Optimal Two-Dimensional Compressed Matching

Amihood Amir* Gary Benson† Martin Farach‡
Georgia Tech Mt Sinai School of Med Rutgers & DIMACS

Abstract

Recent proliferation of digitized data and the unprecedented growth in the volume of stored and transmitted data motivated the definition of the compressed matching paradigm. This is the problem of efficiently finding a pattern P in a compressed text T without the need to decompress.

We present the first *optimal* two-dimensional compressed matching algorithm. The compression under consideration is the two dimensional run-length compression, used by FAX transmission.

We achieve optimal time by proving new properties of two-dimensional periodicity. This enables performing duels in which no witness is required. At the heart of the dueling idea lies the concept that two overlapping occurrences of a pattern in a text can use the content of a *predetermined text position* or *witness* in the overlap to eliminate one of them. Finding witnesses is a costly operation in a compressed text, thus the importance of witness-free dueling.

Key words: Matching, two-dimensional, witness, periodicity, compression

1 Introduction

Recent developments in multimedia have led to an unprecedented increase in digitally stored information. This increase and the projected growth in the volume of telecommunications have made it critically important to store and transmit files in a **compressed** form. The need to quickly access this data has given rise to a new paradigm in searching, that of *compressed matching*. Previously, the main thrust in the study of data compression has been to achieve compressions that are efficient in packing while also being practical in time and space usage. This is no longer sufficient. In [1, 2], a new goal for compression was introduced. The compression must have the additional property of allowing **pattern matching** in the **compressed data** without the need to decompress.

In traditional pattern matching, all occurrences of pattern P in text T are sought. The pattern and text are explicitely given. In compressed pattern matching the goal is the same, but, the

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; 404-894-3152; `amir@cc.gatech.edu`; Partially supported by NSF grant IRI-90-13055.

†Department of Biomathematical Sciences, Mount Sinai School of Medicine; 212-241-5777; `benson@ecology.biomath.mssm.edu`; Partially supported by NSF grant DMS-90-05833.

‡Department of Computer Science, Rutgers University & DIMACS; 908-932-4580; `farach@cs.rutgers.edu`; Supported by DIMACS under NSF contract STC-88-09648.

text is given in compressed form. Let c be a compression algorithm, and let $c(T)$ be the result of c compressing text T. A compressed matching algorithm is *optimal* if its time complexity is $O(|c(T)|)$.

The problem as defined above is equally applicable to text (one dimensional) and image (two dimensional) data. Recently, it was shown that the LZW compression used by UNIX has an optimal compressed matching algorithm [5]. The LZW compression is an adaptive one dimensional compression scheme.

The compressed matching problem is considered even more crucial in image databases. In fact, the initial definition of the compressed matching paradigm was motivated by the *two dimensional run-length compression* used in FAX transmissions. It is defined as follows:

Let $S = s_1 s_2 \cdots s_n$ be a string over some alphabet Σ. The *run-length compression* 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 the concatenation of k *segments*, 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. The *two-dimensional run-length compression* is the concatenation of the run-length compression of all the matrix rows (or columns).

In [1], a compressed matching algorithm for the two dimensional run-length compression was presented, whose running time is $O(|c(T)| \log |c(T)|)$. Although optimal time was not achieved in [1], a new tool was developed, that of *two dimensional periodicity.* Two dimensional periodicity has since played an important role in two dimensional matching. In [3], it was used to achieve the first linear-time, alphabet-independent, two dimensional text scanning. Later, in [10] it was used for a linear-time witness table construction. In [4] it was used to achieve the first parallel, time and work optimal, CREW algorithm for text scanning. A simpler variant of periodicity was used by [9] to obtain a constant-time CRCW algorithm for text scanning. Recently in [13] an attempt has been made to generalize periodicity analysis to higher dimensions and in [11] two-dimensional matching techniques have been applied to three dimensions.

The two dimensional compressed matching problem is a critical one. The search for its solution has already produced a powerful technique in two dimensional matching. Yet, an optimal compressed matching algorithm had proven elusive. The bottleneck had been the need for many *duels*, each requiring a *witness*. The idea of **dueling** has been useful in string matching since its introduction in 1985 [14]. At its heart lies the concept that two overlapping potential occurrences of a pattern string in a text can use the content of a *predetermined text position* or *witness* in the overlap to eliminate one of them. In uncompressed matching a witness location in the text can be accessed in constant time. This is not true in a compressed text. Thus, the need to reduce the number of duels and to eliminate many candidates **without** a text look-up.

The **main contributions** of this paper are:

- The **first known** optimal algorithm for two dimensional compressed matching. Our algorithm finds all occurrences of pattern P in run-length compressed text in time $O(|c(T)|)$. This optimal result is obtained by using new theorems on two dimensional periodicity, as well as fast look-up via finger trees.

- A new and deeper understanding of two dimensional periodicity. This gives the surprising ability to eliminate some candidates without looking at **any** witnesses.

The remainder of this paper is organized as follows. In section 2, we formally state the problem we solve. In section 3, we give an overview of our algorithm. In section 4 we give a brief description of two-dimensional periodicity, witness tables and the pattern preprocessing step. In section 5, we present new properties of two dimensional periodicity that make our linear time algorithm possible. In sections 6 and 7 we present our text analysis algorithm for two dimensional compressed matching.

2 Problem Description

The *Two-Dimensional Run-Length Compressed Matching Problem* is defined as follows:

Input: Two-dimensional run-length compressed text array T; Pattern array P. The uncompressed size of T is $n \times n$. The compressed size is $|c(T)|$. The size of the pattern is $m \times m$. The elements in the text and pattern are taken from an alphabet set Σ.

Output: All locations in T where P occurs, *i.e.* the set $\{(i,j) : T[i+k, j+l] = P[k,l]$ $k,l = 0 \ldots m-1\}$.

Although stated in terms of square arrays, the problem (and our solution) generalizes to rectangular arrays. We require the following definitions for pattern arrays.

Definition 1 A *trivial pattern* is one in which every element contains the same character. A *seam* is an occurence of two different, adjacent characters in the same row or column. A *stripe* is a row or column without a seam.

In order to make the running time insensitive to the size of the output, we do not allow trivial patterns. If the text and the pattern are both trivial with the same character, then the output size is $O(T)$. Under this condition, for any compression scheme, no algorithm can be efficient. All other patterns contain a seam. Seams are important because they appear explicitly in the run-length compression. Since the number of seams in the text is $O(|c(T)|)$ for run length compression, our algorithm will be unaffected by output size.

In order to deal with stripes, we require both a row compressed form and a column compressed form of the text. We assume that both are available.

3 Algorithm Overview

Below, we outline our algorithm for the two-dimensional run-length compressed matching problem. It runs in time $O(|c(T)| + |P|)$. Note that this algorithm is truly linear in that it runs in the increasingly prominent *alphabet independent model* in which the alphabet is unordered and only equality of characters can be tested. Thus it matches the extremely weak model assumed by e.g.[12].

Definition 2 A *candidate pattern* is a subarray of the text which we suspect may be a copy of the pattern. A *candidate* is the location in the text that corresponds to the element $P[0,0]$ of a candidate pattern (*i.e.* the *origin* of the candidate pattern). A candidate pattern is an *occurrence* if it actually is a copy of the pattern. An element of the text is *active* if it is a candidate. A *block* or subarray of text is active if it contains an active element. Two candidates are *compatible* if actual copies of the pattern origination at the candidates would not overlap or would overlap without mismatch in the overlapping characters (independent of what may actually be in the text). A set of candidates is compatible if every pair of candidates in the set is compatible.

Algorithm Overview:

Our algorithm consists of a *pattern preprocessing* part and a *text scanning* part.

