# 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 &
INRIA-Orsay, Equipe Gemo
philippe.rigaux@dauphine.fr

Thomas Schwarz
Univ. Santa Clara
tjschwarz@scu.edu



## ABSTRACT

We propose a novel string search algorithm for data stored once and read many times. Our search method combines the sublinear traversal of the record (as in Boyer Moore or Knuth-Morris-Pratt) with the agglomeration of parts of the record and search pattern into a single character – the algebraic signature – in the manner of Karp-Rabin. Our experiments show that our algorithm is up to seventy times faster for DNA data, up to eleven times faster for ASCII, and up to a six times faster for XML documents compared with an implementation of Boyer-Moore. To obtain this speed-up, we store records in encoded form, where each original character is replaced with an algebraic signature.

Our method applies to records stored in databases in general and to distributed implementations of a Database As Service (DAS) in particular. Clients send records for insertion and search patterns already in encoded form and servers never operate on records in clear text. No one at a node can involuntarily discover the content of the stored data.


## 1. INTRODUCTION

We describe a novel string (pattern) matching principle, called *n-gram search*, first proposed in preliminary form in [10]. We designed our method for databases and files where records are stored once and searched many times. Our search algorithm combines the basic sub-linear method of Boyer Moore, Quick Search *et al.* [4] with a Karp-Rabin like agglomeration of several characters into a single signature of the same size as a character. Our algorithm traverses the record similar to Boyer Moore (BM), but instead of comparing characters, it compares signatures of $n$-grams and thus allows us to compare $n$ characters at once. Compared with BM, our algorithm performs especially well when the signature-based comparison is much more discriminative than that of single characters. We gain this speed advantage because our method tends to use much larger shifts than BM. In the case of DNA records, stored as usual as an ASCII file, the characters themselves can be only one of four values and comparing signatures of 4-grams is much more effective. As a result, we measured our method to be up to seventy times faster than BM for DNA data. Our algorithm showed itself to be at least several times faster than BM also for ASCII and XML text.

To speed up signature calculation, we store the record in encoded form, but in place, i.e., without additional storage overhead. In our scenario, the trade-off between the lowered costs of searches and the single encoding costs during record generation and decoding for reads is advantageous.

Our method presents two advantages. First, as we said above, it is very fast. The second advantage lies in its applicability for 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. He finds himself in the position of someone in possession of an opened letter not addressed to himself.

These features make 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 that differ in the amount of encoding and offer a trade-off between the speed of searches and the amount of decoding necessary when reading a record.

Below, Section 2 gives a bird eye's view of our novel scheme. Section 3 explains our record encoding and reviews the basic properties of algebraic signatures that we use. We present the two variants of the algorithm in Sections 4 and 5, respectively. Section 6 contains analytical and experimental performance analysis, including a comparison to Boyer-Moore. For experimental analysis we use DNA records, ASCII text and XML documents. We then discuss possible improvements (Section 7), present related


*Work partially funded by the EGov IST Project and by the WISDOM project, *http://wisdom.lip6.fr*.





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

work (Section 8) and conclude.

## 2. N-GRAM SEARCH

Boyer's and Moore's pattern search algorithm (BM) [2] tries to match a search pattern against a given position in a record. It first looks at the character in the record on top of the last character in the pattern. Chances are that they do not match. In this case, the occurrence of that character in the pattern determines how far we can shift the pattern to the right without missing a potential match. In Figure 1(a), we are searching for the pattern "Dauphine". The first matching attempt places the "e" in "Dauphine" under an "i". Since the "i" appears in the pattern two characters to the left, we shift the pattern by two characters to the right under the record. In this position, the last letter of pattern and record substring match, but a comparison with the full pattern shows that this is not a match. Since there is no further "e" in "Dauphine", we now shift by the full length of the pattern. In addition to this "bad character" shift, BM also has a "good suffix" shift, but this has no equivalence in our scheme. The BM variant without the latter is known as Quick Search [4].

```
(a) Universite de Technologie Paris Dauphine
    Dauphine  Dauphine   Dauphine   Dauphine
      Dauphine   Dauphine        Dauphine

(b) Universite de Technologie Paris Dauphine
    Dauphine      Dauphine     Dauphine
           Dauphine      Dauphine   Dauphine
```

**Figure 1: (a) Boyer-Moore Type Search and (b) $n$-gram Search with $n = 2$ .**

The efficacy of the "bad character" shift depends on the size of the character set and the frequency statistics of the characters in the data set. At the first match attempt in Figure 1(a) we compare "i" against "e" in the pattern and shift by 2. Using digrams, Figure 1(b), we compare "si" against "ne" and since there is no "si" in the pattern, we can shift by the maximum amount of 7. The following shifts are also by the maximum amount until we hit the actual match. In contrast, the BM variant only once used its maximum shift of 8. Depending on the data, we can often do better with longer $n$-grams. Unfortunately, a naïve implementation of this idea creates a much bigger BM shift table and is not practical. However, algebraic signatures compress the information of an $n$-gram into a single character. Distribution of $n$-gram signatures is more even than distribution of characters in actual data sets and consequentially, $n$-gram signatures are more discriminating than single characters. In consequence, average shift size goes up and the search performs faster.

