Randomized Swap Matching in $O(m \log m \log |\Sigma|)$ time *

Richard Cole[†] Ramesh Hariharan[‡]

September 27, 1999

Abstract

We give a randomized algorithm for the *Pattern Matching with Swaps* problem which runs in $O(m \log m \log |\Sigma|)$ time on a text of length $2m-1$ and a pattern of length m drawn from an alphabet set of size $|\Sigma|$. This algorithm gives the correct answer with probability at least $1-\frac{1}{m}$ and does not miss a match. The best deterministic algorithm known for this problem takes $O(m^{4/3}\text{polylog}(m))$ time.

1 Introduction

The *Pattern Matching with Swaps* problem (the *Swap Matching* problem, for short) requires finding all occurrences of a pattern of length m in a text of length $2m-1$. The pattern is said to match the text at a given location l, if adjacent text characters can be swapped, if necessary, so as to make the substring of the text starting at location l identical to the pattern. All the swaps are constrained to be disjoint, i.e., each character is involved in at most one swap.

Amir et al. [?] obtained the first non-trivial results on this problem. They showed that the case when the size of the alphabet set Σ exceeds 2 can be reduced to the case when it is exactly 2 with a time overhead of $O(\log^2 |\Sigma|)$. They then showed how to solve the problem for alphabet sets of size 2 in time $O(m^{4/3} \log m)$, which is the best bound known to date. Amir et al. [?] also give certain special cases for which $O(m\text{polylog}(m))$ time can be obtained. However, these cases seem rather restrictive.

Recently, Amir et al. [?] have improved the reduction from large alphabet sets to alphabet sets of size 2. This reduction now requires a time overhead of $O(\log |\Sigma|)$ only. The crucial step in this reduction is to obtain $|\Sigma|$ codes of length $O(\log |\Sigma|)$ each so that the following property holds for any ordered triple of codes: the first two codes in the triple are identical and different from the third code on at least one bit. Amir et al. [?] use the ϵ-biased k-wise independent sample space of Naor and Naor [?] to obtain these codes in $O(|\Sigma|)$ time. They use the following parameter values: $k=3$ and $\epsilon < 1/8$.

Our contribution is two-fold.

First, we give a randomized algorithm for the case of alphabet sets of size 2. This algorithm produces the right answer with probability at least $1-\frac{1}{m}$ in time $O(m \log m)$ and never misses a match.

*This work was supported in part by NSF grants CCR-9503309 and CCR-9800085.

[†]Courant Institute, New York University, cole@cs.nyu.edu.

[‡]Indian Institute of Science, Bangalore, ramesh@csa.iisc.ernet.in. Work done in part while visiting NYU.

Second, we give an alternate direct construction of the above codes of length $O(\log |\Sigma|)$. Our construction does not need ϵ-biased k-wise independent sample spaces and runs in $O(|\Sigma|)$ time.

Together, these give an $O(m \log m \log |\Sigma|)$ time algorithm for arbitrary alphabet sets. However, we do not have a procedure to verify the correctness of the output in $O(m \text{ polylog}(m))$ time. Therefore, for longer texts (i.e., when the text has length $n >> m$), the time taken will be $O(n \log n)$ (and not $O(n \log m)$) and correctness guaranteed with probability at least $1 - \frac{1}{n}$.

This paper is organized as follows. Section **??** gives a key property distilled from Amir et al. [**?**] which gives necessary and sufficient conditions for the pattern to match at a text location. The randomized algorithm is described in Sections **??** and **??**. Section **??** gives our construction of codes for large alphabet sets.

2 The Key Property

We assume that the text and the pattern both have only as and bs. Partition the text and pattern into maximal *streams*, where a stream is an alternating substring, i.e., no two consecutive characters are identical. By maximality, the character to the left of the leftmost character x in a stream, if any, is identical to x, and similarly, the character to the right of the rightmost character y, if any, is identical to y. The following lemma appears in a different form in [**?**] and describes all matches of the pattern in the text.

