FA{S,T}TER SCALED MATCHING

Amihood Amir* Gruia Calinescu†
Georgia Tech Georgia Tech

GIT–CC–93/40

July 1993

Abstract

The rapidly growing need for analysis of digitized images in multimedia systems has lead to a variety of interesting problems in multidimensional pattern matching. One of the problems is that of **scaled matching**, finding all appearances of a pattern in a text in all discrete sizes. Another important problem is **dictionary matching**, quick search through a dictionary of preprocessed patterns in order to find all dictionary patterns that appear in the input text.

In this paper we provide a very simple algorithm for two dimensional scaled matching. Our algorithm is the first linear-time alphabet-independent scaled matching algorithm. Its running time is $O(|T|)$, where $|T|$ is the text size, and is independent of $|\Sigma|$, the size of the alphabet.

Our technique generalizes to produce the first known algorithm for scaled dictionary matching. We can find all appearances of all dictionary patterns that appear in the input text in any discrete scale. The time bounds of our algorithm are equal to the best known exact (no scaling) two dimensional dictionary matching algorithms.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; (404) 853-0083; amir@cc.gatech.edu; Partially supported by NSF grant IRI-90-13055 and CCR-92-23699.

†College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; (404) 853-9389; gruia@cc.gatech.edu.

1 Introduction

Recently the world has been witnessing a strong convergence of market forces such as the computer industry, television, photo/films, library, and telephone. All these diverse industries need to digitize and analyse digitized images [14]. A lacking crucial element in this multimedia effort is the equivalent of "text string searching" in an image database [17].

Much theoretical progress has been made in recent years in the area of multidimensional pattern matching. Exact two dimensional matching can now be done in in alphabet-independent linear time [1, 16, 3, 2]. Approximate two dimensional matching, where one seeks all appearances of a pattern P in text T with some possible errors, can be doneefficiently under various definitions of error [4, 8]. Scaled matching, where all appearances of a pattern P *scaled* to any discrete size are sought in text T, can also be solved in linear time for fixed finite alphabets [9].

Much progress has been made with *dictionary matching* as well. In the traditional pattern matching model a single pattern is sought in a single text. Dictionary matching allows preprocessing of a (possibly vast) dictionary of patterns. Subsequently, appearances of dictionary patterns in various input texts are to be found quickly. Various algorithms for two dimensional matching (static, dynamic, square patterns, rectangular patterns) were developed [5, 6, 11, 13]. The two remaining fundamental problems in dictionary matching are 1) Scaled dictionary matching and 2) Approximate dictionary matching.

This paper addresses the scaled two dimensional dictionary matching problem. The techinques of [9] for scaled matching can not be generalized to multidimensions. The main idea was analysing the text with the aid of *power columns*. Those are the text columns appearing $m-1$ columns apart, where P is an $m \times m$ pattern. This dependence on the pattern size make the power columns useless where a dictionary of different sized patterns is involved.

In this paper we present a **novel idea** for solving the scaled dictionary matching problem for a single pattern. This idea is significantly **simpler** than the algorithm of [9] and has an additional advantage of being **alphabet-independent**. The previous scaled matching algorithm had a worst case running time of $O(|T| \log |\Sigma|)$, where $|T|$ and $|\Sigma|$ are the respective sizes of the text and the alphabet. Our algorithm runs in time $O(|T|)$ for every alphabet.

We also show that this idea can be further developed to dictionary matching. We present what is, to our knowledge, **the first algorithm** for **scaled two dimensional dictionary matching**. Our algorithm solves the scaled dictionary matching problem for a static dictionary of square pattern matrices in time similar to that of the non scaled algorithm [5]. Our times are: $O(|D| \log k)$ preprocessing, where $|D|$ is the total dictionary size and k is the number of patterns in the dictionary, and $O(|T| \log k)$ text scanning time, for input text T. This is identical to the time at [5], the best **non-scaled** matching algorithm for a static dictionary of square patterns. It is somewhat surprising that scaling **does not add** to the complexity of single matching nor dictionary matching.

This paper is organized as follows. In section 2, we give the problem definition and some

preliminaries. In section 3, we give the alphabet independent single pattern two dimensional scaled matching algorithm. In section 4 we give the scaled dictionary matching algorithm. We conclude with open problems and future research.

