Fast Filters for Two Dimensional String Matching Allowing Rotations

Kimmo Fredriksson * Gonzalo Navarro † Esko Ukkonen *

March 3, 2000

Abstract

We give faster algorithms for searching a 2-dimensional pattern in a 2-dimensional text allowing rotations, mismatches and/or insertion/deletion errors.

1 Introduction

The problem of searching a two-dimensional pattern of size $m \times m$ in a two-dimensional text of size $n \times n$ when the pattern can appear in the text in rotated form was firstly addressed from a combinatorial point of view in [3]. They present an online algorithm for searching a pattern allowing rotations that takes $O(n^2)$ average time [5].

In [4] they extend the problem so that there may be a limited number (k) of mismatcehs between the pattern and its occurrence. They give an $O(k^{3/2}n^2)$ average time algorithm to find the occurrences. On the other hand, more general edit distances for two dimensions have been considered in [7], where insertions and deletions along rows or columns are handled, but they have not been combined with rotations.

In this paper we present fast filters of all these problems. Our main results are:

- We give an $O(n^2/m)$ search algorithm for the problem of searching allowing rotations (this raises to $O(n^2 \log(m)/m)$ for other exact matching models, see next section).
- We present a filter for searching allowing rotations and mismatches that is $O(n^2\sqrt{k}/m)$ if $k < m^2/(2\log_\sigma m)^2$. It still works better than the naive algorithm for higher error levels, although the complexity changes. The idea is to reduce the problem to exact searching of pieces of the pattern.
- The same is obtained for the more general model of searching allowing rotations, mismatches, insertions and deletions.

*Dept. of Computer Science, University of Helsinki.

†Dept. of Computer Science, University of Chile. Work developed while the author was in a postdoctoral stay at the Dept. of Computer Science, Univ. of Helsinki. Partially supported by the Academy of Finland and Fundación Andes.

- For the problem of rotations and mismatches we present another filter based on reducing the problem to inexact searching of pattern pieces. The filter is applicable for high error levels and in a range of k values it is better than the one based on exact searching.
- Finally, for one of the models of mismatches we present a filter based on coarsening the gray levels of the image, which makes the problem independent on the number of gray levels.

2 Matching Models

There are many different models for matching a rotated pattern considered in [3, 5, 6]. These are

A Each text cell involved should match the pattern cell that covers its center. The center of the pattern matches a text center.

B Idem A, but the center of the pattern needs not match a text center.

C Each pattern cell should match the text cell that covers its center. The center of the pattern matches a text center.

D Idem C, but the center of the pattern needs not match a text center.

E The text color must be between the minimal and maximal pattern colors among the 9 pattern cells surrounding that containing the text center. (The alternate model on 9 text cells has not been considered but it is very similar.)

F The pattern cell must match one of the text cells around that containing the pattern center. The pattern center does not need to match a text center.

We consider mainly the model A, and add comments for models B, E and F. The number of matching possibilities for a given pattern position is $O(m^3)$ in models A, C and F; $O(m^7)$ in models B and D; and $O(m)$ in model E.

3 Exact Search Allowing Rotations

As explained in [3], 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.

This property is used in [3] to build a filter for the search: one feature per angle is extracted and the text is scanned row-wise for the occurence of some feature, and upon such an occurrence the whole pattern is checked in the appropriate angle.

The verification takes $O(m)$ time on average and $O(m^2)$ in the worst case in [3]. 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). As shown in [3], the angles where those changes occur are truly different and therefore there are $O(m^3)$ different rotations for the pattern under model A. 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 [5] the possibility of using features of length $u \leq m$ is considered, since it reduces the space and number of rotations. In which 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.

3.1 Faster Search

Following [7, 2] we propose to search more features of the pattern to reduce the number of text rows to consider. This has obvious advantages since the search time per character is independent on the number of patterns if an Aho-Corasick machine (AC) [1] is used. In [2] a 2-dimensional search algorithm (not allowing rotations) is proposed by searching all the pattern rows in the text, so that only the text rows multiples of m need to be considered because one of them must contain some pattern row in any occurrence.

We propose the same now. Instead of taking the $O(u^2)$ features that cross the center of the pattern, we also take those not crossing the center. For each angle, we take r features at consecutive (rotated) pattern rows. Figure 1 illustrates. This allows us to search only one out of r text rows, but there are $O(ru^2)$ 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 from $m/2$ from the center, since in the case of unrotated patterns this is the limit. Therefore we have the limit $r \leq m$. If we take m features per angle then the shortest ones (for the pattern rotated at 45 degrees) are of length $(\sqrt{2}-1)m = \Theta(m)$.

