Real Two Dimensional Scaled Matching

Amihood Amir[1⋆], Ayelet Butman[2], Moshe Lewenstein[2], and Ely Porat[2]

[1] Bar-Ilan University
amir@cs.biu.ac.il
[2] Bar-Ilan University
{ayelet,moshe,porately}@cs.biu.ac.il

Abstract. *Scaled Matching* refers to the problem of finding all locations in the text where the pattern, proportionally enlarged according to an *arbitrary real-sized* scale, appears. Scaled matching is an important problem that was originally inspired by Computer Vision.
Finding a *combinatorial* definition that captures the concept of real scaling in discrete images has been a challenge in the pattern matching field. No definition existed that captured the concept of real scaling in discrete images, without assuming an underlying continuous signal, as done in the image processing field. We present a combinatorial definition for real scaled matching that scales images in a pleasing natural manner. W e also present efficient algorithms for real scaled matching. The running time of our algorithm is as follows. If T is a two-dimensional $n \times n$ text array and P is a $m \times m$ pattern array, we find in T all occurrences of P scaled to any real value in time $O(nm^3 + n^2 m \log m)$.

1 Introduction

The original classical string matching problem [7,11] was motivated by text searching. Indeed practically every text editor uses some variant of the Boyer-Moore algorithm [7].
Wide advances in technology, e.g. computer vision, multimedia libraries, and web searches in heterogeneous data, point to a glaring lack of a theory of multidimensional matching [15].
The last decade has seen some progress in this direction. Issues arising from the digitization process were examined by Landau and Vishkin [14]. Once the image is digitized, one wants to search it for various data. A whole body of literature examines the problem of seeking an object in an image.
In reality one seldom expects to find an exact match of the object being sought, henceforth referred to as the *pattern*. Rather, it is interesting to find all text locations that "approximately" match the pattern. The types of differences that make up these "approximations" are:

⋆ Partially supported by ISF grant 282/01. Part of this work was done when the author was at Georgia Tech, College of Computing and supported by NSF grant CCR-01-04494.

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 353–364, 2003.

1. *Local Errors* - introduced by differences in the digitization process, noise, and occlusion (the pattern partly obscured by another object).
2. *Scale* - size difference between the image in the pattern and the text.
3. *Rotation* - The pattern image appearing in the text in a different angle.

Some early attempts to handle local errors were made in [12]. These results were improved in [5]. The algorithms in [5] heavily depend on the fact that the pattern is a rectangle. In reality this is hardly ever the case. In [4], Amir and Farach show how to deal with local errors in non-rectangular patterns.

The rotation problem has proven quite difficult. There is currently no known asymptotically efficient algorithm for finding all rotated occurrences of a pattern in an image. Fredriksson and Ukkonen [8], give a reasonable definition of rotation in discrete images and introduce a filter for seeking a rotated pattern.

More progress has been made with scaling. In [6] it was shown that all occurrences of a given rectangular pattern in a text can be found in *all discrete scales* in linear time. By discrete scales we mean natural numbers, i.e. the pattern scaled to sizes $1, 2, 3, \ldots$. The algorithm was linear for fixed bounded alphabets, but was not linear for unbounded alphabets. This result was improved in [2].

The above papers dealt with *discrete scales* only. There is some justification for dealing with discrete scales in a combinatorial sense, since it is not clear what is a fraction of a pixel. However, in reality an object may appear in non-discrete scales. It is necessary to, both, define the combinatorial meaning of such scaling, and present efficient algorithms for the problem's solution. A first step in this direction appeared in [1], however that paper was limited to *string* matching with non-discrete scales. There was still no satisfactory rigorous definition of scaling in an "exact matching" sense of combinatorial pattern matching.

In this paper we present a definition for *scaled pattern matching* with arbitrary real scales. The definition is pleasing in a "real-world" sense. We have scaled "lenna" to non-discrete scales by our definition and the results look natural (see Figure 1). This definition was inspired by the idea of digitizing analog signals by sampling, however, it does not assume an underlying continuous function thus stays on the combinatorial pattern matching field. We believe this is the natural way to define combinatorially the meaning of scaling in the signal processing sense.

