in *Proceedings of the the International Conference on Music Information Retrieval,* London, England, September 11-15, 2005

POLYPHONIC MUSICAL SEQUENCE ALIGNMENT FOR DATABASE SEARCH

Bryan Pardo and Manan Sanghi
Computer Science Department
Northwestern University
1890 Maple Ave., Evanston, IL, USA
pardo,manan@northwestern.edu

ABSTRACT

Finding the best matching database target to a melodic query has been of great interest in the music IR world. The string alignment paradigm works well for this task when comparing a monophonic query to a database of monophonic pieces. However, most tonal music is polyphonic, with multiple concurrent musical lines. Such pieces are not adequately represented as strings. Moreover, users often represent polyphonic pieces in their queries by skipping from one part (the soprano) to another (the bass). Current string matching approaches are not designed to handle this situation. This paper outlines approaches to extending string alignment that allow measuring similarity between a monophonic query and a polyphonic piece. These approaches are compared using synthetic queries on a database of Bach pieces. Results indicate that when a monophonic query is drawn from multiple parts in the target, a method which explicitly takes the multi-part structure of a piece into account significantly outperforms the one that does not.

1. INTRODUCTION

Finding the best matching database target to a melodic query has been a subject of great recent interest in the music IR world [1-4]. Query-by-humming (QBH) systems [2, 3, 5-10] are, perhaps, the most common application of current techniques to find the best match for a query melody. QBH systems let users pose queries to a database by singing or humming. Figure 1 shows a system diagram outlining a typical QBH system. Here, a sung query is transcribed into a string. Targets are indexed by one or more melodic sequences. A similarity ranker compares the transcribed query to the targets in the database and returns a ranked similarity list.

Both query and target are typically represented as a sequence of symbols drawn from a finite alphabet. Such sequences are commonly called strings. The dominant melodic query matching techniques investigated in the literature have been n-grams, edit-distance based string matching, and Markov models.

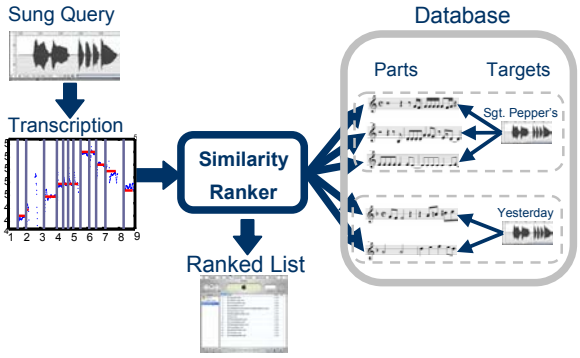

Figure 1. A typical Query-by-humming (QBH) system.

Pickens [11] found n-grams superior to language models, however results from n-grams were only good when the full contents of a piece in the database were passed in as a query. Downie and Nelson [12] performed a systematic investigation of the best length n-gram to use, given queries and targets encoded as sequences of pitch intervals. Uitenbogerd and Zobel [13] compared two kinds of n-gram measures to local string alignment with a fixed match score. Local string alignment was found to beat the performance of n-gram based matching. Other researchers have investigated stochastic methods for music retrieval, particularly Markov models [2, 10, 14] and probabilistic string matchers [9].

With Markov models, a monophonic query is treated as an observation sequence and a theme is judged similar to the query if the associated Markov model has a high likelihood of generating the query. Markov models described in the literature are currently based on monophonic melodic themes.

Existing probabilistic string matchers compare monophonic query strings with monophonic database targets, but take into account possible user error through the use of a *match score matrix.* This matrix determines the edit distance between a query element

and a target element by encoding the probability that a particular substitution would be made, based on prior training examples.

Pardo and Birmingham [9] and Dannenberg, et al. [15] found no statistically significant difference between the precision and recall of systems based on probabilistic string matching and those based on Markov models, while the string matchers consistently performed on the order of 100 times faster on real-world queries.

Given the speed advantage of string matching over Markov models and the performance advantage of string matching over n-gram based techniques, this paper concentrates on extending probabilistic string-matching techniques to a new category of problem: alignment of monophonic queries to polyphonic targets.

2. MULTIPLE ALIGNMENT PROBLEM

Standard string matching [16-19] measures the distance between two strings, S_1 and S_2, as the number of edit operations required to turn S_1 into S_2, given a fixed set of allowable operations. A typical set of edit operations would include deletion of a string element, insertion of an element, and matching between an element of S_1 and an element of S_2. Given a query string and a set of target strings drawn from a database, the closest target is the one with the lowest cost transformation into the query.

