Rotation and Lighting Invariant Template Matching

Kimmo Fredriksson[1], Veli Mäkinen[2⋆], and Gonzalo Navarro[3] ⋆⋆

[1] Department of Computer Science, University of Joensuu.
kfredrik@cs.joensuu.fi
[2] Department of Computer Science, University of Helsinki.
vmakinen@cs.helsinki.fi
[3] Center for Web Research, Department of Computer Science, University of Chile.
gnavarro@dcc.uchile.cl

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 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 pattern and 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, 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 [3] 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 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 less 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 [10, 11]. Since then, several fast filters have been developed for diverse matching models [12, 6, 13, 8, 7, 9]. These represent large performance improvements over the FFT-based approach. The worst-case complexity of the problem was also studied [1, 8]. However, lighting invariance has not been considered in this scenario.

⋆ A part of the work was done while visiting University of Chile under a researcher exchange grant from University of Helsinki.
⋆⋆ Funded by Millenium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile.

On the other hand, *transposition invariant* string matching was considered in music retrieval [4, 14]. 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 in [16, 15].

We note that transposition invariance becomes lighting invariance when we replace musical notes by gray levels of cells in an image. Hence, the aim of this paper is to enrich the existing algorithms for rotation invariant template matching [8] with the techniques developed for transposition invariance [16] 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 transposition invariant distances can be computed incrementally taking the computation done with the previous rotation into account in the next rotation angle.

Let us now determine which are the reasonable matching models. In [8], some of the models considered were useful only for binary images, a case where obviously we are not interested in this paper. We will address models that make sense for gray level images. We define three transposition-invariant distances: $d_{\mathrm{H}}^{\mathrm{t},\delta}$, which counts how many pattern and text cells differ by more than δ; $d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$, which is the maximum color difference between pattern and text cells when up to κ outliers are permitted; and $d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$, which is the sum of absolute color differences between pattern and text cells permitting up to κ outliers. Table 1 shows our complexities to compute these distances for every possible rotation of a pattern centered at a fixed text position. Variable σ is the number of different gray levels (assume $\sigma = \infty$ if the alphabet is not a finite discrete range). We remark that a lower bound to this problem is $O(m^3)$, and this is achieved in [9] without lighting invariance.

Distance	Complexity
$d_{\mathrm{H}}^{\mathrm{t},\delta}$	$\min(\log m, \sigma + (\delta+1))m^3$
$d_{\mathrm{MAD}}^{\mathrm{t},\kappa}$	$(\min(\kappa,\sigma) + \log\min(m,\sigma))m^3$
$d_{\mathrm{SAD}}^{\mathrm{t},\kappa}$	$(\min(\kappa,\sigma) + \log\min(m,\sigma))m^3$

Table 1. Worst-case complexities to compute the different distances defined.

We also define three search problems, consisting in finding all the transposition-invariant rotated occurrences of P in T such that: there are at most κ cells of P differing by more than δ from their text cell (δ-matching); the sum of absolute difference 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 results (the latter are valid only on finite integer alphabets). Without transposition invariance the worst cases are all $O(m^3n^2)$ [9]. In the same paper they give average case algorithms for δ-matching with $\delta = 0$ and for γ-matching for $\kappa = 0$. The respective average complexities are $O(n^2(\kappa + \log_\sigma m)/m^2)$ and $O(n^2(\gamma/\sigma + \log m)/m^2)$. The former is indeed average-optimal and the latter is almost (and conjectured) average-optimal, as shown in [9]. Hence our complexities are rather close to be optimal.

We remark that we have developed algorithms that work on arbitrary alphabets, but we have also taken advantage of the case where the alphabet is a discrete range of integer values.

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. A particular

Problem	Worst case	Average case
δ-matching	$\min(\log m, \sigma + (\delta+1))m^3n^2$	$n^2\kappa \log_{\sigma/(2\delta+1)}(m)/m^2$, for $2\delta+1 < \sigma$ and $\kappa \leq m/\sqrt{2}$
	$(\min(\kappa,\sigma) + \log\min(m,\sigma))m^3n^2$	
γ-matching	$(\min(\kappa,\sigma) + \log\min(m,\sigma))m^3n^2$	$n^2(\kappa + \gamma/\sigma)\log(m)/m^2$, for $\gamma \leq (m/\sqrt{2} - \kappa)/2$
(δ,γ)-matching	$(\min(\kappa,\sigma)\sqrt{\gamma} + \log\min(m,\sigma))m^3n^2$	best of the two above

Table 2. Complexities for different search problems. (δ, γ) matching and average complexities are valid only for integer alphabets.

case of interest is that of Σ being a finite integer range of size σ. 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 (*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 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.

