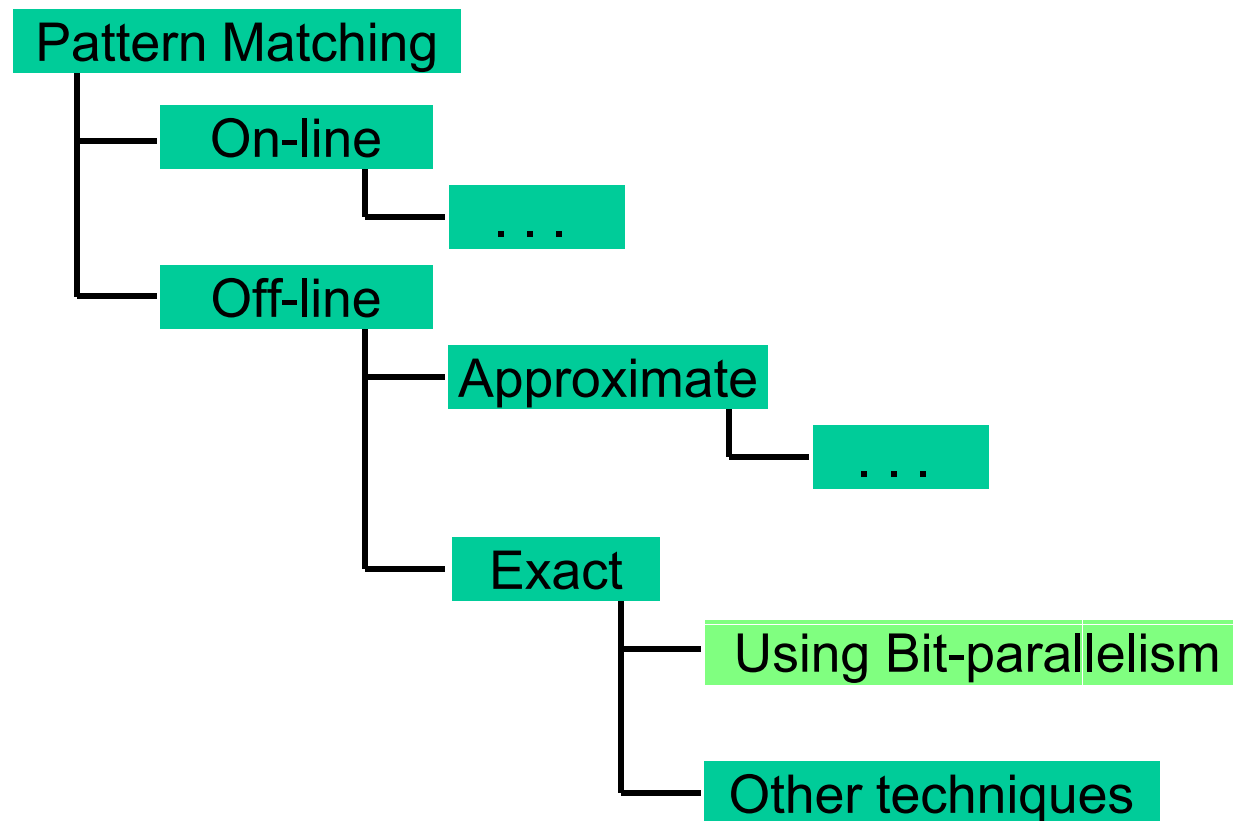


Overcoming Computer Word Size Limitation in Bit-parallel Pattern Matching

M. Oğuzhan Külekci
TÜBİTAK-UEKAE
National Research Institute of
Electronics & Cryptology, Turkey

kulekci@uekae.tubitak.gov.tr
www.busillis.com/o_kulekci

The area of research



Bit-parallelism ?

- Computers perform bitwise operations very fast.
- Designing algorithms that benefit from that intrinsic property of processors

Previous bit-parallel pattern matching algorithms

- Shift-or algorithm
 - Shift-or (SO), (*Baeza-Yates&Gonnet,1992*)
 - Fast (FSO), Average optimal (AOSO), and Fast AOSO (FAOSO), (*Fredriksson&Grabowski,2005*)
- BNDM algorithm
 - Actually (BDM + SO) → BNDM (*Navarro&Raffinot,2000*)
 - SBNDM (*Peltola&Tarhio,2003*)
 - SBNDM2 (*Holub&Durian,2005*)

Problems in Bit-parallel Pattern Matching

- Lack of shift mechanism (in original idea)
 - BNDM solved it.
 - Recent SO variants (AOSO, FAOSO) also include shift mechanisms.
- **Patterns are required to be no longer than the computer word size !**

What causes that limitation?

- The way that the bits are used!
- In previous approaches, each bit marks the position of a character in the pattern.
- If pattern is longer than the computer word size, more words are needed.
⇒ **significant drop in efficiency**

Bitmasks in previous algorithms

- Mask creation in BNDM (SO is also similar) :

```
unsigned long B[ALPHABET_SIZE];  
for (a ∈ Σ) B[a] = 0;  
for j=1..m B[pj] = B[pj] | (1<<(m-j));
```

- Bits in mask **B[c]** express the location of character **c** in the pattern,

- e.g. For pattern P = abaab

```
B[a] = 0....10110  
B[b] = 0....01001
```

How to overcome ?

