SIA(M)ESE: An efficient algorithm for transposition invariant pattern matching in multidimensional datasets

Geraint A. Wiggins[◇] , Kjell Lemström[◇] , David Meredith[◇]
[◇] Department of Computing, City University, London
Northampton Square, London EC1V0HB, England
[*] Department of Computer Science
FIN-00014 University of Helsinki, Finland
{geraint,kjell,dave}@soi.city.ac.uk

May 16, 2001

Abstract

In this paper, we study pattern matching in multidimensional datasets. The aim is to find *translation* (transposition) invariant occurrences of a given query pattern, called template, in an arbitrary multidimensional dataset. Between the points in the dataset that have been found to match the consecutive points in the template, there may be any finite number of other intervening datapoints. For this task, we introduce an algorithm, called SIA(M)ESE, which is based on the SIA pattern induction algorithm (Meredith *et al.*, prep). The algorithm is first introduced in abstract mathematical form, then we show how we have implemented it using sophisticated techniques and equipped it with sensible heuristics. The resulting efficient algorithm has a worst case running time of $O(mn\log(mn))$, where m and n are the size of the template and the dataset, respectively. We consider several application domains, such as cognitive modeling of music and matching of polyphonic music and bitmap images, and show the flexibility of SIA(M)ESE. It not only solves the problem it is developed for, but without any change to its original time complexity, it can also simulate the working of several existing algorithms developed for distinct pattern matching problems.

1 Introduction

The SIA algorithm (Meredith *et al.*, prep) is intended for the induction of significant structures from databases of (musical) data. However, it can straightforwardly be modified for use in searching for patterns in (musical) databases, making assessment comparisons between datasets, and other similar areas. In the pattern search application, our matching algorithm, which we call SIA(M)ESE, performs to a level comparable with existing, more restricted, algorithms based on, for example, dynamic programming (DP) calculating edit distance: SIA(M)ESE runs in $O(mn\log(mn))$ worst-case time, where m and n are the sizes of the template and the dataset, respectively, but produces more useful information than the DP methods. In this paper, we mainly concentrate on two specific applications: music student assessment and polyphonic music searching. These applications have aspects which are significantly problematic for the existing matching methods, which is not, however, the case with SIA(M)ESE. Moreover, SIA(M)ESE is applicable to other pattern matching problems, e.g., to matching bit-map images.

The assessment problem is as follows. In intelligent systems for music education, such as **Tapper** (Wiggins and Trewin, 2000), there is a requirement to test and assess students so as to judge their progress. Testing is a crucial and sensitive part of the education process: not only is correct assessment important in choosing a path for the student through the curriculum, but incorrect assessment and/or feedback can seriously damage a student's learning experience. In using **Tapper**, students are required to tap out different aspects of rhythms that are played to them by the system, and in future versions, they will be required to

sing in response to more complicated exercises. This presents the intelligent tutoring system with a two-stage task: match the notes in the model answer up with the input by the student, and then decide what errors and/or misconceptions gave rise to any mismatches detected.

There are special difficulties here in comparison with the more common task of matching a template supplied by a practiced and competent musician against a body of data. In particular, there is a tendency for beginners to repeat sections of a melody as they attempt to get it right. This presents a problem for the DP-based matching methods, because it becomes difficult to detect which of the extra notes so produced should be viewed as extraneous. SIA(M)ESE's inbuilt grouping of the notes into subsets (see below) can help overcome this problem.

The task in our second application is to match a polyphonic musical template against members of a dataset corresponding to a database of polyphonic music. Traditionally, this problem has been approached by extending methods based on string matching (Dovey, 1999; Holub *et al.*, 1999; Lemström and Tarhio, 2000). Specifying the problem as one of string matching has intrinsic drawbacks: there is no natural way of representing polyphonic music exactly with linear strings (we return to this point, with an illustration, in Section 5.1). Specifically, the string-based methods rely on linear contiguity between successive symbols in a string, and there is no notion of co-occurrence. Either of these constraints is enough to preclude straightforward representation of polyphonic music databases. However, because our new algorithm is inherently multidimensional, polyphonic matching can be performed trivially, without extension.

In the rest of this paper, we survey related literature, and then present the SIA algorithm modified first as SIA(M), an inefficient but easy-to-understand version, and then as the more efficient but more complicated SIA(M)E. We then introduce some simple heuristics for selection and evaluation of high-quality[1] matches, giving the full SIA(M)ESE algorithm. Before giving the conclusions, we present some examples of the algorithm functioning in the contexts introduced above and illustrate how it can be used to simulate the working of some related algorithms.

2 Related Work

In the literature, one can find a considerable number of papers focusing on music information retrieval, or MIR, for short. The problem has frequently been transformed into a combinatorial pattern matching problem, which is a well studied discipline in computer science, see e.g. Crochemore and Rytter (1994). In this way, it is reasonably straightforward to find occurrences of a monophonic musical template within datasets corresponding to monophonic pieces of music. For the current state of the art in string matching techniques for music retrieval, we refer the reader to Crawford *et al.* (1998) and Lemström (2000).

Let us now recall some preliminaries of string combinatorics. A string is a sequence of zero or more symbols from an alphabet Σ. The set of all sequences over Σ is denoted by Σ^*. A string X of length n is represented by $X_1^n = x_1 \cdots x_n$, where $x_i \in \Sigma$ for $1 \leq i \leq n$. The length of the string X is denoted as $|X|$, *viz.*, $|X_1^n| = n$. Let the sequence X be of form $X = \alpha\beta\gamma$, where $\alpha, \beta, \gamma \in \Sigma^*$. We say that α is a *prefix*, β a *substring*, and γ a *suffix* of X. A string X' is a *subsequence* of X if it can be obtained from X by deleting zero or more symbols, i.e., $X' = x_{i_1} x_{i_2} \cdots x_{i_m}$, where $i_1 \ldots i_m$ is an increasing sequence of indices in X. We say that the substring X_i^l is a *covering substring* of $Y = y_1 \cdots y_m$ *at* i, if Y is a subsequence of X_i^l, $x_i = y_1$, and $x_l = y_m$. Furthermore, X_i^l is the *minimal covering substring* of Y *at* i if X_i^l is a covering substring of Y and there is not another covering substring $X_i^{l'}$ of Y such that $l' < l$.