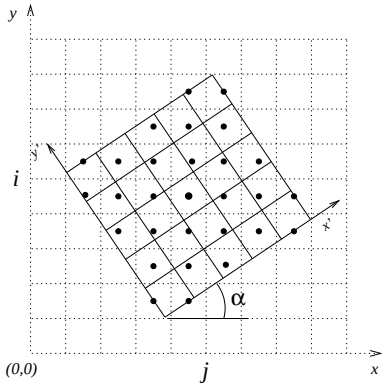

Fig. 1. Each text cell is matched against the pattern cell that covers the center of the text cell.

More precisely, 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 $M(T[r,s]) = P[r',s']$, that is, our algorithms compare the cell $T[r,s]$ of T against the cell $M(T[r,s])$ of P.

Hence the *matching function* M is a function from the cells of T to the cells of P. 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 in [10] that this happens $O(m^3)$ times, when P rotates full 2π radians. This result was shown to be also a lower bound in [1]. Hence

there are $\Theta(m^3)$ relevant orientations of P to be checked. The set of angles for $0 \leq \theta \leq \pi/2$ is

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

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

$$\mathcal{A} = A \ \cup \ A + \pi/2 \ \cup \ A + \pi \ \cup \ A + 3\pi/2.$$

Furthermore, pattern P matches at location $((i, j), \theta)$ with lighting invariance if there is some integer transposition t such that $T[r, s] + t = P[r', s']$ for all $[r', s']$ in the area of 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-invariance 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 $T[r, s] \neq P[r', s']$ occurs, over all the cells of P, that is

$$d_{\mathrm{H}}(i, j, \theta, t) = \sum_{r', s'} \textbf{if } T[r, s] + t \neq P[r', s'] \textbf{ then } 1 \textbf{ else } 0$$
$$d^{\mathrm{t}}_{\mathrm{H}}(i, j, \theta) = \min_t d_{\mathrm{H}}(i, j, \theta, t)$$

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, $T[r, s] + t \notin [P[r', s'] - \delta, P[r', s'] + \delta]$.

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

$$d_{\mathrm{MAD}}(i, j, \theta, t) = \max_{r', s'} |T[r, s] + t - P[r', s']|$$
$$d^{\mathrm{t}}_{\mathrm{MAD}}(i, j, \theta) = \min_t d_{\mathrm{MAD}}(i, j, \theta, t)$$

This can be extended to distance $d^{\kappa}_{\mathrm{MAD}}$ and its transposition-invariant version $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$, so that up to 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 $|T[r, s] - P[r', s']|$ values over all the cells of P, that is,

$$d_{\mathrm{SAD}}(i, j, \theta, t) = \sum_{r', s'} |T[r, s] + t - P[r', s']|$$
$$d^{\mathrm{t}}_{\mathrm{SAD}}(i, j, \theta) = \min_t d_{\mathrm{SAD}}(i, j, \theta, t)$$

Similarly, this distance can be extended to $d^{\kappa}_{\mathrm{SAD}}$ and its transposition-invariant version $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$, where up to κ pattern 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 $d^{\mathrm{t}}_{\mathrm{MAD}}(i, j, \theta) \leq \delta$. A tolerance κ can be permitted, so that we only require $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}(i, j, \theta) \leq \delta$. Observe that this condition is the same as $d^{\mathrm{t},\delta}_{\mathrm{H}}(i, j, \theta) \leq \kappa$.

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

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

3 Efficient Worst-Case Algorithms

In [1] it was shown that for the problem of the two dimensional pattern matching allowing rotations the worst case lower bound is $\Omega(n^2m^3)$. We have shown in [8] a simple way to achieve this lower bound for any of the distances under consideration (without lighting invariance).

