Alphabet Independent Optimal Parallel Search for Three-Dimensional Patterns

Marek Karpinski *
Department of Computer Science, University of Bonn

Wojciech Rytter †
Institute of Informatics, Warsaw University and
Department of Computer Science, Univeristy of Liverpool
Chadwick Building, Peach Street, Liverpool L69 7ZF, U.K.

Abstract

We give an alphabet-independent optimal parallel algorithm for the searching phase of three-dimensional pattern-matching. All occurrences of a three dimensional pattern P of shape $m \times m \times m$ in a text T of shape $n \times n \times n$ are to be found. Our algorithm works in $\log m$ time with $O(N/\log(m))$ processors on a *CREW PRAM*, where $N = n^3$. Some ideas from [3] are used. We explore classification of two-dimensional periodicities of faces of the cubic pattern. Some projection techniques are developed to deal with three dimensions. The nonperiodicity implies some sparseness properties, while periodicity implies other special useful properties (*i.e.* monotonicity) of the set of occurrences. Both types of properties are used in deriving our algorithm. The advantage of our approach is that it is essentially two-dimensional, no special properties related to three dimensions and no new complicated data structures are considered, the resulting algorithm is rather simple.

The search phase is preceded by the preprocessing phase (computation of the witness table). Our main results concern the searching phase, however we present shortly a new approach to the second phase also. Usefulness of the dictionaries of basic factors (*DBF*'s, see [9]), in the computation of the three dimensional witness table is presented.
Our algorithms can be easily adjusted to the case of unequally sided patterns.

*Supported in part by the Int. Computer Science Institute, Berkeley, by the DFG Grant KA 673/4-1, and by the ESPRIT BR Grants 7097 and ECUS 030, e-mail: marekcs.uni-bonn.de

†Supported in part by the project KBN 2-1190-91-01

1 Introduction

The problem of *three dimensional matching* (3D-matching, in short) is to find all occurrences of a three dimensional pattern array P in a text array T. By an occurrence we mean the position of the specified corner of P in T in a full exact-match of P against T. For simplicity of exposition we assume that all sides are equal, sides of P are of length m and sides of T are of length n. Assume $m < n$. The total size of T is $N = n^3$ and the total size of P is $M = m^3$. The 3D-matching is a natural generalization of the classical string matching and two-dimensional pattern-matching problems, and aside of applications, of independent algorithmic interest.

The pattern-matching usually consists of two quite independent parts: preprocessing and searching phase. The main role of the preprocessing is the computation of the so called *witness table*, which will be defined later. Let Σ be the underlying alphabet. In two dimensions there are two approaches to compute this table efficiently: use the suffix trees (see [2]), which is a factor $\log|\Sigma|$ slower than linear time, and the linear time alphabet independent algorithms of [11] and [7]. The alphabet independent algorithms are extremely complicated. They would be even more complicated in three dimensions. On the other hand if Σ is large then we can replace $\log|\Sigma|$ by $\log m$. We show a simple approach through the *dictionary of basic factors* (*DBF*, in short). This is a useful data structure introduced in [14]. It has received the name *DBF* and its usefulness in the design of string algorithms was shown in [9]. The advantage of the DBF is that it can be very easily extended to the three dimensional situation. For large alphabets the complexity of the DBF approach is not inferior to that of the suffix trees. In the three dimensional case the DBF works in much simpler way as the suffix trees approach.

Our model of parallel computations is the *Concurrent Read Exclusive Write Parallel Random Access Machine* (*PRAM*, in short, see [12]). In the paper we concentrate on the first phase of the pattern-matching: the searching phase. Amir, Benson and Farah were the first to give alphabet-independent linear time searching phase, see [2]. They have also given in [3] an alphabet-independent searching in $logM$ time with $O(N/\log(M))$ processors of the *CREW PRAM*. We refer to this algorithm as the algorithm ABF. The algorithm ABF needs only the *witness table* from the preprocessing phase. An $O(1)$ time optimal algorithm was given recently in [7], however it needs additional data structure from the preprocessing phase: so called *deterministic sample*.

2 Periodicities, Witnesses and Duels

Our algorithm for the 3-dimensional matching is based on properties of the structure of 2-dimensional periodicities. This structure is quite complicated. Its precise and detailed description would require too much space. We shall use some known algorithms and methods as a kind of a *black box*. Therefore in many places we refer to bibliography for facts and details about the structure of 2-dimensional periods.

The basic precomputed data structure needed in our algorithm is (similarly as in the algorithm *ABF*) the *witness table* WIT. The entries of WIT correspond to vectors (potential periods). The components of each vector are integers, the size of the vector $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ is $|\alpha| = \max(|\alpha_1|, |\alpha_2|, |\alpha_3|)$.

As potential periods the vectors of size at most $c{\cdot}m$ are considered, assuming throughout the paper that $c = 1/8$. We call such vectors *short.* A vector α is a *period* of P if and only if $P(x) = P(x - \alpha)$ for each position x in P, whenever both sides of the equation are defined (correspond to positions in the pattern).

If α is not a period then $WIT(\alpha) = x$ is a witness (to this fact) if $P(x) \neq P(x - \alpha)$.
If α is a period then by convention we set $WIT(\alpha) = 0$.

The nonperiodicity is explored using the operation of a *duel.* If two positions $u, v \in T$ are related through a short vector $\alpha = u - v$ and α is not a period then the operation $duel(u, v)$ "kills" one of these positions in constant time, as a candidate for a match of the whole pattern P inside T.

The occurrence of the pattern cannot start both at u and v. Let $Wit[\alpha] = x$. Then the copies of the pattern placed at the positions u and v (as starting points) both contain the position $v + x$ of the text. However the corresponding positions in these copies of P, which are tested against this position are x and $x - \alpha$, which are distinct due to the definition of the *witness.* Obviously two distinct symbols cannot both match the same symbol, hence one of them disagree and the corresponding candidate point is removed. We refer the reader to [2] and [8] for the details about the *dueling.*

We introduce also the relation $\equiv_c$ of *consistency* between pairs of positions. We write $x \equiv_c y$ if $x - y$ is a period of P. In other words two positions x, y are consistent , if and only if overlaps of copies of P placed in positions x, y agree with each other (though they could disagree with the actual parts of T). The relation $\equiv_c$ defined above is usually not transitive (and not an equivalence relation).

