

Exploiting a Web Cache for User Recommendations

Peter Edwards*, Murray Steele, Simon E. Clare & Darren M. Johns

Department of Computing Science
King's College
University of Aberdeen
Aberdeen, AB24 5UE
Scotland
+44 (0)1224 272270

{pedwards, mst, sec, djohns}@csd.abdn.ac.uk

Abstract

In this paper we describe work in progress on the development of a central data store with integrated support for recommender agents. The Hitchcock proxy is detailed, and two recommender agent systems constructed to use the data held in the cache are discussed. The COWBRA agent supports a form of collaborative Web browsing, by building profiles of user behaviour which are then used to make recommendations from pages visited by other users of the proxy. The second agent (TMA1) performs user matchmaking and displays the results graphically. TMA1 also provides support for communication between users. The paper concludes with a discussion of proposed Hitchcock extensions.

1 Introduction

Agent-based recommender systems [21] automate forms of interaction within human society which are typically referred to as “word of mouth” recommendations. People regularly use advice received via such mechanisms to influence the choices they make. An agent recommender system requires a series of inputs (descriptions of users, documents, products) together with some means of aggregating them together or identifying patterns between them. Outputs take the form of a list of recommended items, which may be the names of people (e.g. experts to consult on a particular topic), documents (e.g. Web pages to visit) or even suggested purchases to be made at an E-Commerce site. Matching functions play a vital role in this process by determining the degree of similarity between items. Another common feature of all recommender systems is their reliance on large amounts of data. The long-term objective of the work outlined in this paper is to construct a data repository with integrated support for programmers wishing to build recommender agents (and eventually other forms of information agents). We chose to base our initial approach around a Web proxy as a means of providing high data volumes, while at the same time imposing minimum overhead on users.

The remainder of this paper is organised as follows: section 2 outlines some background to the work; section 3 describes the proxy architecture; section 4 introduces an agent-based URL recommender; section 5 presents a user matchmaker and communication facilitator; section 6 briefly discusses related work; and finally section 7 reviews the status of the work, and highlights future directions.

2 Background

Our experience with the construction of a variety of intelligent information agents in recent years [16, 17, 11, 7, 8] has been that there is significant redundancy within such systems. For example, it is common to see components which handle data acquisition, data abstraction, user-modelling, classification/prediction, and user feedback. It

*Author to whom all correspondence should be addressed.

would clearly be desirable to create a central data resource which also included support for as many of these functions as possible; such a facility would allow *lightweight* agents to be constructed, built upon an agent application programming interface (AAPI).

Any approach of this type assumes that users will be willing to give up some privacy in return for access to (value-added) agent services. In other words, they have an incentive in allowing data about them to be logged. However, privacy is still an important issue; very few users are likely to be happy at the prospect of their credit card details or other sensitive information being stored within a central, publicly accessible repository. The data store must therefore be under some form of user control. One approach we considered was that used within the COLLABCLIO [12] system that supports sharing of Web browsing histories amongst users. COLLABCLIO provides controls for users to set the level of accessibility of data which relates to them. In our initial work we have chosen not to provide users with direct access to the contents of the store, instead we have provided a facility which allows them to deactivate logging of their actions.

In the discussion that follows, we present the architecture of the proxy and discuss a rudimentary AAPI which offers some support for agent developers. Two very different recommender systems have been built to investigate the scope of the API. It was always our intention to construct an initial version which would be enhanced following our experiences in the development of practical recommender systems. With this goal in mind, it was important that the agents reflected different aspects of recommender system technology and different application contexts. The scope of the agents are outlined in Table 1.

Agent	Inputs	Outputs	Representation of User Models	Models per User	Recommendation Engine
<i>Collaborative Web Browser</i>	URLs	URLs	Bayesian Model	≥ 1	Simple Bayesian Classifier
<i>User Matchmaker</i>	URLs	Users	Sparse Binary Vector	1	Similarity Metric

Table 1: Characteristics of the recommender agents.

3 The Hitchcock Proxy/Cache

Hitchcock¹ was developed to provide a structured and reusable data source (see Figure 1). It simply observes a user's browsing activity and records these *browse-events* as abstractions of the page content in a cache. The user has the option of turning on and off the monitoring function; with monitoring off, Hitchcock behaves as a standard Web proxy.

