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

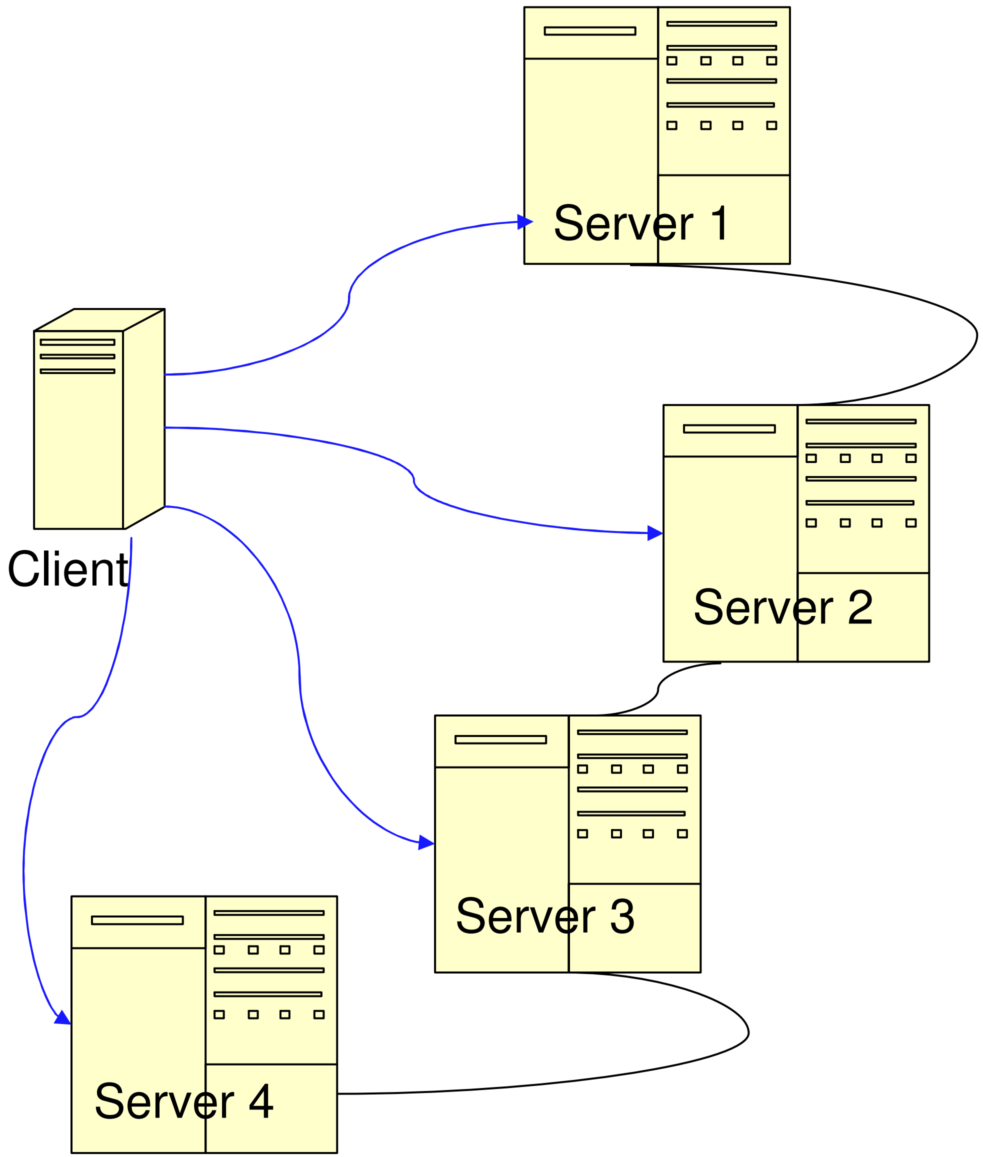
W. Litwin (Dauphine),
R. Mokadem (Dauphine),
Ph. Rigaux (Dauphine)
T. Schwarz (U. Santa Clara)

Plan

- Problem Statement
- Our Proposal
- Key Idea
  - Algebraic Signatures
  - Record Encoding
  - Pattern Preprocessing
- Search Example
- Performance Study
- Conclusion

Problem

- String Search (Pattern Matching) in A Database or File
  - Find every record matching pattern = “Dauphine”
  - What about record “Universite de Technologie Paris Dauphine” ?
- Records are searched often, and updated rarely
  - We especially target large Scalable and Distributed DBs and Files
  - on Grids and P2P networks

