Efficient String Matching Algorithms for Combinatorial Universal Denoising

S. Chen* S. Diggavi† S. Dusad† S. Muthukrishnan*

Abstract

Inspired by the combinatorial denoising method DUDE *[13], we present efficient algorithms for implementing this idea for arbitrary contexts or for using it within subsequences. We also propose effective, efficient denoising error estimators so we can find the best denoising of an input sequence over different context lengths. Our methods are simple, drawing from string matching methods and radix sorting. We also present experimental results of our proposed algorithms.*

1 Introduction

We have a noise-free, unknown sequence $x^n = (x_1, \ldots, x_n)$ and the observed noisy sequence $z^n = (z_1, \ldots, z_n)$. We assume that the noisy sequence is observed as the output of a channel with known statistics. The *denoising problem* is to generate a posited $\hat{x}^n$ given z^n and the channel statistics. A number of approaches for the general denoising problem are known: based on transforms such as wavelets or based on similarity distances, etc. We focus on a recently proposed combinatorial approach for denoising when the channel corrupting the input sequence is a discrete memoryless channel (DMC). In this case, both the noise-free sequence as well as the noisy observations are assumed to belong to an alphabet $\mathcal{S}$ of size[1] m. The authors in [13] proposed an algorithm called DUDE that relies on computing for each string position z_i, certain empirical distributional value based on counting occurrences of its *context*, *i.e.*,, $z_{i-k}, \ldots, z_{i-1}, z_{i+1}, \ldots, z_{i+k}$. DUDE uses the empirical values together with the noise model to determine the likely x_i's. The authors [13] prove that DUDE is a universal denoiser in the sense that it asymptotically has the same performance as the best finite-order sliding window denoiser for both individual sequences as well as for stochastic input sequences.

Our research here is inspired by the elegance and effectiveness of DUDE and motivated by the following concerns. First, the algorithm as presented in [13] takes time[2]

*Dept. of Computer Science, Rutgers University, Email: {suche,muthu}@cs.rutgers.edu.

†School of Computer and Communication Sciences, Swiss Federal Institute of Technology (EPFL), Email: {suhas.diggavi,sanket.dusad}@epfl.ch. S. Dusad was supported in part by SNSF Grant # 200021-105640/1.

[1]The size of the input and output alphabet can be different provided under regularity conditions [13].

[2]The authors, in the most recent version of [13], state the running time to be $O(\min\{n, m^{2k+1}\}m^2)$ but

$O(nk/\log_m n + nm^2)$. This is quite efficient for small context k's (say $k = o(\log_m n)$) and small m's, but we have several examples we later show where the best denoising is obtained for significantly large k. Therefore, we need to choose the best context length when we are not necessarily in the asymptotic regime for given input sequences. Unfortunately, for $k = \omega(\log_m n)$, the running time is superlinear in n. Second, how do we find the suitable context length, or even the *optimal* one, that is, one that minimizes the error between the denoised sequence and x^n? For this, we need efficient methods to determine the cumulative loss function that quantifies the denoising error of DUDE for a given context length k. Since this loss function depends on the unknown input x^n, we need to have estimators for the loss function that depend *only* on the observed noisy sequence z^n. Recently in [10], an estimator for the cumulative loss function has been presented for the denoising error; however, it is rather expensive, taking time $O(nm^5)$, and hence more efficient estimators are needed specially for large alphabet sizes. Finally, it might be possible to denoise portions of the sequences using different context lengths locally or it might be useful to denoise only a portion of the noisy string using a specified context length. Hence, rather than denoising the entire sequence z^n with a single context length, a flexible way is needed to browse through the sequence z^n and explore data structures that enable the denoising of particular portions of the noisy sequence using specified context lengths. This is the denoising analog of the problem of compressing substrings studied in [1].

We address these concerns by studying a general data structural problem for subsequence denoising using `DUDE`. Formally, we study the following problem.

