DYNAMIC PROGRAMMING IN TRANSPOSITION AND TIME-WARP INVARIANT POLYPHONIC CONTENT-BASED MUSIC RETRIEVAL

Mika Laitinen
Department of Computer Science
University of Helsinki
mikalait@cs.helsinki.fi

Kjell Lemström
Department of Computer Science
University of Helsinki
klemstro@cs.helsinki.fi

ABSTRACT

We consider the problem of transposition and time-warp invariant (*TTWI*) polyphonic content-based music retrieval (CBMR) in symbolically encoded music. For this setting, we introduce two new algorithms based on dynamic programming. Given a query point set, of size m, to be searched for in a database point set, of size n, and applying a search window of width w, our algorithms run in time $O(mnw)$ for finding exact *TTWI* occurrences, and $O(mnw^2)$ for partial occurrences. Our new algorithms are computationally more efficient as their counterparts in the worst case scenario. More importantly, the elegance of our algorithms lies in their simplicity: they are much easier to implement and to understand than the rivalling sweepline-based algorithms.

Our solution bears also theoretical interest. Dynamic programming has been used in very basic content-based retrieval problems, but generalizing them to more complex cases has proven to be challenging. In this special, seemingly more complex case, however, dynamic programming seems to be a viable option.

1. INTRODUCTION

In this paper we study how to search for excerpts of music in a large database resembling a given query pattern. We allow both the query pattern and the database to be polyphonic. Typically the query pattern constitutes a subset of instruments appearing in the database while the database may represent a full orchestration of a musical piece. The general setting requires methods based on symbolic representation capable of dealing with true polyphonic subset matching; audio-based methods are only applicable to rudimentary cases where queries are directed to clearly separable melodies.

Except for some trivial cases, the straightforward CBMR approach of linear string representation combined with a string matching algorithm does not properly capture the polyphonic CBMR problem. Recently, a more appropriate, geometric modeling of music has been succesfully used by several authors [5–7]. This approach models polyphonic music very naturally, but usually also takes into account another important feature intrinsic to the problem: the matching process ignores extra intervening notes in the database that do not appear in the query. Extra notes may occur because of different polyphonic arrangements, musical decorations and unexpected noise. Recent geometric methods [2, 3, 6] have challenged different timing problems. In the first setting, the occurrences may be tranposed and time-scaled copies of the query [2, 6]. Under the *transposition and time-scale invariance* (the *TTSI* setting), however, the queries need to be given exactly in tempo. In a realistic application local time jittering occur in every note-onset in the query, and a stronger, *transposition and time-warp invariance* is required for a successful matching (the *TTWI* setting). The latter is the setting for our algorithms to be introduced. The first solutions for the TTWI setting was recently presented by Lemström and Laitinen [3].

Our algorithms are based on the pitch-against-time representation of note-on information (see Fig 1). The musical pieces in a database are concatenated in a single geometrically represented file, denoted by T; $T = t_0, t_1, \ldots, t_{n-1}$, where each element $t_j \in \mathbb{R}^2$ for $0 \leq j \leq n-1$ and the elements are sorted in the lexicographic order. Any symbolic music file is convertible in this representation. Later it may be possible to convert audio files and sheet music by using audio transcription and optical music recognition. Although both processes are error prone, it may be the case that the resulting representations are usable due to the robustness of our algorithms against noise. In a typical retrieval case the query pattern P, $P = p_0, p_1, \ldots, p_{m-1}$; $p_i \in \mathbb{R}^2$ for $0 \leq i \leq m-1$, to be searched for is monophonic and much shorter than the polyphonic database T to be searched; our algorithms, however, deal equally well with monophonic and polyphonic input. Sometimes a search window w is

