Audio Quality Assurance: An Application of Cross Correlation*

Bolette Ammitzbøll Jurik
The State and University Library
Victor Albecks Vej 1
DK-8000 Aarhus C, Denmark
bam@statsbiblioteket.dk

Jesper Sindahl Nielsen
MADALGO[1] and
The State and University Library
Victor Albecks Vej 1
DK-8000 Aarhus C, Denmark
jasn@madalgo.au.dk

ABSTRACT

We describe algorithms for automated quality assurance on content of audio files in context of preservation actions and access. The algorithms use cross correlation to compare the sound waves. They are used to do overlap analysis in an access scenario, where preserved radio broadcasts are used in research and annotated. They have been applied in a migration scenario, where radio broadcasts are to be migrated for long term preservation.

1. INTRODUCTION

As part of the SCAPE audio quality assurance work, we have developed a tool called `xcorrSound`, which can be applied in a number of scenarios. The *SCAlable Preservation Environments* (SCAPE) project aims to develop scalable services for planning and execution of institutional preservation strategies for large-scale, heterogeneous collections of complex digital objects. To drive the development and evaluation of a number of key outputs from the SCAPE Project, specific real life preservation scenarios have been defined [7].

In this paper we describe two audio preservation cases. The first case is 'access to preserved radio broadcasts for research purposes'. The broadcasts are transcoded for streaming, and an *overlap analysis* is performed to provide a graphical user interface with coherent radio programs.

In the second case the radio broadcasts are to be migrated from MP3 to WAV for long time preservation purposes, and we want to perform *automated Quality Assurance (QA)* on the migrated files. We need to determine if the two audio files (the original and the migrated one) are the same with respect to their content. This scenario is the SCAPE *LSDRT6 Migrate mp3 to wav* scenario [5].

There are several ways of designing heuristics that can give some assurance that the migration process went well such as checking if the length is the same before and after the migration. But such 'trivial' measures do not take into account the possibility of just getting white noise as the migrated file, which obviously is a flaw. We will use old and well known techniques from signal processing to catch such errors and report them. The methods we present are easily scalable as well as reasonably reliable.

The algorithms presented in this paper have been implemented in the `xcorrSound` tool package available at [8]. The tool `xcorrSound` finds the overlap between two audio files. `soundMatch` is a tool to find all occurrences of a shorter wav within a larger wav. `migrationQA` is a tool that splits two audio files into equal sized blocks and outputs the correlation for each block (a_i, b_i), if a and b was the input. The tools all make use of cross correlation, which can be computed through the Fourier transform.

We first present the background for the algorithms in Section 2. Next the algorithms and their applications are described in Section 3. The scenarios are then described in Section 4. In Section 5 we present the experiments for the two scenarios and give the results along with a discussion of these. The non-technical reader should skip Section 2 and Section 3, but for those interested in the implementation details they can be found in those two sections.

2. PRELIMINARIES

The Fourier transform is used in many contexts within digital signal processing. Our algorithms rely on being able to compute the cross correlation of two functions efficiently which can be done using the Fourier transform. Cross Correlation, as the name suggests, gives a measure of how similar two waves are at all offsets (shifting one wave in time and comparing for all time shifts). The peak in the cross correlation is the offset at which the two waves have the highest similarity. This is going to be useful for our algorithms, hence we will recall the mathematical background of these.

Definition 1 (Discrete Fourier Transform). *Given a sequence of N values $x_0, x_1, \ldots, x_{N-1}$ the Discrete Fourier*

[1]Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Transform are the complex coefficients

$$X_k = \sum_{n=0}^{N-1} x_j e^{-2i\pi kn/N} \quad (1)$$

for all $k \in \{0, 1, \ldots, N-1\}$. *We will denote the Fourier Transform of* $X = \{x_n\}_{n=0}^{N-1}$ *as* $\mathcal{F}(X)$.

Straight forward computation of the Fourier transform requires $\mathcal{O}(N^2)$ arithmetic operations, but using the FFT algorithm [11] we can compute it using only $\mathcal{O}(N \log N)$ arithmetic operations.

