Using Edit Distance in Point-Pattern Matching*

Veli Mäkinen
Department of Computer Science
P.O Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
vmakinen@cs.Helsinki.FI

Abstract

Edit distance *is a powerful measure of similarity in string matching, measuring the minimum amount of insertions, deletions, and substitutions to convert a string into another string. This measure is often contrasted with time warping in speech processing, that measures how close two trajectories are by allowing compression and expansion operations on time scale. Time warping can be easily generalized to measure the similarity between 1D point-patterns (ascending lists of real values), as the difference between* i^{th} *and* $(i-1)^{th}$ *points in a point-pattern can be considered as the value of a trajectory at the time* i*. However, we show that edit distance is more natural choice, and derive a measure by calculating the minimum amount of space needed to insert and delete between points to convert a point-pattern into another. We show that this measure defines a metric. We also define a substitution operation such that the distance calculation automatically separates the points into matching and mismatching points. The algorithms are based on dynamic programming. The main motivation for these methods is two and higher dimensional point-pattern matching, and therefore we generalize these methods into the 2D case, and show that this generalization leads to an NP-complete problem. There is also applications for the 1D case; we discuss shortly the matching of tree ring sequences in dendrochronology.*

1 Introduction

Point-pattern matching problems arise in many applications. For example, in two-dimensional gel electrophoresis proteins are separated to form spots in a plane, and we want to match the same proteins from different experiments. Gels may contain partly different proteins and proteins may be distorted non-uniformly.

*Supported by the Academy of Finland under grant 22584.

In computational geometry, point-pattern matching problems under uniform transformations (like translation, rotation, and scaling) are widely studied (see [2] for a survey). Allowing non-uniform transformations complicates the problem substantially; the problem has been shown to be NP-complete [1].

In practise, greedy methods are used to cope with non-uniform transformations: The most likely matching pairs are fixed and the other point set is interpolated so that the fixed pairs become aligned. This procedure is repeated until all possible matching pairs are determined. Usually, the fixed pairs are determined interactively with the user. This kind of procedure was first proposed by Appel et. al. [3].

Although automated matching under non-uniform transformations is an NP-complete problem, one can hope to find working solutions at least for more limited definitions of the problem. Hoffman et. al. [10] have used the same definition for non-uniform matching as Akutsu et. al. [1] (see Sect. 2.2 for definition), but in addition they have constrained the slope difference between the matching pairs by a constant. The constant slope difference is a reasonable assumption when limited to a small portion of the point-pattern at a time. Dividing the point-pattern into small portions and finding for each portion a list of good local matches, one can finally get a robust global matching by voting among the local matches.

In this paper, we concentrate on a special case of non-uniform matching, namely on the one-dimensional case. Although the one-dimensional case is much simpler than the higher dimensional cases, it is still instructive to study the one-dimensional case for several reasons, e.g.:

(1) A general problem in many definitions describing point-pattern matching problems is that usually the mismatching points are separated from the matching points with the aid of user-given parameters. We argue that these parameters could be avoided by proper definitions in many problems. We demonstrate that one-dimensional non-uniform matching problem has a natural definition of this kind, with an efficient algorith-

mic solution.

(3) The one-dimensional case is interesting itself; there is an application in the matching of tree ring sequences in dendrochronology.

We discuss our definition for 1D non-uniform point-pattern matching in Sect. 2. First, we give a general overview of our method, then review some related work, and finally present our method in detail and consider the computational issues. In Sect. 3, we discuss the application in the matching of tree ring sequences in dendrochronology. In Sect. 4, we show that our definition leads to an NP-complete problem in 2D. We conclude in Sect. 5.

2 Non-uniform Point-Pattern Matching in 1D

2.1 Deriving an Edit Distance for 1D Point-Patterns

In string matching, edit distance is used to measure how close two strings are. Formally, let Σ be a vocabulary containing *characters*, and define *strings* $A, B \in \Sigma^*$. The *edit distance* between A and B is the minimum amount of character insertions, deletions, and substitutions that are needed to convert A to B [14].

The above definition can be applied to point-pattern matching considering *discrete point-patterns*, i.e. $\Sigma = 0, 1$ and $P \in \Sigma^*$, $P = p_1 p_2 \ldots p_n$, where $p_i = 1$ denotes a point in location $i \in [1 \ldots n]$. However, a straightforward adaptation of edit distance for point-patterns does not consider the special nature of the underlying problem; namely, the costs of edit operations should be different for values 1 and 0, because we are only interested in matching points (values 1) into each others. Therefore, we cannot simply apply methods developed for matching run-length compressed strings [7, 4, 5, 12].

First of all, the substitution operation should be forbidden, because substituting a point (1) with empty space (0) or vice versa would be too easy compared to insertions and deletions. Moreover, insertions and deletions should only be applied to empty space (0), because inserting or deleting a point (1) compares to the substitution operation, which is forbidden. However, as we want to compare point-patterns of different cardinality, we still need operations for inserting and deleting points. These operations can be defined by using the operations for inserting and deleting empty space: If all the space between two points is deleted, then the two points are *combined*. The inverse operation is to *double* a point and insert empty space in between. Finally, the limitation to discrete point-patterns is unnecessary, because we can simply redefine the costs of insertions and deletions as *Euclidean distances* corresponding to the amount of space that is inserted or deleted.