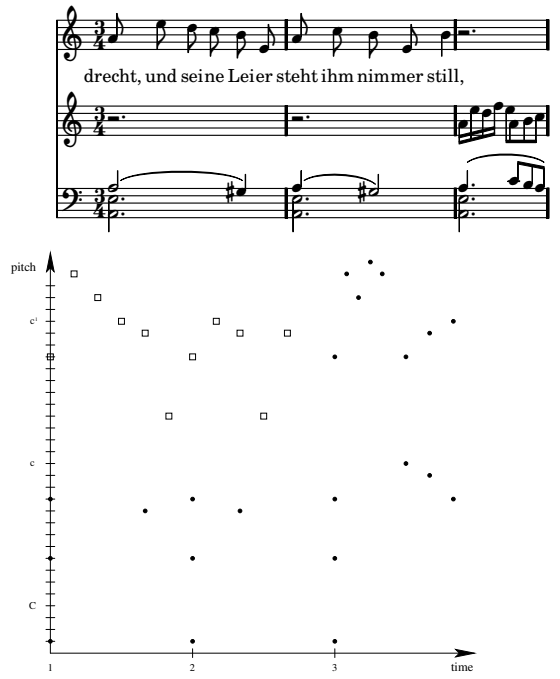

Figure 1. On top, an excerpt from Schubert's *Der Leiermann*. Below, the related point-set representation. The points associated with the vocal part are depicted by squares.

applied and typically $w \leq m$, i.e. $w \leq m \ll n$.

The problems under consideration are modified versions of two problems originally represented in [7]. Below we give the original problems P1 and P2 (pure transposition invariance, *TI*), their transposition and time-scale invariant versions S1 and S2 (*TTSI*), and the transposition and time-warp invariant modifications W1 and W2 under consideration (*TTWI*). For the partial matches in P2, S2 and W2, one may either use a threshold α to limit the minimum size of an accepted match, or to search for maximally sized matches only.

- Find *pure* (P1) / *time-scaled* (S1) / *time-warped* (W1) *translations* of P such that each point in P matches with a point in T.
- Find *pure* (P2) / *time-scaled* (S2) / *time-warped* (W2) *translations* of P that give a partial match of the points in P with the points in T.

Fig. 2 gives six query patterns to be searched for in the excerpt of Fig. 1, exemplifying the six problems P1, S1, W1, P2, S2 and W2 given above.

Ukkonen et al. introduced online algorithms for problems P1 and P2 that run in times $O(mn)$ and $O(mn \log m)$ in the worst case, respectively, and in $O(m)$ additional space [7]. Lemström et al. [4] showed that the practical performance can be improved at least by an order of magnitude by combining sparse indexing and filtering. P2 is known to belong

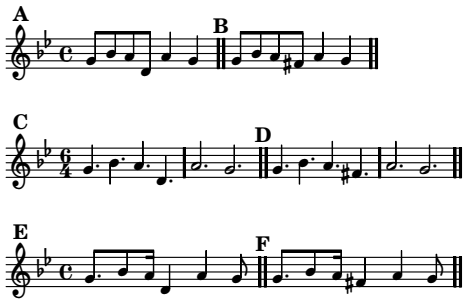

Figure 2. Example queries. For query **A** an occurrence in Fig. 1 would be found in all the six problem cases P1-2, S1-2, W1-2; for **B** in cases P2, S2, W2; for **C** in S1-2, W1-2; for **D** in S2, W2; for **E** in W1-2 and for **F** in W2 only.

to a problem family for which $o(mn)$ solutions are conjectured not to exist. Nevertheless, there is an online approximation algorithm for it running in time $O(n \log n)$ [1].

In [6], Romming and Selfridge-Field gave a geometric-hashing based algorithm for S2 working in time $O(wnm^3)$ and space $O(w^2 n)$. Lemström [2] generalized algorithms P1 and P2 to the time-scaled problems S1 and S2. The algorithms work in $O(m\Sigma \log \Sigma)$ time and $O(m\Sigma)$ space, where $\Sigma = O(wn)$ when searching for exact occurrences and $\Sigma = O(nw^2)$ when searching for partial occurrences.

The first algorithms for W1 and W2 were introduced only very recently in [3]. The sweepline-based algorithms are further generalizations of those above. In this TTWI case the windowing takes an invaluable role; the number of false positives would grow uncontrollably without it. The asymptotic time and space complexities, however, remain the same as with the solution for S1 and S2.

