Including Interval Encoding into Edit Distance Based Music Comparison and Retrieval

Kjell Lemström; Esko Ukkonen
Department of Computer Science; P.O.Box 26, FIN-00014 University of Helsinki, Finland
{klemstro,ukkonen}@cs.helsinki.fi

Abstract

A general framework for defining distance functions for monophonic music sequences is presented. The distance functions given by the framework have a similar structure, based on local transformations, as the well-known edit distance (Levenshtein distance) and can be evaluated using dynamic programming. The costs of the local transformations are allowed to be context-sensitive, a natural property when dealing with music. In order to understand transposition invariance in music comparison, the effect of interval encoding on some distance functions is analyzed. Then transposition invariant versions of the edit distance and the Hamming distance are constructed directly, without an explicit conversion of the sequences into interval encoding. A transposition invariant generalization of the Longest Common Subsequence measure is introduced and an efficient evaluation algorithm is developed. Finally, the necessary modifications of the distance functions for music information retrieval are sketched.

1 Introduction

The *translation-invariant* representation of integer sequences is useful in some application domains of string matching algorithms. In music retrieval, for example, it is often natural to require that the retrieval results are invariant with respect to pitch level transposition. In geometric shape detection the contour of a geometric object should be represented as a sequence that is invariant under rigid motions of the object.

Such an invariance can be achieved by converting the original 'absolute' sequence into relatively encoded form where each element of the sequence is encoded relative to its predecessor. The differences between successive elements are perhaps the simplest form of such an encoding. As an example, the absolute sequence $3, 7, 5, 5, 8, 7, 7, 5, 3$ is relatively encoded as $4, -2, 0, 3, -1, 0, -2, -2$.

In general string matching, the *edit distance* with its many variations is the most common measure to expose the (dis)similarity between two strings. The edit distance can be calculated by using dynamic programming; see e.g. (Crochemore and Rytter, 1994; Gusfield, 1997). The concept has also been adapted to music retrieval with relative encoding; see e.g. (Mongeau and Sankoff, 1990; Crawford et al., 1998).

The paper presents, in Section 3, a general framework for defining distance functions for monophonic music comparison and retrieval. In our framework one can use context-sensitive costs for the local transformations, that is a useful novel property when dealing with music. Analyzing the context of music usually requires a preprocessing phase which executes intelligent context analyzing programs. The result of such a preprocessing phase can be stored into some internal representation; see e.g. (Lemström and Laine, 1998).

In Section 4, we study the effect of the relative encoding on the values of some well-known and novel sequence distance functions. It turns out that typically there is no strong relation between distances measured using either absolute or relatively encoded sequences.

In Section 5, we find out, that an explicit relative encoding is not necessary. Transposition invariant distance functions can be constructed directly, by adopting transposition invariant cost functions for the local transformations used in the distance function. The cost functions now become context sensitive.

Section 6 introduces a new distance function for comparing music. It is a generalization of the classical longest common subsequence measure for comparing strings of symbols. One could call it as a 'Longest Common Hidden Melody' measure. An efficient evaluation algorithm is developed for this distance measure, improving the $O(m^2n^2)$ running time of the straightforward solution into $O(mn)$.

Finally in Section 7, we comment on using our distance functions in music information retrieval, that is, in finding the (approximate) occurrences of a short piece of music in a large music database.

2 Encodings of Music

We use a simplified representation of monophonic music that gives only the pitch levels of the notes, but not their durations. The pitch levels in semitones are given as integer numbers. Hence a piece of music containing n notes is represented as a sequence of n integers.

More formally, the integers that can be used in the sequences, form our *alphabet* Σ. In practice, we could for example have $\Sigma = \{0, 1, \ldots, 127\}$ (as in MIDI) or $\Sigma = \{0, 1, \ldots, 11\}$ (which reduces all notes into one octave). Any sequence $A = (a_1, a_2, \ldots, a_m)$ where each a_i is in Σ represents a piece of monophonic music. For simplicity, we will write such sequences in the sequel as $a_1 a_2 \cdots a_m$ instead of $(a_1, a_2, \ldots, a_m)$. The set of all sequences over Σ is denoted Σ^*. Hence $a_1 a_2 \cdots a_m$ is a member of Σ^*. The length of A is denoted $|A|$.

Two equally long sequences $A = a_1 \cdots a_m$ and $A' = a'_1 \cdots a'_m$ are *transpositions* of each other if there is an integer c such that $a'_1 = a_1 + c, a'_2 = a_2 + c, \ldots$, and $a'_m = a_m + c$. Then we write $A' = A + c$.

