AN EFFICIENT ALGORITHM FOR 2-DIMENSIONAL PATTERN MATCHING PROBLEM

Abstract: Pattern matching is the area of computer science which deals with security and analysis of data. This work proposes two 2D pattern matching algorithms based on two different input domains. The first algorithm is for the case when the given pattern contains only two symbols, that is, binary symbols 0 and 1. The second algorithm is in the case when the given pattern contains decimal numbers, that is, the collection of symbols between 0 and 9. The algorithms proposed in this manuscript convert the given pattern into an equivalent binary or decimal number, correspondingly find the cofactors of the same dimension and convert these cofactors into numbers if a particular cofactor number matches indicate the matching of the pattern. Furthermore, the algorithm is enhanced for decimal numbers. In the case of decimal numbers, each row of the pattern is changed to its decimal equivalent, and then, modulo with a suitable prime number changes the decimal equivalent into a number less than the prime number. If the number mismatched pattern does not exist, the complexity of the proposed algorithm is very low as compared to other traditional algorithms.

Keywords: 2-dimensional, algorithm, cofactors, complexity
1. Introduction

In the context of computers, pattern matching is the activity of inspecting or scrutinizing a predefined arrangement of tokens for the occurrence of the whole or part of the pattern. The concept of pattern matching is one of the oldest activities performed by almost all programming language compilers. Computer languages use pattern matching in checking and verifying syntax errors through parsing process, identification of tokens, sequences, language-specific grammars, and other similar activities.

One of the classic definitions of pattern matching given by Dunn as, “Pattern matching is essential for achieving knowledge of external actions, events, and objects, knowledge of causal regularities that have been repeatedly recognized as the same, knowledge of the contextual meanings of discrete empirical observations, knowledge of the fit between a curve and a set of data points, knowledge of the extent to which two or more theories cohere, and knowledge of the extent to which theories correspond to data” [10].

Concepts of one dimensional and 2-dimensional patterns are shown in fig. 1 and fig. 2 respectively. Normally in the context of one-dimensional pattern matching, the problem is to find appearances of a given pattern in a text string. Most of the solutions available in the literature for one-dimensional pattern matching focus on searching the text from either left to right or from right to left. The concept of a 2-dimensional pattern requires a different approach. The 2D pattern matching can be categorized into six major categories: a) Exact 2D matching, b) Approximate 2D matching of rectangular patterns, c) Approximate 2D matching of non-rectangular patterns, d) Scaled matching, e) Compressed matching, and f) Dictionary matching.

![Fig. 1. One Dimensional Pattern Matching](image1)

![Fig. 2. Two Dimensional Pattern Matching](image2)
In this work, we present two efficient 2D pattern matching algorithms for considering two aspects: a) when the given pattern contains only two symbols, that is, binary symbols 0 and 1, and, b) when the given pattern contains decimal numbers, that is, the collection of symbols between 0 and 9.

This paper is organized as follows: after discussing the Introduction in section 1, the related work is covered in section 2. Section 3 discusses the proposed algorithms and a numeric simulation based on these algorithms, including an experimental case study, and finally, the conclusion is given in Section 4.

2. Background Work

Some basic algorithms available in the literature for the problem of pattern matching are ‘Knuth, Morris, Pratt algorithm’ [13], ‘Bayer and Moore algorithm’ [5] and ‘Karp and Rabin’ [11]. These algorithms were presented in the context of the 1D pattern. Baker [4] suggested a logical framework useful for dealing with exact matches with 2D arrays. Baker presented a generalized solution based on the Morris-Pratt matching algorithm in a detailed scope. Their algorithm includes multi-pattern arrays, irregularly shaped arrays, and some “real-time matching”. Baker’s algorithm is a linear-time algorithm for 2D pattern matching. The author claimed algorithm as “online in the sense that each position of the input is scanned only once, and that before the next character is read, all matches at the previous position have been found” [4].

Amir et al. [2] proposed a pattern matching algorithm to find the upper and lower bounds of the occurrences of a text in a pattern based on the concept of rotation. According to him, 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”. Bird [6] proposed an algorithm to match 2D arrays, including two steps: column matching and row matching. The overall running time of his algorithm is $O(m^2 + n^2)$, where $m$ is the number of columns and $n$ is the number of rows, and it takes $O(m + n^2)$ space to run. In the algorithm, the author used the concept of the Knuth-Morris-Pratt (KMP) algorithm as a subprogram. Author generalized the KMP algorithm for 1D array into a 2D array. In row matching of the pattern, he used “Aho and Corasick’s algorithm.”

