Faster Two Dimensional Pattern Matching with Rotations

Amihood Amir* Oren Kapah† Dekel Tsur‡

Abstract

The most efficient currently known algorithms for two dimensional pattern matching with rotations have a worst case time complexity of $O(n^2m^3)$, where the size of the text is $n \times n$ and the size of the pattern is $m \times m$. In this paper we present a new algorithm for the problem whose running time is $O(n^2m^2)$.

Key Words: Design and analysis of algorithms, two dimensional pattern matching, rotation.

1 Introduction

One of the main motivation for research in two dimensional pattern matching is the problem of searching aerial photographs. The problem is a basic one in computer vision, but it was felt that pattern matching can not be of use in its solution. Such feelings were based on the belief that pattern matching algorithms are only good for *exact matching* whereas in reality one seldom expects to find an exact match of the pattern. Rather, it is interesting to find all text locations that "approximately" match the pattern. The types of differences that make up these "approximations" in the aerial photograph case are: (1) Local Errors — introduced by differences in the digitization process, noise, and occlusion (the pattern partly obscured by another object). (2) Scale — size difference between the image in the pattern and the text, and (3) Rotation — angle differences between the images. Progress has been made on local errors and scaling (e.g. [3, 5, 7, 10, 17, 18, 4]), but rotation had proven challenging.

The *two dimensional pattern matching with rotations problem* (or *rotated matching* for short) is that of finding all occurrences of a two dimensional pattern in a text, in all possible rotations. An efficient solution to this problem proved elusive even though many researchers were thinking about it for over a decade. Part of the problem was lack of a rigorous definition to capture the concept of rotation in a discrete pattern.

The major breakthrough came when Fredriksson and Ukkonen [14] gave an excellent combinatorial definition of rotation based on geometric interpretation of the text and pattern. A slightly different definition of rotation was given in Fredriksson et al. [12]. These two definitions give rise to the "pattern over text" and "pattern under text" models, and will be described in Section 2.

Most of the algorithms for rotated matching are filtering algorithms that behave well on average but have bad worst case complexity (e.g. [14, 12, 15]). In two papers [13, 2] there is an $O(n^2m^3)$ worst case algorithm for rotated matching, where the size of the text is $n \times n$ and the size of the pattern is $m \times m$. Amir et al. [2] also give a lower bound of $\Omega(n^2m^3)$ on the output size (the lower bound applies to both models of rotation).

In this paper we present the first algorithm for rotated matching whose time complexity is better than $O(n^2m^3)$. We give an algorithm for the "pattern under text" model whose time complexity is $O(n^2m^2)$.

*Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel and Georgia Tech. `amir@cs.biu.ac.il`.
Partly supported by NSF grant CCR-01-04494 and ISF grant 282/01.
†Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel. `kapaho@cs.biu.ac.il`.
‡Caesarea Rothschild Institute of Computer Science, University of Haifa, Haifa 31905, Israel. `dekelts@cs.haifa.ac.il`.

The space complexity of the algorithm is $O(m^4)$. We are able to "break" the lower bound on the output size by showing using geometric considerations that many of the rotated pattern occurrences are consecutive and thus need not be explicitly written.

The result above assumes that the alphabet is $\{1, \ldots, n^2\}$. The case of an arbitrary alphabet can be handled by first sorting the characters of the text and the pattern, and then replacing each character of by a number from $\{1, \ldots, n^2\}$. In the general case, the time complexity increases by $O(n^2 \log n)$.

Related work Bird [9] and Baker [8] independently gave an $O(n^2 \log \sigma)$-time algorithm for the standard two dimensional matching problem, where σ is the size of the alphabet. An $O(n^2)$ time algorithm is obtained by combining the results of [1] and [16].

The two dimensional matching with scaling problem can be solved in $O(n^2)$ time [6], when the allowed scales are integers. For the case of real scales, Amir et al. [4] give an algorithm whose time complexity is $O(nm^3 + n^2 m \log m)$.

2 Preliminaries

In this section we define the two models of rotation. We first describe the "pattern under text" model, which is the model that we use for our algorithm. Then, we describe the "pattern over text" model, and compare the two models.

