Two Dimensional String Matching for Non-Rectangular Patterns*

R. Friedrich† Th. Ottmann† S. Schuierer†

Abstract

The two dimensional string matching problem is to find the occurrences of a two dimensional pattern in a two dimensional text. Usually, the pattern is assumed to be a rectangle or even a square. In this paper we show how to adapt existing two dimensional string matching algorithms to triangular and other non-rectangular patterns without loss of efficiency. Moreover, we charaterize the patterns that fulfill a necessary consistency condition used in most two dimensional string matching algorithms and show how to recognize efficiently those patterns that fulfill the consistency condition.

1 Introduction

String matching is a very well studied area in computer science [Gal85]. The classical string matching problem is to find all locations where a pattern string can be aligned with a text string such that the aligned characters of the pattern and the text match. Numerous variations of this problem have been considered and a large number of efficient algorithms have been developed for this problem.

In recent years multidimensional string matching has become a field of more and more interest, largely motivated by problems in low-level image processing [RK82]. In two dimensional string matching the problem is to find the occurrences of a two dimensional pattern in a two dimensional

*This work was supported by the Deutsche Forschungs Gemeinschaft under Grant No. Ot 64/8-1.

†Institut für Informatik, Universität Freiburg, Rheinstr. 10, D-79104 Freiburg, FRG. email: {ottmann,schuierer}@informatik.uni-freiburg.de

text. It has applications in digital picture processing, picture databases, computer vision and many other fields. Formally, the two dimensional string matching problem is defined by a text $T[0..n-1, 0..n-1]$ and pattern $P[0..m-1, 0..m-1]$. The text and pattern contain symbols from an alphabet Σ and we have to find all occurrences of P in T. In our generalization, we allow the pattern to contain several "don't care" symbols denoted by "$*$" where a $*$ matches with every symbol in Σ. So our pattern is given in an $m \times m$ array P and the elements of P are in $\Sigma \cup \{*\}$.

In 1977/78 the problem was first attacked by Bird [Bir77] and Baker [Bak78]. Both independently propose a reduction to the algorithm of Aho and Corasick [AC75] for the one dimensional dictionary matching problem. Since the algorithm of Aho and Corasick is automata based, the total running time for this approach is $O(n^2 \log |\Sigma|)$. For a long time this remained the fastest solution before Zhu and Takaoka [ZT89] were able to improve the time bound to $O(n^2)$ by using a stronger arithmetic model.

Several other techniques have been applied to the two dimensional pattern matching problem. One approach makes use of subword trees [AF92, ALV90]. Since operations on subwords depend on the alphabet size, the algorithm achieves a running time of $O(n^2 \log |\Sigma|)$. If "don't care" characters are allowed in the pattern, convolutions turn out to be a powerful tool. Amir and Landau developed a convolution based two dimensional string matching algorithm with complexity $O(|\Sigma| n^2 \log m)$ that allows "don't care" characters [AL91] .

Finally, there are two algorithms which are based on the idea of two dimensional periodicity which was first introduced by Amir and Benson [AB92]. The first one is an algorithm developed by Amir, Benson, and Farach [ABF92] which uses two dimensional periodicity for the text analysis phase but in order to preprocess the pattern it makes use of a suffix tree. This leads to a running time of $O(n^2 + m^2 \log |\Sigma|)$; the second is an algorithm developed by Galil and Park [GP92] which employs two dimensional periodicity for the pattern analysis phase as well as the text analysis phase and was, therefore, termed "truly alphabet independent". It achieves optimal $O(n^2 + m^2)$ time.

In this paper we show how to adapt the algorithm by Amir, Benson, and Farach [ABF92], ABF-algorithm for short, to non-rectangular

patterns. We characterize the class of patterns which fulfil the following consistency condition. If pattern P_2 is to the right and below pattern P_1 and pattern P_3 is to the right and below pattern P_2 such that the symbols in the region of overlap among P_1 and P_2 as well as among P_2 and P_3 match, then the symbols in the region of overlap between P_1 and P_3 match also. This condition is not only crucial in the ABF-algorithm but also in the algorithm by Galil and Park.

The paper is organized as follows. In Section 2 we give a high-level description of the algorithm by Amir, Benson, and Farach. In particular, we investigate which parts of the algorithm have to be changed when non-rectangular patterns are used. Section 3 deals with patterns that satisfy the above consistency condition and it is shown how to recognize them. In Section 4 we show how to adapt the ABF-algorithm to consistent patterns.

2 The ABF-Algorithm

In this section we give a concise description of the two dimensional string matching algorithm by Amir, Benson, and Farach. It is based on the concept of two dimensional periodicity. The main idea is to find locations other than the origin $P[0,0]$, where the pattern can be superimposed on itself without a mismatch. Such a location is called a *source*. There are two different kinds of sources:

1. (i,j) is a *top-source* if P is superimposed upon itself such that $P[0,0]$ is aligned with $P[i,j]$ and in the region of overlap there is no mismatch;

2. (i,j) is a *bottom-source* if P is superimposed upon itself such that $P[m-1,0]$ is aligned with $P[i,j]$ and in the region of overlap there is no mismatch.

With the help of a suffix tree all sources can be computed in time $O(m^2 \log |\Sigma|)$. The result is represented by the two arrays *top-witness* and *bottom-witness* where

top-witness$(i,j) = (m,m)$ if $P[i,j]$ is a top-source and

top-witness$(i,j) = (r,c)$ if $P[r,c] \neq P[r+i, c+j]$.

bottom-witness$(i,j) = (m,m)$ if $P[i,j]$ is a bottom-source and

bottom-witness$(i,j) = (r,c)$ if $P[r,c] \neq P[r+i-(m-1), c+j]$.

