# Approximate String Matching with Stuck Address Bits

Amihood Amir[1,4], Estrella Eisenberg[1], Orgad Keller[1], Avivit Levy[2,3], and Ely Porat[1]

[1] Department of Computer Science, Bar Ilan University, Ramat Gan 52900, Israel
{amir,kellero,porately}@cs.biu.ac.il
[2] Department of Software Engineering, Shenkar College, 12 Anna Frank, Ramat-Gan, Israel
avivitlevy@shenkar.ac.il
[3] CRI, University of Haifa, Mount Carmel, Haifa 31905, Israel
[4] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218

**Abstract.** A string $S \in \Sigma^m$ can be viewed as a set of pairs $\{(s_i, i) \mid s_i \in S,\ i \in \{0, \ldots, m-1\}\}$. We follow the recent work on *pattern matching with address errors* and consider approximate pattern matching problems arising from the setting where errors are introduced to the location component $(i)$, rather than the more traditional setting, where errors are introduced to the content itself $(s_i)$. Specifically, we continue the work on string matching in the presence of address bit errors. In this paper, we consider the case where bits of $i$ may be stuck, either in a consistent or transient manner. We formally define the corresponding approximate pattern matching problems, and provide efficient algorithms for their resolution.

## 1 Introduction

*Background.* Over 30 years ago, one of the co-authors of this paper was busy writing a program that points an antenna to a given moving location. Having written a program that converts latitude and longitude to the appropriate azimuth, taking all geodesic information into consideration, the program was finally tested.

The frustrated programmer noticed that the antenna was pointing to the west, when it was supposed to point north. In those days, de-bugging meant halting the computer and looking at the memory contents through a panel register. The programmer halted the program after it loaded the bus with the azimuth and immediately prior to giving the device the signal to load the azimuth, and checked the value. To his surprise, the value matched his calculations. He then resumed running the program and the antenna pointed exactly to the required direction. However, running the program from beginning to end again achieved a wrong result.

The problem was that some bits on the bus settled on their value faster than others, thus when those bits had a 0 value and the value was changed to 1, it took longer to settle than when a 1 was changed to a 0, or when the value was not changed. A short wait helped.

## 1.1 Pattern Matching with Address Errors

*Motivation.* An important implicit assumption in the traditional view of pattern matching was that there may indeed be errors in the *content* of the data, but the *order* of the data is inviolate. Consider a text $T = t_0 \dots t_{n-1}$ and pattern $P = p_0 \dots p_{m-1}$, both over an alphabet $\Sigma$. Traditional pattern matching regards $T$ and $P$ as *sequential* strings, provided and stored in sequence (e.g., from left to right). However, some non-conforming problems have been gnawing at the basis of this assumption. An example is the *swap* error, motivated by the common typing error where two adjacent symbols are exchanged [20,7,8,11], which does not assume error in the content of the data, but rather, in the order.

Computational biology has also added several problems wherein the "error" is in the order, rather than the content. During the course of evolution areas of genome may be shifted from one location to another. Considering the genome as a string over the alphabet of genes, these cases represent a situation where the difference between the original string and resulting one is in the locations rather than contents of the different elements. Several works have considered specific versions of this biological setting, primarily focusing on the sorting problem (*sorting by reversals* [13,14], *sorting by transpositions* [12], and *sorting by block interchanges* [15]).

The inherently distributed nature of the web is already causing (in Bit Torrent and Video on Demand) the phenomenon of transmission of a stream of data in tiny pieces from different sources. This creates the problem of putting scrambled data back together again.

Finally, in computer architecture, address errors are of no less concern than content errors [17]. It is by no means taken for granted that when seeking a word from a given address, no errors will occur in the address bits. This problem is relevant even when reading a buffer of consecutive words since these words are not necessarily consecutive in the disk or in an interleaved cache.

Therefore, the traditional view of strings is becoming, at times, too restrictive.

*The Model.* In such cases, it is more natural to view the string as a set of pairs $(\sigma, i)$, where $i$ denotes a location in the string, and $\sigma$ is the value appearing at this location. Given this view of strings, the problem of *approximate pattern matching* has been reconsidered in the last few years, and a new pattern matching paradigm – *pattern matching with address errors* – was proposed in [2]. In this model, the pattern *content* remains intact, but the relative positions (addresses) may change. Efficient algorithms for several different natural types of rearrangement errors were presented [3,9,4,19] (see also [10]). These types of address errors were inspired by biology, i.e., pattern elements exchanging their locations due to some external process.

*Address Bit Errors.* Another broad class of address errors inspired by computer architecture was studied by [1,6]. They consider errors which arise from a process