Efficient algorithm for two dimensional pattern matching problem (non-square pattern)

Article · March 2012

DOI: 10.1109/ICITeS.2012.6216622

3 authors, including:

Jaber Alwidian
PSUT, Jordan
19 PUBLICATIONS **108** CITATIONS

Efficient Algorithm for Two Dimensional Pattern Matching Problem (non-square pattern)

Jaber Alwidian
ITC department
Arab Open University
Riyadh- Saudi Arabia
j.alwidyan@arabou.edu.sa

Hussein Abu-Mansour
ITC department
Arab Open University
Riyadh- Saudi Arabia
hmansour@arabou.edu.sa

Moshira Ali
ITC department
Arab Open University
Riyadh- Saudi Arabia
moshira@arabou.edu.sa

Since the recent advancements in computer storage capacities and processing speed have reached enhanced levels, researches are exerting more demand on higher storage capacity of multidimensional data and elevated computational strengths for research areas. Consequently, two-dimensional pattern matching is a primary topic of current research making good use of current technologies. In this paper, we will examine two-dimensional pattern solution. We are proposing an algorithm to convert the two-dimensional problem into a one-dimensional problem prior to the searching phase. Our proposed algorithm handles black and white images in general and text images in particular. Work in this phase will eliminate the need for soaring computational expense observed in preprocessing phase. Our experimental findings display results superior to both Brute-force algorithm, and RK-KMP algorithm. Suggested algorithm results show a clear reduction in comparisons' numbers in multiple cases. Moreover, applied on one-dimensional text, our proposed algorithm is proportional to Θ (n) in time suggesting linear time.

Keywords: pattern matching, two-dimensional pattern matching, RK-KMP algorithm, Brute-force algorithm

I. INTRODUCTION

The mid-seventies and mid-eighties have been active years for different proposal of equally efficient pattern matching algorithms. Notably, KMP algorithm (Knuth et al., 1977), BM algorithm (Bayer and Moore, 1977), and RK algorithm (Davies and Bowsher, 1986) running single dimensional pattern matching algorithms. Baker (Baker, 1978) and in parallel Bird (Bird, 1977) were pioneers in introducing worst case linear time algorithm for two dimensional pattern matching. Their suggested algorithm then started with a preliminary step running finite automation aiming at a linear text scan. A move on to two or even more dimensional pattern matching problems became a natural step forward, especially with the great advancements in computational technology of computers and their tremendous capacity for multi-dimensional data as would be in photo scans processing (Zdarek and Melichar, 2006). Multiple researchers worked on several algorithms for the exact two dimensional pattern matching problems as reported in (Kouzinopoulos and Margaritis, 2008).

A general classification of two dimensional pattern matching problem would include the following: (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.

II. PROBLEM STATEMENT

When considering string matching in text processing, we could specify its main usage as allocating one-dimensional pattern (string) in the specific text. With the breakthrough in image processing, graphical imaging and multimedia, a need to work in higher levels of pattern matching dimensions was noticed. Two and multi-dimensional pattern matching was to be introduced and implemented (Charalampos et al., 2008). However, this unique approach faced multiple difficulties in implementation since it required handling extremely large numbers of comparisons to locate similarities and specified occurrences in both two-dimensional patterns and text. In most suggested solutions, a relapse to the one-dimensional pattern down from the two-dimensional problem is carried out. These solutions, though, require preprocessing phase for structure re-constructing before text searching could be applied. (Charalampos et al., 2008).

Different text types can be utilized in two-dimensional patterns matching problem as: binary alphabets, alphabets of size 8, English alphabet, and DNA alphabet (Michailidis and Margaritis, 2000) and (Charalampos et al., 2008). This research focuses on the use of binary text. For example, consider the following two matrices *X* and *Y* represented in Figure 1, *X* as a Text and *Y* as a Pattern. 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

Figure1: Two binary matrices X and Y.

Foremost proposal in this research is to achieve an efficient algorithm in exact 2D pattern matching problem when alphabet is binary {0, 1} while keeping arrays in two-dimensional formats, thus eliminating the need to pre-process arrays to reduce them to one-dimensional format. Over-riding this phase will increase efficiency and obliterate originally-needed pre-processing phase of structure constructing before the onset of text search can commence (Charalampos et al., 2008). This algorithm's significance is within the ability to detect an object in black and white images. Such detection is considered as a major obstacle in both pattern recognition and image processing (Brown, 1992). Moreover, the importance of binary data and its newly acquired place in the domain of data analysis should be not be overlooked (Li, 2005).