If *top-witness*$(i,j) = (r,c)$ or *bottom-witness*$(i,j) = (r,c)$, then (r,c) is called a *witness* against (i,j).

The analysis of the pattern is necessary in order to compute efficiently where the pattern can occur in the text. A *candidate* is a location (i,j) in the text T where the pattern may occur. Let (i_1,j_1) and (i_2,j_2) be two candidates. The two candidates (i_1,j_1) and (i_2,j_2) are *consistent* if $(|i_1-i_2|, |j_1-j_2|)$ is a source or if they do not overlap. If (i_1,j_1) and (i_2,j_2) are consistent, then we write $(i_1,j_1) \doteq (i_2,j_2)$.

For instance, if $i_1 \geq i_2$, $j_1 \geq j_2$, and $P[i_1-i_2, j_1-j_2]$ is not source, say *top-witness*$(i_1-i_2, j_1-j_2) = (r,c)$, then $P[r,c] \neq P[r+i_1-i_2, c+j_1-j_2]$. If we consider text position $T[i_1+r, j_1+c]$, then $T[i_1+r, j_1+c]$ is compared with $P[r,c]$ for candidate (i_1,j_1) and with $P[i_1-i_2+r, j_1-j_2+c]$ for candidate (i_2,j_2). One of them yields a mismatch and candidate (i_1,j_1) or candidate (i_2,j_2) (or both) can be discarded. Hence, given two candidates it is possible to decide in constant time if they are consistent and, if not, to discard at least one of them that has a mismatch with the text.

If we are given a set K of pairwise consistent candidates, then we only have to compare each symbol of T once with a pattern symbol. For if a text position (i_1,j_1) does not match with one of the patterns that overlap it, then all the patterns that overlap (i_1,j_1) in T can be removed since pairwise consistent candidates expect the same characters in the text in their region of overlap. Consistent candidates in K can be verified with a single scan over the text in $O(n^2)$ time.

In order to compute K the following two lemmata are crucial.

Lemma 2.1 *If* $1 \leq i_1 \leq i_2 \leq i_3 \leq n$, $1 \leq j_1 \leq j_2 \leq j_3 \leq n$, $(i_1,j_1) \doteq (i_2,j_2)$, *and* $(i_2,j_2) \doteq (i_3,j_3)$, *then* $(i_1,j_1) \doteq (i_3,j_3)$.

Lemma 2.2 *If* $1 \leq i_1 \leq i_2 \leq i_3 \leq n$, $1 \leq j_3 \leq j_2 \leq j_1 \leq n$, $(i_1,j_1) \doteq (i_2,j_2)$, *and* $(i_2,j_2) \doteq (i_3,j_3)$, *then* $(i_1,j_1) \doteq (i_3,j_3)$.

We now describe how to compute K. This is done in two steps. First we compute a set of consistent candidates for each column and in the second step we compute a set of consistent candidates among these for

each row. More precisely we do the following. For each column j in the text, a doubly linked list C_j of consistent candidates is computed in the following way.

1. If (i, j) is consistent with the first candidate in C_j, then add (i, j) to C_j.
2. If (i, j) is inconsistent with the first candidate (i', j) of C_j, then
   (a) If (i, j) is rejected, take $(i-1, j)$ as the next candidate in column j.
   (b) If (i, j) is not rejected and $C_j \neq \emptyset$, then remove the first element of C_j and check the consistency of (i, j) and the new first element in C_j.
   (c) If (i, j) is not rejected and $C_j = \emptyset$, then add (i, j) to C_j.

Note that if (i, j) is consistent with the first candidate in C_j, it is also consistent with every other candidate in C_j because of Lemma 2.1. But a candidate in C_j is not necessarily consistent with the candidates of another list $C_{j'}$, so we have to check the consistency of candidates in different lists.

In order to do so the column lists C_j are processed from right to left. There are two phases in processing C_j, a bottom up phase and a top down phase. In the bottom up phase we keep only those candidates of C_j that are consistent with all candidates that are to the right of it and below it and in the top down phase we keep only those candidates of C_j that are consistent with all candidates that are to the right of it and above it. A candidate (i, j) that passes both phases is then added to a row list R_i. Below we give a more detailed description.

1. Bottom up for column j: From bottom to top we step simultaneously through the candidates in C_j and the row lists R_i. Note that the row lists only contain candidates of the colums $C_{j+1}, \ldots, C_{n-m}$. Let (i, j) be the current candidate in C_j starting with the lowest candidate. We have to test consistency with candidates (i', j') below and strictly to the right of (i, j), that is, with $i' \geq i$ and $j' > j$ which is described below. For the sake of a simple presentation we assume that the

highest candidate of C_j is $(-1, j)$ and that it is consistent with all patterns.

Algorithm Row Lists
(1) $i' := n - 1$;
(2) $(i, j) :=$ the bottommost candidate in C_j;
(3) **while not** $i' = 0$ **do**
$(i', j') :=$ the leftmost candidate of $R_{i'}$;
(4) **if** $(i, j) \doteq (i', j')$ **then**
(4a) **if** $i' > i$
then $i' := i' - 1$;
else $(i, j) :=$ next candidate of C_j;
(4b) **else** compare (i, j) and (i', j') with T and
depending on the outcome:
remove either (i, j) or (i', j');
if (i, j) is removed
then $(i, j) :=$ next candidate of C_j;
end while;

The **while**-loop in Step (3) has the following invariants:

(a) $i' \geq i$,

(b) if (i_1, j) is a candidate of C_j with $i_1 > i$, then (i_1, j) is consistent with all candidates in the row lists R_{i_2} with $i_2 \geq i_1$;

(c) if (i_1, j) is a candidate of C_j with $i_1 < i$, then (i_1, j) is consistent with all candidates in row lists R_{i_2} with $i_2 \geq i$ that are are consistent with (i, j).

