1992

Pattern Matching with Mismatches: A Simple Randomized Algorithm and its Analysis

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Philippe Jacquet

Wojciech Szpankowski
Purdue University, spa@cs.purdue.edu

Report Number:
92-043

PATTERN MATCHING WITH MISMATCHES: A SIMPLE RANDOMIZED ALGORITHM AND ITS ANALYSIS

Mikhail J. Atallah
Philippe Jacquet
Wojciech Szpankowski

CSD-TR-92-043
July 1992

PATTERN MATCHING WITH MISMATCHES: A SIMPLE RANDOMIZED ALGORITHM AND ITS ANALYSIS*

July 21, 1992

Mikhail J. Atallah†
Dept. of Computer Science
Purdue University
W. Lafayette, IN 47907
U.S.A.

Philippe Jacquet‡
INRIA
Rocquencourt
78153 Le Chesnay Cedex
France

Wojciech Szpankowski§
Dept. of Computer Science
Purdue University
W. Lafayette, IN 47907
U.S.A.

Abstract

The study and comparison of strings of symbols from a (possibly very large) alphabet is relevant to various areas of computer science. In particular, the problem of finding all positions, in a text string of length n, at which a pattern string of length m "almost occurs" is of great practical importance. Here by "almost occurs" we mean that some fixed percentage of the characters of the pattern (for example, 90% of them) are equal to their corresponding characters in the text. In this paper we give an algorithm that (i) has $O(n \log m)$ time complexity, and (ii) computes with high probability all of the almost-occurrences of the pattern in the text *irrespective of the probabilistic characteristics of the pattern and text.* We use a probabilistic framework to design some parameters of the algorithm.

*A preliminary version of this research was presented at the 1992 Combinatorial Pattern Matching Conference, Tucson, Arizona. The contents of this paper are solely the responsibility of the authors and do not necessarily represent the official views of the funding agencies.

†This researcher was supported by the Office of Naval Research under Contracts N00014-84-K-0502 and N00014-86-K-0689, the Air Force Office of Scientific Research under Contract AFOSR-90-0107, and the National Library of Medicine under Grant R01-LM05118.

‡This research was primary supported by NATO Collaborative Grant 0057/89.

§This author's research was supported by AFOSR Grant 90-0107 and NATO Collaborative Grant 0057/89, and, in part by the NSF Grant CCR-8900305, and by Grant R01 LM05118 from the National Library of Medicine.

1. INTRODUCTION

Pattern matching is one of the most fundamental problems in computer science. The version of this problem we consider here is the following one. Consider two strings, the text string $\mathbf{a} = a_1 a_2 ... a_n$, and the pattern string $\mathbf{b} = b_1 b_2 ... b_m$, such that symbols a_i and b_j belong to a V-ary alphabet $\Sigma = \{1, ..., V\}$. Let $C_{\mathbf{a},\mathbf{b}}(i)$ be the number of positions at which the string $a_i a_{i+1} ... a_{i+m-1}$ agrees with the pattern $\mathbf{b}$, where an index j that is out of range is understood to stand for $1 + (j \bmod n)$. That is, $C_{\mathbf{a},\mathbf{b}}(i) = \sum_{j=1}^{m} equal(a_{i+j-1}, b_j)$ where $equal(x, y)$ is one if $x = y$, zero otherwise. We are interested in computing all i at which $C_{\mathbf{a},\mathbf{b}}(i) \geq \rho m$, for some constant ρ $(0 < \rho \leq m)$. The constant ρ is supplied to the algorithm as part of its input. Although the algorithm we give works for any input constant ρ, we propose what a suitable choice for ρ should be in order to correspond to the intuitive notion of almost-occurrence. Many related problems have been considered in the literature, notably approximate pattern matching (cf. [12], [14]) and pattern matching with mismatches (cf. [18], [19]). However, in our formulation of the problem the number of mismatches is $O(m)$ while in most existing algorithms (cf. [18], [19], [20]) the number of mismatches is $O(1)$ with the exception of [12] which treats the case $O(m/\log m)$.