Because the conventional string matching methods were originally developed for text matching, and texts do not share many of the features that are intrinsic to music, the whole framework needs to be modified in order to obtain musically meaningful matching results, see e.g. Mongeau and Sankoff (1990) and (Lemström, 2000). The most salient feature of music that is not pertinent to text but is certainly pertinent to our study, is that real music is (almost) always polyphonic. A straightforward application of the conventional string matching methods would start by encoding music as a string of pairs $\langle a, b \rangle_i$, where a and b correspond to the onset time and to the pitch of the ith note, respectively. Let $T = t_1, \ldots, t_m$ and $D = d_1, \ldots, d_n$ be strings corresponding to a monophonic template and a polyphonic dataset string encoded in this way (where each $d_i = \langle a, b \rangle_i$, for $1 \leq i \leq n$), and let D_i^l be the minimal covering

[1] Quality of match is, of course, application dependent.

substring of T at i (in other words, there is an occurrence of the template within the dataset starting at i) . Now, it is likely that $|D_i^l| > m$, that is, there may be extra intervening elements within D_i^l (because of the polyphony), and therefore a conventional string matching algorithm may not find the match.

One possible solution would be to apply the techniques proposed by Crochemore *et al.* (2000). They allow gaps (without paying any penalty) between the elements in the dataset that match with the consecutive elements of the template. The time complexity of their technique for a parametrized maximum spacing (in time) between the consecutive matching elements within the dataset is $O(mn)$, where m and n are the length of the template and the length of the dataset when encoded as above. In the literature, one can also find techniques that have been designed with polyphonic music in mind right from the start. We shall now briefly review these approaches, where the datasets are represented as strings of chords $D' = D'_1 \cdots D'_{n'}$, each of whose element contains the musical events having their onset time simultaneously.

2.1 Techniques for Locating Monophonic Templates within Polyphonic Datasets

Dovey (1999) also allows parametrized spacing between the consecutive elements in the dataset, but the datasets he considers may be polyphonic. Because the algorithm tries to find an occurrence of the given template recursively in every possible location with every allowed spacing within the dataset, its time complexity becomes impractical for very large datasets; the worst case takes $O(n'm(t+1)^{(m-1)})$, where m is as above, and n' and t are the length of the dataset in chords and the length of the allowed spacing, respectively. Moreover, to find transposed occurrences, the algorithm has to be reiterated for each possible transposition. In Dovey's algorithm, the template to be searched for may itself be polyphonic.

Uitdenbogerd and Zobel (1998) combined ideas from music psychology with the straightforward application of conventional string matching methods. Their aim was to develop a heuristic capable of capturing the (monophonic) musical line that best represents a passage of polyphonic music. In their experiments with human listeners, a heuristic that always chooses the highest note of a chord performed the best. Thus, combining any conventional string matching method with their heuristic may be applied to the problem at the cost of losing information.

Holub *et al.* (1999) presented an algorithm based on the so-called *bit-parallel* algorithmic technique. They started by using the well-known shift-or algorithm of Baeza-Yates and Gonnet (1992) to find *multi-templates* (several templates combined in a single query) within monophonic datasets, and arrived at an algorithm capable of finding multi-templates within polyphonic datasets in $O(nq)$ time with a preprocessing phase taking $O(rm + |\Sigma'|)$ time [2]. Here q, r and $|\Sigma'|$ denote the maximum number of events within any chord, the number of templates, and the number of symbols used in the templates, respectively. Holub *et al.* did not, however, consider transposition invariance. From a similar starting point, Lemström and Tarhio (2000) introduced an algorithm, called MONOPOLY, capable of finding all transposed occurrences of a monophonic template within polyphonic datasets. The essential part of their algorithm runs in linear, $O(n)$, time (with an $O(nq)$ preprocessing and an $O(nq(q+m))$ postprocessing phase) [3].

2.2 A Technique for Pictorial and Other Higher Dimensional Matching

Another problem considered in combinatorial pattern matching is pattern matching in two-dimensional structures. The frequently used motivation for this is its relationship to image processing and retrieval. The idea is to locate an $\hat{m} \times \hat{m}'$ rectangular template inside an $\hat{n} \times \hat{n}'$ dataset array taken from an alphabet of size c. Figure 1 exemplifies a case with bit-map images. Without loss of generality, let us assume that $\hat{n} = \hat{n}'$ and $\hat{m} = \hat{m}'$. Kärkkäinen and Ukkonen (1994) have showed that $O(\frac{\hat{n}^2}{\hat{m}^2} \log_c \hat{m}^2)$ is the lower bound for expected running time for an optimal string matching algorithm. They also presented an optimal algorithm achieving the bound with additional $O(m^2)$ time for preprocessing. Tarhio (1996) presented an algorithm that achieves the lower bound also in the worst case (with $O(cm^2)$ time for preprocessing) and being at least as fast as the former when $\hat{m} \geq 20$, in practice.

[2] More precisely, the algorithm have a factor $\lceil \frac{m}{w} \rceil$ in its time complexity, where w denotes the size of computer words in bits. Thus, the best time complexity is achieved only in cases where $m \leq w$.

[3] See Footnote 2.

These techniques are applicable also in higher dimensions and for any kind of data that can be encoded such a way. Nevertheless, they do not work for real numbers for they depend on the size of the containing space rather than the number of datapoints.

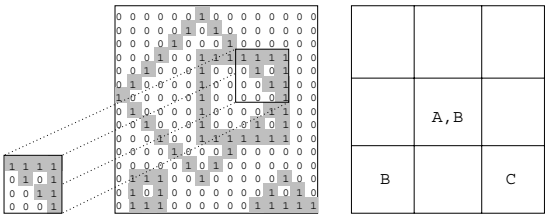

Figure 1: *Example of pattern matching in a bitmap image. The 1 bits are emphasized to show the objects. To the left is the template whose occurrence is found in the dataset (in the middle) with alignment (4,9). To the right, a symbolic picture (using a* 3×3 *grid) representing the objects in the bitmap image.*

Some image retrieval techniques are based on a structural approach (Chang *et al.*, 1987, 1988; Lee *et al.*, 1989). These techniques use the spatial relationships among objects to represent the content of an image. First, a preprocessing phase locates manually or semi-automatically the objects in a given image, which are subsequently classified and labelled using symbols of an alphabet. The symbols are used to form a symbolic picture, i.e., an $M \times N$ grid from which the spatial relationships among the objects are obtainable (see Figure 1). Finally, the objects and their relationships are represented by so-called 2-D strings. By expressing the query templates in the same 2-D string notation, the image retrieval problem is transformed to a 2-D string subsequence matching problem.

