Fast Pattern Matching for Entropy Bounded Text *

Shenfeng Chen and John H. Reif
Department of Computer Science
Duke University
Durham, NC 27708

Abstract

We present the first known case of one-dimensional and two-dimensional string matching algorithms for text with bounded entropy. Let n be the length of the text and m be the length of the pattern. We show that the expected complexity of the algorithms is related to the entropy of the text for various assumptions of the distribution of the pattern. For the case of uniformly distributed patterns, our one dimensional matching algorithm works in $O(n \log m/(pm))$ expected running time where H is the entropy of the text and $p = 1 - (1 - H^2)^{H/(1+H)}$. The worst case running time T can also be bounded by $\frac{n \log m}{p(m+\sqrt{V})} \leq T \leq \frac{n \log m}{p(m-\sqrt{V})}$ if V is the variance of the source from which the pattern is generated. Our algorithm utilizes data structures and probabilistic analysis techniques that are found in certain lossless data compression schemes.

1 Introduction

1.1 Pattern matching problem

Given a text of length n and a pattern of length m, the pattern matching problem is to find all occurrences of the pattern in the text. There have been various efficient algorithms designed for both one and higher dimensional pattern matching problems [19,14,17,15]. [15] gave an algorithm which works in $O(n/m)$ time assuming the text is generated from a source of uniform distribution. Many of these algorithms run in $O(n)$ time in the worst case. Two approaches have been adopted in designing these algorithms: to base the analysis of the algorithm either on the frequency of matching or on the frequency of mismatching. In this paper. we analyze the complexity of the algorithms based on the frequency of mismatching. The reason behind this is that it is unlikely for the pattern to match an arbitrary section of the text which has the

*Email addresses: reif@cs.duke.edu and chen@cs.duke.edu. This work was supported under ARPA/SISTO contracts N00014-91-J-1985, N00014-92-C-0182 under subcontract KI-92-01-0182, Rome Labs Contract F30602-94-C-0037, and NSF Grant NSF-IRI-91-00681.

1068-0314/95$4.00 © 1995 IEEE

same length. Thus we are able to skip text sections after a mismatch happens. In other words, when we try to match a pattern against a text, we keep on comparing the characters of the pattern against those of the text until there occurs a mismatch. Then we decide how far we can move the pattern ahead without comparisons.

1.2 Matching on compressed text

Recently, Amir, Benson and Farach[3] studied the text matching problem on compressed text. They gave several implementations of the algorithm where the text is compressed using LZW compression. If u is the length of the *compressed* text and m is the length of the pattern, their algorithm finds the first pattern occurrence in time $O(u + m^2)$ or $O(u \log m + m)$ (the optimal solution of $O(u + m)$ time and space remains open). Amir and Benson[1] developed an almost optimal $O(u \log m)$ algorithm for two dimensional run-length compression, and Amir, Benson and Farach[2] gave an optimal $O(u)$ algorithm for this problem.

1.3 Matching on entropy bounded text

In this paper, we consider the new problem of matching on text which is not compressed but with bounded compressibility. For classes of large data file found in practice, the compressibility of the file is indeed bounded within a constant, no more than say 4 up to 20(see discussion in [6]). This fact has been widely accepted and some novel algorithms have been designed based on this assumption. For example [6], when the large data is not highly compressible, the time for indexing each data according to a pre-computed dictionary can be decreased substantially. Consequently the time for sorting can be decreased to $O(n \log(\frac{\log n}{H}))$ using a sorting scheme similar to the bucket sort where n is the size of the inputs and H is the entropy of the data.

Some previous algorithms also assume that the text is generated from a source which is uniformly distributed (i.e. each character appears in the text with equal probability). We find that this is not a reasonable assumption for practical files and we are not assuming uniform distribution for the text throughout the paper. Instead, we assume that the text is generated from a stationary ergodic source S_T. Also we assume the text has a bounded entropy, i.e. it is not highly compressible. This paper investigates the complexity of the pattern matching algorithm based on KMP algorithm[15] for entropy-bounded text. Therefore, the implementation of our algorithm could mean a substantial improvement for pattern matching in practice.

In our algorithms, the text is considered relatively big and subject to change from time to time. Thus only the pattern is preprocessed in order to make fast matching while text remains unprocessed. This is acutally the case in practice where maintaining the preprocessed data structure for the text could be too expensive comparing to the time saved in pattern matching. We assume that the pattern is generated from a probabilistic source S_P which may be or may not be independent of S_T.

