A Sublinear Algorithm for Two-Dimensional String Matching

Jorma Tarhio*

Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Abstract

A simple algorithm based on the Boyer-Moore idea is presented for two-dimensional string matching. The algorithm examines a strip of columns at a time, and the shift of the pattern is based on a string of several characters on a row. The expected running time is shown to be sublinear for random texts and patterns. The algorithm is easy to implement, and it works well in practice.

Key Words and Phrases: Two-dimensional string matching, Boyer-Moore algorithm

1 Introduction

The task of two-dimensional string matching is to find all occurrences of a two-dimensional pattern P ($m \times m$ characters) in a two-dimensional string text T ($n \times n$ characters) in an alphabet of c characters.

The trivial algorithm for two-dimensional string matching needs $O(m^2n^2)$ time in the worst case. Baker (1978) and Bird (1977) independently gave the first linear time algorithm working in $O(n^2 + m^2)$ time. Later Galil and Park (1992) presented a better linear time algorithm which is independent of the alphabet.

In the following we will concentrate on the expected time complexity, because the average case is more important in practice. Zhu and Takaoka (1989) introduced an algorithm that scans a preprocessed text in sublinear expected time. Baeza-Yates and Régnier (1993) discovered the first on-line sublinear algorithm which runs in $O(n^2/m+m^2)$ expected time. Recently Kärkkäinen and Ukkonen (1994) showed that $O(\frac{n^2}{m^2}\log_c m^2)$ is the lower bound for expected running time and presented an optimal algorithm which achieves this bound

*This work was supported by the Academy of Finland and the Finnish Cultural Foundation, and was done while the author was visiting the University of California at Berkeley. Electronic mail: Jorma.Tarhio@Helsinki.FI.

with additional $O(m^2)$ time for preprocessing. The lower bound agrees with the one for one-dimensional case presented by Yao (1979). These analyses are valid for random texts and patterns.

The algorithm presented by Boyer and Moore (1977) with its many variations is an efficient solution for one-dimensional string matching. It scans the pattern from right to left and is able to skip portions of the text achieving sublinear average behavior. We will present an algorithm for two-dimensional string with a similar shift heuristic as presented by Horspool (1980) for the one-dimensional Boyer-Moore algorithm. Our algorithm examines a strip of r columns at a time, $r \leq m$. Instead of inspecting a single character for shift at each stop of the pattern, the algorithm examines a d-gram, a string of d characters on a row.

We will show that the expected running time of our algorithm is $O(\frac{n^2}{m^2}\lceil\log_c m^2\rceil + cm^2)$, when $m/2 \leq r \leq m$ and $d = \lceil\log_c(rm)\rceil$. The scanning time $O(\frac{n^2}{m^2}\lceil\log_c m^2\rceil)$ matches with the optimal bound of the Kärkkäinen-Ukkonen algorithm. Recently a related but different method has been (independently) developed by Kim and Shawe-Taylor (1993).

The Boyer-Moore approach has also been applied in earlier algorithms for two-dimensional matching. Zhu and Takaoka (1989) employ it along columns, and Baeza-Yates and Régnier (1993) use it in two ways: Every m^{th} row of the text is inspected, and on each such row, the rows of the pattern are searched for by using the one-dimensional Boyer-Moore approach for multiple patterns.

The rest of the paper is organized as follows. The basic algorithm is presented in Section 2 and analyzed in Section 3. A linear time version and other modifications are presented in Section 4. Our experiments are reviewed in Section 5 before concluding remarks in Section 6.

2 Algorithm

The characteristic feature of the Boyer-Moore algorithm for one-dimensional string matching is the right-to-left scan over the pattern. At each alignment of the pattern with the text, characters of the text below the pattern are examined from right to left, starting by comparing the rightmost character of the pattern with the character in the text currently below it. Between alignments, the pattern is shifted from left to right along the text. In the variation presented by Horspool (1980), the shift is based on character x in the text below the rightmost character of the pattern. If x does not occur in the pattern, the pattern is shifted beyond x, otherwise the pattern is shifted to the right until x is below an occurrence of the same character in the pattern. We call this method the Boyer-Moore-Horspool algorithm or the BMH algorithm.