As an example, consider the case represented in Figure 1. Each object is thought to be located at its center-point for which a symbol of the alphabet is associated. In this example, let us represent the diamond at (6,6) by the symbol A, the two squares at (13,2) and (6,9) by B's, and the triangle at (13,12) by C . Let us use the so-called *augmented 2-D strings*. They are of form $(x; y; p)$ where the x and y are 1-D strings respectively representing the left-right and below-above orderings of the objects, while p is a permutation string binding the corresponding elements in x and y. Moreover, a special symbol of the alphabet, "$<$", is used to specify spatial relationship *left-right* or *below-above*. Using this coding, the image in Figure 1 is represented by the following augmented 2-D string

$$(B < AB < C; BC < AB; 1342).$$

The three different kind of occurrences that are to be searched are called type-0; type-1; and type-2 occurrences. To define them, a concept of rank is introduced; *rank*, $r(a)$, of symbol a is one plus the number of "$<$"s preceding it in the resulting 1-D string. Let A and B be 1-D strings of the template and the dataset, respectively. Now, A is a *type-i 1-D subsequence* of B, if

- A is a subsequence of B, and
- if $a_i w a_j$ is substring of A, where a_i matches b_k in B and a_j matches b_l in B, then

type-0: $r(b_l) - r(b_k) \geq r(a_j) - r(a_i) \ \vee \ r(a_j) - r(a_i) = 0$;
type-1: $r(b_l) - r(b_k) \geq r(a_j) - r(a_i) \ \vee \ r(b_l) - r(b_k) = r(a_j) - r(a_i) = 0$;
type-2: $r(b_l) - r(b_k) = r(a_j) - r(a_i)$.

Type-i occurrences are then defined in a rather obvious way. Let $T = (x; y; p)$ and $D = (x'; y'; p')$ be the 2-D string representations of the template and dataset, respectively. T has a *type-i occurrence* in D, if x and y are type-i 1-D subsequences of x' and y', respectively, and if x_i in x matches x'_j in x' then x_{p_i} of y matches $x_{p'_i}$ of y'.

The augmented 2-D string, introduced by Chang *et al.* (1987), have since been modified to enable more sophisticated relations between the objects. We refer the reader to Chang *et al.* (1988) and Lee *et al.* (1989) for further information.

3 The New Algorithm

We begin by stating the logic of our algorithm in a naïve and relatively inefficient formulation, which nonetheless makes it easier to understand (Section 3.1). If implemented directly, this would lead to an $O((mn)^2)$ time complexity, where m is the template size and n is the dataset size. We call this stage SIA(M), for SIA(Matching), SIA being the Structure Induction Algorithm of Meredith *et al.* (prep). In Section 3.2, we present two versions of an efficient control procedure, which result in algorithms with time complexities $O(mn)$ in the average case, and $O(mn\log(mn))$ in the worst case.

3.1 The Logic of SIA(M)

Equations (1) and (2), below, express the standard SIA algorithm (Meredith *et al.*, prep). The structure set $\mathcal{S}$ is computed from the dataset, D, where D is any set of points with any number of dimensions. The dimensions may be measured on any finitely expressible metric so long as it is possible to give a total ordering, $\prec$, on all the points in the vector space defined by D.

$$P = \{\langle p_1, V\rangle \mid p_1 \in D \wedge V = \{\overline{v} \mid \exists p_2.p_2 \in D \wedge p_1 \prec p_2 \wedge \overline{v} = p_2 - p_1\}\} \quad (1)$$

$$\mathcal{S} = \{\langle \overline{v}_1, M\rangle \mid \exists p_1, V_1.\langle p_1, V_1\rangle \in P \wedge \overline{v}_1 \in V_1 \wedge M = \{p_2 \mid \exists \overline{v}_2, V_2.\langle p_2, V_2\rangle \in P \wedge \overline{v}_2 \in V_2 \wedge \overline{v}_1 = \overline{v}_2\}\} \quad (2)$$

The idea is to find all the maximal subsets of D, which are translated by a non-zero vector in the space defined by D's dimensions, to another subset of D. The ordering, $\prec$, prevents wastage of effort and duplication of results by removing repetition under symmetry. The output, S, is in the form of a set of $\langle$vector,point-set$\rangle$ pairs, relating each subset to the vector which translates it.

Compare this with the logic of SIA(M), where we compute the match set $\mathcal{M}$ using (3) and (4), from a template set, T, whose occurrences in the dataset, D, we wish to find.

$$P = \{\langle p_1, V\rangle \mid p_1 \in T \wedge V = \{\overline{v} \mid \exists p_2.p_2 \in D \wedge \overline{v} = p_2 - p_1\}\} \quad (3)$$

$$\mathcal{M} = \{\langle \overline{v}_1, M\rangle \mid \exists p_1, V_1.\langle p_1, V_1\rangle \in P \wedge \overline{v}_1 \in V_1 \wedge M = \{p_2 \mid \exists \overline{v}_2, V_2.\langle p_2, V_2\rangle \in P \wedge \overline{v}_2 \in V_2 \wedge \overline{v}_1 \simeq \overline{v}_2\}\} \quad (4)$$

There are just two small differences between these pairs of definitions. The first, and the more significant, is that in SIA(M), the vectors, $\overline{v}_i$, in P are not specified in terms of two points, p_1 and p_2, in the dataset, D, ordered under $\prec$ to avoid duplication under symmetry as shown in (1). Rather, the vectors are generated from two separate sets of points inhabiting the same vector space: the template, T, and the dataset, D. The separation of these two sets means that the efficiency measure of removing equivalents under symmetry in generating P no longer makes sense, so the $\prec$ term of (1) is not present in (3).

The only difference between the subsequent calculations of $\mathcal{S}$ and $\mathcal{M}$ (see (2) and (4), respectively) is that the final equality term $\overline{v}_1 = \overline{v}_2$ has been replaced by a term $\overline{v}_1 \simeq \overline{v}_2$ to admit a more relaxed notion of comparison. For example, in the case where we are matching a human performance against an idealized, quantized database, we may want to relax equality a little to avoid having to deal with jitter explicitly in the match-finding process (see Section 4).

In the simplest case where the aim is to find an exact occurrence of T in D, $\simeq$ in equation (4) is defined as identity and an occurrence is found if and only if Condition 5 is also satisfied:

$$\exists M.\langle \overline{v}, M\rangle \in \mathcal{M} \wedge |M| = m. \quad (5)$$

However, the output of SIAM can be used for more complex analyses, as will be seen in Section 4. Henceforth, the next definition will be used frequently.

Definition 1 *Let* t *be a point in the template* T*, and* $\overline{v}$ *a vector in the set of vectors* V *as defined in (3). We say that* t is translatable by $\overline{v}$ *if* $t + \overline{v} = d$*, for some* d *in the dataset* D.