2 Problem Definition

Before we define the scaled matching problem, we introduced the following notation. The string $aa...a$ where the symbol a is repeated k times (to be denoted a^k), is referred to as *scaling of the singleton string a by multiplicative factor k*, or simply as *a scaled to k*. Similarly, consider a string $A = a_1 \cdots a_l$. *A scaled to k* (A^k) is the string $a_1^k, ..., a_l^k$.

The **scaled string matching problem** is defined as follows:

Input: Pattern $P = p_1 \cdots p_m$ and text $T = t_1 \cdots t_n$ where $n > m$.

Output: All positions in T where an occurrence of P scaled to k starts, for any $k = 1, ..., \lfloor \frac{n}{m} \rfloor$.

Let $P[m \times m]$ be a two-dimensional matrix over alphabet Σ (not necessarily bounded). Then P *scaled to* s (P^s) is the $sm \times sm$ matrix where every symbol $P[i,j]$ of P is replaced by a $s \times s$ matrix whose elements all equal the symbol in $P[i,j]$. More precisely,

$$P^s[i,j] = P[\lceil \frac{i}{s} \rceil, \lceil \frac{j}{s} \rceil]$$

Follwing [9] we define the problem of **two-dimensional pattern matching with scaling** as follows:

Input: Pattern matrix $P[i,j] \quad i = 1, ...m; j = 1, ..., m$ and Text matrix $T[i,j] \quad i = 1, ..., n; j = 1, ..., n$ where $n > m$.

Output: all locations in T where an occurrence of P scaled to s (an *s-occurrence*) starts, for any $s = 1, ..., \lfloor \frac{n}{m} \rfloor$.

We defined the scaled matching problem on square texts and patterns for the sake of simplicity only. Our alphabet independent result is also true for rectangular texts and patterns.

In [9], this problem was solved in time $O(n^2)$. However, the alphabet was finite. Over an infinite alphabet, the Amir-Landau-Vishkin algorithm will achieve time $O(n^2 \log m)$. Added deficiencies of that algorithm are that it is extremely complicated, and that the size of the pattern m plays a crucial role in the text processing. Our algorithm is simple and straightforward, and its time complexity is $O(n^2)$ for every alphabet.

Traditional Pattern Matching has dealt with the problem of finding all occurrences of a single pattern in a text (under some definition of the word "occurrence"). While the case of a pattern/text pair is of fundamental importance, the single pattern model is not always appropriate. One would often like to find all occurrences of a *set* of patterns in a text. We call such a set of patterns a *dictionary*. In addition to its theoretical importance, dictionary matching has many practical applications. For example, in computer vision, one is often

interested in matching a template to a picture. In practice, one needs to match an enormous set of templates against each picture. Clearly one would like an algorithm which is minimally dependent on the size of the database of templates.

The **static two dimensional dictinary matching problem** is defined as follows. Given a A set D (the *dictionary*) of patterns $P_1[m_1 \times m_1], P_2[m_2 \times m_2], \ldots, P_k[m_k \times m_k]$ of total length $d = \sum_{i=1}^{k} m_i^2$ all over alphabet Σ. Preprocess the set in a way that enables solution to the following problem:

Input: Text $T[n \times n]$ over alphabet Σ.

Output: All ordered pairs $([x, y], j)$ such that pattern P_j matches the subsquare of text beginning at location $T[x, y]$.

Amir and Farach [5] provided an algorithm for the static two dimensional dictionary matching problem where the dictionary preprocessing was done in time $O(d \log k)$ and subsequent text scans took time $O(n^2 \log k)$. In [6], the dynamic two dimensional dictionary matching problem was considered. In the dynamic version, patterns may be inserted to and deleted from the dictionary. An algorithm was provided where a dictionary update takes time $O(|P| \log d)$ (P is the pattern inserted or deleted) and the text scanning takes time $O(n^2 \log d)$. Giancarlo [11] also obtained an algorithm for the dynamic two dimensional dictionary matching problem, but his algorithm is less efficient. In [7] simple randomized algorithm for this problem were presented.

All above algorithms assume that the patterns are squares (although the text need not be). Idury and Schäffer [13] showed an algorithm for dynamic two dimensional dictionary matching of general rectangular patterns.

