Fast nGram-Based String Search Over Data Encoded Using Algebraic Signatures

Witold Litwin
Univ. Paris Dauphine
witold.litwin@dauphine.fr

Riad Mokadem
Univ. Paris Dauphine
riad.mokadem@dauphine.fr

Philippe Rigaux
Univ. Paris Dauphine
philippe.rigaux@dauphine.fr

Thomas Schwartz
Univ. Santa Clara
tjschwarz@scu.edu

ABSTRACT

We propose a novel string search algorithm for data stored once and read many times. We encode the data records coming for storage into their cumulative algebraic signatures. We define two variants of our algorithm that we analyse theoretically and experimentally. The experiments compares the speed of our algorithm to the Boyer-Moore scheme, usually the fastest method known. Our search was up to a seventy times faster for DNA data, up to eleven times faster for ASCII, and up to a six times faster for XML documents.

Our method applies to databases in general and to scalable distributed data structures, P2P and grid systems used for the Database as Service (DAS) context especially. The client sends the data for storage at the (remote) servers encoded. The servers match the stored data for the pattern requested by the client without decoding. No local user can then involuntarily discover the content of the stored data.

1. INTRODUCTION

We describe n-gram search that is novel string (pattern) matching principle first proposed in preliminary form in [9]. We intend our method for databases and files where records are stored once and searched many times. An application record is encoded in a record of same size as a *Cumulative Algebraic Signature* (CAS). We recall that an Algebraic Signature (AS) of a string is an element of a Galois Field (GF), typically $GF(2^8)$ in what follows [12]. When encoding, we replace each symbol s with the AS of the prefix ending with s. If we use the *full* CAS, then the prefix starts with the first symbol in the record. If we use the *partial* CAS, then the prefix is n symbols long. Converting a record to or from one of the CAS forms is an operation with linear complexity and without storage overhead.

The search algorithm works as follows. We pre-process the pattern to match, calculating the AS of every n-gram in the pattern. Next, we attempt to match the pattern within the record. Each attempt first compares the AS of the last n-gram in the pattern with the AS of some n-gram in the record. The successful match continues with one or more attempts to find out whether the entire pattern matches. If an attempt fails, we shift the pattern, similarly but not identical to the Boyer-Moore (BM) algorithm. We recall that BM is probably the most used pattern matching scheme and usually the fastest. In tendency, our shifts are longer, making our search more sub-linear. The reason is that the n-gram signatures are typically more discriminative than single symbols. Especially, when the size of the alphabet used is smaller than the size of GF used (DNA records) or when only a subset of characters is frequently used. By prudently selecting our parameter n, we can bring the average shift size close to the minimum of the pattern length and the GF size, i.e., to a shift by almost 256 symbols for longer patterns.

Our method presents two advantages. First, it is fast. In particular, the experiments show it is typically several times faster than BM. The ratio is larger for smaller alphabets. More precisely, our search speed appears up to seventy times faster for DNA alphabet. It is also up to eleven times faster for ASCII, and, finally, up to a six times faster for XML documents.

The second advantage lies in the use of distributed storage. A client locally encodes the record and sends it to a remote server. For the search, the client sends an encoding of the pattern to all the servers in parallel. The servers search for the pattern without decoding. No involuntary or accidental data disclosure on the server or on the way to the client is possible. A determined adversary with access to the server can decode the stored data, but if caught, cannot credibly claim unintentional possession. This feature makes the method suitable for Scalable Distributed Data Structures (SDDS) over a grid or a structured P2P system. More generally it is suitable for a Database As Service (DAS) environment.

We present two variants of the algorithm. The first determines n-gram signatures dynamically from a full CAS. The second encodes the n-grams into the partial CAS directly. The search with the latter method is faster. However, the former approach makes up for a longer search time, by storing the records in a form amenable to other fast searches such as prefix searches and longest common substring searches [11].

Below, Section 2 recalls the basics of our method. We

VLDB '07, September 23-28, 2007, Vienna, Austria.

present the two variants of the algorithm in Sections 3 and 4, respectively. Section 5 contains analytical and experimental performance analysis, including comparison to Boyer-Moore. For experimental analysis we use DNA records, ASCII text and XML documents. We then discuss possible improvements (Section 6), present related work (Section 7) and conclude.

2. PRELIMINARIES

We briefly recall here the basic features of algebraic signatures. More details can be found in [LR04].

2.1 Algebraic signatures

Let G be $GF(2^f)$, the Galois Field with 2^f elements. We call them *symbols* and identify them as bytes ($f = 8$) or words ($f = 16$ or 32, etc). Let α be a primitive element in G. The algebraic signature (AS) of any record $r_1 \cdots r_i$ is given by $r_1\alpha \oplus \cdots \oplus r_i\alpha^i$. (This is the one component signature sig_α in [12]). The addition here is the GF addition and implemented as the familiar XOR. The implementation of the multiplication is more involved as we show later in this section. If the AS of two records of the same length differ, then we know for sure that the records are different, whereas if they are the same, (and if the records are not random) then we conclude probabilistically that they are the same.

2.2 Cumulative Algebraic Signature

Let R_M be a record of M symbols $r_1 \cdots r_M$. Consider for each r_i the AS of the prefix ending in r_i, i.e., the AS $r'_i = r_1\alpha \oplus \cdots \oplus r_i\alpha^i$. The record R'_M with symbols $r'_1 \cdots r'_M$ is the *full Cumulative Algebraic Signature* (CAS) of R_M. The full CAS replaces each individual symbol r in the encoded string with another symbol r' encoding not only the current symbol but also additional knowledge of all the symbols preceding r. A comparison of original symbols at same offset in two strings only allows to conclude about the equality or difference of these symbols. One needs additional processing to conclude anything about the matching of the symbols preceding the visited ones, hence of the strings themselves. The comparison of the ASs in a CAS yields in contrast the information about likely equality or for sure inequality of the entire prefixes ending with the matched symbols. The result is naturally much more discriminative for a pattern search. This is the rationale for our approach.

The ASs have two basic properties that we exploit and discuss now, as well as some others we introduce progressively. First, we have:

$$r'_i = r'_{i-1} \oplus r_i\alpha^i \tag{1}$$

We may thus calculate the CAS in linear time. Reversely, we have

$$r_i = (r'_i \ominus r'_{i-1})/\alpha^i \tag{2}$$

The decoding of CAS into the original record can be thus in linear time as well. Note, that in a Galois field $GF(2^f)$, addition is the same as subtraction. We again postpone the discussion of multiplication, as well as of the division by the powers of α.