The *interval representation* of a sequence $A = a_1 \cdots a_m$ in Σ^* is a sequence

$$\overline{A} = (a_2 - a_1, a_3 - a_2, \ldots, a_m - a_{m-1}) = \overline{a}_1 \cdots \overline{a}_{m-1}.$$

Each symbol $\overline{a}_i$ in $\overline{A}$ is a difference between two integers in Σ. Hence $\overline{A}$ is a sequence in $\overline{\Sigma}^*$ where $\overline{\Sigma} = \{a - b \mid a, b \in \Sigma\}$ is the *interval alphabet*.

A *reduced interval alphabet* is often sufficient and useful in music information retrieval. For example, the so-called *octave equivalence* is achieved by using interval alphabet $\{a \bmod 12 \mid a \in \overline{\Sigma}\}$. If the intervals are known to be imprecise, like in query-by-humming systems, a rough classification of the intervals into possibly overlapping classes such as small, medium, and large upwards and downwards intervals might be sufficient (Ghias et al., 1995; Lemström and Laine, 1998).

The crucial property of the interval representation is that it is *transposition invariant*. This means, that if A and A' are transpositions of each other, then, obviously, they have the same interval representation, i.e.,

$$\overline{A} = \overline{A'}.$$

In music comparison and retrieval, it is often natural to have transposition invariant measures for the distance between sequences. Formally, a sequence distance function D is transposition invariant if

$$D(A, B) = D(A + a, B + b)$$

for any sequences A, B in Σ^* and any possible a, b in Σ.

Naturally, if the note durations are to be considered as well, an invariance under different tempi can be obtained by using a sequence $\hat{A}$, such that $\hat{A} = (\frac{a_2}{a_1}, \frac{a_3}{a_2}, \ldots, \frac{a_m}{a_{m-1}})$, that consists of the ratios of the original durations.

3 A Framework for Sequence Comparison

3.1 The General Scheme

We will introduce several different distance functions to measure the dissimilarity between sequences. All are variations of the well-known edit distance (also known as *Levenshtein distance* or *evolutionary distance*) widely used in approximate string matching. The underlying general framework for sequence distance functions is as follows; c.f. (Ukkonen, 1985).

To define a distance between sequences in Σ^*, one should first fix the set of *local transformations* $T \subseteq \Sigma^* \times \Sigma^*$ and a *cost function* w that gives for each transformation t a cost $w(t)$ which is a non-negative integer (or a real). Note that each t in T is a pair of sequences $t = (\alpha, \beta)$. It is convenient to extend w to all pairs (α, β) in $\Sigma^* \times \Sigma^*$: if $(\alpha, \beta) \notin T$, then we define $w(\alpha, \beta) = \infty$.

We usually write such a $t = (\alpha, \beta)$ as $(\alpha \to \beta)$ to emphasize another view on t: it can be seen as a rewriting rule that allows one to replace α by β inside a sequence that contains α.

The actual definition of the distance is based on the concept of *trace*. A trace gives a correspondence between two sequences. Formally, let $A = a_1 \cdots a_m$ and $B = b_1 \cdots b_n$ be sequences in Σ^*. A trace between A and B is formed by splitting A and B into equally many, say p, parts some of which may be empty sequences. Hence a trace can be written as:

$$\tau = (\alpha_1, \alpha_2, \ldots, \alpha_p; \beta_1, \beta_2, \ldots, \beta_p),$$

such that $A = \alpha_1 \alpha_2 \cdots \alpha_p$, and $B = \beta_1 \beta_2 \cdots \beta_p$, and each α_i, β_i is a possibly empty sequence over Σ. The trace suggests a transformation from A to B: part α_1 is transformed to β_1, α_2 to $\beta_2, \ldots$, and α_p to β_p. Stated otherwise, sequence B is obtained from A by local transformation steps $\alpha_1 \to \beta_1,\ \alpha_2 \to \beta_2, \ldots, \alpha_p \to \beta_p$.

Now the cost of the trace τ is $w(\tau) = w(\alpha_1 \to \beta_1) + \cdots + w(\alpha_p \to \beta_p)$. Note that if some $\alpha_i \to \beta_i$ is not in T, then $w(\tau)$ becomes ∞.

Finally, the distance between A and B, denoted $D_{T,w}(A, B)$, is defined as the minimum cost over all possible traces:

$$D_{T,w}(A, B) = \min\{w(\tau) \mid \tau \text{ is a trace between } A \text{ and } B\}.$$

3.2 Functions D_L and D_H

As an example, this scheme gives the familiar *unit-cost edit distance* (Levenshtein distance) if the local transformations are of the forms $a \to b,\ a \to \lambda$, and $\lambda \to a$ where a and b are any members of Σ, and λ denotes the empty sequence.[1] The costs are given as $w(a \to a) = 0$ for all a, and $w(a \to b) = 1$ for all $a \neq b$, and

[1] The local rules $a \to b, a \to \lambda$, and $\lambda \to a$ are often called 'replacement', 'deletion', and 'insertion'.