In this paper we show that our new scaled matching algorithm can be adapted to solve the static two dimensional dictionary matching problem for square patterns in the **same time complexity** as that of the **exact matching** case. The same technique can be used to generalize the algorithms for other versions of the dictionary matching problem to the scaled case.

3 Alphabet Independent Scaled Matching

3.1 Algorithm's Idea and Data Structures

Our algorithm is based on the alphabet independent exact two dimensional matching algorithm of Amir, Benson and Farach [1]. That algorithm has two stages, the *candidate consistency* stage and the *candidate verification* stage.

In the candidate consistency stage, we check for every pair of text location if both locations can be the start of a pattern appearance. A preprocessed witness table allows constant time elimination of one of the two candidate sources, if they do not agree in every location of the overlap. In [3, 16] there are two different algorithms that construct the witness table in time $O(m^2)$. The candidate consistency stage takes time $O(n^2)$ [1].

In the candidate verification stage, a *wave* is employed to verify which of the non-conflicting sources are indeed starts of pattern appearances. This stage is also done in time $O(n^2)$.

Main Idea of Algorithm:

For every scale $s = 1, \ldots, \lfloor \frac{n}{m} \rfloor$ we divide the text into a $\lfloor \frac{n}{s} \rfloor \times \lfloor \frac{n}{s} \rfloor$ grid of $s \times s$ squares. Call such a square an *s-block*. We do a constant amount of work per s-block. The total amount of time is then

$$\sum_{s=1}^{\lfloor \frac{n}{m} \rfloor} \frac{n^2}{s^2} = n^2.$$

It should be noted that the Amir-Landau-Vishkin scaled matching algorithm also tries to employ this strategy. The problem is that the division into s-blocks is very complicated there. We simply divide into the natural grid, and adjust the exact location at the end.

Our algorithm assumes the pattern is *non-trivial*. A *trivial pattern* is a pattern where all rows are equal or all columns are equal. It is easy to find all scaled appearances of a trivial pattern. It is clear that in a non-trivial pattern there exist at least two consecutive rows that are different and one of which has at least two different symbols.

Definition: Position $[i, j]$ of P is a *pivot* if the following conditions hold:

1. $i > 1$, $m > j > 1$.
2. The strings $P[i-1; j-1, \ldots, m]$ and $P[i; j-1, \ldots, m]$ are different.
3. one of the strings $P[i-1; j-1, \ldots, m]$ and $P[i; j-1, \ldots, m]$ contains two different symbols.

The main property exploited by our algorithm is the following lemma.

Lemma 1: Consider an s-block starting at position $T[p, q]$ (i.e. $T[p, \ldots, p+s-1; q, \ldots, q+s-1]$). If the pivot of a pattern's s-occurrence starts within this s-block, then there is at most one position in the s-block where that pivot may starts. That position is independent of the pattern.

Proof: Five cases should be considered:

1. All characters of the s-block are the same.

   Let l be the smallest column such that $l > q$ and such that $T[i, l-1] \neq T[i, l]$, for i ranging from $q-1$ to $q+s-1$, and let i_0 be the row where l appears. In words, l is the closest change in symbol in the rows of the s-block, and the row preceeding it. If the pivot is in the s-block, we are guaranteed such a change in the s-block or the block immediately preceeding it.

The pivot must start in a column of the s-block that is sx columns away from l. Thus the pivot, if it exists, must start in column $q + ((l - q)(\mathrm{mod} s))$. The pivot's row is $\max(p, i_0)$.

(It is possible that no pivot exists in this s-block, but our concern is that we identify no more than one possibility **if** a pivot exists.)

2. The s-block has two distinct symbols a and b, with all elements in the first i rows equal to a and all elements in the last $s - i$ rows equal to b.

   It is clear that if the pivot starts in the s-block, it has to start in row $p + i$, we only need to establish the column. The smallest l such that $l > q$ and such that either $T[q + i - 1, l - 1] \neq T[q + i - 1, l]$ or $T[q + i, l - 1] \neq T[q + i, l]$ determines the column $q + ((l - q)(\mathrm{mod} s))$ as in the previous case.

3. The s-block has two distinct symbols a and b, with all elements in the first i columns equal to a and all elements in the last $s - i$ columns equal to b.

