Matching Numeric Strings under Noise

Veli Mäkinen[1*], Gonzalo Navarro[2†], and Esko Ukkonen[1*]

[1] Department of Computer Science, P.O Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland.
e-mail: {vmakinen,ukkonen}@cs.helsinki.fi

[2] Center for Web Research, Department of Computer Science, University of Chile
Blanco Encalada 2120, Santiago, Chile.
e-mail: gnavarro@dcc.uchile.cl

Abstract. *Numeric string* is a sequence of symbols from an alphabet $\Sigma \subseteq \mathbb{U}$, where $\mathbb{U}$ is some numerical universe closed under addition and subtraction. Given two numeric strings $A = a_1 \cdots a_m$ and $B = b_1 \cdots b_n$ and a distance function $d(A, B)$ that gives the score of the best (partial) matching of A and B, the *transposition invariant distance* is $\min_{t \in \mathbb{U}}\{d(A + t, B)\}$, where $A + t = (a_1 + t)(a_2 + t) \ldots (a_m + t)$. The corresponding *matching* problem is to find occurrences j of A in B where $d(A+t, B_{j' \ldots j})$ is smaller than some given threshold and $B_{j' \ldots j}$ is a substring of B. In this paper, we give efficient algorithms for matching numeric strings — with and without transposition invariance — *under noise*; we consider distance functions $d(A, B)$ such that symbols $a \in A$ and $b \in B$ can be matched if $|b - a| \leq \delta$, or the κ largest differences $|b - a|$ can be discarded.

Keywords: approximate matching, transposition invariance, (δ, γ)–matching

1 Introduction

Transposition invariant string matching is the problem of matching two strings when all the characters of either of them can be "shifted" by some amount t. By "shifting" we mean that the strings are sequences of numbers and we add number t to each character of one of them.

Interest in transposition invariant string matching problems has recently arisen in the field of music information retrieval (MIR) [CIR98, LT00, LU00]. In music analysis and retrieval, one often wants to compare two music pieces to test how similar they are. A reasonable way of modeling music is to consider the pitches and durations of the notes. Often the durations are omitted, too, since it is usually possible to recognize the melody from a sequence of pitches. Hence, our focus is on distance measures for pitch sequences (of monophonic music) and their computation.

We studied the computation of edit distances under transposition invariance in [MNU03]. We noticed that sparse dynamic programming is useful in transposition

[*]Supported by the Academy of Finland under grant 22584.
[†]Supported by Millenium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile.

invariant matching, and obtained e.g. an $O(mn \log\log m)$ algorithm for transposition invariant longest common subsequence problem.

In this paper, we complement our earlier results by studying "non-gapped" distance measures for numeric strings. That is, we study measures where the ith symbol of the source is matched with the ith symbol of the target. To allow some noise in the values to be compared, we study measures that either allow matching symbols that approximately match (i.e. values are within δ distance apart), or allow discarding some amount (κ) of largest differences. We show how to compute the transposition invariant *Hamming distance* under noise in $O(m \log m)$ time, and transposition invariant sum of absolute differences (SAD) and maximum absolute difference (MAD) distances under noise in $O(m + \kappa \log \kappa)$ time, where m is the length of both strings to be compared.

For the corresponding search problems we only give the trivial algorithm that repeats the distance computation at each of the n text positions. However, the upper bound obtained this way for SAD distance is in fact the same as what is known *without* transposition invariance (see [Mut95], "weighted k–mismatches problem"). We also consider the combined search problem with SAD and MAD distances, known as the (δ, γ)*–matching* problem; we give an $O(mn)$ algorithm for the transposition invariant case of this problem. Again the best known upper bound for (δ, γ)–matching without transpositions is $O(mn)$ (because of the SAD distance).

In addition to the distance-specific results we introduce a more general approach to tackle with noise; many distance measures that allow matching two characters a and b for free when $|b-a| \leq \delta$ can be computed easily once the *set of possible matches* $|\mathbb{M}^\delta| = |\mathbb{M}^\delta|(A, B) = \{(i, j) \mid |b_j - a_i| \leq \delta, a_i \in A, b_j \in B\}$ has been computed. We show how to construct this set in $O(m \log |\Sigma| + n \log |\Sigma| + |\mathbb{M}^\delta| \min(\log(\delta + 2), \log\log m))$ time, where Σ is the alphabet of the two strings to be compared. After the set $\mathbb{M}^\delta$ is constructed, Hamming and MAD distances and (δ, γ)–matching under noise can be computed in time linear in the size of the set.

In the transposition invariant case, the construction of the sets of possible matches for all relevant transpositions is useful as well (e.g. for edit distance under noise). We show how to do this in linear time in the overall size of these sets (plus some additive factors of m,n, and $\log |\Sigma|$).

Some of the results of this paper appear in a technical report [MNU02].

2 Definitions

Let Σ be a finite numerical alphabet, which is a subset of some universe $\mathbb{U}$ that is closed under addition and subtraction. Let $A = a_1 a_2 \ldots a_m$ and $B = b_1 b_2 \ldots b_n$ be two *numeric strings* over Σ^*, i.e. the *symbols* (*characters*) a_i, b_j of the two strings are in Σ for all $1 \leq i \leq m, 1 \leq j \leq n$. We will assume w.l.o.g. that $m \leq n$. String A' is a *substring* of A if $A' = A_{i \ldots j} = a_i \ldots a_j$ for some $1 \leq i \leq j \leq m$. String A'' is a *subsequence* of A, denoted by $A'' \sqsubseteq A$, if $A'' = a_{i_1} a_{i_2} \ldots a_{i_{|A''|}}$ for some indexes $1 \leq i_1 < i_2 < \cdots < i_{|A''|} \leq m$.