Calculating $n$-gram signatures directly from the record takes time. To optimize search times, we can replace every character in the record with an $n$-gram signature. This avoids calculating $n$-gram signatures altogether. We can still convert an encoded record in linear time to the original. Alternatively, we can replace a character with the algebraic signature of the record up to that character. This still allows us to calculate $n$-gram signatures quickly, but also enables other searches such as longest prefix search. We describe an algorithm based on the latter approach in Section 4 and on the former in Section 5. The next Section recalls the basic definitions and properties of Algebraic Signatures. More details can be found in [12].

## 3. ALGEBRAIC SIGNATURES

We assume that our records are encoded in a character set where each character is a bit string of length $f$. We interpret these characters as elements or *symbols* of a Galois Field (GF) of size $2^f$ called $GF(2^f)$ in the following. We recall that a Galois field is a finite set that supports addition, denoted $\oplus$, and multiplication, denoted $\otimes$. These operations are associative, commutative and distributive, have neutral elements, and there exist inverse elements for both. Identifying the character set of the records with a Galois field provides a convenient mathematical context to condense the contents of a substring into a single character, as we will see.

A symbol $\alpha$ is *primitive* if, for any element $\beta \neq 0$ of the GF there exists some $i$, $0 \leq i \leq 2^f - 2$, such that $\beta = \alpha^i$. In other words, the powers of $\alpha$ enumerate all the non-zero elements of the GF. The algebraic signature (AS) of a record $r_1 \cdots r_i$ with respect to primitive $\alpha$ is given by

$$r_1\alpha \oplus r_2\alpha^2 \oplus \ldots \oplus r_i\alpha^i$$

As usual, we implement Galois field addition as the familiar XOR. The implementation of the multiplication is more involved and we postpone its discussion to Section 3.3. 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.

### 3.1 Cumulative Algebraic Signature

Let $R_M$ be a record of $M$ symbols $r_1 \cdots r_M$. For each $r_i$ we calculate 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 $R_M$. Thus, the full CAS replaces each individual symbol $r$ in the encoded string with another symbol $r'$ encoding not only the knowledge of the current symbol but also additional knowledge of all the symbols preceding $r$. Comparison of the ASs in a CAS yields information about likely equality or certain inequality of the entire prefixes ending with the matched symbols. The information contained in a single character in the CAS encoding includes information about all preceeding characters in the string. Using it for pattern matching promises much higher efficiency than using the original record.

The algebraic properties of AS allow us to quickly calculate the AS of an $n$-gram from the CAS encoded record. First

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

We can thus calculate the CAS encoding in linear time from the original record. Reversely, we have

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

This allows us to recover the original record from the CAS in linear time. We call this process *decoding.* Note that in a Galois field $GF(2^f)$, addition is the same as substraction.

### 3.2 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 $i \in \{n, \cdots, M\}$. The *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 $r_i$, for $i \geq n$, or over the $i$-gram terminating with $r_i$ for $i < n$. Formally, we have:

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

The following algebraic properties allow us to calculate the AS of any $n$-gram in $R_M$ from a full CAS and to convert between the partial and full CAS of $R_M$. 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:

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

### 3.3 Implementing GF Multiplication and Division

There are several methods for multiplying and dividing in a GF. In our context, the use of single logarithm and antilogarithm tables appears to be the most efficient [11]. The tables precalculate the log and antilog values. The logarithm of a GF element $\beta \neq 0$ is the (unique) integer $i$, $0 \leq i \leq 2^f - 2$, such that $\alpha^i = \beta$. We define the logarithm of 0 to be $2^f - 1$. We implement Equation (4) as

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

Here, the operator $\oplus$ denotes GF addition and substraction. The other additions/substractions in the formula are the usual integer operations. Similarly, we move between the original record and its full CAS by calculating

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

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

## 4. PATTERN MATCHING IN FULL CAS

We present now the first variant of our algorithm which works on records encoded in their (full) CAS form. We recall that encoding is done in linear time using using property (1) and that we design for the insert once, search often scenario. Our search method first pre-processes the pattern, generating a *shift table* that is used in the second phase to find all matches of the pattern in the records. We only discuss the matching process within a single record in this paper.

### 4.1 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 $K \geq n$. Starting from the beginning, we determine for every possible signature in the pattern the shift amount of $P$ against $R$. We store this information in an auxiliary data structure, the *shift table,* $T[0, \ldots, 2^f - 1]$. We have $T[h] = s$, where $h$ is the logarithm of an $n$-gram signature ($h = log_\alpha(AS(p_{i-n+1}, p_{i-n+2}, \cdots, p_i))$ and $s$ is the shift length. We call $h$ the the *Logarithmic Algebraic Signature* (LAS) of the $n$-gram. Using the LAS instead of the AS of an $n$-gram

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

**Table 1: Shifts for each 2-grams in `Dauphine`**

avoids taking an antilogarithm whenever we calculate the AS of an $n$-gram in the matching phase of our algorithm discussed in the next Section. From Equation (5) we obtain

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

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.

1. We preset every entry $h$ to $T[h] = K - n + 1$. This represents the maximal value of the shift: if the LAS $h$ is found in the (encoded) record but not in the pattern, then the next matching attempt is $K - n + 1$ position to the right.

2. We compute the shift for the signatures found in the pattern. For every LAS $h$ of an $n$-gram in $P$ other than the last one, we set $T[h]$ to the offset to the end of $P$ of the rightmost $n$-gram with $h$ as LAS.

The second rule means that if a signature $h$ is found in both the (encoded) record $R$ and in the (encoded) pattern $P$, then the next matching attempt shifts $P$ such that the positions of $h$ coincide. A special case occurs when $h = V$. 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.