Problem 1 *We have a sequence z^n and a noise model in the form of a probability transition matrix. We are allowed to preprocess z^n to build efficient data structures to support* `DUDE-DENOISE`(i, j, k) *query that denoises the portion $[i, j]$ with context k specified in the query, or* `DUDE-CONTEXT`(i, j) *which returns the context size k such that* `DUDE-DENOISE`(i, j, k) *has the smallest error w.r.t $x^n[i, j]$.*

This is a general problem and we can query `DUDE-DENOISE`$(1, n, k)$ to solve the original denoising problem in [13] or `DUDE-CONTEXT`$(1, n)$ to find the best context length for the sequence z^n or search for these for any contiguous subsequence $[i, j]$. Our main results are as follows.

- For supporting the query `DUDE-DENOISE`$(1, n, k)$, we present a $O(n + \mathcal{C}_k m)$ time and $O(n)$ space algorithm, where $\mathcal{C}_k$ is the number of distinct contexts for k; $\mathcal{C}_k \le n$, often significantly smaller. This is nearly linear in n, independent of the context length k whereas the previous algorithm `DUDE` takes $\Omega(nk/\log_m n)$ time.
- We extend the algorithm above to support query `DUDE-DENOISE`(i, j, k) in time $O(|j - i + 1| \log\log n + \mathcal{C}_k m)$ where $\mathcal{C}_k$ is the number of distinct contexts for k in $z^n[i, j]$, for any $1 \le i < j \le n$.
- We present for any context size k, measures for characterizing the denoising error using only the noisy observations. We propose a denoiser performance estimator

in standard RAM model, the running time can be seen to have the $O(nk/\log_m n)$ term for arbitrary context k because operating on $2k + 1$ long strings for state names and pointers needs as many RAM words.

having only $O(n)$ complexity. As we show by an experimental study with synthetic and real data sets, we can derive an accurate performance estimator from it. This can be combined with $\texttt{DUDE-DENOISE}(i, j, k)$ to solve the $\texttt{DUDE-CONTEXT}(i, j)$.

All our results are fairly simple algorithmically. They use natural string matching techniques together with simple data structures and repeated radix sorting. They are highly efficient to implement.

In Section 2, we will present preliminaries and review DUDE [13] as well as the cumulative loss function error estimator proposed in [10]. In Section 3, we will present our denoising algorithms and in Section 4, we will present our denoising error estimator and performance results. Concluding remarks and open problems are in Section 5.

2 Preliminaries and background

2.1 Notation

For simplicity we will define the denoising problem on the same input and output alphabet $\mathcal{S}$, with the alphabet size $m \stackrel{def}{=} |\mathcal{S}|$. Therefore the noise-free sequence $(x_1, \ldots, x_n)$, and the observed noisy sequence $(z_1, \ldots, z_n)$ belong to the alphabet $\mathcal{S}$. We will denote the n-length sequence $(z_1, \ldots, z_n)$ by z^n and also denote a contiguous sub-sequence $(z_p, \ldots, z_q)$ by $z[p, q]$. We denote the probability transition matrix of the discrete memoryless channel (DMC) corrupting x^n by the $m \times m$ matrix $\mathbf{Q}$, where the $\mathbf{Q}(p, q)$ is the probability of observing the output z to be the p^{th} alphabet when the input x was the q^{th} alphabet in $\mathcal{S}$. We want to design a denoiser as defined in [13] which is a mapping $\hat{X}^n : \mathcal{S}^n \to \mathcal{S}^n$ from the observed noisy sequence z^n to an estimate of the noise-free sequence. The denoised sequence is denoted by $(\hat{x}_1, \ldots, \hat{x}_n)$ which as defined above, will be a function of the observed noisy sequence z^n. We define a cumulative loss function between the denoised sequence and the underlying noise-free sequence as [13]

$$L(x^n, \hat{x}^n) = \frac{1}{n} \sum_{i=1}^{n} \Lambda(x_i, \hat{x}_i), \tag{1}$$

where in the loss matrix Λ , $\Lambda(p, q)$ denotes the loss incurred by estimating the p^{th} input symbol as the q^{th} denoised symbol in the alphabet $\mathcal{S}$. For many cases we use the Hamming loss matrix, which uses $\Lambda(p, q) = 1 - \delta_{p,q}$, where $\delta_{p,q}$ is the Kronecker delta function.

