

“Recording on Rails”

The Quest for an Intelligent Recording Environment

by

Bennett Kolasinski

Submitted in partial fulfillment of the requirements for the
Master of Music in Music Technology
in the Department of Music and Performing Arts Professions
in The Steinhardt School
New York University
Advisor: Dr. Juan Pablo Bello
DATE: 2007/12/19

Introduction	3
<i>Personal Motivation</i>	<i>5</i>
<i>Structure</i>	<i>7</i>
A Framework for Automatic Mixing	8
<i>Overview</i>	<i>8</i>
<i>Cost Function</i>	<i>9</i>
<i>Cost Function Evaluation</i>	<i>13</i>
<i>Cost Function Development</i>	<i>14</i>
<i>Genetic Optimization</i>	<i>17</i>
<i>Representation</i>	<i>19</i>
<i>Initialization</i>	<i>20</i>
<i>Evolution</i>	<i>21</i>
<i>Reproduction functions: Crossover and Mutation</i>	<i>21</i>
<i>Selection</i>	<i>23</i>
Test Corpus	23
Tests	25
Results	28
Future Work	38
Conclusion	41
Code	42
Works Cited	43

Introduction

The line between content producers and consumers is blurring more rapidly by the day. This modern-day phenomenon can be attributed to two major factors-- the ease of access to information via the internet and the ever-lowering bar of entry. Not even a decade ago, basic video and audio editing and composition were relegated to top-of-the-line computers with expensive outboard gear or dedicated hardware; today, such features are available on the most basic entry-level computers in ever-shrinking sizes and price points.

Conversely, computers are capable of handling more and more audiovisual information than ever before. Audio editing software on higher-end computers today can handle the recording and playback of dozens or even hundreds of audio tracks with effects on each of them with ease.

Whether you are a novice user trying to make edits and mix music into your latest podcast or you are an expert music editor trying to manage a hundred tracks in a professional recording, you will at some point be faced with dealing with a daunting amount of data. Most audio software provides a familiar interface-- a track view and a mixer view, or some combination thereof-- that is the user's window into his mix. It is up to the recordist to negotiate that interface and decide what to do with each track.

There are a number of ways to minimize the 'grunt work' of getting a mix up and running. Recording software typically ships with a number of templates and presets and allows the recordist to create his own as well. For example, upon launching Garageband, Apple's entry-level recording software, the user is presented with the

option to create a new podcast episode. If this option is selected, a new recording session is created with an arrangement for a typical podcast-- male and female voiceover tracks, a music track, and a 'podcast' track for any additional multimedia tracks to be added. Furthermore, the voiceover tracks have a number of effects applied, such as compression and equalization. This allows the novice user to get a more 'professional' sound without having to fully know how that sound is achieved.

A typical professional recording studio will have a number of templates and presets for different workflows the studio typically handles. Like the 'new podcast' preset in Garageband, templates and presets in a recording studio can save engineers hours of creating tracks, busses, and inserting effects on a typical session.

However, these templates can only take the session so far. No matter how consistent and reproducible the recording session's configuration may be, every session is different. Different talent performs at different volumes and has different timbral qualities. Presets typically have a fixed setting and do not automatically adjust themselves based on the source material being fed to them. This means that the experienced engineer who knows what to listen for will have to make adjustments to each track to achieve the desired sound, and the inexperienced engineer's sound will be defined by the fixed preset and not the ideal sound for the source material.

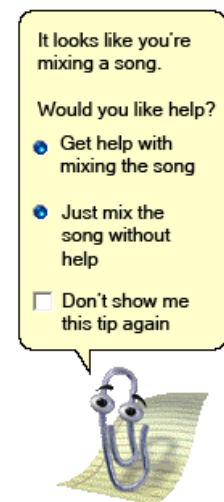
The field of music information retrieval (MIR) is developing tools that may help with this. MIR is devoted to instilling a deeper understanding of musical and audio content by computers. Using clever combinations of ideas from fields as diverse as computer science, music theory, statistics, speech recognition, and even biology, the

burgeoning field of MIR has already led to fairly robust music classification, recommendation, and identification systems.

Recording software stands to benefit greatly from the techniques developed for this field. A system that extracts relevant features from the incoming audio signal and adjusts its presets based on those features would represent a dramatic leap over the static presets commonly found in recording software. Years ago, the notion of a computer making decisions based on such things as recording style and genre seemed far-fetched; now, thanks to great leaps made in timbral similarity and music identification and classification tasks, this does not seem so far off.

Personal Motivation

Many tech-savvy people or experienced recordists are put off at the mere prospect of letting a computer make decisions for them. Some experienced computer users to whom I mentioned this project reacted as if I was trying to develop a ‘Clippy’ system for recording software, referring to an ‘office assistant’ system developed by Microsoft for its Office suite of products that was widely maligned for the often unhelpful suggestions it would make based on basic contextual information it gleaned from what the user was typing¹. And the mere mention of a system that would “make intelligent decisions on the audio signal, greatly reducing ‘knob-twiddling’ and repetitive tasks” on a popular online forum for



¹ http://en.wikipedia.org/wiki/Office_Assistant

recordists prompted rampant skepticism and at least a few hostile responses (“Who asked them to do this anyway?????????”).²

It is not the goal of this line of research to replace recordists or the human ear, nor is it to produce a ‘Clippy’ for ProTools. When I began looking into ways in which MIR could be brought into recording environments, I took a great deal of inspiration from a popular web development framework called Ruby on Rails.³ Ruby on Rails became popular very quickly due to the relatively low barrier to entry to creating powerful data-driven websites quickly and easily, as well as providing a robust framework for very advanced development and extensibility as well. One principle that is central to Ruby on Rails and is particularly interesting in my motivation for this ‘smart studio’ research is the notion of ‘scaffolding’. A scaffold takes a data structure and provides a fully usable web-based interface based on that structure, including the ability to perform common functions with the data defined by that interface (‘CRUD’, or Create, Read, Update, Delete). This scaffolding can be good enough to have a fully functional website based off of it, but the real purpose of the scaffolding is to allow for the quick creation of a website with basic functionality and then for the scaffolding to be incrementally removed as the site develops. The application of this paradigm to a recording environment makes sense. Having a computer set up a recording session and a workable mix as a starting point using a combination of templates and information gleaned from the audio tracks in the session could be a great convenience to a recordist. It is because of this that the working title of this project has become ‘Recording on Rails’.

² <http://recforums.prosoundweb.com/index.php/t/19437/2497/>

³ <http://rubyonrails.org>

Structure

This paper presents a framework for and an implementation of the automatic mixing of sounds based on a desired target sound. The framework combines timbral similarity measures with an optimization algorithm to accomplish the mixing task. The timbral similarity measures are inspired heavily by Elias Pampalk's work, and the genetic optimization algorithm used was derived from a MATLAB library developed by Christopher Houck at North Carolina State University.

The first section describes the components of this framework, followed by the details of its implementation in MATLAB. Results from a number of tests performed using the system are analyzed, and finally, future work and potential improvements to this system are discussed.

A Framework for Automatic Mixing

Overview

As an initial step towards the 'Recording on Rails' vision of a smarter studio, a framework was developed for the automatic mixing of sounds in a multitrack recording session. The approach taken is unique in that it does not directly analyze the features of the individual tracks in the mixing stage; rather, it mixes by calculating the similarity of the mixes it creates to a target sound. The mixing algorithm adjusts coefficients that are applied to each track. This implementation takes into account setting the volume levels of each track, but the basic concept could be followed and such features as panning could easily be incorporated.

This 'top-down' approach was chosen over the perhaps more intuitive approach of analyzing each track in a recording and making the mix based on a track-by-track analysis for a number of reasons. This approach is elegant in that it doesn't require knowledge of the different types of tracks that may be encountered in the mixing task. This means that a completely unknown set of sounds may be mixed using only the target sound. It also demonstrates the re-applicability of a technique originally developed for music classification as a tool to help in the creation of music.

The most immediate application of this framework is in mix recreation. The quantitative tests performed thus far have sought to find how close this implementation can come to recreating a known mix. Once all elements of the system are in place and controlled, the more subjective task of mixing a number of tracks to sound like a novel target sound can be addressed.

The mixer implementation is depicted in **Figure 1**. A gain coefficient (α) is applied to each track and then all the tracks are summed together, just as in a recording environment:

$$\sum_{i=1}^n track_i * \alpha_i$$

An optimization algorithm is then used to figure out the best gain coefficients to make the mix sound as close to the target as possible.

