Optimal Exact and Fast Approximate Two Dimensional Pattern Matching Allowing Rotations

Kimmo Fredriksson[1] *, Gonzalo Navarro[2] **, and Esko Ukkonen[1] ***

[1] Department of Computer Science, University of Helsinki.
Email: {kfredrik,ukkonen}@cs.helsinki.fi
[2] Department of Computer Science, University of Chile.
Eamil: gnavarro@dcc.uchile.cl

Abstract. We give fast filtering algorithms to search for a 2–dimensional pattern in a 2–dimensional text allowing any rotation of the pattern. We consider the cases of exact and approximate matching under several matching models, improving the previous results. For a text of size $n \times n$ characters and a pattern of size $m \times m$ characters, the exact matching takes average time $O(n^2 \log m/m^2)$, which is optimal. If we allow k mismatches of characters, then our best algorithm achieves $O(n^2 k \log m/m^2)$ average time, for reasonable k values. For large k, we obtain an $O(n^2 k^{3/2} \sqrt{\log m}/m)$ average time algorithm. We generalize the algorithms for the matching model where the sum of absolute differences between characters is at most k. Finally, we show how to make the algorithms optimal in the worst case, achieving the lower bound $\Omega(n^2 m^3)$.

1 Introduction

We consider the problem of finding the exact and approximate occurrences of a two–dimensional *pattern* of size $m \times m$ cells from a two–dimensional *text* of size $n \times n$ cells, when all possible rotations of the pattern are allowed. This problem is often called *rotation invariant template matching* in the signal processing literature. Template matching has numerous important applications in image and volume processing. The traditional approach [6] to the problem 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 2–dimensional (2D) case, and $O(m^3)$ in the 3D case, which makes the FFT approach very slow in practice. However, in many applications, "close enough" matches of the pattern are also accepted. To

* Work supported by ComBi and the Academy of Finland.
** Partially supported by the Millenium Center for Web Research.
*** Work supported by the Academy of Finland.

this end, the user may specify a parameter k, such that matches that have at most k differences with the pattern should be accepted.

Efficient two dimensional combinatorial pattern matching algorithms that do not allow rotations of the pattern can be found, e.g., in [5, 2, 4, 14]. Rotation invariant template matching was first considered from a combinatorial point of view in [10]. In this paper, we follow this combinatorial line of work. If we consider the pattern and text as regular grids, then defining the notion of matching becomes nontrivial when we rotate the pattern: since every pattern cell intersects several text cells and vice versa, it is not clear what should match what. Among the different matching models considered in previous work [10–12], we stick to the simplest one in this paper: (1) the geometric center of the pattern has to align with the center of a text cell; (2) the text cells involved in the match are those whose geometric centers are covered by the pattern; (3) each text cell involved in a match should match the value of the pattern cell that covers its center.

Under this *exact matching* model, an online algorithm is presented in [10] to search for a pattern allowing rotations in $O(n^2)$ average time.

The model (a 3D version) was extended in [12] such that there may be a limited number k of mismatches between the pattern and its occurrence. Under this *mismatches* model an $O(k^4 n^3)$ average time algorithm was obtained, as well as an $O(k^2 n^3)$ average time algorithm for computing the lower bound of the distance; here we will develop a 2D version whose running time is $O(k^{3/2} n^2)$. This works for any $0 \leq k < m^2$. For a small k, an $O(k^{1/2} n^2)$ average time algorithm was given in [9].

Finally, a more refined model [13, 9, 12] suitable for gray level images adds up the absolute values of the differences in the gray levels of the pattern and text cells supposed to match, and puts an upper limit k on this sum. Under this *gray levels* model average time $O((k/\sigma)^{3/2} n^2)$ is achieved, assuming that the cell values are uniformly distributed among σ gray levels. Similar algorithms for indexing are presented in [13].

In this paper we present fast filters for searching allowing rotations under these three models. Table 1 shows our main achievements (all are on the average). The time we obtain for exact searching is average-case optimal. For the k–mismatches model we present two different algorithms, based on searching for pattern pieces, either exactly or allowing less mismatches. For the gray levels model we present a filter based on coarsening the gray levels of the image, which makes the problem independent on the number of gray levels, with a complexity approaching that of the k–mismatches model.

