Sweepline and Recursive Geometric Algorithms for Melodic Similarity

Kjell Lemström, Niko Mikkilä, Veli Mäkinen and Esko Ukkonen
C-BRAHMS Group, Department of Computer Science
P.O.Box 68 (Gustaf Hällströmin katu 2b)
FIN-00014 University of Helsinki, FINLAND
{klemstro,mikkila,vmakinen,ukkonen}@cs.helsinki.fi

Abstract

This extended abstract gives an overview on two content-based retrieval algorithms for symbolic music, developed earlier in the C-BRAHMS group [1], that took part in the Symbolic Melodic Similarity and Query by Singing/Humming tasks of the MIREX 2006 contest [3, 4]. Given two excerpts of symbolically encoded monophonic or polyphonic music, the *query pattern* and the *target music*, the purpose of these algorithms is to find musically relevant occurrences of the query pattern within the target music.

Keywords: MIREX 2006, Melodic Similarity, Geometric Matching

1. Introduction and Background from MIREX 2005

In the previous MIREX contest organized in 2005 we submitted both a basic monophonic string-matching algorithm and the same geometric algorithm that is described in this abstract. Last year the Symbolic Melodic Similarity (SMS) task only included monophonic music from the RISM A/II collection of incipits and the results were compared to a human-generated ground truth. Both our string-matching algorithm and the more complex geometric algorithm seemed to work equally well, with the geometric algorithm performing only slightly better. One of the reasons for that performance was probably the fact that the string-matching algorithm does not use any rhythmic information at all and the geometric method is not time-scale (tempo) invariant; it requires that both the query and target melodies are played at the same speed.

This year in MIREX 2006 there were two Symbolic Melodic Similarity subtasks: a monophonic task based on the RISM collection and both exact (quantized in pitch and rhythm) and hummed queries; and a similar polyphonic task based on MIDI files harvested from the Internet [3]. We were especially interested in the polyphonic task and therefore we submitted the same geometric P3 algorithm that we submitted last year, only this time as two entries: the original one (P3) and a brute force tempo scaling version (ScaledP3) that runs the original algorithm multiple times with the pattern scaled in time by predefined constant factors and retrieves the best match across the runs. An overview of these algorithms is given in section 2.

We also intended to submit ScaledP3 to the Query by Singing/Humming (QBSH) task but after testing various approaches with the data set, we decided to use a recursive exhaustive search algorithm instead. The MIDI data available represented some challenges and the alternative would have been to process the queries with our own melody extraction method which we do not have yet. The ES algorithm is described in section 3.

2. Sweepline Algorithm

Our submission to the SMS task is based on a geometric sweepline technique that is applied on a piano-roll type representation of the musical score [2]. The intuition behind this algorithm is to slide the bar-lines representing the query over the piano-roll representation of target and to find the position that gives the maximal common shared time (see Figures 1 and 2).

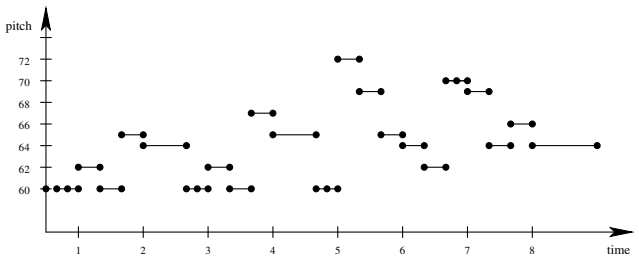

Figure 1. Query in piano-roll representation.

To this end, the piano-roll representations of the query and the target music are given to the algorithm as lexicographically ordered turning points that are calculated based on the start and end points of the bar-lines representing the query and the target.

The algorithm first populates a priority queue with two translation vectors for each turning point in the pattern. In the beginning the vectors point to first starting and ending point in the target. After this initialization the algorithm loops through all possible translation vectors between the