2.3 Partial CAS and n-grams

An n-gram within R_M is any substring of length n, i.e., $r_{i-n+1}r_{i+n-2}\cdots r_i$, where $i \in \{n, \cdots, M\}$. A *partial CAS* of R_M consists of the record where each symbol, let it be r''_i, is either the AS over the n-gram terminating with r_i, or over the i-gram terminating with r_i for $i < n$, i.e., at the beginning of R_M. Formally, we have:

$$r''_i = \begin{cases} r_1\alpha \oplus \cdots \oplus r_i\alpha^i & for\, i < n \\ r_{i-n+1}\alpha \oplus \cdots \oplus r_i\alpha^n & otherwise \end{cases} \tag{3}$$

The following algebraic properties of our signatures are further useful for our goal. They determine the AS of any n-gram in R_M from a full CAS. They thus also define the relationship between the partial and full CAS of R_M, i.e., R'_M and R''_M contents. First, for any $i \geq n$, we have :

$$r'_i \ominus r'_{i-n} = r_{i-n+1}\alpha^{i-n+1} \oplus \cdots \oplus r_i\alpha^i$$

Therefore, the searched n-gram signature is:

$$AS(r_{i-n+1}, r_{i-n+2}, ..., r_i) = (r'_i \ominus r'_{i-n})/\alpha^{i-n} \tag{4}$$

2.4 Calculating the GF Multiplication and Division

There are several methods for multiplying and dividing in a GF. In our context, the use of logarithm and antilogarithm tables appeared to be most efficient [10]. The tables precalculate the log and antilog values. The logarithm of a GF element $x \neq 0$ is the (unique) integer i, $0 \leq i \leq 2^{f-2}$, such that $\alpha^i = x$. We define the logarithm of 0 to be 2^{f-1}. The calculus of (4), in our implementation, is then:

$$AS(r_{i-n+1}, r_{i-n+2}, \cdots, r_i) = antilog_\alpha[log_\alpha[r'_i \oplus r'_{i-n}] - i + n \bmod (2^f - 1)] \tag{5}$$

Here, the operator $\oplus$ denotes XORing, i.e., a GF addition and substraction. Other additions/subtractions are the usual ones. Similarly, we move between the original record and its full CAS through the calculus:

$$r'_i = r'_{i-1} \oplus antilog_\alpha[(log_\alpha[r_i] + i) \bmod (2^f - 1)] \tag{6}$$

$$r_i = antilog_\alpha[log_\alpha[r'_i \oplus r'_{i-1}] - i \bmod (2^f - 1)] \tag{7}$$

3. PATTERN MATCHING IN FULL CAS

We present now the first variant of our algorithm. We assume the records encoded in their CAS forms, using property (1, when coming for storage. As we said, we presume a typical database behavior where a record is stored, hence encoded, once, and searched many times. Our search starts with the pattern pre-processing phase, followed by the actual matching calculus through the records. We only discuss the processing within a record, discussion of inter-record navigation is beyond the scope of this paper.

3.1 Pattern Pre-processing

We use n for the number of symbols in an n-gram. Typically $n \in \{1, 2, 3, 4\}$, Let $P = (p_1, p_2, \cdots, p_K)$ be the pattern to match. We only search for patterns of length $K \geq n$. We start with the encoding of P into its CAS P'. In the distributed case, this is done at the client. We hash every n-gram in the pattern, starting from the beginning, into a value h that is an entry in an auxiliary table $T[0, .., 2^f]$ called *shift table*. We use T to encode the shifts of the search pattern against the records. The h value is the logarithm log_α of the n-gram's AS (we call it *Logarithmic AS*, and note LAS). Equation (5) now becomes:

2-gram	Shift
da	6
au	5
up	4
ph	3
hi	2
in	1
ne	0
All other digrams	7

Table 1: The shifts for each 2-grams in Dauphine

$$\begin{aligned} h &= LAS(p_{i-n+1}, p_{i-n+2}, \cdots, p_i) \\ &= log_\alpha[p'_i \oplus p'_{i-n}] - i + n \, mod \, (2^f - 1) \end{aligned} \quad (8)$$

We store the LAS of the final n-gram in a variable V, and the LAS of the full pattern in a variable W. We then set each $T[i]$ according to the following rules. First we preset every entry to $T[i] = K - n + 1$. For every LAS h for an n-gram in P other than the last one, we set $T[h]$ to the offset in P with respect to K, of the rightmost n-gram with h as LAS. If $h = V$ that is thus the LAS of last n-gram, we set $T[h]$ to the offset of the previous occurrence in P of an n-gram with $h = V$ provided there is such occurrence. Notice that this structure of T differs from that in [L+06], turning out to be faster. Here is the preprocessing algorithm.

Algorithm PREPARESEARCH
Input: a pattern P, the ngram size n
Output: the encoded pattern P', the shift table T
begin
  // First encode the pattern
  for ($i := 1$ to $size(P)$)
    // Apply equation (1)
    if ($i = 1$)
      $P'[i] := \alpha P[1]$
    else
      $P'[i] := P'[i-1]\text{XOR}\alpha^i P[i]$
    endif
  endfor
  // Compute table T. First initialize with the maximal shift
  for ($i = 0$ to $size(GF)$)
    $T[i] := size(P) - n + 1$
  endfor
  // For each ngram, add en entry [log(ngram), shift] in T
  for ($i = n$ to $size(P)$)
    $T[LAS(P[i-n+1] \cdots P[i])] := size(P) - n - i$
  endfor
end

EXAMPLE 1. *Consider the pattern P = `Dauphine`. We choose $n = 2$, i.e., we intend to perform the 2-gram (digram) based search. We initialize every $T[i]$ to $K - n + 1 = 7$. We then set $T[LAS(\texttt{da})] = 6$, $T[LAS(\texttt{au})] = 5$, etc. Table 1 below illustrates the result, (thought it does not show the actual i for each shift).*

□

3.2 Pattern Search Processing

We now describe the search for $P = (p_1, p_2, \cdots, p_K)$ within an encoded record $R = (r_1, r_2, \cdots, r_M)$ of length M, or a non-key field of size M of a record, e.g., in an SDDS. Let $R_i^n = (r_{i-n+1}, r_{i-n+2}, \cdots, r_i)$ denote the n-gram in R ending with r_i. Similarly, we use $P_i^n = (p_{i-n+1}, p_{i-n+2}, \cdots, p_i)$ to denote the n-gram in P ending with p_i.

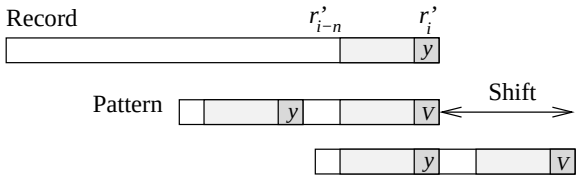

Figure 1: The n-gram shift, with $y = LAS(r_{i-n+1}, \cdots, r_i)$ and $V \neq y$

We begin by attempting to match R_K^n and P_K^n. We do this by comparing $LAS(R_K^n)$ computed according to (8) applied to the symbols in the record, with $LAS(P_K^n)$ that is in V.

1. If there is the match, then we match $LAS(R_K^K)$ and $LAS(P_K^K)$ that is in W. If again we have the match, then we report a likely successful search.
2. If $LAS(R_K^n) \neq LAS(P_K^n)$, then we lookup table T with index $i = LAS(R_K^n)$. We then shift P by $j = T[i]$ positions to the right. We follow with the attempt to match R_{K+j}^n and P_K^n. We repeat the whole process until the shift reaches or attempts to exceed r_M.