2 Problem complexity

There exists a general lower bound for d–dimensional exact pattern matching. In [17] Yao showed that the one–dimensional string matching problem requires at least time $\Omega(n \log m/m)$ on average, where n and m are the lengths of the string and the pattern respectively. In [14] this result was generalized for the

Model	Previous result	Our results
Exact matching	$O(n^2)$	$O(n^2 \log_\sigma m/m^2)$
k Mismatches	$O(n^2 k^{3/2})$	$O(n^2 k \log_\sigma m/m^2)$, $k < m^2/(3\log_\sigma m)^2$
		$O(n^2 m^3/\sigma^{m/\sqrt{k}})$, $k < m^2/(5\log_\sigma m)$
		$O(n^2 k^{3/2}\sqrt{\log m}/m)$, $k < m^2\ (1-\Theta(1/\sigma))$
Gray levels	$O(n^2(k/\sigma)^{3/2})$	$O(n^2(k/\sigma)\log_\sigma m/m^2)$, $k < m^2\sigma/(9e\ln^2 m)$
		$O(n^2(k/\sigma)^{3/2}\sqrt{\log m}/m)$, $k < m^2\sigma/(5e\ln m)$

Table 1. The (simplified) average case complexities achieved for different models.

d–dimensional case, for which the lower bound is $\Omega(n^d \log m^d/m^d)$ (without rotations).

The above lower bound also holds for the case with rotations allowed, as exact pattern matching reduces (as a special case) to the matching with rotations. To search for P exactly, we search it allowing rotations and once we find an occurrence we verify whether or not the rotation angle is zero. Since in 2D there are $O(m^3)$ rotations [10], on average there are $O(n^2 m^3/\sigma^{m^2})$ occurrences. Each rotated occurrence can be verified in $O(1)$ average time (by the results of the present paper). Hence the total exact search time (et) is that of searching with rotations (rt) plus $O(n^2 m^3/\sigma^{m^2}) = o(n^2 \log_\sigma m/m^2)$ for verifications. Because of Yao's bound, $et = \Omega(n^2 \log_\sigma m/m^2) = rt + o(n^2 \log_\sigma m/m^2)$, and so $rt = \Omega(n^2 \log_\sigma m/m^2)$ as well. This argument can be easily generalized to d dimensions because there are $O(m^{O(d)}/d^m)$ matches to verify at $O(1)$ cost.

In Sec. 4 we give an algorithm whose expected running time matches this lower bound.

A lower bound for the k differences problem (approximate string matching with $\leq k$ mismatches, insertions or deletions of characters) was given in [7] for the one dimensional case. This bound is $\Omega(n(k + \log m)/m)$, where n is the length of the text string and m is the length of the pattern. This bound is tight; an algorithm achieving it was also given in [7].

This lower bound can be generalized to the d–dimensional case also. By [14], exact d–dimensional searching needs $\Omega(n^d \log m^d/m^d)$ time, and this is a special case of approximate matching. Following [7], we have that at least $k+1$ symbols of a window of the size of P in T have to be examined to guarantee that the window cannot match P. So a second lower bound is $\Omega(kn^d/m^d)$. The lower bound $\Omega(n^d(k + \log m^d)/m^d)$ follows.

3 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 ordered finite alphabet Σ. The size of the alphabet is denoted by $\sigma = |\Sigma|$. 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 (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 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.

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 $M(T[r,s]) = P[r',s']$. So 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 360 degrees. This result was shown to be also a lower bound in [3]. 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

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

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

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

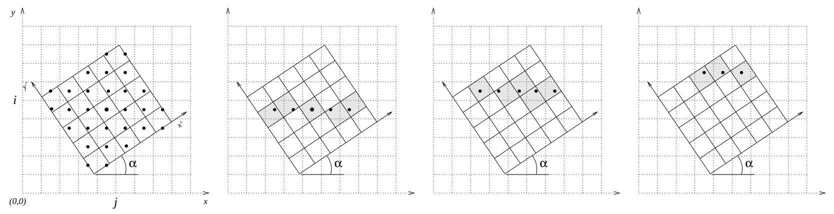

Fig. 1. Each text cell is matched against the pattern cell that covers the center of the text cell. For each angle θ, a set of features is read from P.

As shown in [10], any match of a pattern P in a text T allowing arbitrary rotations must contain a so-called "feature", i.e. a one–dimensional string obtained by reading a line of the pattern in some angle and crossing the center. These features are used to build a filter for finding the position and orientation of P in T.

