On Approximate Pattern Matching with Thresholds

Peng Zhang^a, Mikhail J. Atallah^b

^a*School of Computer Science, Georgia Tech*

^b*Department of Computer Science, Purdue University*

Abstract

In the traditional version of the problem of approximate pattern matching, a pattern symbol is considered to match a text symbol if the two symbols are equal. Such a notion of exact equality is not suitable for situations where the text and pattern symbols are imprecise, e.g., obtained from an analog source, distorted by additive noise, etc. In such situations it is more appropriate to consider two alphabet symbols to match *even if they are not equal*, as long as they do not differ by more than a given threshold θ . The goal is then to compute the number of matches of the length- M pattern with all length- M substrings of the length- N text, i.e., to compute a vector of $N - M + 1$ scores, where the i th score is the number of matches between the pattern and the substring that begins at text position i . The main result of this paper is to show that this threshold version of the problem can be solved by recursively solving $3 + 2 \log \theta$ instances of the traditional (i.e., zero-threshold) version of the problem, which is much-studied in the literature and for which there are many efficient (typically randomized) solutions of time complexity close to $O(N \log M)$. This paper's result therefore implies the first randomized $O(N \log M \log \theta)$ solution for the threshold version of the problem. It also implies that any future improvement to the traditional (zero-threshold) version of the problem automatically translates into a simi-

Email addresses: pzhang60@gatech.edu (Peng Zhang), mja@cs.purdue.edu (Mikhail J. Atallah)

Portions of this work were supported by National Science Foundation Grants CPS-1329979, Science and Technology Center CCF-0939370; by an NPRP grant from the Qatar National Research Fund; and by sponsors of the Center for Education and Research in Information Assurance and Security. The statements made herein are solely the responsibility of the authors.

The work was done while the first author was studying at Purdue University.

Preprint submitted to Information Processing Letters

January 8, 2017

lar improvement to the arbitrary-threshold case. Further, we show that the factor $\Omega(\log \theta)$ is tight by recursive algorithms. [\[Added a sentence for lower bound.\]](#)

Keywords: pattern matching, threshold, recursion

1. Introduction

Recall that, in the approximate pattern matching problem, the input is a pair of strings: a text string $T = t_1 \cdots t_N$, and a pattern string $P = p_1 \cdots p_M$, both over an alphabet Σ . The desired output is a similarity *score vector* S of length $N - M + 1$ such that

$$S[i] = \sum_{k=1}^M \delta(t_{i+k-1}, p_k)$$

for $1 \leq i \leq N - M + 1$, where $\delta(x, y)$ is 1 if $x = y$ and is 0 if $x \neq y$. This problem has been widely studied in theory, and has a variety of applications in DNA sequence analysis, web search, image processing, and computer vision, etc. [\[Added application / background.\]](#)

The best known deterministic algorithm for this problem takes $O(N\sqrt{M \log M})$ time [1, 17]. Efficient randomized algorithms of time complexity close to $O(N \log M)$ were given in [3, 5]. Besides, [16] gave algorithms which compute each score with an ϵ relative error in time $O((N/\epsilon^2) \log^c M)$ for a small constant c . The entries of S for which the similarity score is at least $M - k$ are known as k -mismatches. Finding only k -mismatches can be done efficiently with an $O(N\sqrt{k \log k})$ time algorithm [2], which was further improved to $O(N + N\sqrt{k/\log N \log k})$ [13]. [\[Added a citation here.\]](#) Although in the worst case k could be proportional to M , in practice one is often only interested in smaller values of k . There had been much prior work on this k -mismatch problem (e.g., [14, 6], to mention a few). A recent refinement to the shift-add approach of [6] was given in [15].

In [4] a generalization of the pattern matching problem was studied, considering that the text pattern symbols might be imprecise if obtained from an analog source or distorted by additive noise, etc. [\[Added motivation of considering approximate pattern matching with thresholds.\]](#) They computed the score vector S for the case where the input also includes a *threshold* value

θ whose significance is that alphabet symbols whose difference does not exceed θ are considered to match. In other words, in the above definition of the score vector S , the δ function is replaced by a δ_θ function where $\delta_\theta(x, y)$ is 1 if $|x - y| \leq \theta$ and is 0 if $|x - y| > \theta$. For example, if the alphabet is $\Sigma = \{0, 1, 2, \dots, 9\}$, $\theta = 2$, $T = 6482$, and $P = 93$, then $S = (1, 0, 2)$. The algorithm given in [4] has an $O(N\sqrt{M \log M})$ time complexity. Unlike the situation for the classical (zero-threshold) version of the problem, randomized more efficient (e.g., close to $O(N \log M)$) time bounds for this problem were not known. Such an efficient solution is a corollary to the main result of the present paper.