We believe this definition, that had been sought by researchers in pattern matching since at least 1990, captures *scaling* as it occurs in images, yet has the necessary combinatorial features that allows developing deterministic algorithms and analysing their worst-case complexity. Indeed we present an efficient algorithm for real scaled two dimensional pattern matching.

The running times of our algorithm is as follows. If T is a two-dimensional $n \times n$ text array and P is a $m \times m$ pattern array, we find in T all occurrences of P scaled to any real value in time $O(nm^3 + n^2 m \log m)$.

The main achievements of this paper are pinning down a rigorous combinatorial definition for exact real scaling in images and producing efficient algorithms for scaled matching. The new techniques developed in this paper are analysis of

Fig. 1. An original image, scaled by 1.3 and scaled by 2, using our combinatorial definition of scaling.

the properties of scaled arrays and two-dimensional dictionary matching with a compressed dictionary.

2 Scaled Matching Definition

Definition 1. *Let T be a two-dimensional $n \times n$ array over some finite alphabet Σ.*

1. *The* unit pixels array *for T (T^{1X}) consists of n^2 unit squares, called pixels in the real plane R^2. The corners of the pixel $T[i,j]$ are $(i-1,j-1),(i,j-1),(i-1,j)$, and (i,j). Hence the pixels of T form a regular $n \times n$ array that covers the area between $(0,0),(n,0),(0,n)$, and (n,n). Point $(0,0)$ is the* origin *of the unit pixel array. The* center *of each pixel is the geometric center point of its square location. Each pixel $T[i,j]$ is identified with the value from Σ that the original array T had in that position. We say that the pixel has a color from Σ. See figure 2 for an example of the grid and pixel centers of a 7×7 array.*
2. *Let $r \in \Re$, $r \geq 1$. The* r-ary pixels array *for T (T^{rX}) consists of n^2* r-squares*, each of dimension $r \times r$ whose* origin *is $(0,0)$ and covers the area between $(0,0),(nr,0),(0,nr)$, and (nr,nr). The corners of the pixel $T[i,j]$ are $((i-1)r,(j-1)r),(ir,(j-1)r),((i-1)r,jr)$, and (ir,jr). The* center *of each pixel is the geometric center point of its square location.*

Notation: Let $r \in \Re$. $\|r\|$ denotes the *rounding* of r, i.e. $\|r\| = \begin{cases} \lfloor r \rfloor & \text{if } r - \lfloor r \rfloor < .5; \\ \lceil r \rceil & \text{otherwise.} \end{cases}$
There may be cases where we need to round 0.5 down. For this we denote:
$|\lfloor r \rfloor| = \begin{cases} \lfloor r \rfloor & \text{if } r - \lfloor r \rfloor \leq .5; \\ \lceil r \rceil & \text{otherwise.} \end{cases}$

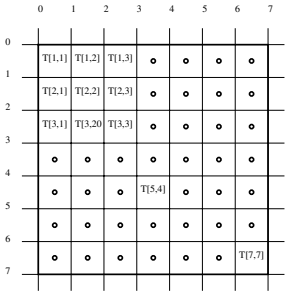

Fig. 2. The grid and pixel centers of a unit pixel array for a 7×7 array.

Definition 2. *Let T be an $n \times n$ text array and P be an $m \times m$ pattern array over alphabet Σ. Let $r \in \Re,\ 1 \leq r \leq \frac{n}{m}$.*
We say that there is an occurrence of P scaled to r *at text location $[i, j]$ if the following condition holds:*
Let T^{1X} be the unit pixels array of T and P^{rX} be the r-ary pixel arrays of P. Translate P^{rX} onto T^{1X} in a manner that the origin of P^{rX} coincides with location $(i-1, j-1)$ of T^{1X}. Every center of a pixel in T^{1X} which is within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ has the same color as the r-square of P^{rX} in which it falls.
The colors of the centers of the pixels in T^{1X} which are within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ define a $\|mr\| \times \|mr\|$ array over Σ. This array is denoted by P^r and called P scaled to r.

