KARC: Radio Research

Florian Pestoni
IBM Almaden Research Center
fpestoni@almaden.ibm.com

Joel L. Wolf
IBM T.J. Watson Research Center
jlwolf@us.ibm.com

Md Ahsan Habib
Purdue University
habib@cs.purdue.edu

Amy Mueller
Massachusetts Institute of Technology
amym@mit.edu

Abstract

In the emerging markets for digital content distribution, traditional models for content programming still predominate. But there is a large potential for innovation and improvement in content selection to better meet consumers' demand for entertainment and news. We have developed a new paradigm – and supporting technology – to customize audience content to their specific preferences.

One can identify several selection models for groups of individuals with similar interests in music, video, or other multimedia content to jointly customize a distribution channel. Our approach represents a balance between the two most widespread models available today, namely broadcasting (in which a small group of people select content for a large, passive audience), and individual playback such as CD/DVD players (with its burden on the listener for content selection).

Using technologies such as data mining, multicasting and smart players, our model gives listeners access to automatic shared playlists. This kind of customized narrowcasting is especially applicable to distribution of content for which there is high demand for repeat listening or viewing, while at the same time being very idiosyncratic. Our approach can provide significant advantages to consumers, distribution channels, content owners and advertisers alike.

In this paper we present the basic collaborative content programming algorithms and describe initial experiences with this new paradigm. Specifically, we describe KARC, a prototype of an Internet multichannel virtual radio station environment deployed at the IBM Almaden Research Center.

1 Introduction

Decisions about the choice of programming for content delivery channels such as radio and TV involve many factors. These typically depend heavily on the business model used to fund the channels. For instance, commerical radio or TV stations are funded via commercials and therefore want to attract as many listeners or viewers in their target audience as possible. This is because advertising budget allocations naturally depend on the size of the station's audience. Premium cable and pay-per-view channels want similarly to attract a large audience, but for a different reason. In this case the audience is a direct source of revenue.

In both cases, it is in the stations' best interest to provide the programming that its audience values most. However, in traditional media there is limited information regarding audience preferences. Ratings and record sales are both after-the-fact measurements of such preferences. By-request programs, while allowing individual listeners and viewers to submit their selections, are usually time consuming and labor intensive for consumers.

1.1 Selection Models

The widespread availability of the Internet is likely to have a profound impact on the delivery of entertainment. Assuming that listeners have access to the Internet, preferably from the same devices used to render the entertainment content, we have developed a feedback mechanism that enables them to jointly decide on what content is delivered to them. (For concreteness we will henceforth focus on radio listeners in this paper, but our comments would apply equally well to TV viewers or a variety of other customer types.) We call this process *collaborative content programming.*

One can readily identify several models, each of which shares some key conceptual characteristics and infrastructure. In the simplest model, listeners share a distribution channel and make requests for content to be delivered. Requests are processed in a FIFO queue and the content is streamed accordingly to the interested parties. We call this the *jukebox* model, by analogy with the coin-operated machines that were popular in the 1950's. This model applies best to content of limited duration (such as a song).

The jukebox model can be thought of as an intermediate concept between traditional programming, in which a small

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

group of people select entertainment content for a large audience, and purely individualized programming in which the end user is responsible for choosing the content. This model allows listeners with similar interests to share the burden of selecting content while at the same time having a direct influence on their experience. It allows them to collectively customize the channel.

A possible implementation would allow the audience to contribute their own content rather than selecting from a predefined list of possible titles. This has many usability and technical advantages ranging from reduced requirements for centralized storage of content to availability of hard-to-find titles in special-interest groups. However, this also raises several issues related to copyright protection, which will be addressed in a later section.

The description above implicitly assumes that enough channels are available for each group with a shared preference. Traditional channels frequently adhere to broad categories, such as rhythm & blues music or folk music; similarly, pre-specified channels for collective content programming may help listeners self-select the group in which they participate. However, in the latter case it is the viewers who select the actual content within the limits of the category. Both mechanisms could share the same downstream delivery mechanism, essentially broadcasting.

Newer streaming technologies and smarter devices allow for significantly more flexibility and personalization than the implementation just described. For instance, IP multicasting can be used on the Internet to stream data on demand to a limited number of listeners. However, these applications typically require significant bandwidth.

An interesting alternative is what has been called *super-radio* mode, in which large amounts of content, such as songs, are transmitted simultaneously on different channels, and a smart receiver retrieves and renders those titles that match the listener's preferences. This is usually combined with some limited buffering to allow for the scheduling of overlapping matches and compensate for times when no titles match the profile.