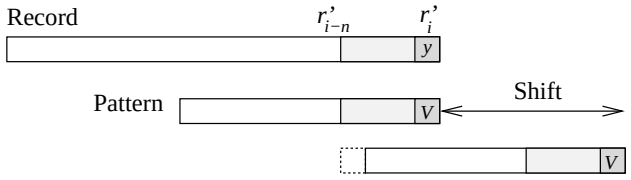

Figure 2: The n-gram shift, when $y = LAS(r_{i-n+1}, \cdots, r_i)$ is not found in the pattern

Figures 1 and 2 illustrate a matching attempt at position i. Variable V stores the value of the final n-gram LAS in P. The encoded record is examined at position i for the values of the CAS r'_{i-n} an r'_i. From Equation (5) we obtain the LAS of the n-gram of R at i: $y = LAS(r_{i-n+1}, \cdots, r_i)$. Now assume that $V \neq y$. Then either y is found in the pattern by looking up table T, and the shift superposes the position of y in the pattern with the current position in the record (Figure 1), or y is not found in the pattern, in which case the shift found in T is $K - n + 1$ (Figure 2).

Since we match the signatures, there is typically, thought not always (see Section 6), small but non-zero probability of a collision, in the orders of $1/2^f$. That is, an n-gram or the whole P may not match, while the LAS does. For sure result, one has to check a likely match symbol by symbol. In an SDDS it should be typically at the client site, where every matching records are sent for the decoding, using property (2) above.

EXAMPLE 2. *Consider the search in a French text, for R = 'Universite de Technologie Paris Dauphine', and P = 'Dauphine', assuming the choice $n = 2$ (Figure 3). We pre-process P, as in Example 1 and Table 1. We set in particular $V = LAS(\texttt{ne})$, since 'ne' is the terminal digram. The processing phase starts with the attempt to match the digrams 'si' ending at the offset R_8, since $K = 8$ for our P, and of 'ne' (underlined at the figure). We calculate $i = LAS(\texttt{si})$, using (5), from the AS found within the symbols*

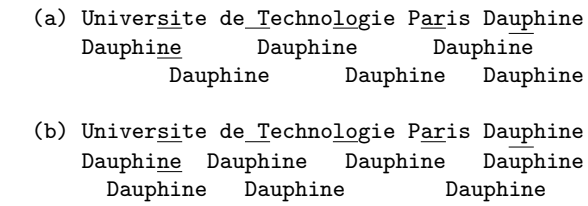

Figure 3: n-gram search in (encoded) French text using (a) $n = 2$ and (b) $n = 1$

of the CAS at offset 8 ('i' at the figure), and that at offset 6 ('r' there). We have $i \neq V$, assuming no collision (what we did not test for this example). We read $T[LAS('si')]$ and find 7 there.

We shift for the next attempt by 7 positions, i.e., to the digram ' T' ending a R_{15}. We calculate $LAS(' T')$. It does not match V, hence we access T at $i = LAS(' T')$. We find 7 again, (also assuming no collision with any digram in 'Dauphine'), etc. We underlined the visited digrams. As Figure 3 shows, the search using $n = 2$ needs 6 attempts and thus 5 shifts. We show the successive shifts one after the other in two lines under the pattern. The 5th attempt using 'up' finds $T[LAS('up')] = 4$ and thus moves the pattern into the correct position under the string. Our algorithm tests that the pattern and the sub-string have the same LAS. Since this test is successful, our algorithm terminates with success.

Our search with $n = 2$ took six attempts (Figure 3.a). Except for the last shift, all shifts are over $K - n + 1 = 7$ symbols (with $K = 8$ and $n = 2$ here). A search with $n = 1$ needs 7 attempts (Figure 3.b). Now two other shifts are smaller than the maximal shift of $K - n + 1 = 8$ and the average shift of 5.8 symbols is almost 20% smaller than for $n = 2$. Our search with $n = 1$ has the same shifts as the Boyer Moore algorithm [2]. □

In our example, using 3-grams would actually lead to an additional matching attempt than using digrams. This illustrates the influence of the choice of n on the behavior of the algorithm. As far as we know, our algorithm is unique in this regard. It appears that in general, larger n work better for smaller alphabets or if only few symbols are frequently used.

EXAMPLE 3. *This example illustrates this facet of our approach for a DNA sequence which is adapted by one in [4] (Figure 4). We have the four-letter alphabet of nucleotides: A, C, G, T. The pattern is 'AGACAGAT'. For such small alphabet, one can choose a representation that leads to zero collisions. This means that any 4-gram has a unique LAS. In our example, choosing $n = 1$ leads to twelve attempts and an average shift of less than 1.5 symbols. Choosing $n = 2$ reduces the search cost threefold to four attempts. Unlike for the text, the best choice is however $n = 3$. It leads to only three attempts with an average shift length of 5.6 symbols. It thus accelerates the search by factor of four. No larger n does better. Notice that Boyer-Moore would need again the same number of attempts as for $n = 1$. However, BM needs less processing for each matching attempt, and the processing should be theoretically faster by a factor of two. This still results in a performance advantage for our algorithm.* □

Figure 4: n-gram search in (encoded) DNA sequence for (a) $n = 3$, then (b) $n = 2$ and (c) $n = 1$.

4. PATTERN MATCHING IN PARTIAL CAS

The full CAS encoding allows for the dynamic choice of n. It is also known as particularly efficient for other useful searches such as prefix search, longest common prefix search, or longest common substring search [10]. The variant we describe now encodes the record into a partial CAS of preset size n. The encoded record contains directly the ASs of all the n-grams in the original record. The pattern matching speeds up, although the other searches above slow down.

4.1 Encoding and Decoding

We denote the i^{th} symbol of the encoded record with r''_i. We define r''_i as in Equation (3). As we said in Section 2, record $R''_M = r''_1 r''_2 \cdots r''_M$ is the *partial CAS* of record R_M. Encoding of R_M can again be computed in linear time. For $2 \leq i \leq n$, we may indeed recursively calculate

$$\begin{aligned} r''_i &= r''_{i-1} \oplus \alpha^i r_i \\ &= r''_{i-1} \oplus antilog[i + log[r_i] \, mod \, 2^f - 1] \end{aligned} \quad (9)$$

for $2 \leq i \leq n$. Otherwise, we observe that $r''_i \oplus \alpha r''_{i+1} = \alpha r_{i-n+1} \oplus \alpha^{n+1} r_{i+1}$. Therefore, our recursion becomes for $i > n$:

$$\begin{aligned} r''_{i+1} &= (\alpha r_{i-n+1} \oplus \alpha^{n+1} r_{i+1} + r''_i)/\alpha \\ &= r_{i-n+1} \oplus \alpha^n r_{i+1} + \alpha^{-1} r''_i \\ &= r_{i-n+1} \oplus antilog[n + log[r_{i+1}] \, mod \, 2^f - 1] \\ &\quad \oplus antilog[log[r''_i] - 1 \, mod \, 2^f - 1] \end{aligned} \quad (10)$$

Decoding a partial CAS is more involved than decoding a complete CAS. First, $r_1 = \alpha^{-1} r''_1$. For $1 \leq i \leq n-1$, $r_{i+1} = \alpha^{-i}(r''_{i+1} - r''_i)$. If $i \geq n$, then

$$\begin{aligned} r_{i+1} &= \alpha^{-n-1}(r''_i \oplus \alpha r''_{i+1} \oplus \alpha r_{i-n+1}) \\ &= \alpha^{-n-1} r''_i \oplus \alpha^{-n} r''_{i+1} \oplus \alpha^{-n} r_{i-n+1}) \\ &= antilog[log[r''_i] - n - 1 \, mod \, 2^f - 1] \\ &\quad \oplus antilog[log[r''_{i+1}] - n \, mod \, 2^f - 1] \\ &\quad \oplus antilog[log[r_{i-n+1}] - n \, mod \, 2^f - 1] \end{aligned} \quad (11)$$

Unlike for complete CAS, decoding a single symbol in the record involves decoding all previous ones.