Another related but different variant of the pattern matching problem is δ -matching, studied broadly in [7, 8, 9, 10, 12]. Instead of outputting a score vector of length $N - M + 1$, δ -matching problem outputs a subset of indices $\mathcal{I}_\delta = \{i : \max_{k=1}^M |t_{i+k-1} - p_k| \leq \delta\}$ where δ is a given threshold. In other words, $\mathcal{I}_\delta = \{i : S[i] = M\}$ by setting $\theta = \delta$ in our problem. Efficient algorithms for δ -matching problem are given in [7, 8, 9, 10]. However, these algorithms are not applicable to the problem studied in this paper, since they lack the similarity scores for indices with mismatches.

2. This Paper's Contribution

The main result of our present paper is to establish that any algorithm of complexity $g(N, M)$ for the traditional (i.e., zero-threshold) version of the problem implies an algorithm of complexity $O(g(N, M) \log \theta)$ for the arbitrary-threshold version of the problem. We establish this by giving an algorithm that solves the with-threshold version of the problem using $3 + 2 \log \theta$ zero-threshold problem instances that have same size (i.e., N and M) as the with-threshold instance (the time spent on creating each instance is linear, hence dominated by $g(N, M)$).

It is easy to see that trying to process, one at a time, the bits of the binary representation of θ , does not work and leads to a dead-end. Our solution bears no resemblance to such an approach. A rough overview of what we do is as follows. We use a recursive approach where, after solving two judiciously defined zero-threshold instances of the problem, we recurse on a problem whose threshold is $\theta/2$. The recursion is on new versions of text and pattern, obtained from the input versions by replacing symbols by other symbols, in a way that resembles an odd-even un-shuffle of a bucketized alphabet (hence some symbols of the alphabet get replaced by much larger

ones, whereas other symbols get replaced by much smaller ones). The process we use for replacing a symbol i depends on the *parity* of $\lfloor \frac{i}{\theta/2} \rfloor$, and the arithmetic used in both the even and odd case involves i , θ , and (for the odd case) $\max_j(\lfloor \frac{j}{\theta/2} \rfloor)$ where the maximization is over all alphabet symbols j that occur in text or pattern. Further, we show that the factor $\Omega(\log \theta)$ is tight by recursive algorithms.

3. The Algorithm

[Formulate the definition of the problem in the following.]

Before stating our algorithm, we formally define the problem of approximate pattern matching with thresholds as follows.

Inputs: A text string $T = t_1 \cdots t_N$, a pattern string $P = p_1 \cdots p_M$, both over a (possibly large) alphabet. A threshold value θ .

Output: A vector S of length $N - M + 1$, where $S[i] = \sum_{k=1}^M \delta_\theta(t_{i+k-1}, p_k)$ and $\delta_\theta(x, y)$ is 1 if $|x - y| \leq \theta$, 0 otherwise.

This section gives a recursive algorithm that solves the problem of approximate pattern matching with thresholds by making use of $3 + 2 \log \theta$ instances of zero-threshold versions of the problem. As the zero-threshold version is much studied in the literature, and has efficient (typically randomized) algorithms for solving it, our result implies efficient algorithms for the threshold version of the problem.

In what follows we assume alphabet symbols and the threshold to be *integers*. There is no loss of generality in this, because otherwise they can all be “scaled up” to become integers without changing the resulting score vector. For example, if the original alphabet is $\{1.73, 2.55, 5.41, 2.17\}$ and θ is 0.13, then after scaling they become $\{173, 255, 541, 217\}$ and (respectively) 13.

To avoid unnecessarily cluttering the exposition, we present the algorithm when θ is a power of 2; there is no loss of generality in doing so, as the algorithm we give can easily be modified for the general case.