It is possible to find all scaled occurrences of an $m \times m$ pattern in an $n \times n$ text in time $O(n^2m^2)$. Such an algorithm, while not trivial, is nonetheless achievable with known techniques. We present below an $O(nm^3 + n^2 m \log m)$ algorithm.
The efficiency of the algorithm results from the properties of scaling. The scaling definition needs to accommodate a conflict between two notions, the continuous (represented by the real-number scale), and the discrete (represented by the array representation of the images). Understanding, and properly using, the shift from the continuous to the discrete and back are key to the efficiency of our algorithms. To this effect we need the following functions.

Definition 3. *Let k be a discrete length of a pattern prefix in any dimension, i.e. the number of consecutive rows starting from the pattern's beginning, or the length of a row prefix. Let $r \in \Re$ be a scale, and let N be the natural numbers. We define the function $D : N \times \Re \to N$ as follows: $D(k, r) = \|kr\|$.*

We would like to define an "inverse" function $D^{-1} : N \times N \to \Re$ with the property $D^{-1}(k, D(k, r)) = r$. However, that is not possible since D is not injective. Claim 2, which follows from the definition, below tells us that for a fixed k there is a structure to the real numbers r that are mapped to the same element $D(k, r)$, namely, they form an interval $[r_1, r_2)$.

Claim. Let $r_1, r_2 \in \Re,\ k \in N$ such that $D(k, r_1) = D(k, r_2)$ and let $r \in \Re,\ r_1 < r < r_2$. Then $D(k, r) = D(k, r_1)$.

Definition 4. *Let $k, \ell \in N$. Define*
$L^{-1}(k,\ell) = \begin{cases} 1 & \text{if } k = \ell; \\ \frac{(\ell-0.5)}{k} & \text{otherwise.} \end{cases}$
and $R^{-1}(k,\ell) = \frac{(\ell+0.5)}{k}$.

It is easy to see that $L^{-1}(k,\ell) = \min\{r \in \Re | D(k,r) = \ell\}$ and that $R^{-1}(k,\ell) = \min\{r \in \Re | D(k,r) = \ell + 1\}$.
The L^{-1} and R^{-1} functions are designed to give a range of scales whereby a pattern sub-range of length k may scale to a sub-range of scale ℓ. The following claim follows from the definition.

Claim. Let P be an $m \times m$ pattern and T an $n \times n$ text. Let $k \leq m$ and $\ell \leq n$, and let $[L^{-1}(k,\ell), R^{-1}(k,\ell))$ be the range of scales defined by L^{-1} and R^{-1}. Then the difference in number of rows (or number of columns) between P^{r_1} and P^{r_2}, for any two $r_1, r_2 \in [L^{-1}(k,\ell), R^{-1}(k,\ell))$ can not exceed $m + 2$.

3 The Number of Different Scaled Patterns

We utilize various properties of the scaled patterns to aid our search. One of the difficulties presented are that it is possible to have several values $r_1, r_2, ..., r_k \in \Re$ for which $P^{r_1}, P^{r_2}, ..., P^{r_k}$ are different matrices yet all have the same dimensions. See Figure 3. The following claim limits the overall number of different possible matrices that represent scaled occurrences of a given pattern matrix P.

Claim. Let P be an $m \times m$ pattern over finite alphabet Σ. Then there are $O(nm)$ different matrices representing the occurrences of P scaled to all $r \in \Re,\ 1 \leq r \leq \frac{n}{m}$.