(d) (i, j) is consistent with all candidates in rows $R_{i''}$ with $i'' > i'$.

To see the correctness of the algorithm note that if a new candidate is chosen in Step (4a), the old candidate (i_{old}, j_{old}) is consistent with all candidates in $R_{i'}$ and the rows below it by Lemma 2.1; furthermore, $i' \geq i$ and (i, j) is consistent with the same candidates of $R_{i'}$ and the rows below it as (i_{old}, j_{old}) by Lemma 2.1 which ensures the invariant of the **while**-loop. A similar remark applies to Step (4b).

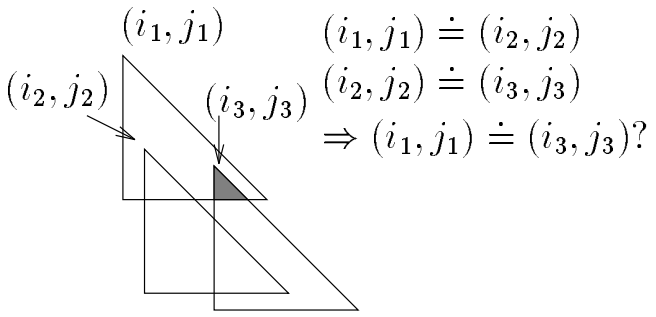

Figure 1: The small shaded triangle may cause inconsistency.

2. Top-down for column j: analogous to the Bottom-up phase.

In the last step, we have to verify if the candidates actually occur in the text. Therefore, we mark every location $T[i, j]$ with a pattern coordinate (i', j') if $(i - i', j - j')$ is a consistent candidate. We then compare $T[i, j]$ with $P[i', j']$ if $T[i, j]$ is labeled with (i', j'). If there is a mismatch, we mark the text position with *false*. Now every candidate which does not overlap a *false*-label represents a pattern occurrence.

The text scanning phase needs $O(n^2)$ time. More details can be found in [ABF92].

3 Transitivity of the Consistency-Relation

In the text scanning phase, that is, the computation and verification of consistent candidates, the transitivity implied by Lemmata 2.1 and 2.2 is crucial to achieve the time bound of $O(n^2)$. Hence, in order to generalize the algorithm of Amir, Benson, and Farach to non-rectangular patterns, we have to check for which patterns these two lemmata hold. We also say that the patterns satisfy the *consistency relation* in this case. Consider the example of triangular patterns as displayed in Figure 1. It may seem that there are cases where $(i_1, j_1) \doteq (i_2, j_2)$ and $(i_2, j_2) \doteq (i_3, j_3)$ but $(i_1, j_1) \not\doteq (i_3, j_3)$.

We can show in the following that this triangular form belongs to a class of pattern forms for which the consistency relation holds and,

therefore, the shaded part does not cause a mismatch independent of the choice of characters for the triangle.

We assume that the triangle is given in a $m \times m$ array with $P[i,j] = *$ for $j > i$. In general we consider patterns as $m_1 \times m_2$ arrays with $P[i,j] \in \Sigma \cup \{*\}$. If we compare a character in Σ with "$*$", no mismatch is caused. We write $a \cong b$, for $a, b \in \Sigma \cup \{*\}$ if $a = b$ or $a = *$ or $b = *$.

We call the elements of P *lattice points*. The point (i,j) is called an *external lattice point*, if $P[i,j] = *$ and otherwise it is called an *internal lattice point*.

If $P[i,j] = a$, with $a \in \Sigma$, we say we have assigned a to the lattice point (i,j). We distinguish between the pattern and the *form* of the pattern. The pattern is a $m_1 \times m_2$ matrix and every point is assigned a character of $\Sigma \cup \{*\}$ as opposed to the form of the pattern where we are only interested in internal lattice points. In the above example the form is a triangle and the pattern is given in a square array.

In the following we are interested in those pattern forms for which Lemmata 2.1 and 2.2 hold for every possible assignment of characters—$\neq *$—to internal lattice points. We start off with a definition of convexity.

Definition 3.1 *The form of a pattern is* convex *if for any two internal lattice points* (i_1, j_1) *and* (i_2, j_2) *the points* $(i_1, j_1) + l((i_1 - i_2)/k, (j_1 - j_2)/k)$, *with* $0 \leq l \leq k$ *and* $k = \gcd(|i_1 - i_2|, |j_1 - j_2|)$, *are also internal lattice points.*

The above definition of discrete convexity has properties similar to the notion of convexity in the plane as the following observation illustrates.

Observation 3.1 *If F is a convex pattern form and $A = (i,j)$, $B = (i + i', j)$, and $C = (i + i', j + j')$ are three internal lattice points, then all the lattice points that are contained in the triangle T with vertices A, B, and C belong to F; in particular, if $D_1 = (i + i'_1, j + j'_1)$ and $D_2 = (i + i'_2, j + j'_2)$ are two points with $i'_1 + i'_2 = i'$ and $j'_1 + j'_2 = j'$, then either D_1 or D_2 belongs to F (see also Figure 2).*

Proof: Without loss of generality we can assume that $A = (0,0)$. We show by induction that if F contains the three internal lattice points $A = (0,0)$, $B = (i,0)$ and $C = (i,j)$, then the geometric triangle defined by A, B, and C contains only internal lattice points.