We now define a *set* of linear features (strings) for P (see Figure 1). 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 the cell $T[i-\frac{m+1}{2}+q, j-\frac{u-1}{2}], \ldots, T[i-\frac{m+1}{2}+q, j+\frac{u-1}{2}]$. Denote them as $t^q_1, t^q_2, \ldots, t^q_u$. Let c^q_i be the value of the cell of P that covers the center of t^q_i. The (horizontal) feature of P with angle θ and row q is now the sequence $F^q(\theta) = c^q_1 c^q_2 \cdots c^q_u$. 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). 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 $\gamma_i \leq \theta < \gamma_{i+1}$. On the other hand, there are $O(|\mathcal{A}|/|\mathcal{B}^q)|$ angles $\beta \in \mathcal{A}$ such that $\gamma_i \leq \beta < \gamma_{i+1}$. If there is an occurrence of $F^q(\theta)$, then P may occur with such angles β.

4 Exact Search Allowing Rotations

In [10] only a set of features crossing the center of P and of length m is extracted from P, i.e. $q = \frac{m+1}{2}$ and $u = m$. The text is then scanned row–wise for the occurrence of some feature, and upon such an occurrence the whole pattern is checked at the appropriate angles.

The verification takes $O(m)$ time on average and $O(m^2)$ in the worst case in [10]. The reason is that there are $O(m^2)$ cells in the pattern, and each one intersects $O(m)$ different text centers along a rotation of 360 degrees (or any other constant angle), so there are $O(m^3)$ different rotations for the pattern. The number of relevant rotations for a feature of $O(m)$ cells is, however, only $O(m^2)$, and therefore there are $O(m)$ different angles in which the pattern has to be tested for each angle in which a feature is found.

In [11] the possibility of using features of length $u \leq m$ is considered, since it reduces the space and number of rotations. In what follows we assume that the features are of length $u \leq m$, and find later the optimal u.

We show now how to improve both search and verification time.

4.1 Faster Search

In [5] a 2–dimensional search algorithm (not allowing rotations) is proposed that works by searching for all the pattern rows in the image. Only every mth row of the image needs to be considered because one of them must contain some pattern row in any occurrence.

We take a similar approach. Instead of taking the $O(u^2)$ features that cross the center of the pattern, we also take some not crossing the center. More specifically, we take features for q in the range $\frac{m-r}{2}+1\ldots\frac{m+r}{2}$, where r is an odd integer for simplicity. For each such q, we read the features at all the relevant rotations. This is illustrated in Fig. 1. This allows us to search only one out of r image rows, but there are $O(rum)$ features now. Figure 1 also shows that the features may become shorter than m when they are far away from the center and the pattern is rotated. On the other hand, there is no need to take features farther away than $m/2$ from the center, since in the case of unrotated patterns this is the limit. Therefore we have the limit $r \le m$. If we take features from $r = m$ rows then the shortest ones (for the pattern rotated at 45 degrees) are of length $(\sqrt{2}-1)m = \Theta(m)$. The features do not cross the pattern center now, but they are still fixed if the pattern center matches a text center.

The search time per character is independent on the number of features if an Aho–Corasick machine (AC) [1] is used. Alternatively, we can use a suffix automaton (DAWG–MATCH algorithm) [8] to get an optimal average search time. The worst case time for the suffix automaton is the same as for the AC automaton.

4.2 Faster Verification

We show how verifications can be performed faster, in $O(1)$ time instead of $O(m)$. Imagine that a feature taken at angle θ has been found in the text. Since the feature has length u and can be at distance at most m from the center, there at most $O(um)$ different angles, whose limits we call γ_1 to γ_K, and we have $\gamma_i \le \theta < \gamma_{i+1}$.

We first try to extend the match of the feature to a match of the complete rotated row of the pattern. There are $O(m^2/(um))$ possible angles for the complete row, which lie between γ_i and γ_{i+1} (as the feature is enlarged, the matching angles are refined). However, we perform the comparison incrementally: first try to extend the feature by 1 cell. There are $O(((u+1)m)/(um)) = O((u+1)/u) = O(1)$ possible angles, and all them are tried. The probability that the $(u+1)$-th cell matches in some of the $O(1)$ permitted angles is $O(1/\sigma)$. Only if we succeed we try with the $(u+2)$-th cell, where there would be $O(((u+2)m)/((u+1)m))$ different angles, and so on.

