A Web Interface for MIT's Library Access to Music Project

Josh Mandel
Course VI AUP

May 7, 2005

1 Introduction

MIT's Library Access to Music Project launched a new kind of music library in October 2004, allowing community members to listen to more than 1700 CDs over the on-campus cable television network. To access LAMP, users select music from a Web site, then listen by tuning a TV to the assigned channel. This paper describes my design of LAMP's Web site, which sits at the interface between the MIT user community and the back-end broadcast equipment.

LAMP has been popular since its October 2004 launch, with over 12,000 songs played, 1200 distinct users in total, and about 100 distinct users each week. This semester I created a campus music map to show how music choices vary across MIT's campus. I also designed an algorithm to recommend new music to users, along with an experiment to determine the algorithm's efficacy. The algorithm's performance was mixed, but novel recommendations were rated significantly better than a random choice. In this paper I also describe design decisions and user-interface challenges in building the Web site.

2 Overall Goals

LAMP's Web site provides thousands of users access to thousands of CDs, and my primary goal was ease of use. The site was designed with a few use cases in mind:

> Jill comes to the site knowing what she wants to hear. She should be able to locate her music and play it over LAMP with very little overhead. If the music is unavailable, she should be able to request a new purchase.
>
> Jack comes to the site because he heard a song he liked on one of LAMP's TV channels. He should be able to easily learn what music it was, find and play similar music, or buy his own copy.
>
> Sam wants to learn more about what MIT students listen to. He should get a broad sense of community preferences, with the ability to drill down into particular sections of the campus.

I designed the site to serve these users, responding quickly to requests and providing plenty of information about CDs, songs, artists, performers, composers, and conductors. I also wanted

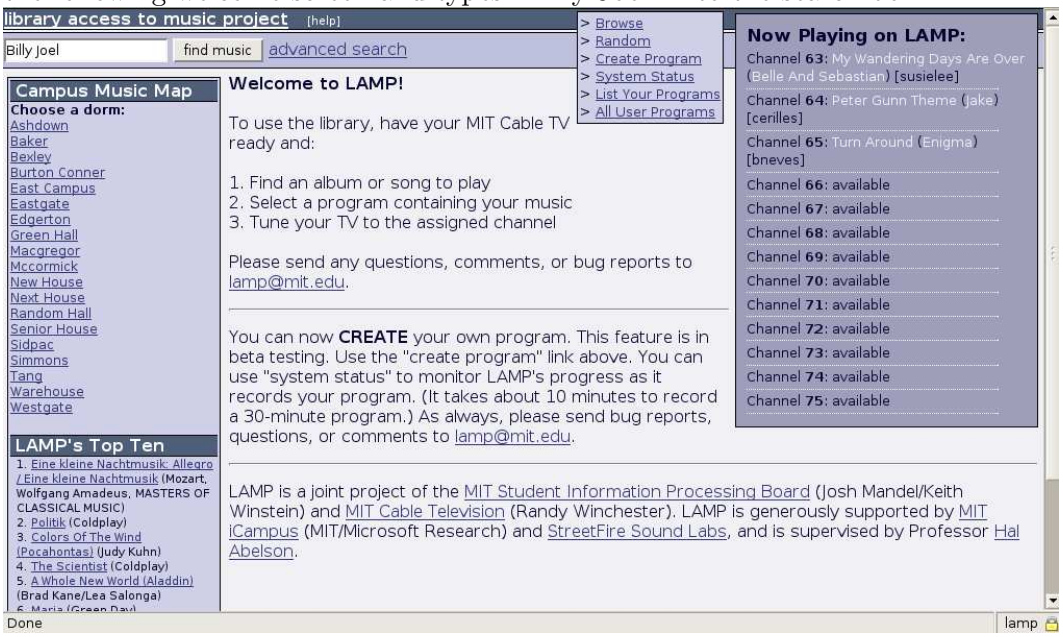
to provide a rich set of links between songs and albums, allowing users to easily navigate a web of music. Every page should provide links to similar pages, allowing users to browse LAMP's collection and discover new music. The site was also designed to foster community. Users should feel connected to one another through music, and information should be available about who's using the system at any time, as well as a historical overview of usage.

3 A Tour of LAMP's Web site

To get a better idea of how the system works, we'll start out with a tour of LAMP's Web site from the user's perspective. From here, we'll delve into the technical details of how the site was designed and how it operates.