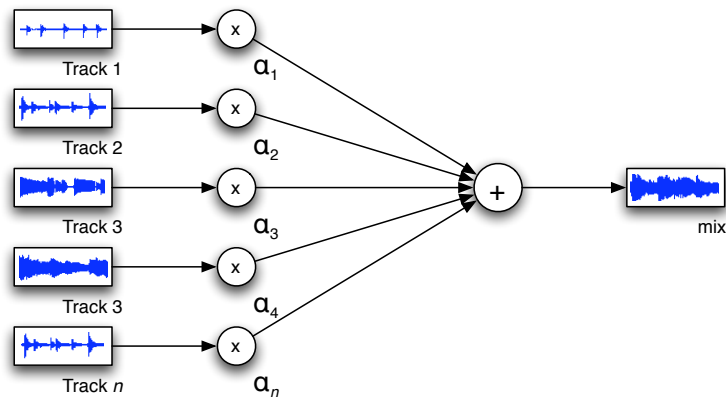


Figure 1: Mixer implementation.

Cost Function

Possibly the most important piece of the automatic mixing project is the cost or fitness function. This is the part that allows the optimization algorithm to determine whether one signal sounds more or less like another in a quantifiable manner. In this case, the distance between a mix and the target sound is measured. This distance is in

turn used by the optimization function to adjust the mix accordingly and to make the mix sound more like the target sound.

This problem can be viewed as an extension of the timbral similarity problem that is being addressed throughout the MIR field. Timbral similarity is being used in everything from music recommendation engines to music identification systems and is a very popular problem in the MIR community. A good overview of the state-of-the-art in timbral similarity measures can be found in Elias Pampalk's PhD thesis (Pampalk, 2006). There is also an annual competition among the MIR community to evaluate new approaches to music information retrieval related tasks such as timbral similarity called the Music Information Retrieval Evaluation eXchange (MIREX).⁴

In "On the Evaluation of Perceptual Similarity Measures for Music", a number of measures are compared for their timbral similarity capabilities (Pampalk, 2003). Interestingly, one of the least computationally expensive measures, the spectral histogram (SH), outperforms all of the others that were tested in music classification tests.

The SH is a histogram of the number of times loudness levels have been exceeded across frequency bands. The frequency bands are defined by the critical bands of the Bark scale and the loudness levels are measured in sones, both of which are psychoacoustic measures: the Bark scale is split up into frequency bands that correspond to the critical bands of human hearing (**Figure 2**), and the sone is a measure of perceived loudness.

⁴ http://www.music-ir.org/mirex/2008/index.php/Main_Page

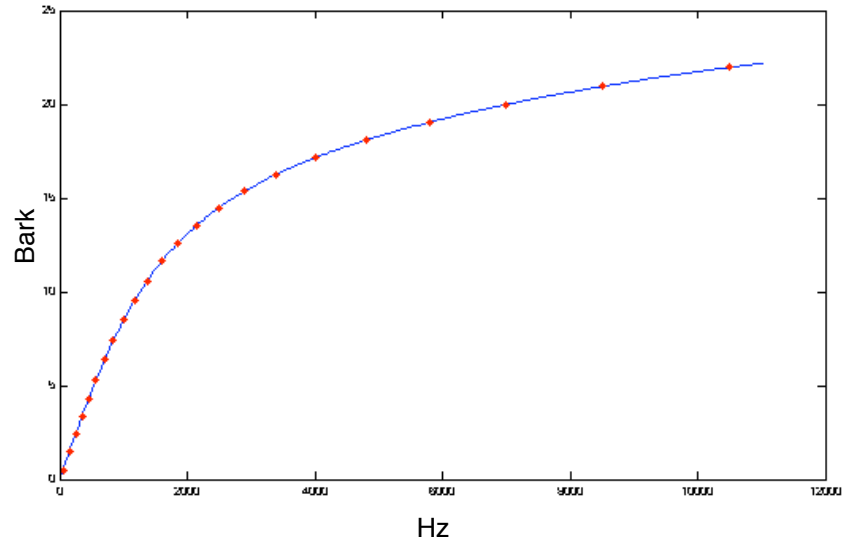


Figure 2: The Bark Scale

Various representations of a piano recording are shown in **Figure 3a**: the waveform of the PCM signal, the overall loudness in sones, and the loudness in sones across the critical bands of the bark scale. The spectral histogram representation of that recording is shown in **Figure 3b**, with the Bark bands across the vertical axis and the loudness levels on the horizontal axis.

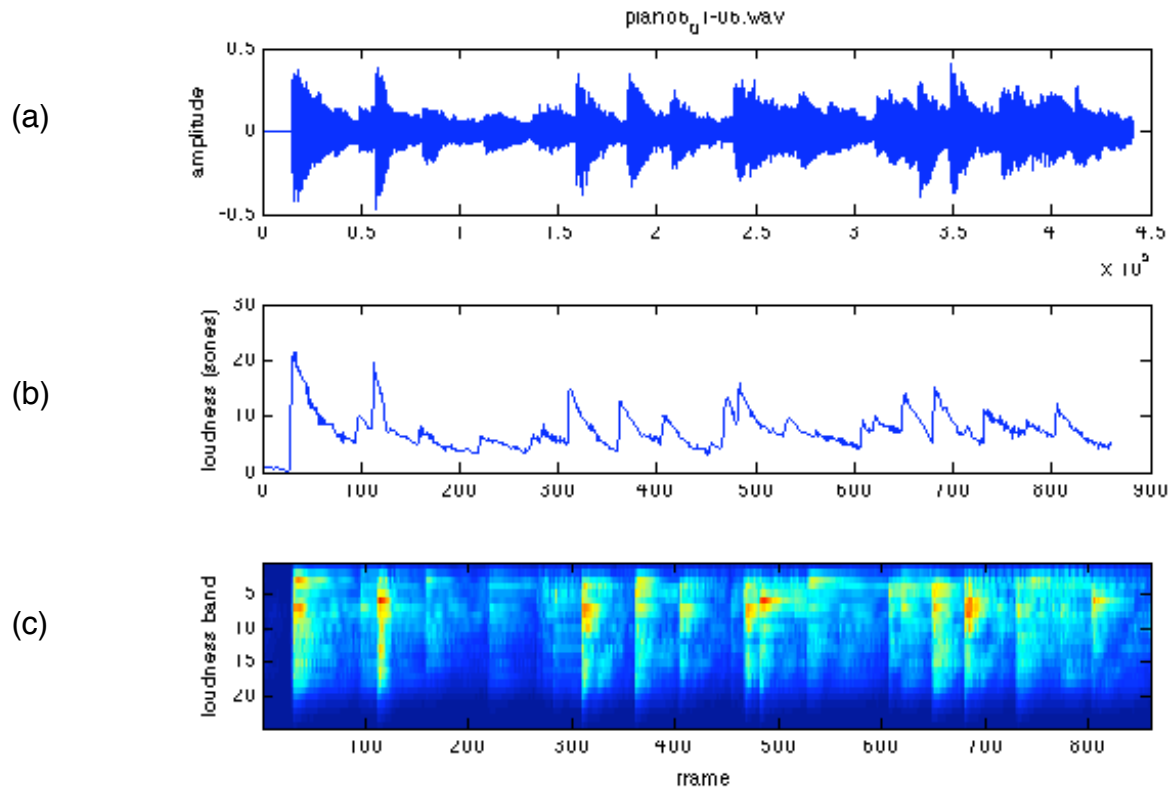


Figure 3a: Three representations of the same piano signal: (a) waveform; (b) loudness in sones; and (c) loudness in sones across the critical bands of the Bark scale.

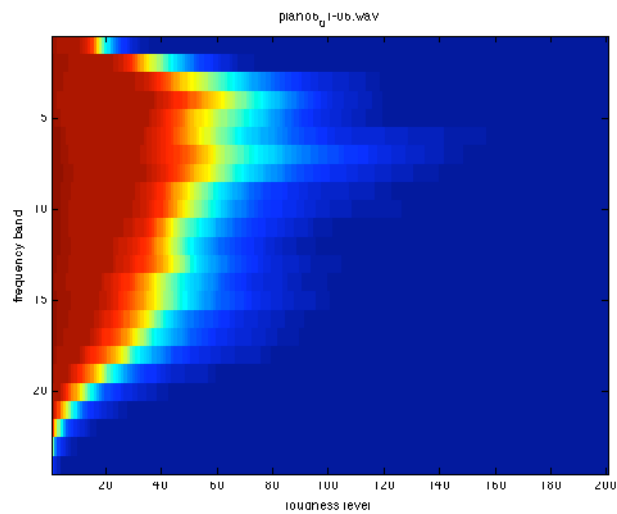


Figure 3b: Spectral Histogram (SH) of a piano signal.

