Two-Dimensional Periodicity in Rectangular Arrays*

Amihood Amir†
College of Computing
Georgia Institute of Technology
amir@cc.gatech.edu

Gary Benson†
Department of Biomathematical Sciences
Mount Sinai School of Medicine
benson@ecology.biomath.mssm.edu

Abstract

String matching is rich with a variety of algorithmic tools. In contrast, multidimensional matching has had a rather sparse set of techniques. This paper presents a new algorithmic technique for two-dimensional matching: *periodicity analysis*. Its strength appears to lie in the fact that it is inherently two-dimensional.

Periodicity in strings has been used to solve string matching problems. Multidimensional periodicity, though, is not as simple as it is in strings and was not formally studied or used in pattern matching. In this paper, we define and analyze two-dimensional periodicity in rectangular arrays. One definition of string periodicity is that a periodic string can self-overlap in a particular way. An analogous concept is true in two dimensions. The *self overlap vectors* of a rectangle generate a regular pattern of locations where the rectangle may originate. Based on this regularity, we define four categories of periodic arrays: *non-periodic, lattice-periodic, line-periodic and radiant-periodic* and prove theorems about the properties of the classes.

We give serial and parallel algorithms that find all locations where an overlap originates. In addition, our algorithms find a *witness* proving that the array does not self overlap in any other location. The serial algorithm runs in time $O(m^2)$ (linear time) when the alphabet size is finite, and $O(m^2 \log m)$ otherwise. The parallel algorithm runs in time $O(\log m)$ using $O(m^2)$ CRCW processors.

*A preliminary version of this paper was presented at the 3rd ACM-SIAM Symposium on Discrete Algorithms, 1992

†Partially supported by NSF grant IRI-9013055.

1 Introduction

String matching is a field rich with a variety of algorithmic ideas. The early string matching algorithms were mostly based on constructing a pattern automaton and subsequently using it to find all pattern appearances in a given text ([KMP-77, AC-75, BM-77]). Recently developed algorithms [G-85, V-85, V-91] use periodicity in strings to solve this classic string matching problem. Lately, there has been interest in various two-dimensional approximate matching problems, largely motivated by low-level image processing ([KS-87, AL-91, AF-91, ALV-90]). Unlike string matching, the methods for solving multidimensional matching problems are scant. This paper adds a *new algorithmic tool* to the rather empty tool chest of multidimensional matching techniques: two-dimensional periodicity analysis.

String periodicity is an intuitively clear concept and the properties of a string period are simple and well understood. Two-dimensional periodicity, though, presents some difficulties. Periodicity in the *plane* is easy to define. However, we seek the period of a *finite rectangle*. We have chosen to concentrate on a periodicity definition that implies the ability for *self-overlap*. In strings such an overlap allows definition of a smallest period whose concatenation produces the entire string. The main contribution of this paper is showing that for rectangles also, the overlap produces a "smallest unit" and a regular pattern in which it appears in the array. The main differences are that this "smallest unit" is a vector rather than a sub-block of the array, and that the pattern is not a simple concatenation. Rather, based on the patterns of vectors that can occur, there are four categories of array periodicity: *non-periodic, line periodic, radiant periodic* and *lattice periodic*. As in string matching this regularity can be exploited.

The strength of periodicity analysis appears to lie in the fact that it is inherently a two-dimensional technique whereas most previous work on two-dimensional matching has reduced the matrix problem to a problem on strings and then applied one-dimensional string matching methods. The two dimensional periodicity analysis has already proven useful in solving several multi-dimensional matching problems [ABF-94a, ABF-93, ABF-94b, KR-94]. We illustrate with two examples.

The original motivation for this work was our research in image preserving compression. We wanted to solve the following problem: Given a two-dimensional pattern P and a two-dimensional text T which has been compressed, find all occurrences of P in T without decompressing the text. The goal is a sublinear algorithm with respect to the size of the original *uncompressed* text. Some initial success in this problem was achieved in [ALV-90], but their algorithm, being automaton based, seems to require a large amount of decompression. In [AB-92b, ABF-94b], we used periodicity to find the first *optimal* pattern matching algorithm for compressed two-dimensional texts.

Another application is the two-dimensional exact matching problem. Here the text is not compressed. Baker [B-78] and, independently, Bird [Bi-77] used the Aho and Corasick [AC-75] dictionary matching algorithm to obtain a $O(n^2 \log|\Sigma|)$ algorithm for this problem. This algorithm is automaton based and therefore the running time of the text scanning phase is dependent on the size of the alphabet. In [ABF-94a] we used periodicity analysis to produce the first two dimensional exact matching algorithm with a linear time *alphabet independent* text scanning phase.

Since the work presented here first appeared [AB-92a], the analysis of radiant periodic patterns has been strengthened [GP-92, RR-93], and periodicity analysis has additionally proven useful in providing optimal parallel two dimensional matching algorithms [ABF-93, CCG+93], as well as in solving a three dimensional matching problem [KR-94].

This paper is organized as follows. In Section 2, we review periodicity in strings and extend this notion to two dimensions. In Section 3, we give formal definitions, describe the classification scheme for the four types of two-dimensional periodicity, and prove some theorems about the properties of the classes. In Section 4 we present serial and parallel algorithms for detecting the type of periodicity in an array. The complexity of the serial algorithm is $O(m^2)$ (linear time) when the alphabet size is finite, and $O(m^2 \log m)$ otherwise. The parallel algorithm runs in time $O(\log m)$ with $O(m^2)$ CRCW processors. In addition to knowing where an array can self overlap, knowing where it can not and why is also useful. If an overlap is not possible, then the overlap produces some mismatch. Our algorithms find a single mismatch location or *witness* for each self overlap that fails.

2 Periodicity in strings and arrays

In a periodic string, a smallest period can be found whose concatenation generates the entire string. In two dimensions, if an array were to extend infinitely so as to cover the plane, the one-dimensional notion of a period could be generalized to a unit cell of a lattice. But, a rectangular array is not infinite and may cut a unit cell in many different ways at its edges.

Instead of defining two-dimensional periodicity on the basis of some subunit of the array, we instead use the idea of *self-overlap*. This idea applies also to strings. A string w is *periodic* if the longest prefix p of w that is also a suffix of w is at least half the length of w. For example, if $w = abcabcabcabcab$, then $p = abcabcab$ and since p is over half as long as w, w is periodic. This definition implies that w may overlap itself starting in the fourth position.

The preceding idea easily generalized to two dimensions as illustrated in figure 1.

Definition 1 Let A be a two-dimensional array. Call a *prefix* of A a rectangular subarray that contains one corner of A. (In the figure, the upper left corner.) Call a *suffix* of A a rectangular subbarray that contains the diagonally opposite corner of A (In the figure, the lower right corner). We say A is *periodic* if the largest prefix that is also a suffix has dimensions at least as large as some fixed percentage d of the dimensions of A.

In the figure, if $d \leq \frac{5}{6}$, then A is periodic. As with strings, if A is periodic, then A may overlap itself if the prefix of one copy of A is aligned with the suffix of a second copy of A. Notice that both the upper left and lower left corners of A can define prefixes, giving A two directions in which it can be periodic. As we will describe in the next section, the classification of periodicity type for A is based on whether it is periodic in either or both of these directions.

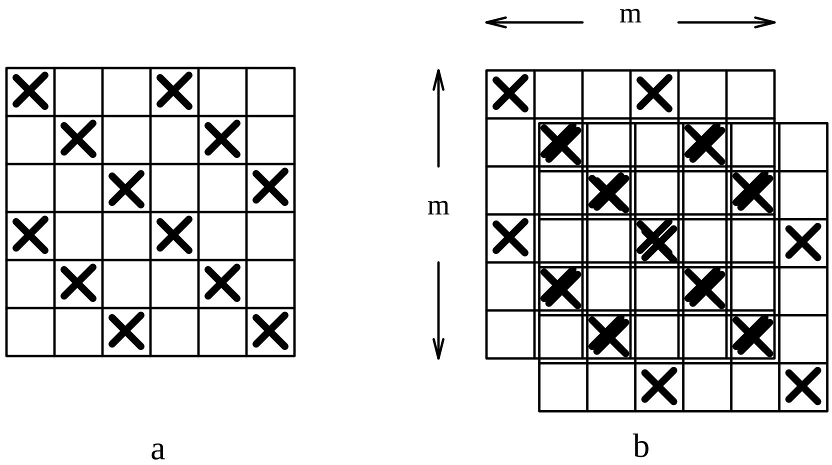

Figure 1:
a) A periodic pattern. b) A suffix matches a prefix.

3 Classifying arrays

Our goal here is classifying an array A into one of four periodicity classes. For clarity of presentation we concentrate on square arrays. We later show how to generalize all results to rectangles. We begin with some definitions of two-dimensional periodicity and related concepts (figure 2).