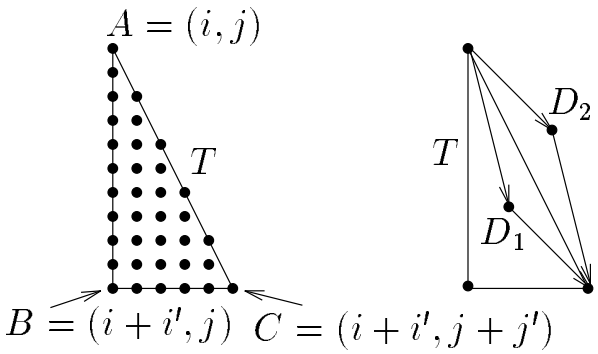

Figure 2: A convex triangle contains all lattice points.

We make use of the following three facts.

Fact 1: The triangle includes only lattice points (i_1, j_1) with $i_1/j_1 \geq i/j$, for $0 < j_1 \leq j$, and points $(i_1, 0)$ with $0 \leq i_1 \leq i$.

Fact 2: For all $0 < i_0 < i$ and $0 < j_0 < j$, we have that
(a) $i/j > i_0/j_0$ if and only if $(i - i_0)/(j - j_0) > i/j$.
(b) $i/j < i_0/j_0$ if and only if $(i - i_0)/(j - j_0) < i/j$.

Fact 3: For all $0 < i_0 < i$ and $0 < j_0 < j$ with $j \leq i$, we have if $i/j > i_0/j_0 > (i + 1)/(j + 1)$, then $i_0/(j_0 - 1) > i/j$.

The proof is by induction on i. Observation 3.1 holds trivially for $i = j = 0$. So assume that the claim is true for all triangles with end points $A = (0, 0)$, $B = (i, 0)$ and $C = (i, j)$ where w.l.o.g. $j \leq i$. The claim is now that the triangle defined by the end points $(0, 0)$, $(i + 1, 0)$ and $(i + 1, j)$ where $j \leq i + 1$ contains only internal lattice points. We prove this claim again by induction, this time on j. For $j = 0$ the claim is obviously true. Hence, assume that all triangles with end points $(0, 0)$, $(i+1, 0)$ and $(i+1, j)$ where $j < i+1$ contain only internal lattice points. Now consider the triangle T_1 which is defined by the end points $(0, 0)$, $(i + 1, 0)$ and $(i + 1, j + 1)$.

If $\gcd(i+1, j+1) > 1$, then we can divide T_1 into two smaller triangles T_2 and T_3 and a rectangle R as shown in Figure 3. The claim holds for T_2 and T_3 by induction and by the convexity of T_1 the rectangle R is contained in it. Hence, we assume in the following that $\gcd(i + 1, j + 1) = 1$.

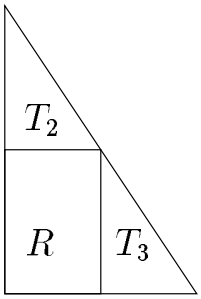

Figure 3: How to split T_1 if $\gcd(i+1, j+1) > 1$.

Note that (i, j) is an internal lattice point and we, therefore, only have to consider points (i_0, j_0) with

$$(*) \quad \frac{i}{j} > \frac{i_0}{j_0} > \frac{i+1}{j+1},$$

for $0 < i_0 < i+1$ and $0 < j_0 < j+1$. In the following we construct a sequence of lattice points $(i_0, j_0), (i_2, j_2), \ldots, (i_{2k}, j_{2k})$, for some $k \geq 0$, with the following properties.

(i) If (i_{2l}, j_{2l}) is an internal lattice point, then $(i_{2(l-1)}, j_{2(l-1)})$ is also an internal lattice point, for $1 \leq l \leq k$.

(ii) (i_{2l}, j_{2l}) belongs to T_1.

(iii) $i_{2l} < i_{2(l-1)}$, for $1 \leq l \leq k$.

(iv) (i_{2k}, j_{2k}) is an internal lattice point.

In order to do so let λ_0 be the unique integer such that $0 \leq i+1-\lambda_0 i_0 < i_0$. We claim that $0 < j+1-\lambda_0 j_0 \leq j_0$.

1. $j+1-\lambda_0 j_0 \leq j_0$: With

$$\frac{i}{j} > \frac{\lambda_0 i_0}{\lambda_0 j_0} \geq \frac{i-i_0}{\lambda_0 j_0} = \frac{i-i_0}{j-(j-\lambda_0 j_0)}$$

we obtain $i/j < i_0/(j-\lambda_0 j_0)$ by Fact 2a. This implies that $j-\lambda_0 j_0 < j_0$.

2. $0 < j+1-\lambda_0 j_0$: Since $\lambda_0 i_0/(j+1) \leq (i+1)/(j+1) < \lambda_0 i_0/\lambda_0 j_0$, we have $j+1 > \lambda_0 j_0$.

We denote the point $(i_1, j_1) = (i+1-\lambda_0 i_0, j+1-\lambda_0 j_0)$ by P_1. Let $D_\lambda = (i+1, j+1) - \lambda P_1$. If D_λ is an internal lattice point, for some $\lambda \geq 1$, then $(\lambda_0 i_0, \lambda_0 j_0)$ and with it (i_0, j_0) is also an internal lattice point. To see this we first note that $(\lambda_0 i_0, \lambda_0 j_0) = D_\lambda + (\lambda - 1)P_1$ and that $P_1 = ((i+1, j+1) - D_\lambda)/\lambda$. Hence, by $(\lambda_0 i_0, \lambda_0 j_0) = (1/\lambda)D_\lambda + (\lambda-1)/\lambda(i+1, j+1)$ the point $(\lambda_0 i_0, \lambda_0 j_0)$ is a convex combination of two internal lattice points.