EXAMPLE 1. *We use the pattern $P$ = `Dauphine`. We choose $n = 2$, i.e., we intend to perform a 2-gram (digram) based search. We initialize every $T[h]$ to $K - n + 1 = 7$. We then set $T[LAS($`da`$)] = 6$, $T[LAS($`au`$)] = 5$, etc. Table 1 illustrates the result.* □

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] := αP[1]
      else
         P'[i] := P'[i − 1] XOR α^i P[i]
      endif
   endfor
   // Compute table T. First initialize with the maximal shift
   for (i = 0 to 2^f − 1)
      T[i] := size(P) − n + 1
```

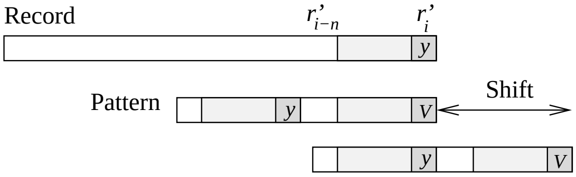


**Figure 2: $n$-gram shift, with $y = LAS(r_{i-n+1}, \cdots, r_i)$ and $V \neq y$**

**endfor**
// For each $n$gram, 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**

## 4.2 Pattern Matching

We now describe the search for $P = (p_1, p_2, \cdots, p_K)$ in an encoded record $R = (r_1, r_2, \cdots, r_M)$ of length $M$. 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 $p_i$.

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 of the record, with $LAS(P_K^n)$ that is in $V$.

1. If there is the match, then we compare $LAS(R_K^K)$ 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 $r_M$.

Figures 2 and 3 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 (8) we obtain the LAS of the $n$-gram of $R$ at $i$ as $y = LAS(r_{i-n+1}, \cdots, r_i)$. 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 2), or $y$ is not found in the pattern, in which case the shift found in $T$ is $K - n + 1$ (Figure 3).

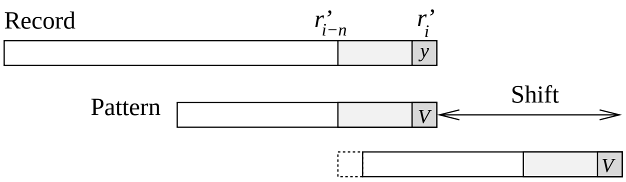


**Figure 3: The $n$-gram shift, when $y = LAS(r_{i-n+1}, \cdots, r_i)$ is not found in the pattern**

Using signatures, there is a possibility of *collisions,* where two non-equal strings have the same signature. The pattern matches reported by our algorithm up to now are only likely matches. We have to make sure that the match is a true one. The simplest method is to finish our algorithm by a

