Two-Dimensional Pattern Matching with Rotations*

Amihood Amir† — AT&T Research Labs and Bar-Ilan University

Ayelet Butman‡ — Bar-Ilan University

Maxime Crochemore§ — University of Marne-La-Vallée

Gad M. Landau¶ — Haifa University and Polytechnic University

Malka Schaps|| — Bar-Ilan University

Abstract

The problem of pattern matching with rotation is that of finding all occurrences of a two-dimensional pattern in a text, in all possible rotations. Given an $m \times m$ array (pattern), and an $n \times n$ array (text), over some finite alphabet Σ, we present two $O(n^2m^3)$ time algorithms for this problem.

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 any use in its solution. Such feelings were based on the belief

*Part of this research was conducted while the first and fourth authors were visiting the University of Marne-La-Vallée supported by Arc-en-Ciel/Keshet, French-Israeli Scientific and Technical Cooperation Program.

†AT&T Labs – Research, Shannon Laboratory, E278, 180 Park Avenue, Florham Park, NJ 07932-0971, (973)360-8120; aamihood@research.att.com; Partially supported by ISF grant 282/01, and an Israel-France exchange scientist grant funded by the Israel Ministry of Science. Part of this work was done when the author was at Georgia Tech, College of Computing and partially supported by NSF grant CCR-01-04494.

‡Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8408, ayelet@cs.biu.ac.il.

§Institut Gaspard-Monge, University of Marne-La-Vallée, http://www-igm.univ-mlv.fr/~mac/ ; partially supported by CNRS action AlBio, NATO Science Programme grant PST.CLG.977017, and by Arc-en-Ciel/Keshet, French-Israeli Scientific and Technical Cooperation Program.

¶Department of Computer Science, Haifa University, Haifa 31905, Israel, phone: (972-4) 824-0103, FAX: (972-4) 824-9331; Department of Computer and Information Science, Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201-3840; email: landau@poly.edu; partially supported by NSF grants CCR-9610238 and CCR-0104307, by NATO Science Programme grant PST.CLG.977017, by the Israel Science Foundation grants 173/98 and 282/01, by the FIRST Foundation of the Israel Academy of Science and Humanities, by IBM Faculty Partnership Award, and by Arc-en-Ciel/Keshet, French-Israeli Scientific and Technical Cooperation Program.

||Department of Mathematics, Bar-Ilan University, Ramat-Gan, Israel, (972-3)531-8408, mschaps@macs.biu.ac.il.

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" 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.
3. *Rotation* - The pattern image appearing in the text in a different angle.

Several of these approximation types have been handled in the past [2, 4, 5, 6, 11, 14]

The problem of pattern matching with rotation is that of finding all occurrences of a two-dimensional pattern in a text, in all possible rotations. There was no known efficient solution for this problem, even though many researchers were thinking about it for over a decade. Part of the difficulty lay in the lack of a rigorous definition to capture the concept of rotation for a discrete pattern.

The first breakthrough came quite recently. Fredriksson and Ukkonen [9] gave an excellent combinatorial definition of rotation. They resorted to a geometric interpretation of text and pattern and provided the following definition.

Let P be a two-dimensional $m \times m$ array and T be a two-dimensional $n \times n$ array over some finite alphabet Σ. The array of *unit pixels* for T consists of n^2 unit squares, called *pixels* in the real plane R^2. The corners of the pixel $T[i,j]$ are $(i-1, j-1), (i, j-1), (i-1, j)$, and (i,j). Hence the pixels of T form a regular $n \times n$ array covering the area between $(0,0), (n,0), (0,n)$, and (n,n). The *center* of each pixel is the geometric center point of its square location. Each pixel $T[i,j]$ is identified with the value from Σ that the original text had in that position. We say that the pixel has a *color* from Σ. See figure 1 for an example of the grid and pixel centers of a 7×7 text.

The array of pixels for 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 is its exact center, the point $(\frac{m}{2}, \frac{m}{2}) \in R^2$. See figure 2 for an example of the rotation pivot of a 4×4 pattern P.

Consider now a rigid motion (translation and rotation) that moves P on top of T. Consider the special case where the translation moves the grid of P precisely on top of the grid of T, such that the grid lines coincide.

Assume that the rotation pivot of P is at location (i,j) on the text grid. The pattern is now rotated, centered at (i,j), creating an angle α between the x-axes of T and P. P is said to be at *location* $((i,j),\alpha)$ *over* T. Pattern P is said to have an *occurrence* at location $((i,j),\alpha)$ if the center of each pixel in P has the same color as the pixel of T under it. See figure 3 for an example of 4×4 pattern P at location $((3,4), 45^o)$ on top of 8×8 text T.

Fredriksson and Ukkonen [9] give a rotation invariant filter for two-dimensional matching. Their algorithm performs well in reality. Combinatorially, though, their algorithm has a worst case time complexity of $O(n^2 m^5)$.