Pattern Preprocessing (Time: $O(|P|)$): For periodicity class and witness tables. (See section 4.3.)

Text Processing (Time: $O(|c(T)|)$): Performed in three phases:

- **Candidate selection:** A text scan to determine active elements. This restricts the number of candidates to $O(|c(T)|)$.
- **Block Compatibility:** A partition of the text into active blocks of size $\frac{m}{d} \times \frac{m}{d}$ (d =constant). Within each block, the set of candidates is compatible. This critical phase of the algorithm uses the periodicity class of the pattern.
- **Verification:** A text scan to verify that candidates are occurrences.

4 Two Dimensional Periodicity, Witness Tables and Pattern Preprocessing

4.1 Two Dimensional Periodicity

Our algorithm uses the periodicity class of the pattern. In [2], periodicity in two-dimensional arrays is defined based on the ability of an array A to overlap itself without mismatch. For simplicity, let A be an $m \times m$ array (although A can be any rectangular array).

Definition 3 Each corner of A is included in a subarray called a *quadrant* (see figure 1) of size $\frac{m}{3} \times \frac{m}{3}$ or less. $A[0,0]$ is included in *quadrant I* and $A[m-1,0]$ is included in *quadrant II*. Two copies of an array are *in register* if some of their elements overlap and there are no mismatches between overlapping elements. Given two copies of A, if $A[0,0]$ overlaps $A[r,c]$ ($c > 0$) and the copies are in register, then $A[r,c]$ is a *quadrant I source* and vector $\vec{v} = r\vec{y} + c\vec{x}$ (or simply (r,c)) is a *quadrant I symmetry vector* where $\vec{y}$ is the unit vector in the direction of increasing row index and $\vec{x}$ is the unit vector in the direction of increasing column index. If $A[m-1,0]$ overlaps $A[r,c]$ ($r < m-1$) and the copies are in register, then $A[r,c]$ is a *quadrant II source* and vector $\vec{v} = (r-m+1)\vec{y} + c\vec{x}$ is a *quadrant II symmetry vector*. The *length* $|\vec{v}|$ of a symmetry vector $\vec{v}$ is the maximum of the absolute values of its coefficients. A *lexicographic ordering* of quadrant I vectors (quadrant II vectors) is accomplished by sorting the vectors first by length, and then for vectors of the same length, by reverse sorting them by column coefficient and then sorting them by row coefficient (by reverse sorting them by absolute value of row coefficient and then sorting them by column coefficient). A *shortest* vector is the smallest in its lexicographic ordering. The *basis vectors* for array A consist of the shortest quadrant I vector (if any) and the shortest quadrant II vector (if any). A symmetry vector is $\frac{m}{d}$ *periodic* if it has length $< \frac{m}{d}$ where $d \geq 3$.

Definition 4 If a copy of a pattern A occurs in a text B, then the location of $A[0,0]$ in that copy is a quadrant I *source of the pattern in the text* and the location of $A[m-1,0]$ is a quadrant II source in the text. Sources of a pattern in a text are *ordered monotonically* if they are non-decreasing in both row and column indices (quadrant I sources) or non-increasing in row index and non-decreasing in column index (quadrant II sources).

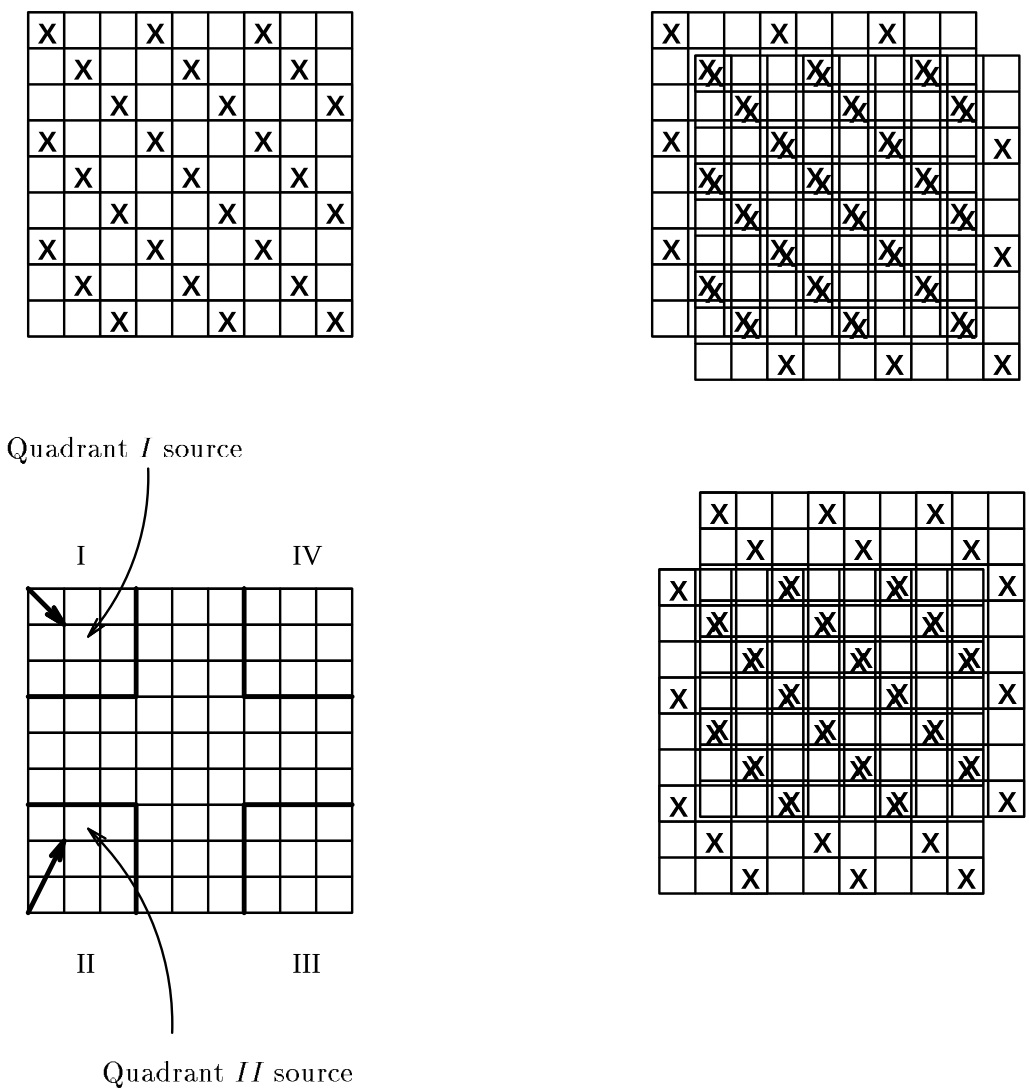


Figure 1:
Each overlap without mismatch defines a *source* and a *symmetry vector*.

There are four classes of periodicity for rectangular arrays [2], determined by the periodicity of the basis vectors. Consider an $\frac{m}{d} \times \frac{m}{d}$ block B of *text*. The periodicity class of pattern A determines the number and arrangement of sources of the pattern (in the descriptions below, either all quadrant I or all quadrant II) that can occur in text block B (figure 2):

1. *Non-Periodic.* Neither basis vector is $\frac{m}{d}$ periodic. Only one source of A can lie in B.
2. *Lattice Periodic.* Both basis vectors are $\frac{m}{d}$ periodic. At least three non-collinear sources of A can lie in B. They must correspond to the *nodes of a lattice* determined by the basis vectors.
3. *Line Periodic.* Only one basis vector is $\frac{m}{d}$ periodic. At least two sources of A can lie in B. Sources must be collinear.
4. *Radiant Periodic.* Only one basis vector is $\frac{m}{d}$ periodic. At least three non-collinear sources of A can lie in B. The sources must be ordered monotonically.