In this paper we introduce new algorithms for the TTWI setting. Our algorithms are based on dynamic programming and their asymptotic worst case complexities are lower than those of the earlier rivals: for the case W1 we have an $O(mnw)$ algorithm; for the W2 case our algorithm runs in time $O(mnw^2)$. In our experiments, however, in usual query settings the sweepline-based algorithms often outperform our dynamic programming algorithms. The main contribution of the new algorithms is in their simplicity which makes them easy-to-understand and easy-to-implement. In addition to this elegance, in the worst-case scenario our new algorithms clearly outperforms the sweepline-based algorithms.

It is also theoretically very interesting to discover that dynamic programming is applicable in the TTWI setting. Applying dynamic progamming for the more straightforward problems, including the TTSI setting, has thus far proven to be too challenging.

```
DPW2(P, T, w)
1  M = A four-dimensional array, filled with −1
2  for i = 0 to P.size − 1
3      for j = 0 to T.size − 1
4          FILL-M(M, P, T, w, i + 1, j + 1, 1, 1)
5  REPORT-RESULTS(M)

FILL-M(M, P, T, w, pcur, tcur, x, y)
1  // Return result if it has been already calculated
2  if M[pcur, tcur, x, y] ≠ −1
3      return M[pcur, tcur, x, y]
4  // Bounds checking, base case for recursion
5  if tcur ≥ T.size or pcur ≥ P.size
6      return 0
7  best = 0
8  // Do the notes under investigation match each other?
9  if T_{tcur}.y − T_{tcur−y}.y == P_{pcur}.y − P_{pcur−x}.y
10     a = FILL-M(M, P, T, w, pcur + 1, tcur + 1, 1, 1)
11     best = max(a + 1, best)
12 // Can we still extend the search inside the window?
13 if y < w
14     a = FILL-M(M, P, T, w, pcur, tcur + 1, x, y + 1)
15     best = max(a, best)
16 // Finally, find the matches with P_p not included
17 a = FILL-M(M, P, T, w, pcur + 1, tcur, x + 1, y)
18 best = max(a, best)
19 M[pcur, tcur, x, y] = best
20 return best
```

Figure 3. Pseudocode illustration for DPW2. In DPW1 lines 17-18 need to be removed.

2. ALGORITHMS

In this section we describe two new algorithms to find exact and partial transposition and time-warp invariant occurrences of a pattern P from a given database T. To distinct our new algorithms from the previous sweepline algorithms W1 and W2 (solving problems W1 and W2), we shall refer to our dynamic programming algorithms by DPW1 and DPW2, respectively.

The new algorithms require the input to be given as a list of notes, where each note is represented by a pair (x, y) in a two-dimensional coordinate system. The x-component of the pair represents the note-on time, the y-component represents the pitch of the note. We assume both P and T to be *lexicographically sorted*, i.e. a precedes b if and only if $a.x < b.x$ or $a.x = b.x$ and $a.y < b.y$.

Let us next introduce some important definitions. A translation of P with vector f results in $P + f = p_0 + f, p_1 + f, \ldots, p_{m-1} + f$, where $p_i + f = (p_i.x + f.x, p_i.y + f.y)$. This translation captures two significant musical phenomena, as $f.x$ aligns the excerpt time-wise, while $f.y$ transposes the excerpt to a lower or higher key. We also define musical time-scaling with σ, $\sigma \in \mathbb{R}^+$. This time-scaling only affects horizontal translation, i.e. scales only the time components.

The following examples and definition illustrate the type of occurrences we aim at finding with the algorithms.

Example 2.1 *Let* $p = \langle 3, 1\rangle$, $f = \langle 2, 5\rangle$ *and* $\sigma = 2$. *Then* $p + \sigma f = \langle 7, 6\rangle$.

Definition 2.2 $t_{\tau_0} \ldots t_{\tau_{m-1}}$, *a subsequence of* T, *is a time-warp occurrence of* $p_{\pi_0} \ldots p_{\pi_{m-1}}$, *a subsequence of* P, *if for each* i, $0 \leq i \leq m-2$, *there is a time-scaling* $\sigma_i \in \mathbb{R}^+$ *such that* $\sigma_i(p_{\pi_{i+1}} - p_{\pi_i}) = t_{\tau_{i+1}} - t_{\tau_i}$ *and* $0 \leq \pi_j < m$, $\pi_j < \pi_{j+1}$, $0 \leq \tau_j < n$ *and* $\tau_j < \tau_{j+1}$ *for all* j.