2.2 DUDE algorithm

In [13], a discrete *universal* denoising scheme was introduced. The scheme took two passes over the noisy observation. The first pass collects the empirical probability distribution,

$$\mathbf{m}(z^n, b^k, c^k)[\beta] = \left|\{i : k + 1 \leq i \leq n - k, z_{i-k}^{i+k} = b^k \beta c^k\}\right| \tag{2}$$

for a single letter in z^n occuring within a double-sided context (b^k, c^k) of length $2k$ symbols. Given the known probability transition matrix of the DMC, this empirical probability

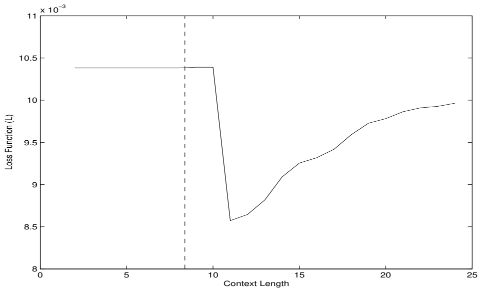


Figure 1: Best context length is larger than $\frac{1}{2}\log_m(n)$ (given by the vertical dashed line).

is used to compute an estimate of the posterior probability of the input symbol given the noisy observations within a double sided window (context). The denoising occurs in the second pass where the noisy observation is parsed sequentially and replaced by the denoised sequence $\hat{x}^n$ as,

$$\hat{x}_i = \arg\min_{\hat{x}\in\mathcal{S}} \mathbf{m}^T(z^n, z_{i-k}^{i-1}, z_{i+1}^{i+k})\mathbf{Q}^{-T}\left[\lambda_{\hat{x}} \odot \mathbf{q}_{z_i}^T\right] \tag{3}$$

where $\odot$ component-wise multiplication of vectors, $\mathbf{q}_{z_i}$ is the z_ith row of the channel matrix $\mathbf{Q}$, and $\lambda_{\hat{x}}$ is the $\hat{x}$th column of Λ. This scheme is proved to asymptotically have the same performance as the best finite-order sliding window denoiser tuned to the clean sequence and the observed noisy sequence. The main component of the DUDE algorithm [13] is to collect the empirical counts as in (2). The first observation we make is that for some sequences (in particular sequences of higher order Markov processes) it might be of advantage to chose a larger context length as shown in Figure 1.

2.3 Loss function estimation

Since we do not have access to the underlying noise-free sequence, to evaluate the denoiser performance, we need to be able to estimate the loss function defined in (1) using only the observed noisy sequence. Such a loss function estimator $\hat{L}_3(x^n, \hat{x}^n)$ was proposed in [10] and recommended the approximation

$$\hat{L}_3(x^n, \hat{x}^n) = \sum_{s_l, s_r}\sum_{\beta\in\mathcal{A}} \mathbf{m}(z^n, \beta, s_l, s_r)\sum_{x\in\mathcal{A}}\mathbf{Q}^{-1}(\beta, x)\sum_{z\in\mathcal{A}}\Lambda(x, g^{z\backslash\beta})\mathbf{Q}(z, x) \tag{4}$$

where the notation is taken from [10]. In order to evaluate (4) we would need $O(nm^5)$ computation since each change in the count results in re-estimating the denoised value.

3 Denoising algorithms using string matching

We first consider supporting $\texttt{DUDE-DENOISE}(1, n, k)$ and let $z[1\ldots n]$ be the input sequence (also referred to as the string). We will assume the alphabet has been renumbered $1..m$, $m \leq n$.

Our overall algorithm is as follows. We will compute a *name* $F(i,-k)$ for $z[i-k,i-1]$; this will have the property that $F(i,-k)=F(j,-k)$ iff $z[i-k,i-1]=z[j-k,j-1]$. Similarly, we will compute a name $F(i,+k)$ for the substring of length k to the right of i. All names will be small integers $1..O(n^c)$, for some small constant c, often 1. Then, we will generate a tuple $(F(i,-k),F(i,+k),z[i])$ using radix sorting. As we go down the list, we can generate for each distinct $(F(i,-k),F(i,+k))$, the count of different symbols in-between, thereby reducing the list to size $\mathcal{C}_k$ for subsequent steps.