1.4 Sketch of this paper

Definitions and the analysis of one-dimensional pattern matching are given in section 2. We show that our algorithm runs in $O(n \log m/(pm))$ expected time where H is the entropy of the text and $p = 1 - (1 - H^2)^{H/(1+H)}$ assuming the text is generated from a stationary ergodic source. Furthermore we show that if the variance of S_P (the probabilistic source from which the pattern is generated) is V, the worst case running time T could be bounded by $\frac{n \log m}{p(m+\sqrt{V})} \leq T \leq \frac{n \log m}{p(m-\sqrt{V})}$. In section 3, we extend our algorithm to two-dimensional case. A possible extension to lossy matching is discussed in section 4.

2 One Dimensional String Matching

2.1 Matching on Text with Bounded Entropy

Let the text be a sequence of characters generated from an alphabet Σ of size c. Also let n and m denote the size of the text and the pattern respectively. Assume the text is entropy bounded with entropy H. Here we define the entropy with respect to the optimal compression ratio of the text ρ_{opt} where $H = 1/\rho_{opt}$. Note that the Lempel-Ziv compression scheme achieves optimal compression ratio with sufficient large inputs, i.e. $\rho \rightarrow \rho_{opt}$ when $n \rightarrow \infty$. Thus the entropy of the inputs can be estimated by applying Lempel-Ziv scheme to compress the inputs.

Our algorithm works similarly as the algorithm designed by [15]. However, we will analyze the complexity of the algorithm with respect to the entropy of the text.

Lemma 1 *Assuming that the entropy of the text is H, an arbitrary interval Δ in the text of length $(1 + 1/H) \log m$ gets a match in the pattern with probability $p \leq 1 - H^2$.*
Proof: Suppose not. Then the probability that the interval Δ gets a match in the pattern is $p > 1 - H^2$. Assume the optimal compression ratio $\rho_{opt} = 1/H$. We now show a new compression method to compress the text. We divide the text into equal sections each with length Δ.

For each interval that gets a match in the pattern, we encode it using $\log m$ bits to index the position where it appears in the pattern. Copy the other intervals into the compressed text as uncompressed. The length of the whole text after the new compression scheme will be the sum of the length of the compressed text $(\frac{pn}{1+\frac{1}{H}})$ and the uncompressed text $((1-p)n)$. Now the compression ratio of the new scheme is

$$\rho' = \frac{n}{\frac{pn}{1+\frac{1}{H}} + (1-p)n} > 1/H = \rho_{opt}$$

Now we have a compression scheme that achieves a compression ratio greater than the optimal compression ratio, which is a contradiction. □

Corollary 1 *An arbitrary interval of length $\log m$ in the text gets a match in the pattern with probability $p \leq (1 - H^2)^{H/(1+H)}$.*

Proof: Assume p is the probability of an arbitrary interval of length $\log m$ in the text getting an match in the pattern. From Lemma 1, we have $p^{(1+1/H)} \leq 1 - H^2$. Thus the corollary follows. □

Our sequential pattern matching algorithm works in the following way. Initially, the pattern is aligned with the left end of the text. We repeatedly try to match the text against the pattern. When a mismatch is found or the pattern is fully matched against the text (in this case we get one match of the pattern in the text), the pattern is shifted to the right. We try to maximize the length of each shift therefore skipping unnecessary matches. Note that in the worst case, the probability of getting a match for the pattern in the text can be small though the compressibility is high. For example, a text composed by alternatively repeating a single word of length m (length of the pattern) and a random sequence of length m. The compression ratio is close to 2 while the probability of matching that single word is $1/2m$.

We expect to shift the whole pattern to the right to skip unnecessary matches when a mismatch occurs. In case of a match, we go on matching without shifting the pattern until there is a mismatch, then we shift the pattern to a certain distance to the right and restart the matching process from the leftmost position of the pattern. The actual complexity of the algorithm therefore depends on the probability of matching which is related to the compressibility of the text. We need to preprocess the pattern so that we can look up the next location in the pattern to restart matching. One way of preprocessing the pattern is to construct a shift table similar to KMP algorithm so that we know where to restart matching process when a mismatch is found. However, it is somehow difficult to analyze the expected shift distance using this shift table.

We adopt another way of preprocessing the pattern which produces a shift table that stores all occurrences of pieces of the pattern with length $\log m$. The pieces are indexed according to alphabetical order. This size of the shift table therefore is $O(m^{\log c})$ where $c \geq 2$ is the size of the alphabet because there are $m - \log m$ different pieces in the pattern and $m^{\log c}$ indices in the shift table. In case of a mismatch, we find the next occurrence in the pattern of the current mismatched piece of the text and continue our matching process. If we assume that the pattern is generated from a random distribution, which is the case in practice, we can show in the following lemma that the complexity of the algorithm depends only on the entropy of the text.