Our main result is the design and analysis of an $O(n \log m)$ time algorithm that finds, with high probability, all the values i such that $C_{\mathbf{a},\mathbf{b}}(i) \geq \rho m$. The claimed performance does not depend on the probabilistic characteristics of the input. As far as the probabilistic analysis is concerned, we distinguish two models. We call the *deterministic model* the one where the input strings are given without any assumptions as to their probabilistic characteristics. This model has an important advantage since the detailed probabilistic characteristics of the alphabet and of the input strings are often not known in practice. We also consider the case when the algorithm works on a family of strings, with known probabilistic characteristics; in such a case we use the usual *Bernoulli model* [22] to analyze the algorithm and to design its parameters. We also exhibit experimental data showing that the algorithm performs very well in practice.

Note: Our usage of the words "deterministic model" should not be confused with the concept of a deterministic algorithm; a deterministic algorithm always gives the right answer, whereas our algorithm gives the right answer only with a high probability, in both the Bernoulli and the deterministic models of analysis. It is the probabilistic analysis of our randomized algorithm that makes the distinction between the deterministic *model of analysis* and the Bernoulli one.

Our algorithm takes $O(n \log m)$ time and its probability of error decays in an exponential

manner. Achieving $O(n \log m)$ time performance is trivial in the case of an alphabet of small size (by using convolution), but for possibly large alphabets the best known deterministic algorithms run in $O(n\sqrt{m}\text{polylog}(m))$ time [1, 17]. Thus our randomized algorithm and the deterministic algorithms in [1, 17] are distinct points in a *speed / quality of answer* tradeoff. Large alphabets arise in many practical situations, for example, when the text represents a time series of physical measurements, prices, ...etc. A number of powerful techniques were developed (see [18], [19], [20], [21], [14], [12]) for the more general problem where insertions and deletions are also allowed (i.e., not only mismatches), but since the number of mismatches under our definition of "almost occurrence" is proportional to m, these techniques cannot be invoked here since all but that of [12] would result in a quadratic time bound (these methods were geared towards the situation where the number of mismatches k is $O(1)$, since they contain an $O(km)$ term in their time complexity). The elegant probabilistic method of [12] requires that the input strings be random, the alphabet size $O(1)$, and that $k < m/\log m + O(1)$, and hence cannot be used in our framework (recall that we make no assumptions about the size of the alphabet, and that we can have k proportional to m).

The paper is organized as follows. In the next section we present the algorithm. Then, we provide a probabilistic analysis of it in two different frameworks: The deterministic model, and the Bernoulli model. We also prove that in both models our randomized algorithm provides the right answer with very high probability. Finally, in Section 4 we present experimental data which supports our analysis. The data will demonstrate that our algorithm performs very well in practice. Section 5 concludes.

2. THE ALGORITHM

The algorithm is conceptually very simple, easy to implement, and performs very well in practice (more on this later). Its main idea is to replace the problem with another problem whose alphabet size is $O(1)$, solve the latter in $O(n \log m)$ time, and from its answer deduce the answer to the original problem. The following is an outline of the algorithm.

1. Generate a random permutation π of the alphabet $\Sigma = \{1, ..., V\}$.

2. Go through the pattern and the text, replacing every symbol $j \in \Sigma$ encountered by the symbol $\lceil (\pi(j) \cdot L)/V \rceil$ where L is an integer *constant* (how to choose L is discussed in the next section). Note that this second step produces a modified pattern $\mathbf{p}$ and text $\mathbf{q}$, both over the alphabet $\Sigma' = \{1, \ldots, L\}$ of constant size L (i.e., $L = O(1)$ whereas the original alphabet size V could be arbitrarily large). We call the above scheme of assigning new symbols from Σ' to $\mathbf{a}$ and $\mathbf{b}$ the *permutation model.* Other

assignment schemes also exist. For example, in the *urn model* every symbol $j \in \Sigma$ randomly selects an integer from the set $\Sigma' = \{1, \ldots, L\}$ which becomes the new alphabet. Both ways of obtaining $\mathbf{p}$ and $\mathbf{q}$ (permutation and urn) will be analyzed in Section 3.