3.2 An Efficient Control Régime for the Algorithm: SIA(M)E

It is not hard to see that any naïve implementation of the specification formed by (3) and (4) would lead to a worst case time complexity of $O((mn)^2)$, where m is the template size and n is the dataset size. This arises first from the computation of the cross product of the two sets, T and D, in (3), and then from the need to scan that cross product of mn vectors once with every mn vector when evaluating (4). However, using a suitable data structure to store the data during the process allows us to circumvent the need for this scan. We shall now introduce the two versions of SIA(M)E, the first of which works in time $O(nm)$, on the average, and latter in time $O(nm \log(nm))$, in the worst case. From the improvement of order of magnitudes follows an intuitive explanation of the name SIA(M)E, for 'SIA(M) Express'.

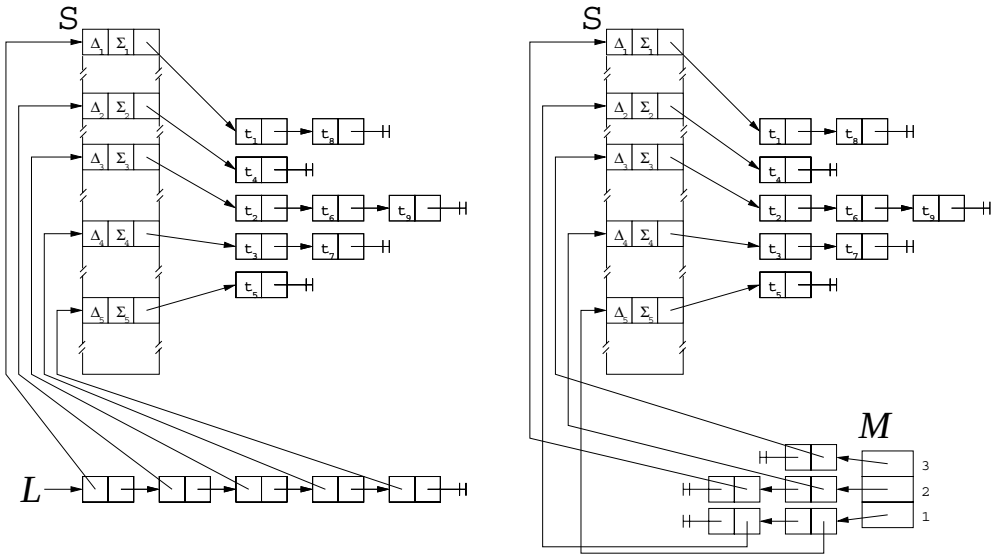

Figure 2: *An illustration of the data structures used in* SIA(M)E.

In Figure 2, above, we illustrate the working of SIA(M)E. Given the points t_i of template T and d_j of dataset D, the aim is to generate the structure $\mathcal{M}$ to the right bottom corner. The first version does this with the aid of an array, S, and a linked list, $\mathcal{L}$; the second version needs only the former. $\mathcal{M}$ stores the found $\langle$vector, point-set$\rangle$ pairs in a decreasing order by the number of points that was found to be translatable by the associated vector.

Let us briefly describe the structures before introducing the pseudo-codes. Each element of the array S contains three fields: ptr, Δ, and Σ. Field "ptr" is a pointer to a linked list comprising t_is that are translatable by a vector $\overline{v}$ which, itself, is stored to field Δ. Σ stores the number of t_is translatable by $\overline{v}$, that is, the size of the subset of T represented by this list.

For the first version of SIA(M)E, it is crucial that the (used) nodes in the array S are reachable in constant time. Hence it maintains a temporary linked list $\mathcal{L}$, whose elements comprise two pointer fields. Field "ptr" points to a used element in S, while "next" points to the next element in the list. $\mathcal{M}$ is an array of pointers, each of which is pointing to a linked list of the same form as that of $\mathcal{L}$.

Let us first introduce a function that shall be called by both versions of SIA(M)E. We denote by square brackets ([]) and an upwards-arrow ($\uparrow$) array indexing and element pointing, respectively. The function NEWLINK, below, takes two parameters, the former of which is either a datapoint or a pointer and the latter a pointer to a linked list. NEWLINK allocates a new node of the element type pointed to by the latter parameter, and adds this created node as the first element of the linked list. The value of the first parameter is stored to the "data" field of the created node. Note that because the newly created node is put at the very beginning of the list, NEWLINK is executed in constant time.

Function NEWLINK($data, next$)
1. $p \leftarrow$ **new**($next$);
2. $p \uparrow$ next $\leftarrow next$;
3. $p \uparrow$ data $\leftarrow data$;
4. **return** p;

3.2.1 Finding Patterns in $O(mn)$ Time on Average

In order to execute SIA(M)E in $O(mn)$ time, we need to choose the right element of S in constant time. A simple solution allocates space for the whole possible value range along each dimensions and use array indirection based on the translation vectors, $\overline{v} = d - t$, which select members of the SIA(M)E output set. This works in constant time, and so is efficient in this respect. The input dataset D for SIA(M)E, however, may be very large in quite ordinary applications. Furthermore, the data in our musical applications, in particular, is really quite sparse. Therefore, not only is there a potential for the data structures to be generated to become of excessive size, but it is very likely that a large proportion of the space tried to be allocated for them is never actually needed. So we have to balance the strictures of space against the time required to access the data for processing both by SIA(M)E and then whatever post-processing is to be done.

In this first version we do so by using a hash function F that hashes the translation vectors into an array of size $O(nm\mathcal{D})$ where m and n are as before and $\mathcal{D}$ is the number of dimensions represented in the input data. We use *closed hashing* (Weiss, 1993), in other words, only identical values are hashed to the same location of the array. To make the hashing work in an *expected* constant time, the frequency of *collisions*[4] should be kept low. This is possible with a hashing array of size approximately twice the number of the items to be hashed (Weiss, 1993). Moreover, a *secondary hashing procedure* (or a *resolution function*) is needed. The details, however, go beyond the scope of the current paper, and the reader is referred to Weiss (1993).

Given T, D, and S as input, the first version of SIA(M)E is as follows.

