Knowledge-based recommender systems

Robin Burke
Department of Information and Computer Science
University of California, Irvine
burke@ics.uci.edu

(To Appear in the Encyclopedia of Library and Information Science.)

1. Introduction

Recommender systems provide advice to users about items they might wish to purchase or examine. Recommendations made by such systems can help users navigate through large information spaces of product descriptions, news articles or other items. As on-line information and e-commerce burgeon, recommender systems are an increasingly important tool. A recent survey of recommender systems is found in (Maes, Guttman & Moukas, 1999). See also (Goldberg et al. 1992), (Resnick, et al. 1994), and (Resnick & Varian, 1997) and accompanying articles.

The most well known type of recommender system is the collaborative- or social-filtering type. These systems aggregate data about customers' purchasing habits or preferences, and make recommendations to other users based on similarity in overall purchasing patterns. For example, in the Ringo music recommender system (Shardanand & Maes, 1995), users express their musical preferences by rating various artists and albums, and get suggestions of groups and recordings that others with similar preferences also liked.

Content-based recommender systems are classifier systems derived from machine learning research. For example, the NewsDude news filtering system is a recommender system that suggests news stories the user might like to read (Billsus & Pazzani, 1999). These systems use supervised machine learning to induce a classifier that can discriminate between items likely to be of interest to the user and those likely to be uninteresting.

A third type of recommender system is one that uses knowledge about users and products to pursue a knowledge-based approach to generating a recommendation, reasoning about what products meet the user's requirements. The PersonalLogic recommender system offers a dialog that effectively walks the user down a discrimination tree of product features.[1] Others have adapted quantitative decision support tools for this task (Bhargava, Sridhar & Herrick, 1999). The class of systems that we will concentrate on in this paper draws from research in case-based reasoning or CBR (Hammond, 1989; Kolodner, 1993; Riesbeck & Schank, 1989). The restaurant recommender Entree (Burke, Hammond & Cooper, 1996; Burke, Hammond & Young, 1997) makes its recommendations by finding restaurants in a new city similar to restaurants the user knows and likes.[2] The system allows users to navigate by stating their preferences with respect to a given restaurant, thereby refining their search criteria.

Each of these approaches has its strengths and weaknesses. As a collaborative filtering system collects more ratings from more users, the probability increases that

[1] <URL: http://www.personallogic.com/>
[2] <URL: http://infolab.ils.nwu.edu/entree/>

someone in the system will be a good match for any given new user. However, a collaborative filtering system must be initialized with a large amount of data, because a system with a small base of ratings is unlikely to be very useful. Further, the accuracy of the system is very sensitive to the number of rated items that can be associated with a given user (Shardanand & Maes, 1995). These factors contribute to a "ramp-up" problem: until there is a large number of users whose habits are known, the system cannot be useful for most users, and until a sufficient number of rated items has been collected, the system cannot be useful for a particular user.

A similar ramp-up problem is associated with machine learning approaches to recommendation. Typically, good classifiers cannot be learned until the user has rated many items. The NewsDude system avoids this problem by using a nearest-neighbor classifier that works with few examples, but the system can only base its recommendations on ratings it has, and cannot recommend stories unless they are similar to ones the user has previously rated.

A knowledge-based recommender system avoids some of these drawbacks. It does not have a ramp-up problem since its recommendations do not depend on a base of user ratings. It does not have to gather information about a particular user because its judgements are independent of individual tastes. These characteristics make knowledge-based recommenders not only valuable systems on their own, but also highly complementary to other types of recommender systems. We will return to this idea at the end of this article.

1.1 Example

Figure 1 shows the initial screen for the Entree restaurant recommender. The user starts with a known restaurant, Wolfgang Puck's "Chinois on Main" in Los Angeles. As shown in Figure 2, the system finds a similar Chicago restaurant that combines Asian and French influences, "Yoshi's Cafe."[3] The user, however, is interested in a cheaper meal and selects the "Less $$" button. The result in Figure 3 is a creative Asian restaurant in a cheaper price bracket: "Lulu's." Note, however, that the French influence has been lost – one consequence of the move to a lower price bracket.

Figures 4 through 7 show a similar interaction sequence with the knowledge-based recommender system at the e-commerce portal site "Recommender.com". The search starts when the user enters the name of a movie that he or she liked, "The Verdict," a courtroom drama starring Paul Newman. The system looks up this movie and finds a handful of others that are similar, one of which appears in Figure 5. The top-rated recommendation is a comedy, however, and the user, in this case, wants something more suspenseful. The "Add Feature" menu seen in Figure 6 allows the user to push the search in a slightly different direction, specifying that that the movie must also have a "Mystery & Suspense" component. Figure 7 shows the results of this search: the system finds "The Jagged Edge." This movie combines courtroom drama with murder mystery.

[3] Note that the connection between "Pacific New Wave" cuisine and its Asian and French culinary components is part of the system's knowledge base of cuisines.

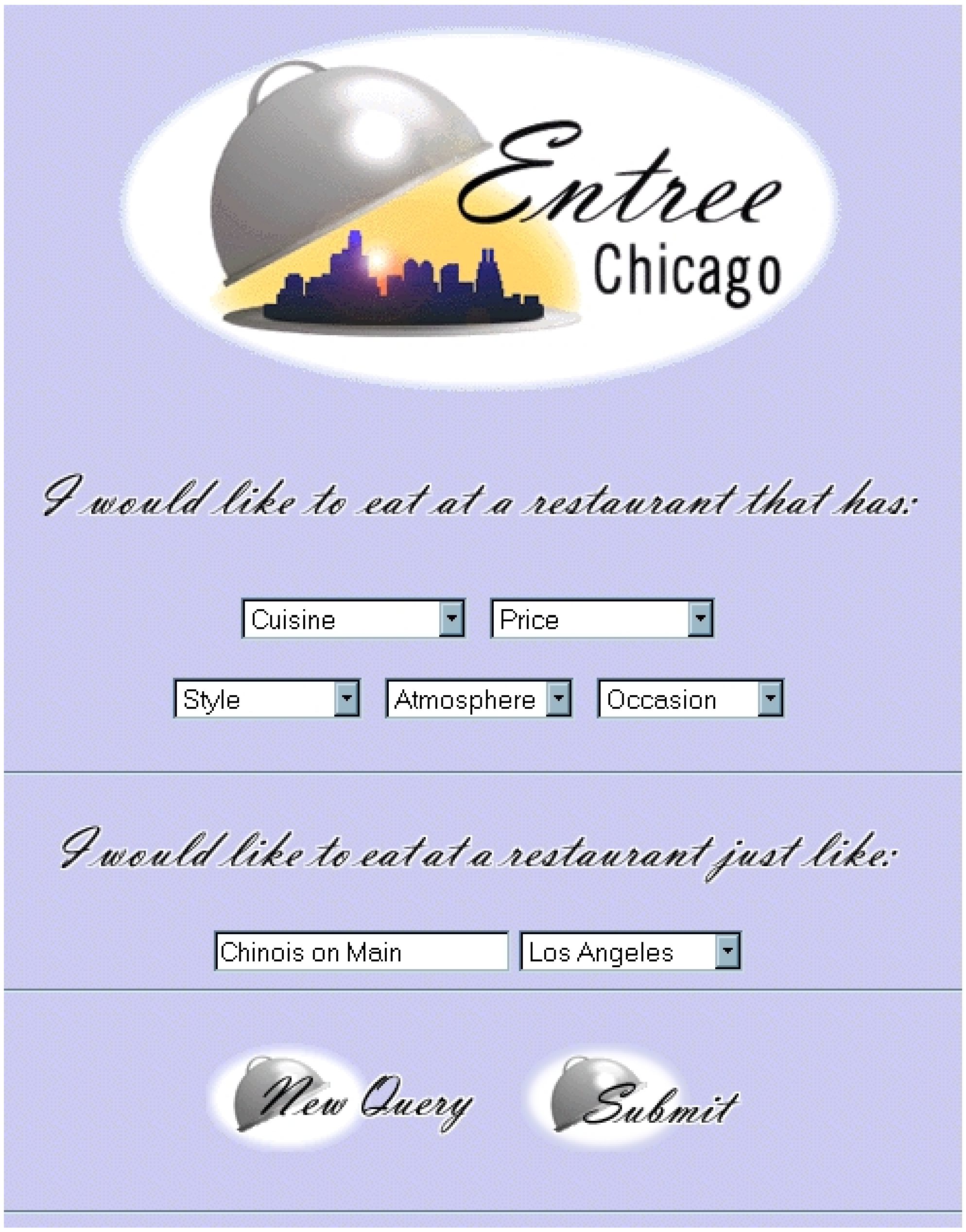
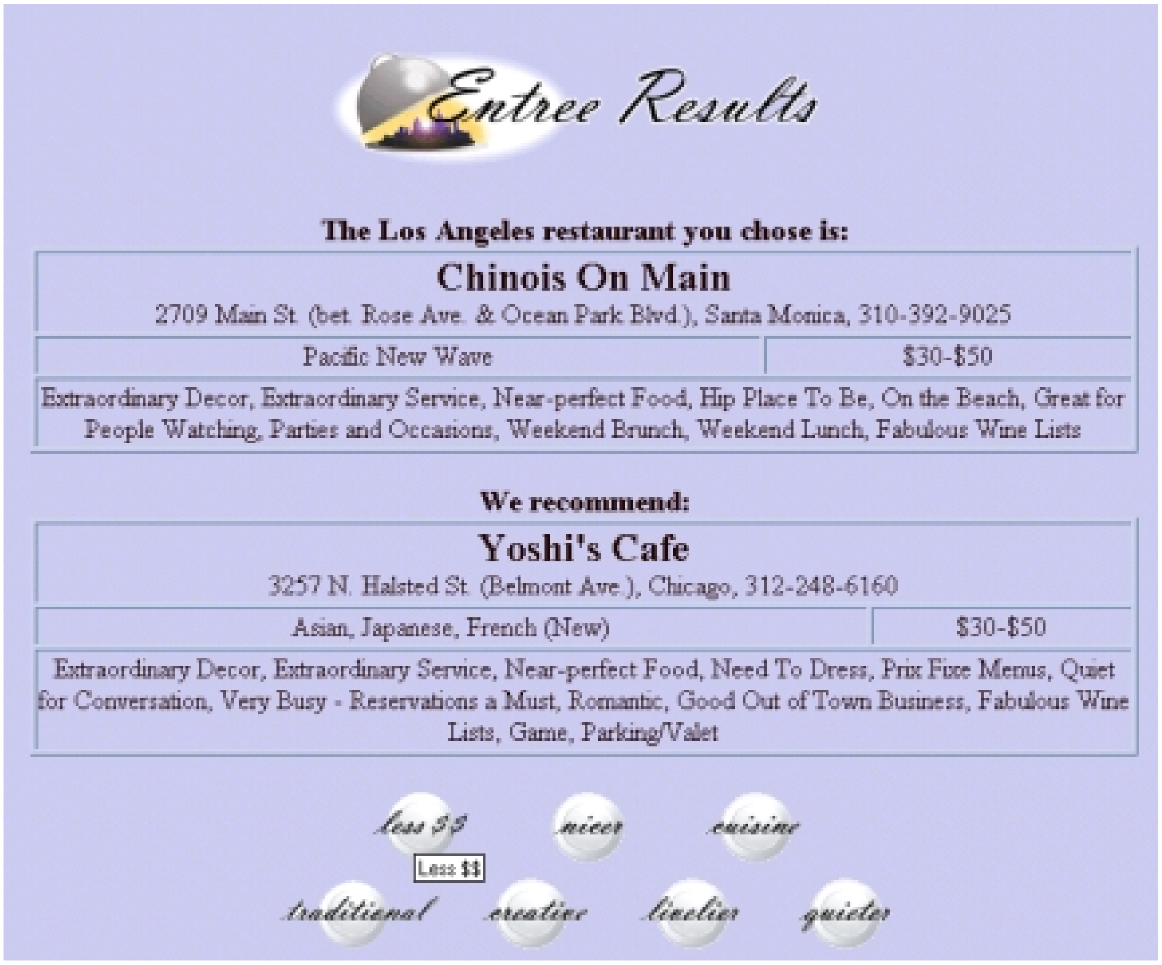

