Searching digital music libraries

David Bainbridge, Michael Dewsnip, and Ian Witten

Department of Computer Science
University of Waikato
Hamilton
New Zealand

Abstract. There has been a recent explosion of interest in digital music libraries. In particular, interactive melody retrieval is a striking example of a search paradigm that differs radically from the standard full-text search. Many different techniques have been proposed for melody matching, but the area lacks standard databases that allow them to be compared on common grounds—and copyright issues have stymied attempts to develop such a corpus. This paper focuses on methods for evaluating different symbolic music matching strategies, and describes a series of experiments that compare and contrast results obtained using three dominant paradigms.

1 Introduction

There has been a recent explosion of interest in digital music libraries—indeed, Apple's iPod has been called the world's first consumer-oriented digital library. In all human societies, music is an expression of popular culture. Different generations identify strongly with different musical styles. People's taste in music reflects their personality. Teenagers, in particular, feel that their musical preferences are strongly bound up with who they are. Many researchers seek to capitalize on this natural interest by building digital music libraries [BD00,DB01].

Digital music libraries are an attractive area of study because they present interesting and challenging technical problems, solutions to which are likely to be highly valued by enthusiastic end-users. This paper addresses the problem of searching a music library for a known melody. In other words, given a fragment of an unknown melody, typically played or sung by a user, return a list of possible matches to a large digital library collection. This operation is not supported by the information structures provided by traditional libraries, except insofar as knowledgeable music librarians are able to provide human assistance. For scholarly work on melody, there is a book that provides a paper index of themes [Par75], but its scope is restricted to the older classical repertoire and it does not provide flexible searching options. And yet the problem of melody retrieval is of great interest to a wide range of potential users—so much so that there are popular radio programs that feature human abilities to "guess that tune".

A practical scheme for searching digital music libraries requires robust implementations of several supporting components. First, it is necessary to assemble a large database of music in searchable form—which implies some kind of symbolic representation. Normally this is accomplished by manually entering a large number of melodies on a keyboard, a highly labor-intensive process. The alternative is to automatically infer notated information from an analog form of the music, such as a recording of a performance, or a paper score. The latter possibility, OMR for "optical music recognition," is a well-advanced technology (e.g. [BB01]) but is not addressed in this paper. Second, the audio query, generated by the user singing, whistling, or humming it—or playing it on a keyboard—must first be transcribed into the same symbolic representation. This is a far easier proposition than inferring a musical score from a recording of a performance, because the input is monophonic—only one voice is present. However, again we do not address the problem in this paper: the transformation is accomplished by standard signal-processing techniques of pitch detection [GR69], followed by quantization of the pitch track in both frequency (placing the notes on a standard musical scale) and time (placing the notes within a standard rhythmical framework). Third, the music searching operation must take place within a context that allows the user to examine search results and request that generalized "documents" be presented. Such documents might include stored musical performances, performances synthesized on the fly from some symbolic representation, facsimile images of the score, scores created on demand from a symbolic representation by musical typesetting techniques, and so on. Suitable general contexts for browsing and document presentation exist (e.g. [BNMW$^+$99]); again, they are not addressed by this paper.

We focus here on the central problem of music matching. We assume that the material to be searched is stored in symbolic form in terms of the notated music. We assume that the audio query has been transcribed into the same symbolic form. We assume that a suitable infrastructure is in place for examining and presenting results.

Three basically different approaches to symbolic music matching have been proposed: dynamic programming [MS90], state matching [WM92], and n-gram-based methods that employ standard information retrieval techniques [WMB99]. All have been used to implement practical melody retrieval systems. Dynamic programming techniques work by calculating, in an efficient manner, the "edit distance" between the query and each melody in the database. The lower the distance, the better the match. Rudimentary edit operations include adding a note, deleting a note and substituting one note for another, along with more subtle changes such as consolidating a series of notes at the same pitch into one note of the composite duration. State based matching also works by adding, deleting and substituting notes to provide an approximate match, but its implementation takes a different form. It uses a matrix of bits that records the state of partial matching so far, and achieves efficiency by encoding the matrix as an array of integers. Given this data-structure only shifts and bitwise Boolean operators are needed to implement the matching progress. Unlike dynamic programming, state-based matching does not keep track of which edits were made, and its running time is proportional to the number of errors that are allowed.

