  // ******************


  if (argc < 4) {
    fprintf(stderr,
        "Usage: %s <session vector length> <threshold> <training runs>\n",
        argv[0]);
    exit (1);
  }


  // verifying the session vector length
  if ((sscanf(argv[1], "%d", &global_session_length)) != 1) {
    fprintf(stderr, "ERROR: can not read vector length (%s)\n", argv[1]);
    exit (2);
  }
  if (global_session_length < 0 || global_session_length > MAX_VAL) {
    fprintf(stderr,
        "ERROR: session vector length not in range {0,..,%ld} (%d)\n",
        MAX_VAL, global_session_length);
    exit (3);
  }
```

```
  // verifying the prediction threshold (delta)
  if ((sscanf(argv[2], "%lf", &global_threshold)) != 1) {
    fprintf(stderr, "ERROR: can not read threshold (%s)\n", argv[2]);
    exit (4);
  }
  if (global_threshold < 0.0 || global_threshold > 1.0) {
    fprintf(stderr,
        "ERROR: prediction threshold not in range [0.0,1.0] (%.2lf)\n",
        global_threshold);
    exit (5);
  }

  // verifying the number of traning runs
  if ((sscanf(argv[3], "%d", &global_trainings)) != 1) {
    fprintf(stderr, "ERROR: can not read number of training runs \
                      (%s)\n", argv[3]);
    exit (6);
  }
  if (global_trainings < 0) {
    fprintf(stderr, "ERROR: number of training runs must not be negative \
                            (%d)\n",
        global_trainings);
    exit (7);
  }

  // ********************
  // starting prediction
  // ********************

  // initializing the memory matrix
  init_phi();

  // read in line after line from standard input
  while (fgets(vector, MAX_VAL, stdin)) {

    // make a prediction with all 1s of vector
    if (actual_vector++ > global_trainings) predict_session(vector, result);

    // memorize vector
    learn_session(vector);
  }

  // printing the overall prediction result of the test runs
  printf("Total Predictions = %ld Prediction Quality = \
                          %4.2lf Threshold = %2.4lf",
        gtotal, (double)grp/gwp, global_threshold);
  printf(" RP = %03ld WP = %03ld RN = %03ld WN = %03ld\n",
        grp, gwp, grn, gwn);

  exit(0);
}
```

A.3 C-Code of the Time-Modeled Generator Module

At this point, the commented source code of the advanced *Generator Module* is presented. The standard version can be found in A.1.

```
// Session Generator
// Randomizer-Module

// part of the prediction testing scenario
// extended modeling of time and document aging

// debugging code excluded

#include <stdio.h>
#include <stdlib.h>

// limit for the number of generated session vectors
#define MAXVEC 100000

main (int argc, char **argv) {
// extended generator program

  int slen, nump, density, value, runs, ri;
  // slen     : length of the generated session vectors
  // nump     : number of predictable values of the session
  //            slen - nump is the number of (semi-)random values
  // density  : number of settings for total random sessions
  // value    : concrete session setting (0 or 1)
  // runs     : number of test runs
  // ri       : variable to count the current run

  double rf, boundary_set, boundary_rand, roh;
  // rf     : random factor, ranges from 0.0, no randomness
  //          to 1.0, total randomness
  // boundary_pred, boundary_rand
  //        : variables to calculate the concrete setting (0 or 1)
  //          from the random values, boundary_pred is for the
  //          predictable one (first nump values)
  //          and boundary_rand for the random ones (rest of the session)
  // roh    : help variable to store parts of the boundary calculation

  double boundary_set_1, boundary_set_0, RCP;
  // boundary_set_1, boundary_set_0
  //        : the boundary for the decision whether a floating value should
  //          become 0 or 1 depends on the value before, therefore we do need
  //          two different boundaries
  // RCP    : the request change probability (models user behavior)

  long num_vec, i, j, lent;
  // num_vec : number of session vectors to be generated
  // i, j    : inner and outer loop variables
  // lent    : length of a time section (this is only needed for subsequent
  //           processing steps, a section break for the time factor)
```

```
int lastvalues[MAXVEC];
// stores the last session vector to be able to deal with RCP

// verify parameters
if (argc != 9) {
  fprintf(stderr, "Usage: %s<slen> <nump> <rf> <density> \
                          <#vectors> <lent> <RCP> <runs>\n",
      argv[0]);
  exit(1);
}