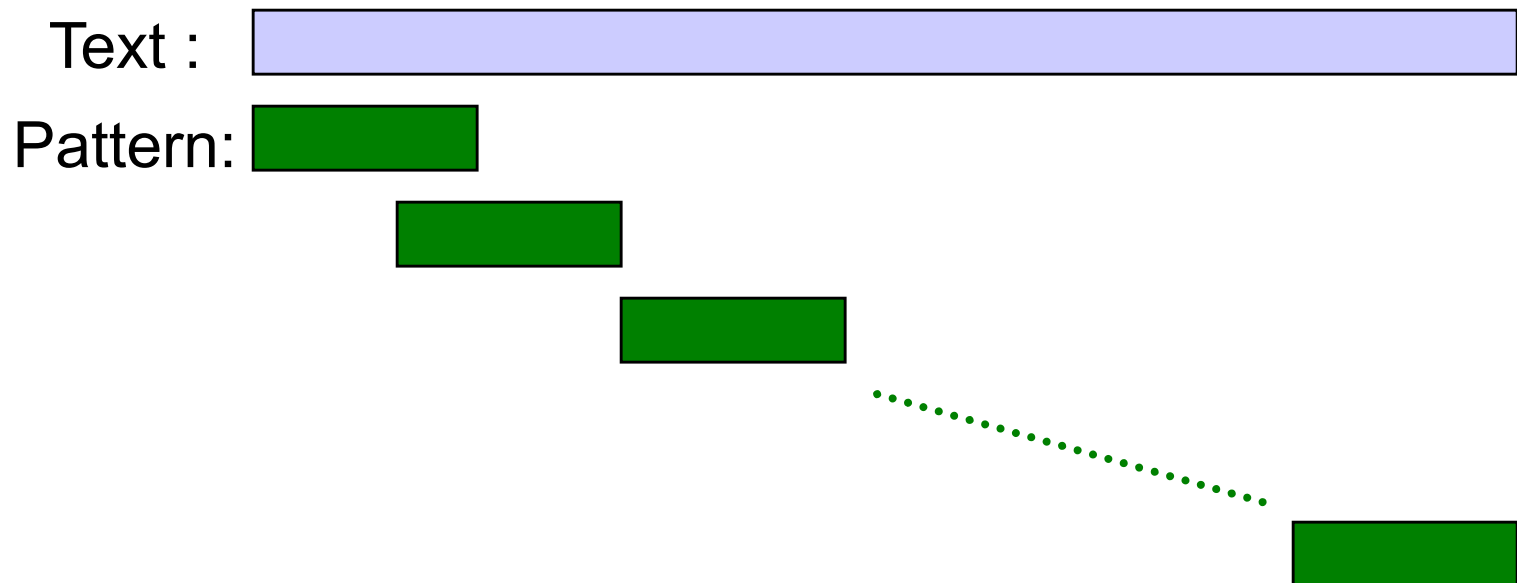
- Load a different information, *which is length independent*, to a single bit.
- Each bit carries information about the whole pattern right shifted some amount in the proposed *bit-parallel length independent matching* (BLIM).

Basic notation

- Text $\mathbf{T} = t_0 t_1 t_2 t_3 \dots t_{n-1}$
- Pattern $\mathbf{P} = p_0 p_1 \dots p_{m-1}$
- Computer Word Size \mathbf{W}
- Σ denotes alphabet

Off-line Pattern Matching (in general...)

- Slide a window over the text
- Check & Shift



Sliding window

	0	1	2	3	$W-m+2$			
0	p_0	p_1	p_2	P_{m-1}				
1		p_0	p_1	p_2	P_{m-1}			
2			p_0	p_1	p_2	P_{m-1}		
⋮									
$W-1$					p_0	p_1	p_2	P_{m-1}

- The window that is to be slid over T .
- W rows, $ws = W+m-1$ columns
- i^{th} row contains i character right shifted P .

Sliding window

P = abaab
W = 8

	0	1	2	3	4	5	6	7	8	9	10	11
0	a	b	a	a	b							
1		a	b	a	a	b						
2			a	b	a	a	b					
3				a	b	a	a	b				
4					a	b	a	a	b			
5						a	b	a	a	b		
6							a	b	a	a	b	
7								a	b	a	a	b

Bitmask Creation

	t_j	t_{j+1}	t_{j+2}	t_{j+3}	t_{j+4}	t_{j+5}	t_{j+6}	t_{j+7}	t_{j+8}	t_{j+9}	t_{j+10}	t_{j+11}	
							b						
0	a	b	a	a	b								b ₀ =1
1		a	b	a	a	b							b ₁ =1
2			a	b	a	a	b						b ₂ =1
3				a	b	a	a	b					b ₃ =0
4					a	b	a	a	b				b ₄ =0
5						a	b	a	a	b			b ₅ =1
6							a	b	a	a	b		b ₆ =0
7								a	b	a	a	b	b ₇ =1
	0	1	2	3	4	5	6	7	8	9	10	11	

Mask[b][6] = 1 0 1 0 0 1 1 1 = A7

Bitmask

- Mask[ch][pos] is a bitvector of W bits as

$$b_{w-1}b_{w-2} \dots b_1b_0$$

where $ch \in \Sigma$, and $0 \leq pos \leq (W+m-1)$

- Bits denote which of the alignments in the investigation window are appropriate when one observes character ch at position pos .