DEFINITION 2 (DISCRETE CROSS CORRELATION). *Let* f *and* gp *be two discrete complex valued functions, the Cross Correlation is then defined as*

$$(f \star g)(t) = \sum_{n=-\infty}^{\infty} \overline{f(n)} \cdot g(n+t) \quad (2)$$

where $\overline{f(n)}$ *denotes the complex conjugate of* $f(n)$

DEFINITION 3 (DISCRETE CONVOLUTION). *Let* f *and* g *be two discrete complex valued functions, the convolution is then defined as*

$$(f * g)(t) = \sum_{n=-\infty}^{\infty} f(t-n) \cdot g(n) \quad (3)$$

Due to the convolution theorem we can efficiently compute the convolution of two waves if we can efficiently compute the Fourier Transform.

THEOREM 1 (CONVOLUTION THEOREM). *Let* f *and* g *be two discrete complex valued functions, then we have*

$$\mathcal{F}(f * g) = (\mathcal{F}(f) \cdot \mathcal{F}(g)) \quad (4)$$

Proofs of this theorem can be found in any book on Fourier Transforms or signal processing.

We want to compute the Cross Correlation between two wav files. We know that for any real valued function f, $\overline{\mathcal{F}(f)(n)} = \mathcal{F}(f)(-n)$. Let f and g be the two wav files we want to compute the cross correlation of, and $h(x) = f(-x)$. Note that f and g are wav files thus they can be considered as real valued functions. The Cross Correlation can efficiently be computed:

$$\begin{aligned}(f \star g)(t) &= \sum_{n=-\infty}^{\infty} \overline{f(n-t)} \cdot g(n) = \sum_{n=-\infty}^{\infty} f(n-t) \cdot g(n) \\ &= \sum_{n=-\infty}^{\infty} h(t-n) \cdot g(n) = (h * g)(t)\end{aligned}$$

Now we apply the Convolution Theorem by taking the Fourier Transform and inverse transform on both sides.

$$\begin{aligned}(f \star g) &= \mathcal{F}^{-1}(\mathcal{F}(f \star g)) = \mathcal{F}^{-1}(\mathcal{F}(h * g)) \\ &= \mathcal{F}^{-1}(\mathcal{F}(h)\mathcal{F}(g)) = \mathcal{F}^{-1}(\overline{\mathcal{F}(f)}\mathcal{F}(g))\end{aligned}$$

One can think of Cross Correlation as taking one of the wave files and sliding it over the other and remember what the best position was so far. Doing it in this way corresponds to computing the Cross Correlation directly from the definition which was $O(N^2)$ arithmetic operations. Intuitively we are searching for the shift that will minimize the euclidean distance between the two wav files.

3. ALGORITHMS

We have slightly different algorithms for handling the different scenarios but they all rely on efficiently computing the cross correlation of two audio clips. In our implementations of the algorithms we have used the FFTW library [13] for computing the Fast Fourier Transform.

3.1 Computing the Cross Correlation

The input to the Cross Correlation is two periodic functions f and g. When providing a discrete representation of a function as $f(0) = x_0, f(1) = x_1, \ldots f(N-1) = x_{N-1}$, it is assumed that $x_N = x_0$. Because of this, we need to zero-pad the wav files with N zeroes, such that the part that has been shifted "outside" does not contribute anything to the cross correlation value at that particular offset. See Figure 1

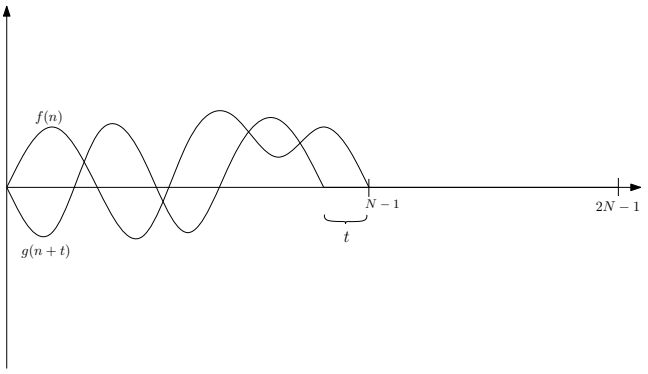

Figure 1: The function g has been shifted t steps in time, and both f and g have been zero padded. Note that from $N-t$ and onwards there is no contribution to the cross correlation, because $g(n) = 0$ for $n \geq N - t$.