Radix sorting takes time $O(n)$ and constant number of passes over the input. We will first describe the naming step in more detail and then the denoising step from the empirical distribution. We will present three solutions for naming.

Solution A. (Randomized algorithm) This approach is quite simple. We use a fingerprint function [7] on substrings of length $2k+1$ of string z^n which has the probability that given the fingerprint $f(z[i-k,i-1])$ of substring $z[i-k,i-1]$, $f(z[i-k,i-1])=f(z[j-k,j-1])$ if $z[i-k,i-1]=z[i-k,i-1]$ and with probability at most $1-1/n^c$, $f(z[i-k,i-1])=f(z[j-k,j-1])$ if $z[i-k,i-1]\neq z[i-k,i-1]$, for any constant c specified by the user. These fingerprints can be computed in $O(n)$ time in one pass and buffer space $O(k)$ as shown in [7]. This gives a linear time algorithm using buffer space $O(k)$ of words each in the range $O(\log n)$. The algorithm succeeds with high probability because of the low probability of error in the fingerprint function.

Solution B. (Deterministic naming). This will follow the ideas in [6]. Without loss of generality we will focus on giving names to string $z[i+1,i+k]$. Our algorithm proceeds in rounds. In jth round, we will have names N_j for the strings $z[i+1,i+2^j]$ for $j=0,\ldots,l$ where l is the largest power of 2 that is no larger than k. All names will be in the range $[1,n]$. Trivially, we have names for $j=0$ from our assumption. Now, assume we have names at the end of round $j-1$ and we will show how to assign names to appropriately longer substrings in round j. We generate tuples $t_i=(N_{j-1}(z[i+1,i+2^{j-1}),N_{j-1}(z[i+2^{j-1}+1, i+2^{j-1}+2^{j-1}]))$ for each i. We sort them by radix sorting and assign the location in this sorted order of earliest tuple that is the same as t_i as $N_j(z[i+1,i+2^j])$. Since we have no more than n tuples in each round, the names will continue to be in the range $[1,n]$. Finally, when k is not a multiple of 2, we need an additional round where we think of $z[i+1,i+k]$ as $z[i+1,i+2^l],z[i+k-2^l,i+k])$ and do another naming by radix sorting. This algorithm takes $O(n\log k)$ time, $O(\log k)$ rounds and $O(n)$ memory of words of size $O(\log n)$.

Solution C. We will present our most efficient, deterministic algorithm.

We will focus on giving names to substrings $z[i+1,i+k]$ first. We will construct the suffix tree T of string z^n. A number of algorithms exist for suffix tree construction. The earliest result is [3] and a simple one is [5]; several implementations exist, *e.g.,* [12]. This takes $O(n)$ time deterministically for alphabet $1..n^c$ for any constant c.

Recall that the suffix tree is a rooted trie with each edge labeled by some substring $z[i,j]$. Define the *string depth* of node v denoted $\sigma(v)$ to be the sum of lengths of substrings labeling the unique path from the root to v.

The algorithm is a depth-first search of T where we do not explore any node v with $\sigma(v)\geq k$ once it is encountered. The tree we get is the truncated version t of T such that each leaf l_t has $\sigma(l_t)\geq k$. We will name each leaf in t from $1...n$ uniquely. Further, each suffix $z[i,n]$ is a leaf in T that descends from precisely one of the leaves $l_t(i)$ in t.

We will assign the name of $l_t(i)$ as the name for the string $z[i, i+k-1]$. It is easy to see that this naming has all the desirable properties. In order to assign names for strings $z[i-k, i-1]$, we do the same procedure as above but reverse the string z^n at the beginning. This algorithm takes $O(n)$ time and space, and makes $O(\log n)$ rounds (in the suffix tree construction part).[3]