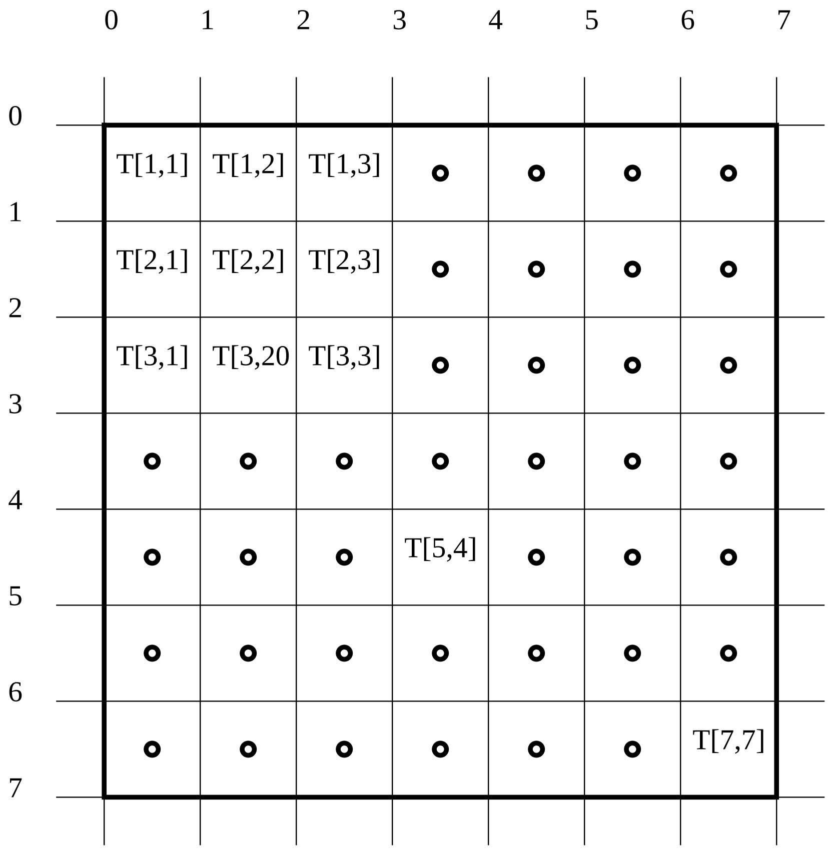

Figure 1: The text grid and pixel centers of a 7×7 text.

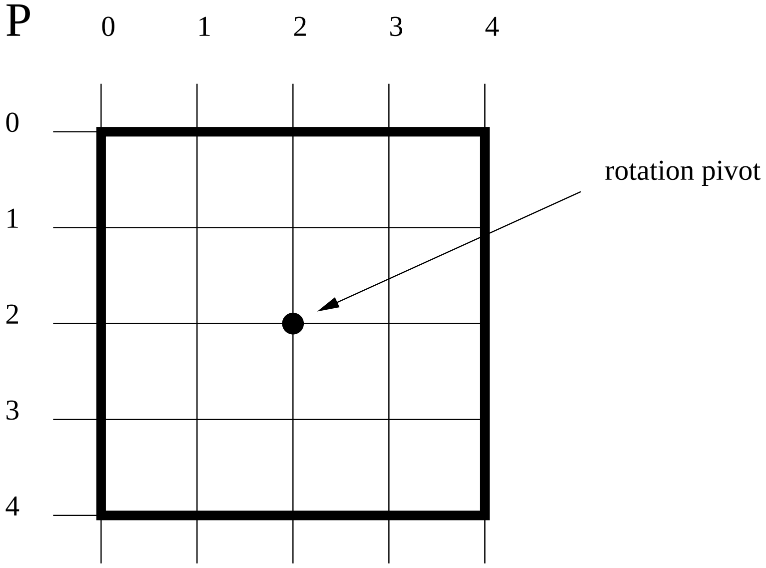

Figure 2: The rotation pivot of a 4×4 pattern P.