In general, the probability of checking the $(u+i+1)$-th cell of the feature is that of passing the check for the $(u+1)$-th, then that of the $(u+2)$-th and so on. The average number of times it occurs is at most

$$\left(\frac{u+1}{u}\right)\frac{1}{\sigma} \times \left(\frac{u+2}{u+1}\right)\frac{1}{\sigma} \times \ldots \times \left(\frac{u+i}{u+i-1}\right)\frac{1}{\sigma} = \left(\frac{u+i}{u}\right)\frac{1}{\sigma^i}$$

and by summing for $i = 0$ to $\Theta(m) - u$ we obtain $O(1)$. This is done in both directions from the center, in any order.

The same scheme can be applied to extend the match to the rest of the pattern. Each time we add a new pattern position to the comparison we have only $O(1)$ different angles to test, and therefore an $O(1/\sigma)$ probability of success. The process is geometric and it finishes in $O(1)$ time on average.

Note that this result holds even if the cell values are not uniformly distributed in the range $1 \ldots \sigma$. It is enough that there is an independent nonzero probability p of mismatch between a random pattern cell and a random text cell, in which case $1/\sigma$ is replaced by $1 - p$.

4.3 Analysis

Using the suffix automaton the average search time is $O(n^2 \log_\sigma ru^2m/(r(u - \log_\sigma ru^2m)))$: there are $O(rum)$ features of length u, meaning that there are $O(ru^2m)$ suffixes to match, so the search enters to depth $O(\log_\sigma ru^2m)$ on average, we scan only every $O(r)$th row of the text, and the shift the automaton makes is on average $O(u - \log_\sigma ru^2m)$.

The verification time per feature that matches is $O(1)$ as explained, and there are $O(rum/\sigma^u)$ features matching each text position on average. This results in a total search cost

$$O\left(n^2 \frac{1}{r}\left(\frac{\log_\sigma ru^2m}{u - \log_\sigma ru^2m} + \frac{rum}{\sigma^u}\right)\right) = O\left(n^2\left(\frac{\log_\sigma ru^2m}{r(u - \log_\sigma ru^2m)} + \frac{um}{\sigma^u}\right)\right)$$

The optimum is at $r = u = \Theta(m)$, which leads to total average time

$$O(n^2(\log_\sigma m/m^2 + m^2/\sigma^m)) = O(n^2 \log_\sigma m/m^2).$$

which is optimal, so the exact matching problem can be solved in optimal average time $O(n^2 \log m/m^2)$. The space requirement of the suffix automaton is $O(m^4)$.

Again, this analysis is valid for non–uniformly distributed cell values, by replacing $1/\sigma$ by $1 - p$, where p is the probability of a mismatch.

5 Search Allowing Rotations and Mismatches

We first present a 2D version of the incremental algorithm of [12] that runs in $O(k^{3/2}n^2)$ average time, to search for a pattern in a text allowing rotations and at most k mismatches.

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 number of mismatches for successive rotations of P incrementally. That is, assume that the number of mismatches has been evaluated for β_i, then to evaluate the number of mismatches 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 number of mismatches 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.

Then consider the k mismatches problem. The expected number of mismatches in N tests is $Np = N\frac{\sigma-1}{\sigma}$. Requiring that $Np > k$ gives that about $N > k/p$ tests should be enough in typical cases to find out that the distance must be $> k$.

This suggests an improved algorithm for the k–mismatches case. Instead of using the whole P, select the smallest subpattern P' of P, with the same center cell, of size $m' \times m'$ such that $m' \times m' > k/p$. Then search for P' to find if it matches with at most k mismatches. If so, then check with the gradually growing subpatterns P'' whether P'' matches, until $P'' = P$. If not, continue with P' at the next location of T. The expected running time of the algorithm is $O(m'^3 n^2)$ which is $O(k^{3/2} n^2)$.

Note that this algorithm assumes nothing of how we compare the cell values, any other distance measure than counting the mismatches can be also used.

We show now how to improve this time complexity.

5.1 Reducing to Exact Searching

The idea is to reduce the problem to an exact search problem. We cut the pattern into j pieces along each dimension, for $j = \lfloor \sqrt{k} \rfloor + 1$, thus obtaining j^2 pieces of size $(m/j) \times (m/j)$. Now, in each match with k differences or less necessarily one of those pieces is preserved without differences, since otherwise there should be at least one difference in each piece, for a total of $j^2 = (\lfloor \sqrt{k} \rfloor + 1)^2 > k$ differences overall. This fact was first utilized in [15, 16]. So we search for all the j^2 pieces exactly and check each occurrence for a complete match.