Regardless of the distribution technology being used, the key concept is that viewers themselves select the actual content. In a more advanced selection model, listeners *subscribe* to one or more *virtual channels*. As content is being presented to them, they can express their opinion by voting for or against it. This makes it possible to receive additional feedback about what listeners in a certain group like or dislike. Over time, as votes accumulate, the channel *learns* the collective preferences of its subscribers, and can use this information to automatically deliver the content that best matches their preferences. We call this *automatic shared playlists.*

Using data mining techniques, patterns are extracted from the selection/voting stream to identify not only a ranking of favorite titles but also temporal relationships (e.g., time of day when certain songs are played or the sequence in which titles are played,) and cross-correlation between groups to proactively *recommend* other selections.

The key goal is to achieve this functionality with a very simple user interface and with modest training requirements. We have developed some prototypes using computer systems embedded in "non-threatening" appliance-like devices, as well as computer simulations of planned physical devices. In order to provide additional value to consumers and attract them to this technology, we are exploring alternative user interfaces such as speech recognition and handheld devices.

1.2 Collaborative Content Programming

Unlike most recommendation systems, automated shared playlists are not passive. A station with a collection of channels is constantly making decisions about what content to feed to each channel. The goal is to take into account a variety of dynamic factors beyond the pure preferences of listeners, including the recent history of played titles, the identities of the listeners currently connected to a channel, availability of content and time of day.

At the heart of collaborative content programming are tried and true collaborative filtering techniques. These algorithms, described in more detail below, take listener's votes and make predictions of what rating a listener would assign to content that she has not rated yet.

We have adopted a multi-dimensional approach that uses a vector of features to identify each piece of content. For music, this vector can include metadata (title, author, album and year it was originally published in) as well as features of the content itself (key, beat, lyrics.) We purposefully avoid arbitrary classifications such as genre, as these tend to be generalizations that not all listeners would agree on. We believe it is better to allow these classifications to be created dynamically by listeners' preferences. This allows us to capture idiosyncratic preferences, such as fans of Van Halen who only listen to songs in which David Lee Roth is the lead singer, and would never accept his replacement by Sammy Hagar.

A cluster of listeners is mapped to a channel. This is a dynamic notion of a cluster, since as listeners' preferences and other conditions change, clusters would be defined differently: a vote by one listener on a song may change the clustering decision affecting several others. Assignment of a listener to a cluster is based on a measure of distance between profiles, implicit in the collaborative filtering algorithms.

Our system introduces a level of randomization to add variety to the sequences. Additionally, it checks the history of the most recent songs played on that channel to avoid excessive repetitions. Listeners can contribute their own content to the station. This makes the system an open one, inasmuch it can grow to adapt to the introduction of new content and to the changing tastes of the audience. A listener introducing new content implicitly counts as a positive vote. The system

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

may then determine the probability that other listeners may like this content and introduces it in the automatic shared playlists for these listeners, who may be assigned to different channels. In turn, these listeners may choose to express whether they like or dislike the new title. In this way, new content propagates through the system, but only on channels where it is likely to be appreciated.

In order to validate the usefulness of the proposed selection model, we built a prototype Internet radio station and deployed it at the IBM Almaden Research Center. We called this radio station *KARC*, following the familiar pattern for radio station call names in the western United States.

Section 2 contains an mathematical description of the KARC algorithms. Implementation issues are discussed in Section 3. Section 4 describes user experiences in Almaden. Section 5 contains conclusions and a list of future work.

2 Descriptions of the KARC Algorithms

In this section we shall describe the algorithms we designed to play appealing and personalized songs for the KARC clients, to choose songs for listeners to rate which *train* the algorithm in a relatively optimal manner, to assign these listeners to appropriate radio stations (*channels*) as they sign onto KARC, and finally to schedule the playlists for those channels appropriately. Our algorithms involve techniques in the area of rating-based *collaborative filtering* and *clustering*.

We begin by introducing a little notation. Suppose that there are N KARC clients (or listeners) and there are K songs which can be rated. In general the number of songs rated by listener i will be *vastly* less than K. Typical rating-based collaborative filtering algorithms require the listener to rate a minimum number on the order of 25 songs, while K might be tens of thousands. Define a 0/1 matrix M by setting $M_{i,j}$ equal to 1 if listener i has rated song j, and 0 otherwise. Define the set $R_i = \{1 \leq j \leq K | M_{i,j} = 1\}$, in other words the songs rated by listener i. Thus $card(R_i)$ is the row sum for the ith row, while the column sum for column j represents the number of listeners who have rated song j. For any matrix element $M_{i,j} = 1$ we denote by $r_{i,j}$ the actual rating of song j by listener i. We will assume that ratings are always integers on a scale from 1 to some fixed value v. Commonly used scales include $v = 7$ and $v = 13$. KARC employs a fairly conservative value $v = 3$. So the choices correspond loosely to *like, neutral* and *dislike*, respectively.