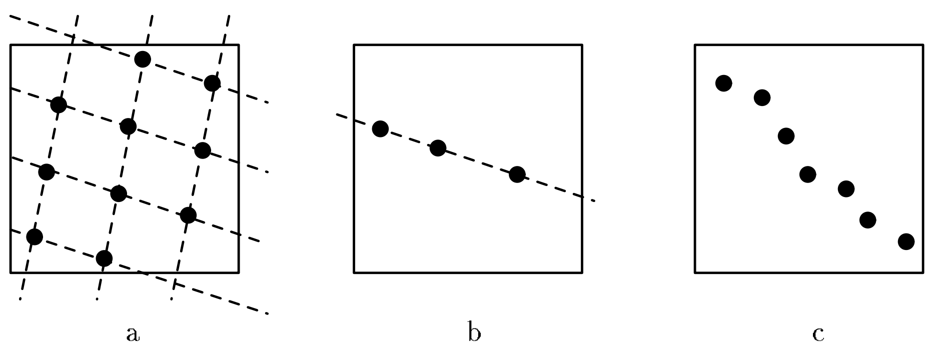


Figure 2:
Examples of how sources of the pattern could appear in an $\frac{m}{d} \times \frac{m}{d}$ block of text.
a) lattice periodic b) line periodic c) radiant periodic

4.2 Witness Tables

An important feature in recent pattern matching algorithms [14, 7, 3, 10, 9] has been the use of a *Witness* table. If we are given two overlapping candidate patterns that are incompatible, the witness table gives us a location in their overlap where the candidates disagree on the character. By looking at the text character at that location, at least one of the candidates can be eliminated. This is the "duel paradigm" introduced by Vishkin [14].

The $Witness$ table we use has been described fully in [2]. Specifically, $Witness$ tells us the following information in constant time (figure 3). Given two candidates, C_1 at text location $T[r, c]$ and C_2 at text location $T[r+i, c+j]$, if $Witness[i, j] = [m, m]$ then the candidate patterns are compatible. Otherwise, $Witness[i, j] = [a, b]$, and text location $T[r+i+a, c+j+b]$ matches at most one of pattern locations $P[a, b]$ (the candidate pattern at C_2) or $P[i+a, j+b]$ (the candidate pattern at C_1). If we know the character at the text location, we can eliminate either one or the other or both

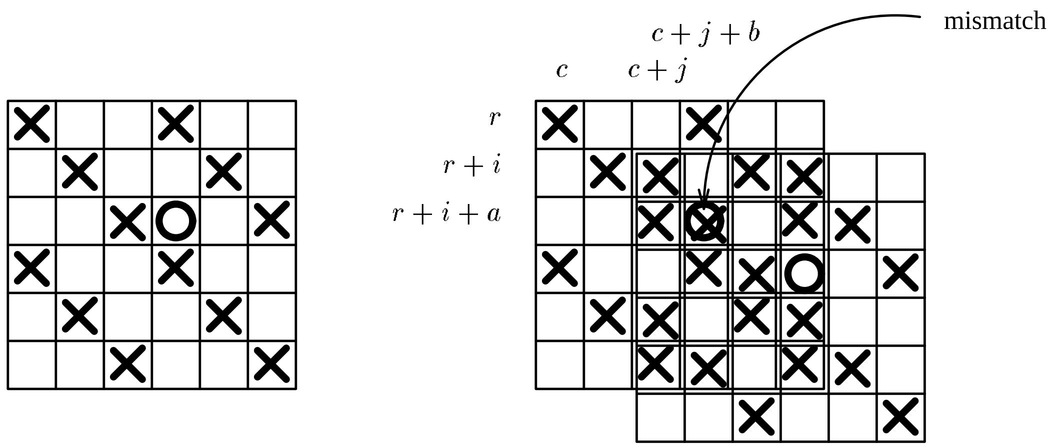


Figure 3:
The witness table gives the location of a mismatch (if one exists) for two overlapping patterns. Here, $Witness[i, j] = (a, b)$.

of the candidates. In [2] we gave a serial algorithm for computing the $Witness$ array that runs in time $O(m^2 \log m)$. The time was improved to $O(m^2)$ by [10].

If the pattern is *lattice periodic*, we have need of an additional table. Suppose we are given two sets S_1 and S_2 of candidates in a block of the text such that all the candidate patterns overlap and all the candidates within S_1 are compatible and all the candidates within S_2 are compatible, but any candidate from S_1 is incompatible with any candidate from S_2. We know that every one of the candidates in at least one set mismatches the text, but which set? By performing duels between each candidate in S_1 and every candidate in S_2, we could determine which set to discard. But, in general, this takes too long. We would like a single witness to do the trick. If the pattern is lattice periodic, it is *always* possible to arbitrarily choose a single candidate from S_1 and a single candidate from S_2 and find a witness that is an element of all the other candidates. Thus a single character in the text eliminates all the candidates in one or the other or both sets. The method requires a second witness table $Alt - Witness$. The construction of this table is straightforward and has been described (along with the proof that a single witness can be used when the pattern is lattice periodic) in [4].

4.3 Pattern Preprocessing

For the pattern preprocessing part of the algorithm, the input is the $m \times m$ pattern P. (For the remainder of this paper, we will assume that the pattern is a square array to simplify the explanation, although, *our algorithm applies to any rectangular array.*)

Definition 5 A *pattern characteristic seam* is the location of any single row seam. An array can have one of two *orientations*, either the original presentation of the array, or a rotation of the array by 90^0 degrees. Given an orientation, rows increase from top down and columns increase from left to right.

Algorithm Pattern Preprocessing

1. Select an orientation of the pattern (and text) so that at least one row seam exists.
2. Choose a pattern characteristic seam. For every pattern row, note the closest non-stripe row below and above it.
3. Analyze pattern for periodicity class and produce $Witness$ table (and $Alt - Witness$ table if the pattern is lattice periodic).

A row seam exists in one of the two orientations of the pattern or else the pattern is trivial. The pattern characteristic seam is used in the candidate selection phase. The location of the closest non-stripe row is used in the verification phase.

Complexity: The pattern classification, $Witness$ table and $Alt - Witness$ table can be obtained in time $O(P)$ [2, 10, 4]. The remaining steps are easily performed in time $O(P)$.

5 Periodicity Properties

Consider again, the case where we are given two sets S_1 and S_2 of overlapping candidates such that all the candidates within S_1 are compatible and all the candidates within S_2 are compatible, but the two sets are incompatible. If the pattern is lattice periodic, we know how to use a single witness to determine which set to discard. Here, we show that when the pattern is line or radiant periodic, **it is possible to discard candidates without looking at any witnesses.**

Definition 6 For a pattern array P, a rectangular subarray $P' \subset P$ is a *maximal* $\frac{m}{d}$ *lattice periodic subarray* if:

1. P' is $\frac{m}{d}$ lattice periodic.
2. any subarray Q formed by extending P' within P by one row or one column is not $\frac{m}{d}$ lattice periodic.

At the end of this section, we show how to find a maximal lattice periodic subarray. Below, we prove three theorems that allow us to accomplish the following:

- If a subarray P' of a pattern P is maximal lattice periodic, then by finding occurrences of P' in the text, we can monotonically order candidates without looking at any witnesses.
- Let R' and C' be subarrays consisting, respectively, of the first r rows and first c columns of a pattern (where the quadrant I basis vector is $\vec{v} = (r, c)$). If monotonically ordered candidate patterns contain occurrences of both R' and C', then we can make the candidates compatible without looking at witnesses.
- Let R' and C' be as above. Let $\hat{R}$ and $\hat{C}$ be the subarrays consisting, repectively of the last r rows and last c columns of a pattern. Let P' and $\hat{P}$ be maximal lattice periodic

subarrays occupying, respectively, the upper left and lower right corners of P. If monotonically ordered candidate patterns contain occurrences of $P', \hat{P}, C'$ and $\hat{C}$, or if monotonically ordered candidate patterns contain occurrences of $P', \hat{P}, R'$ and $\hat{R}$, then we can make candidates compatible without looking at witnesses.