Entree
Chicago
I would like to eat at a restaurant that has:
Cuisine
Price
Style
Atmosphere
Occasion
I would like to eat at a restaurant just like:
Chinois on Main
Los Angeles
New Query
Submit

Figure 1: Entry point for the Entree system



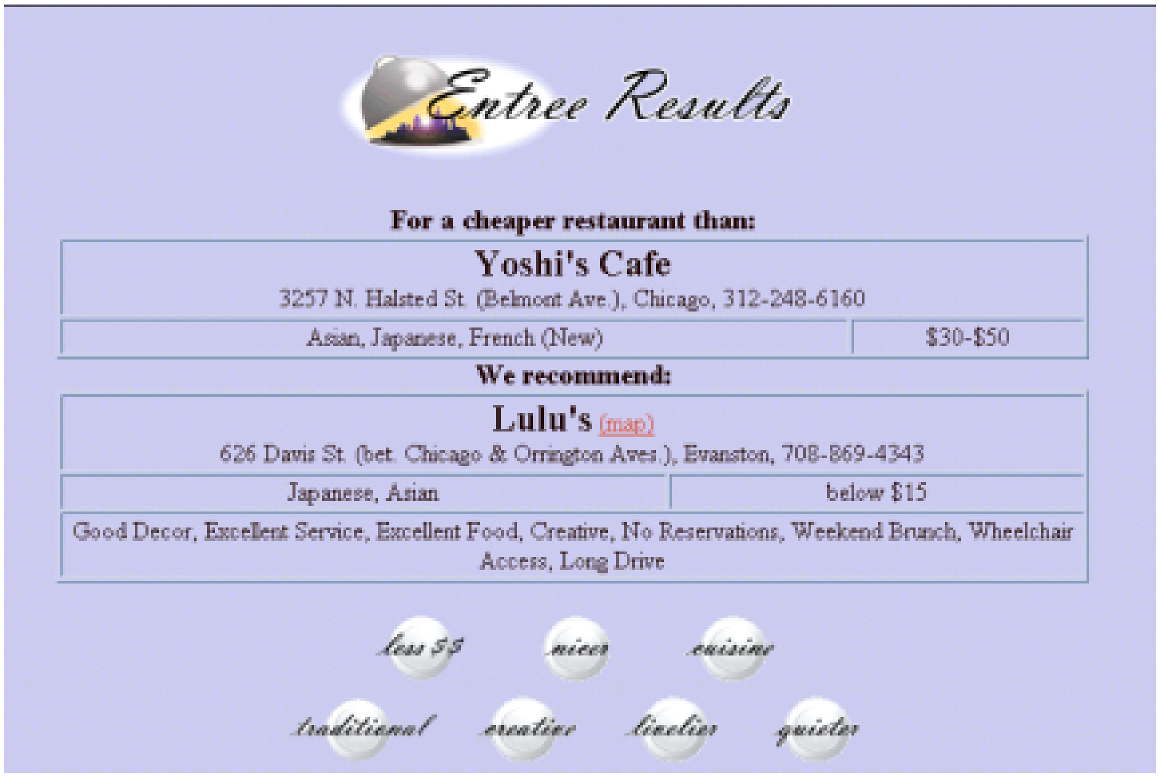
The Los Angeles restaurant you chose is:

Chinois On Main 2709 Main St. (bet. Rose Ave. & Ocean Park Blvd.), Santa Monica, 310-392-9025	
Pacific New Wave	$30-$50
Extraordinary Decor, Extraordinary Service, Near-perfect Food, Hip Place To Be, On the Beach, Great for People Watching, Parties and Occasions, Weekend Brunch, Weekend Lunch, Fabulous Wine Lists	

We recommend:

Yoshi's Cafe 3257 N. Halsted St. (Belmont Ave.), Chicago, 312-248-6160	
Asian, Japanese, French (New)	$30-$50
Extraordinary Decor, Extraordinary Service, Near-perfect Food, Need To Dress, Prix Fixe Menus, Quiet for Conversation, Very Busy - Reservations a Must, Romantic, Good Out of Town Business, Fabulous Wine Lists, Game, Parking/Valet	

Figure 2: Similarity-based retrieval in Entree



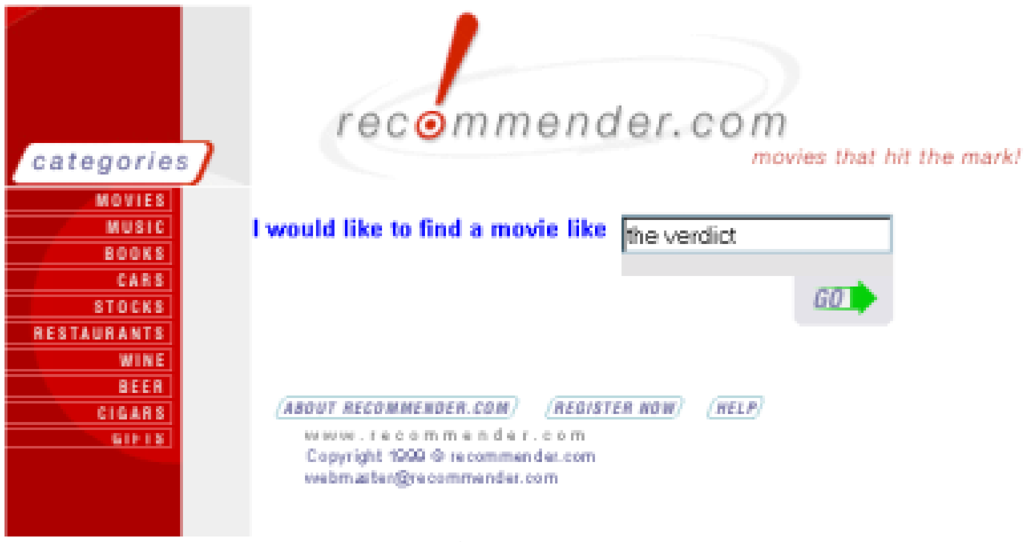
For a cheaper restaurant than:

Yoshi's Cafe 3257 N. Halsted St. (Belmont Ave.), Chicago, 312-248-6160	
Asian, Japanese, French (New)	$30-$50

We recommend:

Lulu's (map) 626 Davis St. (bet. Chicago & Orrington Aves.), Evanston, 708-869-4343	
Japanese, Asian	below $15
Good Decor, Excellent Service, Excellent Food, Creative, No Reservations, Weekend Brunch, Wheelchair Access, Long Drive	

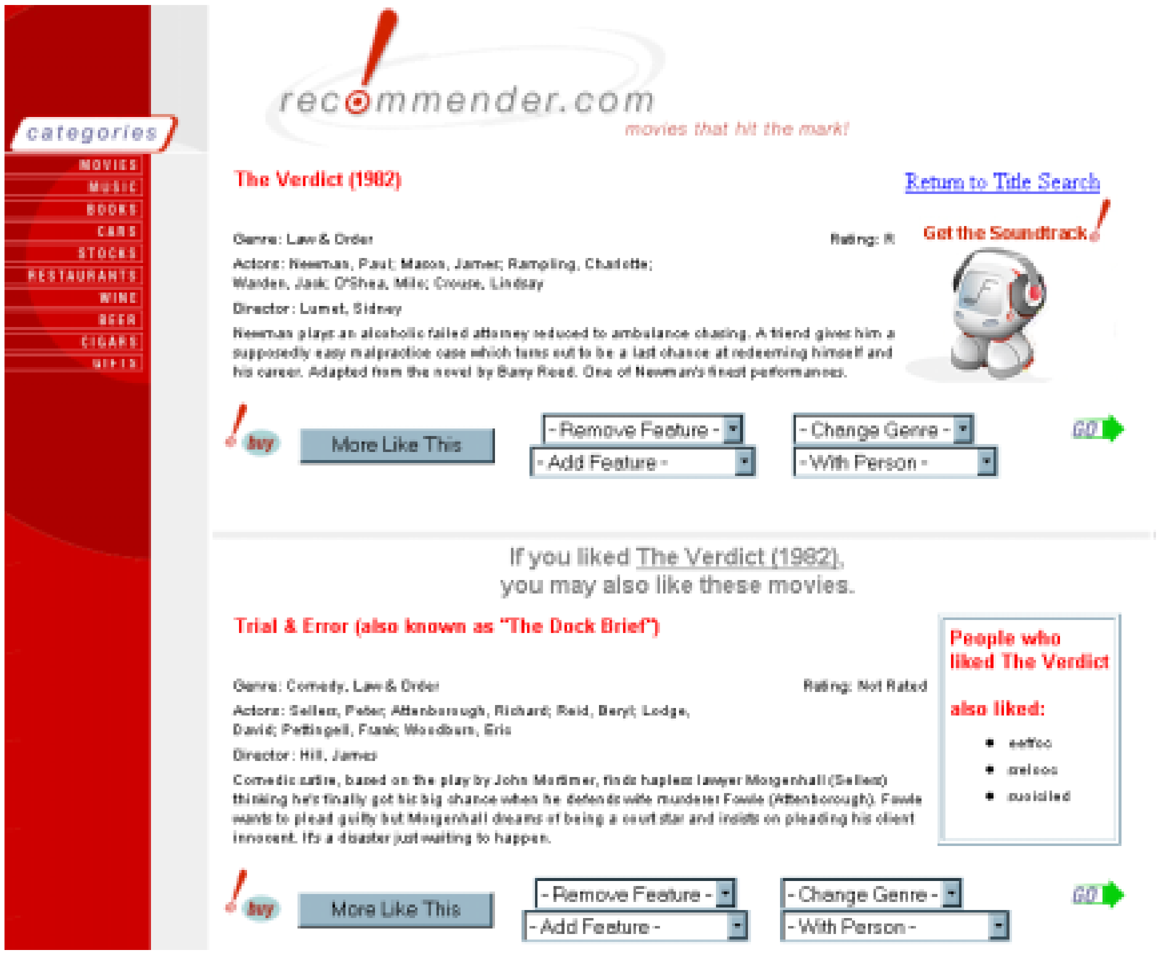
Figure 3: Navigation using the "Less $$" tweak



I would like to find a movie like the verdict

www.recommender.com
Copyright 1999 © recommender.com
webmaster@recommender.com

Figure 4: Entry point for Recommender.com movie recommender



The Verdict (1982)

Genre: Law & Order Rating: R

Actors: Newman, Paul; Mason, James; Rampling, Charlotte; Warden, Jack; O'Shea, Milo; Crouse, Lindsay

Director: Lumet, Sidney

Newman plays an alcoholic failed attorney reduced to ambulance chasing. A friend gives him a supposedly easy malpractice case which turns out to be a last chance at redeeming himself and his career. Adapted from the novel by Barry Reed. One of Newman's finest performances.

If you liked The Verdict (1982),
you may also like these movies.

Trial & Error (also known as "The Dock Brief")

Genre: Comedy, Law & Order Rating: Not Rated

Actors: Sellers, Peter; Attenborough, Richard; Reid, Beryl; Lodge, David; Pettingell, Frank; Woodburn, Eric

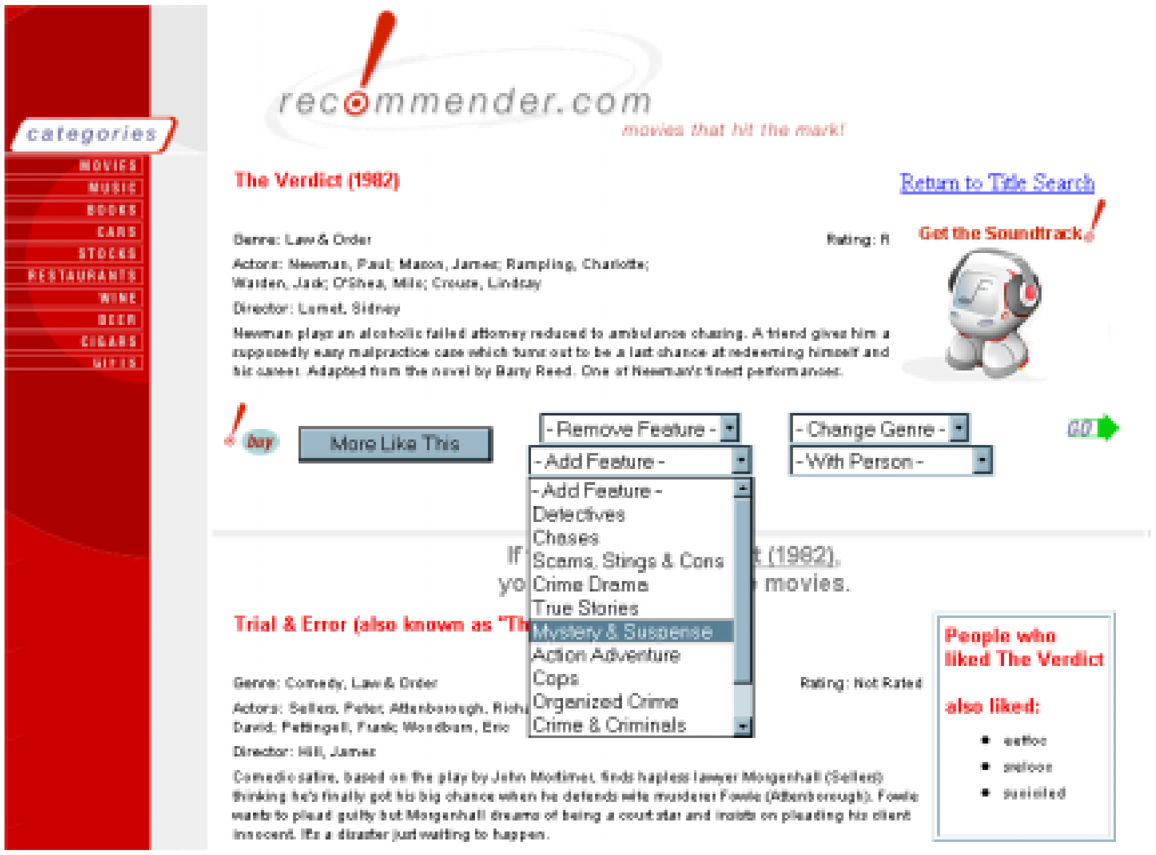
Director: Hill, James

Comedic satire, based on the play by John Mortimer, finds hapless lawyer Morgenhall (Sellers) thinking he's finally got his big chance when he defends wife murderer Fowle (Attenborough). Fowle wants to plead guilty but Morgenhall dreams of being a court star and insists on pleading his client innocent. It's a disaster just waiting to happen.