   Clearly, the pivot starts in column $p + i$. We need to establish the row. By definition, the pattern row preceeding the pivot is different from the pattern row which the pivot is on. Let i_0 be the row in the s-block for which there exists the minimum l such that $T[i_0, j] = T[i_0 - 1, j], \ \ \forall j = 1, \ldots, l$. i_0 is the row of the pivot.

4. There exist four alphabet symbols a, b, c, d, at least three of which are distinct, and there exists location $[i, j]$ in the s-block, such that

$$T[x, y] = \begin{cases} a, & \text{if } p \leq x < i; \ \ q \leq y < j; \\ b, & \text{if } p \leq x < i; \ \ j \leq y < q + s - 1; \\ c, & \text{if } i \leq x \leq p + s - 1; \ \ q \leq y < j; \\ d, & \text{if } i \leq x \leq p + s - 1; \ \ j \leq y < q + s - 1; \end{cases}$$

   In this case if there is a pivot it must start in location $T[i, j]$.

5. All other cases - no pivot. □

We need to decide the start of the pivot in constant time for each s-block. The following data structures allow us to do this.

Definition: Let $[i, j]$ be a position on $n \times n$ text T.

1. Let $B_r[i, j]$ be the largest integer l for which the two strings $T[i; j, j + 1, ..., n]$ and $T[i + 1; j, j + 1, ..., n]$ are equal. In words, $B_r[i, j]$ gives the longest common prefix of rows i and $i + 1$ starting at column j.

2. Let $C_r[i, j]$ be the largest integer l for which $T[i, j] = T[i, j + 1] \cdots = T[i, j + l - 1]$. In words, $C_r[i, j]$ gives the number of times that the symbol in $T[i, j]$ appears consecutively starting at position $[i, j]$.

It is easy to construct B_r and C_r in time $O(n^2)$. We preprocess the columns of B_r and C_r for *range minimum queries.*

Let $L = [l_1, ..., l_n]$ be an array of n numbers. A *Range Minimum* query is of the form:

> Given a range of indices $[i, ..., j]$, where $1 \leq i \leq j \leq n$, return an index $k \quad i \leq k \leq j$ such that $l_k = \min\{l_i, ..., l_j\}$.

In [10] it was shown that a list of length n can be preprocessed in time $O(n)$ such that subsequent range minimum queries can be answered in constant time. It is not hard to see that a constant number of range minimum queries on B_r and C_r can handle the cases in the lemma.

3.2 The Algorithm

The algorithm is a modification of the alphabet independent two dimensional matching algorithm of [1]. The modification allows running the algorithm on an *s-coarse text*. An s-coarse text is an $\lfloor \frac{n}{s} \rfloor \times \lfloor \frac{n}{s} \rfloor$ grid of s-blocks. We will discover all s-occurrences of the pattern by a constant amount of work per s-block in the $\lfloor \frac{n}{s} \rfloor \times \lfloor \frac{n}{s} \rfloor$ grid. The algorithm below is to be run separately for all $s = 1, \ldots, \lfloor \frac{n}{m} \rfloor$.

Algorithm for Exact Matching of P^s in an s-coarse Text

Step 1 Calculate the unique pivot position for every s-block.

Step 2 Calculate the unique possible starting position of P^s for every s-block. (Done by copying the pivot position of that pattern appearance.)

Step 3 Candidate Consistency Stage.

Step 4 Candidate Verification Stage.

end Algorithm

As was shown in section 3.1 Steps 1 and 2 can be done in time $O(\lfloor \frac{n}{s} \rfloor^2)$. We will now show a $O(\lfloor \frac{n}{s} \rfloor^2)$ implementation of the Consistency and Verification Stages.

Consistency Stage

In the consistency stage, all pairs of possible pattern starting positions (*sources*) are compared using the witness table. Either the two sources are compatible, i.e. agree on the overlap, or one of the sources is eliminated.

When comparing two s-blocks, each one has a unique possible starting position. We call this position its *offset*. If the witness table for the two sources identifies a conflict, it is sufficient to check the bottom rightmost element of the s-block where the conflict lies. Regardless of the offset of the sources, the last element lies in the scaled version of the conflicting position.

