AreWeThereYet? - A Temporally Aware Media Player

Matt Adcock, Jaewoo Chung, Chris Schmandt
MIT Media Lab
20 Ames St, Cambridge, MA 02142, USA
{matta, jaewoo, geek}@media.mit.edu

Abstract

In this paper we describe the design and implementation of the AreWeThereYet? (AWTY) Player - a (digital) audio player that composes a program of audio media that is extremely likely to fit within the user's available listening time. AWTY uses time compression and track selection techniques to help the listener make more efficient use of their time. More importantly, it possesses an awareness of the listener's temporal context. It forms an estimate of the available listening time and uses this prediction to compose a playlist of a suitable length. We hope that this research prototype will inspire others to further investigate the ways in which temporally aware computing might be employed.

Keywords: Temporal Awareness, Time Compression, Location Based Computing, Audio, Context Awareness.

1 Introduction

We often have 'idle' time in which we consume various types of audio media such as radio, audio books, mp3s, CDs, podcasts… However, in order to 'commit' time to listening, we may well prefer to know that there will be a natural conclusion or break when we need to stop listening.

In this paper we present the AreWeThereYet? (AWTY) Player - a (digital) audio player that can compose a program of audio media that is extremely likely to fit within the user's available listening time. This differs from many previous approaches to personalized media scheduling and play list composition in that it is not trying to guess personal preferences or play content snippets based on location. Instead, the audio player's 'intelligence' comes from its ability to use a listener's current location and their predicted destination to calculate an estimate of the available listening time.

Additionally, for spoken word audio, the AWTY Player uses time compression in order to maximize listening efficiency. Pitch invariant time compressed speech is comprehensible to the listener at rates of 1.5x and sometimes 2x the normal play rate (Foulke and Sticht 1969). Additionally, after some exposure to this type of time-compressed speech, people actually prefer it to uncompressed speech (Beasley and Maki 1976).

1.1 Example Usage Scenario

Katlyn works at a travel agency, and usually walks to work each morning. She's a little behind in listening to her podcasts, so she flicks on her AWTY Player. The AWTY player connects wirelessly with her phone which is tracking her location and providing estimates of her remaining trip time. The AWTY Player subsequently assembles a program of some of her latest podcasts that it thinks will play for a similar duration to her journey. Then, while she walks, it monitors her progress and automatically adjusts the play list and play speed based on her phone's successive estimations of her arrival time. The podcasts finish playing just a couple of meters from the door to her shop, and her AWTY Player goes to sleep. It's just like clockwork.

2 Time Matching Techniques

There are two main ways that a system like this could provide media to fit within a specific time window. It is important to note that the two techniques described below are not mutually exclusive.

The first method is to use time compression. Time compression works pretty well for most spoken word audio, but has artistic hurdles where music is concerned. There are a number of methods that permit speech to be played at speeds in excess of 1.5×, while maintaining the original pitch. Speech playing at around 1.4× normal speed usually maintains a high level of comprehensibility.

The second method is to employ track selection or, in other words, simply choose tracks with play lengths that add to the required time. This relies on a large source of audio material and can be used for music as well as speech. The experience for the user is quite similar to hitting 'shuffle' or 'random' on today's commercially available media players.

The initial implementation of the AWTY system focuses on the playing of audio books. In this case the chapter order is predetermined, so the track selection method is not possible. Instead, it relies solely on time compression. In doing so, we also demonstrate the technique that many people would be least familiar with.

We assume that the listening preferences are already defined by the user in their previously created playlist(s). We also assume that, given enough idle time, the user is happy to listen to all of the content.

2.1 Speech Audio: Track Selection with Time Compression

The primary algorithm for selecting how many speech tracks to play, and at what rate to play them, is as follows:

1. Determine the maximum number of tracks (including any partial current track) that can be played within the estimated listening time. This is done by simply adding track lengths and dividing by the user's preset maximum time compression threshold.
2. Divide the estimated amount of listening time by the total (normal speed) play times of the selected tracks. This will give the play rate needed for the tracks to play until the estimated end of the journey.
3. This resulting play rate (or time compression factor) is then used until either the estimated remaining listening time diverges from the remaining play time or there is user interaction, such as rewinding. If either of these occurs, we return to step 1.