The structure of this paper is as follows: Section II covers the problem statement. Section III, on the other hand, presents the related work. The proposed algorithm is discussed in Section IV, whereas Section V presents and discusses experimental results. The conclusion is given in Section VI.

III. Related Work

The two dimensional pattern matching problem is considered the first generalization of the one dimensional pattern matching (Charalampos et al., 2008). One dimensional pattern matching problem begins text searching using a single-direction focus – mainly from left to right – thus locating all pattern occurrences in the named text string. Most known algorithms in this area adopt roughly the same procedure (Hudaib et al., 2008) . The same cannot be applied to two-dimensional pattern matching since there is the necessity of locating all two-dimensional pattern occurrences in a two-dimensional text matrix where pattern occurrences might be in the left, right, top and bottom. Such procedure is presented in Figure 2.

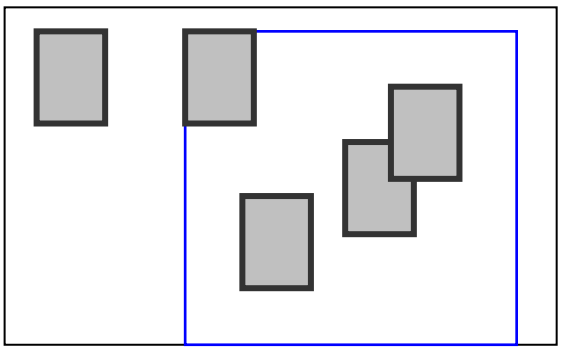


Figure 2: Two dimensional pattern matching

A classification of two-dimensional pattern matching problem according to the way of matching results in 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 (Amir, 1992). However in practice, some types are more prominent than others, the most important ones being exact two dimensional pattern matching problem and two dimensional approximate pattern matching (Zdarek and Melichar, 2006). This section focuses on types one, two and three. A definition of the exact two dimensional pattern matching problem (Amir, 1992) assumes the following: 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.

Several applications adopt the exact 2D 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 (Fan and Su, 1995). One such application that clearly displays the significance of pattern matching problem is compressed matching problem, defined in (Amir and Benson, 1992). The named definition presents the problem of finding all occurrences of a pattern in compressed images. Motivated by the vast increase of stored compressed data, compressed matching problems concentrated on the possibility of finding all pattern occurrences in a compressed text without the need for image decompression (Amir and Benson, 1992). The problem was further significantly shown in another application that searches aerial photographs (Amir, 1992). In such application, the main objective is to read an aerial photograph along with a template of some patterns. Expected output being the spotting of all locations on the aerial photograph where the template object appears.

Several studies and research tracks concentrated on exact 2D pattern matching problem with binary alphabets. The problem was mainly observed in pattern recognition and image processing where the goal was detection of an object in a black and white image (Brown, 1992). As binary data assumes a prominent position in the data analysis domain, the same observed problem surfaced (Li, 2005). Exact 2D matching algorithms were classified in (Amir, 1992) according to two criteria (1) Reduction of the 2D pattern matching problem into one dimension such as (Bird and Baker, Weiner's or Creight's algorithm) and (2) The analysis of the 2D structure of the pattern and its usage in text scanning step such as (2D periodicity idea).

IV. The Proposed Algorithm

This section comprehensively explains our proposed algorithm with details regarding its two phases and proof of work. The proposed algorithm spans two main phases: the preprocessing

phase and the matching phase. Figure 3 below illustrates the block diagram of these two main steps of the proposed algorithm.

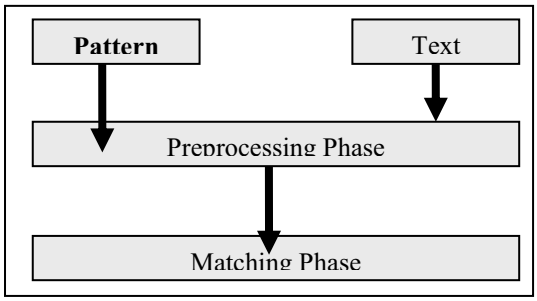


Figure 3: The main steps of the proposed algorithm

A. Preprocessing Phase

In lieu of what have been declared in the research objectives earlier, the proposed algorithm works with '0's and '1's matrices (white and black images). The preprocessing step constructs the structure essential for the following step of text search.