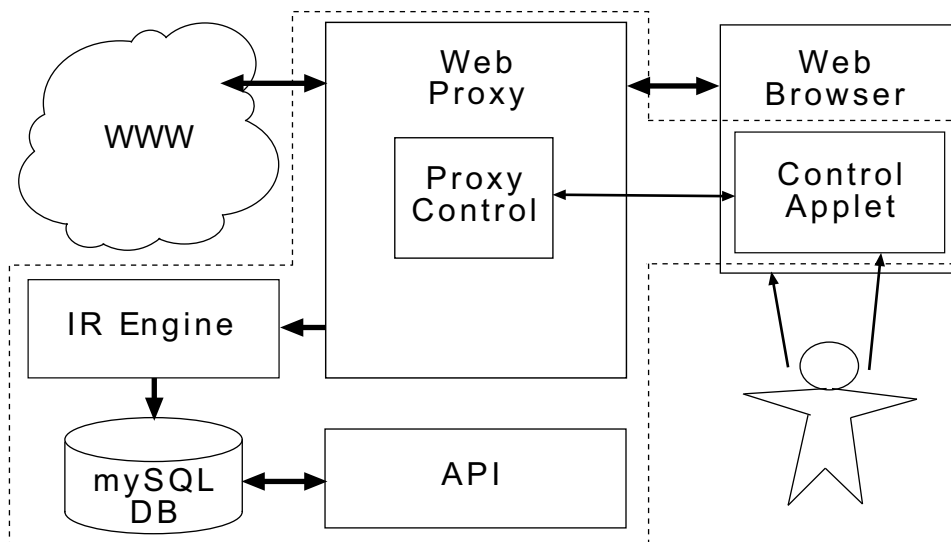


Figure 1: The Hitchcock architecture.

¹We felt that instead of an obscure acronym Hitchcock required a familiar name to present to its users. We chose Hitchcock because it embodied a sense of benevolent watchfulness and intelligence.

```

HTTP GET www.javaintro.com/ url: http://www.javaintro.com/
1.0:Simon title: Java Intro
<html> user: Simon
  <head> date: 22-03-2000
    <title>Java Intro time: 13:20:21
  </head> alt: This is Duke, the Java
  <body> Mascot
    <h1>Java Intro</h1> keyword: Java,3,1.0;
    <p>This page tells you programming,2,0.67;
    all you will ever need language,2,0.67;
    to know about the Java Sun,1,0.33;
    programming language. Microsystems,1,0.33;
    </p> Intro,1,0.33;
    <hr>
    <p>Java is a simple
    programming language
    developed by Sun
    Microsystems.</p>
  </body>
</html>

```

Figure 2: An example Hitchcock *browse-event* with the original HTML source shown to the left.

3.1 The Web Proxy

The Hitchcock proxy takes HTTP requests and fetches the relevant URLs from the Web by passing the request on to an existing proxy cache, such as Squid [2]. However, it differs in one significant way from a normal proxy, as it serves a control applet to the user's browser to allow them to switch the caching component on and off. As shown in Figure 1, the applet connects to a proxy control centre and sends a signal whenever the user changes the state of the applet. The proxy control centre maintains a list of users that are currently connected to Hitchcock and have caching activated. This list contains the Hitchcock userid of the user and the address of their machine. HTTP requests are compared against this list to determine whether or not data should be cached.

3.2 The Cache

The efficient storage of data in the Hitchcock system is a priority, as is effective analysis of this data. A decision was made very early in the development of the system to summarise user browsing activities as discrete *browse-events*. A *browse-event* is defined as a request by a Web client to view a specific page and is made up of one or more HTTP get requests. This event is then analysed (using simple IR techniques) and the result stored in a relational database system (MySQL [1]). Data corresponding to error messages, redirections or other mechanisms for administration of the Web are not logged; Hitchcock information and administration pages are also ignored.

Once a page has been accepted its individual elements are analysed. If the data is in any form other than text (e.g. images, applets and scripting sections) it is discarded. Although we acknowledge that such data might prove useful when determining the content of a page, it is outside the current scope of Hitchcock to analyse such data. The remaining page content is then parsed to extract material for storage. Text within the body of the page is gathered for further analysis as are tags and the <TITLE> tag. A decision was taken to ignore data associated with <META> tags. The contents of the <TITLE> tag are stored verbatim. The text associated with the alt attribute of tags is extracted; this data is useful as many HTML authors provide descriptions of images for users of text based browsers. A standard stop list that has been augmented with commonly occurring Web vocabulary (e.g. link, click) is then applied to the body text to remove non-content terms. The most frequently occurring terms are then identified and their frequencies and normalised frequencies recorded.

The information extracted from the page, together with its URL, details of the user who looked at the page and the date/time it was accessed are then instantiated as a Java object *JURLData*. This object, which represents a discrete browsing event, is then sent to a buffer for storage. When the system is ready each *browse-event* is converted into SQL statements to be executed on the database. Figure 2 shows an example Hitchcock *browse-event*.

3.3 The Hitchcock API

As stated earlier, one of our central aims in developing Hitchcock was that it should include an interface for use by developers of agent recommender systems. The API in its present form provides an initial set of objects and

methods for accessing the data collected by the proxy. At the lowest level the API provides a simple connection to the database and allows SQL queries to be executed. The API also provides classes which represent database records as objects, as well as methods to convert between them. A single *browse-event* is stored as a *URLData* object. This object closely mirrors the contents of the database and has object representations of all the database fields. These *URLData* objects are collected together in *URLDataLists* which provide list methods for accessing single objects within them.