Let us next illustrate the essence of the definition, where we have an *exact time-warping occurrence* of P.

Example 2.3 *Let* $p_0 = \langle 2, 7\rangle, p_1 = \langle 4, 8\rangle, p_2 = \langle 6, 8\rangle, p_3 = \langle 9, 7\rangle$ *and* $t_0 = \langle 1, 1\rangle, t_1 = \langle 2, 3\rangle, t_2 = \langle 3, 2\rangle, t_3 = \langle 4, 2\rangle, t_4 = \langle 5, 1\rangle$. *Then* t_0, t_2, t_3, t_4 *is an exact time-warping occurrence of* p_0, p_1, p_2, p_3 *with* $\sigma_0 = 1$, $\sigma_1 = \frac{1}{2}$ *and* $\sigma_2 = \frac{1}{3}$.

Had we had $t_4 = \langle 5, 0\rangle$ in Example 2.3, then t_0, t_2, t_3 would have been a partial time-warping occurrence of P, matching p_0, p_1 and p_2.

In [3], Lemström and Laitinen defined two problems: finding exact and partial translation and time-warp invariant occurrences of P from T. The exact nature of an occurrence is captured in definition 2.2. These problems can be described followingly: in the exact case, we aim to find a subsequence $t_{\tau_0}, t_{\tau_1}, \ldots, t_{\tau_{m-1}}$ so that for each p_i, $i < m-1$, $p_{i+1}.y - p_i.y = t_{\tau_{i+1}}.y - t_{\tau_i}.y$ holds. In the partial case, we aim to find longest subsequence from P for which we can find a matching subsequence from T, as in the definition 2.2.

In our setting, it is useful to apply a windowing restriction, which states that two consecutive notes in the database subsequence cannot be more than w notes away from each other in the database. The window size w is designed to limit the number of senseless occurrences, and it is also able to significantly speed up the algorithms.

Our algorithms are recursive in nature, and are very similar to each other. We will cover the more complex DPW2 in depth, and pinpoint the differences to DPW1.

In the beginning, the aim of the algorithms is to fill the M-table by calling function FILL-M (see Fig. 3) with appropriate base states. FILL-M takes 8 parameters, 4 of which are variables: $pcur, tcur, x$ and y. These variables define the state FILL-M is currently solving.

FILL-M returns the length of the longest occurrence we can construct from the state it was given. In the case of

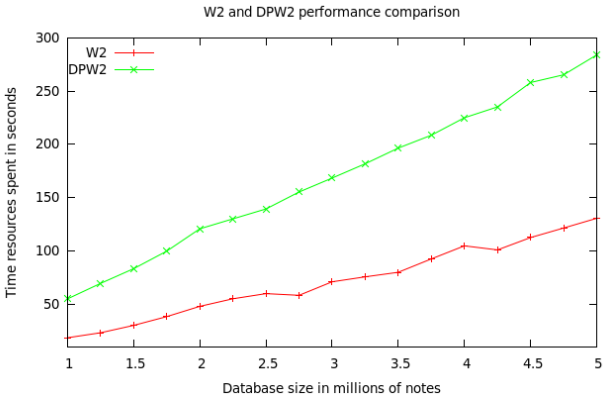

Figure 4. Time resource performance comparison of W2 and DPW2.

Figure 5. Time resource performance comparison of W2 and DPW2. Database used represented the worst case scenario for W2.

DPW2, the current state is defined by the four parameters. The parameters define the state followingly: p_{pcur-x} is the last note chosen from the pattern, t_{tcur-y} is the last note chosen from the database, whereas $p_{pcur}, p_{pcur+1}, \ldots, p_{m-1}$ and $t_{tcur}, t_{tcur+1}, \ldots, t_{n-1}$ are the notes that can be selected in future from pattern and database, respectively.

In the case of DPW2, FILL-M has at maximum three possible options in any state. FILL-M evaluates, which of the options is the best one, and returns the length of the longest occurrence. If the note under investigation can be legally added to the pattern, then the algorithm adds the note, and moves on to find new ones. Also, if we have not yet reached the windowing limit, then we can move on without adding any notes, and finding a new candidate further away in the database. Our third option, which is available in the case of DPW2, is skipping p_{pcur} altogether and not including it to the match at all. In the case of DPW1, we can never skip any p_{pcur}, since otherwise the match being constructed would not be exact anymore.