The idea is that we will 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.

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 must hit a cell boundary at β_i. 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 time for evaluating the distance for P centered at some position in T, for all possible angles, is $O(\sum_i |\mathcal{C}_i|)$. This is $O(m^3)$ because each fixed cell center of T, covered by P, can belong to some $\mathcal{C}_i$ at most $O(m)$ times. To see this, note that when P is rotated the whole angle 2π, any cell of P traverses through $O(m)$ cells of T.

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 $O(n^2m^3\sigma)$. 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 maintaining 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 can we achieve this for each of the distances under consideration.

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 manage to avoid verifying a given text center more than once, then we can apply our verification technique and ensure that, overall, we cannot pay more than $O(n^2m^3f(m))$.

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

As proved in [15], 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 $d_{\mathrm{H}}^{\mathbf{t},\delta}$. Let us now separate the cases of integer and general alphabets.

Integer alphabet. The original algorithm [15] 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).

In our case, we have to pay $O(\sigma + m^2)$ in order to find the optimal transposition for the first rotation angle. The problem is how to recompute the optimal transposition once some text cell $T[r, s]$ changes its value (due to a small change in rotation angle). The net effect 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 $O(\sigma)$ possible transpositions. Each transposition can be classified according to the number of votes it has. There are m^2+1 lists L_i, $0 \le i \le m^2$, containing the transpositions that currently have i votes. Hence, when a range of transpositions loses/gains one vote, the $2\delta+1$ transpositions are moved to the lower/upper list. We need to keep control of which is the highest-numbered non-empty list, which is easily done in constant time per operation because transpositions move only from one list to the next/previous. Initially we pay $O(\sigma + m^2)$ to initialize all the lists and put all the transpositions in list L_0, then $O((\delta+1)m^2)$ to process the votes of all the cells, and then $O(\delta+1)$ to process each cell that changes. Overall, when we consider all the $O(m^3)$ cell changes, the scheme is $O(\sigma + (\delta+1)m^3)$. This is our complexity to compute distance $d_{\mathrm{H}}^{\mathrm{t},\delta}$ between a pattern and a text center, considering all possible rotations and transpositions.

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 $d_{\mathrm{H}}^{\mathrm{t},\delta}(i,j,\theta) \le \kappa$. In fact, the final state of the lists (rotation of 2π) is equal to their state when built for the first rotation (angle zero), so it is possible to turn back to the initial state at cost $O(m^2)$. Hence we can move to the next text cell without paying again the $O(\sigma)$ initialization time. This means that our overall search time is $O(\sigma + (\delta+1)n^2m^3)$.

General alphabet. Let us resort to a more general problem of *dynamic range voting*: In the static case we have a multiset $S = \{[\ell, r]\}$ of one-dimensional closed ranges, and we are interested in obtaining a point p that is included in most ranges, that is $\mathrm{maxvote}(S) = \max_p |\{[\ell,r] \in S \mid \ell \le p \le r\}|$. In the dynamic case a new range is added to or an old one is deleted from S, and we must be able to return $\mathrm{maxvote}(S)$ after each update.

Notice that our original problem of computing $d_{\mathrm{H}}^{\mathrm{t},\delta}$ from one rotation angle to another is a special case of dynamic range voting; multiset S is $\{[P[r',s'] - T[r,s] - \delta, P[r',s'] - T[r,s] + \delta] \mid M(T[r,s]) = P[r',s']\}$ in one rotation angle, and in the next one some $T[r,s]$ changes its value. That is, the old range is deleted and the new one is inserted, after which $\mathrm{maxvote}(S)$ is requested to compute the distance $d_{\mathrm{H}}^{\mathrm{t},\delta} = m^2 - \mathrm{maxvote}(S)$ in the new angle.

We show that dynamic range voting can be supported in $O(\log |S|)$ time, which 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.

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 $|\{[\ell,r] \mid \ell \le e \le r, [\ell,r] \in S\}|$, 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 replace 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

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

where path$(root, e)$ is the set of nodes in the path from the root to a leaf e (including the leaf). Values maxdiff(v) are defined recursively as

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

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 easily notices that

$$\text{maxvote}(v) = \text{maxdiff}(v) + \sum_{v' \in \text{path}(root,v)} \text{diff}(v'), \tag{3}$$

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 cases 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.