This paradigm works well when comparing a monophonic query to a database of monophonic pieces, as the query and the targets are all appropriately represented as strings. However, most tonal music is polyphonic, with multiple concurrent musical lines. Such pieces are not adequately represented by individual strings. The Bach chorale in Figure 2 is a typical example of polyphonic music.

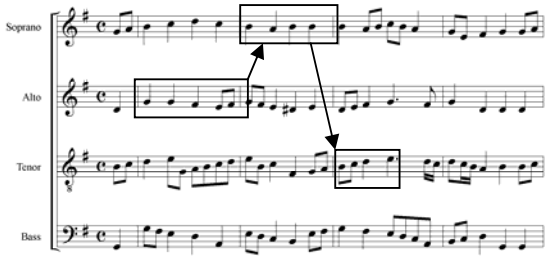

Figure 2. To God On High All Glory Be – Melody: N. Decius, Arrangement: J. S. Bach, measures 1 through 4

Existing string matching methods can be used to compare a monophonic queries to a polyphonic database by keying each polyphonic piece (hereafter referred to as a *score*) with a collection of monophonic parts, corresponding to the individual parts (such as the bass line and vocal line) in the written score. This is illustrated in Figure 1. The query can then be matched against each individual part in the song. The edit distance score for the best-matching part is then used as the similarity estimate for the entire song.

This approach works well as long as the user sings a query drawn only from a single track in the song. However, users often represent polyphonic pieces in their queries by skipping from one part to another. It is not unusual, for example, for a person imitating a rock song to sing a portion of the bass line, interspersed with portions of the melody.

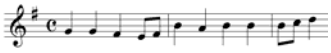

Figure 3. Query based on To God On High All Glory Be

Consider the query expressed as written notation in Figure 3. This query corresponds to the sequence of linked boxes in Figure 2. Current string matching approaches are not designed to handle such a query.

Recently, researchers have explored methods to measure the similarity between polyphonic pieces using n-grams [7]. This work has inspired us to look at approaches to measuring polyphonic similarity using edit-distance based string matching algorithms. We now describe these approaches.

3. DEFINITION OF TERMS

Figure 4 shows the same passage as Figure 2 in piano-roll style notation. Here, each note is indicated by a horizontal line. The length of the line indicates the duration of the note. The horizontal placement indicates the onset time. The vertical placement indicates pitch. The pitch class for each note is indicated by a letter at the onset of the note.

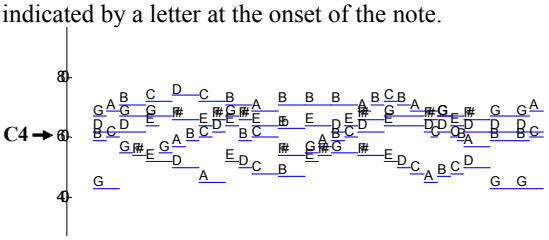

Figure 4. To God On High All Glory Be in piano roll notation

A musical score, in its most generic form, is a collection of *parts*, such as the Soprano and Bass parts in Figure 2. Each part is a collection of notes. The organization of a song as a collection of one or more (possibly polyphonic) parts mirrors that of the standard MIDI Type 1 file [20], in which songs are organized into (possibly polyphonic) tracks. A MIDI Type 0 file contains only a single track.

We define a note $\boldsymbol{n_i}$ by an ordered 3-tuple $(\boldsymbol{s_i}, \boldsymbol{e_i}, \boldsymbol{p_i})$ where $\boldsymbol{s_i}$ is the start time of the note, $\boldsymbol{e_i}$ is the time at which the note ends and $\boldsymbol{p_i}$ is the pitch of the note. For example consider the two-part musical score represented as piano-roll in Figure 5. In the figure, notes in part $\boldsymbol{P_1}$ begin with a circle. Notes in part $\boldsymbol{P_2}$ begin with a diamond. Start times are defined by the order of note onsets. Thus, a start time of 3 indicates the third note onset. The pitch is encoded as MIDI pitch number and listed above the note.

The score and its two parts can be expressed as follows:

$\boldsymbol{S} = \{\boldsymbol{P_1}, \boldsymbol{P_2}\}$

$\boldsymbol{P_1}$ ={(1,3,38), (2,3,36), (4,6,38), (6,8,38)}

$\boldsymbol{P_2}$={(1,2,24), (1,3,22), (3,5,22), (6,7,24)}

The division of notes into parts is something that is routinely available in musical scores encoded as Finale, Sibelius, and MIDI files. Parts captures basic underlying musical structure and we would like our matching algorithms to be sensitive to this information. In the absence of part information, the score can be thought of to be consisting of just one part with all the notes belonging to that single part.