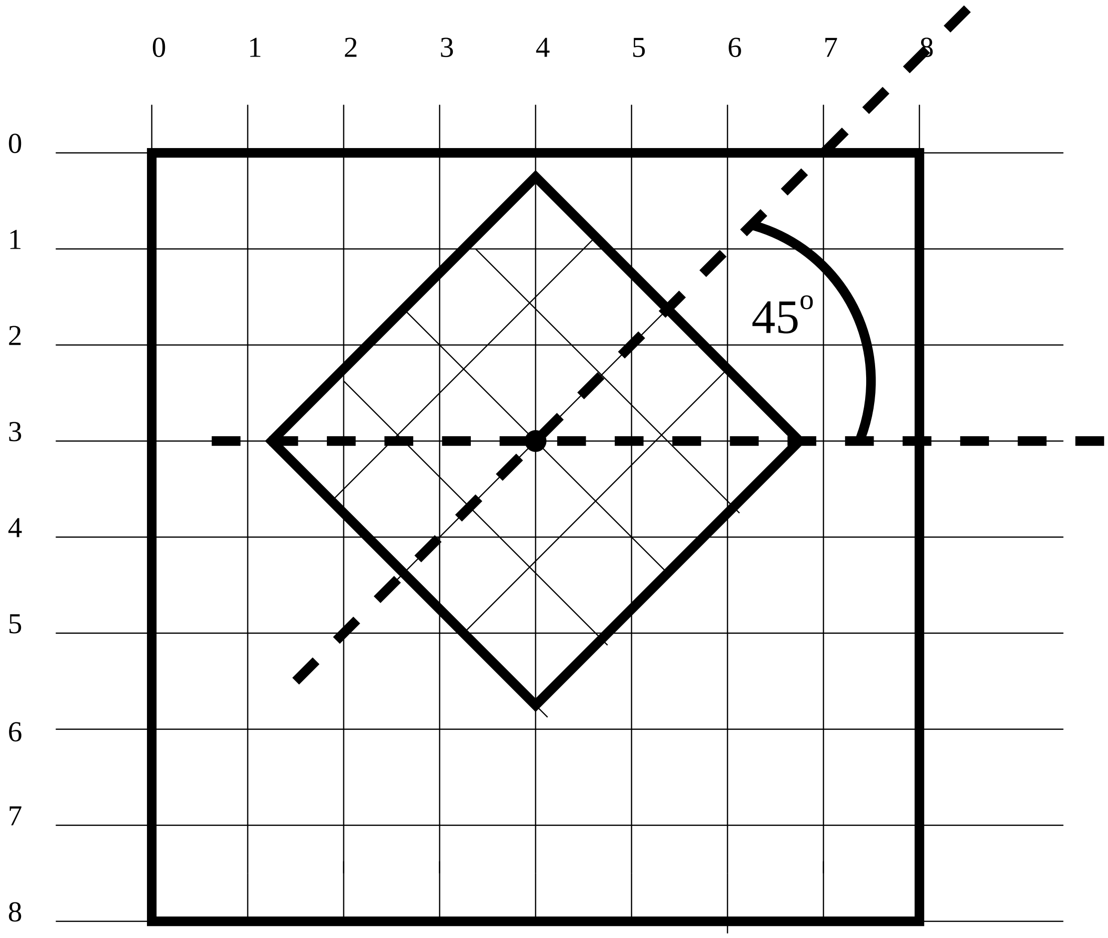

Figure 3: 4×4 pattern P at location $((3,4),45^o)$ over 8×8 text T.

In a subsequent paper, Fredriksson, Navarro and Ukkonen [8] discuss an index for two-dimensional matching involving rotation. They give an indexing scheme for the rotated search problem. Their scheme allows a rotated search in expected time $O(n^2m^3)$, but the worst case is $O(n^2m^5)$.

In that paper ([8]) Fredriksson, Navarro and Ukkonen give two possible definitions for rotation. One is as described above and the second is, in some way, the opposite. P is placed *under* the text T. More precisely, assume that the rotation pivot of P is under location (i, j) on the text grid. The pattern is now rotated, centered at (i, j), creating an angle α between the x-axes of T and P. P is said to be at *location* $((i, j), \alpha)$ *under* T. Pattern P is said to have an *occurrence* at location $((i, j), \alpha)$ if the center of each pixel in T has the same color as the pixel of P under it.

While the two definitions of rotation, "over" and "under", seem to be quite similar, they are not identical. For example, there exist angles for which two pattern pixel centers may find themselves in the same text pixel. Alternately, there are angles where a text pixel does not have in it a center of a pattern pixel, but all text pixels around it have centers of pattern pixels. Thus there is a text pixel without a pattern center "surrounded" by text pixels with pattern centers. In the "pattern under text" model the center of the text pixel is the indicator of the rotated pattern, thus it is impossible to have a rotated pattern with "don't cares" surrounded by symbols, whereas in the "pattern over text" model, this situation may happen. Figure 4 shows an example where text location $T[3, 4]$ does not have any pattern pixel center in it, but $T[2, 5]$ and $T[3, 6]$ have pattern pixel centers. On the other hand, if the text pixel center is the one to decide the color of the pixel, then all contiguous text pixels that have pattern area under them define a rotated pattern.

Although all our results apply to both definitions of rotation, to avoid confusion our paper will henceforth deal only with the rotation of pattern over the text.

This paper considers the worst-case behavior of the rotation problem. We propose a simple strategy of searching for all different patterns that represent possible rotations. To this end we prove an upper and lower bound on the number of such different possible rotated patterns. Subsequently, by using appropriate data structures, we were able to develop two deterministic algorithms whose worst-case complexity is $O(n^2m^3)$. We present both algorithms since we believe that the problem can be solved more efficiently. Some of the methods we present may prove fruitful in further improving the solution for this problem.

The rotation problem is an interesting one since it brings together two research areas that deal with similar problems by using different means – pattern matching and computational geometry. Recent geometry papers have made use of pattern matching techniques to solve geometric problems see e.g. [15, 13]). We believe this is the first pattern matching problem that makes heavy use of geometry to solve pattern matching problems.

This paper is organized as follows. In Section 3 we give a tight bound on the number of different patterns that represent all possible rotations of a given pattern. This leads to a simple $O(n^2m^3 \log m)$ algorithm for the rotated matching problem.