Let us partition the whole text cube T into *cubic windows*, each of the same shape $c{\cdot}m \times c{\cdot}m \times c{\cdot}m$. It is enough to show how to find all occurrences in a fixed window in $O(M)$ time. We have $O(N/M)$ windows. Then the total work would be $O(N)$. Let us fix one window W to the end of the section. The occurrence in W does not mean that the whole P is in W, it just means that the specified corner of an instance of P is in W. Assume this specified corner is fixed. Let it be for example the lower left corner of the top face. We can say that the occurrence is a *starting position* of a match.
Denote by $Occ(W)$ the set of all positions in W which start a match (an occurrences) of P in T.

Assume that the *witness table* WIT has been already precomputed, and consider a set $C \subseteq W$ of candidates for a match (positions which are candidates to be in $Occ(W)$). Consider only patterns which start in W.

We say that a set $C \subseteq W$ is *valid* iff $Occ(W) \subseteq C$. Obviously the set $C = W$ is valid.

We say that a set $C \subseteq W$ is *consistent* iff $x \equiv_c y$ for each $x, y \in C$.

The duels are used:

(1) to remove one of the candidates for an occurrence of the pattern,

(2) to get information that two candidates are consistent.

Hence after each application of a duel to a valid set C of positions C remains valid. If no element can be removed by a duel and C is valid, then C becomes valid and consistent. Such value of C is the outcome of the first substage of the pattern searching.

In the searching phase there are two main goals to achieve.

Goal 1: construct any valid and consistent set $C \subseteq W$.

Goal 2: given a valid and consistent set C of positions of W compute $Occ(W)$

The searching phase has two basic subphases:

Subphase (I): realize Goal 1. **Subphase (II):** realize Goal 2.

Subphase (II) is rather simple compared with (I), and can be done for three dimensions essentially in the same way as for two dimensions, see [2] and [7].

Lemma 2.1 *Assume we have a valid consistent set C of positions in a given window W. Then we can find all occurrences starting in W in $O(\log M)$ time with $O(M/\log(M))$ processors of the* CREW PRAM.

Proof: The basic point is the reduction to the search of a *unary pattern* P' in a *binary text* T'. *Unary* means that P' is a cube consisting of the same symbol "1" repeated. The computation of such patterns essentially reduces to the calculation of runs of consecutive 1's, or to the computation of the first "0" (which is easy in parallel). The reduction to the unary case works in three dimensions essentially in the same way as in two dimensions, see [2].

For each position $x \in T$ we find any element y of C which "covers" this position. This means that the pattern placed at y contains the position x. We place '1' if the symbol on a given position x agrees with the pattern placed at the covering element y. Then the computation is reduced to the pattern-matching problem for unary patterns. This is reduced to several applications of an algorithm computing the longest *runs* of ones. We refer to [2]. •

By a (planar) face of a given cube we mean a set of its points with one of the coordinates fixed. The faces can be external faces or internal faces of the cube.

Let H be a face of the window W, hence it is an $cn \times cn$ square parallel to two of the three axes of the coordinates. We consider all (global) periods of P parallel to H, *i.e.* the vectors of the type $x - y$, where $x, y \in H$. We can classify these periods (with respect to H) in the same way as periodicities in two dimensions. We refer to [2] for definitions of periodicity types.

So the face H can be:

nonperiodic, *lattice periodic*, *radiant periodic* or *line periodic*.

We say also that P has a given (one of four possible) periodicity type with respect to the face H.

We emphasize that the periods $\alpha = x - y$ considered above are parallel to H and have a *planar nature* but they are *global* periods with respect to the whole pattern P which is a 3-dimensional object. *Global* means that, if both sides of the equation are defined, $P(z) = P(z - \alpha)$, for each $z \in P$, not only for $z \in H$.

Our three dimensional matching uses in essential way the classification of (two-dimensional) periodicities of the pattern cube P with respect to its faces.

The set C is called here *consistent with respect to* a face H iff all positions in $C \cap H$ are pairwise consistent.

If we consider only positions on a fixed face H of W then we can treat the pattern as 2-dimensional. Each maximal line of P orthogonal to H at some position x can be treated as a long *composed symbol.* We can use duels between positions of H in the 2-dimensional sense. On a given face the duels can be treated as two-dimensional duels, though the outcome of each duel is determined (in a constant time) somewhere deep in a 3-dimensional object. Then a valid consistent set of positions on a given face can be computed by applying the algorithm ABF, or the algorithm from [7]. This shows the following result.

Lemma 2.2 *Assume that the* witness table WIT *has been already precomputed. For a given face* H *of* W *we can compute in* $O(\log M)$ *time with* $O(m^2/\log(M))$ *processors a valid set* C *which is consistent with respect to* H*.*

We say that P is *1D-nonperiodic* iff it has no short period parallel to one of its edges.

Lemma 2.3 *The 3D-matching can be reduced in* $\log M$ *time using* $O(N/\log(M))$ *processors to the case of* 1D-nonperiodic *patterns.*

Proof: We can decompose the cube P into smaller subcubes if P is *1D-periodic.* These smaller subcubes will be *1D-nonperiodic.* The same argument as reducing periodic to nonperiodic case in one dimensional matching can be applied, see [10]. •

Due to lemma 2.3 we can assume that P is *1D-nonperiodic.* Let us make duels between positions on each line in W parallel to some edge of the cube W. There are $O(m)$ positions on one line. They can be eliminated except at most one position per line by processing each line independently. A given line needs $O(m/\log(m))$ processors to process it in $\log(m)$ time. There are $O(m^2)$ lines, altogether the computation is optimal.

Remark. There are sets $C \subseteq W$ which have quadratic number of points and none two of their points lie on a same line parallel to an edge of W.

Let us also apply the algorithm from Lemma 2.2 to each of $O(m)$ faces H of W. Hence we can assume that we start with some known initial valid set C of positions which satisfies the conditions:

- **(A)** For any line L parallel to an edge of W there is at most one position in $C \cap W$,
- **(B)** C is consistent with respect to each (external or internal) face of W.

