The Truth, the Whole Truth, and Nothing but the Truth: Alphabet Independent Two Dimensional Witness Table Construction

Amihood Amir* — Georgia Tech
Gary Benson† — U. of Southern California
Martin Farach‡ — DIMACS

GIT–CC–92/52

August 1992

Abstract

The new technique of two dimensional periodicity has been useful in many recent two dimensional matching results. A key step in using periodicity is computing the witness table. In this paper, an optimal, linear time, alphabet independent, two dimensional witness table construction algorithm is presented.

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.

†Department of Mathematics, University of Southern California, DRB 155, 1024 W. 36th Pl., Los Angeles, CA 90089-1113; (213) 740-2404; gbenson@hto.usc.edu; Partially supported by NSF grant IRI-90-13055.

‡DIMACS, Box 1179, Rutgers University, Piscataway, NJ 08855; (908) 932-5928; farach@dimacs.rutgers.edu; Supported by DIMACS under NSF contract STC-88-09648.

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. A lacking crucial element in this multimedia effort is the equivalent of "text string searching" in an image database.

Most of the results in multidimensional pattern matching (e.g. [9, 8, 11, 5, 6]) have been a reduction of the multidimensional problem to some string matching problem. Recently, Amir and Benson [2] introduced an **inherently two dimensional tool**, that of *two dimensional periodicity.* The two dimensional periodicity idea has been instrumental in several interesting two dimensional results. Amir, Benson and Farach [3] introduced the first alphabet independent two dimensional text scanning algorithm. Amir and Benson [1] gave an algorithm that searches for appearances of a two dimensional pattern in a compressed text. Amir, Benson, and Farach [4] gave the first optimal parallel CREW two dimensional text scanning algorithm. This was followed by Muthukrishnan and Ramesh [13] and, independently, Crochemore, Gasieniec, and Rytter [10] who showed a constant time optimal parallel CRCW two dimensional text scanning algorithm.

The periodicity analysis in all its different variations uses a *witness table* (it should be remarked that not all two dimensional algorithms use witness tables. See for example the recent Baeza-Yates and Régnier result [7]). The idea of a witness table was first introduced for strings by Vishkin [15]. The witness table indicates, for every possible translation of the pattern, whether the overlap of the pattern with the translation matches or not. If it does not match, the table gives a location that witnesses a mismatch in the overlap. In [2], a $O(m^2 \log \sigma)$ algorithm was presented for witness table construction, where m^2 is the size of the pattern, and $\sigma = \min(m, |\Sigma|)$ ($|\Sigma|$ is the alphabet size). In [14], an alphabet independent witness table construction was presented.

The **main contribution** of this paper is a **new alphabet independent** algorithm for two dimensional witness table construction. We compute the witness table of an $m \times m$ pattern in time $O(m^2)$ regardless of the alphabet size. The advantages of our algorithm are that it is **conceptually simple** and seems to be **scalable to higher dimensions**.

This paper is constructed as follows. In section 2, we give a brief overview of two dimensional periodicity. In section 3, we give an outline of the algorithm. In section 4 we give the details of the lattice periodic case. Section 5 handles the radiant periodic case.

2 Preliminaries

In [2], periodicity in two-dimensional arrays is defined. The pattern is divided into four quadrants. Any location in a quadrant where a second copy of the pattern could originate is termed a *source*. In quadrant I (upper left) for example, the origin or upper left corner is a source. If other sources exist in quadrant I, then a periodicity vector is defined from the origin to the nearest such source. Once the lengths of the periodic vectors (or *basis vectors*) in quadrants I and II (lower left) have been determined, the periodicity class of the pattern is known. The basis vectors for quadrants I and II are, respectively, $\vec{v}_1 = r_1\vec{y} + c_1\vec{x}$ and $\vec{v}_2 = r_2\vec{y} + c_2\vec{x}$, 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. It was shown that there are four classes of periodicity for rectangular arrays. We summarize below the periodicty classes.

For the sake of a smooth exposition, we assume a square pattern. All results in this paper generalize to rectangular arrays. Let $A[0..m-1, 0..m-1]$ be an $m \times m$ square *pattern* array. Consider any $\frac{m}{2} \times \frac{m}{2}$ block B of *text*. The periodicity class of A determines the number and arrangement of copies of A that could originate in B:

1. *Non-Periodic.* Only one copy of A can originate in B.

2. *Lattice Periodic.* Multiple copies of A can originate in B, but identical elements in the copies must lie on the *nodes of a lattice* determined by A.