3.1 Playing A song

Jill, then, logs into LAMP because she wants to play some music by Billy Joel. She first connects to `https://lamp.mit.edu` with an MIT certificate installed on her web browser, where she sees the following welcome screen and types "Billy Joel" into the search box.



A list of eleven albums is returned, along with a link to a list of 156 songs.

Jill decides to look at the list of available songs, so she clicks on the "Songs" link.

Finding the song "Worse Comes to Worst," she clicks the "List Programs" link to show all programs containing the song:

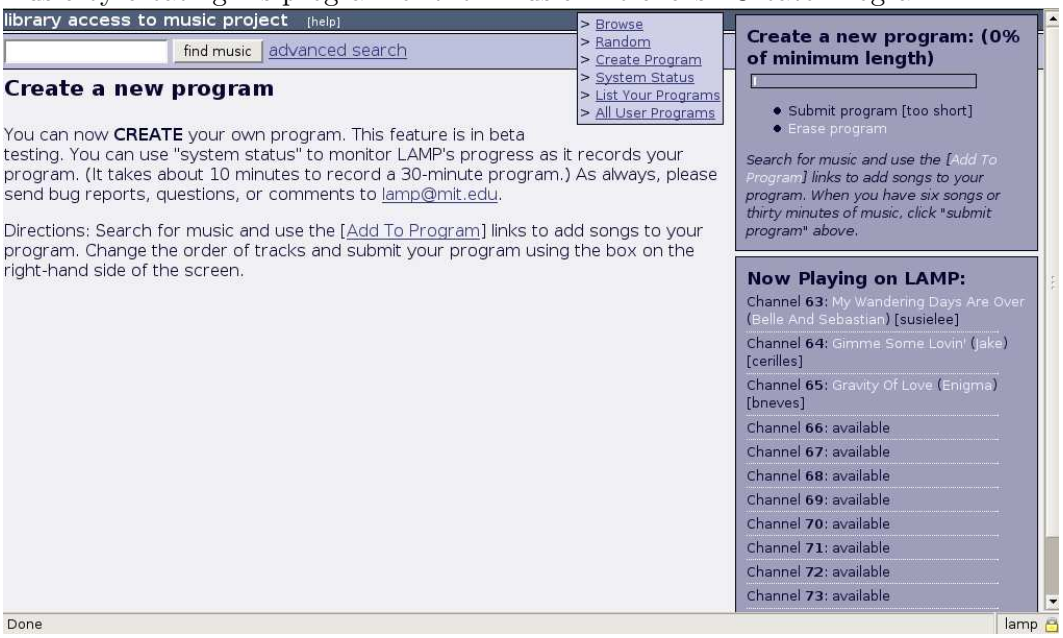
Finally, she clicks on "Play Program 2" and tunes her TV to listen.

3.2 Campus Music Map, Making Programs

Now consider Sam, who wants to learn more about music on campus. After logging into LAMP's Web site, he chooses the campus map to see what's popular at Random Hall:

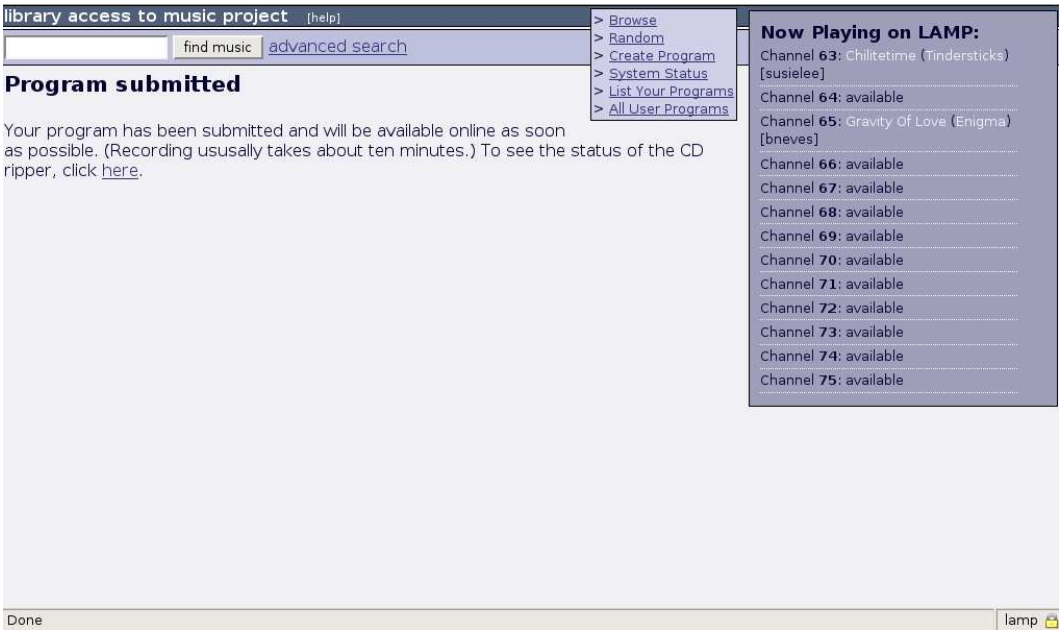
Sam sees that They Might Be Giants are the top band, and decides to listen to some of their