// slen = length of session vectors
if (sscanf(argv[1], "%d", &slen) != 1) {
  fprintf(stderr, "Error: could not read \"slen\"\n");
  exit(2);
}
if (slen < 0 || slen >= MAXVEC) {
  fprintf(stderr, "Error: \"slen\" not within {0,..,%ld}\n", MAXVEC);
  exit(3);
}

// nump = number of predefined settings for rf = 0.0
if (sscanf(argv[2], "%d", &nump) != 1) {
  fprintf(stderr, "Error: could not read \"nump\"\n");
  exit(4);
}
if (nump < 0 || nump > slen) {
  fprintf(stderr, "Error: \"nump\" not in range {0,..,%d} \
                          (%ld)\n", slen, nump);
  exit(5);
}

// rf = random range [0.0,1.0]
if (sscanf(argv[3], "%lf", &rf) != 1) {
  fprintf(stderr, "Error: could not read \"rf\"\n");
  exit(6);
}
if (rf < 0.0 || rf > 1.0) {
  fprintf(stderr, "Error: \"rf\" not in range [0.0,1.0] (%.2lf)\n", rf);
  exit(7);
}

// density = number of settings for rf = 1.0
if (sscanf(argv[4], "%d", &density) != 1) {
  fprintf(stderr, "Error: could not read \"density\"\n");
  exit(8);
}
if (density < 0 || density > slen) {
  fprintf(stderr, "Error: \"density\" not in range {0,..,%d} (%ld)\n",
      slen, density);
  exit(9);
}
```

```
// num_vec = number of vectors to be calculated
if (sscanf(argv[5], "%ld", &num_vec) != 1) {
  fprintf(stderr, "Error: could not read \"number of vectors\"\n");
  exit(10);
}
if (num_vec < 0) {
  fprintf(stderr, "Error: \"number of vectors\" must no be negative\
                     (%ld)\n",
      num_vec);
  exit(11);
}

// lent = length of a time section
if (sscanf(argv[6], "%ld", &lent) != 1) {
  fprintf(stderr, "Error: could not read \"lent\"\n");
  exit(12);
}
if (lent <= 0) {
  fprintf(stderr, "Error: \"lent\" must be positive (%ld)\n", lent);
  exit(13);
}

// RCP = probabilty to change values
if (sscanf(argv[7], "%lf", &RCP) != 1) {
  fprintf(stderr, "Error: could not read \"RCP\"\n");
  exit(14);
}
if (RCP < 0.0 || RCP > 0.5) {
  fprintf(stderr, "Error: \"RCP\" not in range [0..0.5] (%.2lf)\n", RCP);
  exit(15);
}

// runs = number of testing runs
if (sscanf(argv[8], "%d", &runs) != 1) {
  fprintf(stderr, "Error: could not read \"runs\"\n");
  exit(16);
}
if (runs <= 0) {
  fprintf(stderr, "Error: \"runs\" must be positive (%ld)\n", runs);
  exit(17);
}

// setting random combined values
// for detailed information of these formulas see chapter 6
roh = 1.0 + rf*((double)density/(double)slen - 1.0);
boundary_set_1 = 1.0+roh*RCP*2.0-2.0*RCP;
boundary_set_0 = 2.0*roh*RCP;
boundary_rand = (double)density/(double)slen*rf;

for (ri = 1; ri <= runs; ri++) {
// outermost loop counts the test runs
```

```
boundary_set = roh;
// this boundary corresponds to the former modeling
// (without time and aging)
// and is still used for the first session of each test run

// generating session vectors
// at first the predictable elements
for (i = 0; i < num_vec; i++) {

  if (i % lent == 0 && i > 0) {
    // set a mark for subsequent prediction task
    // simulates a time span, e.g. another day
    // (as sign for the time factor)
    for (j = 0; j < slen; j++) printf("-");
      printf("\n");
  }

   // generation of the first nump values
   for (j = 0; j < nump; j++) {

     // checking random value
     if (i) {
       // from the second session of each run the modeling
       // the RCP requires the value of the last request
       // to calucalte the probabilty of the next
       if (lastvalues[j] == 0) {
         boundary_set = boundary_set_0;
       } else {
         boundary_set = boundary_set_1;
       }
     }