Karp et al. [12] presented a technique for matching repeated appearances of ‘$n \times n$’ sub-arrays in a text. Their algorithm takes $n2\log n$ time to execute. Karp et al. solution is based on the matching of square sub-arrays where firstly rows are matched, and then the columns are matched.

Knuth et al. [14] given an algorithm to find the given string in the text. Their algorithm’s execution time is $O(n + m)$. In their algorithm, the pattern is the 2D array which is to be matched from the given text, which is also divided into rows and columns.

Apostolico and Giancarlo [3] presented an updated version of ‘Boyer-Moore-Galil’ algorithm to find occurrences of a pattern in the given text. They offered that their algorithm makes in total ‘$2n-m+1$’ comparisons and will execute in linear time.
Manea and Schmid [15] described a series of results based on some efficient pattern matching techniques with variables. The authors also give the extensions of this matching with the variable problem. They propose their study by considering structural parameters. Their study focuses on some state-of-the-art concepts, including defining the hardness of the matching problem, structural restrictions for patterns, pattern matching by graph morphism, matchable classes of patterns, and computation of structural parameters for parameters. The authors also elaborated on the concept of faster pattern matching, which includes patterns with low scope coincidence degree, and patterns with low locality number. The authors included pattern matching beyond bounded treewidth as well as graph parameters. Further, they discussed the extension of the problem involving injectivity and word equations.

Charalampopoulos et al. [8] proposed solutions for circular pattern k-mismatch problem with worst-case upper bounds. The authors proposed two efficient algorithms having \(O(nk)\)-time and \(O(n + n/m k^5)\)-time complexity. The algorithms proposed by authors are motivated by the fact of the importance of circular matching k mismatch problems in real-life applications like bioinformatics and image processing. The authors give the solution for pattern matching at the given position in the text that is, an anchor, “which belongs to potential k-mismatch circular occurrences of P”. In their study, they include matching problems for non-periodic samples, periodic samples, and periodic-periodic samples.

Cáceres et al. [7] provided indexes in the context of huge data sets for variable length-gapped (VLG) patterns. These variable-length gapped patterns include two or more sub-patterns. The authors described that the results of their study are much better and faster in terms of searching patterns and texts and showed that “in practice, on a broad range of inputs typical in real applications of VLG matching, simple algorithms based on intersecting ranges of the suffix array corresponding to subpattern occurrences can be made very fast in practice, and comfortably outperform state-of-the-art methods based on wavelet trees.”

In Clifford et al. [9], the authors described their work by considering dynamic and online variations of 2D pattern matching where pattern size is \(m \times m\), and the size of the text is \(n \times n\). In this work, the authors claim that in the dynamic 2D match, individual symbols can be updated in \(O(\log^2 n)\) and queries can be performed in \(O(\log^2 m)\). In their work, authors focused on Dynamic Text Static Pattern Matching in 2D, Online Exact Hamming Distance in 2D, Online Approximate Hamming Distance in 2D, Sub-matrix Near Neighbour Problem.

Amir et al. [1] introduced the concept of maximal 2-dimensional replications. In their work, authors propose an efficient algorithm on an upper limit for the replications of occurrences of patterns in \(n \times n\) array. The algorithm proposed by authors is efficient in terms of runtime that is \(O(n^2 \log n + \rho)\), “where \(n^2\) is the size of the input array and \(\rho\) is the number of maximal 2D repetitions in the output”.
3. Proposed Algorithms

In this paper, two new and efficient 2D pattern matching algorithms are proposed. The first algorithm is developed considering the 2D pattern of binary sequences, that is, the sequence of 0 and 1 and the second algorithm for decimal sequences. These proposed algorithms are linear run time algorithms.

3.1. 2D Pattern Matching Algorithm for Binary Patterns

The objective of this algorithm is to find the 2D pattern in a given text. This algorithm has the scope of finding patterns of binary sequences, that is, for black and white images.