Algorithm SIA(M)E(T, D, S)
1. $p \leftarrow$ **nil**; $\mathcal{L} \leftarrow$ **nil**;
2. **for each** $t \in T$ **do**
3. **for each** $d \in D$ **do**
4. $\overline{v} \leftarrow d - t$;
5. $p \leftarrow S[F(\overline{v})]$;
6. $p \uparrow$ ptr $\leftarrow$NEWLINK($t, p \uparrow$ ptr);
7. $p \uparrow \Delta \leftarrow \overline{v}$;
8. $p \uparrow \Sigma \leftarrow p \uparrow \Sigma + 1$;
9. **if** $p \uparrow \Sigma = 1$ **then** $\mathcal{L} \leftarrow$NEWLINK($p, \mathcal{L}$);
10. $p \leftarrow \mathcal{L}$;
11. **while** $p \neq$ **nil do**
12. $tmp \leftarrow p \uparrow$ data $\uparrow \Sigma$;
13. $\mathcal{M}[tmp] \leftarrow$NEWLINK($p \uparrow$ data , $\mathcal{M}[tmp]$);
14. $p \leftarrow p \uparrow$ next;
15. **return** $\mathcal{M}$;

In the nested loops at lines 2–9, SIA(M)E operates by comparing each point t with each point d and uses the main structure S to store the ⟨vector, point-set⟩ pairs. The hashing function F (including also the resolution function) is used at line 5 to find the index in S corresponding with $\overline{v}$. After a new node storing the value t is added to the linked list associated with the vector, then the fields of S, at the element $F(\overline{v})$,

[4] A collision occurs when two different input values p_1 and p_2, $p_1 \neq p_2$, have an identical hashed value, $F(p_1) = F(p_2)$.

are updated. If the current vector has not been met before, a new node is allocated for the new vector, and a another node pointing to the one just allocated is added to the linked list $\mathcal{L}$.

Having executed these nested loops, the main structure S contains the $\langle$vector, point-set$\rangle$ pair information, and the list elements of $\mathcal{L}$ point to the nodes of S corresponding to the vectors that were found to be present in the input data. The length of the list $\mathcal{L}$ is $O(mn)$.

The next phase is to go through the $\langle$vector point-set$\rangle$ pairs (lines 11–14) and sort them according to their size counts. The pairs are stored in the structure $\mathcal{M}$ of size $O(mn)$. To give an example, see Figure 2, where $\Sigma_3 = 3; \Sigma_1 = \Sigma_4 = 2$; and $\Sigma_2 = \Sigma_5 = 1$).

The total expected time complexity of this first version of SIA(M)E is $O(mn)$. This is because the execution of line 5 takes a constant time on average [5], and the remaining lines within the nested **for** loops are executable in constant time. Thus, the execution of lines 2–9 takes $O(mn)$ on average, while the loop at lines 11–14 is clearly executable in $O(mn)$ time, even in the worst case.

3.2.2 Finding Patterns in $O(mn\log(mn))$ Time in the Worst Case

In the former solution, S comprised an array of size $2nm$ for each dimension of the vectors. It is in our interest to reduce that still further for our databases may be very large (consider, for example, databases of entire symphonies and oratorios). Our second version needs an array of size nm. On average it may be slower than the former version, but in the worst case it needs $O(mn\log(mn))$ time, where m is usually very small. The second version of SIA(M)E is as follows.

Algorithm SIA(M)E$_2(T, D, S)$
1. $p \leftarrow$ **nil**; $\mathcal{L} \leftarrow$ **nil**;
2. $i \leftarrow 0$;
3. **for each** $t \in T$ **do**
4. **for each** $d \in D$ **do**
5. $S[i].\Delta \leftarrow d - t$;
6. $S[i].\,ptr \leftarrow$ NEWLINK$(t, S[i].\,ptr)$;
7. $i \leftarrow i + 1$;
8. $S \leftarrow$MERGESORT(S);
9. $\mathcal{M} \leftarrow$MERGEDUPLICATES(S);
10. **return** $\mathcal{M}$;

This version of SIA(M)E first stores all the vectors with the associated t_i in S. Then S is sorted with respect to the vectors by the conventional MERGESORT sorting algorithm [6]. Finally, the function MERGEDUPLICATES, below, is executed.

Function MERGEDUPLICATES(S)
1. $j \leftarrow 0$;
2. **while** $j < (m \times n)$ **do**
3. $k \leftarrow 1$;
4. **while** $S[j].\Delta = S[j + k].\Delta$ **do**
5. $S[j + k].\,ptr \uparrow next \leftarrow S[j].ptr$;
6. $S[j].\,ptr \leftarrow S[j + k].ptr$;
7. $k \leftarrow k + 1$;
8. $S[j].\Sigma \leftarrow k$;
9. $\mathcal{M}[k] \leftarrow$NEWLINK$(S[j], \mathcal{M}[k])$;
10. $j \leftarrow j + k$;
11. **return** $\mathcal{M}$;

[5] In the worst case, however, it takes $O(mn)$ time and, therefore, the worst case time complexity for this version is $O((mn)^2)$.

[6] Being faster on average, QUICKSORT has the worst-case time-complexity of $O(n^2)$. Another reason for preferring here MERGESORT over QUICKSORT is because the implementation could be based on linked lists, which is a sensible case for QUICKSORT.

If the vectors at the consecutive indices in S are identical, MERGEDUPLICATES merges them; all these vectors are collected to the location, say j, where such an vector first occurred in S. Then the Σ field is updated, and an element at the corresponding index of $\mathcal{M}$ is created to point to $S[j]$.

The worst case time complexity for this second version of SIA(M)E is $O(mn \log(mn))$. The nested loops at lines 3–7 take time $O(mn)$, and it is well-known that MERGESORT has a worst case time complexity of $N \log N$ for sorting N objects. The function MERGEDUPLICATES runs in time $O(nm)$, since every location of S is visited exactly once (note that the inner loop is executed k times, after which the outer loop variable j is updated to $j + k$).

Instead of using MERGESORT and MERGEDUPLICATES, one possibility would have been sort S "on-the-fly" within the nested loops of SIA(M)E$_2$ by using, e.g., INSERTIONSORT (Weiss, 1993). This would, however, lead to a worst-case time-complexity of $O((nm)^2)$ (the case where the vectors are given in reversed order).

4 Using SIA(M)E Output to Find Matching Data

In this section, we explain how we reason about the structures produced by the SIA(M)E algorithm to generate descriptions of matches with particular properties between a template and a dataset. To do this, we introduce the notion of a *cover*, which allows us formally to describe the corresponding elements in a match, and some useful properties of covers which can be used to determine particular kinds of match.

The process works by selecting covers of a template T from $\mathcal{M}$ under a heuristic evaluation function composed from various possible elements, some of which are described below. Since we are now adding Selection and Evaluation to SIA(M)E, we call this completed algorithm SIA(M)ESE.

4.1 An Example Task: Template and Data

To exemplify and develop SIA(M)ESE in the context of educational error diagnosis, we use a familiar dataset: the French folk song *Frère Jacques*. The template is shown in Common Music Notation (CMN) in Figure 3.