```
(a) AGCATATAAAGCGAGTGCGGAGCAT
    AGACAGAT       AGACAGAT
          AGACAGAT      AGACAGAT

(b) AGCATATAAAGCGAGTGCGGAGCAT
    AGACAGAT AGACAGATAGACAGAT
           AGACAGAT AGACAGAT

(c) AGCATATAAAGCGAGTGCGGAGCAT
    AGACAGAT    AGACAGAT
     AGACAGAT    AGACAGAT
      AGACAGAT    AGACAGAT
       AGACAGAT    AGACAGAT
        AGACAGAT    AGACAGAT
         AGACAGAT
          AGACAGAT
```

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

character to character comparison after decoding records. In the distributed case (e.g. DAS), this final verification occurs at the client. Section 7 gives variants that optimize this final verification step.

Example 2. *We now revisit our introductory example of matching $P$ = 'Dauphine' in record $R$ = 'Universite de Technologie Dauphine'. We already pre-processed the pattern in Example 1. The value of $V$ is $LAS($ne$)$. Figure 1 shows how the pattern traverses the record. In the initial position, we calculate $i = LAS($si$)$, using (8). Since $i \neq V$ (unless there is a collision,) we consult our shift table and find $T[LAS($'si'$)] = 7$. Accordingly, we shift the pattern by 7 characters to the right. Proceeding in this manner, our fifth attempt leads to the digram 'up' in the record. In this case, the shift table has $T[LAS($'up'$)] = 4$ and the shift moves the pattern into the correct position under the string. Our algorithm now tests whether this is a likely match by comparing further algebraic signatures. Because of collisions (when signatures of different $n$-grams coincide) there is a small possibility of a false positive. Figure 1 (a) shows the same search with $n = 1$. The shifts are the same as for Quick Search, i.e. BM without the (rare) good suffix shift.* □

Example 3. *Our next example illustrates our approach for DNA sequences. It is adapted from one in [4] (Figure 4). We have the four-letter alphabet of nucleotides: A, C, G, T. The pattern is 'AGACAGAT'. If we use bytes to store our signatures, then we can actually find a collision-free way of calculating signatures of $n$-grams with $n \leq 4$. In our example, choosing $n = 1$ leads to twelve attempts and an average shift of less than 1.5 symbols. Choosing $n = 2$ results in four attempts, but choosing $n = 3$ only results in three attempts for an average shift length of 5.6 symbols.* □

These and others examples show the impact of the selection of $n$ on the number of shifts. Usually, the signature contains more information for larger values of $n$ (such as $n = 3$ or $n = 4$) and yields longer shift, but on the other hand, the size of the pattern and $n$ limit the maximum shift. If it happened that $n = K$, the length of the pattern, then we would directly look for likely matches, but would also only be able to shift by one character to the right. It would thus traverse the record as in Karp's and Rabin's method, but would obtain the signatures directly or more directly from the encoded record instead of calculating them from the last signature.

# 5. PATTERN MATCHING WITH PARTIAL CAS

The full CAS encoding allows for the dynamic choice of $n$. It is also particularly efficient for other useful searches such as prefix search, longest common prefix search, or longest common substring search [11]. Our second variant encodes the record directly into the AS of $n$-grams and avoids the LAS calculations of our first variant. However, the search algorithm can no longer dynamically choose $n$ and other searches are now much more expensive.

## 5.1 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 3, 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 can calculate recursively

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

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 $i > n$:

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

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

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

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

## 5.2 Pattern Pre-processing and Matching

Pre-processing pattern $P$ proceeds in the same manner as in the first variant. The search itself proceeds in a similar manner by attempting to match $n$-grams. However, now there is no need to calculate the AS for an $n$-gram in the record, since they are already directly encoded in $R''$. If matching with $V$ is successful, we need to confirm whether this possible match extends to the whole pattern. Since calculating the AS for the current string in $R$ that might match from $R''$ would be cumbersome, we instead compare the AS of all $n$-grams in $P''$ and the possible match in $R''$. If all of these comparisons succeed, then we conclude probabilistically that we have a match. (See Section 7.3) If any match attempt fails during this process, we calculate the shift of the terminal $n$-gram signature from $T$. We confirm a successful match, as in Section 4.2 by a character by character verification after decoding our record. The verification concerns the first $n-1$ characters only, for the reasons discussed in Section 7.3.

# 6. PERFORMANCE ANALYSIS

We first derive analytically the average shift size in dependence on $n$ before we report on the results of our extensive experiments.

