Syracuse University
SUrface

Electrical Engineering and Computer Science Technical Reports

L.C. Smith College of Engineering and Computer Science

8-23-1989

Two-Dimensional Pattern Matching with k Mismatches

Sanjay Ranka
Syracuse University

Todd Heywood

Follow this and additional works at: http://surface.syr.edu/eecs_techreports

Part of the Computer Sciences Commons

Recommended Citation
Ranka, Sanjay and Heywood, Todd, "Two-Dimensional Pattern Matching with k Mismatches" (1989). *Electrical Engineering and Computer Science Technical Reports.* Paper 62.
http://surface.syr.edu/eecs_techreports/62

This Report is brought to you for free and open access by the L.C. Smith College of Engineering and Computer Science at SUrface. It has been accepted for inclusion in Electrical Engineering and Computer Science Technical Reports by an authorized administrator of SUrface. For more information, please contact surface@syr.edu.

TWO-DIMENSIONAL PATTERN MATCHING WITH k MISMATCHES

Sanjay Ranka
Todd Heywood

School Of Computer and Information Science
Syracuse University
Syracuse, NY 13244-4100

(315)-443-4457

ranka@top.cis.syr.edu
heywood@top.cis.syr.edu

August 23, 1989

SU-CIS-89-12

Two-Dimensional Pattern Matching with k Mismatches

Sanjay Ranka Todd Heywood

August 1989

School of Computer and Information Science
Syracuse University
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

Abstract

We give an algorithm which finds all occurrences of an $m_1 \times m_2$ pattern array embedded as subarrays in an $n_1 \times n_2$ array of text, where at most k mismatches are allowed per occurrence. The algorithm runs in time $O((k+a)(b \log b + n_1 n_2))$, where $a = \min(m_1, m_2)$ and $b = \max(m_1, m_2)$. This improves upon the previously best known algorithm, and is asymptotically optimal for $k \approx a$.

1 Introduction

Two-dimensional pattern matching with k mismatches consists of taking two rectangular arrays, an $m_1 \times m_2$ *pattern* array and an $n_1 \times n_2$ *text* array, and finding all occurrences of the pattern embedded as subarrays in the text, where k mismatches of symbols are allowed per occurrence. The problem is important in the object recognition domain of computer vision, where the text is equivalent to a pixel representation of a scene, and the text alphabet is the range of values a pixel can have. Similarly, the pattern is a pixel representation (using the same alphabet) of an object which we want to recognize in the scene. Since errors ("noise") are likely to be present in the scene, or slight differences in perspective between a stored pattern and a scene object to exist, it is important to allow a measure of error to occur in pattern matching, thus the k allowed mismatches between symbols (pixels) of the pattern and the text [1,5].

Krithivasan and Sitalakshmi [2] have given the only previous algorithm for two-dimensional pattern matching with k mismatches. Their algorithm has complexity $O(km_1m_2 \log m_2 + m_1^2m_2 + km_1n_1n_2)$. The $m_1^2m_2$ factor stems from taking advantage of the fact that two rows of the pattern may be identical or almost identical; Krithivasan and Sitalakshmi state that, in practice, this reduces the total time. Since we do not analyze the pattern for this information in our algorithm, and in order to give an accurate comparison of the running time of our algorithm with theirs, we remove this factor. In this case, the complexity of their method will be $O(km_1(m_2 \log m_2 + n_1n_2))$. The complexity of our algorithm is $O((k + m_1)(m_2 \log m_2 + n_1n_2))$, an improvement of $O(km_1/(k + m_1))$ over that of [2]. Our algorithm requires $O(k(m_2 + n_2))$ space, also an improvement over the $O(m_1(km_2 + kn_2 + m_1 \max(m_2, k)))$ space requirement of [2] (ignoring the $O(m_1m_2)$ and $O(n_1n_2)$ space required by both algorithms to store the pattern and text arrays).

Between m_1 and m_2, m_1 has the greater affect on the time complexity of our algorithm, and that of [2], because of its n_1n_2 factor. We could compensate for this by transposing the pattern and text arrays prior to pattern matching whenever $m_1 > m_2$ (guaranteeing that m_1 always corresponds to the smaller pattern dimension), resulting in a change to the above complexities by substituting $\min(m_1, m_2)$ for m_1 and $\max(m_1, m_2)$ for m_2. We return to this in the concluding section of the paper.

Landau and Vishkin [3,4] have considered the one-dimensional case of string matching in the presence of errors. In [3], they give algorithms for two problems: string matching

with k mismatches, and string matching with k differences. The k-differences problem allows three types of errors to occur: a mismatch between a pattern symbol and a text symbol, a pattern symbol pattern corresponding to *no* text symbol, and a text symbol corresponding to *no* pattern symbol. In [4], an efficient and elegant algorithm is presented for finding all occurrences of an m-length pattern in an n-length text string, with k mismatches allowed per occurrence. The algorithm runs in $O(k(m \log m + n))$ time.

In this paper, we restrict ourselves to the k-mismatches problem, since the pattern is a strictly rectangular array occurring as an $m_1 \times m_2$ subarray of the text. We extend the techniques of [4] to two dimensions.

Our algorithm consists of a pattern analysis stage and a text analysis stage. The output of the pattern analysis is used in the text analysis. Since the pattern analysis is a variation of the text analysis, we present the text analysis first, in Section 2. Section 3 contains the pattern analysis, and Section 4 concludes the paper. For clarity, we have organized the presentation similarly to that of [4].

2 Text Analysis

The input to the text analysis stage consists of:

1. The pattern array $p_{i,j}$, $1 \leq i \leq m_1$, $1 \leq j \leq m_2$
2. The text array $t_{i,j}$, $1 \leq i \leq n_1$, $1 \leq j \leq n_2$
3. The output of the pattern analysis, which consists of a pair of two-dimensional PATTERN-MISMATCH (PM) arrays
   (a) PM-I$[1, \ldots, m_2 - 1; 1, \ldots, 2k + 1]$
   (b) PM-J$[1, \ldots, m_2 - 1; 1, \ldots, 2k + 1]$

Output from the text analysis consists of a pair of two-dimensional TEXT-MISMATCH (TM) arrays

1. TM-I$[0, \ldots, n_2 - m_2; 1, \ldots, k + 1]$
2. TM-J$[0, \ldots, n_2 - m_2; 1, \ldots, k + 1]$

In the following, we use the notation PM-(IJ) and TM-(IJ) to refer to the pairs of PATTERN-MISMATCH and TEXT-MISMATCH arrays. These arrays store i and j indices of positions in the two-dimensional pattern and text arrays. PM-I$[x,y]$ and PM-J$[x,y]$ (TM-I$[x,y]$ and TM-J$[x,y]$) together record an i,j position in the pattern (text); PM-(IJ)$[x,y]$ (TM-(IJ)$[x,y]$) is just abbreviated notation for this.