If $j+1-\lambda_0 j_0 = j_0$, then D_{λ_0+1} is an internal lattice point. In order to prove this we observe that $j+1-(\lambda_0+1)(j+1-\lambda_0 j_0) = j+1-(\lambda_0+1)j_0 = 0$, $i+1-(\lambda_0+1)(i+1-\lambda_0 i_0) = \lambda_0((\lambda_0+1)i_0 - i + 1) > 0$, and $i+1-(\lambda_0+1)(i+1-\lambda_0 i_0) \leq i+1-(\lambda_0+1)0 = i+1$. This implies that D_{λ_0+1} a point in the first column of T_1. Hence, we can assume in the following that $j+1-\lambda_0 j_0 < j_0$.

Since $\lambda_0 i_0/\lambda_0 j_0 > (i+1)/(j+1)$, we have $i_1/j_1 < (i+1)/(j+1)$ by Fact 2b. Now we choose λ_1 maximal, so that $i_2 = i+1-\lambda_1 i_1 > 0$ and $j_2 = j+1-\lambda_1 j_1 > 0$. Note that this implies that either $i_2 < i_1$ or $j_2 < j_1$. Furthermore, applying Fact 2a to $\lambda_1 i_1/\lambda_1 j_1 > (i+1)/(j+1)$ yields $i_2/j_2 > (i+1)/(j+1)$.

We show that either (i_2, j_2) is an internal lattice point or it satisfies Conditions (i)–(iii) above. There are two cases to consider.

Case 1: $i_2/j_2 \geq i/j$**.** In this case $(i_2, j_2) = D_{\lambda_1}$ is indeed an internal lattice point and as we argued above so is (i_0, j_0) .

Case 2: $i/j > i_2/j_2 > (i+1)/(j+1)$**.** We have to show that Conditions (i)–(iii) from above hold. Condition (i): If $(i_2, j_2) = D_{\lambda_1}$ is an internal lattice point, then (i_0, j_0) is an internal lattice again by the argument above. Condition (ii): Clearly, (i_2, j_2) belongs to T_1. Condition (iii): Either $i_2 < i_1 < i_0$ by the choice of λ_1 and Condition (iii) is immediately fulfilled or $j_2 < j_1 < j_0$ and $i_0/j_2 \geq i_0/(j_0-1) > i/j > i_2/j_2$ by Fact 3 which again implies $i_2 < i_0$.

We now recur on (i_2, j_2) instead of (i_0, j_0). Because of Condition (iii) this can be done only a finite number of times and there exists an index k for which (i_{2k}, j_{2k}) is an internal lattice point which in turn implies that $(i_{2k}, j_{2k}), (i_{2(k-1)}, j_{2(k-1)}), \ldots, (i_0, j_0)$ are all internal lattice points. □

An analogous observation holds for the points $A = (i, j)$, $B = (i, j + j')$, and $C = (i + i', j + j')$.

Convexity is a necessary but not a sufficient condition for the transitivity of the consistency relation. In order to ensure the transitivity of the consistency relation pattern forms have to satisfy an additional requirement.

Definition 3.2 *Let P be a pattern. If $A = (i_1, j_1)$ and $B = (i_2, j_2)$ are two internal lattice points of P with $i_1 \leq i_2$, then we call (i_1, j_2) the* north corner of A and B *and denote it by $N(A, B)$ and, similarly, we call (i_2, j_1) the* south corner of A and B *and denote it by $S(A, B)$.*

We say a point A in pattern form F is NS-grounded *if, for all other points B in F, either $N(A, B)$ or $S(A, B)$ or both belong to F. A pattern form F is* NS-grounded *if all internal lattice points are NS-grounded.*

In the following we show that Lemmata 2.1 and 2.2 hold for a pattern form F if and only if F is convex and *NS*-grounded. We state this as a theorem.

Theorem 3.2 *The consistency relation is transitive for pattern form F if and only if F is convex and NS-grounded.*

Proof: To see the only if direction suppose the consistency relation is transitive but the form of the pattern is not convex. We construct a pattern, for which either Lemma 2.1 or Lemma 2.2 does not hold. There exist two internal lattice points (i_1, j_1) and (i_2, j_2) with $i_1 \geq i_2$ with $k = \gcd(|i_1 - i_2|, |j_1 - j_2|)$ and $i_{\text{gcd}} = (i_1 - i_2)/k$, $j_{\text{gcd}} = (j_1 - j_2)/k$ such that $(i, j) = (i_1, j_1) + l_0(i_{\text{gcd}}, j_{\text{gcd}})$ is an external lattice point, for some $0 < l_0 < k$.

Suppose that $j_1 \geq j_2$. We assign character $a \in \Sigma$ to all internal lattice points of the form $(i, j) = (i_1, j_1) + l(i_{\text{gcd}}, j_{\text{gcd}})$ with $0 \leq l < l_0$ and character $b \in \Sigma$, $b \neq a$ to all internal lattice points of the form $(i, j) = (i_1, j_1) + l(i_{\text{gcd}}, j_{\text{gcd}})$ with $l_0 < l \leq \min\{m/(i_1 - i_2), m/(j_1 - j_2)\}$ and a to the rest of the internal lattice points. Hence, $(i_{\text{gcd}}, j_{\text{gcd}})$ is a source since every a is mapped onto an a or a $*$ and every b is mapped onto a b or a $*$. If we apply Lemma 2.1 k-times, then $(i_1 - i_2, j_1 - j_2)$ is also a source which contradicts $P[i_2, j_2] = b \neq a = P[i_1, j_1]$. If $j_1 < j_2$, then a similar argument applies by using Lemma 2.2.

A similar argument holds for *NS*-groundedness.

Now we show the if direction. More precisely, we show that if $P[i_1, j_1]$ and $P[i_2, j_2]$ are top-sources, then $P[i_1 + i_2, j_1 + j_2]$ is also a top-source. This is equivalent to the claim of Lemma 2.1. For Lemma 2.2 an analogous argument holds.