2.1 The Prediction Algorithm

The goal of the basic atomic query is to accurately predict the rating of song j for listener i. By making repeated calls to this query, KARC can generate a playlist of songs on a channel that the current listeners to that channel will probably like: It simply chooses a list of those songs not recently played which have the highest average predicted ratings. One can easily add a constraint specifying the minimum acceptable predicted rating of a song for the individual listeners.

A number of collaborative filtering algorithms have appeared in the literature, among them [5] and [7]. While we believe the *mentor*-based techniques described in these papers are sufficiently fast and scalable, it would appear that these algorithms require more listener ratings than absolutely necessary to make accurate predictions. Note that making accurate ratings predictions in the presence of sparse data is by no means a trivial issue: Listeners are *not* willing to invest much time or effort in rating the songs, and this is unquestionably a major drawback of rating-based collaborative filtering.

Our approach to collaborative filtering is fundamentally different from the above two algorithms, based instead on a graph theoretic model originally described in [2]. The idea is to form and maintain a graph whose nodes are the listeners and whose edges correspond to a notion called *predictability*. This predictability is strong in the sense that relatively more common songs are required to be rated in common by pairs of listeners. It is generic in the sense that it accommodates not only pairs of listeners whose ratings are genuinely close, but also listener pairs whose ratings are similar except that one listener is more effusive with his or her ratings than the other listener. It also accommodates listener pairs who rate songs more or less oppositely, as well as combinations of these scenarios. Whenever one listener predicts another listener, a linear transformation is constructed which *translates* real ratings from one listener to predicted ratings for the other. The inverse of that transformation translates ratings back the other way. The ultimate idea is that predicted rating of song j for listener i can be computed as the average computed rating via a few reasonably short *paths* joining multiple listeners. Each of these paths will connect listener i at one end with another listener k who has rated song j at the other end. No other listeners along the path will have rated song j. However, each arc in the graph will have the property that there is some real rating predictability from one listener to the other. The overall rating for this path is computed by starting with the rating $r_{k,j}$ and translating it via the composite of the various transformations corresponding to the path.

More specifically, our approach to generating good quality collaborative filtering predictions involves a pair of concepts which we will call *cohorting* and *predictability*. We have already described the notion of predictability at a high level. We say that listener i_1 *cohorts* listener i_2 provided $card(R_{i_1} \cap R_{i_2}) \geq G$, where G is some predetermined threshold. Notice that *cohorting* is symmetric but not transitive: That is, if listener i_1 cohorts listener i_2 then listener i_2 also cohorts listener i_1. (Because of this we are able to call listeners i_1 and i_2 *cohorts*, a more natural use of the word.)

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

However, if listener i_1 cohorts listener i_2 and listener i_2 cohorts listener i_3 it does *not* follow that listener i_1 cohorts listener i_3. The intent here is that if listeners i_1 and listener i_2 are cohorts then there is enough commonality among the jointly rated songs (from, say, listener i_1's perspective) to *decide* if listener i_2 predicts listener i_1 in some fashion or not. In [2] we introduced this basic approach to collaborative filtering, but we have refined it here in several significant ways.

We now make the notion of predictability more precise. For a given pair $s \in \{-1, +1\}$ and $t \in \{t_{s,0}, ..., t_{s,1}\}$, there is a natural linear transformation of ratings $T_{s,t}$ defined by $T_{s,t}(r) = sr + t$. (Here, $t_{-1,0} = 2$, $t_{-1,1} = 2v$ are chosen so that $T_{-1,t}$ keeps at least one value within $\{1, ...v\}$ within that range. The value of t will typically be close to the average of 2 and $2v$, namely $v + 1$. Similarly, $t_{1,0} = 1 - v$, $t_{1,1} = v - 1$ are chosen so that $T_{1,t}$ keeps at least one value within $\{1, ..., v\}$ within that range. The value of t will typically be close to the average of $1 - v$ and $v - 1$, namely 0.) We say that listener i_2 *predicts* listener i_1 provided i_1 and i_2 are cohorts, and provided there exist $s \in \{-1, +1\}$ and $t \in \{-2v + 1, ..., 2v - 1\}$ such that the distance function $\mathcal{D}_{s,t}(i_1, i_2) = [\sum_{j \in R_{i_1} \cap R_{i_2}} |r_{i_1,j} - T_{s,t}(r_{i_2,j})|]/card(R_{i_1} \cap R_{i_2}) < U$, where U is some fixed threshold. The term $\mathcal{D}_{s,t}(i_1, i_2)$ is simply the *manhattan* or L_1 distance between the ratings of listener i_1 and the transformed ratings of listener i_2 on the set of songs they both rate, normalized by the number of those songs. We call this the *manhattan segmental* distance. Note that if $s = 1$ and $t = 0$, listener i_2 behaves more or less like listener i_1. (This would correspond to our interpretation of closeness, similar to that in [7].) In the idealized situation that $\mathcal{D}_{1,0}(i_1, i_2) = 0$, listener i_2 behaves *exactly* like listener i_1. If $\mathcal{D}_{1,1}(i_1, i_2) = 0$, listener i_1 is simply more effusive with praise (by one rating unit) than listener i_2. On the other hand, if $\mathcal{D}_{1,-1}(i_1, i_2) = 0$, listener i_2 is more effusive than listener i_1. Finally, if $\mathcal{D}_{-1,v+1}(i_1, i_2) = 0$, listener i_2 behaves in a manner precisely opposite to listener i_1. Nevertheless, listener i_2 predicts listener i_1 perfectly.