The command set is currently confined to *Select* operations through the implementation of the *HitchcockSelectURL* interface. Two API classes implement this interface: *ProxyDataConnection* and *URLDataList*. *ProxyDataConnection* allows data to be fetched direct from the database and returns *URLDataLists* of the results of the query. *URLDataList* applies the API methods to itself and returns sublists of its contents. The command set is relatively simple, yet as all methods return *URLDataLists* they can be ‘chained’ to create more sophisticated search operations. Tables 2 and 3 provide details of the API classes and methods respectively.

API Class	Purpose
<i>HitchcockConnection</i>	Provides a low level connection to the database allowing data to be retrieved using any SQL statement.
<i>HitchcockSelectURL</i>	Provides an interface for <i>Select</i> statements to retrieve <i>browse-events</i> . All methods return <i>URLDataLists</i> and have Date range based variants.
<i>ProxyDataConnection</i>	A higher level connection to the database that sits on top of a <i>HitchcockConnection</i> to provide implementations of the <i>HitchcockSelectURL</i> .
<i>URLData</i>	A data construct to represent the <i>browse-events</i> stored in the database.
<i>URLDataList</i>	A list construct for holding <i>URLData</i> items. Implements the <i>HitchcockSelectURL</i> .
<i>Converters</i>	A static class to allow conversions between API objects and database records.

Table 2: Summary of Hitchcock API classes.

Method	Purpose
<code>getUrls()</code>	Retrieves all <i>browse-events</i> .
<code>getUserUrls()</code>	Retrieves all <i>browse-events</i> generated by a specific user.
<code>get[FIELD]Urls()</code>	Retrieves all <i>browse-events</i> with a specific term in the specified FIELD. Variants exists to allow searching for the appearance of any term from a list. There are versions of this method for searching in the <code>keyword</code> , <code>title</code> or the <code>alt</code> data fields.
<code>getDistinctUrls()</code>	Returns a set of <i>browse-events</i> corresponding to the most recent visit to each URL.
<code>filter()</code>	Takes a <i>URLDataList</i> and returns all the <i>browse-events</i> that do not occur in the parameter. There is no Date range version of this.

Table 3: Summary of *HitchcockSelectURL* methods.

4 Collaborative Web Browsing

COWBRA² is a recommender agent which suggests URLs to a user based on their previous browsing behaviour. It implements a form of collaborative Web browsing by exploiting the browsing activities of a group of users to achieve this. Central to the approach is the assumption that within a group of Internet users it is likely that there is some degree of cross-over of interests. The agent (Figure 3) maintains a ‘family’ of interest profiles for each user; these profiles are used to assess *browse-events* generated by other users and thus to highlight URLs of interest. This is a different model of collaborative Web browsing to that used by systems such as Let’s Browse [13] which assume that browsing is taking place in a social setting, e.g. a family using WebTV. Let’s Browse uses the combined interest profiles of the group to recommend pages that they should visit next.

²Collaborative Web Browsing Recommender Agent.

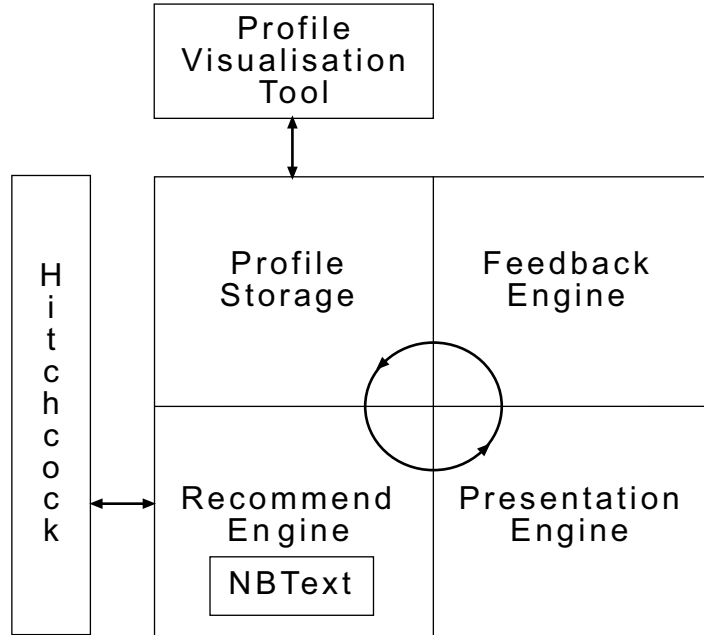


Figure 3: Components of the COWBRA agent.

4.1 Profiles & Recommendations

Rather than represent each user by a single monolithic interest profile, COWBRA employs a family of simple profiles. Each of these focuses on one specific interest area. Profiles are in fact collections of *Hitchcock browse-events* that have been labelled by the user as either ‘like’ or ‘dislike’. Profiles are stored independently of the classifier used to make recommendations; as a result, any algorithm capable of classifying textual data can be used. The only requirements are that conversion routines exist to translate between COWBRA profiles and the input/output formats of the classifier, and that a method exists to support the profile visualisation process - see section 4.2.