4.2 Pattern Pre-processing and Processing

Pre-processing of the pattern P proceeds in the same manner as in the first variant. In an SDDS environment, P should typically arrive at the server encoded by the client into its partial CAS P''. The scan of P'' fills in table T. However, one hashes now each AS, not the LAS.

The pattern search processing is also similar with respect to the n-gram match attempts. There is no more however the AS calculus for the n-grams in the record, since they are directly encoded in R''. This speeds up the matching. If the matching attempt with V succeeds, we need to match, as before, the AS of the entire P towards that of the potentially matching string under the consideration in R. It would be cumbersome to calculate these AS from the n-grams in P'' and R''. Instead, we attempt therefore now to match the AS of the nearest n-gram to the left of the one currently successfully matched. If the match works, we move to the next adjacent n-gram in P'' till we reach the beginning of P''.

If any match attempt fails during this process, we calculate from T the shift of the terminal n-gram signature. If the successful matches reach the beginning of P'', the search is presumed likely successful or successful if the probability of the collision is known to be zero, as in Section 6. In the former case, the client may terminate with the systematic test, as before.

The examples from the previous section apply as is to the partial encoding. Observe on this basis, that since there is no more the n-gram LAS calculus, the attempts to match an individual n-gram is faster. The attempt to match the entire P is in contrast about K/n times slower, increasingly thus for a longer P. The end result depends thus on the actual data. More the n-grams are discriminative, faster the partial CASs should be. A detailed analysis of both variants obviously requires experiments, like those reported in Section 5.2.

5. PERFORMANCE ANALYSIS

We now analyze the behavior of our method. First, we study it analytically. As usual, the calculus complexity forces simplified modeling assumptions and limits the results. We overview the basic performance factors and guidelines for the choice of n. We study the average shift size, crucial for our performance factors, under the usual assumptions of the randomness of symbol values. Next, we complete the formal analysis with extensive experimental campaign. This one considers the data in our examples, i.e., ASCII text and DNA, but also the XML text, more and more popular. We compare the n-gram search to BM. The outcome shows the superiority of n-grams, especially for long patterns.

5.1 Analytical Study

The pre-processing of the pattern costs $O(2Kn + 1)$, involving the (linear) encoding and the creation of T. Likewise the encoding or decoding of a record costs $O(M)$. The pattern matching, i.e., the search cost after the pre-processing, is $O(N)$, where N is the number of attempts to match. The figure assumes most attempts unsuccessful, hence reaching the record's end. The N value depends mostly on the average shift (sized in the number of symbols), let it be A. We have $N = (M - K)/A$, where M is the record size, we recall. Despite the same $O(N)$ cost formulae, the search speed is in practice faster for the n-gram (partial) CAS, since we avoid the XOR calculus and log/antilog calculus. It seems about impossible to determine however analytically any of those. The experiments we report later show some values and the about simple to double difference in general.

The average shift is basically longer when the matching probability is smaller. Our examples illustrated that some choices of $n = 2$, 3, or 4 make n-grams quite selective, in the sense of the matching probability of the signatures close to (the minimal possible) of $1/2^f$ or $1/256$ for our favorite GF. The actual value depends also on the pattern size as we'll see soon in depth. The n value itself depends on the alphabet size, or the number of symbols actually most used. For DNA, we have 4 symbols. Let us consider that these symbols are equally likely to appear and not correlated. The corresponding probability p_1 is $1/4$. This would lead for a 1-gram (or BM) to the average shift of $A = 4$ symbols, whenever $K > 4$. For $n = 4$, the similar (signature) matching probability p_n for the n-gram is $p_n = (\frac{1}{4})^4 = 1/256$. This leads to $A = 256$ for a pattern long enough. For a shorter pattern, with $K < 256$, but $K >> 4$ still, it obviously leads to $A \approx K - n + 1$, where the latter is the maximal shift, i.e., to $A = K - 3$ in the occurrence.

Larger n does not improve p_n under these assumptions. While it decreases the maximal shift, hence A as well. The result, i.e. $n = 4$ being the optimal value basically holds even if the distribution of symbol values is somehow non-uniform. The signature calculation randomizes indeed the distribution. A larger n may however perhaps be useful for a highly skewed case.

Consider now the ASCII text example. Assume, little arbitrarily, that 17 values are by far the most typical. Now $n = 2$ appears best choice. It leads indeed to $p_n = (\frac{1}{17})^2 = 1/289$. Enough for GF used. The choice should increase the A value, with respect to $n = 1$ (and BM thus), already for K about 10 (as the experiments confirm). If less symbols are typical, e.g. 8 only, then n should be 3. Etc.

The above examples point to the possibility of zero-collision probability for the chosen n. The necessary condition is that the number of possible n-gram values is under the GF size. This can be the case or not, e.g., is for DNA, but not for ASCII above.

We now evaluate A more in depth, given is crucial importance for the search cost that appeared above. We come out with an approximate formula, under our modeling assumptions, i.e., of the set up of the best n, leading to the n-gram matching probability of $1/2^f$.

We note $p = 2^{-f}$ and $q = 1 - p$. First, we calculate the average shift B, assuming the shift of the pattern P by 1 symbol if we have an initial match between the rightmost n-gram in P and the n-gram in the record R (Figure 5.a and 5.b). In this case, we always have a shift of 1 if the rightmost n-gram matches to the one in R, which happens with probability p. We shift by 2, if the rightmost n-gram in P does not match, but the next one does. And so on. Finally, if the n-gram in R does not match any of the n-grams in P, then we shift by $K - n + 1$. This happens with probability q^{K-n}. As a result, we obtain

```
a) GGTCAAGTTTAGGCCCCCAGCGTAAAAAAGACGTAAGCCTA
   TAAAAAGTT

b) GGTCAAGTTTAGGCCCCCAGCGTAAAAAAGACGTAAGCCTA
    TAAAAAGTT

c) GGTCAAGTTTAGGCCCCCAGCGTAAAAAAGACGTAAGCCTA
        TAAAAAGTT
```

Figure 5: Shifting after first hit (a) either by shift size 1 (b) or by trying to match the field 'AGTT' in the search pattern, which leads to a maximal shift by 6.

$$B = 1 \cdot p \quad + \quad 2 \cdot p \cdot q + 3 \cdot p \cdot q^2 + \cdots \\ + \quad (K-n) \cdot p \cdot q^{K-n-1} + (K-n+1) \cdot q^{K-n}$$

Using a formula for the derivative of a finite geometric series (or Mathematica), we can simplify to

$$\begin{aligned} B &= (1-q) \cdot (1 + 2q + 3q^2 + \cdots + (K-n)q^{K-n-1}) \\ &\quad +(K-n+1) \cdot q^{K-n} \\ &= (1-q) \cdot \frac{(K-n)q^{K-n+1} - (K-n+1)q^{K-n} + 1}{(q-1)^2} \\ &\quad +(K-n+1) \cdot q^{K-n} \\ &= \frac{(K-n) \cdot q^{K-n+1} - (K-n+1) \cdot q^{K-n} + 1}{1-q} \\ &\quad + \frac{(K-n+1) \cdot q^{K-n} - (K-n+1) \cdot q^{K-n+1}}{1-q} \\ &= \frac{1-q^{K-n+1}}{1-q} \\ &= \frac{1-q^{K-n+1}}{p} \end{aligned}$$

In fact when the n-gram in R matches the rightmost n-gram in P, we do not shift by one. Instead, we shift by T value depending on whether the n-gram in R appears elsewhere in P. The value of A is then calculated from B by

$$A = B \cdot p \quad + \quad 1 \cdot p \cdot q + 2 \cdot p \cdot q^2 + \cdots \\ + \quad (K-n) \cdot p \cdot q^{K-n} + (K-n+1) \cdot q^{K-n+1}$$

Algebraic simplification shows that $A \approx B$. The A value is also approximatively bound by $min(K-n+1, 1/p)$, the former being the maximal shift for a pattern and n value used, we recall. Figure 6 plots A for $GF(256)$ and any small n considered so far. As one sees, A is substantially smaller than its bound for middle-sized patterns. It is also several times larger than a single symbol match in our examples, showing the potential for much faster search, confirmed in depth below.