In the output of the pattern analysis, row j of the PM-(IJ) arrays contains the i and j indices of the first $2k+1$ locations in which the following two symbol sequences (of equal length) have different symbols:

$p_{1,j+1}, p_{2,j+1}, \ldots, p_{m_1,j+1},$	$p_{1,j+2}, p_{2,j+2}, \ldots, p_{m_1,j+2},$	$\cdots$	$p_{1,m_2}, p_{2,m_2}, \ldots, p_{m_1,m_2}$

$p_{1,1}, p_{2,1}, \ldots, p_{m_1,1},$	$p_{1,2}, p_{2,2}, \ldots, p_{m_1,2},$	$\cdots$	$p_{1,m_2-j}, p_{2,m_2-j}, \ldots, p_{m_1,m_2-j}$

In other words, PM-I$[j,v]$=f and PM-J$[j,v]$=g means that $p_{f,j+g} \neq p_{f,g}$ and this is the vth mismatch from top down, left to right (column-major order) between the two subpatterns. If there are only $c < 2k+1$ mismatches then the default values m_1+1 and m_2+1 have been stored in PM-I$[j; c+1, \ldots, 2k+1]$ and PM-J$[j; c+1, \ldots, 2k+1]$ respectively.

The boxes in the above sequences delimit different columns of symbols in the pattern array. Let $*$ denote the sequence $1, 2, \ldots, m_1$. Then column x of the pattern may be denoted by $p_{*,x}$, and we may say that row j of PM-(IJ) contains the first $2k+1$ locations in which $p_{*,j+1}, p_{*,j+2}, \ldots, p_{*,m_2}$ has different symbols than $p_{*,1}, p_{*,2}, \ldots, p_{*,m_2-j}$. It should be clear that PM-(IJ) as a whole records the first $2k+1$ mismatches between two copies of the pattern over all (m_2-1) horizontal offsets possible between the copies.

The text analysis algorithm consists of $n_1 - m_1 + 1$ iterations. In each iteration we find all occurrences of the pattern in a different $m_1 \times n_2$ subarray of the text. This section of the text is analyzed into the TM-(IJ) arrays. After iteration i, $0 \leq i \leq n_1 - m_1$, row j of the TM-(IJ) arrays contains the i and j indices of the first $k+1$ locations in which the following two sequences have different symbols:

$t_{i+1,j+1}, t_{i+2,j+1}, \ldots, t_{i+m_1,j+1},$	$t_{i+1,j+2}, t_{i+2,j+2}, \ldots, t_{i+m_1,j+2},$	$\cdots$	$t_{i+1,j+m_2}, \ldots, t_{i+m_1,j+m_2}$

$p_{1,1}, p_{2,1}, \ldots, p_{m_1,1},$	$p_{1,2}, p_{2,2}, \ldots, p_{m_1,2},$	$\cdots$	$p_{1,m_2}, \ldots, p_{m_1,m_2}$

After iteration i, TM-I$[j,v]$=f and TM-J$[j,v]$=g means that $t_{i+f,j+g} \neq p_{fg}$ and this is the vth mismatch from top down, left to right (column-major order) between the pattern and

the text subarray with upper left corner $i+1, j+1$ and lower right corner $i+m_1, j+m_2$. If there are only $c < k+1$ mismatches then we enter the default values m_1+1 and m_2+1 in TM-I$[j; c+1, \ldots, k+1]$ and TM-J$[j; c+1, \ldots, k+1]$ respectively.

As with the pattern, the boxes in the above sequences delimit different columns of symbols in m_1 contiguous rows of the text array. Recall that we have defined $*$ to denote the sequence $1, 2, \ldots, m_1$. Thus we may say that, after iteration i of the text analysis, row j of TM-(IJ) contains the first $k+1$ locations in which $t_{i+*,j+1}, t_{i+*,j+2}, \ldots, t_{i+*,j+m_2}$ has different symbols than $p_{*,1}, p_{*,2}, \ldots, p_{*,m_2}$.

Note that the end of every i-iteration, we have found all occurrences of the pattern in rows $i+1$ through $i+m_1$ of the text array, since TM-I$[j, k+1]$=m_1+1 and TM-J$[j, k+1]$=m_2+1 $(0 \leq j \leq n_2 - m_2)$ means that there is an occurrence of the pattern in the text (having at most k mismatches) with upper left corner $i+1, j+1$ and lower right corner $i+m_1, j+m_2$. At the end of every i-iteration we report (output) all pattern occurrences by examining the value in column $k+1$ of every row of the TM-(IJ) arrays, since the TM-(IJ) arrays are reused in the next i-iteration (alternatively, we could use three-dimensional TM-(IJ) arrays, using the i variable as the additional index, and only report the pattern occurrences at the end of the algorithm).