It should be clear from this discussion that (1) predictability is a more general notion than mere closeness, but (2) the cohort requirement in the definition imposes a relatively rigid requirement that the predictability be based on sufficient data to be legitimate. As with cohorting, predictability is symmetric but not transitive.

If listener i_2 predicts listener i_1, we define $s\star$ and $t\star$ to be the values of s and t, respectively, which minimize $\mathcal{D}_{s,t}(i_1, i_2)$. (There are only $4v - 2$ possible pairs (s, t), so this optimization is performed quickly via exhaustive search.) The pair $(s\star, t\star)$ will be called the *predictor* values. We will use the linear transformation $T_{s\star,t\star}$ to translate ratings from listener i_2 to listener i_1. Note again that listener i_1 predicts listener i_2 via the inverse of this transformation.

Let H_i denote the set of listeners who are cohorted by listener i. Let P_i denote the set of listeners who predict listener i. Thus $P_i \subseteq H_i$. In order to implement our prediction process we will need to employ three distinct data structures. These are as follows:

- For each song j we will maintain an *inverted index* of all the listeners who rate song j. For details on inverted indexes, see [4]. We will insist that the inverted indexes be sorted in increasing order of listeners i. (By this we mean that we will equate the index of listener i with a unique identification given to that listener, and sort in terms of that id.) We store the actual ratings in this inverted index as well.

- We will also maintain a *graph* in which the nodes are the listeners and there is an arc between listener i_1 and listener i_2 provided listeners i_1 and i_2 predict each other. We will store the corresponding predictor values in the graph, as well as the songs rated by the various listeners and the ratings themselves. We will call these two graphs the *cohort* and *predictability* graphs, respectively. Clearly the latter is a subgraph of the former.

- Finally, we maintain an *update list* consisting of those new and modified ratings of songs for the various listeners which have not yet been incorporated into the inverted indexes and the graph. This is because that update operation is modestly labor intensive, and we do not wish to always perform it in real time. (The timing of this update is under the control of a separate scheduler module. It is typically performed in the middle of the night.) When the update operation does occur we simultaneously flush the list.

Now, consider the atomic rating query: We wish to predict the rating of song j for listener i. First we check the inverted index and the update list to determine if listener i has already rated song j. What happens next depends on whether or not listener i has any (other) songs remaining in the update list. Equivalently, the question is whether or not listener i has recently added to or modified his or her ratings.

If so, we wish to incorporate these ratings into the prediction process, but we are willing to employ a slightly stale view of the other listeners by ignoring the remaining songs in the update list. The prediction algorithm now proceeds in three stages: First, the inverted indexes for all songs in the revised R_i are examined via a merge and count operation, so that the set H_i of listeners who are cohorts of listener i can be quickly computed. Notice that success and/or failure of the cohorting question can often be determined without checking all the inverted indexes. Thus, since the inverted indexes are arranged in increasing order of size, the largest inverted indexes will frequently not need checking.

Second, those members of H_i which are also in P_i are found via the appropriate calculations. Third, a shortest path in the predictability graph from any listener in P_i to a listener who rates song j is computed.

