# A Boyer-Moore Approach for Two-Dimensional Matching

Jorma Tarhio*

Computer Science Division
University of California
Berkeley, CA 94720

**Abstract**

An simple sublinear algorithm is presented for two-dimensional string matching, where occurrences of a pattern of $m \times m$ characters are searched for in a text of $n \times n$ characters in an alphabet of $c$ characters. The algorithm is based on the Boyer-Moore idea and it examines a strip of $r$ columns at a time, $m/2 \leq r \leq m$. The shift of the pattern is based on a string of $d$ characters, $d = \lceil \log_c(rm) \rceil$. The expected running time of the algorithm is shown to be $O(n^2 \lceil \log_c m^2 \rceil / m^2 + cm^2)$ for random texts and patterns. The algorithm is easy to implement, and results of experiments are reported to show its practical efficiency.

*Work supported by the Academy of Finland and Finnish Cultural Foundation, and in part by Department of Energy, Office of Energy Research Grant No. DE-FG03-90ER60999. Permanent address: Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland. Electronic mail: tarhio@CS.Berkeley.EDU or Jorma.Tarhio@Helsinki.FI.

# 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 [Bak78] and Bird [Bir77] independently gave the first linear time algorithm working in $O(n^2 + m^2)$ time. Galil and Park [GaP92] developed a linear time algorithm which is independent of the alphabet. Recently Crochemore et al. [CGR93] presented a sampling technique that works in linear time for almost all patterns.

In the following we will concentrate on the expected time complexity, because the average case is more important in practice. Zhu and Takaoka [ZhT89] introduced an algorithm that scans a preprocessed text in sublinear expected time. Baeza-Yates and Régnier [BaR90, BaR93] discovered the first on-line sublinear algorithm which runs in $O(n^2/m+m^2)$ expected time. Kärkkäinen and Ukkonen [KäU93] 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 with additional $O(m^2)$ time for preprocessing. The lower bound agrees with the one for one-dimensional case [Yao79]. These analyses are valid for random texts and patterns.

The Boyer-Moore algorithm [BoM77] 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 heuristics as in the Horspool version of the Boyer-Moore algorithm [Hor80]. Our algorithm is simple but efficient. The 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, at a time. A similar approach has been used for one-dimensional string matching, e.g. by Baeza-Yates [Bae89]. Because a $d$-gram is a kind of a fingerprint of the pattern, our approach is in a way a combination of the Boyer-Moore idea and the fingerprint method presented by Karp and Rabin [KaR87]. 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 and analyzed by Kim and Shawe-Taylor [KiS93]. Their bound for the expected running time is $O((\frac{n^2}{m^2} + m^2)\log_2 m^2)$.

The Boyer-Moore approach is also applied in earlier algorithms for two-dimensional matching. In the Zhu-Takaoka algorithm [ZhT89] the Boyer-Moore principle is used along columns. The Baeza-Yates-Régnier algorithm [BaR93] uses the Boyer-Moore technique in two ways: Every $m^{\text{th}}$ row of the text is inspected, and on each such row, the rows of the pattern are searched for by using 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. The final section reviews our experiments.

# 2 Algorithm

The characteristic feature of the Boyer-Moore algorithm [BoM77] 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 Horspool version [Hor80] 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 filtration of potential matches and the shifting of the pattern are

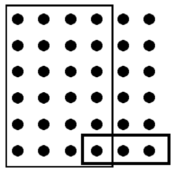

Figure 1: *A 3-gram is covered by $rm = 4 \cdot 6$ alignments of the pattern.*

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 [BoM77], the end of the pattern is probed first. So $d$-gram $x$, corresponding to the lower right corner of the pattern (see Fig. 1), 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 except those positions included in $x$. Because $x$ may occur in $r$ positions of 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 $m$.

After a shift, the $d$-gram under the lower right corner of the pattern is again inspected. Note that a $d$-gram probe is covered by $rm$ alignments of $P$ (see Fig. 1), which means that $rm$ alignments can be skipped, when the probe does not occur in $P$.