3 Searching for 3-Dimensional Patterns

Assume in this section that the *witness table* WIT has been already precomputed. The only operation to eliminate elements of W is of the type $duel(u,v)$: if $u \not\equiv_c v$ then one of positions u, v is removed from C. Our aim is to apply, in parallel, some number of such operations and receive a valid consistent set $C \subseteq W$.

The rough idea how to construct a valid consistent set is: start with $C = W$, then use more and more duels to reduce the size of C, if no duel "kills" any element of C then C is the required set which is valid and consistent. However we cannot make too many duels. We are allowed to make in total $O(M) = O(m^3)$ duels in a fixed window.

Observe that if we know a valid set $C \subseteq W$ such that C is small ($|C| \leq m^{3/2}$) then we can perform duels between each pair in C simultaneously and we are done. We perform at most M duels in total.

Such a situation occurs if P is nonperiodic with respect to (at least) one of its faces H: there is no short period parallel to this face. In this case on each face parallel to H there is at most one element of the candidate set C, if C satisfies property (B). Then $|C| = O(m)$ and we can make duels between each pair of positions in C. Hence in this case we realize easily Goal 1 in $\log M$ time with $O(M/\log(M)$ processors in a given window W.

Therefore we can assume now that P is periodic with respect to each of its faces, also we have a candidate set C which is valid and satisfies (A) and (B).

3.1 Lattice-periodic case

We show that the algorithm in this case is very simple. Its rough idea is as follows: consider all faces parallel to a fixed (lattice-periodic side of a window), choose a candidate from each of these faces and perform duels only between the chosen candidates. If a candidate of some face is *killed* then all other candidates lying on the same face are killed automatically. The effectiveness follows from the fact that we perform small number of duels.

However the proof of correctness is rather technical and we have to consider how *mismatches* can *travel* by shifting them using lattice-periodicity.

Let H be a fixed external 2-dimensional face of W. Assume without loss of generality that

$$H = \{x = (x_1, x_2, x_3) \,:\, 1 \leq x_1, x_2 \leq cm \text{ and } x_3 = 1\}.$$

We can assume also without loss of generality that the window starts at a corner of the cube T. Assume, till the end of this section, that we have a set C satisfying (A), (B) and P is periodic with respect to each of its faces. The *lattice-periodicity* means here that there is a short period parallel to H in quadrant (I) and a short period in quadrant (II), and these vectors are different, see [2] and [8] for details. Denote by $H_k = \{(x_1, x_2, x_3) \in W \,:\, x_3 = k\}$. Let $C_k = H_k \cap C$. So H_k is the k-th

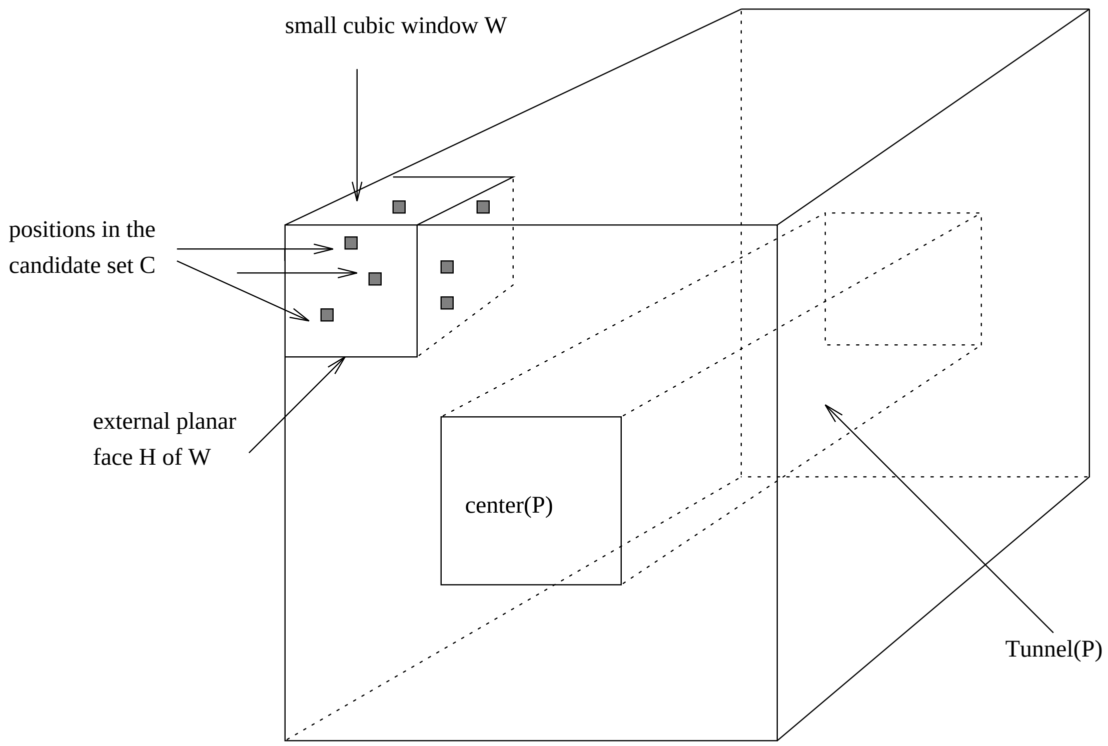

Figure 1: A schematic illustration of H, W and $Tunnel(P)$.

face of W parallel to H and C_k is the subset of all positions in C lying on H_k.

Lemma 3.1
Assume that P is lattice-periodic with respect to the face H, $x \in C_k$ and $y \in C_l$. Then:

(1) *If $x \not\equiv_c y$ and x is removed in the duel between x, y, then all other positions $u \in C_k$ can be removed as candidates for a match.*

(2) *If $x \equiv_c y$ then $u \equiv_c v$ for each $u \in C_k$, $v \in C_l$.*

(3) *The relation $\equiv_c$ restricted to the set $C_k \cup C_l$ is an equivalence relation.*

Proof: Place the pattern P in such a way that it starts at the corner of the face H of the cubic window W. The external face P' of P is parallel to the face H of the window W, see Figure 1.

We can assume that P' is the face corresponding to all points with the third coordinate equal to 1, and H is the part of P' contained in the window W. Let C be a candidate set satisfying (A) and (B) and C' be its projection onto H.