To calculate the distance between SHs, the Euclidean distance is taken:

$$\sqrt{SH(mix)^2 - SH(target)^2}$$

Other measures such as the Mahalanobis distance may prove to be useful and merit further investigation. In “Fuzzy Audio Similarity Measures Based on Spectrum Histograms and Fluctuation Patterns”, Klaas Bosteels evaluates a number of fuzzy similarity measures to calculate the distance between SHs. Some implementations proposed in Bosteels’ paper reduce the computational time needed and perform almost as well as the Euclidean distance between SHs (Bosteels, 2007).

The Spectral Histogram implementation utilized in this project comes from Elias Pampalk’s MA Toolbox (Pampalk, 2004). The parameters passed to the functions for extracting the sones and the spectral histogram from the signal (*ma_sone* and *ma_sh*, respectively) are the defaults for them, with two exceptions: one, the sampling rate (*p.fs*) matches the sampling rate of the recordings (44100 in these tests, rather than the default of 11025) and two, the histogram resolution (*p.hist_res*) is set to 200. The histogram resolution is typically set to 25 or 50 for music classification tasks; the higher histogram resolution in this case, however, allows for more accurate mixing.

Cost Function Evaluation

An evaluation function was designed to determine the viability of a cost function in an automatic mixing system. Two tracks were randomly selected from a collection of instrument tracks that would be typically found in a multitrack recording session. Each

track was assigned a random gain value that was within a reasonable dynamic range of a gain that would be applied to the track while mixing-- for example, -20dB to +10dB. This combination of the two tracks was mixed using the summing algorithm described in **Figure 1** and stored as the target sound.

A mesh was then generated by iterating over all the possibilities of gain values in 1 dB increments through the dynamic range for both tracks. This mesh was used to determine first if the cost function can accurately identify the gain values used to generate the target. If the global minimum of the mesh coincides with target's gains, then the cost function is considered as a candidate for use in the automatic mixing system.

Cost Function Development

The Euclidean distance between spectral histograms produced smooth gradients when run through the evaluation function. While the SH sometimes would uniquely identify the target, the slope leading to the target was typically not steep. Two characteristic meshes generated by the SH are shown in **Figure 4**.

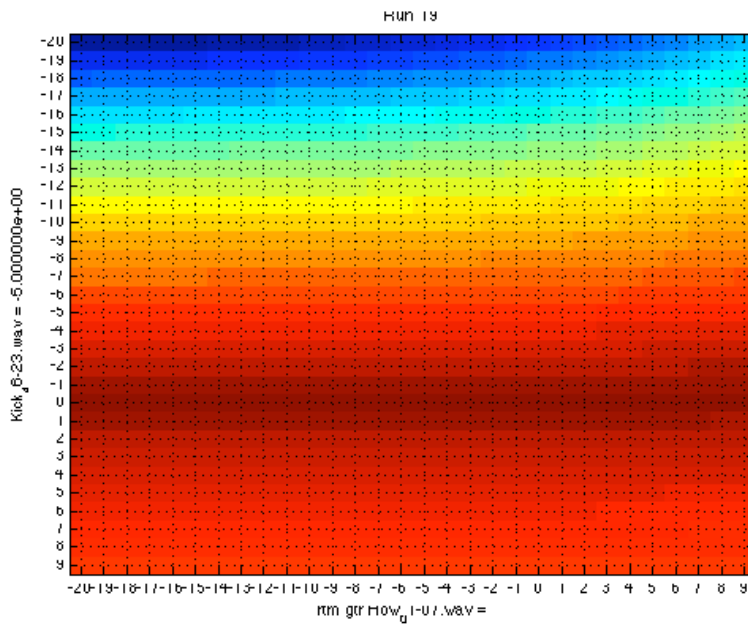
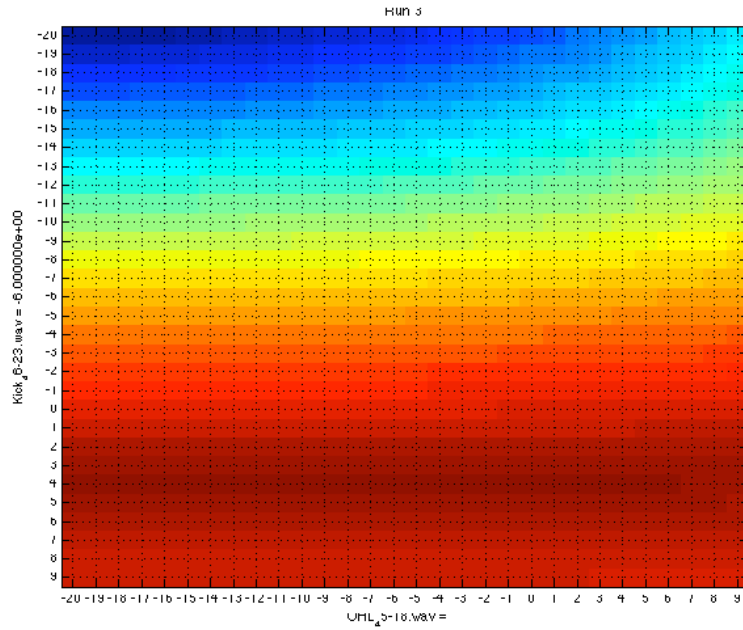


Figure 4: Mesh Produced by Unmodified Spectral Histogram.

Note the nearly straight dark red line that extends through these meshes. This is not desirable because optimization algorithms typically function best when there is a smooth slope leading towards the solution. This line was found to be caused by

normalization that is performed in the calculation of the SH. While this normalization is useful in comparing pieces of music that may have different overall loudness levels or comparing segments of different durations, the loudness data is desirable in calculating the gains for each track.

Simply removing the normalization in the SH resulted in the spectrogram quickly becoming saturated, as can be seen in **Figure 5**.

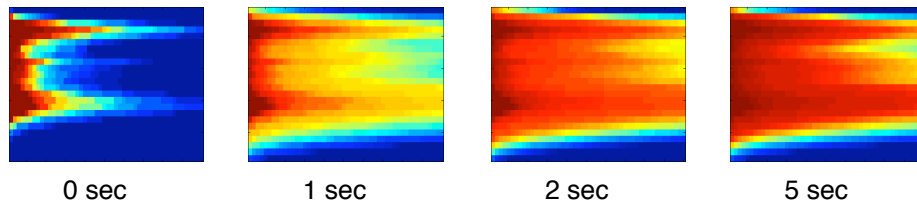


Figure 5: Non-normalized SH over time.

However, by scaling the SH by the overall mean loudness of the mix, the SH maintains its overall shape and still carries loudness data. The overall loudness of the track is also calculated in sones. This scaling proved to be what was needed to get the gradients to come to a nice peak at the target (**Figure 6**).

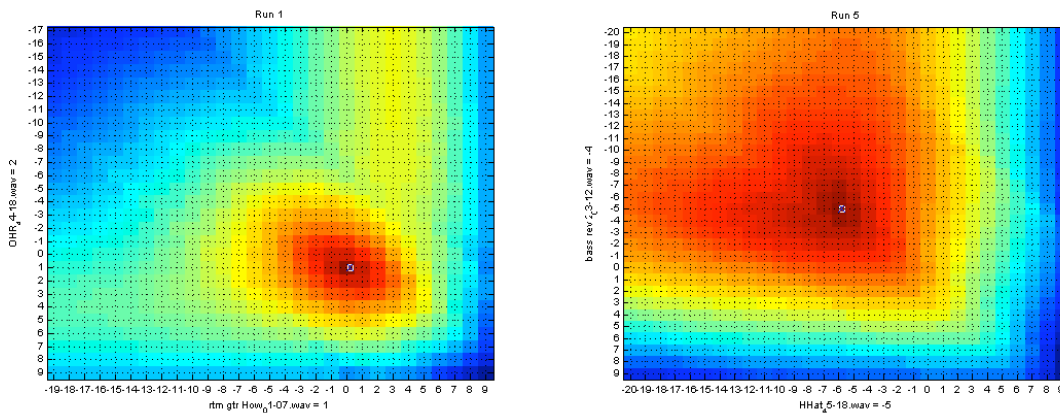


Figure 6: Mesh produced by SH scaled by mean loudness of tracks.

Note that the genetic optimization algorithm used in the automatic mixing task is a maximization algorithm, so the cost of the Euclidean distance between the mix's SH and the target's SH is subtracted from zero. A cost of zero means that the mixes are exactly the same.

Genetic Optimization

Genetic optimization is a combinatorial optimization technique modeled after the biological field of genetics. Genetic algorithms (GAs) model their systems just like genes in a biological system, and the genes undergo a number of transformations like mutation and crossover just as they do through sexual reproduction in the living world. Genetic optimization has been shown to be very successful at navigating unwieldy search spaces with many local minima that would confuse normal hill-climbing optimization algorithms (Houck, 1996:2).