When $m = n$, the following distances can be defined. The *Hamming distance* d_{H} between strings A and B is $d_{\mathrm{H}}(A, B) = m - |\{(i, i) \mid a_i = b_i, 1 \leq i \leq m\}|$. The *maximum absolute difference distance* d_{MAD} between A and B is $d_{\mathrm{MAD}}(A, B) = \max_{1 \leq i \leq m}\{|a_i - b_i| \mid 1 \leq i \leq m\}$. The *sum of absolute differences distance* d_{SAD} between A and B is $d_{\mathrm{SAD}}(A, B) = \sum_{i=1}^{m} |a_i - b_i|$. Note that d_{MAD} is in fact the maximum metric (l_∞ norm) and d_{SAD} the Manhattan metric (l_1 norm) when we interprete A and B as points in m dimensional Euclidean space.

String A is a *transposed copy* of B (denoted by $A =^t B$) if $B = (a_1 + t)(a_2 + t) \cdots (a_m + t) = A + t$ for some $t \in \mathbb{U}$. The transposition invariant versions of the above distance measures d_* where $* \in \{\mathrm{H}, \mathrm{MAD}, \mathrm{SAD}\}$ can now be stated as $d^{\mathrm{t}}_*(A, B) = \min_{t \in \mathbb{U}} d_*(A + t, B)$.

So far our definitions allow either only exact (transposition invariant) matches between some characters ($d^{\mathrm{t}}_{\mathrm{H}}$) or approximate match between all characters ($d^{\mathrm{t}}_{\mathrm{MAD}}$ and $d^{\mathrm{t}}_{\mathrm{SAD}}$). To relax these conditions, we introduce a constant $\delta > 0$. We write $a =^\delta b$ when $|a - b| \leq \delta$, $a, b \in \Sigma$. By replacing the equality $a = b$ with $a =^\delta b$ in the definition of $d^{\mathrm{t}}_{\mathrm{H}}$, we get a more error-tolerant version of the distance; let us denote the new distance $d^{\mathrm{t},\delta}_{\mathrm{H}}$. Similarly, by introducing another constant $\kappa > 0$, we can define distances $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}, d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ such that the κ largest differences $|a_i - b_i|$ are discarded.

The *approximate string matching problem*, based on the above distance functions, is to find the minimum distance between A and any substring of B. In this case we call A the *pattern* and denote it $P_{1\ldots m} = p_1 p_2 \cdots p_m$, and call B the *text* and denote it $T_{1\ldots n} = t_1 t_2 \cdots t_n$, and usually assume that $m << n$. A closely related problem is the *thresholded search problem* where, given P, T, and a threshold value $k \geq 0$, one wants to find all the text positions j such that $d(P, T_{j'\ldots j}) \leq k$ for some j'. We will refer collectively to these two closely related problems as the *search problem*.

Notice that searching under Hamming distance is known as the *k–mismatches problem* [Abr87, ALP01, BYG94, BYP96, GG86, LB86]. Also, a search problem related to distances d_{MAD} and d_{SAD} is known as the (δ, γ)–matching problem [CCIMP99, CILP01, CILPR02] in which occurrences j are searched for such that $d_{\mathrm{MAD}}(P, T_{j'\ldots j}) \leq \delta$ and $d_{\mathrm{SAD}}(P, T_{j'\ldots j}) \leq \gamma$.

Our complexity results are different depending on the form of the alphabet Σ. We will distinguish two cases. An *integer* alphabet is any alphabet $\Sigma \subset \mathbb{Z}$. For integer alphabets, $|\Sigma|$ will denote $\max(\Sigma) - \min(\Sigma) + 1$. A *real* alphabet will be any other $\Sigma \subset \mathbb{R}$, and then $|\Sigma|$ denotes the cardinality of Σ. For any string $A = a_1 \ldots a_m$, we will call $\Sigma_A = \{a_i \mid 1 \leq i \leq m\}$ the alphabet of A.

Last, we will need some orders for a set of pairs $P = \{(i, j)\}$, where $a_i \in A$ and $b_j \in B$. The *row order* of P is such that P is sorted first by i (in increasing order) and secondary by j (in increasing order). In *column order* P is sorted first by j and secondary by i. In *diagonal order* P is sorted first by $j - i$ and secondary by i.

3 Matching under Noise without Transposition Invariance

We will now present a general and efficient method that can be used with little modifications for solving both the k–mismatches problem and the (δ, γ)–matching problem. The time complexities will depend on the number of possible matches between pattern and text characters. A similar approach will also be used later in the transposition invariant case.

Let $\mathbb{M}^\delta(P, T) = \mathbb{M}^\delta = \{(i, j) \mid |p_i - t_j| \leq \delta\}$ be the *set of possible matches*. Let us assume that we are given $\mathbb{M}^\delta$ in diagonal order. By one traversal over $\mathbb{M}^\delta$ one can easily compute values $S(d)$ and $N(d)$ for each diagonal d, where $S(d) = \sum\{|p_i - t_j| \mid$

$(i,j) \in \mathbb{M}^\delta, j-i=d\}$ and $N(d) = |\{(i,j) \mid (i,j) \in \mathbb{M}^\delta, d = j-i\}|$.