Definition 2 Let $A[0 \ldots m-1, 0 \ldots m-1]$ be an $m \times m$ square array. Each element of A contains a symbol from an alphabet Σ. A subarray of A is called a *block*. Blocks are designated by their first and last row and column. Thus, the block $A[0..m-1, 0..m-1]$ is the entire array. Each corner of A defines a *quadrant*. Quadrants are labeled counterclockwise from upper left, quadrants I, II, III and IV. Each quadrant has size q where $1 \leq q \leq \lceil \frac{m}{2} \rceil$. (Quadrants may share part of a row or column). Quadrant I is the block $A[0 \ldots q-1, 0 \ldots q-1]$. The choice of q may depend on the application. For this paper, $q = \lceil \frac{m}{3} \rceil$.

Definition 3 Suppose we have two copies of A, one directly on top of the other. The copies are said to be *in register* because some of the elements overlap (in this case, all the elements) and overlapping elements contain the same symbol. If the two copies can be repositioned so that $A[0,0]$ overlaps $A[r,c]$ ($r \geq 0, c > 0$) and the copies are again in register, then we say that the array is *quadrant I symmetric*, that $A[r,c]$ is a *quadrant I source* and that vector $\vec{v} = r\vec{y} + c\vec{x}$ is a *quadrant I symmetry vector*. Here, $\vec{y}$ is the vertical unit vector in the direction of increasing row index and $\vec{x}$ is the horizontal unit vector in the direction of increasing column index. If the two copies can be repositioned so that $A[m-1,0]$ overlaps $A[r,c]$ ($r < m-1, c \geq 0$) and the copies are again in register, then we say that the array is *quadrant II symmetric*, that $A[r,c]$ is a *quadrant II source* and that $\vec{v} = (r-m+1)\vec{y} + c\vec{x}$ is a *quadrant II symmetry vector*.

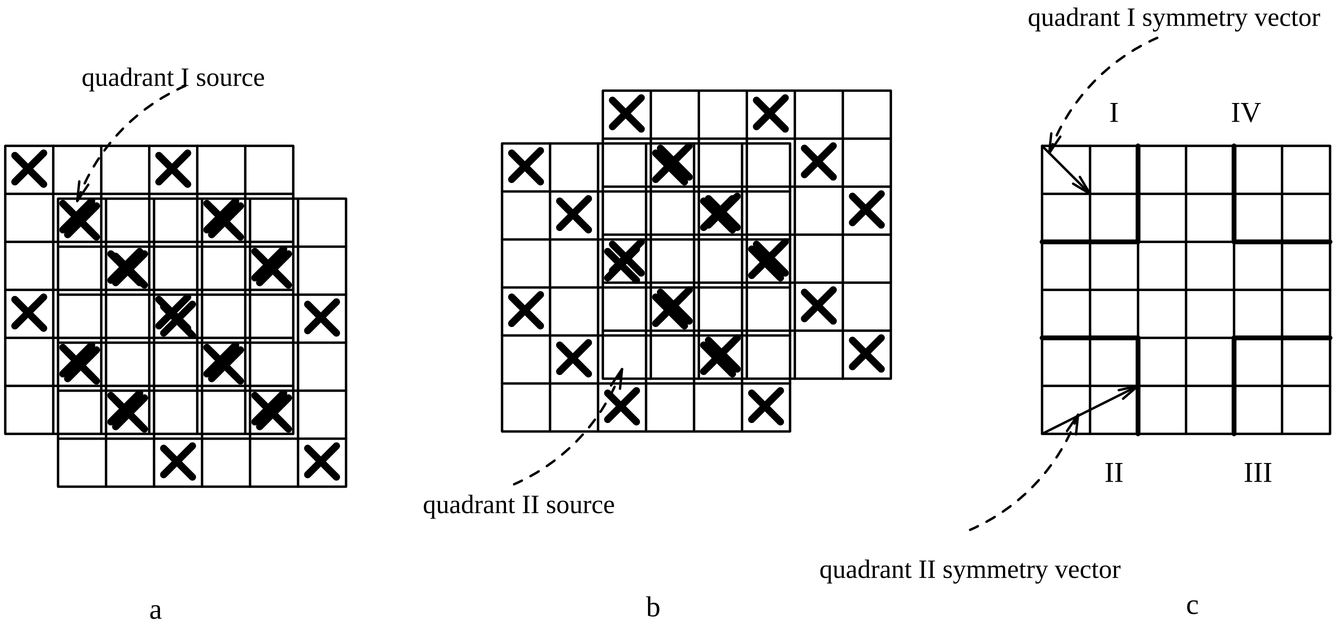

Figure 2:
Two overlapping copies of the same array.
a) A quadrant I source. b) A quadrant II source. c) The symmetry vectors.

Analagous definitions exist for quadrants III and IV, but by symmetry, if $\vec{v}$ is a quadrant $III(IV)$ symmetry vector, then $-\vec{v}$ is a quadrant $I(II)$ symmetry vector. We will usually indicate a vector $\vec{v} = r\vec{y} + c\vec{x}$ by the ordered pair (r, c). Note that symmetry vector (r, c) defines a mapping between identical elements, that is, (r, c) is a symmetry vector iff $A[i, j] = A[i + r, j + c]$ wherever both elements are defined. In particular, if (r, c) is a symmetry vector, then it maps the block $A[i..j, k..l]$ to the identical block $A[i + r..j + r, k + c..l + c]$.

In the remainder of this paper, we use the terms source and symmetry vector interchangeably.

Definition 4 The *length* of a symmetry vector is the maximum of the absolute values of its coefficients. The *shortest* quadrant I (quadrant II) vector is the smallest one in lexicographic order first by row and then by column (first by column and then by absolute value of row). The *basis vectors* for array A are the shortest quadrant I vector (r_1, c_1) (if any) and the shortest quadrant II vector (r_2, c_2) (if any). If the length of a symmetry vector is $< p$ where $p = \lceil \frac{m}{3} \rceil$ then the vector is *periodic.*

We are now ready to classify a square array A into one of four periodicity classes based on the presence or absence of periodic vectors in quadrants I and II. Following the classification we prove some theorems about the properties of the classes. In Section 4 we present algorithms for finding *all* the sources in an array.