Lemma 1 *The pattern matches in a particular alignment if and only if one of the following holds for each stream A in the text and each stream B in the pattern overlapping A.*

1. *The overlap between A and B has even length.*
2. *The overlap between A and B has odd length, and the portion of B overlapping A matches A exactly, i.e., without any swaps.*
3. *The overlap between A and B has odd length, and the portion of B overlapping A does not match A. However, either B is the leftmost stream in the pattern and A extends further to the left of B, or B is the rightmost stream in the pattern and A extends further to the right of B.*

Proof. First, we show the *if* part. If the above overlap is even then the overlapping portions of A and B can be made identical by swapping, if necessary. Note that these swaps stay within the overlapping portion so other streams are not affected. If the overlap is odd and the overlapping portions match exactly then nothing needs to be done. And if the overlap is odd, the overlapping portions do not match exactly, B is the leftmost (rightmost, respectively) strip in the pattern, and A extends further to the left (right, respectively) of B, then swapping characters in A starting from the character to the left (right, respectively) of the overlapping portion and proceeding to the right (left, respectively), ensures that the overlapping portions of A and B can be made identical without affecting the other streams. So clearly, if one of the above conditions holds for each overlapping pair of streams then the pattern matches.

Consider the *only if* part now. Suppose none of the above conditions hold for A and B. Then their overlap has odd length and, without loss of generality, the overlap has $u = (ab)^*a$ in the text and $v = (ba)^*b$ in the pattern. Then, for a match to occur, either the leftmost a

in u must be swapped with the character to the left or the rightmost a in u must be swapped with the character to the right. Consider the former case (the latter is similar).

Clearly, the portion of A overlapping B cannot be a prefix of A for the above swap to be effective (the character preceding A is an a). It follows that the portion of B overlapping A must be a prefix of B. If B is not the leftmost stream in the pattern, then again the above swap is ineffective (the character preceding B is a b). Thus B must be the leftmost stream in the pattern and A must extend further to the left of B. But this contradicts the assumption that none of the above conditions hold for A, B. $\square$

3 Algorithm Outline

We show how to find all matches of the pattern beginning at odd locations in the text. Matches beginning at even locations are found similarly.

We will set up two pairs of strings with the property that matches beginning at odd locations can be determined by convolving each pair of strings separately. The two pairs are called *odd-even* and *even-odd*, respectively.

A stream in the text is called *odd* if the as in it occur at odd text locations (if there are no as, then the only b in it occurs in an even text location). Otherwise, this stream is called *even*. An analogous definition is made for pattern streams.

The Odd-Even Pair. One string t_o in this pair is obtained from the text, the other p_e from the pattern.

Consider the text first. Put a 1 at each location which is part of an odd stream and a 0 at each location which is part of an even stream. Next, consider the pattern. In each even stream, put an alternating sequence of 1 and -1; puts 0s at locations in odd streams. The following lemma is obvious.

Lemma 2 *Consider any placement of the pattern in the text and consider any two overlapping streams A in the text and B in the pattern. Let j denote the dot product of the portions of t_o and p_e which correspond to the overlap of the streams A and B. Then j has the following property.*

1. *If either A is an even stream or B is an odd stream then $j = 0$.*
2. *If the overlap has even length then $j = 0$.*
3. *If the overlap has odd length, A is an odd stream, and B is an even stream, then $j = \pm 1$.*

The Even-Odd Pair. Consider the text first. Put a 1 at each location which is part of an even stream and a 0 at each location which is part of an odd stream. This gives the string t_e. Next, consider the pattern. In each odd stream, put an alternating sequence of 1 and -1; puts 0s at locations in even streams. This gives p_o. Lemma **??** is the even-odd analogue of Lemma **??**.