Given the arrays $S(0 \ldots n-m)$ and $N(0 \ldots n-m)$, one can solve various problems. For example, all values d such that $S(d) \leq \gamma$ and $N(d) = m$, correspond to a (δ,γ)–match starting at position $d+1$ of the text. Similarly, if $N(d) \geq m-k$ when computed for $\mathbb{M}^0$, then there is an occurrence starting at position $d+1$ of the text for the k–mismatches problem.

Thus we have an $O(|\mathbb{M}^\delta| + n)$ algorithm for several problems, if we just manage to construct $\mathbb{M}^\delta$ in linear time in its size.

Theorem 1 *Given numeric strings P (pattern) and T (text) of lengths m and n ($m << n$), the set of possible matches $\mathbb{M}^\delta(P,T) = \{(i,j) \mid |p_i - t_j| \leq \delta\}$ can be constructed in time $O(|\Sigma| + m + n + |\mathbb{M}^\delta| \min(\log(\delta+2), \log\log m))$ on an integer alphabet, and in time $O(m \log|\Sigma| + n \log|\Sigma| + |\mathbb{M}^\delta| \min(\log(\delta+2), \log\log m))$ on a real alphabet. Within the same bounds, the set $\mathbb{M}^\delta$ can be constructed in row, column, or diagonal order.*

Proof. Let us first consider the integer alphabet with $\delta = 0$. We construct an array $L(1 \ldots |\Sigma|)$, where each entry $L(c)$ stores an increasing list of all positions of P, where character c occurs. Array L can obviously be constructed by one traversal over P in $O(|\Sigma| + m)$ time. The set $\mathbb{M}^0$ can then be constructed in column order in one traversal over T by concatenating lists $L(t_1), L(t_2), \ldots L(t_n)$. The running time is $O(m + n + |\Sigma| + |\mathbb{M}^0|)$.

For $\delta > 0$, we construct the array L as above but the traversal over T is now more complicated. To construct the column j of $\mathbb{M}^\delta$ we need to merge the $2\delta+1$ lists $L(t_j - \delta), \ldots, L(t_j + \delta)$ into a single list. This merging can be done using a priority queue $\mathcal{P}$ as follows. Add the first element, say i, of each list $L(c)$ into $\mathcal{P}$ by using i as the priority and c as the key. Then repeat the following until all lists are empty: Take the element with minimum priority, say (i,c), from $\mathcal{P}$, and add the next element from list $L(c)$ into $\mathcal{P}$. Column j of $\mathbb{M}^\delta$ is constructed by inserting pair (i,j) at the end of $\mathbb{M}^\delta$ at each step. The operations on a priority queue can be supported in $O(\log(\delta+2))$ time by using some standard implementation.

Since the priority values that need to be stored are in the range $[1,m]$, we can implement the priority queue more efficiently using a data structure of van Emde Boas [vEB77]. It supports, among other operations, retrieving the smallest value, inserting a new value, and deleting the smallest value, in $O(\log\log m)$ amortized time on values in the range $[1,m]$. We can store the values i using this data structure. Then we can repeat retrieving and deleting the smallest value i until the structure is empty, adding (i,j) at the end of $\mathbb{M}^\delta$ at each step. Thus the claimed bound on the integer alphabet follows.

When the alphabet is real, we can use exactly the same procedure, expect that the array L needs to be replaced by a binary search tree. It takes $O(m \log|\Sigma|)$ time to construct this search tree. For each character of T we need to do a range query on this tree to retrieve the lists of positions that correspond to characters in range $[t_j - \delta, t_j + \delta]$. This will take $O(n \log|\Sigma|)$ time. Merging can be done similarly as in the case of an integer alphabet, so the claimed bound follows.

Finally, the set is in column order after the above construction. Other orders can be constructed easily from the column order in time $O(m + n + |\mathbb{M}^\delta|)$. $\square$

The above theorem gives e.g. an $O(|\Sigma| + m + n + |\mathbb{M}^0|)$ time solution for the k–mismatches problem on an integer alphabet. This can be $\Theta(mn)$, but in the expected case it is much smaller. An expected bound $\Theta(mn/|\Sigma|)$ is easy to prove; see e.g. [BYP96], where the above algorithm was originally proposed for the k–mismatches problem.

4 Matching under Noise and Transposition Invariance

For this section, let $\mathbb{T} = \{t_i = b_i - a_i \mid 1 \le i \le m\} = \{t_i\}$ be the set of transpositions that make some characters a_i and b_i match. Note that the optimal transposition does not need, in principle, to be included in $\mathbb{T}$, but we will show that this is the case for $d_{\mathrm{H}}^{\mathrm{t}}$ and $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$. Note also that $|\mathbb{T}| = O(|\Sigma|)$ on an integer alphabet and $|\mathbb{T}| = O(m)$ in any case.

4.1 Hamming Distance

Let $A = a_1 \ldots a_m$ and $B = b_1 \ldots b_m$, where $a_i, b_i \in \Sigma$ for $1 \le i \le m$. We consider the computation of transposition invariant Hamming distance $d_{\mathrm{H}}^{\mathrm{t},\delta}(A, B)$. That is, we search for a transposition t maximizing the size of set $\{i \mid |b_i - (a_i + t)| \le \delta, 1 \le i \le m\}$.