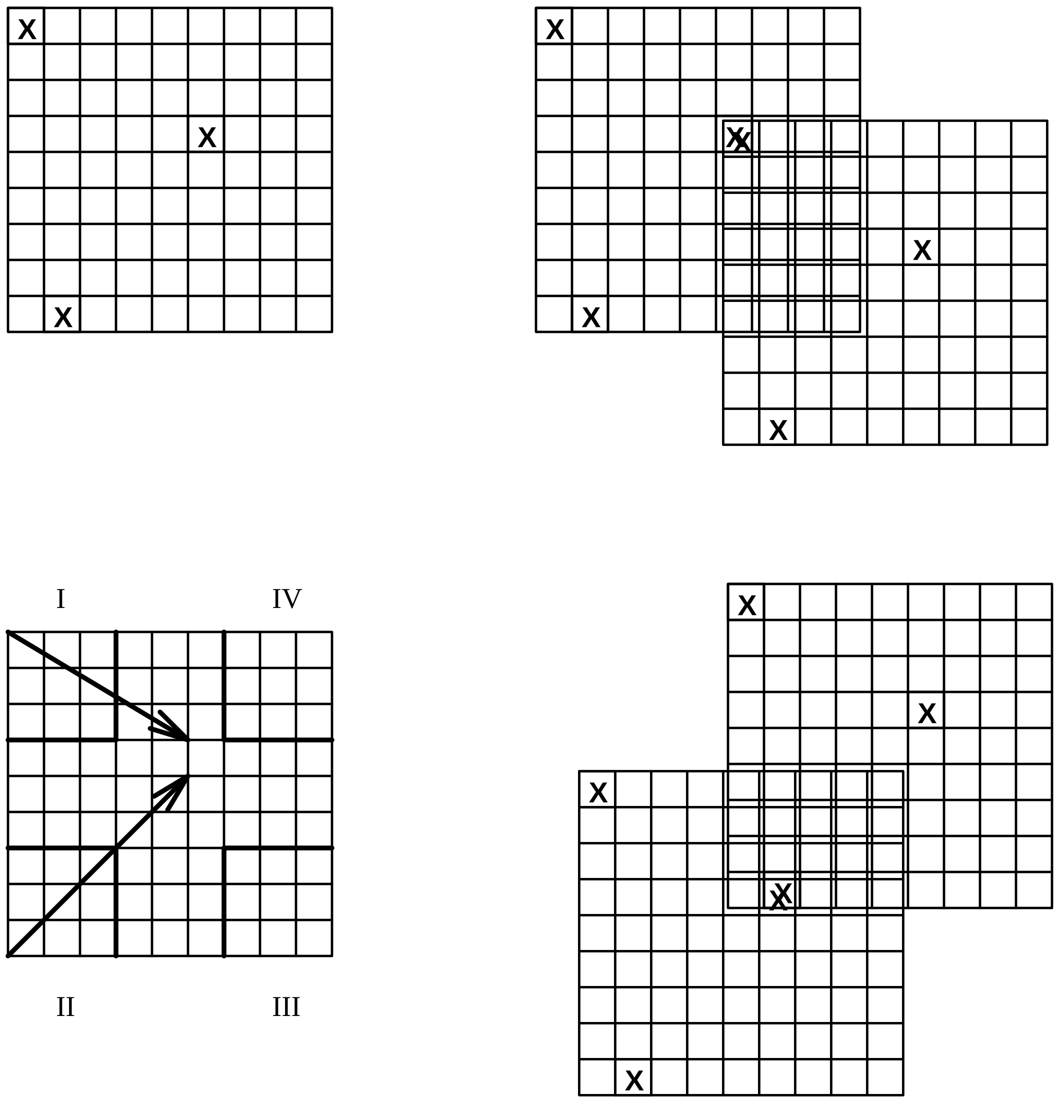

Figure 3:
Non-periodic array.

The four classes of two-dimensional periodicity are (figures 3 – 6):

- *Non-periodic* – The array has no periodic vectors.

- *Lattice periodic* – The array has periodic vectors in both quadrants. All quadrant I sources which occur in quadrant I fall on the nodes of a lattice which is defined by the basis vectors. The same is true for quadrant II sources in quadrant II. Specifically, let $\vec{v}_1 = (r_1, c_1)$ and $\vec{v}_2 = (r_2, c_2)$ be the periodic basis vectors in quadrants I and II respectively. Then, an element in quadrant I is a quadrant I source iff it occurs at index $A[ir_1 + jr_2, ic_1 + jc_2]$

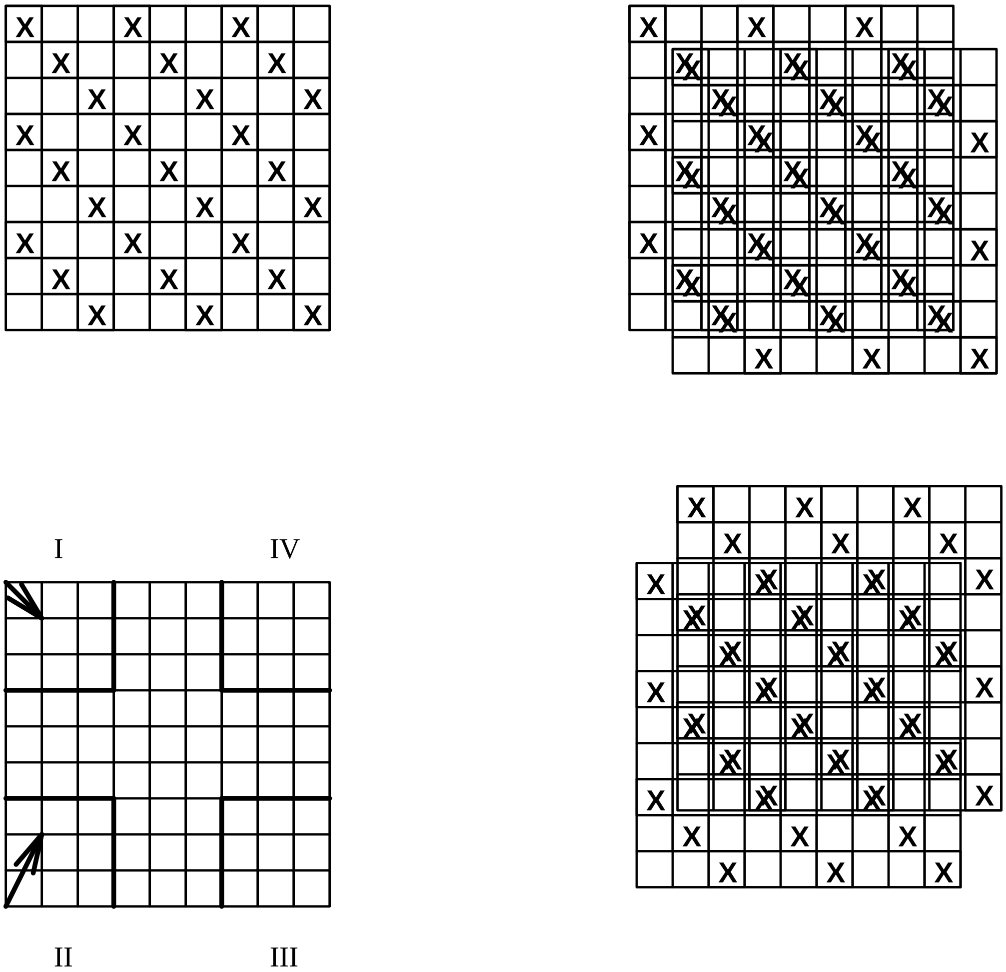

Figure 4:
Lattice periodic array.

for integers i, j. An element in quadrant II is a quadrant II source iff it occurs at index $A[m-1+\hat{\imath}r_1+\hat{\jmath}r_2, \hat{\imath}c_1+\hat{\jmath}c_2]$ for integers $\hat{\imath}, \hat{\jmath}$.

- *Line periodic* – The array has a periodic vector in only one quadrant and the sources in that quadrant all fall on one line.

- *Radiant periodic*—This category is identical to the line periodic category, except that in the quadrant with the periodic vector, the sources fall on several lines which all radiate from the quadrant's corner. We do not describe the exact location of the sources for this class, but see [GP-92] for a detailed analysis of the source locations.

Next, we prove some theorems about the properties of the classes. All the theorems are stated in terms of square arrays for clarity. At the end of the theorems we explain how they can be modified to apply to any $n \times m$ rectangular array.

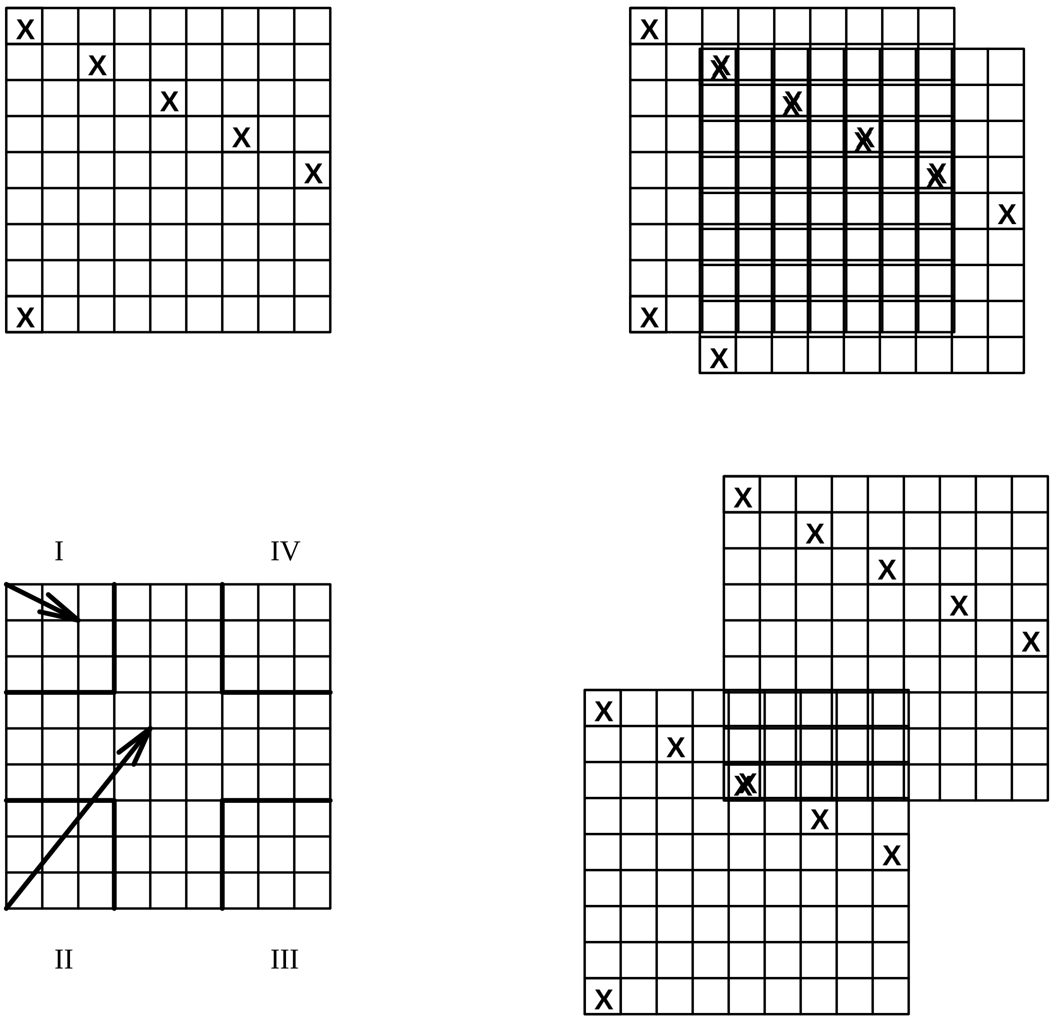

Figure 5:
Line periodic array.