COWBRA currently uses a modified naive Bayes [15] classifier called NBText. NBText further simplifies the naive Bayes classifier by making the assumption that the probability of encountering a specific term is independent of its position. This has the advantage of reducing the number of probability terms that need to be calculated, while also increasing the number of examples available for predicting the probabilities (and hence the reliability of those estimates). NBText calculates the most probable class for a new *browse-event* using Equation (1). For each class v_j in the set of classes V , it calculates the product of the probability of seeing class v_j and the prior probabilities of seeing each term W_k given class v_j (Equation 2). N_k is the number of times term W_k has been seen with class v_j , N is the total number of terms seen for class v_j , and *vocabulary* is the set of distinct terms seen during training.

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{k \in \text{positions}} P(W_k | v_j) \quad (1)$$

$$P(W_k | v_j) = \frac{N_k + 1}{N + |\text{vocabulary}|} \quad (2)$$

The COWBRA Recommendation Engine builds a set of recommendations for a user by applying each of that user’s profiles, in turn, to *browse-events* generated by other users of the Hitchcock proxy. NBText predicts a label (like/dislike) for each event. Any that have been classified as ‘like’ are presented to the user as recommendations.

4.2 Profile Management & Visualisation

COWBRA allows the user to manage their profiles via the Feedback Engine. If a URL is recommended and the user agrees with this recommendation, they can give positive feedback, thereby reinforcing the profile. This is achieved by adding the *browse-event* corresponding to the recommendation to the appropriate profile with a ‘like’ label attached. If, however, the user disagrees with the recommendation they have the option to supply negative feedback. Such feedback can take a variety of forms. For example, the user may dislike the content of

