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

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 $\mathcal{O}(N/\log(m))$ processors of a *CREW PRAM*, where $N = n^3$. The ideas from [3] are used. Surprisingly, the extension of the two dimensional matching to the three dimensional one is not a trivial modification. The searching phase in three dimensions explores classification of two-dimensional periodicities of the cubic pattern. Some projection techniques are developed to deal with three dimensions. The periodicites of the patern with respect to its faces are investigated. The nonperiodicities imply some sparseness properties, while periodicities imply other special useful properties (*i.e.* monotonicity) of the set of occurrences. Both types of properties are useful in deriving an efficient algorithm.

The search phase is preceeded 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. Usefullness of the dictionaries of basic factors (*DBF*'s), see [7], in the computation of the three dimensional witness table is presented. The *DBF* approach gains simplicity at the expense of a small increase in time. It gives a (nonoptimal) $\mathcal{O}(\log(m))$ time algorithm using m processors of a *CRCW PRAM*. The alphabet-independent optimal preprocessing is very complex even in the case of two dimensions, see [9]. For large alphabets the *DBF*'s give assymptotically 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.

The algorithm can be easily adjusted to the case of unequally sided patterns.

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* (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 [9] and [6]. 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 [12]. It has received the name *DBF* and its usefulness in string algorithms was shown in [7]. 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 sufix trees. In the three dimensional case the DBF works in much simpler way as the suffix trees approach.

In the paper we concentrate mostly 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 $\mathcal{O}(M/\log(M))$ processors of a *CREW PRAM*. We refer to the latter 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 [6], however it needs additional data structure from the preprocessing phase: so called *deterministic sample*. 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 $|\alpha| = \max(|\alpha_1|, |\alpha_2|, |\alpha_3|)$.

Usually only (potential periods) vectors of size at most $c \times m$ are considered, assume here that $c = 1/8$. We call such vectors *short*. The vector α is a *period* of P iff $P(x) = P(x + \alpha)$ 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 let $WIT(\alpha) = 0$.

We say that P is *1D-nonperiodic* iff it has no short period parallel to one of edges of the pattern cube. Let H be a face of P, it is an $n \times n$ square parallel to two of the three axes of the coordinates. By a face of a given cube we mean a set of its points with one of the coordinates fixed. The faces can be boundary faces or internal faces of the cube. If we consider all (global) periods of P parallel to H then we can classify them in the same way as periodicities in two dimensions. 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) periodicity type w.r.t. face H. We emphasize that we consider *global* periods, so the period w.r.t. H is parallel to H but works globally in the cube P. We refer to [2] for definitions of periodicity types. Our three dimensional matching uses in essential way the classification of

(two-dimensional) periodicities of the pattern cube P with respect to its faces.

2 An alphabet-independent optimal parallel algorithm for the searching phase.

Throughout this section assume that the *witness table* WIT has been already precomputed. Using WIT we can easily decide if P is *1D-periodic*. Also we know (one of four) types of the periodicity for each face of P.

Lemma 2.1 *The 3D-matching can be reduced in* $\log M$ *time using* $\mathcal{O}(N/\log(M))$ *processors (independently of the alphabet) 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 [8]. We omit the details. □

Recall that by short vectors we mean vectors whose size is at most $c \cdot m$. Let us partition the whole text array 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 $\mathcal{O}(M)$ time. We have $O(N/M)$ windows. 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 say that two positions $x, y \in W$ are consistent (and write $consistent(x, y)$), iff overlaps of copies of P placed in positions x, y agree each with other (though they could disagree with the actual parts of T). The searching phase has two basic subphases:

Subphase (I):
compute a consistent set $CAND$ of candidate positions in the window W: if $x, y \in CAND$ then $consistent(x, y)$. No position outside $CAND$ can be an occurrence of P.

Subphase (II):
Verification of $CAND$: check which positions $x \in CAND$ are real occurrences (P placed at x matches the corresponding part of the text cube T).

