Efficient Algorithm for Two Dimensional Pattern Matching Problem (Square Pattern)

Hussein Abu-Mansour[1], Jaber Alwidian[1], Wael Hadi[2]
[1]ITC department
Arab Open University
Riyadh- Saudi Arabia
[2]CIS department
Philadelphia University
Amman- Jordan
{hmansour, j.alwidyan}@arabou.edu.sa,whadi@philadelphia.edu}

ABSTRACT: *In recent decades, our computers tolerate multidimensional data to be stored and maintained depending on the high computational strength of these computers. As a result, the two-dimensional pattern matching considered as one of the hot research areas. In this paper, we investigate the exact two dimensional pattern matching problem. A new algorithm to solve the main problem in this field is proposed which converts the two dimensional problem into one dimensional problem before searching process which required high computational cost in preprocessing phase. The proposed algorithm deals with white and black images, specifically, with text images. Experimental results show the superiority of the proposed algorithm when comparing with (Brute-force algorithm, RK-KMP algorithm). The results of our algorithm point to the number of comparisons have been reduced in many cases. In another side, when we apply the proposed algorithm on one dimensional text, its time is proportional to* $\theta(n)$*, that means our time in this case is linear.*

Keywords: Pattern Matching, Two-dimensional Pattern Matching, RK-KMP Algorithm, Brute-force Algorithm

Received: 25 January 2012, Revised 19 March 2012, Accepted 26 March 2012

1. Introduction

Three major efficient pattern matching algorithms were proposed in the recent decade, KMP algorithm [13], BM algorithm [4], and RK algorithm [8] these algorithms are one dimensional pattern matching algorithms. Baker [3] and independently Bird [5] introduced the first worst case linear time algorithm for two dimensional pattern matching. The main step in their algorithm is to run finite automation in order to perform a linear scan on text. In recent years, there has been unceasing interest in two and more dimensional pattern matching problems; such interest is substantiated by the growing computational strength of the computers allowing multidimensional data e.g. scans photographs to be processed [17].Several algorithms have been presented by researchers for the exact two dimensional pattern matching problem, which have been reported in [14].

The two dimensional pattern matching problem can be classified into six types as follow: (1) Exact two dimensional matching, (2) Approximate matching of rectangular patterns, (3) Approximate matching of non rectangular patterns, (4) Scaled matching, (5) Compressed matching. And (6) Dictionary matching.

2. Problem Statement

String matching is an important problem in text processing and is commonly used to locate one dimensional pattern (string) in a text. The expansion of imaging, graphics and multimedia required the use of pattern matching to higher dimensions leading to the two- and multi-dimensional pattern matching problem [7]. The main problem of two dimensional pattern matching is the huge number of comparisons which are needed to find occurrence of two-dimensional pattern in two- dimensional text. Most of the algorithms that have been proposed to solve this problem are used to reduce the two dimensional problem into one dimensional problem, and this process is considered as new problem in these algorithms because of the need to for preprocessing phase to construct the necessary structure before searching the text [7].

In two dimensional patterns matching problem there are different types of text which can be used such as: binary alphabet, alphabet of size 8, English alphabet, and DNA alphabet [16] and [7]. In this research we will focus on binary text. For example, consider two matrices *X* and *Y* defined as show in Figure 1, *X* as a Text and *Y* as a Pattern and we need to find the occurrence of *Y* in *X*.

X =

0	0	1	0	0
0	1	1	1	0
1	1	0	0	1
0	1	1	0	0
0	0	0	1	1

Y =

1	0	0
1	1	0
0	0	1

Figure 1. Two binary matrices *X* and *Y*

The main objective of this paper is to propose an efficient algorithm in exact 2D pattern matching problem when alphabet is binary {0, 1} without the reduction of the two dimensional arrays into one dimension ones. In fact, reducing the two dimensional pattern into one dimensional pattern is considered as a problem in many algorithms because of the need to preprocessing phase to construct the necessary structures before searching the text [7]. The importance of this algorithm lies in the detection of an object in white and black images which is considered as one of the main problems in pattern recognition and image processing [6]. Also, binary data have been occupying a special place in the domain of data analysis [15].

This paper is structured as follows: the problem statement is presented in Section II. In Section III, the related work is presented. The proposed algorithm is discussed in section IV. Experimental results are presented and discussed in section V. Finally, conclusions are given in Section VI.

3. Related Work

The two dimensional pattern matching problem is the first generalization of the one dimensional pattern matching [7]. In one dimensional pattern matching problem we need to find all occurrences of a pattern in a text string where most of algorithms in this field focus on pattern, and the searching in the text is done by using one direction (from left to right) [10] as shown in Figure 2a. But in two dimensional pattern matching we need to find all occurrences of a two dimensional pattern in a two dimensional text matrix as shown in figure 2b.