## 6.1 Analytical Study

Pattern pre-processing costs $O(2Kn+1)$, as it involves (linear) encoding and the creation of $T$. Similarly, encoding or decoding of a record costs $O(M)$. Pattern matching itself is $O(N)$, where $N$ is the number of attempts to match. We have $N = (M-K)/A$, where $M$ is the record size and $A$ the average amount of a shift. Despite the same $O(N)$ cost formula, the search speed turns out to be faster when using $n$-gram (partial) CAS, since we avoid the XOR calculus and log/antilog calculus. While obtaining theoretical results seems to be impractical, our experiments show a speed up between one and two.

The average shift $A$ is longer when the probability of matching $n$-gram signatures is smaller. Our examples illustrated this for some choices of $n = 2, 3$, or 4. For $n = 4$, the probability of one 4-gram from a typical record matching another one is already close to the minimal probability of $2^{-f}$ or 1/256 for our favorite GF with 256 elements. In addition, the average shift is limited by the size of the pattern and can be at most $K - n + 1$.

We now calculate $A$ under the assumption that all signatures are equally likely and independent from each other. Despite these simplifying assumptions, our result seems to be confirmed by our experimental values. We set $p = 2^{-f}$ and $q = 1-p$. Assume that we have a match on the rightmost signatures of the pattern and the record. After determining whether to report this match as a likely match, we shift the pattern to the left by $B$ characters. With probability $p$, the second $n$-gram from the right in the pattern matches and we shift by 1. With probability $qp$, the lead $n$-gram in the record is two from the right in the pattern, and so on. With probability $q^{K-n}$, the lead $n$-gram does not appear elsewhere in the pattern and we shift by $K-n+1$. All together

$$B = \sum_{i=0}^{K-n-1} \Big((i+1)pq^i\Big) + (K-n+1)q^{K-n}.$$

Using a formula for the derivative of a finite geometric series (or Mathematica™), this simplifies to

$$\begin{aligned} B &= p \cdot \sum_{i=0}^{K-n-1} \Big((i+1)q^i\Big) + (K-n+1)q^{K-n} \\ &= p \cdot \frac{(K-n)q^{K-n+1} - (K-n+1)q^{K-n} + 1}{(q-1)^2} \\ &\quad + (K-n+1) \cdot q^{K-n} \\ &= \frac{1 - q^{K-n+1}}{p} \end{aligned}$$

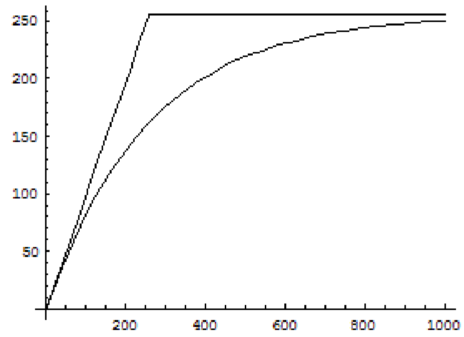


**Figure 5: Graph of $A$ depending on pattern length and of the bound $min(K-n+1, 256)$. $n$ is 4.**

In the general case, the rightmost signatures of the pattern and in the string coincide with probability $p$ leading (after evaluation) to a shift of $B$. With probability $pq$, we shift by one, with probability $pq^2$ by two, etc. Therefore

$$A = pB + \sum_{i=1}^{K-n}(ipq^i) + (K-n+1)q^{K-n+1}$$

The second and third addend are equal to $qB$ and therefore

$$A = B$$

As $K \to \infty$, $A \to 1/p$. It is bound from above by $min(K - n + 1, 1/p)$, but, as Figure 5 shows for $n = 4$, the fit is not tight.

## 6.2 Experimental Analysis

We performed extensive experiments in order to evaluate our method for actual records. We compare the following algorithms:

1. The Boyer-Moore algorithm, denoted by BM;
2. The NGRAM algorithm based on full algebraic signature (full CAS), denoted by $\text{NGR}_{full}$;
3. The NGRAM algorithm based on partial algebraic signature (partial CAS), denoted by $\text{NGR}_{part}$.

We recall that the difference between $\text{NGR}_{full}$ and $\text{NGR}_{part}$ lies in the encoding of the record. In the first case, full CAS, each symbol in the encoded record is the signature of all characters in the record up to and including this position. In the second case, partial CAS, each symbol in the encoded record is the signature of the $n$-gram that ends at $r$.

## Experimental Setting

The code for Ngram search can be downloaded from *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 (*www-igm.univ-mlv.fr/~lecroq*). This choice should guarantee that we compare with a first class, fully optimized, general implementation of BM. We ran all the experiences under either a mono-processor machine under Linux, or a bi-core computer 2.2 GHz Turion 64b under Windows XP. We obtained the results reported below on the latter machine. 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 initialization overhead and any other side effects such as CPU or memory contention from OS processes, each search is performed repeatedly until the search cost stabilizes. We report the minimal search time for one search.

In our results, we distinguish the following phases for each algorithm:

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 both phases independently because – depending on the context – pre-processing might be performed only once. This would be the case in an distributed environment where a client application pre-processes the pattern and sends it and the shift table to all storage servers. We used the following data sets:

1. An ASCII text, a plain text version of the *Book of Common Prayer* of size 941KB.
2. Human DNA chromosome 17 with 167K characters.
3. A 143KB collection of XML and XSL descriptions of French government certificates in which fathers recognize out-of-wedlock children.

We can summarize the main conclusions of our experiments as follows. First the outcome fully confirms our expectation that our method is faster than BM. The gain increases for larger patterns as it should. The $\text{NGR}_{part}$ 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 $\mu$s.