Proof. There are $n - m$ different possible sizes of matrices representing a scaled P whose maximum size is $n \times n$. By lemma 1 below, each one of the $n - m$ different matrix sizes has at most m possibilities of matrices representing scaled versions of P, for a total of $O(nm)$. □

Lemma 1. *Let $m \times m$ pattern P be scaled to size $\ell \times \ell,\ \ell \geq m$. Then there are k different intervals, $[a_1, a_2), [a_2, a_3), ..., [a_k, a_{k+1}),\ k \leq m, 1 \leq a_1 < a_2 < \cdots < a_{k+1}$ for which the following hold:*
1. *$P^{r_1} = P^{r_2}$, if $r_1, r_2 \in [a_i, a_{i+1})$, $1 \leq i \leq k$.*
2. *$P^{r_1} \neq P^{r_2}$, if r_1 and r_2 are in different intervals.*
3. *P^r is an $\ell \times \ell$ matrix iff $r \in [a_1, a_{k+1})$.*

Proof. Omitted for space reasons.

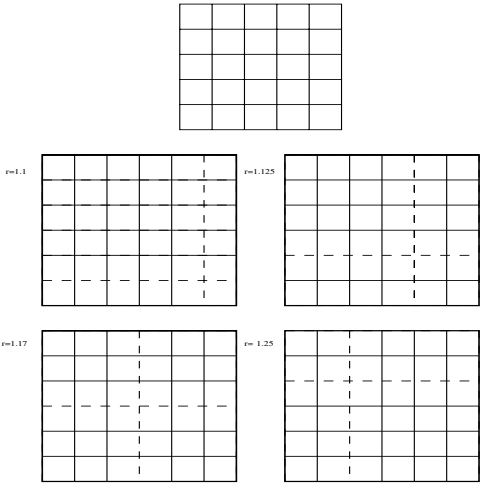

Fig. 3. The 5×5 pattern scaled to $1.1, 1.125, 1.17$ and 1.25 produces a 6×6 pattern. In each of these cases some row and some column needs to be repeated. The dashed grid line indicates the repeating row or column (both rows or columns on the two sides of the dashed grid line are equal).

4 The Scaled Matching Algorithm's Idea

The naive, straightforward idea is to construct a dictionary of the $O(nm)$ different possible scaled occurrences of P and use a two-dimensional dictionary matching algorithm (e.g. [3,10,9]) that can scan the text in linear time and find all dictionary occurrences. The trouble with this idea is that the different matrices in the dictionary range in sizes from m^2 to n^2 which will make the dictionary size $O(n^3m)$, which we are not willing to pay.
Our idea, then, is to keep the dictionary in a compressed form. The compression we use is *run-length* of the rows.

Definition 5. *Let $S = \sigma_1\sigma_2 \cdots \sigma_n$ be a string over some alphabet Σ. The* run-length representation *of string S is the string $S' = \sigma_1'^{r_1}\sigma_2'^{r_2} \cdots \sigma_k'^{r_k}$ such that: (1) $\sigma_i' \neq \sigma_{i+1}'$ for $1 \leq i < k$; and (2) S can be described as concatenation of the symbol σ_1' repeated r_1 times, the symbol σ_2' repeated r_2 times, ..., and the symbol σ_k' repeated r_k times.*

We denote by $S'^{\Sigma} = \sigma_1'\sigma_2' \cdots \sigma_k'$, the symbol part *of S', and by $c(S)$, the vector of natural numbers $r_1, r_2, ..., r_k$, the* run-length part *of S'. We say that the* number of runs of S is k *and denote it by $|c(S)|$. For j, $1 \leq j \leq k$ denote* $\mathsf{prefixsum}(S, j) = \sum_{i=1}^{j} r_i$.

Since every scaled occurrence has at most m different rows, each repeating a certain number of times, and each row is of run-length at most m, then we can encode the information of every pattern scaled occurrence in space $O(m^2)$. Now our dictionary is of size $O(nm^3)$. The construction details are left for the

journal version. The challenge is to perform dictionary matching using a compressed dictionary. We show in Section 5 that such a search can be done in time $O(n^2 m \log m)$.