Theorem 2 *Given two numeric strings A and B, both of length m, there is an algorithm for computing distance $d_{\mathrm{H}}^{\mathrm{t},\delta}(A, B)$ in $O(|\Sigma| + m)$ time on an integer alphabet, or in $O(m \log m)$ time on a general alphabet.*

Proof. It is clear that the Hamming distance is minimized for the transposition in $\mathbb{T}$ that makes the maximum number of characters match. What follows is a simple voting scheme, where the most voted transposition wins. Since we allow a tolerance δ in the matched values, t_i votes for range $[t_i - \delta, t_i + \delta]$. Construct sets $S = \{(t_i - \delta, \text{“open”}) \mid 1 \le i \le m\}$ and $E = \{(t_i + \delta, \text{“close”}) \mid 1 \le i \le m\}$. Sort $S \cup E$ into a list I using order

$$(x', y') <^{\mathrm{H}} (x, y) \ \text{ if } \ x' < x \text{ or } (x' = x \text{ and } y' < y),$$

where “open”<“close”. Initialize variable *count* $= 0$. Do for $i = 1$ to $|I|$ if $I(i) = (x, \text{“open”})$ then *count* $=$ *count*$+1$ else *count* $=$ *count*-1. Let *maxcount* be the largest value of *count* in the above algorithm. Then clearly $d_{\mathrm{H}}^{\mathrm{t},\delta}(A, B) = m - \mathit{maxcount}$, and the optimal transposition is any value in the range $[x_i, x_{i+1}]$, where $I(i) = (x_i, *)$, for any i where *maxcount* is reached. The complexity of the algorithm is $O(m \log m)$. Sorting can be replaced by array lookup when Σ is an integer alphabet, which gives the bound $O(|\Sigma| + m)$ for that case. $\square$

4.2 Sum of Absolute Differences Distance

We shall first look at the basic case where $\kappa = 0$. That is, we search for a transposition t minimizing $d_{\mathrm{SAD}}(A + t, B) = \sum_{i=1}^{m} |b_i - (a_i + t)|$.

Theorem 3 *Given two numeric strings A and B, both of length m, there is an algorithm for computing distance $d^{\mathrm{t}}_{\mathrm{SAD}}(A,B)$ in $O(m)$ time on both integer and general alphabets.*

Proof. Let us consider $\mathbb{T}$ as a multiset, where the same element can repeat multiple times. Then $|\mathbb{T}| = m$, since there is one element in $\mathbb{T}$ for each $b_i - a_i$, where $1 \leq i \leq m$. Sorting $\mathbb{T}$ in ascending order gives a sequence $t_{i_1} \leq t_{i_2} \leq \ldots \leq t_{i_m}$. Let t_{opt} be the optimal transposition. We will prove by induction that $t_{opt} = t_{i_{\lfloor m/2 \rfloor + 1}}$, that is, the optimal transposition is the median transposition in $\mathbb{T}$.

To start the induction we claim that $t_{i_1} \leq t_{opt} \leq t_{i_m}$. To see this, notice that $d_{\mathrm{SAD}}(A+(t_{i_1}-\epsilon),B) = d_{\mathrm{SAD}}(A+t_{i_1},B)+m\epsilon$, and $d_{\mathrm{SAD}}(A+(t_{i_m}+\epsilon),B) = d_{\mathrm{SAD}}(A+t_{i_m},B)+m\epsilon$, for any $\epsilon \geq 0$.

Our induction assumption is that $t_{i_k} \leq t_{opt} \leq t_{i_{m-k+1}}$ for some k. We may assume that $t_{i_{k+1}} \leq t_{i_{m-k}}$, since otherwise the result follows anyway. First notice that, independently of the value of t_{opt} in the above interval, the cost $\sum_{l=1}^{k} |b_{i_l} - (a_{i_l} + t_{opt})| + \sum_{l=m-k+1}^{m} |b_{i_l} - (a_{i_l} + t_{opt})|$ will be the same. Then notice that $\sum_{l=k+1}^{m-k} |b_{i_l} - (a_{i_l} + t_{i_{k+1}} - \epsilon)| = \sum_{l=k+1}^{m-k} |b_{i_l} - (a_{i_l} + t_{i_{k+1}})| + (m-2k)\epsilon$, and $\sum_{l=k+1}^{m-k} |b_{i_l} - (a_{i_l} + t_{i_{m-k}} + \epsilon)| = \sum_{l=k+1}^{m-k} |b_{i_l} - (a_{i_l} + t_{i_{m-k}})| + (m-2k)\epsilon$. This completes the induction, since we showed that $t_{i_{k+1}} \leq t_{opt} \leq t_{i_{m-k}}$.

The consequence is that $t_{i_k} \leq t_{opt} \leq t_{i_{m-k+1}}$ for maximal k such that $t_{i_k} \leq t_{i_{m-k+1}}$, that is, $k = \lceil m/2 \rceil$. When m is odd, it holds $m-k+1 = k$ and there is only one optimal transposition, $t_{i_{\lceil m/2 \rceil}}$. When m is even, one easily notices that all transpositions t_{opt}, $t_{i_{m/2}} \leq t_{opt} \leq t_{i_{m/2+1}}$, are equally good. Finally, the median can be found in linear time [BFPRT72]. □

To get a fast algorithm for $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ when $\kappa > 0$ largest differences can be discarded, we need a lemma that shows that the distance computation can be incrementalized from one transposition to another. Let $t_{i_1}, t_{i_2}, \ldots, t_{i_m}$ be the sorted sequence of $\mathbb{T}$.