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

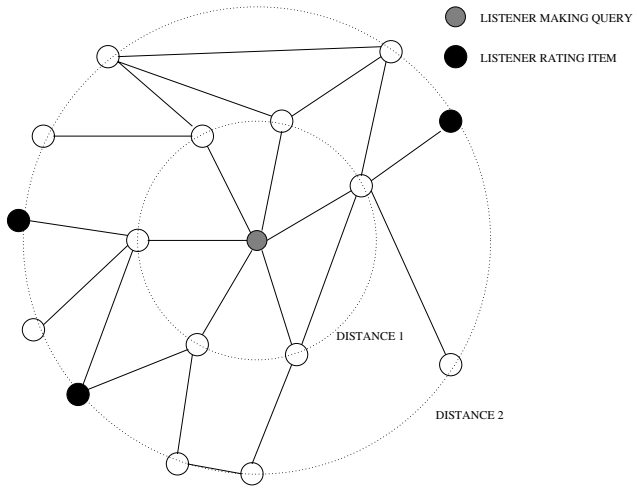

Figure 1: Using the Predictability Graph

See Figure 1 to visualize the shortest path process. In this figure the arcs correspond to predictability. The listener "making the query" is in the center of the illustrated portion of the predictability graph, listeners who are a distance 1 away are are shown on the inner circle, and those who are a distance 2 away are shown on the outer circle. At distance 2 there are 3 listeners who have rated the song in the query, and there are 4 paths from the query listener to these 3 listeners.

The shortest path algorithm can be implemented via breadth first search. If the overall collaborative filtering process is going to be effective, the length of such a path will typically be small, and can be found effectively by the breadth first search algorithm. In fact, if a path is not found within some very small distance D (KARC uses $D = 4$), the algorithm terminates with the failure: Song j cannot be rated for listener i. For details on shortest path algorithms see, for example, [3]. This path allows a rating computation based on the composition of transformations of a predicted value for song j by listener i. So, for instance, given a path $i \rightarrow i_1 \rightarrow ... \rightarrow i_l$, with predictor values $(s_1\star, t_1\star), ...(s_l\star, t_l\star)$, the predicted value for the rating of song j will be $T_{s_1\star,t_1\star} \circ ... \circ T_{s_l\star,t_l\star}(r_{i_l,j})$. In our algorithm the average of these predicted values computed from all such paths of minimal length is taken as the final predicted rating. Thus in Figure 1 the ultimate rating of the song would be averaged over 4 predictions across paths of length 2.

If listener i has not changed or added to his ratings since the update list was last flushed, then the first two steps are not necessary. Only the third step need be performed.

Note that because we are willing to accept slightly stale data associated with listeners other than listener i, only those arcs emanating from that listener need be computed on the fly. The rest are prestored and do not change during the prediction process. (Allowing this staleness seems to us to be in the proper spirit of the algorithm, since we are keeping listener i's ratings fully current and all other listener's ratings consistent from a recent point in time and *nearly* current. But an implementation not willing to accept this would simply keep the update process running in real time, thus maintaining current versions of the inverted indexes and the graph.)

2.2 The Training Algorithm

As stated thus far one might suspect that the cohort and predictability graphs might be hopelessly sparse, perhaps containing many disconnected components. Fortunately we have some control over the training songs we present to the KARC listener. In this subsection we describe techniques for presenting the listener with those songs which the system would most like to have rated. The goal is, of course, to reduce the collaborative filtering learning curve time, to increase the frequency of making successful prediction queries, and finally to improve the accuracy of those queries.

Fundamentally the approach is to partition the collection of songs into a small *hot* set and a large *cold* set. In KARC we choose a hot set of songs with a cardinality on the order of the square root of the total number of songs. The remaining songs make up the cold set. In a 10,000 song library, for example, there might be 100 hot songs and 9,900 cold songs. Songs in the hot set are chosen based on their popularity, and they are presented to many listeners in order to increase *commonality* of rated songs. This has the obvious effect of increasing the number of cohorts and thus simultaneously the number of listeners who predict each other. Ultimately it can have the effect of decreasing the average distance amongst listeners in the predictability graph, which is goodness for our algorithm. Songs in the cold set are presented to the listeners in order to increase *coverage* of the remaining rated songs. The goal is to increase the number of songs which can then be "reached" via a short enough path from the query listener.

An actual listener training session proceeds in two phases, one for the hot set and one for the cold.

In the first phase songs from the hot set are played for the listener until the listener node becomes connected to other listeners in the predictability graph. Though the selection of these songs starts off randomly, it quickly becomes less so: Based on the actual ratings of the listener at any point, it is an easy calculation to compute which other listeners have the currently best predictability coefficients, and the listener is then offered hot songs rated by these listeners. Note that the manhattan segmental distance $\mathcal{D}_{s,t}(i_1, i_2)$ defined earlier makes sense as long as $card(R_{i_1} \cap R_{i_2}) \neq \emptyset$, and so for a training listener i_1 we can easily compute the minimum of $\mathcal{D}_{s,t}(i_1, i_2)$ over all other listeners i_2 and all parameter pairs s and t. (A listener may refuse to rate any song at any time – the next most appropriate song is then offered.) Finally the listener becomes connected to the predictability graph, and the goal changes subtly. At this point we compute, for a given candidate hot set song yet to be rated by the listener, the