Subphase (II) is rather simple compared with (I), so we concentrate later mostly on the first subphase.

Lemma 2.2 *Subphase (II) can be implemented to work in* $\mathcal{O}(\log M)$ *time with* $O(N/\log(M))$ *processors of a* CREW PRAM.

Proof: The basic point here 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). It has to be done in each window independently.

The reduction to the unary case works in three dimensions essentially in the same way as in two dimensions, see [2]. Each position in the text "finds" any element of $CAND$ which "covers" this position. The important point is that *any* covering position represents all such positions due to consistency. We place '1' if the symbol on a given position agrees with the pattern placed at the covering element of $CAND$. We omit details. □

The nonperiodicity is explored using the operation of a *duel*. If two positions x, y are related through the vector $\alpha = x - y$ and α is not a period then the operation $DUEL(x, y)$ "kills" one of positions in constant time. The witness table is used. If α is a period then we know that x and y are consistent. We refer the reader to [2] for the details about the *duelling*. The rough idea how to construct $CAND$ 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 $CAND$. However we cannot make too many duels. We are allowed to make in total $\mathcal{O}(M) = O(m^3)$ duels in a fixed window.

Observe that if we know a set C such that $CAND \subseteq C$ and 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.

Due to lemma 1 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 $\mathcal{O}(m)$ positions on one line. They can be eliminated except at most one position per line by processing each line independently. A given line needs $\mathcal{O}(m/\log(m))$ processors to process it in $\log(m)$ time. There are $\mathcal{O}(m^2)$ lines, altogether the computation is optimal.

Therefore we can assume now that in an initial set C of candidates, each line (parallel to an edge of W) contains at most one position of C.

Remark. Unfortunately C can be too large in this moment. We construct such large set C of candidates using the *latin square* strategy. Each row and each column of such square is a permutation of integers $1 \ldots n$. *Pile* m squares one on the other. The first one has candidates on positions containing 1 in the latin square, the second on positions containing 2, etc. Hence the set C has quadratic number of points in a cube and no two points are on the same line (parallel to one of three orthogonal directions).

Lemma 2.3 (simple case)
Assume that at least one of the faces of P is nonperiodic. Then we can find all occurrences of P in $\log M$ time with $\mathcal{O}(N/\log(M)$ processors (independently of the alphabet).

Proof: Assume the face H is nonperiodic. Partition the window W into disjoint squares of shape $c \cdot m \times c \cdot m$. Each square is an external or internal face parallel to H. On each of them we can apply the two-dimensional algorithm *ABF*. Only one candidate remains on any face, due to nonperiodicity w.r.t. H. We have now together (on all internal faces) at most m candidates in the set C of survivors. For each pair $x, y \in C$ we perform $DUEL(x, y)$. The whole computation has $\mathcal{O}(M)$ work since C is small. C is a set of pairwise consistent positions. This completes subphase (I). The second subphase can be done optimally due to Lemma 2.2. □

Assume to the end of the section that P is periodic w.r.t. each of its faces (otherwise we could apply the lemma above). Assume also P is nonperiodic w.r.t. each line parallel to an edge of P. We can preprocess each set of points on each face independently using the two-dimensional algorithm *ABF*. Then we have reduced the situation to the one satisying the following:

- we have an initial set of candidates $C \subseteq W$. Positions outside C in W are known to be nonoccurrences
- for each two points x, y if $x, y \in C \cap H$ for some face H, then $consistent(x, y)$
- there are no two elements of C on the same line parallel to an edge
- P is globally periodic w.r.t. each of its faces.