The sequence of operations that minimizes the distance between the point-patterns, gives also the *matching* of the point-patterns, i.e. a mapping of points from one point-pattern into another. This mapping is not necessarily a one-to-one mapping, as the points that are combined in one point-pattern are mapped into the same point in the other point-pattern. Nevertheless, a one-to-one mapping would be more useful in most applications. To get a one-to-one mapping, we need a method to separate points into two sets: matching points and mismatching points. We redefine the substitution operation for this purpose. As we have the costs of insertions and deletions already fixed, we have two criteria that fix also the cost of a substitution operation: (1) we seek for a one-to-one matching, (2) there should be as few mismatches as possible.

In Sect. 2.3, we give more precise formulations for the proposed distance measures, and prove some of their properties. First we review some related work to get our work in a context.

2.2 Related Work

Akutsu et. al. [1] have defined non-uniform matching problem as follows:

Definition 1 *Point matching under non-uniform distortion is, given positive real value ϵ, two point sets $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$ in d-dimensional Euclidean space, find a maximum matching $M = \{(a_{i_1}, b_{j_1}), \ldots, (a_{i_l}, b_{j_l})\}$ satisfying*

$$(\forall\, k)(\forall\, h \neq k)\left(\frac{1}{1+\epsilon} < \frac{|b_{j_h} - b_{j_k}|}{|a_{i_h} - a_{i_k}|} < 1 + \epsilon\right),$$

where $|a - b|$ denotes the Euclidean distance between a and b.

They prove that this problem is NP-hard when $d \geq 2$. For $d = 1$, they claim that a dynamic programming algorithm solves this problem with time complexity $O(m^2 n^2)$. They seem to have assumed that in an optimal one-dimensional solution the points in the matching M are in increasing order. However, this does not hold, which can be seen from the reduction given by Jokisch and Müller [11], to prove that Def. 1 is an NP-hard problem even when $d = 1$. Actually they prove that the *distance-congruence* (decision version of Def. 1) is NP-complete, i.e. to decide whether two equal size point sets can have a perfect matching such that a similar ϵ-condition as in Def. 1 is fullfilled. The NP-hardness of Def. 1 follows, because maximizing the size of M would solve the decision problem, too.

While Def. 1 is a reasonable definition for the higher dimensional cases, in 1D the situation is usually different; the point-patterns are sequences, where a change in the middle affects the whole end-sequence equally. An example of this kind of an application is the matching of tree ring sequences in dendrochronology. We will apply our method for this application in Sect. 3, where we also compare our method to another related work, which is a time warping approach to the problem.

Finally, we note that in a recent paper by Bookstein, Klein, and Raita [6], a new dissimilarity measure framework was proposed, called Fuzzy Hamming Distance. This framework has some connections to our edit distance for point-patterns, when discrete point-patterns are considered; in cases where two points are combined or a point is doubled, our space insertion and deletion operations could be defined in their framework. However, the difference comes in the *shift* [6] operation. In Fuzzy Hamming Distance the cost of a shift is proportional to the length of the shift, which is measured as the absolut difference in the locations of the corresponding "1"-bits (i.e. points in our definition). Our definition for a shift is relative to the previous matched pair, and thus works in situations where point-patterns can be seen as a sequences, where a change in the middle affects the whole end-sequence. Whereas the Fuzzy Hamming Distance works in situations where no inter-point relations exist.

2.3 Properties of the 1D Distance Measures

The distance measures introduced in Sect. 2.1 can be formalized as minimizing problems as follows:

Definition 2 *Point matching minimizing the edit distance under space insertions and deletions is, given two point sets $A = \{a_1, \dots, a_m\}$ and $B = \{b_1, \dots, b_n\}$ in ascending order in 1D Euclidean space, find a matching $M_{ID} = \{(a_{i_1}, b_{j_1}), \dots, (a_{i_l}, b_{j_l})\}$ of length $l \geq \max(m, n)$ that satisfies $a_{i_1} \leq \dots \leq a_{i_l}$ and $b_{j_1} \leq \dots \leq b_{j_l}$, and minimizes the cost $d_{ID}(A, B)$ of edit operations that are needed to convert A into B, where*

$$d_{ID}(A,B) = |a_{i_1} - b_{j_1}| + \sum_{k=2}^{l} \left| a_{i_k} - a_{i_{k-1}} - \left(b_{j_k} - b_{j_{k-1}}\right) \right|.$$

Similar definition can be given for the edit distance measure allowing substitutions:

Definition 3 *Point matching minimizing the edit distance under space insertions and deletions, and substitutions is, given two point sets $A = \{a_1, \dots, a_m\}$ and $B = \{b_1, \dots, b_n\}$ in ascending order in 1D Euclidean space, find a matching $M_{IDS} = \{(a_{i_1}, b_{j_1}), \dots, (a_{i_l}, b_{j_l})\}$ that satisfies $a_{i_1} \leq \dots \leq a_{i_l}$ and $b_{j_1} \leq \dots \leq b_{j_l}$, and minimizes the cost $d_{IDS}(A, B)$ of edit operations that are needed to convert A into B, where*

$$\begin{aligned} d_{IDS}(A,B) &= |a_{i_1} - b_{j_1}| + c(S_1) \\ &+ \sum_{k=2}^{l} \big(\left| a_{i_k} - a_{i_{k-1}} - (b_{j_k} - b_{j_{k-1}}) \right| \\ &\qquad + c(S_k) \big) \\ &+ c(S_{l+1}), \end{aligned}$$

and sets S_k contain all the mismatching points with cost $c(S_k)$ assigned to each set. The sets are defined as follows: $s \in S_1 \Rightarrow ((s \in A \land s < a_{i_1}) \lor (s \in B \land s < b_{j_1}))$, $(\forall\, k \in [2 \dots l])\, s \in S_k \Rightarrow ((s \in A \land a_{i_{k-1}} < s < a_{i_k}) \lor (s \in B \land b_{j_{k-1}} s < b_{j_1}))$, $s \in S_{l+1} \Rightarrow ((s \in A \land a_{i_l} < s) \lor (s \in B \land b_{j_l} < s))$.