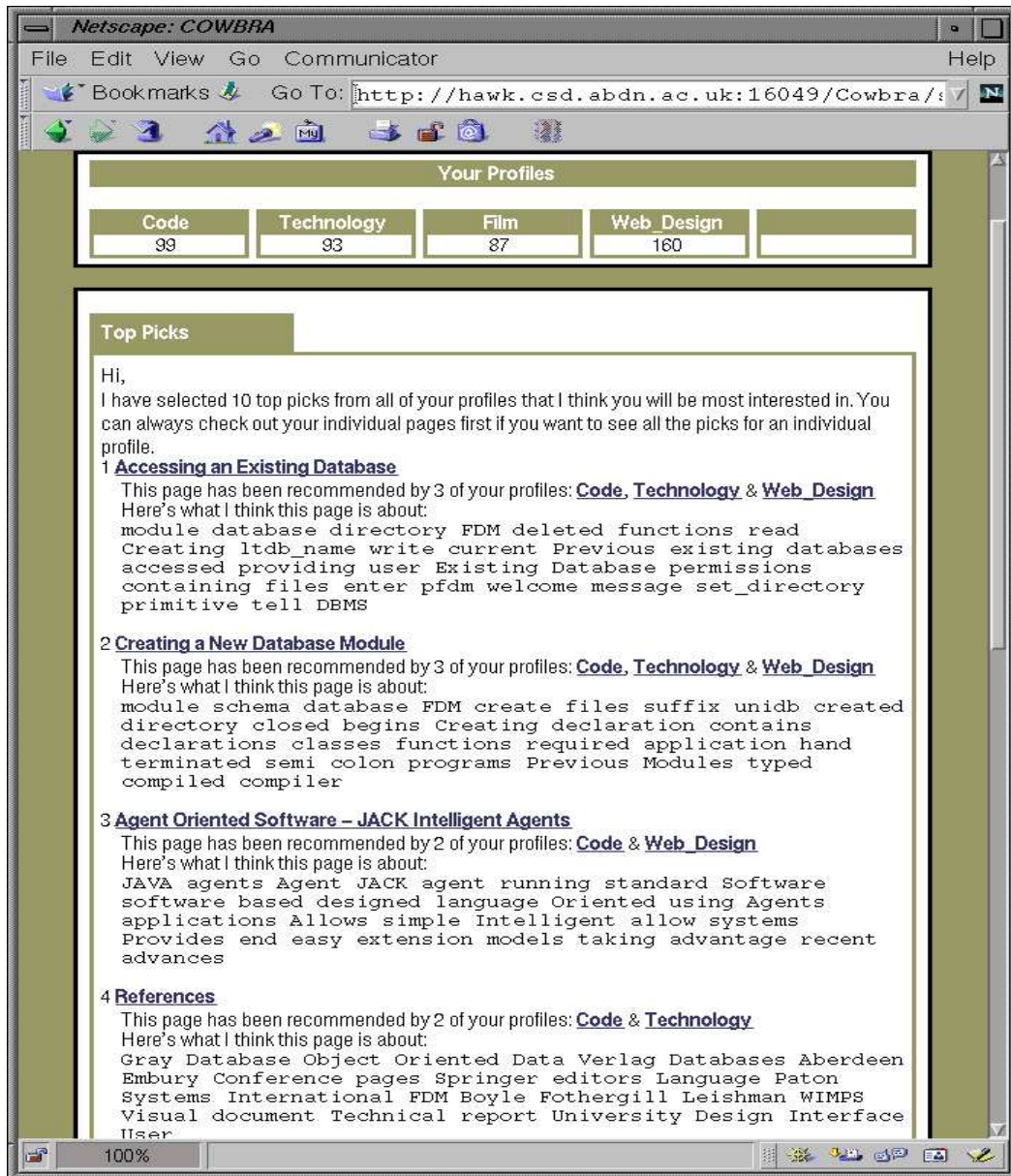


Figure 5: The COWBRA recommendations page.

4.3 The COWBRA Interface

As COWBRA has been designed to recommend URLs, it seemed only natural to create a Web-based interface for the agent - the Presentation Engine. Instead of placing all its recommendations on a single page, COWBRA separates them into pages for each profile. Each of these pages show the recommendations for a specific profile and allow the profile visualisation tool to be launched as an applet. The user is also presented with a 'Top Picks' page, as shown in Figure 5. This presents the top ten recommendations of the user's family of profiles. A recommendation generated by more than one profile is either of great interest to the user (as it covers more than one of their interests) or may be an indication that one or more of the profiles is flawed. In either case the user should have such recommendations brought to their attention.

Recommendations are presented to the user as a title and abstraction (set of keywords) corresponding to a Hitchcock *browse-event*. The title is converted into a link to allow the user to visit the page. When the user follows a link they are presented with a browser window containing two frames. The topmost frame contains the URL they are visiting, while the lower frame contains the COWBRA feedback interface. This interface lists all the profiles that recommended the URL and allows the user to signal whether they liked or disliked the recommendation. In the case of negative feedback the interface allows the user to supply a justification (see section 4.2).

The interface is updated dynamically by COWBRA so that once a recommendation has been viewed and

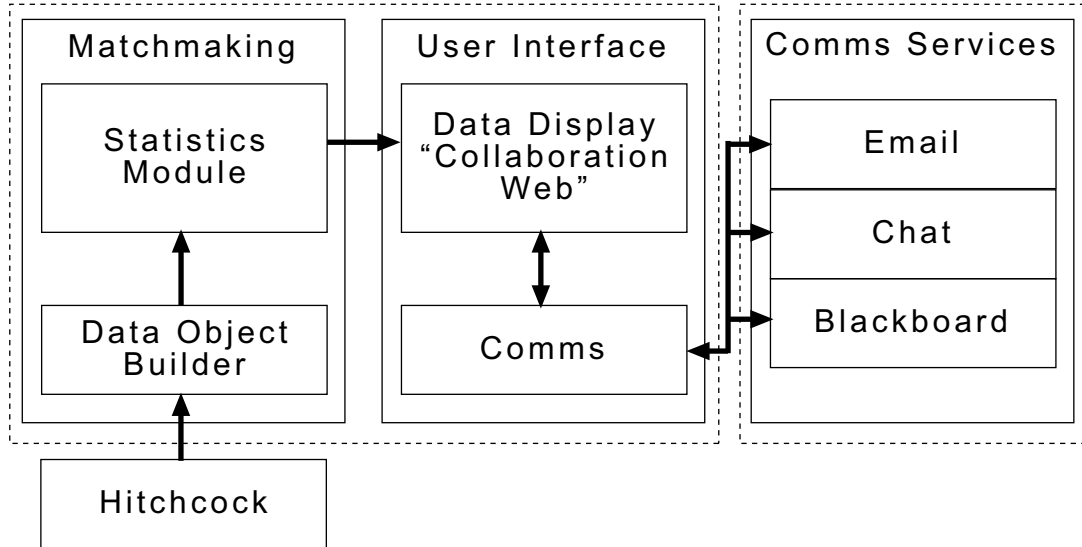


Figure 6: Components of the TMA1 agent.

feedback given by a user it is no longer presented. This also causes the ‘Top Picks’ page to be updated. Once a recommendation has been seen by a user, it is moved down the list or replaced by another ‘Top Pick’.

4.4 Evaluation

COWBRA’s effectiveness can only be measured by rolling it out to a ‘live’ environment. As this is a work in progress this has not yet been possible. However, there is a regular user group for Hitchcock and a large quantity of data has been collected. This data set has been used to test profile creation and management during the development of COWBRA. Early trials of the profile visualisation tool have shown it to work effectively and user feedback has been positive. The effectiveness of the Recommendation Engine has still to be tested fully. Our experience during development implies that it is behaving as expected, but only when the existing Hitchcock users begin to use COWBRA in earnest will we be certain.