The algorithmic steps presented in Section 6 closely follow these theorems. Below, we assume that candidates A and B occur within a block of text no larger than $\frac{m}{d} \times \frac{m}{d}$.

Theorem 1 Let A and B be two incompatible, non-monotonically ordered candidates patterns for P with A left of and below B. Additionally, let A and B both contain occurrences of a maximal lattice periodic subarray $P' \subset P$. If P' has width $< m$, then A is not an occurrence of P and if P' has height $< m$ then B is not an occurrence of P.

Proof: (See figure 4). Let Q be P' with one additional row or column added. That is if P' is $k \times l$ then Q is either $(k+1) \times l$ or $k \times (l+1)$. Now, two copies of Q at A and B can not be compatible, so Q_A and Q_B must have a witness. If Q is $k \times (l+1)$, then the witness must occur in the part of column $l+1$ relative to A that overlaps Q_B. But, this actually overlaps P'_B, so the witness agrees with candidate B, not A. Similarly, if Q is $(k+1) \times l$ then the witness must occur in the part of row $k+1$ relative to B that overlaps Q_A. But this actually overlaps P'_A, so the witness agrees with candidate A, not B. In the first case, P' has width $< m$ and in the second case, P' has height

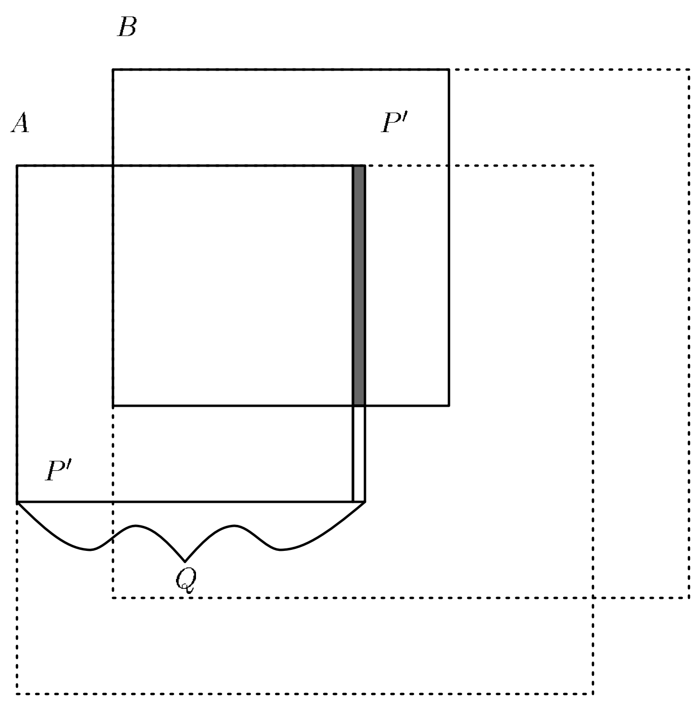


Figure 4:
The witness occurs in the shaded region and therefore must agree with B.

$< m$. Since P contains one or the other or both cases of Q, either A or B or both can not be an occurrence of P. ▮

Corollary 2 If A and B are incompatible and are not monotonically ordered, then we can determine which one is not an occurrence of P by the size of P' without looking at a witness.

Theorem 3 Let A and B be two monotonically ordered incompatible candidate patterns, with B below and/or right of A. Let $\vec{v} = (r, c)$ be the quadrant I basis vector of the pattern. Then a witness for A and B occurs within the first r rows or the first c columns of B.

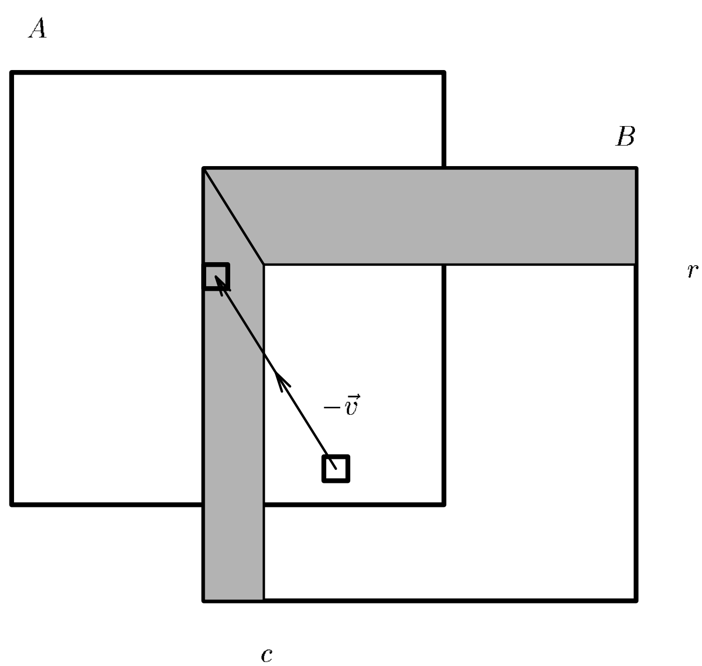


Figure 5:
Any witness can be mapped into the first r rows or c columns of B
by one or more applications of $-\vec{v}$.

Proof: (See figure 5). For a pattern P, let $P_{(r,c)}$ be the elements of the first r rows and the first c columns. Let $P_{\vec{v}}$ be the set $\{u \in P : u - \vec{v} \in P\}$. In other words, $P_{\vec{v}}$ is all of P except $P_{(r,c)}$. The witness w occurs somewhere within $S = A \cap B$. If w occurs within $S_{(r,c)} \subset B_{(r,c)}$ then we are done. Suppose it does not. Then w occurs somewhere in $S_{\vec{v}} = A \cap B_{\vec{v}}$. But every element w in $S_{\vec{v}}$ can be mapped to an identical element w' in $S_{(r,c)}$ by one or more applications of $-\vec{v}$ Since $S_{(r,c)} \subset B_{(r,c)}$, if A and B differ at w, they also differ at w'. ▮

Corollary 4 If A and B are incompatible, monotonically ordered and the first r rows and first c columns of B match the text, then A can not be an occurrence of P.

Theorem 5 Let A and B be two monotonically ordered incompatible candidates with B below and/or right of A. Let $\vec{v} = (r, c)$ be the quadrant I basis vector of the pattern. Let P' and $\hat{P}$ be two maximal lattice periodic subarrays of P of size at least $\frac{3m}{5} \times \frac{3m}{5}$ with P' containing $P[0,0]$ and $\hat{P}$ containing $P[m-1, m-1]$. Let A and B both contain occurrences of P' and $\hat{P}$. Then, a witness for A and B occurs within either the first c columns of B or the last c columns of A or within the first r rows of B or the last r rows of A.