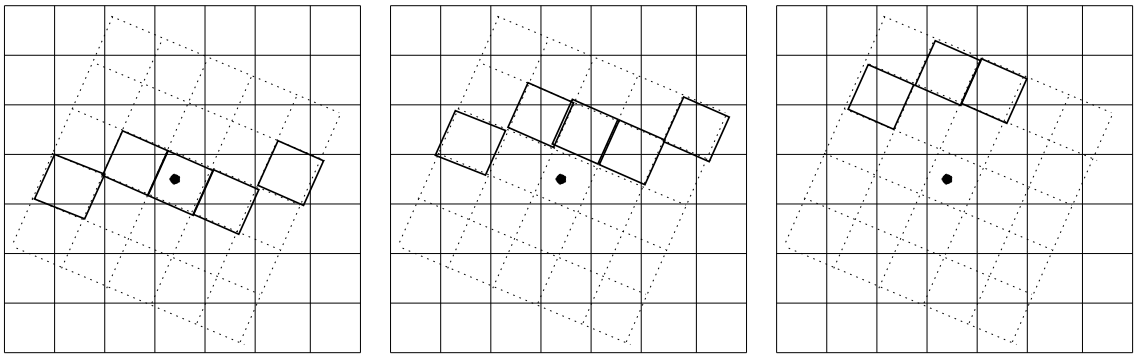

Figure 1: The pattern (dotted) is read from the text at a given angle. We not only extract the feature that crosses its center (left plot) but also others (centeral and right plot).

The features do not cross the pattern center now, but they are still fixed if the pattern center matches a text center.

3.2 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 α has been found in the text. Since the feature has length u, there are only

$O(u^2)$ different angles, whose limits we call γ_1 to γ_{u^2}, and we have $\gamma_i \leq \alpha < \gamma_{i+1}$.

We first try to extend the match of the feature to a match of the complete rotated row of the pattern. There are $O(m^2/u^2)$ 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 character. There are $O((u+1)^2/u^2) = O(1)$ possible angles, and all them are tried. The probability that the $(u+1)$-th character matches in some of the $O(1)$ permitted angles is $O(1/\sigma)$. Only if we succeed we try with the $(u+2)$-th character, where there would be $O((u+2)^2/(u+1)^2)$ different angles, and so on.

In general, the probability of checking the $(u+i+1)$-th character 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

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

and by summing for $i = 0$ to $(m+u)/2$ we obtain $O(1)$. This is done in both directions from the center, which is still $O(1)$.

The same scheme can be applied 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.

3.3 Analysis

The search time for the features is $O(n^2/r)$ since we inspect one text row out of r and the cost per inspected text character is constant. The verification time per feature that matches is $O(1)$ as explained, and there are $O(ru^2/\sigma^u)$ features matching each inspected text position on average. This means that the total search cost is $O(n^2/r(1+ru^2/\sigma^u)) = O(n^2(1/r + u^2/\sigma^u))$. This shows that the optimum is $r = u = \Theta(m)$ for a total search cost of $O(n^2/m)$. However, the same complexity is achieved if $u = x \log_\sigma m$ with $x > 1$, which may be preferable because much less space is necessary to store the features.

Under model B, there are not $O(m^3)$ different matches at the same center positions, but $O(m^7)$. The mechanism of incrementally testing the rotations can be extended to account for center displacement too, still keeping its $O(1)$ complexity because the number of choices increases polynomially with m. The rest of the analysis is also the same since we still have to optimize $poly(u)/\sigma^u$.

If we use the model E, then it is shown in [5] that a trie of color ranges can be built which is used similarly to an Aho-Corasick machine, and the complexity would be the same as for the restricted model. On the other hand, they show that if the failure links are not included then the automaton can be compacted into a DAG, which takes less space, but the search time becomes $O(\log m)$ operations per character (since one has to enter into the trie of $poly(m)$ features again for each text position). The analysis in this case is similar: there are $O(u)$ rotations, one verification costs $O(1)$, and there are $O(ru)$ features to check, and the search for the features takes $O(n^2 \log_{\sigma'}(ru)/r)$. Here $\sigma' < \sigma$ should be taken as the inverse probability that a random cell falls into the range of colors of 9 neighbors at a random text position.