Observe that this time the pieces cannot be searched for using the center to center assumption, because this holds only for the whole pattern. However, what is really necessary is not that the piece center is aligned to a text center, but just that there exists a fixed position to where the piece center is aligned. Once we fix a rotation for the whole pattern, the matching function of each pattern piece gets fixed too. Moreover, from the $O(m^3)$ relevant rotations for the whole pattern, only $O(mu)$ are relevant for each one-dimensional feature of length u. There is at most one matching function for each relevant rotation (otherwise we would have missed some relevant rotations). Hence we can work exactly as before when matching pieces, just keeping in mind that the alignment between the pattern center and the text center has to be shifted accordingly to the angle in which we are searching for the feature. The same considerations of the previous section show that we can do the verification of each matching piece in $O(1)$ time.

The search algorithm can look for all the features of all the j^2 patterns together. Since there are j^2 pieces of size $(m/j)^2$, there are $r = O(m/j)$ feature sets, which when considering all their rotations make up $O(j^2(m/j)mu) = O(jm^2u)$

features of length u that can match. So the suffix automaton takes time

$$O\left(n^2 \frac{\log_\sigma jm^2u^2}{\frac{m}{j}\left(\frac{m}{j} - \log_\sigma jm^2u^2\right)}\right).$$

The verification of the whole piece once each feature is found takes $O(1)$. Hence the total verification time is $O(n^2j^2(m/j)mu/\sigma^u)$. Note that for each piece of length $(m/j)^2$ there will be $O(m(m/j)^2)$ relevant rotations, because the piece may be far away from the pattern center.

Once an exact piece has been found (which happens with probability $O(m(m/j)^2/\sigma^{m^2/j^2}))$ we must check for the presence of the whole pattern with at most k differences. Although after comparing $O(k)$ cells we will obtain a mismatch on average, we have to check for all the possible rotations. A brute force checking of all the rotations gives $m^3/(m(m/j)^2) = j^2$ checks, for a total $O(kj^2)$ verification time for each piece found.

We can instead extend the valid rotations incrementally, by checking cells farther and farther away from the center and refining the relevant rotations at the same time. Unlike the case of exact searching, we cannot discard a rotation until k differences are found, but the match will disappear on average after we consider $O(k)$ extra cells at each rotation. Hence, we stop the verification long before reaching all the $O(m^3)$ rotations.

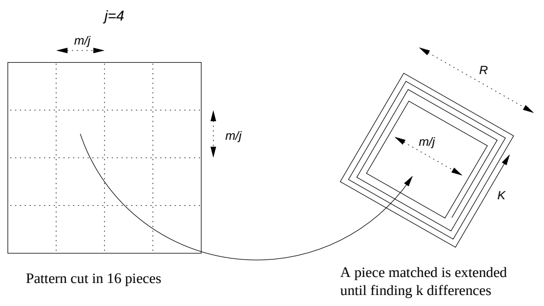

Fig. 2. On the left, the pattern is cut in $j^2 = 16$ pieces. On the right, a piece of width m/j found exactly is extended gradually until finding k differences.

Let K be a random variable counting the number of cells read until k differences are found along a fixed rotation. We know that $\overline{K} = O(k)$. Since we enlarge the match of the piece by reading cells at increasing distances from the center, by the point where we find k differences we will have covered a square of side R where $R^2 - (m/j)^2 = K$ (see Figure 2). The total number of rotations considered up to that point is $O(mR^2/(m(m/j)^2)) = O(1 + Kj^2/m^2)$. Since this is linear on K we can take the function on the expectation $\overline{K}$, so the average number of rotations considered until finding more than k differences is

$O(1 + kj^2/m^2)$. We consider that we check all these rotations by brute force, making $\overline{K} = O(k)$ comparisons for each such rotation. Then the verification cost per piece found is $O(k + k^2j^2/m^2)$. This verification has to be carried out $O(j^2m(m/j)^2n^2/\sigma^{m^2/j^2}) = O(m^3n^2/\sigma^{m^2/j^2})$ times on average. Therefore the total search time is of the order of

$$n^2 \left(\frac{\log_\sigma jm^2u^2}{\frac{m}{j}\left(\frac{m}{j} - \log_\sigma jm^2u^2\right)} + \frac{j^2(m/j)mu}{\sigma^u} + \frac{km^3}{\sigma^{m^2/j^2}} + \frac{k^2j^2m^3}{m^2\sigma^{m^2/j^2}} \right)$$

where all the terms increase with j. If we select $j = \Theta(\sqrt{k})$, and $u = \Theta(m/j) = \Theta(m/\sqrt{k})$, the cost is

$$n^2 \left(\frac{\log_\sigma m^4/\sqrt{k}}{\frac{m}{\sqrt{k}}\left(\frac{m}{\sqrt{k}} - \log_\sigma m^4/\sqrt{k}\right)} + \frac{m^3}{\sigma^{m/\sqrt{k}}} + \frac{km^3}{\sigma^{m^2/k}} + \frac{k^3m}{\sigma^{m^2/k}} \right)$$