The BMH algorithm has a simple code and is in practice better than the original Boyer-Moore algorithm. Based on the BMH algorithm, we will derive a new algorithm for two-dimensional string matching. The text is split in $\lceil(n-m)/r\rceil + 1$ strips of r columns, $r \leq m$. Each strip is examined separately applying the BMH approach to filtrate potential matches, which are then processed by the trivial algorithm, which checks positions of P in order until a character mismatch is found or until a match of P is completed. Both the

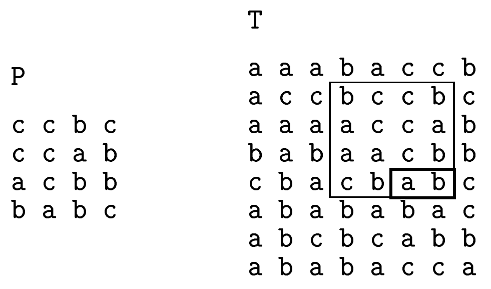

Figure 1: *An example.*

filtration of potential matches and the shifting of the pattern are based on d-grams, i.e. a string of d characters on a row, $r + d \leq m + 1$.

Let us consider a stop of the pattern. For simplicity, let us assume that $r + d = m + 1$. According to the basic idea of Boyer and Moore (1977), the end of the pattern is probed first. So d-gram x, corresponding to the lower right corner of the pattern, is read from the text and compared with the last row of P. If x occurs on the last row of P, we have a potential match, and the corresponding alignment will be further checked by the trivial algorithm (excluding those positions already included in x). Because x may occur in r positions on the last row of P, there can be up to r potential matches at each stop.

For shifting we use a precomputed table D, which follows the shift heuristic of the BMH algorithm. Entry $D[x]$ tells the distance of the closest occurrence of d-gram x in $Y = P[1 : m-1, 1 : m]$ from the last row of P. (Here $P[i_1 : i_2, j_1 : j_2]$ denotes the rectangular region of P with (i_1, j_1) and (i_2, j_2) as the opposite corners.) If x does not occur in Y, $D[x]$ is m.

Fig. 1 shows a situation where the pattern has been aligned at $(2, 4)$ and d-gram $x = T[5, 6 : 7] =$ ab has been read. Because x occurs on the last row of P, there is a potential match at $(2, 5)$, which turns out to be an actual match. Because x appears also on the second row of P, the length of shift will be $m - 2 = 2$ and the next d-gram to be probed will be $T[7, 6 : 7]$.

Above we assumed that $r + d = m + 1$. In a general case we have $r + d \leq m + 1$ and the probe string corresponds to $P[m, r : r + d - 1]$ which is not necessarily in the lower right corner of P.

For the filtration of potential matches we use a preprocessed table M. The entry $M[x]$ tells the starting column of an occurrence of d-gram x in $Z = P[m, 1 : r + d - 1]$, $r + d \leq m + 1$. If x does not occur in Z, $M[x] = 0$. Another preprocessed table N of size m contains a linked list of other occurrences of x in Z, as well as the corresponding linked lists for other d-grams occurring in Z. So $M[x]$ is the first occurrence of x and $N[M[x]]$ is the second occurrence etc.

```
Algorithm 1: Preprocessing of P.
1.  for $i := 0$ to $c^d$ do begin
2.      $D[i] := m$;
3.      $M[i] := 0$ end;
4.  for $i := 1$ to $m$ do begin
5.      $x := P[i, 1]$;
6.      for $k := 2$ to $d$ do $x := x \cdot c + P[i, k]$;
7.      for $j := 1$ to $r$ do begin
8.          if $i = m$ then begin
9.              $N[r - j + 1] := M[x]$;
10.             $M[x] := r - j + 1$ end;
11.         else if $D[x] > m - i$ then $D[x] := m - i$;
12.         if $j < r$ then
13.             $x := (x - P[i, j] \cdot c^{d-1}) \cdot c + P[i, j + d]$ end end;
```

Algorithm 1 describes the computation of tables D, M, and N. The processing of D is based on definition

$$D(y) = \min\{k \mid k = m \text{ or } (k > 0 \text{ and } y = P[m - k, h : h + d - 1], 1 \leq h \leq r)\}.$$