Fig. 2 shows a situation where the pattern has been aligned at $(2, 4)$ and and $d$-gram $x = T[5, 6 : 7] = \text{ab}$ 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 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 preprocessed table $M$.

P

| | | | |
|---|---|---|---|
| c | c | b | c |
| c | c | a | b |
| a | c | b | b |
| b | a | b | c |

T

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| a | a | a | b | a | c | c | b |
| a | c | c | b | c | c | b | c |
| a | a | a | a | c | c | a | b |
| b | a | b | a | a | c | b | b |
| c | b | a | c | b | a | b | c |
| a | b | a | b | a | b | a | c |
| a | b | c | b | c | a | b | b |
| a | b | a | b | a | c | c | a |

Figure 2: *An example pattern and text.*

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.*

for $i := 0$ to $c^d$ do begin
    $D[i] := m$;
    $M[i] := 0$ end;
for $i := 1$ to $m$ do begin
    $x := P[i, 1]$;
    for $k := 2$ to $d$ do $x := x \cdot c + P[i, k]$;
    for $j := 1$ to $r$ do begin
        if $i = m$ then begin
            $N[r - j + 1] := M[x]$;
            $M[x] := r - j + 1$ end;
        else if $D[x] > m - i$ then $D[x] := m - i$;
        if $j < r$ then
            $x := (x - P[i, j] \cdot c^{d-1}) \cdot c + P[i, j + d]$ 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 ≤ n − m + r do begin
4.       i := m;
5.       while i ≤ n do begin
6.           x := T[i, j];
7.           for k := j + 1 to j + d − 1 do x := x · 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. [Bae89b]). 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 occurs in $r$ $d$-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 in [TaU93], 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 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. [Bae89b]). In other cases at most $d$ additional may be checked. Hence we have $\bar{E} \leq 2+d$, because $c \geq 2$.

Putting these together, we get

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

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

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

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 $k > 1$,

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

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

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 $O(m^2 n^2)$ 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 by Bird's algorithm [Bir77] (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 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 it is easy to show that 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.

**Shifting.** In our algorithms, multiplication is used to form the representation of a $d$-gram as an integer. Another alternative is to apply shifting. Against common belief there is no difference in efficiency between multiplication and shifting in many computing environments.

**Hashing.** A useful modification is to apply hashing to save space and preprocessing time especially, when $c$ is large. However, hashing makes Algorithm 2 a bit slower. One may fix the sizes of $D$ and $M$ and use some hash function $h$ when referencing to these tables. Note that the minimum value should be stored in $D$ and the link chains in $M$ and $N$ should be united in the case of a collision.

If the actual size of the alphabet is unknown, $m^2$ may be used for $c$. When $c^d$ is so large that is cannot represented as integer, the mod operation should be applied to computation of the integer representations of $d$-grams (see [KaR87].

**Bitmap pictures.** In many microcomputers a bitmap picture is represented so that a byte corresponds to eight consecutive pixels on a row. Our method is easy to adopt to take advantage of that. When $d = 8k$ for $k = 1, \ldots$, the algorithm works for $m \geq 8 + d - 1$ and $r = 8 \cdot \lceil (m + 1 - d)/8 \rceil$.

As the only additional change, the first strip must be handled in a different way: the initial value of $j$ should be $r_0 = r - d + 1$.

**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 could be performed with the original pattern. Also the scanning direction of the algorithm (left-to-right, top-down) can be made optional.

# 5 Experiences

Our algorithm is efficient, conceptionally simple, and easy to implement. Experimental results of the behavior our algorithm on random strings are shown in Table 1, where we compare Algorithm 2 with the Baeza-Yates-Régnier algorithm (BYR) [BaR93] 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. The values are median times of ten repeats of the test with a new set of patterns. The algorithms were coded in C and the experiments were carried out in a Sun4 workstation.

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. For values $m \geq 10$ our algorithm was unquestionably the best.

Algorithm 2 works also quite well with non-optimal values of $r$ and $d$. When $d$ is fixed, the most advantageous value for $r$ is not always $m - d + 1$, as one might expect, but depends on $c$. For example for $d = 1$ and $m = 20$, the best value for $r$ is 1 for $c < 13$, grows with $c$ and reaches 20 when $c = 170$ according to our estimate (1) of the Section 3. Our practical experiments confirm this phenomenon.