The first term of the expression dominates for $k < m^2/(3\log_\sigma^2 m)$, up to where the whole scheme is $O(n^2k\log_\sigma m/m^2)$ sublinear time. After that point the whole scheme is $O(n^2m^3/\sigma^{m/\sqrt{k}})$ time for $k < m^2/(4\log_\sigma m)$, and $O(n^2k^3m/\sigma^{m^2/k})$ time for larger k.

5.2 Reducing to Approximate Searching

Since the search time worsens with j we may try to use a smaller j, although this time the pieces must be searched for allowing some differences. More specifically, we must allow $\lfloor k/j^2 \rfloor$ differences in the pieces, since if there are more than $\lfloor k/j^2 \rfloor$ differences per piece then the total exceeds k.

The $O(k^{3/2}n^2)$ time incremental search algorithm can be used here. Since we search for j^2 pieces with k/j^2 differences, the total search cost for the pieces is $O(n^2j^2(k/j^2)^{3/2}) = O(n^2k^{3/2}/j)$.

However, the incremental algorithm assumes that the center of P coincides with some center of the cells of T, and this is not necessarily true when searching for pieces. We now present a filter that gives a lower bound for the number of mismatches.

Assume that P is at some location $((u,v),\theta)$ on top of T, such that $(u,v) \in T[i,j]$ is not a center–to–center translation, and that the number of mismatches is k for that position of P. Then assume that P is translated to $((i,j),\theta)$, that is, center–to–center becomes true while the rotation angle stays the same. As a consequence, some cell centers of T may have moved to the cell of P that is one of its eight neighbors. Now compute the number of mismatches such that $T[r,s]$ is compared against $M(T[r,s])$ and its eight neighbors as well. If any of those nine cells match with $T[r,s]$, then we count a match, otherwise we count a mismatch. Let the number of mismatches obtained this way be k'.

This means that $k' \le k$, because all matches that contribute to $m^2 - k$ must be present in $m^2 - k'$ too. The value of k' can be evaluated with the incremental technique using β_s instead of θ where s is such that $\beta_s \leq \theta < \beta_{s+1}$, because the matching functions are the same for θ and β_s by our construction. Hence $k' \leq k$.

Hence we use the algorithm with the center–to–center assumption, but count a mismatch only when the text cells differs from all the 9 pattern cells that surround the one it matches with. The net result in efficiency is that the alphabet size becomes $\tau = 1/(1-1/\sigma)^9$, meaning that a cell matches with probability $1/\tau$.

For the verification cost of the pieces, we need to know the probability of a match with k differences. Since we can choose the mismatching positions and the rest must be equal to the pattern, the probability of a match is $\leq \binom{m^2}{k}/\tau^{m^2-k}$. By using Stirling's approximation to the factorial and calling $\alpha = k/m^2$, we have that the probability can be bounded by γ^{m^2}/m^2, where $\gamma = 1/(\alpha^{\alpha/(1-\alpha)}(1-\alpha)\tau)^{1-\alpha} \leq (e/((1-\alpha)\tau))^{1-\alpha}$. This improves as m grows and α stays constant. On the other hand, $\alpha < 1 - e/\tau$ is required so that $\gamma < 1$.

If we are searching for a piece of size $(m/j)^2$, then the matching probability is $O(\gamma^{(m/j)^2})/(m/j)^2$, which worsens as j grows. On the other hand, we have to multiply this probability by $j^2 m (m/j)^2 = m^3$ to account for all the rotations of all the pieces. Once a piece matches we check the complete match, which as explained in Section 5.1 takes $O(k + k^2 j^2/m^2)$ time. The total cost is

$$n^2 \left(\frac{k^{3/2}}{j} + \gamma^{m^2/j^2} m j^2 \left(k + \frac{k^2 j^2}{m^2} \right) \right) = n^2 k (\sqrt{k}/j + \gamma^{m^2/j^2} j^2 (m + k j^2/m))$$

whose optimum is $j = m/\sqrt{4 \log_{1/\gamma} m + 1/2 \log_{1/\gamma} k}(1 + o(1))$, which can be achieved whenever it is smaller than $\sqrt{k}$, i.e. for $k > m^2/(5 \log_\sigma m)(1 + o(1))$ (for smaller k the scheme reduces to exact searching and the previous technique applies). For this optimum value the complexity is $O(n^2 k^{3/2} \sqrt{\log_{1/\gamma} m/m})$.