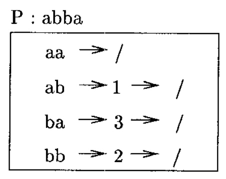

Figure. 1: Example of a shift table.

Lemma 2 *Assuming that the pattern of length m is generated from a random distribution and the alphabet size is c, the expected shift distance after each mismatch* *using the second shift table is $O(m)$ for $c \geq 2$.*

Proof: The total number of different pieces of length $\log m$ is $m^{\log c}$. Therefore the expected number of one single piece of length $\log m$ appearing in the pattern is $max(1, m/m^{\log c})$. The expected distance between two consecutive identical pieces is $min(m, m^{\log c})$. For $c \geq 2$, this is $O(m)$. According to the algorithm, this is also the shift distance when mismatch occurs. □

Theorem 1 *The pattern matching algorithm takes $O(n \log m/(pm))$ expected running time where H is the entropy of the text and $p = 1 - (1 - H^2)^{H/(1+H)}$.*

Proof: Each comparison of the text string takes $O(\log m)$ time because of the length of Δ. Because we shift the pattern to the right only when a mismatch occurs, the total number of necessary comparisons is $O(n/(pm))$. Thus the total sequential time follows. □

Futhermore, we can assume that the text and pattern are generated from the same source with certain distribution of entropy H. We can bound the expected shift distance after each mismatch with respect to the entropy H. We have the following theorem.

Theorem 2 *Assuming that the text and the pattern are generated from the same source with certain distribution of entropy H. The pattern matching algorithm takes $O(n \log m)$ expected running time where H is the entropy of the text.*

Proof: The analysis is the same as in last theorem. However, this time the expected length of shift distance is different since the pattern is assumed to be entropy bounded by H. Applying Lemma 1 to the pattern, the expected number of occurrence of any string of length $\log m$ is $O(pm)$. Therefore the expected distance between any two consecutive identical string of length $\log m$ is $\Omega(1/p)$. The probability p of matching a string of length $\log m$ in the pattern to the text remains the same as last theorem. □

2.2 Matching on Pattern with Bounded Variance

The algorithm described above have an expected running time which is related to the entropy of the text. Here we consider the case where the variance of the pattern is bounded so that the worst case running time could also be bounded.

We define the random variable X as follows: given an arbitrary string of length $O(\log m)$ in the pattern, X is the distance between this string and the next identical string in the pattern (in case there is no such identical string, the distance is $\log m$). The variance we are considering is $Var[X]$. Assume the mean of the distribution X is μ and the length of each matched prefix x_i. Note we have the following property:

$$Var[X] = \frac{\sum_{i=1}^{k}(x_i - \mu)^2}{k} \leq \frac{(\sum_{i=1}^{k}(x_i - \mu))^2}{k} \tag{1}$$

Thus,

$$\sqrt{kVar[X]} \geq \sum_{i=1}^{k}(x_i - \mu) \geq -\sqrt{kVar[X]} \tag{2}$$

and

$$k\mu + \sqrt{kVar[X]} \geq \sum_{i=1}^{k} x_i \geq k\mu - \sqrt{kVar[X]}. \tag{3}$$

Since the complexity of our algorithm depends on the shift distance after each mismatch, the above bound on the total shift distance gives the worst case bound of our algorithm. Applying the above to the time complexity analysis of our algorithm, we have the following:

Theorem 3 *Assuming that the variance of the pattern (as defined above) is V and pattern is of uniform distribution, the running time T of the one dimensional matching algorithm is bounded by the following:*

$$\frac{n \log m}{p(m + \sqrt{V})} \leq T \leq \frac{n \log m}{p(m - \sqrt{V})}.$$

Proof: First we loose the bound in (3) to

$$k(\mu + \sqrt{Var[X]}) \geq \sum_{i=1}^{k} x_i \geq k(\mu - \sqrt{Var[X]}).$$

Then the total number k of shifts can be bounded by

$$\frac{n}{\mu + \sqrt{V}} \leq k \leq \frac{n}{\mu - \sqrt{V}}.$$

Note n denotes the length of the text and $\mu = m$ by Lemma 2. Then we apply the same analysis of Theorem 1 to get the lower and upper time bounds. □.

3 Two Dimensional Matching

We assume that the dimensional text is generated from an alphabet of size c. Without loss of generality, we assume the size of the pattern is n where $n = s^2$.