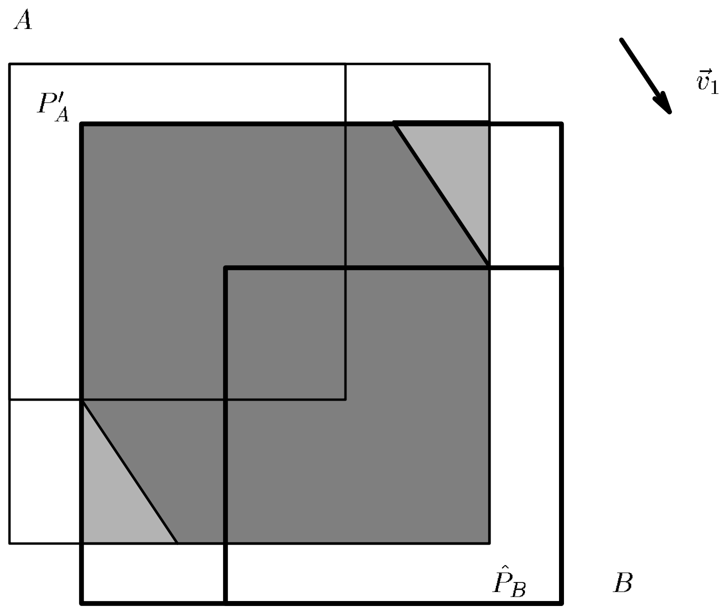


Figure 6:
A witness can not occur within the darkly shaded region because it could be mapped by $\vec{v}_1$ into either the intersection of P'_A and P'_B or into the intersection of $\hat{P}_A$ and $\hat{P}_B$.

Proof: (See figure 6.) First, notice that if the witness could be mapped into the overlap of P'_A and P'_B or into the overlap of $\hat{P}_A$ and $\hat{P}_B$ then the two copies of P' or the two copies of $\hat{P}$ could not both exist in the text. Since $P'_A \cap P'_B = P'_A \cap B$ and $\hat{P}_B \cap \hat{P}_A = \hat{P}_B \cap A$ and since the size of both P' and $\hat{P}$ is at least $\frac{3m}{d} \times \frac{3m}{d}$, the region where witnesses can not occur occupies the entire middle of the overlap of A and B. Only two triangular regions remain that could contain witnesses. If a witness occurs in the lower left triangular region, then that witness can be mapped by $\vec{v}_1$ into both the first c columns of B and the last r rows of A. Similarly, if the witness occurs in the upper right triangular region, then that witness can be mapped by $\vec{v}_1$ into both the last c columns of A and the first r rows of B. ▮

Corollary 6 Let A and B be incompatible and monotonically ordered and let both contain occurrences of P' and $\hat{P}$. If the first r rows of B match the text and if a witness occurs in the upper right triangular region, or if the first c columns of B match the text and a witness occurs in the lower left triangular region, then A can not be an occurrence of P.

Similarly, if the last r rows of A match the text and a witness occurs in the lower left triangular region, or the last c columns of A match the text and a witness occurs in the upper right triangular region, then B can not be an be an occurrence of P.

Observation 7 Let P be $\frac{m}{d}$ line or radiant periodic with a quadrant I basis vector $\vec{v} = (r, c)$. If P is radiant periodic, it is non-periodic in a block of size $r \times c$ and if it is line periodic, it is non-periodic in both a block of size $r \times \frac{m}{d}$ and a block of size $\frac{m}{d} \times c$.

5.1 Maximal Lattice Periodic Subarrays

For the algorithm presented in Section 6, we need a maximal lattice periodic subarray if the pattern is radiant periodic. In [10] it was shown that an $m \times m$ array that is $\frac{m}{d}$ radiant periodic ($d \geq 4$) has a lattice periodic subarray of size at least $\frac{(d-2)m}{d} \times \frac{(d-2)m}{d}$. This subarray can be extended to a maximal lattice periodic subarray.

The next theorem establishes that if an $m \times m$ array P is $\frac{m}{d}$ periodic in quadrant I only and P' a subarray of P of size at least $\frac{3m}{d} \times \frac{3m}{d}$ is $\frac{m}{d}$ periodic in both quadrants, then if P' is extended within P and the quadrant II basis vector changes, then the new basis vector (if any) cannot also be $\frac{m}{d}$ periodic. In other words, if we originally know the quadrant II basis vector, then when we observe that it changes, we do not have to worry that the new basis vector (if any) is also $\frac{m}{d}$ periodic. Thus, to find a maximal lattice periodic subarray, we start with a known subarray that is lattice periodic with known basis vectors. Then we extend the array one row at a time (at the top and bottom) until the quadrant II basis vector $\vec{v}_2 = (r_2, c_2)$ changes. We back up one row and extend by one column at a time (at the left and right sides) until the quadrant II basis vector changes again and then back up by one column. To determine if the basis vector changes, we merely test the character x in $P[r, c]$ (where r or c is the new row or column) to see if it matches the corresponding character at $P[r + r_2, c + c_2]$ or $P[r - r_2, c - c_2]$.

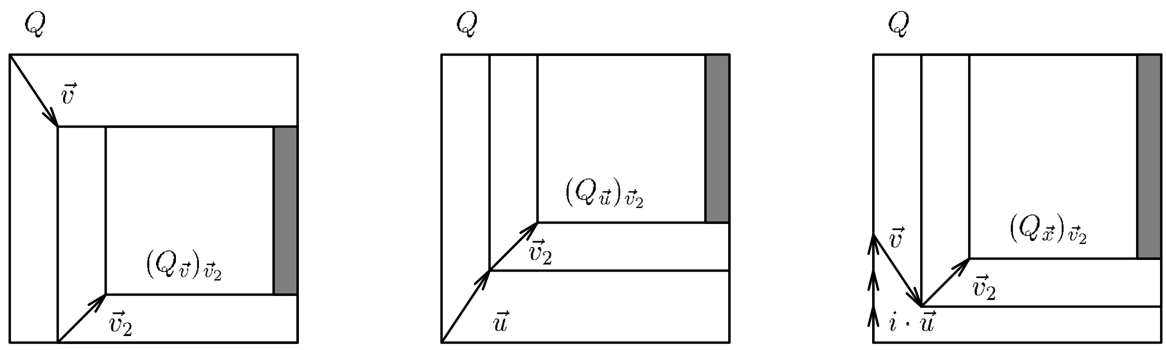
Although the following theorem is written in terms of quadrant I, by symmetry, it applies also to quadrant II.

Theorem 8 Let P be an $m \times m$ array that is $\frac{m}{d}$ periodic in quadrant I only (with $d \geq 5$). Let P' be an $\frac{m}{d}$ lattice periodic subarray of P of size at least $\frac{3m}{d} \times \frac{3m}{d}$ and let its quadrant II basis vector be $\vec{v}_2$. Then no extension Q of P' by one row or one column within P can have a quadrant II basis vector $\vec{u}$ such that $\vec{u} \neq \vec{v}_2$ and $|\vec{u}| < \frac{m}{d}$.

Proof: (See figure 7.) Let the periodic basis vector of P be $\vec{v} = (r, c)$. Wolog, suppose Q is P' with an additional column along the right side. (Other cases can be reflected or rotated into this case.) Let P' be of size $k \times l$ and let the quadrants I and II basis vectors of P' be $\vec{v}_1 = (r_1, c_1)$ and $\vec{v}_2 = (r_2, c_2)$. Let Q have a quadrant II basis vector $\vec{u} \neq \vec{v}_2$ with $|\vec{u}| < \frac{m}{d}$. Note that $|\vec{u}| \geq |\vec{v}_2|$. We show that $\vec{v}_2$ is a symmetry vector of Q and therefore $\vec{u}$ is not a basis vector.