music by creating his program of their music. He clicks “Create Program”:



Now he browses the collection, choosing songs to add:

Once the program is long enough, he submits it to be ripped by LAMP’s CD changers:



4 Technical Design

LAMP's Web site is the interface between MIT community members and the back-end broadcast system. The site shows users what music is available, and allows them to control playback. Authorized requests are sent to a back-end broadcast server using the short set of commands below.

Command	Effect
`lamp-play`	Play specified song on specified channel
`lamp-pause`	Pause music on specified channel
`lamp-resume`	Resume playback on specified channel
`lamp-stop`	Stop playback on specified channel
`lamp-channel-status`	Mark specified channel as "available" or "in use"

In Web development, LAMP's acronym has another meaning: "Linux, Apache, MySQL, and PHP." Given the eponymous name and my background using the platform, LAMP was a natural fit for LAMP.

In broad terms, LAMP's Web site works according to Figure 1. Users authenticate before entering the site; from there, a central Display module provides links to all of LAMP's functions. Requests are sent to a separate handling facility which processes them and returns the user to the Display module, which now provides the appropriate results. We'll take a concrete example of this kind of transaction shortly.

Most projects of this scope would require passwords for authentication and cookies to track state. LAMP, however, is fortunate enough to serve a community with a better option: SSL certificates. Essentially every member of the MIT community already has an SSL certificate, signed by MIT's key. LAMP's Web site uses `apache-ssl`, requiring users to present their MIT-signed certificate for

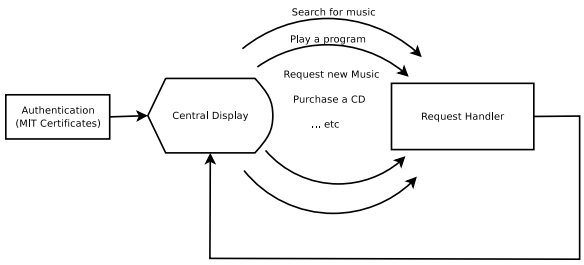


Figure 1: LAMP Web site Overview.

authentication. As an added bonus, certificates can also be used to keep track of state, eliminating the need for cookies: since every exchange between LAMP and the user is made over SSL, the certificate is presented with each request, and the unique `athena` username is used as a session identifier. This approach means that a single user cannot participate simultaneously in multiple LAMP sessions, but the constraint is natural enough for the domain: a user is only allowed, after all, to control music on one channel at a time.

Display and processing are done in PHP, using an object-oriented approach. Objects representing songs, albums, and programs are built on-the-fly from persistent data stored in a MySQL database. A depiction of this mechanism and a map of objects are shown in Figure 2. For instance, in a `songs` table in the database, each row holds the ID, title, and performer for one song. A PHP Song object is created on-the-fly by loading the appropriate data into memory from the database. The object then provides methods to display data, find related albums, generate web links, etc.

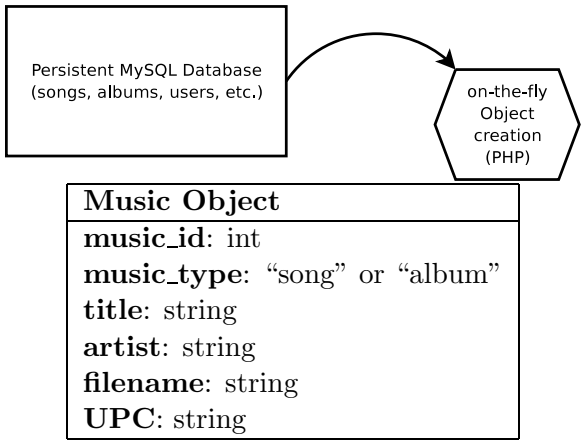


Figure 2: On-the-fly object creation from persistent data, and a set of objects.

