Parikh Matching in the Streaming Model

Lap-Kei Lee[1], Moshe Lewenstein[2], and Qin Zhang[3]

[1] Department of Computer Science, University of Hong Kong
lklee@cs.hku.hk
[2] * Department of Computer Science, Bar-Ilan University, Israel
moshe.lewenstein@gmail.com
[3] MADALGO**, Aarhus University
qinzhang@cs.au.dk

Abstract. Let S be a string over an alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$. A *Parikh-mapping* maps a substring S' of S to a $|\Sigma|$-length vector that contains, in location i of the vector, the count of σ_i in S'. *Parikh matching* refers to the problem of finding all substrings of a text T which match to a given input $|\Sigma|$-length count vector.

In the streaming model one seeks space-efficient algorithms for problems in which there is one pass over the data. We consider Parikh matching in the streaming model. To make this viable we search for substrings whose Parikh-mappings approximately match the input vector. In this paper we present upper and lower bounds on the problem of approximate Parikh matching in the streaming model.

1 Introduction

A *fingerprint* of a string is a smaller, more condensed identifier string. The notion of fingerprints of text substrings dates back to the early pattern matching papers of Karp, Miller and Rosenberg [13] and Karp and Rabin [12]. In the prior it was used for the classical pattern matching problem and was a building block of the then-introduced *renaming technique*. The idea of fingerprint and renaming has been used widely to solve many pattern matching applications, e.g. it is used implicitly in suffix array construction algorithms [16,14]. In the latter fingerprints were used to efficiently implement a randomized pattern matching algorithm. The fingerprints in the former are numerical identifiers for substrings of lengths of power 2 and the fingerprints in the latter are formed by transforming pattern-length substrings into small numbers by hashing them via polynomials into small numbers using modulo of random numbers.

Amir et al. in [1] presented *fingerprint matching* that produces a set of fingerprints for a text for a special subset of the substrings. The choice of the subset and the type of the fingerprints are motivated by *Parikh-mappings*. For a given

* The author was on a research visit to MADALGO during work on this paper. This research was supported by the Israel Science Foundation (grant no. 1848/04).

** MADALGO is the Center for Massive Data Algorithmics, a center of the Danish National Research Foundation.

L. Calderón-Benavides et al. (Eds.): SPIRE 2012, LNCS 7608, pp. 336–341, 2012.
© Springer-Verlag Berlin Heidelberg 2012

string S over alphabet Σ, Parikh-mappings are maps from the substrings S' of S into $|\Sigma|$-length arrays containing the count of the character σ_i in S' in location i. Fingerprints, in the model of Amir et al. are binary $|\Sigma|$-length arrays for S', where location i is 0 if σ_i does not appear in S' and 1 if it does. Amir et al. were interested in the unique set of fingerprints. The problem of finding them was dubbed *text fingerprinting* and has also appeared under the name of the problem of *character sets* [6]. In this setting several problems are posed:

1. Preprocess S to quickly answer queries of the form: Given k compute the number of distinct fingerprints of size k.
2. Preprocess S to quickly answer queries of the form: Given a subset $\phi \in \Sigma$, compute the number of substrings in S with fingerprint ϕ.
3. Compute all maximal locations of substrings having a given fingerprint.
4. Compute for each fingerprint ϕ in S all the maximal locations of ϕ.

Text fingerprinting has applications in lexical categorization and the above-mentioned problems are used for finding common intervals in multiple sequences [5,19], for permutation pattern discovery in biosequences [8], and other computational biology tasks. Extensions of text fingerprinting to graphs have been addressed in [15,9].

The more general problem where the Parikh-mappings themselves are considered is defined as follows:

Parikh Matching

Input: A string S of length n over alphabet Σ and a $|\Sigma|$-length array A of numbers, such that the sum of numbers is k.

Output: All locations i where the k-length substring S' beginning at that location has Parikh-mapping A.

This problem has been explored under the names of *Abelian pattern matching* [7] and *jumbled pattern matching* [3]. It has also been implicitly used in several pattern matching algorithms, e.g. filtering for approximate pattern matching [11]. It is straightforward to solve this problem in $O(n)$ time, but has been addressed in the case where there are many patterns to seek in parallel [3]. We are interested in this problem in the setting of the streaming model.