Lemma 4 *Once values S_j and L_j such that $d_{\mathrm{SAD}}(A + t_{i_j}, B) = S_j + L_j$, $S_j = \sum_{j'=1}^{j-1} t_{i_j} - t_{i_{j'}}$, and $L_j = \sum_{j'=j+1}^{m} t_{i_{j'}} - t_{i_j}$, are computed, the values of S_{j+1} and L_{j+1} can be computed in $O(1)$ time.*

Proof. Value S_{j+1} can be written as

$$S_{j+1} = \sum_{j'=1}^{j} t_{i_{j+1}} - t_{i_{j'}} = \sum_{j'=1}^{j} t_{i_{j+1}} - t_{i_j} + t_{i_j} - t_{i_{j'}} = j(t_{i_{j+1}} - t_{i_j}) + S_j.$$

Similar rearranging gives

$$L_{j+1} = \sum_{j'=j+2}^{m} t_{i_{j'}} - t_{i_{j+1}} = (m-j)(t_{i_j} - t_{i_{j+1}}) + L_j.$$

Thus both values can be computed in constant time given the values of S_j and L_j (and $t_{i_{j+1}}$). □

Theorem 5 *Given two numeric strings A and B both of length m, there is an algorithm for computing distance $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}(A,B)$ in $O(m + \kappa \log \kappa)$ time on both integer and general alphabets. On integer alphabets, time $O(|\Sigma| + m + \kappa)$ can also be obtained.*

Proof. Consider the sorted sequence $t_{i_1}, t_{i_2}, \ldots, t_{i_m}$ as in the proof of Theorem 3. Clearly the candidates for the κ outliers (largest differences) are $M(k', k'') = \{t_{i_1}, \ldots, t_{i_{k'}}, t_{i_{m-k''+1}}, \ldots t_{i_m}\}$ for some $k' + k'' = \kappa$. The naive algorithm is then to compute the distance in all these $\kappa+1$ cases: Compute the median of $\mathbb{T} \setminus M(k', k'')$ for each $k' + k'' = \kappa$ and choose the minimum distance induced by these medians. These $\kappa + 1$ medians can be found as follows: First select values $t_{\kappa+1}$ and $t_{m-\kappa}$ using the linear time selection algorithm [BFPRT72]. Then collect and sort all values smaller than $t_{\kappa+1}$ or larger than $t_{m-\kappa}$. After selecting the median $m_{0,\kappa}$ of $\mathbb{T} \setminus M(0, \kappa)$ and $m_{\kappa,0}$ of $\mathbb{T} \setminus M(\kappa, 0)$, one can collect all medians $m_{k',k''}$ of $\mathbb{T} \setminus M(k', k'')$ for $k' + k'' = \kappa$, since the $m_{k',k''}$ values are those between $m_{0,\kappa}$ and $m_{\kappa,0}$. The $\kappa + 1$ medians can thus be collected and sorted in $O(m + \kappa \log \kappa)$ time, and the additional time to compute the distances for all of these $\kappa + 1$ medians is $O(\kappa m)$. However, the computation of distances given by consecutive transpositions can be incrementalized using Lemma 4. First one has to compute the distance for the median of $\mathbb{T} \setminus M(0, \kappa)$, $d_{\text{SAD}}(A + m_{0,\kappa}, B)$, and then continue incrementally from $d_{\text{SAD}}(A + m_{k',k''}, B)$ to $d_{\text{SAD}}(A + m_{k'+1,k''-1}, B)$, until we reach the median of $\mathbb{T} \setminus M(\kappa, 0)$, $d_{\text{SAD}}(A + m_{\kappa,0}, B)$ (this is where we need the medians sorted). Since the set of outliers changes when moving from one median to another, one has to add value $t_{i_{k'}} - t_{i_m}$ to S_m and value $t_{i_m} - t_{i_{k''}}$ to L_m, where S_m and L_m are the values given by Lemma 4 (here we need the outliers sorted). The time complexity of the whole algorithm is $O(m + \kappa \log \kappa)$. On an integer alphabet, sorting can be replaced by array lookup to yield $O(|\Sigma| + m + \kappa)$. $\square$

4.3 Maximum Absolute Difference Distance

We consider now how $d^{\text{t},\kappa}_{\text{MAD}}$ can be computed. In case $\kappa = 0$, we search for a transposition t minimizing $d_{\text{MAD}}(A + t, B) = \max_{i=1}^{m} |b_i - (a_i + t)|$. In case $\kappa > 0$, we are allowed to discard the k largest differences $|b_i - (a_i + t)|$.

Theorem 6 *Given two numeric strings A and B both of length m, there is an algorithm for computing distance $d^{\text{t},\kappa}_{\text{MAD}}(A, B)$ in $O(m + \kappa \log \kappa)$ time on both integer and general alphabets. On integer alphabets, time $O(|\Sigma| + m + \kappa)$ can also be obtained.*

Proof. When $\kappa = 0$ the distance is clearly $d^{\text{t}}_{\text{MAD}}(A, B) = (\max_i\{t_i\} - \min_i\{t_i\})/2$, and the transposition giving this distance is $(\max_i\{t_i\} + \min_i\{t_i\})/2$. When $\kappa > 0$, consider again the sorted sequence $t_{i_1}, t_{i_2}, \ldots, t_{i_m}$ as in the proof of Theorem 3. Again the κ outliers are $M(k', k'')$ for some $k' + k'' = \kappa$ in the optimal transposition. The optimal transposition is then the value $(t_{i_{m-k''}} + t_{i_{k'+1}})/2$ that minimizes $(t_{i_{m-k''}} - t_{i_{k'+1}})/2$, where $k' + k'' = \kappa$. The minimum value can be computed in $O(\kappa)$ time, once the $\kappa + 1$ smallest and largest t_i values are sorted. These values can be selected in $O(m)$ time and then sorted in $O(\kappa \log \kappa)$ time, or $O(|\Sigma| + \kappa)$ on integer alphabets. $\square$

4.4 Searching

Up to now we have considered distance computation. Any algorithm to compute the distance between A and B can be trivially converted into a search algorithm for P in T by comparing P against every text window of the form $T_{j-m+1\ldots j}$. Actually, we do not have any search algorithm better than this.