Hence, we have to show that $P[i, j] \cong P[i + i_1 + i_2, j + j_1 + j_2]$, for all i, j. So suppose $P[i, j] \neq *$ and $P[i + i_1 + i_2, j + j_1 + j_2] \neq *$.

Either $P[i + i_1 + i_2, j]$ or $P[i, j + j_1 + j_2]$ is an internal lattice point by the *NS*-groundedness of F, say $P[i + i_1 + i_2, j]$. Furthermore, by Observation 3.1 either $P[i + i_1, j + j_1]$ or $P[i + i_2, j + j_2]$ belongs to the triangle T spanned by $P[i, j]$, $P[i + i_1 + i_2, j + j_1 + j_2]$, and $P[i + i_1 + i_2, j]$, say $P[i + i_1, j + j_1]$. Again by Observation 3.1 T belongs to F. Since $P[i_1, j_1]$ and $P[i_2, j_2]$ are sources, we have $P[i, j] = P[i + i_1, j + j_1] = P[i + i_1 + i_2, j + j_1 + j_2]$. □

A pattern form that satisfies Theorem 3.2 is called a *consistent pattern form.*

3.1 Recognition of Consistent Pattern Forms

Above we have given necessary and sufficient conditions for a pattern form to be consistent. In this section we want to address the question of how to recognize if a given pattern form is consistent. We assume that we are given a description of the pattern form as two sequences of intervals $C_1, \ldots, C_{m_2}$ and $R_1, \ldots, R_{m_1}$ where interval $C_k = [i_k, i'_k]$ is the interval of internal lattice points in column k and $R_k = [j_k, j'_k]$ is the interval of internal lattice points in row k.

In order to test the convexity of F we first compute the convex hull of F which can be done in time $O(\min\{m_1, m_2\})$ because the boundary points of F which delimit the intervals C_k and R_k are already sorted [PS85]. Given the convex hull of F is easy to check whether an external lattice is contained in the hull or not. If there is an external lattice point X that is contained in the convex hull of F, then we can clearly find three internal lattice points as in Observation 3.1 such that the triangle T spanned by them contains X and, hence, F is not convex.

The second condition of Theorem 3.2 is not as easy to check. If we consider one of the left-, top-, right-, bottommost internal lattice points

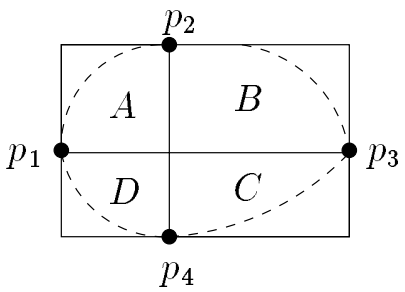

Figure 4: Decomposing a pattern form into four subforms.

$p_1 \ldots, p_4$ of the pattern, it is easy to see that the pattern can be decomposed into the four rectangles A, B, C, and D as illustrated in Figure 4. The dashed line indicates the boundary of the pattern form. Note that such a decomposition can always be found because of the *NS*-groundedness of the points $p_1, \ldots, p_4$. The second condition of Theorem 3.2 holds trivially for all pieces of our decomposition since, for instance, A is S-grounded. But if we put the rectangles together, problems may arise. If we put A together with B or D, then it is again easy to see that every point is *NS*-grounded. Hence, we only have to check the second condition of Theorem 3.2 for boundary points of A against all boundary points of C and analogous for boundary points of B and D.

We demonstrate how to test for *NS*-groundedness between the boundary points of A and C. We consider the leftmost point in each row of A which we assume to be numbered from 1 to A_{max}. Let W_k be the leftmost point and X_k the rightmost point of F in row k. Let Y_k be the lowest point in the column of X_k and Z_k the lowest point in the column of W_k. With the information provided by the intervals C_j and R_i all four points can be computed in constant time. We have the following lemma.

Lemma 3.3 *Let F be a convex pattern form decomposed into the rectangles A, B, C, and D and k an index of a row that belongs to A. W_k is NS-grounded in F if and only if either $Z'_k = S(W_k, Y_k)$ belongs to F or $X'_k = N(Y_k, Z_k) + (1, 1)$ does not belong to F.*

Proof: Note that for all boundary points Y that belong to C and that are to left of Y_k, i.e., the column index of Y is less than the column index

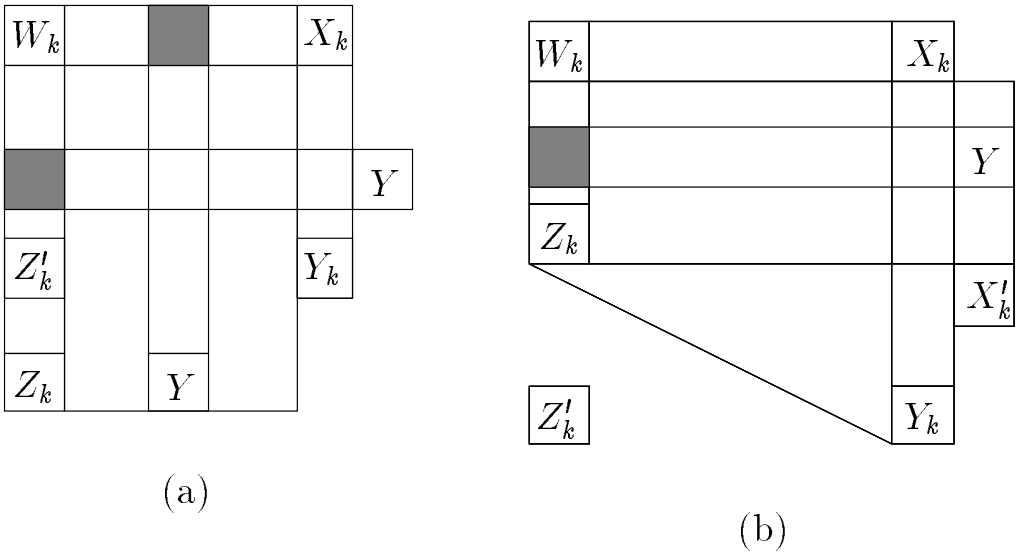

Figure 5: The two cases of Lemma 3.3.