Bitmask

- i^{th} bit of `Mask[ch][pos]` gathers info whether the i character right shifted placement of pattern matches with the observed `ch` at position `pos`.

$b_i = 0$, if $(0 \leq \text{pos}-i < m)$ and $(\text{ch} \neq p_{\text{pos}-i})$

$b_i = 1$, otherwise

Sample Bitmask

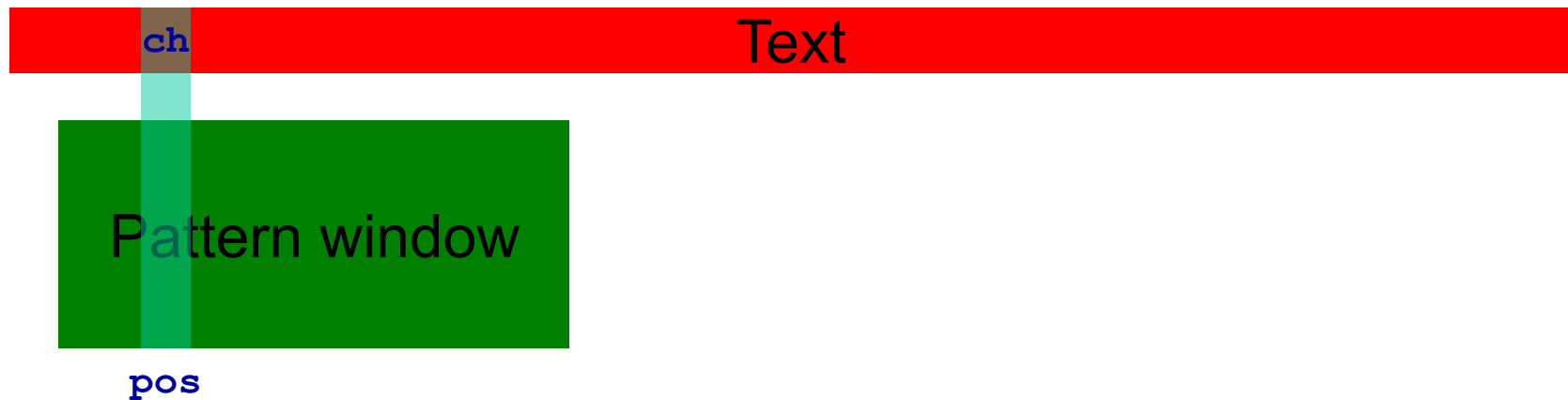
$P = \text{abaab}$, $W = 8$,
 $\Sigma = \{a,b,c,d\}$ (ch)
 $ws = W + m - 1 = 12$ (pos)
 $\text{Mask}[ch][pos] = \dots$

		<i>pos</i>											
		0	1	2	3	4	5	6	7	8	9	10	11
<i>ch</i>	a	FF	FE	FD	FB	F6	ED	DB	B7	6F	DF	BF	7F
	b	FE	FD	FA	F4	E9	D3	A7	4F	AF	3F	7F	FF
	c	FE	FC	F8	F0	E0	C1	83	87	8F	1F	3F	7F
	d	FE	FC	F8	F0	E0	C1	83	87	8F	1F	3F	7F

Up to now, we created the sliding window,
and the associated bitmasks.

How those masks are used for
matching followed by a shift procedure?

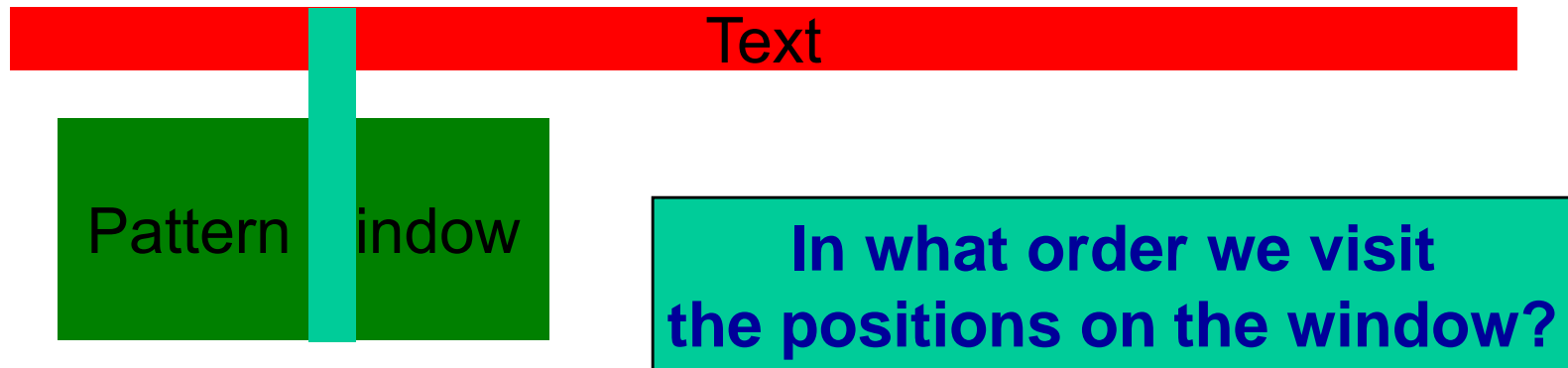
Checking...