Let $center(P')$ be the central $m/2 \times m/2$ subarray of P'. The following claim says that if a two-dimensional $m \times m$ pattern P' is lattice periodic then witnesses for all nonperiodic short vectors can be found in $center(P')$.

The proof was essentially presented in [3]. Roughly speaking, if a 2-dimensional face is lattice-periodic then we can *move* any point to an equivalent point in the central part of the face using the periods. The points $x = WIT[\alpha]$ and $x - \alpha$ can be moved in such a way to points x', y'. The

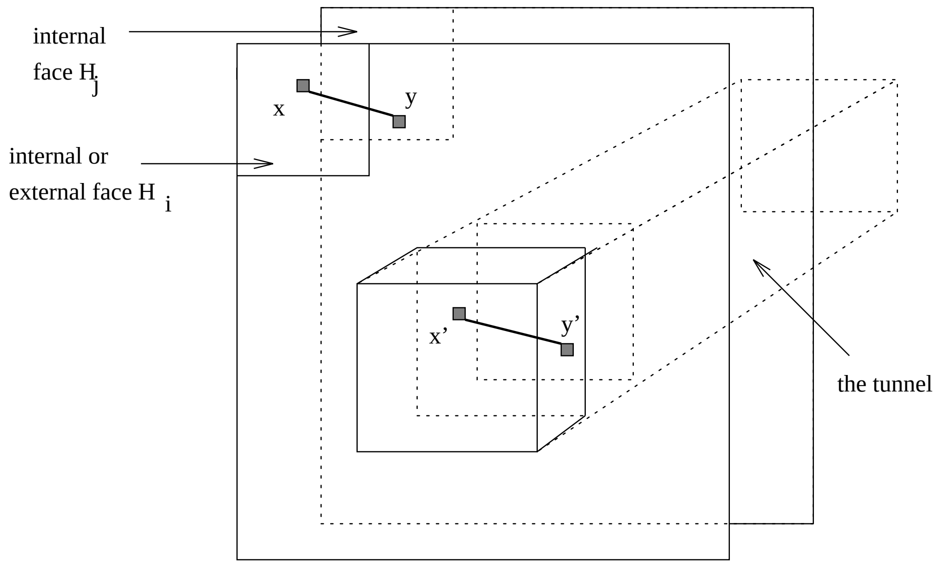

Figure 2: Let $\alpha = x - y$. Assume x, y are not consistent. Then there are corresponding points $x', y' \in Tunnel(P)$.

coordinates of the points x', y' can be computed in constant time by a simple arithmetic. We omit the details and refer to [3].

Claim A Assume a 2-dimensional pattern P' is lattice-periodic. We can modify the table WIT in such a way that if a short vector α is not a period of P' then $WIT[\alpha] = x'$, where $x', x' - \alpha \in Center(P')$. The modification of the table WIT can be done in $\log m$ time with $O(m^2/\log(m))$ processors.

We introduce a 3-dimensional counterpart of the *2-dimensional center*. Let P' be the face of P containing the face H. Denote by $Tunnel(P)$ the set of all positions in P whose projection onto P' lies in $Center(P')$. The same argument as for Claim A works for the following claim.

Claim B (moving into the *tunnel*)
Assume a 3-dimensional pattern P is lattice-periodic with respect to its face H. We can modify the table WIT in such a way that if a short vector α is not a period of P then $WIT[\alpha] = x'$, where $x', x' - \alpha \in Tunnel(P)$. The modification of the table can be done in $\log m$ time with $O(m^3/\log(m))$ processors.

The thesis follows now from the fact that we can move all possible *mismatches* related to any given nonperiodic short vector to $Tunnel(P)$. Also the part of $Tunnel(P)$ intersected by a copy of P placed at any position of a fixed face H_i, depends only on this face, so it is the same for all points in H_i. The latter fact is caused by the small size of the window with respect to P. If we move the window W at distance at most $c \cdot m$ then it will not touch the *tunnel*. More formally we can express it as follows. Let us consider the cube P_1 inside T corresponding to the pattern P which starts at $x \in C_k$, and the cube P_2 corresponding to the situation when P starts at some other point $y \in C_k$. Then observe that

$Tunnel(P_1) \subseteq P_2$ and $Tunnel(P_2) \subseteq P_1$.

This follows from the fact that the face of the window is very small (of shape $\frac{m}{8} \times \frac{m}{8}$). Shifting the patterns in such small window does not affect the inclusion of $Tunnel(P_1)$ in P_2 and *vice versa.*

Due to the common overlap with the *tunnel* all positions of C lying on the same face are equivalent, with respect to the duels with positions on other faces.

We prove now point (1). Assume that $x \in C_k$ is a loser in a duel between x and y. Then there is a vector α such that $x - y = \alpha$ and $WIT[\alpha] \neq 0$. According to Claim B there are two points: $x' = WIT[\alpha] \in Tunnel(P)$, and $y' = x' - \alpha \in Tunnel(P)$.

At one of these points the pattern disagrees with T and due to that the point x is removed. Take some position $u \in C_k$. Due to the observation above, the positions x', y' are also in a copy of P placed at u, since this copy contains the *tunnel* of the copy placed at x.

We know that a copy of P placed at x disagrees with at least one of the points x', y'. A copy placed at u contains both points and is consistent with x, so it should also disagree. Hence the point u should be removed as a candidate. This completes the proof of point (1). Point (2) can be proved by a similar argument. It is enough to prove that if $x \equiv_c y$ and $u \in C_k$ then $u \equiv_c y$. However in this case the points u, x are consistent, so any inconsistence between u and y can be brought to the *tunnel* and affect the consistence between u and x. Hence u is consistent with any position $y \in C_l$, assuming $x \equiv_c y$. The point (3) follows directly from points (1) and (2). This completes the proof. •

Lemma 3.2 *Assume that P is lattice-periodic with respect to some face. Then we can compute $Occ(W)$ in time $\log M$ with $O(M/\log(M))$ processors.*

Proof: Due to Point (3) of Lemma 3.1 the computation of a valid consistent set can be implemented by choosing a representative from each set C_k and then by making duels between all possible pairs of representatives. Each killed representative in some group C_k consequently kills all members of C_k. There are $O(m)$ representatives, one per each C_k. We can make duels between all of them in one parallel step with $O(m^2)$ processors. This completes the proof. •

3.2 The row minima problem for special monotone arrays

We use a simple version of a *row minima problem*, see [4], for special monotone arrays. The processing of the radiant-periodic (the most difficult) case in the next subsection is reduced to this problem.