$w(a \to \lambda) = w(\lambda \to a) = 1$ for all a. We denote this edit distance function as D_L.

The *Hamming distance* is obtained exactly as D_L but the allowed local transformations are only $a \to b$ where a and b are any members of Σ, with cost $w(a \to a) = 0$ and $w(a \to b) = 1, a \neq b$. We denote the Hamming distance as D_H.

3.3 Distance Evaluation and Transformation Graph

Distances $D_{T,w}(A, B)$ are useful in practice, because they can efficiently be evaluated using dynamic programming. Such a procedure tabulates all distances $d_{ij} = D_{T,w}(a_1 \cdots a_i, b_1 \cdots b_j)$ between the prefixes of A and B. The distance table (d_{ij}), where $0 \leq i \leq m$ and $0 \leq j \leq n$, can be evaluated by proceeding row-by-row or column-by-column and using the recurrence

$$\begin{cases} d_{00} = 0 \\ d_{ij} = \min\{d_{r-1,s-1} + w(a_r \cdots a_i \to \\ \quad b_s \cdots b_j) \,|(a_r \cdots a_i \to b_s \cdots b_j) \in T\}. \end{cases} \tag{1}$$

Finally, d_{mn} equals the distance $D_{T,w}(A, B)$.

It is not difficult to see that the evaluation of $D_{T,w}(A, B)$ in this way may in the worst case take time at least proportional to $m^2 n^2$ which is relatively slow[2]. In practice, fortunately, T is often very sparse which can be utilized to speed-up the dynamic programming. For example, the edit distance $D_L(A, B)$ can be evaluated in time proportional to mn (in time $O(mn)$, for short) from the well-known recurrence

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + (\text{if } a_i = b_j \text{ then } 0 \text{ else } 1). \end{cases} \end{aligned}$$

An alternative view which sometimes is very useful, is given by interpreting (d_{ij}) as a weighted graph as follows. The d_{ij}'s form the nodes of the graph. Moreover, there is a weighted arc from any node $d_{r-1,s-1}$ to node d_{ij} if $(a_r \cdots a_i \to b_s \cdots b_j)$ belongs to T. This arc has weight $w(a_r \cdots a_i \to b_s \cdots b_j)$. Then $D_{T,w}(A, B)$ equals the smallest possible total weight of a path in this graph from d_{00} to d_{mn}; formal proof is a simple induction. We call such a path a *minimizing path*. The weighted graph is called the *transformation graph* and denoted $G_{T,w}(A, B)$.

3.4 Context-Sensitive Cost Functions

In our distance scheme as described above the cost $w(\alpha \to \beta)$ of each local transformation is independent of the context in which α and β occur in A and B. Transformation $\alpha \to \beta$ has always the same cost. When comparing musical sequences, however, a context-sensitive cost function seems sometimes useful. The cost of $\alpha \to \beta$ should depend on the (tonal, harmonic, rhythmic) context around α and β; see e.g. (Rolland and Ganascia, 1996; Coyle and Shmulevich, 1998).

This can be formalized by including the context in the definition of the cost function w. There are several possibilities as regards limiting the size of the context that can influence the cost. We can imagine that sometimes the total context of α and β, that is, the whole sequences A and B are relevant.

We suggest a parameterized limitation of the context. Assume that A can be written as $A = \mu\varphi\alpha\varphi'\mu'$ where $|\varphi| = u$ and $|\varphi'| = v$. Then α has (u, v) *context* (φ, φ'). Now we say that w is a *cost function with* (u, v) *context* if w is an integer valued function defined on $T \times \Sigma^u \times \Sigma^v \times \Sigma^u \times \Sigma^v$; here Σ^x denotes the set of sequences of length x over Σ. When using such a function w for evaluating the cost of a trace, the (u, v) context of each local rule is taken into account. If $\alpha \to \beta$ occurs in a trace such that the (u, v) context of α and β is (φ, φ') and (ϱ, ϱ'), respectively, then $w(\alpha \to \beta, \varphi, \varphi', \varrho, \varrho')$ is the cost associated with this particular occurrence of $\alpha \to \beta$.

The (u, v) context of each candidate transformation $\alpha \to \beta$ can easily be retrieved during dynamic programming when evaluating each d_{ij}. Hence using a cost function with (u, v) context is possible in the dynamic programming algorithm. The run time obviously gets slower but the overall architecture of the evaluation process remains the same.

It will turn out later in this paper that our transposition invariant sequence distance functions use cost functions with $(1, 0)$ context.