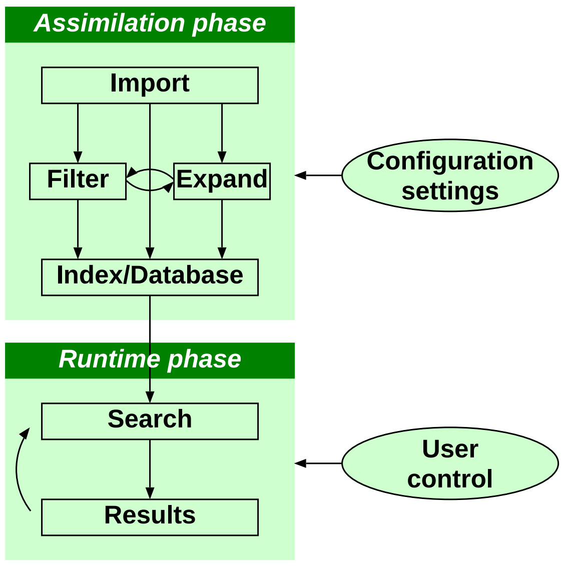


Fig. 1. A workbench for symbolic music information retrieval.

N-gram-based methods work by mapping both queries and melodies to textual words (n-letters long) and then using full-text retrieval to locate documents in the database that contain the “words” included in a given query.

The aim of this paper is to provide a comparative evaluation of these three methods for searching digital music libraries. We explore three orthogonal axes. The first measures “query length uniqueness”, and is the minimum length of query (in notes) needed to unambiguously determine a unique melody. The second is ranked return—how high up the sought-after melody appears in the ordered list of returned matches. The third is the computational efficiency of the searching method.

We begin by introducing the workbench we have developed to perform these experiments, and then describe the experiments themselves and the results obtained. We conclude with a summary of our findings.

2 A workbench for symbolic music information retrieval

To support practical research in this area we have developed a workbench for symbolic music information retrieval. Figure 1 gives on overview of the system, which fits into a larger digital library software architecture, Greenstone [WRBB00,WBB01]. Work is divided into two phases: the assimilation phase and the runtime phase. The former is responsible for collecting files together and creating from them the necessary indexes and/or databases. The latter, guided by user input, supports experimentation and evaluates performance measures.

While assimilation is typically performed once for a given set of experiments, the runtime phase is executed many times to gather results from different experiments.

The assimilation phrase is controlled by a configuration file that determines what files are gathered together and how they are processed and indexed. It begins with an "import" process that is capable of reading the plethora of different file formats associated with music data and normalizing them by converting them into a canonical format. For this we use the XML version of Guido [HRG01]: it is general, expressive enough for our needs, and straightforward to parse.

The next steps, which are optional, are to filter and expand the normalized input files. Filtering reduces the stream of musical data—an example is to retain only those tracks that are monophonic in a given MIDI input. Expanding increases the musical data—an example is to generate versions of a melody in all different keys. While it is very useful conceptually to distinguish between filtering and expanding, from an implementation standpoint the difference is insignificant and both categories use the same basic implementation. The two entities are represented separately in the design because it is useful to be able to differentiate between them during the configuration phase.

The workbench implements three broad types of algorithm for symbolic music information retrieval: state-based matching, dynamic programming, and text-based information retrieval of n-grams. These require different levels of support at assimilation time, and the configuration settings that govern the assimilation phase dictate what indexes and databases are built. For example, little work is needed by the assimilation phase to support interval matching with the dynamic programming algorithm, since the Guido format closely resembles the data needed by this matching algorithm. In comparison, to support contour mapping by n-grams the assimilation phase needs to map the Guido notes and durations to a textual representation suitable for text information retrieval, and build the appropriate full-text indexes.