Similar work to the automatic mixing problem has been performed on estimating the parameters for FM synthesis. In a paper published in the *Computer Music Journal*, Andrew Horner et al use a genetic algorithm to adjust the parameters of FM synthesis to closely match the characteristics of an acoustic signal (Horner, 1993). Tan and Lim successfully found parameters for double FM synthesis using genetic annealing, a variant on genetic optimization (Tan and Lim, 1996). In both groups' experiments, they compared the harmonics of the acoustic sounds that they were trying to model to the harmonics generated by the FM synthesis as the fitness function. This is analogous to

the automatic mixing task in that both papers use a GA to optimize the distance between a system's output and a known target sound.

A toolbox for genetic optimization was used in this implementation. The "Genetic Algorithm Optimization Toolbox" (GAOT) implements a genetic optimization function as well as a number of other functions used by the genetic algorithm for mutation, crossover, and selection. It also provides a framework for customizing each of these functions as needed (Houck, 1996).

The GA implementation adapted from Houck's paper and used in these experiments is presented in **Figure 7**. Each colored block represents a gene; each sequence of genes represents an individual. The fitness of the individual is stored in the last element of its genetic code; the fitness value improves as it approaches zero. A population is comprised of all of the individuals in one generation.

```
Generate initial population  $P_0$  of  $N$  individuals;
calculate cost of each individual
let  $i = 0$ 
do until termination criteria is met
     $P'_i = \text{selection function}(P_{i-1})$ 
     $P_i = \text{reproduction function}(P'_i)$ 
loop
```

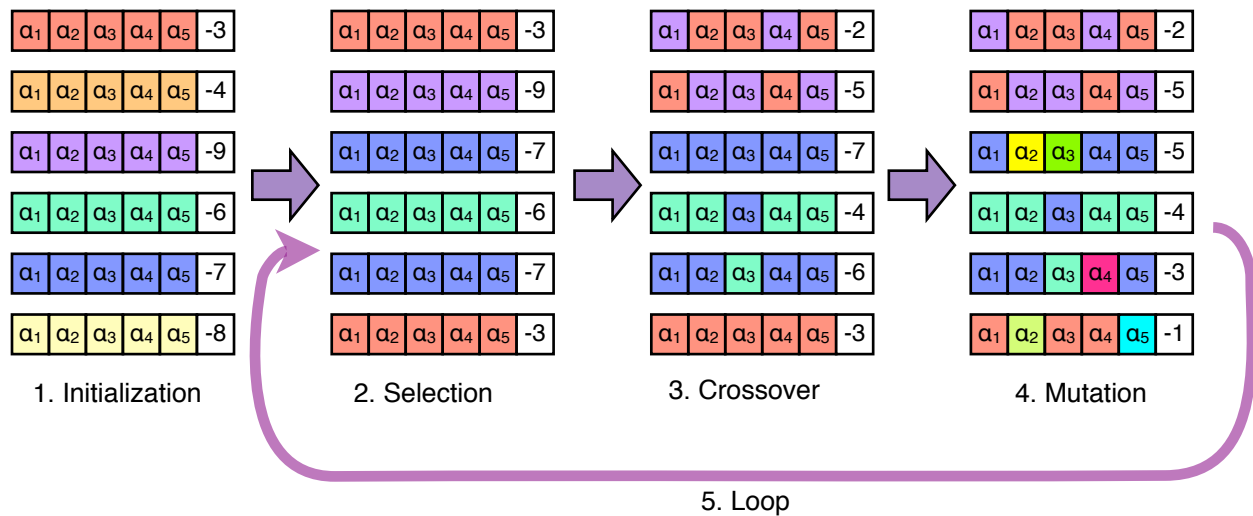


Figure 7: Genetic Algorithm Overview.

Each component of this algorithm is discussed below.

Representation

Just like how the base pairs in a sequence of DNA make up genes that define an organism's attributes, a means of encoding information about a system is needed to serve as the genetic algorithm's 'DNA'.

For the first step of this automatic mixing project, a monaural mix is optimized to sound similar to a target mix using gain adjustments on each track. The gain was defined in voltage change that would be applied to each track; i.e. a gain value of 1 would indicate no gain change and a gain of 0.5 would indicate a -6dB gain applied to the track.

A collection of multiple tracks is thus represented as a sequence of gains to be applied to each track. For every iteration of the genetic algorithm, the fitness of each individual in the population is calculated by applying the gains to each track and then

summing the tracks together, as depicted in **Figure 1** of this paper. The cost function is then utilized to calculate the distance between the resulting mix and the target mix; this fitness is recorded and used by the GA's selection function.

Initialization

Initialization in a GA typically assigns all the individuals of a population to random values within a known range. This is partly responsible for the ability of the GA to search out large and uneven spaces for the globally optimal solution. In the case of the automatic mixing project, the bounds of the variables define the dynamic range of the signal; for example, they could be defined as [0.001 1.5] to allow for a -60dB to +3.5dB range.

While this random initialization allows for a wide-spanning search, it can also lead to unpredictable results. A few factors could be used to potentially improve the initialization of the population. While this somewhat goes against the 'black box' approach taken by this project, some knowledge of the tracks could be taken into account to help in initialization. For example, each track could be scanned for its loudness, either by simply scanning for its peak or by using a basic windowed RMS or perceptually weighted loudness function. The dynamic range for initialization for that track could then be adjusted accordingly. However, this approach does not take into account stylistic decisions made by the recordist to allow certain tracks to be more prominent than others while avoiding clipping. This approach therefore may be no better than a simply random initialization.

A more ambitious initialization technique would be to incorporate some sort of track identification or classification scheme. A robust system for accomplishing this would require a large amount of prior knowledge about different mixing styles and typical track settings and levels; such a system is discussed in the 'Future Work' section of this paper.

Evolution

An initial population is created of a predetermined size. The genetic optimization function then takes that population and performs a number of operations on it to produce new generations of viable populations until a termination criteria is reached. The termination criteria can be defined as a maximum cost being achieved, evolving for a number of generations without seeing any improvement, or evolving for a specified maximum number of generations. Since the maximum cost achieved by the GA is dependent on a wide array of factors, in particular the number of tracks being mixed, an ideal maximum cost varies from task to task and as such a fixed number of generations was used instead.

Reproduction functions: Crossover and Mutation

For each generation, the GA applies crossover and mutation operators to members of the population at a specified rate. In biology, crossover and mutation are responsible for introducing variations in offspring.

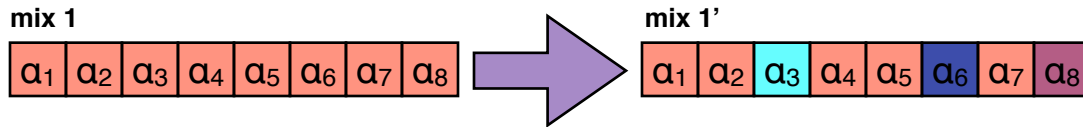


Figure 8a: Mutation.

Mutation (**Figure 8a**) will randomly alter one individual's genes to be within specified bounds. Depending on the mutation function, the mutation can be selected from a uniformly random distribution within those bounds or can be selected from a probability distribution. The latter option can exploit the robust global search features of a typical genetic algorithm; as the number of generations reaches a maximum number of generations, the amount of change introduced by the mutation can decrease so a search for a local minimum rather than the global minimum is performed.

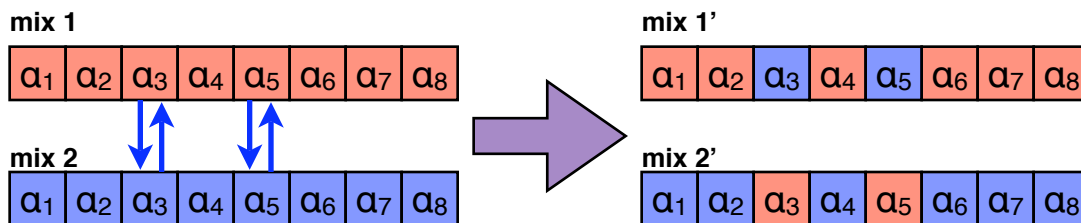


Figure 8b: Crossover.

Likewise, crossover (**Figure 8b**) provides for variation in offspring by transferring portions of one parent's genetic material to the other parent's. A crossover function can randomly transfer genes and gene sequences of varying length between parents, or information about each parent's fitness can be used as a heuristic for determining which parent provides the genetic material for crossover and how much material should be used in the crossover.