3. *Line Periodic.* Multiple copies of A can originate in B, but identical elements in the copies must lie on a *single line* determined by A.

4. *Radiant Periodic.* Similar to line periodic, except that identical elements are *monotonic* meaning that they can be ordered so that the row and column indices are both non-decreasing. Intuitively, monotonic sources can form a jagged line in B.

3 Overview

We want to fill out an array $Witness[-m-1..m-1, 0..m-1]$. For each location $A[i,j]$ that is a quadrant I source, $Witness[i,j] = [m,m]$. Otherwise, $Witness[i,j] = [r,c]$ where $[r,c]$ identifies some mismatch. Specifically $A[r,c] \neq A[i+r, j+c]$. For each location $A[i,j]$ that is a quadrant II source, $Witness[i-(m-1), j] = [m,m]$, otherwise $Witness[i-(m-1), j] = [r,c]$ where $A[r,c] \neq A[i-(m-1)+r, j+c]$.

We will construct the witness table for the first quadrant. This suffices for all known applications since resulting sources that do not overlap in the first quadrant will only overlap in at most half of their locations. Thus the slowdown resulting in only knowing the witnesses in the first quadrant is no more than a constant multiple (9).

The general idea of our algorithm is the following: We can find the witness table in linear time if our alphabet is bounded, say $\Sigma = \{0,1\}$. This can be done using suffix trees (as in [2]). We will use a bounded alphabet witness table algorithm as a building block of a general alphabet witness table construction.

Algorithm Overview

1. Convert the pattern P into pattern P' as follows:

$$P'[i,j] = \begin{cases} 1, & \text{if } P[i,j] = P[0,0]; \\ 0, & \text{if } P[i,j] \neq P[0,0]. \end{cases}$$

2. Find the witness table for P'. By [2], P' may be of one of four forms: non periodic, line periodic, lattice periodic, and radiant periodic.

   (a) **Non periodic:** We are done. The witness table for P' is good for P.

(b) **Line periodic:** Using ideas similar to Main-Lorentz [12], it was shown in [2] that the witness table for strings over general alphabets can be constructed in linear time. The line periodic two dimensional pattern can be treated very similarly to a string, since all sources are on a one dimensional line.

(c) **Lattice Periodic:** The main idea here is to treat every lattice cell as a symbol. For simplicity's sake let us assume that the lattice is orthogonal, i.e. every lattice cell is a rectangle. Considering every cell as a symbol, we get a new pattern $P^{(1)}$ whose size is at most half the size of P (unless every lattice cell is of size 1×1, in which case we are done).

Construct $P^{(1)'}$ from $P^{(1)}$ in a similar manner to the construction of P' from P., and recurse step 2 (find the witness table for $P^{(1)'}$). This eliminates many internal sources and we are left with a non periodic or line periodic case (that we know how to handle) or a lattice that is at most **half** the size of $P^{(1)}$.

We will show that the time for each level of the recursion is proportional to the number of cells in the lattice. The total algorithm time is then at most $\sum_{i=0}^{\log m} m^2/2^i = O(m^2)$, independent of the alphabet.

(d) **Radiant Periodic:** The idea here is simple. In the radiant periodic case all the sources are also placed on a lattice. The only difference is that not all lattice points are sources. Thus there is a motivation to treat this case as the lattice periodic case. The only trouble is that we need to assure a recursion. This is done by the following observation. The lattice upon which all sources lie can be determined by the three "closest" source locations. These are found at the rightmost bottom corner. Once this lattice is identified and the case is treated like a lattice periodic case, we may have a situation where the "dense" part of the lattice, i.e. the one near the right bottom, is complete. All sources there are verified. But this may not be the case in other locations. The monotonicity of the lattice density means that we can now ignore the verified locations, and handle the sources in the next sparser part of the lattice.

4 The Lattice Periodic Case:

We begin by analysing the orthogonal grid.The additional issues pertaining to a general lattice will be discussed later. We provide the details of creating $P^{(1)'}$ from $P^{(1)}$, and then $P^{(i)'}$ from $P^{(i)}$.

4.1 The Base Case - Orthogonal Lattice

Assume we created P' and found its witness table. Assume also that the result is an $m_1 \times m_1$ orthogonal grid. Our problem is that the upper left elements in each lattice rectangle do not have witnesses. They may or may not be sources of a pattern appearance, depending on the symbols different from $P[0,0]$, that were all labeled 0. We need to find witnesses for these elements. We will show that in time proportional to m_1^2 these witnesses can be found for at least $\frac{m_1^2}{2}$ of the grid points. Let the *top* of a rectangle be its upper left element.