In the streaming setting one is given a large streaming text for which space is constrained and one desires to answers queries as the data moves by on-the-fly. This model has been shown to be useful for numerous problems such as frequency moments, heavy hitters, counting distinct elements etc. Also the problem of pattern matching in the streaming model has been addressed efficiently [18,2]. Therefore, it would seem that it makes sense that Parikh matching, at least the approximate version, would be potentially achievable with reasonable space. This would be very useful for problems such as large DNA data, or other large data, streamed through on small memory devices. Unfortunately, we show that not too much can be achieved. We then continue to show almost-tight upper bounds to match the lower bounds we present.

2 Parikh Matching in a Data Stream

Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ be an alphabet where $k = |\Sigma|$. For any string S, let $f(S) = (s_1, s_2, \ldots, s_k)$ be the Parikh-mapping of S which contains the count s_i of σ_i in S for all i. Given two strings X and Y of the same size, where $f(X) = (x_1, x_2, \ldots, x_k)$ and $f(Y) = (y_1, y_2, \ldots, y_k)$, the L_p distance (for $0 < p \leq 2$) between X and Y is defined to be

$$L_p(X, Y) = (\textstyle\sum_{i=1}^{k} |x_i - y_i|^p)^{1/p} .$$

In Parikh matching one is given a text T and pattern P both over alphabet Σ such that A is the Parikh-mapping of P. One desires to find all locations i of T where the n-length substring has Parikh-mapping A. The approximate version seeks locations i of T where the n-length substring T_i has a Parikh-mapping B such that $L_p(A, B)$ is bounded by a parameter to be described later.

We study approximating Parikh matching in the streaming model. We are given a character stream containing a pattern P followed by a text T. Let $n = |P|$ and $m = |T|$. Let T_i denote the substring $T[i..i + n - 1]$. Given a threshold $1 \leq \theta \leq 2n$ and an error bound $0 < \epsilon < 1$, we want to decide for each text position $1 \leq i \leq |T| - n + 1$, whether $L_p(P, T_i) \geq \theta$ (output 1) or $L_p(P, T_i) \leq (1 - \epsilon)\theta$ (output 0) . The decision can be arbitrary for any value in the middle.

3 Lower Bound

This section presents space lower bounds for deterministic and randomized algorithms. The core idea is that deciding whether $L_p = 0$ requires a lot of memory.

Theorem 1. *For any $0 < p \leq 2$ and any threshold $1 \leq \theta \leq \frac{n}{4}$, any deterministic algorithm A requires $\Omega(\frac{n}{\theta} \cdot 2^{1/p})$ bits of memory.*

Proof. Suppose A requires S bits of space. For some integer b (to be defined), we show how to use $S + 2\log b$ bits to store a b-bit vector B exactly. Thus, $S = \Omega(b)$.

We construct the pattern P and text T as inputs of A, as follows. Let $\Sigma = \{0, 1\}$. Let $x > 0$ be an integer to be defined later. A *0-block* is x consecutive 0's, and a *1-block* is x consecutive 1's. The pattern P contains b 0-blocks followed by b 1-blocks, so $f(P) = (bx, bx)$. The text T begins with b blocks: if the i-th bit of B is 1, the i-th block is a 1-block; otherwise, it is a 0-block. We also count the number of 0's and 1's in B, denoted by C_0 and C_1, so $C_0 + C_1 = b$. Then, we append C_1 0-blocks and then C_0 1-blocks to T. Recall that T_i is the substring $T[i..i + |P| - 1]$. Thus, $f(T_0) = (bx, bx)$ and we have $L_p(P, T_0) = 0$. The counters C_0, C_1 and algorithm A require $S + 2\log b$ bits of space.

We can recover B as follows. We first copy the current memory state of A, feed this copy with a 0-block, and then make a query. (*Case 1*) If B's first bit is 1, the substring T_x contains $b + 1$ 0-blocks and $b - 1$ 1-blocks, which implies $L_p(P, T_x) = (x^p + x^p)^{1/p} = 2^{1/p}x$, which is at least θ by setting $x = \lceil 2^{-1/p} \cdot \theta \rceil$. As $n = |P| = 2bx$, we have $b = \frac{n}{2x} = \Theta(\frac{n}{\theta} \cdot 2^{1/p})$. (*Case 2*) If B's first bit is 0,

$L_p(P, T_x) = 0$, which is less than $(1-\epsilon)\theta$ since $\epsilon > 0$ and $\theta \geq 1$. Thus, we can determine B's first bit using the copy of A. We let the original copy of A read a 0-block if B's first bit is 0, and a 1-block otherwise. The above procedure can be repeated to determine other bits in B. Therefore, $S = \Omega(b) = \Omega(\frac{n}{\theta} \cdot 2^{1/p})$. □

We extend the above result to randomized algorithms using an idea from [4].