Let Pattern size be *n*m*, Time is *O (3*n*m)*. This divides into three parts as follows:

1. let

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

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

Principally pattern must be converted into unique values using the following formula:

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

Using this formula, we will have a fragment from the geometric form or full geometric form as shown in following formula:

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

To demonstrate, let's assume have the following pattern:

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

By applying formula 1, unique value of converted pattern is calculated as follows:

$= 1*(2^1) + 0*(2^2) + 1*(2^3) + 1*(2^4) + 0*(2^5) + 0*(2^6)$

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

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

Consequently the obtained value is considered as geometric form fragment as was shown in formula 2, which is unique value. The above concludes that there should not be any other pattern having the same above value but 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 be 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 changing any element will give unique value.

If the size of the Pattern is equal n*m, the time of this part is proportional to O (n*m). The Pseudo Code is presented 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 with 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 ٤: 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 used in the previous part where its time is proportional of *O (n*m)*. For example, let

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

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

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

Get unique 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

```
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 with 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: Pseudo code of part 2

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 2^1 to 2^{n*m}. To demonstrate we say,

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).

```
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

B. *Matching Phase*

The proposed algorithm is built basically on and derived from the Karp and Rabin algorithm. Comparisons are sequentially performed in order from left to right until a complete match occurs.

This is further explained in an example as follows:
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 unique value of the pattern and the first pattern that is derived from the text should be done for matching purposes. If no matching is resolved, further steps should be applied to obtain a match:

1. Firstly a new pattern must be derived from the text. A shift to the right by one column should occur to the whole pattern, if possible. Figure 3.5 demonstrates:

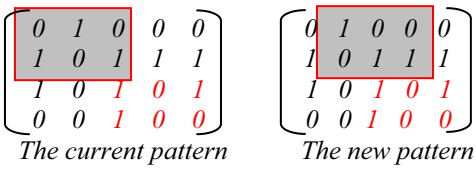


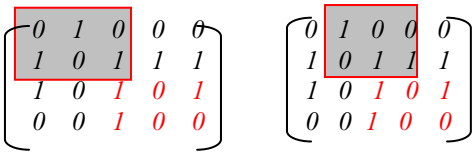
Figure 3.5: create new pattern by shifting to the right one column.

Presently, the following formula is used to calculate new pattern's value:

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



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 the right by one column is no longer available, a down shift by one row from the current pattern should be done if possible. The shift will come up with a new pattern from a certain text as shown in figure 3.7.

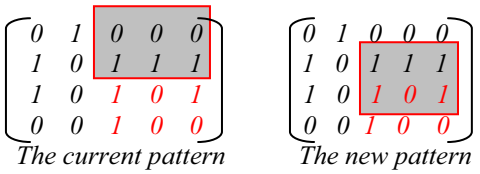


Figure 3.7: create new pattern by shifting down one row.

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

Value of the Pattern= (PV - FRPP)/2^m + LRNP

Where

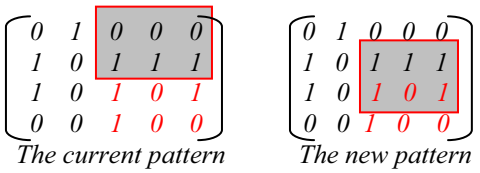
PV is Previous Value

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



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

$$\begin{bmatrix} 0 & 0 & 0 \\ 2^4 & 2^5 & 2^6 \end{bmatrix} - \boxed{0\ 0\ 0} = \begin{bmatrix} 2^4 & 2^5 & 2^6 \end{bmatrix} /2^3$$

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

3. In the case of finding no match between the patterns that has been created from step2 and the pattern, a shift to the left by one column should be done. Further left shifts should be repeated until either a match with the pattern is obtained, or no more left shifting is possible as end of columns was reached. This is demonstrated in figure 3.8.

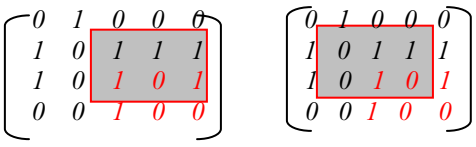


Figure 3.8: create new pattern by shifting to the left by one column.