Lattice Periodic Witness Construction Algorithm

Step 1 Scan all the grid rectangles and create a matrix $P^{(1')}$ where

$$P^{(1')} = \begin{cases} 1, & \text{if rectangle [i,j] equals the top rectangle;} \\ 0, & \text{otherwise.} \end{cases}$$

Implementation: We scan the rectangles row by row, and column by column (both orientations are necessary in order to handle edges). If an equality is achieved we designate the rectangle with a "1". If there is no equality, the cell is designated with a "0". In addition, it is marked with the pairs $\langle 1, j\rangle$ and $\langle 1, k\rangle$. The "1" designates the 1st level in the recursion, j is the first element where this cell is **not equal** to the top cell, in a row-major scan, and k is the first unequal element in a column-major scan.

Time: At most $O(m^2)$, if all cells are equal. In that case, we are done. If not all cells are equal we only check the parts of the cells that equal the top cell (and remember them). These parts will never be checked again.

Space: Every one of the m_1 cells uses a constant space.

Step 2 Find the witness table of $P^{(1')}$.

Implementation: Wherever a lattice point is eliminated, it is because a "1" was expected and a "0" found, or vice versa. The pair $\langle 1, j\rangle$ of the "0" involved provides information on the exact witness location in P.

Edge Blocks: Special care should be given to the last cell in each row as well as the last row of cells. These are called *edge cells*. These cells may be "broken", i.e. only part of them is in the pattern. For some edge cells this is not a problem, because the inequality with the top may be within the available part. These edge blocks could be assigned a "0". However, there may be edge blocks that agree with the top block, but may differ from it in the broken off part. These cells can be assigned neither a "1" nor a "0". Thus, each comparison in the witness table construction of $P^{(1')}$ (this may be a suffix tree LCA query as in [2]), will only be applied to unbroken cells. Each time a suffix tree query is executed (or an AC match is found) it will be followed by a constant time comparison of the edge cell and the next expected cell in the substring that was queried. For example, suppose we are checking if there is a source in location i of the row, we do a suffix tree query followed by comparing the edge cell with block $m_1 - i + 1$. The comparison is done in constant time as follows:

If the pattern cell (cell $m_1 - i + 1$ in the example) is a "0" then we check its column-major pair $\langle 1, k\rangle$. If k is outside the bounds of the edge cell, this is a match. If the pattern cell is a "1" and the edge cell agrees with the top cell everywhere within its bounds, then it is a match. All other cases are a mismatch and create a witness.

The last row is handled similarly, but with a witness table construction of the column-major orientation of $P^{(1')}$. This will double the work, but all edge cells will be properly handled.

4.2 Recursion - Orthogonal Lattice

Assume we have a lattice $P^{(i)}$, of size $m_i \times m_i$ at the i-th level of the recursion.

Data Structure: Every cell is composed of k_i level-$(i-1)$ cells. We need to know, for each cell, if it equals the top cell or not. If it is equal it is designated with a "1". If it is not equal, it is designated with a "0" and is attached an i-tuple $\langle i, k_{i-1}, k_{i-2}, \ldots, k_0\rangle$ where $\langle i, k_{i-1}, \ldots, k_0\rangle$ indicates the location where the cell differs from the top cell. The meaning is as follows:

There is an error at level-$(i-1)$ cell k_{i-1}. The error there is in its level-$(i-2)$ cell k_{i-2}, etc., down to level-0 cell (element of P) k_0.

Even if we keep all $\log m$ levels of the data structure its total size is

$$\sum_{i=0}^{\log m} \frac{im^2}{2^i} = O(m^2).$$

The Algorithm

Step 1 Create matrix $P^{(i')}$.

Implementation: Similar to creating $P^{(1')}$. The only difference is equality comparison. We compare each level-i cell to the level-i top cell block-by-block. Each block is a level-$(i-1)$ cell.
Cases:

1. The $(i-1)$-tuples are different. This implies inequality.
2. The level-$(i-1)$ cells are both designated "1". This implies equality.
3. The $(i-1)$-tuples are equal. This means that both are different from the level-$(i-1)$ top cell at the same location, but we do not know their relationship *to each other*. In this case, we compare the elements in P (level-0) where they differ from the level-$(i-1)$ top cell. If they are not equal then this implies inequality. If they are equal we keep checking element by element until either an inequality is found or block equality is established.