5 Matchmaking Users

Matchmaking services are a natural extension of most recommender systems. At sites such as Firefly³, music and video recommendations provide the starting point for new relationships to develop via on-line chat services. TMA1 (Figure 6) is a simple matchmaking agent which identifies groupings within a Hitchcock user community based upon browsing behaviour. It attempts to promote communication and collaboration by highlighting users who share common interests.

5.1 The Statistics Module

TMA1 uses data gathered from the Hitchcock API to construct user profiles. Each user’s profile consists of a series of Hitchcock *browse-events*, represented within TMA1 as sparse binary vectors⁴. Once these profiles have been created a method is required to measure the degree of similarity between them. There are many ways to compare individual *browse-events* so a modular system was devised which allows different metrics to be used. The similarity between two profiles (A, B) is calculated using Equation 3, where $f(Be_i, Be_j)$ is the function used to determine the degree of similarity between two *browse-events*. Currently, the only function implemented is a variant of the simple (Hamming) overlap metric.

$$AvgSim = \frac{\sum_{i \in profile_A} \sum_{j \in profile_B} f(Be_i, Be_j)}{|profile_A| \cdot |profile_B|} \quad (3)$$

³<http://www.firefly.com>

⁴We acknowledge that using the data held within Hitchcock it should be possible to construct vectors containing term frequency values (or other measures of term weight). However, binary vectors were felt to be sufficient for the first version of TMA.

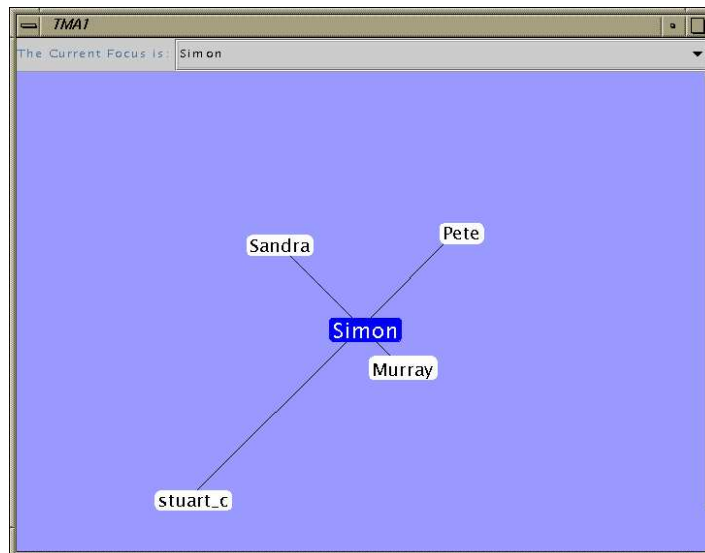


Figure 7: The TMA1 matchmaking display.

5.2 The TMA1 Interface

Displaying the relationships between users and facilitating communication between them is a critical goal of TMA1. Users are displayed in a *collaboration web*. This is a graph with a centralised node representing the current user (the *focus*) and a series of nodes clustered around it which represent the other users in the system (see Figure 7). A user only appears in the web if their similarity to the *focus* is above a threshold value (set by the user). The greater the degree of similarity between a user and the *focus*, the shorter the distance between their nodes on the display. Users are able to interact directly with the nodes using a mouse, allowing the *focus* to be changed or information about a specific user to be displayed. These details include name, email address and list of terms in common with the *focus* node. This user display window is also the key to communication in TMA1, allowing users to email one another from within the system; a simple chat function has also been implemented, as has a message blackboard.

5.3 Evaluation

TMA1 acts as a visual recommender agent by highlighting interest groups among users. At present, it is only able to create groups based on the global similarity of user profiles, i.e. the system does not consider the fine-grained detail of each user's browsing activities. In tests with a small initial user group, this was identified as a weakness as users felt that TMA1 was unable to form specialist thematic groupings, only very general ones.

6 Related Work

A range of strategies have been employed by recommender systems to suggest URLs to users. *PHOAKS* [25] mines the content of Usenet newsgroup discussions to identify Web page recommendations using a set of simple heuristics. Results are then presented on the PHOAKS site, together with details of who recommended them and why. Pazzani and Billsus [18] describe an agent which recommends documents (Web pages, PostScript or PDF files) related to those already visited at a site. A set of heuristics are used to determine when two items are related, and a user profile then employed to select amongst the alternative recommendations. *Fab* [3] is a hybrid URL recommender system that uses both content-based and collaborative techniques. Unlike COWBRA, it actively searches the Web for pages using a dynamically changing population of agents that represent topics that are of interest to users. These agents supply pages to a central router that passes them on to selection agents where they are scored against each user's personal profile. Recommendations are also made when a user with a similar profile rates a page highly. *Jasper* [5] supports collaborative Web browsing using one profile per user, instead of the family of profiles employed by COWBRA. Another difference is that pages are only stored by Jasper at the instigation of the user. Consequently, some pages that might be of potential interest to other users will not be stored and hence cannot be recommended. *Do-I-Care* [24] recommends pages (from a user-defined lists of URLs) that contain "interesting" updates. The system acquires a model of what constitutes an interesting change based upon user feedback.