Let T be a two-dimensional string of size $n \times n$ over some finite alphabet Σ. The array of *unit pixels* for T consists of n^2 unit squares, called *pixels* in the real plane $\mathbb{R}^2$. The pixel that corresponds to the character $T[i, j]$ is the pixel whose corners are are $(i-1, j-1)$, $(i, j-1)$, $(i-1, j)$, and (i, j). Hence the pixels for T form an $n \times n$ array covering the area between $(0, 0)$, $(n, 0)$, $(0, n)$, and (n, n). We will use $[i, j]$ to denote the pixel that corresponds to $T[i, j]$. The *center* of a pixel $[i, j]$ is the geometric center point of the pixel, namely the point $(i - 0.5, j - 0.5)$. The *color* of the pixel $[i, j]$ is the value of $T[i, j]$. We also say that the center of $[i, j]$ has the color $T[i, j]$. See figure 1 for an example of these definitions.

The array of pixels for the pattern P is defined similarly. A different treatment is necessary for patterns with odd sizes and for patterns with even sizes. For simplicity's sake we assume throughout the rest of this paper that the pattern is of size $m \times m$ and m is even. The *rotation pivot* of the pattern array is its exact center, namely the point $(\frac{m}{2}, \frac{m}{2}) \in \mathbb{R}^2$.

Consider now a rigid motion (translation and rotation) that moves the pixel array of P with respect to the the pixel array of T. Consider the special case where the translation moves the rotation pivot of P to a point $(i, j) \in \mathbb{R}^2$, where i and j are integers. The pattern array is now rotated counterclockwise, centered at (i, j), creating an angle α between the x-axes of the pixel arrays of T and P. We say that the array of P is at *location* $((i, j), \alpha)$. There is an *occurrence of P at location* $((i, j), \alpha)$ if the center of each pixel in T has the same color as the pixel of P under it, if there is such a pixel. When the center of a text pixel is exactly over a vertical (horizontal) border between text pixels, the color of the pattern pixel left (below) to the border is chosen. An example for this definition is given in Figure 1.

Consider some occurrence of P at location $((i, j), \alpha)$. This occurrence defines a non-rectangular substring of T that consists of all the characters of T that correspond to text pixels whose centers are inside the pattern pixels. We call this string P *rotated by* α, and denote it by P^α. Note that there is an occurrence of P at location $((i, j), \alpha)$ if and only if P^α occurs at location (i, j) of T.

In the "pattern over text" model, we say that there is an occurrence of P at location $((i, j), \alpha)$ if the center of each pixel in the rotated pixel array of P, has the same color as the pixel of T under it (see Figure 2). While the two models of rotation seem to be quite similar, they are not identical. For example, in the "pattern over