Now we turn to the denoising part. For each distinct context $(F(i,-k), F(i,+k))$ from the list of size $\mathcal{C}_k$, we consider the frequencies of various alphabet symbol $\alpha \in \mathcal{S}$ found in-between the context $(F(i,-k), F(i,+k))$ throughout the string z^n and denote it $f(F(i,-k), F(i,+k), \alpha)$. Now we perform a vector multiplication given by equation (3) and determine the most likely symbol to replace all those occurrences of α. All functions $f(\cdot)$'s can be computed in $O(n)$ time and the vector multiplication takes $O(m)$ time per unique $f(\cdot)$, so $\mathcal{C}_k m$ time in the worst case in all. Summarizing our discussion so far,

Theorem 2 *A given string z^n and probability transition matrix* $\mathbf{Q}$*, denoising by* `DUDE` *algorithm can be implemented in time $O(n + \mathcal{C}_k m)$ deterministically using suffix trees or with high probability by randomized Karp-Rabin fingerprints or in $O(n \log k + \mathcal{C}_k m)$ time by deterministic naming.*

Finally, we study `DUDE-DENOISE`(i, j, k). The algorithm for this problem will follow the outline above, but now our challenge is that given the suffix tree T of z^n built during the preprocessing, we need to name *only* the positions in $[i, j]$ for the query context k. We could ignore the suffix tree altogether and read in $z[i-k, j+k]$ and run either the randomized or deterministic naming methods, but we will propose a more efficient way using the suffix tree. The algorithm considers each position ℓ, $i \leq \ell \leq j$, in turn. For each ℓ, we need to determine the leaf $l_t(\ell)$ in the truncated tree t. We do not have the time to perform a depth-first-search of the suffix tree, instead, we will obtain $l_t(\ell)$ directly by posing a question about `WA`$(l_T(\ell), k)$ where $l_T(\ell)$ is the leaf in T that corresponds to $z[\ell, n]$ and `WA`(v, l) for leaf v and integer l, return the deepest node in T with string depth at most l that is an ancestor of v. Here, `WA` stands for the weighted ancestor for the obvious reason that we seek the ancestor of v with weighted depth close to l. It is clear that `WA`$(l_T(\ell), k) = l_t(\ell)$. Thereafter the algorithm is identical to above, but we only work on the induced tree t obtained by `WA`$(l_T(\ell), k)$ for ℓ, $i \leq \ell \leq j$. The bottleneck is the procedure for computing `WA`$(l_T(\ell), k)$ which currently takes $O(\log \log n)$ time per ℓ using [4]. Summarizing,

Theorem 3 *A given string z^n and probability transition matrix* $\mathbf{Q}$ *which can be preprocessed, a query* `DUDE-DENOISE`(i, j, k) *can be answered in time $O(|j-i+1| \log \log n + \mathcal{C}_k m)$ deterministically using suffix trees where $\mathcal{C}_k$ is the number of distinct contexts for k in $[i, j]$.*

[3] This is philosophically interesting since suffix tree does potentially much more, *i.e.,* sort all the suffixes of a string, and using it to only name the substring, *i.e.,* partition them into equivalence classes based on equality, may seem to be an overkill, but we do not yet know of an $O(n)$ time deterministic algorithm for our problem without using the power to "sort" the suffixes.

4 Algorithm for $\texttt{DUDE-CONTEXT}(1, n)$

First we will present estimators for denoiser performance for any given context k in Section 4.1. We present how such estimators can be used to find the optimal context k necessary for $\texttt{DUDE-CONTEXT}(1, n)$ in Section 4.2. Finally in Section 4.3 we present experimental results on $\texttt{DUDE-CONTEXT}(1, n)$.

4.1 Loss function estimators

We propose an alternate measure to the ones proposed in (4) and [13] that tries to capture the performance of the denoiser given only the noisy observation.

Note that the measure (which was observed in [13])

$$\hat{L}_1(x^n, \hat{x}^n) = \frac{1}{n} \sum_{u[-k,k] \in \mathcal{S}^{2k+1}} \lambda^T_{f(u[-k,k])} \mathbf{q}^T_{u[0]} \odot \mathbf{Q}^{-1} \mathbf{m}(z^n, u[-k,-1], u[1,k]), \tag{5}$$

can be very efficiently computed using $O(m^2 \mathcal{C}_k)$ operations computed using the observed noisy sequence. The drawback of this estimator is that it is accurate only for $km^{2k} = o(n/\log n)$, which makes it unattractive for $\texttt{DUDE-CONTEXT}(1, n)$.