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

Value of the next Pattern= (PV – LCPP)*2 + FCNP

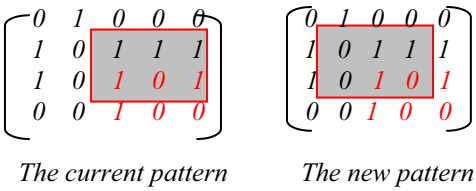
Where

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



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.

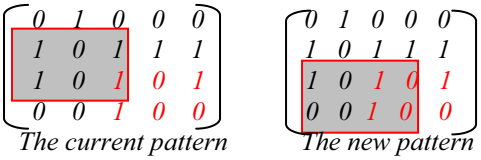


Figure 3.9: create new pattern by shifting down by one row.

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

Upon reaching this stage, steps 1 to 4 will keep repeating until either a match is found or no new patterns can be created.

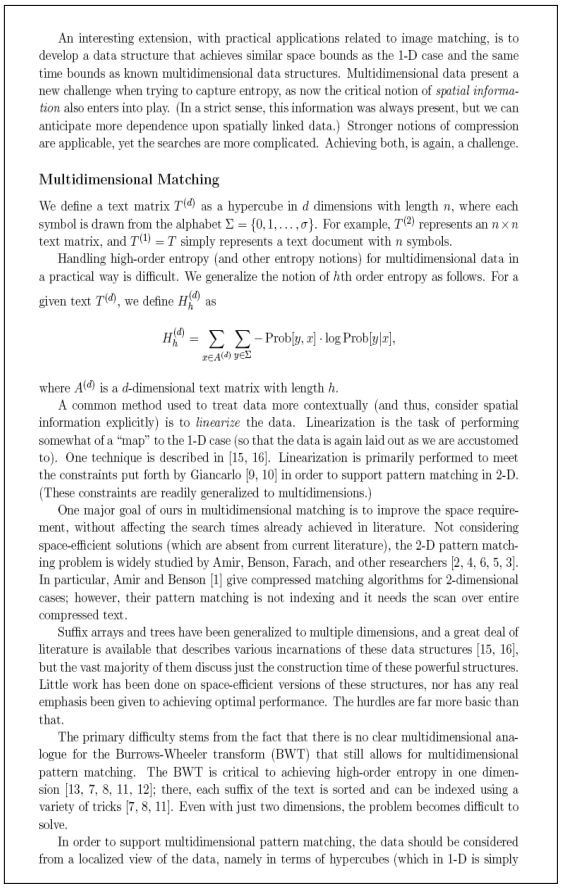
The achievements behind using these formulas center on the unnecessary call for all array elements visitations. Match inspection is not required on all Patterns elements. We minimally need to visit 2 columns or 2 rows. The comparison that proceeds between these values and the value of the original Pattern using one operation will identify the existence or non-existence of a match. In this manner, the number of needed 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. To explain we will have the following text with size 1*18: "1011000101010000101". We are searching for a Pattern of length 1*8. We will compute the value of the 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 required comparisons. The worst case time for this application is Θ (M).

V. EXPERIMENTAL RESULTS

In this section, we evaluate the results of our approach in comparison with the results of both Brute force and RK-KMP algorithms proposed by Rabin and Karp in [Karp and Rabin, 1987]. These algorithms are tested on text images that have white and black colors with non square patterns that have different sizes as shown in figures 7 and 8.



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.)

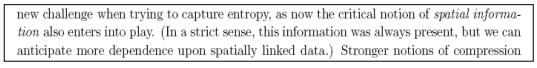
One major goal of ours in multidimensional matching is to improve the space requirement, without affecting the search times already achieved in literature. Not considering space-efficient solutions (which are absent from current literature), the 2-D pattern matching problem is widely studied by Amir, 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 space-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



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

Figure 8: The pattern

A. Experimental Methodology

We present, in this section, the testing methodology which is used in our experiments for the comparative purposes between our algorithm and the main algorithms in the two dimensional pattern matching problem. There are a set of parameters which describe the performance of these algorithms such as: Size of text, Size of pattern and Size of alphabet.

Various data types can be implemented in two-dimensional pattern matching problems. Our research focuses on binary alphabet where $\sum = \{0, 1\}$. The motivation behind this focus is the vast increase of stored white and black images available and the difficulties in finding patterns occurrences in these images. The number of character comparisons is applied as measures throughout our comparative experiment of the two-dimensional pattern matching algorithm. Such measures as in [Smith, 1991] are used in string matching problem to create comparison between some of algorithms.