In the runtime phase, users issue commands to interact with the workbench. They can provide sample inputs and match them against the database using different matching algorithms, examining and comparing the results. Each matching method has optional arguments that modify its behavior. For example, one can seek matches only at the start of melodies, rather than at any position within them. Instead of using exact pitch intervals one can match a pitch contour, which records for each pitch change whether it rises and falls, rather than the amount by which it rises or falls. The workbench implements many other matching options. The outcome of a search is held as a result set from which statistics are extracted, graphs plotted, tables generated, and so on.

Interactive use has its limitations, particularly when setting up and running large experiments. Consequently there is a facility for users to develop a "script" that defines a particular series of experiments. This script is then run by the workbench in batch mode, and the results are recorded in files for later examination by the user.

A third mode of operation is to allow a different process, rather than an online user or a pre-prepared script, to access the facilities of the workbench. The workbench can be accessed through a web-based user interface, using the CGI mechanism, to perform music-content retrieval and format the data returned in a suitable format. This allows its use directly by digital library software, for example, Greenstone. The advantage is that exactly the same implementation and options are used for live retrievals as have been evaluated in interactive and off-line experiments.

The workbench design is capable of supporting polyphonic matching. However, the experiments reported here focus on monophonic-to-monophonic matching.

The music information retrieval workbench will be released under the GNU public license. It will provide a uniform basis for evaluating melody matching algorithms. More importantly, other research groups will be able to add their retrieval algorithms to it, allowing a comprehensive comparison of their strengths and weaknesses against the prior state of the art without the need to continually re-implement earlier methods. An alternative strategy, which has been adopted in other communities (e.g. text compression [AB97] and machine learning [BKM98]), is to develop a standard corpus of material against which different algorithms are evaluated, and publish the results of these evaluations. However, in the context of music, where source material is heavily copyrighted, the distribution of an evaluation workbench seems more likely to achieve the desired effect.

3 Experimentation

The purpose of our experiments is to shed light on how well commonly-used music information retrieval algorithms perform under a wide variety of conditions. This provides the basic information needed to design and configure a digital music library. Such information is necessary to make a sensible choice of any algorithms used to support query by music content in practice; it is also necessary to fine-tune particular parameter settings. Conditions differ from one digital library to the next, depending on factors such as the user community being served, the computer infrastructure that underpins the service, and the type and size of the collection. Our aim is to provide design data for digital music libraries. If, in addition, a library uses our workbench to respond to queries, the implementation is guaranteed to be the same as was used to produce the design data.

3.1 Dataset

For evaluation, we need to use standard corpora of melodies. Recall the legal difficulties, mentioned above, of creating and distributing corpora. In the absence of a globally used corpus we have used a dataset of folksongs that is available to us internally. The dataset combines songs from the Essen and Digital Tradition collections [BNMW$^+$99] to form a dataset of nearly 10,000 songs.

3.2 Summary of experiments

Our experiments are based on earlier work by McNab [McN96], Downie [Dow99], and Rand *et al.* [RB01]. Where possible, we follow the same experimental procedures, expanding and extending the details appropriately.

In the first experiment we examine how many notes each method requires for the list of matches to contain just one tune—the sought-after one. This is repeated for a range of different circumstances. The second experiment studies the ranking performance for each method under more realistic user conditions, albeit simulated. The experiment quantifies retrieval quality in terms of the position in which the sought-after tune appears in the returned list, and thereby helps establish how useful each method is from a user's perspective. The third experiment considers computation cost, and introduces a hybrid approach. It is known that music-based n-gram systems are computationally very efficient and have high recall, but suffer from low precision [Dow99]. Motivated by this observation, the final experiment evaluates a system that first performs an n-gram query and then applies (in one variation) the dynamic programming approach to refine the resulting set of search results, and (in a second variation) the state-based matching algorithm.