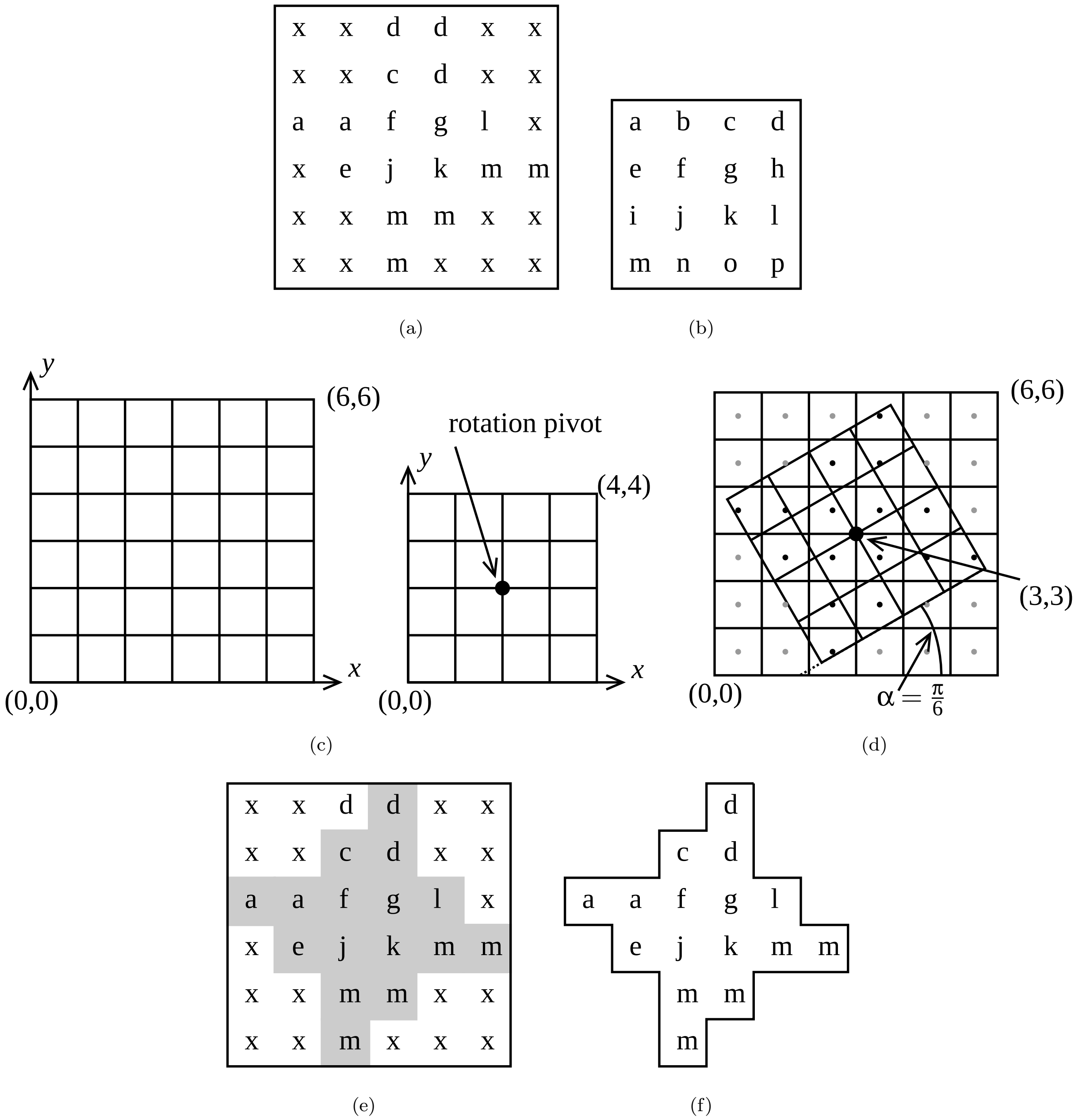
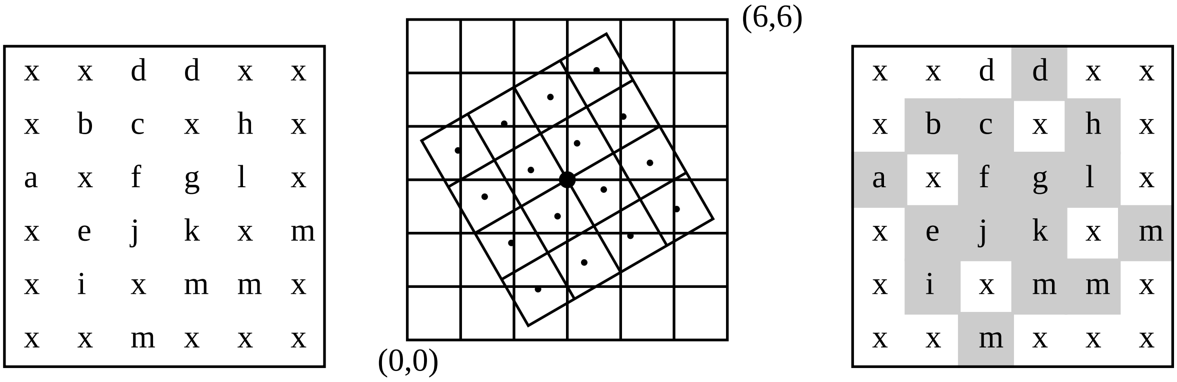


Figure 1: An example for the definition of rotation in the “pattern under text” model. The text T and pattern P are given in Figures (a) and (b), respectively. Figure (c) shows the pixel arrays of T and P. In Figure (d), the pattern array is at location $((3,3),\pi/6)$. The pixel centers of T that fall inside pixels of P are drawn in black, while the rest of the pixel centers of T are drawn in gray. Figure (e) shows the occurrence of the pattern P at location $((3,3),\pi/6)$. The pattern $P^{\pi/6}$ is shown in Figure (f).



(a) (b) (c)

Figure 2: An example for the definition of rotation in the "pattern over text" model. The text T is given in Figure (a), and the pattern P is the same as in Figure 1. An occurrence of P at location $((i,j),\pi/6)$ is shown in Figures (b) and (c). Note that the text pixel $[5,4]$ does not contain a pattern pixel center.

text" model there are angles for which two pattern pixel centers are over the same text pixel. Alternately, there are angles where there are "holes" in the rotated pattern, namely there is a text pixel that does not have in it a center of a pattern pixel, but all text pixels around it have centers of pattern pixels. See Figure 2 for an example.