People who liked The Verdict also liked:

- [illegible]
- [illegible]
- [illegible]

Figure 5: Similarity based retrieval in the movie recommender



The Verdict (1982)

Genre: Law & Order — Rating: R

Actors: Newman, Paul; Mason, James; Rampling, Charlotte; Warden, Jack; O'Shea, Milo; Crouse, Lindsay

Director: Lumet, Sidney

Newman plays an alcoholic failed attorney reduced to ambulance chasing. A friend gives him a supposedly easy malpractice case which turns out to be a last chance at redeeming himself and his career. Adapted from the novel by Barry Reed. One of Newman's finest performances.

If ... t (1982), yo ... movies.

Trial & Error (also known as "Th

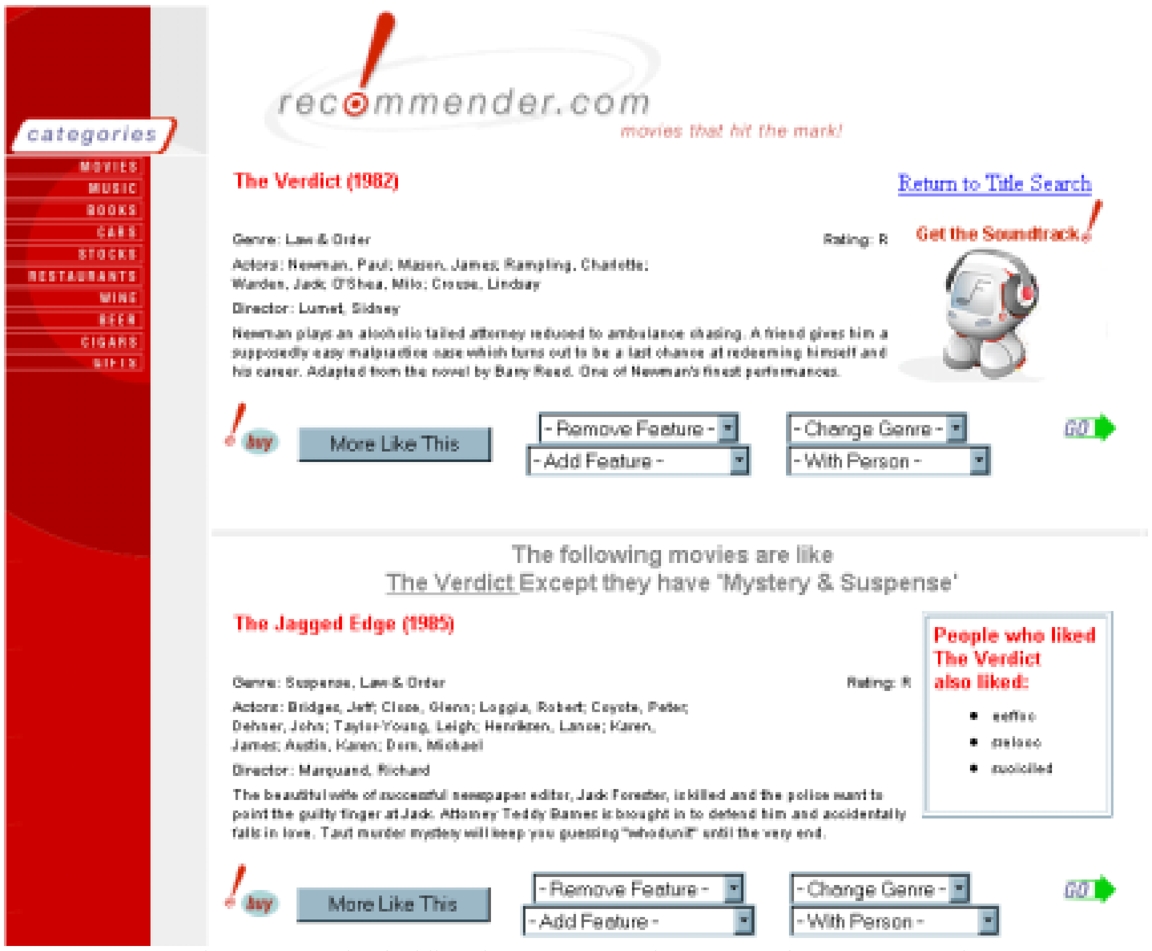
Genre: Comedy, Law & Order — Rating: Not Rated

Actors: Sellers, Peter; Attenborough, Richa... David; Pettingell, Frank; Woodburn, Eric

Director: Hill, James

Comedic satire, based on the play by John Mortimer, finds hapless lawyer Morgenhall (Sellers) thinking he's finally got his big chance when he defends wife murderer Fowle (Attenborough). Fowle wants to plead guilty but Morgenhall dreams of being a court star and insists on pleading his client innocent. It's a disaster just waiting to happen.

Figure 6: Applying the "Add Feature" tweak



The Verdict (1982)

Genre: Law & Order | Rating: R

Actors: Newman, Paul; Mason, James; Rampling, Charlotte; Warden, Jack; O'Shea, Milo; Crouse, Lindsay

Director: Lumet, Sidney

Newman plays an alcoholic failed attorney reduced to ambulance chasing. A friend gives him a supposedly easy malpractice case which turns out to be a last chance at redeeming himself and his career. Adapted from the novel by Barry Reed. One of Newman's finest performances.

The following movies are like
The Verdict Except they have 'Mystery & Suspense'

The Jagged Edge (1985)

Genre: Suspense, Law & Order | Rating: R

Actors: Bridges, Jeff; Close, Glenn; Loggia, Robert; Coyote, Peter; Dehner, John; Taylor-Young, Leigh; Henriksen, Lance; Karen, James; Austin, Karen; Dorn, Michael

Director: Marquand, Richard

The beautiful wife of successful newspaper editor, Jack Forester, is killed and the police want to point the guilty finger at Jack. Attorney Teddy Barnes is brought in to defend him and accidentally falls in love. Taut murder mystery will keep you guessing "whodunit" until the very end.

People who liked The Verdict also liked:

- [illegible]
- [illegible]
- [illegible]

Figure 7: Result of adding the "Mystery and Suspense" feature as a tweak

2. History

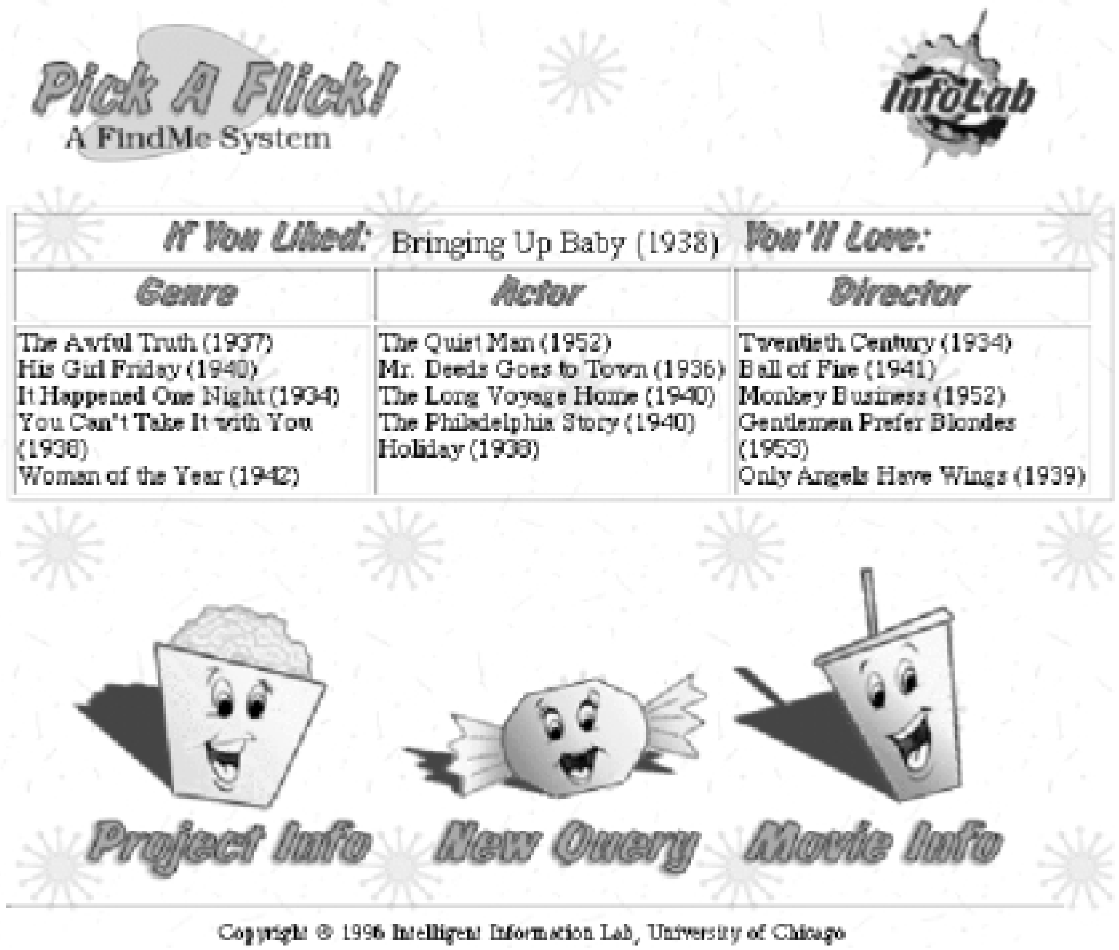
Both Entree and Recommender.com are FindMe knowledge-based recommender systems. FindMe systems are distinguished from other recommender systems by their emphasis on examples to guide search and on the search interaction, which proceeds through tweaking or altering the characteristics of an example.