Assume we have an $m \times m$ zero-one matrix A. We say that A is *strongly monotone* iff the entries of A are in the nonincreasing order along each row (left-to-right) and in the nondecreasing order along each column (top-down). It means, more formally, that:

$$A[i,k] = 1 \text{ implies } A[i,p] = A[q,k] = 1 \quad \text{for any } p \leq k,\ q \geq k.$$

The row minima are given by a vector Θ_A such that $\Theta_A(i)$ is the smallest index of an entry containing 0 in the i-th row . If there is no such entry then it equals $n+1$.

The *row minima problem* consists in computing the row minima vector Θ_A for a strongly monotone zero-one array.

Example
For the array A presented below we have:

$\Theta_A = [1, 3, 3, 4, 6, 6]$

$$\mathrm{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Observation
If A is strongly monotone then the row minima vector is monotone in the following sense:

$$(*) \qquad\qquad i \leq j \text{ implies } \Theta(i) \leq \Theta(j).$$

Observe that the *total* size of the input problem is m^2, but according to the next lemma, Θ_A can be computed only with $O(m)$ work, so the complexity of the computation is only output-sensitive. We assume that the matrix A is already in the memory.

Lemma 3.3
Assume A is a strongly monotone array which is already in the memory, then the row minima problem can be computed in $\log m$ time with $O(m/\log(m))$ processors.

Proof: Observe that one processor can compute $\Theta_A(i)$, for a given i, in logarithmic time using a kind of binary search. So n processors can do the whole job in logarithmic time. We can reduce the number of processors to $m/\log(m)$ using standard techniques. Using $m/\log(m)$ processors we can compute $\Theta(i)$ for all i of the form $k \cdot \log(m)$, for $1 \leq k \leq r$, where $r = m/\log(m)$. We can use the monotonicity property (*) of the vector Θ_A.

Let $\Theta_A(k \cdot \log(m)) = j_k$ for $k \in [1 \ldots r]$. Then we have a sequence of smaller subrectangles:

$$A[0..\log(m), 0..i_1], A[log(m)..2 \cdot \log(m), i_1..i_2], \ldots, A[(r-1)log(m)..r \cdot \log(m), i_{r-1}..i_r],$$

where $A[p..q, l..s]$ denotes the subarray consisting of all entries $A[i, j], p < i \leq q, l < j \leq s$.
The number of resulting subrectangles is $m/\log(m)$, but they can be still too large. We cut each of them (if its width is larger than $\log(m)$) into $\log(m) \times \log(m)$ subsquares. There are altogether $O(m/\log(m))$ such subsquares. Each of them can be easily processed by one processor in logarithmic time. It is similar to the algorithm from [4], however the situation here is simpler due to the applicability of binary search. This completes the proof. •

3.3 Projections and weighted 2-dimensional points

It will be more convenient to deal with 2-dimensional objects, instead of 3-dimensional. It can be done by treating the third component of the points as a *weight*. Let us project the set C onto the face H. The point (x_1, x_2, x_3) is projected onto the point $project(x_1, x_2, x_3) = x = (x_1, x_2)$ of H. The third component is associated with x as its *weight*. We have $weight(x) = x_3$. We write also (x, k), for a point with weight k.

Denote $\Gamma = project_H(C)$, hence Γ is the collection of projected points on H together with their weights.

Due to properties (A) and (B), the points in Γ satisfy the following conditions:

1. each point in C is projected onto a different point in H;
2. if $(x_1 = y_1)$ or $(x_2 = y_2)$ then $weight(x_1, x_2) \neq weight(y_1, y_2)$;
3. if $weight(x) = weight(y)$ then $x \equiv_c y$.

The points (1), (2) follow from the property (A) and the point (3) follows from the property (B) of the candidate set C.

It seems that we reduced the problem to a simpler two-dimensional one. However we have only changed terminology to a more convenient one. We have a collection Γ of points of the two-dimensional square array H. Also we have a witness table for them. It refers to three dimensions but all we need is the operation $DUEL$ which works in constant time for any two points. Hence the *dueling* can be treated as two-dimensional since it involves points on a two-dimensional array. We have to eliminate some points from Γ and be left with the subset of pairwise consistent element, which means that for any two points of H a duel will eliminate none of them. One could try to apply in this situation the two-dimensional algorithm *ABF*. Unfortunately it doesn't work in a straightforward way. The algorithm *ABF* is based on some partial transitivity properties of the consistency relation, see also [8]. These properties are here more complicated due to weights which correspond to the third dimension (and which cannot be neglected).

3.4 Radiant-periodic and line-periodic cases

Let $\Gamma = project_H(C)$. We say that Γ is *row-monotonic* if the weights of points in Γ are increasing in each row or are decreasing in each row of the face H. Analogously define *column-monotonicity* of Γ.

If the two-dimensional pattern is *line-* or *radiant-periodic* then it is known, see [1], that any set of consistent candidates in the 2D-text is monotonic in an unweighted-sense. This means that one of the coordinates is a monotonic function of the second one. Assume that P is radiant-periodic or line-periodic w.r.t. each of its faces. Then the property above holds for all faces. On each face

