Rotation and Lighting Invariant Template Matching [*]

Kimmo Fredriksson[†] Veli Mäkinen[‡] Gonzalo Navarro[§]

Abstract

We address the problem of searching for a two-dimensional pattern in a two-dimensional text (or image), such that the pattern can be found even if it appears rotated and it is brighter or darker than its occurrence. Furthermore, we consider approximate matching under several tolerance models. We obtain algorithms that are almost optimal both in the worst and the average cases simultaneously. The complexities we obtain are very close to the best current results for the case where only rotations, but not lighting invariance, are supported. These are the first results for this problem under a combinatorial approach.

1 Introduction

We consider the problem of finding the occurrences of a two-dimensional *pattern* of size $m \times m$ cells in a two-dimensional *text* of size $n \times n$ cells, when all possible rotations of the pattern are allowed and also the pattern and the text may have differences in brightness. This stands for *rotation and lighting invariant template matching.* Text and pattern are seen as images formed by cells, each of which has a gray level value, also called a color.

Template matching has numerous important applications from science to multimedia, for example in image processing, content based information retrieval from image databases, geographic information systems, and processing of aerial images, to name a few. In all these cases, we want to find a small subimage (the pattern) inside a large image (the text) permitting rotations (a small degree or any). Furthermore, pattern and text may have been photographed under different lighting conditions, so one may be brighter than the other.

The traditional approach to this problem [5] is to compute the cross correlation between each text location and each rotation of the pattern template. This can be done reasonably efficiently using the Fast Fourier Transform (FFT), requiring time $O(Kn^2 \log n)$ where K is the number of rotations sampled. Typically K is $O(m)$ in the two-dimensional (2D) case, and $O(m^3)$ in the 3D case, which makes the FFT approach very slow in practice. In addition, lighting-invariant features may be defined in order to make the FFT insensitive to brightness. Also, in many applications, "close enough" matches of the pattern are also accepted. To this end, the user may specify, for

[*]A short version of this paper appeared in *Proc. LATIN 2004*, pp. 39–48, LNCS 2976.

[†]Department of Computer Science, University of Joensuu. `kfredrik@cs.joensuu.fi`.

[‡]Department of Computer Science, University of Helsinki. `vmakinen@cs.helsinki.fi`. Part of this work was done while visiting University of Chile under a researcher exchange grant from University of Helsinki.

[§]Department of Computer Science, University of Chile. `gnavarro@dcc.uchile.cl`. Funded by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.

example, a parameter κ such that matches that have at most κ differences with the pattern should be accepted, or a parameter δ such that gray levels differing by no more than δ are considered equal. The definition of the matching conditions is called the "matching model" in this paper.

Rotation invariant template matching was first considered from a combinatorial point of view in [16, 17]. Since then, several fast filters have been developed for diverse matching models [18, 9, 19, 14, 13, 15]. These represent large performance improvements over the FFT-based approach. The worst-case complexity of the problem was also studied [1, 14, 15]. However, lighting invariance has not been considered in this scenario.

On the other hand, *transposition invariant* string matching was considered in music retrieval [7, 20]. The aim is to search for (one-dimensional) patterns in texts such that the pattern may match the text after all its characters (notes) are shifted by some value. The reason is that such an occurrence will sound like the pattern to a human, albeit in a different scale. In this context, efficient algorithms for several approximate matching functions were developed [21]. Recently, average-optimal algorithms for several variants of the problem have appeared [10, 11].

We note that transposition invariance becomes lighting invariance when we replace musical notes by gray levels of cells in an image. This is of course just a general statement. Not only human perception of light is not linear with the gray level value, but also lighting involves a non-linear transformation of gray levels. There exist, however, several well-known techniques to transform the gray level values to another scale so that perceptual changes due to lighting conditions become approximately linear in the transformed gray level [24]. Two examples of such techniques are histogram equalization and variance normalization. In this paper we disregard this aspect and assume that lighting introduces a constant shift in the gray level values.

The aim of this paper is to enrich the existing algorithms for rotation invariant template matching [14, 15] with the techniques developed for transposition invariance [21, 10, 11] so as to obtain rotation and lighting invariant template matching. It turns out that lighting invariance can be added at very little extra cost. The key technique exploited is *incremental distance computation*: We show that several lighting-invariant distances can be computed incrementally by taking the computation done for the previous rotation into account in the next rotation angle. This problem cannot be solved by straightforwardly combining techniques from previous work.

Let us now determine which are reasonable matching models for our case. In [14, 15], some of the models considered were useful only for binary images, a case where obviously we are not interested in in this paper. We will consider models that make sense for gray level images. We define three lighting-invariant distances: *Hamming* distance $d_{\mathrm{H}}^{\mathrm{t},\delta}$, which counts how many pattern and text cells differ by more than δ; *Maximum Absolute Differences* distance $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$, which is the maximum color difference between pattern and text cells when up to κ outliers are permitted; and *Sum of Absolute Differences* distance $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$, which is the sum of absolute color differences between pattern and text cells permitting up to κ outliers.

We consider two types of cell values. *General* values means that the cell contents are real numbers, while *discrete* values means that the cell contents belong to a range of integers of size σ. Any complexity achieved for general cell values is valid for discrete cell values as well. Table 1 shows the time complexities (under the word RAM computation model) of our algorithms for computing these distances for every possible rotation of a pattern centered at a fixed text position. We remark that a lower bound to this problem is $\Omega(m^3)$, and an algorithm whose time complexity matches this lower bound was given in [14, 15] without lighting invariance (see Section 3 for more on lower bounds). On the other hand, in the integer case it is trivial to obtain $O(\sigma m^3)$ time by trying out each possible transposition (i.e., difference among gray levels).

Distance	General	Discrete
$d_{\text{H}}^{\text{t},\delta}$	$m^3 \log m$	$m^3(\delta+1)$
$d_{\text{MAD}}^{\text{t},\kappa}$	$m^3(\kappa + (\log\log m)^2)$	$m^3(\kappa + \log\log\sigma)$
$d_{\text{SAD}}^{\text{t},\kappa}$	$m^3(\kappa + (\log\log m)^2)$	$m^3(\kappa + \log\log\sigma)$

Table 1: Worst-case time complexities ($O(\cdot)$ omitted) to compute the different distances defined. We give complexities for general and discrete cell values, under the word RAM model.

We also define three search problems, consisting in finding all the lighting-invariant rotated occurrences of P in T such that: there are at most κ cells of P differing by more than δ from their corresponding text cell (δ-matching); the sum of absolute differences between cells in P and T, except for κ outliers, does not exceed γ (γ-matching); and P matches both criteria at the same time, for a given transposition and set of outliers ((δ,γ)-matching).

Table 2 shows our worst-case and average-case search complexities (the latter are valid only for integer cell values independently and uniformly distributed over their σ possible contents). Without transposition invariance the worst cases are all $O(m^3n^2)$, which are optimal [14, 15] [1]. Again, it is trivial to obtain $O(\sigma m^3 n^2)$ on integer cell values, by simply trying out every possible transposition. Algorithms for δ-matching with $\delta = 0$ (but permitting κ outliers) and for γ-matching with $\kappa = 0$, without lighting invariance in both cases, are given in [15] (see also [11]). The respective average complexities are $O(n^2(\kappa+\log_\sigma m)/m^2)$ and $O(n^2(\gamma/\sigma+\log_{\frac{\sigma}{1+\gamma/m}} m)/m^2)$. Both are average-optimal [15, 11]. Thus our complexities are rather close to be optimal.

2 Definitions

Let $T = T[1..n, 1..n]$ and $P = P[1..m, 1..m]$ be arrays of unit squares, called *cells*, in the (x, y)-plane. Each cell has a value in an alphabet called Σ, sometimes called its gray level or its color. Two types of alphabets are of interest: *general alphabets* assume $\Sigma \subseteq \mathbb{R}$; while *discrete alphabets* assume finite $\Sigma \subset \mathbb{Z}$ and $\max(\Sigma) - \min(\Sigma) = \sigma$. The corners of the cell for $T[i, j]$ are $(i-1, j-1), (i, j-1), (i-1, j)$ and (i, j). The center of the cell for $T[i, j]$ is $(i - \frac{1}{2}, j - \frac{1}{2})$. The array of cells for pattern P is defined similarly. The center of the whole pattern P is the center of the cell in the middle of P. Precisely, assuming for simplicity that m is odd, the center of P is the center of cell $P[\frac{m+1}{2}, \frac{m+1}{2}]$.

Assume now that P has been moved on top of T using a rigid motion (translation and rotation), such that the center of P coincides exactly with the center of some cell of T (we call this the *center-to-center assumption*). The location of P with respect to T can be uniquely given as $((i, j), \theta)$ where (i, j) is the cell of T that matches the center of P, and θ is the angle between the x-axis of T

[1]Recently, an algorithm whose worst case *scanning time* is $O(m^2n^2)$ was obtained [2]. The algorithm is for exact matching only, and it is based on linearizing all the pattern rotations, and then relying on one-dimensional linear time dictionary matching algorithms. However, if all the occurrences at their angles must be reported, any algorithm still requires $\Omega(m^3n^2)$ time in the worst case.