of Y_k, $N(W_k, Y)$ is between W_k and X_k. Hence, by the convexity of F $N(W_k, Y)$ belongs to F (see Figure 5a). Let Y be a boundary point of C to the right of Y_k. Y is at least as high as Y_k, i.e., the row index of Y is less or equal to the row index of Y_k. If Z'_k belongs to F, then Y is above Z'_k and, hence, $S(W_k, Y)$ is between W_k and Z_k which again implies that $S(W_k, Y)$ belongs to F.

Now assume that X'_k does not belong to F (see Figure 5b). This implies that the row index of Y is lower than that of X'_k. Since X'_k has a row index that is one higher than that of Z_k, $S(W_k, Y)$ is between W_k and Z_k, hence, because of the convexity belongs to F.

Finally, assume that Z'_k does not belong to F but X'_k does. Since $S(W_k, X'_k)$ is one below Z_k, it does not belong to F. Furthermore, since $N(W_k, X'_k)$ is one to the right of X_k it does also not belong to F. Hence, W_k is not *NS*-grounded. □

With above lemma we can solve the problem in the following way.

Algorithm ***NS*-groundedness**
for $k := 1$ **to** A_{max} **do**
  compute W_k, X_k, Y_k, and Z_k;
  $Z'_k := S(W_k, Y_k)$;

$X'_k := N(Y_k, Z_k) + (1,1)$;
if Z'_k is an external lattice point
and X'_k is an internal lattice point
then F is not *NS*-grounded
end for;

We conclude that provided an interval representation of the pattern is available it can be checked in time $O(\min\{m_1, m_2\})$ if the pattern form is consistent or not.

4 The ABF-Algorithm for Non-Rectangular Arrays

In this section we show how to generalize the two dimensional string matching algorithm of Amir, Benson, and Farach to consistent pattern forms. W.l.o.g. suppose that the pattern is given in a square $m \times m$ array.

The first step is to identify all sources of the pattern. In order to do so let $r_k = P[k, i_k]P[k, i_k + 1] \cdots P[k, i'_k]$ be the kth row of F. We construct the word $M := r_0 r_1 \cdots r_{m-1}$, i.e., the concatenation of the rows without "$*$". For M we construct a suffix tree in time $O(m^2 \log |\Sigma|)$. In the following we show how to compute all sources (and witnesses against other locations) in column j. We define $s_j(i)$ as the suffix of row i starting at column j and $p_j(i)$ as the prefix of row i of length $m - j$. $P[i, j]$ is a top-source, if $s_j(i) \cong p_j(0), s_j(i+1) \cong p_j(1), \ldots, s_j(m-1) \cong p_j(m-1-i)$. The strings $s_j(i)$ and $p_j(k)$ can be compared in constant time because the region of overlap with characters $c \neq *$ of $s_j(i)$ and $p_j(k)$ can be compared in constant time by making a lowest common ancestor query in the suffix tree for M as described in [ABF92]. A witness is the character which we find if $s_j(i) \neq p_j(k)$.

Let $P_j = p_j(0)p_j(1) \cdots p_j(m-1)$ and $S_j = s_j(0)s_j(1) \cdots s_j(m-1)$ where $p_j(k)$ and $s_j(k)$, $0 \leq k \leq m-1$ are now considered as *characters* of the words P_j and S_j. Let $S_j(i) = s_j(i) \cdots s_j(m-1)$. With a variation of the algorithm by Knuth, Morris, and Pratt [KMP77] we now compute, for each i, the longest prefix of the word P_j that matches a prefix of $S_j(i)$.

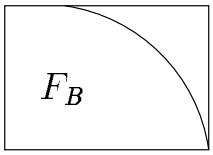

Figure 6: The form F_B of B.

This can be done with $O(m)$ comparions between the strings $s_j(k)$ and $p_j(l)$. If this prefix has the length $m - i$, then $P[i, j]$ is a top-source.

The computation of the list of pairwise consistent candidates K is exactly as described earlier.

Finally, we have to consider the problem of the verification of the candidates. We again divide the pattern into the previously defined rectangles A, B, C, and D and solve the problem for each class of rectangles separately. So we split the list of consistent candidates up into for lists K_A, K_B, K_C, and K_D. We illustrate the algorithm for the candidates in K_B. W.l.o.g. we assume that the upper left corner of B has coordinates $(0, 0)$. The form F_B of B is as displayed in Figure 6. The problem is to assign one coordinate to all text positions that are overlapped by candidates in K_B. The main idea is to mark the left and right boundary of the *union* of candidates which have their left boundary in the same column. Let (i, j) be a text position. There may be several candidates that contain (i, j) on their left boundary since candidates are allowed to overlap. Among these we choose the candidate with the lowest row number and mark its right boundary since its right boundary has the largest column index j' (see Figure 7). By choosing j' we can be sure that if we mark all the points between (i, j) and (i, j'), then none of the text positions that are in the region of candidates which have (i, j) on their left boundary will be missed.