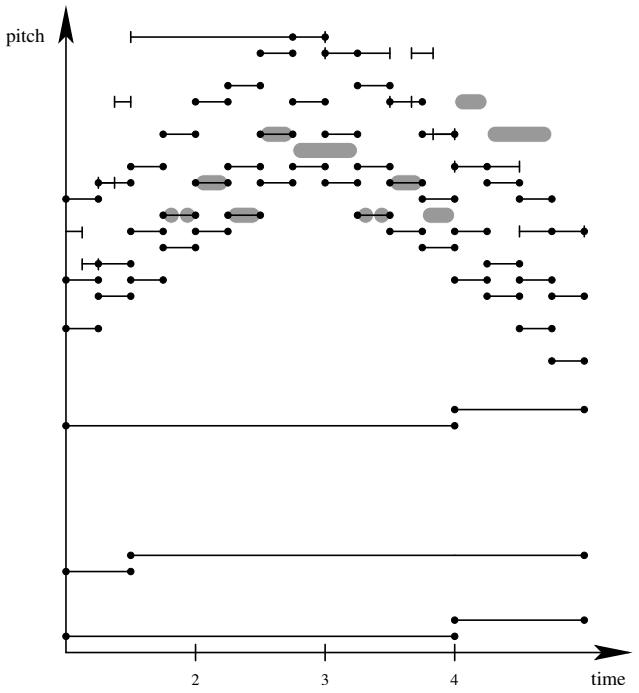

Figure 2. Target in piano-roll representation. The first twelve notes of the query in Figure 1 are shown by shading in a translated position such that the total length of the overlapping is six quarter notes.

query and the target. At each iteration the first vector in lexicographic order is retrieved from the queue and replaced by the next corresponding vector (a vector for the same turning point in the query) that has not yet been inserted into the queue.

When the algorithm iterates over translation vectors, it counts common time between the query and the target on each vertical translation level to ensure transposition invariance. The time is counted by using a linear slope that the translation vectors adjust, and the maximal overlapping is simply checked for at each iteration. Finally, normalizing the maximal overlapping by the combined length of query or target notes (whichever is smaller) results in a value that expresses the similarity of the query and the target.

The brute force pattern-scaling version of the algorithm simply uses the method described above multiple times with the notes in either the pattern or the target scaled in time. We used scaling factors 0.5, 0.667, 0.8, 1.0, 1.25, 1.5 and 2.0. The selection of these values is not based on extensive experimentation, so they are probably not the optimal factors for the task. Heuristics could perhaps be used to select a suitable scale or to at least limit the range.

The overall best match is chosen simply by comparing normalized overlap lengths of the best matches returned from the runs with each scaling factor. With such a low number of factors, good matches usually stand out from the rest. Using more scaling factors would allow smoother matching, but the algorithm execution time would also quickly increase unbearably. This brute force method should give slightly better results than the original one whenever note timing in the query is not expected to be the same as timing in the potential matches, just like in the SMS and QBSH tasks. Of course, this will not help much with tempo changes that may occur within hummed queries.

P3 runs in $O(mn \log m)$ time where m and n denote the number of musical events (notes) in query and target, respectively. The scaled version of the algorithm increases the execution time by a constant factor of 7.

3. Recursive Algorithm

The ES algorithm that we submitted to the QBSH task also uses a piano-roll representation of music, but it does not maximize the overlapping in the same way as P3 does. Instead it performs an exhaustive depth-first search trying to scale the pattern note-by-note to 'fit' the target song, with costs applied to local time-scaling, note duration changes and pitch-shifting. Clearly irrelevant branches are cut with simple heuristics while searching, which keeps the average running time in an usable range, although the worst case time complexity is $O(n^m)$.

First the algorithm divides the piano-roll representation into tiles that have a height of one MIDI pitch level and a width chosen so that there would not be many notes in one tile. A pointer to the first note that starts in each tile is stored to a table and subsequent notes at the same pitch level are linked together. This tile table is used for hashing: quickly finding notes that start within a specific range in the target music. The tile table size is a compromise between quick lookups and space consumption. We used a static tile length of 100 ms but a more optimal value for each pitch level could be chosen by scanning through the target music.

Next the target music is searched for the best occurrance of the pattern by checking recursively for a match at each note in the music, starting with each note in the pattern. For note T_i in the target music and note P_j in the pattern, the recursive check is started by calculating the expected pitch and starting time interval of the next matching note, or multiple notes when gaps are allowed in the matches. All potentially matching notes are looked up from the tile table, match score is updated and the same check is executed recursively for each of the notes, starting at the next position in the pattern.

The most adjustable part of the algorithm is the way how the following potentially matching notes are picked and scored at each recursion level. This procedure can be weighted by the already matched part of the pattern or it can be done independently for each position. To calculate the expected pitch level, we simply take the pitch interval between P_{j+1} and P_j, and add that to the pitch of T_i. Similarly, the difference between start times of the consecutive notes in the

pattern is scaled in proportion to previously matched notes and added to the start time of T_i.