5.2 Experimental analysis

We performed extensive experiments in order to evaluate the robustness of the analytical study. We compare the following algorithms:

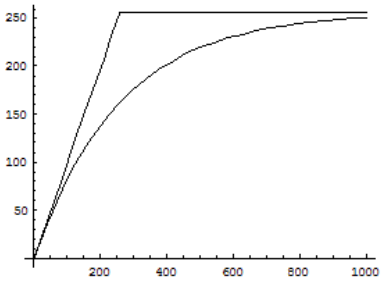

Figure 6: Graph of A and of the bound $min(K-n+1, 256)$

1. The Boyer-Moore algorithm, denoted BM;
2. The NGRAM algorithm based on full algebraic signature, denoted NGR_{full};
3. The NGRAM algorithm based on partial algebraic signature, denoted NGR_{part}.

The difference between NGR_{full} and NGR_{part} lies in the encoding of records. In the first case each symbol in the record is encoded by the cumulative signature from the beginning of the record. In the second case each symbol s is encoded by the signature of the n-gram that ends at s.

Experimental setting

The code for Ngram search can be downloaded from *http://www.lamsade.dauphine.fr/rigaux/ngram.zip*. All the algorithms are written in C. For Boyer Moore we use the C implementation provided by T. Lecroq (*http://www-igm.univ-mlv.fr/ lecroq*). We run all the experiences under either a mono-processor machine under Linux, or a bi-processor computer 2.2 GHz Turion 64 Bytes under Windows XP. The results reported below are obtained under the latter setting. The data sets consist of ASCII, DNA and XML files. All files are pre-encoded (calculation of signatures uses $GF(2^8)$) and loaded in main memory before the measurements. This phase does not influence the result. The search algorithms then execute on the in-memory files. In order to avoid the initialization overhead and any other side effects, each search is performed repeatedly until the search cost stabilizes. We report the minimal search time for one search.

For each algorithm we distinguish the following phases:

1. Pre-processing: the computation of the bad character and good suffixes table for BM, and the encoding of the pattern and the computation of the shift table for the NGRAM variants.
2. Processing: the search phase itself.

We measure independently both phases, since, depending on the context, preprocessing might be performed once but applied to many searches. This is the case for example in a distributed environment where a server transmits both the encoded pattern and the shift table to the storage units who perform locally the search phase. Table 2 shows the records used in our experiments. The ASCII file is a plain text version of the *Book of Common Prayer*.

Name	Type	Size
ascii_bcp	ASCII	940,678 bytes
dna_hs	DNA	167,280 bytes
xml_egov	XML	143,108 bytes

Table 2: Records used in experiments

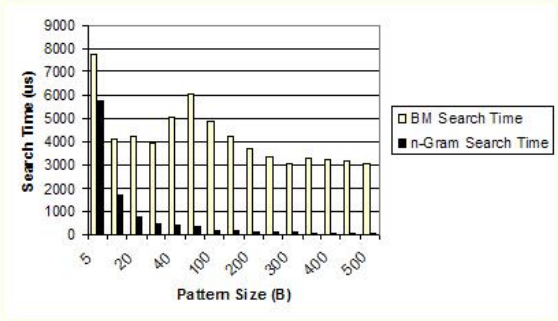

Figure 7: Search time for n-gram searchs and Boyer-Moore

The main conclusions of the experimental study are summarized as follows. First the outcome fully confirms the theory. The gain increases for larger patterns as it should. The NGR$_{part}$ algorithm based on partial algebraic signature appears particularly efficient for longer patterns in the context of the database search. The precise results depend on the data type.

In the following experiments, the n-gram size is set to 4. The elapsed time are in μs.

Search in DNA records

Table 3 shows the results for a pattern search in a DNA file, with variable pattern size. Here, "Ngram search" refers to the NGR$_{part}$ algorithm. The columns "Prepr. Time" and "Elapsed time" denote respectively the preprocessing and search time (the latter excluding the preprocessing phase). Column "Nb shifts" represents the number of matching attempts, whereas the column "Sum shifts" is the sum of the shift values, for all the shifts performed during a search over a file. Finally column "Ratio" is the ratio of the elapsed time of BM with the elapsed time of NGR$_{part}$.

Note that the elapsed times (and therefore the ratio) are machine-dependent, while the other figures are not because they only depend on the algorithmic features and the input.

We also show in the table the theoretical shift size, obtained analytically from the performance study of Section 5, and reported in Figure 6.

In our experiment, the number of shifts in NGR$_{part}$ strongly decreases as the length of the pattern increases. When a mismatch occurs, the algorithm searches for a match between the signatures of the final n-gram in the record at the current position and one of the n-grams in the pattern. When the pattern is small, it is unlikely to find a match and we shift by almost the length of the pattern. As the pattern becomes longer, the chances for a match increases, but on the other hand, the shift amount increases even more leading to a quick scan of the file. BM does not exhibit this behavior. As a result the search time (Figure 7) becomes much better for our algorithm as the pattern size increases. The

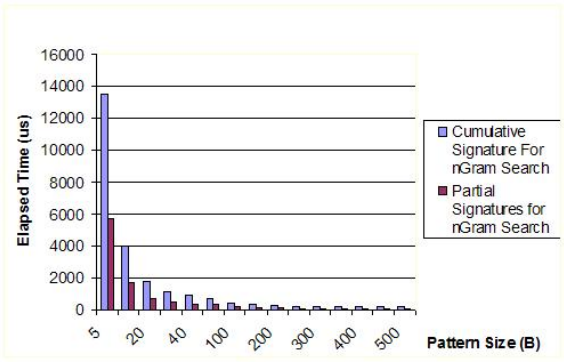

Figure 8: Comparison of elapsed time for Ngr$_{full}$ and Ngr$_{part}$

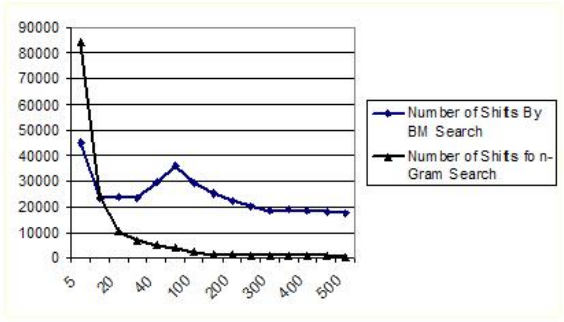

Figure 9: Evolution of the number of shifts with the size of the pattern

comparison between NGR$_{part}$ and NGR$_{full}$ shows clearly the advantages obtained by the former, due to the direct access to the n-gram signature in the encoded record (Figure 8). From now on the Ngram algorithm considered is NGR$_{part}$.