In order to remedy this we propose a measure which is based on the question of how well the denoised sequence "explains" the observed noisy sequence. Therefore, we estimate the empirical channel transition probability matrix $\hat{\mathbf{Q}}$ assuming the denoised sequence $\hat{x}^n$ as the input and the observed noisy sequence z^n as the output. If the denoiser is performing well, *i.e.,* is close to the noise-free input sequence x^n. the estimate $\hat{\mathbf{Q}}$ should be "close" to the known transition probability matrix $\mathbf{Q}$. Thus, an appropriate function[4] f of $(\mathbf{Q} - \hat{\mathbf{Q}})$ should give a measure of how well the denoiser is doing. Therefore we propose to use the measure

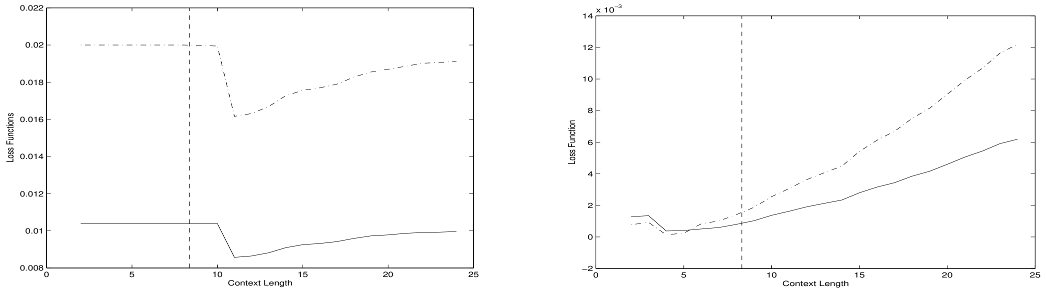
$$\hat{L}_2(x^n, \hat{x}^n) = f(\mathbf{Q} - \hat{\mathbf{Q}}), \tag{6}$$

to determine the performance of the denoiser using the observed noisy sequence alone. We used several matrix norms such as the l_1 and Frobenius norms in order to evaluate this measure. Note that this measure only captures the behavior of the cumulative loss function but does not predict the loss function accurately. Nevertheless, this estimator can be used to predict the context size k for which the denoising error is minimized, as we show in the numerical experiments in Section 4.2 even for large context sizes. The advantage of this scheme is that it can be efficiently computed using $O(n)$ computation.

4.2 $\texttt{DUDE-CONTEXT}(1, n)$ algorithm

Given a maximal context size K_{max}, the algorithm $\texttt{DUDE-CONTEXT}(1, n)$ does the following: i) For each context length $k = 1, \ldots, K_{max}$ run the $\texttt{DUDE-DENOISE}(1, n, k)$ denoiser

[4] Consider a Bernoulli(θ) source corrupted by a BSC with crossover probability δ ($\theta, \delta < 1/2$) and Hamming distance as the loss function. If $\delta \leq \theta$, $\hat{x}^n = z^n$, $L(x^n, \hat{x}^n) \stackrel{n\to\infty}{\longrightarrow} \delta$ and $\mathbf{Q} - \hat{\mathbf{Q}} \approx \delta \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$. Similarly if $\delta > \theta$, $\hat{x}^n = 0^n$, $L(x^n, \hat{x}^n) \stackrel{n\to\infty}{\longrightarrow} \theta$ and $\mathbf{Q} - \hat{\mathbf{Q}} \approx \begin{bmatrix} \theta(1-2\delta) & \delta \\ -\theta(1-2\delta) & (1-\delta) \end{bmatrix}$.