The procedure we use within an i-iteration is a direct adoption of the string matching algorithm of [4]. The pattern is treated as an $m_1 \times m_2$ length string and the text as an $m_1 \times n_2$ length string by following column-major ordering through these two-dimensional arrays. We now give our two-dimensional text analysis algorithm, and then explain it in the following discussion.

```
Algorithm 2D-TEXT-ANALYSIS
  for i = 0 to n_1 - m_1 do
    {text being acted on in this iteration is t_{i+*,j}, 1 ≤ * ≤ m_1, 1 ≤ j ≤ n_2}
    {Initialize TM-(IJ) arrays and auxiliary variables}
    TM-I[0,...,n_2 - m_2; 1,...,k+1] ← m_1 + 1
    TM-J[0,...,n_2 - m_2; 1,...,k+1] ← m_2 + 1
    r, i', j' ← 0
    for j = 0 to n_2 - m_2 do
      b ← 0
      if j < j' or (j = j' and i < i') then
        MERGE(i,j,r,i',j',b)
      end if-then
      if b < k + 1 then
        r ← j
        EXTEND(i,j,i',j',b)
      end if-then
    end for-do
    {report matches for t_{i+*,j}, 1 ≤ * ≤ m_1, 1 ≤ j ≤ n_2}
  end for-do
```

In each j-iteration (inner for-do loop), 2D-TEXT-ANALYSIS checks whether there are $> k$ mismatches between the pattern and the subarray of the text with upper left corner $i+1, j+1$ and lower right corner $i+m_1, j+m_2$, i.e. between the sequences $p_{*,1}, p_{*,2}, \ldots, p_{*,m_2}$ and $t_{i+*,j+1}, t_{i+*,j+2}, \ldots, t_{i+*,j+m_2}$. Thus this loop slides the pattern to the right one column, or m_1 text symbols, at a time. The variables i' and j' record the row and column, respectively, of the rightmost, lowermost text symbol that we have scanned in previous j-iterations (this is the rightmost symbol scanned when thinking of the text as a string obtained by following the $m_1 \times n_2$ subarray in column-major order). The variable r records the iteration in which we arrived at this position. Procedure MERGE finds mismatches between $t_{i+1,j+1}, \ldots, t_{i'j'}$ and $p_{1,1}, \ldots, p_{i'-i,j'-j}$, again following column-major ordering through the text subarray $t_{i+*,j}$ $(1 \leq * \leq m_1, 1 \leq j \leq n_2)$ and the pattern

array. MERGE reports in b the number of mismatches found. If $b < k+1$, then procedure EXTEND is called to process the text from location $i'+1, j'+1$ on. EXTEND operates by scanning the text symbols and comparing them to the corresponding pattern symbols. It continues until it finds the $(k+1)^{\text{th}}$ mismatch of this j-iteration (in which case it resets i' and j' to the text position of this mismatch) or until it arrives at $t_{i+m_1, j+m_2}$ (in which case i' and j' are reset to $i+m_1$ and $j+m_2$ respectively). In the latter situation, if the comparison with the corresponding pattern symbol does not result in the $(k+1)^{\text{th}}$ mismatch, we have found an occurrence of the pattern in the text with upper left corner $i+1, j+1$ and lower right corner $i+m_1, j+m_2$ which has at most k mismatches. Note that the current iteration number j is saved in r whenever EXTEND is called, since EXTEND always resets i' and j'.

We now discuss and present the MERGE procedure. Following this, we give the EXTEND procedure and then conclude the section with the analysis of the complexity of 2D-TEXT-ANALYSIS.

In outer iteration i, inner iteration j, MERGE records the mismatches it finds in TM-(IJ)$[j; 1, \ldots, b]$ $(b \leq k+1)$ by using the following information computed in previous j-iterations of this i-iteration:

1. Mismatches occurring from a previous match attempt of the pattern on the text subarray with upper left corner $i+1, r+1$, which occurred in iteration r. However, mismatches which occurred in this iteration which were in text locations $< i+1, j+1$ in column-major order are irrelevant when checking for an occurrence of the pattern starting at $(\geq)$ location $i+1, j+1$ in the text (as we are doing in the present iteration). Thus, letting q be the smallest integer satisfying TM-J$[r, q] > j - r$, MERGE uses the information in TM-(IJ)$[r; q, \ldots, k+1]$. This is illustrated in Figure 1.

2. Mismatches occurring between two copies of the pattern offset horizontally by $j - r$ columns. Let s be the the largest integer $(\leq 2k+1)$ satisfying both PM-I$[j - r, s] \leq i' - i$ and PM-J$[j - r, s] \leq j' - j$. Then MERGE uses the information in PM-(IJ)$[j - r; 1, \ldots, s]$. This is illustrated in Figure 2.

Figure 3 shows the relationship between the information in TM-(IJ)$[r; q, \ldots, k+1]$ (Figure 1) and PM-(IJ)$[j - r; 1, \ldots, s]$ (Figure 2), and how MERGE uses it to compute TM-(IJ)$[j; 1, \ldots, b]$. It also illustrates the scope of EXTEND.

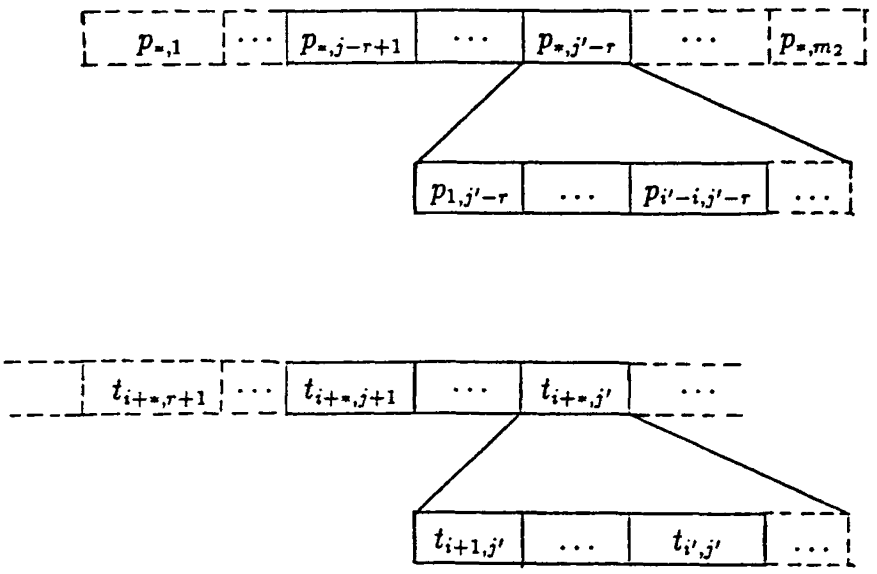

Figure 1: TM-(IJ)$[r; q, \ldots, k+1]$ gives all the mismatches between these sequences.