The challenges of "discretizing" a continuous image are not simple. In the Image Processing field, stochastic and probabilistic tools need to be used because the images are "smoothed" to compensate for the fact that the image is presented in a far coarser granularity than in reality. The aim of the pattern matching community has been to fully discretize the process, thus our different definitions. However, this puts us in a situation where some "gut" decisions need to be made regarding the model that best represents "reality". It is our feeling that in this context the "pattern under text" model is more intuitive since it does not allow anomalies such as having two pattern centers over the same text pixel (a contradiction) nor does it create "holes" in the rotated pattern. We note that the algorithm we present in this paper cannot be applied to the "pattern over text" model due to the "holes".

3 The New Algorithm

The main idea of the algorithm is to partition the pattern into two parts: an *inner part* and an *outer part*. By carefully choosing the partition, we will be able to efficiently search for occurrences of each part in T, and then combine the results of the two searches. More precisely, our partition has the following properties:

1. There are $O(m)$ different shapes of the inner part. The number of different shapes is smaller by a factor of $O(m)$ than the number of shapes in all the rotations of the pattern. This fact will allow us to reduce the time complexity for matching the inner parts by a factor of $O(m)$.
2. Each outer part contains $O(m)$ characters. Again, since the number of characters is smaller by a factor of $O(m)$ than the number of characters in the pattern, we obtain an $O(m)$ speedup.

Let B_0 be the square in $\mathbb{R}^2$ whose bottom corners are $(0,0)$, $(0,m)$, (m,m), and $(m,0)$. For a positive integer i, let B_i denote the square whose corners are (i,i), $(i,m-i)$, $(m-i,m-i)$, and $(m-i,i)$. We denote by B_i^α the square obtained by rotating B_i counterclockwise by an angle of α with rotation center at $(m/2,m/2)$. Recall that we assume that m is even.

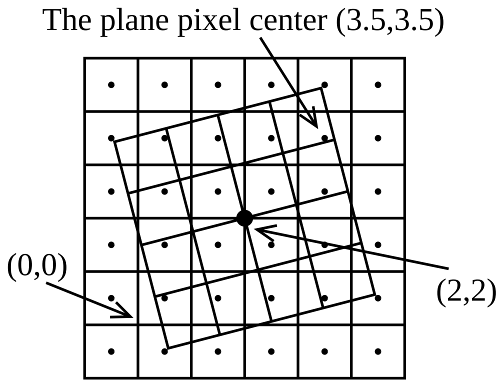


Figure 3: Example showing the angle α_1 for a pattern of size 4×4. The plane pixel center $(3.5, 3.5)$ was inside the pixel $[4, 4]$ of the pattern pixel array rotated by $\alpha_1 - \epsilon$, while it is inside the pixel $[4, 3]$ of the pattern pixel array rotated by α.

For the following definitions, consider the pixel array of P, and an infinite array of pixels that covers the plane, which will be called the *plane pixel array.* Consider a continuous rotation of the array of P counterclockwise. During this rotation, 3 types of events occur:

1. A plane pixel center which was inside the pixel $[i, j]$ of the pattern pixel array rotated by $\alpha - \epsilon$ (for all small enough $\epsilon > 0$), is inside the pixel $[i', j'] \neq [i, j]$ of the pattern pixel array rotated by α.
2. A plane pixel center which was outside the square $B_0^{\alpha-\epsilon}$ (in other words, the center was not inside any pixel of the pattern pixel array rotated by $\alpha - \epsilon$), is inside the square B_0^{α}.
3. A plane pixel center which was inside the square $B_0^{\alpha-\epsilon}$, is outside the square B_0^{α}.

Let $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k < 2\pi$ be the angles in which these events occur (see Figure 3). We assume that in each α_i, only one event occurs for a single plane pixel center (when several events occur simultaneously, we assign them into angles $\alpha_i, \alpha_{i+1}, \ldots$ such that $\alpha_i = \alpha_{i+1} = \cdots$). We also define $\alpha_0 = 0$.

Definition 1 (outline)**.** *Let $i_0 = 0$. We define indices $i_1, i_2, \ldots, i_{l-1}$ and sets $I_1, I_2, \ldots, I_l$ as follows. For $j \geq 1$, the* outline *I_j is the set of all plane pixels whose centers are inside $B_1^{\alpha_{i_{j-1}}}$. For $j \geq 1$, i_j is the minimum index such that there is a pixel in I_j whose center is outside the square $B_0^{\alpha_{i_j}}$, if such index exists. Finally, l is the minimum value such that i_l is not defined.*

Examples for Definition 1 are shown in Figure 4.

Definition 2 (inner and outer parts)**.** *Let $s \leq k$ be some index, and let j be the index such that $i_{j-1} \leq s < i_j$. The* inner part *of P^{α_s} is the string that consists of all the characters $P[a, b]$ such that $[a, b] \in I_j$. The* outer part *of P^{α_s} consists of all the characters of P^{α_s} that are not in the inner part.*

Examples for Definition 2 are given in Figure 5.