One challenge was to keep track of the results of users' search requests. For instance, searching LAMP for "The Beatles" finds about twenty albums and three hundred songs. In this case, the result is broken down into several pages, and to preserve state across page requests, I save entire list as a "Searchset" in the database. Subsequently, search results are displayed one page at a time by loading the Searchset from the database and extracting the appropriate page.

A concrete example goes a long way in elucidating the system. When a user searches for music by Fiona Apple, the display module generates a link to the handling facility at the address: `lamp/process-requests?req=basicsearch&query=Fiona%20Apple`. The handling facility

logs the request and performs a search with the following code, where `$t` is "Fiona Apple":

```
function process_basic_search($t) {
        $s = new Searchset("basicsearch", $t);
        return $s->link();
}
```

A Searchset is created, and subsequently provides a link back to the Display module. Inside the Searchset constructor, the search term is tokenized into the two terms "Fiona" and "Apple," stored in the array `$words`. These are used to construct a database query which provides a set of album IDs:

```
// construct query by AND'ing together token words
$which_tokens = " where ";
for ($i = 0; $i < sizeof($words); $i++) {
  $words[$i] = " tokens like '%$words[$i]%'  ";
}
$which_tokens .=  implode($words, " and ");

// execute query using the derived $which_tokens string
$this->albumquery =  ``SELECT album_id FROM albums $which_tokens
                                            order by album_upc'';
```

At this point, the Searchset is established, saved to the database, and finally returns a link to itself. The user is directed via this link back to the Display module.

This approach provides a relatively clean separation between display code and internal logic. On the display side, I took a light-weight approach to site design, cutting out unnecessary exchanges and keeping page size small and browser requirements low. LAMP uses HTML and CSS for formatting, displaying images only when text is impossible (i.e. for corporate logos and campus maps). The site does not rely on java or javascript.

5 User-interface Design

Creating an intuitive Web interface for LAMP was challenging because the service does not mesh well with users' existing paradigms. First, no one is used to a service where music is selected over the Web but played back over cable TV. Next, users are not familiar with LAMP's notion of "programs." For copyright reasons, users must play an entire program of songs in sequence and as a unit; so although the service is on-demand, the basic unit of playback is the program, not the song. The natural paradigm is to search for a song and play it back, but LAMP requires users first to search for a song, then to select a program containing that song, and finally to play back the program.

I tried to forestall these concerns by providing documentation and on-line guidance at decision points. This guidance begins with LAMP's welcome screen, where the basic work-flow is explained as a three-step process (Figure 3). When a user searches for a song, results are displayed alongside links to available programs. In observing users unfamiliar with LAMP, I found that their comprehension was quite sensitive to the way these links were named. For instance, confusion arose

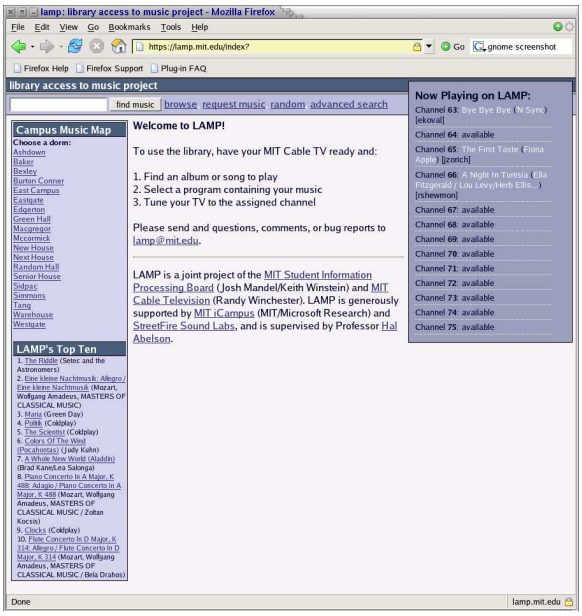


Figure 3: LAMP Web site welcome screen with instructions.

when a search returned links to "Find Programs," but changing the link to "List Programs" helped clarify that clicking this link was the next step in the process.