The question now is how to select the costs $c(S_k)$ in Def. 3. As we earlier reasoned, there are two criteria that can be used: (1) we seek for a one-to-one matching, (2) there should be as few mismatches as possible. Consider now a cost $c(S_k)$ (costs for $c(S_1)$ and $c(S_{l+1})$ can be derived similarly). If we assign a higher cost for $c(S_k)$ than what would cost to include the points in S_k into the matching M_{IDS} by matching them with space insertion and deletion operations to the nearest matching points $a_{i_{k-1}}$, $b_{j_{k-1}}$, a_{i_k}, or b_{j_k}, then substitutions would never take a place. On the other hand, if we assign any smaller cost, substitutions would always be preferred in situations where many-to-one matchings could be formed. With this kind of an assignment, we always get a one-to-one matching and criterion (1) is satisfied. To satisfy criterion (2) we just take a cost equal to the cost of including all the points in S_k into the matching M_{IDS}, and prefer substitutions in even situations.

Now, it is clear that $d_{ID} = d_{IDS}$ if the costs for substitutions are selected as above. To minimize d_{IDS}, we can select any $M_{IDS} \subset M_{ID}$ such that all the *one-to-one subsets* of M_{ID} are included and a pair is included from each *many-to-one subset* of M_{ID}. With a one-to-one subset of M_{ID} we mean a pair (a_{i_k}, b_{j_k}) such that $(a_{i_{k'}}, b_{j_{k'}}) \notin M_{ID}$ if for $k \neq k'$, $a_{i_k} = a_{i_{k'}}$ or $b_{j_k} = a_{i_{k'}}$. With a many-to-one subset of M_{ID} we mean a sequence $(a_{i_k}, b_{j_k}), (a_{i_{k+1}}, b_{j_{k+1}}), \dots (a_{i_{k'}}, b_{j_{k'}})$ of maximum length such that either $a_{i_k} = a_{i_{k+1}} = \dots = a_{i_{k'}}$ and $b_{j_k} < b_{j_{k+1}} < \dots < b_{j_{k'}}$, or $b_{j_k} = b_{j_{k+1}} = \dots = b_{j_{k'}}$ and $a_{i_k} < a_{i_{k+1}} < \dots < a_{i_{k'}}$. Note that there are no many-to-many subsets in an optimal matching M_{ID} because any many-to-many subset could be replaced by two many-to-one subsets to obtain a matching with smaller cost.

There may still be many ways to choose $M_{IDS} \subset M_{ID}$ (and there may also be many matchings M_{ID} that have the same cost d_{ID}). All of them have the same overall cost, but the cost comes from two different sources: cost for matches and cost for mismatches. A natural way to select the best matching among these equal cost matches is to select the one that has the smallest cost for matches. The following example illustrates:

Example 4 *Consider point-patterns* $A = \{1, 6, 10, 11\}$ *and* $B = \{1, 2, 3, 12, 13\}$. *The edit distance allowing space insertions and deletions (Def. 2) gives four optimal matchings:*

$$
\begin{aligned}
M_{ID}^1 &= \{(1,1),(1,2),(1,3),(6,12),(10,13),(11,13)\} \\
M_{ID}^2 &= \{(1,1),(1,2),(1,3),(6,12),(10,12),(11,13)\} \\
M_{ID}^3 &= \{(1,1),(1,2),(6,3),(10,12),(11,13)\} \\
M_{ID}^4 &= \{(1,1),(6,2),(6,3),(10,12),(11,13)\}
\end{aligned}
$$

There are several ways to choose an optimal one-to-one matching M_{IDS}*, but choosing* $M_{IDS} = \{(1,3),(10,12),(11,13)\} \subset M_{ID}^2$ *gives the smallest score for the cost of matches.*

The following theorem gives further justification for our distance measure, as it proves that it is mathematically well-defined.

Theorem 5 *The edit distance for point-patterns under space insertions and deletions (Def. 2) is a metric.*

Proof. The following metric axioms hold (in the following, $d(A, B) = d_{ID}(A, B)$):

(0) **Identity:** if $A = B$ then $d(A, B) = 0 + \sum_{k=2}^{l} 0 = 0$, otherwise there is at least one term that is > 0, and so $d(A, B) > 0$ if $A \neq B$. All points are assumed to be finite, and there are finite number of points, and so $d(A, B) < \infty \; \forall\, A, B$.