The situation is more complicated where the witness table indicates consistency. Here different offsets may still cause a conflict. We handle the three possible cases:

1. Equal offsets. There is no conflict between the sources.

2. The two offsets differ in one index. Without loss of generality we may assume that index is the column index. The situation is: Source c_1 at location $[x, y_1]$, and source c_2 at location $[x, y_2]$. If P^s starting at c_1 overlaps P^s starting at c_2 then there are two possible cases:
   (a) The c_1 appearance of P^s agrees with the entire area where it overlaps the c_2 occurrence. In this case, both sources are compatible.
   (b) There is a location in the overlap where the c_1 and c_2 occurrences conflict. This last situation can only happen if somewhere in the overlap there is an s-scaled appearance of a symbol a followed by an s-scaled appearance of a different symbol b. This location can be found in constant time by a range minimum query of a B_r array of the pattern. Checking this text location will eliminate one of the two sources.
3. The two offsets differ in their two indices. The situation here is very similar to the previous one. Either the entire area of the overlap consists of a single symbol, in which case there is no conflict, or we can decide in constant time where two different symbols should occur and eliminate one of the two options.

Remark: At the conclusion of the Consistency Stage we have in each s-block at most one possible source. Moreover, we also know that if there are pattern appearances at any of these sources, they can not conflict on the overlap.

Important Observations:

1. Let c_1, c_2, c_3 be three nested nonconflicting sources resulting from the Consistency phase of scale s, with c_3 within c_2 and c_2 within c_1. If c_3 is verified to be a pattern occurrence (in time proportional to the number of s-blocks), then we know that the area where c_2 and c_1 overlap c_3, need not be checked. However, the s-blocks on the top edge and left edge of c_3 were checked only relative to c_3, so if c_1 and c_2 have different offsets, these s-blocks need to be checked again. The important thing to note is that each s-block need only worry about the source immediately above it, the source immediately to the left of it and the source diagonally to its top-left. Any other sources will agree with those three on the overlap. Thus, verifying at most three symbols for every s-block is sufficient.
2. Let c_2 be a source nested within c_1. If we check some s-block in the overlap while trying to verify c_2, and find a mismatch, this generally means a mismatch for c_1, since they agree on the overlap. The possible exceptions are if this s-block is on an edge row or column of c_1, and the mismatch occured at the offset **outside** the boundary of c_1. This will necessitate separate handling of the first and last row and column of a pattern occurrence.

Verification Stage

The verification is similar to the exact matching algorithm presented in [1]. The verification algorithm for scale s follows.

4.1 Run a horizontal wave on the s-scale grid recording for each s-block, its position relative to the nearest leftmost overlapping source, and the offset of that source. In an s-block with a source, record its position and offset **and** the position and offset of its nearest leftmost overlapping source.

4.2 Run a similar vertical wave recording the position relative to the source of the s-block above as well as the offset. In case of a source change, record both.

4.3 For every s-block $[x, y]$, record the relative position and offset of s-block $[x-1, y-1]$.

4.4 For every s-block, verify that each of the $s \times s$ blocks starting at each offset is an s-occurrence of the appropriate pattern symbol. (Implemented in constant time by a range minimum query to T_r).

4.5 Do a back wave to propagate the mismatches and cancel the appropriate sources. The back wave should advance only $m-1$ s-blocks, since the edge row and column need to be checked separately.

4.6 Find all scaled s appearances of the first and last pattern row and column (string scaled matching, can be done in linear time by [9] and appropriate range minima queries). Discard all remaining sources where the edges do not match.

Verification Time: $O(\lfloor \frac{n}{s} \rfloor^2)$.

4 Dictionary Scaled Matching

We are now ready to solve the problem of scaled dictionary matching for a static dictionary of squares. This is the scaled version of the problem solved by Amir and Farach in [5]. In that paper an algorithm (henceforth refered to as AF) was presented that preprocessed the dictionary in time $O(d \log k)$, where d is the sum of the sizes of all dictionary patterns and k is the number of patterns. The AF algorithm then finds all occurrences of all dictionary patterns that appear in text T in time $O(|T| \log k)$.

The main idea for scaling the AF algorithm is similar to the single pattern scaled matching we presented before. Simply divide the text into a grid of s-blocks for each scale $s = 1, \ldots, \lfloor \frac{n}{m} \rfloor$. Subsequently, we run an adapted version of AF for each scale s, where the time complexity per s-block is constant.