     // this calucaltion is the same as for the standard modeling
     // (without time and aging)
     if (boundary_set > rand()/(RAND_MAX+1.0)) {
       value = 1;
     } else {
       value = 0;
     }

     // remember the last setting
     lastvalues[j] = value;
     printf("%d", value);
  }

  // for the random settings of the session vector nothing has changed
  for (j = nump; j < slen; j++) {
    // checking random value
    if (boundary_rand > rand()/(RAND_MAX+1.0)) {
      value = 1;
    } else {
      value = 0;
    }
```

```
        printf("%d", value);
      }
      printf("\n");
    }
  }
  exit(0);
}
```

A.4 C-Code of the Time-Modeled Prediction-Module

Here, the programming code of the advanced prediction module with modeling of time and aging can be found. Identical procedures to the standard approach are omitted to increase the readability of the code.

```
// Prediction-Module
// predicts and evaluate the test sessions

// part of the prediction testing scenario
// advanced modeling time and document aging

// debugging code excluded

// ***********************************************
// fixed testing scenario settings
// ***********************************************
// MAX_VAL describes the length of the session vector
#define MAX_VAL 10000

// MINUMUM for matrix values
#define MINIMUM 0.0001

// fixed value for positive requests
#define SETTING '1'

// define double "zero"
#define DZERO 0.0000001

// caution: the max value of unsigned short is system dependend
// this is also an (unchecked) limit for the number of requests
#define INIT_VAL 0.0

// definition of the memory matrix phi
double phi[MAX_VAL][MAX_VAL];

// definition of the last access session vector
long lasv[MAX_VAL];

// actual session vector length
int global_session_length;

// fixed threshold delta
float global_threshold;
```

```
// number of traing-vectors
int global_trainings;

// global counters
long grp = 0, grn = 0, gwp = 0, gwn = 0;
// grp : number of correctly predicted requests
// grn : number of data sets that were not requested and not predicted
// gwp : number of incorrectly predicted requests
// gwn : number of requests that were not predicted

// global number of request-predictions
long gtotal = 0;

// global time factor
double timefactor;

// counter for time sections
long timecounter = 0L;

// ***********************************************
// helping sub-routines
// ***********************************************

init_phi () {
// initialization of the matrix phi
  int i,j;
  for (i = 0; i < global_session_length; i++) {
    for (j = 0; j < global_session_length; j++) {
      phi[i][j] = INIT_VAL;
    }
  }
  for (i = 0; i < global_session_length; i++) lasv[i] = 0L;
  return 0;
}

// ***********************************************
// procedures for prediction tasks
// ***********************************************

// evaluate_prediction(char vector[], double result[], int position)
// predict_with(int value, double result[])
// predict_session(char vector[], double result[])
// all those functions are the same as for the simplified version

learn_timeline (char vector[], int number) {
// learning of a request while considering the time modeling
// the function is needed for learn_timesession
  int i = 0;
  long timespans;
  char c;
```

```
  while ((c = vector[i]) != '\0' && c != '\n') {
    if (c == SETTING) phi[number][i] ++;
    else {
      // the timing factor is only applied to 0s of the session vector
      timespans = timecounter - lasv[i];
      phi[number][i] -= timespans*timefactor;
      if (phi[number][i] < MINIMUM) phi[number][i] = INIT_VAL;
    }
    i++;
  }
}

learn_timesession(char vector[]) {
// store vector into the memory matrix phi
   int i = 0;
   char c;

   while (c = vector[i++]) {
     if (c == SETTING) learn_timeline(vector, i-1);
   }
   return 0;
}

// ***********************************************
// main prediction program
// ***********************************************
main (int argc, char **argv) {
// predictioner program

  char vector [MAX_VAL+1];
  // storage for the current session vector

  double result [MAX_VAL+1];
  // storage for the probability vector result from the classification step

  char c;
  // variable for the current vector elements

  int actual_vector = 0;
  // counter for the number of vectors

  // ******************
  // checking parameters
  // ******************
  if (argc < 5) {
    fprintf(stderr,
        "Usage: %s <svector length> <threshold> \
                      <training runs> <timefactor>\n",
        argv[0]);
    exit (1);
  }
```