If we have two functions f and g given as a sequence of N values indexed by 0 to $N-1$ then we will zero-pad them such that $f(n) = g(n) = 0$ for $n \geq N$. Now we can compute the Cross Correlation as was described in Section 2 because we have a black box (FFTW library [13]) for computing the Fourier Transform.

The Cross Correlation in itself does not provide a measure between $[0, 1]$ describing how much two wav files are alike. We want to normalize it to a value between $[0, 1]$. To do this we divide by $(f \star g)(t)$ by $\frac{1}{2}\sum_{n=0}^{N-t} g(n+t)^2 + f(n)^2$. The resulting value is always less than or equal to 1. The term we divide by can be found efficiently by storing two arrays,

one for each function. The j'th entry in the array is the sum of the first j values squared. The two prefix sums require only a few arithmetic operations to compute pr entry so this will not slow down the computation significantly.

3.2 Overlap algorithm

The input is two wav files where there might be an overlap between the end of the first wav file and the beginning of the second. We are guaranteed that if there is an overlap, it is not longer than a fixed amount of time (2 minutes). We look at the last few minutes of the first wav file and the first few minutes of the second wav file and we compute the cross correlation of these two smaller files. If there is a significant peak, we find it and report that there is an overlap, otherwise we report where the highest peak was, but that it was not very significant. The measure of significance is a value in $[0, 1]$, which is a normalisation of the cross correlation values.

This algorithm was implemented as the `xcorrSound` tool.

3.3 Quality Assurance algorithm

We have two waves and we want to determine whether they are equal or not. Let $X = \{x_n\}_{n=0}^{N-1}$, $Y = \{y_n\}_{n=0}^{N-1}$ be the two audio files. We Split these into smaller equal size pieces: $X_0 = \{x_n\}_{n=0}^{B-1}$, $X_1 = \{x_n\}_{n=B}^{2B-1}, \ldots, X_{N/B} = \{x_n\}_{n=N-B}^{N-1}$ (assuming B divides N) and likewise $Y_0 = \{y_n\}_{n=0}^{B-1}$, $Y_1 = \{y_n\}_{n=B}^{2B-1}, \ldots, Y_{N/B} = \{y_n\}_{n=N-B}^{N-1}$. Now we compute the cross correlation for each (X_j, Y_j) pair for $j = 0, \ldots, N/B$ and find the peaks. We remember the first peak, and if any of the following blocks' peak position differs by more than 500 samples from the first block's peak we conclude that the files are not similar, otherwise they are similar. We chose B to be 5 seconds worth of samples.

Why is it important to split the files into blocks? The intuition is that if we cross correlate the two files as is, then their similarity may be quite high even if some small parts have very bad correlation which could happen if an error occurred such that there was pure noise for a couple of seconds somewhere in the wav file.

3.4 Analysis

The quality assurance algorithm runs in $\mathcal{O}(N \log B)$ time since we split the N samples into N/B blocks of size B then each cross correlation will take $\mathcal{O}(B \log B)$ time to compute, hence the execution time follows. We, however, care a great deal about the constants. For every block we need to perform three Fourier transforms and roughly $4B$ multiplications and divisions. Notice that the unit for N and B is samples. One way to speed up the tools is to simply have lower sample rates and then there will be a trade-off between the quality of the results and the sample rate. The intuition is that radio broadcasts probably do not need 48kHz sample rate and if we have two wave files that are very similar then down sampling should not change the similarity significantly.

We are also interested in the robustness of the migration algorithm. The primary question is, how degraded is the material allowed to become when migrating? Cross Correlation is quite robust wrt. artifacts (eg. extra noise in the background) appearing in addition to what was supposed to be in the result file. By robust, we mean that the algorithm will likely still find the correct offset, but the value of the match decreases as more noise is present. One way to solve degradations like this is either to output some of the blocks that had a low match value for later manual checking or do experiments on degraded signals and fix a parameter that can decide whether the migrated file has an acceptable quality. The last method has the disadvantage that when migrating the same file through a number of intermediate steps it will (maybe) be unrecognizable in the end, though every intermediate step was within the acceptable parameters. Think of this as making a copy of a copy of a ... of a copy of a newspaper article.