Lemma 7 *For distances $d^{\mathrm{t},\delta}_{\mathrm{H}}$, $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$, and $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$, if the distance can be evaluated in $O(f(m))$ time, then the corresponding search problem can be solved in $O(f(m)n)$ time.*

On the other hand, it is not immediate how to perform transposition invariant (δ,γ)–matching. We show how the above results can be applied to this case.

Note that one can find in $O(mn)$ time all the occurrences $\{j\}$ such that $d^{\mathrm{t}}_{\mathrm{MAD}}(P, T_{j-m+1\ldots j}) \leq \delta$, and all the occurrences $\{j'\}$ where $d^{\mathrm{t}}_{\mathrm{SAD}}(P, T_{j'-m+1\ldots j'}) \leq \gamma$. The (δ,γ)–matches are a subset of $\{j\} \cap \{j'\}$, but identity does not necessarily hold. This is because the optimal transposition can be different for $d^{\mathrm{t}}_{\mathrm{MAD}}$ and $d^{\mathrm{t}}_{\mathrm{SAD}}$.

What we need to do is to verify this set of possible occurrences $\{j\} \cap \{j'\}$. This can be done as follows. For each possible match $j'' \in \{j\} \cap \{j'\}$ one can compute limits s and l such that $d_{\mathrm{MAD}}(P+t, T_{j''-m+1\ldots j''}) \leq \delta$ for all $s \leq t \leq l$: If the distance $d = d_{\mathrm{MAD}}(P+t_{opt}, T_{j''-m+1\ldots j''})$ is given, then $s = t_{opt} - (\delta - d)$ and $l = t_{opt} + (\delta - d)$. On the other hand, note that $d_{\mathrm{SAD}}(P+t, T_{j''\ldots j''+m-1})$, as a function of t, is decreasing until t reaches the median of the transpositions, and then increasing. Thus, depending on the relative order of the median of the transpositions with respect to s and l, we only need to compute distance $d_{\mathrm{SAD}}(P+t, T_{j''-m+1\ldots j''})$ in one of them ($t = s$, $t = l$, or $t = t_{\lceil m/2 \rceil}$). This gives the minimum value for d_{SAD} in the range $[s, l]$. If this value is $\leq \gamma$, we have found a match.

One can see that using the results of Theorems 3 and 6 with $\kappa = 0$, the above procedures can be implemented so that only $O(m)$ time at each possible occurrence is needed. There are at most n occurrences to test.

Theorem 8 *Given two numeric strings P (pattern) and T (text) of lengths m and n, there is an algorithm for finding all the transposition invariant (δ,γ)–occurrences of P in T in $O(mn)$ time on both integer and general alphabets.*

4.5 Set of Possible Matches Revisited

Recall that an edit distance between two strings A and B is the cost of single symbol insertions, deletions, and substitutions to convert A into B. The unit cost or *Levenshtein distance* [Lev66] assigns cost 1 to each operation. If substitutions are forbidden and other operations have cost 1 the resulting distance is related to the longest common subsequence (LCS) of A and B. See e.g. [MNU03] and the references therein (like [Sel80]) for an introduction and formal definition of these edit distances.

For the sequel, it is only necessary to know the fact [MNU03] that the above edit distances can be computed efficiently once the set of possible matches $\mathbb{M} = \{(i,j) \mid a_i = b_j, a_i \in A, b_j \in B\}$ is given. Since we gave an efficient algorithm in Sect. 3 for constructing $\mathbb{M}^\delta = \{(i,j) \mid |b_j - a_i| \leq \delta\}$ we immediatedly have algorithms for edit distances under noise; just use the *sparse dynamic programming* algorithms of [MNU03] (or others' cited therein) on $\mathbb{M}^\delta$ instead of on $\mathbb{M}$. The effect of parameter δ is that two symbols can be matched if their values are close enough. For example, the method sketched above can be used to compute the *longest approximately common subsequence* of two numeric strings.

Now we focus on the transposition invariant edit distances under noise. Let us denote the size of $\mathbb{M}^\delta$ as $r = r(A, B, \delta) = |\mathbb{M}^\delta(A, B)|$. Let us redefine $\mathbb{T}$ in this section to be the set of those transpositions that make some characters between A and B

exactly δ apart, that is $\mathbb{T} = \{b_j - a_i \pm \delta \mid 1 \leq i \leq m, 1 \leq j \leq n\}$. The match set corresponding to a transposition $t \in \mathbb{T}$ is $\mathbb{M}_t^\delta = \{(i,j) \mid |b_j - a_i - t| \leq \delta\}$. Notice that there is always some $t \in \mathbb{T}$ whose match set $\mathbb{M}_t^\delta$ is equal to $\mathbb{M}_{t'}^\delta$, where $t' \in \mathbb{U}$. For most edit distances (like Levensthtein distance or LCS) same match set means that the distance will also be the same.