B. Results of preprocessing phase

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

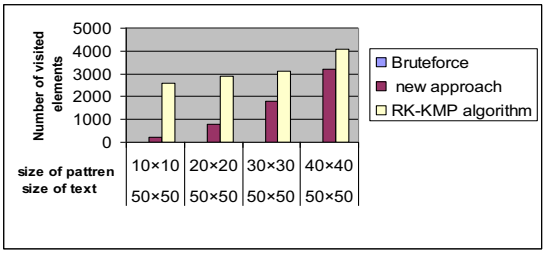


Figure 9: comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in the preprocessing phase when text has size (50*50).

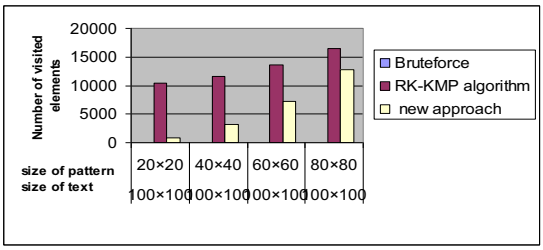


Figure 10: comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in the preprocessing phase when text has size (100*100).

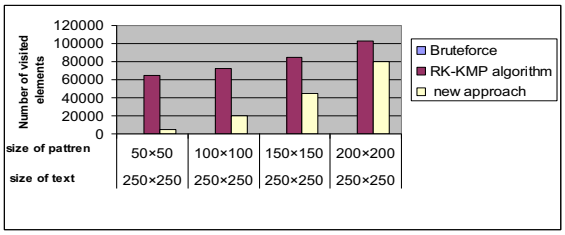


Figure 11: comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in the preprocessing phase when text has size (250*250).

Brute Force algorithm, in figures 9, 10, and 11, shows no need for a preprocessing phase. It only searches for the pattern at all possible positions in the text. The new approach shows better results than the *RK-KMP* algorithm in all experiments for the reason that the preprocessing time of the new approach is proportional to *O (2*m1*m2),* whereas the preprocessing time of the RK-KMP algorithm is proportional to *O (m1*m2 + n1*n2).*

C. Results of exact matching with non square size of pattern

In this section, the new approach will be compared to the other algorithms when the pattern is non-square. Non-square pattern has two different forms. First one when columns are more than rows, second one when rows are more than columns. In this part the first form will be applied.

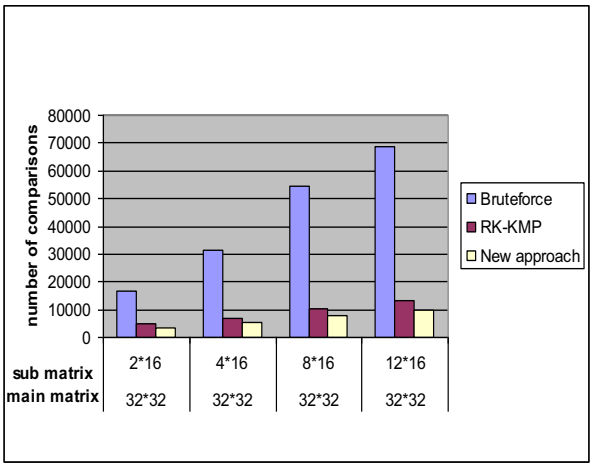


Figure 12: comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (32*32).

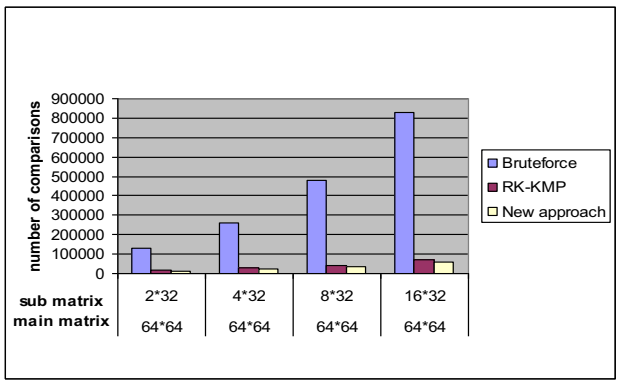


Figure 13: comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (٦٤*٦٤).