## Search in DNA records

Table 2 shows the results for a pattern search in a DNA file, with variable pattern size. Here, "Ngram search" refers to the $\text{NGR}_{part}$ 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 $\text{NGR}_{part}$.

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 6, and reported in Figure 5.

In our experiment, the number of shifts in $\text{NGR}_{part}$ strongly decreases as the length of the pattern increases. When a mismatch occurs, the algorithm searches for a match between

| | 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 2: Results for DNA search**

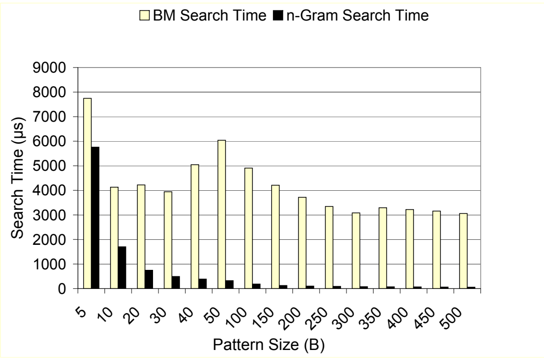


**Figure 6: Search time for $n$-gram searchs and Boyer-Moore**

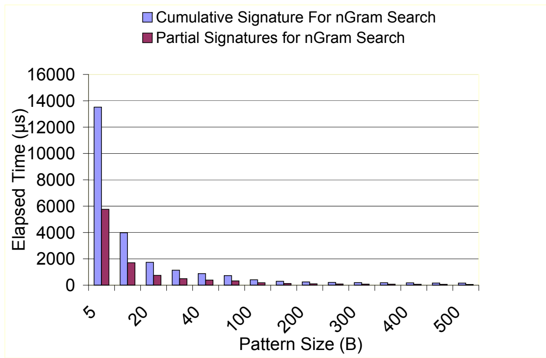


**Figure 7: Comparison of elapsed time for Ngr$_{full}$ and Ngr$_{part}$**

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 increase, 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, search time (Figure 6) becomes much better for our algorithm as the pattern size increases. The 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 7). From now on the $n$-gram algorithm considered is NGR$_{part}$.

Figure 8 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 9 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

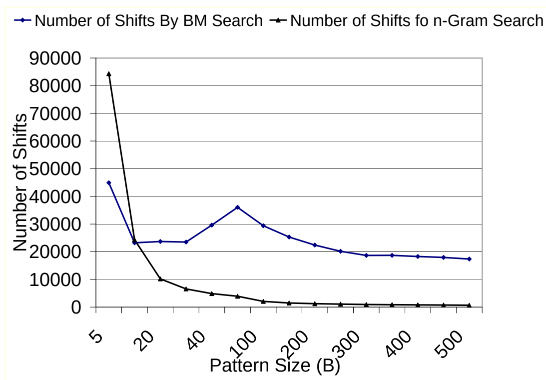


**Figure 8: Evolution of the number of shifts with the size of the pattern**

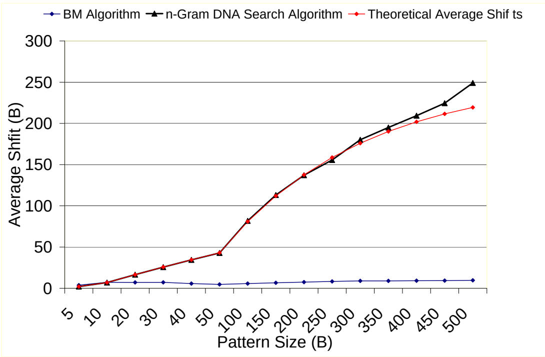


**Figure 9: Average shift size for DNA search**

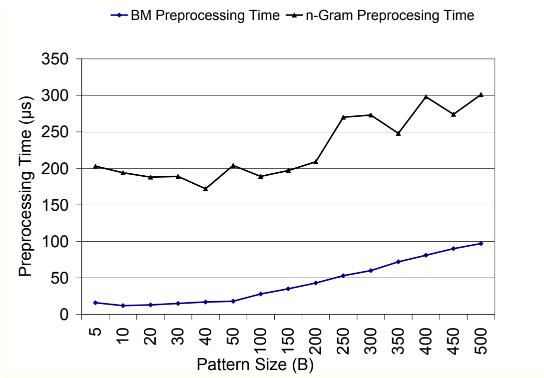


**Figure 11: Preprocessing time**

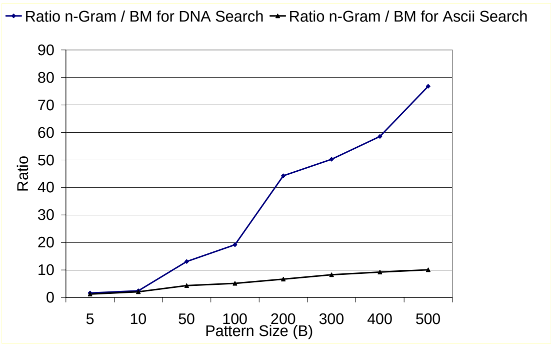


**Figure 10: Ratio of search time**

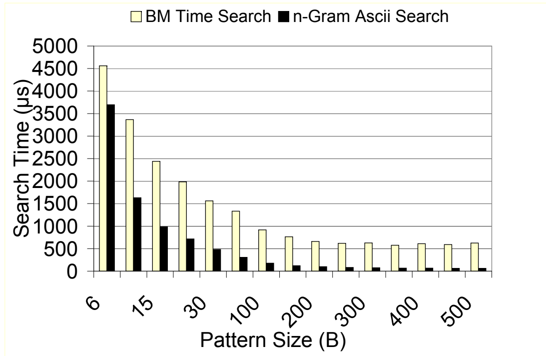


**Figure 12: Elapsed time for ASCII search**

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 10 shows the ratio between $\text{NGR}_{part}$ 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 in our 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 the encoded form of the record. However, the number of shifts necessary is the same and still yields an advantage over BM. 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 $n$.

Figure 11 shows pre-processing time. BM preprocessing appears several times faster, e.g. almost 4 times towards the largest pattern. However, in both cases the pre-processing cost is negligible with respect to the search cost.

## Search in ASCII Records

Table 3 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 the DNA file. Of course, the BM algorithm behaves better with alphabets of large sizes, since a single character contains much more discriminative. Still, $n$-gram search performs at least as fast as BM for small patterns, and turns out to be almost twice as fast for large patterns. Figure 12 shows the compared curves of elapsed time for both algorithms.

The values of the number of shifts and average shift size for BM and $\text{NGR}_{part}$, algorithms respectively are shown in Figure 13 and 14.

The figures clearly illustrates the good behavior of $n$-gram as pattern size increases. As already mentioned for DNA files, the size of the shift is roughly that of the pattern for

| | 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 3: Results for ASCII search**

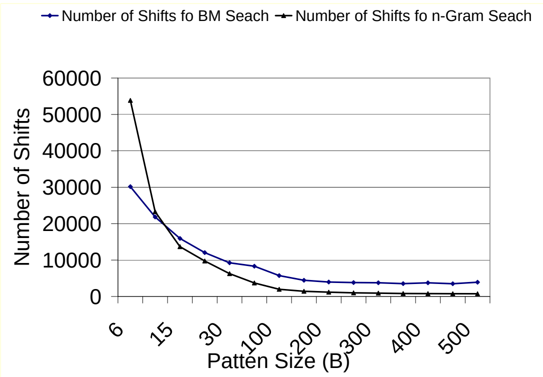


**Figure 13: Number of shifts for ASCII search**

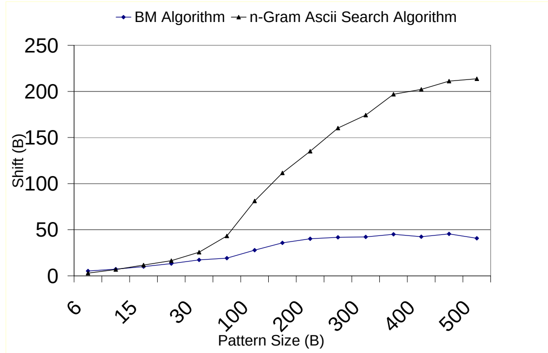


**Figure 14: Average shift size for ASCII search**

small pattern size. Indeed the $n$-gram signature found in the record is unlikely to match any $n$-gram from the pattern. While this probability increases with long patterns, so does the size of the shift.

## Search in XML records

We searched, as for ASCII, for various patterns of different length in our XML collection. Patterns included tags. Table 4 and Figures 15 and 16 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 less than for ASCII text. In fact both methods sped up compared to their performance with ASCII records. However, the BM algorithm gains systematically more. Apparently, the reason is the repetitive presence of quite similar long XML tags in our benchmark. The "good suffix" case of BM takes then over more often, leading to longer shifts. The $n$-gram takes lesser advantage of these tags as they are longer than $n$.

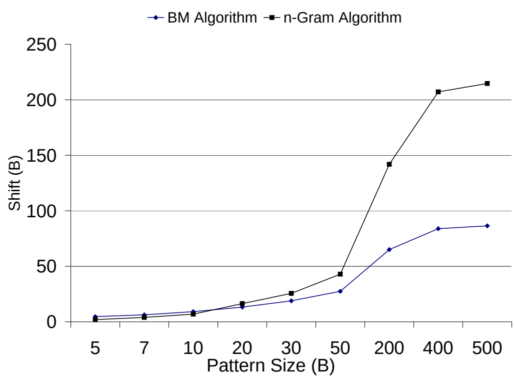


**Figure 15: Average shift size for XML search**

| | 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 4: Results for XML search**

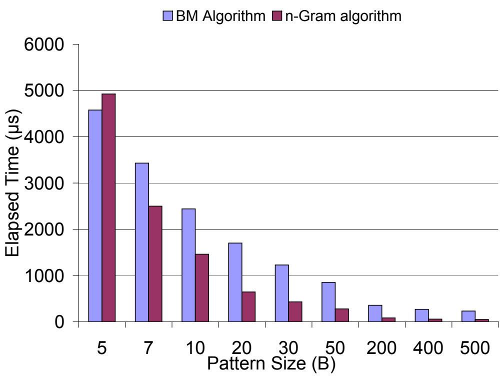


**Figure 16: Search time for XML**

# 7. ELIMINATION OF FALSE POSITIVES

Use of signatures is prone to *collisions,* which occur when two different strings have the same signature. We show in this section how we can avoid collisions for 2-grams for ASCII records encoded in Unicode and for 4-grams in DNA records. Even if this is not possible, we show how to eliminate almost or completely false positives in our pattern matching using the encoded form of records only.