W bits
flag = 1111 ... 1

flag = flag & Mask[ch][pos]

Checking...



- Continue until flag becomes zero or all the positions are visited.
- If all positions visited, there are some matches.
- The index of the bits that are 1 on the flag determine which of the alignments are observed.

Scan Order

- A heuristic approach to visit minimum number of characters in case of a mismatch

ScanOrder = { $m-i, 2m-i, \dots, km-i$ }

– **$i = 1, 2, \dots, m$**

– **$(km-i) < ws$ ($ws = W+m-1$)**

Scan Order

a	b	a	a	b	b	b	b	b	b	b	b
a	a	b	a	a	a	a	a	a	a	a	a
b	b	a	b	b	b	b	b	b	b	b	b
0	1	2	3	4	5	6	7	8	9	10	11

ScanOrder = 4, 9, 3, 8, 2, 7, 1, 6, 11, 0, 5, 10

Shifting...

Text

Pattern window

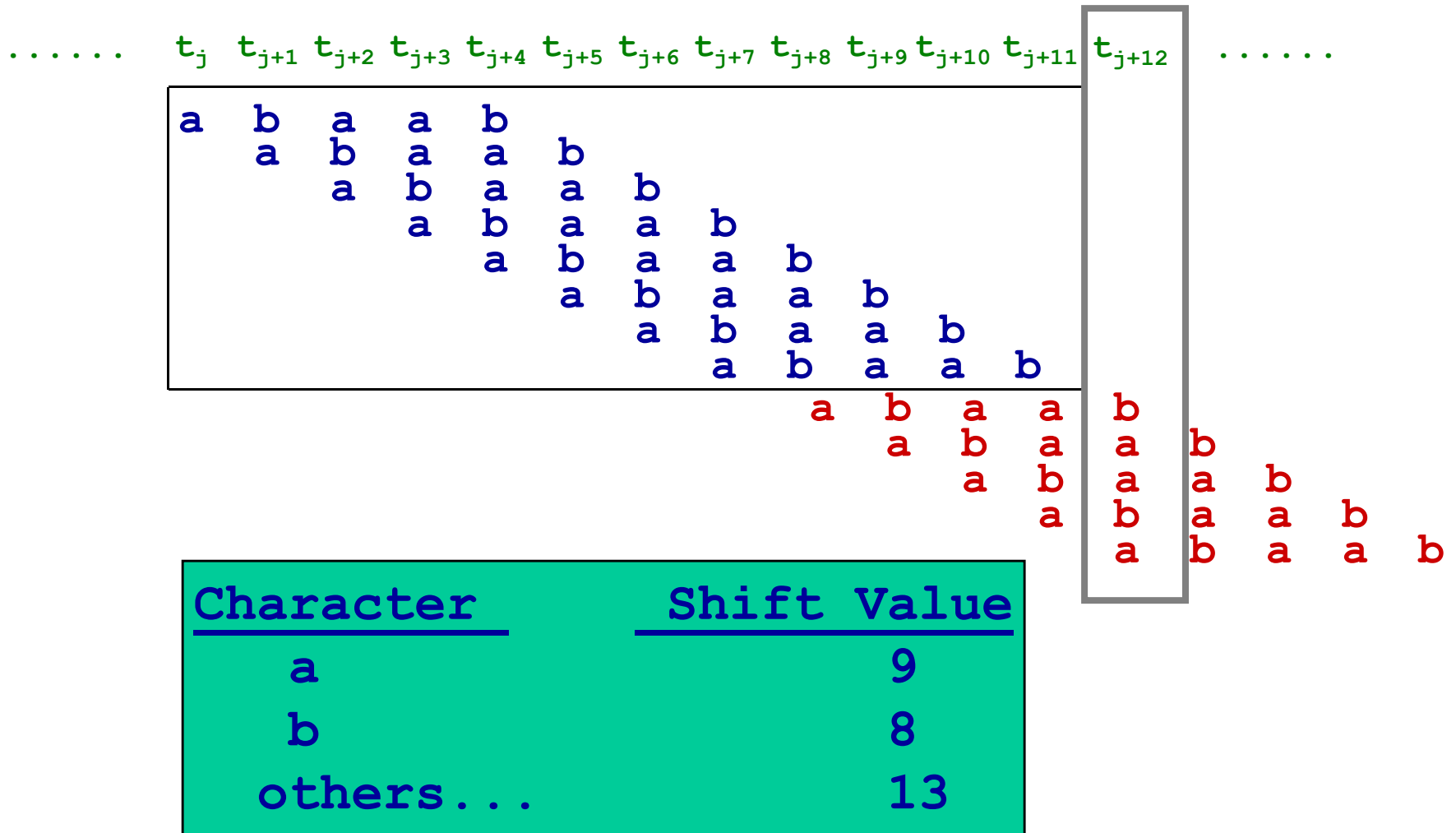
Pattern window

The amount of shift ?

Shift Mechanism

- Same as Sunday's quick search
- Move right according to the immediate text character succeeding the current window under investigation

Shift Mechanism

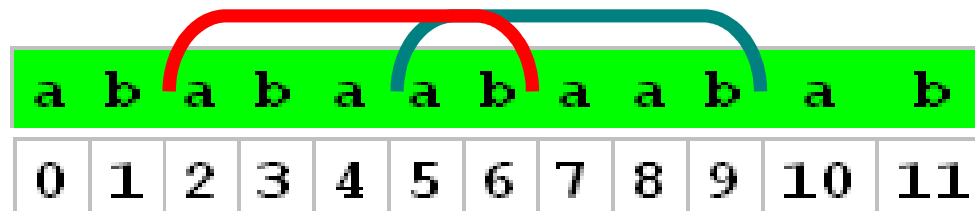


BLIM Algorithm

```
Ws = W+m-1;
Compute Mask;
Compute ScanOrder;
Compute Shift;
Pad text T with ws number of NULL characters;
i=0;
while(i<n){
    flag = Mask[T[i+ScanOrder[0]]][ScanOrder[0]];
    for(i=j;j<ws;j++){
        flag &= Mask[T[i+ScanOrder[j]]][ScanOrder[j]];
    }
    if (flag){