Let H be an (external) face of the window W. Assume w.l.o.g. that

$$H = \{x = (x_1, x_2, x_3) \ : \ 0 \leq x_1, x_2 < cm \text{ and } x_3 = 0\}.$$

Let us project the set C onto the face H. Assume that H is parallel to the first two axes. The point (x_1, x_2, x_3) is projected onto the point $x = (x_1, x_2)$ of H. The third component is associated with x as its *weight*. We have $weight(x) = x_3$. Let

$$\Gamma = project_H(C)$$

be the collection of projected points on H together with their weights. We write (x, k), for a point with weight k. We write also $weight(x) = k$. Due to the fact that no two points in C are on the same line each point in C is projected onto a different point in H. However Γ can contain many points on the same line, though none two of them can have the same weights.

In a certain sense we reduced the problem to a two-dimensional one. We have a collection Γ of points on the two-dimensional square 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 *duelling* 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 x, y $DUEL(x, y)$ will eliminate none of x, y. 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. These properties are here more complicated due to weights which correspond to the third dimension (and which cannot be neglected). At this moment we can assume that all faces are periodic, otherwise Lemma 2.4 can be applied. The searching depends on the type of periodicity. The *lattice-periodicity* means that a 2D-pattern has a short periodic vector in quadrant (I) and a short periodicity in quadrant (II). However it could happen that both periods are equal, but then the pattern is line periodic according to one of the axes. In the lemma below such possibility is excluded by assuming that the whole pattern is not *1D-periodic*.

Lemma 2.4 *Assume that a* 1D-nonperiodic *cubic pattern P is lattice-periodic w.r.t. one of its faces H. Then we can find all occurrences of P in time* $\log M$ *with* $\mathcal{O}(N/\log(M))$ *processors (independently of the alphabet).*

Proof: Let Γ_k be the set of points in Γ of weight k. We know that all elements of Γ_k are pairwise consistent for a fixed k. If P is *1D-nonperiodic* and lattice-periodic w.r.t. H then the following properties can be proved :

Claim A

Assume $x \in \Gamma_k$ and $y \in \Gamma_l$ for $k \neq l$. Then

1) If in $DUEL(x, y)$ x is "killed" then all positions in Γ_k can be also "killed". They certainly do not start any occurrence of the pattern.

2) If in $DUEL(x, y)$ both elements survive then all positions in $\Gamma_k \cup \Gamma_l$ are pairwise consistent.

We omit the proof of the claim. It is reduced to the following observation:

if a two-dimensional pattern P' is *lattice-periodic* but not *line-periodic* then for each position in the left-upper window of shape $c \cdot m \times c \cdot m$ there is a witness in the central subarray of shape $(1-cm) \times 1-cm)$, if there is any witness at all. This is the only place where we need the constant c to be rather small, $c = 1/8$ is sufficiently small.

Due to the claim the computation of the consistent set $CAND$ can be solved by choosing a representative from each set Γ_k and then by making duels between all possible pairs of representatives. Each killed representative in some group Γ_k consequently kills all memebres of Γ_k. Let Γ' be the set of remaining elements. Each element in fact corresponds to a three dimensional point (we are undoing the projection). This gives the required set $CAND$ as the three-dimensional version of Γ'. Then $CAND$ is the input to the subphase (II). The whole searching can be done optimally due to Lemma 2.2. This completesthe proof. □

Lemma 2.5 *Assume that the pattern P is quadrant-periodic or line-periodic w.r.t. each face of P. Then we can find all occurrences of P in time $\log M$ with $\mathcal{O}(N/\log(M))$ processors independently of the alphabet.*

Proof: Define Γ to be *row-monotonic* if the weights of points in Γ are increasing in each row or are decreasing in each row of H. Analogously define *column-monotonicity* of Γ.
If the two-dimensional pattern is *line-* or *radiant-periodc* then it is known, see [1], that any set of consitent 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. Such property holds for all faces orthogonal to H. The successive points are at agrowing or decreasing distance from H. This implies the following property.
Claim 1. If P is *line-periodic* or *quadrant-periodic* w.r.t. each of its faces then Γ is *row-monotonic* and *column-monotonic.*