Theorem 2. *For any $0 < p \leq 2$ and any threshold $1 \leq \theta \leq \frac{n}{4}$, any randomized algorithm with success probability at least $1-\delta$ (for $\delta < \frac{1}{2}$) requires $\Omega(\frac{n}{\theta} \cdot 2^{1/p} + \log(1-2\delta))$ bits of memory.*

Proof. We use an analogous version of Yao's minimax principle for Monte Carlo randomized algorithms [17]: For $\delta < \frac{1}{2}$, $\frac{1}{2}$ of the expected space complexity of the optimal deterministic algorithm that returns an answer within the error bound ϵ with probability at least $1-2\delta$ for an arbitrarily chosen input distribution $\mathbf{p}$ is a lower bound on the expected space complexity (under the worst input) of the optimal randomized algorithm with error bound ϵ and success probability $1-\delta$.

As shown in the proof of Theorem 1, a deterministic algorithm for the streaming problem can be used to store a bit vector of size $b = \Theta(\frac{n}{\theta} \cdot 2^{1/p})$. Let $\mathbf{p}$ be the uniform distribution over all bit vectors of the same size b. Consider a deterministic algorithm A that returns an answer within the error bound ϵ with probability at least $1-2\delta$ over $\mathbf{p}$. Then, A needs to differentiate at least $1-2\delta$ fraction of the input bit vectors. As a result, the expected space complexity of the optimal deterministic algorithm is at least equal to the optimal coding length of these $1-2\delta$ fraction of input bit vectors. The latter is at least equal to the entropy of these bit vectors, which is $\Omega(\log((1-2\delta) \cdot b)) = \Omega(\frac{n}{\theta} \cdot 2^{1/p} + \log(1-2\delta))$. □

4 Near-Tight Algorithms

We show that it is possible to obtain space efficient algorithms for L_1 and L_2.

Theorem 3. *There exists an algorithm which for any $0 < \epsilon, \delta < 1$, maintains the correct approximate estimates at any time with probability $1-\delta-1/k$ and uses $O(\frac{n}{\theta} \cdot \log(mk/\epsilon\delta) \log(k/\epsilon\delta) \log(1/\delta)/\epsilon^3)$ bits of random access storage. The processing time per item is $O(\frac{n}{\theta} \cdot \log(mk/\epsilon\delta) \log(1/\delta)/\epsilon^3)$.*

Theorem 3 is a direct application of a theorem by Indyk [10]. Let S be a string and $f(S)$ be its Parikh-mapping.

Theorem 4. *([10]) There is an algorithm in the turnstile date stream model which for any $0 < \epsilon, \delta < 1$ estimates $L_1(f(S))$ or $L_2(f(S))$ up to a factor $(1 \pm \epsilon/4)$ [1] with probability $1-\delta-1/k$ and uses $O(\log(mk/\epsilon\delta) \log(k/\epsilon\delta) \log(1/\delta)/\epsilon^2)$ bits of random access storage. The processing time per item is $O(\log(mk/\epsilon\delta) \cdot \log(1/\delta)/\epsilon^2)$.*

[1] The constant 4 in $\epsilon/4$ is just to make this theorem easily applicable in the proof of Theorem 3.

Indyk's algorithm uses random linear mappings to map the Parikh-mapping $f(S)$ of length k to $\ell = O(\log(1/\delta)/\epsilon^2)$ random variables each of which can be stored using $O(\log(mk/\epsilon\delta))$ bits, and then take the median of these random variables. Let $\phi_1, \phi_2, \ldots, \phi_\ell$ denote such ℓ random linear mappings. These linear mappings can be computed in the streaming model using $O(\log(mk/\epsilon\delta) \cdot \log(k/\epsilon\delta)\log(1/\delta)/\epsilon^2)$ bits of storage (including storing the random bits used in the random mappings).