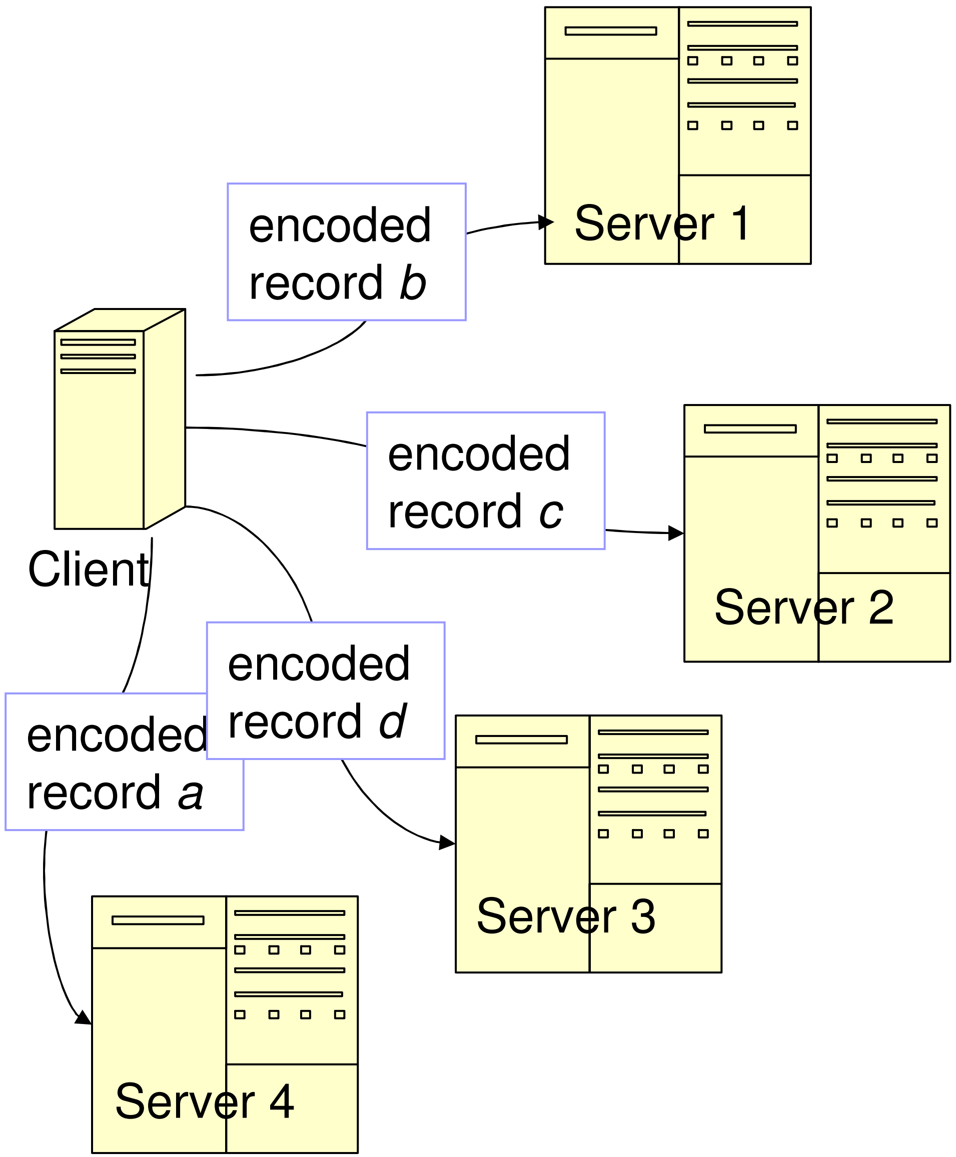

Server 1
Client
Server 2
Server 3
Server 4

Our Proposal

- Fast String Search Method
  - Several Times Faster than Boyer-Moore
    - In our experiments:
      - Up to eleven times for ASCII
      - Up to six times for XML
      - Up to seventy times for DNA

Key Idea : Pre-processing

- We aggregate (encode) all *n*-symbol long substrings (*ngrams*) in visited strings (*records*) and in the searched pattern into single-symbol *algebraic signatures*
  - Records are encoded while coming for storage
  - Pattern is encoded during search pre-processing


encoded
record *b*
Server 1
Client
encoded
record *c*
Server 2
encoded
record *d*
encoded
record *a*
Server 3
Server 4

Key Idea : Search

- We compare signatures for attempted matches and shifts like Boyer-Moore (BM) does
  - “Bad character” shift
- However, matching *n*gram signatures ⇔ matching *n* symbols at the time

Key Benefit

- Matching attempts usually more discriminative than matching a single (original) symbol at the time.
  - The latter is the current approach
    - BM and all other major pattern matching algorithms we are aware of
      - KMP, Quick Search, KR…

Key Benefit

- Longer shifts
- Fewer comparisons
- Faster search
- Local search over encoded data only
- No local user can claim unintentional disclosure of stored data
  - Important for P2P
  - Thought determined fraud is not that difficult
- Idem for the data transfer to the client

Algebraic Signature

ICDE 2004

- Condenses information in a string into a single character
- Defined over Galois Fields (GF) of size 2^f
  - Elements are bit strings of length f
  - In our case, typically $f = 8$
  - Hence our symbols are bytes
  - We realize GF addition $\oplus$ as XOR
  - We realize GF multiplication through log/antilog tables

Algebraic Signature

$$AS(r_1 \ldots r_k) = r_1\alpha \oplus r_2\alpha^2 \oplus \cdots \oplus r_k\alpha^k$$