In Lemmas 1-3, we establish the fact that if we have symmetry vectors for both quadrants I and II, and they meet a pair of constraints on the sum of their coefficients, then every linear combination of the vectors defines another symmetry vector.

Lemma 1 If (r_1, c_1) and (r_2, c_2) are symmetry vectors from quadrants I and II respectively, and $c_1 + c_2 < m$ and $r_1 + |r_2| < m$, then $(r_1 + r_2, c_1 + c_2)$ is either a quadrant I symmetry vector $(r_1 \geq |r_2|)$ or a quadrant II symmetry vector $(r_1 < |r_2|)$.

Proof: We prove for the case $r_1 \geq |r_2|$. The proof for the other case is similar. We show that $S = (r_1 + r_2, c_1 + c_2)$ is a quadrant I source.

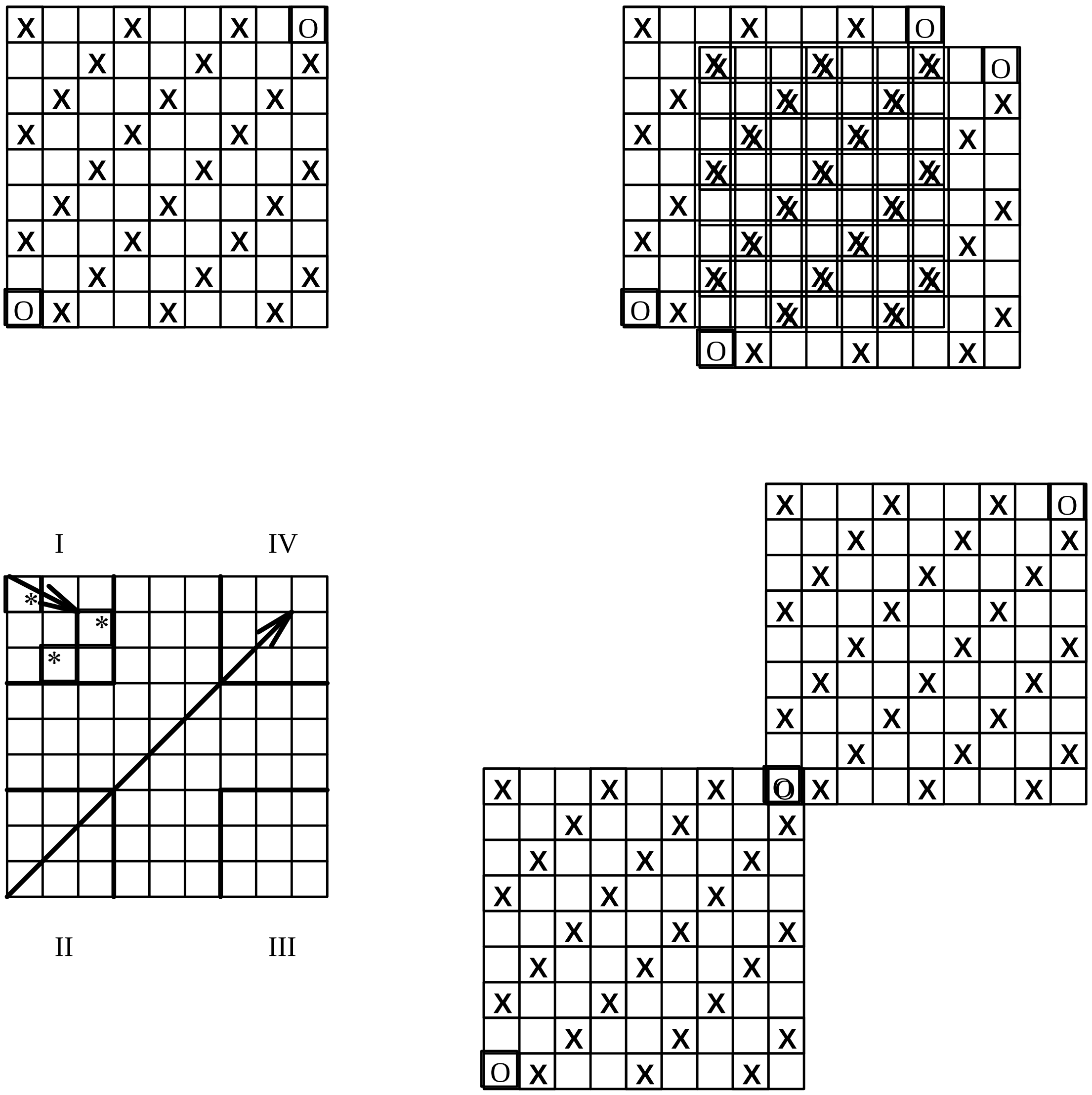

Figure 6:
Radiant periodic array. Three non-colinear sources are starred.

First, by the constraint on the c_i, the fact that r_2 is negative and the assumption that $r_1 \geq |r_2|$, S is an element of A. Next, we show via two pairs of mappings that the quadrant I prefix block $A[0..m-r_1-r_2-1, 0..m-c_1-c_2-1]$ is identical to the suffix block $A[r_1+r_2..m-1, c_1+c_2..m-1]$.

First pair: (r_1, c_1) maps block $A[0..m-r_1-1, 0..m-c_1-c_2-1]$ to block $A[r_1..m-1, c_1..m-c_2-1]$. (r_2, c_2) maps the resultant block to block $A[r_1+r_2..m+r_2-1, c_1+c_2..m-1]$.

Second pair: (r_2, c_2) maps block $A[m-r_1..m-r_1-r_2-1, 0..m-c_1-c_2-1]$ to $A[m-r_1+r_2..m-r_1-1, c_1..m-c_2-1]$. (r_1, c_1) maps the resultant block to $A[m+r_2..m-1, c_1+c_2..m-1]$. $\Box$

Lemma 2 If (r_1, c_1) and (r_2, c_2) are two quadrant k symmetry vectors ($k = I, II$) and $r_1+|r_2| < m$ and $c_1 + c_2 < m$, then $(r_1 + r_2, c_1 + c_2)$ is also a quadrant k symmetry vector.

Proof: We prove for quadrant I. The proof for the other quadrant is similar. We show that $S = (r_1 + r_2, c_1 + c_2)$ is a quadrant I source.

First, by the restraints on the r_i and the c_i, S is an element of A. Next, by a pair of mappings, we show that the quadrant I prefix block $A[0..m - r_1 - r_2 - 1, 0..m - c_1 - c_2 - 1]$ is identical to the suffix block $A[r_1 + r_2..m - 1, c_1 + c_2..m - 1]$. Recall that both r_1 and r_2 are postive.

First mapping: (r_1, c_1) maps the block $A[0..m - r_1 - r_2 - 1, 0..m - c_1 - c_2 - 1]$ to the block $A[r_1..m - r_2 - 1, c_1..m - c_2 - 1]$. Second mapping: (r_2, c_2) maps the resultant block to the block $A[r_1 + r_2..m - 1, c_1 + c_2..m - 1]$. $\square$

Lemma 3 If $\vec{v}_1 = (r_1, c_1)$ and $\vec{v}_2 = (r_2, c_2)$ are symmetry vectors from quadrants I and II respectively, and $c_1 + c_2 < m$ and $r_1 + |r_2| < m$, then for all integers i, j such that $A[ir_1 + jr_2, ic_1 + jc_2]$ is an element of A, $(ir_1 + jr_2, ic_1 + jc_2)$ is a quadrant I symmetry vector. Similarly, for all $\hat{\imath}, \hat{\jmath}$ such that $A[m-1+\hat{\imath}r_1+\hat{\jmath}r_2, \hat{\imath}c_1+\hat{\jmath}c_2]$ is an element of A, $(\hat{\imath}r_1+\hat{\jmath}r_2, \hat{\imath}c_1+\hat{\jmath}c_2)$ is a quadrant II symmetry vector.