        Check bits of the flag to locate occurrences
    }
    i+=Shift[T[i+ws]];
}
```

Sample Run

0 1 2 3 4 5 6 7 8 9 10 11	position	Mask[ch] [pos]	flag
a b a b a a b a a b a b	ScanOrder [0] = 4	Mask[a] [4] = 11110110	11110110
a b a b a a b a a b a b	ScanOrder [1] = 9	Mask[b] [9] = 00111111	00110110
a b a b a a b a a b a b	ScanOrder [2] = 3	Mask[b] [3] = 11110100	00110100
a b a b a a b a a b a b	ScanOrder [3] = 8	Mask[a] [8] = 01101111	00100100
a b a b a a b a a b a b	ScanOrder [4] = 2	Mask[a] [2] = 11111101	00100100
a b a b a a b a a b a b	ScanOrder [5] = 7	Mask[a] [7] = 10110111	00100100
a b a b a a b a a b a b	ScanOrder [6] = 1	Mask[b] [1] = 11111101	00100100
a b a b a a b a a b a b	ScanOrder [7] = 6	Mask[b] [6] = 10100111	00100100
a b a b a a b a a b a b	ScanOrder [8] = 11	Mask[b] [11] = 11111111	00100100
a b a b a a b a a b a b	ScanOrder [9] = 0	Mask[a] [0] = 11111111	00100100
a b a b a a b a a b a b	ScanOrder [10] = 5	Mask[a] [5] = 11101101	00100100
a b a b a a b a a b a b	ScanOrder [11] = 1	Mask[b] [1] = 11111101	00100100



00100100
76543210

Complexity

- Best case :

Minimum number of character comparison

$$O\left(\left\lceil \frac{n}{ws+1} \right\rceil \times \left\lceil \frac{ws}{m} \right\rceil\right) \approx O\left(\frac{n}{m}\right)$$

Maximum shift

- Worst Case:

Maximum number of character comparison

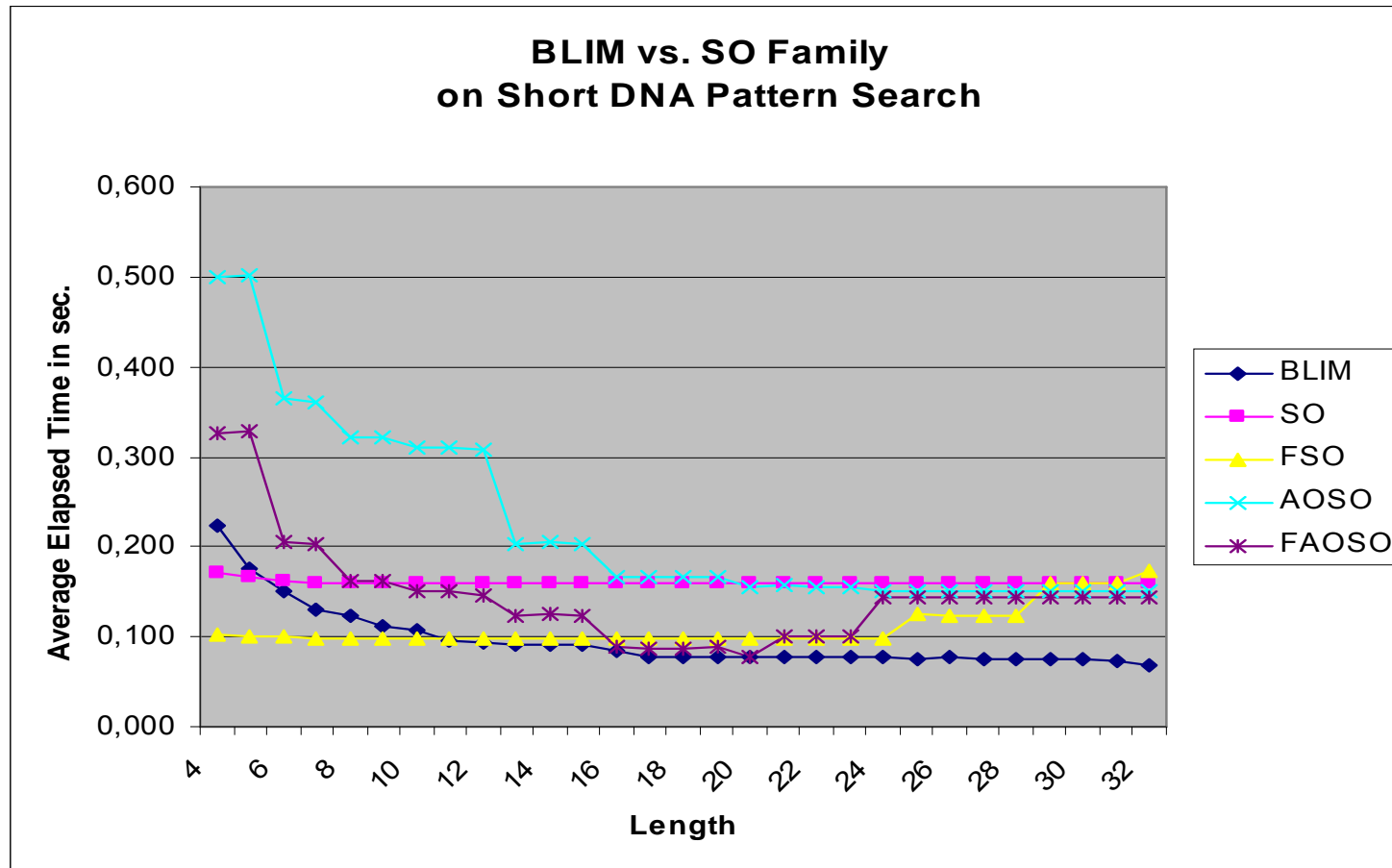
$$O\left(\left\lceil \frac{n}{w} \right\rceil \times ws\right)$$