Selection

A genetic algorithm works by employing a 'survival of the fittest' algorithm. Crossover and mutation result in offspring with variable viability; it is the job of the selection function to choose which offspring survive until the next generation. The selection function ranks the individuals in a population based on their fitness level, which is calculated by the cost function described above. It then decides which members survive until the next generation.

Test Corpus

A significant hurdle in developing this project was developing a corpus for testing. The problem is twofold: one is the lack of freely-available multitrack recording sessions. This is compounded by the fact that recordists are notoriously secretive and protective of their works; many like to consider what they do as their personal 'trade secrets' and are loath to share their sessions with anyone even for research purposes. A far-reaching call to professional recordists and mastering engineers received no responses.

A few professional sample recording sessions were found. One is a number of multitrack sessions that Trent Reznor has released over the years of Nine Inch Nails' music in various formats.⁵ Another is a number of demonstration files that ship with ProTools software.⁶ These sessions as well as a number of personal recordings that I have accumulated over the years were used as the corpus for the basic development of this algorithm.

⁵ www.nin.com

⁶ www.digidesign.com

The other issue, however, is getting these recording sessions into MATLAB. MATLAB provides for the basic reading of WAV files but makes no provisions for interoperability with any multitrack recording software. Furthermore, multitrack recording software packages use proprietary formats that define the exact placement of files in a recording session. The only 'open' formats I could find that are interoperable with ProTools and Logic⁷ are OMF and AAF, both of which are typically used for transferring video files with multitrack audio mixes between systems⁸. No MATLAB library for reading these formats was found and the effort to develop such a library would have been too great an effort for this task, so all of the data used in the development of this project was prepared by hand.

To export a multitrack session for mixing within MATLAB using ProTools, each track was either bounced individually or bussed with effects and then bounced in real-time to disk, which can be a very time-consuming process. Logic Pro has a feature to export all tracks as audio files offline; that saved some time in the file preparation process.

One very important facet of this is that since MATLAB is simply reading in audio files with no timecode information, the files must be synchronized with one another-- they must start at exactly the same point in time. Multitrack recording software can place fragments of sounds in arbitrary positions on the timeline, saving a great amount of storage space over having a full-resolution audio file that spans the duration of the project for each track.

⁷ The Nine Inch Nails files were released in Garageband format, which does not directly support OMF/AAF export, but Logic Pro can read Garageband files and then export the files as OMF/AAF.

⁸ <http://www.aafassociation.org/>

The files were all exported to WAV format and converted to 16-bit resolution and a sampling rate of 44100 Hz. Segments of multitrack sessions ranging in size from four tracks to sixteen tracks were exported for testing.

Obviously this field of research could benefit greatly from an open repository of recordings as well as cross-platform libraries for interpreting those files. Hopefully this dearth of data will change as more people become aware of this effort and collaborate on sharing recordings as well as developing interoperable file formats.

Tests

Each run of the automatic mixing algorithm prints out the cost of the final mix as well as the best gains found for each track.

A test was created to determine the effect of number of tracks and total number of generations on the mixing algorithm's efficacy.

A number of tracks was randomly selected from a 10-track multitrack rock recording. Twenty random mixes containing four randomly selected tracks and twenty random mixes containing eight randomly selected tracks were produced. The gains used to generate each mix were recorded and stored as the target gains. Each of these mixes was then used as the target for the automatic mixing system. The same combination of four or eight tracks as the target was fed into the system and allowed to run for a number of generations.

To better visualize how the optimization algorithm was performing, a function was created that displays the gains for each track each time an improved optimal population was found by the GA. A bar graph display is used to simulate how the faders would be

positioned in a recording console. A movie is generated from those graphs to show how the optimization functions over time.

The genetic algorithm was called with the following parameters:

```
ga(bounds, 'shDistGA', [], b.initPop, [1e-6 1 1], 'maxGenTerm',  
    b.generations, 'normGeomSelect', [0.08],  
    'heuristicXover', [b.xoverFreq b.heuristicRetries],  
    'multiNonUnifMutation', [b.mutFreq b.generations 2]);
```

The parameters are as follows:

bounds: A matrix of size [2 x number of tracks], each element of which contains the minimum and maximum gain for each track. These values are all set to [.001 1.5], or -60dB to +3.5dB.

shDistGa: The fitness / cost function explained above; calculates the Euclidean distance between the current track's modified SH and the target's modified SH.

b.initPop: A data structure containing the initial population. The population had a fixed size of 25 for all tests. Each individual of the population is represented as:

[Gain₁ Gain₂ Gain₃ ... Gain_n Cost]

The cost of the individual is stored in the *number of gains + 1* position; the GA uses this cost in the selection function.

maxGenTerm: The termination function used; simply terminates after the maximum number of generations has been reached.

b.generations: The number of generations for which the GA will run.

normGeomSelect: A selection function provided by the GA toolbox (GAOT). Ranks and selects individuals based on the normalized geometric distribution of their costs.

heuristicXover: A crossover function provided by GAOT. This function utilizes the fitness information of the parents by attempting to create a child by crossing a random number of genes from the fitter parent to the less fit parent. It is called **b.xoverFreq** times each generation, which is fixed at 50% ($b.populationSize / 2$) for these tests. It attempts to generate a viable child **b.heuristicRetries** times; if no viable offspring is produced, the fitter parent is returned. **b.heuristicRetries** is fixed to 1 for these tests.

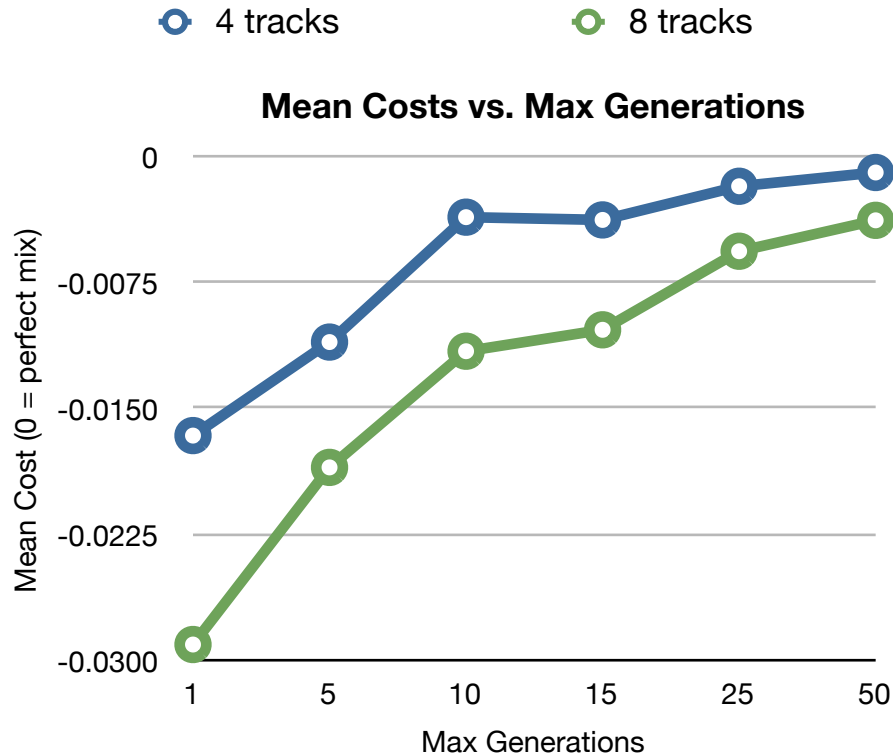
multiNonUnifMutation: A mutation function provided by GAOT. This function mutates a randomly selected number of genes within a member of the population; it is applied to **b.mutFreq** members of the population. **b.mutFreq** was set to 50% of the population size ($b.populationSize / 2$) for these tests. This mutation function is useful in that it initially starts off generating large mutations and then as **b.generations** is approached, the size of the mutation gets smaller. This takes advantage of the GA's ability to find a global minimum relatively quickly and then provides a local search operator as the mutations become smaller.

Results

As expected, adding more tracks to a mix degrades the performance of the mixing algorithm. Furthermore, increasing the maximum number of generations that the mixing algorithm is allowed to run increases its performance. Mixing performance versus number of tracks and maximum number of generations is demonstrated in the following figures.