The idea behind the text searching is as follows: For every text location $[i, j]$, we assume that there is a pattern scaled occurrence beginning at that location. Subsequently, we handle every one of the pattern m rows separately (in time $O(\log m)$ for each row). For every row, we establish the number of times this row repeats in the text. This allows us to narrow down the range of possible scales for this occurrences and compare the row to an appropriately scaled pattern row from the dictionary. The high-level description of the text searching algorithm appears below. We denote the ℓth row of P by P_ℓ.

Scaled Text Scanning (high level)

For every text location $[i, j]$ do:
  Set $k \leftarrow i$.
  For pattern row $\ell = 1$ to m do:

1. Establish the number of times the subrow of T starting at location $[k, j]$ and whose run-length equals the run-length of P_ℓ repeats in the text.
2. If this number of repetitions is incompatible with the numbers for rows $1, ..., \ell - 1$ then halt – no scaled occurrence possible at $[i, j]$.
3. Binary search all the dictionary rows P_ℓ in the appropriate scale compared to the run-length $c(P_\ell)$ subrow starting at $T[k, j]$. If no match then halt – no scaled occurrence at $[i, j]$.
   update the possible range of scales and k.

  EndFor
EndFor

end Scaled Text Scanning

5 Scaled Search Algorithm Implementation

We describe the details of efficiently computing the high-level steps of Section 4.

5.1 Computing Text Subrow Repetition

Consider a given text location $[i, j]$. We are interested in ascertaining whether there exists a $r \in \Re$ for which P^r occurs starting at location $[i, j]$. If such an r exists, the first pattern row starts at location $[i, j]$ and repeats $\|r\|$ times. Since we do not know the scale r, we work backwards. If we know the number of repetitions of the first subrow, we will be able to derive $\|r\|$. In [6] a method was presented that preprocesses an $n \times n$ text matrix in time $O(n^2)$ and subsequently allows answering every subrow repetition query in constant time. A *subrow repetition query* is defined as follows: Given an $n \times n$ matrix T,
Input: Location $[i, j]$ and natural numbers k, ℓ.

Output: Decide whether the substring $T[i,j], T[i,j+1], ..., T[i,j+\ell-1]$ repeats k consecutive times starting at column j in rows $i, i+1, ..., i+k-1$ of T. Formally, decide whether for every natural number $y \in \{0, ..., \ell-1\}$ it is true that $T[i+x, j+y] = T[i, j+y], \quad x = 1, ..., k-1$.

The Scale Range Computation – Overview. Every text subrow gives us *horizontal* scaling information, by analysing how many times each symbol repeats in the subrow, and *vertical* scaling information, by computing how many times the subrow repeats until it changes. Note that both the horizontal and vertical scaling information are exact up until the last run-length symbol and the last unique row. Both may repeat in the text for longer than the scale. However, assuming that there is either a row or a column of run-length at least 3, the "inner part" of the run-length places some limitations on the scale range.
The strategy of our algorithm is as follows. For text location $[i,j]$, we compute the range induced by the horizontal scale of the first row, and update the range by the vertical scale of the first row, then move on to the next distinct row, and continue until all pattern rows are accounted for. This means we handle $O(m)$ distinct rows per text location $[i,j]$. The horizontal scale calculations are a constant number of numeric operations per row. The vertical scale computation utilizes the subrow repetition queries. However, not knowing a-priori how many times a row repeats, we do a binary search on the values induced by the horizontal scale range. Claim 2 guarantees that the range of these values can not exceed $m+2$.