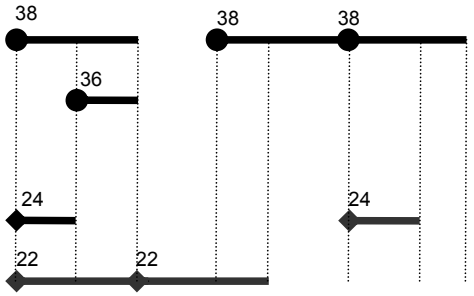

Figure 5. A simple score in piano roll notation

We classify a score into three categories: *monophonic*, *homophonic* and *polyphonic*. A score is monophonic if no new note in the score begins until the current note has finished sounding. In a monophonic score S, there is a total ordering of notes. For every pair of notes $n_i=(s_i,e_i,p_i)$ and $n_j=(s_j,e_j,p_j)$ in S, $s_i \neq s_j$, $e_i \neq e_j$ and if $s_i<s_j$ then $e_i<s_j$.

In homophonic music, notes may sound simultaneously, but they must start and end at the same time. For every pair of overlapping notes (i.e. concurrently sounding notes) $n_i=(s_i,e_i,p_i)$ and $n_j=(s_j,e_j,p_j)$ in a homophonic score**,** it must hold that $s_i=s_j$, and $e_i=e_j$.

If a score is neither monophonic, nor homophonic, it is polyphonic. In this case, there are at least two notes in the score that sound concurrently but either do not end or do no start concurrently, violating the constraints for both monophonic and homophonic scores. The Bach score in Figure 2 is polyphonic.

4. FINDING TARGETS IN A DATABASE

We are interested in finding relevant targets in a database in response to a user query.

A *query,* Q, is a monophonic, one part score. We define a *database*, D, as set of scores $\{S_1,\ldots,S_r\}$. We will consider separately the cases when the scores in the database are monophonic, homophonic or polyphonic.

Given a database of scores, D, and a query, Q, we assume there is a target set $T \subseteq D$ that corresponds to the query. This is the set of scores that the user would like to access using the query. For the purpose of this paper, we assume that T consists of a single correct score and that this score is the one which is most similar to the query. The question then becomes one of defining a similarity function whose value is maximized for the target.

An order may be imposed on the scores in D by measuring the similarity between each score $S \in D$ and the query Q and then ordering the set by similarity. We now define several versions of a similarity function $\sigma(S,Q)$ based on the idea of edit-cost.

4.1 Monophonic Alignment

The typical problem formulation in the query by humming literature assumes the query and all scores are monophonic. If one assumes monophonic scores consist of a single part, one can represent the scores as strings over a suitably defined alphabet and employ standard string matching techniques [9, 21].

Given two strings, $S=s_1s_2\ldots s_n$ and $Q=q_1q_2\ldots q_m$, drawn from alphabet Σ, string matchers find the best alignment between string S and string Q by finding the highest-reward alignment of S to Q in terms of edit operations. (insertion, deletion or matching of characters). The highest-reward is a measure of the similarity between the two strings.

Central to string matching algorithms is the development of a match score function, $\mu(x,y)$, that gives a numerical value corresponding to goodness of the match between two characters, x and y, drawn from the alphabet Σ.

Recent work in the music IR community [19, 21] has used the log-odds approach to determining the match score. Such a match score function is shown in Equation 1.

$$\mu(x,y) = \log\left(\frac{P(x,y \mid match)}{P(x,y \mid chance)}\right) \qquad (1)$$

This function returns a negative value when the probability of a meaningful match is below that of a random co-occurrence. Similarly, the value is positive when a meaningful match is more likely than random chance.

Developing good estimates of the probabilities required for this function can be a challenge and is task-dependent. For an example of how this may be done see [9].

The best alignment of S to Q can be found in $O(mn)$ time Here, m and n correspond to the length of the query and the score. This is done by applying dynamic programming. This has been used for over 30 years to align gene sequences based on a common ancestor [18]. We now describe a local edit-distance variant.

Construct a matrix M, of size $n+1$ by $m+1$, which is indexed from 0,0 to n,m. Element $M_{i,j}$ contains the score of the best alignment between the initial segment s_1 through s_i of S and the initial segment q_1 through q_j of Q.