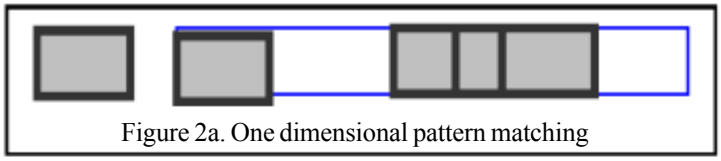


Figure 2a. One dimensional pattern matching

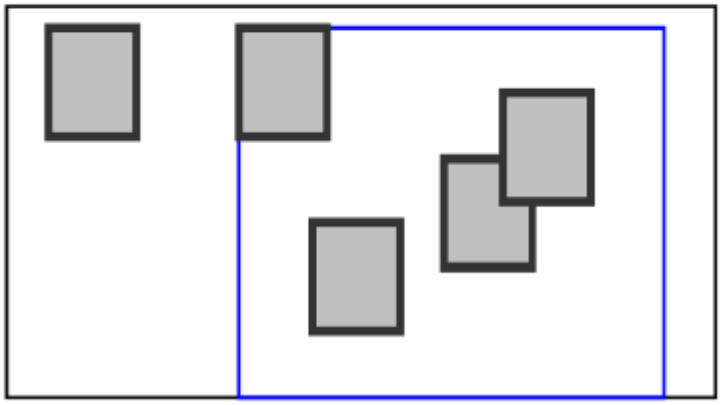


Figure 2b. Two dimensional pattern matching

The two dimensional pattern matching problem can be classified into six types: (1)Exact two dimensional pattern matching, (2) Approximate matching of rectangular pattern, (3) Approximate matching of non rectangular pattern, (4) Scaled matching, (5) Compressed matching, and (6) Dictionary matching [1]. The most important types of two dimensional pattern matching problem in practice are exact two dimensional pattern matching problem and two dimensional approximate pattern matching [17]. In this section we will focus on the first, second, and third types. The exact two dimensional pattern matching problem is defined in [1] as: Let q be an alphabet, given a text array T (n * n) and pattern array P (m * m), report all locations (i, j) in T where there is an occurrence of P, i.e. T (I + k, j + l) = P (k, l) for 0 <= k, l <= n and m <= n.

Many of applications adopts the exact 2-D pattern matching algorithms such as the edge detecting process for any type of images, where a set of arrays of edge detectors are matched against the picture [9]. One of the applications that presents the importance of pattern matching problem is compressed matching problem, defined in [2] it is the problem of finding all occurrences of pattern in compressed images. The compressed matching problems motivated by the vast increase of stored compressed data, is the problem of finding all occurrences of pattern in a compressed text without the need to decompress this image [2]. Another application that shows the importance of the problem was presented in [1] which is about searching aerial photographs. The practical goal of this application is to read an aerial photograph and a template of some patterns, and the output is all locations on the aerial photograph where the template abject appears.

Many of the studies focus on exact 2-D pattern matching problem when the alphabet is binary. This problem appears in the detection of an object in white and black image that is considered as one of the main problem in pattern recognition and image processing [6]. Also, Binary data have been occupying a special place in the domain of data analysis [1]. In [15] the exact 2-D matching algorithms were classified based on two criteria (1) Reduction of the 2-D pattern matching problem into one dimension such as (Bird and Baker, Weiner's or Creight's algorithm) and (2) The analysis of the 2-D structure of the pattern and make use of it in text scanning step such as (2-D periodicity idea).

4. The Proposed Algorithm

In this section, a comprehensive explanation about the proposed algorithm is presented. The proposed algorithmconsists mainly of two phases, the preprocessing phase and the matching phase. Figure 3 depicts the block diagram of the main steps of the proposed algorithm.

4.1 Preprocessing Phase

With regard to what have been mentioned in the research objectives the proposed algorithm treats with '0's and '1's matrices

white and black images), our algorithm conducts preprocessing step to construct the necessary structure before searching the text. Let Pattern size is $n * m$, Time is $O(3 * n * m)$ which can be divided into three parts as follows:

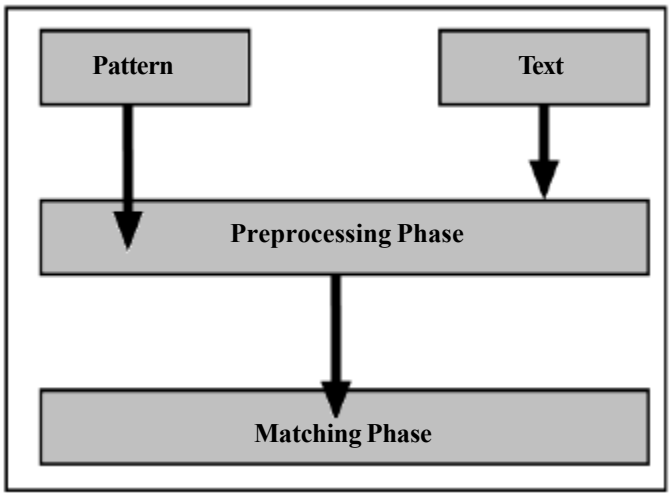


Figure 3. The main steps of the proposed algorithm

1. let

$$Pattern = \begin{bmatrix} a11 & a12 & .... & a1m \\ a21 & a22 & .... & a2m \\ \vdots & \vdots & & \vdots \\ an1 & an2 & .... & anm \end{bmatrix}$$

$$Text = \begin{bmatrix} b11 & b12 & .... & b1m \\ b21 & b22 & .... & b2m \\ \vdots & \vdots & & \vdots \\ bn1 & bn2 & .... & bnm \end{bmatrix}$$

We need to convert the pattern into unique value first by using the following formula:

$$Pattern\ value = a11 * (2^1) + a12 * (2^2) + ...anm * (2^{(n*m)}) \quad (1)$$

By using this formula we will have fragment from the geometric form or full geometric form as shown in following formula:

$$2^1 + 2^2 + 2^3 + ... + 2^n = (2^{(n+1)} - 2) \quad (2)$$

For example, if we have the following pattern:

$$Pattern = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

By using formula 1, value of this pattern = $1 * (2^1) + 0 * (2^2) + 1 * (2^3) + 1 * (2^4) + 0 * (2^5) + 0 * (2^6)$

$= 1 * (2^1) + 1 * (2^3) + 1 * (2^4)$

$= 2^1 + 2^3 + 2^4$

The resulted value is considered as geometric form fragment as shown in formula 2, which is unique value , that means there should not be any pattern that has the same value with different geometric form fragment.

Proof 1: assume the Pattern = 101100, then

- Change the element which comes after the last '1' in the Pattern to '1'. In this example the pattern will become 101110.

- Change all elements that come before the last '1' in the new Pattern to zeroes. In this example the pattern will become 000010.
- By using formula 1 and 2, the value will be greater than the previous value. So that change any element will give unique value.