Algorithm Scores(T, P, θ)

Method: The algorithm is recursive, and works by making two calls to zero-threshold instances of the problem, followed by a recursive call on a problem instance having a threshold of $\theta/2$.

1. If $\theta = 1$ then do the following.
 - (a) Let S_0 be the score vector returned by $\mathbf{Scores}(T, P, 0)$.
 - (b) Obtain T' from T by replacing every occurrence of a symbol i in T with $i + 1$. Let S_1 be the score vector returned by $\mathbf{Scores}(T', P, 0)$.
 - (c) Obtain P' from P by replacing every occurrence of a symbol i in P with $i + 1$. Let S_2 be the score vector returned by $\mathbf{Scores}(T, P', 0)$.
 - (d) Return $S_0 + S_1 + S_2$.
 2. If $\theta > 1$ then do the following, where every alphabet symbol i is said to belong to *bucket* j where $j = \lfloor \frac{i}{\theta/2} \rfloor$. (It does not matter that some buckets are empty, because we do not create a list of buckets – we merely use the concept of a bucket number for symbols that actually occur in P or T .)
 - (a) Compute the contributions, to the desired score vector, of the alignments between symbols that are in *adjacent buckets* (i.e., one is in bucket j and the other in bucket $j + 1$, hence they “match” because they are within θ of each other). This is done by using two instances of a zero-threshold algorithm, as follows.
 - i. Obtain T' from T by replacing every occurrence of a symbol i in T with $\lfloor \frac{i+\theta/2}{\theta/2} \rfloor$ (which is $1 + i$'s bucket number). Obtain P' from P by replacing every occurrence of a symbol i in P with its own bucket number $\lfloor \frac{i}{\theta/2} \rfloor$. Let S_1 be the score vector returned by $\mathbf{Scores}(T', P', 0)$.
Note: S_1 captures the contribution of all matches of a symbol x in P with a symbol y in T such that the bucket number for x exceeds the bucket number for y by 1.
 - ii. Obtain T'' from T by replacing every occurrence of a symbol i in T with its own bucket number $\lfloor \frac{i}{\theta/2} \rfloor$. Obtain P'' from P by replacing every occurrence of a symbol i in P with $\lfloor \frac{i+\theta/2}{\theta/2} \rfloor$ (which is $1 + i$'s bucket number). Let S_2 be the score vector returned by $\mathbf{Scores}(T'', P'', 0)$.
Note: S_2 captures the contribution of all matches of a symbol x in P with a symbol y in T such that the bucket number for y exceeds the bucket number for x by 1.
- The contribution, to the desired score vector, of the alignments between symbols that are in adjacent buckets, is $S_1 + S_2$.

Note: Of course the desired score vector contains contributions of matches that are not accounted for by $S_1 + S_2$, namely, matches between pairs of symbols that have same bucket number (always a “match”) or whose bucket numbers differ by 2 (can be either a match or a mismatch). These other contributions are taken care of by the next step.

- (b) Compute the contribution, to the desired score vector, of the alignments between symbols that are either in the same bucket or are in separate non-adjacent buckets. This is done by recursively solving a problem instance of threshold $\theta/2$, $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$, as we explain next. First, observe that an alignments of symbols that are in different non-adjacent buckets can be a match only if the symbols’ respective bucket numbers differ by 2 (because if their bucket numbers differ by more than 2 then they are farther than θ apart and therefore do not match). Therefore such symbols that are in odd-numbered buckets cannot match symbols in non-adjacent even-numbered buckets. This observation is the key insight for the construction of $\hat{T}$ and $\hat{P}$, which are obtained from T and (respectively) P by replacing the symbols in T and P with other symbols. Which symbol replaces a symbol i depends on whether the bucket number of i (call it j) is even or odd. The replacement is done as follows (where B is the largest bucket number of any symbol occurring in P or T , assumed to be even for convenience).
 - i. If i ’s bucket number j is even (i.e., $j = 2t$) then i is replaced with $i - t\theta/2$. Intuitively, this is like “pulling out” all the odd-numbered buckets and then “compressing to the left” the even-numbered buckets. The symbols in bucket 0 (if any) do not change. The symbols in bucket 2 are left-shifted by $\theta/2$, those in bucket 4 are left-shifted by θ , those in bucket 6 are left-shifted by $3\theta/2$, etc. Note that this effectively places the symbols previously in even-numbered buckets in buckets numbered 0 to $B/2$.
 - ii. If i ’s bucket number j is odd (i.e., $j = 2t+1$) then i is replaced with $i + ((B/2)+1-t)\theta/2$. Intuitively, this is like “pulling out” all the even-numbered buckets and then “compressing to the right” the odd-numbered buckets so they become separated by a gap of at least $\theta/2$ from those that were left-compressed in the previous step. The symbols in bucket 1 are thereby right-

symbols in (T, P)	0	1	2	3	4	5	6	7	8	9
symbols in $(\hat{T}, \hat{P})$	0	1	8	9	2	3	10	11	4	5