(1) **Symmetry:** A counterargument proves the result; if $d(A, B) < d(B, A)$, we get $d(A, B) = |a_{i_1} - b_{j_1}| + \sum_{k=2}^{l} \left|a_{i_k} - a_{i_{k-1}} - \left(b_{j_k} - b_{j_{k-1}}\right)\right| = |b_{j_1} - a_{i_1}| + \sum_{k=2}^{l} \left|b_{j_k} - b_{j_{k-1}} - \left(a_{i_k} - a_{i_{k-1}}\right)\right| = d'(B, A) < d(B, A)$, which is a contradiction, because $d(B, A)$ is the minimum. Case $d(A, B) > d(B, A)$ goes similarly.

(2) **Triangle inequality:** Let M' be a matching of A to Z that minimizes $d(A, Z)$ (denote $d^{M'}(A, Z)$) and M'' a matching of Z to B that minimizes $d(Z, B)$ (denote $d^{M''}(Z, B)$). From these matchings, we can obtain a matching M''' of A to B in many ways (see Fig. 1 for an example). Not depending on the way we choose the matching M''' of A to B, we can associate with each pair in M''' a point in Z. For example, we can choose $Z' = \{z_p | (a_i, b_j) \in M''', (a_i, z_p) \in M', (z_p, b_j) \in M'', \nexists (a_i, z_{p'}) \in M'$ such that $z_{p'} < z_p\} = \{z_{r_1}, z_{r_2}, \ldots, z_{r_l}\}$ (see Fig. 1). Now,

$$
\begin{aligned}
d^{M'''}(A, B) &= |a_{i_1} - b_{j_1}| \\
&+ \sum_{k=2}^{l} \left|a_{i_k} - a_{i_{k-1}} - \left(b_{j_k} - b_{j_{k-1}}\right)\right| \\
&= |a_{i_1} + z_{r_1} - z_{r_1} - b_{j_1}| \\
&+ \sum_{k=2}^{l} \left|a_{i_k} - a_{i_{k-1}} + z_{r_k} - z_{r_k}\right. \\
&\quad \left. + z_{r_{k-1}} - z_{r_{k-1}} - \left(b_{j_k} - b_{j_{k-1}}\right)\right| \\
&\leq |a_{i_1} - z_{r_1}| \\
&+ \sum_{k=2}^{l} \left|a_{i_k} - a_{i_{k-1}} - \left(z_{r_k} - z_{r_{k-1}}\right)\right| \\
&+ |z_{r_1} - b_{j_1}| \\
&+ \sum_{k=2}^{l} \left|z_{r_k} - z_{r_{k-1}} - \left(b_{j_k} - b_{j_{k-1}}\right)\right| \\
&\leq d^{M'}(A, Z) + d^{M''}(Z, B),
\end{aligned}
$$

where we have used the facts that $Z' \subseteq \{z_p | (a_i, z_p) \in M'\}$ and $Z' \subseteq \{z_p | (z_p, b_j) \in M''\}$. Therefore, a matching M_{opt} that minimizes $d(A, B)$ satisfies $d^{M_{opt}}(A, B) \leq d^{M'''}(A, B) \leq d^{M'}(A, Z) + d^{M''}(Z, B)$. $\square$

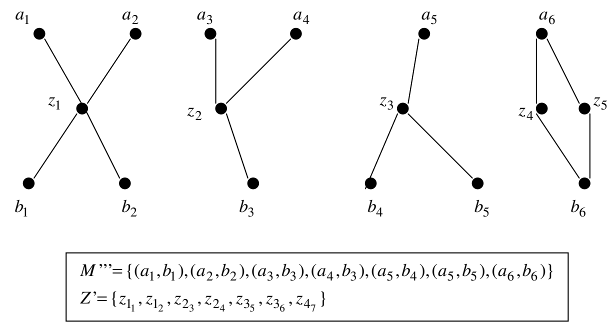

Figure 1. Obtaining a matching for A to B from matchings A to Z and Z to B.

Obvious corollary from Theorem 5 is that also d_{IDS} is a metric if the costs for substitutions are selected as described before.

2.4 Dynamic Programming Algorithms for the Distance Measures

The minimizing problems introduced in the previous section, can be solved by using dynamic programming. We first describe the method for the edit distance for 1D point-patterns under space insertions and deletions (Def. 2) based on the following recurrence:

$$
\begin{aligned}
d_{1,1} &= |a_1 - b_1|, \\
d_{i,1} &= d_{i-1,1} + a_i - a_{i-1}; \qquad 2 \le i \le m, \\
d_{1,j} &= d_{1,j-1} + b_j - b_{j-1}; \qquad 2 \le j \le n, \\
d_{i,j} &= min(d_{i-1,j-1} + |(a_i - a_{i-1}) - (b_j - b_{j-1})|, \\
&\qquad d_{i-1,j} + a_i - a_{i-1}, \\
&\qquad d_{i,j-1} + b_j - b_{j-1}); \qquad 2 \le i \le m, \\
&\qquad\qquad 2 \le j \le n.
\end{aligned} \tag{1}
$$

Theorem 6 *The recurrence (1) solves the minimizing problem in Def. 2, i.e.* $d_{ID}(A,B) = d_{m,n}$.

Proof. We use induction for the proof. The cases (1) $j = 1$, $1 \le i \le m$ and (2) $i = 1$, $1 \le j \le n$, are evident, because there is only one permitted matching for each pair of values i,j, and therefore it is minimum.