3. Compute the $\widetilde{\mathbf{C}}_{\mathbf{p},\mathbf{q}}$ array. Recall that $\widetilde{\mathbf{C}}_{\mathbf{p},\mathbf{q}}(i) = \sum_{\ell=1}^{m} equal(p_{i+\ell-1}, q_\ell)$ where by definition $equal(x, y)$ is one if $x = y$, zero otherwise. It is well known [15] that computing such an array when the alphabet is of size $L = O(1)$ can be done within $O(Ln \log m) = O(n \log m)$ time and linear space; this was in fact one of the crucial ingredients in the elegant scheme of Abrahamson [1] and (independently) Kosaraju [17].

4. We go through the $\widetilde{\mathbf{C}}_{\mathbf{p},\mathbf{q}}$ array and we "mark" every i for which

$$\widehat{C}_{\mathbf{a},\mathbf{b}}(i) = \frac{\widetilde{C}_{\mathbf{p},\mathbf{q}}(i) - mQ}{1 - Q} \geq \rho m \ , \tag{1}$$

where Q is the probability of an "illegal match", that is, a match between symbols in the new alphabet Σ' that does not have a corresponding "legal" match in the original alphabet Σ. This probability is computed in the next section for different probabilistic models. We output every marked position i satisfying (1) as one where the pattern $\mathbf{b}$ almost-occurs in the text $\mathbf{a}$. (The analysis will show that the left-hand side of the above inequality is a good estimate of the percentage of agreement between the pattern and the ith m-substring of the text.)

The probabilistic analysis of the above scheme is given in the next section, where it is also explained how to choose L. In addition, the next section also contains a discussion of which choices of ρ are in agreement with the intuitive notion of almost-occurrence (although ρ is just an input parameter for the above algorithm). The analysis will show that the algorithm finds, with very high probability, all the positions in the text at which the pattern almost-occurs.

3. PROBABILISTIC ANALYSIS

Before discussing ρ and L, we present a probabilistic analysis of the algorithm. The analysis uses elementary statistics to design an unbiased estimate of the number of matches at each position of the pattern in the text (i.e., of each $C_{\mathbf{a},\mathbf{b}}(i)$). Other probabilistic analyses of pattern matching problems can be also found in [7], [6], [8] and [9].

For simplicity of the presentation, we write C and $\widetilde{C}$ for $C_{\mathbf{a},\mathbf{b}}(i)$ and $\widetilde{C}_{\mathbf{p},\mathbf{q}}(i)$, respectively. Note that the value $\widetilde{C}$ is related to the true value C of the number of matches by the following stochastic equation

$$\widetilde{C} = C + \sum_{i=1}^{m-C} X_i \,, \tag{2}$$

where $X_i = 1$ when there is an illegal match, and zero otherwise. By an illegal match we mean a match that occurs between symbols of Σ' that did not occur between symbols of the original alphabet Σ. Note that $EX_i = Q$ where Q is the probability of the illegal match.

An analysis of (2) depends on the probabilistic model of both strings, the text string $\mathbf{a}$ and the pattern string $\mathbf{b}$. In particular, the probability of illegal match Q depends on the underlying model. We consider the two usual analytical models, namely: the deterministic model (i.e., both strings are given) and the Bernoulli model (i.e., both strings are random).

Deterministic Model. In this model both strings are given and known to us (they are deterministic). Note, however, that even in this case the equation (2) is a stochastic one due to the term involving random variables X_i. The distribution of X_i depends further on the way we distribute symbols of the alphabet Σ of size V to the new alphabet Σ' of size L. In Section 2 we concentrated on the so called *permutation model* in which a random permutation $\pi(\cdot)$ of Σ is divided into L blocks that form the new alphabet, that is, the index of the ith block becomes the ith symbol in Σ'. In another model, called the *urn model*, every symbol from the alphabet Σ is randomly assigned to one out of L urns. Symbols that fall into the ith urn are labelled as the ith letter of the new alphabet Σ'. It is easy to estimate the probability Q in both models (cf. [13]). Indeed, in the permutation model $Q = \frac{V}{L(V-1)} - \frac{1}{V-1} \approx \frac{1}{L}$, while in the urn model $Q = 1/L$.

Bernoulli Model. We now assume that both strings are random, and *the ith symbol from Σ occurs in any position of either string, independently of any other position of either string, and with probability p_i*. If $p_i = 1/V$ for all $i \in \Sigma$, then the Bernoulli model is called *symmetric*; otherwise we refer to an *asymmetric* Bernoulli model.