Figure 9 shows how the number of shifts evolves with the size of the pattern. For large patterns, a few attempts suffice to process the pattern search.

Figure 10 compares the values of the average shift for BM and NGR$_{part}$, algorithms. With NGR$_{part}$, the shift is equal to $K-(n-1)$, with K the size of the pattern, and n the size of ngram (we use $n=4$). Therefore the shift is $N-3$ when the n-gram is not found in the shift table, else the shift is the value found in the shift table which results from the preprocessing phase. We also plot on the same figure the theoretical shift results. For smaller pattern length, the match turns out to be almost perfect. For larger ones, the experimental values surpass the prediction, mainly because the n-grams are not evenly distributed.

Figure 11 shows the ratio between NGR$_{part}$ and BM. This ratio was about 2 for a pattern of size 6. This ratio is much higher for long pattern. On a bi-processor machine, the ratio reaches 72 for a pattern size of 500 symbols. This is directly related to the difference in the number of shifts, as reported on the previous figure.

The speed of our algorithm is lower when we use cumulative signatures. The cause are the additional XOR operations and log table accesses needed to obtain the n-gram from the CAS stored in record. However, the number of shifts necessary is the same and still yields an advantage over BM.

	Boyer-Moore search				Ngram search					
Pattern size	**Prepr. time**	**Elapsed time**	**Nb shifts**	**Avg. shifts**	**Prepr. time**	**Elapsed time**	**Nb shifts**	**Avg. shifts**	**Theor. shift**	**Ratio**
5	16	7745	44936	3.72	203	5758	84342	1.99	1.996	1.3451
10	12	4128	23223	7.20	194	1702	24312	6.91	6.918	2.4254
20	13	4221	23693	7.06	188	747	10187	16.49	16.47	5.6506
30	15	3943	23499	7.12	189	493	6554	25.62	25.67	7.9980
40	17	5043	29622	5.65	172	388	4874	34.45	34.45	12.9974
50	18	6038	36048	4.64	204	324	3919	42.84	43.01	18.6358
100	28	4907	29403	5.69	189	185	2053	81.74	80.86	26.5243
150	35	4208	25307	6.60	197	125	1483	113.09	111.99	33.6640
200	43	3715	22409	7.46	209	102	1223	137.14	137.59	36.4216
250	53	3343	20166	8.28	270	90	1077	155.58	158.63	37.1444
300	60	3080	18668	8.93	273	77	929	180.17	175.94	40.0000
350	72	3291	18702	8.93	248	72	858	195.05	190.17	45.7083
400	81	3217	18284	9.13	298	67	800	209.35	201.87	48.0149
450	90	3156	17941	9.30	274	62	745	224.51	211.49	50.9032
500	97	3057	17367	9.60	301	57	672	249.07	219.40	53.6316

Table 3: Results for DNA search

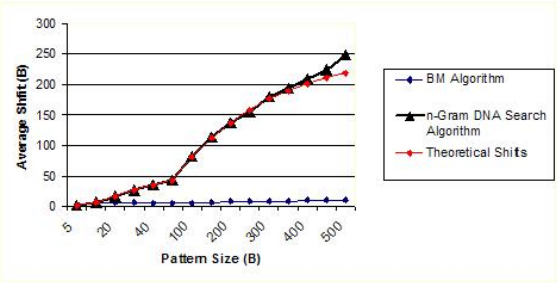

Figure 10: Average shift size for DNA search

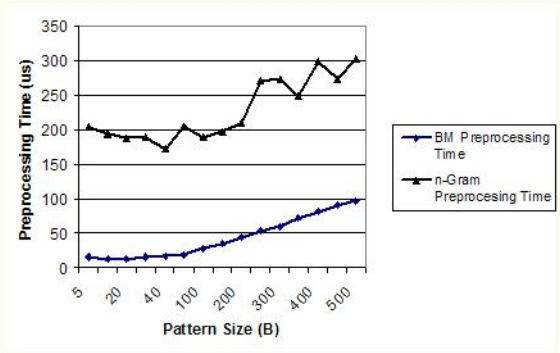

Figure 12: Preprocessing time

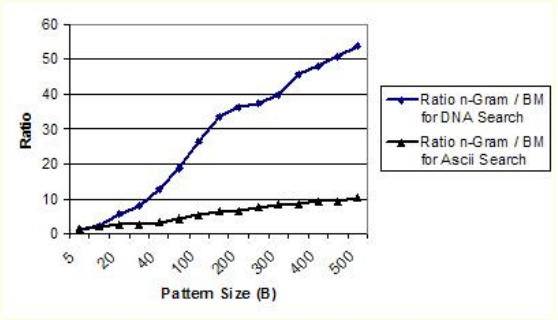

Figure 11: Ratio of search time

We recall again that the full CAS method allows other fast searches such as prefix searches. In addition, the CAS encoding works for all choices of n. The figure below shows pre-processing time. In both cases, the pre-procesing cost is negligible with respect to the search cost. There is no significant differences with respect to the pre-processing costs between Ngram search and BM search.

Figure 12 shows pre-processing time. BM preprocessing appears several times faster, e.g. almost 4 times towards the largest pattern. Howerever, in both cases the pre-procesing cost is negligible with respect to the search cost.

Search in ASCII records

Table 4 summarizes the results for searching ASCII texts. Recall that we search a plain text version of the Book of Common Prayer whose size is approximately 1 MB.

The difference is less impressive than with a DNA file. Actually the BM algorithm behaves better with alphabets of large sizes, since a single character is highly characteristic. Still, Ngram search performs at least as fast as BM for small patterns, and turns out to be almost twice as fast for large patterns. The following figure shows the compared curves of elapsed time for both algorithm. Figure 13 shows the compared curves of elapsed time for both algorithms.

The values of, respectively, the number of shifts and average shift size for BM and NGR_{part}, algorithms are shown

	Boyer-Moore search				Ngram search				
Pattern size	**Prepr. time**	**Elapsed time**	**Nb shifts**	**Avg. shifts**	**Prepr. time**	**Elapsed time**	**Nb shifts**	**Avg. shifts**	**Ratio**
6	14	4560	30168	5.30	211	3697	53868	2.98	1.2334
10	11	3364	21843	7.32	202	1630	23275	6.90	2.0638
15	13	2439	15946	10.03	171	982	13686	11.73	2.4837
20	13	1984	12051	13.27	165	716	9751	16.46	2.7709
30	15	1562	9250	17.28	201	481	6267	25.61	3.2474
50	17	1333	8324	19.20	202	308	3711	43.25	4.3279
100	25	917	5735	27.86	198	178	1976	81.19	5.1517
150	35	763	4458	35.81	233	121	1437	111.58	6.3058
200	43	660	3974	40.18	212	99	1186	135.18	6.6667
250	49	619	3821	41.78	222	83	1000	160.28	7.4578
300	58	628	3774	42.26	243	76	918	174.27	8.2632
350	66	576	3534	45.11	248	68	813	196.88	8.4706
400	75	609	3758	42.44	281	66	792	202.18	9.2273
450	86	591	3505	45.46	282	63	758	211.16	9.3810
498	92	624	3916	40.71	330	62	747	213.71	10.0645