4. SCENARIOS

Both the access scenario and the migration scenario are well known in relation to digital preservation [19]. Transcoding or migrating audio and video for access is done as the 'preservation master' is usually too big a file to share with users, maybe it cannot be streamed online, and the "popular" online access formats change [12]. The overlap analysis is relevant in our context as audio broadcasts were recorded in two hour chunks with a few minutes of overlap, and we want to find the exact overlap to make an interface to the broadcasts without strange repetitions every two hours. Migration of audio from MP3 to WAV is done primarily as the WAV is the IASA (International Association of Sound and Audiovisual Archives) recommended preservation format [10].

4.1 Finding Overlap

In connection with the LARM project WP2, the *overlap analysis* issue arose. The LARM project [4] is a collaboration between a number of research and cultural institutions in Denmark. The project provides research data and meta data to a digital infrastructure facilitating researchers' access to the Danish radio-phonic cultural heritage.

The addressed and problematic collection is Danish radio broadcast from 1989 till 2005 from four different radio channels. The recordings were made in two hour chunks on Digital Audio Tapes (DAT), and were recently digitized. Our library got MP3 (and not WAV) copies of these files primarily due to storage limitations. High resolution WAV-files also exist within the broadcasting company. The MP3 files have sampling rate 48 kHz and bit depth 16. The collection is roughly 20 Tbytes, 180000 files and 360000 hours.

In order not to loose content originally, one tape was put in one recorder and a few minutes before it reached the end, another recorder was started with another tape. The two tapes thus have a short overlap of unknown duration, as do the digitized files.

The task is to find the precise overlaps, such that the files can be cut and put together into 24 hour blocks or other relevant chunks correctly.

4.2 Migration QA

The Danish radio broadcast MP3 files are also addressed in the SCAPE *LSDRT6 Migrate mp3 to wav* scenario [5]. They are part of the Danish cultural heritage the Danish State and University Library preserves. They are used as examples of a very large MP3-collection well knowing that original WAV

files actually exist for this collection. We have other collections in both MP3 and other compressed and/or older audio formats that could and should be migrated to WAV at some point in time but chose to work with the same collection for the two scenarios to ease the work. This means that the library would like to migrate the files to WAV (BWF) master files, as is the IASA recommendation [10]. This format has been chosen as the preferred preservation format as this is a raw format, which needs less interpretation to be understood by humans, and is also a robust format. The actual migration is done using FFmpeg [1]. The decompression presents a preservation risk in itself, which is why keeping the original MP3s and performing quality assurance (QA) on the migrated files is recommended.

The QA is done in a number of steps. The first step is validation that the migrated file is a correct file in the target format. We currently use JHOVE2 [3] for this validation.

The second step is extraction of simple properties of the original and the migrated files, and comparing these properties to see if they are 'close enough'. We currently use FFprobe to extract properties. FFprobe is a multimedia streams analyzer integrated in FFmpeg. The properties that are checked are sampling rate, number of channels, bit depth and bit rate.

We could add a third step of extracting more advanced properties using a tool such as e.g. Cube-Tec Quadriga Audiofile-Inspector [15] and comparing these properties. Note however that tools such as Cube-Tec Quadriga Audiofile-Inspector do not compare content of audio files, but rather provides an analysis of a single audio file. We are evaluating significant properties, property formats, property extractors and comparators for possible addition to the workflow.

We have run the migration, validation and property comparison workflow on some of the Danish radio broadcast MP3 files creating a small test set for further QA. The workflow is written as a Taverna [9] workflow and is available on myExperiment [16]. The workflow used SCAPE web services which are set up locally. The used SCAPE web services are the FFmpeg, JHOVE2 and FFprobe web services defined in the scape GitHub repository [6]. Comparison of the extracted properties is done with a Taverna bean shell. The workflow input value is a fileURL containing a list of input MP3 URLs. The output is a list of Wav fileURLs, a list of validation outputs (valid / not valid) and a list of comparison outputs (properties alike / not alike).