Assume w.l.o.g. that the weights of points in Γ are increasing in rows left-to-right and decreasing in columns top-down. Consider a point x in Γ. We refer the reader to Figure 1. We explain how to make duells between x and all points in Γ to the right of x. There are $\mathcal{O}(m^2)$ such points, hence making all possible duells needs quadratic work for a single point x. Altogether it gives $\mathcal{O}(m^4)$ work as there are possible $\mathcal{O}(m^2)$ points x. However we make duells in an implicit way. The processing for X is done in three seprate areas denoted by A, B and C in Figure 1. In area A weights are smaller than $weight(x)$ and in two other areas the weights are larger. We explain only how we process part A, other parts are processed similarly. Each column in A is processed independently. Let us fix some column L, see Figure 1. Let y, z be two points on L, where z is further from x. It is easy to observe the following:

$$P(x) \cap P(z) \subseteq P(y) \cap P(z).$$

where $P(x), P(y)$ and $P(z)$ denote the copies of the cubic pattern P which start at the three dimensional points in T corresponding to x, y and z, respectively (they do not have to match T).
This observation is due to the fact that $weight(x) > weight(y) > weight(z)$ and due to the way how x, y, z are situated on H. It implies the following properties:
(not $consistent(x,z)$) $\Longrightarrow$ (not $consistent(x,y)$) ;
$consistent(x,y) \Longrightarrow consistent(x,z)$.

Denote by $top_incons(x,L)$ the topmost *point* $y \in L \cap \Gamma$ which is inconsistent with x and which is in a row not below x. The above two properties imply that for each $z \in L \cap \Gamma$:
(z is *above* $top_incons(x,L)$) $\Longrightarrow consistent(x,z)$;
(z is *below* $top_incons(x,L) \Longrightarrow$ (not $consistent(x,z)$).

In a certain sense $top_incons(x,L)$ is the "stronger fighter" for x in $L \cap A$. We perform:
if $top_incons(x,L)$ "kills" x in a single duell then x is removed;
otherwise we know (without more duells) that x "kills" all $z \in L \cap \Gamma$ which are below $top_incons(x,L)$ and it does not kill any z which is above.

All that can be done easily by a parallel algorithm for all x's together. Hence it is essentially enough to compute the values $top_incons(x,L)$ for each column L to the right of x. We start with an algorithm which is optimal within logarithmic factor, afterwards we explain how to remove this factor.
Almost optimal algorithm: assume x is fixed. Assign one processor to each column L in area A. This processor finds $top_incons(x,L)$ in $\log m$ time by a binary search method. For a single x we need m processors. Altogether it works in $\log m$ time with $\mathcal{O}(m^3)$ processors.
Optimal algorithm: we have waisted the work by a logarithmic factor due to independency in computing values of $top_incons(x,L)$. Let us now compute these values together for all $x \in K$, where k is some column

preceeding L, see Figure 2. If we can do it for all $x \in K$ in $\log m$ time with $\mathcal{O}(m/\log(m))$ processors then we will have optimality. We use a standard trick of partitioning columns into $\mathcal{O}(m/logm)$ small segments and processing each of them by one processor in logarithmic time. We refer the reader to Figure 2. The column L is partitioned into logarithmic segments, denote the partitioning points by y_k's.. Each partitioning point y_k computes its value $x_k = bottom_incons(y, K)$, where *bottom_incons* is the table analogous to *top_incons* but working for column K against L and in a bottom-down manner. Then for $x \in L$ we know the following:

if x is between x_{k-1} and x_k in K then $top_incons(x)$ is between y_{k-1} and y_k in L.

Unfortunately the segments implied in column K can be larger than logarithmic. We overcome it by refining L with horizontal lines containing points y_k's, see Figure 2. In this way K is divided into $\mathcal{O}(m/\log(m))$ segments, each of logarithmic size and the values *top_incons* for points in a given segment are contained in a known segment of logarithmic size. Then we asign one processor to each segmenet in K which in top-down way computes required values for all members of the segment sequentially in logarithmic time. We use $\mathcal{O}(m/\log(m))$ processors and time is logarithmic. There is the quadratic number of pairs K, L. Altogether $\mathcal{O}(m^3/\log(m))$ processors are enough.