(a) $k_{opt} > \frac{1}{2}\log_m(n)$

(b) $k_{opt} < \frac{1}{2}\log_m(n)$

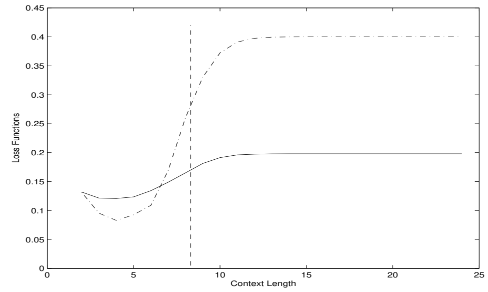
Figure 2: Comparison of behavior of actual loss function L and estimated measure $\hat{L}_2$. The dashed curve is the measure of the denoiser performance $\hat{L}_2$ whereas the solid curve is the true cumulative loss function. The vertical line indicates $k = \frac{1}{2}\log_m(n)$.

algorithm. ii) Evaluate the denoiser performance measure $P_k = \hat{L}_2$ for the given context length. iii) Pick the context length with the smallest measure P_k.

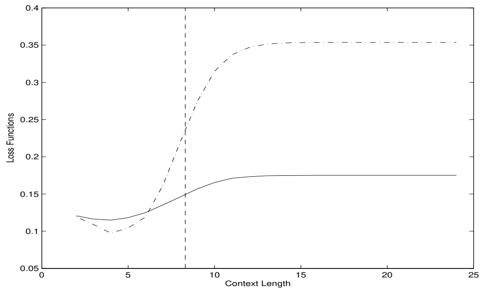
In order for this algorithm to perform well, we need to show that the measure $\hat{L}_2$ given in (6) accurately captures the behavior of the loss function. Note that this measure does *not* give the actual cumulative loss function but just the behavior as illustrated in Figure 2. We compare its behavior with the actual loss function computed using the noise-free input sequence and find that both when the best context size k is smaller and larger than $\frac{1}{2}\log_m(n)$ the trends were well captured by this measure. This behavior was fairly consistently observed across a wide variety of data sets used. In particular, in Figure 2(a) we generated a binary Markov source of order 11 and of length $n = 10^5$. The context size k for which the denoising error is minimum is larger than $\frac{1}{2}\log_m(n)$. Similar behavior was observed in other higher order Markov sources generated as binary auto-regressive processes as well. However, for lower order Markov processes or processes with shorter memory, as one would expect, short context sizes k were sufficient to get good denoising performance as illustrated in Figure 2(b). The estimate of the denoiser performance using $\hat{L}_2$ followed the behavior of the actual cumulative loss function as seen Figure 2.

4.3 Performance results

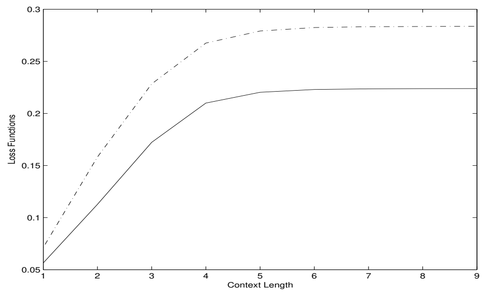
In this section the performance of DUDE−DENOISE(i, j, k) and DUDE−CONTEXT (i, j) implemented using the string matching algorithms described in Section 3 is evaluated over several data sets. We use two types of data sets: i) Synthetic data sets generated using Markov processes of given order. ii) Text data set obtained from Project Guttenberg by concatenating the works of Leo Tolstoy. Figure 3(a) plots the performance of running DUDE−CONTEXT$(1, n)$ on synthetic data generated by a first order Markov source corrupted a symmetric DMC. The same source was used in another scenario in Figure 3(b) where the channel is an asymmetric binary channel. Using DUDE−CONTEXT$(1, n)$ and the loss estimator $\hat{L}_2$, we can pick out the optimal context size in both these cases. In Figures 3(c)-(d), the performance of DUDE−CONTEXT$(1, n)$ is demonstrated for text sources,



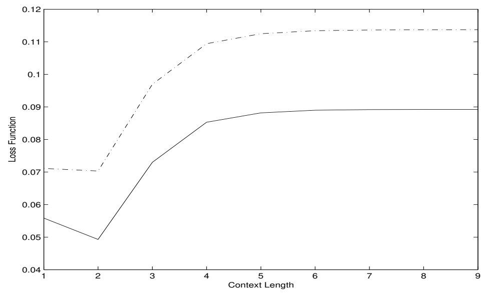
(a) Binary Markov source and BSC(0.2)



(b) Binary Markov source and asymmetric DMC.