Problem	Worst case (general)	Worst case (discrete)
δ-matching	$m^3n^2\log m$ $m^3n^2(\kappa + (\log\log m)^2)$	$m^3n^2(\delta+1)$ $m^3n^2(\kappa + \log\log\sigma)$
γ-matching	$m^3n^2(\kappa + (\log\log m)^2)$	$m^3n^2(\kappa + \log\log\sigma)$
(δ,γ)-matching		$m^3n^2((\kappa+1)\sqrt{\gamma} + \log\log\sigma)$

Problem	Average case (discrete)
δ-matching	$\frac{n^2}{m^2}(\kappa + (1+\frac{\kappa}{m})\log_{\frac{\sigma}{\delta+1}}((\delta+1)m))$, if $\delta < \frac{\sigma-2}{4}$ and $\kappa < \frac{m^2}{4}$
γ-matching	$\frac{n^2}{m^2}(\frac{\kappa+\gamma+\kappa\gamma/m}{\sigma} + (\kappa+1)\log_{\frac{\sigma}{1+\gamma/m}}(\gamma+m))$, if $\kappa \le m/(2\sqrt{2})$ and $\gamma/\kappa \le \sigma m/(2\sqrt{2}e)(1-O(1/\sigma))$, or $m/(2\sqrt{2}) \le \kappa = O(m^2/\log m)$ and $\gamma\kappa/m^2 \le \sigma m/(16\sqrt{2}e)(1-O(1/\sigma))$
γ-matching	$\frac{n^2}{m^2}(\kappa + (1+\frac{\gamma}{\sigma}+\frac{\kappa}{m})\log(\sigma m))$, if $\kappa \le m^2/8$ and $\gamma \le m\sigma/(8\sqrt{2})$
(δ,γ)-matching	Best of all the above

Table 2: Complexities for different search problems (conditions of applicability for average cases are simplified).

and the x-axis of P. The (approximate) occurrence between T and P at some location is defined by comparing the values of the cells of T and P that overlap. We will use the centers of the cells of T for selecting the comparison points. That is, for the pattern at location $((i,j),\theta)$, we look which cells of the pattern cover the centers of the cells of the text, and compare the corresponding values of those cells. Figure 1 illustrates.

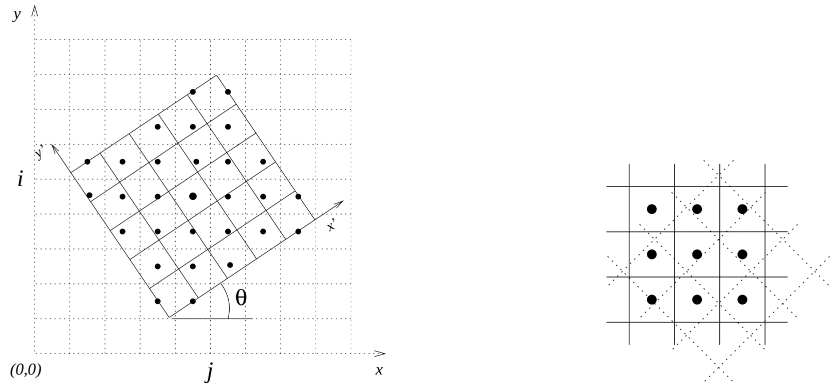


Figure 1: Each text cell is matched against the pattern cell that covers the center of the text cell. As a consequence, some pattern cells may have to match several text cells simultaneously, whereas some others may not have to match any text cell at all (right).

More precisely, we define a *matching function* M from the cells of T to the cells of P as follows. Assume that P is at location $((i,j),\theta)$. For each cell $T[r,s]$ of T whose center belongs to the area covered by P, let $P[r',s']$ be the cell of P such that the center of $T[r,s]$ belongs to the area covered by $P[r',s']$. Then $M([r,s]) = [r',s']$.

Now consider what happens to M when angle θ grows continuously, starting from $\theta = 0$. Function M changes only at the values of θ such that some cell center of T hits some cell boundary of P. It was shown [16] that this happens $O(m^3)$ times as P rotates full 2π radians (or within any fixed angle). A lower bound of $\Omega(m^3)$ was also proved [1]. Hence there are $\Theta(m^3)$ relevant orientations of P to be checked. The set of angles for $0 \le \theta \le \pi/2$ is

$$A = \{\beta, \pi/2 - \beta,\ \beta = \arcsin \frac{h+\frac{1}{2}}{\sqrt{i^2+j^2}} - \arcsin \frac{j}{\sqrt{i^2+j^2}};$$
$$i = 1,2,\ldots,\lfloor m/2 \rfloor; j = 0,1,\ldots,\lfloor m/2 \rfloor; h = 0,1,\ldots,\lfloor \sqrt{i^2+j^2} \rfloor\}.$$

By symmetry, the set of possible angles θ, $0 \le \theta < 2\pi$, is

$$\mathcal{A} = A \ \cup\ (A+\pi/2)\ \cup\ (A+\pi)\ \cup\ (A+3\pi/2)\ ,$$

where $A + c\pi$ is the set of angles in A with $c\pi$ added to each angle.

Furthermore, pattern P matches at location $((i,j),\theta)$ with lighting invariance if there is some integer transposition t such that $T[r,s] + t = P(M[r,s])$ for all $[r,s]$ covered by P.

Once the position and rotation $((i,j),\theta)$ of P in T define the matching function, we can compute different kinds of *distances* between the pattern and the text. Lighting-invariant versions of the distances choose the transposition minimizing the basic distance. The following distances are interesting for gray level images.

Hamming Distance (H): The number of times $T[r,s] + t \ne P[r',s']$ occurs, over all the covered cells of T, that is

$$\begin{aligned} d_{\mathrm{H}}(i,j,\theta,t) &= \sum_{r,s} \textbf{if } T[r,s]+t \ne P(M[r,s]) \textbf{ then } 1 \textbf{ else } 0 \\ d^{\mathrm{t}}_{\mathrm{H}}(i,j,\theta) &= \min_t d_{\mathrm{H}}(i,j,\theta,t) \end{aligned}$$

This can be extended to distance d^{δ}_{H} and its transposition-invariant version $d^{\mathrm{t},\delta}_{\mathrm{H}}$, where colors must differ by more than δ in order to be considered different, that is, $T[r,s]+t \notin [P(M[r,s])-\delta, P(M[r,s])+\delta]$.

Maximum Absolute Difference (MAD): The maximum value of $|T[r,s]+t-P[r',s']|$ over all the covered cells of T, that is,

$$\begin{aligned} d_{\mathrm{MAD}}(i,j,\theta,t) &= \max_{r,s} |T[r,s]+t-P(M[r,s])| \\ d^{\mathrm{t}}_{\mathrm{MAD}}(i,j,\theta) &= \min_t d_{\mathrm{MAD}}(i,j,\theta,t) \end{aligned}$$

This can be extended to distance $d^{\kappa}_{\mathrm{MAD}}$ and its transposition-invariant version $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$, so that κ pattern cells are freed from matching the text. Then the problem is to compute the MAD distance with the best choice of κ outliers that are not included in the maximum.

Sum of Absolute Differences (SAD): The sum of the $|T[r,s]+t-P[r',s']|$ values over all the covered cells of T, that is,

$$\begin{aligned} d_{\mathrm{SAD}}(i,j,\theta,t) &= \sum_{r,s} |T[r,s]+t-P(M[r,s])| \\ d^{\mathrm{t}}_{\mathrm{SAD}}(i,j,\theta) &= \min_t d_{\mathrm{SAD}}(i,j,\theta,t) \end{aligned}$$

Similarly, this distance can be extended to $d^{\kappa}_{\mathrm{SAD}}$ and its transposition-invariant version $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$, where κ cells can be removed from the summation.

Once the above distances are defined, we can define the following search problems:

δ-Matching: Report triples (i,j,θ) such that $d^{\mathrm{t}}_{\mathrm{MAD}}(i,j,\theta) \le \delta$. A tolerance κ can be permitted, so that we only require $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}(i,j,\theta) \le \delta$. Observe that this condition is the same as $d^{\mathrm{t},\delta}_{\mathrm{H}}(i,j,\theta) \le \kappa$.

γ-Matching: Report triples (i,j,θ) such that $d^{\mathrm{t}}_{\mathrm{SAD}}(i,j,\theta) \le \gamma$. Again, permitting tolerance κ means requiring $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}(i,j,\theta) \le \gamma$.

(δ,γ)-Matching: Report triples (i,j,θ) such that $d_{\mathrm{MAD}}(i,j,\theta,t) \le \delta$ and $d_{\mathrm{SAD}}(i,j,\theta,t) \le \gamma$ for some t. Tolerance κ can be handled similarly, but the κ excluded cells must be the same for both distances.

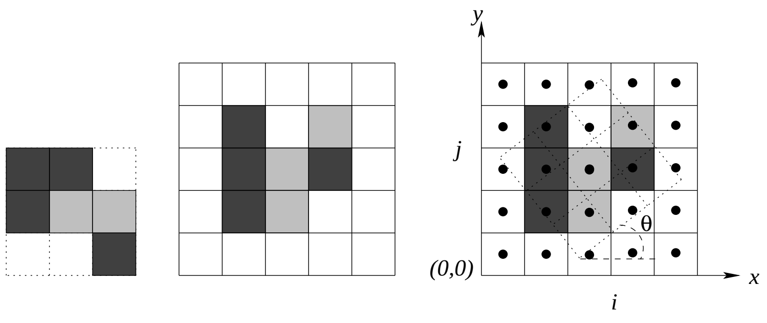
Figure 2 illustrates some cases.

3 Efficient Worst-Case Algorithms

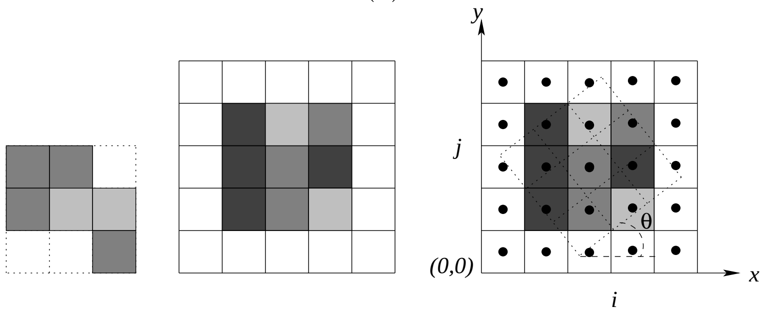
In [1] it was shown that the worst case lower bound for the problem of the two dimensional pattern matching allowing rotations is $\Omega(n^2m^3)$. A simple way to achieve this lower bound for any of the distances under consideration (without lighting invariance) is shown in [14, 15].

The idea is that we check each possible text center, one by one. So we have to pay $O(m^3)$ per text center to achieve the desired complexity. What we do is to compute the distance we want for each possible rotation, by reusing most of the work done for the previous rotation. Once the distances are computed, it is easy to report the triples (i,j,θ) where these values are smaller than the given thresholds (δ and/or γ). Only distances d_{H} (with $\delta = 0$) and d_{SAD} (with $\kappa = 0$) were considered in [14, 15].