```
// verifying the session vector length
if ((sscanf(argv[1], "%d", &global_session_length)) != 1) {
  fprintf(stderr, "ERROR: can not read vector length (%s)\n", argv[1]);
  exit (2);
}
if (global_session_length < 0 || global_session_length > MAX_VAL) {
  fprintf(stderr, "ERROR: session vector length not in range \
                  {0,..,%ld} (%d)\n",
      MAX_VAL, global_session_length);
  exit (3);
}

// verifying the prediction threshold (delta)
if ((sscanf(argv[2], "%lf", &global_threshold)) != 1) {
  fprintf(stderr, "ERROR: can not read threshold (%s)\n", argv[2]);
  exit (4);
}
if (global_threshold < 0.0 || global_threshold > 1.0) {
  fprintf(stderr, "ERROR: prediction threshold not in range [0.0,1.0] \
              (%.2lf)\n",
      global_threshold);
  exit (5);
}

// verifying the number of training runs
if ((sscanf(argv[3], "%d", &global_trainings)) != 1) {
  fprintf(stderr, "ERROR: can not read number of training runs \
                      (%s)\n", argv[3]);
  exit (6);
}
if (global_trainings < 0) {
  fprintf(stderr, "ERROR: number of training runs must not be negative \
                           (%d)\n",
      global_trainings);
  exit (7);
}

// verifying the time factor
if ((sscanf(argv[4], "%lf", &timefactor)) != 1) {
  fprintf(stderr, "ERROR: can not read the time factor (%s)\n", argv[3]);
  exit (8);
}
if (timefactor < 0.0) {
  fprintf(stderr, "ERROR: time factor must not be negative \
                  (
  exit (9);
}

// ******************
// starting prediction
// ******************

// initializing the memory matrix
init_phi();
```

```
  // read in line after line from standard input
  while (fgets(vector, MAX_VAL, stdin)) {

    // verifying the time mark
    if (vector[0] == '-') {
       timecounter++;
       continue;
    }

    // make a prediction with all 1s of vector
    if (actual_vector++ > global_trainings) predict_session(vector, result);

    // memorize vector
    learn_timesession(vector);
  }

  // printing the overall prediction result of the test runs
  printf("Total Predictions = %ld Prediction Quality =  \
            %4.2lf Time factor = %4.2lf",
        gtotal, (double)grp/gwp, timefactor);
  printf("Threshold = %2.4f RP = %03ld WP = %03ld RN = %03ld WN = %03ld\n",
        global_threshold, grp, gwp, grn, gwn);

  exit(0);
}
```

Appendix B

Hyperlink-Proposal Sourcecode

B.1 Java-Sourcecode of an Hyperlink Proposal Module

In the following, the commented Java source code for an *Hyperlink Proposal Module* HPM is presented. For simplicity, we only show the core functionalities of the HPM object.

```
/**
 * Hyperlink Proposal Module HPM
 * Main Class
 *
 * @version 1.0.0
 * @since JDK 1.1
 */
class HPM {

    /* ****************************
     * definition of global constants
     * ****************************
     */

    // thresholds for attributes and classification
    // validity attribute
    private static final double VALATTRIB_DEFAULT = 0.7;
    // departmental structure attribute
    private static final double STRATTRIB_DEFAULT = 0.7;
    ...
    private static final double KEYWATTRIB_DEFAULT = 0.4;
    private static final double CLASSIFICATION_DEFAULT = 0.85;

    // special attributes
    private static final int SPEC_ATTRIBS =
                                <number of special attributes>;
    private static final String ATT_STRUCTURE =
                                "Departmental Structure";
    private static final String ATT_VALIDITY =
                                "Validity";
    ...
```

```
// value to equalize computation inexactness
private static final double EQUALIZER = 0.000001;

// default maximum number of proposals
private static final int MAX_PROPOSALS = 20;

/* ************************************
 * definition of private class variables
 * ************************************
 */

// thresholds for attributes and classification
// validity attribute threshold
private double valattrib_threshold = VALATTRIB_DEFAULT;
// departmental structure attribute threshold
private double strattrib_threshold = STRATTRIB_DEFAULT;
// keyword attribute threshold
private double keywattrib_threshold = KEYWATTRIB_DEFAULT;
// link classification threshold
private double class_threshold = CLASSIFICATION_DEFAULT;