A graphical example of this algorithm, using chapters from LoudLit.org's reading of "Adventures of Huckleberry Finn", and a maximum time compression threshold of 1.5 can be seen in Figure 1. In this example, an ETA (Estimated time to Arrival) of 500 seconds would result in (only) chapter 1 of the audio book being played at a rate of about 1.2 times normal speed. If, however, the ETA was 1000 seconds, there would be enough time to play chapters 1 and 2, provided we played them at a rate of about 1.35 times normal speed. As a further example, an ETA of 2000 seconds would permit chapters 1-5 to be played at around 1.4 times the normal rate.

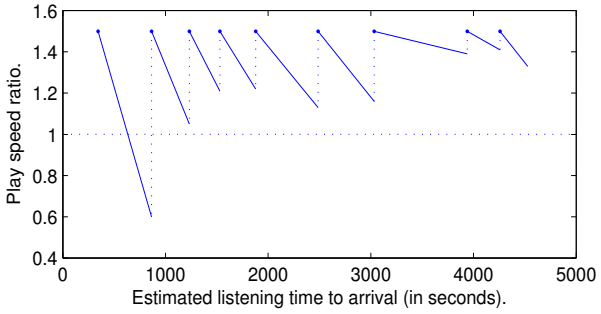

Figure 1: For a given set of track lengths, this graph shows a snapshot of the relationship between the estimated available listening time and the resulting audio play rate. If the estimated listening time stays constant for the entire journey, the play rate will also remain constant.

As can be seen in the Figure, the longer the estimated remaining listening time, the more likely the compression factor will be close to the user's maximum compression threshold (assuming the tracks are of somewhat similar length).

The graph in Figure 1 is only a snapshot. As time progresses, the ETA should decrease, and the remaining duration of the currently playing audio track will also decrease. It is therefore important to realize that if the ETA never diverges from its initial value (i.e. it was correct from the beginning), the play rate will remain exactly constant throughout the entire journey.

If the estimated time to arrival is small compared to the lengths of the given tracks, it is possible that there is not enough time to play even the first track at maximum play rate. It is also possible that a play rate of less than normal speed is needed to exactly match the estimated listening time. In both cases, our solution is to first warn the user that the system performance will be sub-optimal, but also to begin playing the track at maximum rate or normal rate respectively.

The system is robust to user intervention during the journey (e.g. a rewind and replay the last two minutes of the current chapter). It simply performs the same track selection and audio calculation and issues warnings if required.

2.2 Track Selection from a Music Library

When playing music tracks, we are likely to want to preserve the play rate and therefore time compression is not used. To calculate a playlist that is as close to the estimated listening time as possible, one option is to exhaustively solve the *knapsack problem* (Cormen et al. 1990) (named after a hypothetical thief trying to choose a set of objects that will best fill his knapsack).

However, if we assume the user has a suitably large set of tracks from which to draw, and also assume that the given journey is an order of magnitude larger than the average length of tracks in the library, we can employ a cheaper (approximate) algorithm, as follows:

0. Create an index of all tracks in the collection ordered by track length, and determine the average track length (this only needs to be done once).
1. Randomly select individual tracks and add them to a list until the list is close to the total estimated listening time (e.g. within half the average track length).
2. Rank the selected tracks by their difference from the average track length (such that the longest and shortest tracks will play first). We do this to increase the chance of 'manoeuvrability' (which is described below).
3. Swap some small number of tracks near the end of the list (the ones closest to the average length) for tracks that will result in the total playtime. For example, if the list generated in step 1 is 90 seconds longer than the estimated listening time, replace the last three songs with tracks that are about 30 seconds shorter.

If the resulting playlist not within some threshold (e.g. 2%) of the estimated remaining listening time, we have the option of falling back on an exhaustive solution to the knapsack algorithm.

As the user's journey progresses, we will obtain further refinement of the ETA. If ΔETA is small (with respect to the ETA used to create the current play list), we can swap the last song(s) in the list accordingly. Alternatively, and especially if ΔETA is large, we recalculate the whole list

(this is possible as we do not disclose the list to the user). Note that if the user is currently listening to a track, we do not interfere with that track.

The simplified algorithm above is based on the assumption that the music library provides sufficient choice. Figure 2 shows the distribution of track lengths in a typical mainstream music library (of 3196 tracks). The average track length is 245s and the closer songs are to that, the greater the chance of having alternative tracks to 'switch to' when ETA refinements are obtained. Additionally, there is a high degree of contiguity in the track lengths. More than 92% of the (non-duplicate) track lengths in the library have another track within 3 seconds of their length.