Assume that, when computing the set of angles $\mathcal{A} = (\beta_1, \beta_2, \ldots)$, we also sort the angles so that $\beta_i < \beta_{i+1}$, and associate with each angle β_i the set $\mathcal{C}_i$ containing the corresponding cell centers that hit a cell boundary at β_i. This is done in a precomputation step that depends only on m, not on P or T. Hence we can evaluate the distance functions (such as d_{SAD}) incrementally for successive rotations of P. That is, assume that the distance has been evaluated for β_i, then to evaluate it for rotation β_{i+1} it suffices to re-evaluate the cells restricted to the set $\mathcal{C}_i$. This is repeated for each $\beta \in \mathcal{A}$. Therefore, the total number of cell (re)evaluations when P is centered at some fixed position in T, for all possible angles, is $\sum_i |\mathcal{C}_i|$. This is $O(m^3)$ because each fixed cell center of



(a)



(b)

Figure 2: Some examples of matches. In (a) we show a 3×3 pattern over three gray levels, a 5×5 text, and a rotated occurrence. Note that every text center covered by some pattern cell matches the color of the corresponding pattern cell, with the exception of the text cell in the third column and second row (counting from below). This text cell is light gray and is aligned with a white pattern cell. This alignment will be declared as an occurrence, for example, if we permit Hamming distance 1, as $d_{\mathrm{H}}(i, j, \theta, 0) = 1$. In (b) we illustrate matching with lighting invariance using four gray levels. The matching cells are the same as in (a) provided we shift all the gray levels of the text cells by $t = 1$ (assuming colors go from black $= 0$ to white $= 3$). That is, $d_{\mathrm{H}}(i, j, \theta, 1) = d^{\mathrm{t}}_{\mathrm{H}}(i, j, \theta) = 1$.

T, covered by P, can belong to at most $O(m)$ different $\mathcal{C}_i$ sets. To see this, note that when P is rotated the whole angle 2π, any cell of P traverses through $O(m)$ cells of T. It is easy to update distances d_{H} and d_{SAD} in constant time upon a cell reevaluation, thus the overall cost is $O(n^2m^3)$.

If we want to add lighting invariance to the above scheme, a naive approach is to run the algorithm for every possible transposition, for a total cost of $O(n^2m^3\sigma)$ on discrete alphabets. In case of a general alphabet there are $O(m^2)$ relevant transpositions at each rotation (that is, each pattern cell can be made to match its corresponding text cell). Hence the cost raises to $O(n^2m^5)$.

In order to do better, we must be able to compute the optimal transposition for the initial angle and then maintain it when some characters of the text change (because the pattern has been aligned over a different text cell). If we take $f(m)$ time to do this, then our lighting invariant algorithm becomes worst-case time $O(n^2m^3f(m))$. In the following we show how we can achieve this for each of the distances under consideration. Additionally, some of our results give relevant complexities for the case of no transpositions, for example for d_{MAD} distance.

This technique can be inserted into the filters that we present later in order to make them near optimal in the worst case. All our filtration algorithms are based on discarding most of the possible (i, j, θ) locations and leaving a few of them to be verified. If we avoid verifying a given text center more than once, then we can apply our verification technique and ensure that, overall, we do not pay more than $O(n^2 m^3 f(m))$.

3.1 Distance $d_{\mathbf{H}}^{\mathbf{t},\delta}$ and δ-Matching

In this section we show how to compute distance $d_{\mathrm{H}}^{\mathrm{t},\delta}$ between a pattern and a text center in time $O((\delta+1)m^3)$, as well as to perform δ-matching with tolerance κ in overall time $O((\delta+1)m^3n^2)$, on integer alphabets of size σ. On general alphabets, we show how to compute distances in $O(m^3 \log m)$ time, and how to perform δ-matching in $O(n^2m^3 \log m)$ time. Note that the search times are independent on κ.

As proved in [21], the optimal transposition for Hamming distance is obtained as follows. Each cell $P[r', s']$, aligned to $T[r, s]$, *votes* for a range of transpositions $[P[r', s'] - T[r, s] - \delta, P[r', s'] - T[r, s] + \delta]$, for which it would match. If a transposition receives v votes, then its Hamming distance is $m^2 - v$. Hence, the transposition that receives most votes is the one yielding distance $d_{\mathrm{H}}^{\mathrm{t},\delta}$. The problem is equivalent to the so-called *point of maximum overlap* in the literature. We are in particular interested in the dynamic version of the problem, and give different solutions for integer and general alphabets.

3.1.1 Integer Alphabet

The algorithm in [21], for one-dimensional transposition invariant string matching, obtains $O(\sigma + |P|)$ time on integer alphabet, by bucket-sorting the range extremes and then traversing them linearly so as to find the most voted transposition (a counter is incremented when a range starts and decremented when it finishes).

We will use a different method to find, in $O((\delta + 1)m^2)$ time, the optimal transposition for the first rotation angle. This method will enable us to recompute, in $O(\delta + 1)$ time, the optimal transposition once some text cell $T[r, s]$ changes its value (due to a small change in rotation angle). The net effect of such a change is that the range of transpositions given by the old cell value loses a vote and a new range gains a vote.

We use the fact that the alphabet is an integer range, so there are at most $2\sigma - 1$ possible transpositions. An array S_t of size $2\sigma-1$ tells the number of votes each transposition has. There are also $m^2 + 1$ counters L_i, $0 \le i \le m^2$, maintaining the number of transpositions that currently have i votes. Hence, when a range of transpositions loses/gains one vote, at most $2\delta + 1$ transpositions are moved to refer to the lower/upper counter (that is, if t loses/gains one vote and $S_t = i$, then S_t is decremented/incremented, L_i is decremented and L_{i-1}/L_{i+1} incremented). We need to keep control of which is the highest-numbered non-zero L_i counter, which is easily done in constant time per operation because transpositions move from one counter to the next/previous.

Arrays S and L are initialized in constant time [22, Section III.8.1], so that we assume that all uninitialized S_t values are zero, and all L_i values are also zero except for $L_0 = 2\sigma - 1$. Then, we spend $O((\delta + 1)m^2)$ time to process the votes of all the cells in angle $\theta = 0$, and then process $O(\delta+1)$ changes for each cell that changes as we rotate P. Overall, when we consider all the $O(m^3)$ cell changes, the total complexity is $O((\delta + 1)m^3)$.

Thus the overall complexity to compute distance $d_{\mathrm{H}}^{\mathrm{t},\delta}$ between a pattern and a text center, considering all possible rotations and transpositions, is $O((\delta+1)m^3)$. δ-Matching can be done simply by computing $d_{\mathrm{H}}^{\mathrm{t},\delta}$ distances at each text center and reporting triples (i,j,θ) where $d_{\mathrm{H}}^{\mathrm{t},\delta}(i,j,\theta) \leq \kappa$. The overall search time is thus $O((\delta+1)n^2m^3)$.

3.1.2 General Alphabet

Our problem is a slight variant of the dynamic point of maximum overlap of a set of intervals. Given a multiset S of one-dimensional closed ranges, we are interested in obtaining a point p that is included in most ranges, that is $\mathrm{maxvote}(S) = \max_p |\{[\ell,r] \in S,\ \ell \leq p \leq r\}|$. Each update consists of a new range that is added to or an old range that is deleted from S, and we must return $\mathrm{maxvote}(S)$ after each update.

Given an algorithm for this problem, we can easily compute $d_{\mathrm{H}}^{\mathrm{t},\delta}$ from one rotation angle to the next. Our multiset is $S = \{[P(M[r,s])-T[r,s]-\delta, P(M[r,s])-T[r,s]+\delta]\}$. From one rotation angle to the next, some $M[r,s]$ changes its value and thus we have to delete the old range and add the new one, after which $\mathrm{maxvote}(S)$ is requested in order to compute distance $d_{\mathrm{H}}^{\mathrm{t},\delta} = m^2 - \mathrm{maxvote}(S)$ for the new angle.

The problem of maintaining the point of maximum overlap upon interval insertions and deletions can be found, for example, in [6, Problem 14-1]. We present here a solution that differs from the one suggested in there and thus can be of independent interest, yet it achieves the same $O(\log|S|)$ time per operation. This immediately gives an $O(m^3 \log m)$ time algorithm for computing $d_{\mathrm{H}}^{\mathrm{t},\delta}$ between a pattern and a text center, considering all possible rotations and transpositions, as well as an $O(\log(m)m^3n^2)$ worst-case time solution for δ-matching with tolerance κ (the complexity is independent of κ).

First, notice that the point that gives $\mathrm{maxvote}(S)$ can always be chosen among the endpoints of ranges in S. We store each endpoint e in a balanced binary search tree with key e. Let us denote the leaf whose key is e simply by (leaf) e. With each endpoint e we associate a value $\mathrm{vote}(e)$ (stored in leaf e) that gives the number $|\{[\ell,r],\ \ell \leq e \leq r, [\ell,r] \in S\}|$, where the set is considered as a multiset (same ranges can have multiple occurrences). In each internal node v, value $\mathrm{maxvote}(v)$ gives the maximum of the $\mathrm{vote}(e)$ values of the leaves e in its subtree. After all the endpoints e are added and the values $\mathrm{vote}(e)$ in the leaves and values $\mathrm{maxvote}(v)$ in the internal nodes are computed, the static case is solved by taking the value $\mathrm{maxvote}(\mathit{root}) = \mathrm{maxvote}(S)$ in the root node of the tree.

A straightforward way of generalizing the above approach to the dynamic case would be to recompute all values $\mathrm{vote}(e)$ that are affected by the insertion/deletion of a range. This would, however, take $O(|S|)$ time in the worst case. To get a faster algorithm, we only store the changes of the votes in the roots of certain subtrees so that $\mathrm{vote}(e)$ for any leaf e can be computed by summing up the changes from the root to the leaf e.

