A Simple Algorithm for Approximating the Text-To-Pattern Hamming Distance

Tsvi Kopelowitz\(^1\) and Ely Porat\(^2\)

\(^1\) University of Waterloo, Waterloo, Canada
kopelot@gmail.com

\(^2\) University of Bar-Ilan, Ramat Gan, Israel
porately@cs.biu.ac.il

Abstract

The algorithmic task of computing the Hamming distance between a given pattern of length \(m\) and each location in a text of length \(n\), both over a general alphabet \(\Sigma\), is one of the most fundamental algorithmic tasks in string algorithms. The fastest known runtime for exact computation is \(\tilde{O}(n \sqrt{m} \log m)\). We recently introduced a complicated randomized algorithm for obtaining a \(1 \pm \epsilon\) approximation for each location in the text in \(O(\frac{n}{\epsilon} \log \frac{n}{\log m} \log |\Sigma|)\) total time, breaking a barrier that stood for 22 years. In this paper, we introduce an elementary and simple randomized algorithm that takes \(O(n \epsilon \log n \log m)\) time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Pattern Matching, Hamming Distance, Approximation Algorithms

Digital Object Identifier 10.4230/OASIcs.SOSA.2018.10

1 Introduction

One of the most fundamental family of problems in string algorithms is to compute the distance between a given pattern \(P\) of length \(m\) and each location in a given larger text \(T\) of length \(n\), both over alphabet \(\Sigma\), under some string distance metric (See [24, 20, 2, 25, 8, 6, 3, 7, 29, 12, 28, 26, 9, 11, 31, 27, 19, 10, 15, 18, 17, 16, 5, 4, 30]). One of the most useful distance metrics in this setting is the Hamming Distance of two strings, which is the number of aligned character mismatches between the strings. Let \(\text{HAM}(X, Y)\) denote the Hamming distance of two strings \(X\) and \(Y\). Abrahamson [1] showed an algorithm whose runtime is \(O(n \sqrt{m} \log m)\). The task of obtaining a faster upper bound seems to be very challenging, and indeed there is a folklore matching conditional lower bound for combinatorial algorithms based on the hardness of combinatorial Boolean matrix multiplication (see [14]). However, for constant sized alphabets the runtime is solvable in \(O(n \log m)\) using a constant number of convolution computations (which are implemented via the FFT algorithm) [20].

The challenge in beating Abrahamson’s algorithm naturally lead to approximation algorithms for computing the Hamming distance in this setting, which is the problem that we consider here and is defined as follows. Denote \(T_j = T[j, \ldots, j + m - 1]\). In the pattern-to-text approximate Hamming distance problem the input is a parameter \(\epsilon > 0\), \(T\), and \(P\). The goal is to compute for all locations \(i \in [1, n - m + 1]\) a value \(\delta_i\) such that \((1 - \epsilon)\text{HAM}(T_i, P) \leq \delta_i \leq (1 + \epsilon)\text{HAM}(T_i, P)\). For simplicity we assume without loss of generality that \(\Sigma\) is the set of integers \(\{1, 2, \ldots, |\Sigma|\}\).

* This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 683064).
Karloff in [22] utilized the efficiency of the algorithm for constant sized alphabets to introduce a beautiful randomized algorithm for solving the pattern-to-text approximate Hamming distance problem, by utilizing projections of $\Sigma$ to binary alphabets. Karloff’s algorithm runs in $\tilde{O}(\frac{n}{\epsilon^2})$ time, and is correct with high probability.

Communication complexity lower bounds

One of the downsides of Karloff’s algorithm is the dependence on $\frac{1}{\epsilon^2}$. In particular, if one is interested in a one percent approximation guarantee, then this term becomes 10000, which is extremely large for many applications. Remarkably, many believed that beating the runtime of Karloff’s algorithm is not possible, mainly since there exist qualitatively related lower bounds for estimating the Hamming distance of two equal length strings (for a single alignment). In particular, Woodruff [32] and later Jayram, Kumar and Sivakumar [21] showed that obtaining a $(1 \pm \epsilon)$ approximation for two strings in the one-way communication complexity model requires sending $\Omega(1/\epsilon^2)$ bits of information. The lower bounds were extended to the two-way communication complexity model as well [13].

In [23] we showed that this belief was flawed, by introducing an $\tilde{O}(\frac{n}{\epsilon})$ time algorithm that succeeds with high probability. In particular, we proved the following.

► Theorem 1 ([23]). There exists an algorithm that with high probability solves the pattern-to-text approximate Hamming distance problem and runs in $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon} \log n \log m \log |\Sigma|)$ time.

Our algorithm in [23] turned out to be rather complicated and borrows ideas from sparse recovery and constructions of specialized families of hash functions.

A simpler algorithm

In this paper we show how to solve the pattern-to-text approximate Hamming distance problem faster (in terms of logarithmic factors) and simpler. The rest of this paper is devoted to proving the following theorem.