4 Absolute vs. Transposition Invariant Distance

The sequence distance functions given by our framework, such as D_L and D_H, can be applied as such both for comparing sequences in Σ^* ('absolute' sequences) and comparing their interval encoded versions in $\overline{\Sigma}^*$ ('relative' sequences). The functions induce two distances in this way. We say for sequences A, B in Σ^*, that $D_L(A, B)$ is the *absolute* edit distance and $D_L(\overline{A}, \overline{B})$ is the *transposition invariant* edit distance between A and B. When comparing music, the transposition invariant distance seems more natural except when we know *a priori* that A and B belong to the same key.

In this section we present some basic results, mostly on the relation between the absolute and the transposition invariant versions of distances D_L and D_H. However, let us start by introducing a *modulation function*. Let $A = a_1 \cdots a_m$. For any $l, 1 \leq l \leq m$, and any integer valued c, sequence

$$\mathcal{M}_l^c(A) = (a_1, \ldots, a_{l-1}, a_l + c, \ldots, a_m + c)$$

is called a *modulation* of A by c at l.

[2]Depending on the representation for T and w, the run time can be much slower.

Note that transposition is a special case: If B is a transposition of A, $B = \mathcal{M}_1^c(A)$.

The following theorem illustrates a strength of the interval encoding. Although one single modulation in the original sequences can change any number of notes, the (edit of Hamming) distance of the interval encoded versions stays small.

Theorem 1 *If B is a modulation of A then $D_L(\overline{A}, \overline{B}) = D_H(\overline{A}, \overline{B}) = 1$.*

Proof Let $B = \mathcal{M}_l^c(A)$. Hence $b_i = a_i$, for $1 \leq i < l$, and $b_i = a_i + c$ for $l \leq i \leq |A|$. Then obviously, $\overline{b}_i = \overline{a}_i$ for $1 \leq i < l-1$, $\overline{b}_{l-1} = \overline{a}_{l-1} + c$, and $\overline{b}_i = \overline{a}_i$ for $l \leq i < |A|$.

Sequences $\overline{A}$ and $\overline{B}$ differ in one position, hence the theorem follows. $\square$

We continue with a technical lemma that points out the local configurations in the transformation graphs in which the transposition invariant distance can locally be larger than the corresponding absolute distance.

Distances D_H and D_L have similar properties, hence we analyze them together. Let D_E denote D_H or D_L. We need to compare the paths in $G_E(A,B)$ and in $G_E(\overline{A}, \overline{B})$. Denote the nodes of $G_E(A,B)$ as $(d_{ij}), 0 \leq i \leq m, 0 \leq j \leq n$. Note that as $\overline{A}$ and $\overline{B}$ are one element shorter than A and B, the first row and column are missing in $G_E(\overline{A}, \overline{B})$ as compared to $G_E(A,B)$. The subgraph of $G_E(A,B)$ with nodes $(d_{ij}), 1 \leq i \leq m, 1 \leq j \leq n$, has the same topological structure as $D_E(\overline{A}, \overline{B})$ but some arc weights may differ. Let us denote this subgraph as $G'_E(A,B)$. Consider some directed path p in $G_E(A,B)$ from d_{00} to d_{mn}. Let p' be the restriction of p to $G'_E(A,B)$. Hence p' is a path from d_{rs} to d_{mn} for some r and s such that $r = 1$ or $s = 1$. Let e and f be two consecutive arcs of p'. We say that f starts a *0-block* of p', if e has weight 1 and f has weight 0.

Lemma 2 *Let g be an arc of p' such that g has weight 0 while the corresponding arc $\overline{g}$ in $G_E(\overline{A}, \overline{B})$ has weight 1. Then g starts a 0-block in p'.*

Proof. If g does not start a 0-block then the arc g' just before g on p' must have weight 0, too. The only local transformations with weight 0 that are available for distance D_E are the identity transformations of the form $a \to a$. But this means that for some i and j, g is an arc from $d_{i-1,j-1}$ to d_{ij} and g' is an arc from $d_{i-2,j-2}$ to $d_{i-1,j-1}$, and $a_i = b_j$ and $a_{i-1} = b_{j-1}$. Hence $a_i - a_{i-1} = b_j - b_{j-1}$ which means that $\overline{g}$ has weight 0 in $G_E(\overline{A}, \overline{B})$. Hence, if $\overline{g}$ has weight 1, g must start a 0-block in p'. $\square$

Theorem 3 $D_E(\overline{A}, \overline{B}) \leq 2 \cdot D_E(A,B)$.

Proof. Let p be the minimizing path in $G_E(A,B)$ from d_{00} to d_{mn}, and let p' be its restriction to $G'_E(A,B)$. Path p' can contain at most $D_E(A,B)$ 0-blocks because there must be before each block an arc with weight 1. Then by Lemma 2, path $\overline{p'}$ that corresponds to p' in $G_E(\overline{A}, \overline{B})$ can have at most $D_E(A,B)$ 1-arcs more than p'.