Proof: We prove for vector $(ir_1 + jr_2, ic_1 + jc_2)$, equivalent to source $S_{i,j} = A[ir_1 + jr_2, ic_1 + jc_2]$. The proof for the other vector is similar. Consider the lattice of elements in A defined by the quadrant I and II vectors and with one element at $A[0, 0]$. (The lattice elements correspond exactly to the elements $S_{i,j}$.) Consider the line l that extends from element $A[0, 0]$ through elements $S_{i,0} = A[ir_1, ic_1]$. We prove the lemma only for those lattice elements on or to the right of l. The remaining elements are treated similarly.

Case 1: $S_{i,0}$ on line l. By induction on i. For $S_{1,0}$, $i = 1$ and (r_1, c_1) is a symmetry vector by hypothesis. Now, assume that (ir_1, ic_1) is a symmetry vector. For $S_{i,0}$ $i > 1$. Since (r_1, c_1) and (ir_1, ic_1) are both quadrant I symmetry vectors, by Lemma 2 $((i + 1)r_1, (i + 1)c_1)$ is a quadrant I symmetry vector.

Case 2: $S_{i,j}$ $j \geq 1$ to the right of line l. Elements $S_{i,j}$ fall on lines l_j which are parallel to line l. We show that the *uppermost* element $S_{i,j}$ is a source. By application of Lemma 2, as in Case 1 above, the remaining sources on l_j are established.

Consider a *cell* of the lattice with sides (r_1, c_1) and (r_2, c_2) and corners S, e_1, e_2, e_4 with S the uppermost lattice element on line l_j (figure 6):

$$\begin{aligned} e_1 &= A[ir_1 + jr_2, ic_1 + jc_2] \\ e_2 &= A[ir_1 + (j + 1)r_2, ic_1 + (j + 1)c_2] \\ e_4 &= A[(i + 1)r_1 + jr_2, (i + 1)c_1 + jc_2] \\ S &= A[(i + 1)r_1 + (j + 1)r_2, (i + 1)c_1 + (j + 1)c_2] \end{aligned}$$

The following are always true:

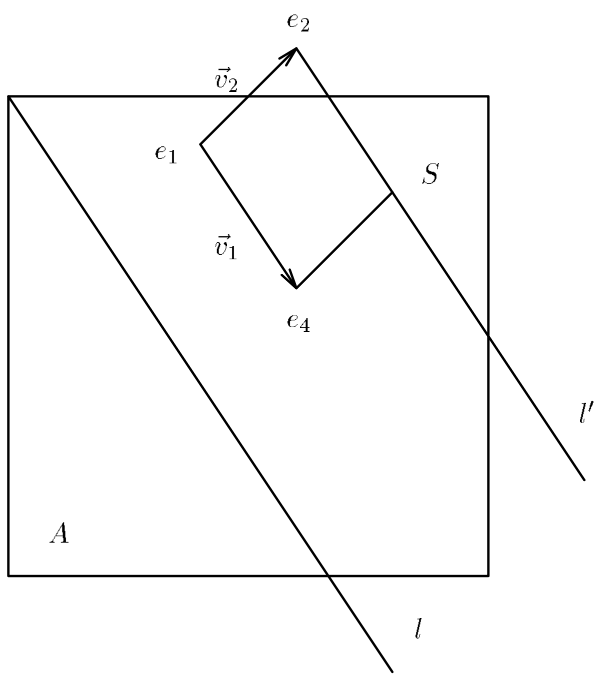

Figure 7:
A candidate source S in lemma 3. Here $|r_1| \geq |r_2|$.

- e_2 is **not** an element of A. Otherwise e_2 - not S - is the top element on its line.
- e_4 **is** an element of A. Otherwise S is not in A, S is not to the right of line l or $r_1 + |r_2| \geq m$.

Two possibilities remain. Either e_1 is an element of A or it is not. Our proof is by induction on i and j. For the base cases we use $\vec{v}_1$ $(i = 1, j = 0)$, $\vec{v}_2$ $(i = 0, j = 1)$ and $\vec{v}_3 = \vec{v}_1 + \vec{v}_2$ which is either a quadrant I vector $(r_1 \geq |r_2|)$ or a quadrant II vector $(r_1 < |r_2|)$ by Lemma 1.

Subcase A: $r_1 \geq |r_2|$.

- e_1 is not an element of A. By the induction hypothesis, $\vec{v}_{e_4} = (i+1)\vec{v}_1 + j\vec{v}_2$ is a symmetry vector. Since e_1 is not on A, $r_{e_4} < r_1$. That is, the row coefficient in $\vec{v}_{e_4}$ is smaller than the row coefficient in $\vec{v}_{1.}$ Apply Lemma 1 to $\vec{v}_{e_4}$ and $\vec{v}_2$ and S is a source.
- e_1 is an element of A. By the induction hypothesis, $\vec{v}_{e_1} = i\vec{v}_1 + j\vec{v}_2$ is a quadrant I symmetry vector. From the base case, $\vec{v}_3 = \vec{v}_1 + \vec{v}_2$ is a quadrant I symmetry vector. Apply Lemma 2 to $\vec{v}_{e_1}$ and $\vec{v}_3$ and S is a source.

Subcase B: $r_1 < |r_2|$.

- e_1 is not an element of A. Impossible, else S is not in A or S is not right of l.
- e_1 is an element of A. Note that S is above row r_1 or else e_2 is on the array. The vector $\vec{v}_{e_2} = i\vec{v}_1 + (j+1)\vec{v}_2$ is a quadrant II symmetry vector (because r_{e_2} is negative) by application of Subcase A to quadrant II. Now, $r_{e_2} + r_1 =$(the row index of S)≥ 0 so $r_1 \geq -r_{e_2}$ or $|r_{e_2}| \leq r_1$. By hypothesis, $r_1 < |r_2|$ and therefore $|r_{e_2}| < |r_2|$. Apply Lemma 2 to $\vec{v}_1$ and $\vec{v}_{e_2}$ and S is a source. $\square$

The proof of Theorem 1 is simplified by the following easily proven observation.

Observation 1 Let (r_1, c_1) and (r_2, c_2) be symmetry vectors from quadrants I and II respectively, and $c_1 + c_2 < m$ and $r_1 + |r_2| < m$, and let L be an infinite lattice of points on the xy-plane also with basis vectors (r_1, c_1) and (r_2, c_2). If we put one copy of A on each lattice point by aligning element $A[0, 0]$ with the lattice point, then the copies are in register and completely cover the plane.

The next Lemma establishes that for a given lattice of elements in A, an element not on the lattice has a shorter vector to some lattice point than the corresponding basis vector for the lattice. (Note that a simplified version of the proof appeared in [GP-92] and we use essentially that same proof here.)

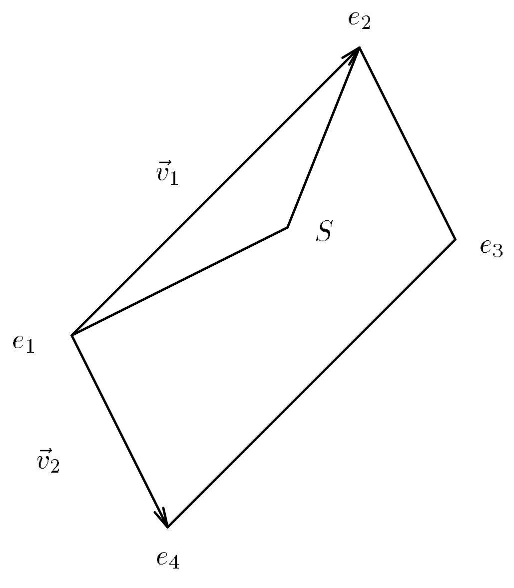

Figure 8:
One of the vectors from e_1 to S or S to e_2 is a quadrant I vector shorter than $\vec{v}_1$.

Lemma 4 Let L be an infinite lattice in the xy-plane with basis vectors $\vec{v}_1 =(r_1, c_1)$ and $\vec{v}_2 =(r_2, c_2)$ (quadrants I and II symmetry vectors respectively) where all the r_i and c_i are integers. Then, for any point $S = (x, y)$ that is not a lattice element, where x and y are integers, there exists a lattice point e such that the vector $\vec{v}$ from e to S (or S to e) is a quadrant I vector shorter than $\vec{v}_1$ or a quadrant II vector shorter than $\vec{v}_2$.