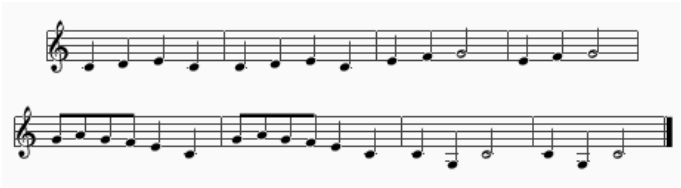

Figure 3: *The template, Frère Jacques, in CMN.*

In this example, we shall use a representation of pitch against onset time – though, as explained by Meredith *et al.* (prep), an arbitrary number of dimensions can be used. Our example representation (this time, for an implementation in Prolog) is shown in Figure 4; onset time in quaver (eighth-note) beats is the x axis, while pitch in scale degrees above middle C is the y axis. These choices of unit are arbitrary, and do not affect the operation of the algorithm. Barlines are indicated, for the convenience of the reader, by Prolog comments.

The dataset pattern we will match against in our first motivating example will be mostly correct apart from one pitch error and some small timing errors. This is not readily represented in CMN, so we give only the Prolog representation. The errors are that the middle two notes in the second bar (measure) are late, there is an incorrect note at the first quaver of bar 5, and there are timing errors between the third and fourth notes in bar 5 and the fifth and sixth notes of bar 6, leading to the entire pulse being delayed and then advanced, respectively. This is, of course, a fairly straightforward dataset for the purposes of example.

```
[event( 0, 0 ),     event( 2, 1 ),      event( 4, 2 ),     event( 6, 0 ),
 /*barline*/        event( 8, 0 ),      event( 10, 1 ),    event( 12, 2 ),
 event( 14, 0 ),    /*barline*/         event( 16, 2 ),    event( 18, 3 ),
 event( 20, 4 ),    /*barline*/         event( 24, 2 ),    event( 26, 3 ),
 event( 28, 4 ),    /*barline*/         event( 32, 4 ),    event( 33, 5 ),
 event( 34, 4 ),    event( 35, 3 ),     event( 36, 2 ),    event( 38, 0 ),
 /*barline*/        event( 40, 4 ),     event( 41, 5 ),    event( 42, 4 ),
 event( 43, 3 ),    event( 44, 2 ),     event( 46, 0 ),    /*barline*/
 event( 48, 0 ),    event( 50, -3 ),    event( 52, 0 ),    /*barline*/
 event( 56, 0 ),    event( 58, -3 ),    event( 60, 0 )]
```

Figure 4: *The template in our Prolog representation*

```
[event( 0, 0 ),        event( 2, 1 ),        event( 4, 2 ),
 event( 6, 0 ),        event( 8, 0 ),        event( 10.2, 1 ),
 event( 12.2, 2 ),     event( 14, 0 ),       event( 16, 2 ),
 event( 18, 3 ),       event( 20, 4 ),       event( 24, 2 ),
 event( 26, 3 ),       event( 28, 4 ),       event( 32, 6 ),
 event( 33, 5 ),       event( 34, 4 ),       event( 35.3, 3 ),
 event( 36.3, 2 ),     event( 38.31, 0 ),    event( 40.32, 4 ),
 event( 41.3, 5 ),     event( 42.29, 4 ),    event( 43.3, 3 ),
 event( 44.3, 2 ),     event( 46.1, 0 ),     event( 48.1, 0 ),
 event( 50.1, -3 ),    event( 52.1, 0 ),     event( 56.1, 0 ),
 event( 58.1, -3 ),    event( 60.1, 0 )]
```

Figure 5: *Example Data 1 in our prolog representation*

4.2 Covering a Matching Subset with a Pattern

Having computed $\mathcal{M}$ by executing the SIA(M)E algorithm, we search in it for a cover of the template, T. The intuition behind the cover is that we want an optimal (in some appropriate sense) set of fragmentary matches to our template, T, such that each datapoint in each of (the largest possible subset of) T and the matching subset of $\mathcal{M}$ is accounted for exactly once. So a cover describes a one-to-one mapping between (the largest possible subset of) T and that subset of $\mathcal{M}$ against which T has been (successfully) matched.

Stated more formally, a cover, $\mathcal{C}$, is a set of $\langle$vector,datapoint-set$\rangle$ pairs (*viz.*, a subset of $\mathcal{M}$), the union of whose datapoint-sets is the largest subset of T derivable thus from $\mathcal{M}$, the intersection of whose datapoint-sets is empty, and none of whose datapoint-sets contains a datapoint which maps, under the translation of its associated vector, to the same point as a member of a different datapoint-set in $\mathcal{C}$ under the translation of its own respective vector. This can be expressed logically as follows:

$$\begin{aligned} & \mathcal{C} \subseteq \mathcal{M} \\ \wedge \quad & \forall \overline{\mathrm{v}}_1, \overline{\mathrm{v}}_2, \mathrm{M}_1, \mathrm{M}_2 . ((\langle \overline{\mathrm{v}}_1, \mathrm{M}_1 \rangle \in \mathcal{C} \wedge \langle \overline{\mathrm{v}}_2, \mathrm{M}_2 \rangle \in \mathcal{C} \wedge \overline{\mathrm{v}}_1 \neq \overline{\mathrm{v}}_2) \rightarrow \\ & \qquad \neg \exists \mathrm{p}_1, \mathrm{p}_2 . (\mathrm{p}_1 \in \mathrm{M}_1 \wedge \mathrm{p}_2 \in \mathrm{M}_2 \wedge (\mathrm{p}_1 = \mathrm{p}_2 \vee \mathrm{p}_1 + \overline{\mathrm{v}}_1 = \mathrm{p}_2 + \overline{\mathrm{v}}_2))) \end{aligned} \tag{6}$$

However, (6) will typically generate many covers, with wildly different structural properties. We need to be able to pick an appropriate one (or ones). Often, we will want to use the *smallest cover* – that is, the one which divides the pattern up into fewest segments. This is because we are working under the assumption that our data will be coherent where it does not contain errors, and because we want to find the closest (*i.e.*, most coherent) match in the case where there is not a unique solution. In general, it will be infeasible to find the smallest cover by searching the possibilities generated by (6) above, because they are, in general, very many. This can be done straightforwardly using a best-first search algorithm (Russell and Norvig, 1995).