The cost under model E becomes $O(n^2(\log_{\sigma'}(ru)/r + u/\sigma'^u))$, which shows that the maximal $r = \Theta(m)$ is optimal. Again, any $u > \log_{\sigma'} m$ is optimal, and using the optimal setting the search

cost becomes $O(n^2 \log_{\sigma'}(m)/m)$.

This is also the cost under the F model, because one cannot use AC but can use a trie of features and test it at each text position. Since on average one enters up to depth $O(\log_{\sigma'} m)$ in the trie, the complexity follows. The result is the same even when there are $O(ru^2)$ features and $O(u^2)$ rotations per feature.

4 Search Allowing Rotations and Mismatches

There are two possible models for this case. In M1 we count the total number of cells that do not match, while in M2 we count the sum of the absolute values of the color differences.

In [4] an $O(k^{3/2}n^2)$ average time algorithm is presented to search a pattern in a text allowing rotations and at most k mismatches under model M1. We show now how to improve this time complexity and how to cope with M2.

In principle any result for M1 holds for M2 because if a pattern matches with k errors in model M2 it also matches with k errors in model M1. However, the typical k values are much larger in M2, so using the same algorithms as filters is not effective if a naive approach is taken.

4.1 Reducing to Exact Partitioning

The idea is to reduce the problem to an exact search problem. We cut the pattern in 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 errors or less necessarily one of those pieces is preserved without errors. So we search for all the j^2 pieces and check each occurrence for a complete match.

The search algorithm can look for all the features of all the j^2 patterns together, so the search time has two parts: the AC machine takes $O(n^2/(m/j))$ time (since the pieces are of $((m/j)^2)$ size); and the verification of the whole piece once each feature is found takes $O(1)$. Since there are j^2 pieces of size $(m/j)^2$, there are $j^2(m/j)u^2$ features that can match, and the total verification time is $O(n^2 j^2 (m/j) u^2/\sigma^u)$.

Once an exact piece has been found (which happens with probability $O((m/j)^3/\sigma^{m^2/j^2})$) we must check for the presence of the whole pattern with at most k errors. Although after comparing $O(k)$ characters 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 $m^3/(m/j)^3 = j^3$ checks, for a total $O(kj^3)$ verification time. We can incrementally check the valid rotations, but unlike the case of exact searching, we cannot discard a rotation until k errors are made. However, if we enlarge the match by checking points farther and farther from the center of the exact match, on average the match disappears when we consider $O(k)$ extra characters at each rotation, which means a square of radius r where $r^2 - (m/j)^2 = O(k)$. The total number of rotations considered up to that point is $r^3/(m/j)^3 \leq (\sqrt{k}j/m)^3$. Hence, we consider that we check all rotations by brute force up to this point, and hence $O(k)$ comparisons are made for each such rotation. Then the verification cost per piece is $O(k^{5/2}/(m/j)^3)$. This verification has to be carried out $O(j^2(m/j)^3 n^2/\sigma^{m^2/j^2})$ times on average. Therefore the total search time is of the order of

$$n^2 \left(\frac{j}{m} + \frac{j^2(m/j)u^2}{\sigma^u} + \frac{k^{5/2}j^2(m/j)^3}{(m/j)^3\sigma^{m^2/j^2}} \right) = n^2 \left(\frac{j}{m} + \frac{jmu^2}{\sigma^u} + \frac{k^{5/2}j^2}{\sigma^{m^2/j^2}} \right)$$

where all the terms worsen as j grows. This is why we prefer to take the minimum possible $j = \Theta(\sqrt{k})$. Any $u \geq x \log_\sigma m$ for $x > 2$ yields optimal performance for u. The first term of the expression dominates while the optimal u is feasible, i.e. for $k \leq m^2/(4 \log_\sigma^2 m)(1 + o(1))$, up to where the whole scheme is $O(n^2\sqrt{k}/m)$ time. After that point we have to select maximal $u = m/j$ and the whole scheme is $O(n^2 m^3/(\sqrt{k}\sigma^{m/\sqrt{k}}))$ time for $k \leq m^2/(5 \log_\sigma m)(1 + o(1))$. Finally, the scheme is $O(n^2 k^{7/2}/\sigma^{m^2/k})$ for larger k.