(c) Text data with more noisy DMC



(d) Text data with less noisy DMC

Figure 3: Plots for data sets obtained from text and synthetic data for various DMC's. The dashed curve is the measure of the denoiser performance $\hat{L}_2$ whereas the solid curve is the true cumulative loss function. The vertical line indicates $k = \frac{1}{2}\log_m(n)$.

where the alphabet size is $m = 26$. For these large alphabet sizes, an efficient estimator of the loss function becomes important. In this case we saw a significant computational advantage of using $\hat{L}_2$ over previously proposed $\hat{L}_3$ [10], whereas the behavior of the denoiser performance prediction seemed to be close for datasets examined.

5 Concluding Remarks

There has been a rich history of interplay between string matching methods and compression [11, 2, 8], so our work in this paper adds to this lore. In particular, string matching methods in [2, 8] use the context like we do here, but do a different, elaborate search within many contexts for a particular position in order to perform compression.

Our work also leaves open a number of technical problems. Can we design a deterministic algorithm for the naming of substrings of length k in $O(n)$ time and $O(\log k)$ rounds? We suspect it is possible but one may need to go into the guts of [5] or [9] and use $2-3$ naming or $2-4$ naming respectively. A conceptual question is to develop an effective denoising method in presence of noise comprising inserts and deletes. Also, it will be of interest to extend `DUDE` to do block-based denoising for block sizes B. There is also the interesting technical problem of whether preprocessing the probabilistic transition matrix can solve the repeated vector multiplications more efficiently than the complexity bound we have quoted. Also it might be important to develop denoising algorithms when the alphabet size m is large, perhaps comparable to the sequence length n. Finally, other novel efficiently computable cumulative loss function estimators with provable properties will be useful to choose the appropriate context length.

References

[1] G. Cormode and S. Muthukrishnan. Substring compression problems. To appear in *Proc. ACM SODA, 2005.*

[2] Z. Cohen, Y. Matias, S. Muthukrishnan, S. Sahinalp, J. Ziv. On the temporal HZY compression scheme. *ACM-SIAM Symp. Discrete Algorithms (SODA)*, 2000, 185-186.

[3] M. Farach-Colton. Optimal Suffix Tree Construction with Large Alphabets. *FOCS*, 1997, 137-143.

[4] M. Farach and S. Muthukrishnan. Perfect Hashing for Strings: Formalization and Algorithms. CPM 1996: 130-140.

[5] J. Karkkinen and P. Sanders. Simple linear work suffix array construction. *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP '03)*, 2003, LNCS 2719, Springer, pp. 943-955.

[6] R. Karp, R. Miller and K. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. *Proc. 4th Ann. ACM Symp. on Theory of Computing*, 1972, pp. 125–136.

[7] R. Karp and M. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 1987, 31(2):249-260.

[8] Y. Matias, S. Muthukrishnan, S. Sahinalp and J. Ziv. Augmenting Suffix Trees, with Applications. *European Symp. on Algorithms (ESA)*, 1998, 67-78.

[9] S. Muthukrishnan Simple Optimal Parallel Multiple Pattern Matching. *J. Algorithms*, 2000, 34(1): 1-13.

[10] E. Ordentlich , M. Weinberger and T. Weissman. Efficient pruning of bi-directional context trees with applications to universal denoising and compression, *IEEE Information Theory Workshop (ITW)*, San Antonio, Texas, Oct 24-29, 2004.

[11] M. Rodeh, V. R. Pratt, S. Even. Linear Algorithm for Data Compression via String Matching. J. ACM 28(1): 16-24 (1981).

[12] S. Skiena. Code: http://www.cs.sunysb.edu/ algorith/files/suffix-trees.shtml; http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK3/NODE131.HTM.

[13] T. Weissman, E. Ordentlich, G. Seroussi , S. Verdu and M. Weinberger. Universal Discrete Denoising: Known Channel, IEEE Transactions on Information Theory, to appear. Most recent version available at http://www.stanford.edu/∼tsachy/pdffiles/finalversion.pdf