Asynchronous Pattern Matching – Metrics (Extended Abstract)*

Amihood Amir

Bar-Ilan University and Georgia Tech
Department of Computer Science
52900 Ramat-Gan
ISRAEL

e-mail: amir@cs.biu.ac.il

Abstract. Traditional Approximate Pattern Matching (e.g. Hamming distance errors, edit distance errors) assumes that various types of errors may occur to the data, but an implicit assumption is that the order of the data remains unchanged.

Over the years, some applications identified types of "errors" were the data remains correct but its order is compromised. The earliest example is the "swap" error motivated by a common typing error. Other widely known examples such as transpositions, reversals and interchanges are motivated by biology.

We propose that it is time to formally split the concept of "errors in data" and "errors in address" since they present different algorithmic challenges solved by different techniques. The "errors in address" model, which we call asynchronous pattern matching, since the data does not arrive in a synchronous sequential manner, is rich in problems not addresses hitherto.

We will consider some reasonable metrics for asynchronous pattern matching, such as the number of inversions, or the number of generalized swaps, and show some efficient algorithms for these problems. As expected, the techniques needed to solve the problems are not taken from the standard pattern matching "toolkit".

1 Motivation

Historically, approximate pattern matching grappled with the challenge of coping with errors in the data. The traditional *Hamming distance* problem assumes that some elements in the pattern are erroneous, and one seeks the text locations where this number of errors is small enough [17, 14, 4], or efficiently calculating the Hamming distance at every text location [1, 16, 4]. The *edit distance* problem adds to the assumption that some elements of the text are deleted, or that noise is added at some text locations [18, 11]. Indexing and dictionary matching under these errors has also been considered [15, 12, 21, 10].

Implicit in all these problems is the assumption that there may indeed be errors in the **content** of the data, but the **order** of the data is inviolate. Data may be

*Partially supported by NSF grant CCR-01-04494 and ISF grant 82/01.

lost or noise may appear, but the relative position of the symbols is unchanged. Data **does not move around**. Even when *don't cares* were added [13], when non-standard models were considered[6, 20, 2] the order of the data was assumed to be ironclad.

Nevertheless, some non-conforming problems have been gnawing at the walls of this assumption. The *swap* error, motivated by the common typing error where two adjacent symbols are exchanged [19, 3], does not assume error in the content of the data, but rather, in the order. The data content is, in fact, assumed to be correct. Recently, the advent of computational biology has added more problems of order error to our repertoire. In evolution, one envisions a whole piece of genome to "detach" and "reconnect" in a different location, or two pieces of genome to "exchange" places. These phenomena, of course, are assumed to take place simultaneously with traditional data content errors, however, their nature is **rearrangement** of the data, rather than corruption of its contents.

It turns out that the overall problem of adding these new rearrangement operators to the content changing operators is extremely difficult. Thus more simplified problems were considered in the literature. The rearrangement operators were isolated and handled separately. Reversals [7], transpositions [5], and block interchanges [9] were explored. The edit distance problem under these new operations is still too difficult, therefore the *sorting permutation* version of these problems was researched.

This research direction led to interesting paths. First, the tools and techniques used were different from the traditional pattern matching tools. The results also seem more varied. The *sorting by reversal problem* is $\mathcal{NP}$-hard [8]. It is still open whether the *sorting by transposition problem* can be efficiently solved deterministically. Christie [9] gives an $O(n^2)$ algorithm for the *sorting by block interchange problem.*

In this paper, for the first time, we explicitely identify and formalize this different pattern matching paradigm, that of **errors in the order** rather than error in the content of the data. The advantages in formalizing this paradigm are:

1. Identifying the types of problems and techniques required, rather than than re-inventing ad-hoc solutions.

2. Understanding the theoretical underpinnings of the problem.

3. Generalizing to other possible rearrangements and possibly providing more general solutions.

One of the immediate understandings from a formal model definition of errors in order, is that one needs to consider appropriate *distance measures*. The error in content measures are not necessarily meaningful in these circumstances. We consider some generic error distances, such as minimum L_1 and L_2 distance on the address of the data. We also illustrate the fact that more specific distance measures are necessary for specific applications.