## 7.1 Unicode

We recall a collision occurs if two different $n$-grams have the same AS. Collisions obviously slow the search speed as they decrease shift sizes and lead to additional verification work. We recall some facts from the theory of the algebraic signatures in [12] that indicate the relative absence of collisions for two interesting cases. First, if two $n$-grams differ by only one symbol, then the probability of the collision is exactly zero (the algebraic signatures were the first signature scheme known for this property but we do not deal with the more general formulation for $l$-symbol signatures). Second, 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

| $n$ | Maximum number of $n$-gram signatures |
|---|---|
| 1 | 4 |
| 2 | 16 |
| 3 | 64 |
| 4 | 256 |
| $\geq 4$ | 256 |

**Table 5: Maximum number of signatures for DNA**

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 made up of Unicode-embedded ASCII characters is zero.

## 7.2 DNA

In the case of DNA, the character set is the alphabet $\Sigma = \{A, C, G, T\}$. Table 5 gives the number of possible $n$-gram signatures for $GF(2^8)$. Since the possible number of $n$-grams is fixed at $4^n$, an encoding that has the maximum $n$-gram signatures is one that has the least collisions. In our case, such a scheme is collision-free for $n \leq 4$. The question arises whether one can find a combination of encoding, GF, and $\alpha$ that realizes the maximum number of signatures. We now give a general construction for such an encoding of DNA records.

Galois field elements are bit strings of length $f$ and we identify them as polynomials 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 $\alpha$. Then, multiplication by $\alpha$ 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 different. With this encoding, we have reduced the probability of $n$-gram signature collision for $n \leq 4$ to zero.

### 7.3 Verification of Possible Matches

In the description of our two algorithm variants, we ran into the problem that our algorithm only yields likely matches of the pattern in the record. We proposed there to use a final verification step based on a character by character match of the decoded record and pattern. For many data sets like text records, a likely match is almost always a true match since collisions are much less likely than for random data. If this is the case, then we might not bother with further verification using encoded records.

If we still want verification and if we know that there are no collisions for $n$-gram signatures, then we can verify by tiling the pattern (and the corresponding part of the record) into $n$-grams and verify these signatures. For instance, for $n = 4$ and $K = 10$, we compare the signatures of $p_7, p_8, p_9, p_{10}$, $p_3, p_4, p_5, p_6$, and $p_1, p_2, p_3, p_4$ in the pattern with the signatures of the corresponding 4-grams in the record. This involves $K/n$ signature comparisons and works for Full and Partial CAS Encoding. In the distributed case, this now happens at the server.

Assume now that we cannot exclude collisions among $n$-gram signatures. We show that we can verify a match for sure if all $K - n + 1$ $n$-gram signatures and the last $n - 1$ characters coincide.

**Proof:** We denote the pattern by $P = p_1, p_2, \dots p_K$ and assume it matches the substring $= a_1, a_2, \dots a_K$ of the record in this manner. This translates into a system of $K - n + 1$ linear equations in $a_i$

$$
\begin{aligned}
\sum_{\nu=1}^{n} \alpha^{\nu} p_{\nu} &= \sum_{\nu=1}^{n} \alpha^{\nu} a_{\nu} \\
\sum_{\nu=1}^{n} \alpha^{\nu} p_{\nu+1} &= \sum_{\nu=1}^{n} \alpha^{\nu} a_{\nu+1} \\
&\dots \\
\sum_{\nu=1}^{n} \alpha^{\nu} p_{\nu+K-n} &= \sum_{\nu=1}^{n} \alpha^{\nu} a_{\nu+K-n}
\end{aligned}
$$

Additional $n - 1$ equations state the equalities $p_{K-n+1} = a_{K-n+1}$, $\dots$ $p_K = a_K$. The $K$ by $K$ coefficient matrix of this system of equations has the form

$$
\begin{pmatrix}
\alpha & \alpha^2 & \dots & \alpha^n & 0 & 0 & \dots & 0 \\
0 & \alpha & \dots & \alpha^{n-1} & \alpha^n & 0 & \dots & 0 \\
0 & 0 & \alpha & \dots & \dots & \alpha^n & & 0 \\
0 & 0 & 0 & \ddots & & & & \\
0 & 0 & \dots & 0 & \alpha & \alpha^2 & \dots & \alpha^n \\
0 & 0 & \dots & 0 & 0 & 1 & & 0 \\
0 & 0 & \dots & 0 & 0 & 0 & \ddots & 0 \\
0 & 0 & \dots & 0 & 0 & 0 & & 1
\end{pmatrix}
$$

This upper-triangular matrix has rank $K$. Therefore, if a possible match passes all our $K$ tests, then it has to be a true match. **q.e.d.**

In the distributed setting, it is important to limit false positives as much as possible at the server. Otherwise, a client might receive a large number of records to verify and decode. If records are encoded in full CAS form, then we can perform all of our $K$ tests at the server and avoid sending false positives altogether. This presupposes that the server can calculate the characters, which in turn requires that the server can divide by powers of $\alpha$. If records are encoded in partial CAS form, we can only perform the $K - n + 1$ signature comparisons at the server without decoding the whole record at the server side. Thus, our observation can no further improve on the algorithm presented in Section 5.

As a further variant, we state without proof another possibility of verification using full CAS for encoding records. For this, the client sends the full CAS form of the pattern. When verifying a potential match, the server calculates the full CAS encoding of that portion of the record using the algebraic properties of signatures. If a character by character comparison of these encodings coincide, then the match is a true one.