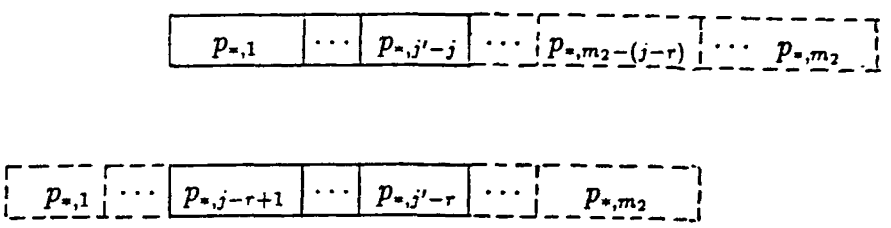

Figure 2: PM-(IJ)$[j-r; 1, \ldots, s]$ gives all $2k+1$ first mismatches between these sequences.

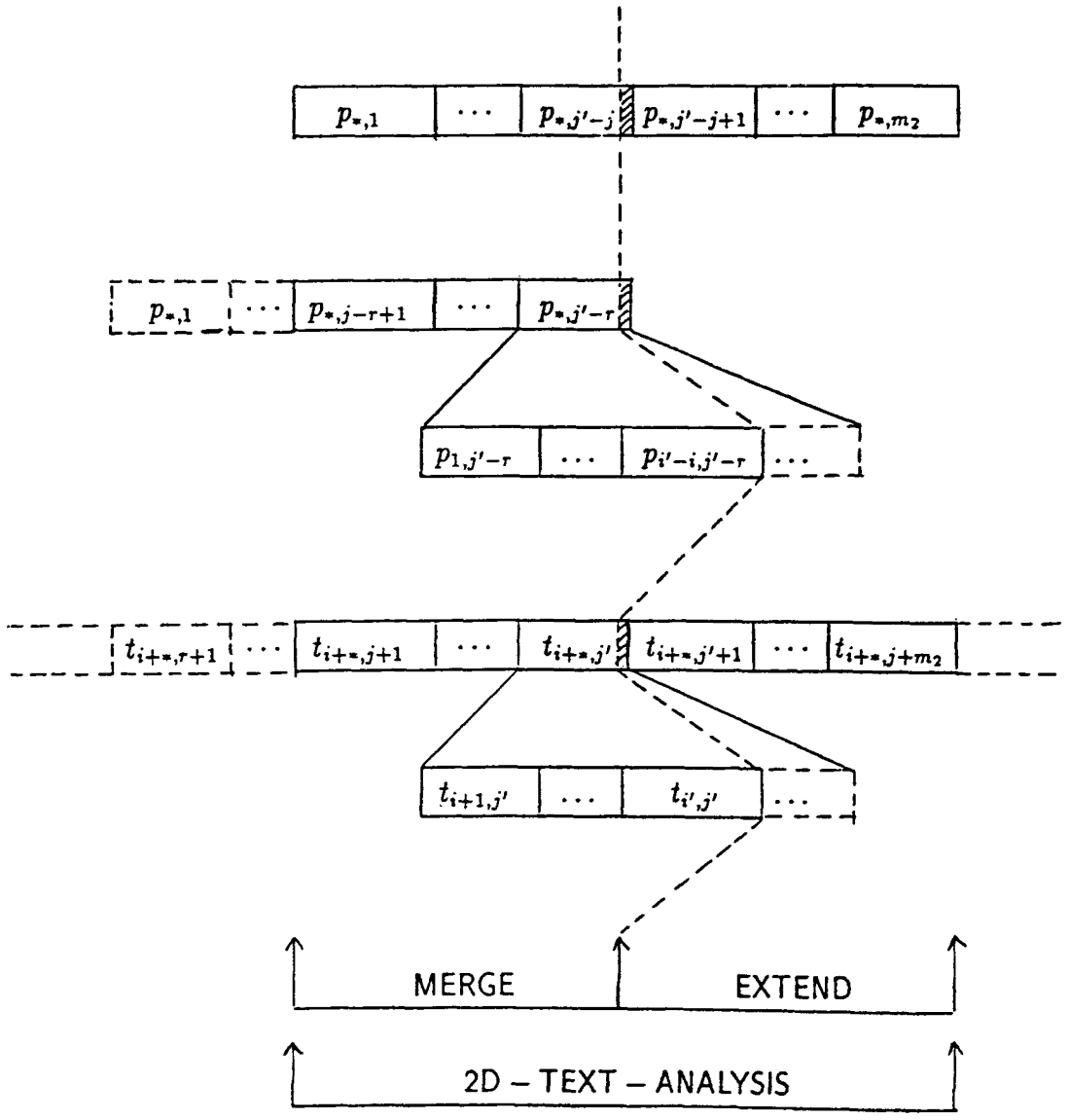

Figure 3: MERGE uses information from Figures 1 and 2 to compute TM-(IJ)$[j; 1, \ldots, k+1]$. If MERGE is unable to complete the job, then EXTEND completes it.

As in [4], case analysis is employed in order to explain how MERGE uses the previously computed information. First, we define two conditions on any location x, y ($i+1 \leq x \leq i'$, $j+1 \leq y \leq j'$) in the text:

Condition 1 A mismatch occurred at x, y in previous iteration r. This means that $t_{x,y} \neq p_{x-i,y-r}$ and for some d, $q \leq d \leq k+1$, $x - i$=TM-I$[r, d]$ and $y - r$=TM-J$[r, d]$. In Figure 3, this corresponds to a mismatch between the bottom line and the middle line.

Condition 2 Consider laying one copy of the pattern starting at (upper left corner at) $t_{i+1,r+1}$ and another copy starting at $t_{i+1,j+1}$. The condition is: A mismatch occurred at x, y in the text between the two copies of the pattern which lay over this position. This means that $p_{x-i,y-r} \neq p_{x-i,y-j}$ and for some f, $1 \leq f \leq s$, $x - i$=PM-I$[j - r, f]$ and $y - j$=PM-J$[j - r, f]$. In Figure 3, this corresponds to a mismatch between the middle and upper lines.

Location x, y may satisfy one of these conditions, both of them, or neither. Thus we have three possible cases.