Initialize M as described in Equation 2 and Equation 3.

$$\underset{0\le j\le m}{M_{0,j}} = 0 \qquad (2)$$

$$\underset{0\le i\le n}{M_{i,0}} = 0 \qquad (3)$$

We define the recurrence relation for M in Equation 4.

$$M_{i,j} = \max \begin{cases} 0 \\ M_{i-1,j} + \mu(-, q_j) \\ M_{i,j-1} + \mu(s_i, -) \\ M_{i-1,j-1} + \mu(s_i, q_j) \end{cases} \quad (4)$$

Here, the "-" character is a blank, which is added to the alphabet of both query and score. Matching to a blank can be thought of as skipping the element matched to the blank. The match score function value for matching a blank is set to the penalty for skipping the element matched to the blank. Skipping an element of the query assumes the query inserted an extra element that was not in the score. Skipping an element of the score assumes that the query deleted that element of the score.

The similarity of the best alignment of ***S*** to ***Q*** is then defined as the highest valued element of ***M***.

$$\sigma(S,Q) = \max_{i,j}(M_{i,j}) \quad (5)$$

4.2 Maximum Single-part Similarity

It is often the case that a user wishes to find a musical piece consisting of multiple parts, querying the database with a monophonic query. Given a multi-part score, as in Figure 2, the easiest way to measure similarity to the query is to use the similarity of its maximally similar part. This is shown in Equation 6.

$$\sigma(S,Q) = \max_{\forall P \in S}(\sigma(P,Q)) \quad (6)$$

This approach is simple to add to any existing similarity measure. The time complexity of this algorithm is ***O*(*tr*)**, where ***t*** is the time required to calculate similarity for a single part and ***r*** is the number of parts. When applied to the monophonic alignment similarity measure from Section 4.1, the time complexity is ***O*(*mnr*).** Here, we assume that the length of the score is the length of the longest part.

4.2 Homophonic Alignment

It is not unusual, for a query to be composed of sections derived from multiple parts in the desired score. The optimal alignment of such a query travels from part to part in the score. For example, a person may sing a bass line, interspersed with portions of the soprano. Maximum single-part similarity does not capture this situation and may fail to produce the highest score for the correct target.

As a first step to handling a query whose optimal alignment travels from part to part, we consider the case of a homophonic score. We can extend the dynamic programming algorithm of Section 4.1 as follows. Instead of each symbol s_i in the string representation of the score representing a single note ***n***, let it represent a note *concurrency*. We define a note concurrency, ***C***, as a set of notes that share the same onset and end time. Each homophonic score in the database may, thus, be represented a sequence of note concurrencies, $S=C_1C_2\ldots C_l$.

To compare the query to the score with alignment algorithm in Equation 4, one must replace the match score function between two notes, $\mu(x,y)$, with a match score function, $\mu_H(x,C)$, for scoring a note ***n*** and a note concurrency ***C*** . Equation 7 defines this.

$$\mu_H(n,C) = \max_{\forall \alpha \in C}(\mu(n,\alpha)) \quad (7)$$

Here, the similarity of the concurrency to the note is determined by the similarity of its most similar note. As with the method from Section 4.1, the similarity score of the best alignment of ***S*** to ***Q*** is the highest valued element of ***M***.

By moving the maximization into the match score function (as opposed to simply selecting the value for the most-similar part), we increase the similarity between a query that travels from part to part and the correct score.

The time complexity of the match score function in Equation 7 depends on the number of notes in the concurrency, ***c***, and is ***O*(*c*)**. If the size of the score, ***n***, is the number of concurrencies in the score and we take ***c*** to be the size of the largest concurrency in the score, then the time complexity is ***O*(*mnc*)**.

This alignment method may also be combined with the method from Section 4.2. The resulting method allows comparison of a monophonic query to a score with multiple homophonic parts.

4.4 Polyphonic Alignment

Homophonic alignment, while an improvement over combining maximum single-part similarity with monophonic alignment, has two weaknesses. It does not account for polyphonic scores, where notes begin and end independently, nor does it capture the concept of a part. We would like the flexibility to impose a penalty on the match for skipping between parts. This lets us preferentially favor an alignment that continues on the same part over one which skips to a new part, while still allowing such skips to take place. We start the description of the polyphonic alignment algorithm by first treating the simpler homophonic case.

Let ***S*** be a multi-part homophonic score with ***r*** parts $P_1,\ldots,P_r$ where each part has the same number of note concurrencies**.** Note that this may require insertion of some silent note concurrencies, i.e. note concurrencies which only contain pitch 0. Let $P_i=p_{i1}\ldots p_{in}$ where p_{ij} is the j^{th} concurrency of P_i. Let $Q=q_1q_2\ldots q_m$ be the query sequence where q_k is the k^{th} note of ***Q***. For a sequence $A=a_1a_2\ldots a_l$, we will use $A[i..j]$ to represent the substring $a_ia_{i+1}\ldots a_j$.