As mentioned earlier, systems such as HOMR/Ringo/Firefly [23] facilitate user matchmaking via recommendations. An alternative approach is that taken by *Yenta* [10, 9] which performs matchmaking using decentralised, self-organising agents, each of which encapsulates a single user interest. These agents group themselves into clusters which are then used to introduce, or find, other users who share this interest. *Yenta* uses labelled clusters of documents to represent interests.

The MEMOIR [6, 19] project is highly relevant to our work as it also aims to create an information management architecture based upon existing Web infrastructure. In MEMOIR, data storage is handled by an object-oriented database system, while data manipulation, user-profiling and other tasks are performed by agents of varying levels of sophistication. Low-level agents typically handle simple data manipulation tasks, such as matching and retrieving URLs. Higher-level (information broker) agents rely on these services to perform their own tasks; for example, finding an expert in a particular domain. The MEMOIR architecture also includes an *interface manager* (acting as a proxy server), a *message router*, and one or more *agent servers* (to manage the various agent groupings). Hitchcock embodies a different philosophy, that of a central data store with fully integrated data management support. We feel that agent developers should concern themselves with the high-level design of their systems and should be able to request the data they require in a suitable format via an appropriately designed API. Within MEMOIR there is an obvious communication overhead between the various agents and sub-systems, which we were keen to avoid.

7 Discussion & Future Plans

We begin by discussing the two agent systems described in this paper, before reflecting on the current level of functionality provided by the proxy cache and its associated API.

COWBRA currently passes much of the overhead associated with profile creation directly onto the user. The agent presents a set of *browse-events* which are first grouped together by the user (into profiles) and then rated as 'like'/'dislike'. A desirable extension is the use of clustering techniques [14] to identify potential groupings of *browse-events*, which can then be assessed by the user. We are also interested in exploring mechanisms that would allow users to manage their profiles via some form of visual abstraction, rather than by interacting with a set of document instances (as is often the case in such systems). The existing profile visualisation tool has been received positively by users and would seem to be an obvious starting point for development of an editing capability. However, there are difficulties in ensuring that the display remains consistent with the underlying profile model.

As discussed earlier, TMA1 currently assesses the global similarity of users. An obvious extension would be a facility for identifying local matches by representing each user as a series of sub-profiles which are then compared. This would allow a user with interests in areas as diverse as *software agents*, *arabian horses* and *thai food* to be matched with another user who only shared one of these interests. TMA1 would currently fail to assign a sufficiently high similarity score to such a match and would therefore be unlikely to bring it to the attention of the user. One strength of TMA1 is its modular design, with use of dynamic class loading to allow new statistical analysis methods to be deployed easily. There is scope for the introduction of various mechanisms here, including perhaps clustering.

Although COWBRA and TMA1 are usable recommender agent systems in their own right, of greater significance to us were the lessons learnt during their design and construction. As anticipated, both agents contain large amounts of code to handle data representation and manipulation and there is clearly scope for the addition of functionality to the existing Hitchcock proxy cache and the API. The list below summarises a number of possible extensions:

Alternative Information Sources Hitchcock should be extended to act as a repository for more than just Web page data. Possibilities are: Usenet articles, XML, PostScript, PDF, and \LaTeX document formats.

Information Extraction Methods The cache should support a more extensive range of techniques, such as stemming of terms [20]; tagging [4]; and methods based upon syntactic pattern matching [22].

Document Profile Management A more extensive set of API objects are needed to represent document instances within the cache. At present, only the *URLData* object exists. We are considering the use of a *DocumentData* class as the root of a hierarchy of such classes which would encompass the range of information sources listed above. Each of the subtypes would support a variety of representations including binary and weighted vectors.

User Profile Management We are exploring whether the proxy cache can be given responsibility for aspects of user profile creation and storage. In other words, whether profiles can be considered as first class objects - *ProfileData*. Profile management would be provided via the API, allowing `create`, `edit` and `remove`

operations. The representation of profile objects within the cache would clearly have to reflect the range of possible “external” roles of such profiles. For example, while some agent developers may be happy with a collection of documents, others may require labels to be attached to each instance, and others may require some higher-level abstraction of the data. The API will need careful design to support such a range of profile “views”.

Object Database Support Hitchcock currently relies upon a relational database server to manage *browse-event* data. It is apparent to us that for future development of the cache we should move to an object database.

Classification & Clustering Support for classification algorithms (such as naive Bayes) would enable data held within the cache to be classified using the contents of appropriately labelled profiles. Clustering methods would allow aggregation of *DocumentData* or *ProfileData* entities; this could serve as a method for the creation of sub-profiles or as a user matchmaking mechanism.