To show that $\vec{v}_2$ is a symmetry vector of Q, we need only show that the upper $k - |r_2|$ elements of the last column of Q can be mapped from identical elements by vector $\vec{v}_2$. That is, we must satisfy the following condition:

$$Q[0..k - |r_2| - 1, l] \text{ matches } Q[|r_2|..k - 1, l - c_2]$$



a b c

Figure 7:
Each of the following subarrays is identical to some subarray of $P'_{\vec{v}_2}$.
a) $(Q_{\vec{v}})_{\vec{v}_2}$ b) $(Q_{\vec{u}})_{\vec{v}_2}$ c) $(Q_{\vec{x}})_{\vec{v}_2}$

Define P_w as the subarray $\{u \in P : u - \vec{w} \in P\}$. If $\vec{w} = (r_w, c_w)$ is a quadrant II symmetry vector, then every element $P[x,y]$ of P_w satisfies $P[x,y] = P[x + |r_w|, y - c_w]$. Specifically, every element of P_{v_2} satisifes $P[x,y] = P[x + |r_2|, y - c_2]$

Observe first that $\vec{v}$ must be a quadrant I periodic vector of Q. (Since Q is a subarray of P.) Therefore, $Q[r,c]$ is a quadrant I source. Since $c > 0$ (a necessary condition for a quadrant I symmetry vector), Q_v is identical to a subarray of P'. Then $(Q_v)_{v_2}$ is identical to a subarray of P'_{v_2}. Therefore, we have satified:

$$Q[r..k - |r_2| - 1, l] \text{ matches } Q[r + |r_2|..k - 1, l - c_2]$$

Case 1: $c_u > 0$.

Q_u is identical to a subarray of P' and $(Q_u)_{v_2}$ is identical to a subarray of P'_{v_2}. Therefore, we have satisfied:

$$Q[0..k - |r_u| - |r_2| - 1, l] \text{ matches } Q[|r_2|..k - |r_u| - 1, l - c_2]$$

Since all of $r, r_2, r_u < \frac{m}{d}$ and $k \geq \frac{3m}{d}$, $0 \leq k - r - |r_u| - |r_2| - 1$ and $r \leq k - |r_u| - |r_2| - 1$. Thus the entire upper $k - |r_2|$ elements of the column have been covered.

Case 2: $c_u = 0$.

Let $i = \min(j : r - j \cdot |r_u| \leq 0, j = 1, 2 \ldots)$. Since $i \cdot \vec{u}$ is a quadrant II vector and $\vec{v}$ is a quadrant I vector, then by Lemma 2 of [2], $\vec{x} = i \cdot \vec{u} + \vec{v}$ is a quadrant II symmetry vector with $c_x = c > 0$ and $|r_x| < \frac{m}{d}$. Thus, $Q_{\vec{x}}$ is a subarray of p', and $(Q_{\vec{x}})_{\vec{v}_2}$ is a subarray of $P'_{\vec{v}_2}$. Therefore, we have satisfied:

$$Q[0..k - |r_x| - |r_2| - 1, l] \text{ matches } Q[|r_2|..k - |r_x| - 1, l - c_2]$$

But, $r \leq k - |r_x| - 1$. Thus again, the entire upper $k - |r_2|$ elements of the column have been covered. ▮

6 Text Analysis: Block Compatibility Phase

The text analysis is performed in three phases as outlined in the overview. We concentrate in this section on the Block Compatibility Phase [6]. This phase is the only one which requires the complicated machinery of two dimensional periodicity. The other phases are straightforward, requiring mostly bookkeeping [1]. They are described in section 7.

The goal is to partition the text into active disjoint blocks containing only compatible candidates. We consruct the blocks in a logarithmic number of steps by merging active regions of the text. To construct a given block B of size $\frac{m}{d} \times \frac{m}{d}$, we first merge candidates within individual columns of B, then we merge the columns.

Conceptually, when we merge within a column (or among columns), the entire column (or final block B) can be represented by the root of a binary tree. Each leaf of the tree represents a single candidate (or a single active column). Merging is done bottom up from the leaves at each internal node. *Each merge requires only a single witness.*

The compatibility phase is performed in stages $i = 1..\log m$. From the selection phase, candidates are initially stored in column lists C_j $j = 1 \ldots n$. These are used as the original block lists. At the end of the each stage, the size of the blocks is doubled and the number of block lists is halved. During stage i, the block lists are scanned to determine the list D_i of duels which should occur. At most a single witness is identified for each duel. The list of witnesses is sorted using a modified radix sort and then the content of each witness is determined by reference to the finger tree of text characters (Section 6.2). Once we have the witness characters, candidate patterns that do not match the text are eliminated. At the end of stage i, each block contains only compatible candidates.

6.1 Specifics for the Candidate Classes

When the pattern is non-periodic, the procedure is straightforward. At every duel, at most one candidate survives to the next stage. The procedure for the remaining periodicity classes is outlined below. For brevity, we only describe the steps when we are merging columns. Merging within a column reduces to either the non-periodic case or the line periodic case. The following lists of assumptions and notation are common to all the procedures below.

Assumptions and Notation:

- P is the original pattern array.
- Whenever we use P or a subarray of P as a pattern, we have precomputed the table *Witness* (and *Alt-Witness* if necessary).
- The subarrays $R', \hat{R}, C'$ and $\hat{C}$ are, respectively, the first r rows, the last r rows, the first c columns and the last c columns of P, where the quadrant I basis vector of P is $\vec{v} = (r, c)$.
- B_1 and B_2 are, respectively, the left and right blocks being merged.
- Each block contains a candidate list ordered by non-decreasing column index.

- Within the candidate lists, c_1 is the last candidate in B_1 and c_2 is the first candidate in B_2.

Lattice Periodic Patterns

Assumptions:

- P is $\frac{m}{5}$ lattice periodic.

1. **Make candidates compatible.** When merging B_1 and B_2, find the witness w for c_1 and c_2 (if it exists) using the witness tables. Every candidate contains element w and at most, one set of candidates can agree with the text character at w.

Radiant Periodic Patterns

Assumptions:

- P is $\frac{m}{5}$ radiant periodic.
- The subarrays P' containing $P[0,0]$ and $\hat{P}$ containing $P[m-1, m-1]$, both of size at least $\frac{3m}{5} \times \frac{3m}{5}$, are maximal $\frac{m}{5}$ lattice periodic.

1. **Make candidates sparse.** Merge blocks up to size $r \times c$. (P is non-periodic for this block size by Observation 7.)
2. **Find P' in text.** Run the entire text analysis using P' as the pattern. Each remaining candidate is an occurrence of P'.
3. **Make candidates monotonic.** Merge blocks up to size $\frac{m}{5} \times \frac{m}{5}$. Here, *we do not look at witnesses.* When combining blocks B_1 and B_2, examine c_1 and c_2. If they are not monotonically ordered, then if P' has width m, eliminate c_1, else if P' has height m eliminate c_2, else eliminate both. Repeat (with new c_1 and/or c_2) until the candidates in both blocks are monotonic (Corollary 2).
4. **Find $\hat{P}$ in text.** Run the entire text analysis using $\hat{P}$ as the pattern. Each remaining candidate is an occurrence of $\hat{P}$.
5. **Make candidates within the same column of $r \times c$ blocks compatible.**
   (a) For each candidate, if it is one of several candidates in the same column of $r \times c$ blocks (from step 1.), verify that it is an occurrence of both R' and $\hat{R}$. (This works within our time bounds because the candidates are already monotonic and therefore at most four copies of R' or $\hat{R}$ overlap any one text row segment.)
   (b) Here, *we do not look at witnesses.* Let the candidates in the same column of $r \times c$ blocks be $d_1, \ldots, d_k$. The procedure call **Compare**(d_1, d_2) (below) makes the candidates compatible (Corollary 6). Let $next(c)$ be the candidate following candidate c. Let $prec(c)$ be the candidate preceding candidate c.
   Compare(a,b)

If $b = null$ exit;
 else if $a = null$ Compare(b, next(b));
 else if a and b are compatible, Compare(b, next(b));
 else if the witness for a and b falls within the upper right triangular region, eliminate a, Compare(b,prec(a));
 else if the witness for a and b falls within the lower left triangular region, eliminate b, Compare(a, next(b));

6. **Make candidates within the same row of $r \times c$ blocks compatible.**
   (a) For each candidate, if it is one of several candidates in the same row of $r \times c$ blocks, verify that it is an occurrence of both C' and $\hat{C}$.
   (b) Similar to step 4(a).
7. **Find R' and C'.** Verify, for each candidate, if it is an occurrence of both R' and C'. If not, discard the candidate. (This works within our time bounds because we have initially made the candidates within the same row or column of $r \times c$ blocks compatible.)
8. **Make candidates compatible.** For each final active block B, scan backwards through its candidate list. Initially, let the current candidate, d, be the last candidate. If d is incompatible with the preceding candidate x in the list, eliminate x, else set $d = x$. Repeat to the beginning of the list (Corollary 4).