We extend the dynamic programming algorithm for homophonic single-part alignment by introducing a function $\gamma(a,b)$ which returns the cost for changing from P_a to P_b. Let S_j denote a length ***j*** prefix of ***S*** which is the set of parts $P_1[1..j], P_2[1..j],\ldots, P_r[1..j]$ and Q_j denote the length ***j*** prefix of ***Q*** which is $Q[1..j]$.

Next we describe the recurrence which uses $\gamma(a,b)$ to score alignment of a monophonic query with a multiple-part homophonic scores. Instead of a two dimensional table as in Section 4.1, we will construct a three dimensional table where the third dimension

corresponds to the different parts in the multi-part score.

Construct a ***(m+1)× (n+1)× r*** matrix ***L***, indexed from ***0,0,1*** to ***m,n,r***. Here, $L_{i,j,k}$ is the score of the optimal local alignment score of Q_i and S_j such that p_{kj} is present in the alignment.

Initialize *L* as described in Equation 8 and Equation 9.

$$L_{\substack{0,j,k \\ 0\le j\le n, 1\le k\le r}} = 0 \tag{8}$$

$$L_{\substack{i,0,k \\ 0\le i\le m, 1\le k\le r}} = 0 \tag{9}$$

We define the recurrence relation for *L* in Equation 10.

$$L_{i,j,k} = \max\begin{cases} L_{i-1,j,k} + \delta(q_i,-) \\ L_{i,j-1,k} + \delta(-,p_{k_j}) \\ L_{i-1,j-1,k} + \delta(q_i,p_{k_j}) \\ L_{i,j-1,l} + \delta(-,p_{k_j}) + \gamma(l,k)\ \forall l \neq k \\ L_{i-1,j-1,l} + \delta(q_i,p_{k_j}) + \gamma(l,k)\ \forall l \neq k \\ 0 \end{cases} \tag{10}$$

The similarity score of the best alignment of ***S*** to ***Q*** is then the highest value element of ***L***.

$$\sigma(S,Q) = \max_{i,j,k}(L_{i,j,k}) \tag{11}$$

In polyphonic scores, we have to deal with the added complication that notes may begin and end independently. Thus, it may be possible to skip between parts in the middle of a sounding note, at the point where a note in another part begins. This happens throughout the Bach example in Figure 2, starting with the very first beat. To account for this we break down the notes into *notebits*.

A currently-sounding note, ***n***, us sokut ubti twi bitebuts at the point where another note begins or ends. It is, perhaps, easiest to see this in Figure 5. Here, vertical lines are placed at every note onset and ending. These lines subdivide the notes in the score into notebits. For example, the first note in part P_1 is divided into two notebits. The first of these corresponds to the *onset* of the note. The second is a continuation of the note as a note in the other part enters.

In our notation, a note ***n*** is decomposed into a sequence of notebits, $n=b_1,b_2,\ldots,b_k$. Each notebit is a 4-tuple **(*s,e,p,o*)** where ***s*** is the start time of the notebit, ***e*** is the end time, ***p*** is the pitch and ***o*** is a Boolean value that defines whether the notebit corresponds to the *onset* of the note. The field ***o*** of a notebit is true only if it is an onset notebit. The first note in part P_1 is ***n*=(1,3,38)**. This is decomposed as $n = b_1,b_2$ = **(1,2,38,*true*), (2,3,38,*false*)**.

Up to this point, we have not defined silence in a score. We do so by allowing notes with pitch 0, defining 0 to mean "silence." This allows us to divide a part into a sequence of consecutive notes, some of which may be silence Each note in a part may be further decomposed into notebits.

Given a score composed of monophonic parts, all parts will have the same number of notebits, since the number of notebits in any part is determined by the number of independent note onsets and endings in the entire piece (including all parts). Our algorithm depends on all parts containing the same number of notebits, as it implicitly uses order in the sequences as its encoding of relative time. Therefore, the *i*th notebit in part P_j is coincident with the *i*th notebit in part P_k, for all ***j*** and ***k***.

Given a score with polyphonic parts, we break each part into monophonic subparts, such that each subpart has no overlapping notebits and all notebits from the same note belong to the same subpart. Note that each part gets divided into ***z*** subparts, where ***z*** is the maximum number of simultaneously sounding notes in the part. In Figure 5, *z*=2 for both parts. For the example in Figure 5, the decomposition of notes into notebits yields the following (here, we substitute ***t*** for ***true*** and ***f*** for ***false***).