Proof: Let S be an element that does not fall on a lattice point. Consider the unit cell of the lattice containing S (figure 8) with nodes labeled e_1, e_2, e_3 and e_4, where

$$
\begin{aligned}
e_4 &= e_1 + \vec{v}_1 \\
e_2 &= e_1 + \vec{v}_2 \\
e_3 &= e_1 + \vec{v}_2 + \vec{v}_3
\end{aligned}
$$

Connect S to the four corners of the unit cell to get four triangles. At least one of these triangles has a right or obtuse angle. Wolog, let the triangle be on points e_1, e_2 and S. Then both the vector from e_1 to S and the vector from e_2 to S is shorter than the vector from e_1 to e_2. Since at least one of the two is a quadrant I vector, we have a quadrant I vector shorter than $\vec{v}_1$. $\square$

Our first main result is the following Theorem. It establishes that if an array has basis vectors in both quadrants, then in a certain block of the array, which depends on the coefficients of the basis vectors, all symmetry vectors are linear combinations of the basis vectors. We state the Theorem in terms of quadrant I for simplicity. Since the array can be rotated so that any quadrant becomes quadrant I, it applies to all quadrants.

Theorem 1 Let A be an array with *basis vectors* (r_1, c_1) and (r_2, c_2) in quadrants I and II respectively with $c_1 + c_2 < m$ and $r_1 + |r_2| < m$. Let L be an infinite lattice with the same basis vectors and containing the element $A[0, 0]$. Then, in the block $A[0..m - r_1 - |r_2|, 0..m - c_1 - c_2]$, an element is a quadrant I source iff it is a lattice element.

Proof: By Lemma 3, if $S = A[r, c]$ is a lattice element, then it is a source. Suppose that S is not a lattice element, but that it is a quadrant I source. We will show that S can not occur within block $A[0..m - r_1 - |r_2|, 0..m - c_1 - c_2]$.

By way of contradiction, assume S does occur in prefix block $A[0..m - r_1 - |r_2|, 0..m - c_1 - c_2]$. There is a quadrant I vector $\vec{v}$ associated with S that is not a linear combination of $\vec{v}_1$ and $\vec{v}_2$. By Observation 1, copies of A can be aligned with the points of lattice L and the copies will be in register and cover the plane. Let $A' = A[r..m - 1, c..m - 1]$, *i.e.* the suffix block originating at element $A[r, c]$. Because S is a source, $\vec{v}$ maps $A[0..m - r - 1, 0..m - c - 1]$ to A'. For each copy of A, remove all but A'. The copies of A' are in register. Since A' has dimensions at least $r_1 + |r_2|$ by $c_1 + c_2$, it is at least as large as a unit cell of the lattice and therefore, the copies of A' also cover the plane. Now every element of the plane is mapped by $\vec{v}$ from an identical element, and there is a complete copy of A at S. S falls within some cell of lattice L. By Lemma 4, there is a quadrant I or quadrant II vector $\vec{v}_3$ from S to some corner e of the cell (or from e to S) which is *shorter* than the corresponding basis vector of L. Since there are complete copies of A at S and e, $\vec{v}_3$ is a symmetry vector and therefore $\vec{v}_1$ and $\vec{v}_2$ are not both basis vectors of A as assumed. $\square$

Since our quadrants are of size $\lceil \frac{m}{3} \rceil \times \lceil \frac{m}{3} \rceil$, they are no greater in size than the smallest block that can contain only lattice point sources. The region that contains only lattice point sources can be larger than the block described in Theorem 1, see [GP-92].

Next, we prove the following important trait about radiant periodic arrays that facilitates their handling in matching applications [AB-92b, ABF-94b, KR-94]. Origins ($A[0,0]$) of complete copies of a radiant periodic array A that overlap without mismatch can be *ordered monotonically.*

Definition 5 A set of elements of an array B can be *ordered monotonically* if the elements can be ordered so that they have column index nondecreasing and row index nondecreasing (ordered monotonically in quadrant I) or row index nonincreasing (ordered monotonically in quadrant II).

Our theorem is stated in terms of quadrant I, but generalizes to quadrant II.

Theorem 2 Let A be a radiant periodic array with periodic vector in quadrant I. Let $S_1, \ldots, S_j$ be quadrant I sources occuring within quadrant I. On each source, place one copy of A by aligning $A[0,0]$ with the source. If every pair of copies is in register, then the sources can be ordered monotonically in quadrant I.

Proof: Suppose two sources $A[c_1, r_1]$ and $A[c_2, r_2]$ cannot be ordered monotonically. That is, $c_1 < c_2$ but $r_2 < r_1$. If there is no mismatch in the copies of A at these sources, then by the fact that $c_2 - c_1 < \frac{m}{3}$ and $r_1 - r_2 < \frac{m}{3}$, $\vec{v} = (r_2 - r_1, c_2 - c_1)$ is a periodic, quadrant II symmetry vector and by definition, A is lattice periodic, a contradiction. $\Box$

As stated earlier, our classification scheme applies to any rectangular array. The major modification is a new definition of length.

Definition 6 The *length* of a symmetry vector of a rectangular array is the maximum of the absolute values of its coefficients *scaled to the dimensions of the array.* Let A be n rows by m columns with $m \geq n$. Let $\vec{v} = (r, c)$ be a symmetry vector in A. Then the length of $\vec{v}$ scaled to the dimensions of the array is $\max(r \cdot \frac{m}{n}, c)$.

4 Periodicity and Witness Algorithms

In this section, we present two algorithms, one serial and one parallel for finding all sources in an array A. In addition, for each location in A which is not a source, our algorithms find a *witness* that proves that the overlapping copies of A are not in register.

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 $A[r,c] \neq A[i+r, j+c]$ (figure 9). 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 $A[r,c] \neq A[i-(m-1)+r, j+c]$.

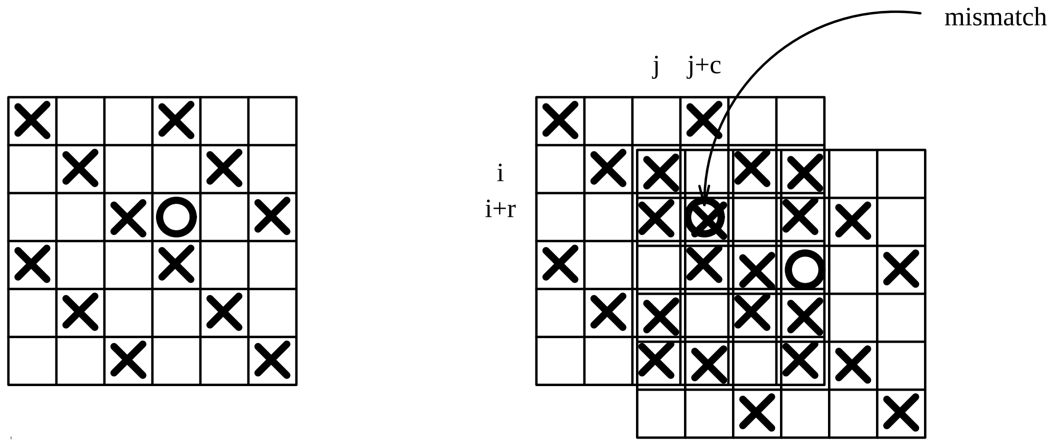

Figure 9:
The witness tables gives the location of a mismatch
(if one exists) for two overlapping patterns: $Witness[i,j] = [r,c]$.

4.1 The Serial Algorithm

Our serial algorithm (Algorithm A) makes use of two algorithms (Algorithms 1 and 2) from [ML-84] which are themselves variations of the KMP algorithm [KMP-77] for string matching. Algorithm 1 takes as input a pattern string w of length m and builds a table *lppattern*$[0..m-1]$ where *lppattern*$[i]$ is the length of the longest prefix of w starting at w_i. Algorithm 2 takes as input a text string t of length n and the table produced by Algorithm 1 and produces a table *lptext*$[0..n-1]$ where *lptext*$[i]$ is the length of the longest prefix of w starting at t_i.

The idea behind Algorithm A is the following: We convert the two-dimensional problem into a problem on strings (figure 10). Let the array A be processed column by column and suppose we are processing column j. Assume we can convert the suffix block $A[0..m-1, j..m-1]$ into a string $T_j = t_0 \ldots t_{m-1}$ where t_i represents the suffix of row i starting in column j. This will serve as the text string. Assume also that we can convert the prefix block $A[0..m-1, 0..m-j-1]$ into a string $W_j = w_0 \ldots w_{m-1}$ where w_i represents the prefix of row i of length $m-j$. This will serve as the pattern string. Now, use Algorithm 1 to produce the table *lppattern* for W_j and Algorithm 2 to produce the table *lptext* for T_j. If a copy of the pattern starting at t_i matches in every row to t_{m-1}, then *lptest*$[i] = m - i$ and $A[i,j]$ is a source. If the pattern doesn't match and the first pattern row to mismatch is row $k < m - i$, then *lptext*$[i] = k$ and $A[i,j]$ is not a source. The mismatch occurs between the prefix of pattern row k and the suffix of text row $i + k$. We need merely locate the mismatch to obtain the witness.