Lemma 3 *Consider any placement of the pattern in the text and consider any two overlapping streams A in the text and B in the pattern. Let j denote the dot product of the portions of t_e and p_o which correspond to the overlap of the streams A and B. Then j has the following property.*

1. *If either A is an odd stream or B is an even stream then $j = 0$.*
2. *If the overlap has even length then $j = 0$.*
3. *If the overlap has odd length, A is an even stream, and B is an odd stream, then $j = \pm 1$.*

Corollary 1 *Consider any placement of the pattern starting at an odd location in the text. Consider any two overlapping streams A in the text and B in the pattern and consider their overlapping portions. The dot product of the corresponding overlapping portions of either t_e, p_o or t_o, p_e is ± 1 if and only if the above overlapping portions violate both conditions 1 and 2 of Lemma* **??**.

Proof. First, consider the *if* part. Since the two conditions are violated, the overlap has odd length and the portions of A and B in this overlapping region are not identical. Recall that we are considering a placement of the pattern beginning at an odd location in the text. Thus A and B cannot be both odd streams or both even streams, otherwise, their overlapping portions would be identical. So one of A, B is odd and the other is even. The corollary follows from Lemmas **??** and **??**.

Next, consider the *only if* part. Without loss of generality, suppose that the dot product of the overlapping portions of A, B in t_o, p_e is ± 1. Then A must be an odd stream and B an even stream. Further, this overlap must have odd length, otherwise the dot product will be 0, by Lemma **??**. Finally, since we are considering only matches of the pattern starting at odd text locations, the overlapping portions of A and B are not identical. Thus conditions 1 and 2 are violated. $\square$

Corollary **??** and Lemma **??** immediately lead to Lemma **??**, which is the crux of our algorithm. Let L, R denote the leftmost and rightmost streams in the pattern, respectively. For any particular placement of the pattern, let L', R' denote the text streams overlapping the left and the right ends of the pattern, respectively.

Lemma 4 *Consider any placement of the pattern starting at an odd location in the text. If the pattern matches in this placement, then the dot product of the overlapping portions of A, B for each stream A in the text and B in the pattern is 0 (with the possible exception of L', L and R', R) in both t_o, p_e and t_e, p_o. If the pattern does not match in this placement then one of the following happens:*

1. *The overlapping portions of L', L are not identical, have odd lengths, and L' does not extend further to the left of L.*
2. *The overlapping portions of R', R are not identical, have odd lengths, and R' does not extend further to the right of R.*
3. *For some pair of overlapping streams A, B other than L, L' and R, R', the dot product of the overlapping portions of A, B is non-zero in at least one of t_o, p_e and t_e, p_o.*

4 The Randomized Algorithm

First, we do some computation for L and R.

Border Computation. The following can be determined easily in linear time.

1. For each l, when the pattern is placed started at the lth text character, whether the stream L' in the text overlapping the left end of the pattern has an odd or even length overlap with L, whether L' extends further to the left of L, and the dot product of the overlapping portions of L' and L in both t_o, p_e and in t_e, p_o. We call these dot products $left-oe(l)$ and $left-eo(l)$, respectively.

2. For each l, when the pattern is placed started at the lth text character, whether the stream R' in the text overlapping the right end of the pattern has an odd or even length overlap with R, whether R' extends further to the right of R, and the dot product of the overlapping portions of R' and R in both t_o, p_e and in t_e, p_o. We call these dot products $right-oe(l)$ and $right-eo(l)$, respectively.

Random Multipliers. We modify t_o, t_e, p_o, p_e slightly. For each stream in the text, two random numbers in the range $1 \ldots m^2$ are assigned; all entries in the portion of t_e corresponding to this stream are multiplied by the first number and all entries in the portion of t_o corresponding to this stream are multiplied by the second number. A similar multiplication with random numbers is done in p_e and p_o. Let $rand_e(A)$ and $rand_o(A)$ denote the random number multiplier for stream A in the even and the odd versions, respectively.

Next, for each placement of the pattern starting at an odd location l in the text, we determine $dp-oe(l)$, the dot product of p_e and the overlapping portion of t_o, and similarly, $dp-eo(l)$, the dot product of p_o and the overlapping portion of t_e. This can be done in $O(m \log m)$ time, using convolution [**?**]. We now have the following claim.

