Approximate Matching in the L_∞ Metric

Ohad Lipsky* Ely Porat**

Bar-Ilan University

Abstract. Let a text $T = t_0, \dots, t_{n-1}$ and a pattern $P = p_0, \dots, p_{m-1}$, strings of natural numbers, be given. In the *Approximate Matching in the L_∞ metric* problem the output is, for every text location i , the L_∞ distance between the pattern and the length m substring of the text starting at i , i.e. $\max_{j=0}^{m-1} |t_{i+j} - p_j|$. We consider the *Approximate $k - L_\infty$ distance* problem. Given text T and pattern P as before, and a natural number k the output of the problem is the L_∞ distance of the pattern from the text only at locations i in the text where the distance is bounded by k . For the locations where the distance exceeds k the output is ϕ . We show an algorithm that solves this problem in $O(n(k + \log(\min(m, |\Sigma|)))) \log m$ time.

1 Introduction

One of the classical approximate pattern matching problems is the *String matching with Mismatches* [3]. The input is a text string $T = t_0, \dots, t_{n-1}$ and pattern string $P = p_0, \dots, p_{m-1}$ and the output is the number of mismatches between the pattern and every length m substring of the text. The number of mismatches is a very intuitive way for comparing strings, motivated by typing errors. However, this measure is not completely suitable in many fields. In computer vision, for example, one might allow the pattern pixel to differ from text pixel within some tolerated distance, without causing a mismatch. In music information retrieval [9] one might consider half tone changes as allowed, and seek for large distortions only. In various other applications the L_∞ Minkowsky norm is in use. The problem of L_∞ matching was discussed by Lipsky [7], and in recent papers [4, 1]. In [7, 1] an $O(n|\Sigma| \log(m + |\Sigma|))$ time algorithm is presented for the L_∞ matching problem. Their technique was also extended to closest pair problems [5] and to some other pattern matching problems [6]. In [4] They showed how to find all locations for which the L_∞ distance is less than some given δ in $O(\delta n \log m)$ time. We present a simpler solution, that not only finds the locations it matches, but also computes the exact L_∞ distance at those locations. Let T^i denote the length m substring of the text starting at i .

* Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel, +972 3 531-8408; ohadlipsky@yahoo.com.

** Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, 972 3 531-7620; porately@cs.biu.ac.il. Partially supported by GIF Young Scientists Program grant 2055-1168.6/2002.

2 The algorithm:

Algorithm Outline:

1. Discard all locations with distance $> 2k$.
2. Reduce the alphabet to an alphabet of size $2k$.
3. Run the L_∞ algorithm with the reduced alphabet.
4. Take distances as cyclic.

We discuss each step in detail.

Discarding locations with too high a distance: We use here an approximation method for L_∞ which approximates the L_∞ values up to a factor of $1 \pm \epsilon$ [8]. We use $\epsilon = \frac{1}{4}$. Then we discard every location with distance greater than $\frac{5}{4}k$. This ensures us that location with distance greater than $\frac{5}{3}k$ are discarded for sure, and location with distance less than k are kept for sure. The locations with distance between k and $\frac{5}{3}k$ might be discarded, and might be kept. In the next steps we will exploit the fact that while there are locations with distance greater than k , since they are bounded by $5k/3$ it will only add a constant factor. The time needed for this step is $O(n \log m \log |\Sigma|)$.

Reducing the alphabet: This is done simply by taking each number modulo $4k$, i.e. use $T' = t_0 \bmod 4k, t_1 \bmod 4k, \dots, t_{n-1} \bmod 4k$ and $P' = p_0 \bmod 4k, p_1 \bmod 4k, \dots, p_{m-1} \bmod 4k$ instead of the original T and P . The idea behind this step is the following lemma:

Lemma 1. *For any integers x, y, k s.t. $|x - y| < 2k$*

$$|x \bmod 4k - y \bmod 4k| = |x - y| \text{ or } |x \bmod 4k - y \bmod 4k| = 4k - |x - y|$$

Corollary 1. *For any integers x, y, k s.t. $|x - y| < 2k$*

$$|x - y| = |x \bmod 4k - y \bmod 4k| \text{ or } |x - y| = 4k - |x \bmod 4k - y \bmod 4k|$$

This step is executed in linear time.

Run L_∞ algorithm with reduced alphabet: We simply use the algorithm from the next section on P' and T' , the alphabet size of T' and P' is bounded by $4k$. This step will take $O(nk \log(m + k))$ time. Let $l_0, \dots, l_{n-m}$ be the output.

Take distances as cyclic: Following from the corollary we know that for each non discarded location i either the L_∞ distance of the pattern from the length m substring of the text starting at location i equals l_i or it equals $4k - l_i$. Since location i is not discarded we know that the output should be less than $2k$, and therefore only one of the two options fit. This step is executed in linear time.

3 L_∞ matching $O(n|\Sigma| \log(m + |\Sigma|))$ Algorithm

The method in this algorithm is encoding the text and the pattern in such a way that in a single convolution, and a linear time pass on the convolution result we compute the output.

Key idea: We look at one text number, t , and one pattern number p . We encode both of them to a $|\Sigma|$ long binary strings. The encoding of t is all 0's except the t -th bit which is 1, and similarly with p , which is encoded to all 0's except the p -th bit. Let $c(i)$ denote the encoded i . Now, we start by $c(p)$ aligned below $c(t)$ and start at position $-|\Sigma|$ (where $c(t)$ fixed to start at position 1). We move $c(p)$ to the right till both 1-bits are one below the other. At this position, the distance between the starting position of $c(t)$ and the starting position of $c(p)$ equals to the difference $|t - p|$, an example is given in Figure 1. If we look at $r = c(t) \otimes c(p)$ we will have either $r[-|t - p|] = 1$ or $r[|t - p|] = 1$. Extending this idea to encoding strings of numbers requires adding leading (or tracing) zeros between the encoded numbers.