In the following sections we present two different ways of improving the time to $O(n^2m^3)$. Finally, we conclude with ideas for further improvements.

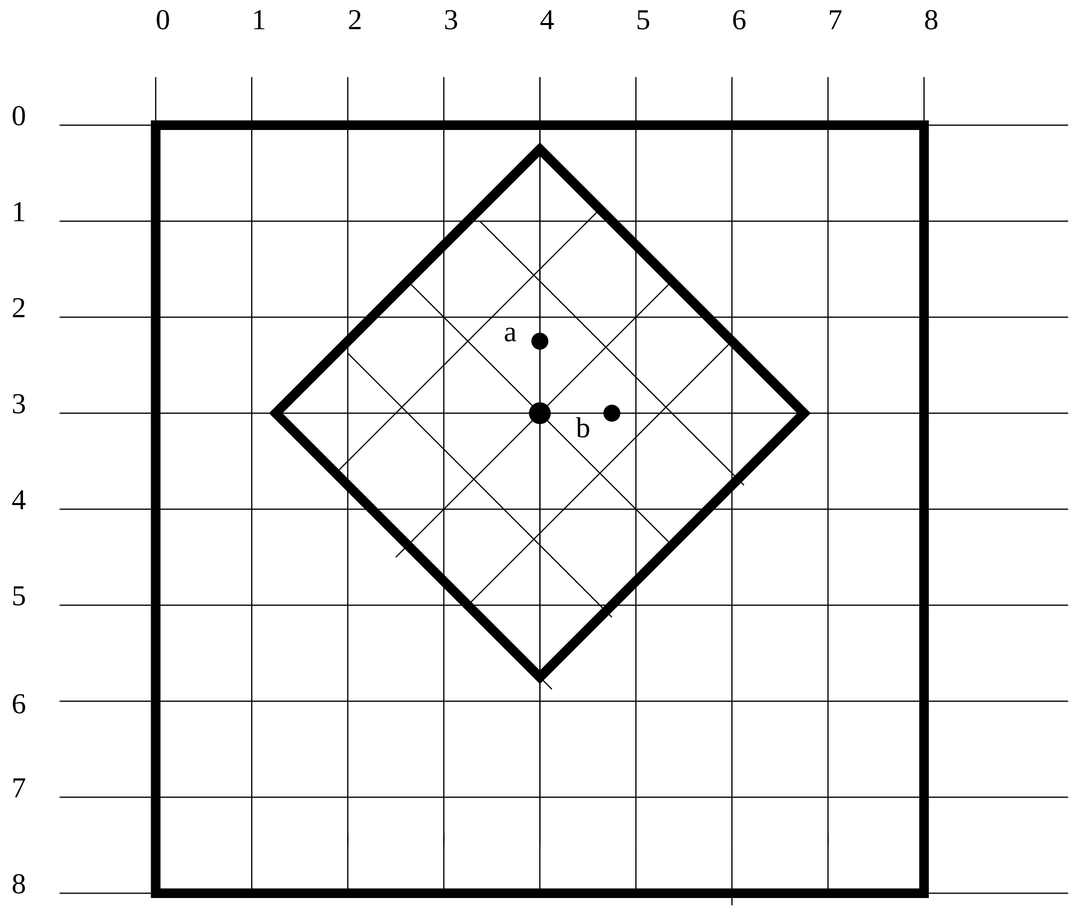

Figure 4: Text pixel $T[3, 5]$ has no pattern pixel over it.

2 The Dictionary-Matching Solution

An immediate natural idea for solving the rotation problem is to use *dictionary matching* solutions. That is, construct a dictionary of all possible rotations of the pattern and then search for them in the text.

We slightly modify the definition given in [9] for *rotated occurrences of P in T*.

Definition 1 *Let P be an $m \times m$ pattern with m even. Place P on the real plane R^2 in the following manner:*

1. *The rotation pivot coincides with the origin $(0,0)$.*
2. *Every pattern element corresponds to a 2×2 square.*

The segments $x = 2i,\ \ i = -\frac{m}{2}, ..., 0, ..., \frac{m}{2};\ \ -m \le y \le m$ and $y = 2j,\ \ j = -\frac{m}{2}, ..., 0, ..., \frac{m}{2};\ \ -m \le x \le m$ are the pattern grid.

The 2×2 squares whose corners, clockwise from the top left, are $(2i, 2j), (2i, 2(j+1)), (2(i+1), 2(j+1))$ and $(2(i+1), 2j),\ \ i, j = -\frac{m}{2}, ..., 0, ..., \frac{m}{2} - 1$ are the pattern grid cells.

The points $(2i+1, 2j+1),\ \ i, j = -\frac{m}{2}, ..., 0, ..., \frac{m}{2} - 1$ are the pattern pixel centers.

The color *of pattern pixel center $(2i+1, 2j+1)$ is $P[i + \frac{m}{2} + 1, j + \frac{m}{2} + 1]$.*

Example: The pattern in figure 5 is of size 8×8. The pattern grid is the dotted (even valued) lines. The pattern pixel centers are the intersection of the solid (odd valued) lines.