Lemma 5 *Consider a particular placement of the pattern starting at an odd location l in the text. If the pattern matches in this location, then the following hold.*

1. $dp-eo(l) = rand_e(L') * rand_o(L) * left-eo(l) + rand_e(R') * rand_o(R) * right-eo(l)$.

2. $dp-oe(l) = rand_o(L') * rand_e(L) * left-oe(l) + rand_o(R') * rand_e(R) * right-oe(l)$.

3. *The overlapping portions of L, L' are either identical, or have even lengths, or L' extends further to the left of L.*

4. *The overlapping portions of R, R' are either identical, or have even lengths, or R' extends further to the left of R.*

If the pattern does not match in this location, then with probability at least $1 - \frac{1}{m^2}$, at least one of following holds.

1. $dp-eo(l) \neq rand_e(L') * rand_o(L) * left-eo(l) + rand_e(R') * rand_o(R) * right-eo(l)$.

2. $dp-oe(l) \neq rand_o(L') * rand_e(L) * left-oe(l) + rand_o(R') * rand_e(R) * right-oe(l)$.

3. *The overlapping portions of L, L' are not identical, have odd lengths, and L' does not extend further to the left of L.*

4. *The overlapping portions of R, R' are not identical, have odd lengths, and R' extends further to the right of R.*

Proof. If the pattern matches at location l then the lemma follows directly from Lemma **??** and Lemma **??**. If the pattern does not match starting at location l, then by Lemma **??**, either conditions 3 and 4 stated above for this case hold, or there exists some pair of streams A, B (other than L, L' and R, R'), the dot product of whose overlapping portions is non-zero in either t_e, p_o or in t_o, p_e. Without loss of generality, suppose this dot product is non-zero in t_e, p_o. Then $dp - eo(l) = rand_e(L') * rand_o(L) * left - eo(l) + rand_e(R') * rand_o(R) * right - eo(l)$ with probability at most $\frac{1}{m^2}$ (if the random multipliers for all streams other than B are fixed then there is at most one value for the random multiplier for stream B which can cause this event to happen). $\square$

Since there are only m values for l in the above lemma, the above algorithm produces the correct result with probability at least $1 - \frac{1}{m}$; in addition, it never misses a match.

Finally, if the text has length $n >> m$, then the random multipliers have to be chosen from $1 \ldots n^2$ and not $1 \ldots m^2$, so that the probability of correctness is at least $1 - \frac{1}{n}$. The time taken in this case is $O(n \log n)$.

5 Coding for Large Alphabet Sets

We show how to construct $|\Sigma|$ codes of length $O(\log |\Sigma|)$ each, with the following property. Our construction will take $O(|\Sigma|)$ time.

Property: For any ordered triple of codes, there must be a bit in which the first two codes agree but differ from the third code.

It is easy to see that the expected number of ordered triples violating the above property is less than 1 for a random assignment of codes (each code bit is set to 1 with probability 1/2), provided the code length is $\Theta(\log |\Sigma|)$. Our linear time algorithm constructs these codes in two steps.

Step 1. $|\Sigma|^{\frac{1}{8}}$ codes with the above property are constructed in this step. Each code has length $O(\log |\Sigma|)$. This is done using the method of Conditional Probabilities [**?**, **?**], which is easily seen to run in $O((|\Sigma|^{1/8})^4 \log^2 |\Sigma|) = o(|\Sigma|)$ time (the number of ordered triples is cubic in $|\Sigma|^{\frac{1}{8}}$, each triple needs to be checked each time a bit is set, the total number of bits over all codes is $O(|\Sigma|^{\frac{1}{8}} \log |\Sigma|)$, checking a triple requires $O(\log |\Sigma|)$ time).

Step 2. Shortly, we will show how to construct x^2 codes with the above property, given x codes with the above property, for any $x > 0$. The lengths of the new codes will be thrice those of the old codes. The time taken will be $O(x^2)$. Step 2 involves 3 iterations of the above construction, starting with the codes obtained in Step 1 (here, $x = |\Sigma|^{\frac{1}{8}}$). Clearly, the lengths of the final codes will be $O(\log |\Sigma|)$ as well. The total time taken will be $O(|\Sigma|)$.