This competes with the reduction to exact searching for high values of k. Reducing to approximate searching is indeed better for $k > m^2/(5 \log_\sigma m)$, i.e., wherever it can be applied. Recall that the scheme cannot be applied for $k > m^2(1 - e/\tau) = m^2(1 - \Theta(1/\sigma))$.

Figure 3 shows the complexities achieved.

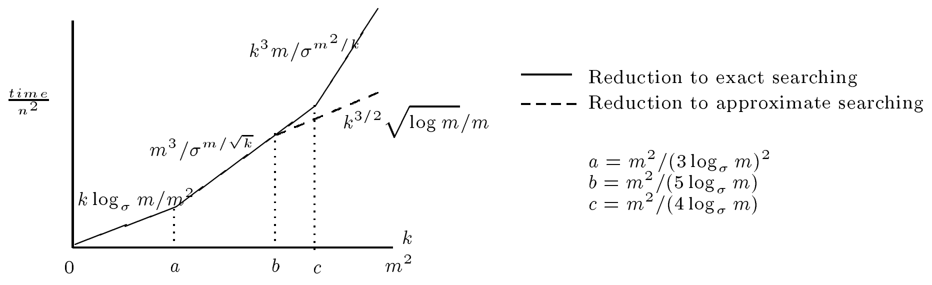

Fig. 3. The complexities obtained for the mismatches model depending on k.

6 Searching under the Gray Levels Model

In principle any result for the mismatches model holds for the gray levels model as well, because if a pattern matches with total color difference k then it also matches with k mismatches. However, the typical k values are much larger in this model, so using the same algorithms as filters is not effective if a naive approach is taken.

In this case, we can improve the search by reducing the number of different colors, i.e. mapping s consecutive colors into a single one. The effect is that σ is reduced to σ/s and k is reduced to $1 + \lfloor k/s \rfloor = \Theta(k/s)$ too. For instance, if we consider reduction to exact searching, the scheme is $O(n^2 k \log_\sigma m/m^2)$ time for $k < m^2/(3\log_\sigma^2 m)$. This becomes now $O(n^2 k/s \log_{\sigma/s} m/m^2)$ time for $k/s < m^2/(3\log_{\sigma/s}^2 m)$. For example binarizing the image means $s = \sigma/2$ and gives a search time of $O(n^2 k/\sigma \log m/m^2)$ for $k < m^2/(3\sigma/\log_2^2 m)$.

This seems to show that the best is to maximize s, but the price is that now we have to check the matches found for potential matches, because some may not really satisfy the matching criterion on the original gray levels. After a match with reduced alphabet is found we have to check for a real match, which costs $O(1)$ and occurs $O(n^2 m^3 \delta^{m^2}/m^2) = O(n^2 m \delta^{m^2})$ times, where $\delta = 1/(\beta^{\beta/(1-\beta)}(1-\beta)\sigma/s)^{1-\beta} \leq (e/((1-\beta)\sigma/s))^{1-\beta}$ and $\beta = \alpha/s = (k/s)/m^2$ (similar to γ in Section 5.2).

It is clear that this final verification is negligible as long as $\delta < 1$. The maximum s satisfying this is $(\sigma + \sqrt{\sigma^2 - 4e\sigma\alpha})/(2e) = \sigma/e(1+O(1/\sqrt{\sigma}))$. The search cost then becomes $O(n^2 k/\sigma \log m/m^2)$ for $k < m^2\sigma/(9e\ln^2 m)$. This means that if we double the number of gray levels and consequently double k, we can keep the same performance by doubling s.

For higher k values, partitioning into exact searching worsens if we divide k and σ by s, so the scheme is applicable only for $k < m^2\sigma/(9e\ln^2 m)$. However, it is possible to resort to reduction to approximate matching, using the $O((k/\sigma)^{3/2} n^2)$ average time algorithm for this model. This cost improves as we increase s, and hence we can obtain $O(n^2 (k/\sigma)^{3/2} \sqrt{\log_{1/\delta} m/m})$ time for $k < m^2\sigma/(5e\ln m)$.