In order to treat the suffix and prefix of a row as a single character, we will build a *suffix tree* for the array. A suffix tree is a compacted trie of suffixes of a string $S = s_1 \cdots s_n$ [W-73]. Each node v has associated with it the indices [a,b] of some substring $S(v) = s_a \cdots s_b$ of S. If u is the Least Common Ancestor (LCA) of two nodes v and w, then $S(u)$ is the longest common prefix of $S(v)$ and $S(w)$ [LV-85]. A tree can be preprocessed in linear time to answer LCA queries in constant

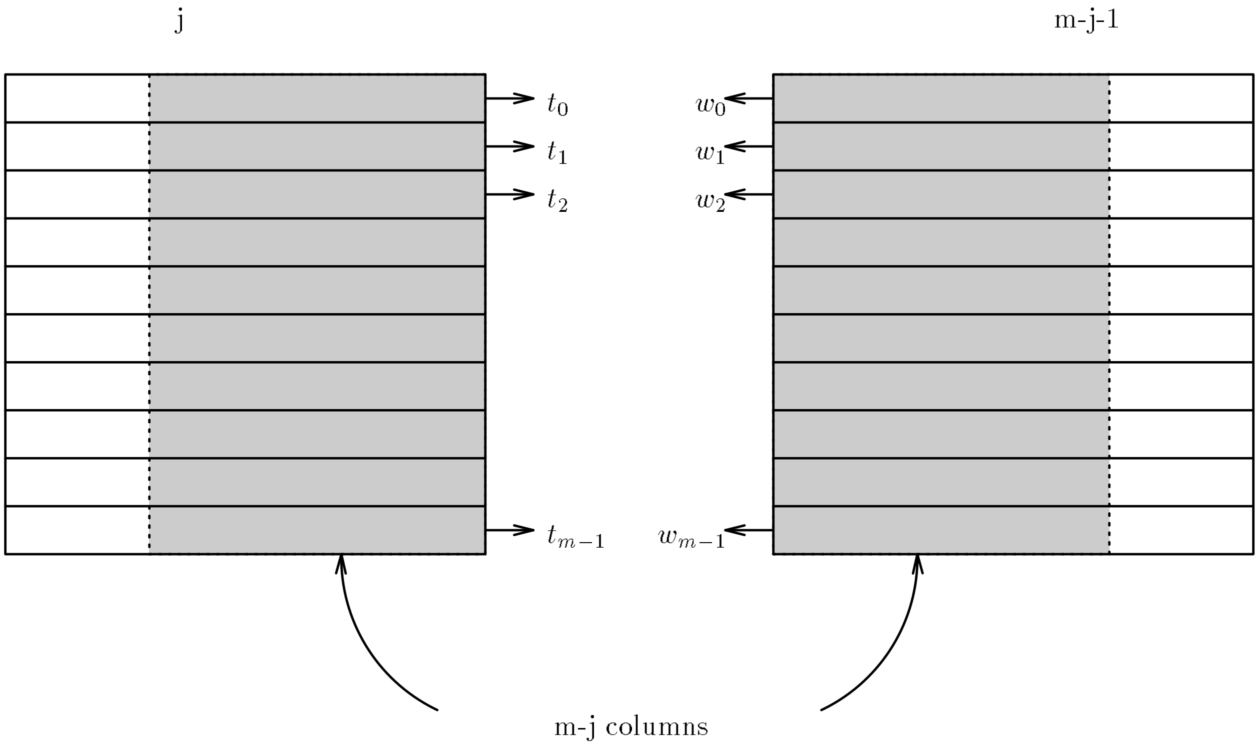

Figure 10:
Representing a block of the array by a string.
$T_j = t_0 \cdots t_{m-1}$ is the text and $W_j = w_0 \cdots w_{m-1}$ is the pattern.

time [HT-84]. Thus, we can answer questions about the length of $S(u)$ in constant time.

Algorithm A Serial algorithm for building a witness array and deciding periodicity class.

Step A.1: Build a suffix tree by concatenating the rows of the array. Preprocess the suffix tree for least common ancestor queries in order to answer questions about the length of the common prefix of any two suffixes.

Step A.2: For each column j, fill out $Witness[0..m-1, j]$ (quadrant I):

Step A.2.1: Use Algorithm 1 to construct the table *lppattern* for $W_j = w_0 \dots w_{m-1}$. Character w_i is the *prefix* of row i of length $m-j$. We can answer questions about the equality of two characters by consulting the suffix tree. If the common prefix of the two characters has length at least $m-j$ then the characters are equal.

Step A.2.2: Use Algorithm 2 to construct the table *lptext* for $T_j = t_0 \dots t_{m-1}$. Character t_i is the *suffix* of row i starting in column j (also of length $m-j$). Again we test for equality by reference to the suffix tree.

Step A.2.3: For each row i, if $lptext[i] = m - i$ then we have found a quadrant I source and $Witness[i,j] = [m,m]$ otherwise, using the suffix tree, compare the suffix of text row $i + lptext[i]$ starting in column j with the prefix of pattern row $lptext[i]$. The length l of the common prefix will be less than $m - j$, and $Witness[i,j] = [lptext[i], l+1]$.

Step A.3: Repeat step 2 for $Witness[-m+1..0, j]$ (quadrant II) by building the automata and processing the columns from the bottom up.

Step A.4: Select quadrant I and quadrant II basis vectors from $Witness$ if they exist.

Step A.5: Use the basis vectors to decide to which of four periodicity classes the pattern belongs.

Theorem 3 Algorithm A is correct and runs in time $O(m^2 \log |\Sigma|)$.

Proof: The correctness of Algorithm A follows from the correctness of Algorithms 1 and 2 [ML-84]. The suffix tree construction [W-73] takes time $O(m^2 \log |\Sigma|)$ while the preprocessing for least common ancestor queries [HT-84] can be done in time linear in the size of the tree. Queries to the suffix tree are processed in constant time. The tables *lppattern* and *lptext* can be constructed in time $O(m)$ [ML-84]. For each of m columns, we construct two tables so the total time for steps 2 and 3 is $O(m^2)$. Step 4 can be done in one scan through the witness array and step 5 requires comparing all vectors to the basis vectors in order to distinguish between the radiant and line periodic classes, so the time for steps 4 and 5 is $O(m^2)$. The total complexity of the pattern preprocessing is therefore $O(m^2 \log |\Sigma|)$. $\square$

Recently, [GP-92] gave a linear time serial algorithm for the witness computation.

4.2 The Parallel Algorithm

Our parallel algorithm (Algorithm B) makes use of the parallel string matching algorithm (Algorithm 3) from [V-85]. Algorithm 3 takes as input a pattern string w of length m and a text string t of length n and produces a boolean table $match[0..n-m-1]$, where $match[i] = true$ if a *complete* copy of the pattern starts at t_i. Algorithm 3 first preprocesses the pattern and then processes the text.

First, for a text of length m, we show how to modify Algorithm 3 to compute $match[0..m-1]$, where $match[i] = true$ if $t_i \cdots t_{m-1}$ is a prefix of the pattern. For simplicity, we assume m is a power of 2. Let

$$\begin{aligned} P_k &= w_0 \cdots w_{\lfloor \frac{m-1}{2^k} \rfloor} \\ S_k &= t_{m-1-\lfloor \frac{m-1}{2^k} \rfloor} \cdots t_{m-1} \\ k &= 0, 1, \ldots, \log m \end{aligned}$$

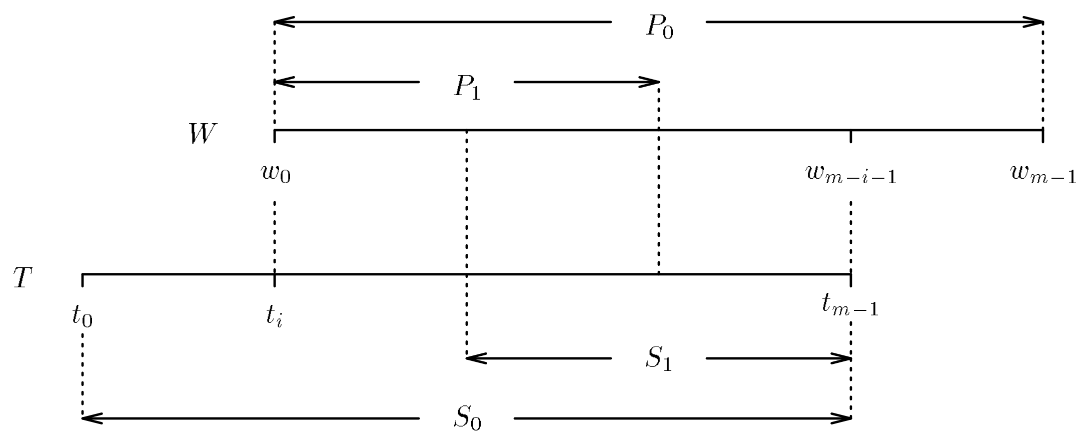

Figure 11:
P_1 is a prefix of $t_i \cdots t_{m-1}$ and S_1 is a suffix of $w_0 \cdots w_{m-i-1}$.