Definition 2 *Let P^c be the lattice whose points are the pattern pixel centers. The* color *of lattice point p is the color of the pattern pixel center that coincides with p. Let P^c_α be the lattice P^c rotated around the origin by angle α. If there is a pattern grid cell with two lattice points of P^c_α in it, then we say that* there is no legal rotated pattern for angle α. *Otherwise, color each pattern grid cell by the color of the (unique) lattice point P^c_α that is in that grid cell, if such exists, and by ϕ if there is no such lattice point. ϕ is the* don't care *or* wildcard *symbol that matches every character in the alphabet.*

Consider the smallest square centered at $(0,0)$ that includes all non-ϕ colored pixels, and construct a matrix of the pixel colors of that square. This matrix is the pattern P rotated by angle α.

While there exist efficient two-dimensional dictionary-matching algorithms (e.g. [3, 10]), none of them works with don't cares. The only known algorithm for efficiently solving string matching with don't cares is the Fischer-Paterson algorithm [7]. The Fischer-Paterson algorithm finds all occurrences of a string of length m in a string of length n, both possibly having occurrences of the don't care symbol, in time $O(n \log^2 m)$. Unfortunately, this method does not generalize to dictionary matching.

Therefore, an immediate suggestion for a rotation algorithm is the following.
Preprocessing: Construct a data base of all possible pattern rotations $P_i,\ i = 1, ..., k$.

Text scanning: Let T be the input text (an $n \times n$ array).
For every pattern rotation P_i in the data base do:
  Find all occurrences of P_i in T.
Time: The algorithm's running time is $O(kn^2 \log m)$. In the next section we prove that the number of different pattern rotations k is $\Theta(m^3)$. Thus the running time of the above algorithm becomes $O(n^2 m^3 \log m)$.

3 The Number of Different Rotations

Let *pattern* $P = P[1..m, 1..m]$ be a two-dimensional $m \times m$ array and let *text* $T = T[1..n, 1..n]$ be an $n \times n$ array over some finite alphabet Σ. We assume that $m < n$.

We want to prove that the upper and lower bounds on the number of legal pattern rotations is $\Theta(m^3)$.

It is clear that the upper bound is $O(m^3)$ since we have m^2 different points in the pattern, each creates no more than m different angles (whenever its orbit crosses an odd coordinate).

It suffices to prove that the order of the different rotations is $\Omega(m^3)$, and that will establish a tight bound of $\Theta(m^3)$.

We will restrict ourselves to points (x, y) in the first quadrant $(0 \leq x, y)$ whose coordinates x and y are *relatively prime*, or *co-prime*, i.e., the greatest common divisor of x and y is 1. We prove that the number of different rotations of just the first quadrant points with coprime coordinates is $\Omega(m^3)$.

Theorem 1 *There are $\Omega(m^3)$ different rotations of an $m \times m$ pattern.*

Proof: We will show that, for every two different first quadrant points X_1 and X_2 having coprime coordinates, it is impossible that they cross a grid line at the same angle. The cardinality of the set $\{(n_0, m_0) \mid 1 \leq n_0, m_0 \leq m,$ and n_0, m_0 are coprime$\}$ is: $\frac{6m^2}{\pi^2} + o(mlogm)$. This is a direct corollary of [theorem 330, [12]]. Since there are $\Theta(m^2)$ such points and $\Omega(m^2)$ of them cross the grid $\Omega(m)$ times, there are $\Omega(m^3)$ different rotations.

Assume, then, that X_1 and X_2 each have coprime integer coordinates. Assume also that if $X_1 = (c, s)$ then $X_2 \neq (s, c)$. This second assumption reduces the number of pairs under consideration by a half, but it is still $\Theta(m^2)$. Rotations moves X_1 to Y_1 and X_2 to Y_2. Assume that both Y_1 and Y_2 just crossed a horizontal text grid line. This means that both Y_1, Y_2 have coordinates of which one is an even integer and one is of the form $\sqrt{2n}$, where n is odd. The reason for this is the following.

Let $X_1 = (c, s)$, and $Y_1 = (c', s')$. Note that c, s are odd integers and therefore can be denoted by $c = 2k_1 + 1, s = 2k_2 + 1$. s' is even so it can be denoted by $s' = 2l_1$. Using the "Pythagoras theorem" we get $c' = (2k_1+1)^2 + (2k_2+1)^2 - 4{l_1}^2 = 4{k_1}^2 + 4k_1 + 1 + 4{k_2}^2 + 4k_2 + 1 - 4{l_1}^2 = 4({k_1}^2 + k_1 + {k_2}^2 + k_2 - l_1) + 2$. Therefore, c' is even. It can be denoted by $c' = 2l_2$. Substituting c' with $2l_2$ we get: $4({k_1}^2 + k_1 + {k_2}^2 + k_2 - l_1) + 2 = 2l_2$. Therefore, $2({k_1}^2 + k_1 + {k_2}^2 + k_2 - l_1) + 1 = l_2$.