Figure 1: The correspondence relationship between symbols in (T, P) and $(\hat{T}, \hat{P})$.

shifted by $(B/2 + 1)\theta/2$, those in bucket 3 are right-shifted by $(B/2)\theta/2$, those in bucket 5 are right-shifted by $(B/2 - 1)\theta/2$, etc. Note that this effectively places the symbols previously in odd-numbered buckets in buckets numbered $(B/2) + 2$ to $B + 1$.

Let S_3 be the score vector returned by the recursive call $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$.

3. Return $S_1 + S_2 + S_3$.

Observe that, at each level of the recursion, the work done *other than the calls to the zero-threshold instances* (i.e., the time to set up the inputs to the zero-threshold instances, and combine the answers they return) is $O(N + M)$ which is dominated by the time taken by the zero-threshold calls. Therefore the overall time complexity is determined by how many calls are made to the zero-threshold instances, and we henceforth focus on analyzing how many such calls are made.

The recurrence for the number of zero-threshold instances that must be solved is

$$f(\theta) = 2 + f(\theta/2) \text{ if } \theta > 1$$

$$f(1) = 3$$

whose solution is $f(\theta) = 3 + 2 \log \theta$.

An example. We now give an example illustrating the above algorithm. Let $\Sigma = \{0, 1, 2, \dots, 9\}$, $T = (3, 5, 0, 9, 9, 1, 4, 3, 5, 8, 8, 4)$, $P = (7, 8, 9, 5, 3, 6)$ and $\theta = 4$.

In $\text{Scores}(T, P, 4)$, $T' = (2, 3, 1, 5, 5, 1, 3, 2, 3, 5, 5, 3)$, $P' = (3, 4, 4, 2, 1, 3)$, and $S_1 = (0, 3, 0, 1, 1, 0, 2)$; $T'' = (1, 2, 0, 4, 4, 0, 2, 1, 2, 4, 4, 2)$, $P'' = (4, 5, 5, 3, 2, 4)$ and $S_2 = (0, 0, 1, 1, 3, 1, 0)$. Then, we compute $\hat{T}$ and $\hat{P}$. The correspondence relationship between symbols in $(\hat{T}, \hat{P})$ and symbols in (T, P) is given in Figure 1. Thus, we get $\hat{T} = (9, 3, 0, 5, 5, 1, 2, 9, 3, 4, 4, 2)$ and $\hat{P} = (11, 4, 5, 3, 9, 10)$. In $\text{Scores}(\hat{T}, \hat{P}, 2)$, we do a similar computation and return value $S_3 = (3, 2, 4, 3, 0, 2, 2)$. Therefore, the output of $\text{Scores}(T, P, \theta)$ is $S = S_1 + S_2 + S_3 = (3, 5, 5, 5, 4, 3, 4)$.

4. Analysis

In this section, we analyze our algorithm $\text{Scores}(T, P, \theta)$ according to the zero-threshold algorithm that we use as a subroutine. In $\text{Scores}(T, P, \theta)$, if $\theta > 1$, we put each symbol i into bucket $\lfloor \frac{i}{\theta/2} \rfloor$. Then, a pair of symbols i and i' matches if they satisfy one of the following three conditions: (1) the bucket numbers of i and i' are the same; (2) the bucket numbers of i and i' differ by 1; (3) the bucket numbers of i and i' differ by 2 *and* the difference between i and i' is no larger than θ . $S_1 + S_2$ corresponds to condition (2), and S_3 corresponds to conditions (1) and (3).