We have now the following total method for two-dimensional string matching.

```
Algorithm 2: Two-dimensional string matching.
1.  compute $D$, $M$, and $N$ with Algorithm 1;
2.  $j := r$;
3.  while $j \leq n - m + r$ do begin
4.      $i := m$;
5.      while $i \leq$ n do begin
6.          $x := T[i, j]$;
7.          for $k := j + 1$ to $j + d - 1$ do $x := x \cdot c + T[i, k]$;
8.          $k := M[x]$;
9.          while $k > 0$ do begin
10.             Check$(i - m + 1, j - r + k)$;
11.             $k := N(k)$ end;
12.         $i := i + D[x]$ end;
13.     $j := j + r$ end
```

Subroutine Check(a, b) on line 10 checks the potential match at (a, b).

3 Analysis

Let us consider the average case complexity of Algorithm 2 without preprocessing. We use the standard random string model, where each character of the text and the pattern is selected uniformly and independently. The time requirement is proportional to C, the number of text characters the algorithm inspects.

Let us estimate $\bar{C}$, the expected value of C. We have

$$\bar{C} = \bar{P}(d + \bar{N}\bar{E})$$

where $\bar{P}$ is the expected number of stops, $\bar{N}$ is the expected number of alignments examined at a stop, and $\bar{E}$ is the expected number of character comparisons for checking of an alignment.

The expected value of shift is $\bar{S} = c(1 - (1 - 1/c)^m)$ for the BMH algorithm (see e.g. Baeza-Yates (1989)). In our approach, we need to replace c by $1/q$, where q is the probability that a d-gram occurs in a $(r + d - 1)$-gram. We will use estimates q_1 and q_2 for q, $q_1 \leq q \leq q_2$, such that

$$q_1 = 1 - (1 - \frac{1}{c^d})^r$$

is the probability that a d-gram is present in a set of r d-grams and

$$q_2 = \frac{rc^{(r+d-1)-d}}{c^{r+d-1}} = \frac{r}{c^d}$$

is the probability that a d-gram without an overlap with itself occurs in a $(r+d-1)$-gram. Now we get

$$\begin{aligned} \bar{S} &= \frac{1 - (1 - q)^m}{q} \\ &\geq \frac{c^d(1 - (1 - q)^m)}{r} \\ &\geq \frac{c^d(1 - (1 - \frac{1}{c^d})^{rm})}{r}. \end{aligned}$$

Let us then consider the number of stops. Let $\bar{P}_1$ be the expected number of stops in one strip. By using similar reasoning as Tarhio and Ukkonen (1993), we get $\bar{P}_1 \leq (n-m+1)/\bar{S}$ for large $n - m + 1$. Thus we get

$$\bar{P} \leq \frac{n - m + 1}{\bar{S}}(\left\lceil \frac{n - m}{r} \right\rceil + 1) \leq \frac{n^2}{r\bar{S}}.$$

When estimating upwards, we can use r/c^d for $\bar{N}$, because r/c^d is the expected number of occurrences of a d-gram in r d-grams.

We have

$$\bar{E} = \frac{c}{c - 1}(1 - \frac{1}{c^{m^2-d}})$$

at the first stop of a strip or when the previous shift is m (see e.g. Baeza-Yates (1989)). In other cases at most d additional positions may be checked. Hence we have $\bar{E} \leq 2 + d$, because $c \geq 2$.

Putting these together, we get

$$\begin{aligned}\bar{C} &= \bar{P}(d+\bar{N}\bar{E}) \\ &\le \frac{n^2 r(d+\bar{N}\bar{E})}{rc^d(1-(1-\frac{1}{c^d})^{rm})} \\ &\le \frac{n^2(d+\frac{r}{c^d}(2+d))}{c^d(1-(1-\frac{1}{c^d})^{rm})}.\end{aligned}$$

When $d = \lceil \log_c(rm) \rceil$ we get

$$\bar{C} \le \frac{n^2(\lceil \log_c(rm) \rceil (1+\frac{r}{rm}) + \frac{2r}{rm})}{rm(1-(1-\frac{1}{rm})^{rm})},$$

because $x(1-(1-1/x)^{rm})$ is an increasing function of x. Then because $(1-\frac{1}{k})^k < \frac{1}{e}$ for all $k > 1$,

$$\bar{C} \le \frac{n^2(\lceil \log_c(rm) \rceil (1+\frac{1}{m}) + \frac{2}{m})}{rm(1-\frac{1}{e})},$$

which is clearly $O(\frac{n^2}{m^2}\lceil \log_c m^2 \rceil)$ when $r \ge m/2$.