Claim 1 *$\measuredangle X_1 O Y_1 \neq \measuredangle X_2 O Y_2$.*

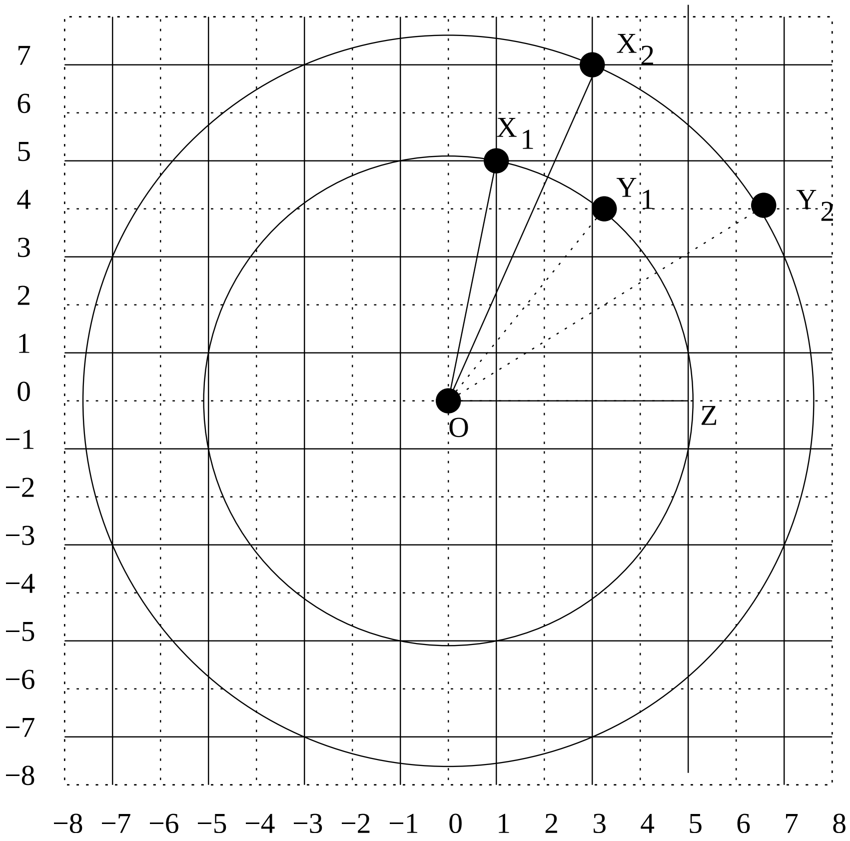

Figure 5: Points X_1 and X_2 each have coprime integer coordinates. Orbits of them under rotation around point O crosses an horizontal line at points Y_1 and Y_2 respectively. Then, $\measuredangle X_1OX_2 \neq \measuredangle Y_1OY_2$ by Claim 1.

proof: It suffices to show that $\measuredangle X_1OX_2 \neq \measuredangle Y_1OY_2$. Consider the three possibilities:

- case 1: Both Y_1, Y_2 have a horizontal coordinate as an even integer coordinate.
  Denote,
  $$X_1 = (c_1, s_1)$$
  $$X_2 = (c_2, s_2)$$
  $$Y_1 = (c'_1, s'_1)$$
  $$Y_2 = (c'_2, s'_2)$$
  then, by the formula $sin(\alpha - \beta) = sin\alpha cos\beta - cos\alpha sin\beta$, we get
  $$sin\measuredangle X_1OX_2 = \frac{s_1c_2 - s_2c_1}{r_1r_2}$$
  $$sin\measuredangle Y_1OY_2 = \frac{s'_1c'_2 - s'_2c'_1}{r_1r_2}$$
  where, $r_1 = \sqrt{c_1^2 + s_1^2}$, $r_2 = \sqrt{c_2^2 + s_2^2}$.
  Assume $sin\measuredangle X_1OX_2 = sin\measuredangle Y_1OY_2$
  then, $s_1c_2 - s_2c_1 = s'_1c'_2 - s'_2c'_1$ where, $s'_1, s'_2 \in 2\mathbb{Z}$,
  $c'_1 = l_1\sqrt{2m_1}$, $c'_2 = l_2\sqrt{2m_2}$ s.t. m_1, m_2 are odd and square free (i.e. they do not have a square factor).
  $d' = s'_1c'_2 - s'_2c'_1 \in \mathbb{Z}[\sqrt{2m_1}, \sqrt{2m_2}]$
  Now, either $m_1 = m_2$ or not. In both cases we will reach a contradiction.
  - If $m_1 \neq m_2$ then $d' \notin \mathbb{Z}$ since, $\sqrt{2m_1}$, $\sqrt{2m_2}$ have non zero coefficients and are linearly independent. This contradicts the fact that $d' \in \mathbb{Z}$ since $d' = s_1c_2 - s_2c_1$.
  - If $m_1 = m_2$ then $d' \in \mathbb{Z}$ iff $d' = 0$ in that case either X_1 and X_2 are the same point, or $c_1 = s_2 and s_1 = c_2$, or one of them does not have coprime coordinates. Both later cases contradict the assumption.
- case 2: Both Y_1, Y_2 have an even vertical coordinate: exchange s'_i with c'_i.
- case 3: One has an even vertical coordinate and one has an even horizontal coordinate: replace cos with sin.