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

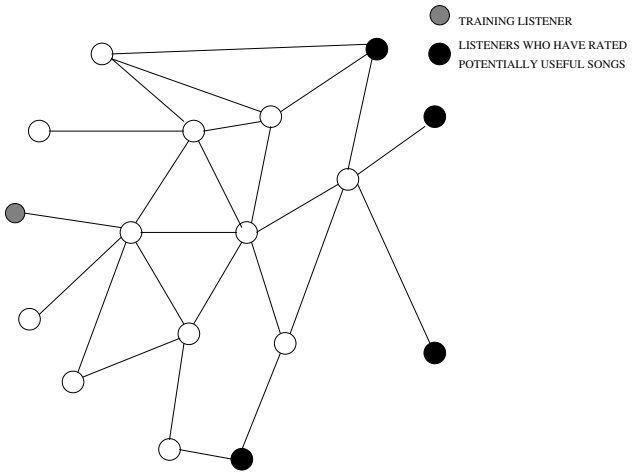

Figure 2: Finding Potentially Useful Training Songs

probability, based on uniformity assumptions, that rating the song will add or remove a given arc from the listener in the predictability graph. Since the objective is now to minimize the expected average distance between all pairs of listener nodes in this random graph, an easy calculation. For details on random graphs see [6]; for details on all pair shortest path algorithms see [3]. The song offered is chosen accordingly.

See Figure 2 for a simplified example of this training algorithm in action. The listener being trained is shown on the left. There are 3 listeners whose distance from the training listener is 4, and hot songs rated by these listeners are likely to be good candidates on which to train the listener.

In the second phase songs from the cold set are chosen for purposes of coverage. Thus we try to find cold set songs which have *not* been rated by any listeners within a distance D of the training listener. These cannot cause the predictability graph to disconnect, but could result in a successful prediction not possible before. In the event that all cold set songs have been rated by a nearby listener the more general goal of finding that cold set song with the least ratings is adopted.

2.3 The Channel Assignment and Scheduling Algorithms

As listeners sign on they must be assigned to a channel. Listeners who are still in the training stage are assigned to their own channel. This allows KARC to play training songs chosen exclusively for them. When assigning listeners past their training stage, KARC tries to cluster them with other listeners with whom they agree most. The manhattan segmental distance $\mathcal{D}_{1,0}(i_1, i_2)$ effectively captures the closeness between a new listener i_1 and another, previously assigned listener i_2, and the goal is to assign listener i_1 to a channel where the distance to other listeners is minimized. This therefore becomes an on-line *projected clustering* problem, where the clusters correspond to the channel assignments. KARC adopts the philosophy that a listener assigned to a channel can be reassigned during the time he or she is active, in order to maintain some sort of balance amongst the cluster sizes. So it employs an online clustering algorithm as described in [1].

Aside from training songs, which are handled on private channels, KARC has to schedule two types of songs. There are listener requests and system-generated songs. These must both be scheduled on the same channels. KARC schedules listener-requested songs with higher priority than system-generated songs. If there are no listener requests pending the channel schedules a song chosen based on its predicted popularity, as above. Otherwise it will empty its playlist of listener requests first, and these are scheduled in round-robin fashion for the multiple listeners.

3 Implementation Issues

The KARC prototype supports IP multicasting to make efficient use of bandwidth. Rather than opening an individual stream between the server and each client computer, a single stream for each channel is broadcast on the subnet. Routers that support multicasting make it possible for the multicast channels to be propagated to other subnets, but only if there is at least one subscriber interested in this content.

The system is composed of three components: a streaming server, a front end, and a recommendation engine. We used off-the-shelf products to provide the streaming functionality. On the server side, we used Icecast [8], an open source audio streaming server. We were able to use Icecast, and its companion software Shout (an mp3 source client) without any changes. On the client side, we support a number of players, including RealPlayer Plugin, Winamp and XMMS.

The front end is based on a Java framework developed in Almaden for embedded applications. This framework includes a minimalistic http server and uses servlets to serve dynamically-generated html pages.

In our object-oriented design, listeners are dynamically assigned to Channels upon connection. Each listener may have multiple Personas. Each Persona must go through a RegistrationProcess before connecting to a regular Channel. A TrainingChannel extends Channel and is a point-to-point stream between the server and a client. This is done to allow a single listener to listen to a (portion of a) Song and proceed to the next one in the Queue as soon as she casts her Vote. A Vote corresponds to a Rating, with values 1,2,3 to represent dislike, neutral, like.