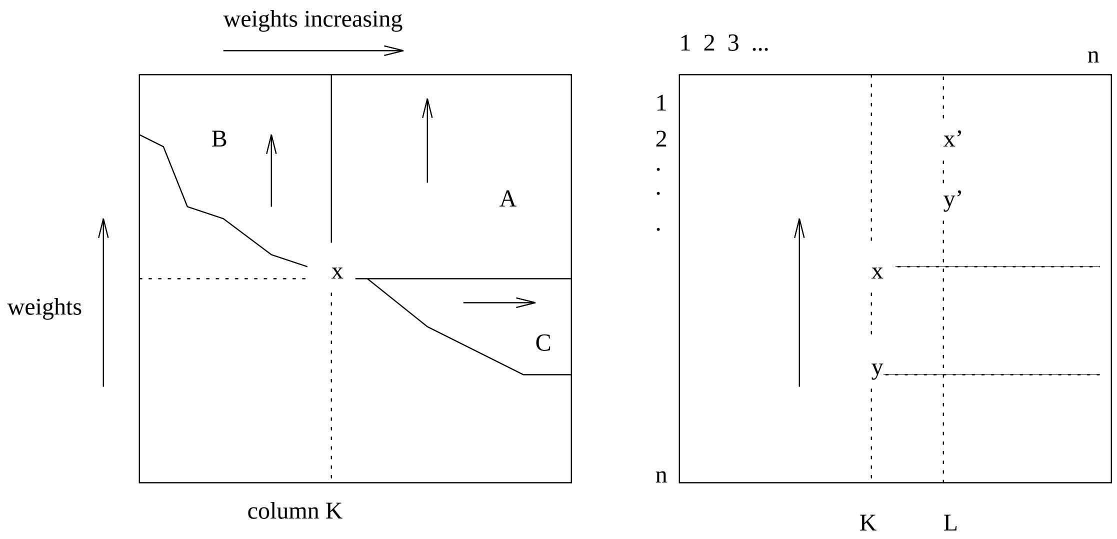

Figure 3: The structure of the set $HeavierThan(x)$ and the relation between $x' = TOP_{K,L}(x)$ and $y' = TOP_{K,L}(y)$.

orthogonal to H the distances of successive points of C from H form monotonic sequences. This implies the validity of the following fact.

Observation.
If P is *line-periodic* or *quadrant-periodic* w.r.t. each of its faces then Γ is *row-monotonic* and *column-monotonic.*

Assume without loss of generality that the weights of points in Γ are increasing in each row left-to-right and increasing in each column bottom-up. The rows are numbered top-down.

For $x, y \in \Gamma$ we write $x \equiv_c y$ iff $(x, k) \equiv_c (y, l)$, where $k = weight(x)$ and $l = weight(y)$. The following observation follows trivially from the symmetry of the relation $\equiv_c$.

Observation
If $x \equiv_c y$ for each two points $x, y \in \Gamma$ such that $weight(x) < weight(y)$, then Γ is a consistent set.

Consider a point x in Γ. We refer the reader to Figure 3 and Figure 4. We explain how to make duels between x and all points in Γ whose weight is larger than the weight of x. Denote the set of these points by $HeavierThan(x)$. Making all possible dells between x and $HeavierThan(x)$ needs quadratic work for a single point x. Altogether it would give $O(m^4)$ work as $|HeavierThan(x)| = O(m^2)$. So we cannot process each point x independently of the others.

The structure of the set $HeavierThan(x)$ is as shown in Figure 3. It consists of 3 parts A, B, C. A cosists of points which are to the right and (horizontally) not below x, B consists of all points in $HeavierThan(x)$ which are to the left of x and C consists of all other points in $HeavierThan(x)$.

In each part the arrows indicate the direction of the weight increase which will be relevant in further computations. We can refer to the part A related to x, as the A-part of x.

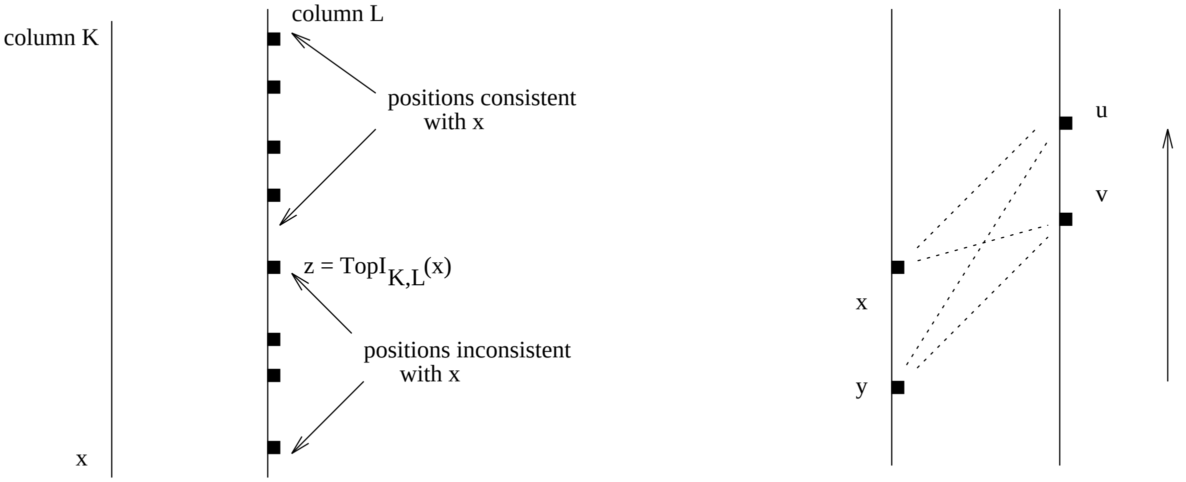

Figure 4: The structure of the positions in $HeavierThan(x) \cap L$

For $x \in K \cap \Gamma$ denote by $TOP_{K,L}(x)$ the topmost position $z \in L \cap \Gamma$ such that z is in the A-part of x and $x \not\equiv_c z$. Such position z is called the *topmost inconsistent position* for x. This position is situated just between consistent and nonconsistent positions with respect to x on this part of line L which is contained in the A-part of x, see Figure 4.

Lemma 3.4

(1) Let $z = TOP_{K,L}(x)$, and assume $u \in \Gamma$ is in the A-part of x and in column L. Then if u is above z then $x \equiv_c u$, otherwise $x \not\equiv_c u$, see Figure 4

(2) if $x, y \in K \cap \Gamma$ and y is below x then $TOP_{K,L}(y)$ is not above $TOP_{K,L}(x)$, see Figure 3.

Proof:

Assume that $x, y, u, v \in \Gamma$. Observe that $x \equiv_c y$ and $u \equiv_c v$, due to the fact that they are pairs of points on the same lines and due to the property (2) of Γ. It is enough to show that for any four points x, y, u, v situated as presented in Figure 4 the following implications hold:

(I1) $x \equiv_c v$ implies $x \equiv_c u$.

(I2) not $x \equiv_c u$ implies *not* $x \equiv_c v$.

(I3) not $y \equiv_c v$ implies *not* $x \equiv_c v$.

We show only the implication *(I1)*.