Using the knowledge (acquired from TM-(IJ)$[r, d]$ and PM-(IJ)$[j - r, f]$) of which conditions location x, y ($i+1 \leq x \leq i'$, $j+1 \leq y \leq j'$) satisfies, we can apply case analysis to x, y to address the question of whether $t_{x,y}$ matches $p_{x-i,y-j}$:

Case 0 Location x, y does not satisfy Condition 1 or Condition 2. In this case, the text symbol at location x, y matches the symbol at location $x-i, y-j$ of the pattern, since $t_{x,y} = p_{x-i,y-r}$ (Condition 1 is not satisfied) and $p_{x-i,y-r} = p_{x-i,y-j}$ (Condition 2 is not satisfied). Thus we do not need to scan and compare $t_{x,y}$ and $p_{x-i,y-j}$.

Case 1 Location x, y satisfies one condition but not the other. In this case, we have a mismatch between $t_{x,y}$ and $p_{x-i,y-j}$, since if $t_{x,y} \neq p_{x-i,y-r}$ (Condition 1 is satisfied) and $p_{x-i,y-r} = p_{x-i,y-j}$ (Condition 2 is not satisfied) then $t_{x,y} \neq p_{x-i,y-j}$, and if $t_{x,y} = p_{x-i,y-r}$ (Condition 1 is not satisfied) and $p_{x-i,y-r} \neq p_{x-i,y-j}$ (Condition 2 is satisfied) then also $t_{x,y} \neq p_{x-i,y-j}$. Here we also do not scan and compare $t_{x,y}$ and $p_{x-i,y-j}$, but we do increment the mismatch count variable b and set TM-(IJ)$[j, b]$.

Case 2 Location x, y satisfies both conditions. Here we cannot make any conclusion as to whether $t_{x,y}$ and $p_{x-i,y-j}$ match or not, and thus have to scan and compare the

symbols. If the comparison results in a mismatch, we have to increment b and set TM-(IJ)$[j, b]$.

Procedure MERGE and its proof of correctness follow:

Procedure MERGE(i, j, r, i', j', b)

{Input: TM-(IJ)$[r; q, \ldots, k+1]$ and PM-(IJ)$[j-r; 1, \ldots, s]$ }

{Initialize variables d and f; d is used in TM-(IJ)$[r, d]$ and f in PM-(IJ)$[j-r, f]$ }

$d \leftarrow q$

$f \leftarrow 1$

while not (Case A or Case B or Case C) do

{Case A: $b = k+1$, in which case $k+1$ mismatches have been found

Case B: $d = k+2$, in which case we have passed location i', j' in the text.

By the method that d is incremented, when it has been assigned the value $k+1$ then in the middle line of Figure 3 we are directly over location i', j' of the bottom line.

Case C: (j+PM-J$[j-r, f] > j'$ or

(j+PM-J$[j-r, f] = j'$ and i+PM-I$[j-r, f] > j'$)) and

TM-I$[r, d] = m_1 + 1$ and TM-J$[r, d] = m_2 + 1$

The first conjunct means that in the top line of Figure 3 we have already passed location i', j' of the bottom line. The second and third conjuncts mean that there is an occurrence of the pattern at $t_{i+1,j+1}$ with $d-1$ mismatches and in the middle line of Figure 3 we have also passed location i', j' of the bottom line. }

if j+PM-J$[j-r, f] > r$+TM-J$[r, d]$ or

(j+PM-J$[j-r, f] = r$+TM-J$[r, d]$ and i+PM-I$[j-r, f] > i$+TM-I$[r, d]$) then

{Case 1: Condition 1 is satisfied: next mismatch is recorded in TM-(IJ); there is no mismatch recorded for this location x, y in PM-(IJ).

Thus $t_{x,y} \neq p_{x-i,y-r}$ and $p_{x-i,y-r} = p_{x-i,y-j}$.}

```
        b ← b + 1
        TM-I[j, b] ← TM-I[r, d]
        TM-J[j, b] ← TM-J[r, d] − (j − r)
        d ← d + 1
    else
        if j+PM-J[j − r, f] < r+TM-J[r, d] or
            (j+PM-J[j − r, f] = r+TM-J[r, d] and i+PM-I[j − r, f] < i+TM-I[r, d]) then
            {Case 1: Condition 2 is satisfied: next mismatch is recorded in PM-(IJ);
            there is no mismatch recorded for this location x, y in TM-(IJ). Thus
            t_{x,y} = p_{x−i,y−r} and p_{x−i,y−r} ≠ p_{x−i,y−j}.}
            b ← b + 1
            TM-I[j, b] ← PM-I[j − r, f]
            TM-J[j, b] ← TM-J[j − r, f]
            f ← f + 1
        else
            {Case 2: next mismatch is recorded in both TM-(IJ) and PM-(IJ).
            Thus for this location x, y, t_{x,y} ≠ p_{x−i,y−r} and p_{x−i,y−r} ≠ p_{x−i,y−j},
            and we have to scan and compare symbols.}
            if p_{PM−I[j−r,f],PM−J[j−r,f]} ≠ t_{i+PM−I[j−r,f],j+PM−J[j−r,f]} then
                b ← b + 1
                TM-I[j, b] ← PM-I[j − r, f]
                TM-J[j, b] ← PM-J[j − r, f]
            end if-then
            f ← f + 1
            d ← d + 1
        end if-then-else
    end if-then-else

end while-do
```

In order to correctly perform its function, MERGE must find all mismatches, up to a maximum of $k+1$, that occur between $t_{i+1,j+1}, \ldots, t_{i'j'}$ and $p_{1,1}, \ldots, p_{i'-i,j'-j}$. Consider

the invocation of MERGE in outer iteration i, inner iteration j of the text analysis.