The test set contains 70 Danish radio broadcast MP3 files. The workflow was run on a test machine with an Intel(R) Xeon(R) CPU X5660 @ 2.80GHz processor and 8GB RAM running Linux 2.6.18 (CentOS). The workflow finished in approximately 5 hours and 45 minutes. This means we have a performance of almost 5 minutes pr file. Earlier tests have shown that the most expensive components in the workflow is the FFmpeg migration and the JHOVE2 validation, while FFprobe characterisation and property comparison is relatively cheap [18].

We note that the Danish Radio broadcasts mp3 collection is 20 TB and around 180000 files. This means that running the basic workflow migrations sequentially on the test machine would take more than 600 days. We do however plan to improve that significantly by using the Scape execution platform instead of doing the migrations sequentially on just one server.

Another related scenario is that The Danish State and University Library have a very small collection of Real Audio (200 files) that are planned to be migrated to wav. The actual FFmpeg migration needs adjustment and we need to find another independent implementation of a Real Audio decoder, but the rest of the workflow as well as the algorithms presented in this paper can be applied to this issue directly.

5. EXPERIMENTS

5.1 Overlap Analysis Tool Use

We have already used the overlap tool on real data sets. The `xcorrSound` tool is used to find the overlaps. The solution must consider

- Some recordings (files) may be missing.
- Noise at both ends of the sound files. Can be both silence and changing to a different station.
- The sound recording may have been started up to 23 minutes early.
- There must be a quality assurance to show that the transformation was successful. The tool used for this QA is also the `xcorrSound` tool. The success criteria are:
  - Good overlap measurement. QA check match value of at least 0.2
  - Length of resulting file is **not** checked, as above check also catches these cases.

The overlap match is done by a script, which first cuts a short interval (1 second) of either end of the files, as this is often noise related to the start or finish of recording, see Fig. 2. Then a time interval of 10 seconds in the second file is cut for later QA analysis. The `xcorrSound` tool is now run on 6 minutes of the end of the first file and the beginning of the second file. The output is a best match position and best match value. Using the best match position, the `xcorrSound` tool is run a second time on the 10 second time interval cut for QA. If the best match value is acceptable, the files are cut and concatenated at the best match position.

The results were initially all checked manually to estimate the acceptable values for the best match. The results where the best match value is not acceptable, are checked manually and the script is tweaked to hopefully provide matches.

5.1.1 Results

Our data set consisted of one month of radio broadcasts recorded in 2 hour chunks. The goal was to cut them into 24 hour chunks instead. The `xcorrSound` tool worked very well. We found that when doing the QA check, if the value produced was below 0.1 there was an error and if the value

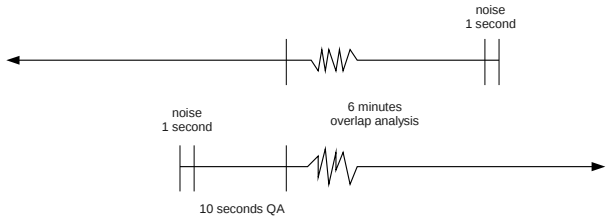

Figure 2: Overlap Analysis

was above 0.2 it was safe to assume the process went correct. We found several actual errors in the content using this tool. Examples include that one file simply contained a wrong radio show (may have happened if a channel was changed), several files in a row were identical, hence they would not overlap in the end and an error would be produced or there was a lot more overlap than the promised few minutes - up to 25 actually. All these errors in the data set was caught and reported. The tool of course only tells when two files do not overlap and the actual reasons have to be manually found. QA values that lie in the range 0.1 - 0.2 are the ones that we are not quite sure of and we would do manual quality assurance on these. However It is rare that the QA values lie in that range and most commonly the QA match is above 0.7.

5.1.2 Discussion

The `xcorrSound` tool has been used and the results were quite good. We found several errors in the collection that we can now correct. As can be seen we have a nice structure on the values of the QA match. We have found that by doing experiments and trying to listen to the broadcasts and comparing with the QA match values we can now run through a large collection and do automatic quality assurance because we have determined the different intervals we can trust for the QA values. We know that when a QA value is below 0.1 there is almost surely an error and when the QA value is above 0.2 there is not an error.