3.3 Uniqueness

It is interesting to establish how many notes a user must sing before a music information retrieval algorithm returns one match—namely, the sought-after tune. We call this measure *uniqueness* since it helps gauge how selective or inclusive a technique is. For instance, when using exact matching one would expect the number of returned tunes to drop off more quickly than with approximate matching, but does this actually happen, and is the difference significant?

Many parameters can be varied in this experiment: with or without duration, interval or contour, match at the start or anywhere within the tune. Within these broad categories further choices must be made. For example, Downie [Dow99] considered four different ways to map musical notes to text characters and varied the size of n-grams from 4 to 6. We have experimented with many variations, but (due to space limitations) we present selective results that indicate the trends we observed, and note which were the most influential parameters and which had little effect.

Figures 2–4 show the uniqueness measure applied to the folksong dataset. We used exact and approximate matching, ignored durations and took them into account, and described pitch in absolute, interval, and contour terms. For 300 melodies chosen at random from the database, an initial query of two notes was extended a note at a time up to a limit of twenty notes, and the resulting number of tunes returned was recorded. The average result for each query length was then calculated and plotted.

Figure 2 tests the idealized circumstance that the notes sung in the query—ignoring rests—are exactly the same as the corresponding notes in the sought-after tune in the database. This is as good as it gets! It represents the best results

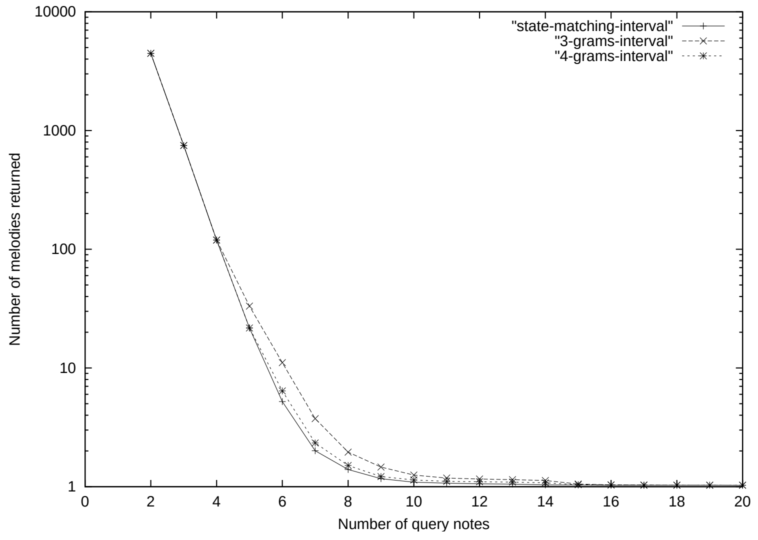


Fig. 2. Testing the uniqueness of matching algorithms versus indexing algorithms (Rest mode: ignored; Location: anywhere; Duration: used; Matching: exact; Pitch mode: interval).

any matching algorithm can hope to achieve (without resorting to additional features of music notation such as dynamic markings). The figure shows state-based matching and two versions of n-gram (3-gram and 4-gram). Dynamic programming is omitted because it produces *exactly* the same result as state-based matching.

The n-gram methods yield slightly higher curves because in the version tested, detected n-grams do not need to occur consecutively. A stricter version of the search can be accomplished by issuing the query as a “phrase search,” in which case it too would produce the lower of the three curves. While dynamic programming and state-based matching are implicitly designed for approximate matching, this is not true for n-gram-based matching. Removing the requirement that n-grams be consecutive is one way to incorporate flexibility, so it is interesting to see how this variation performs. The second category of experiment (see Section 3.4) evaluates how relaxing this requirement affects the quality of the search results.