The FindMe technique is one of knowledge-based similarity retrieval. There are two fundamental retrieval modes: similarity-finding and tweak application. In the similarity case, the user has selected a given item from the catalog (called the *source*) and requested other items similar to it. To perform this retrieval, a large set of candidate entities is initially retrieved from the database. This set is sorted based on similarity to the source and the top few candidates returned to the user.

Tweak application is essentially the same except that the candidate set is filtered prior to sorting to leave only those candidates that satisfy the tweak. For example, if a user responds to item X with the tweak “Nicer,” the system determines the “niceness” value of X and rejects all candidates except those whose value is greater.

The first FindMe system was the Car Navigator, an information access system for descriptions of new car models. In this system, cars were rated against a long list of criteria such as horsepower, price or gas mileage, which could be directly manipulated. Retrieval was performed by turning the individual criteria into a similarity-finding query to get a new set of cars. After some experimentation with this interface, we added the capability of making large jumps in the feature space through buttons that alter many variables at once. If the user wanted a car “sportier” than the one he was currently examining, this would imply a number of changes to the feature set: larger engine, quicker acceleration, and a willingness to pay more, for example. The introduction of these buttons marked the beginning of what is now the FindMe signature: conversational interaction focused around high-level responses to particular examples, rather than on retrieval based on fine-grained details. Although direct manipulation of the features was appealing in some situations, we found that most users preferred to use these tweaks to redirect the search.

For our next prototype, we turned our attention to the more complex domain of movies, which had already gotten attention from collaborative filtering researchers. Here we returned to a retrieval approach, letting users find movies similar to ones they already knew and liked. Our movie recommender PickAFlick made several sets of suggestions, introducing the idea of multiple retrieval strategies, different ways of assessing the similarity of items. If a PickAFlick user entered the name of the movie “Bringing Up Baby,” a classic screwball comedy starring Cary Grant and Katharine Hepburn, the system would locate similar movies using three different strategies. First, it would look for movies that are similar in genre: other fast-paced comedies. As Figure 8 shows, it finds “His Girl Friday,” another comedy from the same era starring Cary Grant, as well as several others. The second strategy looks for movies with similar casts. This strategy will discard any movies already recommended, but it finds more classic comedies, in particular “The Philadelphia Story,” which features the same team of Grant and Hepburn. The director strategy returns movies made by Howard Hawks, preferring those of a similar genre.


Pick A Flick!
A FindMe System
InfoLab
Project Info
New Query
Movie Info
Copyright © 1996 Intelligent Information Lab, University of Chicago

If You Liked: Bringing Up Baby (1938) You'll Love:

Genre	Actor	Director
The Awful Truth (1937) His Girl Friday (1940) It Happened One Night (1934) You Can't Take It with You (1938) Woman of the Year (1942)	The Quiet Man (1952) Mr. Deeds Goes to Town (1936) The Long Voyage Home (1940) The Philadelphia Story (1940) Holiday (1938)	Twentieth Century (1934) Ball of Fire (1941) Monkey Business (1952) Gentlemen Prefer Blondes (1953) Only Angels Have Wings (1939)

Figure 8: Multi-strategy retrieval in PickAFlick

The following system RentMe was an apartment-finding recommender system. Unlike cars and movies, there is no easy way to name particular apartments, so our standard entry point, a known example, was not effective in this domain. We had to present a fairly traditional set of query menus to initiate the interaction. The list of apartments meeting these constraints forms the starting point for continued browsing. RentMe used natural language processing to generate its database, starting from a text file of classified ads for apartments. The terse and often-agrammatical language of the classified ads would have been difficult to parse rigorously, but a simple expectation-based parser (Schank & Riesbeck, 1981) worked well, much better than simple keyword extraction.

Entree was our first FindMe system that was sufficiently stable, robust and efficient to serve as a public web site. All of the previous FindMe systems were implemented in Common Lisp and kept their entire corpus of examples in memory. While this design had the advantage of quick access and easy manipulation of the data, it was not scalable to very large data sets. The Entree system was written C++ and used an external database for its restaurant data. It has been publicly accessible on the web since August of 1996.

Kenwood, the last domain-specific FindMe system, allowed users to navigate through configurations for home theater systems. The user could browse among the configurations by adjusting the budget constraint, the features of the room or by adding, removing or replacing components. Our database was not of individual stereo components and their features, but rather entire configurations and their properties. Since we were dealing with configurations of items, it was also possible to construct a system component by component and use that system as a starting point. This made the search space somewhat different than the other systems discussed so far, in that every combination of features that can be expressed actually exists in the system.[4]

3. Recommender Personal Shopper

The evolution of FindMe systems demonstrates several characteristics they share: (i) the centrality of examples, (ii) conversational navigation via tweaks, (iii) knowledge-based similarity metrics, and (iv) task-specific retrieval strategies. The recommendation engine of the Recommender.com site, the Recommender Personal Shopper (RPS), represents the culmination of the FindMe research program.[5] It is a domain-independent implementation of the FindMe algorithm that interfaces with standard relational databases. Our task in building RPS was to create a generic recommendation capability that could be customized for any domain by the addition of product data and declarative similarity knowledge.

3.1 Similarity

Our initial FindMe experiments demonstrated something that case-based reasoning researchers have always known, namely that similarity is not a simple or uniform concept. In part, what counts as similar depends on what one's goals are: a shoe is similar to a hammer if one is looking around for something to bang with, but not if one wants to extract nails. FindMe similarity measures therefore have to be goal-based, and consider

[4] An adapted version of Kenwood was part of the web presence for Kenwood, USA in 1997-1998.

[5] See also (Burke, 1999).

multiple goals and their tradeoffs. Typically, there are only a handful of standard goals in any given product domain. For each goal, we define a *similarity metric*, which measures how closely two products come to meeting the same goal. Two restaurants with the same price would get the maximum similarity rating on the metric of price, but may differ greatly on another metric, such as quality or type of cuisine.

Through the various FindMe prototypes, we looked at the interactions between goals, and experimented with combinations of metrics to achieve intuitive rankings of products. We found there were well-defined priorities attached to the most important goals and that they could be treated independently. For example, in the restaurant domain, cuisine is of paramount importance. Part of the reason is that cuisine is a category that more or less defines the meaning of other features – a high-quality French restaurant is not really comparable to a high-quality burger joint, partly because of what it means to serve French cuisine.

We can think of the primary category as the most important goal that a recommendation must satisfy, but there are other goals that must be factored into the similarity calculation. For example, in the Entree restaurant recommender system, the goals were cuisine, price, quality, and atmosphere applied in rank order, which seemed to capture our intuition about what was important about restaurants. It is of course possible that different users might have different goal orderings or different goals altogether. A FindMe system may therefore have several different *retrieval strategies*, each capturing a different notion of similarity. A retrieval strategy selects the goals to be used in comparing entities, and orders them giving rise to different assessments of similarity. PickAFlick, for example, created its multiple lists of similar movies by employing three retrieval strategies: one that concentrated on genre, another focused on actors, and a third that emphasized direction.

3.2 Sorting algorithm

The FindMe sorting algorithm begins with the source entity S, the item to which similarity is sought, such as the initial entry point provided by the user, and a retrieval strategy R, which is an ordered list of similarity metrics $M_1..M_m$. The task is to return a fixed-size ranked list of target entities of length n, $T_{1..n}$, ordered by their similarity to S. Our first task is to obtain an unranked set of candidates $T_{1..j}$ from the product database. This retrieval process is discussed in the next section.

Similarity assessment is an alphabetic sort, using a list of buckets. Each bucket contains a set of target entities. The bucket list is initialized so that the first bucket B_1 contains all of $T_{1..j}$. A sort is performed by applying the most important metric M_1, corresponding to the most important goal in the retrieval strategy. The result is a new set of buckets $B_{1..k}$, each containing items that are given the same integer score by M_1. Starting from B_1, we count the contents of the buckets until we reach n, the number of items we will ultimately return, and discard all remaining buckets. Their contents will never make it into the result list. This process is then repeated with the remaining metrics until there are n singleton buckets remaining (at which point further sorting would have no effect) or until all metrics are used.