5.2 Migration QA Tests

In order to test the `migrationQA` tool we needed a data set. The test data set contains 70 two-hour radio broadcast files which were migrated using FFmpeg[1], see Section 4.2. The average file size of the original mp3 files is only 118Mb, but the migrated wav files are approximately 1.4Gb. Three of them were replaced by a randomly generated file with a 'correct' wav-header, such that the `migrationQA` tool was able to process them. We assume that checks such as correct header information are performed before invoking the `migrationQA` tool. Five of the remaining 67 files were kept intact except for a few seconds a few places within the file which were replaced by randomly generated bytes. The other 62 files were kept as they were after migrating through FFmpeg. We have an inherent problem using this data set because it is quite artificial. We imagine that the data set contains errors that *might* occur during a migration, but we have no basis for this as we have never seen any erroneous migrations. To use the `migrationQA` tool we need to 'play' or interpret the files, just as a human needs to 'play' or interpret an mp3 file to hear the sound. We currently use MPG321[2] to 'play' the original mp3 files. MPG321 is an independent implementation of an mp3-decoder, thus independent from FFmpeg, which was used to migrate the files. The migrated files are already in wav format and are used directly.

The *migrationQA SCAPE Web Service including MPG321 decoding Workflow* [17] on myExperiment takes an mp3 file and a wav file as input. The mp3 file is then played into a temporary wav file using MPG321, and the temporary file and the input wav file are compared using the `migrationQA` tool. This workflow however only works on one pair of files.

We have tested the tool on a list of 70 pairs of mp3 and migrated wav test files using a bash script. The *migrationQA including MPG321 workflow bash script* was run on a test machine with an Intel(R) Xeon(R) CPU X5660 @ 2.80GHz processor and 8GB RAM.

5.2.1 Results

The script ran for 4 hours and 45 minutes. This gives us a performance of just over 4 minutes pr. file. This is roughly equally divided between the MPG321 migration and the `migrationQA` comparison.

In total there were 12 reported errors, which is 4 more than we expected. All the files that were supposed to be found during this QA check were found, so we only have some false positives left (or false negatives depending on your view). We investigated the additionally reported errors. The 'limit' of 500 samples difference from the first block may in fact be too low. On one pair of files the best offset was 1152 samples during the first 6850 seconds of the file (00:00:00-01:54:10) but during the remaining part of the file it changed to having the best offset at 3456 samples and a cross correlation match value of nearly 1 (0.999-1.0).

5.2.2 Discussion

The fact that partly through the file the best offset changed suggests that either one of the converters has a bug or there were some artifacts in the original mp3 file that is not following the standard and thus they simply do not recover from this in the same manner. Of course when there are 48000 samples/second we cannot hear the difference between an offset on 3456 and 1152 (4.8 milliseconds). Now the question is as much political as it is implementational. Was the migration correct or was it not? Obviously one can argue that since we cannot hear any difference between the two files, the migration went as it should. On the other hand, one of the files we ran the `migrationQA` program on must have had some errors, if we accept that one of the files must be a correct migration. Ultimately the question is up to the definition of a correct migration, which is a subject we have carefully avoided in this paper. One solution is to let the `migrationQA` program take a parameter that decides how much difference from the first block is allowed, rather than fixing a magic constant of 500 samples. Another solution is to try to identify what exactly is happening inside the migration tools (FFmpeg and MPG321) to find out why they differ and check if one of them has a bug or if it was in fact the original mp3 file that did not follow the standard.

One might argue that the `migrationQA` program is as much

a validation tool of other programs that migrate audio files to wav to check if they agree as it is a Quality Assurance tool for migrated files. This happens when we accept one migration tool to be correct and then try migrating a lot of files using that tool and another we want to test correctness of. If they agree on the output, then we can have some confidence the other migration tool is correct as well.

In this paper we had two ways of migrating an mp3 file to wav, but we were unsure whether any of them were correct. If we assume that the migration tools are independent implementations this should intuitively provide some assurance that they do not have the same error (if they have any). Hence, if they agree on the output we have some confidence that the migration went as it should. The question is, if it is a reasonable assumption that they do not have the same error if they are independent implementations. They are after all implementing the same algorithm, which likely has some parts that are non trivial to implement and others that are trivial.