Lemma 1. *The score vector of $(\hat{T}, \hat{P}, \theta/2)$ counts the matches of pairs of symbols in (T, P, θ) that satisfy condition (1) or (3).*

Proof. Recall that in the process of constructing $(\hat{T}, \hat{P})$ from (T, P) , we replace symbol i with $i - t\theta/2$ if the bucket number of i is even (say, $j = 2t$), and we replace symbol i with $i + ((B/2) + 1 - t)\theta/2$ if the bucket number of i is odd (say, $j = 2t + 1$), where B is the largest bucket number over all symbols occurring in T or P .

We first show that for any two symbols in (T, P) , say i and i' (without loss of generality, assume $i < i'$), that satisfy condition (1) or (3), the difference between their respective symbols in $(\hat{T}, \hat{P})$, say $\hat{i}$ and $\hat{i}'$, is at most $\theta/2$. If i and i' satisfy condition (1), that is, i and i' have same bucket number j . According to the replacement rules, $i - \hat{i}$ and $i' - \hat{i}'$ are both $t\theta/2$ (if $j = 2t$) or $-((B/2) + 1 - t)\theta/2$ (if $j = 2t + 1$). Since $|i - i'| \leq \theta/2$, we know that $|\hat{i} - \hat{i}'| \leq \theta/2$. If i and i' satisfy condition (3), that is, the bucket number of i and i' differ by 2 and $|i - i'| \leq \theta$. We know that their bucket numbers must be both even or both odd. If both i and i' have even bucket numbers (say, i 's bucket number is $j = 2t$ and i' 's bucket number is $j' = 2(t + 1)$), then $\hat{i} = i - t\theta/2$ and $\hat{i}' = i' - (t + 1)\theta/2$. The difference between $\hat{i}$ and $\hat{i}'$ is $i' - i - \theta/2$, which is at most $\theta/2$. If both i and i' have odd bucket numbers (say, i 's bucket number is $j = 2t + 1$ and i' 's bucket number is $j' = 2(t + 1) + 1$), then $\hat{i} = i + ((B/2) + 1 - t)\theta/2$ and $\hat{i}' = i' + ((B/2) + 1 - (t + 1))\theta/2$. The difference between $\hat{i}$ and $\hat{i}'$ is $i' - i - \theta/2$, which is at most $\theta/2$.

We then show that for any two symbols in (T, P) , say i and i' (without loss of generality, assume $i < i'$), that satisfy neither condition (1) nor condition (3), the difference between their respective symbols in $(\hat{T}, \hat{P})$, say $\hat{i}$ and $\hat{i}'$, is larger than $\theta/2$. There are two cases of i and i' : the bucket numbers of i and i' differ by 1; or the difference between i and i' is larger than θ . If

the bucket numbers of i and i' differ by 1, then one bucket number is even and the other one is odd. No matter which one is even, by the replacement rules, $|\hat{i} - \hat{i}'| \geq (B/2)\theta/2 + 1$, which is larger than $\theta/2$. If the difference between i and i' is larger than θ , according to the replacement rules, $|\hat{i} - \hat{i}'| \geq \min\{\theta/2 + 1, i' - i - \theta/2\}$, which is larger than $\theta/2$. It completes the proof. $\square$

Theorem 1. *If the zero-threshold algorithm used by $\text{Scores}(T, P, \theta)$ is correct and deterministic, then $\text{Scores}(T, P, \theta)$ is correct and deterministic. The running time of $\text{Scores}(T, P, \theta)$ is $O(g(N, M) \log \theta)$, where $g(N, M)$ is the running time of the zero-threshold algorithm with input lengths N and M .*

Proof. It is obvious that $\text{Scores}(T, P, \theta)$ is deterministic if the zero-threshold algorithm we use is deterministic, since we do not have any randomness elsewhere. We prove the correctness of $\text{Scores}(T, P, \theta)$ by induction on θ .

If $\theta = 1$, it is trivial to see that $S_0 + S_1 + S_2$ in $\text{Scores}(T, P, \theta)$ captures all contribution of matches, thus it is correct. Assume that $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$ (for θ is equal to some $\theta' \geq 2$) is correct. Consider the three conditions of matches listed at the beginning of this section. It is easy to see that the contribution of matches of pairs of symbols in (T, P, θ) that satisfy condition (2) is exactly captured by $S_1 + S_2$. By Lemma 1, the contribution of matches of pairs of symbols that satisfy condition (1) or (3) is captured by $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$ (i.e., S_3). By the hypothesis of induction, $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$ is correct. Thus, $S_1 + S_2 + S_3$ captures the contribution of all matches correctly. It means that $\text{Scores}(T, P, \theta)$ is correct.