Case (i) is handled by storing in each leaf an additional counter end(e). It 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. The $+1$ vote induced by the new range whose endpoint e is, will be handled in case (ii). To make vote$(e) = \sum_{v' \in \text{path}(root,e)} \text{diff}(v')$, we fix diff$(e)$ so that vote$(e) =$ diff$(e) + \sum_{v' \in \text{path}(root,v)} \text{diff}(v')$, where v is the parent of e. Once the maxdiff() values are updated in the path from e to the root, we can conclude that all the necessary updates are done in $O(\log |S|)$ time.

Let use then consider case (ii). Recall the one-dimensional range search on a balanced binary search tree (see e.g. [5], Section 5.1). We use the fact that one can find in $O(\log |S|)$ time the minimal set of nodes, say 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 them. 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). 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 $O(\log |S|)$.

Finally, let us consider case (iii). Counters diff(v) are affected by rotations, but in case a 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 rotation can be carried out. Subtree maxima are easily maintained through 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 $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and δ-Matching

Let us start with $\kappa = 0$. As proved in [15], 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 $P[r', s'] - T[r, s]$. Then, the optimal transposition is the average between the minimum and maximum vote. The $d^{\mathrm{t}}_{\mathrm{MAD}}$ distance yielded is the difference of maximum minus minimum, divided by two. Hence an $O(|P|)$ algorithm was immediate.

We need $O(m^2)$ to obtain the optimal transposition for the first angle, zero. 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 minimum and maximum votes. Every time a text character changes, a vote disappears and a new vote appears. We can simply maintain balanced search trees with all the current votes so as to handle any insertion/deletion of votes in $O(\log(m^2)) = O(\log m)$ time, knowing the minimum and maximum at any time. If we have an integer alphabet of size σ, there are only $2\sigma + 1$ possible votes, so it is not hard to obtain $O(\log \sigma)$ complexity. Hence $d^{\mathrm{t}}_{\mathrm{MAD}}$ distance between a pattern and a text center can be computed in $O(m^3 \log m)$ or $O(m^3 \log \min(m, \sigma))$ time, for all possible rotations and transpositions.

In order to account for up to κ outliers, it was already shown in [15] 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 $v_1 \ldots v_{m^2}$, then distance $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ is the minimum among distances $d^{\mathrm{t}}_{\mathrm{MAD}}$ computed in sets $v_1 \ldots v_{m^2-\kappa}$, $v_2 \ldots v_{m^2-\kappa+1}$, and so on until $v_{\kappa+1} \ldots v_{m^2}$. Moreover, the optimum transposition of the i-th value of this list is simply the average of maximum and minimum, that is, $(v_{m^2-\kappa-1+i} + v_i)/2$.

So our algorithm for $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ is as follows. We make our tree threaded, so we can easily access the $\kappa + 1$ smallest and largest votes. After each change in the tree, we retraverse these $\kappa + 1$ pairs and recompute the minimum among the $v_{m^2-\kappa-1+i} - v_i$ differences. This takes $O(m^3(\kappa + \log m))$ time. In case of an integer alphabet, since there cannot be more than $O(\sigma)$ different votes, this can be done in time $O(m^3(\min(\kappa, \sigma) + \log \min(m, \sigma)))$.

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 $O((\kappa + \log m)n^2m^3)$ or $O((\min(\kappa, \sigma) + \log \min(m, \sigma))n^2m^3)$ time algorithm to solve the δ-matching problem.

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

Let us first consider case $\kappa = 0$. This corresponds to the SAD model of [15], 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 $d^{\mathrm{t}}_{\mathrm{SAD}}$. Then 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, and then have to manage to maintain the median transposition and current distance when some text cells change their value due to small rotations.

We maintain a balanced and threaded binary search tree for the votes, plus a pointer to the median vote. 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. When insertion and deletion occur at different halves of the sorted list of votes (that is, one is larger and the other smaller than the median), the median may move by one position. This is done in constant time since the tree is threaded.