□

4 Getting Rid of the log Factor

We provide two different methods for solving the problem of pattern matching with rotation in time $O(n^2m^3)$. This improves on the immediate $O(n^2m^3 \log m)$ solution of section 2.

4.1 Updating Changes Method

We have established in Section 3 that there are $\Theta(m^3)$ different discrete patterns that represent all possible rotations. We consider the situation now from a different point of view. Suppose P is at location $((i,j),\alpha)$. As we rotate P, the center of its pixels moves from one text pixel to another. We are interested in quantifying this shift through text pixels.

Definition: Let P be at location $((i,j),\alpha)$. The *changes resulting from rotation* ϵ are the pattern pixel centers that go through (above) more than one different text pixel during rotation ϵ. The *number of changes resulting from rotation* ϵ is the sum of the number of different text pixels that each change traversed during rotation ϵ.

Claim 2 *Let P be at location $((i,j),0)$. The number of changes resulting from a 2π radian rotation is $O(m^3)$.*

Proof: During a complete 2π radian rotation, the center of every pattern pixel moves through at most $O(m)$ text pixels. Since there are m^2 pattern elements then the total number of changes is $O(m^3)$. □

Our idea is to create, for a given m, a list of angles that produce changes, together with all the changes each such angle produces. Specifically, for every angle, we want to store all pixels that moves from a text pixel to another text pixel and keep track of these pixels. Note that this list is identical for all $m \times m$ patterns over all alphabets. Claim 2 guarantees that the size of this list is $\Theta(m^3)$.

Our algorithm's pattern preprocessing is then:

Preprocessing

For every pattern pixel center do:

Generate the formula of the circle created by that center during a 2π rotation (its orbit).

Mark all angles and changes – the intersection of the circle with the text grid.

endFor

Merge the angles in the m^2 lists, to form a list of changes increasing by rotation angle.

end Preprocessing

Note that the above is a *structural preprocessing.* Unlike most pattern preprocessing in the pattern matching literature, our preprocessing does not use any information on the pattern symbols. The only information needed is the pattern dimension. This type of structural preprocessing is used in the algorithm of subsection 4.2 as well.

Implementation Remarks: The angle precision is given by the problem definition. The preprocessing can accommodate any precision that is at least $\log m$ bits since the exact angle is not necessary for the preprocessing. The only necessary information is differentiating between different angles. For this, a precision of $\log m$ bits is sufficient since there are $O(m^3)$ different angles.

Time for Preprocessing: The preprocessing running time can be made $O(m^3)$.

The text scanning involves two operations. The first is finding the number of mismatches that the original, unrotated pattern has at every text location. The second step is, for every text location, updating the number of mismatches as it changes for every one of the $O(m^3)$ relevant rotation angles. For every text location and every angle where the number of mismatches is 0 is a rotated match.

Text Scanning

Compute the number of mismatches of P (unrotated) at every text location.

For every text pixel do:

For every angle in the precomputed list of changes do:

Update the number of mismatches reflected by this angle's change.

Report a match if the number of mismatches is 0.

endFor

endFor

end Text Scanning

Implementation Remarks: The preprocessing assures us that every angle has a list of pattern pixel centers that changed a text pixel at that angle. Comparing the symbol in the pattern pixel with the symbols of the past and present text pixels allows us to compute whether the number of mismatches remain the same (in case the pattern symbol either matches both text symbols or mismatches both text symbols), increases by one (if the pattern symbol matches the old text symbol but mismatches the new one) or decreases by one (if the pattern symbol mismatches the old text symbol but matches the new one).

Time for Text Scanning: The initial mismatch count of P in T can be done in time $O(n^2 m\sqrt{\log m})$ using Abrahamson's algorithm [1]. The ensuing mismatch update takes constant time per change in the list. Since the list has length $O(m^3)$ by claim 2, and since the entire list is scanned for every text location, the total algorithm time is $O(n^2m^3)$.

4.2 Invalid-Intervals Method

Definition: The *relevant text pixels for P at location* $((i,j),\alpha)$ are all text pixels that are under at least one pattern pixel center. The *relevant text pixels for P at text location* (i,j) are all relevant text pixels for P at location $((i,j),\alpha)$, for all $\alpha \in [0, 2\pi)$.

Let P be centered at some fixed text location (i,j), and let $\langle (x_P, y_P), (x_T, y_T)\rangle$ be a pair where (x_P, y_P) is a pattern location and (x_T, y_T) is a relevant text pixel of P at location (i,j). The *angle interval of pair* $\langle (x_P, y_P), (x_T, y_T)\rangle$ is the interval $[\alpha, \beta], 0 \leq \alpha < \beta \leq 2\pi$, where α is the smallest angle for which the center of pattern pixel (x_P, y_P) is on text pixel (x_T, y_T) at location $((i,j),\alpha)$, and β is the largest angle for which the center of pattern pixel (x_P, y_P) is on text pixel (x_T, y_T) at location $((i,j),\beta)$.