4.3 Inferring Matches and Differences from SIA(M) output

However, the concept of smallest cover does not quite give us the full range of SIA(M)'s capabilities. There may be several smallest covers, and they may have different properties which may be useful in different circumstances. For example, in the context of database search (see Section 5.1), we would want the sets in our cover to match (almost) time-contiguous sets in the data, so that they constitute a complete musical unit. Alternatively, in the learning case, we might want a match which covered as much of the temporal area of the dataset as possible, matching beginning and end most strongly, since these are cognitively the most memorable parts of a musical phrase. We call these the time-minimal and the time-maximal smallest covers, respectively. Note that while these and other data-dependent properties might be built in to the basic algorithms for SIA(M) (and SIA) we have chosen to keep them separate from the symmetrical multidimensional specification for reasons of clarity and neatness.

```
[pattern(vector(0,0),      [event(0,0), event(2,1), event(4,2), event(6,0),
                            event(8,0), event(14,0),event(16,2),event(18,3),
                            event(20,4),event(24,2),event(26,3),event(28,4),
                            event(33,5),event(34,4)]),
 pattern(vector(0.2,0),    [event(10,1),event(12,2)]),
 pattern(vector(0,2),      [event(32,4)]),
 pattern(vector(0.29,0),   [event(35,3),event(36,2),event(38,0),event(40,4),
                            event(41,5),event(42,4),event(43,3),event(44,2)]),
 pattern(vector(0.1,0),    [event(46,0),event(48,0),event(50,-3),event(52,0),
                            event(56,0),event(58,-3),event(60,0)])]
```

Figure 6: *Best fit for error set for Example Data 1*

The time-minimal smallest cover for the current example is, in fact, unique. It is shown in Figure 6. It can be expressed in various ways, depending on the $\simeq$ relation used to generate $\mathcal{M}$ (in this case, an arbitrary variance of ± 0.05 units around equality in both of the represented dimensions): the variance means that it becomes possible to represent the same set of points equivalently with different vectors, notably the pattern translated by the vector $\langle 0.29, 0 \rangle$ in Figure 6.

Another property of covers which is likely to be useful is that which we call *entanglement*. A cover, $\mathcal{C}$, is entangled if there exist two datapoints in T positioned in such a way that the order in which they lie along any of the dimensions represented is reversed under translation by their respective vectors in $\mathcal{C}$.

Of course, it is possible to use entanglement as a metric, not just a predicate: for example, one could count the number of distinct entanglements, or, for a finer measure, calculate the total angle between the entangled vectors – in both cases, the greater the value, the more questionable is the match which the cover represents for many purposes.

Because our current example is rather well-behaved, we do not need to appeal to entanglement to disambiguate our output. However, in the examples below, its use will be seen.

5 Applying SIA(M)E to Different Domains

5.1 Retrieving Polyphonic Music

One natural application domain for SIA(M)E is music information retrieval. As discussed in Section 2, there are only a few MIR algorithms capable of matching polyphonic data. Although some of the previous MIR algorithms may be more efficient in certain cases, as for instance the MONOPOLY algorithm in finding exact occurrences of a monophonic template, none of them is as flexible and versatile MIR algorithm as SIA(M)E. In the following, we illustrate this by showing how the working of the previous MIR algorithms can be simulated by using SIA(M)E without the need of changing SIA(M)E's original time complexity. In some cases, as in gapping, the original aim would not necessarily have been what have been yielded. In any case, we are still able to simulate the result which is not the case the other way around. None of

the previous algorithms can simulate the working of SIA(M)E, at least, not without major modification leading to a considerable increase of time complexity

Gapping Simulation. Even though it is superior to be able to deal with unlimited gaps, if wanted, we can simulate the gapping used in the algorithms by Crochemore *et al.* (2000) and Dovey (1999) (recall Subsection 2.1) with SIA(M)E. In this case, we use a mechanism that we call *labelling*. We define a label to be an attribute attached to every data point, but which is not be considered as another dimension by SIA(M)E. Thus, we are able to represent dimensions not exhibiting translation invariance.

In this case, SIA(M)E attaches an extra label, $K(p)$, for each two-dimensional point p in the dataset. The mapping K is a surjection to a range $0, \ldots, n'$, such that $K(p_1) > K(p_2)$ if and only if the *onset time* [7] of data point p_1 occurs later than that of p_2, and $K(p_1) = K(p_2)$ if and only if the onsets of p_1 and p_2 occur simultaneously.

Now, the modification to SIA(M)E is slight. Recall the logical expressions we presented on page 5. Having calculated Equations (3) and (4), instead of observing if there is an M in $\mathcal{M}$ such that $|M| = m$ (Equation (5)), the truth of the following sentence should be observed:

$$\exists M.\langle \overline{v}, M\rangle \in \mathcal{M} \wedge |M| = m \bigwedge_{i=1}^{m-1} (K(p_{i+1}) - K(p_i) \leq t). \tag{7}$$

Naturally, if (7) is true, an occurrence with spacing t has been found.

Whereas the allowing of a parametrized spacing makes the string matching approach more complex, for SIA(M)E it does not have such an affect; it causes only a minor practical overhead for the execution. When dealing with big databases and large values for t, SIA(M)E will clearly outperform Dovey's algorithm. Recall also that, as opposed to Dovey's algorithm, SIA(M)E will find also the transposed occurrences without any reiterations.

Matching Multiple Templates Simultaneously. Simultaneous searching for multiple musical templates, considered by (Holub *et al.*, 1999), can also be solved with SIA(M)E. Let $T_1 \cdots T_h$ be the h templates to be searched for in D, simultaneously within one execution of SIA(M)E. With SIA(M)E the solution is very straightforward; it needs no modification to the problem specification or to the implementation, whatsoever. We simply consider the numbering of the templates as another dimension to the problem. Because the points in the dataset have a constant value along this dimension, it is not possible for any two points p_i and p_j of two distinct templates to be translatable with a same vector.

Moreover, recall that the algorithm by Holub *et al.* (1999) allows no gaps between the consecutive elements under consideration. If this really is what is wanted, the gapping mechanism introduced above can naturally be considered here as well, and the desired solution is achieved by setting the gapping parameter as $t = 0$.

5.2 Other Applications

Being able to deal with any data represented as points in a multidimensional space, SIA(M)E has naturally many other application domains apart from the ones considered above. Next we compare it with the optimal combinatorial pattern matching algorithm for multidimensional data, introduced earlier, and show how SIA(M)E can be applied to retrieving pictorial data.

Matching Data in Higher Dimensions Including Bitmap Images. When dealing with bitmap images, it is not easy to compare the efficiencies of SIA(M)E and the optimal combinatorial pattern matching algorithms (recall Subsection 2.2, henceforth denoted by CPM) without running experiments with them. There is, however, something that can be observed without any experiments. Firstly, with the dimensionality of two, these images are rarely so large enough that the array version of SIA(M)E could not cope with them and, therefore, the $O(nmD)$ (D being the dimensionality of the problem) expected complexity is achievable. Secondly, by intuition, the best dataset images for SIA(M)E should be those containing a lot

[7] This is a domain spesific term, used here without loss of generality.

of empty space broken only with occasional occurrences by bits of interest (which is the case, for instance, with facsimiles). Nevertheless, this may not be the ideal case for CPM algorithms because, although being sparse, they have to represent the whole rectangular area. On the other hand, the sophisticated CPM algorithms also benefit from the sparseness, for in the matching process there are more areas that do not need closer examination.