The median value itself can change. One change is due to the fact that one of the votes changed 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}. Another change is due to the fact that the median position can change from a value in the sorted list to the next or previous. It was shown in [15] 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 v_j to v_{j+1}, then all the j smallest votes increase their value by $v_{j+1} - v_j$, and the $m - j$ largest votes decrease by $v_{j+1} - v_j$. Hence distance d_{SAD} at the new transposition is the value at the old transposition plus $(2j - m)(v_{j+1} - v_j)$.

Hence, we can traverse all the rotations in time $O(m^3 \log m)$. This can be reduced to $O(m^3 \log \min(m, \sigma))$ on finite integer alphabet, by noting that there cannot be more than $O(\sigma)$ different votes, and taking some care in handling repeated values inside single tree nodes.

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 $m - \lceil \kappa/2 \rceil \ldots m + \lceil \kappa/2 \rceil$ in the list of sorted votes, where m is the position of the median for the whole list.

Hence, instead of maintaining a pointer to the median, we maintain two pointers to the range of $\kappa+1$ medians that could be relevant. It is not hard to maintain left and right pointers when votes are inserted and deleted in the set. All the median values can be changed one by one, and we can choose the minimum distance among the $\kappa+1$ options. This gives us an $O(m^3(\kappa+\log m))$ time algorithm to compute $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$. On integer alphabet, this is $O(m^3(\kappa+\log\min(m,\sigma)))$, which can be turned into $O(m^3(\min(\kappa,\sigma)+\log\min(m,\sigma)))$ by standard tricks using the fact that there are $O(\sigma)$ possible median votes that have different values.

This immediately gives an $O((\kappa+\log m)n^2m^3)$ or $O((\min(\kappa,\sigma)+\log\min(m,\sigma))n^2m^3)$ time algorithm 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

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 $t\in[t_1^{\mathrm{MAD}},t_2^{\mathrm{MAD}}]=[t^{\mathrm{MAD}}-(\delta-d^{\mathrm{t}}_{\mathrm{MAD}}(i,j,\theta)),t^{\mathrm{MAD}}+(\delta-d^{\mathrm{t}}_{\mathrm{MAD}}(i,j,\theta))]$.

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) $t_1^{\mathrm{MAD}}\leq t^{\mathrm{SAD}}\leq t_2^{\mathrm{MAD}}$, in which case the occurrence can be reported; (ii) $t^{\mathrm{SAD}}<t_1^{\mathrm{MAD}}$, in which case we report the occurrence only if $d_{\mathrm{SAD}}(i,j,\theta,t_1^{\mathrm{MAD}})\leq\gamma$; (iii) $t^{\mathrm{SAD}}>t_2^{\mathrm{MAD}}$, in which case we report the occurrence only if $d_{\mathrm{SAD}}(i,j,\theta,t_2^{\mathrm{MAD}})\leq\gamma$.

As in the worst case we may have to check $O(m^3n^2)$ 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 $O(m^5n^2)$, 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 [15], 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 i transpositions from the median, we have i votes contributing in one unit each to d_{SAD}, so after we move i times d_{SAD} has increased in $O(i^2)$ (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, search time is $O((\sqrt{\gamma}+\log\min(m,\sigma))n^2m^3)$.

The situation is more complex if we permit κ outliers. Fortunately, both in $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ 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 $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ is not the same producing distance $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$, it holds that if there is a selection that produces a $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ distance of at most δ and a $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ 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 the $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ distances can only decrease if we replace the votes in the initial selection by excluded minimum or maximum votes.

Now we compute $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ distances and consider every triple (i,j,θ) where both criteria coincide. 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 already have $d^{\mathrm{t},\kappa}_{\mathrm{MAD}}$ and $d^{\mathrm{t},\kappa}_{\mathrm{SAD}}$ 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 $O((\min(\kappa,\sigma)\sqrt{\gamma}+\log\min(m,\sigma))n^2m^3)$. We remark that this works only for integer alphabets.

4 Features

As shown in [10, 8], any match of a pattern P in a text T allowing arbitrary rotations must contain some so-called "features", i.e., 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. See Figure 2.

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 $F^q(\theta)=c_1^qc_2^q\cdots c_u^q$. Note that this value depends only on q, θ and P, not on T.