This multi-level sort can be replaced by a single sort, provided that the score for each target entity can be made to reflect what its position would be in the more complex version. Consider the case of two metrics, M_1 and M_2. Let b_i be the upper bound on the score for comparing a target entity against S with metric M_i, that is $b_i > \max (M_i(S, T)$, for any target entity T). The single-pass scoring function for the combination of these two metrics would be $S(S, T) = M_2(S, T) + M_1(S, T) * b_2$. With this function, we can sort the target entity list and end up with the same set of buckets that we would have obtained with a two-pass sort applying first M_1 and then M_2. In the general case, the scoring function becomes

$$S(S, T) = \sum_{i=1..m} \left(M_i(S, T) * \prod_{j=i+1..m} b_j\right)$$

where m is the number of metrics.[6]

A final optimization to note is that we are rarely interested in a complete sort of the candidate list. Generally, we are returning a small set of the best answers, five in the case of the movie recommender. We can get the top n targets by performing n O(L) max-finding operations where L is the length of the candidate list. When the list is large ($L > 2^n$), this is faster than performing an O ($L \log L$) complete sort.

The max finding operation can be optimized for this comparison function by applying metrics in decreasing order of importance (and multiplier magnitude). High- and low-scoring targets may not need more than one or two metric applications to rule them in or out of the top n.

3.3 Retrieval algorithm

Our original implementations of the FindMe algorithm retrieved large candidate sets. We used promiscuous retrieval deliberately because other steps (such as tweaking steps) filtered out many candidates and it was important not to exclude any potentially useful target. In our Lisp implementations, the use of a large candidate set was reasonably efficient since the candidates were already in memory. We found this not to be true as we moved to relational databases for storing entity data. Queries that return large numbers of rows are highly inefficient, and each retrieved entity must be allocated on the heap. Employed against a relational store, our original algorithms yielded unacceptable response times, sometimes greater than 10 minutes. It was necessary therefore to retrieve more precisely – to get back just those items likely to be highly rated by the sort algorithm.

Our solution was a natural outgrowth of the metric and strategy system that we had developed for sorting, and was inspired by the CADET system, which performs nearest-neighbor retrieval in relational databases (Shimazu, Kitano & Shibata, 1993). Each metric became responsible for generating retrieval constraints based on the source entity. These constraints could then be turned into SQL clauses when retrieval took place. This approach was especially powerful for tweaks. A properly-constrained query for a tweak such as “cheaper” will retrieve only the entities that will actually pass the “cheaper” filter, avoiding the work of reading and instantiating entities that would be immediately discarded.

The retrieval algorithm works as follows. To retrieve candidates for comparison against a source entity, each metric creates a constraint. The constraints are ordered by the priority of the metric within the current retrieval strategy. If the query is to be used for a tweak, a constraint is created that implements the tweak and is given highest priority.

[6] As a practical matter, it should be noted that if there are too many metrics with too large a scoring range, this function will become very large. For example, six metrics of range 50 already exceeds the capacity of a 32 bit unsigned integer: $50^6 > 2^{32}$.

This constraint is considered “non-optional.” An SQL query is created conjoining all the constraints and is passed to the database. If no entities (or not enough) are returned, the lowest priority constraint is dropped and the query resubmitted. This process can continue until all of the optional constraints have been dropped.

The interaction between constraint set and candidate set size is dramatic: a four-constraint query that returns nothing will often return thousands of entities when relaxed to three constraints. We are considering a more flexible constraint scheme in which each metric would propose a small set of progressively more inclusive constraints, rather than just one. Since database access time dominates all other processing time in the system, we expect that any additional computation involved would be outweighed by the efficiencies to be had in more accurate retrieval.

3.4 Product data

The generic FindMe engine implemented in RPS knows nothing about restaurants, movies or any other domain of recommendation. It simply applies similarity metrics to entities that are described as feature sets. Our architecture has therefore decomposed the task of creating a recommender system into two parts: the creation of a product database in which unique items are associated with sets of features, and the specification of the similarity metrics and retrieval strategies that are appropriate for those items.

An entity is represented in RPS simply as a set of integer features. This representation is extremely generic, compact and efficient, and it can be easily stored in a relational database. The product database is single table that associates an entity ID with its features.

To create a feature set for a product, we must make use of whatever information is available about the item’s qualities. In the domains where RPS has been applied, product databases typically consist of a handful of fields describing a product’s qualities, such as its price, and chunks of natural language text intended as a product description. Natural language processing is needed to make use of the descriptive information. It is important to note that we are interested only in comparing descriptions against each other: Is the dining experience at restaurant A like the experience at restaurant B? Is the experience of watching movie X similar to that of movie Y? We do not build sophisticated linguistic structures, but instead transform each natural language description into atomic features like those used to represent any other aspect of an entity.

Product descriptions in general tend to be not very complex syntactically, consisting largely of descriptive adjectives and nouns. Typically, there are several categories of information that are of interest. For restaurants, it might be qualities of the atmosphere (“loud”, “romantic”) or qualities of the cuisine (“traditional”, “creative”, “bland”); for wines, descriptions of the flavor of the wine (“berry”, “tobacco”), descriptions of the wine’s body and texture (“gritty”, “silky”), etc. For each category, we identify the most commonly used terms, usually nouns. We also identify modifiers both of quantity (“lots”, “lacking”) and quality (“lovely”, “ugly”). For some applications, this level of keyword identification is sufficient. In other cases, more in-depth analysis is required, including phrase recognition and parsing. Descriptions of wines, with their evocative language, have been the most difficult texts that we have tackled.

3.5 Metrics

Similarity metrics and retrieval strategies are really the heart of knowledge-based recommendation in a FindMe system. Metrics determine what counts a similar when two items are being compared; retrieval strategies determine how important different aspects of similarity are to the overall calculation. The creation of a new FindMe system requires the creation and refinement of these two crucial kinds of information.

A similarity metric can be any function that takes two entities and returns a value reflecting their similarity with respect to a given goal.[7] Our original FindMe systems implemented the similarity metric idea in many different domain-specific ways. For RPS, we have created a small set of metric types general enough to cover all of the similarity computations used in our other FindMe systems. One example is included here to give a flavor for the kinds of comparisons these metrics perform.

Price is an obvious candidate for a similarity metric, because most consumer items can be compared by price,. However, price is not a simple as it might seem. A user looking for a restaurant similar to restaurant X at price Y is indicating that he or she is willing to spend at least Y. Prices below Y shouldn't be penalized as different the way that prices above Y should be. Neither should prices below Y necessarily be preferred, since the user is evidently willing to spend that amount. A price comparison can therefore be implemented as a *directional scalar metric*, and has the following form:

Let S be the source entity, the item that the user has chosen.

Let T be the target entity that we are comparing against the source.

Let M be a directional metric with a decreasing preference for features in the set F (such as the set of price features).

Let $f_s, f_t \in F$ be features found in S and T, respectively.

Let b be the cardinality of the set F.

The score returned by the metric, $M(S, T)$, is given by

b, if $f_t <= f_s$

$b - (f_t - f_s)$, otherwise.[8]

If restaurant X has a price in the $30-50 price bracket and restaurant Y in the $50 and up price bracket, this metric will report that the restaurant Y gets a score of 7 (out of a possible 8) since it is one price bracket more expensive than X. Restaurant Z whose typical tab is in the $15-30 price bracket would get the maximum score of 8 for price – we do not penalize it for being cheaper than X.

Price is something of a special case since a product will usually have only one price. Similarity metrics must also handle cases in which there are multiple features to be compared in source and target. The multiple feature version of the metric goes through all of the target features, aligning each to the feature in the source that gives the best score. The scalar metric can also be non-directional. For example, when comparing shirts, it is possible to compare the weight of different fabrics as a scalar quantity, but no direction of preference can be assumed: shirts of different weights are just different. The score becomes the absolute value of $m - (f_t - f_s)$, in all cases.

[7] All FindMe metrics return integer values to facilitate the bucket sort. Larger numbers mean a better match.

[8] Note that this metric depends on the actual numeric difference between the integer features. Not all metrics impose a semantics on the mapping of features to integers, but where scalar metrics such as this one are used, all of the features in a category must be mapped into an integer range.

A feature such as cuisine in restaurants presents a more complex matching problem. The Entree system represents cuisines in a semantic network, and uses marker passing to calculate distances between them. For performance reasons, we opted not to use such networks in the RPS metric set. Instead, there is a table metric, which can represent a network through an adjacency matrix that records the distance from one feature to all other features reachable from it. The matrix can, of course, represent not just semantic networks but any mapping from a feature-pair to an integer.

Retrieval strategies are the arrangement of similarity metrics into priority relationships. Usually the most important metric is obvious: cuisine for restaurants, grape varietal for wines, genre for movies. Selecting and ordering the lower-priority metrics is harder and often requires some experimentation. However, it is easy to get users to respond to questions of the form "Which of Y or Z is the most similar to item X?" Surveys consisting of well-chosen comparisons are very useful in determining what priority to assign to different aspects of an item. Obviously, different individuals may hold goals in different relative priorities. The most obvious example is the goal of not paying too much money for something. In restaurants, we distinguish three strategies: a normal retrieval strategy that puts money second after cuisine, an epicurean strategy that puts money third after cuisine and quality, and a "money no object" strategy that does not consider money at all.

Ultimately, what a FindMe system "knows" about a domain is fairly shallow: different ways that items can be similar to each other, and standard ways of prioritizing these individual assessments into overall strategies. None of the steps required to gather this knowledge is particularly complex, and part of our longer-term research agenda is the creation of tools to automate the majority of the process. At this stage in the development of the technology, the most significant obstacle to the construction of effective FindMe systems is the very practical problem of getting high-quality up-to-date product data.