The details of the AF adaptations are similar to those presented above for the single pattern scaled matching algorithm, and are left to the journal version of the paper. However, we need to answer a major difficulty.

The "pivotal" reason our single pattern algorithm worked is that there could be at most one position in an s-block where the pivot can start. While that unique position was found

independent of the pattern (lemma 1), it did assume knowledge of the pivot's location in the pattern. When dealing with a dictionary, the pivot of different patterns may occur in different locations, and then lemma 1 will no longer guarantee at most one starting location per s-block.

We solve this problem by guaranteeing that the pivot is in the same location in all dictionary patterns. We will need a slight modification of the pivot definition, and an efficient method of converting the patterns to and from a set of patterns with the same pivot location.

Assumption: We again assume that the patterns are *non-trivial*. If trivial patterns appear in the dictionary they are taken out during preprocessing, and handled separately.

Definitions:

1. A *diagonal pivot* of $m \times m$ matrix P is the largest i such that Rows $P[1;1,\ldots,m],\ldots,P[i-1;1,\ldots m]$ are either all equal to each other or each consists of one repeating symbol, and columns $P[1,\ldots,m;1],\ldots,P[1,\ldots m;i-1]$ are either all equal to each other or each consists of one repeating symbol.

2. Let P be an $m \times m$ matrix, and let i be the diagonal pivot of P. The *reduced matrix* $P^{[i]}$ of P is the square matrix $P[i-1,\ldots,m;i-1,\ldots,m]$.

Observations: Every non-trivial square matrix has a diagonal pivot. The pivot of every reduced pattern $P^{[i]}$ is in location $P^{[i]}[2,2]$.

Lemma 2: Consider an s-block starting at position $T[p,q]$ (i.e. $T[p,\ldots,p+s-1;q,\ldots,q+s-1]$). If the diagonal pivot of a pattern's s-occurrence starts within this s-block, then there is at most one position in the s-block where that pivot may starts. That position is independent of the pattern.

Proof: Similar to the proof of lemma 1.

Outline of Scaled Dictionary Matching Algorithm

Step 1 **Preprocessing:** Construct dictionary D^R whose elements are the reduced matrices of the patterns in dictionary D. (The pivots of all patterns in dictionary D^R are in the same location, $[2,2]$.) Preprocess this dictionary in a manner similar to the dictionary preprocessing of AF.

Step 2 **Text Scanning:**

2.1 For $s = 1,\ldots,\lfloor\frac{n}{m}\rfloor$ construct the grid of s-blocks, and run a modification of algorithm AF.

2.2 For every reduced pattern found, efficiently verify and output all original dictionary patterns that start in a text position corresponding to the location of the reduced part.

Implementation: In addition to preprocessing for range minimum queries over the B and C arrays, we also need suffix trees [15, 18] of the dictionary patterns and text, preprocessed for constant time lowest common ancestor queries [12]. The suffix tree construction adds the $\log k$ multiplicative factor to our complexity.

Using suffix trees of the run-length representation of text and pattern, as in [9], we can answer the following queries in constant time:

Input: Text location $T[x, y]$, and pattern subrow (or subcolumn) $R = P[z; w, \ldots, w + j]$.

Decide: If there is an s-occurrence of R starting at $T[x, y]$.

Time: It is not difficult to see that Step 1 can be done in time $O(d \log k)$. A modification of AF on lines similar to our algorithm in section 3 enables accomplishing Step 2.1 in time $O(|T| \log k)$. We will show below that Step 2.2 can be implemented in time $O(|T| \log k)$ and that will conclude our algorithm.

Lemma 3: Let $P^{[i]}$ be the reduced pattern of P. If there is an s-occurrence of $P^{[i]}$ in location $[x, y]$ of T, it can be verified in constant time whether there is an s-occurrence of P in location $[x - s(i-1)\ ,\ y - s(i-1)]$.

Proof: We distinguish between three types of diagonal pivot:

1. The first $i - 1$ rows of the pattern are equal and the first $i - 1$ columns of the pattern are equal. This case can be easily verified by two range minima queries on B arrays of the text (to the right, and to the bottom).

2. Each of the first $i - 1$ rows is a single repeating symbol but there are two consecutive unequal rows. (In this case, the first $i - 1$ columns are all equal.) This situation can be verified by a range minimum query on the B array to the bottom (for the columns), and a range minimum query on the B array to the bottom (for the rows).