Line Periodic Patterns

Assumptions:

- P is $\frac{m}{5}$ line periodic.
- If the central $\frac{3m}{5} \times \frac{3m}{5}$ subarray of P is not $\frac{m}{5}$ lattice periodic, then $P_{central}$ is this subarray. Else, $P_{central}$ is the central subarray extended so that it is maximal $\frac{m}{5}$ lattice periodic.

1. **Make candidates sparse.** Merge blocks up to size $r \times \frac{m}{5}$ and independently up to size $\frac{m}{5} \times c$. (P is non-periodic for these block sizes by Observation 7.)
2. **Find R' and C'.** Verify, for each candidate, if it is an occurrence of both R' and C'. If not, discard the candidate. (This works within our time bounds because we have initially made the candidates sparse.)
3. **If $P_{central}$ is maximal lattice periodic.**
   (a) **Find $P_{central}$.** Run the entire text analysis using $P_{central}$ as the pattern. Each remaining candidate is an occurrence of $P_{central}$.
   (b) **Make candidates monotonic.** Merge blocks up to size $\frac{m}{5} \times \frac{m}{5}$. Here, *we do not look at witnesses.* When combining blocks B_1 and B_2, examine c_1 and c_2. If they are not monotonically ordered, then if $P_{central}$ has width m, eliminate c_1, else if $P_{central}$ has

height m eliminate c_2, else eliminate both. Repeat (with new c_1 and/or c_2) until the candidates in both blocks are monotonic (Corollary 2).

(c) **Make candidates compatible.** For each final active block B, scan backwards through its candidate list. Initially, let the current candidate, d, be the last candidate. If d is incompatible with the preceding candidate x in the list, eliminate x, else set $d = x$. Repeat to the beginning of the list (Corollary 4).

4. **If $P_{central}$ is not lattice periodic.**

(a) **Make candidates compatible** Merge blocks up to size $\frac{m}{5} \times \frac{m}{5}$. When combining blocks B_1 and B_2, examine c_1 and c_2.

(i.) If they are not monotonically ordered, then find the witness w for c_1 and c_2 using the witness table for $P_{central}$. (Such a witness exists because $P_{central}$ is not lattice periodic.) Every candidate contains element w and at most one set of candidates can agree with the text character at w.

(ii.) If c_1 and c_2 are monotonically ordered but not compatible, then *we do not use any witness*. Instead, discard c_1 and repeat (with new c_1) until c_1 and c_2 are compatible (Corollary 4).

Theorem 9 The Block Compatibility phase is correct and runs in time $O(|c(T)|)$.

Proof: The correctness follows from the theorems and corollaries given in section 5 and [1, 2, 4]. The total number of block merging steps is $O(|c(T)|)$. Each merge requires a single witness or contains extra steps that do not require looking up witnesses, but are bounded in total by the number of candidates. All verifications can be done in time $O(|c(T)|)$ (see section 7.2). One difficulty that arises is retrieving the text characters at the witness locations. In the next section, we show how to do this in time $O(|c(T)|)$. Thus, the total time is $O(|c(T)|)$. ∎

6.2 Looking up witnesses

Here, we bound the time required to look up witnesses. In particular, we will have $\log m$ batches of witnesses from the duel lists with $O(|c(T)|/2^i)$ witnesses in batch i, and thus $O(|c(T)|)$ witnesses in total. We define the *Batch Dictionary Lookup Problem* to be:

Preprocess: A list $L[1..n]$ of n distinct sorted numbers.

Given: A sequence $W_1, \ldots, W_{\log n}$ of sorted lists such that there are no more than $n/2^i$ numbers in W_i.

Output: A new sequence $N_1, \ldots N_{\log n}$ of lists such that $N_i[j] = k$ if $L[k-1] \leq W_i[j] < L[k]$. Furthermore, we must compute the N_i in batches, that is, we must compute N_i after seeing W_i but before seeing W_{i+1}.

In our case, L contains the locations of the $O(|c(T)|)$ initial characters in the runs of the row compressed run length compression of T. The W_i contain witness locations from the duels. Each N_i tells us where to look up the value of the text characters at the witness locations.

6.2.1 Sorting the witnesses

The values in L are given to us in sorted order, first by row and then by column. Initially, the W_i are not sorted. We sort them using a modified radix sort. (The first pass is on the column indices and the second pass is on the row indices – we show how to do the first pass). Recall that T has uncompressed size $n \times n$ and $n \leq O(|c(T)|)$.

Witness Sort

1. Bucket sort the keys in W_i into n buckets $B_1, \ldots, B_n$, and keep a non-duplicative index list L of buckets that contain keys.
2. Bucket sort the indices in L into $\frac{n}{2^i}$ buckets $I_1, \ldots, I_{\frac{n}{2^i}}$. Each bucket contains at most 2^i indices and there are at most $O(|c(T)|/2^i)$ indices in total.
3. Sort each of the buckets $I_1, \ldots, I_{\frac{n}{2^i}}$ using a comparison based sorting algorithm.
4. Pick up the buckets I in order and use the sorted indices to pick up the buckets B that contain keys in order.

Theorem 10 The time to sort all the keys in the W_i is $O(|c(T)|)$.

Proof: To sort a single W_i containing at most $O(|c(T)|/2^i)$ keys takes $O(|c(T)|/2^i)$ time in steps 1, 2 and 4. Each index in step 3 is sorted using $O(\log 2^i) = O(i)$ comparisons and there are $O(|c(T)|/2^i)$ indices, for a total time in step 3 and thus in all the steps of $O(|c(T)|i/2^i)$. For all the W_i we have $\sum_{i=1}^{\log m} O(|c(T)|i/2^i) = O(|c(T)|)$. ∎

6.2.2 Finger trees

Theorem 11 The *Batch Dictionary Lookup Problem* can be solved in $O(n)$ time.

Proof: Note that since both L and the W_i are sorted, the lookup problem is equivalent to merging the two lists. Brown and Tarjan [8] showed a linear time finger tree construction algorithm. They further showed that sorted lists of size n and m can be merged in time $O(m \log(n/m))$ once we have built a finger tree on the list of length n. In our case, we build a finger tree on L in time $O(n)$, and then compute N_i for list W_i of size $m \leq n/2^i$ in time $O(\frac{n}{2^i}\log(2^i)) = O(ni/2^i)$. The total work is therefore $O(n) + \sum_{i=1}^{\log n} O(ni/2^i) = O(n)$. ∎

7 Text Analysis: Candidate Selection and Verification Phases

The remaining phases of the text analysis algorithm perform the initial selection of candidates and the final verification of copies of the pattern. The techniques are straightforward, constituting mostly scans of the text and data structures to handle bookkeeping

7.1 Text Analysis – Candidate Selection

Each candidate must contain a pattern characteristic seam. Recall that a candidate is the row and column index of the upper left corner of a candidate pattern. We store candidates on column lists $C_1, \ldots, C_{n-m+1}$ and row lists $R_1, \ldots R_{n-m+1}$ for use in the remaining stages of the text analysis.