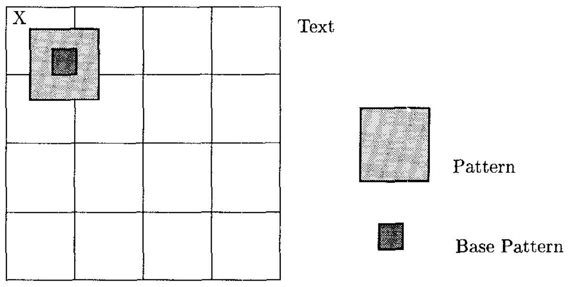

Figure 2: Two-dimensional pattern matching.

In two-dimensional case, we need to match the pattern in two different directions: horizontal and vertical. The search path is a tree-like path starting with the root of the left-upper corner of the text. Each node d in this tree is represented by a point in the text with coordinates (x, y). Each child of d is the next possible matching with coordinates (d_x, d_y) where $d_x \geq x$ and $d_y \geq y$. The matching is accomplished in a fashion similar to a breadth-first search in this tree.

3.1 Preprocessing of Pattern

Given a pattern P of size $m = t^2$ generated from an alphabet of size c, we produce a data structure to speed up the matching process. For every base pattern of size k where $k = (1 + 1/H)\log n$, we construct a set of pointers pointing to different positions the base pattern occurring in the square. We assume that the base pattern is of size k where $k < m$. For example, the base pattern in Fig. 3 have 3 different occurrences in the pattern.

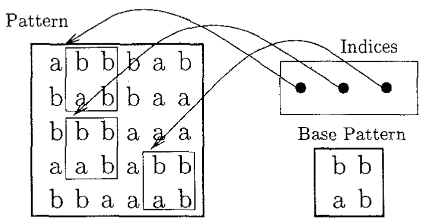

Figure 3: Preprocessing of pattern.

The indices are all possible occurrences of the base pattern. The pointers for each index are the upper-left positions of the base pattern occurring in m. We need to go through every position in the whole pattern to recognize all occurrences of the base patterns. Each comparison takes $O(m)$ time. Therefore, the preprocessing time is $O(mk)$.

3.2 Description of the Algorithm

First we divide the text into blocks each of size m. Then we proceed the search separately in each block. We illustrate the process of search in the most upper-left block X in the text (see Figure 2).

We first find the index of the base pattern in the lower-right corner of that block. We then match the pattern against the entire block. If there is a match, we are done. If not, we slide the pattern to the next possible matching position where the base pattern in the lower-right corner overlaps with the block. This position is just the next pointer for the base pattern in the preprocessed data structure. Then we check if there is a match. This process repeats until there is no further possible position to match.

We will show the complexity of this algorithm is also related to the compressibility of the text. We also define entropy of the 2-dimensional text as the inverse of the optimal compression ratio of the text, i.e. $H = 1/\rho_{opt}$.

Lemma 3 *Assuming the text has a bounded entropy H, the probability of an arbitrary base pattern of size $k = (1 + 1/H)\log n$ appearing in an arbitrary position of the text is $\leq 1 - H^2$.*
Proof: The proof is similar to the one-dimensional case. We assume the optimal compression ratio is $\rho_{opt} = 1/H$. Suppose not, we show there is a compression scheme that achieves higher compression ratio than the optimal compression which is a contradiction. □

Theorem 4 *Assuming the pattern is generated from a random source and the entropy of the text is bounded by H, the two-dimensional pattern matching algorithm takes an expect time of $O(nk(1 - H^2)/c^k)$ where H is the entropy of the text and $k = (1 + 1/H)\log n$.*
Proof: The number of expected matching in each block is $O((1 - H^2)m/c^k)$. The number of total blocks is $O(n/m)$. We assume that each match checking between a particular part of the text and the base pattern takes an expected time of $O(k)$. The time complexity thus follows. □

3.3 Parallelization of 2-D Pattern Matching

The sequential 2-D pattern matching described above can be extended to a parallel algorithm in a straightforward fashion. Note that the pattern matching in each block of the 2-D text is carried out separately. Therefore, we assign n/m processors to work on the matching for each block simultaneously. The total work is the same as of the sequential algorithm.