4. Hybrid recommender systems

Knowledge engineering of the type described above is necessary for building a knowledge-based recommender system. This is the inevitable "price of admission" for a knowledge-based approach, a price that is not incurred by knowledge-weak method such as collaborative filtering or machine learning. However, these weak methods suffer from the ramp-up problem mentioned earlier. The differing strengths of these approaches suggest that they may be complementary rather than competing approaches for the generation of recommendations. A particular benefit of FindMe systems is that they gather preference information without requiring that users make their ratings explicit. Rather than requiring the user to input his or her preferences as a starting point, FindMe systems let the user browse through a catalog using qualitative ratings as navigation aids. Each navigation step informs the system about the user's preferences at a finer grain of detail than a single "buy" decision, and a user is likely to make several such navigation steps (an average of 3 in Entree) while using the system.

Table 1 contrasts the collaborative filtering and knowledge-based approaches, identifying the positive and negative aspects of each. The third row suggests what might be achieved in an ideal hybrid that combines the techniques. Despite the necessary investment in knowledge engineering, such a hybrid could offer good performance even with little or no user data, and the benefits of collaborative filtering as data is collected.

Technique	Pluses	Minuses
Knowledge-based	A. No ramp-up required B. Detailed qualitative preference feedback (in FindMe systems) C. Sensitive to preferences changes	H. Knowledge engineering. I. Suggestion ability is static.
Collaborative filtering	D. Can identify niches precisely. E. Domain knowledge not needed. F. Quality improves over time. G. Personalized recommendations.	J. Quality dependent on large historical data set. K. Subject to statistical anomalies in data. L. Insensitive to preference changes
Ideal Hybrid	A, B, C, D, F, G	H

Table 1: Tradeoffs between knowledge-based and collaborative-filtering recommender systems.

The possible synergy with FindMe systems appears particularly promising, since these systems, through preference-based browsing, permit the collection of detailed user ratings even for rarely purchased items like automobiles or houses.

4.1 Combining recommendation techniques

Within FindMe systems, it is often the case that a retrieval strategy fails to discriminate the returned items completely. The system might be required to arbitrarily select 10 items to return, for example, out of a topmost bucket of size 20. In such a case, we consider the result "under-discriminated." Eliminating under-discriminated results in FindMe systems can be difficult because it requires the addition of one or more new similarity metrics, with attendant knowledge-engineering tasks, or it may require a more in-depth representation of the products. Collaborative filtering, however, can add additional discrimination without requiring knowledge engineering.

Consider a system that, in addition to the FindMe recommender component, has also a collaborative filtering engine, where ratings are obtained by recording each example the user has seen and the user's reaction to it. If a user names an item as a starting point, we can consider that a very high rating, since the user is seeking something similar to what he or she has seen and liked before. The exit point of the system could also be considered a high rating, since the user stops searching, but this is less reliable since the user may have given up without finding anything satisfactory. Each tweak along the way we can consider a negative rating, since the user has found something to dislike. With this technique, we can accumulate ratings (typically many negative and a few positive) for users with only the overhead of logging their FindMe activity.

These ratings can be used to compute correlations between users. The operation of this recommender is likely to be weak if it starts with a small amount of data, so we would not want to present its suggestions directly to users. However, we will not make a bad suggestion if we only examine the items in the topmost under-discriminated bucket. After we have performed the normal FindMe ranking, we look at the user's profile to rate and rank each item in the topmost bucket. There is little risk in applying collaborative filtering here. In the worst case, if the ratings from the collaborative filter are random, we will still be selecting items that are equally similar as far as our knowledge-based system is concerned.

Consider the following example: Alice connects to a version of Entree that includes the collaborative filtering component. She registers as a new user, and starts browsing for Chicago restaurants by entering the name of her favorite restaurant at home, "Greens

Restaurant" in San Francisco. "Greens" is characterized as serving "Californian Vegetarian" cuisine. The top recommendation is "302 West," which serves "Californian Seafood." It turns out that Alice is, in fact, a vegetarian, so she tweaks the system's cuisine choice and moves back towards vegetarian recommendations.

After the system has built up a bigger user base, another new user Ben approaches the system with the same starting point: "Greens." Since the recommendation given to Alice was under-discriminated, her feedback and that of other users allows the system to more fully discriminate Ben's recommendation, and return "Jane's," a vegetarian restaurant, preferring it over "302 West."

This thought experiment suggests that a combination of knowledge-based and collaborative-filtering techniques may produce a recommender system with many of the characteristics of an ideal hybrid. Initial suggestions are good, since there is a knowledge base to rely on. As the system's database of ratings increases, it can move beyond the knowledge base to characterize users more precisely. Because the knowledge base is always present, users are not trapped by their past behavior. If Alice decides to stop being a vegetarian, the system will not make it difficult for her to get recommendations for steakhouses.

To test the validity of this approach, we used data gathered from the Entree system over 3 years of public use, a total of 20,000 interactive sessions. We treat each session as an individual user, since Entree has no way to identify unique returning users. (If individually identified data were available, it would increase the accuracy of collaborative filtering, since we would have fewer users and more ratings per user.) We used 10,000 users for training, and from the remaining 10,000 selected "active" users, those who had rated at least 15 restaurants. There were about 100 of these highly active users. For each active user, we performed five trials selecting four, six or eight examples to provide training data for the user and seven examples for testing, including one restaurant that we know the user likes. The goal is for the system to predict out of the seven test examples the one that the user would rate positively. This task is a reasonable correlate for what we would like the system to actually perform, that is, to select the best recommendation from an unordered bucket.

Figure 9 shows the results for three conditions compared using the precision of the top suggestion: the percent of times that the correct restaurant is selected as the one the user would like. In the random condition, the recommendation is chosen at random from the test set. This is effectively what an unaugmented FindMe system would do. In the average condition, the system chooses the recommendation that has the highest average rating for all users. The collaborative filtering condition (CF) chooses its suggestion by correlating the user's known ratings with those of other users to perform the prediction. Both collaborative filtering and using the average preference always suggest better than chance. Once even a small number of ratings have been collected for a user, collaborative filtering contributes substantially.

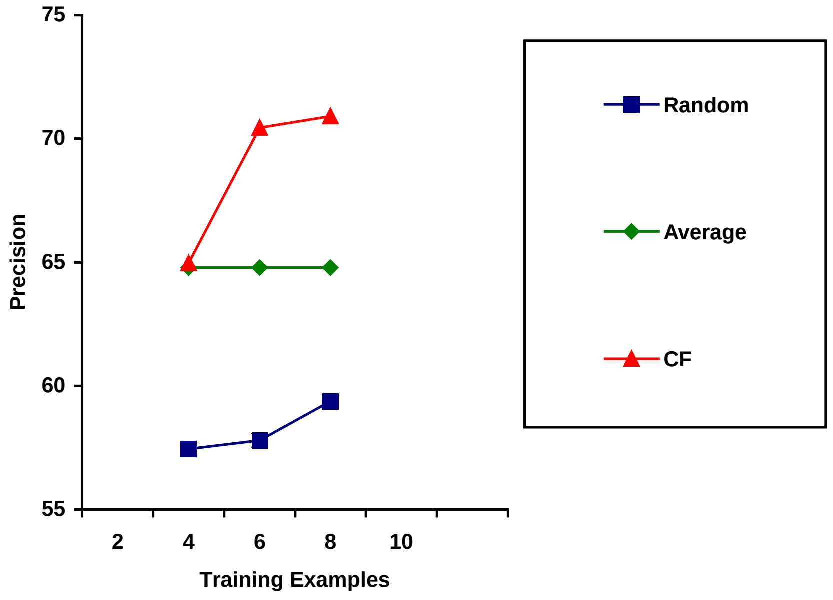


Figure 9: Results of hybrid recommender experiments (precision of top ranked item)

While these findings suggest that the type of hybrid recommender system suggested above will be effective, it does not incorporate all of the data available from navigational actions in FindMe systems. The collaborative filtering technique used in this experiment ignores the difference between user A not liking restaurant X because it is too expensive, and user B not liking restaurant X because its cuisine is too traditional. Ideally, we would like to aggregate users based on their reasons for disliking restaurants, also. The difficulty in using the qualitative tweak data is that it increases the sparseness of an already sparse data set. One of our next research directions for FindMe systems will be to explore ways to manage the increased dimensionality of qualitative ratings. We are particularly interested in the possible application of singular value decomposition (Deerwester, et al. 1990).

5. Related work

Knowledge-based recommender systems have gotten relatively little research attention to date. As discussed earlier, the closest precedent for our use of knowledge-based methods in retrieval comes from case-based reasoning. A case-based reasoning system solves new problems by retrieving old problems likely to have similar solutions. Researchers working on the retrieval of CBR cases have concentrated on developing knowledge-based methods for precise, efficient retrieval of well-represented examples. For some tasks, such as case-based educational systems, where cases serve a variety of purposes, CBR systems have also explored multiple goal-based retrieval strategies like those discussed in this paper (Burke & Kass, 1995).