We care a great deal about efficiency, and just over 4 minutes per file is at the moment acceptable. The algorithm is not easy to make parallel but it is easy to have several instances of the same algorithm running. This is a feasible solution because the algorithm can be implemented to use a limited amount of memory. All that needs to be in memory at any point is the match value and offset of the first block, the current block being processed and some buffers for speeding up the I/O. Our implementation uses roughly 50mb memory when running. If we have a machine that can run multiple instances of the program, it might be the I/O operations that become the bottle neck of the program.

6. CONCLUSION AND FURTHER WORK

We presented algorithms for doing Quality Assurance on audio when migrating from one file format to another. We also gave an algorithm to eliminate overlap between audio files such that potential listeners do not need to hear the same bit twice. The experiment for QA showed that the tool works well on the constructed input. Since we do not have any data where the migration goes bad we cannot speak to how good the tool actually is, but we believe that it will work very well. The experiment also showed that there is not one single algorithm that will fit all. It might be necessary to fiddle with parameters depending on the data set being processed. Further work in this area is to try to develop even faster algorithms and develop better metrics for comparing audio. We used the Cross Correlation metric, which is a relatively expensive metric to compute, perhaps there are cheaper ones that work just as well or more expensive ones that can give better guarantees. For doing the overlap analysis we could possibly have adopted a finger printing scheme (such as [14]) that would have worked just as well, though that solution is a lot more complex than our suggested approach. The technique of applying cross correlation is general and might have application elsewhere for doing QA. It is worth investigating if we can reuse the same ideas for other areas as well.

Acknowledgements

Thanks to Bjarne Andersen, Henning Böttger and Asger Askov Blekinge for all their work on the experiments and help with the paper.

7. REFERENCES

[1] FFmpeg (2012), ffmpeg.org
[2] Homepage of mpg321 (2012), mpg321.sourceforge.net
[3] JHOVE2 (2012), jhove2.org
[4] LARM audio research archive (2012), www.larm-archive.org/about-larm/
[5] LSDRT6 migrate mp3 to wav (2012), wiki.opf-labs.org/display/SP/LSDRT6+Migrate+mp3+to+wav
[6] SCAPE project repository (2012), https://github.com/openplanets/scape
[7] SCAPE scenarios - datasets, issues and solutions (2012), wiki.opf-labs.org/display/SP/SCAPE+Scenarios+-+Datasets%2C+Issues+and+Solutions
[8] Scape xcorrsound tools (2012), https://github.com/openplanets/scape-xcorrsound
[9] Taverna workflow management system (2012), taverna.org.uk
[10] Committee, I.T.: Guidelines on the production and preservation of digital audio objects. standards, recommended practices and strategies, iasa-tc 04, www.iasa-web.org/tc04/audio-preservation
[11] Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of Computation 19(90), 297–301 (1965)
[12] Elsheimer, S.: Introduction to transcoding: Tools and processes (2011), www.prestocentre.org/library/resources/introduction-transcoding-tools-and-processes, presentation at Screening the Future 2011
[13] Frigo, M., Johnson, S.G.: FFTW library (April 2012), http://www.fftw.org/
[14] Haitsma, J., Kalker, T.: A highly robust audio fingerprinting system. In: Proceeding of the International Symposium on Music Information Retrieval (ISMIR) (2002)
[15] Houpert, J., Lorenz, T., Wiescholek, M.: Quadriga - audiofile-inspector - cube-tec international (August 2012), http://www.cube-tec.com/quadriga/modules/audiofileinspector.html
[16] Jurik, B.: Workflow entry: Migrate mp3 to wav validate compare list to list (May 2012), www.myexperiment.org/workflows/2914.html
[17] Jurik, B.: Workflow entry: migrationqa scape web service including mpg321 decoding workflow (March 2012), http://www.myexperiment.org/workflows/2806.html
[18] Pitzalis, D.: Quality assurance workflow, release 1 & release report (March 2012), http://www.scape-project.eu/deliverable/d11-1-quality-assurance-workflow-release-1-release-report-draft
[19] Wright, R.: Preserving moving pictures and sound. dpc technology watch report 12-01 march 2012. Tech. rep., The Digital Preservation Coalition (DPC) (2012)