The algorithm needs $O(c^d) = O(cm^2)$ space. The computation of D, M, and N takes time $O(m^2 + c^d) = O(cm^2)$, where the initialization of D and M takes $O(cm^2)$ time and the rest of the preprocessing $O(m^2)$ time.

We have shown the following result.

Theorem 1 *Algorithm 2 finds the occurrences of an $m \times m$ pattern P in an $n \times n$ text T in expected time $O(\frac{n^2}{m^2}\lceil \log_c m^2 \rceil + cm^2)$ and in space $O(cm^2)$ in an alphabet of c characters.*

4 Modifications

Linear time version. Because the trivial algorithm is used for checking, Algorithm 2 needs $O(m^2n^2)$ time in the worst case. There is a way to make our approach work in linear time also in the worst case without changing the average complexity.

Let us consider how to modify Algorithm 2. Let k be a positive constant. When kr text characters have been inspected at a stop of the pattern at (i, j), the processing of Algorithm 2 at that stop is ceased and region $T[i : i+2m-1, j : j+r+m-1]$ is processed according to the algorithm by Bird (1977) (or some other linear time algorithm), and after that Algorithm 2 is resumed with a shift of m. The preprocessing for Bird's algorithm is performed only once during the first call.

The modified algorithm clearly works in linear time, because the amortized number of text positions the modified algorithm inspects for each $r \times m$ region of T is at most a constant times rm.

Asymptotically, Bird's algorithm is applied very seldom, when k is large enough, and therefore the average complexity will remain the same.

Rectangular shapes and higher dimensions. Algorithm 2 can easily be modified to work with patterns and texts of arbitrary rectangular shape. The square shape was only used to make the presentation clear.

The generalization to higher dimensions is also obvious.

Bitmap pictures. A bitmap picture is often represented so that a byte corresponds to eight consecutive pixels on a row. It is easy to adopt our method to take advantage of this representation. When $d = 8k$ for $k = 1, \ldots$, the algorithm works for $m \geq 8 + d - 1$ and $r = 8 \cdot \lfloor (m + 1 - d)/8 \rfloor$. As the only additional change, the first strip must be handled in a different way: the initial value of j should be $r_0 = r - 7$.

Use of subpatterns. If the bottom row of the pattern contain d-grams that occur frequently in the text, the checking phase is repeatedly started, which is not desirable for our algorithm. To avoid this phenomenon, a suitable subpattern could be selected for the filtration phase according to some heuristic. For example, a subpattern with a bottom row without d-grams 0 and $c^d - 1$ is usually advantageous for scanning of a bitmap picture. The actual checking is always performed with the original pattern. Also the scanning direction of the algorithm (left-to-right, top-down) can be made optional.

5 Experiences

In Table 1 we compare Algorithm 2 with the algorithm by Baeza-Yates and Régnier (1993), denoted by BYR, and the trivial algorithm in the binary alphabet for $n = 1000$ and for $2 \leq m \leq 64$. For Algorithm 2, we selected $r = \min\{k \mid m + 1 - k \geq \log_2(km)\}$ and $d = \lceil \log_2(rm) \rceil$. The figures in Table 1 represent total execution times in seconds containing preprocessing and checking but excluding loading of the pattern and the text. Each time reported is a median of the results for ten different patterns of the same size. The algorithms were coded in C and the experiments were carried out in a Sun4 workstation.

In this test the trivial algorithm was the best for $m < 5$. Within the range $5 \leq m < 10$ Algorithm 2 and the BYR algorithm were in practice equally good. Algorithm 2 was the best for values $m \geq 10$.