Recall that in our recursion, each time we halve the threshold, we use two zero-threshold instances of lengths N and M to compute S_1 and S_2 . Thus, the number of the zero-threshold instances that we solve in $\text{Scores}(T, P, \theta)$ satisfies the following equations:

$$\begin{aligned} f(\theta) &= 2 + f(\theta/2) \text{ if } \theta > 1 \\ f(1) &= 3 \end{aligned}$$

Hence, $f(\theta) = 3 + 2 \log \theta$. Besides, the time used for constructing the new pattern vector and text vector, as well as adding three returned score vectors together, is $O(N + M)$, dominated by the running time of the zero-threshold algorithm with input lengths N and M . Therefore, the running time of $\text{Scores}(T, P, \theta)$ is $O(g(N, M) \log \theta)$. $\square$

Theorem 2. *If the zero-threshold algorithm used by $\text{Scores}(T, P, \theta)$ is a correct Las Vegas algorithm, then $\text{Scores}(T, P, \theta)$ is a correct Las Vegas algorithm. The expected running time of $\text{Scores}(T, P, \theta)$ is $O(g(N, M) \log \theta)$,*

where $g(N, M)$ is the expected running time of the zero-threshold algorithm with input lengths N and M .

The proof is similar to the proof of Theorem 1, thus we omit it here.

Theorem 3. *If the zero-threshold algorithm used by $\text{Scores}(T, P, \theta)$ is an unbiased Monte Carlo algorithm with variance $O(\sigma^2)$, then $\text{Scores}(T, P, \theta)$ is an unbiased Monte Carlo algorithm with variance $O(\sigma^2 \log \theta)$. The running time of $\text{Scores}(T, P, \theta)$ is $O(g(N, M) \log \theta)$, where $g(N, M)$ is the running time of the zero-threshold algorithm with input lengths N and M .*

Proof. We show the output of $\text{Scores}(T, P, \theta)$ is unbiased by induction on θ . When $\theta = 1$, the output of $\text{Scores}(T, P, 1)$ is unbiased due to the linearity of expectation and the unbiasedness of the zero-threshold algorithm. Assume that the output of $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$ (where θ is equal to some $\theta' \geq 2$) is unbiased. In the proof of Theorem 1, in $\text{Scores}(T, P, \theta)$, the contribution of matches is exactly captured by $S_1 + S_2 + S_3$. S_1 and S_2 are unbiased due to the unbiasedness of zero-threshold algorithm. By the hypothesis of induction, S_3 (i.e., the output of $\text{Scores}(\hat{T}, \hat{P}, \theta/2)$) is unbiased, thus, by the linearity of expectation, $S_1 + S_2 + S_3$ (i.e., the output of $\text{Scores}(T, P, \theta)$) is unbiased.

In the proof of Theorem 1, we know that the number of zero-threshold instances that must be solved in $\text{Scores}(T, P, \theta)$ is $3 + 2 \log \theta$. Since each zero-threshold instance runs independently, the variance of the output of $\text{Scores}(T, P, \theta)$ is the sum of the variance of each zero-threshold instance, that is, $O(\sigma^2 \log \theta)$ (where $O(\sigma^2)$ is the variance of the output of each zero-threshold instance). The running time of $\text{Scores}(T, P, \theta)$ is $O(g(N, M) \log \theta)$ (where $g(N, M)$ is the running time of the zero-threshold instance of same input lengths). $\square$

As a corollary, we give an efficient randomized algorithm by using the randomized algorithm in [5] as the zero-threshold algorithm.

Corollary 1. *There is a randomized algorithm for estimating the score vector S between a length- N text vector and a length- M pattern vector with a threshold θ , in time $O(kN \log M \log \theta)$, where k is a parameter controlling the tradeoff between running time and variance. The output of the randomized algorithm is unbiased and the variance of $S[i]$ is at most $((3 + 2 \log \theta)M - S[i])M/k^2$.*

Proof. We use the randomized algorithm in [5] as the zero-threshold algorithm in $\text{Scores}(T, P, \theta)$. It is a Monte Carlo algorithm with running time $O(kN \log M)$, where k is the number of iterations in the algorithm. The output score vector is unbiased and each entry s has a variance at most $\min\{M/k, M - s\}(M - s)/k$.