For example, P_1 is the prefix of w of length $\frac{m}{2}$ and S_1 is a suffix of t of the same length. The following observation embodies the key idea (figure 11):

Observation 2 If $t_i \cdots t_{m-1}$ is a prefix of w of length between m and $\frac{m}{2}$, then P_1 is a prefix of $t_i \cdots t_{m-1}$ and S_1 is a suffix of $w_0 \cdots w_{m-i-1}$. Similarly, if $t_i \cdots t_{m-1}$ is a prefix of w of length between $\frac{m}{2}$ and $\frac{m}{4}$, then the prefix and suffix are P_2 and S_2, etc.

Now, for each $k \geq 1$, we attempt to match P_k in S_{k-1} and S_k in P_{k-1}. If a matched prefix begins at t_i and a matched suffix ends at w_{m-i-1} then $t_i \cdots t_{m-1}$ is a prefix of w.

Using Algorithm 3, we first preprocess the P_k and S_k as patterns and then use these to process the appropriate segments as text. We can additionally modify Algorithm 3 so that at every index where a prefix or suffix does not match, we obtain the location of a mismatch. Since the sum of the lengths of the P_i and S_i are no more than a linear multiple of the length of w, the modification does not increase the complexity of the algorithm and therefore the time complexity of the modified Algorithm 3 is $O(\log m)$ using $O(\frac{m}{\log m})$ CRCW processors, the same as the unmodified algorithm [V-85]. In our parallel algorithm, only steps 2 differs from the serial algorithm.

Algorithm B Parallel algorithm for finding sources and building a witness array

Step B.2: For each column j, fill out $Witness[0..m-1, j]$ (quadrant I):

Step B.2.1: For each $k = 1, \ldots, \log m$:

Step B.2.1.1: Use W_j to form P_k and P_{k-1} and T_j to form S_k and S_{k-1}. Use modified Algorithm 3 to match P_k in S_{k-1} and S_k in P_{k-1}. As in the serial algorithm, use the suffix tree to answer questions about equality.

Step B.2.1.2: For each row i for $m-1-\frac{m-1}{2^{k-1}} \leq i < m-1-\frac{m-1}{2^k}$. If P_k matches beginning at t_i and S_k matches ending at w_{m-i-1}, then $Witness[i,j] = [m,m]$. Otherwise, using the row r of mismatch from modified Algorithm 3, refer to the suffix tree to find the column c of mismatch and set $Witness[i,j] = [r,c]$.

Theorem 4 Algorithm B is correct and runs in time $O(\log m)$ using $O(m^2)$ CRCW processors.

Proof: The suffix tree construction [AILSV-87] and preprocessing for LCA queries [SV-88] is done in time $O(\log m)$ using $O(m^2)$ CRCW processors. Step 2 is done in time $O(\log m)$ using $O(\frac{m^2}{\log m})$ CRCW processors [V-85]. Finding the basis vectors is done by prefix minimum [LF-80] in time $O(\log m)$ using $O(\frac{m^2}{\log m})$ processors. Distinguishing the line and radiant periodic cases can be done in constant time using $O(m^2)$ processors. The total complexity is therefore $O(\log m)$ time using $O(m^2)$ CRCW processors.

References

[AB-92a] A. Amir and G. Benson, "Two-Dimensional Periodicity and its Application", *3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992, pp. 440-452.

[AB-92b] A. Amir and G. Benson, "Efficient Two-Dimensional Compressed Matching", *Data Compression Conference*, 1992, pp. 279-288.

[ABF-93] A. Amir, G. Benson and M. Farach, "Optimal Parallel Two Dimensional Text Searching on a CREW PRAM," *5th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1993.

[ABF-94a] A. Amir, G. Benson and M. Farach, "An Alphabet Independent Approach to Two-Dimensional Matching", *SIAM Journal of Computing*, July 1994.

[ABF-94b] A. Amir, G. Benson and M. Farach, "Optimal Two Dimensional Compressed Matching," *21st International Colloquium on Automata, Languages and Programming*, 1994.

[AC-75] A.V. Aho and M.J. Corasick, "Efficient String Matching", *C. ACM,* Vol. 18, No. 6, 1975, pp. 333-340.

[AF-91] A. Amir and M. Farach, "Efficient 2-dimensional Approximate Matching of Non-rectangular Figures", *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, 1991, pp. 212-223.

[AL-91] A. Amir and G.M. Landau, "Fast Parallel and Serial Multidimensional Approximate Array Matching", *Theoretical Computer Science*, Vol. 81, 1991, pp. 97-115.

[ALV-90] A. Amir, G.M. Landau and U. Vishkin, "Efficient Pattern Matching with Scaling", *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, 1990, pp. 344-357.

[AILSV-87] A. Apostolico, C. Iliopoulos, G.M. Landau, B. Schieber, and U. Vishkin, "Parallel Construction of a Suffix Tree with Applications", *Algorithmica*, Vol. 3, 1988, pp. 347-365.

[B-78] T.P. Baker, "A Technique For Extending Rapid Exact-Match String Matching to Arrays of More Than One Dimension", *SIAM J. Comput.,* Vol. 7, No. 4, 1978, pp. 533-541.

[Bi-77] R.S. Bird, "Two Dimensional Pattern Matching", *Information Processing Letters,* Vol. 6, No. 5, 1977, pp. 168-170.

[BM-77] R.S. Boyer and J.S. Moore, "A Fast String Searching Algorithm", *Comm. ACM,* Vol. 20, 1977, pp. 762-772.

[CCG+93] R. Cole, M. Crochemore, Z. Galil, L. Gąsieniec, R. Harihan, S. Muthukrishnan and K. Park, "Optimally fast parallel algorithms for preprocessing and pattern matching inone and two dimensions," *34th IEEE Symp. on Foundations of Computer Science*, 1993.

[CR-92] M. Crochemore and W. Rytter, "Note on two dimensional string matching by optimal parallel algorithms," *Int. Conf. on Image Processing, Lecture Notes in Computer Science*, Springer-Verlag, Vol 654, 1992.

[G-85] Z. Galil, "Optimal Parallel Algorithms for String Matching", *Information and Control*, Vol. 67, 1985, pp 144-157.

[GP-92] Z. Galil and K. Park, "Truly Alphabet Independent Two-Dimensional Pattern Matching," *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 1992, pp 247-256.

[HT-84] D. Harel and R.E. Tarjan, "Fast Algorithms for Finding Nearest Common Ancestors", *SIAM J. Computing,* Vol. 13, No. 2, 1984, pp. 338-355.

[KR-94] M. Karpinski and W. Rytter, "Alphabet Independent Optimal Parallel Search for 3-Dimensional Patterns," *5th Annual Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science*, Springer-Verlag, vol. 807, 1994, pp125-135.

[KMP-77] D.E. Knuth, J.H. Morris and V.R. Pratt, "Fast Pattern Matching in Strings", *SIAM J. Comp.,* Vol. 6, 1977, pp. 323-350.

[KS-87] K. Krithivasan and R. Sitalakshmi, "Efficient Two Dimensional Pattern Matching in the Presence of Errors", *Information Sciences*, Vol. 47, 1987, pp. 169-184.

[LF-80] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation", *JACM,* Vol 27, 1980, pp. 831-838.

[LV-85] G.M. Landau and U. Vishkin, "Efficient string matching in the presence of errors", *Proc. 26th IEEE FOCS*, 1985, pp 126-136.

[ML-84] M.G. Main and R.J. Lorentz, "An $O(n \log n)$ Algorithm for Finding all Repetitions in a String", *J. of Algorithms,* 1984, pp. 422-432.

[RR-93] M. Regnier and L. Rostami. "A Unifying Look at d-Dimensional Periodicities and Space Coverings," *Proc. 4th Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science*, Springer-Verlag, Vol 684, 1993.

[SV-88] B. Schieber and U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization", *SIAM J. Comput.*, Vol. 17, 1988, pp. 1253-1262.

[V-85] U. Vishkin, "Optimal Parallel Pattern Matching in Strings", *Information and Control,* Vol. 67, 1985, pp. 91-113.

[V-91] U. Vishkin, "Deterministic Sampling–a new technique for fast pattern matching", *SIAM J. Comput.*, Vol. 20, 1991, pp. 303-314.

[W-73] P. Weiner, "Linear Pattern Matching Algorithms", *Proc. 14th IEEE Symposium on Switching and Automata Theory,* 1973, pp. 1-11.