Figure 3(a), for state-based matching, compares the result of using absolute pitch (so the query needs to match the exact key the song in is), intervals, and contours. The contour mapping used recorded if a note’s pitch was above, below or the same as the previous note’s pitch (category C3 in Downie’s work [Dow99]). Not surprisingly, absolute pitch queries require fewer notes than interval queries, which in turn need fewer than contour queries. For example, a user who sings a query and wants only four songs returned (which can then be checked manually

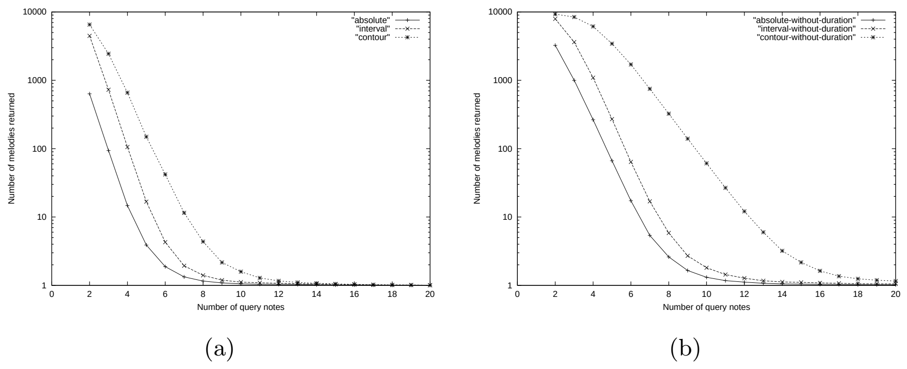


Fig. 3. Uniqueness experiment for state-based matching with exact matching anywhere in melody and rests ignored (a) with duration (b) without duration.

by playing them) must sing 5 notes for the absolute match, 6 notes for the interval match and 8 notes for the contour match.

Figure 3(b) repeats the experiment but ignores note duration. The progression is as before, but contour matching is considerably worse. To return just four songs a further two more notes must be sung for absolute and interval matching, but five for contour matching.

Repeating these two experiments with dynamic programming and consecutive n-gram matching yields exactly the same results. The trend for the more relaxed n-gram version is to return a greater number of melodies than the comparable versions of the other two algorithms, and for the disparity to be at its most significant when the query is 8–14 notes long.

So far we used the dynamic programming and state-based matching algorithms in an "exact matching" mode. The shows how well things go in an ideal situation, and allows comparison with n-gram algorithms that are inherently exact-match based. We now measure uniqueness for approximate matching.

Figure 4 shows what happens when 1, 3 and 5 mismatches are allowed between the state-based matching algorithm and the dataset. Naturally more melodies are returned than before for the same query, and the number increases as more mismatches are allowed. If only four songs are to be returned, users must now sing on average 8, 11, and 14 notes respectively. Using approximate contour matching the values increased further to 11, 15 and more than 20.

The dynamic programming algorithm yields similar trends. However, the two are not equivalent because of differences in how the cost function is calculated.

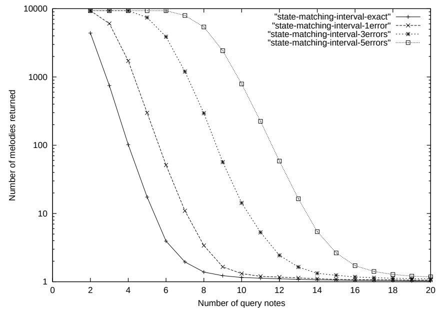


Fig. 4. Exact matching versus approximate matching for the state-based technique with duration, matching anywhere in melody and ignoring rests.

3.4 Ranking

In the uniqueness experiments, the sample queries were replicated excerpts of the sought-after tune in the database. This does not reflect the reality of a digital music library, where the user's query may not match any tune in the database for many reasons—different arrangements of the same song, imprecise recollection, the original does not exist in notated form, and so on. This section describes an experiment that takes into account such errors. It establishes how useful a matching algorithm is from a user's perspective by studying tune ranking, that is, the position in which the sought-after melody appears in the list of returned melodies.