By Theorem 3, the output of $\text{Scores}(T, P, \theta)$ is unbiased. The variance is the sum of the variance of each zero-threshold instance solved in our algorithm. Let s_i^j be the exact score at position i in the j th zero-threshold instance. The variance of $S[i]$ is at most $\sum_1^{3+2\log\theta} (M - s_i^j)M/k^2$. Since the contribution of match of each pair of symbols is exactly computed once, $\sum_1^{3+2\log\theta} s_i^j = S[i]$. Thus, the variance of our estimate of $S[i]$ is at most $((3 + 2\log\theta)M - S[i])M/k^2$. $\square$

5. Lower Bound by Recursion

In this section, we show that our result is tight for recursive algorithms. More specifically, in $\text{Algorithm Scores}(T, P, \theta)$, for any bucket size (say, $g(\theta)$) we choose (e.g., in our algorithm in Section 3, $g(\theta) = \theta/2$), it requires to solve $\Omega(\log \theta)$ distinct zero-threshold instances of same input lengths of N and M . We prove it by recursion tree.

bucket #	0	1	2	3	4	5	6
symbol	0,1,2	3,4,5	6,7,8	9,10,11	12,13,14	15,16,17	18,19,20

Figure 2: An example of threshold $\theta = 9$ and bucket size $g(\theta) = 3$. For symbol 10, symbols in buckets numbered by 1,2,3,4,5 match 10 and satisfy the first condition; symbols 1,2,18,19 in buckets 0 or 6 match 10 and satisfy the second condition. In $\hat{T}$ and $\hat{P}$, the new order of buckets is 0,3,6,*,1,4,*,2,5, where “*” represents a dummy bucket.

$\text{Algorithm Scores}(T, P, \theta)$ can be generalized as follows. If $\theta > 1$, we put every alphabet symbol i into bucket $\lfloor \frac{i}{g(\theta)} \rfloor$. Then, a pair of symbols matches iff they satisfy one of the following two conditions: (1) their bucket numbers differ by at most $\theta/g(\theta) - 1$; (2) their bucket numbers differ by $\theta/g(\theta)$ and their difference is at most θ . For the first condition, we replace each symbol with its bucket number, forming new (T', P') and recursively solve $\text{Scores}(T', P', \theta/g(\theta) - 1)$. For the second condition, similar to our algorithm in Section 3, suppose i 's bucket number $j = (\theta/g(\theta))t + r$, we form $\hat{T}$ and $\hat{P}$ from T and P by replacing each i with $i + ((\frac{B}{(\theta/g(\theta))} + 1)r - t)\theta/2$ (where B is the largest bucket number of any symbol occurring in T or

P , assumed to be a multiple of $\theta/g(\theta)$ for convenience). Intuitively, we respectively “pull out” all the buckets whose numbers j have equal value of $(j \bmod \theta/g(\theta))$ and “compress” them together, then arrange them by separating “compressed” groups by a dummy bucket of size $g(\theta)$, forming new $(\hat{T}, \hat{P})$. Figure 2 gives an example of threshold $\theta = 9$ and bucket size $g(\theta) = 3$. Recursively solving $\text{Scores}(\hat{T}, \hat{P}, g(\theta))$ gives the contribution of the matches that satisfy the second condition. We roughly ignore the reduplicate scores of matches of two symbols that have same bucket number in (T, P) , which does not change the result asymptotically. Until the threshold is some constant $c \geq 1$, we stop and solve it directly through solving constant instances of zero-threshold version of the problem. We can represent the whole process as a recursion tree, see Figure 3. Each node represents a call of $\text{Scores}(T, P, \theta)$. On the leaves of the tree, threshold θ is constant c . Thus, the number of the zero-threshold instances that need to solve can be bounded by the number of leaves of this tree, denoted by k .

Let θ_v be the threshold in the recursion call at node v . Fixed a non-leaf node v , let l and r be its left child and right child respectively. We have $\theta_v = (\theta_l + 1)\theta_r \leq 2\theta_l\theta_r$. Thus, we know that $(2c)^{\Theta(k)} \geq \theta$, that is, $k = \Omega(\log \theta)$. That is, the number of zero-threshold instances required for recursive algorithms is $\Omega(\log \theta)$.