// core datastructure: the relevance matrix, dynamically growing
private Matrix psi = null;

// a link-list: base for the classification results
private LinkList links = new LinkList();

// the attribute list as vector
private Vector attributes = new Vector();

// flag for local or remote use of the HPM
boolean local = true;

// URI of remote website as base for the HPM
String base = null;

/* ************************************************
 * set and get methods to manipulate private variables
 * ************************************************
 */

/**
 * Retrieve threshold for the validity attribute
 * @param
 * @return double validity attribute threshold between 0.0 and 1.0
 */
public double getValAttrib() {
      return valattrib_threshold;
}
```

```
/**
 * Retrieve threshold for the departmental structure attribute
 * @param
 * @return double departmental structure attribute threshold
 *                    between 0.0 and 1.0
 */
public double getStrAttrib() {
      return strattrib_threshold;
}

...

/**
 * Retrieve threshold for the keyword attribute
 * @param
 * @return double keyword attribute threshold between 0.0 and 1.0
 */
public double getKeywAttrib() {
      return keywattrib_threshold;
}

/**
 * Retrieve class threshold
 * @param
 * @return double class threshold between 0.0 and 1.0
 */
public double getClassThreshold() {
      return class_threshold;
}

/**
 * Set threshold for the validation attribute
 * @param double threshold for the validation attribute
 * @return
 */
public void setValAttrib(double threshold) {
      valattrib_threshold = threshold;
      return;
}

/**
 * Set threshold for the departmental structure attribute
 * @param double threshold for the dep. structure attribute
 * @return
 */
public void setStrAttrib(double threshold) {
      strattrib_threshold = threshold;
      return;
}

...
```

```
/**
 * Set threshold for the keyword attribute
 * @param double threshold for the keyword attribute
 * @return
 */
public void setKeywAttrib(double threshold) {
      keywattrib_threshold = threshold;
      return;
}

/**
 * Set class threshold
 * @param double class threshold
 * @return
 */
public void setClassThreshold(double threshold) {
      class_threshold = threshold;
      return;
}

/* ***********************
 * usefule private methods
 * ***********************
 */

/**
 * Initializes the internal variables called by class constructors
 * @param
 * @return
 */
private void initializeData() {
      // the matrix is defined for two initial columns (see below)
      psi = new Matrix(0, SPEC_ATTRIBS);

      // the attribute list starts with the validity and the
      // document type entries as minimum
      attributes.addElement(ATT_VALIDITY);
      attributes.addElement(ATT_STRUCTURE);
      ....

      return;
}

/* ****************************
 * attribute manipulating methods
 * ****************************
 */
```

```
/**
 * adding a new keyword to the system
 * @param Keyword the new keyword to be added
 * @return
 */
private void addKeyword(Keyword w) {
    // adding keyword to the list of attributes
    attributes.addElement(w);

    // preparing the new column for the matrix
    double[] newcol = new double[links.linkCount()];

    // new columns are initialized by zero
    for (int i = 0; i < newcol.length; i++) newcol[i] = 0.0;

    // adding a corresponding column to the matrix
    try {
        psi.addCol(newcol);
    } catch (Exception e) {
        System.out.println ("Error adding a new keyword:  " + e);
    }
    return;
}

/* ***********************************
 * link (solution) manipulating methods
 * ***********************************
 */

 /**
  * adding a new link to the system
  * @param Link the new link to be added
  * @return
  */
 private void addLink(Link l) {
     // verifying links for efficiency
     if (!isValidLink(l, local, links)) return;

     // adding the new link to the current link list
     links.addLink(l);

     // preparing a new link weight row for the matrix
     double[] newrow = new double[psi.getColCount()];

     // initializing the new row with same weights for all attrbiutes
     for (int i = 0; i < newrow.length; i++) {
             newrow[i] = 1/(double)newrow.length;
     }
```

```
        // adding a corresponding row to the matrix
        try {
            psi.addRow(newrow);
        } catch (Exception e) {
            System.out.println ("Error adding a new link:  " + e);
        }
    }

    /* ************************************
     * Matrix or vector manipulating methods
     * ************************************
     */

    /**
     * the attribute separation method that distinguished between fulfilled and
     * contradictory attributes by resetting the latter ones to 0
     * @param Vector the attribute vector
     * @return Vector the resulting separated attribute vector
     */
    private Vector zeta (Vector attVec) {
        Vector result = attVec.clone();

        // every attribute is compared to the corresponding
        // threshold value

        // the special attributes are treated first
        if (result.get(0) < valattribthreshold)
                                        result.set(0, new Double(0.0));
        if (result.get(1) < strattribthreshold)
                                        result.set(0, new Double(0.0));
        ...