A good example of a self-documenting feature is LAMP's program creation page. When a user chooses to create a new program, she is immediately presented with a concise set of instructions in the main page body (see Figure 4), as well as a persistent summary remains on-screen throughout the process:

> Search for music and use the [Add To Program] links to add songs to your program.
> When you have six songs or thirty minutes of music, click "submit program" above.

I also provided a more detailed user's guide that explains how to find, play, and listen to music on LAMP. This guide is available from every LAMP page with one click.

My goal at all times was to keep the user interface simple. LAMP's pages load quickly and strive to provide a large number of well-labeled links.

6 LAMP Usage

LAMP's original launch in 2003 was much acclaimed but short-lived. That service, which allowed users to play individual songs rather than programs, saw more than three hundred distinct users on its first day of operations, and over a hundred distinct users on each of the following two days. The current service was launched in 2004 with less fanfare, and fewer users. Figure 5 shows weekly usage since the 2004 launch, indicating an early peak of 160 users per week and a slow decline to about 100. The most noticeable gaps are due to school holiday (December to January) and a system outage when LAMP's server was compromised by Belizean hackers (March). (Because the back-end broadcast system is insulated from the public interface, the hackers could not access LAMP's stored music.)

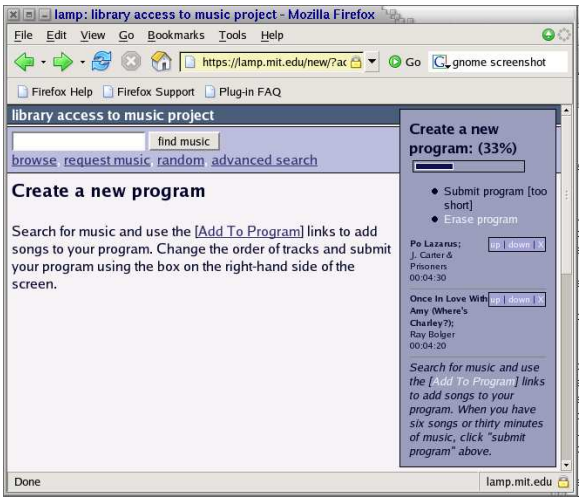


Figure 4: LAMP Web site program creation page.

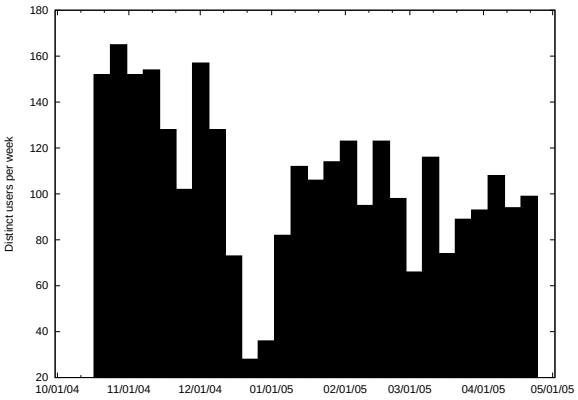


Figure 5: Web site weekly usage

LAMP sees a wide variety of users, from the merely curious to the die-hard addicts, as shown in Figure 6. Of 1200 total users, about 375 played just one program; about 80 are power-users with more than three-hundred song plays; and the distribution falls off steeply in between.

On the one hand, just watching LAMP on TV is vegetative. But on the other hand, there's something fulfillingly mesmerizing about the smooth, cyclic scrolling. And people do watch what others are playing, with some very interesting results! One night in April, LAMP filled to capacity with users all playing an album by Paul Simon. Users also influence each other on a smaller, less orchestrated scale. Indeed, it happens often that a user will see an interesting song on one channel and then be compelled to play something similar on her own channel. One example of this phenomenon: a user played Beethoven's Ninth Symphony on LAMP, and six minutes later another played Eine Kleine Nachtmusik on a different channel.