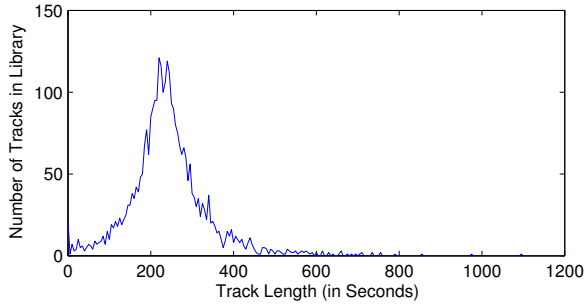

Figure 2: A histogram showing the track lengths of a typical music collection (containing 3196 album-version songs totaling about 18GB). Tracks are bucketed at intervals of 5 seconds.

3 Implementation Details

The initial 'proof of concept' demo was implemented as a windows application, written in Java (see Figure 3). It uses the Winamp Media Player to manage audio playing and the PaceMaker Winamp Plugin to stretch or shrink audio without affecting the pitch. The main body of Java code communicates with Winamp and PaceMaker via the Java Native Interface and a custom Dynamic Link Library (DLL). This DLL sends commands and requests to Winamp and PaceMaker through the Windows Messaging interface.

The system is largely based on a Model-View-Controller design. Additional Java components of the system include an Application Controller, GUI, Map Display, and a Fake Trip Generator. This last component, which is implemented in its own separate thread, is used as a surrogate for real trip time estimations.

When running in demonstration mode the system uses 'fake' trip data. In this mode, the Map Display indicates the 'location along the route' and 'estimated time until arrival'. It also provides for user control of a proxy traveller so as to demonstrate the dynamics of the system at various points along the route.

Figure 4 shows an example of similar stages along the same journey, and in each case the person is travelling at two slightly different speeds.

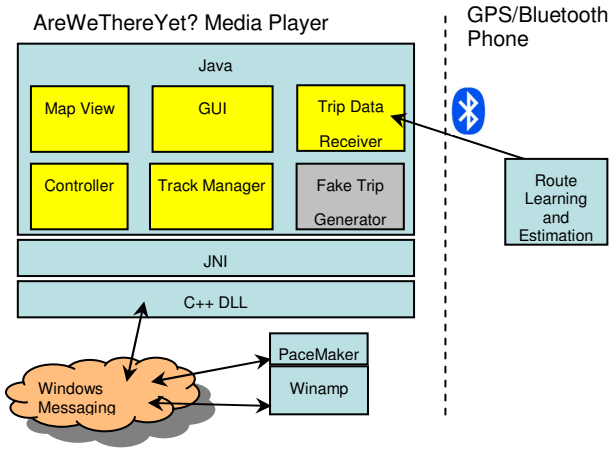

Figure 3: The AWTY System Architecture.

4 The Working Mobile System

We have implemented a mobile version of the AWTY Player on an OQO pocket computer, a WindowsXP device that is about the same size as today's lager portable MP3 jukeboxes.

4.1 Route Prediction and Wireless Connection

To calculate the user's available listening time, we used the estimated time to arrival (ETA) for an established travel route. This ETA is calculated by Contella (Chung 2006), a route prediction application which runs on a Motorola i870 mobile phone. In order for the application to detect a given route, the user is first required to train the system by travelling along that route. This is ideally done as part of the user's daily routine. Then, when the user revisits a given route, the application is able to predict the user's ultimate destinations as well as the associated ETA.

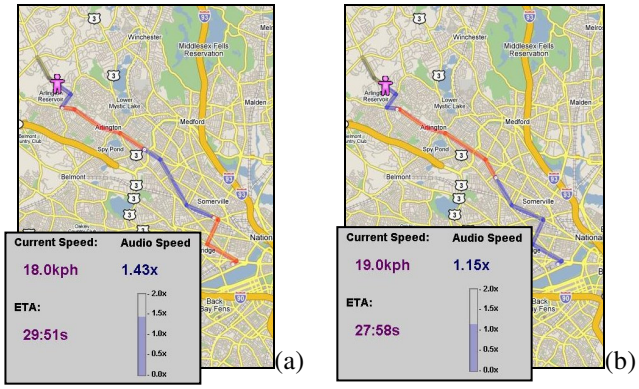

Figure 4: (a) The map and status indicators from the AreWeThereYet? GUI. The red and blue lines show the sections of the journey where it is anticipated that each respective track will play. (b) After a slight increase in speed, one track has been dropped from the play list and the audio compression level has, in turn, decreased.

The Contella application uses Motorola's JAVA API to retrieve GPS fixes every five seconds (in both the training and general use situations). The GPS fixes are used to collect information about intermediate locations every 50 metres, and the series of intermediate points (iPoints) is used to generate a route template (as shown in Figure 5).