Claim 3 *Let P be centered at some fixed text location (i,j). There are $O(m^3)$ pairs $\langle (x_P, y_P), (x_T, y_T)\rangle$, where (x_P, y_P) is a pattern location and (x_T, y_T) is a relevant text pixel of P at location (i,j), that* *have non-empty angle intervals.*

Proof: For a fixed text location there are $O(m^2)$ relevant text pixels. Therefore the total number of pairs is $O(m^4)$. However, some pattern elements cannot match some text pixels in a rotation centered at fixed text location (i, j). As noted in the proof of claim 2, every pattern pixel center goes through $O(m)$ text pixels in a full 2π radians rotation. Since there are m^2 different pattern pixels, this means that the number of pairs that generate non-empty angle intervals is $O(m^3)$. $\square$

The preprocessing in the *invalid-intervals* method is also structural. Fix a text location, as the center of rotations. Compute the angle interval for each one of the $O(m^3)$ pairs. In the text scanning phase, for every text location as center, we are guaranteed that every interval where the pattern symbol does not equal the text symbol in its pair, cannot have a match. Any angle not covered by an invalid interval is an angle where there is a rotated match.

Our algorithm's pattern preprocessing is then:

Preprocessing

For every pattern pixel center do:

Generate the formula of the circle created by that center during a 2π rotation.

For every text pixel intersecting with the circle, mark the angle interval.

endFor

end Preprocessing

Implementation Remarks: As remarked for the preprocessing of subsection 4.1, the preprocessing can accommodate any angle precision that is at least $\log m$ bits. In fact, we designate the angles by $\{1, ..., 2m^3\}$ (with an attached conversion table of appropriate precision).

Time for Preprocessing: The preprocessing running time can be made $O(m^3)$.

The text scanning involves ascertaining which are the invalid angle intervals and computing the union of these intervals. Any remaining angle is a rotated match. The invalid intervals are declared via a brute-force check. The union is computed by sorting and merging the intervals.

Text Scanning

For every text pixel do:

For every pair $\langle (x_P, y_P), (x_T, y_T) \rangle$ do:

If $P[x_P, y_P] \neq T[x_T, y_T]$ then mark the interval $[\alpha, \beta]$ of pair $\langle (x_P, y_P), (x_T, y_T) \rangle$ as **invalid**.

endFor

Bucket Sort the invalid intervals by starting angle.

Merge overlapping intervals into a single larger interval.

If the interval $[0, 2\pi]$ is achieved, then there is no rotated match at text pixel. Otherwise, all angles outside the merged intervals are rotated matches.

endFor

end Text Scanning

Implementation Remarks: Recall that our angles are denoted by numbers in $\{1, ..., 2m^3\}$, thus bucket sort is appropriate in the above algorithm.

Time for Text Scanning: For each of the n^2 text pixels the algorithm makes $O(m^3)$ comparisons. Bucket sorting the resulting $O(m^3)$ invalid intervals is done in time $O(m^3)$, and merging is done in linear time, for a total of $O(m^3)$ per text pixel. The total algorithm time is, therefore, $O(n^2m^3)$.

5 Future Work

We have presented two strategies to search for rotated two-dimensional patterns in arrays. Although our algorithms improve on the previous solution given by Fredriksson and Ukkonen [9], their time complexity is not entirely satisfactory and seems to leave room for further improvements. Our opinion is based on the fact that each position in the text array is examined independently of other positions. We have been unable to take into account the information collected at one position to accelerate the test at next positions, as it is classical in most pattern matching methods. However, we believe that this is possible.

References

[1] K. Abrahamson. Generalized string matching. *SIAM J. Comp.*, 16(6):1039–1051, 1987.

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

[3] A. Amir and M. Farach. Two dimensional dictionary matching. *Information Processing Letters*, 44:233–239, 1992.

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

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

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

[7] M.J. Fischer and M.S. Paterson. String matching and other products. *Complexity of Computation, R.M. Karp (editor), SIAM-AMS Proceedings*, 7:113–125, 1974.

[8] K. Fredriksson, G. Navarro, and E. Ukkonen. An index for two dimensional string matching allowing rotations. In *Prof. IFIP International Conference on Theoretical Computer Science (IFIP TCS)*, volume 1872 of *LNCS*, pages 59–75. Springer, 2000.

[9] 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.

[10] R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. *Information and Computation*, 130(2):151–182, 1996.

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

[12] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford at the Clarendon Press, fifth edition, 1979.

[13] P. Indyk, R. Motwani, and S. Venkatasubramanian. Geometric matching under noise: Combinatorial bounds and algorithms. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 354–360, 1999.

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

[15] L. Schulman and D. Cardoze. Pattern matching for spatial point sets. *Proc. 39th IEEE FOCS*, pages 156–165, 1998.