If the size of the Pattern equal n*m, the time of this part is proportional to O (n*m). The Pseudo Code is as shown in Figure 4.

```
1  Position =1
2  Hashsub = 0
3  // Hashsub is the value of the pattern
4  // size if pattern = n * m
5  // n is number of rows
6  // m number of columns
7  // tem is one dimension array include values from 2^1 to 2^(n*m)
8  // sub is a Pattern
9  For i =1 to n
10 For j =1 to m
11 Hashsub = Hashsub + sub(i) (j)*tem ( position)
12 End for
13 End for
```

Figure 5(a). The pseudo code of part 1

2. Convert the first pattern which can be created from the text into one value by using the same way which used in the previous part where its time is proportional of $O(n * m)$. For example, let

$$Text = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The first pattern that can be created from the Text will be:

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Value of this pattern by using formula $1 = 0 * (2^1) + 1 * (2^2) + 0 * (2^3) + 1 * (2^4) + 0 * (2^5) + 1 * (2^6)$
$= 1 * (2^2) + 1 * (2^4) + 1 * (2^6)$

The Pseudo Code of this part is as shown in figure 5.

3. Create one dimensional array that has the same size of pattern (n*m). for example, if the pattern has size equal 2 * 3 , then the size of one dimensional array will be equal 6. This array contains values from 21 to 2^{n*m} . For example,

$$\text{If the pattern} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The size of new one dimensional pattern will be equal 6 elements, and these elements are {2, 4, 8, 16, 32, and 64}. The Pseudo Code of this part is as shown in figure 6, where time of this part is proportional to O (n*m).

4.2 Matching Phase

The basic procedure of the proposed algorithm is derived from the Karp and Rabin algorithm. The order of comparisons is performed from left to right until a complete match occurs.

This example will be explained in an example as follows:

```
1  Hashmain = 0
2  Position =1
3  // Hashmain is the value of the first pattern which is created from the Text
4  // size if pattern = n * m
5  // n is number of rows
6  // m number of columns
7  // tem is one dimension array include values from 2^1 to 2^(n*m)
8  // largematrix is a Text
9  For i =1 to n
10 For j =1 to m
11 Hashmain = Hashmain + largematrix (i)(j) * tem ( position)
12 End for
13 End for
```

Figure 5(b). Pseudo code of part 2

Let

$$Text = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$Pattern = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

In this example, a comparison between the value of the pattern and the first pattern that is derived from the text should be done in order to find a match between them. In the case of no match, further steps should be applied to find the match:

1. To get a new pattern from the text, a shift to the right by one column should occur to the whole pattern if possible. As shown in figure 3.5.

```
1 tem (1) = 2
2 // tem is one dimensional array
3 // n*m size of the pattern
4 For I = 2 to n*
5 tem (i) = tem (i -1) * 2
6 End for
```

Figure 6. The pseudo code of part 3

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The current pattern The new pattern

Figure 3.5. Create new pattern by shifting to the right one column

In this case, the following formula is used to calculate the value of this new pattern:

$$\text{Value of the next Pattern} = (PV - FCPP)/2 + LCNP$$

Where

PV is Previous Value

FCPP is First Column of the Previous Pattern

LCNP is Last Column of the Next Pattern

Applying formulas 1 and 3, to the example:

Let

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The current pattern The new pattern

First step (PV - FCPP) / 3 should be found:

$$\begin{bmatrix} 0 & 2^2 & 0 \\ 2^4 & 0 & 2^5 \end{bmatrix} - \begin{bmatrix} 0 \\ 2^4 \end{bmatrix} = \begin{bmatrix} 2^2 & 0 \\ 0 & 2^6 \end{bmatrix} / 2$$

(PV) (FCPP) (PV - FCPP) / 2

Second step, summation of (PV - FCPP)/2 and LCNP should be found:

$$= \begin{bmatrix} 2^1 & 0 \\ 0 & 2^5 \end{bmatrix} - \begin{bmatrix} 0 \\ 2^6 \end{bmatrix} = \begin{bmatrix} 2^2 & 0 & 0 \\ 0 & 2^5 & 2^6 \end{bmatrix}$$

(PV - FCPP)/3 LCNP = (PV - FCPP)/2 + LCNP

2. in the case of shifting pattern to right by one column is no longer available, a shift down by one row to the current pattern should be done if possible, in order to come up with new pattern from a certain text. As shown in figure 3.6.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The current pattern The new pattern

Figure 3.7. Create new pattern by shifting down by one row

In this case, the following formula is used to calculate the value of this new pattern.

$$\text{Value of the Pattern} = (PV - FRPP) / 2^m + LRNP$$

Where

PV is Previous Value

FRPP is First Row of the Previous Pattern

LRNP is last row of the Next Pattern

m is the number of columns in the Pattern

Applying formulas 1 and 4 to the example

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

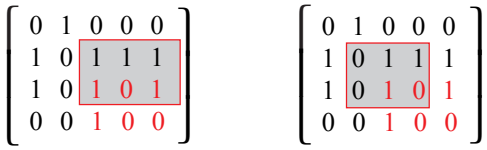
The current pattern The new pattern

Using the formulas 1 and 4, the result will be as follow:

$$\begin{bmatrix} 0 & 0 & 0 \\ 2^4 & 2^5 & 2^6 \end{bmatrix} - [0\ 0\ 0] = [\,2^4 \;\; 2^5 \;\; 2^6\,]$$

$$= [\,2^1 \;\; 2^2 \;\; 2^3\,] + [\,2^4 \;\; 0 \;\; 2^6\,] = \begin{bmatrix} 2^1 & 2^2 & 2^3 \\ 2^4 & 0 & 2^6 \end{bmatrix}$$

3. In the case of no match between the patterns that has been created from step 2 and the pattern, a shift to the left by one column should be done, and shifting should be repeated to the left has to be done until a match with the pattern is found or no more shifting is available due to no more new columns in the text. As shown in figure 3.8.