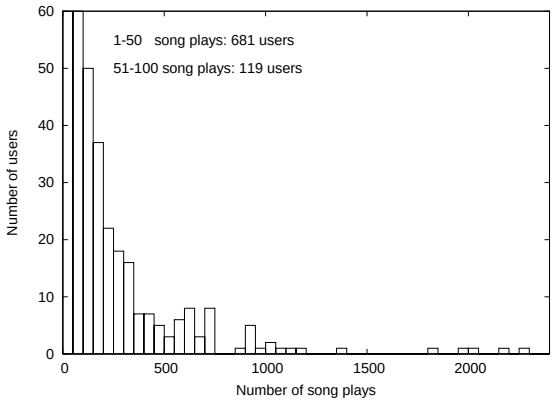


Figure 6: Histogram of song plays per user

7 A new campus map

With usage data from LAMP, I constructed a campus music map to show how musical tastes vary from dorm to dorm at MIT. The map is a playful take on campus culture, giving users a qualitative overview of who plays what, and where.

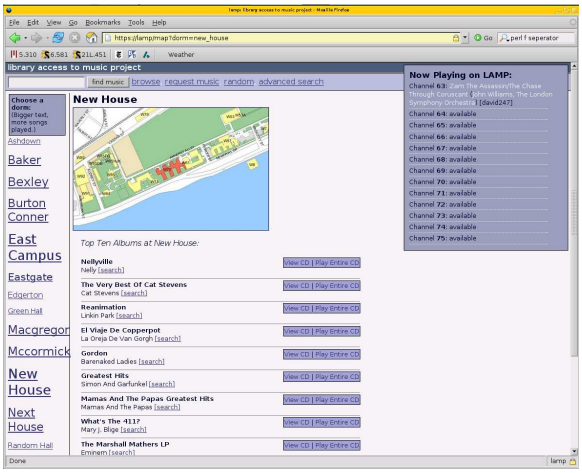


Figure 7: Campus music map

The map was constructed by finding the "Top Ten" most-played albums for users in each dorm, and most had about ten users supporting them. The map also provides images taken from `whereis.mit.edu`, displaying the location of each dorm on campus along with the music.

One surprising result demonstrated by the map is LAMP's popularity with graduate students as well as undergrads. Grad dorms are represented strongly, with Sidney Pacific, Tang, and Westgate each playing more songs than some undergrad dorms.

The map also shows that users come from all parts of campus, and broadly speaking, tastes vary dramatically from place to place. On the one hand, the map indicates trends likely to re-enforce campus stereotypes: for example, two of the top albums at Random Hall are by the nerdy They Might Be Giants. But at the same time, the map reveals some "unexpected" selections: at Bexley,

The Phantom of The Opera, Joseph and the Amazing Technicolor Dreamcoat, and Jesus Christ Superstar all made the top ten.

8 Predicting Musical Preferences: an Experiment

With over 100,000 song plays by 1200 users, LAMP's usage data contain a wealth of information about musical preferences on campus. To harvest some of this information, I designed an algorithm to recommend new music to users based on their choices and those of others. To evaluate the effectiveness of my algorithm, I ran an experiment asking users to listen to and rate the suggested music on a few criteria. A control group was given music selected at random from LAMP's collection.

8.1 Music Recommendation Algorithm

My algorithm for suggesting music is perfectly simple: I constructed a matrix of (All Albums vs. All Albums) and populated it by iterating over LAMP's users. For every pair of albums $(i, j), i > j$ played by one user, I incremented the value of the matrix at location (i, j). Once constructed, the matrix contained at location (i, j) the number of users who had played both album i and album j on LAMP, which I call the score, $S(i, j)$. Recommendations were made for a particular album i by searching row and column i of the matrix for cells with the highest values and recommending the corresponding columsn and rows, as in the example in Figure 8.

—	—	—	—	—
$S(1,0) = 2$	—	—	—	—
$S(2,0) = 2$	$S(2,1) = 5$	—	—	—
$S(3,0) = 4$	$S(3,1) = 0$	$S(3,2) = 1$	—	—
$S(4,0) = 1$	$S(4,1) = 1$	$S(4,2) = 2$	$S(4,3) = 3$	—

Figure 8: Example score table for a five-album collection. A search for recommendations based on Album 3 would return, in order, Album 0, Album 4, and Album 2.

8.2 The Experiment

The pressing question, of course, is how does such a simple algorithm perform? Looking at the algorithm's recommendations casually, it seemed that most were pretty good; in most cases the recommended album at least fell in the same genre as the input album. Some recommendations were particularly astute (If you like an album by The Counting Crows, you might like one by Dave Matthews Band), but others seemed awkward (If you like Paul Oakenfold's Tranceport, you might like Frank Sinatra). To objectively evaluate the algorithm's performance, I ran the following experiment.