The evaluation of the probability of an illegal match is more intricate in the Bernoulli model. We consider in details only the permutation model since the urn model can be treated in a similar manner. To describe our results, we need the following definition of an *eligible permutation*. Let $\pi(\cdot)$ be a permutation of $\Sigma = \{1, 2, \ldots, V\}$. Let also for simplicity V be a multiply of L, and define $K = V/L$. We consider a random permutation $\pi(1), \pi(2), \ldots, \pi(V)$ of Σ and divide it into L blocks of size K. A permutation $\widetilde{\pi}(\cdot)$ is eligible if and only if permutation of elements inside a block and permutation of whole blocks lead to the same eligible permutation. Clearly, the number of distinct eligible permutations is

equal to $V!/(L!(K!)^L)$ (cf. [13]).

Now we can compute the probability of the illegal match in the Bernoulli model. Noting that blocks in the eligible permutation contains symbols of the original alphabet that become a single symbol in the new alphabet, we arrive at the following formula

$$Q = \frac{L!(K!)^L}{V!} \sum_{\widetilde{\pi}} \sum_{k=1}^{K} p_{\widetilde{\pi}(kK+1)} \cdots p_{\widetilde{\pi}((k+1)K)}, \tag{3}$$

where the summation in the above is over all eligible premutations (e.g., $p_{\widetilde{\pi}(kK+1)}$ represents the occurrence probability of the $\widetilde{\pi}(kK+1)$ symbol in the $kK+1$ block). This formula simplifies significantly for the symmetric alphabet. In this case

$$Q = \frac{1}{L} - \frac{1}{V} \,. \tag{4}$$

Now we ready to continue our analysis of our basic equation (2). For uniform treatment of both deterministic and Bernoulli models, we assume that $C = c$. Then, $E\widetilde{C} = c + (m - c)Q$. This leads to the following unbiased estimate $\widehat{C}$ of $\widetilde{C}$

$$\widehat{C} = \frac{\widetilde{C} - mQ}{1 - Q} \,. \tag{5}$$

Formula (5) is in the agreement with (1) from Section 2. Note that $E\widehat{C} = c$ and $\mathrm{var}\widehat{C} = (m - c)Q/(1 - Q)$, hence $\widehat{C}$ is an unbiased estimate of C.

How to choose ρ and L? We begin with the issue of ρ.

Although the algorithm works for any input constant ρ, ultimately it is the user who must supply the algorithm with some ρ, and hence we need to discuss what a suitable choice for ρ might be. Clearly, the main goal of pattern matching with mismatches is to identify a correlation (i.e., homology or similarity) between the pattern and the text. One can argue that in order for two strings to be correlated, and hence homologous, the similarity between them should be much higher than a correlation between two *randomly selected* strings; certainly, it makes sense that the "almost-occurrence" should be clearly separated from the matching produced by two random and statistically independent strings **a** and **b**. Now, let $M_{m,n} = \max_{1 \le i \le n-m}\{C_{\mathbf{a},\mathbf{b}}(i)\}$ in the Bernoulli model with the text string and the pattern string being statistically independent. In [8] and [9] we proved that $M_{m,n} \sim mP + \Theta(\sqrt{m \log n})$ in probability (pr.) for $\log n = o(m)$, where $P = \sum_{i=1}^{V} p_i^2$ is the probability of a match in a given position of random strings. For $\log n = \Omega(m)$, Arratia *et al.* [7] proved that $M_{m,n} = \Theta(\log n)$. To attach any significance to the pattern matching of *similar* (i.e., homologous) strings (other than pure chance), it makes sense to consider that

a "significant" correlation occurs if $\rho m \gg M_{m,n} \sim mP + \Theta(\sqrt{m \log n})$. This condition is typically satisfied even for moderate choices for ρ because P is typically small (for example, in the symmetric case we have $P = 1/V$).