Minimum shift

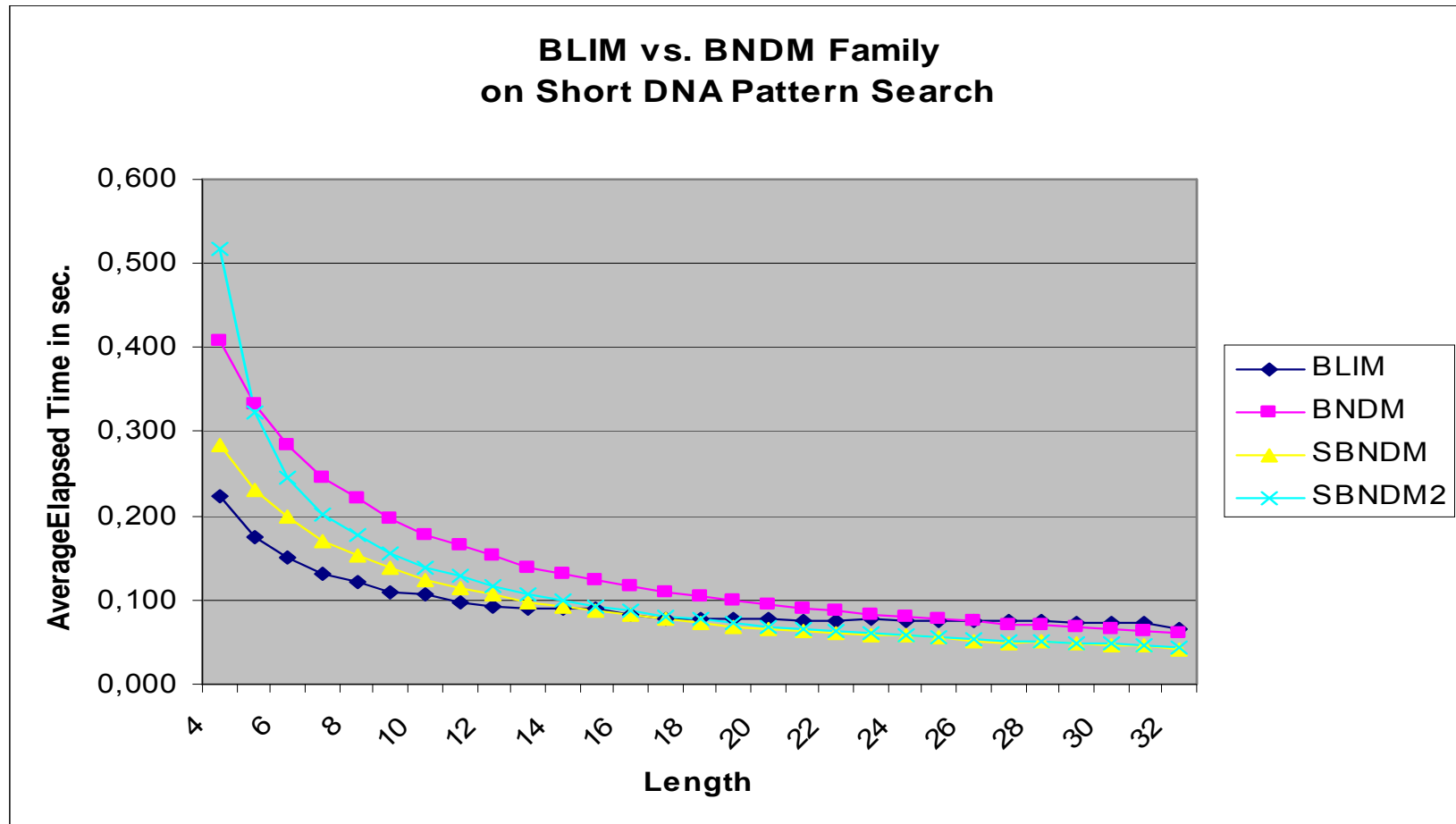
Experimental Results

- On DNA sequences (Manzini's DNA compression corpus)
- On natural language text (enwik8.txt)
- 100 sample pattern for each length tested
- gcc -O3
- Intel Xeon 2.4 Ghz, 3GB memory