As noticed in [MNU03] (in the case $\delta = 0$) one could compute the above edit distances by running the basic dynamic programming algorithms [Sel80] over all pairs $(A+t, B)$, where $t \in \mathbb{T}$. In case $\delta > 0$, one would just interpret symbols a be b the same when $|b - a| \leq \delta$. One can obtain a more efficient method using advanced algorithms at each transposition. Let us first assume that $\delta = 0$ and let $r(A,B) = r(A,B,0)$. The following connection was shown in [MNU03]:

Lemma 9 ([MNU03]) *If an algorithm computes a distance $d(A,B)$ in $O(r(A,B)f(m,n))$ time, then there is an algorithm that computes the transposition invariant distance $d^{\mathrm{t}}(A,B) = \min_{t \in \mathbb{T}} d(A+t,B)$ in $O(mnf(m,n))$ time.*

As a consequence of the above lemma, we have $O(mn\,\mathrm{polylog}(n))$ time algorithms for different edit distances, since we manage to construct the match sets for all transpositions in $O(mn\,\mathrm{polylog}(n))$ time [MNU03]. In our noisy case, the above lemma extends to giving an $O(\sum_{t \in \mathbb{T}} |\mathbb{M}_t^\delta| f(m,n))$ algorithm, which equals $O(mn\,\mathrm{polylog}(n))$ when $\delta = 0$. To achieve total running time $O(\sum_{t \in \mathbb{T}} |\mathbb{M}_t^\delta| f(m,n))$, we still need to show that the sets $\mathbb{M}_t^\delta$ can be constructed in linear time in their overall size.

Theorem 10 *The* match sets $\mathbb{M}_t^\delta = \{(i,j) \mid a_i + t = b_j\}$, *each sorted in the column order, for all transpositions $t \in \mathbb{T}$, can be constructed in time $O(|\Sigma| + \delta mn)$ on an integer alphabet, and in time $O(m \log |\Sigma_A| + n \log |\Sigma_B| + |\Sigma_A||\Sigma_B| \log(\min(|\Sigma_A|, |\Sigma_B|)) + \sum_{t \in \mathbb{T}} |\mathbb{M}_t^\delta|)$ on a real alphabet.*

Proof. (We extend the proof given in [MNU03] for the case $\delta = 0$.) On an integer alphabet we can proceed naively to obtain $O(|\Sigma| + mn)$ time using array lookup to get the transposition where each pair (i,j) has to be added. For $\delta > 0$ each pair (i,j) is added to entries from $b_j - a_i - \delta$ to $b_j - a_i + \delta$, in $O(|\Sigma| + \delta mn)$ time.

The case of real alphabets is solved as follows. Let us first consider the case $\delta = 0$. Create a balanced tree $\mathcal{T}_A$ where every character $a = a_i$ of A is inserted, maintaining for each such $a \in \Sigma_A$ a list $\mathcal{L}_a$ of the positions i of A, in increasing order, such that $a = a_i$. Do the same for B and $\mathcal{T}_B$. This costs $O(m \log |\Sigma_A| + n \log |\Sigma_B|)$. Now, create an array $R(1 \ldots |\Sigma_A||\Sigma_B|)$, where each $R(k)$ stores the subset of the match set $\mathbb{M}_{t_k}$ (in column order), where $t_k = b - a$, $b_j = b$, and $a_i = a$ for all $(i,j) \in R(k)$. There is an entry in R for each possible pair (a,b), where $a \in \Sigma_A$, $b \in \Sigma_B$. Clearly R can be constructed in $O(mn)$ time once $\mathcal{T}_A$, $\mathcal{T}_B$, and the associated lists $\mathcal{L}$ are given. However, many pairs can produce the same transposition, thus we have to (i) sort R based on values t_k and (ii) merge the partial match sets that correspond to the same transposition. Phase (i) can be implemented to run in $O(|\Sigma_A||\Sigma_B| \log(\min(|\Sigma_A|, |\Sigma_B|)))$ time; consider w.l.o.g. that $|\Sigma_A| \leq |\Sigma_B|$. For fixed $a \in \Sigma_A$, we can get the $|\Sigma_B|$ transpositions $b - a$, $b \in \Sigma_B$, in increasing order by a depth-first search on $\mathcal{T}_B$. Thus we have $|\Sigma_A|$ lists, each containing $|\Sigma_B|$ transpositions already in order. Merging these lists (using standard techniques) takes $O(|\Sigma_A||\Sigma_B| \log |\Sigma_A|)$ time. Phase (ii) can be implemented to run in $O(mn)$ time; we can traverse through B and for each b_j add a new column to each $\mathbb{M}_t$, where $b_j - a = t$, $a \in \Sigma_A$. The correct set $\mathbb{M}_t$ can be found in constant time since we can maintain suitable pointers when sorting R in phase (i).

Finally, let us consider the case where $\delta > 0$. As discussed earlier, each pair (a, b) produces two relevant transpositions, $b - a - \delta$ and $b - a + \delta$. We proceed as before until array R is constructed and sorted. Consider sliding a window of length 2δ over the transpositions t_k in R. Let the middle point of current window be at t. Clearly, the pairs that are included in the current window produce the whole match set for transposition t. That is, partial match sets $R(l), R(l+1), \ldots, R(r)$ are merged into match set $\mathbb{M}_t^{\delta}$, where $t_l = b_j - a_i \geq t - \delta$ for (all) $(i, j) \in R(l)$, $t_r = b_{j'} - a_{i'} \leq t + \delta$ for (all) $(i', j') \in R(r)$, and $[l, r]$ is maximal range of R where this holds. The match sets change only when the middle points of the sliding window are from set $\mathbb{T} = \{b - a \pm \delta \mid a \in \Sigma_A, b \in \Sigma_B\}$. We can construct this set in $O(|\Sigma_A||\Sigma_B|)$ time. After sorting it, we can slide the window of length 2δ stopping at each middle point $t \in \mathcal{T}$, and construct each match set $\mathbb{M}_t^{\delta}$ by merging the match sets in the entries of R that are covered by the current window.