Figure 5 shows the ranked position of the sought-after melody when some notes are omitted from the query, which is originally ten notes long. The x-axis shows the percentage of notes omitted, and values are averaged over 100 queries.

For dynamic programming, state-based matching and 3-grams the sought-after melody appears in the top 10 when zero or one note is missing from the query. Beyond that point the plots rise steeply indicating a marked worsening of the melody's ranked position. The same trend appears in the 4-gram method, only its ranked scores are notably poorer.

Towards the end of the graph (percentage error 70%–80%) both the 3- and 4-grams improve slightly. However, the improvement is of little practical import because it moves the sought-after tune up to rank 3,000—still very far from the start of the list. To see why the effect occurs, consider the size of the n-gram at this point. In the case of 4-grams, by the time the experiment has dropped 7 or more notes, the query is shorter than the length of the n-gram. In order to support queries of two or three notes—something a user could reasonably expect—we modified the n-gram algorithm to also calculate 3- and 2-grams, and it is this part of the index that performs the matching in the experiment's final stages.

Repeating the experiment for contour matching produces a similar graph to Figure 5. The main difference is a softening of the gradient when moving from

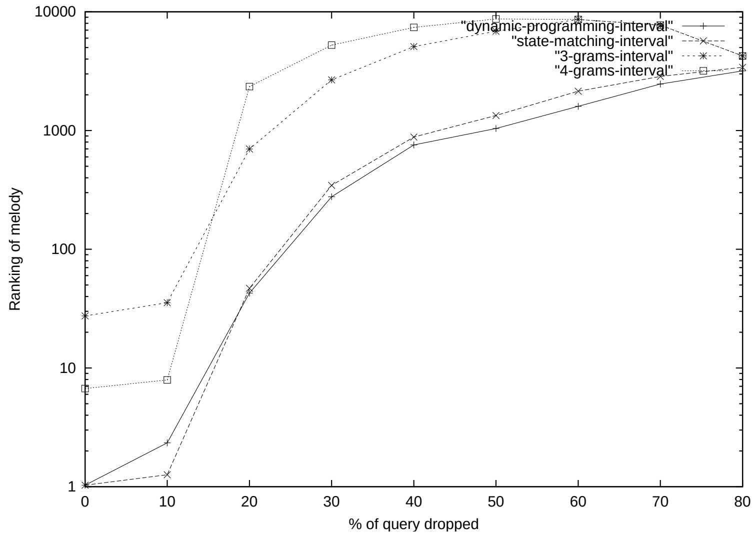


Fig. 5. Simulation of dropping notes with approximate matching of intervals where rests are ignore and duration is used.

one to two missing notes. This is because in all cases the ranked position when one note is missing is higher than the equivalent interval ranked position, but the when two notes are missing the ranked positions are comparable.

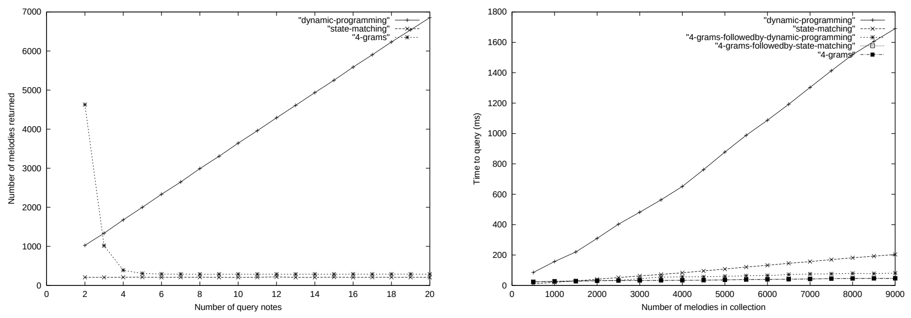
3.5 Computational cost

Now we examine the computational cost of the three methods. Figure 6(a) shows the stark difference between dynamic programming and state-based matching. The 4-gram method's cost is high at first, but quickly reduces before stabilizing at a consistent value that is a little larger than that for state matching. The initial expense is caused, once again, by the fact that when the length of the n-gram (4) exceeds that of the query, we drop down to 3-grams and 2-grams.