The induction assumption is that the recurrence holds for $i' + j' < i + j$. For $d_{i,j}$, we can write the recurrence

$$
\begin{aligned}
d_{i,j} = min \quad (\quad & d_{i,j'} + b_j - b_{j'}, \\
& d_{i',j} + a_i - a_{i'}, \\
& d_{i-1,j-1} + |(a_i - a_{i-1}) - (b_j - b_{j-1})|),
\end{aligned} \tag{2}
$$

where $1 < i' < i$ and $1 < j' < j$. This follows from the induction assumption and from the fact that all the matchings that are permitted in Def. 2 are included.

Using the induction assumption again, we have

$$
\begin{aligned}
d_{i,j-1} &\le d_{i,j-2} + b_{j-1} - b_{j-2} \\
&\le d_{i,j-3} + b_{j-1} - b_{j-3} \\
&\le \cdots \\
&\le d_{i,1} + b_{j-1} - b_1.
\end{aligned}
$$

Similar inequalities hold for $d_{j,i-1}$. Using these inequalities with recurrence (2), we get the recurrence (1), which completes the induction. □

We can evaluate the recurrence (1) by using dynamic programming, i.e. filling a table $(d_{i,j})$ row by row, so that each value $d_{i,j}$ can be calculated in constant time. Finally, value $d_{m,n}$ corresponds to the distance between point-patterns A and B. The calculation takes $O(mn)$ time.

An extension of the problem in Def. 2 is to find the best match for a short point-pattern $P = A$ inside a long point-pattern $S = B$. For this problem, we can use the same trick that is used for k-errors problem in string matching; the cells in the first row of matrix (d_{ij}) are assigned value zero. After calculating matrix (d_{ij}), the smallest value in the last row corresponds to the best match.

The minimizing problem in Def. 3 also has a dynamic programming solution. The following recurrence holds:

$$
\begin{aligned}
d_{i,j} \;=\; & min(d_{i',j'} + |(a_i - a_{i'}) - (b_j - b_{j'})| \\
& + substitute((i',j'),(i,j)),
\end{aligned} \tag{3}
$$

where $0 \le i' < i \le m, 0 \le j' < j \le n$, and $a_0 = b_0 = 0$, $d_{0,0} = 0$. The function $substitute((i',j'),(i,j))$ returns the cost of mismatches between points $a_{i'}$ and a_i, and between points $b_{j'}$ and b_j, as defined in Def. 3. The correctness of the recurrence follows from the definition. We can solve $d_{m,n}$ by using dynamic programming in $O(m^2n^2)$ time assuming that substitution costs can be evaluated in constant time.

If we use the substitution costs derived for getting the maximal one-to-one matching, we can speed up the calculation of $d_{IDS}(A,B)$ by calculating $d_{ID}(A,B)$ first and selecting $M_{IDS} \subset M_{ID}$ as described in Sect. 2.3. To get the M_{IDS} that also minimizes the cost of the matches, we have to examine all matchings M_{ID} that produce optimal d_{ID}. For each optimal M_{ID}, we can use one-dimensional dynamic programming to find M_{IDS} as follows: Divide M_{ID} into many-to-one/one-to-one subsets $M_{ID} = M_1 M_2 \ldots M_h$. For each pair $(a,b) \in M_1$, calculate the cost $c((a,b)) = |a - b|$. For each pair $(a,b) \in M_2$ calculate the cost $\min\{|a - a' + (b - b')| + cost((a',b')) \mid (a',b') \in M_1\}$, etc., until costs are calculated for each pair in M_h. The matching M_{IDS} can now be backtracked starting from the pair in M_h that has the smallest cost.

If we have s different matchings M_{ID} that produce optimal d_{ID}, the time complexity of the above method is $O(s\,(m^2 + mn + n^2))$. This follows from the fact that the time complexity of the one-dimensional dynamic programming algorithm can be expressed as $\sum_{i=1}^{h-1} x_i x_{i+1}$ under the constraints $\sum_{i=1}^{h} x_i < m + n$ and $x_i < \max(m,n)$. Thus, $\sum_{i=1}^{h-1} x_i x_{i+1} < \sum_{i=1}^{h-1} x_i \max(m,n) < \max(m,n)\,(m + n) < m^2 + mn + n^2$.

In the worst case, value s above can grow exponentially. When calculating d_{ID} one can also calculate the value of s. In cases $s > mn$, one can use directly recurrence (3) to get M_{IDS}. However, the cost of substitutions can not be calculated in constant time for all parameter values $((i',j'),(i,j))$, and the time complexity achieved this way becomes $O(m^3n + m^2n^2 + mn^3)$. We shall now discuss how this result is achieved.

The problem we are trying to solve is that of calculating the value $substitute((i',j'),(i,j))$ in recurrence (3) for all different parameter values. We remind that we consider the special case, where the substitution costs are defined to equal the cost for matching the mismatching points with space insertion and deletion operations to the nearest matching points $a_{i'}, b_{j'}, a_i$, or b_j. We consider different ranges for the values i' and j' for fixed i, j, separately. First, if $i' = 0$ (case $j' = 0$ is symmetric), then $a_{i'+1}, a_{i'+2}, \ldots, a_{i-1}$ are mismatches. In this case, we can assume that $j = 1$, because otherwise we could only produce matchings that have *crossing pairs* (e.g $A = 1, 3$, $B = 2, 4$, points $1, 2$ are mismatches; we could only produce a matching $M = \{(1, 4), (3, 2), (3, 4)\}$, where pairs $(1, 4)$ and $(3, 2)$ are crossing because $4 > 2$). It is easy to see that a matching that has crossing pairs, can always be replaced by a non-crossing one that has smaller cost. In our case this would mean that some mismatching points would be matched with each others. We do not need to consider such situations further, because the calculation of recurrence (3) already includes these possibilities with some other values for i' and j'. Thus, we may assume that for $i' = 0$, the value of j is 1. This gives only one possible matching for the mismatching points, $M_{i'=0} = \{(a_1, b_1), \ldots, (a_i, b_1)\}$, and the substitution cost becomes $c(M_{i'=0}) = |a_1 - b_1| + |a_i - a_1| - |b_1 - a_i|$, where the last term is subtracted because it is already calculated in the cost for matches.