We refer the reader to [8], pages 258-260, for the properties of partial transitivity of the relation $\equiv_c$ in the 2-dimensional case. The implication above is satisfied in the 2-dimensional case, if the weights are disregarded. Here the crucial point is the monotonicity of weights (which corresponds to the monotonicity in the third dimension - the depth):

$$\text{if } i = weight(x),\ j = weight(v) \text{ and } k = weight(u), \text{ then } i \leq j \leq k.$$

Let $Cube(z)$ be the $m \times m \times m$ cube which starts (with its specified corner) at z. Then it is easy to see that if $x = (x_1, x_2), v = (v_1, v_2), u = (u_1, u_2)$ are situated as in Figure 4 and their corresponding weights i, j, k are nondecreasing then:

$$Cube(x_1, x_2, i) \cap Cube(u_1, u_2, k) = Cube(x_1, x_2, i) \cap Cube(v_1, v_2, j) \cap Cube(u_1, u_2, k).$$

The front faces of the cubes are situated one over the other, and their distance (depth) from the first face is growing. Hence each relation between (related to points in) $Cube(x)$ and $Cube(u)$ can be *decomposed* into a relation between $Cube(x)$ and $Cube(v)$ and a relation between $Cube(v)$ and $Cube(u)$. This completes the proof of the implication *(I1)*. Two other implications can be proved in an analogous way. •

Lemma 3.5 *Assume that the pattern P is quadrant-periodic or line-periodic with respect to each of its faces. Then we can find all occurrences of P in time $\log M$ with $O(N/\log(M))$ processors.*

Proof: We cannot make directly all possible duels for each point x independently, so we make them in an *implicit* way. The processing for x is done in four separate areas.

We explain only how we process the A-parts for all points. Other parts are processed similarly. Each column K is processed independently. Let us fix some column L, see Figure 3.

It is enough to show how to perform duels between all positions x in K against positions in L, which are heavier than x.

Claim.
The vector $TOP_{K,L}$ can be computed in logarithmic time with $O(m/log(m))$ processors.
Proof of the claim.

Assume that the the sequences of points of Γ lying on the line K and line L, in a top-down order, are respectively $x_1, x_2, \ldots x_p$ and $y_1, y_2, \ldots y_q$. Construct the $p \times q$ zero-one matrix A such that:

$$A[i,j] = 1 \text{ iff } x_i \equiv_c y_j \text{ and } y_j \text{ is not in a row below } x_i.$$

Due to Lemma 3.4 the matrix A is monotone. Compute the row minima vector Θ_A for A. Then $\Theta_A(i) = y_j$ iff $TOP_{K,L}(x_i) = y_j$.

In this way the table $TOP_{K,L}$ for two lines K and L is computed , due to Lemma 3.3, in logarithmic time with $O(m/log(m))$ processors. This completes the proof of Claim.

There is the quadratic number of pairs K, L. Altogether $O(m^3/\log(m))$ processors are enough to perform (*implicitly*) duels between each point $x \in \Gamma$ and each point in the A-part of x.

The computations related to the other parts B, C are carried out in the same way. However, for the part B we group points x in rows, instead of columns, and use the horizontal monotonicity in B, see Figure 3.

Now we do not need to make duels between x and all positions in the A-part of x. It is enough to make one *essential* duel between x and the position $TOP_{K,L}(x)$, which represents all positions in the A-part of x lying in the column L. The same works for other parts. There are only few *essential* duels altogether. The number of all such duels is $O(m^2)$. This completes the proof. •

Observe that the complexity of all algorithms considered above did not depend on the size of the alphabet. The series of lemmas above implies immediately our main result.

Theorem 3.6 (main result)
Assume that the witness table is precomputed. Then the 3D-matching problem can be solved by an optimal parallel algorithm working in $\log(M)$ *time on a CREW PRAM. The complexity does not depend on the size of the alphabet.*

Remark It is possible to compute Θ_A in $\log\log m$ time with $O(m/\log(\log(m)))$ processors. Also the algorithm ABF works optimally in $\log\log m$, if the model of computations is a $CRCWPRAM$. Hence our 3-dimensional searching algorithm can be implemented on a $CRCWPRAM$ as an optimal $O(\log\log m)$-time algorithm.

4 Preprocessing the pattern: the DBF approach.

The *dictionary of basic factors* (DBF, in short) is a useful data structure in text algorithms, see [9] for details about DBF and its applications. The DBF approach gains simplicity at the expense of a small increase in time. It gives a (nonoptimal) $O(\log(M))$ time algorithm using $O(M)$ processors of a *CRCW PRAM*. However the alphabet-independent optimal preprocessing is very complex even in the case of two dimensions, see [11]. For large alphabets the DBF's give asymptotically the same complexity as the (alphabet-dependent) suffix trees approach (but avoids suffix trees and is simpler). However the basic advantage of the DBF approach is simplicity of dealing with three (or more) dimensions.

Let $S = \{w_1, \ldots, w_r\}$ be a set of strings. The total size of S, denoted by $||S||$, is the total length of words $w_1, \ldots, w_r$. We want to give consistent names to all subwords of words in S. Each subword z of a word in S can be specified by three integers: a number k of a word w_k which contains z, a position p, where it starts in w_k, and the length l of z. There are quadratic number of such objects with respect to $||S||$. The basic idea of the *dictionary of basic factors* is to have identifiers only for a small subset of all subwords (so called *basic factors*), in a way which enables to identify easily any other subword.

The *basic factors* are subwords whose length is a power of two. The first advantage of basic factors is that there are only $O(||S||\log||S||)$ basic factors, while there are $O(||S||^2)$ subwords in total.

The *dictionary of basic factors* for S, denoted by $DBF(S)$, is a data structure which assigns to each basic factor corresponding to a triple (k, p, l) a unique name $ID(k, p, l)$. The names are integers in the range $1 \dots ||S||$ and two words of the same length are equal (as strings) if and only if their names are the same. The folowing fact was shown in [9].

Lemma 4.1 *$DBF(S)$ can be computed in $\log ||S||$ time with $O(||S||)$ processors of a CRCW PRAM.*

The power of the DBF relies on three facts:

1: *DBF* is small, it stores explicitly information only about $O(||S|| \log(||S||))$ objects.