The Scale Range Computation – Details Terminology: We denoted the rows of pattern P by $P_1, P_2, ..., P_m$. We can conceptualize the pattern as a string with every row a symbol. Consider the run-length representation of this string, $(P'^{\Sigma}[1])^{c(P)[1]}(P'^{\Sigma}[2])^{c(P)[2]} \cdots (P'^{\Sigma}[k])^{c(P)[k]}$. We call this presentation the *row run-length presentation of the pattern*. We will calculate the subrow length for the rows grouped according to the row run-length representation.
Notation: Denote by $T_{i,j}$ the subrow $T[i,j], T[i,j+1], ..., T[i,n]$.
The details of the scale range computation follow:
For every location $[i_0, j_0]$ in the text calculate the scale range in the following manner:
Initialization:

Set the text row variable tr to i_0
Initialize the pattern row pr to 1.
Initialize the pattern run-length row rr to 1.
Initialize the scale range to $[1, \frac{n}{m})$.

Assume that the scale range $[a,b)$ has been computed so far, and the algorithm is now at text row tr, pattern row pr, and pattern run-length row rr.
Update Scale Range:
Distinguish between three cases:
1. The run-length of pattern row P_{pr} is 1.
2. The run-length of pattern row P_{pr} is 2.
3. The run-length of pattern row P_{pr} is 3 or more.

Each of the three cases is handled somewhat differently: At any stage of the algorithm if an intersection is empty, the algorithm is halted for location $[i_0, j_0]$ – no scaled occurrence. We omit the details of the simple cases 1 and 2 for space reasons and present the interesting case, case 3.

Case 3:

Let the new values of $[a, b)$ be the intersections of the current $[a, b)$ and $[L^{-1}(s, s'), R^{-1}(s, s'))$ where $s = \mathsf{prefixsum}(P_{pr}, |c(P_{pr})| - 1), s' = \mathsf{prefixsum}(T_{tr,j_0}, |c(P_{pr})| - 1)$

Using a "binary search on $[a, b)$" (see Section 5.1) determine the number k of row repetitions.

$tr \leftarrow tr + k$; $pr \leftarrow pr + c(P)[rr]$; $rr \leftarrow rr + 1$.

Check if text subrow is equal to pattern row (see Section 5.2).

end Case 3

We do not want to explicitly check that all symbols in the text match the pattern, since that could take time $O(m)$. So we just make sure the text row repeats a sufficient amount of times. In Section 5.2 we show how to compare the text and pattern rows quickly.

Implementation Details: The prefixsum computations and several computations of the L^{-1} and R^{-1} functions are performed on a run-length compression of the text and pattern, whereas the subrow repetition queries are performed on the uncompressed text and pattern. An initial preprocessing stage will create compressed text and pattern, with double pointers of locations in the compressed and uncompressed text and pattern. All such preprocessing can be done in linear time and space using standard techniques.

Time: The running time for every text location is $O(mt)$, where t is the time it takes to do the binary search. We will see in the next section that the binary search is done in time $O(\log m)$. Thus the total search time is $O(n^2 m \log m)$.

Binary Search on $[a, b)$. $[a, b)$ is an interval of real numbers, thus it is impossible to do "binary search" on it. When we say "binary search" on $[a, b)$ we mean the discrete number of rows that P_{pr} repeats scaled to r for $r \in [a, b)$. This number is at most $\lfloor \frac{n}{m} \rfloor$, and we will show that we can actually limit it to $O(m)$. In any event, we have a finite set of natural numbers to search.

The values we need to search are when a block of k equal subrows of length ℓ occur starting in text location $[tr, j_0]$. It is easy to see that D is non-decreasing in both variables. Therefore, consider a (k, ℓ) block to be *valid* if the k subrows of length ℓ are all equal. The monotonicity of k in ℓ guarantees that if a (k, ℓ) block is not valid, no greater block with a greater ℓ (or k) is valid. If a (k, ℓ) block is valid, all smaller blocks are valid. Thus, it is possible to do a binary search on k or ℓ to find the largest k that gives a valid block.

The only information still needed is how to compute the k and ℓ from interval $[a, b)$. Note that the D function computes from a number of repetitions in P and a given real scale r, the number of times those repetitions will scale to r. It seems like the function we need, and it actually does the job for computing the ℓ. However, we have a problem with the k. The definition of D assumes that the repetitions start at the *beginning* of the pattern on the pixel array, and the rounding is done at the end. Now, however, we find ourselves in the middle of

the pattern. This means that the pattern rows we are interested in may start in the middle of a pixel in the text pixel array. The D function would assume they start at the beginning of a pixel in the text pixel array and provide an answer that may be incorrect by 1. An example can be seen in Figure 4.