In a similar test with a different implementation, Algorithm 2 was as fast as the Kärkkäinen-Ukkonen algorithm for values $m \geq 20$ and slightly slower for smaller patterns.

6 Concluding Remarks

We have extended the Boyer-Moore-Horspool algorithm into two dimensions. Our algorithm is efficient, conceptionally simple, and easy to implement. The expected scanning time of the algorithm is $O(\frac{n^2}{m^2} \log_c m^2)$. Practical experiments on random strings agree with the theoretical bound.

Our algorithm is related to the algorithm by Kärkkäinen and Ukkonen (1994), denoted by KU, and the algorithm by Kim and Shawe-Taylor (1993), denoted by KST. We will shortly compare our algorithm with these.

Table 1: *Experimental results (in seconds) for $c = 2$ and $n = 1000$.*

m	Trivial	BYR	Alg. 2
2	3.03	9.58	7.50
4	2.70	3.83	2.94
6	2.63	1.81	1.58
8	2.58	1.06	1.01
12	2.55	0.63	0.50
16	2.55	0.55	0.28
24	2.50	0.60	0.13
32	2.45	0.78	0.08
48	2.38	1.52	0.07
64	2.30	2.60	0.07

Our algorithm is functionally different from the KU algorithm, because our algorithm determines the test entries dynamically, whereas the KU algorithm uses a static scheme, where samples of d-grams are taken at fixed grid points of the text.

Our algorithm uses integers to represent d-grams. The KST algorithm uses only tries, and Kärkkäinen and Ukkonen consider the both ways. The number of character comparisons is less for the trie than for the integer representation. However, an implementation based on the trie is in practice slower because of the overhead of a more complicated data structure.

The shape of a test area is a line in our algorithm. Kärkkäinen and Ukkonen consider also a square shape and Kim and Shawe-Taylor also a triangle shape. The line shape seems to be the most efficient in practice.

The expected scanning times of our algorithm and the KU algorithm are the same. Kim and Shawe-Taylor show that the expected scanning time of the KST algorithm is $O(\frac{n^2}{m^2}\log_2 m^2)$ for a fixed alphabet.

Acknowledgement. The author thanks Ricardo Baeza-Yates and Juha Kärkkäinen for providing the the implementations of their algorithms.

References

Baeza-Yates, R. (1989). String searching algorithms revisited. In: *Proceedings of Workshop on Algorithms and Data Structures* (ed. F. Dehne et al.), Lecture Notes in Computer Science 382, Springer-Verlag, Berlin, 75–96.

Baeza-Yates, R. and M. Régnier (1993). Fast two dimensional pattern matching. *Information Processing Letters* **45**, 51–57.

Baker, T. (1978). A technique for extending rapid exact-match string matching to arrays more than one dimension. *SIAM Journal on Computing* **7**, 533–541.

Bird, R. (1977). Two dimensional pattern matching. *Information Processing Letters* **6**, 168–170.

Boyer, R. and S. Moore (1977). A fast string searching algorithm. *Communications of the ACM* **20**, 762–772.

Galil, Z. and K. Park (1992). Truly alphabet-independent two-dimensional pattern matching. In: *Proceedings of the 33st IEEE Annual Symposium on Foundations of Computer Science*, IEEE, 247–256.

Horspool, N. (1980). Practical fast searching in strings. *Software Practice & Experience* **10**, 501–506.

Kärkkäinen, J. and E. Ukkonen (1994). Two and higher dimensional pattern matching in optimal expected time. In: *Proceedings of the Fifth Symposium on Discrete Algorithms*, ACM-SIAM, 715–723.

Kim, J. and J. Shawe-Taylor (1993). Fast expected two dimensional pattern matching. In: *Proceedings of the First South American Workshop on String Processing* (ed. R. Baeza-Yates and N. Ziviani), 77–92.

Tarhio, J. and E. Ukkonen (1993). Approximate Boyer-Moore string matching. *SIAM Journal on Computing* **22**, 243–260.

Yao, A. (1979). The complexity of pattern matching for a random string. *SIAM Journal on Computing* **8**, 368–387.

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