For now on, we refer to $\mathrm{vote}(e)$ and $\mathrm{maxvote}(v)$ as virtual values, and implement them with counters $\mathrm{diff}(v)$ and values $\mathrm{maxdiff}(v)$. Counters $\mathrm{diff}(v)$ are defined implicitly so that for all leaves of the tree it holds

$$\mathrm{vote}(e) \quad = \sum_{v \in \mathrm{path}(\mathit{root},e)} \mathrm{diff}(v), \qquad (1)$$

where path$(root, e)$ is the set of nodes in the path from the root to a leaf e (including e). We note that there are several possible ways to choose diff(v) values so that they satisfy the definition. Values maxdiff(v) are defined as the maximum sum of differences across a path from a child of v to a leaf. It is easy to see that

$$\text{maxdiff}(v) \quad = \quad \max(\text{maxdiff}(v.left) + \text{diff}(v.left), \text{maxdiff}(v.right) + \text{diff}(v.right)), \qquad (2)$$

where $v.left$ and $v.right$ are the left and right child of v, respectively. In particular, maxdiff$(e) = 0$ for any leaf node e. One also easily notices that

$$\text{maxvote}(v) \quad = \quad \text{maxdiff}(v) + \sum_{v' \in \text{path}(root, v)} \text{diff}(v'),$$

which also gives as a special case Equation (1) once we notice that maxvote$(e) =$ vote(e) for each leaf node e.

Our goal is to maintain diff() and maxdiff() values correctly during insertions and deletions. We have three different subproblems to consider: (*i*) How to compute the value diff(e) for a new endpoint of a range, (*ii*) how to update the values of diff() and maxdiff() when a range is inserted/deleted, and (*iii*) how to update the values during rotations to rebalance the tree. An insertion involves subproblems (*i–iii*), while a deletion involves only (*ii*) and (*iii*).

Problem (*i*) is handled by storing in each leaf an additional counter end(e) that gives the number of ranges whose rightmost endpoint is e. Assume that this value is computed for all existing leaves. When we insert a new endpoint e, we either find a leaf labeled e or otherwise there is a leaf e' after which e is inserted. In the first case vote(e) remains the same and in the latter case vote$(e) =$ vote$(e') -$ end(e'), because e is included in the same ranges as e' except those that end at e'. Notice also that vote$(e) = 0$ in the degenerate case when e is the leftmost leaf. To make vote$(e) = \sum_{v' \in \text{path}(root,e)} \text{diff}(v')$, we define diff$(e) =$ vote$(e) - \sum_{v' \in \text{path}(root,v)} \text{diff}(v')$, where v is the parent of e. Once the maxdiff() values are updated in the path from e to the root, we have solved the subproblem in $O(\log |S|)$ time. Note that the $+1$ vote induced by the new range whose endpoint is e has not yet been considered, as it is handled as subproblem (*ii*).

Let us then consider subproblem (*ii*). Recall the one-dimensional range search on a balanced binary search tree (see, e.g., [8, Section 5.1]). We use the fact that one can find in $O(\log |S|)$ time the minimal set of nodes F such that the range $[\ell, r]$ of S is *partitioned* by F: The subtrees starting at nodes of F contain all the points in $[\ell, r] \cap S$ and only those. It follows that when inserting (deleting) a range $[\ell, r]$, we can set diff$(v) =$ diff$(v) + 1$ (diff$(v) =$ diff$(v) - 1$) at each $v \in F$. This is because all the values vote(e) in these subtrees change by ± 1 (including leaves ℓ and r). Note that some diff(v) values may go below zero, but this does not affect correctness.

To keep also the maxdiff() values correctly updated, it is enough to recompute the values in the nodes in the paths from each $v \in F$ to the root using Equation (2); other values are not affected by the insertion/deletion of the range $[\ell, r]$. The overall number of nodes that need updating is $O(\log |S|)$. To see this, note that the nodes in F are either left children of a unique rightwards path, or right children of a unique leftwards path. Therefore the set of ancestors of F is of size $O(\log |S|)$.

Finally, let us consider subproblem (*iii*). Counters diff(v) are affected by tree rotations, but in case a tree rotation involving e.g. subtrees $v.left$, $v.right.left$ and $v.right.right$ takes place, values diff(v) and diff($v.right$) can be "pushed" down to the roots of the affected subtrees, and hence they become zero. Then the tree rotation can be carried out without further considerations. Note that here we are taking advantage of the fact that the diff(v) values need not be unique as long as we maintain their path sums. Subtree maxima are easily maintained through tree rotations.

Hence, each insertion/deletion takes $O(\log |S|)$ time, and maxvote(S) = maxdiff($root$) + diff($root$) is readily available in the root node.

3.2 Distance $d^{\mathbf{t},\kappa}_{\mathbf{MAD}}$ and δ-Matching

In this section we show how to compute distance $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ between a pattern and a text center in time $O((\kappa + (\log\log m)^2)m^3)$ on general alphabets. On integer alphabets we can also obtain $O((\kappa+\log\log\sigma)m^3)$. These methods yield a δ-matching algorithm alternative to that of Section 3.1, needing overall time $O((\kappa + (\log\log m)^2)n^2m^3)$ on general alphabets and $O((\kappa + \log\log\sigma)n^2m^3)$ on integer alphabets. This time the complexities are sensitive to κ.

Let us start with $\kappa = 0$. As proved in [21], the optimal transposition for distance $d^{\mathrm{t}}_{\mathrm{MAD}}$ is obtained as follows. Each cell $P[r', s']$, aligned to $T[r, s]$, votes for transposition $t = P[r', s'] - T[r, s]$. Then, the optimal transposition is the average between the minimum and maximum votes. The corresponding $d^{\mathrm{t}}_{\mathrm{MAD}}$ distance is the difference of maximum minus minimum, divided by two. Hence an $O(|P|)$ algorithm is immediate.

In our case, we need $O(m^2)$ time to obtain the optimal transposition for the first angle, $\theta = 0$. Then, in order to address changes of text characters (because, due to angle changes, the pattern cell was aligned to a different text cell), we must be able to maintain the minimum and maximum votes. Every time a text character changes, a vote disappears and a new vote appears. This can be solved with min- and max-priority queues supporting insertion, deletion, and min/max operations.

In the case of integer alphabets, the transpositions belong to a universe of size $O(\sigma)$. Thus van Emde Boas priority queues [26, 25] permit implementing each operation in time $O(\log\log\sigma)$, using $O(\sigma)$ space. On general alphabets, it is possible to obtain $O((\log\log m)^2)$ time per operation on the word RAM model [4] [2]. Hence $d^{\mathrm{t}}_{\mathrm{MAD}}$ distance between a pattern and a text center can be computed in $O(m^3 \log\log\sigma)$ time on integer alphabets or $O(m^3(\log\log m)^2)$ time on general alphabets, for all possible rotations and transpositions.

In order to account for up to κ outliers, it was shown [21] that it is optimal to choose them from the pairs that vote for maximum or minimum transpositions. That is, if all the votes are sorted into a list $t_1 \ldots t_{m^2}$, then distance $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ is the minimum among distances $d^{\mathrm{t}}_{\mathrm{MAD}}$ computed in sets $t_1 \ldots t_{m^2-\kappa}$, $t_2 \ldots t_{m^2-\kappa+1}$, and so on until $t_{\kappa+1} \ldots t_{m^2}$. Moreover, the optimum transposition of the i-th value of this list is simply the average of maximum and minimum, that is, $(t_{m^2-\kappa-1+i} + t_i)/2$.

So our algorithm for $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ is as follows. We maintain plain sorted arrays S and L with the $\kappa + 1$ smallest and largest votes, respectively. All the other votes not in $S \cup L$ are maintained in a priority queue Q. Upon an insertion, we determine in constant time which of the three cases apply: (i) the element must be inserted into S and the largest element of S must be moved to Q, (ii) the element must be inserted into L and the smallest element of L must be moved to Q, (iii)

[2] This solution is in AC^0. If we wish to stick to a weaker computation model, we can still solve the problem using a balanced search tree in $O(\log(m^2)) = O(\log m)$ time. Note in passing that this weaker computation model is assumed for the results in [21]. In particular, the $O(m \log m)$ complexity for $d^{\mathrm{t},\delta}_{\mathrm{H}}$ and $O(\kappa \log \kappa)$ terms for $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ in [21] correspond to sorting, and they become respectively $O(m \log\log m)$ and $O(\kappa \log\log \kappa)$ on the stronger model [3].

the element must be inserted into Q. Similarly, upon a deletion we might have to delete from S or L (in which case the minimum or maximum of Q must be moved to S or L), or we might have just to delete the element from Q. In any case the cost of the insertion/deletion is $O(\kappa + \log\log\sigma)$ on integer alphabets and $O(\kappa + (\log\log m)^2)$ on general alphabets.

After each cell change (deletion plus insertion), we retraverse the $\kappa + 1$ pairs in S and L and recompute the minimum among the $t_{m^2-\kappa-1+i} - t_i$ differences. Overall, the process takes $O((\kappa + \log\log\sigma)m^3)$ on integer alphabets and $O((\kappa + (\log\log m)^2)m^3)$ on general alphabets. Note that on integer alphabets the result is interesting only if $\kappa < \sigma$, as otherwise a trivial algorithm obtains $O(\sigma m^3)$ time, by just trying out each transposition.

The δ-matching problem can be alternatively solved by computing this distance for every text cell, and reporting triples (i, j, θ) where $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}(i, j, \theta) \leq \delta$. This gives an alternative $O((\kappa + \log\log\sigma)n^2m^3)$ or $O((\kappa + (\log\log m)^2)n^2m^3)$ time algorithm to solve the δ-matching problem.

Note, on the other hand, that a similar algorithm solves the problem of computing $d^{\kappa}_{\mathrm{MAD}}$ and doing δ-matching, without lighting invariance, with the same complexity. Instead of votes, we maintain all the $|P[r', s'] - T[r, s]|$ values in a max-priority queue and find the smallest maximum across all rotations. Outliers are handled similarly by using an array L of largest differences.

3.3 Distance $d^{\mathbf{t},\kappa}_{\mathbf{SAD}}$ and γ-Matching

In this section we show how to compute distance $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ between a pattern and a text center within the same time complexities obtained for $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ in Section 3.2. This in turn yields a γ-matching algorithm with the same complexity of the δ-matching algorithm of Section 3.2.