Theorem 5 *The parallel two-dimensional pattern matching algorithm takes $O(mk(1-H^2)/c^k)$ expected parallel time using $O(n/m)$ processors where H is the entropy of the text, n and m are the size of the text and the pattern respectively, $k = (1+1/H)\log n$.*
Proof: Since the processor works independently on each block, the expected parallel time follows directly from the analysis of the sequential algorithm. The total work is the same as the sequential time. The number of the processors needed is the same as the number of the blocks $O(n/m)$. □

4 Lossy Pattern Matching

Pattern matching algorithm can be lossy in the sense that there could be less than or equal to k mismatches in the matched text where k is a preset constant. Precisely, suppose pattern is $x_1x_2...x_m$ and the matched string from the text is $y_1y_2...y_m$, the number of i that satisfies $x_i \neq y_i$ where $1 \leq i \leq m$ must be less than or equal to k. The sequential algorithm proceeds basically the same way but we now allow matching process to continue until at least k mismatches have appeared. Then we shift pattern as much as possible according to the shift table.

Also there have been various randomized pattern matching algorithms that find the pattern in the text with certain probability. [14] gave three randomized algorithms based on fingerprinting techniques. The match is now between fingerprints which are much shorter than the orginal string. Such algorithms find exact match of the pattern in the text with high likelihood using less storage space. The family of fingerprinting function is carefully chosen so that it is true with high probability that two strings with the same fingerprint are identical. We could apply the similar method in our algorithms, i.e., encode the pieces of string of length $\log m$ using fingerprint functions and then compare the fingerprints instead of original string.

5 Conclusion

We have presented several algorithms and their analysis for fast matching text with respect to the entropy of the text in one and two dimensional cases. We found that the complexity of the algorithms is related to the compressibility of the pattern as well as the text. An interesting open problem is to find an algorithm that has a complexity independent of the pattern entropy without assuming random distribution. Also the case where the complexity of the algorithm is independent on the size of alphabet remains open. The implementation of those algorithms are also being carried out at the same time on both SPARC workstation and CM-5 massively parallel computer.

References

[1] A.Amir and G.Benson. Efficient two dimensional compressed matching. *Proc. of Data Compression Conference, Snow Bird, Utah*, pages 279-288, Mar 1992.

[2] A.Amir and G.Benson and M.Farach. Witness-free dueling: The perfect crime! (or how to match in a compressed two-dimensional array). submitted for publication, 1993.

[3] A.Amir and G.Benson and M. Farach. Let sleeping files lie: pattern matching in Z-compressed files. *Symposium on Discrete Algorithms*, 1994.

[4] A.V. Aho and M.J. Corasick. Efficient string matching: and aid to bibliographic search. *Communications of the ACM*, Vol 18, 6:333-340, 1975.

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

[6] S. Chen and J. H. Reif. Using difficulty of prediction to decrease computation: Fast sort, priority queue and computational geometry for bounded-entropy inputs. *34th Symposium on Foundations of Computer Science*, 104–112, 1993.

[7] J.G. Cleary and I.H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Tran. on Communications*, 32:396–402, 1984.

[8] R. Cole and R. Hariharan. Tighter bounds on the exact complexity of string matching. *33th Symposium on Foundations of Computer Science*, 600–609, 1992.

[9] Z. Galil. A constant-time optimal parallel string-matching algorithm. *24th Symposium on Theory of Computation*, 69-76. 1992.

[10] Z. Galil and Kunsoo Park. An improved algorithm for approximate string matching. *SIAM J. Comput.*, Vol 19, 6:989-999, December 1990.

[11] R.G. Gallager. Variations on a theme by Huffman. *IEEE Trans. on Information Theory*, 24:668–674, 1978.

[12] M.C. Harrison. Implementation of the substring test by hashing. *Communications of the ACM*, 14:777-779, 1971.

[13] JáJá, J. *An introduction to parallel algorithm.* Addison Wesley, 1992.

[14] R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. IBM J. Res. Develop. Vol 31, 2:249-260, March 1987.

[15] D.E.Knuth and J.H.Morris and V.R.Pratt. Fast pattern matching in strings. *SIAM J. Comput*, 8:323-350, 1977.

[16] T. Linder, G. Lugosi, and K. Zeger. Universality and rates of convergence in lossy source coding. *preliminary draft*, 1992.

[17] S. Muthukrishnan and K. Palem. Highly efficient dictionary matching in parallel. *extended abstract*, 7:51–75, 1992.

[18] T. Raita and J. Teuhola. Predictive text compression by hashing. New Orleans, LA.

[19] M. Rodeh, V.R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *J. of ACM*, 28:1:16–24, 1981.

[20] F. Rubin. Experiments in text file compression. *Communication of the ACM*, 19:11:617–623, 1976.

[21] J.S.Vitter and P.Krishnan. Optimal prefetching via data compression, *Thirty-second Annual IEEE Symposium on Foundations of Computer Science*, 1991.

[22] A.C.C. Yao. The complexity of pattern matching for a random string. *SIAM J.Comput*, 8:3:368–387, 1979.