Time: As long as equality is checked, the comparison is charged to a lower level. Note that our counters move only **forward**, never back. Thus the only charge to level-$(i-1)$ (remember that checking equality of blocks in level-i is always charged to level-$(i-1)$) is checking the $(i-1)$-tuple, i.e. level-$(i-1)$ is scanned $O(i-1)$ times. The total algorithm time is then no more than

$$c\sum_{i=0}^{\log m} \frac{im^2}{2^i} = O(m^2).$$

Step 2 Find the witness table of $P^{(i')}$.

4.3 General Lattices

Our discussion so far assumed that the lattice produced an orthogonal grid. In, general, this is clearly not the case. We now show how to reduce the general case to the orthogonal case.

Every stage in the orthogonal grid algorithm had two phases:

1. The *cell verification phase*, where each cell was checked for equality with the top cell.

2. The *lattice contraction phase*, where the lattice over $\{0,1\}$ was prunned to a sparser lattice.

4.3.1 Cell Verification Phase

The key idea here is that while the lattice may not be orthogonal, the underlying pattern is (it is a matrix). We will therefore consider "almost" rectangles that correspond to the lattice cells, and cover the lattice.

Consider the cell whose upper left corner has coordinates $[x_1, y_1]$ and the coordinates of the remaining corners, in clockwise order, are: $[x_2, y_2], [x_3, y_3]$, and $[x_4, y_4]$. Note that vector $\overline{[x_1, y_1][x_2, y_2]}$ is the quadrant IV vector described in [2] and $\overline{[x_1, y_1][x_4, y_4]}$ is the quadrant I vector.

The "almost grid" we construct is roughly the tiling we get by drawing a horizontal line to the right of each lattice point and a vertical line down from each lattice point until we hit the horizontal or vertical line emanating from a neighboring lattice point. Formally, let

$i = y_2 - y_1 - 1$
$j = x_4 - x_2 - 1$
$k = x_1 - x_2 - 1$
$l = y_2 - y_4$ (this may be 0 if $y_2 = y_4$ or negative if $y_2 < y_4$).

The corners of the new cell are (clockwise from top left): $[x_1, y_1], [x_1, y_1 + i], [x_1 + j - k, y_1 + i], [x_1 + j - k, y_1 + i - l]$ (this is lattice point $[x_4, y_4]$), $[x_1 + j, y_1 + i - l], [x_1 + j, y_1]$ (and then back to $[x_1, y_1]$). This is a rectangle with a "bite" taken out of it either on the top right or the bottom right, by the nighboring rectangle.

We now compare each cell to the top cell, in a similar fashion to the orthogonal case. If they all match and the borders are consistent with what is expected, we are done. If not, the result is a lattice, with a 1 or 0 at every point.

4.3.2 Lattice Contraction Phase

As in the orthogonal case, the contraction is done recursively. We can "rotate" the 0-1 lattice in a manner that results in an orthogonal lattice. This rotation is accomplished by transforming the axes so that the vectors defining the lattice are the x and y axes.

We now proceed in a manner similar to the orthogonal case. There are two differences:

1. Our grid is now a diamond rather than a rectangle. The first row is the smallest, the rows extend in both directions as we descend in the diamond, and at some point their length begins to diminish.

2. There may now be more than one edge cell per row.

Solutions:

1. We divide the diamond into four right triangles and look for appearances of each of the triangles separately. A KMP-like algorithm using suffix trees will work because the rows in each triangle are *increasing* in lenght, preserving the necessary transitivity in KMP.

2. We ignore the broken blocks and do the lattice contraction phase only on complete blocks.

   Cases:

   (a) We achieve a sparser lattice. In this case continue with the recursion. The base cases of the recursion will be discussed next.

   (b) All unbroken blocks are sources, i.e. none of them has a witness following the lattice contraction phase. This case is almost solved. The only problem now may be having witnesses in the edge cells. However, in this case the edge cells *must* all equal the top cell. Any difference is a witness for all sources to the left and higher than the witness.

   (c) We get a radiant case. The radiant case is discussed in the next section.

The recursion can now proceed similarly to the orthogonal case, merging quasi-rectangles after every lattice contraction. The only necessary addition is handling the base case.

The Base Case:
The lattice contraction can proceed as long as there is still a lattice of unbroken cells (at least two rows). The base case is when there is only one row left with unbroken cells. There are two possibilities for such a situation:

- The row of unbroken cells extends from the top left and slanting in the direction of the bottom right of the first quadrant. This is the line periodic case and can be solved in time $O(n^2)$ as described in section 3.