$\Rightarrow$ α is a primitive element, e.g., $\alpha = 2$

$\Rightarrow$ if $AS(R_1) \neq AS(R_2)$ then $R_1 \neq R_2$ for sure

$\Rightarrow$ if $AS(R_1) = AS(R_2)$ then for sure or very likely $R_1 = R_2$

- The latter case is a *collision*

Record Encoding

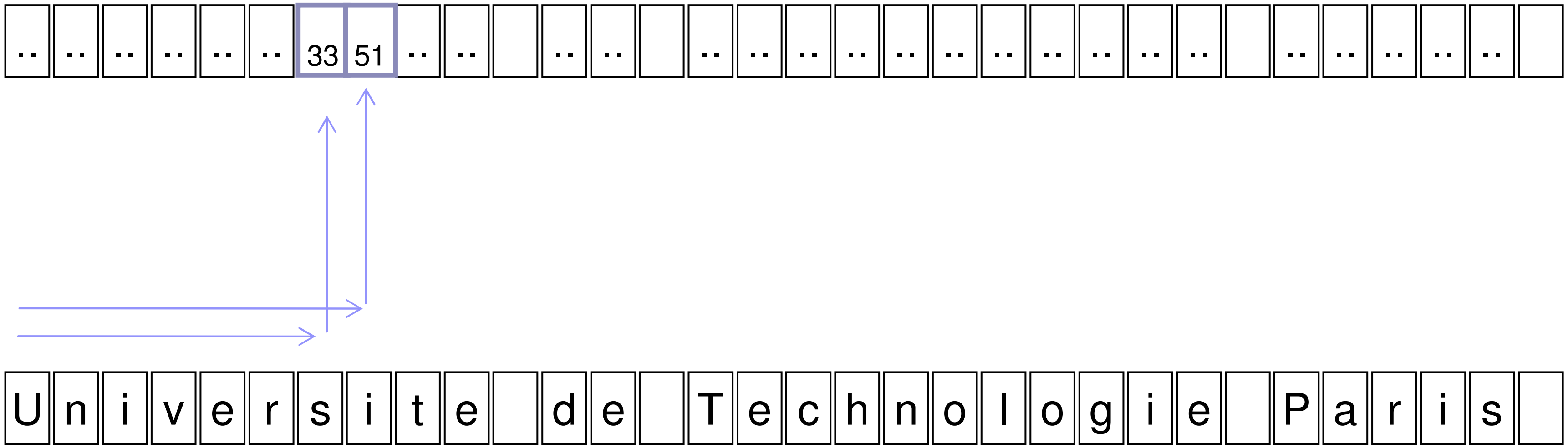
- We encode every stored record : $r_1 \ldots r_K$
  - Either into full *Cumulative Algebraic Signature*

$$r'_k = r_1\alpha \oplus r_2\alpha^2 \oplus \cdot\cdot\cdot \oplus r_k\alpha^k$$

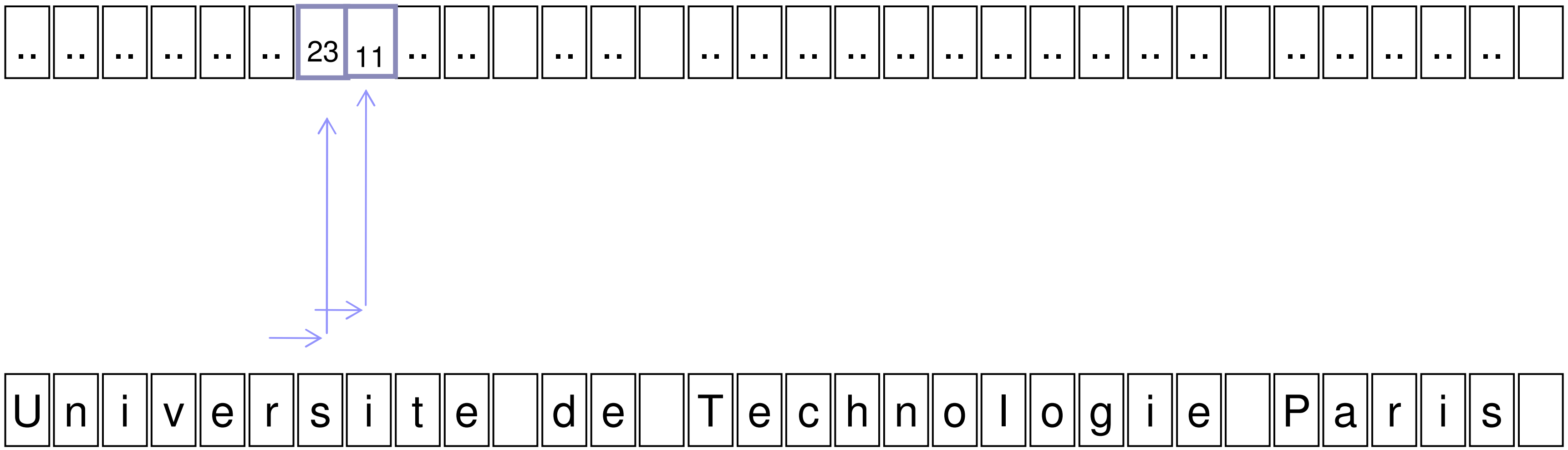
  - Or into partial (moving) CAS of ngrams

$$r'_k = r_{k-n+1}\,\alpha \oplus \cdot\cdot\cdot \oplus r_k\alpha^n$$

Full CAS



Partial CAS for $n = 2$



- Partial CAS can be stored or dynamically calculated from full CAS
  - See the paper

Pattern Preprocessing

- We aggregate ngram signatures in the pattern in a BM-like shift table T
- Conceptual result for "Dauphine"
- Actually:
  - shift table size is *f* and entry is by AS value
  - Rightmost ngram value is in variable *V*