Fig. 1. $c(p)$ moved below $c(t)$ till the 1-bits are aligned.

In detail: first, define $\chi_{\neq 0}(x) = 1$ if $x \neq 0$ and 0 otherwise. Next, define For every $x \in \Sigma = \{1, \dots, n\}$, $c^t(x) = c^t(x)_1, \dots, c^t(x)_{2|\Sigma|}$ where $c^t(x)_i = 1$ if $i = |\Sigma| + x$ and 0 otherwise. Similarly define $c^p(x) = c^p(x)_1, \dots, c^p(x)_{2|\Sigma|}$ where $c^p(x)_i = 1$ if $i = x$ and 0 otherwise.

Algorithm Steps

1. Construct $c^t(T) = c^t(t_1) \cdots c^t(t_n)$
2. Construct $c^p(P) = c^p(p_1) \cdots c^p(p_m)$
3. Compute $R = c^t(T) \otimes c^p(P)$
4. For $i = 1, \dots, n - m + 1$
 $O[i] \leftarrow \max_{s=-|\Sigma|}^{|\Sigma|} \chi_{\neq 0}(R[(2i-1)|\Sigma| + 1 + s])|s|$

Claim. At the end of the algorithm $O[i] = \max_{j=1}^m |t_{i+j-1} - p_j|$.

Proof: First, we show that $O[i] \geq |t_{i+j-1} - p_j|$ for every $j \in \{1, \dots, m\}$. In order to see that, it is enough to show that for every $j \in \{1, \dots, m\}$ it holds that $R[(2i-1)|\Sigma| + 1 + t_{i+j-1} - p_j] \neq 0$ (since, then, the value of $|t_{i+j-1} - p_j|$ is one of the values for the *max* taken in step 4). Now, since $R[(2i-1)|\Sigma| + 1 + t_{i+j-1} - p_j] = \sum_{k=1}^{2m|\Sigma|} c^t(T)_{(2i-1)|\Sigma|+1+t_{i+j-1}-p_j+k-1} c^p(P)_k$ and for $k = p_j$ we have $c^t(T)_{(2i-1)|\Sigma|+1+t_{i+j-1}} = 1$ and $c^p(P)_{p_j} = 1$ (from the way we defined the encoding) it holds that $R[(2i-1)|\Sigma| + 1 + t_{i+j-1} - p_j] \neq 0$. We have left to show that $O[i] = |t_{i+j-1} - p_j|$ for some $j \in \{1, \dots, m\}$. Let s_m be the value for which $O[i] = \chi_{\neq 0}(R[(2i-1)|\Sigma| + 1 + s_m])|s_m|$. We can assume that $R[(2i-1)|\Sigma| + 1 + s_m] \neq 0$ (otherwise $O[i] = 0$ and $0 \leq \max_{j=1}^m |t_{i+j-1} - p_j|$ and our proof is done). The fact that $R[(2i-1)|\Sigma| + 1 + s_m] \neq 0$ implies that for some $j' \in \{1, \dots, m\}, k' \in \{0, \dots, 2|\Sigma| - 1\}$ we have $c^t(T)_{(2i-1)|\Sigma|+1+s_m+2j'|\Sigma|+k'-1} = c^t(T)_{(2(i+j')-1)|\Sigma|+s_m+k'} = 1$ and $c^p(P)_{2j'|\Sigma|+k'} = 1$, which in turn implies that for j' it holds that $|t_{i+j'-1} - p_{j'}| = s_m$. This complete our proof.

Time: The time needed to convolve 2 strings of size $n|\Sigma|$ and $m|\Sigma|$ is $O(n|\Sigma| \log n)$. The computation of $O[i]$ takes $2|\Sigma|$ steps, and $i = 1, 2, \dots, n - m + 1$ so this step takes $O(n|\Sigma|)$. Both steps together take $O(n|\Sigma| \log n)$. We can slightly improve the time by using the technique of cutting the text into n/m overlapping segments, each of length $2m$ to a total time of $O(n|\Sigma| \log(m + |\Sigma|))$.

References

1. Amihood Amir, Ohad Lipsky, Ely Porat, and Julia Umanski. Approximate matching in the l_1 metric. In Apostolico et al. [2], pages 91–103.
2. Alberto Apostolico, Maxime Crochemore, and Kunsoo Park, editors. *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, volume 3537 of *Lecture Notes in Computer Science*. Springer, 2005.
3. Alberto Apostolico and Zvi Galil, editors. *Combinatorial Algorithms on Words*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1985.
4. Peter Clifford, Raphaël Clifford, and Costas S. Iliopoulos. Faster algorithms for delta, gamma-matching and related problems. In Apostolico et al. [2], pages 68–78.
5. Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 782–792. Springer, 2004.
6. Piotr Indyk, Ohad Lipsky, and Ely Porat. Approximate translation matching, manuscript. 2004.
7. Ohad Lipsky. Efficient distance computations. Master’s thesis, Bar-Ilan University, Department of Computer Science, 2003.
8. Ohad Lipsky and Ely Porat. Approximate l_∞ matching. Technical report, Department of Computer Science, 2003.
9. Ilya Shmulevich, O. Yli-Harja, E. Coyle, D. Povel, and K. Lemstrom. Perceptual issues in music pattern recognition - complexity of rhythm and key fining. 1999.