Theorem 1 *If there are $\geq k+1$ mismatches in locations $\geq i+1, j+1$ and $\leq i', j'$ (in column-major order), then* MERGE *finds the first $k+1$ of them. If there are $< k+1$ mismatches, then* MERGE *finds all of them.*

Proof: Condition 1 holds for $\leq k+1$ locations which are $\geq i+1, j+1$ and $\leq i', j'$. Let z be the number of locations $\geq i+1, j+1$ and $\leq i', j'$ for which Condition 2 holds. We know nothing about z. Suppose PM-(IJ)$[j-r; 1, \ldots, z]$ records all of the mismatches occurring between the two copies of the pattern which are offset horizontally from each other by $j-r$ columns. Then by the case analysis, MERGE finds all of the mismatches in locations $i+1, j+1$ through i', j', since mismatches can only occur when Condition 1 or Condition 2 or both hold, all ($\leq k+1$) mismatches occurring under Condition 1 are recorded in TM-(IJ), and all z mismatches under Condition 2 are recorded in PM-(IJ). But PM-(IJ)$[j-r; 1, \ldots, 2k+1]$ contains no more than $2k+1$ mismatches, i.e. for MERGE to find all mismatches ($\leq k+1$) in the range $i+1, j+1$ through i', j', we must have $z \leq 2k+1$. Thus, we now need to show that we never need more than $2k+1$.

If

$$\text{PM-J}[j-r, 2k+1] > j'-j \text{ or}$$
$$(\text{PM-J}[j-r, 2k+1] = j'-j \text{ and PM-I}[j-r, 2k+1] \geq i'-i\ )$$

then PM-(IJ) records all mismatches under Condition 2 between the two copies of the pattern offset by $j-r$ columns which occur in locations $\leq i', j'$, and the Theorem follows. Otherwise, we have

$$\text{PM-J}[j-r, 2k+1] < j'-j \text{ or}$$
$$(\text{PM-J}[j-r, 2k+1] = j'-j \text{ and PM-I}[j-r, 2k+1] < i'-i\ ),$$

which means that there are $2k+1$ locations $\geq i+1, j+1$ and $< i', j'$ for which Condition 2 holds. Condition 1 holds for $\leq k$ locations in this range since, by the way variable d is incremented in MERGE, when $d = k+1$ then in the middle line of Figure 3 we are directly over location i', j' of the bottom line. This means that we are left with $(2k+1-(\leq k) =)$ $\geq k+1$ locations, which are $\geq i+1, j+1$ and $< i', j'$, where Condition 2 holds and Condition 1 does not, i.e. at least twice as many as the number of mismatches we are looking for. Thus we can satisfy Case 1 with $\leq k+1$ mismatches recorded in TM-(IJ)

(Condition 1), and $\leq 2k+1$ mismatches recorded in PM-(IJ) (Condition 2). Thus, all of these locations, plus those that satisfy Case 2 (where mismatches are found by scanning and comparing), suffice to establish that there is no occurrence of the pattern with $\leq k$ mismatches and the Theorem follows. □

Procedure EXTEND finds mismatches between (column-major ordered) sequences

$$t_{i'+1,j'}, \ldots, t_{i+m_1,j+m_2} \text{ and } p_{i'-i+1,j'-j}, \ldots, p_{m_1,m_2} \text{ (if } i' < i+m_1),$$

or the sequences

$$t_{i+1,j'+1}, \ldots, t_{i+m_1,j+m_2} \text{ and } p_{1,j'-j+1}, \ldots, p_{m_1,m_2} \text{ (if } i' = i+m_1).$$

This is done by explicitly scanning and comparing corresponding pattern and text symbols. EXTEND finishes once it finds the $(k+1)^{\text{th}}$ mismatch, or when it arrives at $t_{i+m_1,j+m_2}$ (and compares it with the corresponding pattern symbol), in which case there is an occurrence of the pattern with at most k mismatches starting at $t_{i+1,j+1}$ (if the comparison does not result in the $(k+1)^{\text{th}}$ mismatch). The procedure follows; its correctness is straightforward.

```
Procedure EXTEND(i,j,i',j',b)
  while b < k + 1 and j' - j < m_2 do
    if i' - i = m_1 then
      j' ← j' + 1
      i' ← i
    end if-then
    while b < k + 1 and i' - i < m_1 do
      i' ← i' + 1
      if t_{i',j'} ≠ p_{i'-i,j'-j} then
        b ← b + 1
        TM-I[j, b] ← i' - i
        TM-J[j, b] ← j' - j
      end if-then
    end while-do
  end while-do
```

We are now ready to determine the complexity of 2D-TEXT-ANALYSIS. We concentrate on a single i-iteration. The initialization of the TM-(IJ) arrays takes time $O(kn_2)$ and the time to report the mismatches is $O(n_2)$ (recall that we do this by examining TM-(IJ)$[0,\ldots,n_2-m_2;k+1]$). Within each j-iteration of the for-do loop, the operations excluding MERGE and EXTEND require $O(1)$ time. MERGE operates on the $2k+1$ entries in PM-(IJ)$[j-r;1,\ldots,2k+1]$ and the $k+1$ entries in TM-(IJ)$[r;1,\ldots,k+1]$. As stated in [4], each of the operations of MERGE can be charged to one of these $3k+2$ entries (actually $6k+4$ in our case, since PM-(IJ) and TM-(IJ) are both really two arrays) in such a way that each entry is charged with $O(1)$ operations. Thus, MERGE takes time $O(k)$ per j-iteration, which means that it requires $O(kn_2)$ over the course of the for-do loop, i.e. per i-iteration. Since EXTEND scans each symbol in an $m_1 \times n_2$ subarray of the text at most once over the course of the for-do loop, it takes time $O(m_1n_2)$ per i-iteration. Therefore, a single i-iteration has complexity $O(kn_2+n_2+kn_2+m_1n_2) = O(n_2(k+m_1))$, and the complexity of 2D-TEXT-ANALYSIS is $O(n_1n_2(k+m_1))$.

2D-TEXT-ANALYSIS requires $O(k(m_2+n_2)+m_1m_2+n_1n_2)$ space, where the first term arises from the $(m_2-1)\times(2k+1)$ and $(n_2-m_2+1)\times(k+1)$ PM-(IJ) and TM-(IJ) arrays, respectively, and the second and third terms from the pattern and text arrays.

3 Pattern Analysis

The text analysis algorithm requires preprocessed pattern information. In this section we present the pattern analysis, where PM-(IJ)$[1,\ldots,m_2-1;1,\ldots,2k+1]$ is computed for input to 2D-TEXT-ANALYSIS. As before, we extend the techniques of [4] to two dimensions.