The current pattern The new pattern

Figure 3.8. Create new pattern by shifting to left by one column

In this case the following formula is used to calculate the value of the new pattern:

$$\text{Value of the next Pattern} = (\text{PV} - \text{LCPP}) * 2 + \text{FCNP}$$

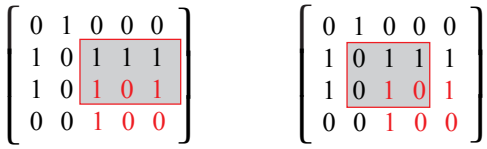
Where

PV is Previous Value

LCPP is Last Column of the Previous Pattern

FCNP is First Column of the Next Pattern

Applying formulas 1 and 5 to the example



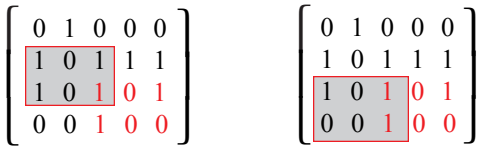
The current pattern The new pattern

Using the formulas 1 and 5, the result will be as follow:

$$\begin{bmatrix} 2^1 & 2^2 & 2^3 \\ 2^4 & 0 & 2^6 \end{bmatrix} - \begin{bmatrix} 2^3 \\ 2^6 \end{bmatrix} = \begin{bmatrix} 2^1 & 2^2 \\ 2^4 & 0 \end{bmatrix} * 2$$

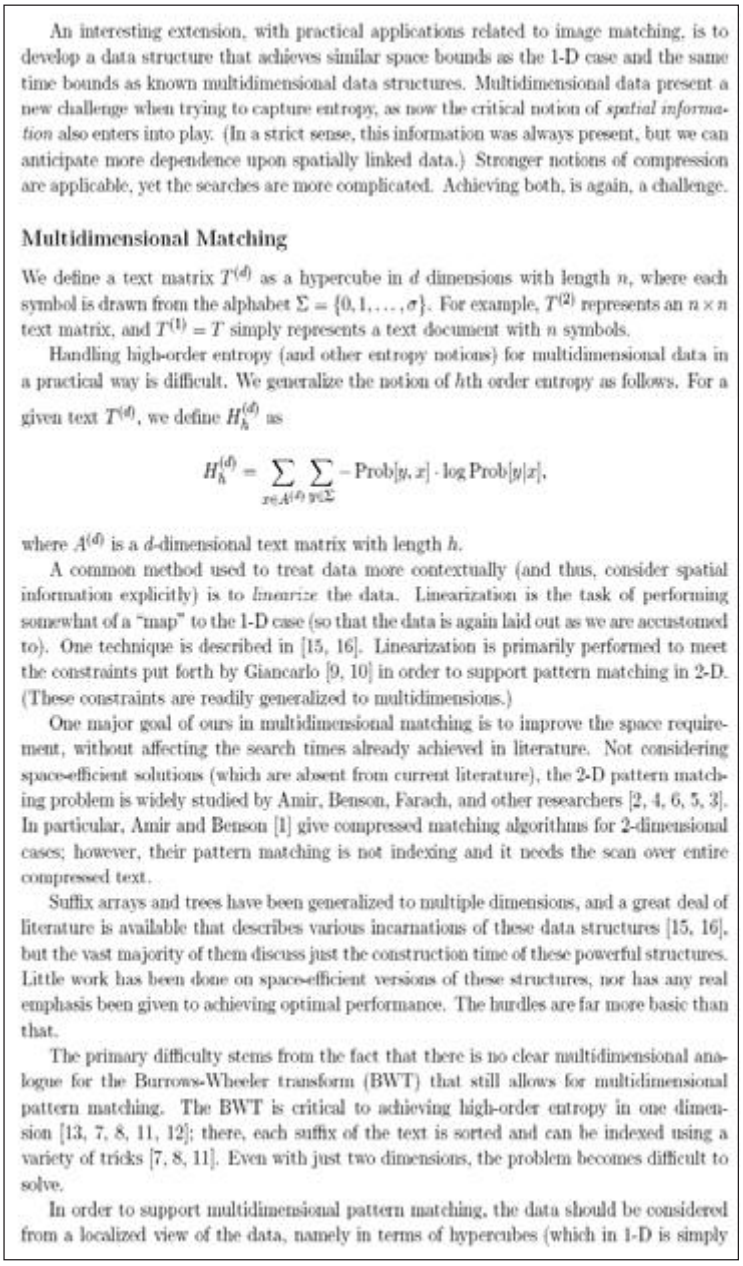
$$= \begin{bmatrix} 2^2 & 2^3 \\ 2^5 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 2^2 & 2^3 \\ 0 & 2^5 & 0 \end{bmatrix}$$

4. In the case of no match between the patterns that have been created in step 3 and the pattern, and when shifting the pattern to the left by one column is no longer available, a shift down by one row should be done in order to come up with new pattern if possible. As shown in figure 3.9.



The current pattern The new pattern

Figure 3.9. Create new pattern by shifting down by one row



An interesting extension, with practical applications related to image matching, is to develop a data structure that achieves similar space bounds as the 1-D case and the same time bounds as known multidimensional data structures. Multidimensional data present a new challenge when trying to capture entropy, as now the critical notion of *spatial information* also enters into play. (In a strict sense, this information was always present, but we can anticipate more dependence upon spatially linked data.) Stronger notions of compression are applicable, yet the searches are more complicated. Achieving both, is again, a challenge.