Constructing x^2 codes from x codes. Let the given codes be denoted by $a_1, a_2, \ldots, a_x$. Note that no two of these codes are identical, otherwise, the required property will not be satisfied. The new codes are given by the following $x \times x$ matrix. Each new code has three components. The first component is determined by the row, the second by the column, and the third by the SW-NE diagonal (with wrap-around).

$$\begin{array}{llllll}
a_1a_1a_1 & a_1a_2a_2 & a_1a_3a_3 & \cdots & a_1a_{x-1}a_{x-1} & a_1a_xa_x \\
a_2a_1a_2 & a_2a_2a_3 & a_2a_3a_4 & \cdots & a_2a_{x-1}a_x & a_2a_xa_1 \\
a_3a_1a_3 & a_3a_2a_4 & a_3a_3a_5 & \cdots & a_3a_{x-1}a_1 & a_3a_xa_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a_{x-1}a_1a_{x-1} & a_{x-1}a_2a_x & a_{x-1}a_3a_1 & \cdots & a_{x-1}a_{x-1}a_{x-3} & a_{x-1}a_xa_{x-2} \\
a_xa_1a_x & a_xa_2a_1 & a_xa_3a_2 & \cdots & a_xa_{x-1}a_{x-2} & a_xa_xa_{x-1}
\end{array}$$

That the x codes obtained above satisfy the required property is shown as follows. Recall that this property involves ordered triples of codes. Consider any one ordered triple. It suffices to prove that one of the following situations holds for at least one of the three components forming each code.

1. No two entries of the ordered triple are identical in this component. This is sufficient because $a_1 \dots a_x$ themselves have the required property.

2. The first two entries of the ordered triple are identical in this component while the third entry is different. This is sufficient because $a_1 \dots a_x$ are all distinct.

There are 3 cases in proving that one of the above situations holds.

Case 1. Suppose all three entries in the ordered triple are either in distinct rows or in distinct columns. Then (1) above holds for either the first component or the second component.

Case 2. Suppose the first two entries in the ordered triple are in one row (column, respectively) and the third entry is in another row (column, respectively). Then (2) above holds for the first component (second component, respectively).

Case 3. The only remaining case is when the first two entries in the ordered triple are neither in the same row, nor in the same column. Further, the third entry is in the same row as one of the above entries (without loss of generality, the first entry) and in the same column as the second entry. Note that the first and third entries will now disagree in the third component, as will the second and the third entries. Then either [1] or [2] will hold for the third component.

6 Open Problems

Whether the above stream multipliers can be chosen in a deterministic manner in $O(m\text{polylog}(m))$ time is not clear. We leave this as an open problem.

References

[1] A. Amir, Y. Aumann, G. Landau, M. Lewenstein, N. Lewenstein. *Pattern Matching with Swaps*, Proceedings of IEEE Symposium on Foundations of Computer Science, pp. 144–153, 1997.

[2] A. Amir, G. Landau, M. Lewenstein, N. Lewenstein. *Efficient Special cases of Pattern Matching with Swaps*, Information Processing Letters, Vol. 68, No. 3, pp. 125–132, 1998.

[3] A. Amir, Private Communication.

[4] M. Fischer, M. Paterson. *String Matching and Other Products*, R.M. Karp (editor), SIAM-AMS Proceedings, 7, pp. 113-125, 1974.

[5] J. Naor, M. Naor. *Small Bias Probability Spaces: Efficient Constructions and Applications*, SIAM J. Computing, 22, pp. 838–856, 1993.

[6] P. Raghavan. *Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs*, Journal of Computer and System Sciences 37, pp. 130–143, 1988.

[7] P. Raghavan, *Randomized Approximation Algorithms in Combinatorial Optimization*, Foundations of Theoretical Computer Science and Software Technology, LNCS 880, pp. 300–317, 1994.

[8] J. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.