Pitch and tempo shifts are handled by retrieving all notes within a certain range from the expected position: $\pm$ 2 pitch levels and the delta time scaled by 0.5 – 2.0. Notes that start outside this area are not considered further at that point of recursion. Each melody line that continues from the retrieved notes is checked recursively and the match scores are updated. Notes that are closest to the expected note position and have similar duration to the corresponding note in the pattern receive the best score. This is done by multiplying together factors derived from all these differences. 1.0 is a perfect match of a note and 0 is a complete mismatch. Therefore the whole pattern has a maximal score of $m - 1$, and match scores are normalized by dividing them with this value.

4. Results and Analysis

In this section we analyze results from the two MIREX 2006 tasks that our algorithms competed in. More information about the tasks and evaluation methods can be found through task descriptions in the MIREX Wiki. Abstracts from all the participants are published in the result pages. [3, 4]

4.1. Symbolic Melodic Similarity

Our algorithms were at the tail of the competition in all the Symbolic Melodic Similarity tasks (see Figures 3, 4 and 5). Comparing the results of our two P3 variations and looking at the results from last year might suggest that there was a problem within our implementation this year that did not surface as strongly last time. The difference may certainly come from different task setup, but since the RISM A/II collection was used both times and other algorithms performed much better, we might have mistuned something.

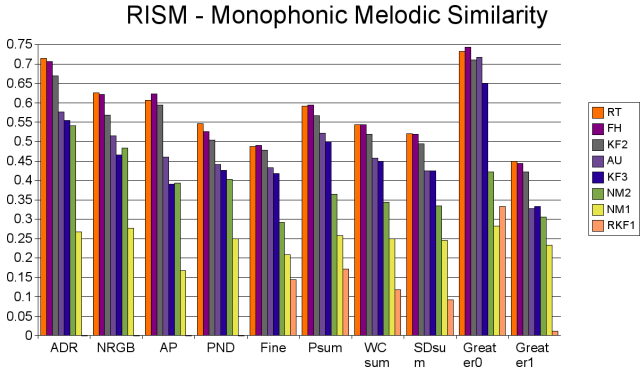

Figure 3. Summary of MIREX 2006 Symbolic Melodic Similarity Monophonic (RISM) task results. The teams and algorithms are FH: Pascal Ferraro and Pierre Hanna, NM1: our original P3 algorithm, NM2: scaled P3, RT: Rainer Typke, Frans Wiering and Remco C. Veltkamp, KF: Klaus Frieler and AU: Alexandra Uitdenbogerd. We thank Rainer Typke for providing these graphs.

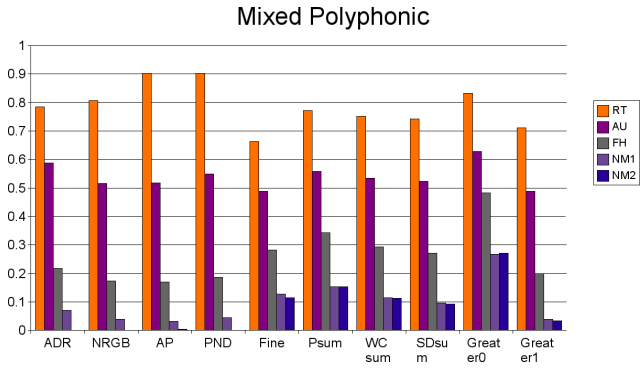

Figure 4. SMS Mixed Polyphonic task results.

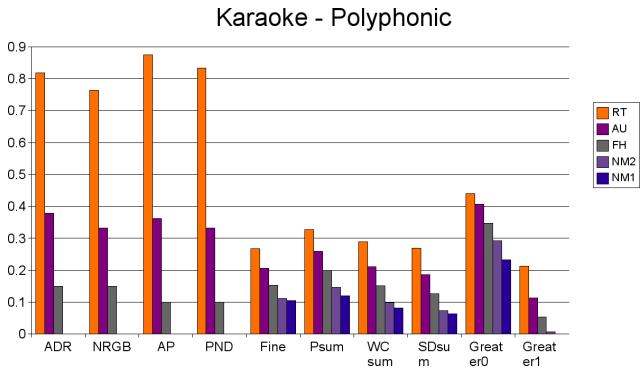

Figure 5. SMS Karaoke Polyphonic task results.

One possible source of problems with the RISM data is the normalization of common time in a match which in our current implementation allows incipits with only a few notes to match the queries better than longer incipits that humans would consider best matches. We first used this normalization scheme last year, when it seemed to work better than simply dividing the common time by pattern duration. After the MIREX 2006 results were published, we compared these two normalization approaches with the evaluation data, and there is a clear difference in favor of the simple normalization by pattern duration. This only affects the results when matching short queries in a database of potentially even shorter incipits, such as the RISM collection. For straight similarity comparison of two songs, the normalization we used is a valid approach.