Another issue making the comparison difficult is the fact that the O-notation hides many thing, e.g., all the constants involved. So it does not make any sense to try to get any strict solution to the inequality $O(nmD) \leq O((\frac{\hat{n}}{\hat{m}})^2 \log_c \hat{m}^2)$, where c is the size of the CPM alphabet — in particular because the input variables to the two algorithms are not even commensurable: there is no direct connection either between n and $\hat{n}$, or m and $\hat{m}$.

Therefore we give only the following loose estimation giving only an indication when SIA(M)E could be expected to perform better than the CPM algorithm, if the bitmap images under consideration are large enough.

Lemma 1 *Let a rectangular dataset bitmap image of $\hat{n}^2$ bits contain n 1s (and $(\hat{n}^2 - n)$ 0s); a rectangular query template of $\hat{m}^2$ bits contain m 1s, and let $m \leq n$. After some point h, such that $\hat{n}^2 \geq h$,* SIA(M)E *can be expected to perform better when $n \leq \frac{\hat{n}}{\hat{m}}$.*

Proof. In this case, clearly $D \leq \log_c \hat{m}^2$ holds. Therefore the following reasoning is also valid:

$$n \leq \frac{\hat{n}}{\hat{m}} \iff n^2 \leq (\frac{\hat{n}}{\hat{m}})^2 \Longrightarrow$$
$$nm \leq (\frac{\hat{n}}{\hat{m}})^2 \implies nmd \leq (\frac{\hat{n}}{\hat{m}})^2 \log_c \hat{m}^2.$$

□

Matching Structural Pictures. At the first sight, it may be rather surprising that SIA(M)E can also partly simulate the working of the somewhat more exotic 2-D string algorithms. This can be done in the case of type-2 matching, as defined in page 4. To make this possible, the two-dimensional datapoints are represented by their ranks in the vertical and horizontal directions. Moreover, for each such point representing an object of the image, the class of the object is represented via our labelling mechanism (Subsection 5.1). In this case only vectors, which connect a template point with a dataset point sharing an equivalent label, are considered and stored in the data structures. Thus, the matching becomes analogous to that of the original SIA(M)E algorithm.

6 Conclusion

In the current paper, we have investigated the problem of finding occurrences of a given query template in multidimensional datasets. Unlike previous approaches to the related problems, we have introduced a framework capable of finding such occurrences without paying any attention, whatsoever, to how many intervening datapoints there might be between the dataset points that match the consecutive template points. This feature is particularly useful, for instance, in music information retrieval, where polyphony and different ornamentations (such as grace notes) can cause such a problem. Moreover, in our framework, the occurrences of the template may be translated along any of the dimensions.

Furthermore, we have presented different possibilities to implement the framework, the most applicable of which runs in $O(mn \log(mn))$ time in the worst case, and requires $O(nm\mathcal{D})$ space, where m is the template size, n the dataset size and $\mathcal{D}$ the number of dimensions represented in the input data. The described algorithm, called SIA(M)E, is then equipped with application dependent selection and evaluation heuristics thus resulting in a version called SIA(M)ESE.

We have shown that SIA(M)E is efficient, versatile, and flexible. It can be applied to many different forms of pattern matching problems, including matching data from an education system, and polyphonic music matching where the cases of parametrized spacing between the consecutive elements and simultaneous searching of multiple templates can also be solved. We have also shown how SIA(M)E could be

applied to certain image retrieval problems, and estimated in which circumstances SIA(M)E could outperform an optimal combinatorial multidimensional string matching algorithm when applied to bitmap image matching.

We conclude that SIA(M)E seems to be superior to the other, previous algorithms considered in the paper. It is not the fastest algorithm for all possible cases, but it is capable of doing nearly anything that any of the others can with comparable or better time complexity, while the opposite direction is not true.

Acknowledgements

This work was partly supported by a grant #48313 from the Academy of Finland (K. Lemström) and EPSRC research grants GR/R25316 (K. Lemström) and GR/N08049 (D. Meredith).

References

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

Chang, S. K., Shi, Q., and Yan, C. W. (1987). Iconic indexing by 2-d strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **9**(3), 413–428.

Chang, S. K., Yan, C. W., Dimitrof, D., and Arndt, T. (1988). An intelligent image database system. *IEEE Transactions on Software Engineering*, **14(**5), 681–688.

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

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

Crochemore, M., Iliopoulos, C. S., Pinzon, Y. J., and Rytter, W. (2000). Finding motifs with gaps. In *First International Symposium on Music Information Retrieval (ISMIR'2000)*, Plymouth, MA. poster paper.

Dovey, M. J. (1999). An algorithm for locating polyphonic phrases within a polyphonic musical piece. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 48–53, Edinburgh.

Holub, J., Iliopoulos, C. S., Melichar, B., and Mouchard, L. (1999). Distributed string matching using finite automata. In *Proceedings of the 10th Australasian Workshop On Combinatorial Algorithms*, pages 114–128, Perth.

Kärkkäinen, J. and Ukkonen, E. (1994). Two and higher dimensional pattern matching in optimal expected time. In *Proceedings of the 5th Symposium on Discrete Algorithms*, pages 715–723. ACM-SIAM.

Lee, S. Y., Shan, M. K., and Yang, W. P. (1989). Similarity retrieval of iconic image databases. *Pattern Recognition*, **22**(6), 675–682.

Lemström, K. (2000). *String Matching Techniques for Music Retrieval*. Ph.D. thesis, Faculty of Science, University of Helsinki, Department of Computer Science. Report A-2000-4.

Lemström, K. and Tarhio, J. (2000). Detecting monophonic patterns within polyphonic sources. In *Content-Based Multimedia Information Access Conference Proceedings (RIAO'2000)*, pages 1261–1279, Paris.

Meredith, D., Lemström, K., and Wiggins, G. (prep.). MuSIA: An efficient algorithm for pattern induction in musical datasets.

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

Russell, S. and Norvig, P. (1995). *Artificial Intelligence – a modern approach*. Prentice Hall, New Jersey.

Tarhio, J. (1996). A sublinear algorithm for two-dimensional string matching. **17**(6), 833–838.

Uitdenbogerd, A. L. and Zobel, J. (1998). Manipulation of music for melody matching. In *ACM Multimedia 98 Proceedings*, pages 235–240, Bristol.

Weiss, M. A. (1993). *Data Structures and Algorithm Analysis in C*. Benjamin/Cummings, Redwood City, CA.

Wiggins, G. and Trewin, S. (2000). A system for the concerned teaching of musical aural skills. In *Proceedings of ITS2000*, number 1839 in LNCS. Springer-Verlag.