The computation for the area C is carried out in the same way. For the array B we group points x in rows, instead of columns. We omit technical details. The computation of the tables *top_incons* and their counterparts for areas B and C were the bottleneck. Other parts can be easily done by an optimal algorithm. This completes the proof. □

The series of lemmas above implies immediately our main result:

Theorem 2.6 *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.*

3 Preprocessing the pattern: the DBF approach.

Let S be a set of strings. Each subword of a word in S is specified by two integers: a position p, where it starts, and the length l. (All words of S can be concatenated, so a single position can determine where and in which word a given subword starts.) *Basic factors* are subwords whose length is a power of two. $DBF(S)$ is a data structure which assigns to each basic factor corresponding to a pair (p, l) a unique name $ID(p, l)$. The names are integers in the range $1 \ldots |S|$ and two words of the same length are equal (as strings) if and only if their names are the same. The following fact was shown in [7].

Lemma 3.1 *$DBF(S)$ can be computed in* $\log|S|$ *time with* $\mathcal{O}(|S|)$ *processors of a CRCW PRAM.*

The power of the DBF relies on two facts:

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

2: Implicitely *DBF* gives information about $\mathcal{O}(|S|^2)$ objects. 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). Equality of two subwords can be checked with $\mathcal{O}(1)$ work.

We demonstrate first usefullnes of the DBF on the 1D-matching and 2D-matching.

1D-matching: Assume 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 binary search. For a given position i names of basic factors whose lengths are decreasing powers of two are compared successively. Each position has one processor (assigned to this position) which finds a *witness* (if there is any) in $\log m$ time.

2D-matching: Assume 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 $m-k$ starting at this position. Observe that $m-k$ can be a nonpower of two (but then it can be decomposed into such powers and have a composed name). Do the same with the first column. In this way we have two strings. We compute witnesses in the second string w.r.t. the first one by the 1D-method. Consider a fixed position x in the k-th column of P. After linearization it becomes some position x' in 1D-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 unequl. The mismatch is found by the binary search method mentioned-above. A *witness* for the position x is found. This approach extends to three dimensions automatically.

Theorem 3.2 *The three dimensional witness table can be computed in* $\log M$ *time with* $\mathcal{O}(M)$ *processors of a CRCW PRAM.*

Proof: Consider the (whole) faces

$$P_k = \{x = (x_1, x_2, x_3) \,:\, 0 \leq x_1, x_2 < m \text{ and } x_3 = k\}$$

for $0 \leq k < m$. (Previously we considered only faces of the 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_0 and P_k the name of the string of size $m-k$ which starts at this position and *goes* in the horizontal direction. We receive the two-dimensional arrays $\widetilde{P_0}$ and $\widetilde{P_k}$. Compute the witnesses of all positions in $\widetilde{P_k}$ against the pattern $\widetilde{P_0}$ 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 $m-k$ going into the *horizontal direction*. The binary search described before 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. □

References

[1] A. Amir, G. Benson. *Two dimensional periodicity in rectangular arrays.* SODA'92, 440-452

[2] A. Amir, G. Benson, M. Farach. *Alphabet independent two dimensional matching.* STOC'92, 59-68.

[3] A. Amir, G. Benson, M. Farach. *Parallel two dimensional matching in logarithmic time.* SPAA'93, 79-85.

[4] 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.

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

[6] 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.* FOCS'93.

[7] 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.

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

[9] Z. Galil, K. Park. *Truly alphabet independent two dimensional matching.* FOCS'92, (1992) 247-256.

[10] Z. Kedem, G. Landau, K. Palem. *Optimal parallel prefix-suffix matching algorithm and application.* SPAA'89 (1989) 388-398.