Recall that rows j of the PM-(IJ) arrays are to contain the i and j indices of the first $2k+1$ locations in which the following two subpatterns have different symbols:

$p_{1,j+1}, p_{2,j+1}, \ldots, p_{m_1,j+1},$	$p_{1,j+2}, p_{2,j+2}, \ldots, p_{m_1,j+2},$	$\cdots$	$p_{1,m_2}, p_{2,m_2}, \ldots, p_{m_1,m_2}$

$p_{1,1}, p_{2,1}, \ldots, p_{m_1,1},$	$p_{1,2}, p_{2,2}, \ldots, p_{m_1,2},$	$\cdots$	$p_{1,m_2-j}, p_{2,m_2-j}, \ldots, p_{m_1,m_2-j}$

and that PM-I$[j,v]=f$ and PM-J$[j,v]=g$ must mean that $p_{f,j+g} \neq p_{f,g}$ and this is the vth mismatch from top down, left to right (column-major order) between the two subpatterns.

If there are only $c < 2k+1$ mismatches then the default values m_1+1 and m_2+1 need to be stored in PM-I$[j; c+1, \ldots, 2k+1]$ and PM-J$[j; c+1, \ldots, 2k+1]$ respectively.

The PM-(IJ) arrays have $m_2 - 1$ rows. Assume (without loss of generality) that m_2 is a power of two. The pattern analysis has $\log m_2$ stages. In stage l, $1 \leq l \leq \log m_2$, we compute the entries of PM-(IJ)$[2^{l-1}, \ldots, 2^l - 1; 1, \ldots, 2k+1]$. In other words, in stage 1 we compute row 1 of PM-(IJ), in stage 2 we compute rows 2 and 3, in stage 3 rows 4, 5, 6, and 7, ..., and in stage $\log m_2$ we compute rows $m_2/2, \ldots, m_2 - 1$.

In each stage of the pattern analysis, we compute the appropriate rows by applying an adjusted version of a single (outer) i-iteration of the 2D-TEXT-ANALYSIS algorithm. We now specify the input, output, and algorithm for stage l of the pattern analysis, and then explain the adjustments made to (differences from) the i-iteration of the text analysis algorithm.

Recall that we have defined * to denote the sequence $1, \ldots, m_1$. The input to stage l, $1 \leq l \leq \log m_2$, of the pattern analysis consists of:

1. The pattern subarray $p_{*,1}, p_{*,2}, \ldots, p_{*,m_2-2^{l-1}}$, which plays the role of the pattern in the pattern analysis (adjusted text analysis) algorithm.

2. The pattern subarray $p_{*,2^{l-1}+1}, p_{*,2^{l-1}+2}, \ldots, p_{m_1,m_2}$, which plays the role of the text.

3. If $l > 1$, the two (PM-I and PM-J) two-dimensional arrays

$$\text{PM-(IJ)}[1, \ldots, 2^{l-1} - 1; 1, \ldots, \min(2^{\log m_2 - (l-1)} 2k + 1, m_1(m_2 - 2^{l-1}))]$$

   which are the output of the previous $l-1$ stages of the pattern analysis, and which play the role of PM-(IJ). (Accordingly, in stage 1 we do not have any pattern-mismatch information yet, so we will not be able to use MERGE in this stage. This is no drawback, since we only compute one row of PM-(IJ) in stage 1.)

The output of stage l consists of the two arrays

$$\text{PM-(IJ)}[2^{l-1}, \ldots, 2^l - 1; 1, \ldots, z]$$

where $z = 2k+1$ when $l = \log m_2$, and $z = \min(2^{\log m_2 - l} 2k + 1, m_1(m_2 - 2^l))$ when $1 \leq l < \log m_2$. These arrays play the role of the TM-(IJ) arrays.

The algorithm for stage l of the pattern analysis follows. Note that procedures MERGE and EXTEND in this algorithm will also need adjusting from their text analysis form.

These adjustments are straightforward, consisting of compensating for the PM-(IJ) and TM-(IJ) role-playing and defining z rather than $k+1$ to be the number of mismatches to find, and thus are not given.

```
Procedure STAGE(l)
   {Note: i is constant and = 0}
   {Initialize arrays and auxiliary variables}
   PM-I[2^(l-1),...,2^l - 1; 1,...,z] ← m1 + 1
   PM-J[2^(l-1),...,2^l - 1; 1,...,z] ← m2 + 1
   r, i', j' ← 2^(l-1)
   for j = 2^(l-1) to 2^l - 1 do
      b ← 0
      if l > 1 and (j < j' or (j = j' and i < i')) then
         MERGE(i,j,r,i',j',b)
      end if-then
      if b < z then
         r ← j
         EXTEND(i,j,i',j',b)
      end if-then
   end for-do
```

To explain the pattern analysis, we discuss the differences between STAGE and a single i-iteration of the text analysis. This also serves as an extension of the proof of correctness of MERGE in the text analysis to the pattern analysis.

The first difference is that MERGE is not used throughout stage 1. This is because MERGE uses PM-(IJ) information which in stage 1 has not yet been computed. Thus, the single row of PM-(IJ) computed in stage 1 must be computed entirely by EXTEND.

The second difference is the number of mismatches that is looked for within a stage. For stage $\log m_2$, we find up to $2k+1$ mismatches. For stage l, $1 \leq l < \log m_2$, we find up to the minimum between $2^{\log m_2 - l}2k+1$ (instead of $k+1$) and $m_1(m_2 - 2^l)$ (implied by the the lengths of the pattern copies to be matched against each other in the next stage) mismatches. In stage l, the largest pattern copies that are matched occur in iteration

$j = 2^{l-1}$, and have length $m_1(m_2 - 2^{l-1})$. In each successive stage, the pattern length in this iteration gets smaller. Thus, in stage $l-1$, we never need to count more than the $m_1(m_2 - 2^{l-1})$ mismatches that could occur between the pattern copies in the following stage l. In other words, in stage l, $1 \leq l < \log m_2$, we never need to count more than $m_1(m_2 - 2^l)$ mismatches, and this is one term of the min function. The input PM-(IJ) to stage l never contain more than $m_1(m_2 - 2^{l-1})$ mismatches, and the output never more than $m_1(m_2 - 2^l)$ mismatches.