What is left is to consider how the merging can be done efficiently. Notice that the match sets corresponding to consecutive transpositions share a lot in common; the merging does not have to be done by brute force. We have two cases depending on whether the consecutive match sets differ (i) only by one entry of R, or (ii) by several entries. In case (i), the range $[l, r]$ of R is changed either to $[l+1, r]$ or to $[l, r+1]$. Both situations can be handled by one traversal over match set corresponding to $[l, r]$ and in the latter case also over $R(r+1)$. In case (ii), the range $[l, r]$ of R is changed either to $[l+k, r]$ or to $[l, r+k]$ for some k (by definition both ranges can not change at the same time). Let us consider the latter situation, since the first is analogous. It follows that $t_{r+1} = \cdots = t_{r+k}$, since otherwise there would be a relevant transposition $t_{r+k'} - \delta$, for some $1 < k' < k$, in between $t_r - \delta$ and $t_{r+k} - \delta$, which conflicts the fact that we are moving from one relevant transposition to the next. What follows is that we can preprocess R just like in the case when $\delta = 0$, merging consecutive entries of R having exactly the same transposition in $O(mn)$ time. After this is done, case (ii) can be handled just like case (i). The time complexity of this merging phase is bounded by $\sum_{t \in \mathbb{T}} |\mathbb{M}_t^{\delta}|$. □

Notice that $\sum_{t \in \mathbb{T}} |\mathbb{M}_t^{\delta}| \leq \delta mn$ on an integer alphabet. So the bound on a real alphabet is analogous to the bound on an integer alphabet.

5 Concluding Remarks

The motivation to study transposition invariant distances comes from music information retrieval. However, there are also other applications where these distance measures are useful. For example, in image comparison one could use the transposition invariant SAD distance to search for the occurrences of a small template inside a large image. With gray-level images the search would then be "lighting invariant". Combining other invariances, such as rotation or scaling invariance, with transposition invariance in a search algorithm, is a major challenge.

References

[Abr87] K. Abrahamson. Generalized string matching. *SIAM J. Computing*, 16(6):1039–1051, 1987.

[ALP01] A. Amir, M. Lewenstein, and E. Porat. Approximate Subset Matching with Don't Cares. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pp. 279–288, 2001.

[BYG94] R. Baeza-Yates and G. Gonnet. Fast string matching with mismatches. *Information and Computation*, 108(2):187–199, 1994.

[BYP96] R. Baeza-Yates and C. Perleberg. Fast and Practical Approximate String Matching. *Information Processing Letters*, 59:21–27, 1996.

[BFPRT72] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *J. Computer and System Sciences*, 7:448–461, 1972.

[CCIMP99] E. Cambouropoulos, M. Crochemore, C.S. Iliopoulos, L. Mouchard, and Yoan J. Pinzón. Algorithms for computing approximate repetitions in musical sequences. In *Proc. 10th Australian Workshop on Combinatorial Algorithms, AWOCA'99*, R. Raman and J. Simpson, eds., Curtin University of Technology, Perth, Western Australia, pp. 129–144, 1999.

[CIR98] T. Crawford, C.S. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology* 11:71–100, 1998.

[CILP01] M. Crochemore, C.S. Iliopoulos, T. Lecroq, and Y.J. Pinzón. Approximate string matching in musical sequences. In *Proc. Prague Stringoly Club (PSC 2001)*, M. Baliik and M. Simanek, eds, Czech Technical University of Prague, pp. 26–36, DC-2001-06, 2001.

[CILPR02] M. Crochemore, C.S. Iliopoulos, T. Lecroq, W. Plandowski, and W. Rytter. Three Heuristics for δ–Matching: δ–BM Algorithms. In *Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM'02)*, Springer-Verlag LNCS 2373, pp. 178–189, 2002.

[GG86] Z. Galil and R. Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17:52–54, 1986.

[LT00] K. Lemström and J. Tarhio. Searching monophonic patterns within polyphonic sources. In *Proc. RIAO 2000* , pp. 1261–1279 (vol 2), 2000.

[LU00] K. Lemström and E. Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *Proc. AISB 2000*, pp. 53–60, 2000.

[Lev66] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady* 6:707–710, 1966.

[LB86] G. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.

[Mut95] S. Muthukrishnan. New results and open problems related to non-standard stringology. In *Proc. 6th Annual Symposium on Combinatorial Pattern Matching (CPM'95)*, LNCS 937, pp. 298–317, 1995.

[MNU02] V. Mäkinen, G. Navarro, and E. Ukkonen. Algorithms for Transposition Invariant String Matching. *TR/DCC-2002-5*, Dept. of CS, Univ. Chile, July 2002, "ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/ti_matching.ps.gz"

[MNU03] V. Mäkinen, G. Navarro and E. Ukkonen. Algorithms for Transposition Invariant String Matching (Extended Abstract). In *Proc. 20th International Symposium on Theoretical Aspects of Computer Science (STACS 2003)*, Springer-Verlag LNCS 2607, pp. 191–202, Berlin, February, 2003.

[Sel80] P. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *J. of Algorithms*, 1(4):359–373, 1980.

[vEB77] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Proc. Letters* 6(3):80–82, 1977.