[11] R. Karp, R. Miller, A. Rosenberg. *Rapid identification of repeated patterns in strings, trees and arrays.* STOC'72 (1972) 125-136.

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

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

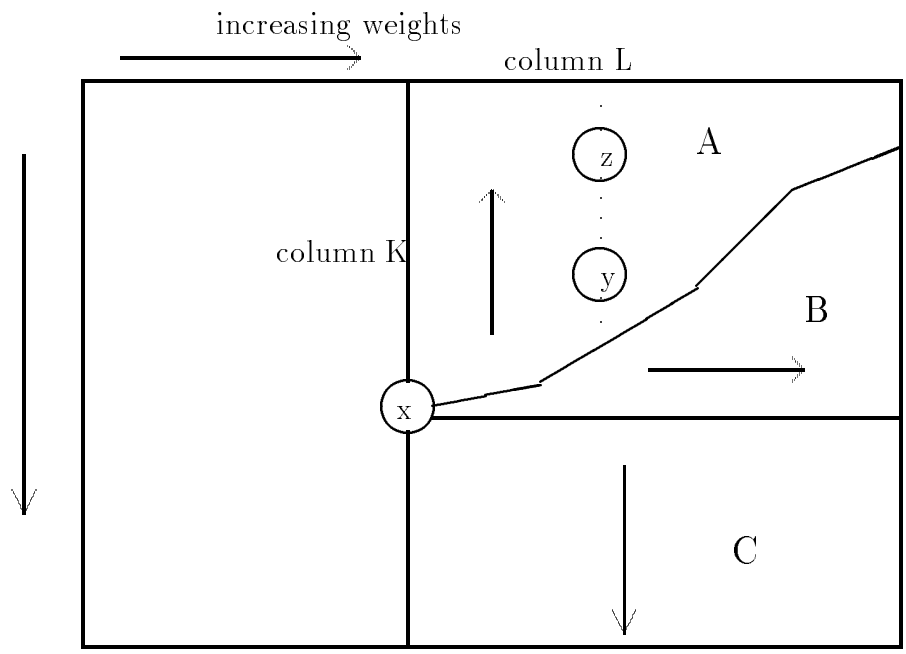

Figure 1: The duels between x and points in L to the right of x are done in three separate areas.

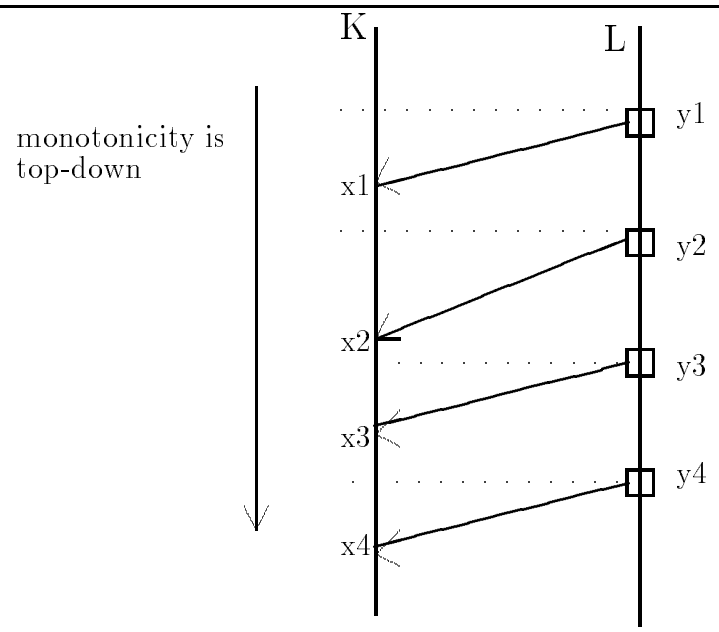

Figure 2: The column L is partitioned into small segments. The inverse pointers partition K. This is refined by the dotted lines.