**Procedure**

A two-dimensional pattern matching problem is to locate pattern \( P \) \((p \times q)\) inside in 2D text \( T \) of size \((m \times n)\) as shown in fig. 3.

\[
\begin{align*}
\text{Pattern } P & = \\
& = \begin{bmatrix}
P_{1,1} & P_{1,2} & \ldots & P_{1,q} \\
P_{2,1} & P_{2,2} & \ldots & P_{2,q} \\
\vdots & \vdots & \ddots & \vdots \\
P_{p,1} & P_{p,2} & \ldots & P_{p,q}
\end{bmatrix} \\
\text{Text } T & = \\
& = \begin{bmatrix}
T_{1,1} & T_{1,2} & \ldots & T_{1,n} \\
T_{2,1} & T_{2,2} & \ldots & T_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
T_{m,1} & T_{m,2} & \ldots & T_{m,n}
\end{bmatrix}
\end{align*}
\]

Fig. 3. Pattern and Text

The procedure to solve the problem consists of the following steps:

**Step 1:** First, we calculate the value of the pattern, using the following equation:

\[
P = \sum_{i=1}^{p} \sum_{j=1}^{q} P[i,j] \cdot d^{(i-1)q+j}
\]
Step 2: Now, the next step is to identify the cofactors in the given text.

As we know, the number of cofactors in the given text ‘T’ of dimension (p x q) will be 
\((m - p + 1) \times (n - q + 1)\), where \(1 \leq p \leq m\) and \(1 \leq q \leq n\).

Step 3: Now, for each cofactor, compute the sum of the first row, second row up to the pth row. Then, compute the sum of the first row and check with the sum of the first row of pattern P.

Step 4: If this sum matches, then go to the second row and repeat the process. If the sum of all the rows of cofactor matches with the sum of rows of the pattern, then there is a possibility of pattern matching, which will further be verified with one to one character matching row-wise.

Step 5: If all the characters match for row one, then proceed for the next row. If the sum for a particular row does not match, discard that cofactor. If character to character for a particular row does not match, discard that cofactor. Row wise sum matches and character to character again matches for each row of cofactor and pattern the pattern exit in the text T, else not.

We can define the algorithm for 2D pattern matching for binary patterns as:

**Algorithm**

1. Count = 0;
2. \(P = \sum_{i=1}^{p} \sum_{j=1}^{q} P[i, j]d^{(i-1)q+j};\)
3. for \(t = 0\) to \((m - p)\) do; // where ‘t’ is the horizontal shift
4. for \(r = 0\) to \((n - q)\) do; // where ‘r’ is the vertical shift
5. \(d = |\Sigma|;\) // Length of alphabet
6. \(COF_n(t, r) = \sum_{i=t+1}^{t+p} \sum_{j=r+1}^{r+q} T[i, j]d^{(i-(t+1)q)+(j-r)};\)
7. if \((P_n = COF_n(t, r));\)
8. Then Pattern occurred with horizontal-vertical shift (t, r);
9. Else Pattern not occurred;

The run time complexity of algorithm is: \(O ((m - p + 1) (n - q + 1))\).

**Numerical simulation**

In this section, we present a numerical simulation of the proposed algorithm. For the implementation of the proposed method, consider the pattern P having dimensions p and q (3 x 3) and the text T having dimensions m and n (6 x 6) as shown in fig. 4.
An efficient algorithm for 2-dimensional pattern matching problem

Pattern P =

|   |   |   |   
|---|---|---|---|
| 1 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

Text T =

|   |   |   |   |   |   
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |

Fig. 4. Pattern and Text

**Step 1:** First, we calculate the value of the pattern. According to the algorithm, the value of pattern can be calculated as:

\[ P = 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 + 0 \times 2^8 + 1 \times 2^9 \]

\[ P = 2 + 4 + 0 + 16 + 32 + 0 + 128 + 512 \]

\[ P = 694 \]

**Step 2:** Now, the next step is to identify the cofactors in the given text. From the given example, as defined in fig. 4, \( m = 6, n = 6, p = 3, \) and \( q = 3. \)

**Step 3:** Now, for each cofactor compute the sum of the first row, second row up to the \( p \)th row. Then, compute the sum of the first row and check with sum of the first row of pattern \( P. \)

**At shift \( t = 0, r = 0: \)**

Starting from the horizontal shift \( t = 0, \) and vertical shift \( r = 0, \) we have:
Fig. 5. At shift $t = 0$, $r = 0$

$$\text{COFn} (0, 0) = 1 \times 21 + 0 \times 22 + 1 \times 23 + 1 \times 24 + 1 \times 25 + 1 \times 26 + 1 \times 27 + 0 \times 28 + 0 \times 29$$
$$= 2 + 0 + 8 + 16 + 32 + 64 + 128$$
$$\text{COFn} (0, 0) = 250$$

**Step 4:** We can observe that $P = 694$, and $\text{COFn} (0, 0) = 250$, therefore $P \neq \text{COFn} (0, 0)$. Hence pattern does not occur.