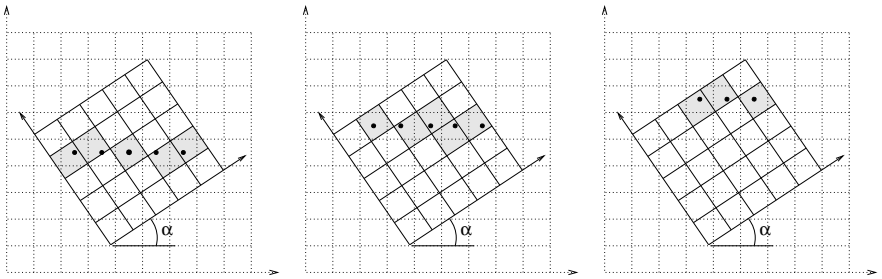

Fig. 2. For each angle θ, a set of features is read from P. We show $F^0(\alpha)$, $F^1(\alpha)$ and $F^2(\alpha)$.

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). Therefore, if a match of some feature $F^q(\theta)$ is found, there are $O(|\mathcal{A}|/|\mathcal{B}^q|)$ possible orientations to be verified for an occurrence 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 $\gamma_i\le\theta<\gamma_{i+1}$. On the other hand, there are $O(|\mathcal{A}|/|\mathcal{B}^q)|$ angles $\beta\in\mathcal{A}$ such that $\gamma_i\le\beta<\gamma_{i+1}$. If there is an occurrence of $F^q(\theta)$, then P may occur with any such angle β.

Our plan is to build a filter based on exact searching for features, but this search must be transposition-invariant. We will consider integer integer alphabets only. Exact transposition-invariant search is very simple, however. Recode the text T into T' such that $T'[i,j]=T[i,j]-T[i,j-1]$ and disregard the first column from T'. Recode the linear features $F^q(\theta)=c_1^q,c_2^q,c_3^q,\ldots,c_u^q$ as $F'^q(\theta)=(c_2^q-c_1^q),(c_3^q-c_2^q),\ldots,(c_u^q-c_{u-1}^q)$. Hence, $F^q(\theta)$ appears in $T[i,j]\ldots T[i,j+u-1]$ under some transposition t if and only if $F'^q(\theta)$ appears in $T'[i,j+1]\ldots T[i,j+u-1]$. Moreover, $t=T[i,j]-F^q(\theta)[1]$.

The statistics of this new text T' are not the same as in T. The probability that a text and a feature character match is not $1/\sigma$ anymore. If we have two contiguous text characters c_1c_2 and two contiguous feature characters f_1f_2 then the probability that the corresponding differences match in the transformed strings is that of c_2-c_1 being equal to f_2-f_1. Since the four variables are independent and uniform over $[1,\sigma]$, it can be seen that this probability is bounded above by $1/\sigma$. Hence we will pessimistically assume that the transformed text and features share the same statistics of the original text and features. Note that in the transformed text and features there is a dependence between consecutive pairs, but in any case all the

probabilities are bounded above by $1/\sigma$, so it is (conservatively) correct to assume that they are independent with probability $1/\sigma$ each (note that, since there are $2\sigma+1$ different values, probabilities add $2+1/\sigma$ in our pessimistic setup).

5 Efficient Average-Case Time Algorithms

Just as in [8], we choose features of length u from r pattern rows around the center, rotate them in all possible ways, and search for all them using a multipattern exact search algorithm. To simplify the presentation we assume from the beginning $u = r = m/\sqrt{2}$, which are in fact the optimal values. Our results are valid only for integer alphabets.

5.1 δ-Matching with Tolerance κ

If we examine one text row out of r, then every occurrence of P must contain a feature. Moreover, if we examine one text row out of $\lfloor r/(\kappa+1) \rfloor$, then every occurrence of P must contain $\kappa+1$ features, and therefore it must contain some feature that δ-matches without any tolerance.

In order to search for a feature $F^q(\theta) = c_1^q, c_2^q \ldots c_u^q$ that δ-matches using an exact search machinery, we must generate all its δ-variants, that is, all the strings that δ-match $F^q(\theta)$. These can be described as the product $[c_1^q - \delta, c_1^q + \delta] \times [c_2^q - \delta, c_2^q + \delta] \times \ldots \times [c_u^q - \delta, c_u^q + \delta]$. This set is of size $(2\delta+1)^u$. If we account for all the $O(ru\max(r,u)) = O(m^3)$ rotations of each feature we arrive at a total of $O(m^3(2\delta+1)^{\Theta}(m))$ strings. This is too much, so we take the following approach.