In the same time we made experiments in one-dimensional string match-

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

| $m$ | Trivial | BYR | Alg. 2 |
|---|---|---|---|
| 2 | 3.03 | 9.58 | 7.50 |
| 3 | 2.83 | 6.18 | 5.45 |
| 4 | 2.70 | 3.83 | 2.94 |
| 5 | 2.65 | 2.56 | 2.24 |
| 6 | 2.63 | 1.81 | 1.58 |
| 7 | 2.60 | 1.30 | 1.35 |
| 8 | 2.58 | 1.06 | 1.01 |
| 10 | 2.57 | 0.78 | 0.70 |
| 12 | 2.55 | 0.63 | 0.50 |
| 14 | 2.55 | 0.58 | 0.37 |
| 16 | 2.55 | 0.55 | 0.28 |
| 20 | 2.52 | 0.55 | 0.20 |
| 24 | 2.50 | 0.60 | 0.13 |
| 28 | 2.48 | 0.68 | 0.10 |
| 32 | 2.45 | 0.78 | 0.08 |
| 40 | 2.42 | 1.11 | 0.07 |
| 48 | 2.38 | 1.52 | 0.07 |
| 56 | 2.35 | 2.03 | 0.07 |
| 64 | 2.30 | 2.60 | 0.07 |

ing using $d$-grams instead of single characters (like Baeza-Yates in [Bae89]). The speed-up in the two-dimensional case looked much better, because the adequate patterns are larger than in the one-dimensional case.

**Acknowledgement.** We thank Ricardo Baeza-Yates for providing the code of his algorithm.

# References


[Bae89] R. Baeza-Yates: Improved string searching. *Software Practice & Experience* **19** (1989), 257–271.

[Bae89b] R. Baeza-Yates: 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, 1989, 75–96.

[BaR90] R. Baeza-Yates and M. Régnier: Fast algorithms for two dimensional and multiple pattern matching. In: *Proceedings of SWAT90, 2nd Scandinavian Workshop on Algorithm Theory* (ed. J. Gilbert and R. Karlson), Lecture Notes in Computer Science 447, Springer-Verlag, Berlin, 1990, 332–347.

[BaR93] R. Baeza-Yates and M. Régnier: Fast two dimensional pattern matching. *Information Processing Letters* **45** (1993), 51–57.

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

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

[BoM77] R. Boyer and S. Moore: A fast string searching algorithm. *Communications of the ACM* **20** (1977), 762–772.

[CGR93] M. Crochemore, L. Gasieniec, and W. Rytter: Two-dimensional pattern matching by sampling. *Information Processing Letters* **46** (1993), 159–162.

[GaP92] Z. Galil and K. Park: Truly alphabet-independent two-dimensional pattern matching. In: *Proceedings of the 33st IEEE Annual Symposium on Foundations of Computer Science*, IEEE, 1992, 247–256.

[Hor80] N. Horspool: Practical fast searching in strings. *Software Practice & Experience* **10** (1980), 501–506.

[KäU93] J. Kärkkäinen and E. Ukkonen: Two and higher dimensional pattern matching in optimal expected time. To be presented in the Fourth Symposium on Discrete Algorithms.

[KaR87] R. Karp and M. Rabin: Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development* **31** (1987), 249–260.

[KiS93] J. Kim and J. Shawe-Taylor: Fast expected two dimensional pattern matching. In: *Proceedings of the First South American Workshop on String Processing* (ed. R. Baeza-Yates and N. Ziviani), Sept. 13–15, 1993, 77–92.

[TaU93] J. Tarhio and E. Ukkonen: Approximate Boyer-Moore string matching. *SIAM Journal on Computing* **22** (1993), 243–260.

[Yao79] A. Yao: The complexity of pattern matching for a random string. *SIAM Journal on Computing* **8** (1979), 368–387.

[ZhT89] R. Zhu and T. Takaoka: A technique for two-dimensional pattern matching. *Communications of the ACM* **32** (1989), 1110–1120.