Now, we repeat the same process for all the cofactors.

**At shift $t = 1$, $r = 0$:**

Similarly, at horizontal shift $t = 1$, and vertical shift $r = 0$, as shown in fig. 6, we have:

$$\text{COFn} (1, 0) = 1 \times 21 + 1 \times 22 + 1 \times 23 + 1 \times 24 + 1 \times 25 + 0 \times 26 + 1 \times 27 + 0 \times 28 + 1 \times 29$$
$$= 2 + 4 + 8 + 16 + 0 + 0 + 128 + 512$$
$$\text{COFn} (1, 0) = 670$$

We can observe that $P = 694$, and $\text{COFn} (1, 0) = 670$. Therefore, $P \neq \text{COFn} (0, 1)$. Hence, the pattern does not occur.
At shift $t = 2, r = 3$:

Similarly, at horizontal shift $t = 2$, and vertical shift $r = 3$, as shown in fig. 7, we have:

\[
\begin{array}{cccccc}
1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Fig. 7. At shift $t = 2, r = 3$

\[
\text{COFn} (2, 3) = 1 \times 21 + 1 \times 22 + 0 \times 23 + 1 \times 24 + 1 \times 25 + 0 \times 26 + 1 \times 27 + 0 \times 28 + 1 \times 29 \\
= 2 + 4 + 0 + 16 + 32 + 0 + 128 + 512 \\
\text{COFn} (2, 3) = 694
\]

We can observe that $P = 694$, and $\text{COFn} (2, 3) = 694$. Therefore, $P = \text{COFn} (2, 3)$. Hence, pattern occurred at shift $t = 2, r = 3$.

**Brute force approach**

The number of cofactors are $N = (m - p + 1) \times (n - q + 1)$. Each cofactor has dimension $(p \times q)$, to check existence of pattern each symbol is matched, so the number of comparison will be $(p \times q)$ for each cofactor, since the number of cofactors is $(m - p + 1) \times (n - q + 1)$. Thus, the complexity will be $O (pq (m - p + 1) \times (n - q + 1))$.

**Lemma:** If $a$ and $b$ are positive integers, then if $a \equiv b \pmod{q}$ then $a \neq b$, where $q$ is a prime number.

**Proof:** Given $a \equiv b \pmod{q}$

Let $a \equiv a1$

$b \equiv b1$

$\Rightarrow a = kq + a1, \quad 0 \leq a1 < q$

$\Rightarrow b = sq + b1, \quad 0 \leq b1 < q$

Method of contradiction

Suppose $a = b$

$\Rightarrow kq + a1 = sq + b1$
\[ (k-s)q = b1 - a1 \]
\[ b1 - a1 = tq \quad \text{where} \quad (k-s) = t, \quad \text{and} \quad t > 0 \]

Since \( a1 \neq b1 \)
\[ b1 - a1 \neq 0 \]
\[ b1 = a1 + tq, \quad t > 0 \]
\[ b1 \geq q, \quad a1 > 0, \quad \text{that is a contradiction, which arises when we take} \quad a = b. \]

Thus \( a \neq b \)

### 3.2. 2D Pattern Matching Algorithm for Decimal Numbers

In this section, we propose the modified algorithm for 2D pattern matching considering the decimal patterns. The objective of this algorithm is to find the 2D pattern in a given text. This algorithm has the scope of finding patterns of decimal sequences.

**Procedure**

A two-dimensional pattern matching problem for decimal patterns is to locate pattern \( P \) (\( p \times q \)) inside in 2D text \( T \) of size \( (m \times n) \), as shown in fig. 8.