We preprocess the set of $O(m^3)$ patterns of length u by collecting all their ℓ-grams (substrings of length ℓ), where ℓ will be determined soon. Hence the total number of substrings collected is bounded by $O(m^4(2\delta+1)^\ell)$. All these strings are stored in a trie data structure [2], which takes $O(m^4\ell(2\delta+1)^\ell)$ space and permits searching for a string in the set in $O(\ell)$ time. We remark that, after generating all the different strings and before inserting them into the trie, we must transform them to their differentially encoded versions to search for them in transposition invariant form.

Then, we slide a window of length u along the text row. At each window, we read its last ℓ characters. If this string belongs to the set of ℓ-grams (which can be determined in $O(\ell)$ time with the trie), then we report the window as a match of all the features that contain the matched ℓ-gram (although it might actually not match any of these features), and shift the window by one position. If, on the other hand, the last ℓ-gram of the window does not belong to the set, then no feature occurrence can overlap this ℓ-gram, so we can safely shift the window by $u-\ell+1$ characters.

For each feature declared to match inside each text window, we must verify the corresponding text center (i,j) so as to determine whether there is a complete occurrence of P at (i,j). The probability of a feature match being declared at a given text position is $O(m^4(2\delta+1)^\ell/\sigma^\ell)$. Since the time to verify a candidate text center is $O((\kappa+\log m)m^3)$, we have that the overall verification cost per text row is on average $O(nm^7(\kappa+\log m)/(\sigma/(2\delta+1))^\ell)$. Since this method works only for $k \le r = O(m)$, this is $O(n\log(m)/m)$ provided $\ell \ge 9\log_{\sigma/(2\delta+1)} m$. So let this be the value of ℓ, and we will see soon why we want to be sure that verification cost is $O(n\log(m)/m)$.

Feature search time can be divided into two parts. We may either advance the window by one position or by $u-\ell+1$. In both cases we pay $O(\ell)$ per window. We consider at most n text windows in a row. The probability of advancing by one position is that of finding the last window ℓ-gram, that is, $O(m^4(2\delta+1)^\ell/\sigma^\ell) = O(1/m^5)$, hence the cost for windows that are advanced by one position is $O(n\log_{\sigma/(2\delta+1)}(m)/m^5)$, which is totally negligible. For the other windows we pay $O(n\ell/(m-\ell)) = O(n\log_{\sigma/(2\delta+1)}(m)/m)$ per text row. So we have previously made ℓ large enough to make verification cost smaller than feature search cost.

Since we have to traverse one row out of $r/(\kappa+1)$, the overall search cost, counting both feature searching and verifications, is

$$O\left(\frac{n^2\kappa\log_{\sigma/(2\delta+1)} m}{m^2}\right)$$

which assumes $2\delta+1<\sigma$ and $\kappa \le m/\sqrt{2}$.

For this value of ℓ, the space for the trie of ℓ-grams is $O(ru^2\max(r,u)\ell(2\delta+1)^\ell) = O(m^4\log(m)m^{9/(\log_{2\delta+1}\sigma-1)})$, which is polynomial in m.

For the sake of simplicity we have disregarded many optimizations that do not change the complexity but improve a practical implementation. For example, with the information given by the feature we do not actually need to try all the $O(m^3)$ rotations but just $O(m)$ of them, which are in the range that is consistent with the angle of the feature. It is also not necessary that we check the whole P once an ℓ-gram is found, but we could first check for the whole feature (one by one). Just these two improvements reduce ℓ to $4\log_{\sigma/(2\delta+1)} m$ and the space requirement to $O(m^{4(1+1/(\log_{2\delta+1}\sigma-1))}\log m)$. This can be further lowered by, for example, building up a data structure to check for all the whole features faster.