2-gram	Shift
33 = AS(da)	6
23 = AS(au)	5
133 = AS(up)	4
24 = AS(ph)	3
07 = AS(hi)	2
62 = AS(in)	1
67 = AS(ne)	0
Any other digram	7

N-Gram Search by Example

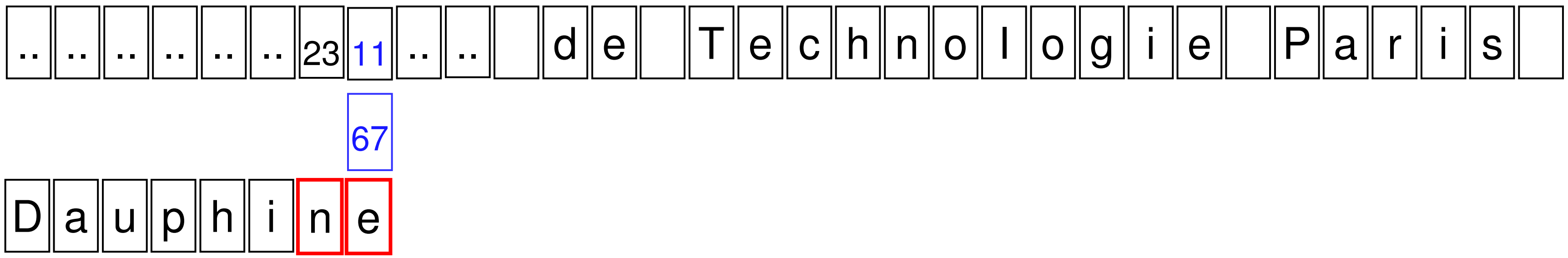
- Pattern = “Dauphine” of length $l = 8$
- Record = “Universite de Technologie Paris Dauphine”
- $n = 2$

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		
D	a	u	p	h	i	n	e																									

- Attempt to match the rightmost 2-gram of pattern against the visited 2-gram in the record
  - AS(ne) =? AS(si) at offset of “i”

N-Gram Search by Example

- Pattern = “Dauphine” of length *l* = 8
- Record = “Universite de Technologie Paris Dauphine”
- *n* = 2



- 67 =? 11
- No
- Lookup shift table T at offset 11 = (AS(si))
  - T shows shift of 7 symbols since AS(si) is not in “Dauphine”
  - Maximal shift here
    - Equal in general to $l - n + 1$

N-Gram Search by Example

- **N-Gram Search**: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s	
							D	a	u	p	h	i	n	e																	

- AS(ne) =? AS(T)
- Mismatch
- What in element AS(T) in table T ?
- Maximal shift by 7
  - Since “ T” is nowhere in “Dauphine”

N-Gram Search by Example

- **N-Gram Search**: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s	
														D	a	u	p	h	i	n	e										

- Idem
- Mismatch
- Shift by 7
  - Again maximal shift since ‘lo’ not in “Dauphine”

N-Gram Search by Example

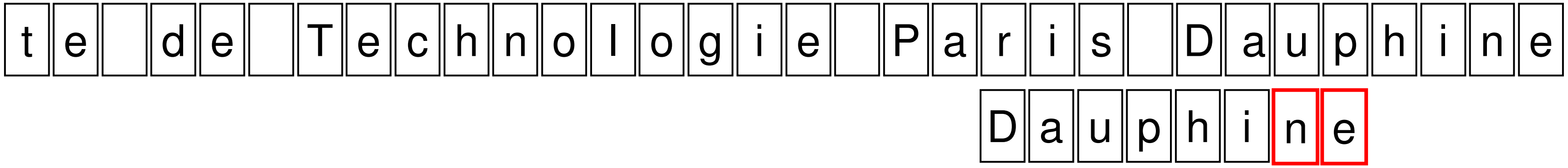
- **N-Gram Search**: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		D	a	u	p	h	i	n	e
													D	a	u	p	h	i	n	e											

  - Idem
  - Mismatch
  - Shift by 7
    - Maximal shift since ‘ar’ not in “Dauphine”

N-Gram Search by Example

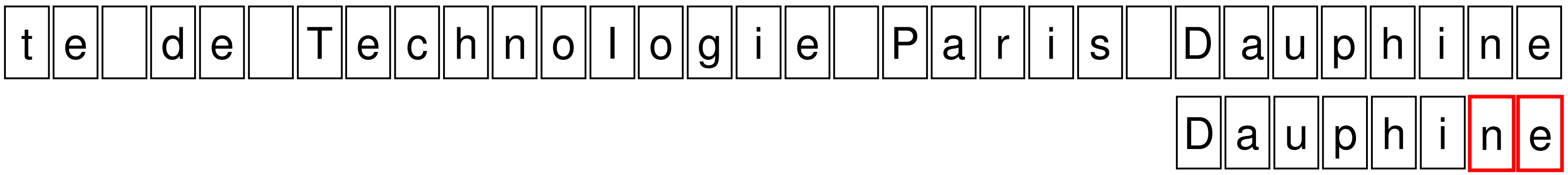
- **N-Gram Search**: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:



  - Compare by signature digrams “ne” and “up”
  - Mismatch
  - shift by 4 according to T
    - To align on ‘up’ in “Dauphine”

N-Gram Search by Example

- **N-Gram Search**: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:



- Match ‘ne’ and ‘ne’, ‘hi’ and ‘hi’, ‘up’ against ‘up’, ‘Da’ and ‘Da’
- Full match

N-Gram Search by Example

- **N-Gram Search**: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		D	a	u	p	h	i	n	e
																								D	a	u	p	h	i	n	e

- Test for false positive : full CAS
  - Compare all the matching symbols at the server
- No test if ngram signatures never collide
  - e.g., through the method proposed for DNA in the paper

N-Gram Search by Example

- **N-Gram Search**: Looking for "Dauphine" in "Universite de Technologie Paris Dauphine:

t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		D	a	u	p	h	i	n	e
																								D	a	u	p	h	i	n	e

- Test for false positive : partial CAS
  - Compare matching symbols at the server except for AS(D) in the record
  - Match D after decoding at the client
    - Remaining $n-1$ leftmost symbols in general
- No test if ngram signatures never collide
  - e.g., through the method proposed for DNA in the paper

BM Search by Example

- Match attempts and shifts compare single symbol at the time

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s	
D	a	u	p	h	i	n	e																								

- Compare right-most character
- Mismatch, hence move Dauphine 2 slots to the right where ‘i’ appears in Dauphine

BM Search Example

- BM: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s	
		D	a	u	p	h	i	n	e																						

- Compare right-most character
- Match, hence compare next character
- Mismatch, hence move Dauphine 7 slots to the right since ‘e’ appears only once in Dauphine

BM Search Example

- BM: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s	
										D	a	u	p	h	i	n	e														

- Compare ‘h’ against ‘e’
- Mismatch, move pattern three to the right

BM Search Example

- BM: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

U	n	i	v	e	r	s	i	t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s	
													D	a	u	p	h	i	n	e											

- Compare ‘l’ against ‘e’
- No ‘l’ in Dauphine, move by 8

BM Search Example

- BM: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		D	a	u	p	h	i	n	e
													D	a	u	p	h	i	n	e											

- No ‘r’ in Dauphine, move by 8

BM Search Example

- BM: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		D	a	u	p	h	i	n	e
																				D	a	u	p	h	i	n	e				

- There is a ‘p’ in Dauphine, move by 5

BM Search Example

- BM: Looking for “Dauphine” in “Universite de Technologie Paris Dauphine:

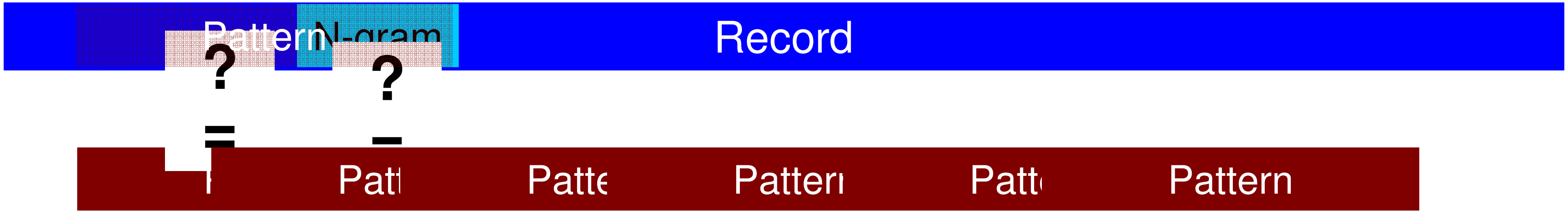
t	e		d	e		T	e	c	h	n	o	l	o	g	i	e		P	a	r	i	s		D	a	u	p	h	i	n	e
																								D	a	u	p	h	i	n	e

- Compare ‘e’ against ‘e’, then ‘n’ against ‘n’, …
- A match

Comparison

- 2-gram search has fewer shifts (6 vs 8)
- The shifts are on average longer
- Even though maximum shift size for 2-gram is here only 7 vs. 8 for BM
- Much larger gain to expect for larger patterns

N-gram Search in Nutshell



- Get N-gram in record
- Compare with V
  - the last N-gram in pattern
- If equal, check whether this is a full match
- If not, use shift table
- Repeat until done

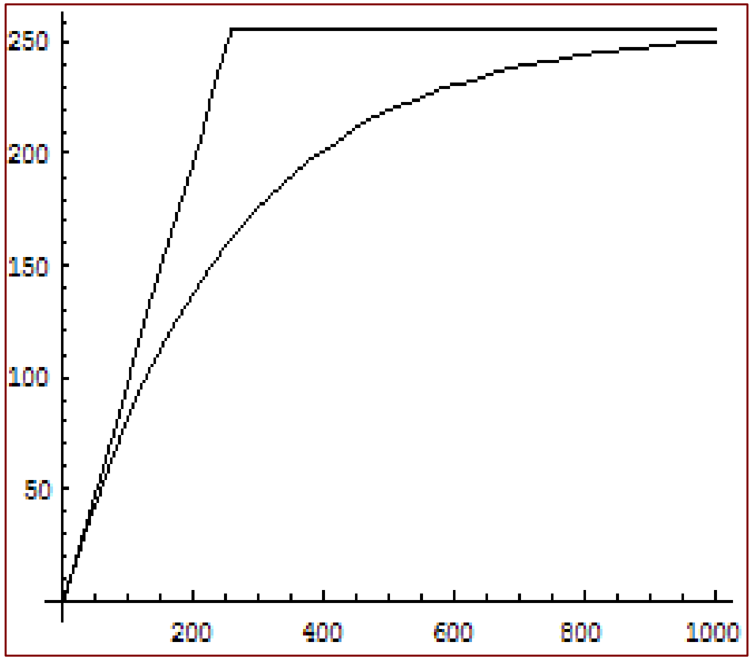
Performance

- Zero Storage Overhead
  - No indexing
  - Like BM, KMP…
  - Unlike suffix trees and arrays or ngram indexes…
- Search cost is O(*s*), *s* the number of shifts
  - Maximal shift size is $l - n + 1$
  - Expected shift size converges towards *f*
    - Galois Field size used for CAS calculus