The algorithm can legally add notes to the occurrence, if note pairs (p_{pcur-x}, p_{pcur}) and (t_{tcur-y}, t_{tcur}) match each other under the translation and time-warp invariances, i.e. $p_{pcur}.y - p_{pcur-x}.y = t_{tcur}.y - t_{tcur-y}.y$. Then we can call FILL-M recursively with a state (p_n, t_n, x_n, y_n) where $x_n = y_n = 1$, $p_n = pcur + 1$ and $t_n = tcur + 1$. This means that in the new state, the previous notes that were picked from pattern and database, were p_{p_n-1} and t_{t_n-1}, respectively. Naturally, in the new state, we can find new matching notes from p_{p_n} and t_{t_n} onwards.

Also, if the parameters for FILL-M are same that have been used previously, then the algorithm can avoid calculating this state again, since every time FILL-M is called with the same parameters, it has to return the same result. Therefore every time we have finished calculating a state, we can store the result, and return the stored result whenever FILL-M is again called with the same parameters.

The case of DPW1 is very similar to that of DPW2. In DPW1, however, we cannot allow the algorithm to skip any notes from the pattern, which means that x will always be 1. As x is not a variable anymore, we do not have to store it in the M-table; it is initialized it to be 3-dimensional.

As FILL-M requires that at least one note has been selected from both the pattern and the database, we must initialize the M-table by calling FILL-M with all possible combinations of first notes (see Fig. 3). Once the M-table is filled, we can construct the matches we are interested in by investigating the M-table in a similar fashion to the way FILL-M does. Also, if we are only interested in the length of the longest occurrence, we do not need to investigate M-table afterwards at all, as FILL-M itself returns the length of the longest occurrence.

The time complexities for DPW1 and DPW2 are $O(mnw)$ and $O(mnw^2)$, respectively. The number of states depends of the possible values of the variables. The variables can vary followingly: $0 \leq pcur < m$, $0 \leq tcur < t$, $1 \leq x \leq w$ and $1 \leq y \leq w$. In DPW1 x is not a variable, so there are $O(mnw)$ states, and in DPW2 we get w times more states, due to the fact that x can vary. Since the amount of calculation in each state is constant, the time complexities become simply the number of states in both cases.

3. EXPERIMENTS

We compared the performance of W2 to that of DPW2 in different scenarios, and also W1 against DPW1 in a typical scenario. In our experiments, we used music data from Mutopia database so that the pieces of music were concatenated together to form a large database. In the worst case comparison databases and patterns were specifically tailored. In

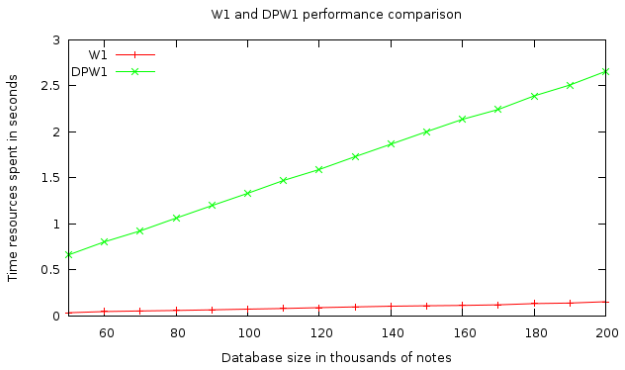

Figure 6. Time performance comparison of W1 and DPW1.

all tests, we kept the pattern and window sizes constant, $m = w = 10$.

It was expected that in cases where the database size is small, W2 would be slightly faster than DPW2, since the complexity difference would not be able to kick in with smaller database sizes, and the ability of being able to skip non-compact matches would outweigh the additional logarithmic term. However, it seemed likely that DPW2 would become gradually faster with larger database sizes when compared with W2.

In our experiments, W2 outpeformed DPW2 in the smaller cases, as expected. With growing database sizes, however, DPW2 was not able to catch up, and instead the performance difference became even larger in favour of W2.