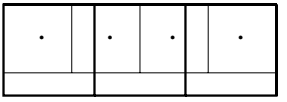

Fig. 4. Assume the pattern's first row is the three symbols *abc*. Consider $P^{1\frac{1}{3}}$. $D(1, 1\frac{1}{3}) = 1$. Yet, the second symbol extends over **two** pixel centers.

Definition 6. *Let k be a discrete length of a sub-pattern in any dimension, starting from location i in the pattern. Let $r \in \Re$ be a scale, and let N be the natural numbers. We define the function $D' : N \times N \times \Re \to N$ as follows: $D'(i, k, r) = D(i + k - 1, r) - D(i - 1, r)$. Since D' is a straightforward generalization of D ($D(k, r) = D'(1, k, r)$), and to avoid unnecessary notation, we will refer to both functions as D.*

It is easy to see that $D(i, k, r)$ is indeed the number of discrete rows or columns that k in position i scales to by r.

Binary Search on $[a, b)$.

Perform a binary search on the values $\ell \in \{D(m, a), ..., \lfloor\lfloor mb \rfloor\rfloor\}$.

For each value ℓ compute the two values k_1, k_2 for which to perform subrow repetition queries starting at location $[tr, j_0]$, with k_1 or k_2 repetitions of length ℓ as follows:

Let $x = L^{-1}(m, \ell)$, $\quad y = R^{-1}(m, \ell)$.

$k_1 = D(pr, c(P)[rr], x)$ and $k_2 = D(pr, c(P)[rr], y)$.

If both subrow repetition queries are positive, this is a valid case. The ℓ value should be increased.

If both subrow repetition queries are negative, this is an invalid case. The ℓ value should be decreased.

If the k_1 subrow repetition query is positive and k_2 is negative the search is done with $k = k_1$.

Correctness: It follows from claims 5.1 and 5.1 that the binary search indeed finds the correct number of repetitions. Their proofs are technical and are omitted for lack of space.

Claim. The binary search algorithm considers all possibilities of k and ℓ.

Claim. Let $r \in \Re$ and assume there is an occurrence of P^r starting at location $[i, j]$ of T. Then the $c(P)[rr]$ rows of the pattern starting at pattern row pr are scaled to the value k discovered by our algorithm.

Time: A subrow repetition query takes constant time. The number of queries we perform is logarithmic in the range of the binary search. By Claim 2, if we know for any number of rows and columns in the pattern, to exactly how many rows or columns they scale, the range is then $O(m)$. Note that in all cases except for Case 1, Claim 2 holds. Even in Case 1, the claim does not hold only for cases where the first pattern rows are all of run-length 1. This is a very trivial pattern and all its scaled occurrences can be easily detected by other easy means. Every other case has at least a row or a column with run-length greater than 1. Without loss of generality we may assume that there is a row r_0 of run-length greater than 1 (otherwise we rotate the text and pattern by 90^o). We consider the pattern P' consisting of rows $P_{r_0}, P_{r_0+1}, ..., P_m$. We compute the ranges of all possible scales of P' for every text location, as described above, and then eliminate all locations where rows $P_1, ..., P_{r_0-1}$ do not scale appropriately. By this scheme there is never a binary search on range greater than $m + 2$. Thus the total time for the binary search is $O(\log m)$.

5.2 Efficient Subrow Comparison

At this stage we know, for every text location, the number of repetitions of every row. But we do not know if the repeating text subrow is, indeed, the appropriate scaled pattern row, for any pattern row whose run-length exceeds two. Checking every row element by element would add a factor of m to our complexity, which we want to avoid. It would be helpful if we could compare entire subrows in constant time. We are interested in a query of the form below.