Figure 9: Mean Cost vs. Max Generations for 4- and 8-track Recordings

Max Generations	4 tracks	8 tracks	Performance difference between 4 and 8 tracks
1	-0.0165493	-0.0289864	57.09%
5	-0.0109831	-0.0184428	59.55%
10	-0.00354805	-0.0115269	30.78%
15	-0.00371195	-0.0102529	36.20%
25	-0.00170817	-0.00557563	30.64%
50	-0.0008888	-0.00373602	23.79%



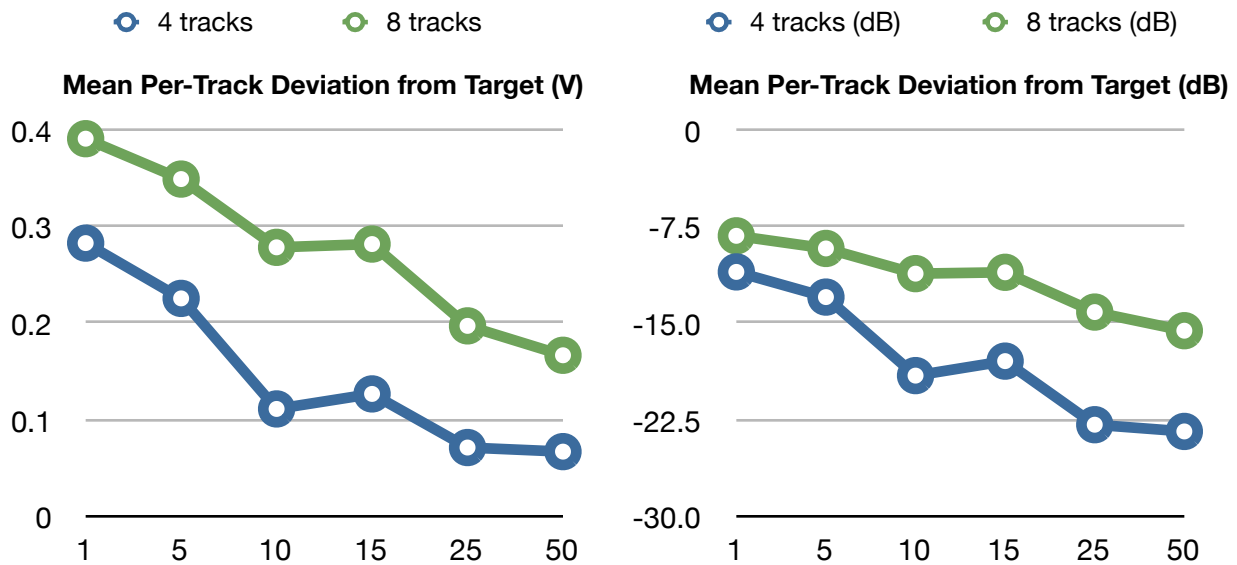
As the number of generations increases, the disparity in performance between the smaller set of tracks (4) and the larger set of tracks (8) decreases. The performance of the GA to level off as the number of generations increases; at this point, the region of the global minimum in the search space has been found and the GA is exploring that area to find the optimal mix.

To observe the correlation between the cost function and the actual deviation from the mix, the mean deviation of each track's gain value from the target gain value was calculated in both voltage and decibels. Note that unlike the cost function that characterizes improvements by increasing towards zero, the ideal deviation between

the target gains and the mix's gains with regards to voltage should decrease towards zero, and the deviation in decibels should be continually decreasing below zero.

Figure 10: Mean Per-Track Deviation from Target

Max Generations	4 tracks (V)	4 tracks (dB)	8 tracks (V)	8 tracks (dB)
1	0.284044	-10.9322876	0.392044	-8.13330377
5	0.226829	-12.8860284	0.350266	-9.11204034
10	0.11248	-18.9784938	0.279655	-11.0675482
15	0.128018	-17.8545792	0.283039	-10.9630744
25	0.0725436	-22.7880179	0.198411	-14.0486851
50	0.0683144	-23.3097548	0.167987	-15.4944865

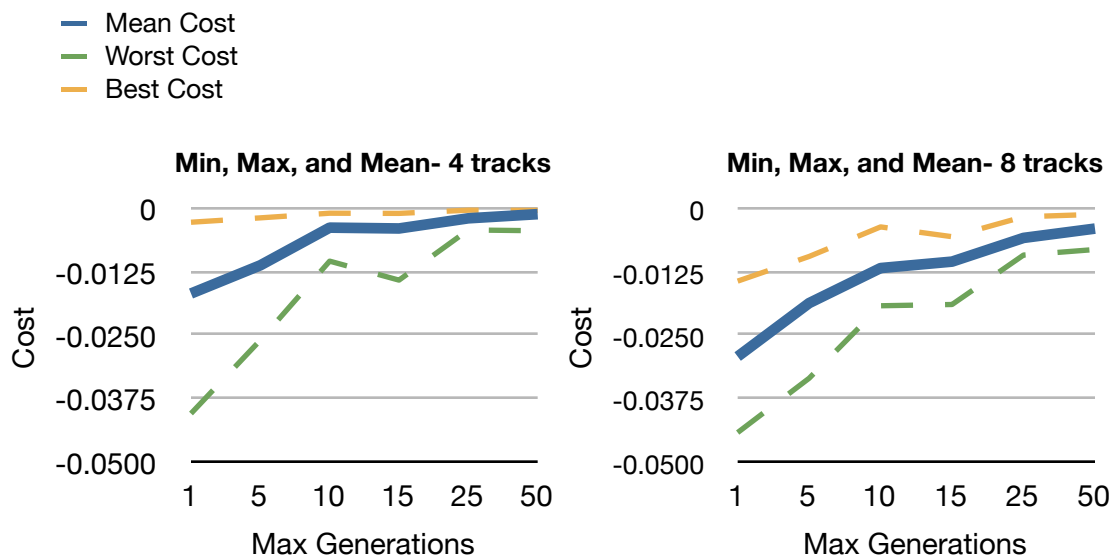


The per-track deviation shows a general similarity with the cost function's performance in that a significant improvement is shown as the maximum number of

generations increases from one to ten and then levels off in performance as the number of generations increases beyond that. There is curiously an increase in the deviation from the target gains between a maximum of ten generations and fifteen generations. Looking at the mean cost alongside the maximum and minimum costs found over the twenty test runs gives more insight into why this is happening:

Figure 11: Mean Cost vs. Maximum and Minimum Costs found over 20 Runs

Max Generations	Mean Cost 4 tracks	Max Cost 4 tracks	Min Cost 4 tracks	Mean Cost 8 tracks	Max Cost 8 tracks	Min Cost 8 tracks
1	-0.0165493	-0.0402356	-0.00248121	-0.0289864	-0.04399	-0.0141027
5	-0.0109831	-0.0257545	-0.00161785	-0.0184428	-0.0332733	-0.00923919
10	-0.00354805	-0.0101597	-0.000700288	-0.0115269	-0.0189671	-0.00340315
15	-0.00371195	-0.0139062	-0.000735112	-0.0102529	-0.0187164	-0.00533533
25	-0.00170817	-0.00400355	-0.000103698	-0.00557563	-0.00895949	-0.00140457
50	-0.0008888	-0.00416336	-0.000152721	-0.00373602	-0.00786479	-0.00093043



When run for fewer generations, the quality of the results from the automatic mixing vary widely. This can be attributed to the random initialization of the gains for each track; sometimes, the initial population generates an individual that is very close to

the target, and other times it takes a number of generations to get close to the global minimum of the search space. A more refined initialization system could potentially help minimize this disparity.

Another test was run to determine the optimal frequency at which the mutation function should be called. One would expect that there would be a breaking point beyond which the mutation rate would be too high and would degrade the system's performance. However, tests that were run interestingly showed that increasing the mutation frequency even up to one hundred percent of the population-- meaning that every member of the population will be mutated during every iteration of the function-- did not degrade performance and may have even improved it. With mutation set to only a few members of the population, it takes a longer number of generations before the local minimum is found, as shown in **Figure 12**:

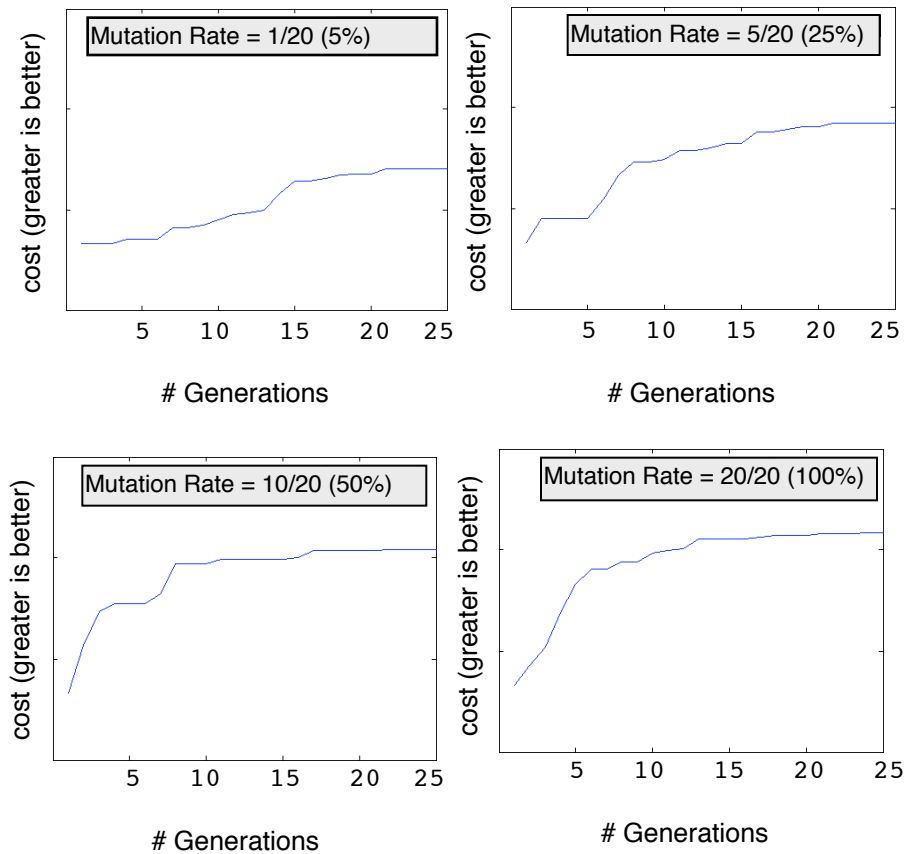


Figure 12: GA Performance versus Mutation Rate (average over 3 runs)

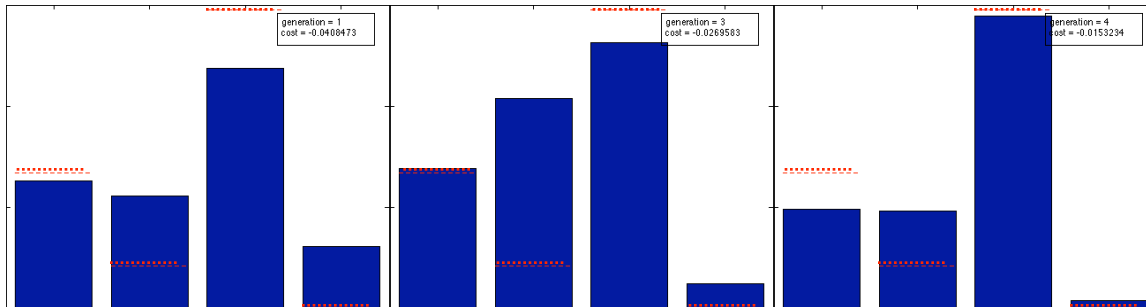
This may be due to the fact that the mutation function is being used as the main search function in the GA and therefore calling it more frequently causes a more aggressive search. The multiNonUnifMutation function being utilized also tapers its search as the maximum number of generations is approached; if another mutation function were used that does not narrow its search space in such a manner, maybe there would be a more noticeable turning point in the tradeoff between mutation frequency and performance as widely varying individuals will be generated in all generations of the genetic algorithm's run. Also, upon closer inspection it was found that the GAOT always preserves the best individual of the previous population-- the

least fit individual in a population is replaced with the most fit individual from the previous generation. This guarantees that the solution found by the GA never gets any worse than the previous generation, which means that an even more aggressive search can only improve the results.

However, increasing the mutation frequency also increases the runtime of each generation as the cost function is called every time a new individual is generated. This means that if the mutation rate is set to one hundred percent of the population, the fitness function is called an additional (number of tracks * population size) times each generation, which can drastically increase the runtime of the mixing algorithm. Because of this, the mutation frequency was set to half the population size for testing.

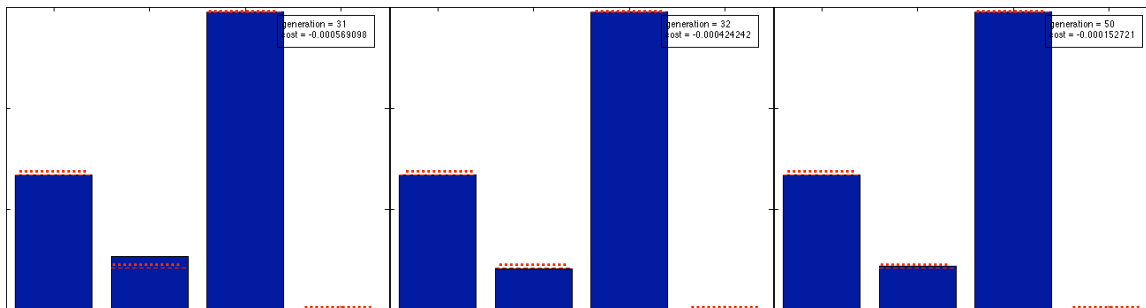
The behavior of the mixing function as well as the narrowing down of the global search space to a localized search by the multiNonUnifMutation function is apparent when watching the evolution of the mix as a bar graph. The search starts out wide as a number of combinations are explored for a global minimum; then, as the maximum number of generations is reached, the search moves in smaller increments. The best-performing situation tested (4 tracks, 50 generations) comes quite close to hitting its target, indicated by the dashed horizontal red lines in the following figures. In the first four generations, the search produces widely varying results:

**Figure 13a: Evolution of Optimization on Four-Track Recording
(Generations 1, 3, and 4)**



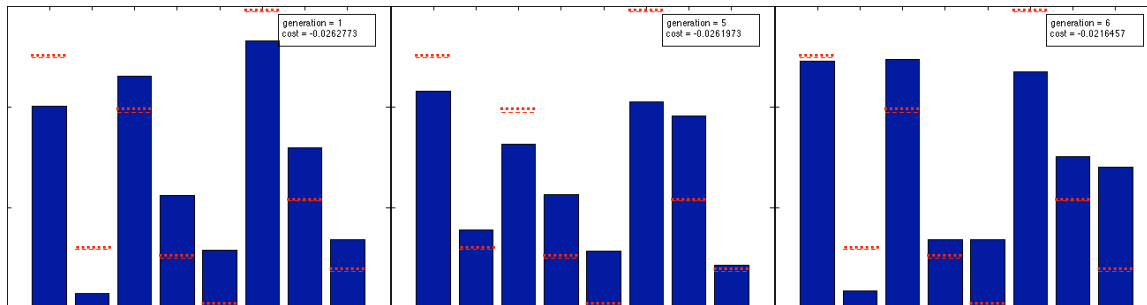
As the maximum number of generations is approached, the search has come close to the target and now fine-tunes its results. The mix from the thirty-first and thirty-second generations are not very different from the final mix from the fiftieth generation:

**Figure 13b: Evolution of Optimization on Four-Track Recording
(Generations 31, 32, and 50)**



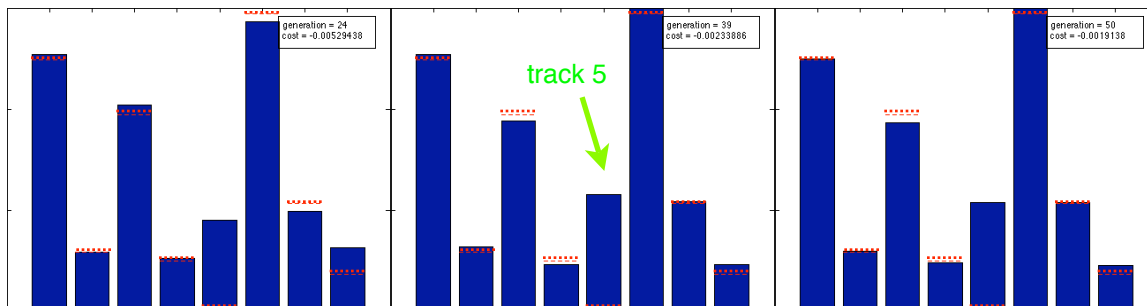
Performance of the mixing algorithm with an eight track recording was not quite as good as with the four track recording, but it is still apparent that the mixes are approaching the target. As with the four track recording, the search for the eight track mix started off on a fairly random course:

**Figure 14a: Evolution of Optimization on Eight-Track Recording
(Generations 1, 5, and 6)**



By the end, it had recreated a reasonably close approximation to the actual mix, with one notable miss:

**Figure 14b: Evolution of Optimization on Eight-Track Recording
(Generations 24, 39, and 50)**



This was a fairly common occurrence when dealing with larger numbers of tracks-- the mixing algorithm appeared to 'lock on' to most but not all of the tracks and was unable to figure out others. Track 5, identified by the green arrow in **Figure 14b**, is a snare drum track. The target gain for this track was randomly generated to be .0067, or -43 dB. The final effect of the snare drum track at -43 dB on the target mix may have been too quiet to have been perceptible by the cost function.