Let us first consider case $\kappa = 0$. This corresponds to the SAD model of [21], where it was shown that, if we collect votes $P[r', s'] - T[r, s]$, then the median vote (either one if $|P|$ is even) is the transposition that yields distance $d^{\mathrm{t}}_{\mathrm{SAD}}$. The actual distance can be obtained by using the formula for d_{SAD}. Hence an $O(|P|)$ time algorithm was immediate.

In this case we have to pay $O(m^2)$ to compute the distance for the first rotation ($\theta = 0$), and then have to maintain the median transposition and current distance when some text cell changes its value due to a small rotation.

We maintain a max-priority queue S and a min-priority queue L. The first contains the smallest $\lceil m^2/2 \rceil$ votes and the second the largest $\lfloor m^2/2 \rfloor$ votes. Then the median vote is always the maximum element in S.

Each time a vote changes because a pattern cell aligns to a new text cell, we must remove the old vote and insert the new one. In either case, we determine which priority queue the insertion and deletion belong to. If they occur at different halves of the set of votes (that is, one is larger and the other is smaller than the median), then we must transfer one element from S to L or vice versa to maintain the invariant on their sizes. This requires a constant number of priority queue operations.

The distance value $d^{\mathrm{t}}_{\mathrm{SAD}}$ itself change upon two events. One event is that any of the votes changes its value. Given a fixed transposition, it is trivial to remove the appropriate summand and introduce a new one in the formula for d_{SAD} (Section 2). The other event is that the median position changes from a transposition in the sorted list to the next or previous. It was shown in [21] how to modify in constant time distance $d^{\mathrm{t}}_{\mathrm{SAD}}$ in this case. The idea is very simple: If we move from transposition t_j to t_{j+1}, then all the j smallest $|P[r', s'] - T[r, s] - t_j|$ summands of d_{SAD} increase their value by $t_{j+1} - t_j$ (as they become $|P[r', s'] - T[r, s] - t_{j+1}|$), while the $m^2 - j$ largest summands decrease by $t_{j+1} - t_j$. Hence distance d_{SAD} at the new transposition is the value at the old transposition plus $(2j - m^2)(t_{j+1} - t_j)$. Thus the distance can be updated in constant time.

Hence, we can traverse all the rotations in time $O(m^3 \log\log \sigma)$ on integer alphabets and $O(m^3 (\log\log m)^2)$ on general alphabets.

If we want to compute distance $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$, we have again that the optimal values to free from matching are those voting for minimum or maximum transpositions. If we remove those values, then the median lies at positions between $\lceil m^2/2 \rceil - \lfloor \kappa/2 \rfloor$ and $\lfloor m^2/2 \rfloor + \lceil \kappa/2 \rceil$ in the sorted list of votes.

We add a new plain array M holding the $\kappa + 1$ intermediate votes $t_{\lceil m^2/2 \rceil - \lfloor \kappa/2 \rfloor} \ldots t_{\lfloor m^2/2 \rfloor + \lceil \kappa/2 \rceil}$. The remaining smallest and largest values are maintained in priority queues S and L, respectively. As in Section 3.2, it is easy to perform the insertions/deletions in the appropriate set S, M, or L, and move elements among them to maintain the size invariants.

We need now to maintain all the $\kappa + 1$ possible median values. Those can be updated one by one in constant time each, and we can choose the minimum distance among the $\kappa + 1$ options. This gives us an $O(m^3(\kappa + \log\log \sigma))$ time algorithm to compute $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ on integer alphabets, and $O(m^3(\kappa + (\log\log m)^2))$ on general alphabets. In addition, this gives us $O((\kappa + \log\log \sigma) m^3 n^2)$ and $O((\kappa + (\log\log m)^2) m^3 n^2)$ time algorithms for γ-matching. It is a matter of computing $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ at each text position and reporting triples (i, j, θ) such that $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}(i, j, \theta) \leq \gamma$.

3.4 (δ, γ)-Matching with Tolerance κ

In this section we show how to perform (δ, γ)-matching with tolerance κ in time $O((\kappa + 1)\sqrt{\gamma} + \log\log \sigma) n^2 m^3)$, on integer alphabets. We have no result for general alphabets.

There are two reasons why solving this problem is not a matter of computing $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ at each text position and reporting triples (i, j, θ) where both conditions $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}(i, j, \theta) \leq \delta$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}(i, j, \theta) \leq \gamma$ hold. One is that the transposition achieving this must be the same, and the other is that the κ outliers must be the same.

Let us first consider the case $\kappa = 0$. A simple (δ, γ)-matching algorithm is as follows. We run the δ-matching algorithm based on $d^{\mathrm{t}}_{\mathrm{MAD}}$ distance, and the γ-matching algorithm based in $d^{\mathrm{t}}_{\mathrm{SAD}}$ distance at the same time. Every time we find a triple (i, j, θ) that meets both criteria, we compute the range of transpositions t such that $d_{\mathrm{MAD}}(i, j, \theta, t) \leq \delta$. This is very simple: Say that $d^{\mathrm{t}}_{\mathrm{MAD}}(i, j, \theta) \leq \delta$, which is obtained at the optimal transposition t^{MAD}. Then, $d_{\mathrm{MAD}}(i, j, \theta, t) \leq \delta$ for $t \in [t^{\mathrm{MAD}}_1, t^{\mathrm{MAD}}_2] = [t^{\mathrm{MAD}} - (\delta - d^{\mathrm{t}}_{\mathrm{MAD}}(i, j, \theta)), t^{\mathrm{MAD}} + (\delta - d^{\mathrm{t}}_{\mathrm{MAD}}(i, j, \theta))]$.

The problem is now to determine whether $d_{\mathrm{SAD}}(i, j, \theta, t) \leq \gamma$ for some t in the above range. As a function of t, $d_{\mathrm{SAD}}(i, j, \theta, t)$ has a single minimum at its optimum transposition t^{SAD} (which does not have to be the same t^{MAD}). Hence, we have three choices: (*i*) $t^{\mathrm{MAD}}_1 \leq t^{\mathrm{SAD}} \leq t^{\mathrm{MAD}}_2$, in which case the occurrence can be reported; (*ii*) $t^{\mathrm{SAD}} < t^{\mathrm{MAD}}_1$, in which case we report the occurrence only if $d_{\mathrm{SAD}}(i, j, \theta, t^{\mathrm{MAD}}_1) \leq \gamma$; (*iii*) $t^{\mathrm{SAD}} > t^{\mathrm{MAD}}_2$, in which case we report the occurrence only if $d_{\mathrm{SAD}}(i, j, \theta, t^{\mathrm{MAD}}_2) \leq \gamma$.

As in the worst case we may have to check $O(m^3 n^2)$ times for a (δ, γ)-match, and computing $d_{\mathrm{SAD}}(i, j, \theta, t)$ takes $O(m^2)$ time, we could pay as much as $O(m^5 n^2)$, which is as bad as the naive approach. However, on integer alphabet, we can do better. As we can recompute in constant time d_{SAD} from one transposition to the next (as explained in Section 3.3), we can move stepwise from t^{SAD} to t_1^{MAD} or t_2^{MAD}. Moreover, as we move away from t^{SAD}, distance d_{SAD} increases and it quickly exceeds γ. As we move j votes away from the median, say from t_j to t_{j+1}, we have j summands contributing each $t_{j+1} - t_j \geq 1$ to d_{SAD}, so after we move j times d_{SAD} has increased by $\Omega(j^2)$ (this assumes that the alphabet is integer and that we pack equal votes so as to process them in one shot). Hence we cannot work more than $O(\sqrt{\gamma})$ before having d_{SAD} out of range. Overall, the search time is $O((\sqrt{\gamma} + \log\log\sigma)n^2m^3)$.

The situation is more complex if we permit κ outliers. Fortunately, both in $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ and $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$ it turns out that the relevant outliers are those yielding the κ minimum or maximum votes, so the search space is small. That is, even when the selection of outliers that produces distance $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ is not the same producing distance $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$, it holds that if there is a selection that produces a $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ distance of at most δ and a $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$ distance of at most γ, then the same is achieved by a selection where only those producing minimum or maximum votes can be chosen. This is easily seen because distances $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ and $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$ can only increase if we replace the votes in their initial selection by excluded minimum or maximum votes.

Now we compute $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ and $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$ distances and consider every triple (i, j, θ) where both matching criteria are met. There are only $\kappa+1$ relevant selections of outliers (that is, choosing κ' smallest and κ'' largest votes such that $\kappa' + \kappa'' = \kappa$). For each such selection we have $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ and $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$ distances already computed. Hence we have to run the above verification algorithm for each triple (i, j, θ) and each of the $\kappa + 1$ selections of outliers. This gives a worst-case search algorithm of complexity $O(((\kappa + 1)\sqrt{\gamma} + \log\log\sigma)n^2m^3)$. We remark that this works only for integer alphabets and that it is interesting only when $\kappa < \sigma$.

4 Features

As shown in [16, 14, 15], any match of a pattern P in a text T allowing arbitrary rotations must contain some so-called "features", that is, one-dimensional strings obtained by reading a line of the pattern in some angle. These features are used to build a filter for finding the position and orientation of P in T. Figure 3 shows features of different lengths taken at different positions. In our algorithms we will take all features of the same length.

The length of a particular feature is denoted by u, and the feature for angle θ and row q is denoted by $F^q(\theta)$. Assume for simplicity that u is odd. To read a feature $F^q(\theta)$ from P, let P be on top of T, on location $((i, j), \theta)$. Consider cells $T[i - \frac{m+1}{2} + q, j - \frac{u-1}{2}], \ldots, T[i - \frac{m+1}{2} + q, j + \frac{u-1}{2}]$. Denote them as $t_1^q, t_2^q, \ldots, t_u^q$. Let c_i^q be the value of the cell of P that covers the center of t_i^q. The feature of P with angle θ and row q is the string $F^q(\theta) = c_1^q c_2^q \cdots c_u^q$. Note that this value depends only on q, θ and P, not on T.