Moreover, let $\overline{p''}$ be the path in $G_E(\overline{A}, \overline{B})$ from $\overline{d}_{11}$ to the start node of $\overline{p'}$. A simple case analysis shows that the total weight of $\overline{p''}$ is at most the weight of the subpath of p that leads from d_{00} to the start node of p'. In summary, this means that the weight W of path $\overline{p''}\,\overline{p'}$ in $G_E(\overline{A}, \overline{B})$ from $\overline{d}_{11}$ to $\overline{d}_{mn}$ is $\leq 2 \cdot D_E(A,B)$. The theorem follows as $\overline{p''}\,\overline{p'}$ is not necessarily the minimizing path in $G_E(\overline{A}, \overline{B})$ and hence $D_E(\overline{A}, \overline{B}) \leq W$. $\square$

The following theorem gives bounds for the difference $D(A,B) - D(\overline{A}, \overline{B})$ when D is D_L or D_H. We also consider a distance $D_{H'}$ which is the Hamming distance augmented with a novel local transformation $ab \to cd$ (called *compensation*), where a, b, c, and d are members of Σ such that $a+b = c+d$. Moreover, $w(ab \to cd) = 1$.

To understand the intuition behind the compensation rule, consider the effect of a single replacement. Let A and B differ only by one replacement (also called a *mismatch*) which we denote $B = \mathcal{V}_l(A)$ where l refers to the mismatching position. Then $D_H(A,B) = 1$ while $D_H(\overline{A}, \overline{B}) = 2$ because a mismatch changes two intervals. By adding the compensation rule this antisymmetry is at least partially relieved, because $D_{H'}(\overline{A}, \overline{B}) = 1$.

Theorem 4 *a) Let $|A| = |B| = n$. Then $-\lfloor \frac{n-1}{2} \rfloor \leq D_H(A,B) - D_H(\overline{A}, \overline{B}) \leq n$.*

b) Let $|A| = |B| = n$. Then $-\lfloor \frac{n-1}{3} \rfloor \leq D_H(A,B) - D_{H'}(\overline{A}, \overline{B}) \leq n$.

c) Let $m = |A| \leq |B| = n$. Then $1 - m \leq D_L(A,B) - D_L(\overline{A}, \overline{B}) \leq n$.

Proof. To prove the upper bounds in all the cases, we note that, clearly, the bound cannot be larger than n because the distances between A and B as well as between $\overline{A}$ and $\overline{B}$ always belong to the range $0, \ldots, n$. To show that the bound is tight, let $A = aa \cdots a$ and $B = bb \cdots b$ for some $a \neq b$ and $|A| = |B| = n$. Then $D_H(A,B) = D_L(A,B) = n$ while $D_H(\overline{A}, \overline{B}) = D_{H'}(\overline{A}, \overline{B}) = D_L(\overline{A}, \overline{B}) = 0$. Hence the bound cannot be smaller than n.

The lower bound follows from the fact that the number of 0-blocks of Lemma 2 is at most $\lfloor \frac{n-1}{2} \rfloor$ in the case of Hamming distance D_H; at most $\lfloor \frac{n-1}{3} \rfloor$ in the case of modified Hamming distance $D_{H'}$; and at most $m-1$ in the case of Levenshtein distance D_L.

All the bounds are again tight. For D_H consider sequences $A = (0,1,0,1,\ldots)$ and $B = (0,2,0,2,\ldots)$; for $D_{H'}$ sequences $A = (0,0,1,0,0,1,\ldots,0,0,1)$ and $B = (0,0,2,0,0,2,\ldots,0,0,2)$; and for D_L sequences

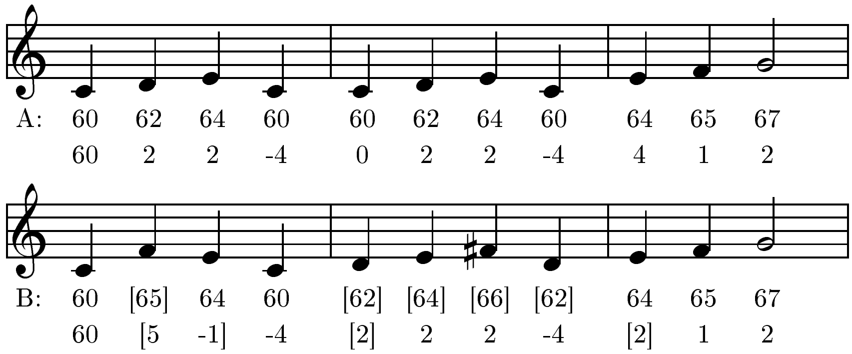

Figure 1: $D_H(A, B)$ vs. $D_{H'}(\overline{A}, \overline{B})$ in music comparison. The sequences below A and B correspond to $\overline{A}$ and $\overline{B}$, respectively.