The number of cofactors in the given text ‘\( T \)’ of dimension \( (p \times q) \) will be \( (m - p + 1) \times (n - q + 1) \), where \( 1 \leq p \leq m \) and \( 1 \leq q \leq n \).

The procedure to solve the problem, the following steps are suggested:

**Step 1:** Consider a prime number say, \( q \).

**Step 2:** Convert the rows in decimal number by cascading the number written in each cell, row-wise. Convert sequence row-wise into a decimal number, taking mod with respect to \( q \). We denote the achieved values as \( K_i \).

**Step 3:** Now, convert \( K_i \)'s into a decimal number and then, take the mod with respect to \( q \). We denote this value as \( K_p \).

**Step 4:** Now, consider the Text, and start from the cofactor \( (0, 0) \). Convert the rows in decimal number by cascading the number written in each cell, row-wise. Convert sequence row-wise into a decimal number, taking mod with respect to \( q \). We denote the achieved values as \( C_i \).

**Step 5:** Now, convert \( C_i \)'s into a decimal number and then take the mod with respect to \( q \). We denote this value as \( C_p \).

**Step 6:** Now check if \( K_p = C_p \).

**Step 7:** If \( K_p \) and \( C_p \) are not equal, then there is no need to check character by character. But if \( K_p \) and \( C_p \) are the same, then it is possible that any cofactor could produce the same value.
of Cp as that of Kp, for different Ci and Ki, that is, for dissimilar symbols of the pattern and text. In such cases, we match the characters of the pattern with the characters of text.

**Algorithm**

When the symbols are decimal numbers, (Decimal (P, T, q)

1. Prime number q.
2. for i = 1 to p do
   
   //Pattern sequence//
   
   (Convert sequence row wise in to decimal number) mod q = Ki
   
   (Convert Ki’s into decimal number and then Kip) mod q = Kp
3. for t= 0 to (m-p) do
4. for r= 0 to (n-q) do
   
   //cofactor C(t, r) sequence//
   
   (Convert sequence row wise in to decimal number) mod q = Ci
   
   (Convert Ci’s sequence in to decimal number-Cic) mod q = Cp
5. if (Kp = = Cp)
6. Count = 0;
7. then for i = 1 to p do
   
   if (Ki = = Ci)
      count++;
   if (Count = = p)
      then match each symbol of cofactor C(t, r) row wise with pattern P.
      if matched pattern exist
      else not;
8. All matches -Pattern occurred with horizontal-vertical shift (t, r);
9. else not;
Complexity

Using Rabin Karp algorithm the complexity to find Kp is \( O(p \cdot q) \) and to find all \( C_p \) is \( O(p \cdot q \cdot (m - p + 1)(n - q + 1)) \) and then matching of Pattern symbol wise takes time \( O(p \cdot q) \). Thus, the total time in worst case would be \( O(p^2 \cdot q^2 \cdot (m - p + 1)(n - q + 1)) \).

Numerical simulation - Decimal Number

In this section, we are presenting a numerical simulation for the proposed algorithm for pattern matching of decimal numbers. As shown in fig. 8, consider the pattern \( P \) having dimensions \( p \) and \( q \) (3 x 3), and the text \( T \) having dimensions \( m \) and \( n \) (6 x 6) as shown in fig. 8. In this simulation, we discuss two cases where the pattern has not occurred and a case when the pattern occurred in the text.

\[
P = \begin{bmatrix}
9 & 4 & 6 \\
4 & 2 & 0 \\
1 & 2 & 1
\end{bmatrix}
\]

\[
T = \begin{bmatrix}
2 & 0 & 0 & 9 & 2 \\
4 & 2 & 1 & 0 & 5 \\
3 & 0 & 2 & 6 & 8 \\
1 & 9 & 4 & 6 & 0 \\
2 & 4 & 2 & 0 & 3 \\
1 & 1 & 2 & 1 & 2
\end{bmatrix}
\]

Fig. 8. Pattern and Text

Let’s assume that the prime number ‘q’ is taken as 7. Convert the rows in decimal number by cascading the number written in each cell, row-wise. Convert sequence row-wise into a decimal number, taking mod with respect to q. We denote the achieved values as \( K_i \), as shown in fig. 9.