The choice of L controls the quality of the algorithm. We can choose it in such a way that with a high level of confidence, say $(1-\alpha)100\% = 90\%$, the real value c of the match is in a small interval called the *confident interval.* This leads to the standard interval estimate (cf. [10]). Since the sum in (2) is composed of i.i.d. random variables (in both considered models), we can apply the *Central Limit Theorem* provided $\rho < 1$. Then, for large m and for given confidence level $1-\alpha$ we obtain the following confidence interval

$$\frac{\widehat{C}}{m} - \frac{z_{\alpha/2}\sigma\sqrt{1-\rho}}{\sqrt{m}} \leq \frac{c}{m} \leq \frac{\widehat{C}}{m} + \frac{z_{\alpha/2}\sigma\sqrt{1-\rho}}{\sqrt{m}}\,, \tag{6}$$

where $z_{\alpha/2}$ is such that $\Pr\{Z > z_{\alpha/2}\} = \alpha/2$ and Z has standard normal distribution, that is, $Z \sim N(0,1)$ (cf. [10]). The parameter L is hidden in $\widehat{C}$ through the probability Q.

Finally, we estimate the error produced by our randomized algorithm. Note that the error occurs if and only if the actual number of matching C satisfies $C < \rho m$, and the algorithm returns $\widehat{C} > \rho m$. Define P_{er} as $P_{er} = \Pr\{\widehat{C} > \rho m \mid C = (\rho-\varepsilon)m\}$, where $\varepsilon > 0$. Using (2) we note that

$$P_{er} = \Pr\left\{ \frac{1}{m(1-\rho+\varepsilon)} \sum_{i=1}^{m(1-\rho+\varepsilon)} X_i > \frac{\varepsilon}{1-\rho+\varepsilon} + \frac{Q(1-\rho)}{1-\rho+\varepsilon} \right\}.$$

Then, by the normal approximation of the binomial distribution (cf. [13], [11]), or by Chernoff's (Hoeffding's) bound (cf. [11]) we obtain for $m \to \infty$

$$P_{er} \leq A \exp\left(\frac{-\varepsilon^2 m}{2(1-\rho+\varepsilon)Q} \right), \tag{7}$$

where A is a constant. In other words, $P_{er} = O(e^{-\varepsilon^2 m})$. The algorithm, as expected, with high probability provides the right answer.

4. EXPERIMENTAL DATA

We have compiled extensive experimental data on the quality of the answer returned by the algorithm: the data displayed in the Tables 1 and 2 is typical of what we obtained. In those tables, the "%Exact" entry is the exact value of the fraction of agreement between the pattern and a typical position of the text at which it almost occurs, whereas the "%Estim." entry is the estimate of that agreement as given by our algorithm. Note from Table 1 that the algorithm performs extremely well in practice, even for moderate values of ρ and L;

Table 1: Simulation results for uniform alphabet with $n = 2000$, $m = 400$ and $V = 100$.

L	$\rho = 0.6$		$\rho = 0.7$		$\rho = 0.8$		$\rho = 0.9$	
	%Exact	%Estim.	%Exact	%Estim.	%Exact	%Estim.	%Exact	%Estim.
5	69.75	70.25	75.25	75.75	83.75	84.00	90.50	90.00
10	69.75	69.25	75.25	75.50	83.75	83.50	90.50	90.50
20	69.75	69.50	75.25	75.25	83.75	83.25	90.50	90.50
30	69.75	70.00	75.25	75.00	83.75	83.25	90.50	90.00
40	69.75	70.00	75.25	75.25	83.50	83.50	90.50	90.50
50	69.75	69.75	75.25	75.25	83.75	83.50	90.50	90.25

Table 2: Simulation results for uniform alphabet with $n = 2000$, $m = 800$, $\rho = 0.8$ and $L = 10$.

V	%Exact	%Estim.
100	83.375	83.375
300	83.125	83.375
500	83.000	83.375
700	83.000	83.375
800	83.000	83.375

that it performs well for high values of these parameters is in agreement with intuition, but it is somewhat surprising that it does so good for moderate values of these parameters. In fact, even for fairly small values of ρ and L, the algorithm still finds the locations of the text where the pattern "almost occurs", although in such cases the estimate of the percentage of agreement is not as good. If for some reason the user needs both a small ρ and a small L, and a very accurate count of the percentage of agreement, then he could view the output of the algorithm as an indication of *where* to apply a more accurate counting procedure (which could even be a brute force one, if there are few enough such "interesting" positions in the text). Note also from Table 2 how good is our algorithm for large alphabets. We expect that the algorithm will work spectacularly well in real applications.