$A = (0, 0, \ldots, 0)$ and $B = (1, 0, 1, 0, \ldots, 1, 0)$ such that $|B| = 2 \cdot |A|$. $\square$

Example 1 *In Fig. 1, the absolutely encoded A and B are comprised of MIDI values. The two sequences are almost identical: there is only one mismatch and two modulations ($B = \mathcal{M}_9^{-2}(\mathcal{M}_5^2(\mathcal{V}_2(A)))$). We have enclosed in brackets the locations where an editing operation is needed. In $D_{H'}(\overline{A}, \overline{B})$, the mismatch is detected by a compensation ($\overline{a}_2 + \overline{a}_3 = \overline{b}_2 + \overline{b}_3 = 4$) and the modulations by a replacement, thus $D_{H'}(\overline{A}, \overline{B}) = 3$. However, $D(A, B) = 5$ because the mismatch is detected by a replacement, but the first modulation has shifted all the values until the other 'normalizing' modulation. Note, that after being lost 'B catches' immediately the correct tune at the second modulation ($a_9 = b_9$), while '$\overline{B}$ realigns' itself one step later.* $\square$

5 Implicit Interval Encoding

We noticed in Section 4 that each distance function for absolute sequences also gives a transposition invariant distance. This is achieved simply by at first converting the sequences into interval encoding. We now complement this by observing that an explicit conversion is not necessary. The cost function for local transformations can be defined such that the resulting distance becomes transposition invariant.

Since the conversion is not needed anymore, some shortcomings of the relative encoding are avoided. When the intervals are calculated 'on-the-fly' from absolute sequences, a deletion or an insertion does not transpose the rest of a sequence as in the case when working with relative encoding.

We restrict the consideration to distances D_H and D_L only. The *transposition invariant unit-cost edit distance* $\overline{D}_L(A, B)$ uses the local transformations $a \to b$, $a \to \lambda$, and $\lambda \to a$ as the distance $D_L(A, B)$. The costs are given as $\overline{w}(a \to \lambda) = \overline{w}(\lambda \to a) = 1$. The cost for $a \to b$ will now be context-sensitive. Let a' be the symbol of A just before a and b' the symbol of B just before b', that is, the $(1, 0)$ contexts of a and b are (a', λ) and (b', λ). Then define

$$\overline{w}(a \to b, a', \lambda, b', \lambda) = \begin{cases} 0, \text{if } a - a' = b - b' \text{ or} \\ \quad a' \text{ or } b' \text{ is missing,} \\ 1, \text{if } a - a' \neq b - b'. \end{cases}$$

The recurrence for evaluating the prefix distance table (d_{ij}) for $\overline{D}_L(A, B)$ becomes

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + (\text{if } a_i - a_{i-1} = b_j - b_{j-1} \\ \qquad \text{then } 0 \text{ else } 1). \end{cases} \end{aligned}$$

Obviously, this table can be evaluated in time $O(mn)$.

Similarly, the *transposition invariant Hamming distance* $\overline{D}_H(A, B)$ uses local transformations $a \to b$ whose cost is defined exactly as for $\overline{D}_L(A, B)$ above.

By induction over the corresponding prefix distance tables (d_{ij}) one easily proves the following theorem which implies that $\overline{D}_L$ and $\overline{D}_H$ really are transposition invariant distance functions according to the definition given in Section 2.

Theorem 5

$$\begin{aligned} \overline{D}_L(A, B) &= D_L(\overline{A}, \overline{B}) \\ \overline{D}_H(A, B) &= D_H(\overline{A}, \overline{B}). \end{aligned}$$

This also includes that the prefix distance table (d_{ij}) for $\overline{D}_L$ is of the same general form as the table for the standard unit-cost edit distance. Hence the very fast bit-parallel algorithms designed for the edit distance can be used for evaluating the transposition invariant unit-cost edit distance as well.

Finally we remark that using both the absolute and relative costs of local transformations simultaneously is possible and seems to give useful distance functions. For example, in the standard unit-cost edit distance we could give cost 0 to $a \to b$ if $a = b$, as usual, but also if $a \to b$ occurs in a context in which the interval for a equals the interval for b. The recurrence becomes

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + (\text{if } a_i = b_j \text{ or } a_i - a_{i-1} = \\ \qquad b_j - b_{j-1} \text{ then } 0 \text{ else } 1). \end{cases} \end{aligned}$$

We denote the resulting distance function as D_N. As D_N has the same local transformations as D_L and $\overline{D}_L$, with possibly smaller costs, it follows that $D_N(A, B) \leq D_L(A, B)$ and $D_N(A, B) \leq \overline{D}_L(A, B)$ for all A and B. On the other hand, D_N is not transposition invariant.