Given a string S of length n over *ordered* alphabet Σ, a *substring comparison query* is defined as follows.

Input: Locations $i, j, \quad 1 \leq i, j, \leq n$ and length ℓ.

Output: Decide whether the substring $S[i], S[i+1], ..., S[i+\ell-1]$ is lexicographically larger, smaller or equal to substring $S[j], S[j+1], ..., S[j+\ell-1]$.

In [13] a method was presented that preprocesses a string of length n in time $O(n \log \sigma)$, where $\sigma = \min\{m, |\Sigma|\}$, and subsequently allows answering *longest common prefix* queries in constant time. A *Longest Common Prefix* query is defined as follows:

Input: Locations $i, j, \quad 1 \leq i, j, \leq n$.

Output: The length of the longest common prefix of the substrings $S[i], S[i+1], ..., S[n]$ and $S[j], S[j+1], ..., S[n]$, i.e. the number m for which $S[i+\ell] = S[j+\ell], \quad \ell = 0, ..., m-1$ and $S[i+m] \neq S[j+m]$ or one of $i+m, j+m$ is greater than n.

Using the Landau and Vishkin method, we can also answer the substring comparison query in constant time, following a $O(n \log \sigma)$-time preprocessing, in the following way.

Given locations $i, j, \quad 1 \leq i, j, \leq n$ and length ℓ, find the length k of the longest common prefix of $S[i], S[i+1], ..., S[n]$ and $S[j], S[j+1], ..., S[n]$. If $k \geq \ell$ then the two substrings $S[i], S[i+1], ..., S[i+\ell-1]$ and $S[j], S[j+1], ..., S[j+\ell-1]$ are equal. Otherwise, compare $S[i+k]$ and $S[j+k]$. If $S[i+k] > S[j+k]$ then the substring $S[i], S[i+1], ..., S[i+\ell-1]$ is lexicographically larger than

$S[j], S[j+1], ..., S[j+\ell-1]$, and if $S[i+k] < S[j+k]$ then the substring $S[i], S[i+1], ..., S[i+\ell-1]$ is lexicographically smaller than $S[j], S[j+1], ..., S[j+\ell-1]$. Our algorithm will do a binary search on a precomputed run-length compress dictionary of all scaled possibilities on the row. While the symbol part of the run-length can indeed be checked in such a dictionary, the numerical part is more problematic. The difficulty is that our text subrow is not wholly given in run-length compressed form. The first and last symbols of the run-length compressed row may occur in the text more times than in the scaled pattern. The details of verifying that the numerical part of the run-length of the pattern row matches the run-length of the text subrow are left for the journal version.

References

1. A. Amir, A. Butman, and M. Lewenstein. Real scaled matching. *Information Processing Letters*, 70 (4) : 185–190, 1999.
2. A. Amir and G. Calinescu. Alphabet independent and dictionary scaled matching. *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM 96)*, pages 320–334, 1996.
3. A. Amir and M. Farach. Two dimensional dictionary matching. *Information Processing Letters*, 44: 233–239, 1992.
4. A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.
5. A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991.
6. A. Amir, G.M. Landau, and U. Vishkin. Efficient Pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.
7. R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.
8. K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. 9th Annual Symposium on Combinatorial Pattern Matching (CPM 98)*, pages 118–125. Springer, LNCS 1448, 1998.
9. R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for Square matrices: Algorithms and applications. *Information and Computation*, 130(2):151–182, 1996.
10. R.M. Idury and A.A Schäffer. Multiple matching of rectangular Patterns. Proc. *25th ACM STOC*, pages 81–89, 1993.
11. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.
12. K. Krithivansan and R. Sitalakshmi. Efficient two dimensional Pattern matching in the presence of errors. *Information Sciences*, 13:169–184, 1987.
13. G. M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.
14. G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12 (3/4):375–408, 1994.
15. A. Pentland. Invited talk. NSF Institutional Infrastructure Workshop, 1992.