8.2.1 Experimental Design

After selecting a program to play on LAMP, users were asked instead to participate in an experiment. They were informed that this was an experiment to evaluate a music recommendation algorithm, and given the option to participate or decline. Users who participated were placed in a control group with 25% probability, or an experimental group with 75% probability. In both cases,

users were asked to listen to a program and rate it on a scale of 1 to 5 overall (1 means dislike strongly, 5 means like strongly). Users also rated the program on similarity to their original selection (1 means strongly dissimilar, 5 means strongly similar), and indicated whether they had ever heard the selection before. Users in the control group were given a program selected at random from LAMP's entire program inventory; users in the experimental group were given a program derived from the album with the highest score according to the recommendation algorithm.

8.3 Results

Overall, about 500 offers were extended to participate in the experiment; of these, about 110 were accepted, but only 72 garnered responses. I evaluated these responses using a few statistical tools. First, the following table shows the mean and standard deviation for overall score and similarity score, as well as the percent of people who had heard the recommended selection before.

Condition	Number Respondents	Overall Score	Similarity Score	Heard before?
Control	15	3.3±1.0	2.4 ± 1.0	53%
Experimental	57	3.6±1.1	3.1 ± 1.4	68%

Clearly these numbers are not astonishing. We see that average scores were higher for the experimental group than the control group, but by a small margin. To determine whether this effect was significant, I created a probabilistic model for each condition based on the distribution of responses, and I ran a 10,000-trial Monte-Carlo simulation to determine how often scores were higher for the experimental group vs. the control group in a 70-person trial. The mean overall score was higher in just 85% of trials, whereas the mean similarity score was higher in 98% of trials. So it seems reasonable to conclude that the algorithm generated programs significantly more similar to the user's original choice than a random selection process; but these were not significantly better liked, using the conventional p=0.05.

If, however, we only consider responses where the user had never heard the music before, a significant effect emerges. In this case, the Monte-Carlo simulation shows that average scores are higher in the experimental group 98% of the time. In other words, the recommendation algorithm beats a random selection when both suggest novel music.

Overall performance of the recommendation algorithm was disappointing, but not necessarily surprising. The approach was perhaps too simple to perform really well, but it was interesting to see how an unsophisticated approach woulddo. And at the same time, the recommendation algorithm performed significantly better than chance at recommending novel music, which is one of its main goals. After all, people take recommendations to learn about music they haven't heard before.

9 Future Improvements

I think a key factor in LAMP's continued popularity will be its ability to keep up with new music. Today, all the CDs in the system are more than one year old, and the absence of current hits will drive some users away. We maintain a list of buy requests that currently contains over 300 UPCs, indicating that users are ready for new music. On the other hand, usage has been quite stable, and the community is obviously enjoying the system. One user appreciated LAMP so much that he donated his entire CD collection to the library!

Another important direction will be to improve the community aspect of LAMP's Web site. When users begin to create their own programs, implementing a rating system will become important so users can more easily find what they want. For instance, users may be able to vote for or recommend a program they like, causing it to appear higher in future search results.

As far as the automated recommendation system, a more effective approach might be to use a collaborative filtering recommendation system. This way, sets of users with similar tastes could be identified, and better recommendations made.

10 Conclusion

This semester I improved LAMP's Web site by adding a system for users to create their own programs, a browse feature, a campus music map, and a system for recommending music. I also improved the user-interface by separating the playback process into steps, describing the process in clear language and adding on-line help. An experiment to determine the effectiveness of the music recommendation system produced mixed results, but the algorithm performed well at recommending novel music.

Overall, LAMP's Web site has been very successful, serving about a hundred distinct users per week. Someone is always using the service, and most nights five or six channels are in use at once. And this overview, of course, discounts LAMP's passive users who simply tune their TV and listen to whatever's on.

11 Acknowledgements

First of all Keith Winstein. Thank you, Keith, for sharing your vision and letting me play a role. It's been a lot of ups and downs over the past four years, and though the lows were low, the highs sure were high. I am forever indebted.

Thanks, also, to Hal Abelson for supervising this work, to iCampus for sponsoring it, and to French House for being such willing guinea pigs.