An approximate version of the dynamic programming algorithm gives exactly the same results, since runtime is not proportional to the error value. An approximate version of state-based matching gives values that are slightly greater than for exact matching, depending on the degree of error. This is because the runtime complexity of this algorithm is directly proportional to the error value.

The great efficiency, but disappointing effectiveness, of n-grams leads one to consider whether they can be used as a pre-filtering step for the more expensive, but more effective, dynamic programming and state-based matching algorithms. The effect of this is shown in Figure 6(b).

In this experiment the collection size was varied from 500 songs to 9000 songs in 500 song increments. For each collection size approximate matching versions



(a) (b)

Fig. 6. Computational cost for matching anywhere in query with duration and ignoring rests (a) increasingly long queries with exact interval matching (b) increasingly large collections with contour matching.

of the dynamic programming and state-based matching algorithms (allowing up to two mistakes) were run and the time taken compared with versions that used 4-gram matching to prefilter the dataset.

The difference observed between dynamic programming and state-based is marked, with dynamic programming taking on average 8 times longer to perform a match. Although it should be remembered that the former is unrestricted in the number of mismatches that can occur, whereas the version of the state-based matching tested allowed only two mistakes.

The cost of matching 4-grams stays consistently low. Although it is hard to make out in Figure 6(b) due to the high density of lines, by the time the dataset contains 1,500 melodies, its cost is cheaper than state matching. For the hybrid methods, once the collection size has crossed the same threshold 1,500 it too represents a faster search that state matching alone. Both versions of the hybrid algorithm fall between the lines plotted for 4-gram state-based matching solutions: 4-gram followed by dynamic programming is roughly twice as time-consuming as the 4-gram method alone; 4-gram followed by state-based matching is so close to the 4-gram base line it cannot be visually distinguished.

4 Conclusion

We conclude this paper by relating the outcomes of our experimentation to forming a digital music library.

The uniqueness experiments—Figures 3(a)–4—help gauge how many notes make a useful query for a digital music library. Turning the issue around, having determined the typical number of notes sung by a user in a query, what are the implications of selecting a particular matching algorithm with certain parameter settings? The ranking experiment—Figure 5—helps gauge how much sifting through the list of returned songs may be required by the user to locate the sought-after melody. Together these experiments help place limits on what parameters are acceptable for a given matching algorithm in a digital library.

Say the designer of a digital music library expects users to sing queries with around 6–8 notes, and—because users rarely go beyond the first page of search results [JCMB00]—wants the sought-after melody to be ranked in the top 10. The graphs of uniqueness and ranking show that this rules out performing any contour matching without duration. It also rules out state-based matching with three or more errors,[1] and 3-grams. This leaves interval matching with or without duration, and contour matching with duration, as strong candidates for use, and 4-grams with interval and duration as a tolerable option.

Issues of computational efficiency are revealed by the third set of experiments. Greater speed without compromising accuracy is a strong factor driving the implementation of a digital music library. Figure 6(b) shows that there is a definite advantage in a digital library system using a hybrid approach, particularly in the case of pre-filtering the dynamic programming algorithm. The initial high cost of n-gram matching for queries with fewer notes than the basic n-gram size is of minor concern and can be handled by warning the user, if they issue such a query, that so few notes are likely to return a high number of songs and asking if they wish to continue.