► Theorem 2 ([23]). There exists an algorithm that with high probability solves the pattern-to-text approximate Hamming distance problem and runs in $O(\frac{n}{\epsilon} \log n \log m)$ time.

2 The Algorithm

For a function $h : \Sigma \rightarrow \Sigma'$ and for any string $S = s_1s_2\ldots s_k$, let $h(S) = h(s_1)h(s_2)\ldots h(s_k)$.

Local versus global operations

The operations that our algorithm performs during the approximation of the Hamming distance at some location $j$ are partitioned into two types. The first type is local operations which are independent of the computations performed for other locations in $T$. The second type is global operations, which are operations that for efficiency purposes are computed as a batch for all of the alignments in $T$. In particular, all of the global operations in our algorithm are to compute $HAM(h(T_j), h(P))$ where $h : \Sigma \rightarrow \lfloor \frac{|\Sigma|}{2} \rfloor$. Such a computation will make use of the following Theorem (which uses the FFT algorithm; see [20]), and is summarized in Corollary 4.
Algorithm 1 The new algorithm.

APPROXHAM($T_j, P, \epsilon$)
1 for $i = 1$ to $c \log n$
2 do Pick a random $h : \Sigma \rightarrow \{1, 2, \ldots, 2^\epsilon\}$
3 compute $x_i = \text{HAM}(h(T_j), h(P))$
4 return $\max_{1 \leq i \leq c \log n} \{x_i\}$

**Theorem 3.** Given a binary text $T$ of size $n$ and a binary pattern $P$ of size $m$, there exists an $O(n \log m)$ time algorithm that computes for all locations $j$ in $T$ the number of times that a 1 in $T_j$ is aligned with a 1 in $P$.

**Corollary 4.** Given a text $T$ of size $n$ and a pattern $P$ of size $m$ both over alphabet $\Sigma$, there exists an $O(|\Sigma| \cdot n \log m)$ time algorithm that computes $\text{HAM}(T_j, P)$ for all locations $j$ in $T$.

The algorithm for Corollary 4 is implemented by considering a separate binary text and binary pattern for each character $\sigma \in \Sigma$. For character $\sigma$ in this set, occurrences of $\sigma$ in $T$ are assigned to 1, while occurrences of other characters are assigned to 0. On the other hand, occurrences of $\sigma$ in $P$ are assigned to 0, while occurrences of other characters are assigned to 1. Applying Theorem 3 on the binary text and pattern defined by $\sigma$ enables computing for every location $j$ the number of times character $\sigma$ in $T_j$ contributes to $\text{HAM}(T_j, P)$. A summation over all the mismatches for all the characters in $\Sigma$ completes the computation of $\text{HAM}(T_j, P)$. Since Theorem 3 is applied $|\Sigma|$ times, the Corollary follows. Notice that when the algorithm of Corollary 4 is applied, then each location in the text is charged an $O(|\Sigma| \log m)$ (global) time cost.

The algorithm

With the goal of easing the presentation of our algorithm, we focus on estimating the Hamming distance between $T_j$ and $P$, and count the cost of global and local operations for this location. Since we are interested in algorithms that succeed with high probability (at least $1 - \frac{1}{n^{\Theta(1)}}$) then it suffices to show that with high probability the algorithm succeeds at location $j$. The pseudo-code for the algorithm is given in Algorithm 1.

Time complexity

Computing the Hamming distance between the projected text and projected pattern in Line 3 takes place by applying the algorithm from Corollary 4 where the alphabet is $[2^\epsilon]$. Thus, the total time cost for location $j$ is $O(\frac{1}{\epsilon} \log n \log m)$, and so the overall time cost for all locations is $O(\frac{n}{\epsilon} \log n \log m)$.

Correctness

Let $d = HAM(T_j, P)$. The goal of the algorithm is to approximate $d$. Notice that the expected value of $x_i$ is $E[x_i] = (1 - \frac{\epsilon}{2})d$, since each mismatch in the original text and pattern remains a mismatch after the projection obtained by applying $h$ with probability $1 - \frac{\epsilon}{2}$. Thus,
A Simple Algorithm for Approximating the Text-To-Pattern Hamming Distance

\[ E[d - x_i] = \frac{\epsilon d}{2}, \] and so by the Markov inequality,

\[ \Pr[x_i < (1 - \epsilon)d] = \Pr[d - x_i > \epsilon d] \leq \frac{E[d - x_i]}{\epsilon d} = \frac{1}{2}. \]

Since the algorithm returns the largest \( x_i \) value, the only way in which the algorithm fails is if all of the \( x_i \) values are less than \((1 - \epsilon)d\). Since the choices of the projections is independent, this happens with probability at most \( n^{-e} \).

References

10. A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Accepted to 56th IEEE Symposium on Foundations of Computer Science (FOCS), 2015.