\[
\begin{array}{cccc}
9 & 4 & 6 & 946 \\
4 & 2 & 0 & 420 \\
1 & 2 & 1 & 121 \\
\end{array} \quad \begin{array}{ccc}
\% & 7 & = 1 \\
\% & 7 & = 0 \\
\% & 7 & = 2 \\
\end{array}
\]

Fig. 9. Values of \( K_i \)
An efficient algorithm for 2-dimensional pattern matching problem

Now, convert Ki’s into a decimal number and then take the mod with respect to q. We denote this value as Kp, represented in fig. 10.

\[
\begin{array}{ccc}
1 & 0 & 2 \\
\hline
102 & \% & 7 \\
\hline
4 & = & Kp
\end{array}
\]

**Fig. 10. Values of Kp**

We find that the value of \(Kp = 4\), which is the decimal equivalent of the given pattern is 4. Now, consider the given text T. In this simulation, we discuss two cases where the pattern has not occurred and a case when pattern occurred in the text. We start from the cofactor \((0, 0)\), as shown in fig. 11.

**At shift \(t = 0, r = 0\):**

\[
\begin{array}{cccc}
2 & 0 & 0 & 9 \\
\hline
4 & 2 & 1 & 0 \\
\hline
3 & 0 & 2 & 6 \\
\hline
1 & 9 & 4 & 6 \\
\hline
2 & 4 & 2 & 0 \\
\hline
1 & 1 & 2 & 1 \\
\end{array}
\]

**Fig. 11. At shift \(t = 0, r = 0\)**

Let’s assume that the prime number ‘q’ is taken as 7. Convert the rows in decimal number by cascading the number written in each cell, row-wise. Convert sequence row-wise into a decimal number, taking mod with respect to q. We denote the achieved values as Ci, shown in fig. 12.

\[
\begin{array}{ccc}
2 & 0 & 0 \\
\hline
200 & \% & 7 \\
\hline
4 & 2 & 1 \\
\hline
421 & \% & 7 \\
\hline
3 & 0 & 2 \\
\hline
302 & \% & 7 \\
\hline
4 & = & C1 \\
\hline
1 & = & C2 \\
\hline
1 & = & C3
\end{array}
\]

**Fig. 12. Values of Ci**

Now, convert Ci’s into a decimal number and then take the mod with respect to q. We denote this value as Cp, presented in fig. 13.
Fig. 13. Values of $C_p$

We can observe that $K_p = 4$, and $C_p = 5$, that is, $K_p \neq C_p$. Hence, the pattern does not exist.

At shift $t = 1$, $r = 3$:

At horizontal shift $t = 1$, and vertical shift $r = 3$, as shown in Fig. 14, we have:

\[
\begin{array}{cccc}
2 & 0 & 0 & 9 \\
4 & 2 & 1 & 0 \\
3 & 0 & 2 & 6 \\
1 & 9 & 4 & 6 \\
2 & 4 & 2 & 0 \\
1 & 1 & 2 & 1 \\
\end{array}
\]

Fig. 14. At shift $t = 1$, $r = 3$

As we have taken the prime number ‘q’ as 7, convert the rows in decimal number by cascading the number written in each cell, row-wise. Convert sequence row-wise into a decimal number, taking mod with respect to q. We denote the achieved values as $C_i$ in Fig. 15.

\[
\begin{array}{cccc}
9 & 4 & 6 & 946 \\
4 & 2 & 0 & 420 \\
1 & 2 & 1 & 121 \\
\end{array}
\]

Fig. 15. Values of $C_i$

Now, convert $C_i$’s into a decimal number, and then, take the mod with respect to q. We denote this value as $C_p$, as shown in Fig. 16.

\[
\begin{array}{cccc}
1 & 0 & 2 & 102 \\
\end{array}
\]

Fig. 16. Values of $C_p$

We can observe that $K_p = 4$, and $C_p = 4$, that is, $K_p = C_p$. Hence, the pattern exists at shift $t = 1$, $r = 3$. 
An efficient algorithm for 2-dimensional pattern matching problem

It is quite possible that any cofactor can produce the same Cp value as that of Kp, for different Ci and Ki, that is, for dissimilar symbols of the pattern and text. Consider the cofactor at shift \( t = 2, r = 0 \), as shown in fig. 17.