## 8. RELATED WORK

In the general computer science literature, pattern matching is among the fundamental problems with many prominent contributions [4]. The popular algorithms do not attempt to encode records, because they were not designed for our "write once read many" database context. BM is accepted as a very versatile search method that often outperforms all other prominent pattern matching methods. For this reason, we compared our method to BM. Our method with $n = 1$ and partial CAS has the same shifts as BM with only "bad character" shifts or Quick Search [4].

Our method falls into the large class of algorithms for string search without indexing. Two state-of-the-art indexing techniques offer an attractive alternative to our work, namely suffix trees [9, 13] and $n$-gram indices [14], both implementing an inverted file. Both methods require several times more space for the index than for the text itself, namely about 20 times for the basic suffix tree, 4-5 times for a suffix array and 2-5 times for a compressed or two-level $n$-gram index. However, search speed is now $O(N)$, with $N$ the length of the pattern, for the suffix trees while the storage optimizations employed in advanced implementations add to the search time.

Our SDDS-2005 system already manages SDDS files in distributed RAM, with records encoded into their full CAS form [11]. 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, which uses a Karp-Rabin like sequential traversal [8, 5]. The algorithm avoids pattern preprocessing other than the calculation of algebraic signatures and the creation of the shift table. For a match, only the result of pre-processing the pattern, but not the pattern itself is send by the client to the servers. Sending and storing data not in the clear can be advantageous to security. Our efforts fall into the recent general direction of the Database As Service (DAS) model [7].

Our encoding of records provides a simple social and legal protection of the stored data and of data in transit. However, a simple frequency attack can determine $\alpha$ and hence decode both if the attacker has some knowledge about the data set. 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 by Bawa et al. [1] and by Chang et al. [3] uses keyword indexes to describe the contents of files. Song et al. [16] 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. [15] propose a different method that trades search capability for much less security.

# 9. CONCLUSION

We have presented a novel search algorithm that is attractive in any scenario where records are inserted once and searched often. Our algorithm uses $n$-gram signatures combined with a sublinear traversal of the record similar to the algorithm by Boyer and Moore. Signatures contain more information than single characters and our method tends to outperform known string search algorithms, in particular Boyer and Moore, with which we compared our method experimentally.

Compared to indexing, our method does not have any storage overhead and the costs of record insertion are very small, but we cannot rival search times. Compared with traditional pattern matching algorithms, we have to encode records, but have almost always better and often much better search times.

Future works includes more complete performance studies including for instance a study of likely match verification at the server and collision probabilities in actual data sets. There is also the possibility of further optimizations such as implementing the equivalence of the Boyer Moore "good shift" for partial CAS or a different organization of the shift table for larger character sets such as Unicode. A particularly promising direction lies in the implementation of approximate matches.

**Acknowledgments**. Support for this work came from EGov IST project FP6-IST-2004-26727. We would like to thank Christopher Jermaine and JoAnne Holliday for their generous help in presenting our work. We are also grateful to anonymous reviewers for helpful comments.

# 10. REFERENCES

[1] M. Bawa, R. Bayardo, and R. Agrawal. Privacy Preserving Indexing of Documents on the Network. In *Proc. Intl. Conf. on Very Large Databases (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 *Proc. 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 *Proc. 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] M.-S. Kim, K.-Y. Whang, J.-G. Lee, and M.-J. Lee. n-Gram/2L: A Space and Time Efficient Two-Level n-Gram Inverted Index Structure. In *Proc. Intl. Conf. on Very Large Databases (VLDB)*, pages 325–336, 2005.

[10] 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.

[11] 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.

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

[13] U. Manber and G. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935, 1993.

[14] E. Miller, D. Shen, J. Liu, and C. Nicholas. Performance and Scalability of a Large-Scale N-gram Based Information Retrieval System. *Journal of Digital Information*, 2000.

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

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