2: Implicitly the *DBF* gives information about $O(||S||^2)$ objects.

3: The construction of the *DBF* is very simple.

Observation. Assume the *DBF* has been computed. Then equality of any two subwords of strings in S can be checked with $O(1)$ work. Each subword can be split into at most two (maybe overlapped) basic factors and get a constant sized name (composed of at most two smaller ones).

We formulate also the problem of witnesses of a pattern P' *against* P. If x is a position in P', then $WIT[x]$ is a position y in P', such that if we place a copy of P over P' at x, then P, P' disagree at y. If $P = P'$ then it is a reformulation of the witness table definition for a single pattern.

First we demonstrate usefullnes of the DBF on the 1D-pattern and 2D-pattern matching.

1D-matching: assume that we want to compute the value of $WIT[i]$ for each position i in a given string P for which the DBF is computed. We can do it with one processor per each position i in logarithmic time by a kind of a binary search, see [9] for details. Each position has one processor (assigned to this position) which finds a *witness* (if there is any) in $\log m$ time.

2D-matching: assume that we are to compute the witness table for a 2D-pattern P. Consider a fixed k-th column of P. We linearize the problem. Compute $DBF(S)$ for the set S of all rows of P. Place at each position in the k-th row the name of the horizontal word of length $k' = m-k+1$ starting at this position. Observe that k' can be a nonpower of two (but then it can be decomposed into two powers of two and have a composed name of size $O(1)$). Do the same with the first column. In this way we have two strings. We compute witnesses of the first string against the second string using the 1-dimensional method. Consider a fixed position x in the k-th column of P. After linearization it becomes a position x' in the corresponding 1-dimensional string. If the witness for x' is in some position j, then we know that the horizontal strings of length k' starting in the first column and the k-th column in row j are unequal. The mismatch to such inequality is found by the binary search method mentioned above.

This approach extends to three dimensions automatically.

Theorem 4.2 *The three dimensional witness table can be computed in $\log M$ time with $O(M)$ processors of a CRCW PRAM.*

Proof: Consider the (whole) faces

$$P_k = \{x = (x_1, x_2, x_3) \, : \, 1 \le x_1, x_2 \le m \text{ and } x_3 = k\}$$

for $1 \le k \le m$. (Previously we considered only faces of a small window W, the windows are not relevant here.) We show how the computation of witnesses for points in P_k can be reduced to a two-dimensional case for a given k. It works in the same way as the reduction of *2D-case* to *1D-case*.

Let us fix k. Assume that the third coordinate corresponds to the *horizontal direction*. Compute the DBF for all horizontal strings in the cube P. Place at each position in P_1 and P_k the name of the string of size $k' = m - k + 1$ which starts at this position and *goes along* the horizontal direction.

We receive the two-dimensional arrays $\widetilde{P_1}$ and $\widetilde{P_k}$. Compute the witnesses of all positions in $\widetilde{P_k}$ against the pattern $\widetilde{P_1}$ using the two-dimensional method described above.

If the witness for position (x_1, x_2) in $\widetilde{P_k}$ is found at (y_1, y_2) then we know that the witness for (x_1, x_2, k) is at a horizontal string starting at (y_1, y_2, k). We apply the one-dimensional method to two strings of size k' going along the *horizontal direction*.

The binary search method can be applied to find a witness of one horizontal string against the other. In this way we reduce the computation of the three dimensional witness table to the independent computation of m two-dimensional witness tables. This completes the proof. •

Remark.
The theorem can be extended to the k-dimensional case, for any natural fixed k. The proof goes essentially in the same way, a kind of projection is realized by replacing one dimensional strings (going along a fixed dimension) by single symbols: identifiers of these strings. The k-dimensional case is reduced to a $(k-1)$-dimensional one using the DBF. Observe that both the searching phase and the preprocessing phase are now implemented by applying different types of projection techniques.

References

[1] A.Amir, G.Benson, Two dimensional periodicity in rectangular arrays. in Proc. 2nd ACM Symp. on Discrete Algorithms, (1992) 440-452

[2] A.Amir, G.Benson, M.Farach. Alphabet independent two dimensional matching, in Proc. 24th ACM Symp. on Theory of Computation, (1992) 59-68.

[3] A.Amir, G.Benson, M.Farach. Parallel two dimensional matching in logarithmic time. in Proc. 5th ACM Symp. on Parallel Algorithms and Architectures (1993), 79-85.

[4] M. Atallah, R. Kosaraju, An efficient parallel algorithm for the row minima of a totally monotone array, Journal of Algorithms 13:2 (1992) 394-413

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

[6] R.S.Bird. Two dimensional pattern matching. Inf. Proc. letters 6, (1977) 168-170.

[7] R.Cole, M.Crochemore, Z.Galil, L.Gasieniec, R.Hariharan, S.Muthukrishnan, K.Park, W.Rytter. Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions. in Proc. 34th IEEE Symp. on Foundations of Computer Science (1993) 248-258.

[8] M.Crochemore, W.Rytter. Text algorithms, Oxford University Press, New York (1994)

[9] M.Crochemore, W.Rytter. Usefullness of the Karp-Miller- Rosenberg algorithm in parallel computations on strings and arrays. Theoretical Computer Science 88 (1991) 59-62.

[10] Z.Galil. Optimal parallel algorithms for string matching. Information and Control 67 (1985) 144-157.

[11] Z.Galil, K.Park. Truly alphabet independent two dimensional matching, in Proc. 34th IEEE Symp. on Foundations of Computer Science , (1992) 247-256.

[12] A.Gibbons, W.Rytter. Efficient parallel algorithms. Cambridge University Press (1988)

[13] R.Karp, R.Miller, A.Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. in Proc. ACM Symp. on Theory of Computation, (1972) 125-136.

[14] R.Karp, M.O.Rabin. Efficient randomized pattern matching algorithms. IBM Journal of Res. and Dev. 31 (1987) 249-260.

[15] M.Karpinski, W.Rytter. Alphabet independent optimal parallel search for 3-dimensional patterns, preliminary version. in Combinatorial Pattern Matching, eds. M.Crochemore, D.Gusfield, Springer-Varlag LNCS 807, (1994) 125-135

[16] U.Vishkin. Optimal pattern matching in strings. Information and Control 67 (1985) 91-113.