Table 4: Results for ASCII search

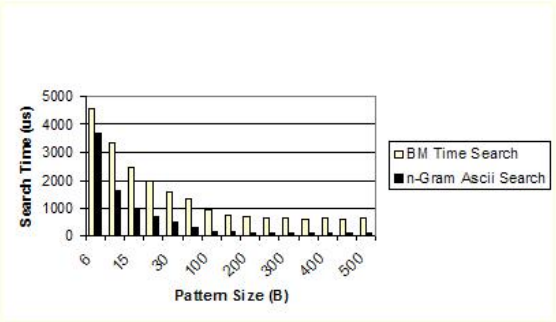

Figure 13: Elapsed time for ASCII search

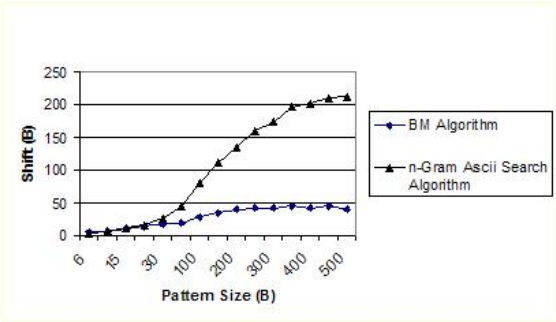

Figure 15: Average shift size for ASCII search

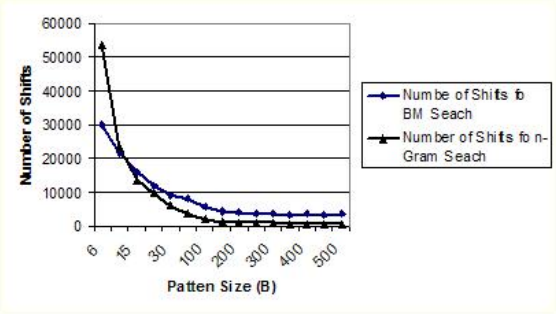

Figure 14: Number of shifts for ASCII search

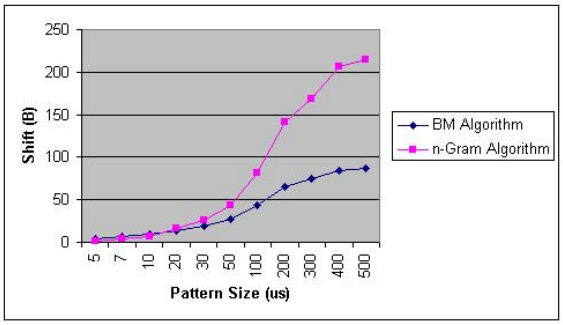

Figure 16: Average shift size for XML search

in Figure 14 and 15.

The figures clearly illustrates the good behavior of Ngram as patterns gets longer. As already mentioned for DNA files, the size of the size is roughly that of the pattern for small pattern size. Indeed the ngram signature found in the record is unlikely to match any ngram from the pattern. While this probability increases with large patterns, so does the size of the shift.

Search in XML records

We searched the collection of XML and XSL descriptions of (specifically) French certificates recognizing an out-of-wedlock child by the father. We searched, as for ASCII, for various patterns of different length. Patterns included tags. Table 5 and Figures 16 and 17 show the results for BM and partial CAS search.

The comparative ratio of search speed is now up to about six in the favor of the n-gram search. It is thus lesser than for ASCII text. In fact both methods accelerated the search with respect to ASCII. However, BM algorithm speeds up systematically more. The reason is apparently the repetitive presence of quite similar long XML tags in our benchmark. The "good suffix" case of BM takes over then more often, generating longer shifts. The n-gram takes a lesser advantage as these tags are longer than n.

	Boyer-Moore search				Ngram search				
Pattern size	**Prepr. time**	**Elapsed time**	**Nb shifts**	**Avg. shifts**	**Prepr. time**	**Elapsed time**	**Nb shifts**	**Avg. shifts**	**Ratio**
5	11	4577	30748	4.65	195	4924	72109	1.99	0.92
7	11	3430	22759	6.29	169	2500	36234	3.97	1.372
10	11	2441	15691	9.12	203	1461	20834	6,9	1.670
20	14	1702	10840	13.2	194	645	8772	16.38	2.638
30	15	1228	7568	18.9	178	432	5610	25.6	2.842
50	18	853	5214	27.43	170	279	3346	42.92	3.057
100	27	542	3239	44,12	185	165	1762	81.49	3.284
200	42	357	2195	65.04	234	84	1010	141.92	4.25
300	58	318	1517	74.08	258	70	851	168.42	4.55
400	75	269	1699	83.91	314	57	691	207.15	4.719
500	90	233	1654	86.43	270	49	668	214.67	6.423

Table 5: Results for XML search

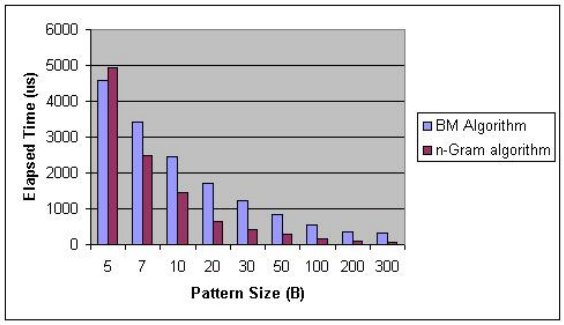

Figure 17: Search time for XML

6. TUNING TO ZERO COLLISION PROBABILITY

As usual, our algorithms are prone to various enhancements. One particularly interesting direction is the reduction to zero of the probability of a collision between n-grams. The absence of the collisions should increase the average shift, but, more interestingly, avoids the systematic final test of the successful matching. We discuss an approach to the more and more popular Unicode and to DNA symbol encoding.

6.1 Unicode

We recall a collision occurs if two different n-grams have the same AS. Collisions obviously slow the search speed. We recall from the theory of the algebraic signatures in [12] that first if two n-grams differ by only one symbol, then the probability of the collision is strictly zero (the algebraic signatures were the first signature scheme known for this property and its more general formulation for l-symbol signatures not dealt with here.). Furthermore, if we switch two characters in an n-gram, then the signatures of the n-gram differ (unless of course the switch does not change the n-grams).