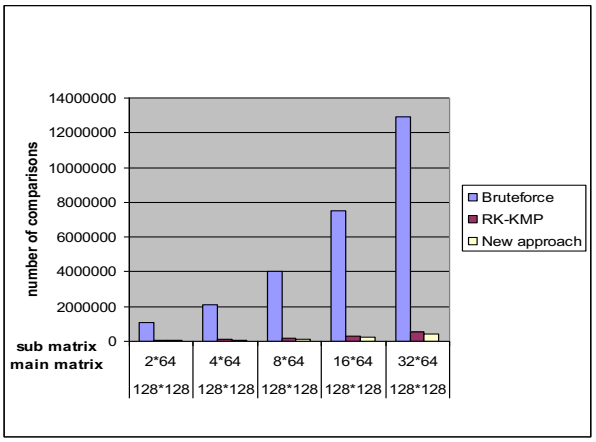


Figure 14: comparison between Brute Force, RK-KMP, and New approach matrix containment algorithms in worst case when text has size (١٢٨*١٢٨).

In figures 12, 13, and 14, the new approach is the best among all of the algorithms that have been tested. The basic operation in the new approach is deleting one column or one row and adding another. And so the number of elements that exist in the columns when the pattern is non-square will be little, and that will give a result that number of visited elements in the matching phase is little as will.

To conclude, we could explain that differences in size between the pattern and the text affect the number of comparisons. The proposed algorithm will not score high as long as the differences in the size between the pattern and the text are slight. The other algorithms work well under these conditions, though. The proposed shifting methodology decreases the need for large numbers of elements visitations, thus resulting in an increase in matching speed.

Further, the other studied algorithms do not show a relation between the number of patterns that can be created from the text and the number of visited elements, whereas the suggested algorithm displays a clear relation. When the difference in the size between the pattern and the text is slim, the number of patterns that can be created from the text will be slim as well as the main operation in the new approach is deleting a column and adding another for each pattern. Usage of that will decrease the number of visited columns and hence the number of visited elements will decrease accordingly.

VI. Conclusion and Future Works

A. Conclusion

In this paper, we demonstrated a speedy exact two dimensional pattern matching algorithm. Experiments results show that this algorithm has better performance in most experiments, speedier than the other algorithms in two cases; (1) when the pattern size is greater than quarter of the Text and (2) when the Pattern and the Text have very small sizes. This shows that our algorithm is affected by 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.

B. Future work directions

Extensive work can be further carried on in the field of our research to add several improvements on the proposed algorithm. Recommended future work can be summarized as follows:

— Enhancing the proposed algorithm to work out methodology of getting not only first pattern occurrence with low number of comparisons, but also getting all pattern occurrences in the text with low number of comparisons.

— Devising new equations to convert the pattern into unique value in the treatment of both colored images and gray scale images.

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

— Build an efficient algorithm to reduce the number of computations 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, *Proc. of Data Compression Conference,* pp. 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*, vol. 7, no. 3, pp. 533–541.

[4] Bayer, R. and Moore, J. (1977), A fast matching algorithm. ACM, vol. 20, no. 10, pp. 762-772.

[5] Bird, R. (1977), Two dimensional pattern matching, *Information Processing Letters*, vol. 6, no.5, pp. 168–170.

[6] Brown, L. (1992), A survey of image registration techniques, ACM Journal, Computing Surveys, vol.24, no.4, pp. 325-376.

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

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

[9] Fan, J. and Su, K. (1995), The design of efficient algorithms for two dimensional pattern matching. IEEE, Transactions on knowledge and data engineering. Vol. 7, no. 2, pp. 318-327.

[10] Hudaib, A., Al-khalid, R., Suleiman, D., Itriq, M., and Al-anani, A.(2008), A fast pattern matching algorithm with tow sliding windows (TSW). Journal of computer science, vol. 4, no. 5, pp. 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. and Rabin, M. (1987), Efficient randomized pattern-matching algorithms. IBM, vol. 31, no. 2, pp. 249-260.

[13] Knuth, D., Morris, J., and Pratt. V. (1977), Fast pattern matching in strings. SIAM Journal of Computing, vol. 6, no.2, pp. 323-350.

[14] Kouzinopoulos, C. and 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. Proc. of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, Chicago, Illinois, USA, pp. 188–197.

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

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