- The row of unbroken cells extends from the bottom left of the first quadrant and angles in the direction of the top right of the first quadrant.

The problem with the second case is that there is not an extra row of unbroken cells for comparison with the first such row. Thus it seems like we will not be able to contract any further. However, now we have at most $O(m)$ positions left without witnesses and we can use more powerful methods. The case is handled recursively, in a similar fashion to the general lattice case, but taking into account the fact that the lattice has only one row of unbroken cells.

Algorithm for a one unbroken row lattice

Initialization:

Orthogonalize the blocks in the manner discussed in section 4.3.1.

Recursive Part:

1 *Cell Verification*

1.1 Compare all blocks with the top cell row by row, starting from the bottom left of each cell and moving to the right (and up). Record the row where the first mismatch occurs. (There will only be a single cell per row which is broken vertically as well as horizontally. All others will be broken only horizontally (i.e. have missing rows on top). This is due to the fact that there is only one unbroken row.)

1.2 Compare all blocks with the top cell row by row, starting from the top left of each cell and moving to the right (and down). Record the row where the first mismatch occurs.

2 *Lattice Contraction*

Do a $\leq$-matching [5] of the first row (all cells but one broken) and the second row (except for the vertically broken cell).

3 *Recursion*

Similar to the recursive stage of a general lattice.

End Algorithm

Time: The only difference between this case and the general lattice case is the lattice contraction. By [5] this can be done in time $O(m\sqrt{m}\log m)$. Since there are no more than $O(\log m)$ possible recursive levels, the total time can not exceed $O(m^2)$.

5 Radiant Periodic Case:

We reduce the radiant periodic case to the lattice periodic case. The key observation that allows this reduction is the following lemma.

Lemma: The density of sources in a radiant periodic pattern increases from least dense at the top left corner, to most dense at the bottom right.

Proof: Follows immediately from the fact that the sources in the radiant periodic case are all linear combinations of the quadrant I vectors. $\square$

When encountering the radiant periodic case, we generate a lattice that agrees with the densest part of the radiant sources. Call this lattice the *dense lattice*. This is done by identifying as the quadrant I and II spanning vectors, the vectors defined by the closest three sources at the bottom right.

Since we now have a lattice, we proceed with the cell verification and lattice contraction phases. The only problem may be if the cell verification phase verifies the dense edges of the dense lattice as equal to the top cell, but the other cells do not all verify. In this case the contraction is done in a simpler way. We generate a new lattice that is defined by the closest three sources to the verified part of the current dense lattice.

Reduction of radiant case to lattice case

1 Identify the quadrant I and II vectors that generate the dense latice, and construct the dense lattice.

2 Perform cell verification as described in section 4

3 If not all cells in the densest part are equal to the top cell, then perform lattice contraction.

4 If all cells in the densest part are equal to the top cell, then thin the lattice by identifying the quadrant I and II vectors of the three closest radiant sources that are not on the verified edge

End Algorithm

Time: For each phase, the reduction time is proportional to the number of elements in the dense lattice. Therefore the total algorithm time is the same as the lattice periodic case, $O(m^2)$.

References

[1] A. Amir and G. Benson. Efficient two dimensional compressed matching. *Proc. of Data Compression Conference, Snow Bird, Utah*, pages 279–288, Mar 1992.

[2] A. Amir and G. Benson. Two-dimensional periodicity and its application. *Proc. of 3rd Symposium on Discrete Algorithms, Orlando, FL*, pages 440–452, Jan 1992.

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

[4] A. Amir, G. Benson, and M. Farach. Optimal parallel two dimensional pattern matching. *Proc. of the 5th ACM Symp. on Parallel Algorithms and Architectures*, 1993.

[5] 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.

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

[7] R. Baeza-Yates and M. Régnier. Fast algorithms for two dimensional and multiple pattern matching algorithms. *Proc. SWAT 90*, pages 332–347, 1990.

[8] T.J. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Comp*, 7:533–541, 1978.

[9] R.S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977.

[10] M. Crochemore, L. Gasieniec, and W. Rytter. Constant time optimal parallel algorithm for 2d pattern matching. Manuscript, 1992.

[11] G. H. Gonnet. Efficient searching of text and pictures. *University of Waterloo Center for the New OED, Technical Report OED-88-02*, 1988.

[12] M.G. Main and R.J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *J. of Algorithms*, pages 422–432, 1984.

[13] S. Muthukrishnan and H. Ramesh. A constant time optimal parallel algorithm for two dimensional matching. Manuscript, 1992.

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

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