We typically choose a GF implementation in which the primitive element is equal to `0x2`. Assume that we have ASCII text embedded in Unicode. The Unicode character code is then equal to the ASCII code with a leading 0-byte. Since $\alpha^8 =$ `0x0100` is also a primitive element (because 8 and $2^{16}-1$ are co-prime), we can form the signature with $\beta = \alpha^8$. The signature of two ASCII embedded Unicode characters `0x00rs` and `0x00tu` (with arbitrary hex-digits r, s, t, and u) is $\beta \cdot 0x00rs \oplus \beta^2 0x00tu = \beta(0xturs)$ which is different for all possible ASCII characters. Therefore, collision probability for digrams is zero.

n	Maximum number of n-gram signatures
1	4
2	16
3	64
4	256
≥ 4	256

Table 6: Maximum number of signatures for DNA

6.2 DNA

In the case of DNA, the character set is the alphabet $\Sigma = \{A, C, G, T\}$. If we are using $GF(2^f)$, then the number of possible n-gram signatures is given in Table 6. The question arises whether we can find an encoding that actually attains the maximum. We now give a general construction. Galois field elements are bit strings of length f and we identify them as a polynomial of degree up to $f-1$ over $\{0,1\}$. For example, the Galois field element 0110 0001 in $GF(2^8)$ is identified with the polynomial $x^6 + x^5 + 1$. Under this identification, Galois field multiplication is polynomial multiplication modulo a generator polynomial $g(x)$ of degree f. We typically use $x^8 + x^4 + x^3 + x^2 + 1$ for $g(x)$, but other choices are possible. With this choice of g (and many other ones), x turns out to be a primitive element and we use it for our α. Then, multiplication by α corresponds to multiplication by x which shifts the whole bit string by one. If there is an overflow, we XOR the result with a number derived from $g(x)$, in our case `0x1d` = 0001 1101. For example, multiplying 0101 0101 by $\alpha =$ 0000 0010 turns out to be 1010 1010, if we multiply again, then we have an overflow and the result is 0101 0100 $\oplus$ 0001 1101, which is 0100 1001. To represent our alphabet $\Sigma = \{A, C, G, T\}$, we use the Galois field elements 0000 0000, 0000 0001, 0001 0000, and 0001 0001. In this encoding, the choice of bits 0 and 4 (from the left) determines the choice of character. Since the signature of a 4-gram (c_3, c_2, c_1, c_0) is $\alpha(c_3 \oplus \alpha c_2 \oplus \alpha^2 c_1 \oplus \alpha^3 c_0)$, we determine the uniqueness of the second factor. In forming it, we shift c_0 three times, this is the maximum shift and it involves no overflow. We can read the bits set in c_0 from bits 3 and 7, the bits in c_1 from bits 2 and 6, the bits in c_2 from bits 1 and 5 and of course the bits in c_3 from bits 0 and 4. Therefore, the signatures of different 4-grams are differ-

ent. With this encoding, we have reduced the probability of n-gram signature collision for $n \leq 4$ to zero.

7. RELATED WORK

In the general computer science literature, the pattern matching is among the fundamental problems with many prominent contributions [4]. The popular algorithms do not attempt to take advantage of any specific record encoding. They were not designed for our "write once read many" database context. BM is accepted as a very versatile and usually the fastest search, outperforming the other prominent pattern matching methods. Its match attempt and shift complexity can be about ours for the partial CAS and $n = 1$. This happens, whenever the "bad character" shift becomes prominent with respect to the "good suffix" suffix case. In the practical conditions similar to those addressed in previous sections, the n-gram search time should be often much faster. It should thus usually similarly outperform the other well-known algorithms. As we said, the exact comparison remains however an open research problem.

The SDDS-2005 system already manages the SDDS files in distributed RAM, with records encoded into their full CAS's [10]. The encoding protects the records at the servers against involuntary viewing, while the data remain available for the parallel, distributed non-key scans. A specialized crypto-attack at a server can decode the data by finding α. But this is legally analogous to steaming a closed envelope, with the same social and penal consequences.

SDDS-2005 offers several other string search algorithms over its CAS encoded records. There is the prefix matching, the longest common prefix or string matching and an alternative pattern matching. That one uses a Karp-Rabin like sequential traversal [8, 5]. The algorithm avoids pattern preprocessing other than AS calculus. The AS is the only data sent to the servers to match. This can be a security advantage on its own. Besides, as in general for a sequential algorithm, lack of preprocessing may, perhaps, lead to a faster search than with n-grams.

Our principles fall into the recent general direction for the distributed database architecture termed Database As Service (DAS) model [7]. The encryption at a DAS server is a must. But one advocates a compromise over the strength/trust, for an efficient parallel processing. Our partial CAS encoding trades somehow minimal DAS security of stored data, towards currently the fastest pattern matching.

Generally, while private and semi-private information on networks has grown rapidly, mechanisms for searching for privacy-protected information have failed to keep pace. One set of solutions explored among others by [1] and Chang et al. [3] uses keyword indexes to describe the contents of files. This is still a surprisingly hard question if hard privacy guarantees are required. Song et al. [14] give the first practical solution to the problem of searching encrypted data by keyword. Golle et al. [6] extends the capabilities to conjunctive key searches. Common to these approaches is the primary concern of not compromising the privacy of the data, but instead restricting the search capability. Schwarz et al. [13] propose a different method that trades search capability for lesser security. Unlike the other methods above, it is thus also a solution for a DAS. Our protection is weaker. But it offers zero storage overhead, we recall and faster processing speed

8. CONCLUSION

Our method shows the potential for the fastest pattern matching string search in databases, especially for longer patterns. It should prove useful new tool for text, image, DNA, etc. Its search without decoding capability should also be valuable for the distributed environments like DAS, P2P, SDDS, or grid.

Further work includes more performance studies, including the zero collision probability encoding in Section 6. It should be also interesting to study variants of selected algorithm specs. For instance, the partial CAS may more efficiently use table T when the rightmost n-grams match (a kind of equivalence to the BM good suffix case). Also a different implementation of T could prove better for the Unicode. A particularly promising direction consists of adding the approximate pattern matching capabilities.

Acknowledgments. The support for this work came from EGov IST project.

9. REFERENCES

[1] M. Bawa, R. Bayardo, and R. Agrawal. Privacy Preserving Indexing of Documents on the Network. In *VLDB*, 2003.

[2] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[3] Y.-C. Chang and M. Mitzenmacher. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *Applied Cryptography and Network Security Conference (ACNS)*, Springer LNCS, 3531, 2005.

[4] M. Crochemore and M. Lecroq. *Pattern Matching and Text Compression ALgorithms*. CRC Press Inc, 2004.

[5] M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.

[6] P. Golle, J. Staddon, and B. Waters. Public Key Encryption with Conjunctive Field Keyword Search. In *Applied Cryptography and Network Security Conference (ACNS)*, Springer LNCS, 3531, 2005.

[7] H. Hacigumus, B. Hore, B. Iyer, and S. Mehrotra. *Search on Encrypted Data*. Springer Verlag, 2007.

[8] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[9] W. Litwin, R. Mokadem, P. Rigaux, and T. Schwarz. Pattern Matching Using Cumulative Algebraic Signatures and n-gram Sampling. In *Intl. Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE'06)*, 2006.

[10] W. Litwin, R. Mokadem, and T. Schwarz. Pre-computed Algebraic Signatures for faster string search. Protection against incidental viewing and corruption of Data in an SDDS. In *Intl. Workshop DBISP2P 2005*, 2005.

[11] W. Litwin, R. Moussa, and T. Schwarz. LH*RS-A Highly-Available Scalable Distributed Data Structure. *ACM Trans. on Database Systems*, 30(3):769–811, 2005.

[12] W. Litwin and T. Schwarz. Algebraic Signatures for Scalable Distributed Data Structures. In *Proc. Intl. Conf. on Data Engineering (ICDE)*, 2004.

[13] T. Schwarz, P. Tsui, and W. Litwin. An Encrypted, Content Searchable Scalable Distributed Data Structure. In *Int. Workshop on Security and Trust in Decentralized / Distributed Data Structures (STD3S)*, 2006.

[14] D. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Security and Privacy Symposium*, 2000.