Multidimensional Matching

We define a text matrix $T^{(d)}$ as a hypercube in d dimensions with length n, where each symbol is drawn from the alphabet $\Sigma = \{0, 1, \ldots, \sigma\}$. For example, $T^{(2)}$ represents an $n \times n$ text matrix, and $T^{(1)} = T$ simply represents a text document with n symbols.

Handling high-order entropy (and other entropy notions) for multidimensional data in a practical way is difficult. We generalize the notion of hth order entropy as follows. For a given text $T^{(d)}$, we define $H_h^{(d)}$ as

$$H_h^{(d)} = \sum_{x \in A^{(d)}} \sum_{y \in \Sigma} -\text{Prob}[y, x] \cdot \log \text{Prob}[y|x],$$

where $A^{(d)}$ is a d-dimensional text matrix with length h.

A common method used to treat data more contextually (and thus, consider spatial information explicitly) is to *linearize* the data. Linearization is the task of performing somewhat of a "map" to the 1-D case (so that the data is again laid out as we are accustomed to). One technique is described in [15, 16]. Linearization is primarily performed to meet the constraints put forth by Giancarlo [9, 10] in order to support pattern matching in 2-D. (These constraints are readily generalized to multidimensions.)

One major goal of ours in multidimensional matching is to improve the space requirement, without affecting the search times already achieved in literature. Not considering space-efficient solutions (which are absent from current literature), the 2-D pattern matching problem is widely studied by Amir, Benson, Farach, and other researchers [2, 4, 6, 5, 3]. In particular, Amir and Benson [1] give compressed matching algorithms for 2-dimensional cases; however, their pattern matching is not indexing and it needs the scan over entire compressed text.

Suffix arrays and trees have been generalized to multiple dimensions, and a great deal of literature is available that describes various incarnations of these data structures [15, 16], but the vast majority of them discuss just the construction time of these powerful structures. Little work has been done on spaces-efficient versions of these structures, nor has any real emphasis been given to achieving optimal performance. The hurdles are far more basic than that.

The primary difficulty stems from the fact that there is no clear multidimensional analogue for the Burrows-Wheeler transform (BWT) that still allows for multidimensional pattern matching. The BWT is critical to achieving high-order entropy in one dimension [13, 7, 8, 11, 12]; there, each suffix of the text is sorted and can be indexed using a variety of tricks [7, 8, 11]. Even with just two dimensions, the problem becomes difficult to solve.

In order to support multidimensional pattern matching, the data should be considered from a localized view of the data, namely in terms of hypercubes (which in 1-D is simply

Figure 7. The main text image

In this case, formulas 1 and 4 will be applied.

Reaching this step and in the case of no matching between the patterns that have been created and the pattern, steps from 1 to 4 should be repeated if possible, until a match is found or no new patterns can be created.

By using these formulas we do not need to visit all elements in the Patterns, just we need to visit 2 columns or 2 rows, and then when we compare between these values and the value of the original Pattern by using one operation it will tell us if they are the same or not. So that number of comparisons will be reduced form Θ (NMnm) time to Θ ((N - n + 1)*(M - m + 1) * 2n) which is proportional to Θ (NMn).

This algorithm can be used for one dimensional Text. For example, if we have the following text with size 1*18: "101100010101000101" , and we are searching for a Pattern of length 1 * 8, we can compute the value of second Pattern which is

"01100010" from the value of the first Pattern which is "10110001" by subtracting the number added for the first '1' of "10110001", i.e. $1 * 2^1$(1 value of first element and 2 is the base we are using), dividing by the base and adding for the last '0' of " 01100010", i.e. $0 * 2^8$. If the substrings are long, this algorithm achieves great savings for number of comparisons. The worst case time for this application is Θ (M).

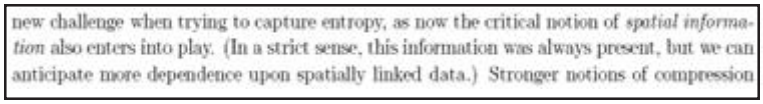


Figure 8. The pattern

5. Experimental Results

In this section, ….compare the results of our approach and the results of Brute force and RK-KMP algorithms which proposed by Rabin and Karp in [12]. These algorithms are tested on text images that have white and black colors with square patterns that have different sizes as shown in figures 7 and 8.

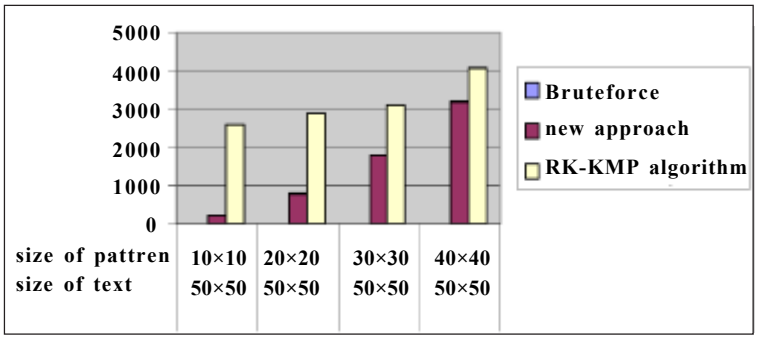


Figure 9. Comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in the preprocessing phase when text has size (50*50)

5.1 Experimental Methodology

In this section we present the testing methodology which is used in our experiments in order to compare our algorithm with the main algorithms in the tow dimensional pattern matching problem. There are set of parameters which describe performance of these algorithms which they are: Size of text, Size of pattern and Size of alphabet .