7 Worst case optimal algorithms

In [3] it was shown that for the problem of the two dimensional pattern matching allowing rotations the worst case lower bound is $\Omega(n^2 m^3)$. Our efficient expected case algorithms above do not achieve this bound in the worst case. However, they are easily modified to do so. This can be done using the $O(m^3)$ time algorithm given in Sec. 5 for the verifications. Each time the filter suggests that there might be an occurrence in some position, we use the $O(m^3)$ time algorithm to verify it, if it is not verified before (which is possible because several features may suggest an occurrence at the same position). As each position is verified at most once, and the verification cost is $O(m^3)$, the total time is at most $O(n^2 m^3)$, which is optimal. This works for both the Hamming and gray levels cases. Moreover, this verification algorithm is very flexible, and can be adapted to many other distance functions.

8 Conclusions and Future Work

We have presented different alternatives to speed up the search for two dimensional patterns in two dimensional texts allowing rotations and differences.

The results can be extended to more dimensions. In three dimensions there are $O(m^{11})$ different rotations for P [12], and $O(um^2)$ features of length u. However, the three–dimensional text must be scanned in two directions, e.g. along the x–axis and along the y–axis, to find out the candidate rotations for P. Only if two features are found (that suggest the same center position of P in T), we enter the verification, see Figure 4. For the exact matching, the method works in $O(n^3 \log m/m^3)$ average time. The other results can be extended also.

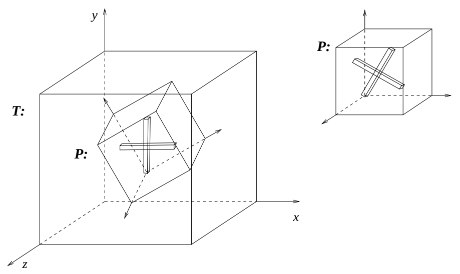

Fig. 4. Matching features in 3D.

It is also possible to make the verification probability lower by requiring that several pieces must match before going to the verification. This means smaller pieces or more differences allowed for the pieces. It is also possible to scan the text in two (in 2D) or in three (in 3D) ways instead of only one or two, using the same set of features than in the basic algorithm.

Note also that, until now, we have assumed that the center of P must be exactly on top of some center of the cells of T. It is also possible to remove this restriction, but the number of matching functions (and therefore the number of features) grows accordingly, see [12]. This, however, does not affect the filtering time, but the verification for the approximate matching would be slower.

Finally, we have considered an error model where only "substitutions" are permitted, i.e. a cell value changes its value in order to match another cell, so we substitute up to k values in the text occurrence and obtain the pattern. More sophisticated error models exist which permit displacements (such as inserting/deleting rows/columns) in the occurrences, and search algorithms for those models (albeit with no rotations) have been developed for two and more dimensions [4]. It would be interesting to combine the ability to manage those types of errors and rotations at the same time.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.
2. A. Amir, G. Benson, and M. Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM J. Comput.*, 23(2):313–323, 1994.
3. A. Amir, A. Butman, M. Crochemore, G.M. Landau, and M. Schaps. Two-dimensional pattern matching with rotations. Submitted for publication, 2002.
4. R. Baeza-Yates and G. Navarro. New models and algorithms for multidimensional approximate pattern matching. *J. Discret. Algorithms*, 1(1):21–49, 2000.
5. R. A. Baeza-Yates and M. Régnier. Fast two-dimensional pattern matching. *Inf. Process. Lett.*, 45(1):51–57, 1993.
6. L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, 1992.
7. W.I. Chang and T. Marr. Approximate string matching with local similarity. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 259–273, 1994.
8. M. Crochemore, A. Czumaj, L. Gąsieniec, T. Lecroq, W. Plandowski, and W. Rytter. Fast practical multi-pattern matching. *Inf. Process. Lett.*, 71(3–4):107–113, 1999.
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 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. J. Kärkkäinen and E. Ukkonen. Two- and higher-dimensional pattern matching in optimal expected time. *SIAM J. Comput.*, 29(2):571–589, 2000.
15. R. L. Rivest. Partial-match retrieval algorithms. *SIAM J. Comput.*, 5(1):19–50, 1976.
16. S. Wu and U. Manber. Fast text searching allowing errors. *Commun. ACM*, 35(10):83–91, 1992.
17. A. C. Yao. The complexity of pattern matching for a random string. *SIAM J. Comput.*, 8(3):368–387, 1979.