The poor performance of our submissions in the polyphonic SMS tasks (Figures 4 and 5) was most likely caused by our decision to ignore all track and channel information available in the MIDI files in the algorithm implementation. With the track information in place, the task can be reduced to almost monophonic similarity comparison where simple dynamic programming methods work well. Of course it could be argued that in the real world the track information is usually available, but then again, there are cases where

	XW1	XW2	RJ	RL	**NM**	CS1	RT2	CS3	AU2	CS2	AU1	RT1
Task I (MRR)	0.926	0.900	0.883	0.800	**0.688**	0.568	0.390	0.348	0.288	0.283	0.205	0.196
Task II (Mean Prec.)	-	-	0.926	-	**0.722**	0.587	0.401	0.415	0.238	0.649	0.163	0.468

Table 1. Query by Singing/Humming task results. We refer the reader to the task description for more information about the test collections and the evaluation method [4]. The teams are AU: Alexandra Uitdenbogerd, CS: Christian Sailer, FH: Pascal Ferraro and Pierre Hanna, NM: us with the ES algorithm, RJ: J.-S. Roger Jang, Nien-Jung Lee and Chao-Ling Hsu, RL: Ernesto Lopez and Martin Rocamora, RT: Rainer Typke, Frans Wiering and Remco C. Veltkamp and XW: Xiao Wu and Ming Li.

the searched pattern is not constrained to one track – here it was in majority of the relevant files considering the queries used. There are also sources such as automatic polyphonic transcriptions of audio recordings, where the instrument information may not even be available at all.

The P3 algorithm is computationally efficient but it does not use any indexing, so searching large databases can be slow. In MIREX 2005, parsing the short MIDI incipits probably took most of the time the algorithm was measured to run for and therefore the execution time of the actual algorithm was not clear. Overall it was the fastest one along with our DP algorithm. This year there were much longer pieces of music in the polyphonic task and the indexing and query execution times were separated for those algorithms that support indexing. It is clear that an indexing scheme is necessary in most applications that require searching massive music collections, even though fast on-line algorithms have their uses as well. Indexing polyphonic music fully and efficiently is still a big challenge.

4.2. Query by Singing/Humming

The ES algorithm that we submitted to the QBSH task is an experimental brute force implementation of ideas that may be useful in symbolic melodic similarity in general when implemented more efficiently. Currently it is too slow for polyphonic matching with patterns longer than 10-20 notes, although it could be used for n-gram searches or index construction. The results from QBSH (see Table 1) are quite encouraging since this algorithm performed fairly well even with the imperfect MIDI queries supplied, while the other purely symbolic algorithms (AU, FH and RT) had more difficulties with them.

Overall, these two MIREX tasks were a teaching experience on music information retrieval from large databases. We will have to consider richer approaches for weighing the potentially matching notes, like the ES algorithm does, but hopefully with much lower time complexity. If that proves to be impossible or difficult, we could use a multi-level approach where a simple and fast algorithm retrieves a long list of potential matches and then a slower algorithm filters those results to pick the best matches. A similar approach might be applicable to polyphonic indexing.

5. Acknowledgments

Our best thanks to IMIRSEL for organizing MIREX 2006, to Rainer Typke and Anna Pienimäki for their work on the Symbolic Melodic Similarity task and to J.-S. Roger Jang and J. Stephen Downie for the Query by Singing/Humming task. We much appreciate the opportunity to compare various music information retrieval methods in a controlled environment and hope that MIREX will continue for years to come.

References

[1] K. Lemström, V. Mäkinen, A. Pienimäki, M. Turkia and E. Ukkonen, "The C-BRAHMS Project," in *ISMIR 2003 Fourth Int. Conf. on Music Inf. Retr. Proc.*, Oct. 2003, pp 237-238, See: http://www.cs.helsinki.fi/group/cbrahms/

[2] E. Ukkonen, K. Lemström and V. Mäkinen, "Sweepline the Music!." *Computer Science in Perspective — Essays dedicated to Thomas Ottmann*, vol 2598, pp 330-342, Springer-Verlag, 2003

[3] R. Typke and Anna Pienimäki, "Symbolic Melodic Similarity," [Web site] 2006, Available: http://www.music-ir.org/mirex2006/index.php/Symbolic_Melodic_Similarity

[4] J. S. Downie, J.-S. Roger Jang and R. Typke, "Query by Singing/Humming," [Web site] 2006, Available: http://www.music-ir.org/mirex2006/index.php/QBSH:_Query-by-Singing/Humming