An important property of the partition above is that there are $O(m)$ different outlines. This is proved in the following lemma.

Lemma 1. *For every j, $1 \leq j \leq l$, $\alpha_{i_j} - \alpha_{i_{j-1}} = \Theta(1/m)$.*

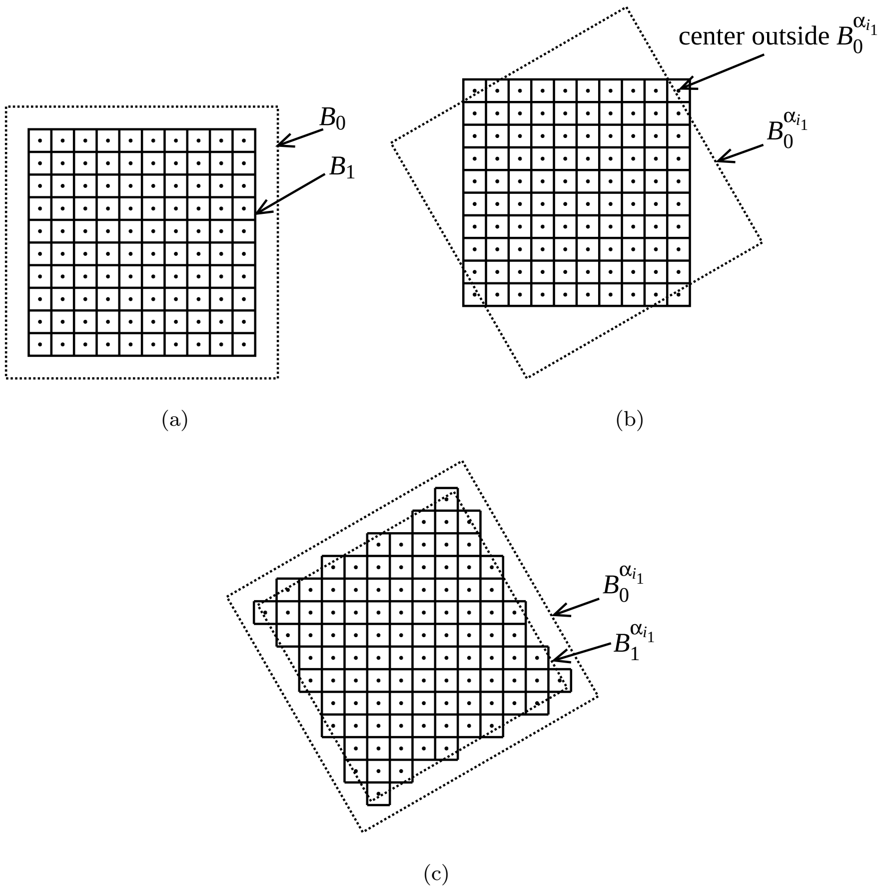


Figure 4: Examples for the definition of outline for a pattern of size 12×12. Figure (a) shows the outline I_1. Figure (b) illustrates the definition of the index i_1. The outline I_2 is shown in Figure (c).