This discussion applies to the folksong dataset. These recommendations can probably be extrapolated to other collections, but some caution must be exercised. The dataset used is monophonic, based on notated form, and is from one genre—folk music. What happens if the genre is different? What happen if the dataset is sourced from MIDI files, a readily available form of music but one that is much noisier. For instance, duration information is less reliable because the song is typically entered using a synthesizer keyboard, and passages of the music that repeat are played out rather than appearing once and being notated as repeating. Further experimentation is required to understand how such changes alter the requirements of a digital music library, and what better way to manage this than through a workbench for music information retrieval!

References

[AB97] R. Arnold and T. Bell. A corpus for the evaluation of lossless compression algorithms. In *Designs, Codes and Cryptography*, pages 201–210, 1997.

[BB01] D. Bainbridge and T. Bell. The challenge of optical music recognition. *Computers and the Humanities*, 2001.

[BD00] D. Bird and J.S. Downie, editors. *Proceedings of the 1st. Int. Symposium on Music Information Retrieval: ISMIR 2000*, Plymouth, Massachusetts, USA, 2000. available through *www.music-ir.org.*

[1] A similar cutoff threshold for dynamic programming can also be determined. However, the necessary graph is not shown in the paper and the value is specific to the cost functions used to calculate edit distance.

[BKM98] C. Blake, E. Keogh, and C.J. Merz. *UCI Repository of Machine Learning Databases.* http://www.ics.uci.edu/~mlearn/mlrepository.html, University of California, Department of Information and Computer Science, Irvine, CA, Irvine, CA, USA, 1998.

[BNMW$^+$99] D. Bainbridge, C. Nevill-Manning, I. Witten, L. Smith, and R. McNab. Towards a digital library of popular music. In *The 4th ACM conference on Digital Libraries*, pages 161–169, 1999.

[DB01] J.S. Downie and D. Bainbridge, editors. *Proc. of the 2nd Int. Symposium on Music Information Retrieval*, Indiana University, Bloomington, IN, USA, 2001. available through *www.music-ir.org.*

[Dow99] J.S. Downie. *Evaluating a Simple Approach to Musical Information Retrieval: Conceiving Melodic N-Grams as Text.* PhD. thesis, University of Western Ontario, Canada, 1999.

[GR69] B. Gold and L. Rabiner. Parallel processing techniques for estimating pitch periods of speech in the time domain. *J. Acoust. Soc. Am.*, 46(2):442–448, 1969.

[HRG01] H. Hoos, K. Renz, and M. Gorg. GUIDO/MIR: An experimental musical information retrieval system based on guido music notation. In J. Stephen Downie and David Bainbridge, editors, *Proc. of the 2nd Int. Symposium on Music Information Retrieval: ISMIR 2001*, pages 41–50, 2001.

[JCMB00] S. Jones, S.J. Cunningham, R.J. McNab, and S. Boddie. A transaction log analysis of a digital library. *International Journal on Digital Libraries*, 3(2):152–169, 2000.

[McN96] R. McNab. *Interactive applications of music transcription.* MSc thesis, Department of Computer Science, University of Waikato, NZ, 1996.

[MS90] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, pages 161–175, 1990.

[Par75] D. Parsons. *The Directory of Tunes and Musical Themes.* Spencer Brown, Cambridge, 1975.

[RB01] W. Rand and W. Birmingham. Statistical analysis in music information retrieval. In J. Stephen Downie and David Bainbridge, editors, *Proc. of the 2nd Int. Symposium on Music Information Retrieval*, pages 25–26, Indiana University, Bloomington, IN, USA, 2001.

[WBB01] I. Witten, D. Bainbridge, and S. Boddie. Greenstone: open source dl software. *Communications of the ACM*, 44(5):44, 2001.

[WM92] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.

[WMB99] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: compressing and indexing documents and images.* Morgan Kaufmann, San Francisco, CA, 1999.

[WRBB00] I. Witten, McNab R., S. Boddie, and D. Bainbridge. Greenstone: a comprehensive open-source digital library software system. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 113–121, San Antonio, Texas, June 2000.