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>0</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 17. At shift \( t = 2, r = 0 \)

As we have taken the prime number ‘\( q \)’ as 7. Convert the rows in decimal number by cascading the number written in each cell, row-wise. Convert sequence row-wise into a decimal number, taking mod with respect to \( q \). We denote the achieved values as \( C_i \), as shown in fig. 18.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>9</th>
<th>2</th>
<th></th>
<th>092</th>
<th>% 7 = 1</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td>105</td>
<td>% 7 = 0</td>
<td>C2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td>268</td>
<td>% 7 = 2</td>
<td>C3</td>
</tr>
</tbody>
</table>

Fig. 18. Values of \( C_i \)

Now, convert \( C_i \)’s into a decimal number and then, take the mod with respect to \( q \). We denote this value as \( C_p \), as shown in fig. 19.

|   | 1 | 0 | 2 |   | 102 | \% 7 = 4 | \( C_p \) |

Fig. 19. Values of \( C_p \)

We can observe that \( K_p = 4 \), and \( C_p = 4 \), that is, \( K_p = C_p \).

In such cases, we match the characters of the pattern with the characters of the text. We find that the first character of pattern is 9, whereas the corresponding character of the cofactor (2, 0) is 0. Hence, the pattern does not occur.

In this section, we describe a case study of matching security codes from the repository of codes. Security code provided to the authorized users is dynamic, that is, it is provided to them at the run time or the demand of the user. The size of the security code provided to the users is of variable length and is of size $3 \times 3$, $9 \times 9$, and $16 \times 16$, that is, security codes may contain 9, 81, or 256 binary or decimal digits. These security codes are matched with the codes available in the repository. The collections of a variety of codes are kept in the form of either binary digits or numbers in the repository. We take the collection of codes in two different forms, as first is the case when the security code contains only binary digits, and second is the case when the security code contains decimal numbers.

**Case 1: When the security code contains only binary digits**

This is the case when the security code is formed using only binary digits, that is, the combinations of 0’s and 1’s. In this case study, we are considering repository of two different sizes: a) Repository with size $250 \times 250$, b) Repository with size $500 \times 500$. In these repositories of different sizes, we consider three different sizes of security code, including $3 \times 3$, $9 \times 9$, and $16 \times 16$. We compare the proposed method with the brute force approach.

**a) Repository having size $250 \times 250$**

In the size of $250 \times 250$ repositories, we have a security code of three variations that is of size $3 \times 3$, $9 \times 9$, and $16 \times 16$. Figure 20 shows the number of comparisons in every situation and compares it with the brute force method.

Here we observe that:

- When the size of the security code is $3 \times 3$, and the size of the repository is $250 \times 250$, then the number of cofactors = 61504.
- When the size of the security code is $9 \times 9$, and the size of the repository is $250 \times 250$, then the number of cofactors = 58564.
- When the size of the security code is $16 \times 16$, and the size of the repository is $250 \times 250$, then the number of cofactors = 55225.

Now, the number of comparisons is calculated using the proposed algorithm and compared with the brute force method.
An efficient algorithm for 2-dimensional pattern matching problem

**Fig. 20.** Repository having size 250 × 250 when security code contains only binary digits

**b) Repository having size 500 × 500**

In the size of 500 × 500 repositories, we have security code of three variations that is of size 3 × 3, 9 × 9, and 16 × 16. Figure 21 shows the number of comparisons in every situation and compares it with the brute force method.

Here we observe that:

When the size of the security code is 3 × 3, and the size of the repository is 500 × 500, then the number of cofactors = 248004.

When the size of the security code is 9 × 9, and the size of the repository is 500 × 500, then the number of cofactors = 242064.

When the size of the security code is 16 × 16, and the size of the repository is 500 × 500, then the number of cofactors = 235225.

Now, the number of comparisons is calculated using the proposed algorithm and compared with the brute force method.
5. Conclusion

The proposed algorithm is applicable to solve the 2D pattern matching problems in which symbols are decimal numbers, binary numbers, characters and dot patterns with slight modification. Further, it can be modified to match the 2D figures and to solve face matching problem. The algorithm is simple and easy, and its complexity is the same as some popular algorithms available to solve the 2D pattern matching problem. The proposed work is divided into two parts. One part is devoted to binary number algorithm, and the other part is for the decimal number algorithm. The worst-case time complexity of the proposed algorithm is \(O(pq (n - p + 1)(m - q + 1))\).
6. References


313