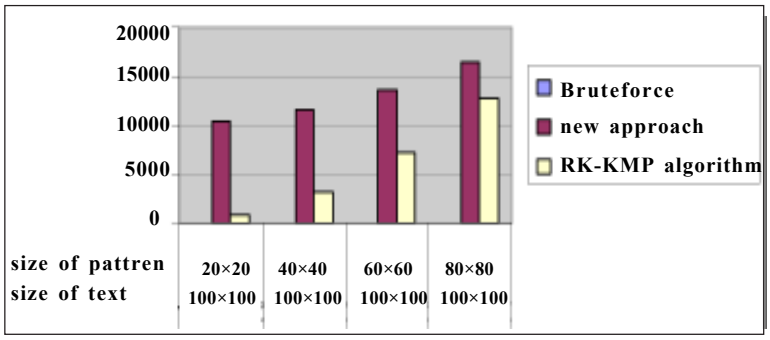


Figure 10. comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in the preprocessing phase when text has size (100*100)

There are different types of data can be used in the tow dimensional pattern matching problem, in our research we will focus on binary alphabet where $\Sigma = \{0, 1\}$. The main motivation of using this alphabet is the vast increase of stored white and black images and the problem of finding occurrence of patterns in these images. For the comparison of the two dimensional pattern

matching algorithms we used the number of character comparisons as measures. This way in [Smith, 1991] is used in string matching problem to create comparison between some of algorithms

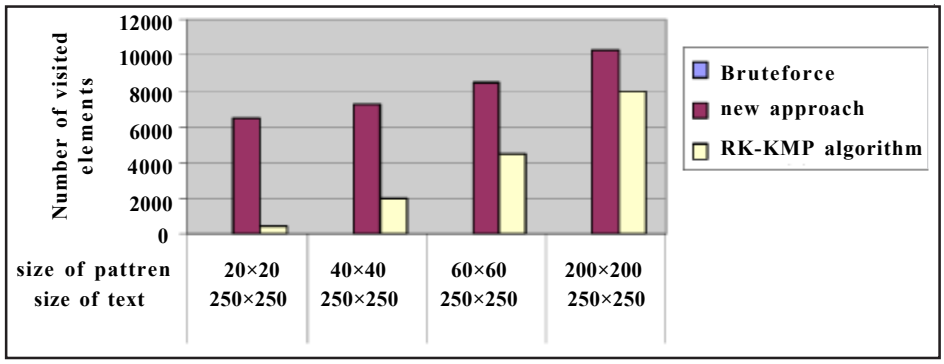


Figure 11. Comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in the preprocessing phase when text has size (250*250)

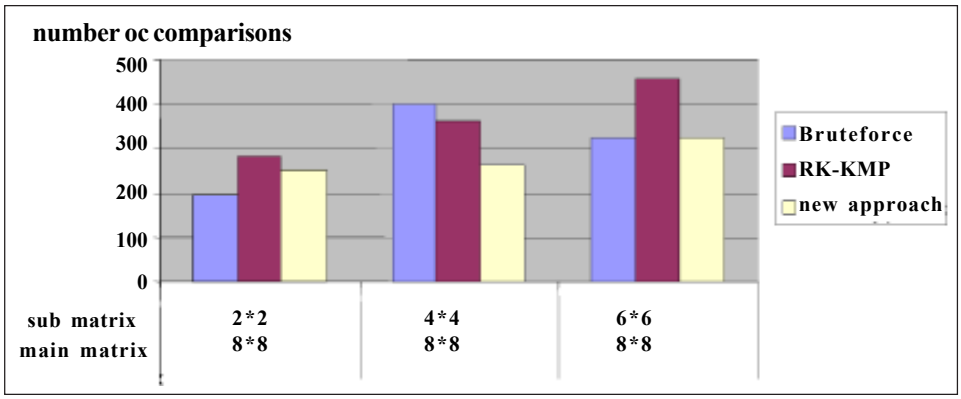


Figure 12. Comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (8*8)

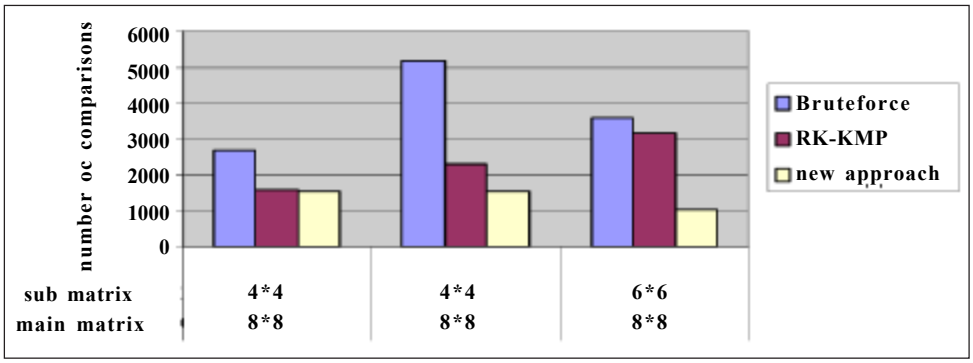


Figure 13. Comparison between Brute Force, RK-KMP, and New Approach matrix containment algorithms in worst case when text has size (16*16)

5.2 Results of preprocessing phase

In this section, the new approach will be compared to the other algorithms in the preprocessing phase.