Our use of tweaks is obviously related to CBR research in case adaptation. Note however, that our use of the term is different. Tweaking in the context of CBR means to adapt a returned case to make it more closely match the problem situation in which it will be applied. The tweaks that a user invokes in FindMe are applied much differently. We cannot invent a new movie or change an existing one to match the user's desires – the best we can do is attempt a new retrieval, keeping the user's preference in mind.

The problem of intelligent assistance for browsing, especially web browsing, is a topic of active interest in the AI community. There are a number of lines of research directed at understanding browsing behavior in users (Konstan, et al. 1997; Perkowitz & Etzioni, 1998), extracting information from pages (Craven, et al. 1998; Knoblock et al. 1998, Cohen, 1998), and automatically locating related information (Lieberman, 1995). Because the web presents an unconstrained domain, these systems must use knowledge-poor methods, typically statistical ones.

In information retrieval research, retrieval is seen as the main task in interacting with an information source, not browsing. The ability to tailor retrieval by obtaining user response to retrieved items has been implemented in some information retrieval systems through retrieval clustering (Cutting, et al., 1992) and through relevance feedback (Salton & McGill, 1983).

Our approach differs from relevance feedback approaches in both explicitness and flexibility. In most relevance feedback approaches, the user selects some retrieved documents as being more relevant than others, but does not have any detailed feedback about the features used in the retrieval process. In FindMe systems, tweaks supply concrete domain-specific feedback. In addition, FindMe systems are not limited to finding items based on similarity alone. The user does not say "Give me more items like this one," the aim of relevance feedback and clustering systems, but instead asks for items that are different from a presented item in some particular way.

Examples have been used as the basis for querying in databases since the development of Query-By-Example (Ullman, 1988). Most full-featured database systems offer the ability to construct queries in the form of a fictitious database record with certain features fixed and others variable. The RABBIT system (Williams, et al. 1982) took this capacity one step further and allowed retrieval by incremental reformulation, letting the user incorporate parts of retrieved items into the query, successively refining it. Like these systems, FindMe uses examples to help the user elaborate their queries, but it is unique in the use of knowledge-based reformulation to redirect search based on specific user goals.

Schneiderman's "dynamic query" systems present another approach to database navigation (Schneiderman, 1994). These systems use two-dimensional graphical maps of a data space in which examples are typically represented by points. Queries are created by moving sliders that correspond to features, and the items retrieved by the query are shown as appropriately-colored points in the space. This technique has been very effective for two-dimensional data such as maps, when the relevant retrieval variables are scalar values. Like RPS, the dynamic query approach has the benefit of letting users discover tradeoffs in the data because users can watch the pattern of the retrieved data change as values are manipulated. Also, as we found in our early Car Navigator experiments, direct manipulation is less effective when there are many features to be manipulated, especially when users may not be aware of the relationships between them.

6. Conclusion

Knowledge-based recommender systems perform a needed function in a world of ever-expanding information resources. Unlike other recommender systems, they do not depend on large bodies of statistical data about particular rated items or particular users. Our experience has shown that the knowledge component of these systems need not be prohibitively large, since we need only enough knowledge to judge items as similar to each other.

Further, knowledge-based recommender systems actually help users explore and thereby understand an information space. Users are an integral part of the knowledge discovery process, elaborating their information needs in the course of interacting with the system. One need only have general knowledge about the set of items and only an informal knowledge of one's needs; the system knows about the tradeoffs, category boundaries, and useful search strategies in the domain.

Knowledge-based recommender systems are strongly complementary to other types of recommender systems. We have shown one way that a hybrid knowledge-based/collaborative system might be successfully constructed, but this is a fertile research area with much room for future experimentation.

Acknowledgements

This article is based in part on “The FindMe Approach to Assisted Browsing” by Robin D. Burke, Kristian J. Hammond, and Benjamin C. Young, which appeared in *IEEE Expert*, 12(4), pp. 32-40, July/August 1997. © 1997 IEEE. The Recommender.com engine was developed by Kristian J. Hammond, Jim Silverstein and the author at Recommender.com with assistance from the National Science Foundation SBIR program under contract #9531395. The FindMe research was supported at the University of Chicago by the Office of Naval Research under grant F49620-88-D-0058.

The interface to the Entree system was designed and created by Robin Hunicke at the University of Chicago. Many others contributed to the FindMe effort at the University of Chicago, including Terrence Asselin, Kai Martin, Kass Schmitt, and Robb Thomas. Daniel Billsus of the University of California, Irvine designed and performed the collaborative filtering experiments reported here. The development of the Recommender.com Personal Shopper system benefited from the insights gained during a JAD workshop with Geneer, Inc. and from the efforts of Jeff Barth, Cate Brady, Drew Scott, Paul Steimle, and Peter Tapling.

References

Bhargava, H. K., Sridhar, S. and Herrick, C. 1999. Beyond Spreadsheets: Tools for Building Decision Support Systems. *IEEE Computer*, 32(3), 31-39.

Billsus, D. & Pazzani, M. 1999. A Hybrid User Model for News Story Classification. In *Proceedings of the Seventh International Conference on User Modeling*. Banff, Canada, June 20-24.

Burke, R, 1999. The Wasabi Personal Shopper: A Case-Based Recommender System. In *Proceedings the 11th Annual Conference on Innovative Applications of Artificial Intelligence*. pp. 844-849. Menlo Park, CA: AAAI Press.

Burke, R., & Kass, A. 1995. Supporting Learning through Active Retrieval of Video Stories. *Journal of Expert Systems with Applications*, 9(5), 361-378.

Burke, R., Hammond, K. & Cooper, E. 1996. Knowledge-based navigation of complex information spaces. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pp. 462-468. Menlo Park, CA: AAAI Press.

Burke, R., Hammond, K., and Young, B. 1997. The FindMe Approach to Assisted Browsing. *IEEE Expert*, 12(4): 32-40.

Cohen, W. W. 1998. A Web-based Information System that Reasons with Structured Collections of Text. In *Proceedings of the Second International Conference on Autonomous Agents*, pp. 400-407. New York: ACM Press.

Craven, M., DiPasquo, D., Freitag, D. , McCallum, A., Mitchell, T., Nigam, K. & Slattery, S. 1998. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 509-516. Menlo Park, CA: AAAI Press.

Cutting, D. R.; Pederson, J. O.; Karger, D.; and Tukey, J. W. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM/SIGIR Conference*, pp. 318-329. New York: ACM Press.

Deerwester, D., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. 1990. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6), 391-407.

Goldberg, D., Nichols, D., Oki, B. M., Terry, D. 1992. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12), 61-70.

Hammond, K. 1989. *Case-based Planning: Viewing Planning as a Memory Task*. Boston, MA: Academic Press.

Knoblock C. A., Minton, S., Ambite, J. L., Ashish, N., Modi, P. J., Muslea, I., Philpot, A. G., and Tejada, S. 1998. Modeling Web Sources for Information Integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 211-218. Menlo Park, CA: AAAI Press.

Kolodner, J. 1993. *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.

Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. 1997. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40(3), 77-87.

Maes, P., Guttman, R. H., and Moukas, A. G. 1999. Agents that buy and sell. *Communications of the ACM*, 42(3), 81-91.

Perkowitz, M. & Etzioni, O. 1998. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 727-731. Menlo Park, CA: AAAI Press.

Resnick, P. and Varian, H. R. 1997. Recommender systems. *Communications of the ACM*, 40(3), 56-58.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. GroupLens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the conference on Computer supported cooperative work*, pp. 175-186. New York: ACM Press.

Riesbeck, C., & Schank, R. C. 1989. *Inside Case-Based Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.

Salton, G., & McGill, M. 1983. *Introduction to modern information retrieval*. New York: McGraw-Hill.

Schank, R.C., & Riesbeck, C. 1981. *Inside Computer Understanding: Five Programs with Miniatures*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Schneiderman, B. 1994. Dynamic Queries for Visual Information Seeking. *IEEE Software*, 11(6), 70-77.

Shardanand, U. and Maes, P. 1995. Social information filtering algorithms for automating "word of mouth" In *CHI-95: Conference proceedings on Human factors in computing systems*, pp. 210-217. New York: ACM Press.

Shimazu, H., Kitano, H. & Shibata, A. 1993. Retrieving Cases from Relational Data-Bases: Another Stride Towards Corporate-Wide Case-Base Systems. In *Proceedings of the 1993 International Joint Conference on Artificial Intelligence*, pp. 909-914. New York: Morgan Kaufmann.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems Vol 1*. Computer Science Press, 1988.

Wiener, T. 1993. *The Book of Video Lists*. Kansas City: Andrews & McMeel.

Williams, M. D., Tou, F. N., Fikes, R. E., Henderson, T., & Malone, T. 1982. RABBIT: Cognitive, Science in Interface Design. In *Fourth Annual Conference of the Cognitive Science Society*, pp. 82-85. Ann Arbor, MI.