Let $\hat{T}_j$ be the prefix of T of length j. Let $\gamma = \theta\epsilon/8n$. We call $\phi_i(f(\hat{T}_{\gamma n}))$, $\phi_i(f(\hat{T}_{2\gamma n})), \ldots$ for each $i \in [\ell]$ milestones of the text T. In our problem we store for each $i \in [\ell]$ the latest $2/\gamma$ milestones of text T. This uses $O(1/\gamma \cdot \log(mk/\epsilon\delta)\log(k/\epsilon\delta)\log(1/\delta)/\epsilon^2)$ bits of storage .

In a sliding window query at time t, let $h = \lfloor t/\gamma n \rfloor \gamma n$. We compute $W_i^h = \phi_i(\hat{T}_h) - \phi_i(\hat{T}_{h-n}) - \phi_i(P)$ for each $i \in [\ell]$ and then choose $\tilde{W}^h$ as the median of $W_1^h, W_2^h, \ldots, W_\ell^h$. The algorithm outputs 1 if $\tilde{W}^h \geq \theta - \epsilon\theta/2$ and 0 otherwise. Let W^h be the exact distance between P and $T[h, h+n-1]$. By Theorem 4 and the fact that inserting/deleting one character can only change L_1 or L_2 distance by 1. We have that with probability at least $1 - \delta - 1/k$,

$$(1 - \epsilon/4)W^h - 2\gamma n \leq \tilde{W^h} \leq (1 + \epsilon/4)W^h + 2\gamma n.$$

Now if $W^h \geq \theta$ then we must have $\tilde{W}^h \geq (1 - \epsilon/4)\theta - 2\gamma n \geq \theta - \epsilon\theta/2$, and if $W^h \leq (1 - \epsilon)\theta$ then we must have $\tilde{W}^h \leq (1 + \epsilon/4)(1 - \epsilon)\theta + 2\gamma n < \theta - \epsilon\theta/2$. This completes the proof of the correctness of the algorithm.

References

1. Amir, A., Apostolico, A., Landau, G.M., Satta, G.: Efficient text fingerprinting via Parikh mapping. J. Discrete Algorithms 1(5-6), 409–421 (2003)
2. Breslauer, D., Galil, Z.: Real-Time Streaming String-Matching. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 162–172. Springer, Heidelberg (2011)
3. Burcsi, P., Cicalese, F., Fici, G., Liptak, Z.: On Approximate jumbled pattern matching in strings. Theory Comput. Syst. 50(1), 35–51 (2012)
4. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM J. Comput. 31(6), 1794–1813 (2002)
5. Didier, G.: Common Intervals of Two Sequences. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 17–24. Springer, Heidelberg (2003)
6. Didier, G., Schmidt, T., Stoye, J., Tsur, D.: Character sets of strings. J. Discrete Algorithms 5(2), 330–340 (2007)
7. Ejaz, T.: Abelian pattern matching in strings. Ph.D. Dissertation, Faculty of Informatics, Technical University of Dortmund (2010)
8. Eres, R., Landau, G.M., Parida, L.: Permutation pattern discovery in biosequences. Journal of Computational Biology 11(6), 1050–1060 (2004)
9. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Upper and lower bounds for finding connected motifs in vertex-colored graphs. J. Comput. Syst. Sci. 77(4), 799–811 (2011)
10. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computation. JACM 53(3), 307–323 (2006)

11. Jokinen, P., Tarhio, J., Ukkonen, E.: A comparison of approximate string matching algorithms. Softw., Pract. Exper. 26(12), 1439–1458 (1996)
12. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development 31(2), 249–260 (1987)
13. Karp, R.M., Miller, R.E., Rosenberg, A.L.: Rapid identification of repeated patterns in strings, trees and arrays. In: Proc. STOC, pp. 125–136 (1972)
14. Karkkainen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. JACM 53(6), 918–936 (2006)
15. Lacroix, V., Fernandes, C.G., Sagot, M.-F.: Reaction Motifs in Metabolic Networks. In: Casadio, R., Myers, G. (eds.) WABI 2005. LNCS (LNBI), vol. 3692, pp. 178–191. Springer, Heidelberg (2005)
16. Manber, U., Myers, E.W.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. 22(5), 935–948 (1993)
17. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press (1995)
18. Porat, B., Porat, E.: Exact and approximate pattern matching in the streaming model. In: Proc. FOCS, pp. 315–323 (2009)
19. Schmidt, T., Stoye, J.: Quadratic Time Algorithms for Finding Common Intervals in Two and More Sequences. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 347–358. Springer, Heidelberg (2004)