We scan the row compressed form of the text, row by row looking for characteristic seams. If such a seam is found, we compute the indices $T[r, c]$ of the candidate and put it on the column list C_c and row list R_r. This phase clearly takes time $O(|c(T)|)$ and limits the initial number of candidates to $O(|c(T)|)$.

7.2 Text Analysis – Verification

In the verification phase, we need to deal with two types of situations. In one situation, we are verifying a possibly large group of overlapping compatible candidates (such as candidates for the entire pattern P or the subarray P'). In the other situation, we are verifying monotonically ordered not necessarily compatible subarrays (such as R' or C'), but only a constant number of the subarrays can overlap. For each block of candidates, we have a candidate list sorted by non-decreasing column index. Each of the candidates is also linked onto a row list R_i from the candidate selection step.

In testing the candidates against the text, we employ the following main ideas:

- For each segment of the text, at most $2(d+1)^2$ blocks must be examined. These blocks are the only ones whose candidates could contain the first or last character of the segment.
- Long segments (with length $> m$) in the text, may represent a stripe in many candidates. We note that a stripe occurs in these candidates without actually marking each one.
- We can verify that row stripes in a candidate pattern contain the correct symbol using a single compressed column of the text.

7.2.1 Verifying compatible candidates

When we verify a group of overlapping compatible candidates in the $\frac{m}{d} \times \frac{m}{d}$ blocks, we proceed as follows. We scan the row compressed form of the text, row by row. For each segment S in text row r, we test S against at most $2(d+1)^2$ blocks whose candidates could contain either the first or last character of S. Let B be one of those blocks. Of the candidates in B, not all may contain row r of the text. In particular, only candidates in rows $r-m+1, \ldots, r$ will contain row r. If B contains characters in rows $< r-m+1$ or in rows $> r$ then those candidates must be removed from consideration before testing the segments in text row r. We use the lists R_i to remove candidates from consideration in two passes through the text, a top-down pass and a bottom-up pass.

Verify Candidates Using Text Rows

/* Top Down */

1. For $r = m+1 \ldots n$

   1.1 Remove candidates on row list R_{r-m} from the block lists.

1.2 Process text row r segments using only blocks which contain no candidates *below* row r.

2. Restore the block lists with the surviving candidates.

/* Bottom-Up */

3. For $r = n - m \dots 1$

3.1 Remove candidates on row list R_{r+1} from the block lists.

3.2 Process text row r segments using only blocks which contain no candidates *above* row $r - m + 1$.

Once we have removed candidates from consideration, let the remaining candidates in B be $(c_1 \dots c_k)$. In the processing steps, we test S against the first (c_1) and last (c_k) candidates in the list. If a mismatch is found at either candidate, that candidate is eliminated and the test is repeated. Ultimately, either both c_1 and c_k match the text and so do those in between (because the candidates are all compatible and agree on the area of overlap) or all the candidates are eliminated.

A complication arises when S is a long text segment that does not start or end within B, but passes through B. Candidates in B actually match S if S is a stripe in those candidates. In order to preserve our time bounds, we do not test S against B's candidates. Instead, we can determine if the stripe is correctly positioned by maintaining a record of the last text row matched against c_1 and c_k and testing this the next time we come back to test c_1 and c_k. (That is why we recorded the closest non-stripe row below and above each row in the pattern preprocessing.) If we ever find that we have skipped a row which is not a stripe, then all the candidates can be eliminated.

At this point, the remaining candidates match the pattern except for the possible occurrence of mismatched stripes. *If the pattern contains no row stripes, we are done.* Otherwise, we do a final test using the column compressed form of the text. For each block list, we scan down a *single* compressed text column that covers all candidate patterns in the list, matching segments as described above. For a long text column segment, we now test every candidate it covers. At this point, the remaining candidates must be occurrences of the pattern. All non-stripe rows have been verified against the row compressed form of the text and every stripe row is verified against one column of the column compressed form of the text.

7.2.2 Verifying candidates that may not be compatible

When we are verifying monotonically ordered subarrays which are not necessarily compatible, but which overlap only a constant number of other subarrays, we proceed as follows. If we are verifying R' or $\hat{R}$, we use the row compressed form of the text. We scan row by row. For each segment S in text row r, we test S against the constant number of candidates that could contain either the first or last character of S and remove any that produce a mismatch. We deal with long text segments as above, using the column compressed form of the text to identify mismatched row stripes in the candidate subpatterns. To verify C' or $\hat{C}$, we proceed in an analagous way, except we use the column compressed form of the text to do the initial scan of segments and the row compressed form to identify mismatched column stripes in the candidate subpatterns.

Theorem 12 The verification phase takes time $O(|c(T)|)$.

Proof: Each text row segment needs to check at most $2(d+1)^2$ blocks (d =constant) with candidates containing the segment. Candidates are removed from consideration in the top down and bottom up scans so only candidates that could contain S are examined. Examination of a candidate is either successful, ending the processing for that segment, or causes the candidate to be eliminated. The number of tests for the row segments is thus bounded by the number of row segments and the number of candidates. The number of tests by the column segments is also bounded by the number of segments and the number of candidates. All tests are bounded by $O(|c(T)|)$, thus this phase takes $O(2(d+1)^2 \cdot |c(T)|) = O(|c(T)|)$. ∎

8 Summary

We have described the compressed matching problem for two-dimensional run-length compression and given an algorithm that solves this problem (for all but trivial patterns) in time $O(|c(T)|+|P|)$. This is the first optimal multidimensional compressed matching algorithm. It is our hope that this modest start will encourage further study of one and multidimensional compressed matching algorithms. Such a study should find efficient search techniques for known compressions as well as stimulate the definition of interesting new compressions that enable efficient compressed search.

References

[1] A. Amir and G. Benson. Efficient two-dimensional compressed matching. *Proceedings Data Compression Conference*, pages 279–288, 1992.

[2] A. Amir and G. Benson. Two-dimensional periodicity and its application. *Proc. of the Third Ann. ACM-SIAM Symp. on Discrete Algorithms*, Jan 1992.

[3] A. Amir, G. Benson, and M. Farach. Alphabet independent two-dimensional matching. *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 59–68, 1992.

[4] A. Amir, G. Benson, and M. Farach. Optimal parallel two dimensional text searching on a crew pram. *Proc. of the Fifth Ann. ACM Symp. on Parallel Algorithms and Architectures*, 1993.

[5] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in z-compressed files. *Proc. of the Fifth Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1994.

[6] A. Amir, G. Benson, and M. Farach. Optimal two-dimensional compressed matching. *Proc. of 21st International Colloquium on Automata, Languages and Programming*, 1994.

[7] D. Breslauer and Z. Galil. An optimal $o(\log\log n)$ time parallel string matching algorithm. *SIAM J. Comput.*, 19:1051–1058, 1990.

[8] M. Brown and R. Tarjan. Design and analysis of a data structure for representing sorted lists. *SIAM Journal on Computing*, 9:594–614, 1980.

[9] M. Crochemore, Z. Galil, L. Gasieniec, S. Muthukrishnan, K. Park, H. Ramesh, and W. Rytter. Constant time optimal parallel algorithm for 2d pattern matching. 1993.

[10] Z. Galil and K. Park. Truly alphabet independent two-dimensional pattern matching. *Proc. of the 33rd IEEE Annual Symp. on Foundation of Computer Science*, 1992.

[11] M. Karpinski and W. Rytter. Alphabet independent optimal parallel search for three dimensional patterns. *Proc. 5th Symp. on Combinatorial Pattern Matching*, 1994.

[12] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.

[13] M. Régnier and L. Rostami. A unifying look at d-dimensional periodicities and spacec coverings. *Proc. 4th Symp. on Combinatorial Pattern Matching*, 1993.

[14] U. Vishkin. Optimal parallel pattern matching in strings. *Proc. 12th ICALP*, pages 91–113, 1985.