The other term of the min function which gives the number of mismatches to find in stage l is $2^{\log m_2 - l}2k+1$. In the text analysis, the correctness proof of MERGE demonstrated that we need to know the first $2k+1$ locations for which Condition 2 holds in order to find the first $k+1$ mismatches, i.e. the number of locations for which Condition 2 holds must be at least twice the number of mismatches we are looking for. In the pattern analysis, we are constructing PM-(IJ) to contain the first $2k+1$ mismatches between pattern copies. In stage $\log m_2$, we find up to $2k+1$ mismatches (there is no $m_1(m_2 - 2^l)$ bound because this is the final stage), therefore we need $2 \cdot 2k + 1$ locations for which Condition 2 holds stored in the rows of PM-(IJ) computed in previous stages. This means that, in stage l, $1 \leq l \leq \log m_2$, we need $2^{\log m_2 - (l-1)}2k + 1$ locations (up to $m_1(m_2 - 2^{l-1})$) for which Condition 2 holds, and we look for $2^{\log m_2 - l}2k + 1$ mismatches (up to $m_1(m_2 - 2^l)$) for $l < \log m_2$). Thus for stage $\log m_2$ we find up to $2k + 1$ mismatches, and for stage l, $1 \leq l < \log m_2$, we find up to $\min(2^{\log m_2 - l}2k + 1, m_1(m_2 - 2^l))$ mismatches.

The two-dimensional pattern analysis algorithm simply consists of incrementing l and calling STAGE(l) ($l = 1, \ldots, \log m_2$). To determine its complexity, we focus on a single stage l. Within this stage, we further concentrate on a single iteration j, $2^{l-1} \leq j \leq 2^l - 1$. The operations excluding MERGE and EXTEND take $O(1)$ time. As in the text analysis, one invocation of MERGE takes time O("number of mismatches we are looking for"), which is

$$O(\min(2^{\log m_2 - l}2k + 1, m_1(m_2 - 2^l))) \leq O(2k2^{\log m_2 - l})$$

Stage l has 2^{l-1} iterations, thus the total time required by MERGE in the stage is $O(2^l 2k 2^{\log m_2 - l})$ or (maximizing l to $\log m_2$) $O(m_2 k)$. The time required by EXTEND through all j-iterations of stage l is $O(m_1 m_2)$. Thus the total time required by STAGE(l) is $O(m_2 k + m_1 m_2) = O(m_2(k + m_1))$, so the time required by the two-dimensional pattern analysis ($\log m_2$ stages) is $O((k + m_1)m_2 \log m_2)$.

The pattern analysis requires space $O(km_2 + m_1m_2)$, where the first term arises from the $(m_2 - 1) \times (2k + 1)$ PM-(IJ) arrays, and the second from the pattern array.

4 Conclusions

Our algorithm for two-dimensional pattern matching with k mismatches consists of pattern analysis stage followed by a text analysis stage. We have shown that the pattern analysis takes time $O((k + m_1)m_2 \log m_2)$ and the text analysis time $O(n_1n_2(k + m_1))$. Therefore the complexity of our algorithm is $O((k+m_1)(m_2 \log m_2 + n_1n_2))$. Between m_1 and m_2, m_1 has a greater affect on the complexity because of its n_1n_2 factor. If $m_1 > m_2$, we could compensate for this by transposing the pattern and text prior to pattern matching, which changes the complexity of the algorithms of this paper and [2] by substituting $\min(m_1, m_2)$ for m_1 and $\max(m_1, m_2)$ for m_2 in the analyses.

The complexity of our algorithm improves upon that of the previously best known algorithm for this problem ([2]) by a factor of $O(k \min(m_1, m_2)/(k + \min(m_1, m_2)))$. We feel justified in claiming that this is not a a small improvement for two reasons. First, the smallest pattern would likely be 10×10 with respect to a 1000×1000 text array. Second, and more importantly, the smallest realistic k would likely be $\min(m_1, m_2)$ (i.e. allowing one mismatch per row or column of the pattern would be the minimum in practical cases), resulting in a factor of improvement of $O((\min(m_1, m_2))^2/(\min(m_1, m_2)+\min(m_1, m_2))) = O(\min(m_1, m_2))$. For example, consider the case $m_1 = m_2$; denote this simply by m. When $k = m$, our algorithm takes time $O(m(m \log m + n_1n_2))$ while the one of [2] takes time $O(m^2(m \log m + n_1n_2))$.

Our algorithm also requires less space than that of [2]. Ignoring the $O(m_1m_2)$ and $O(n_1n_2)$ space required by both algorithms for storing the pattern and text arrays, our algorithm uses $O(k(m_2 + n_2))$ space, while the one of [2] uses $O(m_1(km_2 + kn_2 + m_1 \max(m_2, k)))$.

It is an open question as to whether a better algorithm can be found for two-dimensional pattern matching with k mismatches. A likely strategy would be employing a MERGE-like function between outer i-iterations to reduce the number of symbol scans performed by EXTEND. The best possible result of such a strategy would be an algorithm which scans each symbol of the text at most once, thus running in time $O(k(m_2 \log m_2 + n_1n_2))$.

Since $O(k(m_2 \log m_2 + n_1n_2))$ is the best complexity one can achieve, our algorithm is

asymptotically optimal if $k \approx m_1$, or $k \approx \min(m_1, m_2)$ if pattern and text transposition is assumed. We feel that in practical cases it will be so that $k \approx \min(m_1, m_2)$ since, as mentioned above, the smallest realistic number of errors allowed would likely be at least of the order of the smaller pattern dimension.

References

[1] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice Hall, 1985.

[2] K. Krithivasan and R. Sitalakshmi, Efficient two-dimensional pattern matching in the presence of errors, *Information Sciences 43*, 1987, pp. 169–184.

[3] G.M. Landau and U. Vishkin, Efficient string matching in the presence of errors, *Proc. 26th IEEE Symposium on Foundations of Computer Science*, 1985, pp. 126–136.

[4] G.M. Landau and U. Vishkin, Efficient string matching with k mismatches, *Theoretical Computer Science 43*, 1986, pp. 239–249.

[5] A. Rosenfeld and A.C. Kak, *Digital Picture Processing*, Academic Press, 1982.