We have two more situations to consider in calculating the value $substitute((i', j'), (i, j))$ in recurrence (3). Consider first situation $0 < i' < i - 1$ and $0 < j' < j - 1$. This means that we have mismatching points in both point-patterns between points $a_{i'}$ and a_i, and between $b_{j'}$ and b_j. As before, we only need to consider matchings for the mismatching points that are non-crossing. There are two alternatives: $M_1 = \{(a_{i'}, b_{j'}), \ldots, (a_{i-1}, b_{j'}), (a_i, b_{j'+1}), \ldots, (a_i, b_j)\}$ and $M_2 = \{(a_{i'}, b_{j'}), \ldots, (a_{i'}, b_{j-1}), (a_{i'+1}, b_j), \ldots, (a_i, b_j)\}$. So the substitution cost becomes $min(c(M_1), c(M_2))$, where $c(M_1) = |a_{i-1} - a_{i'}| + |(b_{j'+1} - b_{j'}) - (a_i - a_{i-1})| + |b_j - b_{j'+1}| - |(b_j - b_{j'}) - (a_i - a_{i'})|$ and $c(M_2) = |b_{j-1} - b_{j'}| + |(a_{i'+1} - a_{i'}) - (b_j - b_{j-1})| + |a_i - a_{i'+1}| - |(a_i - a_{i'}) - (b_j - b_{j'})|$, where the last terms are subtracted because they are already calculated in the cost for matches.

The remaining situation is $i' = i - 1$ (case $j' = j - 1$ is symmetric). This means that we only have mismatching points in one point-pattern. To match these mismatching points, there are $j - j' - 1$ matchings that need to be considered, namely $M_{j''} = \{(a_{i'}, b_{j'}), \ldots, (a_{i'}, b_{j''}), (a_i, b_{j''+1}), \ldots, (a_i, b_j)\}$ for all $j' < j'' < j$. For each of these matchings the substitution cost can be calculated in constant time, $c(M_{j''}) = |b_{j''} - b_{j'}| + |(b_{j''+1} - b_{j''}) - (a_i - a_{i'})| + |b_j - b_{j''+1}| - |(b_j - b_{j'}) - (a_i - a_{i'})|$, where again the last term is subtracted because it is already calculated in the cost for matches. We take $min\{c(M_{j''}) \,|\, j' < j'' < j\}$ as the substitution cost.

We have now considered all different parameter values for calculating the value $substitute((i', j'), (i, j))$ in recurrence (3). To summarize the time comlexity of this algorithm, we have for the case $i' = 0, j = 1$, m values for i where we need constant time calculation, for $j' = 0, i = 1$ we have n values for j where we need constant time calculation, for $0 < i' < i - 1, 0 < j' < j - 1$ we have mn values for different i, j and $(i - 2)(j - 2)$ values for different i', j' for each fixed i, j, and constant time calculation at each fixed i', j', i, j, and finally for $i' = i - 1$ (case $j' = j - 1$ is symmetric) there are mn values for i, j where $j - 2$ values must be examined to choose j' and for each fixed i, j, j', there are $j - j' - 1$ different matchings to be considered. Altogether, we have the time complexity $O(m) + O(n) + O(mnmn) + O(mnnn) + O(mnmm) = O(m^3 n + m^2 n^2 + mn^3)$.

3 Application to the Matching of Tree Ring Sequences in Dendrochronology

In dendrochronology tree ring sequences are aligned in order to *cross-date* the sequences. The alignment is based on the fact that a year's tree ring is about the same size for trees that grow in similar conditions. Due to changes in climate, a tree may sometimes produce two rings per year, or no ring at all. A *double ring* is produced e.g. when the growth slows down in the middle of the year because of cold weather, and then starts to grow again when the weather warms up. In some years, a tree might not grow at all, which produces a *missing ring*.

The goal is to get the total time series (master series) of the tree samples. As a subgoal, an optimal alignment between two tree ring sequences is needed. Also, a method to match a new tree ring sequence against a master series, is needed.

Statistical methods, like cross-correlation, can be applied to the cross-dating [8]. Usually these methods are robust, but they have problems in detecting missing or double rings.

A time warping approach has been introduced to cope with these missing and double rings [15, 16]; a transformation is looked for that transforms a sequence of tree ring widths A into a sequence that is as close to the compared sequence B as possible. The allowed transformations are I (inserting a missing ring), M (merging a double ring), and N (identity operation). The width of an inserted ring is taken as the average of the two surrounding rings. No costs are assigned for the operations, but the closeness is measured by taking the sum of the squared distances between the widths in the transformed sequence and the corresponding widths in the compared sequence.