The sets of angles for the features are obtained the same way as the set of angles for the whole pattern P. Note that the set of angles $\mathcal{B}^q$ for the feature set F^q is subset of $\mathcal{A}$, that is $\mathcal{B}^q \subset \mathcal{A}$ for any q. The size of $\mathcal{B}$ varies from $O(u^2)$ (the features crossing the center of P) to $O(um)$ (the features at distance $\Theta(m)$ from the center of P). In other words, the matching function M can change as long as $F^q(\theta)$ does not change.

More precisely, assume that $\mathcal{B}^q = (\gamma_1, \ldots, \gamma_K)$, and that $\gamma_i < \gamma_{i+1}$. Therefore, feature $F^q(\gamma_i) = F^q(\theta)$ can be read using any θ such that $\gamma_i \leq \theta < \gamma_{i+1}$. If there is an occurrence of $F^q(\theta)$, then

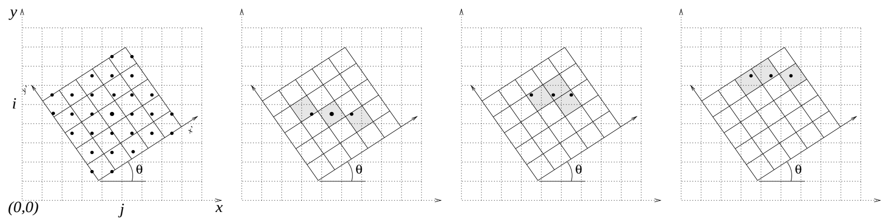


Figure 3: Some features read from P at angle θ. We show $F^0(\alpha)$, $F^1(\alpha)$, and $F^2(\alpha)$.

P may occur with any angle $\beta \in \mathcal{A}$ such that $\gamma_i \leq \beta < \gamma_{i+1}$. We say that those angles β are *compatible* with θ, that is, they belong to the same range $[\gamma_i, \gamma_{i+1})$.

The idea of using features is as follows. Assume we read from P a range of features $F^i(\theta)$, $\frac{m-r}{2} \leq i \leq \frac{m+r}{2}$ for some odd r. Then, if we scan one text row out of r, every occurrence of P at an angle compatible with θ will contain some feature $F^i(\theta)$ within some scanned row. Moreover, if we scan one text row out of $\lfloor r/j \rfloor$, then every occurrence of P at an angle compatible with θ will contain at least j features $F^i(\theta)$ within some scanned row [3]. Therefore, a multipattern search for the features in the selected text rows will spot all the possible occurrences of P in T.

These results can be extended to account for γ tolerance and κ outliers in the occurrences. The following lemmas are a generalization of other well-known results in approximate string matching [23].

Lemma 1 Assume we read from P a range of features $F^i(\theta)$, $\frac{m-r}{2} \leq i \leq \frac{m+r}{2}$ for some odd r. Then, if we scan one text row out of $\lfloor r/j \rfloor$, every occurrence of P at an angle compatible with θ with up to κ non-matching positions will contain at least p features $F^i(\theta)$ matching with at most $\lfloor \frac{\kappa}{j-p+1} \rfloor$ non-matching positions each, within some scanned row. This holds for any $1 \leq p \leq j$.

Proof. Assume otherwise. Consider a particular occurrence of P. As we scan one text row out of $\lfloor r/j \rfloor$, there are j features of P that appear in the scanned rows, corresponding to this occurrence. If the lemma does not hold, then there are at least $j-p+1$ features requiring at least $1 + \lfloor \frac{\kappa}{j-p+1} \rfloor > \frac{\kappa}{j-p+1}$ mismatches to occur. Therefore, just matching those features requires strictly more than κ mismatches in total, and therefore the whole P cannot match with just κ mismatches. $\square$

We note that Lemma 1 holds verbatim if we consider γ-tolerance instead of κ outliers. In addition, δ-tolerance and transposition invariance do not affect at all its correctness.

Lemma 1 is the key to the algorithms of the next section, where we rely on existing one-dimensional string matching algorithms. Let us first review the non-transposition invariant algorithms. In [12] it is shown how to perform approximate searching (under edit distance) of r patterns of length u in a text of length n, with distance at most k, in optimal average time $O(n(k+\log_\sigma(ru))/u)$, for $k/u \leq 1/2 - O(1/\sqrt{\sigma})$. In [15] the same technique is applied to Hamming distance, obtaining the same (optimal) complexity for $k/u \leq 1/2 - O(1/\sigma)$. In [10, 11], δ-matching is considered, as well as δ-matching combined with edit, Hamming, and other distances. The resulting average complexity is $O(n(k + \log_{\frac{\sigma}{\delta+1}}(ru))/u)$, with the same limits as before on k/u, and

[3]There is no guarantee that those features will be different if they repeat in P.

the constraint $2\delta + 1 < \sigma$. This result is shown to be average-optimal. On the other hand, γ-matching is considered in [15], where they obtain $O(n(\gamma/\sigma + \log_{\frac{\sigma}{1+\gamma/m}}(ru))/u)$ average time[4] for $\gamma/m \leq \sigma/(4e) - O(1)$. In [10, 11] they show that it is difficult to combine this γ-matching algorithm with k outliers.

Several transposition-invariant versions of the above algorithms are given in [10, 11]. They obtain, with the same restrictions as above on δ, γ and k, $O(n(k + \log_{\frac{\sigma}{\delta+1}}(ru(\delta+1)))/u)$ for δ-matching with outliers and $O(n(\gamma/\sigma + \log_{\frac{\sigma}{1+\delta/m}}(r(\gamma+u)))/u)$ for γ-matching. On the other hand, the approximate search algorithms with edit, Hamming, and other similar distances, stay with their optimal average complexity $O(n(k + \log_\sigma(ru))/u)$ when transpositions are allowed.

5 Efficient Average-Case Time Algorithms

Following [15], we choose features of length u from r pattern rows around the center, at all relevant rotation angles θ, and search for them all using different multipattern one-dimensional search algorithms permitting transpositions [10, 11]. To simplify the presentation we assume from the beginning $u = r = m/\sqrt{2}$, which are in fact the optimal values[5]. The number of relevant rotations is $O(ru\max(r,u)) = O(m^3)$ [14, 15].

The results of this section are valid only for integer alphabets where text and pattern cell values are independently and uniformly chosen over σ different values. We remind that, as the techniques consist essentially on leaving a few (i,j) text centers to check, we can maintain the worst cases of all the algorithms of this section within the bounds obtained in Section 3. It is a matter of running the worst-case-oriented distance computation algorithms only for the text centers (i,j) we could not discard, taking care of not verifying the same text center twice. As there are at most n^2 text centers to verify, the worst-case complexities follow.

5.1 δ-Matching with Tolerance κ

In this section we show how to perform transposition-invariant δ-matching with tolerance κ on integer alphabets of size σ in average time $O(n^2(\kappa + (1+\kappa/m)\log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m^2)$, whenever (roughly) $4\delta + 2 < \sigma$ and $\kappa < m^2/4$. By following the same procedure without transposition invariance, we obtain $O(n^2(\kappa + \log_{\frac{\sigma}{\delta+1}} m)/m^2)$, which is shown to be average-optimal.

We follow Lemma 1. We extract $r = O(m)$ features from P, at all the possible $O(m^2)$ rotations for each, totalizing $O(m^3)$ one-dimensional strings to search for. Then we scan one text row out of r with the one-dimensional transposition invariant algorithm that permits multiple patterns, δ-matching, and κ mismatches. According to Lemma 1 (with $j = p = 1$), every occurrence of P will be spotted by the occurrence of a feature in our one-dimensional search. Thus, it is enough to check for a complete occurrence of P in T only upon finding the occurrence of a feature. The matching feature and its position indicates the text center cell that must be considered, as well as the range of angles $[\gamma_i, \gamma_{i+1}) \cap \mathcal{A}$ to try.

[4]They give the base of the logarithm in the form $1/x$ and then quickly switch to the worst case $x = \Theta(1)$. We present here a more refined version.

[5]Actually these are the maximum possible values that guarantee that we can take r features of length u at any possible rotation.

Note, however, that $0 \le \kappa \le m^2$, and thus searching for a feature of length u permitting κ outliers might be too permissive (in particular the feature will match everywhere if $\kappa \ge u$). In this case, we scan one text row out of $\lfloor r/j \rfloor$, for some j to be determined soon. Now Lemma 1 guarantees that at least one feature ($p = 1$) will appear with at most $\lfloor \kappa/j \rfloor$ mismatches. Thus we have to use the one-dimensional algorithm with tolerance $\lfloor \kappa/j \rfloor$, which gives us the possibility of adjusting j so that the tolerance is low enough.

To analyze this algorithm we have to consider both the scanning and the verification cost. Let us start with the latter. Every time a feature matches we have to verify the possible occurrence of the complete pattern. We start by analyzing the probability of matching.

Let us consider a given feature F of length u. To upper bound the number of strings that δ-match F with tolerance k, we note that there are $\binom{u}{k}$ ways to choose the nonmatching positions, and then σ^k ways to choose the characters at those nonmatching positions. The other $u - k$ positions must δ-match the feature and thus there are $(2\delta + 1)^{u-k}$ ways to choose them. If we consider in addition transposition invariance, each of those strings matches at most $2\sigma - 1$ other strings, for a grand total of at most $(2\delta + 1)^{u-k}\binom{u}{k}2\sigma^{k+1}$. As each of those strings match with probability $1/\sigma^u$, the probability of each feature matching at a given text position is at most

$$\frac{(2\delta+1)^{u-k}\binom{u}{k}2\sigma}{\sigma^{u-k}} \quad = \quad O\left(\left(\frac{2\delta+1}{\alpha^{\frac{\alpha}{1-\alpha}}(1-\alpha)\sigma}\right)^{m(1-\alpha)/\sqrt{2}} \sigma\right),$$

where we have defined $\alpha = k/u$ ($0 \le \alpha < 1$) and used Stirling's approximation to convert $\binom{u}{k} = \left(\frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}}\right)^u \Theta(1)$. For α bounded away from 1, the probability has the form $O(a^m\sigma)$, and we are interested in determining the condition on α to ensure $a^m\sigma = O(c^m)$ for some constant $c < 1$, that is, $a \le c\,\sigma^{-1/m}$. We simplify the formula by noting that $\alpha^{\frac{\alpha}{1-\alpha}} \ge 1/2$ if $0 \le \alpha \le 1/2$ (the method is not used beyond this limit), and then it is easy to obtain condition $\alpha \le 1 - \frac{2(2\delta+1)}{c\sigma^{1-1/(\sqrt{2}m)}} = 1 - O((\delta+1)/\sigma)$. Note that it is necessary that $4\delta + 2 < \sigma$ for the limit on α to be nonempty (asymptotically on m).