Under model B we can also check points farther and farther from the center, although this time the cost grows as $O(r^7)$ for radius r. Hence the verification cost per position is $O(j^2 k^{9/2}/(m/j)^7)$. Therefore the total cost now is $O(n^2(j/m + j^2(m/j)u^7/\sigma^u + k^{9/2}j^2/\sigma^{m^2/k})$. All the complexities are similar (that of the first term is the same) and the cut points differ only in constant factors.

In the models E and F all the terms and cut points are similar, but the first term of the search time is $O(n^2\sqrt{k}\log(m/\sqrt{k})/m)$.

4.2 Reducing to Inexact Partitioning

Since the search time worsens with j we may try to use a smaller j, although this time the pieces must be searched allowing some errors. More specifically, we must allow $\lfloor k/j^2 \rfloor$ errors in the pieces.

In [4] an $O(k^{3/2}n^2)$ search algorithm is given to search with k errors. Since we search j^2 pieces with k/j^2 errors, the total search cost for the pieces is $O(n^2 j^2 (k/j^2)^{3/2}) = O(n^2 k^{3/2}/j)$.

For the verification cost of the pieces, we need to know the probability of a match with k errors. Since we can choose the mismatching positions and the rest must be equal to the pattern, the probability of a match is $\leq \binom{m^2}{k}/\sigma^{m^2-k}$, which has to be multiplied by m^3 to account for rotations. By using Stirling's approximation to the factorial and calling $\alpha = k/m^2$, we have that the probability can be bounded by $\gamma^{m^2} m^3$, where $\gamma = 1/(\alpha^{\alpha/(1-\alpha)}(1-\alpha)\sigma)^{1-\alpha} \leq (e/((1-\alpha)\sigma)^{1-\alpha}$. This improves as m grows and α stays constant. On the other hand, $\alpha < 1 - e/\sigma$ is required so that $\gamma < 1$.

Once a piece matches we check the complete match, which as explained before takes $O(k^{5/2}/(m/j)^3)$.

In our partitioning method α stays constant, which means that the verification time worsens as j grows, since m is replaced by m/j in the formula of the matching probability. The search time, on the other hand, improves with j. The total cost is

$$n^2 \left(\frac{k^{3/2}}{j} + j^2 \gamma^{m^2/j^2} (m/j)^3 \frac{k^{5/2}}{(m/j)^3} \right) = n^2 k^{3/2} (1/j + j^2 k \gamma^{m^2/j^2})$$

whose optimum is $j = \Theta(m/\sqrt{\log_{1/\gamma} m})$, that 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 $O(n^2 k^{3/2}\sqrt{\log_\sigma m}/m)$.

This competes where the area of reducing to exact searching where the third term dominates, so we would like to compare it against that third term. Reducing to inexact partitioning is indeed better for $k > m^2/(3 \log_\sigma m)(1 + o(1))$. The results should be quite similar with the other models.

Under the model M2 all these complexities also hold by changing k by $3k/\sigma$. This is because the expected difference between two random pixel values is $\sigma/3$.

4.3 Reducing the Resolution

As explained, the problem of using our algorithms to reduce the number of errors to the M2 model is that the number of errors allowed k may be too high, to account reasonably for differences in absolute values of gray levels. 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. In this case 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 partitioning, the scheme is $O(n^2\sqrt{k}/m)$ time for $k < m^2/(4\log_\sigma^2 m)$. This becomes now $O(n^2\sqrt{k/s}/m)$ time for $k/s < \Theta(m^2/\log_{\sigma/s}^2 m)$. For example binarizing the image means $s = \sigma/2$ and gives a search time of $O(n^2\sqrt{k/\sigma}/m)$ for $k < \Theta((m/\log_2 m)^2\sigma)$.

This seems to show that the best is to maximize s, but the price is that we now 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 $O(m^2)$ and occurs $O(n^2m^3\gamma'^{m^2})$ times, where γ' is the same as γ where σ is replaced by σ/s.

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

The same should happen with inexact partitioning and models B, E and F.

5 Search Allowing Rotations and Differences

The model allowing not only differences but also insertions and deletions along rows or columns [7] can be extended to permit rigid rotations as well. Without rotations, the pattern can be searched in $O(m^4n^2)$ time using dynamic programming. A naive extension which runs this algorithm using all the $O(m^3)$ relevant angles takes $O(m^7n^2)$.