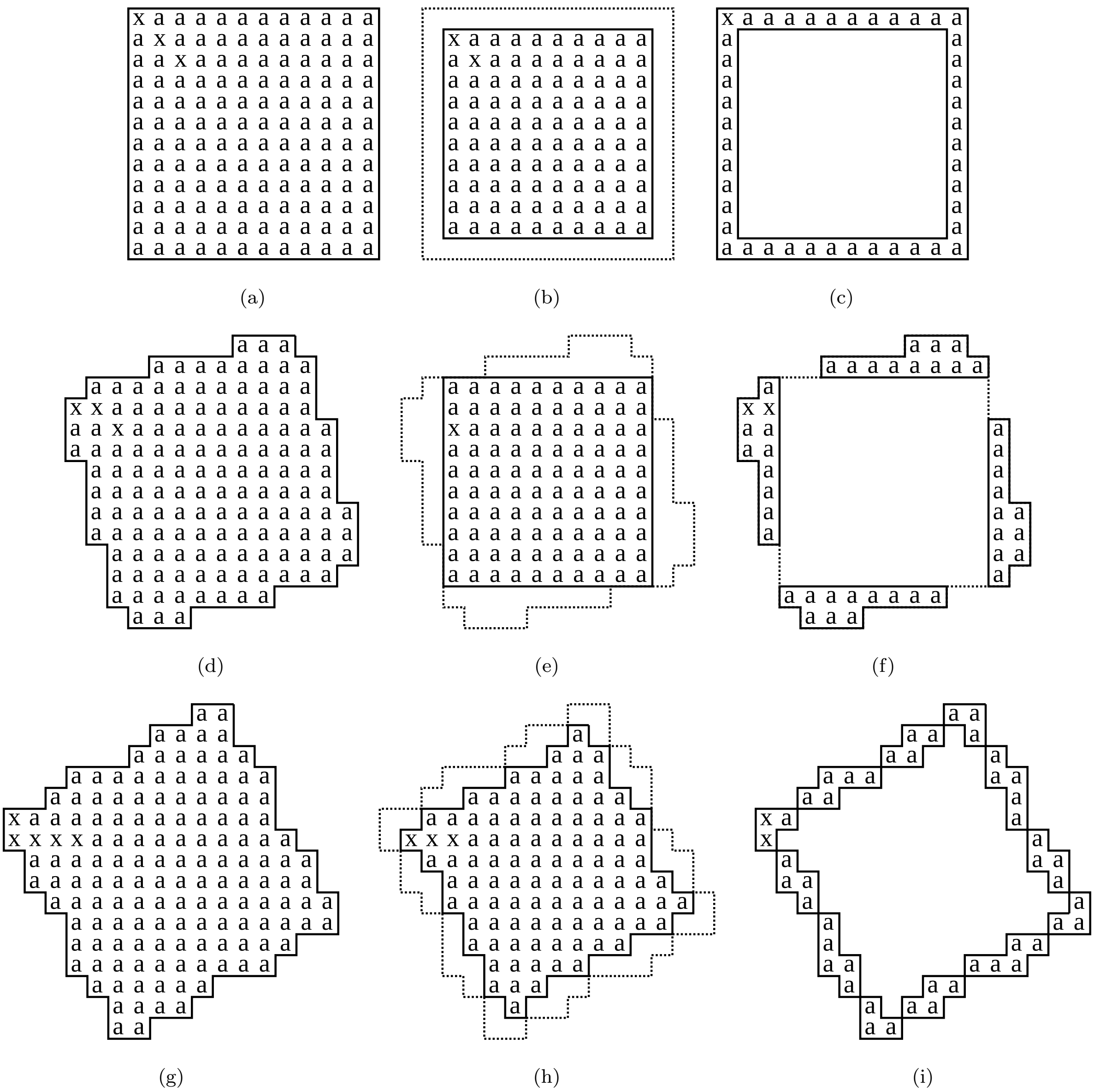


Figure 5: Examples for the definition of inner and outer parts. The pattern $P = P^{\alpha_0}$, its inner part, and its outer parts are shown in Figures (a)–(c), respectively. Note the three 'x' characters at the top left corner of P. Figures (d)–(f) show the pattern P^{α_s} and its inner and outer parts, for some $s < i_1$. Note that while the outline of the inner parts of P^{α_s} and P^{α_0} are identical, the characters of these two inner parts are different. Figures (g)–(i) show the pattern $P^{\alpha_{i_1}}$ and its inner and outer parts.

Proof. Let A be the set of all plane pixel centers that are inside $B_1^{\alpha_{i_{j-1}}}$, and let A' be the set obtained by rotating all the points in A by an angle of $-\alpha_{i_{j-1}}$. By definition, $\alpha_{i_j} - \alpha_{i_{j-1}}$ is equal to the minimum angle $\alpha > 0$ such that the set A intersects with the line segments of $B_0^{\alpha_{i_{j-1}}+\alpha}$. It is also equal to the minimum $\alpha > 0$ such that A' intersects with B_0^{α}.

We first show an upper bound on α. Every $\sqrt{2} \times \sqrt{2}$ rectangle contained in B_1 contains at least one point of A'. In particular, there is a point $(a, b) \in A'$ such that $a \geq m - 1 - \sqrt{2}$ and $b \geq m - 1 - \sqrt{2}$. Define β to be the minimum angle such (a, b) intersects with B_0^{β}. Clearly, $\alpha \leq \beta$, so it suffices to give an upper bound on β. Let $(m, \frac{m}{2} + y)$ be the point on the right segment of B_0 which is moved to (a, b) when B_0 is rotated by β. The points (a, b) and $(m, \frac{m}{2} + y)$ have an equal distance to $(\frac{m}{2}, \frac{m}{2})$, namely $\sqrt{(m/2)^2 + y^2} = \sqrt{(a - m/2)^2 + (b - m/2)^2} \geq \sqrt{2}(\frac{m}{2} - 1 - \sqrt{2})$. It follows that $y \geq \sqrt{\frac{m^2}{4} - 2(1 + \sqrt{2})m}$. For large enough m, $y \geq \frac{m}{3}$.