Each Channel Thread makes sure that the Queue always contains a minimum number of Songs by requesting recommendations from the Engine. The Queue sends Songs to the Player, which in turn interfaces with the streaming server. When the Player is done with a Song, it requests the next from the Queue. The Queue prefetches

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

Songs from their location on the network ahead of time, storing them in a local Cache.

Listeners connect to the server via their web browsers, and must have one of the supported players installed. (See Figure 3 for a screenshot.) A VotingApplet, loaded from the server, communicates back to it to send and receive Events, such as a Vote by the current Persona or a change in the current Song. We use the TSpaces middleware [9] for this purpose. Additionally, a ChatApplet developed as part of the WBI project [10] allows listeners connected to a Channel to exchange short messages. An avatar represents each listener graphically in a virtual room, which provides a "meeting place" for people with like tastes in music.

The recommendation engine presents a well-defined API to the front end. Some of the classes developed for the front end (e.g. Channels, Listeners) were reused in the implementation of the recommendation engine. By loosely coupling these two components, we may replace the Engine without changing the front end. The main class in this component is the Oracle, which implements the algorithms described above. The Oracle can answer queries such as *"What is the best channel for listener Amy?", "What is the best song to play in channel 5?"* and *"What songs should I have Joel vote for during registration?"* Additionally, the Oracle can find the likely Rating a listener would give to a Song that has not been rated yet, or the set of listeners that more closely match another listener's preferences.

The Oracle uses knowledge about listeners' votes, but also other information such as the history of recently played songs and the set of listeners currently connected to each channel. We are currently working on a version that considers the metadata associated with each title.

4 Listener experience at IBM Almaden Research Center

KARC was made available to researchers at the Almaden Research Center. The goal was to determine the viability of the concept and to discover listeners' experiences.

One issue that we faced was that of using copyrighted material. We are currently engaged in several copy-protection research efforts geared towards preserving copyrights in an electronic distribution system. In the meantime we we wanted to restrict any possibility of abuse, even behind our corporate firewalls. For this reason, we had to restrict the number of participants and the availability of content. We wanted listeners to interact with the systems in conditions as close as possible to what they might be in the real world. Therefore, we avoided targeted experiments and decided to measure the performance of system in indirect ways.

We focused on usability and effectiveness of the channel assignment and prediction algorithms. The former was measured in qualitative ways, and was used as feedback for our user-interface design. For instance, listeners who were given no instructions were able to use the voting applet with essentially the same ease as those listeners who were coached or referred to written instructions. (See Figure 4 for a screenshot of the GUI used for registration.) However, some listeners preferred different labels for the buttons for positive, negative and neutral votes than those provided in our prototype.

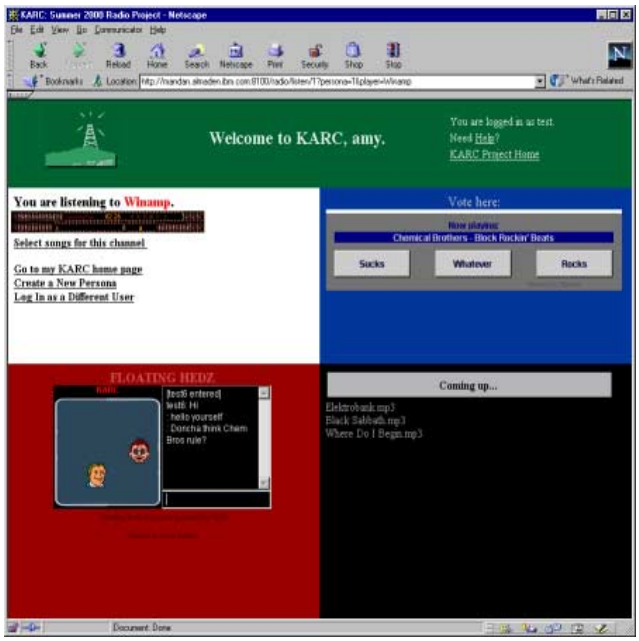

Figure 3: Graphical User Interface for a Regular Channel

Most listeners found the initial transition from training to listening confusing. During training, as soon as the listener casts a vote, the system proceeds to the next song. But in normal listening mode, votes do not affect the playback of the *current* song. While this made sense to all listeners eventually, in many cases it required some explanation. We added specific instructions, which are only shown to first-time listeners when they finished their initial training session.

Overall, the user interface was evaluated as being quite good, but somewhat complex for first-time listeners. Because our sample population was drawn out of a group of scientists, it is very likely not to be representative of a more general listener base. We found that users were quite active in terms of listening, voting and contributing songs when they first signed up. The registration process was well received by all listeners, and several expressed their preference for this approach over being presented with a list of titles to vote from.