Let us now consider the cost of a verification. Even if we check all the $O(m^3)$ possible rotations (instead of restricting to the relevant angle $[\gamma_i, \gamma_{i+1})$), we would pay $O(m^3(\log\log m)^2) = o(m^4)$ time by using the algorithm of Section 3.2 to compute $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$ at the proper text center. On the other hand, we are searching for $O(m^3)$ features, each of which triggers verifications independently. Therefore, the total average cost of verifications, run over n/r text rows of length n, is $o(n^2m^4m^3a^m\sigma/r) = o(n^2m^6\sigma a^m)$. This is negligible whenever $a \le c\,(2\sigma)^{-1/m}$, that is, $\alpha \le 1 - \frac{2(2\delta+1)}{c\sigma^{1-1/(\sqrt{2}m)}}$ and $\alpha \le 1/2$.

Let us now consider scanning time. The transposition-invariant multipattern one-dimensional δ-matching scanning algorithm given in [10, 11] will search for the $O(ru\max(r, u)) = O(m^3)$ (rotated) features of length $u = \Theta(m)$ permitting k mismatches in $O(n(k + \log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m)$ average time per text row, provided $\alpha = k/u \le 1/2 - O(1/\sigma)$ and $2\delta + 1 < \sigma$.

If $\kappa/u \le 1 - O((\delta+1)/\sigma)$ and $\kappa/u \le 1/2 - O(1/\sigma)$, then we can use $j = 1$ and traverse one text row out of r. Adding the scanning time over all the $O(n/r)$ text rows, we obtain the final complexity $O(n^2(\kappa + \log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m^2)$.

As explained, if κ turns out to be too large, we must scan one text row out of $\lfloor r/j \rfloor$, for some sufficiently large j. As we use the one-dimensional algorithm with tolerance $k = \lfloor \kappa/j \rfloor$, the total scanning time becomes $O(\frac{n}{r/j}\ n(\kappa/j + \log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m) = O(n^2(\kappa + j\log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m^2)$.

A j value that satisfies both restrictions on k/u is $\Theta((\kappa/m)/(1-(\delta+1)/\sigma))$. Using this value the scanning time is $O(n^2(\kappa + \frac{\kappa}{m}\log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m^2$.

Considering both cases, we get complexity $O(n^2(\kappa + (1+\frac{\kappa}{m})\log_{\frac{\sigma}{\delta+1}}((\delta+1)m))/m^2$. The limit of applicability of this method is reached when $j = r$, as we cannot increase it anymore. At this point we can apply the algorithm provided $k/u \leq 1/2 - O(1/\sigma)$ where $k = \lfloor \kappa/r \rfloor$, that is, for $\kappa \leq \frac{m^2}{4}(1-O(1/\sigma))$. The other condition on r yields $\kappa \leq \frac{m^2}{2}(1-O((\delta+1)/\sigma))$. All the limits are constant on κ/m^2. Finally, it is necessary that $4\delta+2 < \sigma$. The space required by the algorithm is polynomial in $m\sigma$.

Finally, we note that, if we do not wish to allow lighting invariance and use the one-dimensional algorithm of [10, 11] without transpositions, the complexity becomes $O(n^2(\kappa + \log_{\frac{\sigma}{\delta+1}} m)/m^2)$. It is easy to see that this is average-optimal by following previous arguments [15, 11]: On one hand, we have that $O(n^2\log_\sigma(m)/m^2)$ complexity is average-optimal for two-dimensional exact matching (allowing rotations or not) [15]. On the other hand, if we can do δ-matching in less than $\Omega(|T|\log_{\frac{\sigma}{\delta+1}}(|P|)/|P|)$ (in one or two dimensions), a simple trick [11] permits doing exact matching in less than $\Omega(|T|\log_\sigma(|P|)/|P|)$, which is optimal. Finally, it is impossible to match in two dimensions allowing κ mismatches in less than $\Omega(n^2\kappa/m^2)$: We must access at least $\kappa+1$ characters in each $m \times m$ text area in order to discard it. Adding both lower bounds, we have the average-case lower bound $\Omega(n^2(\kappa + \log_{\frac{\sigma}{\delta+1}} m)/m^2)$.

5.2 γ-Matching with Tolerance κ

In this section we show how to perform γ-matching with tolerance κ on integer alphabets of size σ in average time $O(n^2((\kappa+\gamma+\kappa\gamma/m)/\sigma + (\kappa+1)\log_{\frac{\sigma}{1+\gamma/m}}(\gamma+m))/m^2)$, whenever (i) $\kappa \leq m/(2\sqrt{2})$ and $\gamma/\kappa \leq \sigma m/(2\sqrt{2}e)(1-O(1/\sigma))$, or (ii) $m/(2\sqrt{2}) \leq \kappa = O(m^2/\log m)$ and $\gamma\kappa/m^2 \leq \sigma m/(16\sqrt{2}e)(1-O(1/\sigma))$. We then present an alternative complexity based on δ-matching and discuss how to do (δ,γ)-matching.

We now make use of the full potential of Lemma 1. We scan one text row out of $\lfloor r/(j+h-1)\rfloor$, for j and h to be determined. Then Lemma 1 (with its j being our $j+h-1$ and p being h) guarantees that each occurrence of P will trigger at least h feature occurrences, each with $k = \lfloor \kappa/j \rfloor$ mismatches. If the occurrence of P γ-matches the text, then at least one of those h features must γ'-match the text, where $\gamma' = \lfloor \gamma/h \rfloor$.

Yet, we are unable to combine γ'-matching and mismatches with an efficient one-dimensional algorithm. Thus, let us *partition* the features into $k+1$ pieces (substrings) of length $\lfloor u/(k+1)\rfloor$. Each feature occurrence must contain the occurrence of at least one piece without any mismatch.

Therefore, we search for the $r(k+1) = O(r(\kappa/j+1))$ pieces permitting γ'-matching and no mismatches. Upon the occurrence of any piece, we verify the corresponding text center(s). Using the multipattern transposition invariant γ' matching algorithm in [11] for $n(j+h-1)/r$ text rows we get complexity $O(n^2(j+h)(\kappa/j+1)(\gamma/(\sigma h) + \log_{\frac{\sigma}{1+\gamma/(mh)}}(\gamma/h+m))/m^2)$, subject to the conditions $j+h \leq r$ and $(\gamma/h)/(u/(\kappa/j)) = \gamma\kappa/(huj) \leq \sigma/(2e) - O(1)$.

By expanding the first summations we get that the term accompanying n^2/m^2 is

$$O\left(\left(\kappa + \frac{\kappa h}{j} + j + h\right)\left(\frac{\gamma}{\sigma h} + \log_{\frac{\sigma}{1+\gamma/(mh)}}(\gamma/h+m)\right)\right).$$

Any j between $\Theta(\min(h, \kappa))$ and $\Theta(\max(h, \kappa))$ yields the same complexity, $O(\kappa + h)$, for the expanded sum. We will manage to maintain j within those bounds, so substituting and expanding again we get

$$O\left(\frac{\gamma\kappa}{\sigma h} + \frac{\gamma}{\sigma} + (\kappa + h)\log_{\frac{\sigma}{1+\gamma/(mh)}}(\gamma/h + m)\right),$$

where it is clear that $h = \Theta(\kappa)$ is the optimum, but maybe this optimum is outside the bounds for h. The best choices turn out to be $j = \min(\kappa + 1, m/(2\sqrt{2}))$ and $h = \min(\kappa + 1, m/(2\sqrt{2}), \gamma + 1)$. It is immediate that all the conditions hold: $j + h \leq m/\sqrt{2}$, $j \leq \kappa + 1$, $h \leq \gamma + 1$, $h \leq j \leq \kappa + 1$. Substituting we obtain the complexity

$$O\left(\frac{\gamma + \kappa + \gamma\kappa/m}{\sigma} + (\kappa + 1)\log_{\frac{\sigma}{1+\gamma/m}}(\gamma + m)\right),$$

which applies as long as $\gamma\kappa/(huj) \leq \sigma/(2e) - O(1)$. This is $\gamma/\kappa \leq \sigma m/(2\sqrt{2}e)(1 - O(1/\sigma))$ if $\kappa + 1 \leq m/(2\sqrt{2})$, and $\gamma\kappa/m^2 \leq \sigma m/(16\sqrt{2}e)(1 - O(1/\sigma))$ otherwise.

We have considered, however, only scanning time. We must derive a sufficient condition on γ and κ to make verification cost insignificant. In [15] it is shown that the probability of two strings of length ℓ to γ'-match each other (without transpositions) is $a = 2(1+\beta)(1+1/\beta)^\beta/\sigma \leq 2(1+\beta)e/\sigma$, where $\beta = \gamma'/\ell$.

We consider $O(m^3)$ rotations, searching for $r(\kappa/j + 1)$ pieces from each, of length $\ell = u/(\kappa/j + 1)$, with $\gamma' = \gamma/(h + 1)$. Each such piece, in addition, matches with transpositions, so we in practice generate σ strings from each. Therefore, the average number of verifications triggered is $O(n^2 m^3 r(\kappa/j + 1)\sigma a^{u/(\kappa/j+1)}) = O(n^2 m^4(\kappa/j)\sigma a^{mj/(\kappa\sqrt{2})})$.