        // then the keyword attributes thresholds are compared
        for (int i = SPEC_ATTRIBS - 1; i < result.length(); i++) {
                if (result.get(i) < keywattribthreshold)
                                        result.set(i, new Double(0.0));
        }
        return result;
    }

    /**
     * a simple information retrieval method based on
     * statistical properties of  the corresponding text
     * @param Meta the text represented by its meta-object
     * @return Vector the attribute vector (not yet separated into
     *                  fulfilled and contradictory attributes)
     */
    private Vector retrieveAttributeVector (Meta text) {
            // the resulting attribute vector
            Vector result = new Vector();
```

```
        // initializing the resulting vector
        for (int i = 0; i < attributes.size(); i++)
                result.add(new Double(0.0));

            // retrieving the list of all keywords
            DocKeyword keywordList[] =
                        text.getKeywords().getArrayOfDocKeywords();

        try {
            // the special attributes are treated first
            result.set(0, getValAttribute(text));
            result.set(1, getStrAttribute(text));
            ...

            // the retrieval of the keyword attributes
            for (int i = SPEC_ATTRIBS - 1; i < attributes.size(); i++) {
              // compare known attributes to the list of
              // keywords appearing in the text
              for (int j = 0; j < keywordList.length; j++)
                if (((Keyword)KeywordList[j]).equals(attributes.elementAt(i)))
                        result.set(i, getKeywordWeight(keywordList[j]));
            }
        } catch (Exception e) {
            System.out.println ("Error retrieving information from text:  " + e);
        }
        return result;
}

/**
 * adapting the relevance weights of matrix psi (within the given row)
 * to map the learning changes adequately
 * @param row the row of psi that is to be changed
 * @param Meta the text as base for the chaning
 * @result
 */
private void changeWeights (int row, Meta text) {
        double result = 0.0;
        // constructing a list with fulfilled attributes indexes
        int fulfilledIndexes[] = getFulfilledAttributeIndexes(text);
        int fulfillatt = fulfilledIndexes.length;

        try {
            // investigating the (constant distributed) weight change
            // for the fulfilled attributes that is to be added
            double addingValue =
                  missingClassificationValue(row, text) / fulfillatt;

            // ... and analogous for the contradictory attributes
            double substractingValue =
                      addingValue * fulfillatt / (attributes.size()-fulfillatt);
```

```
            // check all attributes and make the relevance adaptation
            // depending on whether the attributes belong
            // to the set of fulfilled ones or not
            for (int i = 0; i < attributes.size(); i++) {
                    if (belongsToArray(fulfilledIndexes, i)) {
                        // the weight adaptation for the fulfilled attributes
                        psi.setEntry(row, fulfilledIndexes[i],
                                psi.getEntry(row, fulfilledIndexes[i]) + addingValue);
                    } else {
                        // the weight adaptation for the contradictory attributes
                        psi.setEntry(row, contradictoryIndexes[i],
                                psi.getEntry(row,contradictoryIndexes[i])
                                        -substractingValue);
                    }
                }
        } catch (Exception e) {
            System.out.println ("Error in changing the relevance weights:  " + e);
        }
        return;
    }

    /* ************************************************
     * Main cognitive methods, learning and classifying
     * ************************************************
     */
    /**
     * The classification method takes a text (represented by its meta-object)
     * and retrieves link probabilities exceeding the threshold ordered
     * by their probability to be useful as hyperlinks for the text
     * @param Meta the text represented by its meta-object
     * @return LinkList the list of links to be proposed
     */
    public LinkList classify(Meta text) {
        // declaring the result vector with probabilities
        // for each link
        Vector probability_vector = null;

        // result finally contains the proposed links
        // orderd by their probability
        LinkList result = new LinkList();

        // verifying the status of the
        // dynamically growing matrix psi
        psi.checkIntegrity ();