The template for any given route also contains information about the time of day when the user was on the route.

Sometimes these route templates are generated under non-ideal conditions. Poor GPS reception can result in some inaccuracies in the GPS coordinates recorded and some gaps (of greater than 100 meters) may exist between iPoints. However, Contella re-samples GPS coordinates every time a route is traversed. It will attempt to replace the GPS coordinates of iPoints' that had initially been recorded with poor accuracy, and plug gaps in which iPoints were missing.

Contella generates a prediction of the ultimate destination based on the user's current direction, the route that the user has travelled so far and the time of day that the user is travelling. In addition, the ETA is calculated not only based on speed, but also the average speed of the user during previous traversals of the respective route.

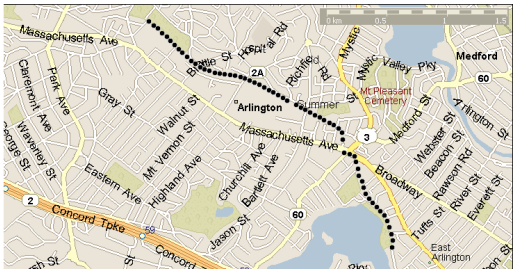

Figure 5: A set of iPoints along a typical route.

Each route template is divided into regions of similar speeds (roughly corresponding to different modes of transport). For each section, the route template maintains a weighted average of the speeds collected during previous journeys along the respective route (giving the most weight to the most recent journey). Whenever a user's current travel speed is within range (+15%/-30%)[1] of the corresponding (weighted average) speed held in the template, that template is used to calculate the ETA. If a user exceeds that speed range, the application calculates the estimated travel time for that region based on the user's current average speed within that region. The total ETA is always calculated using the template values for any subsequent regions.

On average, the algorithm updates its prediction every 40 seconds. When the route prediction application updates a prediction, it sends the ETA to the OQO device via Bluetooth. The ETA message is a simple string containing the estimated number of seconds. Given the update frequency, if the device is within one minute of the estimated arrival point, we just maintain a constant time compression factor and play until the end of the track.

5 Related Work

AWTY Player contains some echoes of the time critical music scheduling systems found in broadcast radio stations. In fact, there are now a number of desktop applications that perform similar functions (such as the GPL'ed GJay software). There are also many systems that aim to automatically generate play lists. Some, such as CUIDADO (Aucouturier and Pachet 2002), base the play list on the current user specified preference. Others, like PATS (Pauws et al. 2002) and Riot! 1831 (Cater et al. 2005) generate a playlist for the specific real world environment or location (respectively) in which the audio will be heard.

It is also worth noting that most car navigation systems already calculate at least one instance of an estimated journey time. This suggests that car stereos (which are often located in close proximity to the GPS navigation systems) would be an ideal benefactor of the techniques described in this paper.

6 Conclusion

In this paper we have described the design and implementation of the AreWeThereYet? Player. AWTY uses time compression and track selection techniques to help the listener make more efficient use of their time. More importantly, it possesses an awareness of the listener's temporal context. It forms an estimate of the available listening time and uses this prediction to compose a playlist of a suitable length. We hope that this research prototype will inspire others to further investigate the ways in which temporally aware computing might be employed.

7 Acknowledgements

Thank you to Pattie Maes for her helpful input. Matt Adcock is on leave from CSIRO Australia while at the MIT Media Lab.

8 References

Aucouturier, J.-J. and F. Pachet (2002) "Scaling up Music Playlist Generation", *IEEE International Conference on Multimedia Expo*, Lausanne (Switzerland).

Beasley, D.S. and Maki, J.E. (1976) Time- and Frequency- Altered Speech. In N.J. Lass, editor, *Contemporary Issues in Experimental Phonetics*, Academic Press, 419–458.

Cater, K., Fleuriot, C., Hull, R., and Reid, J. (2005) Location Aware Interactive Applications. In: ACM SIGGRAPH 2005, Conference Abstracts and Applications, ACM.

Chung, J. (2006) Will You Help Me: Enhancing personal safety and security utilizing mobile phones. Masters Thesis, MIT Media Lab.

Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

Foulke, W., and Sticht, T.G. (1969) Review of research on the intelligibility and comprehension of accelerated speech *Psychological Bulletin*, 72, 50–62,

Pauws, Steffen, and Eggen, B. (2002) "PATS: Realization and user evaluation of an automatic playlist generator", *Proceedings of the 3rd International Conference on Music Information Retrieval*, 222-230.

[1] This speed range was reached through trial and error.