These results may also highlight a critical shortcoming in the approach of automatic mixing by evaluating the entire mix rather than on a track-by-track basis. The fact that track 5's gain is set much higher than the target value yet changing its value doesn't appear to have much of an effect on the overall mix's cost could be because of the percussive nature of the track. The vast majority of the energy of the snare track comes from each hit, with the energy rapidly decaying after the hit. The hits occur relatively infrequently when compared to a more continuous sound such as the strumming of a guitar or a vocal melody. Properly handling a sparse track such as this in an automatic mixing task may require a track-by-track evaluation and initialization before the mixing takes place.

One other possible impediment to picking up very quiet tracks in an overall mix is the masking model used in the score calculation. The snare drum track at -43 dB in the randomly-generated mix may have been masked by the rest of the mix, making it perceptually irrelevant to the overall mix. Modifications to the initialization and mutation functions to work in a logarithmic rather than a linear fashion to more accurately mimic human hearing may improve the chances of finding such a quiet track in the GA's search, and it also may make sense as more human-generated mixes with known targets are tested.

Future Work

This system for automatic mixing has been shown to be successful at finding the ideal gains for a preexisting mix. Extending this algorithm to mixing novel combinations of tracks is perhaps the most exciting application of this project but is also quite a different problem to qualify. Such an application would almost certainly require subjective testing to determine the system's efficacy. The system's performance in recreating existing mixes as well as in the creation of novel mixes could also be compared to mixes created by professional recordists and mixers.

Addressing panning is another logical next step for this algorithm. Panning values could be added relatively easily to the genetic code; just like the gain values, they could be represented as real numbers from $[0..1]$, with 0 referring to panning fully to the left and 1 referring to the right. This value could follow each track's gain coefficient in the genetic code and the functions that read and modify the genetic code would have to be modified to take this into account.

Tzanetakis discusses some interesting uses of panning information in his paper "Stereo Panning Features for Classifying Recording Style" (Tzanetakis, 2007). His success in identifying engineers, artists, and genres of music by using stereo panning features shows how intrinsic the use of panning can be to a mix's sound. The Stereo Spanning Spectrum (SPS) used in Tzanetakis' work describes a recording's stereo spectrum; this measure uses frequency domain information across both stereo channels. A number of features from the SPS are used in his classification tasks;

perhaps these features could be used in an extension of the automatic mixing system proposed here.

The system that was implemented in this paper creates a static mix for the entire song to be mixed; a good mix, however, is almost never completely static as gain levels are altered throughout to ensure a consistent sound as well as to emphasize certain parts of the mix to create interesting contours as the song develops. The mix provided by this algorithm could be used as the 'scaffolding' mentioned in the introduction to this paper, providing the recording engineer with a solid starting point from which he can build the rest of the mix rather than starting completely from scratch. However, this system could be combined with a windowing function to create a mix that will maintain the ideal levels for tracks throughout a song. Perhaps an initial mix could be created by using an exhaustive search for an ideal sound; the maximum number of generations could be set to be very high for this initial mixing task. Once this initial mix is found, this algorithm could then go through the rest of the song incrementally and use the first ideal mix as its initial population. This would create a sequence of gains that could define the mix for the entire song.

As the results from these experiments have shown, the genetic optimization algorithm's strengths lie in quickly finding the region of a global minimum in a large search space. However, once the algorithm has found the global minimum's region, it takes a much longer time to converge on the actual minimum. In fact, none of the tests I ran while developing this program ever converged on the exact target. This is mentioned in Houck's paper, which cites a number of papers that support this shortcoming of the GA (Houck, 1996:8). In his paper, Houck achieves very accurate

results in one optimization example by combining a GA with Sequential Quadratic Programming (SQP), available in a MATLAB toolbox. He uses the GA to find the overall region of the global minimum in a wide search space with a number of local minima, then uses SQP to find the actual minimum of that region.

An interesting complement to this project would be to address the automatic mixing problem from a 'bottom up' approach. Like typical presets that ship with recording software, this effort would address shaping a mix's sound on a track-by-track basis. One approach would be to classify each track, either broadly in terms of, say, its percussivity or its frequency spectrum; another more ambitious (but with more impressive applications from the end user's perspective) task would be to attempt to specifically identify each track and adjust the preset parameters accordingly.

This system could be used in tandem with the 'top down' approach outlined in this paper. The track-by-track approach could serve as a means of initializing each track and then the optimization function can tweak the mix until it is ideal. If this vision is implemented, it could be a complete system for creating a mix.

Conclusion

The field of Music Information Retrieval has developed a number of techniques that could vastly improve the intuitiveness, efficiency, and utility of recording software. A framework for one application of this was presented here. This system for the automatic mixing of a multitrack recording combines a MIR-derived timbral classification system with a machine learning algorithm. The automatic mixing system has been shown to successfully recreate existing mixes under a number of situations with varying degrees of success; it remains to be seen what it will take to extend this system to creating novel mixes based on unknown targets.

This paper sought to shed light on the promising integration of MIR and the recording studio. It is my hope that this will lead to a more widespread effort towards achieving a 'smarter' recording studio. The development of an open file format for recording session exchange and libraries to support such a format across a number of platforms would be helpful to both the research community as well as software companies that create professional recording software. Furthermore, a collaborative repository of multitrack recording sessions would be of great use to all those who are pursuing this line of research.

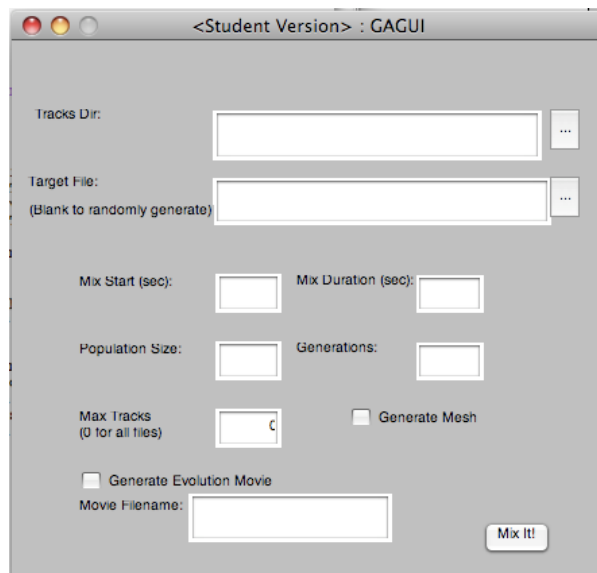
A recording studio that incorporates machine listening and MIR techniques stands to benefit both the amateur recordist as well as the professional. Not only could a smarter studio save hours of time, but it will allow you to focus on what matters most: creativity.

Code

All code developed for this project is on the companion CD-ROM to this document and will be released as a MATLAB toolbox to encourage further pursuit of automatic mixing. The toolbox will be available from New York University's Music Technology Research homepage:

<http://www.nyu.edu/projects/mtr/>

The example tracks used in the development and testing of this algorithm will also be provided, as will a basic GUI to allow for simple usage of the mixing algorithm.



The code used in this project requires both Elias Pampalk's MA Toolbox and Chris Houck's GAOT toolbox, both of which are freely available for download online.

MA Toolbox: <http://www.pampalk.at/ma/>

GAOT: <http://www.ise.ncsu.edu/mirage/GAToolBox/gaot/>

Works Cited

- Bosteels, Klass et al. "Fuzzy Audio Similarity Measures Based on Spectrum Histograms and Fluctuation Patterns." International Conference on Multimedia and Ubiquitous Engineering April 2007: 361-365.
- Horner, Andrew et al. "Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis." Computer Music Journal 17.4 (2003): 17-29.
- Houck, Christopher et al. "A Genetic Algorithm for Function Optimization: A Matlab Implementation." (1996).
- Pampalk, Elias et al. "On the Evaluation of Perceptual Similarity Measures for Music." Proceedings of the 6th International Conference on Digital Audio Effects (DAFx'03) 8 (2003).
- Pampalk, Elias. "A Matlab Toolbox to Compute Music Similarity from Audio." (2004).
- Pampalk, Elias. "Computational Models of Music Similarity and their Application in Music Information Retrieval." Diss. Johannes Kepler University, 2006.
- Tan, B.G. and Lim, S.M. "Automated Parameter Optimization for Double Frequency Modulation Synthesis Using the Genetic Annealing Algorithm". Journal of the Audio Engineering Society 44.1/2 (1996): 3-15.
- Tzanetakis, George et al. "Stereo Panning Features for Classifying Recording Production Style." Proc. Int. Conf. on Music Information Retrieval (ISMIR) (2007): 441-444.