Clearly, $a = \frac{m}{2} + \frac{m}{2}\cos\beta - y\sin\beta$. Using Taylor's expansion on the right-hand side of the equality, we obtain that $a = m - y\beta - O(m\beta^2) \leq m - \frac{m}{3}\beta$. It follows that $\beta \leq \frac{3}{m}(m - a) \leq \frac{3}{m}(1 + \sqrt{2})$.

For the lower bound on α, we have that $\alpha \geq \gamma$, where γ is the minimum angle such that the right segment of B_0^{γ} intersect the line $y = m - 1$. Using arguments similar to the ones above, we obtain that $\gamma = \Omega(1/m)$. $\square$

Corollary 1. *$l = O(m)$.*

3.1 Matching the outer parts

Lemma 2. *There is a constant c such that for every i, the plane pixel centers that correspond to the characters of the outer part of P^{α_i} are outside the square $B_c^{\alpha_i}$.*

Proof. W.l.o.g. we will prove the lemma for $i < i_1$. The rightmost point of $B_0^{\alpha_i}$ is the bottom-right corner. Its x-coordinate is $\frac{m}{2} + \frac{m}{2}\cos\alpha_i + \frac{m}{2}\sin\alpha_i \leq m + c_0$ for some constant c_0, where the inequality follows from the fact that, by Lemma 1, $\alpha_i \leq \alpha_{i_1} = O(1/m)$. Let (a, b) be some plane pixel center that corresponds to the outer part of P^{α_i}, namely, (a, b) is inside the square $B_0^{\alpha_i}$ and outside the square B_1. Assume w.l.o.g. that $a \geq m - 1$. There is a point (a', b) on $B_0^{\alpha_i}$ whose distance to (a, b) is $a' - a \leq c_0 + 1$. Therefore, the distance between (a, b) and $B_0^{\alpha_i}$ is at most $c_0 + 1$, and the lemma follows. $\square$

Let $\alpha_{o_1}, \ldots, \alpha_{o_{l'}}$ be the angles in which the outer part of the pattern changes. From Lemma 2, the outer part depends on $O(m)$ pixels of the pattern. Each pattern pixel contains $O(m)$ different plane pixel centers at some step of the rotation. It follows that $l' = O(m^2)$. Therefore, we can check for all occurrences of the outer parts at some text location (a, b) in T in $O(m^2)$ time using the algorithm of [2].

3.2 Matching the inner parts

The multiple pattern matching problem (for one dimensional strings) is as follows:

Input: A set of strings (dictionary) $D = \{D_1, \ldots, D_k\}$ and a string T over alphabet Σ.
Output: All the occurrences of the strings of D in T.

Let n be the length of T, and let d be the sum of the lengths of the strings of D. For the case when $\Sigma = \{1, \ldots, n\}$, the multiple pattern matching problem can be solved in $O(n + d)$ time and space using the suffix tree construction algorithm of [11].

The algorithm for matching the inner parts is based on reducing the problem to two instances of the multiple pattern matching problem. First, build a dictionary D containing all the rows of all the inner parts.

Enumerating all these rows is done by maintaining the pattern P^{α_i} when i goes from 0 to k. When i is increased by 1, a single character in the rotated pattern changes, and the row of the inner part of P^{α_i} that contains this character is added to D.

For each string in D assign a distinct label from $1, \ldots, |D|$. Then, find all the occurrences of the strings in D in the rows of T. In other words, for each location (a, b) in T, we store a table $R_{a,b}$ such that $R_{a,b}[i]$ is the label of the dictionary string of length i that begins in row a of T in position b, if there is such a string, and 0 otherwise.

Lemma 3. $|D| = O(m^3)$. *The length of every string in* D *is* $O(m)$.

Proof. P has m rows. For each i, P^{α_i} differs from $P^{\alpha_{i+1}}$ in exactly one row and by [2] there are $\Theta(m^3)$ different angles α_i. Note also that the length of every string in D is at most $\lceil\sqrt{2}m\rceil$. □

Therefore, the stage described above takes $O(m^4 + n^2m)$ time.

We now need to put together rows that create an inner part. We will handle each different outline separately. Fix an outline I_j and consider the set of its inner parts, namely, the inner parts of $P^{\alpha_{i_j}}, \ldots, P^{\alpha_{i_{j+1}-1}}$. Denote this set of inner parts by P_j. Each inner part in P_j can be encoded by a string of length at most $\lceil\sqrt{2}m\rceil$ by writing the labels of its rows from the top row to the bottom row. Construct a dictionary D_j containing the strings that encode the inner parts in P_j (note that the strings in D_j all have the same length).