In figure 9, 10, and 11, as shown in all experiments, the Brute Force algorithm no need preprocessing phase. It just looks for the pattern at all possible positions in the text. The new approach is better than the *RK-KMP* algorithm in all experiments because the preprocessing time of the new approach is proportional to $O(2 * m1 * m2)$, where the preprocessing time of the RK-KMP

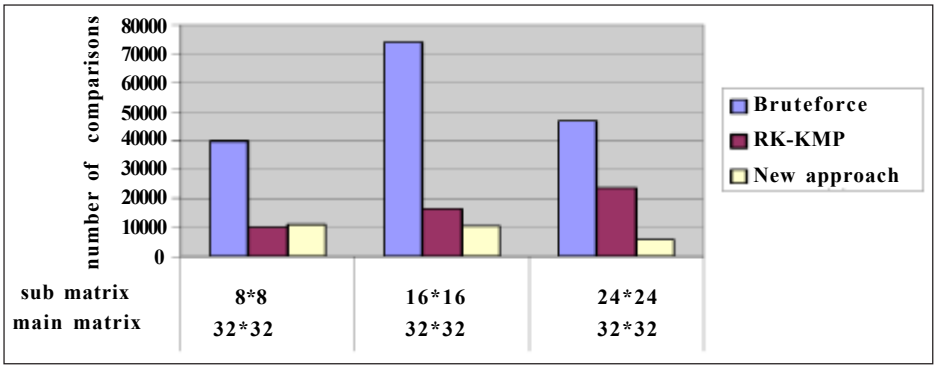


Figure 14. Comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (32*32)

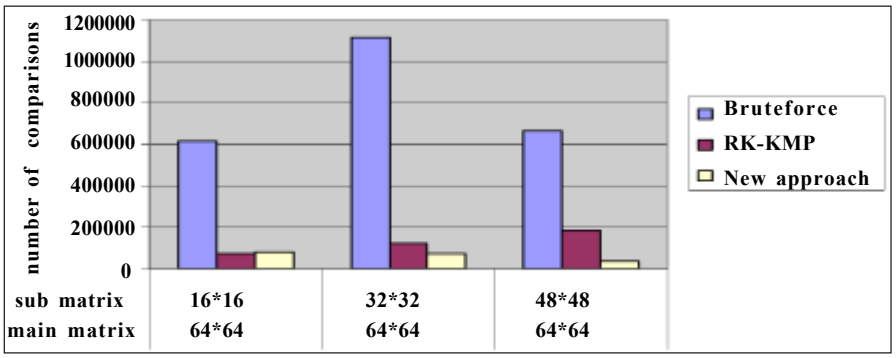


Figure 15. Comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (64*64)

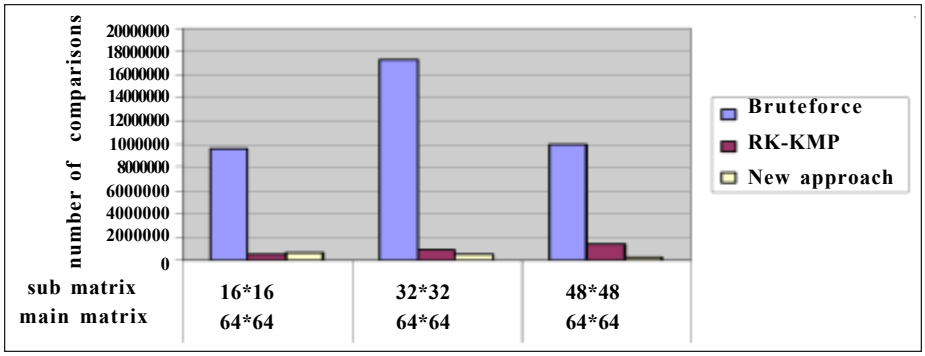


Figure 16. Ccomparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (128*128)

algorithm is proportional to $O(m1 * m2 + n1 * n2)$.

5.3 Results of exact matching with square size of pattern

In this section, the new approach will be compared to the other algorithms when the pattern is square ($m = n$).

In figure 13, as shown in the first experiment where size of the pattern (2 * 2) and the size of the text (8 * 8), the BF algorithm is better than the RK-KMP algorithm and the new approach, and it is because that the BF algorithm does not need the preprocessing phase while the others need it in different ways, also the pattern contains only two columns that prevent the other two algorithms to work properly, as mentioned before.

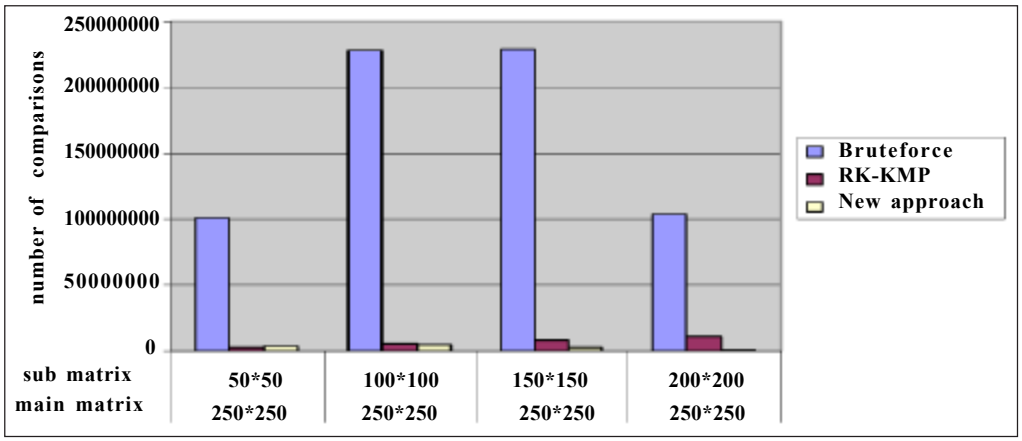


Figure 17. Ccomparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (250*250)