Performance

- Depends on tuning of *n*
  - Larger *n* decreases the maximum shift
  - But makes ngrams more discriminative
  - Up to some value of *n*
    - depending on the alphabet size, symbol value distribution…
- Our experiments show:
  - N=4 for DNA records
  - N=2 for ASCII & XML in natural language text

Analytical Calculus



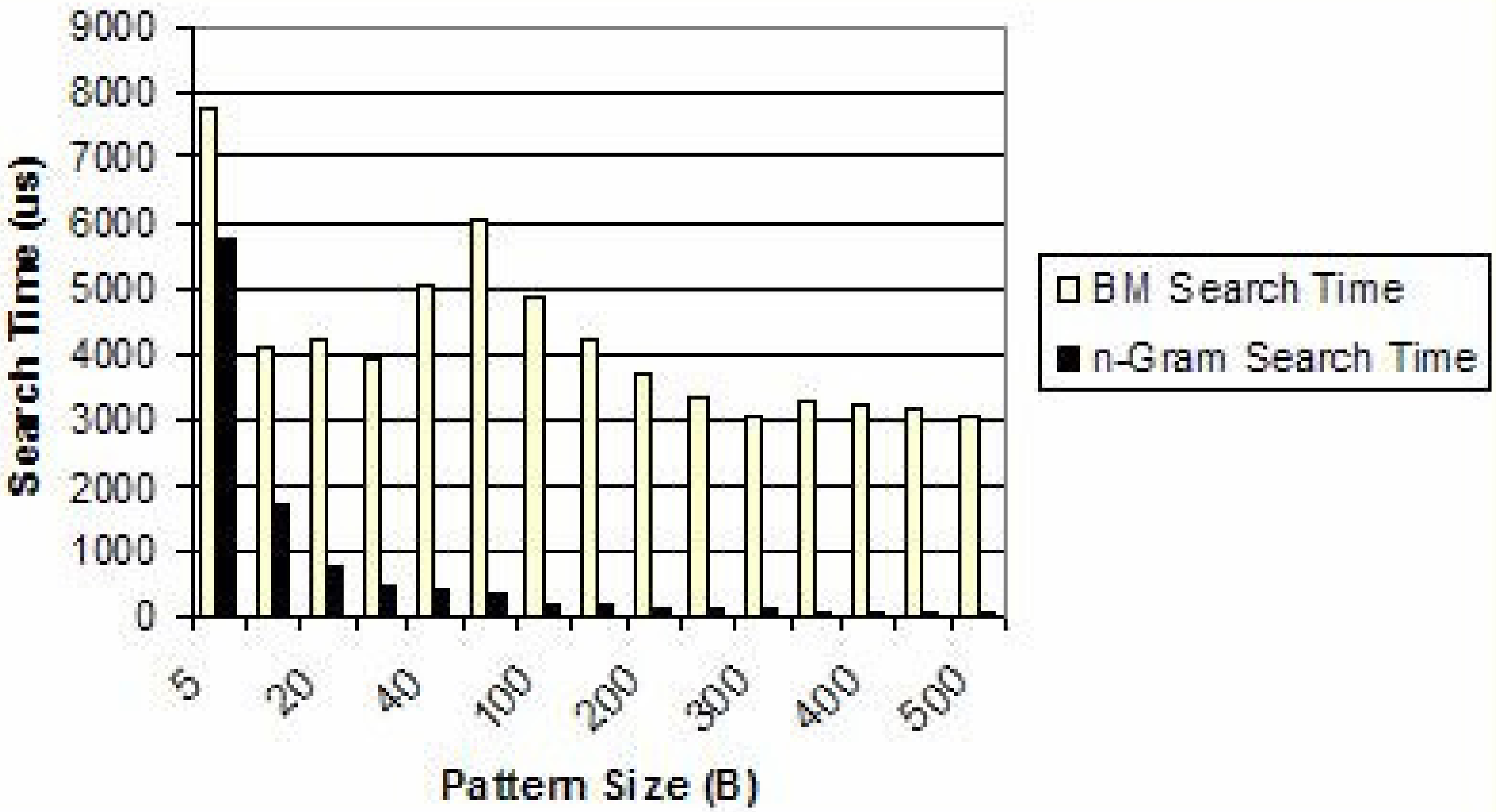
Expected Shift Size for 4-gram search on DNA