        // core classification process
        try {
            probability_vector =
                psi.multiply(zeta(retrieveAttributeVector(text)));
        } catch (Exception e) {
            System.out.println ("Error in core classification:  " + e);
        }
```

```
        // the probabilities have to be compared to the
        // class threshold and the final resulting link list is built
        try {
            // a proposal list contains indexes of Links within psi
            // together with their probability
            ProposalList proplist = new ProposalList();
            for (int i = 0; i < links.linkCount(); i++) {
                double probabilty = probabilty_vector.get(i);

                // result beyond threshold leads to a proposal
                if (probability + EQUALIZER > class_threshold) {
                    proplist.addElement(i, probability);
                }
            }

            // order the links according to their probability
            proplist.sortIndexes();

            // generating the final result linklist
            for (int i = 0;
                  i < min(proplist.proposalsCount(), MAX_PROPOSALS);
                  i++) result.addLink(links.getLink(proplist.getLinkIndex(i)));
        } catch (Exception e) {
            System.out.println ("Error in calculating link proposals:  " + e);
        }
        return result;
    }

    /**
     * The main learning method takes a text as input (represented by its meta-object)
     * and adapts the relevance weights so that a subsequent classification step would
     * result in the given linklist
     * @param Meta the text represented by its meta-object
     * @param LinkList the list of links that are to be classified
     * @return
     */
    public void learn(Meta text, LinkList propLinks) {
        // The first step consists of updating the relevance matrix psi
        // so that all links within the link list are known to the system.

        // The number of rows of the matrix is increased dynamically.
        updateRelevanceMatrix(text, propLinks);

        // an a priori classification step is needed to compare
        // the results to the given list of links
        LinkList classLinks = classify(text);

        // finally, all the matrix weigths are updated that belong
        // to links (rows) where a link should be proposed
        // but is not classified yet
        for (int i = 0; i < links.linkCount(); i++) {
```

```
            // find out, if the current link is within
            // the proposed or the classified list
            Link currentLink = links.getLink(i);
            boolean linkIsProposed = propLinks.isElement(currentLink);
            boolean linkIsClassified = classLinks.isElement(currentLink);

            // case treatment
            if (linkIsProposed && linkIsClassified) {
                // the current link is proposed and classified,
                // the learning algorithm has nothing to do,
                // but it might be adequate to do some statistics
                ...
            } else if (!linkIsPropoased && linkIsClassified) {
                // in the first step of the development of the HPM
                // this case was not considered; it is a result of
                // the synergy to see that in this case
                // document aging can be modeled
                ...
            } else if (linkIsPropoased && !linkIsClassified) {
                // the real learning case; the matrix weights
                // have to be adpated
                changeWeights (i, text);
            } else {
                // corresponds to: !linkIsProposed && !linkIsClassified
                // again, this is not learning case
            }
        }
        return;
    }

    /* ***************
     * HPM Constructors
     * ***************
     */

    /**
     * Neutral HPM constructor for local classification
     * (local website, no special URI-base)
     * @param
     * @return
     */
    public HPM () {
        initializeData();
        local = true;
    }
```

```
    /**
     * External use of the HPM (for remote sites)
     * @param String a URI as base for the subsequent
     *                  classifcation steps
     * @return
     */
    public HPM (String URI) {
        initializeData();
            local = false;
        base = new String(URI);
    }
}
```

Index

Q

R

S

T

U

V

W

Tabellarischer Bildungsgang

Name:	Ernst-Georg Alfons Haffner
Geburtsdatum:	30.11.1966
Gebutsort:	Hermeskeil
1973-1977	Besuch der Grundschule Gusenburg
1977-1986	Besuch des Gymnasiums Hermeskeil Abschluss: Abitur
1986-1987	Grundwehdienst
1987-1993	Studium der Informatik an der Universität Kaiserslautern
Februar 1993	Diplom der Informatik
1993-1997	Angestellter der zentralen IT-Abteilung eines Fensterherstellers betraut mit dem Design, der Entwicklung und Wartung von computergestützter Anwendungssoftware zur Auftragsabwicklung und zur automatisierten Fensterproduktion
1997-2001	Wissenschaftlicher Mitarbeiter im Institut für Telematik in Trier unter der Betreuung von Prof. Dr. Christoph Meinel