The main contributions of this research are: We give a formal framework of *rearrangement operators* and the distance measure they define. We also provide efficient algorithms for several natural operators and distance measures. It is exciting to point out that some techniques we use are totally new to pattern matching. This reinforces the realization that this new model is needed, as well as gives hopes to new research directions and paths in the field of pattern matching.

2 The New Model

We begin by an illustration of different applications requiring different rearrangement operators.

An Example. At the Formula-one races, cars and their designated drivers queue behind the start line at a precise, predetermined order. Suppose that the cars arrive at random order, and then have to rearrange into order. There is only a single passing lane, so that at any given time only one pair of cars can swap locations. What is the minimal number of swaps necessary in order to complete the rearrangement? Suppose that instead of reshuffling cars, the cars stay in place, and the drivers exchange cars. To do so, the drivers meet mid way and swap keys. In this case, multiple swaps can occur in parallel. What is the minimal number of parallel steps necessary in order to get all the drivers in order? Customarily, race cars and drivers are divided into groups. Suppose that the initial queuing order determines the ordering by group, not by specific car and driver. What is the minimum number of steps for the rearrangement in this case (sequential and parallel)?

Our new model considers how to efficiently answer these and similar questions.

Rearrangement Systems and Distances. Consider a set A and let x and y be two n-tuples over A. We wish to formally define the process of converting x to y through a sequence of *rearrangement* operations. A *rearrangement operator* π is a function $\pi : [1..n] \to [1..n]$, with the intuitive meaning being that for each i, π moves the element currently at location i to location $\pi(i)$. Let Π be a set of rearrangement operators, and let $w : \Pi \to R^+$ be a cost function, associating a non-negative cost with each operator. We call the pair (Π, w) a *rearrangement system*. Consider two vectors $x, y \in A^n$ and a rearrangement system $\mathcal{R} = (\Pi, w)$, we define the distance from x to y under $\mathcal{R}$. Let $s = (\pi_1, \pi_2, \ldots, \pi_k)$ be a sequence of rearrangement operators from Π, and let $\pi_s = \pi_1 \circ \pi_2 \circ \cdots \circ \pi_k$ be the composition of the π_j's. We say that s *converts* x *into* y if for any $i \in [1..n]$, $x_i = y_{\pi_s(i)}$. That is, y is obtained from x by moving elements according to the designated sequence of rearrangement operations. The *cost* of the sequence s is the sum of costs of the different operators in s, $w(s) = \sum_{j=1}^{k} w(\pi_j)$. The distance from x to y under $\mathcal{R}$ is defined as:

$$d_R(x, y) = \min\{w(s) | s \text{ converts } x \text{ to } y \}$$

If there is no sequence that converts x to y then the distance is ∞.

We extend the definition to tuples of different lengths. In this case, we define the distance between the two vectors to be the minimum distance between the shorter of the two and the closest contiguous subsequence of the longer.

We consider several natural rearrangement systems and the resulting distances. For these systems we provide efficient algorithms to compute the distances.

The Swaps Distance. We first consider the set of rearrangement operators were in each operation the location of exactly two entries can be swapped (as in the car rearrangement example above). The cost of each swap is 1. We call the resulting distance the *swaps distance*. We prove:

Theorem 1. *For tuples x and y of sizes m and n respectively ($m \leq n$) where all entries of x are distinct the swaps distance can be computed in time $O(m(n-m+1))$.*

The Parallel-Swaps Distance. Next we consider the case were in each rearrangement operation multiple pairs can be swapped, but any element can participate in at most one swap per operation (as in the drivers swap example above). Formally, this corresponds to the set of all permutation with cycles of length at most 2. The cost of each such permutation is 1. We call the resulting distance the *parallel swaps distance*, denoted by $d_{p\text{-}swap}(\cdot,\cdot)$. We prove:

Theorem 2. *For any two tuples x and y, either $d_{p\text{-}swap}(x,y) = \infty$ or $d_{p\text{-}swap}(x,y) \leq 2$.*

This means that for any two tuples x and y that are identical as multi-sets, it is possible to convert one to the other using only two parallel steps of swap operations! We also prove:

Theorem 3. *For tuples x and y of sizes m and n respectively ($m \leq n$) with k distinct entries in x, the parallel swaps distance can be computed deterministically in time that is the minimum of $O(k^2 n \log m)$ and $O(m(n-m+1))$.*

and,

Theorem 4. *For tuples x and y of sizes m and n respectively ($m \leq n$), the parallel swaps distance can be computed randomly in expected time that is the minimum of $O(n \log m)$ and $O(m(n-m+1))$.*

The L_1 Rearrangement Distance. Consider the set of rearrangement operations where in each operation exactly one element is moved. The element can be moved to any other location, and the cost of the operation is the distance the element is moved. We call this the L_1 *Rearrangement System* and the resulting distance the L_1 *Rearrangement Distance.* We prove:

Theorem 5. *For tuples x and y of sizes m and n respectively ($m \leq n$), the L_1 Rearrangement Distance can be computed in time $O(m(n-m+1))$. If all entries of x are distinct, then the distance can be computed in time that is the minimum of $O(n \log\log m)$ and $O(m(n-m+1))$*

The L_2 Rearrangement Distance. Consider the same set of operations as in the L_1 Rearrangement System, only that the cost of an operation is the square of the distance. We call this the the L_2 *Rearrangement System*, and the resulting distance the L_2 *Rearrangement Distance.* We prove:

Theorem 6. *For tuples x and y of sizes m and n respectively ($m \leq n$), the L_2 Rearrangement Distance can be computed in time that is the minimum of $O(n \log m)$ and $O(m(n-m+1))$*

References

[1] K. Abrahamson. Generalized string matching. *SIAM J. Comp.*, 16(6):1039–1051, 1987.

[2] A. Amir, A. Aumann, R. Cole, M. Lewenstein, and E. Porat. Function matching: Algorithms, applications, and a lower bound. In *Proc. 30th ICALP*, pages 929–942, 2003.

[3] A. Amir, R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. Overlap matching. *Information and Computation*, 181(1):57–74, 2003.

[4] A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 2004.

[5] V. Bafna and P.A. Pevzner. Sorting by transpositions. *SIAM J. on Discrete Mathematics*, 11:221–240, 1998.

[6] B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th Annual ACM Symposium on the Theory of Computation*, pages 71–80, 1993.

[7] P. Berman and S. Hannenhalli. Fast sorting by reversal. In D.S. Hirschberg and E.W. Myers, editors, *Proc. 8th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1075 of *LNCS*, pages 168–185. Springer, 1996.

[8] A. Carpara. Sorting by reversals is difficult. In *Proc. 1st Annual Intl. Conf. on Research in Computational Biology (RECOMB)*, pages 75–83. ACM Press, 1997.

[9] D. A. Christie. Sorting by block-interchanges. *Information Processing Letters*, 60:165–169, 1996.

[10] R. Cole, L. Gottlieb, and M. Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. 36th annual ACM Symposium on the Theory of Computing (STOC)*, pages 91–100. ACM Press, 2004.

[11] R. Cole and R. Hariharan. Approximate string matching: A faster simpler algorithm. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 463–472, 1998.

[12] P. Ferragina and R. Grossi. Fast incremental text editing. *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 531–540, 1995.

[13] M.J. Fischer and M.S. Paterson. String matching and other products. *Complexity of Computation, R.M. Karp (editor), SIAM-AMS Proceedings*, 7:113–125, 1974.

[14] Z. Galil and R. Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17(4):52–54, 1986.

[15] M. Gu, M. Farach, and R. Beigel. An efficient algorithm for dynamic text indexing. *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 697–704, 1994.

[16] H. Karloff. Fast algorithms for approximately counting mismatches. *Information Processing Letters*, 48(2):53–60, 1993.

[17] G. M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.

[18] V. I. Levenshtein. Binary codes capable of correcting, deletions, insertions and reversals. *Soviet Phys. Dokl.*, 10:707–710, 1966.

[19] R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *J. of the ACM*, pages 177–183, 1975.

[20] S. Muthukrishnan and H. Ramesh. String matching under a general matching relation. *Information and Computation*, 122(1):140–148, 1995.

[21] S. C. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. *Proc. 37th FOCS*, pages 320–328, 1996.