- Random distribution of symbol values

Experiments

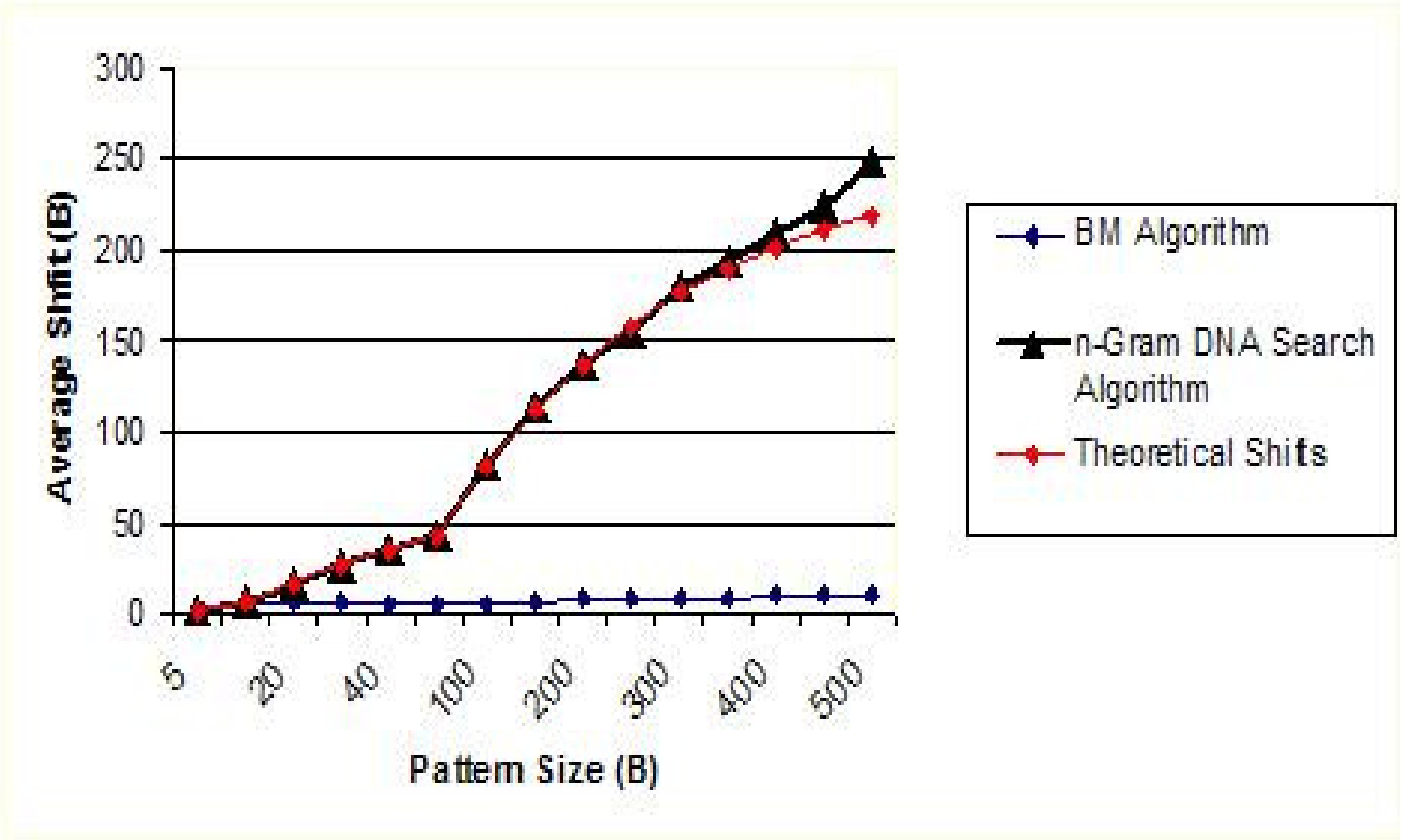
- We compare experimentally performance of N-gram search with BM
- We use mostly partial CAS encoding for:
  - DNA
  - ASCII natural language text
  - XML code

Experiments: DNA (homo sap.)