3. Each of the first $i-1$ columns is a single repeating symbol but there are two consecutive unequal columns. (In this case, the first $i - 1$ rows are all equal.) This situation is analagous the the previous one. □

The above lemma provides a linear time solution for Step 2.1, in case every reduced pattern is obtained from a single dictionary pattern. In the general case, it is possible that several dictionary patterns have the same reduced pattern. In this case, we have no *apriori* knowledge of i, and the situation becomes more complicated.

Lemma 4: Let P' be the reduced pattern of patterns $P_1, \ldots, P_j$. If there is an s-occurence of P' in location $[x, y]$, then we can find the largest P_i whose s-occurrence appears in the appropriate location of T in time $O(\log j)$.

Proof: We can divide the patterns $\{P_1, \ldots, P_j\}$ into three sets corresponding to the three types of diagonal pivot defined in lemma 3. We will present the first type. Similar treatment can be given to the other types.

1. This case can happen only if there is a square diagonal (left and above) to the start of the s-occurrence of P', consisting of a single repeating letter. The maximum such square can be found by $O(\log m)$ range minima queries, where m is the size of the maximum $P_i \in \{P_1, \ldots, P_j\}$. This will essentially be a binary search. A series of $O(\log m)$ more range queries will pick the maximal number of equal rows and columns. A final binary search on the patterns of $\{P_1, \ldots, P_j\}$ will enable choosing the largest matching pattern in time $O(\log j) \leq O(\log k)$. □

5 Future Work

The pivot idea seems to be extendable to scale more general dictionary matching algorithms that AF. We think that every known dictionary matching algorithm can be scaled without degradation in its time complexity.

A very interesting remaining open problem is efficiently finding all scaled occurrences for **all scales**, not just discrete scales. This is open even for one dimensional non-dictionary string matching.

References

[1] A. Amir, G. Benson, and M. Farach. Alphabet independent two dimensional matching. *To appear, SIAM J. Comp.*, 1992.

[2] A. Amir, G. Benson, and M. Farach. Alphabet independent two dimensional matching. *Proc. 24th ACM Symposium on Theory of Computation*, pages 59–68, 1992.

[3] A. Amir, G. Benson, and M. Farach. The truth, the whole truth, and nothing but the truth: Alphabet independent two dimensional witness table construction. Technical Report GIT-CC-92/52, Georgia Institute of Technology, 1992.

[4] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of non-rectangular figures. *Proc. of 2nd Symposium on Discrete Algorithms, San Francisco, CA*, pages 212–223, Jan 1991.

[5] A. Amir and M. Farach. Two dimensional dictionary matching. *Information Processing Letters*, 44:233–239, 1992.

[6] A. Amir, M. Farach, R.M. Idury, J.A. La Poutré, and A.A Schäffer. Improved dynamic dictionary matching. *Proc. 4th ACM-SIAM SODA*, pages 392–401, 1993.

[7] A. Amir, M. Farach, and Y. Matias. Efficient randomized dictionary matching algorithms. *Proc. 3rd. Combinatorial Pattern Matching Conference*, pages 259–272, 1992. Tucson, Arizona.

[8] A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991.

[9] A. Amir, G.M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.

[10] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. *Proc. 16th ACM Symposium on Theory of Computing*, 67(135-143), 1984.

[11] R. Giancarlo. The suffix of a square matrix, with applications. *Proc. 4th SODA*, pages 402–410, 1993.

[12] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestor. *Computer and System Science*, 13:338–355, 1984.

[13] R.M. Idury and A.A Schäffer. Multiple matching of rectangular patterns. *Proc. 25th ACM STOC*, page to appear, 1993.

[14] R. Jain. Workshop report on visual information systems. Technical Report Technical Report, National Science Foundation, 1992.

[15] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23:262–272, 1976.

[16] K. Park and Z. Galil. Truly alphabet-independent two-dimensional pattern matching. *Proc. 33rd IEEE FOCS*, pages 247–256, 1992.

[17] A. Pentland. Invited talk. NSF Institutional Infrastructure Workshop, 1992.

[18] P. Weiner. Linear pattern matching algorithm. *Proc. 14 IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.