According to Section 3.3, we can verify a text center in time $O(m^3(\log\log m)^2)$, and thus the overall average verification cost is $O(n^2 m^7(\log\log m)^2(\kappa/j)\sigma a^{mj/(\kappa\sqrt{2})})$. For this not to affect the average scanning time, it is sufficient to make it $O(n^2/m^2)$. Thus the condition is $\sigma a^{mj/(\kappa\sqrt{2})} = O(j/(\kappa m^9(\log\log m)^2)$.

We wish to make $a \leq c\sigma^{-1/\ell}$, for some $c < 1$, so that $\sigma a^\ell = c^\ell < 1$. For this sake, it is sufficient that $\beta = (\gamma/(h+1))/(u/(k/j+1)) < \sigma^{1-1/\ell}/(2e) - 1$. This is asymptotically the same condition we have been considering for the feature scanning algorithm.

Once this holds, the condition to make the average verification cost negligible is $c^{mj/(\kappa\sqrt{2})} = O(j/(\kappa m^9(\log\log m)^2))$. There are two cases: if $\kappa+1 \leq m/(2\sqrt{2})$, then $j = \kappa+1$ and the condition is $c^{m/\sqrt{2}} = O(m^{-9}(\log\log m)^{-2})$, always true; yet if $\kappa + 1 > m/(2\sqrt{2})$, then $j = m/(2\sqrt{2})$ and the condition becomes $c^{m^2/(4\kappa)} = O(m^{-10}(\log\log m)^{-2})$, that is $\kappa/m^2 \leq \frac{1}{40\log_{1/c} m}(1 + o(1))$.

Let us consider some particular cases of our results. If we do not permit outliers, $\kappa = 0$, the cost of our algorithm is $O(n^2(\gamma/\sigma + \log_{\frac{\sigma}{1+\gamma/m}}(\gamma + m))/m^2)$ for $\gamma \leq m\sigma/(2\sqrt{2}e)(1 - O(1/\sigma))$. This is the same complexity obtained without transpositions in previous work [15], yet for a stricter condition on γ.

On the other hand, our results with κ outliers without transposition invariance are also relevant. By using the feature scanning algorithm of [15] (that does not permit transpositions) we arrive at essentially the same complexity, except the argument of the logarithm is m instead of $\gamma + m$.

Finally, it is not hard to adapt the lower bounds in [11] to show that a lower bound on the transposition-invariant version of this problem is $\Omega(n^2(\kappa + \gamma/\sigma + \log_{\sigma m/\gamma} m)/m^2)$, not far away from what we have obtained.

An alternative method based on δ-matching. Another idea for γ-matching is to search for the features using δ-matching, with $\delta = \lfloor \gamma/h \rfloor$. This will obviously spot all the γ'-occurrences, where $\gamma' = \delta$. Then, for each occurrence of a feature, we check the corresponding text center for a γ-occurrence of P. We can use the scanning algorithm of Section 5.1, as well as most of the analysis because the verification costs are identical (Section 3.3). That is, we obtain average search time $O(n^2(j+h)(\kappa/j + (1 + \frac{\kappa/j}{m}) \log_{\frac{\sigma}{1+\gamma/h}}(\sigma m))/m^2)$, provided $j \geq 2\sqrt{2}\kappa/m(1 - O(1/\sigma))$, $j \geq \sqrt{2}\kappa/m(1 - O((1+\gamma/h)/\sigma))$, $4\gamma/h + 2 < \sigma$, and $j + h \leq r$.

By calling $L = \log_{\frac{\sigma}{1+\gamma/h}}(\sigma m)$ and distributing the sums, we get that the term accompanying n^2/m^2 is

$$O(\kappa + h\kappa/j + jL + hL + \frac{\kappa}{m}L + \frac{h\kappa}{jm}L).$$

By setting $j = h$, the formula reduces to $O(\kappa + hL + \frac{\kappa}{m}L)$. We choose h as small as possible, $h = 1 + 4\gamma/(\sigma - 2)$. With this choice, the final complexity is

$$O\left(n^2\left(\kappa + \left(1 + \frac{\gamma}{\sigma} + \frac{\kappa}{m}\right)\log(\sigma m)\right)/m^2\right).$$

Let us check whether j and h satisfy the boundary conditions. It holds $j + h \leq r$ if $\gamma \leq (m - 2\sqrt{2})(\sigma - 2)/(8\sqrt{2})$. The strictest lower bound on j is $j \geq 2\sqrt{2}\kappa/m(1 - O(1/\sigma))$. If this does not hold, we instead increase j and h to $j = h = 2\sqrt{2}\kappa/m$. The complexity stays the same, and now condition $j + h \leq r$ becomes $\kappa/m^2 \leq 1/8$. Thus the (slightly simplified) final conditions are $\gamma \leq m\sigma/(8\sqrt{2})$ and $\kappa \leq m^2/8$, much looser on κ than with the previous technique, and we have obtained an alternative complexity where κ does not multiply the logarithm. On the other hand, the logarithm is multiplied by γ/σ, and we do not yet reach the lower bound we have proved.

Note also that we can use this algorithm without transposition invariance, with the only difference that the σ inside the logarithm disappears.

(δ, γ)-matching with tolerance κ. We can just use either the δ-matching algorithm of Section 5.1 or the γ-matching algorithm of Section 5.2, using the verification algorithm of Section 3.4. The resulting complexity is the minimum among those we obtained for δ- or γ-matching.

6 Conclusions and Future Work

We have presented the first combinatorial approach to the problem of two-dimensional template matching permitting rotations and lighting invariance, where in addition there is some tolerance for differences between the pattern and its occurrences. We have defined a set of meaningful distance measures and search problems, which extend previous search problems [14, 15]. We have built on top of previous rotation-invariant (but not lighting-invariant) search techniques [14, 15] and of previous one-dimensional transposition-invariant search algorithms [21, 10, 11].

We have developed algorithms to compute the defined distances, as well as algorithms for all the search problems, which are at the same time efficient in the worst and average case. We have shown that adding lighting invariance poses a small computational price on top of previous rotation invariant search algorithms [14, 15], several of which are already optimal. In particular, we have obtained in some cases average complexities that match the optimal existing results that do not permit lighting invariance.

The results can be extended to more dimensions. In three dimensions, for example, there are $O(m^{12})$ different matching functions for P [18], and $O(um^2)$ features of length u. The worst-case time algorithms retain their complexity as long as we replace $O(m^3n^2)$ by $O(m^{12}n^3)$. Average case algorithms also retain their complexity if we replace $O(n^2/m^2)$ by $O(n^3/m^3)$.

It is also possible to remove some restrictions we have used for simplicity, such as the center-to-center assumption. In this case the number of relevant rotations and small displacements grows up to $O(m^7)$ [9], so the worst case complexities shift to $O(\ldots m^7n^2)$. Average case complexities are not affected.

Future work involves trying to close the gap between our complexities and the known lower bounds, pushing in either way, both for worst-case and average-case complexities. Finally, it would be good to obtain an algorithm for (δ, γ)-matching that works for general alphabets, as the one we have proposed only works for integer alphabet.

References

[1] A. Amir, A. Butman, M. Crochemore, G. Landau, and M. Schaps. Two-dimensional pattern matching with rotations. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, LNCS 2676, pages 17–31, 2003.

[2] A. Amir, O. Kapah, and D. Tsur. Fast two-dimensional pattern matching with rotations. In *Proc. 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004)*, LNCS v. 3109, pages 409–419, 2004.

[3] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? In *Proc. 27th Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 427–436, 1995.

[4] A. Andersson and M. Thorup. Tight(er) worst-case bounds on dynamic searching and priority queues. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC'00)*, pages 335–342, 2000.

[5] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, 1992.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.

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

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 2nd rev. edition, 2000.

[9] K. Fredriksson. Rotation invariant histogram filters for similarity and distance measures between digital images. In *Proc. 7th String Processing and Information Retrieval (SPIRE 2000)*, pages 105–115. IEEE CS Press, 2000.

[10] K. Fredriksson, V. Mäkinen, and G. Navarro. Flexible music retrieval in sublinear time. In *Proc. 10th Prague Stringology Conference (PSC'05)*, pages 174–188, 2005.

[11] K. Fredriksson, V. Mäkinen, and G. Navarro. Flexible music retrieval in sublinear time. In *Proceedings of the 10th Prague Stringology Conference (PSC'05)*, pages 174–188, 2005. Extended version to appear in *IJFCS*.

[12] K. Fredriksson and G. Navarro. Average-optimal single and multiple approximate string matching. *ACM Journal of Experimental Algorithmics (JEA)*, 9(1.4), 2004.

[13] K. Fredriksson, G. Navarro, and E. Ukkonen. *Faster than FFT: Rotation Invariant Combinatorial Template Matching*, volume II, pages 75–112. Transworld Research Network, 2002.

[14] K. Fredriksson, G. Navarro, and E. Ukkonen. Optimal exact and fast approximate two dimensional pattern matching allowing rotations. In *Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002)*, LNCS 2373, pages 235–248, 2002.

[15] K. Fredriksson, G. Navarro, and E. Ukkonen. Sequential and indexed two-dimensional combinatorial template matching allowing rotations. *Theoretical Computer Science A*, 347(1–2):239–275, 2005.

[16] K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. 9th Combinatorial Pattern Matching (CPM'98)*, LNCS 1448, pages 118–125, 1998.

[17] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. *Patt. Recog. Letters*, 20(11–13):1249–1258, 1999.

[18] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate pattern matching under rotations and translations in 3d arrays. In *Proc. 7th String Processing and Information Retrieval (SPIRE 2000)*, pages 96–104. IEEE CS Press, 2000.

[19] G. Navarro K. Fredriksson and E. Ukkonen. An index for two dimensional string matching allowing rotations. In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *IFIP TCS2000*, LNCS 1872, pages 59–75, 2000.

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

[21] V. Mäkinen, G. Navarro, and E. Ukkonen. Transposition invariant string matching. *Journal of Algorithms*, 56(2):124–153, 2005.

[22] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* Springer Verlag, 1984.

[23] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[24] C. Russ. *The Image Processing Handbook.* CRC Press, 4th edition, 2002.

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

[26] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.