6 Transposition Invariant LCS

The *longest common subsequence* $LCS(A, B)$ of two sequences A and B can be used for measuring the similarity of A and B: the longer is $LCS(A, B)$, the more similar are the sequences A and B.

To define $LCS(A, B)$, we say that sequence C is a *subsequence* of a sequence A if C can be obtained from A by deleting zero or more symbols. Then $LCS(A, B)$ is the longest sequence that is a subsequence of both A and B. For music comparison and retrieval we need a transposition invariant generalization of this concept.

We say that a sequence C in Σ^* is the *longest common transposition invariant subsequence* of two sequences A and B in Σ^* if C is the longest possible sequence such that $\overline{C} = \overline{A'}$ and $\overline{C} = \overline{B'}$ and A' is a subsequence of A and B' is a subsequence of B. Then we write $C = LCTS(A, B)$. Note that $LCTS(A, B)$ is unique only up to arbitrary transposition: if $C = LCTS(A, B)$ then any $C + c$ is $LCTS(A, B)$.

The sequence $C = LCTS(A, B)$ can be seen in musical terms as the longest common melody that is hidden in both A and B. To obtain C, we must delete the smallest number of elements from A and B such that the remaining two sequences are identical after some transposition, that is, their interval encoded representations are identical. This seems like a natural concept in the context of music comparison. We can think, for example, that the deleted elements are musical decorations used differently in the two variations A and B of the same melody C [3].

Let $D_{LCS}(A, B)$ denote the total number of deletions needed to obtain $LCS(A, B)$ from A and B. Then it is well-known that

$$|LCS(A, B)| = \frac{|A| + |B| - D_{LCS}(A, B)}{2},$$

where $D_{LCS}(A, B)$ is a distance function that is given by our general scheme by using local transformations $a \to \lambda$, $\lambda \to a$, and $a \to a$ with costs $w(a \to \lambda) = w(\lambda \to a) = 1$, and $w(a \to a) = 0$.

Similarly, it turns out that

$$|LCTS(A, B)| = \frac{|A| + |B| - D_{LCTS}(A, B)}{2},$$

where $D_{LCTS}(A, B)$ is a distance function again given by our scheme. Now any rule $\alpha \to \beta$ where α, β are *non-empty* sequences in Σ^* is a local transformation. The associated cost function w_{LCTS} has $(1, 0)$ context. Let a be the last symbol of α and b the last symbol of β and let the $(1, 0)$ context of α and β be (a', λ) and (b', λ), respectively.

Then we define

$$w_{LCTS}(\alpha \to \beta, a', \lambda, b', \lambda) = \begin{cases} k-1+l-1, \text{if } a-a' = b-b' \\ k+l, \text{if } a-a' \neq b-b', \end{cases}$$

where $k = |\alpha|$ and $l = |\beta|$.

The recurrence for tabulating (d_{ij}) to get $D_{LCTS}(A, B)$ is

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min_{1 \leq k \leq i; 1 \leq l \leq j} \{ d_{i-k,j-l} + \\ &\qquad w_{LCTS}(\alpha \to \beta, a_{i-k}, \lambda, b_{j-l}, \lambda) \}, \end{aligned}$$

where $\alpha = a_{i-k+1} \cdots a_i$ and $\beta = b_{j-l+1} \cdots b_j$. To evaluate d_{ij} directly from the recurrence needs $O(ij)$ operations, hence the total time requirement becomes as high as $O(m^2 n^2)$. We show next how to reduce the computation of $D_{LCTS}(A, B)$ to a repeated computation of the distance D_{LCS}.

The key observation is, that $LCTS(A, B)$ must be equal to a $LCS(A, B + c)$ for some suitably selected constant c.

This gives the following solution: For all c in $\overline{\Sigma}$, compute $D_{LCS}(A, B + c)$ with the standard $O(mn)$ algorithm. Then $|D_{LCTS}(A, B)| = \max_c \{ |D_{LCS}(A, B + c)| \}$. This method takes time proportional to $|\overline{\Sigma}| \cdot mn$ which is $O(mn)$ because $|\overline{\Sigma}|$ is independent of m and n. Hence we have:

Theorem 6 *$D_{LTCS}(A, B)$ and hence $|LCTS(A, B)|$ can be computed in time $O(mn)$.*

In Fig. 2 we give an example of calculating an LCTS between two musical sequences (the sequences are obtained from (Cambouropoulos et al., 1999); the lower sequence is transposed from the original key of C major to D major).

7 Music Retrieval

In the content-based information retrieval problem for monophonic music we are given a long database $S = s_1 s_2 \cdots s_n$ of monophonic music and a relatively short

[3] To really achieve this goal needs further elaboration of the LCTS model that goes beyond the present paper.