5.2 γ-Matching with Tolerance κ

Let us search one text row out of $\lfloor r/(\kappa+h)\rfloor$. Since $\kappa+h$ features will appear inside every occurrence, at least h of them will appear without outliers. Hence at least one feature must appear with d_{SAD} distance of at most $\lfloor\gamma/h\rfloor$. Otherwise, each of the h features match with at least $\lfloor\gamma/h\rfloor+1$ outliers, and hence the total number of outliers exceeds γ since $h(\lfloor\gamma/h\rfloor+1) > h(\gamma/h)=\gamma$.

Hence, we run our δ-matching algorithm for $\delta=\gamma/h$, that is, we are much less restrictive and permit distance γ/h at each character instead of the overall feature. Since verification for γ-matching costs the same as for δ-matching, we can use the analysis of δ-matching verbatim (except that we traverse $n(\kappa+h)/r$ text rows) and have a search cost of

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

where it is clear that the minimum h is the optimum. However, it might be that $2(\gamma/h)+1 \ge \sigma$, which contradicts our precondition for δ-matching. Hence we need $h > 2\gamma/(\sigma-1)$. This gives a search cost of

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

which works whenever $\gamma \le (m/\sqrt{2}-\kappa)/2$.

5.3 (δ,γ)-Matching with Tolerance κ

In this case we can use any of the above filters (that is, the one giving best complexity or maybe the one that can be applied), and change only verification to check for the (δ,γ)-condition. The higher complexity of checking does not affect overall search time.

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 difference between the pattern and its occurrence. We have defined a set of meaningful distance measures and search problems, which extend previous search problems [9]. We have built on top of previous rotation-invariant (but not

lighting-invariant) search techniques [9] and of previous one-dimensional lighting-invariant search algorithms [16].

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 [9], several of which are already optimal.

The results can be extended to more dimensions. In three dimensions, for example, there are $O(m^{12})$ different matching functions for P [12], 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 as long as 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 $O(m^7)$ [6], so the worst case complexities shift to $O(\ldots m^7n^2)$. Average case complexities are not affected.

On the other hand, our average-case results can be applied to the one-dimensional lighting-invariant search too. If we split the pattern into $\kappa+1$ pieces, then some piece must match without outliers. A multipattern search for those pieces, with δ-tolerance, enables an $O(n\kappa \log_{\sigma/2\delta+1}(m)/m)$ average-time algorithm for δ-matching. Similarly, we can get $O(n(\kappa+\gamma/\sigma)\log(m)/m)$ for γ-matching. Finally, (δ,γ)-matching can be done with the best of these two complexities.

A technique used in [9] to obtain optimal search times was to reduce the problem to *approximate* rather than *exact* search for pattern features. This is promising as far as we are able to develop optimal average-case algorithms for the one-dimensional version of the problem. For example, if we attempt to use the average case complexities given in the previous paragraph, the average complexity of the two-dimensional search stays the same.

Finally, it would be good to obtain an algorithm for (δ,γ)-matching that works for general alphabets, as the current one 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, 2003. To appear.
2. A. Apostolico. Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings. *Inf. Process. Lett.*, 23(2):63–69, 1986.
3. L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, 1992.
4. T. Crawford, C. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:71–100, 1998.
5. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd rev. edition, 2000.
6. 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.
7. K. Fredriksson, G. Navarro, and E. Ukkonen. *Faster than FFT: Rotation Invariant Combinatorial Template Matching*, volume II, pages 75–112. Transworld Research Network, 2002.
8. 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.
9. K. Fredriksson, G. Navarro, and E. Ukkonen. Sequential and indexed two-dimensional pattern matching allowing rotations. Technical Report TR/DCC-2003-2, Dept. of Computer Science, Univ. of Chile, May 2003.
10. 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.
11. K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. *Patt. Recog. Letters*, 20(11–13):1249–1258, 1999.

12. 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.
13. 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.
14. 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.
15. V. Mäkinen, G. Navarro, and E. Ukkonen. Algorithms for transposition invariant string matching. Technical Report TR/DCC-2002-5, Dept. of Computer Science, Univ. of Chile, July 2002.
16. V. Mäkinen, G. Navarro, and E. Ukkonen. Algorithms for transposition invariant string matching. In *Proc. 20th International Symposium on Theoretical Aspects of Computer Science (STACS 2003)*, LNCS 2607, pages 191–202, 2003.