In order to keep track of the candidates that overlap (i, j) with their left boundary we maintain a queue L. We also make use of the interval representation of B with interval $R_i = [0, j'_i]$ for row i. The indizes of the rows are in the range from 0 to B_{max}. For convenience we assume that the indizes of the candidates are directly written into the text T. In the beginning we have $T[i, j] = nil$, for all i, j. The algorithm can now be

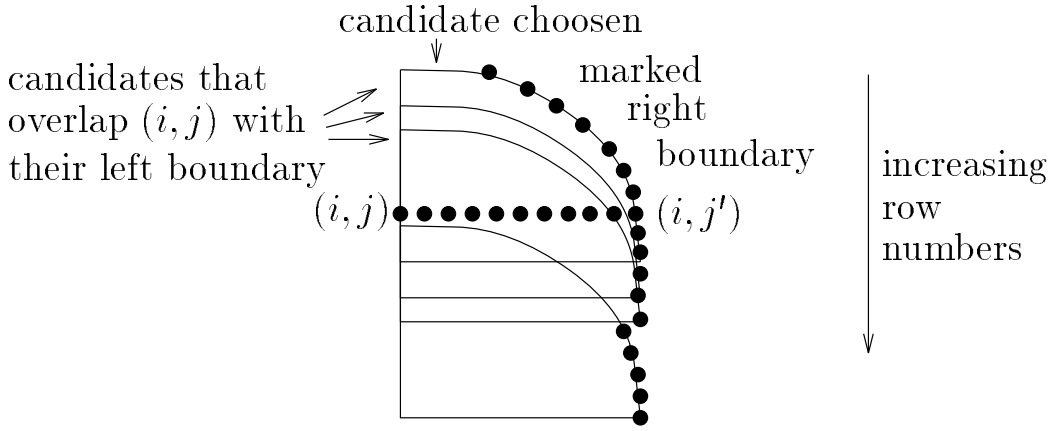

Figure 7: How to assign candidates to text positions without missing any overlapped text position.

described as follows. $i_{first(L)}$ is used to denote the row index of the first element in L.

Algorithm Candidate Verification
(1) **for all** $(i,j) \in K_B$ **do** $T[i,j] := (0,0)$;
(2) **for** $j := n - B_{max}$ **downto** 0 **do**
  $L := \emptyset$;
  for $i := 0$ **to** $n-1$ **do**
    if $T[i,j] = (0,0)$
      then $L := L \cup \{(i,j)\}$;
    if $L \neq \emptyset$ **and** $i - i_{first(L)} = B_{max}$
      then $L := L \setminus \{first(L)\}$;
    {mark right boundary }
    if $L \neq \emptyset$
      then $i' := i - i_{first(L)}$; $j' := j'_{i'}$
        $T[i, j+j'] := (i', j')$;
    {mark left boundary }
    if $T[i,j] \neq (0,0)$ **and**
      $T[i-1,j] = (k,0)$ **and** $k < B_{max}$
      then $T[i,j] := (k+1, 0)$;
  end for;

```
    end for;
(3) for i := 0 to n − 1 do
      for j := n − 2 downto 0 do
        if T[i, j + 1] = (i', j') and j' > 0
          then if T[i, j] = nil or
                    (T[i, j] = (i'', j'') and j'' < j')
                    then T[i, j] := (i', j' − 1);
      end for;
    end for;
```

In Step (1) we mark the left upper end points of all candidates in K_B. With Step (2) we mark the right and left boundary of all candidates. Finally, in Step (3) we mark in each row i the positions that are covered by a candidate from right to left where the column index of the marking indicates how far the left boundary of the candidate is away. We always choose the candidate with the greatest distance to its left boundary.

It is easy to see that the algorithm is correct since in the end the text position $T[i, j]$ contains pair (i', j') if j' is the greatest column index of a character in the pattern which overlaps $T[i, j]$ which is the invariant of the second **for**-loop.

After comparing each text position (i, j) with $P(i', j')$ if (i, j) was marked with (i', j') we have to mark all the candidates that overlap mismatches. This can be done analogously to the candidate verification algorithm.

5 Conclusions

In this paper we show how to adapt the algorithm by Amir, Benson, and Farach to triangular and other non-rectangular patterns that satisfy the transitivity of the consistent relation. We are able to characterize all pattern forms that are consistent and give efficient algorithms to recognize them. It seems that this delimits the classes of patterns for which the techniques developed by Amir, Benson, and Farach work. New ideas if not a totally new approach are needed if other classes of patterns are to be considered.

References

[AB92] A. Amir and G. Benson. Two-dimensional periodicity and its applications. In *Proc. 3rd ACM Symp. on Discrete Algorithms*, pages 440 – 452, 1992.

[ABF92] A. Amir, G. Benson, and M. Farach. Alphabet independent two dimensional matching. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 59 – 68, 1992.

[AC75] A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333 – 340, 1975.

[AF92] A. Amir and M. Farach. *Two Dimensional Dictionary Matching.* Technical Report, Georgia Tech, November 1992. to appear in Information Processing Letters.

[AL91] A. Amir and G.M. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 97 – 115, 1991.

[ALV90] A. Amir, G.M. Landau, and U. Vishkin. Efficient pattern matching with scaling. In *Proc. 1st ACM Symp. on Discrete Algorithms*, pages 344 – 357, 1990.

[Bak78] T.J. Baker. A technique for extending rapid exakt-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533 – 541, 1978.

[Bir77] R.S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168 – 170, 1977.

[Gal85] Z. Galil. *Combinatorial Algorithms on Words*, chapter Open Problems in Stringology, pages 1–8. Volume 12 of *NATO ASI Series F*, Springer-Verlag, 1985.

[GP92] Z. Galil and K. Park. Truly alphabet-independent two-dimensional pattern matching. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 247 – 256, 1992.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323 – 350, 1977.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[RK82] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, New York, 1982.

[ZT89] R.F. Zhu and T. Takaoka. A technique for two-dimensional pattern matching. *Communications of the ACM*, 32(9):1110 – 1120, 1989.