6. Conclusion

We established a connection between the traditional version of pattern matching (where symbols match iff they are equal) and the threshold version, in which different symbols match iff they do not differ by more than a given threshold θ . We showed constructively that the time complexity of the threshold version of the problem is within a factor of $\log \theta$ of the traditional version. This implies the first efficient randomized algorithm for the threshold version of the problem, having complexity $O(N \log M \log \theta)$.

Reference

- [1] Karl Abrahamson, “Generalized string matching,” *SIAM Journal of Computing*, 16, 1987, pp. 1039–1051.
- [2] Amihoud Amir, Moshe Lewenstein, Ely Porat, “Faster algorithms for string matching with k mismatches,” *Proceedings of the eleventh annual*

Figure 3: Recursion tree

ACM-SIAM symposium on Discrete algorithms, San Francisco, January 2000, pp. 794–803.

- [3] Mikhail J. Atallah, Frederic Chyzak, Philippe Dumas, “A randomized algorithm for approximate string matching, *Algorithmica* 29, 3 (2001), pp. 468-486.
- [4] Mikhail J. Atallah, Timothy W. Duket, “Pattern Matching in the Hamming Distance with Thresholds,” *Information Processing Letters*, 111 (14), (2011) pp. 674–677.
- [5] Mikhail J. Atallah, Elena Grigorescu, Yi Wu, “A Lower-Variance Randomized Algorithm for Approximate String Matching,” *Information Processing Letters*, 113 (18), 2013, pp. 690-692.
- [6] Ricardo A. Baeza-Yates, Gaston H. Gonnet, “A new approach to text searching,” *Commun. ACM*, 35(10), 1992, pp. 74–82, 1992.
- [7] Domenico Cantone, Salvatore Cristofaro, Simone Faro, “Efficient Algorithms for the delta-Approximate String Matching Problem in Musical Sequences,” *Stringology*, 2004, pp. 33–47.

- [8] Peter Clifford, Raphaél Clifford, Costas Iliopoulos, “Faster algorithms for δ, γ -matching and related problems,” *Combinatorial Pattern Matching*, 2005, pp. 68–78.
- [9] Richard Cole, Costas Iliopoulos, Thierry Lecroq, Wojciech Plandowski, Wojciech Rytter, “On special families of morphisms related to δ -matching and don’t care symbols,” *Information Processing Letters*, 85(5), 2003, pp. 227–233.
- [10] Maxime Crochemore, Costas S. Iliopoulos, Thierry Lecroq, Yoan J. Pinzon, Wojciech Plandowski, Wojciech Rytter, “Occurrence and substring heuristics for δ -matching”, *Fundamenta Informaticae*, 56(1), 2003, pp. 1–21.
- [11] Michael J. Fischer, Michael S. Paterson, “String matching and other products,” in *Complexity of computation*, Richard M. Karp (Ed.), American Mathematical Society, Society for Industrial and Applied Mathematics, 1974.
- [12] Kimmo Frederiksson, Szymon Grabowski, “Efficient Algorithms for (δ, γ, α) and $(\delta, k_\Delta, \alpha)$ -matching,” *International Journal of Foundations of Computer Science*, 19(1), 2008, pp. 163–183.
- [13] Kimmo Fredriksson, Szymon Grabowski, “Exploiting word-level parallelism for fast convolutions and their applications in approximate string matching,” *European Journal of Combinatorics*, 34(1), 2013, pp. 38–51.
- [14] Zvi Galil, Raffaele Giancarlo, “Improved string matching with k mismatches,” *SIGACT News*, 17(4), 1986, pp. 52–54.
- [15] Szymon Grabowski, Kimmo Fredriksson, “Bit-parallel string matching under Hamming distance in $O(n\lceil m/w \rceil)$ worst-case time,” *Information Processing Letters*, 105(5), 2008, pp. 182–187.
- [16] Howard Karloff, “Fast algorithms for approximately counting mismatches,” *Information Processing Letters*, 48(2), 1993, pp. 53–60.
- [17] S.R. Kosaraju, “Efficient string matching,” manuscript, 1987.