P_{11}={(1,2,38,t), (2,3,38,f), (3,4,0,t), (4,5,38,t), (5,6,38,f), (6,7,38,t), (7,8,38,f)};

P_{12}={(1,2,0,t), (2,3,36,t), (3,4,0,t), (4,5,0,f), (5,6,0,f), (6,7,0,f), (7,8,0,f)};

P_{21}={(1,2,24,t), (2,3,0,t), (3,4,0,f), (4,5,0,f), (5,6,0,f), (6,7,24,t), (7,8,0,t)}; and

P_{22}={(1,2,22,t), (2,322,f), (3,4,22,t), (4,5,22,f), (5,6,0,t), (6,7,0,f), (7,8,0,f)}.

Notice that the in the notebit representation, polyphonic scores look very similar to the homophonic scores with ***N*** parts, where ***N*** is the total number of subparts in the score (for our running example, ***N***=4). Therefore we can create a convenient representation of the score in terms of set of ***N*** strings $S_1,\ldots,S_N$ where $S_i=s_{i1}\ldots s_{in}$ and s_{ij} is the ***j***th notebit in subpart S_i. Let p_{ij} be the pitch corresponding to the notebit s_{ij} and let o_{ij} represent its corresponding boolean onset field. We set up the dynamic programming table for polyphonic alignment in a similar fashion and use the following recurrence where the change-track penalty for two subparts derived from the same original part is set to be 0. This is expressed in Equation 12.

if o_{kj}= **false**, $L_{i,j,k} = L_{i,j-1,k}$,

if o_{kj} = **true,**

$$L_{i,j,k} = \max\begin{cases} L_{i-1,j,k} + \delta(q_i,-) \\ L_{i,j-1,k} + \delta(-,p_{k_j}) \\ L_{i-1,j-1,k} + \delta(q_i,p_{k_j}) \\ L_{i,j-1,l} + \delta(-,p_{k_j}) + \gamma(l,k)\ \forall l \neq k \\ L_{i-1,j-1,l} + \delta(q_i,p_{k_j}) + \gamma(l,k)\ \forall l \neq k \\ 0 \end{cases} \tag{12}$$

For all the alignment algorithms described in this paper, we need to fill up a dynamic programming table. Computing each entry in the table takes ***O(1)*** time for monophonic case and ***O(N)*** time for the

polyphonic case. So the total time taken is the product of time taken per entry and the total number of entries. Therefore monophonic alignment takes $O(mn)$ time and the polyphonic alignment takes $O(mnN^2)$ time, where m is limited by the number of notes in the query, n is limited by the number of notes in the score, and N is the number of subparts in the score.

We have described a series of alignment algorithms, culminating in the polyphonic alignment algorithm in Equation 12. This algorithm is the most general method we have described, allowing for alignment of a monophonic query that skips from part to part of an arbitrary polyphonic score. In Section 5, we measure the performance improvement this algorithm provides over the maximum single-part similarity measure in Section 4.2.

5. EXPERIMENTAL SECTION

In order to estimate the potential performance gain for finding the appropriate polyphonic, multi-part target in a database in response to a part-skipping monophonic query, we constructed a small corpus of Bach Chorales and a set of synthetic queries that skip from part-to-part. We then compared the performance of a similarity measure based on the polyphonic alignment algorithm from Section 4.4 with the maximum single-part similarity measure from Section 4.2. This experiment is described in this section.

As our database, we chose 300 Bach four-part chorale harmonizations, encoded as MIDI files. These are typical sopranto-alto-tenor-bass vocal arrangements, and a small subset of them are alternate harmonizations of the same melody. The midi files are available at http://www.jsbchorales.net/. The complete list of chorales selected for this study is available at http://www.cs.northwestern.edu/~pardo.

While three hundred chorales makes for a small corpus, the point of the experiment is not to measure absolute performance of a single method, but rather relative performance improvement. For this reason, we felt a smaller database, composed of known pieces with full scores was a better choice.

We are interested in creating algorithms that allow for effective comparison of monophonic queries to polyphonic, multi-part scores. For this experiment, we were interested in measuring the relative performance of these methods as a query is increasingly likely to skip between parts. In order to control this likelihood we constructed synthetic queries, based on targets in the corpus.