We will perform the matching in a brute force fashion, spending $O(m)$ time for every text location. For each text location (a, b) we build a string $T_{a,b,j}$ as follows: Let P' be some string in P_j. If the i-th row of P' has length r, and its leftmost pixel is $[x, y]$, then the i-th letter of $T_{a,b,j}$ is $R_{a+x-m/2,b+y-m/2}[r]$. Clearly, a string $P'' \in P_j$ appears at location (a, b) in T if and only if $T_{a,b,j}$ is equal to the string in D_j that corresponds to P''. Thus, by concatenating the strings $T_{a,b,j}$ for all a, b, and j to a string T', and solving the multiple pattern matching problem on D_j and T', we can find all the occurrences of the inner parts of P_j in T. The time complexity of this stage is $O(n^2ml)$. From Corollary 1, the time complexity for matching the inner parts is $O(n^2m^2)$.

We now describe how to merge the output from the two stages of the algorithm. For $j \leq l$, let S_j be the set of all intervals $[o_i, o_{i+1} - 1]$ that intersect with the interval $[i_j, i_{j+1} - 1]$. For each inner part $P' \in P_j$, let $L_{P'}$ be the set of all indices i such that the inner part of P^{α_i} is T'. For each $S \in S_j$ build a list $L_{P',S} = L_{P'} \cap S$. To output all the occurrences of P at position (a, b) of T, go over all $j \leq l$. For each j, if there is an inner part $P' \in P_j$ that matches to T at position (a, b), output all the lists $L_{P',S}$ for every set $S \in S_j$ whose corresponding outer part occurs in T at position (a, b) (for $S = [o_i, o_{i+1} - 1]$, the corresponding outer part is the outer part of $P^{\alpha_{o_i}}$). Note that the algorithm needs to output only a pointer to each list, so the time complexity does not depend on the lengths of the lists.

The total time complexity of the algorithm is $O(n^2m^2)$. The space complexity is (n^2m^2), which can be reduced to $O(m^4)$ by splitting the string T into overlapping substrings of size at most $3m \times 3m$, and running the algorithm on each substring separately.

4 Conclusion and Open Problems

We have provided an $O(n^2m^2)$ algorithm for the rotated matching problem. Our algorithm exploits geometric properties of the problem. Nevertheless, it seems like there is more knowledge to be exploited. For example, we did not make use of some point (a, b) in the text to obtain some properties about its neighboring points, even though they share a vast number of characters. Thus it is our feeling that the rotated matching problem can be solved more efficiently than $O(n^2m^2)$.

References

[1] A. Amir, G. Benson, and M. Farach. Alphabet independent two dimensional matching. *Proc. 24th ACM Symposium on Theory of Computation*, pages 59–68, 1992.

[2] A. Amir, A. Butman, M. Crochemore, G. M. Landau, and M. Schaps. Two-dimensional pattern matching with rotations. *Theoretical Computer Science*, 314(1–2):173–187, 2004.

[3] A. Amir, A. Butman, and M. Lewenstein. Real scaled matching. *Information Processing Letters*, 70(4):185–190, 1999.

[4] A. Amir, A. Butman, M. Lewenstein, and E. Porat. Real two dimensional scaled matching. In *Proc. 8th Workshop on Algorithms and Data Structures (WADS)*, pages 353–364, 2003.

[5] A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991.

[6] A. Amir, G.M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Proceedings of First Symposium on Discrete Algorithms, San Fransisco, CA*, pages 344–357, 1990.

[7] A. Apostolico and Z. Galil (editors). *Pattern Matching Algorithms.* Oxford University Press, 1997.

[8] T.J. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Comp.*, 7:533–541, 1978.

[9] R.S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977.

[10] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[11] M. Farach. Optimal suffix tree construction with large alphabets. *Proc. 38th IEEE Symposium on Foundations of Computer Science*, pages 137–143, 1997.

[12] K. Fredriksson, G. Navarro, and E. Ukkonen. An index for two dimensional string matching allowing rotations. In *Proc. IFIP International Conference on Theoretical Computer Science*, LNCS 1872, pages 59–75, 2000.

[13] 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)*, LNCS 2373, pages 235–248, 2002.

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

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

[16] Z. Galil and K. Park. Alphabet-independent two-dimensional witness computation. *SIAM J. Comp.*, 25(5):907–935, October 1996.

[17] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[18] G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12(3/4):375–408, 1994.