8 Acknowledgements

We would like to express our thanks to all the members of the initial Hitchcock test group: Catriona, Sandra, Claire, John, Roma and Stuart. Darren Johns acknowledges financial support provided by IEE Publishing & Information Services.

References

- [1] MySQL Database Server. www.mysql.com. Version 3.23.
- [2] Squid Web Proxy Cache. www.squid-cache.org. Version 2.3.
- [3] BALABANOVIC, M., AND SHOHAM, Y. Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM* 40, 3 (1997), 66–72.
- [4] BRILL, E. Some Advances in Transformation-Based Part of Speech Tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI94)* (1994).
- [5] DAVIES, N., WEEKS, R., AND REVETT, M. Information Agents for the World Wide Web. *BT Technology Journal* 14, 4 (1996), 105–114.
- [6] DEROURE, D., HALL, W., REICH, S., PIKRAKIS, A., HILL, G., AND STAIRMAND, M. An Open Framework for Collaborative Distributed Information Management. In *Proceedings of Seventh International World Wide Web Conference (WWW7)* (1998), Elsevier, pp. 624–625.
- [7] EDWARDS, P., BAYER, D., GREEN, C., AND PAYNE, T. Experience with Learning Agents which Manage Internet-Based Information. In *AAAI Spring Symposium on Machine Learning in Information Access (SS-96-05)* (1996), Menlo Park, CA:AAAI, pp. 31–40.
- [8] EDWARDS, P., GREEN, C., LOCKIER, P., AND LUKINS, T. Exploiting Learning Technologies for World Wide Web Agents. In *IEE Colloquium on Intelligent World Wide Web Agents, Digest No: 97/118* (1997), pp. 3/1–3/7.
- [9] FONER, L. Yenta: A Multi-Agent, Referral-Based Matchmaking System. In *Proceedings of the First International Conference on Autonomous Agents (Agents’97)* (1997).
- [10] FONER, L., AND CRABTREE, I. Multi-Agent Matchmaking. *BT Technology Journal* 14, 4 (1996), 115–123.
- [11] GREEN, C., AND EDWARDS, P. Using Machine Learning to Enhance Software Tools for Internet Information Management. In *AAAI-96 Workshop on Internet-Based Information Systems (WS-96-06)* (1996), Menlo Park, CA:AAAI, pp. 48–55.
- [12] LAU, T., ETZIONI, O., AND WELD, D. Privacy Interfaces for Information Management. *Communications of the ACM* 42, 10 (1999), 89–94.
- [13] LIEBERMAN, H., VAN DYKE, N., AND VIVACQUA, A. Let’s Browse: A Collaborative Web Browsing Agent. In *Proceedings of the 1999 International Conference on Intelligent User Interfaces* (1999), Redondo Beach, CA, USA.

- [14] MARTIN, J. Clustering Full Text Documents. In *Proceedings of IJCAI-95 Workshop on Data Engineering for Inductive Learning* (1995).
- [15] MITCHELL, T. *Machine Learning*. McGraw-Hill, 1997, ch. 6, pp. 177–184.
- [16] PAYNE, T., AND EDWARDS, P. Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface. *Applied Artificial Intelligence* 11, 1 (1997), 1–32.
- [17] PAYNE, T., EDWARDS, P., AND GREEN, C. Experience with Rule Induction and k -Nearest Neighbour Methods for Interface Agents that Learn. *IEEE Transactions on Knowledge and Data Engineering* 9, 2 (1997), 329–335.
- [18] PAZZANI, M., AND BILLSUS, D. Adaptive Web Site Agents. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)* (1999).
- [19] PIKRAKIS, A., BITSIKAS, T., SFAKIANAKIS, S., HATZOPOULOS, M., DEROURE, D., HALL, W., REICH, S., HILL, G., AND STAIRMAND, M. MEMOIR - Software Agents for Finding Similar Users by Trails. In *Proceedings of PAAM98 - The Third International Conference on The Practical Application of Intelligent Agents and Multi-Agents* (1998), London, UK, pp. 453–466.
- [20] PORTER, M. F. An Algorithm for Suffix Stripping. *Program: Automated Library and Information Systems* 14, 1 (1980), 130–137.
- [21] RESNICK, P., AND VARIAN, H. Recommender Systems - Guest Editorial. *Communications of the ACM* 40, 3 (1997), 56–58.
- [22] RILOFF, E. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI93)* (1993), AAAI/MIT Press, pp. 811–816.
- [23] SHARDANAND, U., AND MAES, P. Social Information Filtering: Algorithms for Automating ‘Word of Mouth’. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)* (1995).
- [24] STARR, B., ACKERMAN, M., AND PAZZANI, M. Do-I-Care: A Collaborative Web Agent. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)* (1996), pp. 273–274.
- [25] TERVEEN, L., HILL, W., AND B., A. Collaborative Filtering to Locate, Comprehend, and Organize Collections of Web Sites. *SIGART Bulletin* 9 (1998), 10–17.