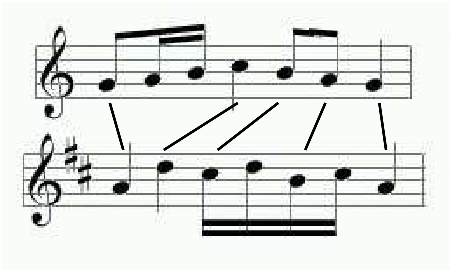

Figure 2: An example of finding $LCTS(A, B)$. The applied $(a \to a + c)$ rules are illustrated by lines. In this case $|LCTS(A, B)| = 5$, and $c = 2$.

'key' $P = p_1 \cdots p_m$. Formally, both S and P are sequences in Σ^*. The problem is to find the locations in S where P occurs as a subsequence. One might be interested in finding for example exact occurrences, transposed occurrences or maybe somehow approximate occurrences.

The framework of Section 3 is for comparing entire sequences. It is, however, easy to modify it using a well-known trick, originally presented by Sellers (1980), for our retrieval problem. Now P should basically be compared against *all subsequences* of S to find the best match. The dynamic programming algorithm almost as such will do this, only a slight change is needed in the initialization of the table (d_{ij}).

More precisely, let $D_{T,w}$ be a distance function given by the scheme of Section 3. Evaluate table $(d_{ij}), 0 \le i \le m, 0 \le j \le n$, as in the recurrence (1) but use initialization

$$d_{0j} = 0$$

for $0 \le j \le n$. Then the music retrieval results can be read from the last row $d_{mj}, 0 \le j \le n$, of the table (d_{ij}). Value d_{mj} gives the smallest possible value of $D_{T,w}(P, P')$ where P' is any subsequence (substring) of S that ends at location j. The sequence P' can be uncovered using (d_{ij}), too.

The length n of the database S can be very large. Therefore it is of crucial importance to find fast implementations for the evaluation of (d_{ij}). The fastest algorithms currently known are based on so-called bit-parallelism (Baeza-Yates and Gonnet, 1992). Bit-parallelism can give a speed-up up to by factor W where W is the length (in bits) of the computer word used. Unfortunately, such algorithms have quite limited applicability: strong restrictions on the set T of local transformations and on the cost function w, seem necessary.

The unit-cost edit distance D_L is an example of a distance function for which bit-parallel implementation is possible. The fastest such algorithms currently known are due to Myers (1998), and Navarro and Raffinot (1998). It turns out that Myers' algorithm can be modified (at least) to the distance function D_N and $\overline{D}_L$ defined in Section 5. We have implemented these in our prototype MIR system under development (Lemström and Perttu, 2000). The algorithms can reach a scanning speed exceeding 10^7 notes per second on current Pentium II computers.

8 Conclusion

We have considered the problem of measuring the distance between two sequences A and B in the context of monophonic music comparison and retrieval. The encoding that we have used is a simplified representation of monophonic music; only the pitch levels are present in the encoding.

We have presented a general framework for sequence comparison. The framework deals with variations of the well-known edit distance measure. Moreover, in our framework one can use context-sensitive cost functions, which we believe that is a very important property in this application area. We also introduced the concept of a transposition invariant distance function and presented some examples of such functions.

Currently we are working on various bit-parallel algorithms for music information retrieval, based on the distance functions presented in this paper.

References

R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10): 74–82, 1992.

E. Cambouropoulos, T. Crawford, and C. S. Iliopoulos. Pattern processing in melodic sequences: Challenges, caveats & prospects. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 42–47, 1999.

E. Coyle and I. Shmulevich. A system for machine recognition of music patterns. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3597–3600, Seattle, WA, 1998.

T. Crawford, C. S. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11: 71–100, 1998.

M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming - musical information retrieval in an audio database. In *ACM Multimedia 95 Proceedings*, pages 231–236, San Francisco, California, 1995.

D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.

K. Lemström and P. Laine. Musical information retrieval using musical parameters. In *Proceedings of the 1998*

International Computer Music Conference, pages 341–348, Ann Arbor, Michigan, 1998.

K. Lemström and S. Perttu. SEMEX - an efficient music retrieval prototype. Manuscript (in preparation), 2000.

M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. In *Proceedings of Combinatorial Pattern Matching*, pages 1–13, Piscataway, New Jersey, 1998.

G. Navarro and M. Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In *Proceedings of Combinatorial Pattern Matching*, pages 14–33, Piscataway, New Jersey, 1998.

P.-Y. Rolland and J.-G. Ganascia. Automated motive oriented analysis of musical corpuses: a jazz case study. In *Proceedings of the 1996 International Computer Music Conference*, pages 240–243, Hong Kong, 1996.

P. H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980.

E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.