A more general approach can be taken: Instead of modeling the tree ring sequences as sequences of tree ring widths, one can model them as point-patterns, where each point corresponds to the distance from the start of the sequence to the border of two tree rings. Now, aligning two tree ring sequences can be carried out by using the distance measures for 1D point-patterns introduced in Sect. 2.3. Also, aligning a new tree ring sequence P against an existing master series S can be done by searching the best match for P in S as described in Sect. 2.4.

The main difference between these two methods is due to the difference between edit distance and time warping: In edit distance, we measure the minimum amount of work to convert an object into another object. In time warping, the goal is to find the sequence of operations that minimizes the distance between objects. Time warping can be used in multi-dimensional applications, where the costs for edit operations cannot be given explicitly, but the distance between objects after the transformation can be measured. The matching of tree ring sequences is however a one-dimensional application, because a tree ring sequence can be described as a point-pattern, and therefore also an edit distance approach can be applied.

There are some application dependent details that must be included into the point-pattern based method that are described in [16] in the context of the time warping approach, like data standardization, and limiting the amount of missing or double rings. We will not cover these here, as they are (together with experimental evaluation) subject of a separate study. Also, it shall be seen whether these methods can be applied to answer the hard multiple alignment problem of constructing the master series.

4 Distance Measures in 2D Point-Pattern Matching

4.1 Generalizing the 1D Edit Distance into 2D

The edit distance measure for point-patterns in 1D can be easily generalized into 2D. The only difference now is that the order in which the points are matched can not be fixed. We have the following definition:

Definition 7 *Point matching minimizing the* 2D movement distance *is, given two point-patterns $A = \{a_1, \dots, a_m\}$ and $B = \{b_1, \dots, b_n\}$, where a point is represented as a vector in 2D Euclidean space, find a matching $M = \{(a_{i_1}, b_{j_1}), \dots, (a_{i_l}, b_{j_l})\}$ of length $l \geq \max(m, n)$ that minimizes the cost $d_M(A, B)$ of movement needed to align all the points in A and B, where*

$$d_M(A,B) = |a_{i_1} - b_{j_1}| + \sum_{k=2}^{l} |a_{i_k} - a_{i_{k-1}} - (b_{j_k} - b_{j_{k-1}})|.$$

The name 2D movement distance is easily explained: Consider two point-patterns such that the other one is in fixed position. Move the other point-pattern above the other until all points become aligned during the movement. The distance d_M gives the minimum movement.

Only a minor modification for the proof of Theorem 5 is needed for showing that the 2D movement distance is also a metric.

Although Def. 2 gives a well-justified measure of similarity for point-patterns in 1D, the situation is usually different in 2D. Definition 7 tries to capture the "secret" sequence order of a point-pattern in which the points influence each others. In 2D, it would be more rightful to think that a point is influenced by its neighboring points, not by an arbitrary single point.

In other words, we would like *local similarity* to preserve. That is, a point is matched nearby where its neighboring points are matched. The Def. 1 in Sect. 2.2 by Akutsu et. al. [1] fulfills this criterion. However, their definition uses a user-given parameter ϵ to separate the mismatches from the matches. We wish to overcome the need of free parameters, and therefore we introduce a distance measure approach to replace Def. 1 so that we could apply the same methodology as we used in the 1D case for separating the mismatches from the matches.

Consider that we have a matching where local similarity is preserved. We can measure how well local similarity is preserved by comparing the matching of a point with the matchings of its neigboring points. How do we measure neighborness? Obviously we can weight the effect of a point to an other point as a function of their distance; the nearer the points are, the more effect there is in between. As the matching of each point is affected by the matchings of all other points, we take the weighted average of these effects. We get the following definition, where also symmetry has been included:

Definition 8 *Point matching minimizing the* Interpolation distance *is, given two point-patterns $A = \{a_1, \dots, a_m\}$ and $B = \{b_1, \dots, b_n\}$, where a point is represented a a vector in Euclidean space, and a weighting parameter $r > 0$, find a matching $M = \{(a_{i_1}, b_{j_1}), \dots, (a_{i_l}, b_{j_l})\}$ of length $l \geq \max(m, n)$ that minimizes the cost $d_I(A, B)$ needed to align all the points in A and B, where*

$$\begin{aligned} d_I(A,B) &= \sum_{k=1}^{l} \Bigg| b_{j_k} - \Bigg(a_{i_k} \\ &\quad + \frac{\sum_{p \neq k} \frac{(b_{j_p} - a_{i_p})}{(|a_{i_k} - a_{i_p}|)^r + (|b_{j_k} - b_{j_p}|)^r}}{\sum_{p \neq k} \frac{1}{(|a_{i_k} - a_{i_p}|)^r + (|b_{j_k} - b_{j_p}|)^r}} \Bigg) \Bigg|. \end{aligned}$$

To separate the mismatches from the matches using the Interpolation distance, we can define the cost of a substitution as the distance to the nearest matching point in the other point-pattern after the interpolation effect of the matching is taken into account. This way all the points that benefit the overall matching are included into the matching, and those that do not benefit are calculated as mismatches with costs equal to the minimum cost of including them into the matching.