However, as shown in [7], we can reduce the problem to exact searching too by cutting the pattern into $j \times j$ pieces so that at least one is unaltered in any approximate occurrence. This is exactly as in Section 4.1, only the verification changes. In particular, we have not devised an incremental test under this model. Redoing the analysis made before yields

$$n^2 \left(\frac{j}{m} + \frac{jmu^2}{\sigma^u} + \frac{j^2m^7}{\sigma^{m^2/j^2}} \right)$$

where now the second term dominates for $k < m^2/(28\log_\sigma m)(1 + o(1))$.

The models B, E and F should be similar.

6 Conclusions and Future Work

We have presented different alternatives to speed up the search of two dimensional patterns in two dimensional texts allowing rotations and errors. The results can be extended to more dimensions. Figure 2 shows the main results obtained.

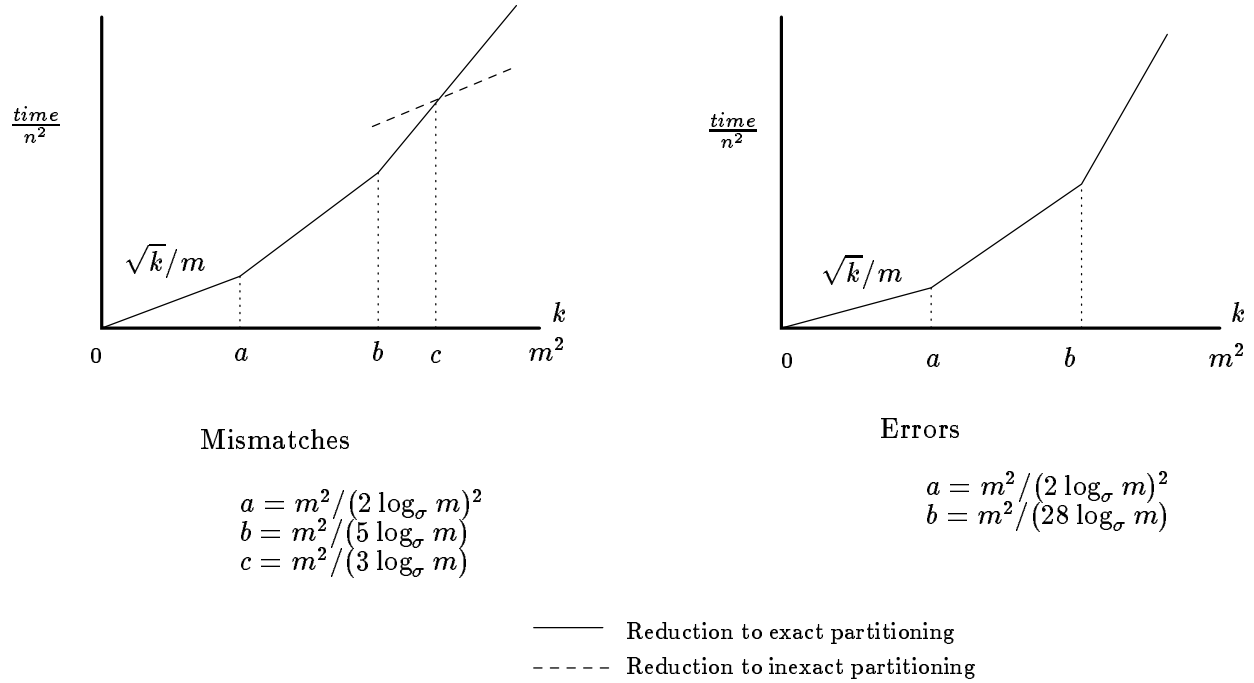

Figure 2: The complexities obtained depending on k, for the model that allows only mismatches and rotations (left) and for the model that allows differences and rotations (right).

References

[1] A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *CACM*, 18(6):333–340, June 1975.

[2] R. Baeza-Yates and M. Régnier. Fast two dimensional pattern matching. *Information Processing Letters*, 45:51–57, 1993.

[3] K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. CPM'98*, LNCS 1448, pages 118–125, 1998.

[4] K. Fredriksson and E. Ukkonen. Algorithms for 2-d Hamming distance under rotations. 1999.

[5] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. 1999.

[6] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. 1999.

[7] G. Navarro and R. Baeza-Yates. A new indexing method for approximate string matching. In *Proc. CPM'99*, LNCS 1645, pages 163–185, 1999.