Given a target score in the database, T, a query was constructed by selecting a subsequence of notes from T whose length was randomly selected (with an equal probability distribution) from the range [5,25]. This length range was based on the range of typical query lengths for sung queries[9]. The initial note was selected from a randomly-chosen part (given an equal probability distribution) in the target. The start position in the selected part was also chosen randomly, so that a query might begin anywhere within the piece. Once started, the query was constructed by adding consecutive notes from the score. The part of each additional note in the query was selected based on a fixed probability of changing parts. If the probability of changing parts was set to 0, then all notes in the query would be selected from the same part. If the probability of changing part were 0.25, then there would be a 25% chance that each additional note in the query would be drawn from a different part than the previously selected note. Given a change in part from the current to the next note in the query, the new part was selected at random. The query in Figure 2 and Figure 3 is an example query drawn from the score with a probability of changing notes of 0.25.

5.1 The Experiment

The maximum single-part similarity algorithm from Section 4.2 forms a simple baseline performance measure, as it implicitly assumes the query is drawn from a single, monophonic part. Let c be the probability that a query skips from part to part. As c increases, the similarity measure based on the maximal single-part similarity should become increasingly ineffective. Conversely, the polyphonic alignment method from section 4.4 should be unaffected by an increased amount of skipping from part to part. Accordingly, we compared these two methods for determining the similarity of a query to each score in the database.

To construct the query set, we selected 150 targets at random from the database. For each target, we constructed five queries, one with $c = 0$, one with $c = 0.25$, one with $c = 0.5$, one with $c = 0.75$, and one with $c = 1.0$. This created a total of 750 queries. We then ranked the similarity of every target in the database to each query, recording the rank of the correct target (hereafter called "right rank").

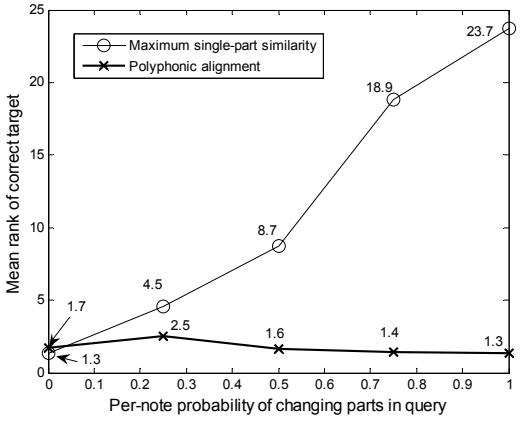

Figure 6. Mean rank of correct target as a function of *c*

Figure 6 shows the mean right rank as a function of c. Each point indicates the mean value for 150 synthetic queries. The number by each point gives the mean value for that point. Since there are 300 scores in the database, random performance would result in a mean right rank of 150. Perfect performance would result in a mean right rank of 1.

As the figure shows, neither method performed perfectly, even when the probability of changing

parts in a query is 0. This is due to two factors: first, several of the targets are alternate harmonizations of the same melody; second, a short query (on the order of 5 notes) may match multiple items in the database, if it is based on a common melodic pattern. This is the case for a number of the queries.

As the probability of changing parts increases, however, the difference between the performances of the algorithms becomes clear. The performance of the maximum single-part similarity measure quickly degrades as the queries increasingly skip between parts, while the polyphonic alignment algorithm's performance remains essentially constant, with its mean varying within a narrow range.

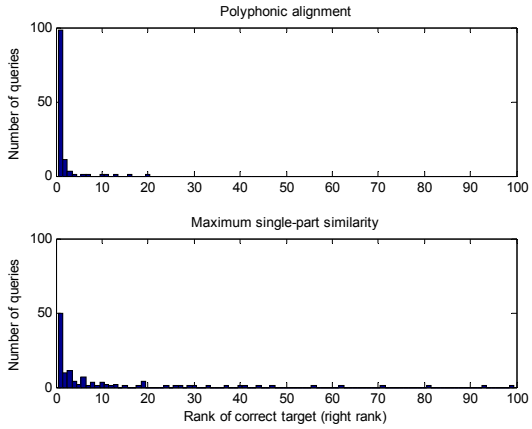

Figure 7. Performance of ranking methods when $c = 0.5$

Figure 7 gives more detail. The figure shows a histogram of the distribution of right ranks for both algorithms in the case where c, the probability of changing parts at each note, is 0.5. The distribution returned by the polyphonic alignment algorithm much tighter, with the vast majority of the 150 queries returning a right rank of 1. Only a third of the queries do so for the maximum single-part similarity method, and the tail of right ranks extends out to the value 100.