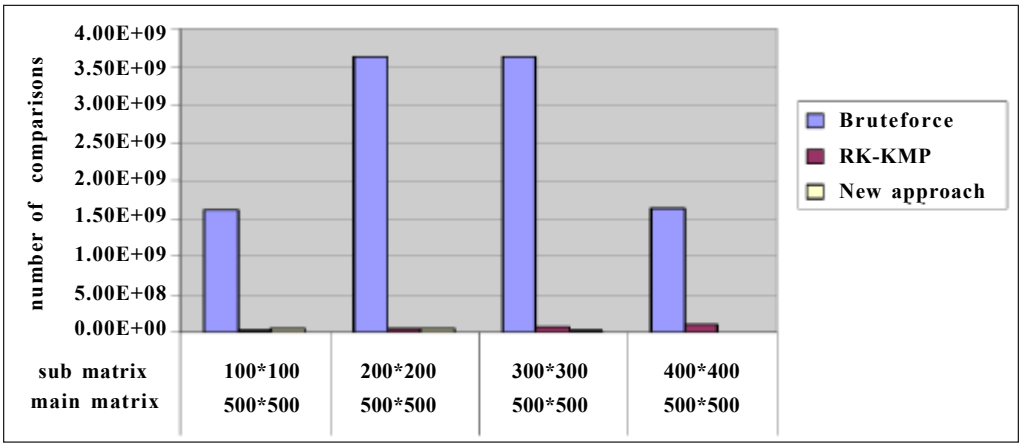


Figure18. Ccomparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (500*500)

In figure 12 and 13, the new approach algorithm is the best algorithms among the others, except the first experiment that is shown in figure 1, where size of the pattern (2*2) and size of the text (8*8).

In figures 14,15,16,17, and 18, the RK-KMP algorithm shows better performance than the other algorithms. the proposed algorithm scored low in the first attempt due to the difference between the size of the pattern and the text, in the first attempt the size of pattern is equal to the quarter size of the text. The proposed scored high with comparing with other algorithms in the rest of the attempts since the size of the pattern is bigger than the quarter size of the text.

As a result, when the difference in the size, between the pattern and the text is little, the number of comparisons will decrease in the new approach, and will increase in the other algorithms. The process of shift to right by one column will decreases visiting large number of elements that represent most of the text which leads to increase the matching speed.

Further, when the difference in the size between the pattern and the text is little, the number of patterns that can be created from the text will be little as well, and because the main operation in the new approach is deleting a column and adding another for each pattern. That will decrease the number of visited columns and then the number of visited elements will be decreased too. While in the other algorithms, there is no relation between the number of patterns that can be created from the text and the number of visited elements.

6. Conclusion and Future Works

6.1 Conclusion

In this paper, we present a fast exact two dimensional pattern matching algorithm. Experiments results shows that this algorithm has better performance in most experiments, faster than the other algorithms in two cases; (1)when the pattern has size greater than quart of the Text and (2) when the Pattern and the Text have very small size, So that, our algorithm is effected with size and form of patterns. The proposed algorithm has the least number of character comparisons in most cases; therefore it is feasible that this method can be used in applications related to exact two dimensional patterns matching in white and black images databases.

6.2 Future work directions

The works that can be done in the field of our research is open and many improvements can be done on the proposed algorithm. Recommended future works can be summarized as following:

- Enhancing the proposed algorithm to find not only the first occurrence of the pattern with low number of comparisons but also find all occurrences of the pattern in the text with low number of comparisons.

- Devising new equation that can convert the pattern into unique value when we treat with colored images or gray scale images.

- Build an efficient algorithm to find all rotated occurrences of pattern in two dimension array when we restricted our alphabet to {0, 1}.

- Build an efficient algorithm to reduce number of computations which needed to find the correlation between image (binary image) and object template.

References

[1] Amir, A. (1992). *Multidimensional pattern matching: A survey*, Technical Report GIT-CC-92/29, Georgia Institute of Technology, College of Computing.

[2] Amir, A. and Benson, G. (1992), Efficient two dimensional compressed matching, *In*: *Proc. of Data Compression Conference,* p. 279-288.

[3] Baker, T. (1978). A technique for extending rapid exact string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7 (3) 533–541.

[4] Bayer, R., Moore, J. (1977), A fast matching algorithm. ACM, 20 (10) 762-772.

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

[6] Brown, L. (1992). A survey of image registration techniques, ACM Journal, *Computing Surveys*, 24 (4) 325-376.

[7] Charalampos S., Kouzinopoulos, Konstantinos G. (2008), Improving the Efficiency of Exact Two Dimensional On-line Pattern Matching Algorithms, *In*: *Proc. of* IEEE Panhellenic Conference on Informatics, Samos, p. 232-236.

[8] Davies, G., and Bowsher, S. (1986). Algorithms for pattern matching. Software Predicates and Experience, 16 (6) (16, 575-601).

[9] Fan, J., Su, K. (1995). The design of efficient algorithms for two dimensional pattern matching. IEEE, Transactions on knowledge and data engineering. 7 (2) 318-327.

[10] Hudaib, A., Al-khalid, R., Suleiman, D., Itriq, M., Al-anani, A. (2008), A fast pattern matching algorithm with tow sliding windows (TSW). *Journal of computer science*, 4 (5) 393-401.

[11] Jain, A. (2003), Fundamental of digital image processing, the tenth Indian reprint, New Delhi-India: Prentice Hall of India private limited..

[12] Karp, R., Rabin, M. (1987). Efficient randomized pattern-matching algorithms. IBM, 31 (2) 249-260.

[13] Knuth, D., Morris, J., Pratt. V. (1977). Fast pattern matching in strings. *SIAM Journal of Computing*, 6 (2) 323-350.

[14] Kouzinopoulos, C., Margaritis, K. (2008). *Exact Two Dimensional On-line Pattern Matching Algorithms: Survey and Experimental Results*. Technical Report, University of Macedonia, Department of Applied Informatics.

[15] Li, T. (2005). A general model for clustering binary data. *In*: Proc. of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, Chicago, Illinois, USA, p. 188–197.

[16] Michailidis, P., Margaritis, K. (2001) On-line String Matching Algorithms: Survey and Experimental Results. *International Journal of Computer Mathematics*, 76 (4) 411-434.

[17] Zdarek, J., Melichar, B. (2006). On Two-Dimensional PatternMatching by Finite Automata. *In: Proc. Of Implementation and Application of Automata* Conference, Springer /Heidelberg. 3845, 329-340.