4.2 Complexity of the 2D Movement Distance

The following shows that the generalization of edit distance into 2D also changes the complexity of the problem:

Theorem 9 *To determine, whether there is a point matching such that the 2D movement distance is less or equal to some constant $k > 0$, is an NP-hard problem.*

Proof. We use reduction from the *path-TSP* [9], which is a special case of the traveling salesperson problem, where every city is traversed, but the salesperson is not required to return to the start of the tour. Let $P = \{p_1, p_2, \ldots, p_m\}$ be a set of points in a plane. Construct point-patterns $A = p_1^1, p_1^2, \ldots, p_1^{m-1}$ and $B = p_2^1, p_3^2, \ldots, p_m^{m-1}$ such that $p_1^i - p_{i+1}^i = p_1 - p_{i+1}$ for all $1 \leq i < m$, and $|p_1^i - p_1^j| > 2k$ for all $i \neq j$. The construction can be done in linear time, e.g. by mapping p_1 into (discretized) real plane in 2k intervals to construct A, and then B by a simple calculation.

Now, if there is a movement of length k or less to align A and B, it must define a matching that is a permutation of $M = \{(p_1^1, p_2^1), (p_1^2, p_3^2), \ldots, (p_1^{m-1}, p_m^{m-1})\}$. Whenever such a matching is found, there is also a path in P of length k or less. □

It should also be noted that even if we know all the pairs that are included in the matching M, finding the right order for the pairs is itself NP-hard. This follows from the proof of Theorem 8, because the reduction gives the pairing.

5 Conclusions and Future Work

We have considered the use of edit distance in point-pattern matching. For the 1D case, we have introduced polynomial time algorithms for finding matchings minimizing the edit distance allowing space insertions and deletions, and substitutions. We have also proven that the distance measures are sensible in a mathematical point of view, as they obey metric properties.

Although there are applications for the 1D case, the actual motivation for these methods has been 2D point-pattern matching. Unfortunately, the generalization of edit distance into 2D gives an NP-complete problem.

We are currently examining complexity issues related to the Interpolation distance. Actually, we can prove that the Interpolation distance is an NP-hard problem, by using a reduction from 3-PARTITION (similar to the one given by Jokisch and Müller [11] for the distance-congruence problem). The reduction also gives an inapproximativeness result for the problem. However the reduction produces point-patterns that are very sparse, and gives no clue of the complexity of problem, when restricted to dense instances, like those in the 2D gel electrophoresis application.

Acknowledgements

I would like to thank Esko Ukkonen and Taneli Mielikäinen for the fruitful discussion and valuable comments.

References

[1] T. Akutsu, K. Kanaya, A. Ohyama, and A. Fujiyama. Matching of Spots in 2D Electrophoresis Images. Point Matching Under Non-uniform Distortions. In *Proc. CPM'99*, LNCS 1645, pp. 212-222, 1999.

[2] H. Alt and L. Guibas. Discrete Geometric Shapes: Matching, Interpolation, and Approximation, In J.-R. Sack, J. Urrutia, editors, *Handbook of Computational Geometry*, pp. 121 - 153. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

[3] R. D. Appel, J. R. Vargas, P. M. Palagi, D. Walther, and D. F. Hochstrasser. Melanie II - A Third Generation Software Package for Analysis of Two-dimensional Electrophoresis Images: II. Algorithms. *J. Electrophoresis* 18, pp. 2735-2748, 1997.

[4] A. Apostolico, G. Landau, and S. Skiena. Matching for run-length encoded strings. *J. of Complexity*, 15:4–16, 1999. (Also at Sequences '97, Positano Italy, June 11–13, 1997).

[5] O. Arbell, G. Landau, and J. Mitchell. Edit distance of run-length encoded strings. Submitted for publication, August 2000.

[6] A. Bookstein, S. T. Klein, and T. Raita. Fuzzy Hamming Distance: A New Dissimilarity Measure. In *Proc. CPM 2001*, LNCS 2089, pp. 86–97, 2001.

[7] H. Bunke and J. Csirik. An improved algorithm for computing the edit distance of run-length coded strings. *Information Processing Letters*, 54(2):93–96, 1995.

[8] E. R. Cook and L. A. Kairiukstis, editors. *Methods of Dendrochronology*. Kluwer Academic Publishers, 1990.

[9] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proc. 8th Annual ACM Symposium on Theory of Computing*, pp. 10-12, May 1976.

[10] F. Hoffmann, K. Kriegel, and C. Wenk. An applied point pattern matching problem: comparing 2D patterns of protein spots. *Discrete Applied Mathematics*, 93, pp. 75-88, 1999.

[11] S. Jokisch and H. Müller. Inter-Point-Distance-Dependent Approximate Point Set Matching. *Research Report* No. 653, University of Dortmund, Department of Computer Science, July 1997.

[12] V. Mäkinen, G. Navarro, and E. Ukkonen. Approximate Matching of Run-Length Compressed Strings. In *Proc. CPM 2001*, LNCS 2089, pp. 31–49, 2001.

[13] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley Publishing Company, 1983.

[14] R. Wagner and M. Fisher. The string-to-string correction problem. *J. Assoc. Comput. Mach.* 21, pp. 168-178, 1974.

[15] P. C. Van Deusen. A dynamic program for cross-dating tree rings. *Canadian Journal of Forest Research*, 20:200-205, 1989.

[16] C. Wenk. Applying an Edit Distance to the Matching of Tree Ring Sequences in Dendrochronology. In *Proc. CPM'99*, LNCS 1645, pp. 223-242, 1999.