6. CONCLUSIONS

Finding the best matching database target to a melodic query has been of great interest in the music IR world. Standard string alignment algorithms work well for this task when comparing a monophonic query to a database of monophonic pieces. However, most tonal music is polyphonic, with multiple concurrent musical lines. Such pieces are not adequately represented as strings. Moreover, users often represent polyphonic pieces in their queries by skipping from part to part. We described a series of algorithms designed to compare the similarity of a monophonic melodic sequence to a homophonic or polyphonic piece of music, culminating in the polyphonic alignment algorithm in Equation 12. This algorithm is the most general method we described, allowing for alignment of a monophonic query to an arbitrary polyphonic score.

We compared the polyphonic approach to the maximum single-part similarity method for matching a polyphonic score to a monophonic query. Results using synthetic queries on the Bach database indicate that the polyphonic method significantly outperforms the maximum single-part method when a monophonic query is drawn from multiple parts in the target. This suggests that the performance of music information-retrieval systems, such as query-by-humming systems, can be improved through the use of a polyphonic-alignment algorithm.

REFERENCES

[1] Mazzoni, D. and R. Dannenberg. Melody Matching Directly From Audio. in ISMIR. 2001. Bloomington, IN.

[2] Meek, C. and W.P. Birmingham. Johnny Can't Sing: A Comprehensive Error Model for Sung Music Queries. in ISMIR 2002. 2002. Paris, France.

[3] Hoos, H., K. Rentz, and M. Gorg. GUIDO/MIR - an Experimental Musical Information Retrieval System based on GUIDO Music Notation. in International Symposium on Music Information Retrieval. 2001. Bloomington, IN.

[4] Song, J., S.Y. Bae, and K. Yoon. Mid-Level Music Melody representation of Polyphonic Audio for Query-by-Humming System. in ISMIR 2002. 2002. Paris, France.

[5] McNab, R.J., L.A. Smith, and e. al. Towards the digital music library: tune retrieval from acoustic input. in Digital Libraries. 1996.

[6] Clausen, M., R. Englebrecht, and e. al. Proms: A web-based tool for searching in polyphonic music. in The International Symposium on Music Information Retrieval. 2002.

[7] Doraisamy, S. and S. Ruger. A Comparative and Fault-tolerance Study of the Use of N-grams with Polyphonic Music. in ISMIR 2002. 2002. Paris, France.

[8] Clarisse, L.P., et al. An Auditory Model Based Transcriber of Singing Sequences. in ISMIR 2002. 2002. Paris, France.

[9] Pardo, B., W.P. Birmingham, and J. Shifrin, Name that Tune: A Pilot Study in Finding a Melody from a Sung Query. Journal of the American Society for Information Science and Technology, 2004. 55(4): p. 283-300.

[10] Birmingham, W.P., et al. Musart: Music Retrieval Via Aural Queries. in ISMIR 2001. 2001. Bloomington, IN.

[11] Pickens, J. A Comparison of Language Modeling and Probabilistic Text Information Retrieval. in International Symposium on Music Information Retrieval. 2000. Plymouth, Massachusetts.

[12] Downie, S. and M. Nelson. Evaluation of a Simple and Effective Music Information Retrieval Method. in 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 2000. Athens, Greece.

[13] Uitdenbogerd, A. and J. Zobel. Melodic Matching Techniques for Large Music Databases. in Seventh ACM International Conference on Multimedia. 1999. Orlando, FL.

[14] Durey, A.S. and M. Clements. Melody Spotting Using Hidden Markov Models. in International Symposium on Music Information Retrieval. 2001. Bloomington, IN.

[15] Dannenberg, R., et al. The MUSART testbed for query-by-humming evaluation. in ISMIR 2003, 4th International Conference on Music Information Retrieval. 2003. Baltimore, Maryland.

[16] Gotoh, O., An improved algorithm for matching biological sequences. Journal of Molecular Biology, 1982. 162: p. 705-708.

[17] Gusfield, D., Algorithms on Strings, Trees, and Sequences. 1997, New York, NY: The Press Syndicate of the University of Cambridge.

[18] Needleman, S.B. and C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 1970. 48: p. 443-453.

[19] Pardo, B. and W.P. Birmingham. Following a musical performance from a partially specified score. in Multimedia Technology Applications Conference. 2001. Irvine, CA.

[20] MIDI-Manufacturers-Association, The Complete MIDI 1.0 Detailed Specification. 1996, Los Angeles, CA: The MIDI Manufacturers Association.

[21] Hu, N., R. Dannenberg, and A. Lewis. A Probabilistic Model of Melodic Similarity. in International Computer Music Conference (ICMC). 2002. Goteborg, Sweden: The International Computer Music Association.