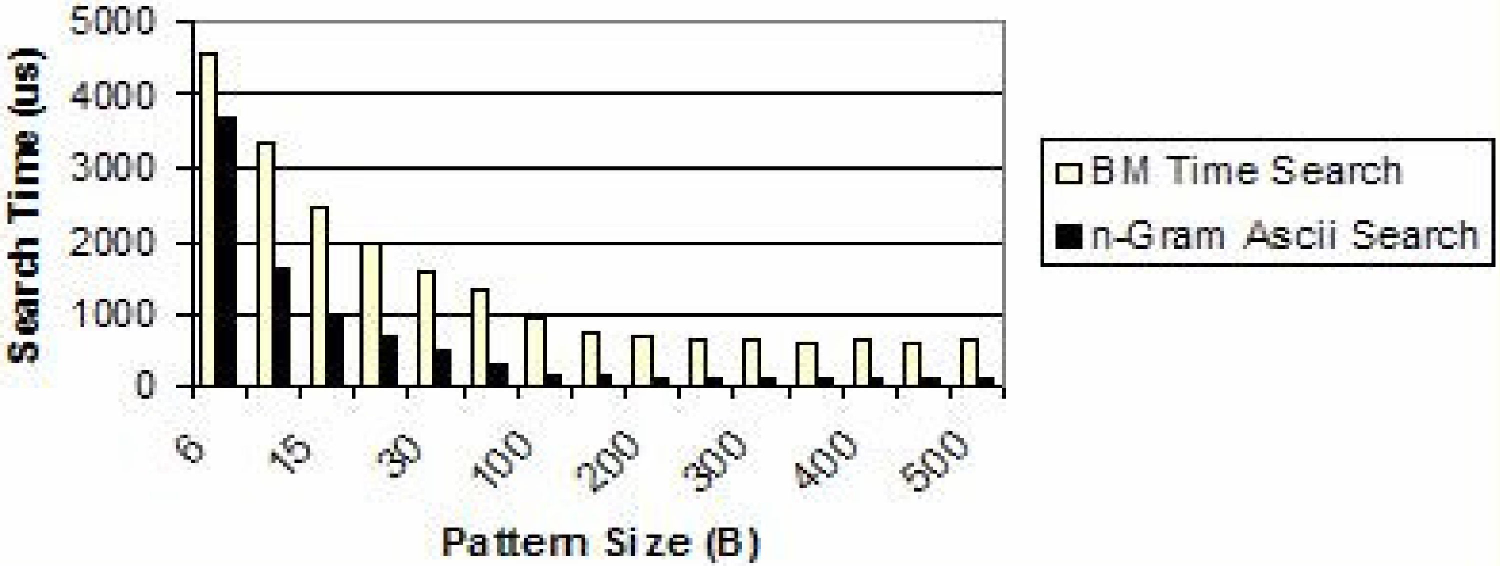
Search Times

Experiments: DNA (homo sap.)

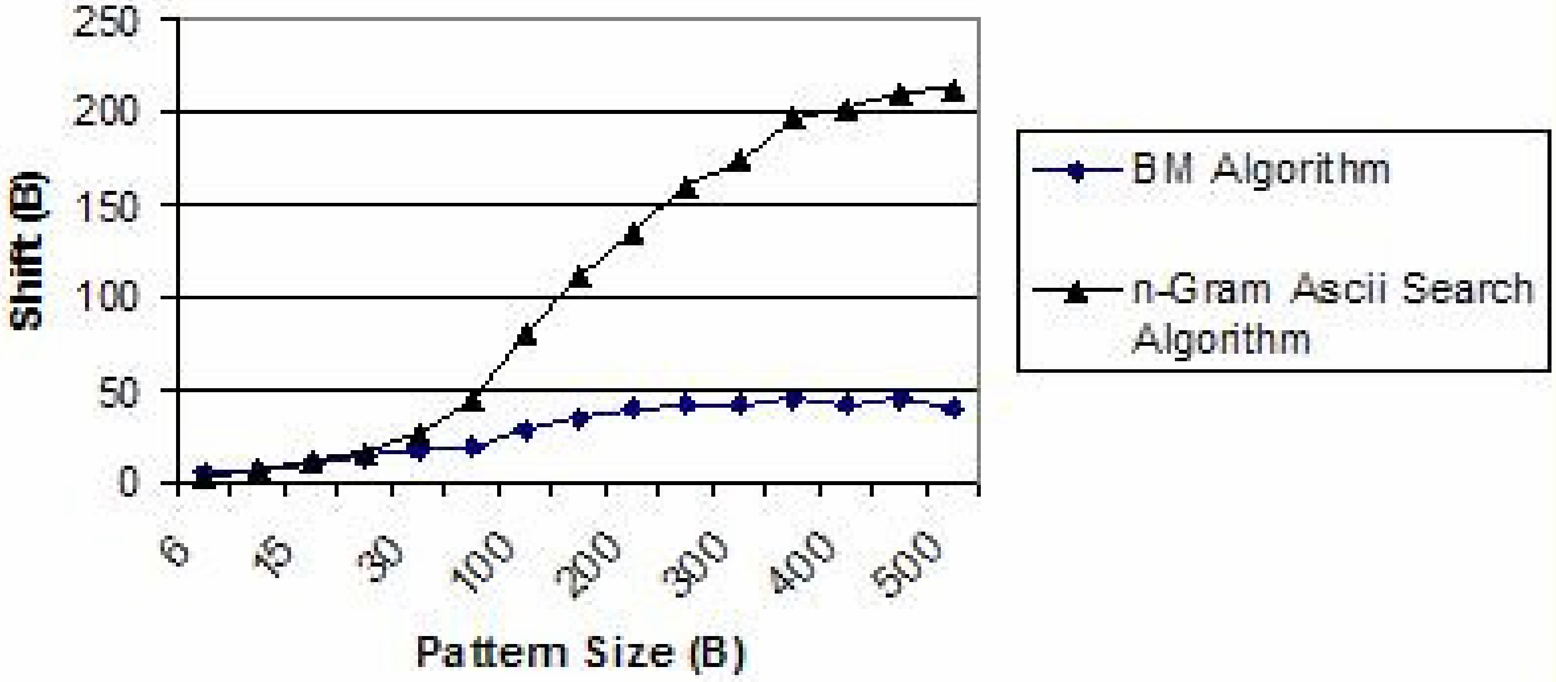


Shifts

Experiments (ASCII nat. lang.)



Experiments (ASCII nat. lang.)



Conclusion

- A new algorithm suitable for data stored once and read many times
  - At least as fast as the most used pattern-matching technique (Boyer-Moore);
  - Much faster for small alphabets and/or large patterns;
  - Search without decoding is valuable for P2Pn and Grid environment.
- Current work on:
  - Approximate string matching
  - Multiple pattern matching
  - Stronger privacy preservation

Thank You

for

Your Attention