It seems that the fact that W2 calculates only the compact matches, while DPW2 calculates exactly all matches, is responsible for the difference. Even though theoretical time complexity suggests that W2 should eventually be slower with larger databases, it seems that in a typical setting the ability of W2 to eliminate matches grows faster than the additional logarithmic term, as depicted in Fig. 4. This suggests that the expected complexity of W2 would be significantly smaller than its worst-case complexity.

The property of being able to skip non-compact matches is even more visible in the exact case, where DPW1 is significantly slower than W1 in a real-world scenario (see Fig. 6). It seems that the penalty for finding all possible matches is even larger here.

To further experiment on the effect of getting rid of additional matches, we constructed the absolute worst case scenario for W2, where all the notes in both the pattern and the database have the same pitch. In this setting, W2 would not be able to eliminate many matches, which results in a large amount of additional work. In Fig. 5, we depict the time usage of the two algorithms in the worst case for W2. From the figure it is evident that W2 uses a significant amount of time in this type of setting, even with very small databases. It is also noteworthy that the time usage of W2 grows quickly.

4. CONCLUSIONS

In this paper we presented two new algorithms for the transposition and time-warp invariant (*TTWI*) content-based polyphonic music retrieval setting. We used the geometric framework where each note is represented as a point in the Euclidean plane (pitch value against on-set time). The framework has several advantages: it is intuitive, it intrinsically deals with polyphonic music, transposition invariance and subset matching. The *TTWI* setting that allows for local time jittering makes the approach usable in real-world applications where queries are always somewhat out of tempo. Our DPW1 algorithm solves the exact matching problem under the *TTWI* setting while DPW2 is for the partial matching problem under the same setting. The algorithms, based on dynamic programming, have better asymptotic worst-case time complexities than their only existing rivals [3], here called W1 and W2, based on the sweepline techique.

Our experiments revealed that in a typical query case W2 is faster than DPW2. This is due to the capability of W2 to eliminate non-compact matches while DPW2 thoroughly scrutinizes every possible match. The impact of the elimination, however, was surprisingly strong given that W2 has an additional logarithmic term in its asymptotic complexity. Nevertheless, when looking for consistent performance, our DPW2 is the choice to be taken as in complex query cases W2 freezes suddenly. The elegance of our new algorithms lie in their simplicity: they, unlike the rivaling algorithms, are very easy both to implement and to understand.

As hinted by Fig. 4, with the future very large music databases, neither W2 nor DPW2 alone would work in an interactive setting. As a future work, we will study a distributed calculation process. Even though the sweepline-based solutions were somewhat faster in typical real-world queries in our experiments, the distributed setting is presumed to be significantly different: dynamic programming algorithms are generally easily distributable, while distributing sweepline-based algorithms may prove to be very challenging.

5. REFERENCES

[1] R. Clifford, M. Christodoulakis, T. Crawford, D. Meredith, and G. Wiggins. A fast, randomised, maximal subset matching algorithm for document-level music retrieval. In *Proc. ISMIR'06*, pages 150–155, Victoria, 2006.

[2] K. Lemström. Towards more robust geometric content-based music retrieval. In *Proc. ISMIR'10*, pages 577–582, Utrecht, 2010.

[3] K. Lemström and M. Laitinen. Transposition and time-warp invariant geometric music retrieval algorithms. In

Proc. ADMIRE'11, Third International Workshop on Advances in Music Information Research, Barcelona, 2011.

[4] K. Lemström, N. Mikkilä, and V. Mäkinen. Filtering methods for content-based retrieval on indexed symbolic music databases. *Journal of Information Retrieval*, 13(1):1–21, 2010.

[5] A. Lubiw and L. Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *Proc. ISMIR'04*, pages 289–296, Barcelona, 2004.

[6] C.A. Romming and E. Selfridge-Field. Algorithms for polyphonic music retrieval: The hausdorff metric and geometric hashing. In *Proc. ISMIR'07*, pages 457–462, Vienna, 2007.

[7] E. Ukkonen, K. Lemström, and V. Mäkinen. Geometric algorithms for transposition invariant content-based music retrieval. In *Proc. ISMIR'03*, pages 193–199, Baltimore, 2003.