Most listeners cast several votes during their first listening experience after registration, but the number of votes decreased over time. This can be attributed to some extent to the novelty of the system during the first few days. Another reason could be that the channel assignment was not perfect

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE

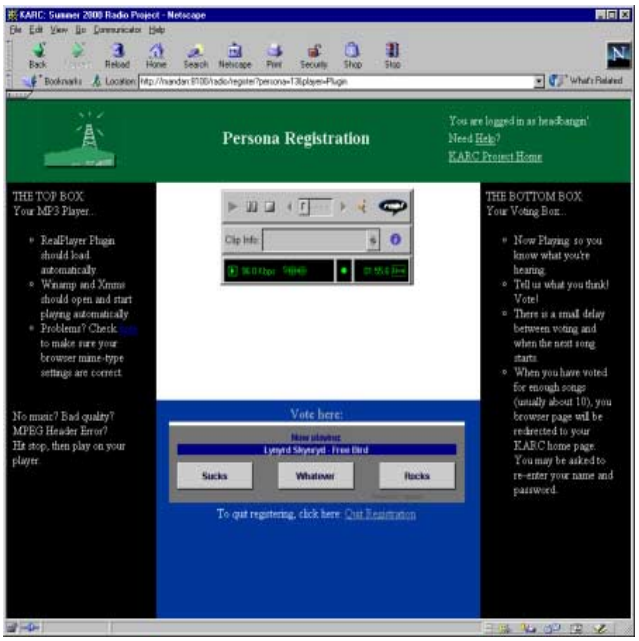

Figure 4: Graphical User Interface for Training

and that listeners had a strong initial incentive to vote to fine tune the recommendations; as the system was trained, this incentive decreased. Listeners were also quite active in contributing new songs to the system, with a temporal pattern similar to that of votes.

Every time a listener rating was received we compared it to the recommendation engine's estimate. We kept a running average of the distance between these, observing values ranging from an excellent 0.2 to a poor 1.3. But this data data should be considered anecdotal. A more rigorous study is needed, with a much larger population.

5 Conclusions and Future Work

We have shown that the notion of collaborative filtering can be extended from passive recommendations to active content programming. The system can manage multiple channels simultaneously, intelligently allocating listeners to channels based on the preferences of the whole community of listeners. Besides optimizing for bandwidth, this provides a simple mechanism for the introduction of new content. In addition, it stresses the community aspects of listening online, attempting to transfer some of the experience from the off-line world.

Our KARC prototype provides a platform for conducting our Radio Research. The current system can be easily fine-tuned to adapt to different bandwidth constraints. Moreover, improvements in the recommendation engine can be seamlessly incorporated as they are made available.

We expect to conduct larger scale experiments within the IBM Research community (at least initially) after we incorporate copy protection schemes. This may require improving or complementing the networking infrastructure, for example by using repeaters in distant locations. These large scale experiments will also allow statistically significant listener surveys and performance measurements.

In terms of user interfaces, we expect to allow listeners to customize their visual experience by defining what elements are displayed, changing labels and the look and feel of the voting applet, and defining their own avatars for the virtual chat room. We have started work on a speech interface, which will allow visually impaired listeners to participate actively.

6 Acknowledgments

The authors would like acknowledge the contributions of Steve Farrell for his Floating Hedz chat applet, John Thomas for debugging TSpaces, Eustus D. Nelson for his proposal of competing DJ algorithms, Dalit Naor for helping meet the deadline, and especially all beta testers at Almaden who contributed their time, effort and suggestions.

References

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu and J. Park, "Fast Algorithms for Projected Clustering", *Proceedings of ACM SIGMOD Conference*, Philadelphia PA, pp. 61-72, 1999.

[2] C. Aggarwal, J. Wolf, K.-L. Wu and P. Yu, "Horting Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering", *Proceedings of the ACM KDD Conference*, San Diego CA, 1998.

[3] T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge MA, 1992.

[4] C. Faloutsos, Access Methods for Text, *ACM Computing Surveys*, Vol. 17, pp. 50-74, 1985.

[5] D. Greening, "Building Consumer Trust with Accurate Product Recommendations", LikeMinds White Paper LMWSWP-210-6966, 1997.

[6] E. Palmer, *Graphical Evolution*, Wiley-Interscience, New York, 1985.

[7] U. Shardanand and P. Maes, "Social Information Filtering: Algorithms for Automating Word of Mouth" *Proceedings of CHI '95*, Denver CO, pp. 210-217, 1995.

[8] www.icecast.org

[9] www.almaden.ibm.com/cs/TSpaces

[10] www.almaden.ibm.com/cs/wbi

Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)
0-7695-1284-4/01 $17.00 © 2001 IEEE