5. FURTEHR REMARKS

Although the algorithm we gave is for 1-dimensional patterns and texts, it can be used for comparisons of multi-dimensional patterns and texts using the techniques of Amir *et al.* [4, 5]. The one-dimensional probabilistic analyses given in this paper and in [8] can also be extended to higher dimensions; although the extensions to higher dimensions are not trivial, they are natural and involve no substantially new ideas, and hence we choose to omit them.

References

[1] K. Abrahamson, Generalized String Matching, *SIAM J. Comput.*, 16, 1039-1051, 1987.

[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[3] Aldous, D., *Probability Approximations via the Poisson Clumping Heuristic*, Springer Verlag, New York 1989.

[4] A. Amir and M. Farach, Efficient 2-dimensional approximate matching of non-rectangular figures, *Proc. 2d ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, California, 1991, 212-223.

[5] A. Amir and G. Landau, Fast Parallel and Serial Multidimensional Approximate Array Matching, *Proc. Intl. Workshop on Sequences, Combinatorics, Compression, Security and Transmission*, Salerno, Italy, June 1988.

[6] Arratia, R., Gordon, L., and Waterman, M., An Extreme Value Theory for Sequence Matching, *Annals of Statistics*, 14, 971-993, 1986.

[7] Arratia, R., Gordon, L., and Waterman, M., The Erdös-Rényi Law in Distribution, for Coin Tossing and Sequence Matching, *Annals of Statistics*, 18, 539-570, 1990.

[8] Atallah, M., Jacquet, P., and Szpankowski, W., A Probabilistic Approach to Pattern Matching with Mismatches, Purdue University, CSD-TR-1007, 1990.

[9] Atallah, M., Jacquet, P., and Szpankowski, W., Pattern Matching with Mismatches: A Probabilistic Analysis and a Randomized Algorithm, *Proc. of the 1992 Combinatorial Pattern Matching Conference*, Tucson, Arizona, 1992, 27-41.

[10] P. Bickel and K. Doksum, *Mathematical Statistics: Basic Ideas and Selected Topics*, Holden-Day, Inc., San Francisco 1977.

[11] E. Coffman, and G. Lueker, *Probabilistic Analysis of Packing and Partitioning Algorithms*, John Wiley & Sons, New York 1991.

[12] Chang, W.I. and Lawler, E.L., Approximate String Matching in Sublinear Expected Time, *Proc. 31st Ann. IEEE Symp. on Foundations of Comp. Sci.*, 116-124, 1990.

[13] Feller, W., *An Introduction to Probability Theory and its Applications,* Vol. II, John Wiley & Sons, New York (1971).

[14] Galil, Z. and Park, K., An Improved Algorithm for Approximate String Matching, *SIAM J. Comp.*, 19, 989-999, 1990.

[15] M. J. Fischer and M. S. Paterson, String-Matching and Other Products. *Complexity of Computation (SIAM-AMS Proceedings 7)*, R. M. Karp ed., American Mathematical Society, Providence, RI, 1974, 113-125.

[16] Knuth, D.E., J. Morris and V. Pratt, Fast Pattern Matching in Strings, *SIAM J. Computing*, 6, 323-350, 1977.

[17] S. R. Kosaraju, Efficient String Matching, manuscript, 1987.

[18] Landau, G.M. and Vishkin, U., Efficient String Matching with k Mismatches, *Theor. Comp. Sci.*, 43, 239-249, 1986.

[19] Landau, G.M. and Vishkin, U., Fast String Matching with k Differences, *J. Comp. Sys. Sci.*, 37, 63-78, 1988.

[20] Landau, G.M. and Vishkin, U., Fast Parallel and Serial Approximate String Matching, *J. Algorithms*, 10, 157-169, 1989.

[21] E.W. Myers, An O(ND) Difference Algorithm and Its Variations, *Algorithmica*, 1, 252-266, 1986.

[22] Szpankowski, W., On the Height of Digital Trees and Related Problems, *Algorithmica*, 6, 256-277, 1991.