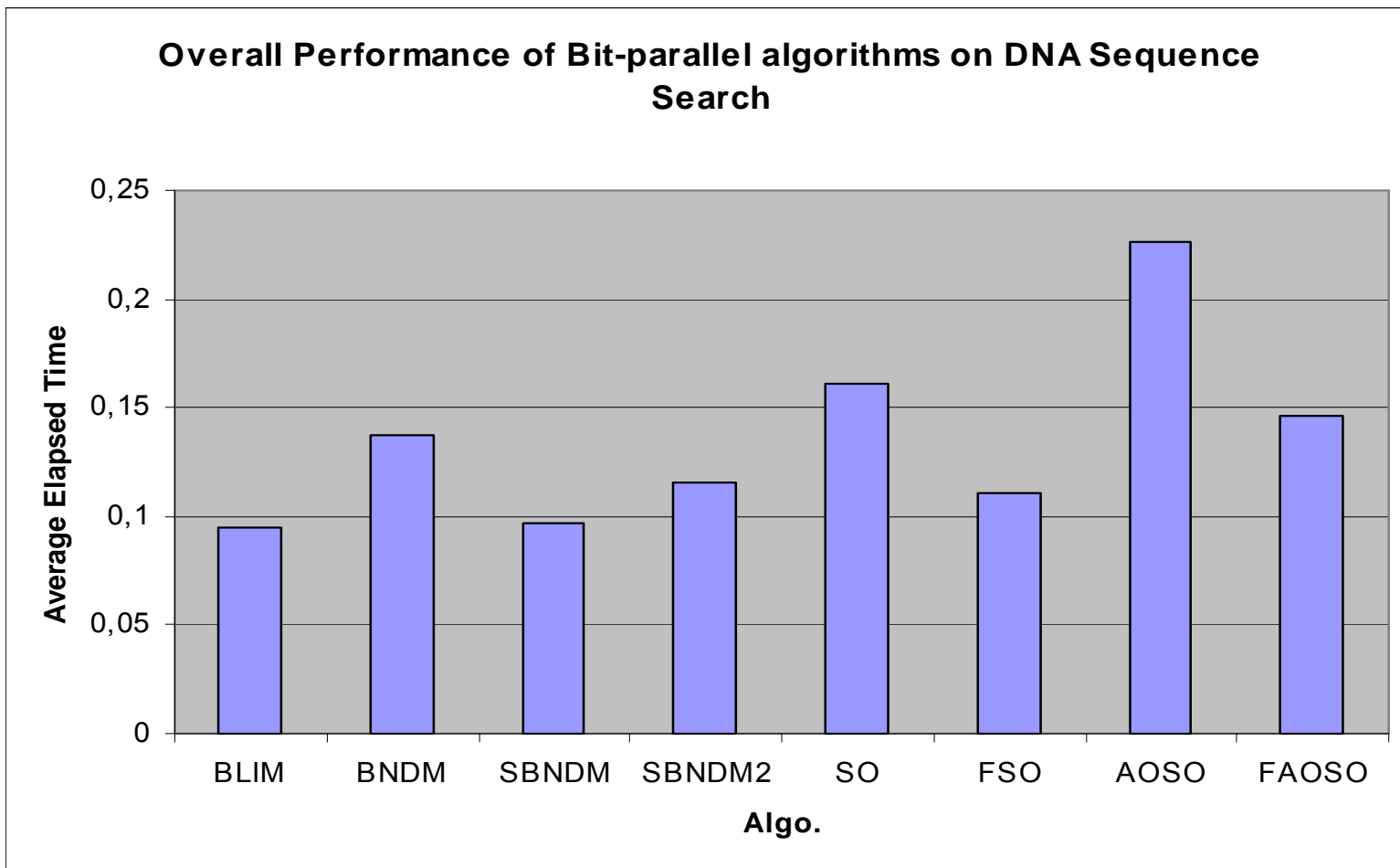
BLIM vs. SO Family on DNA



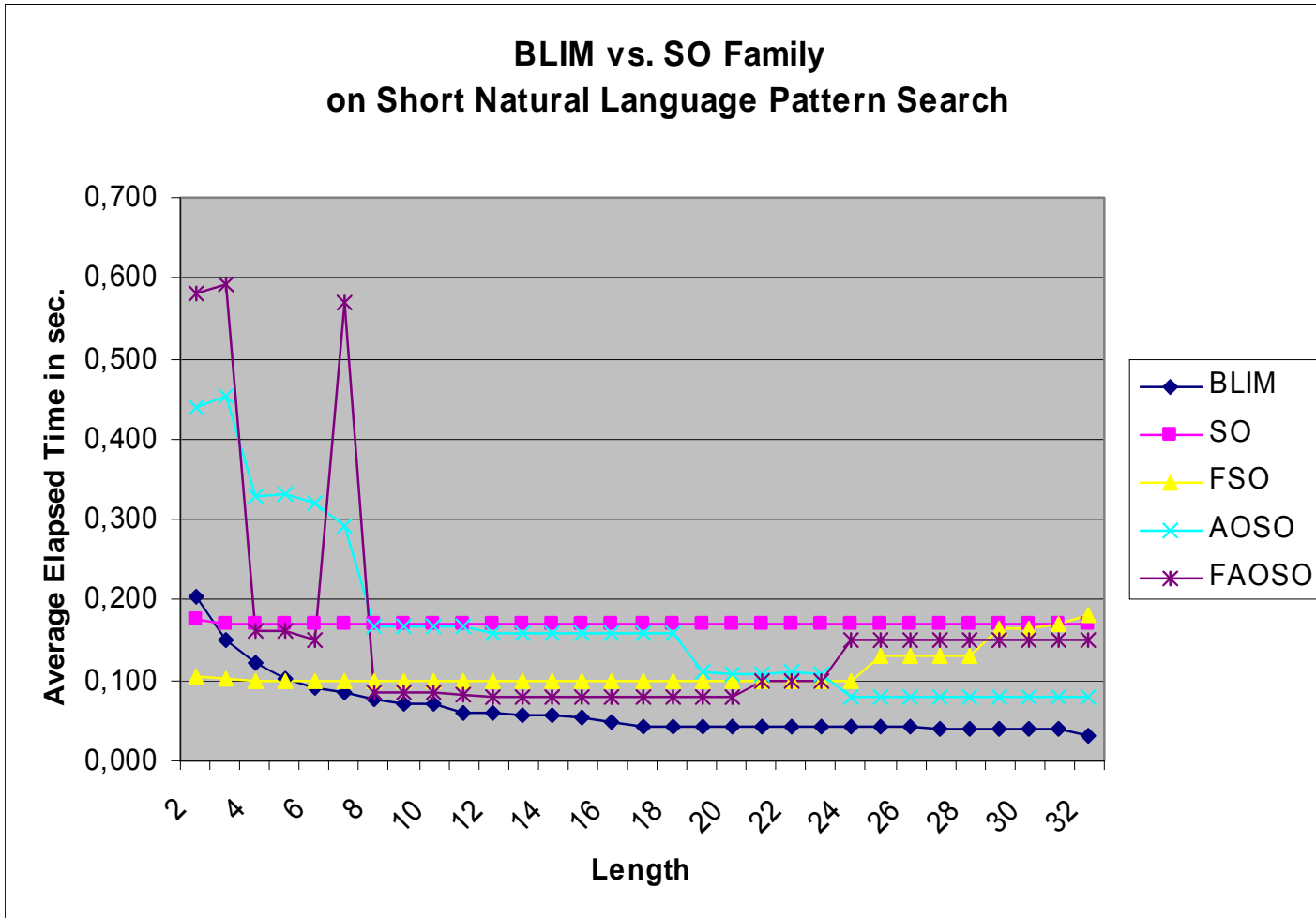
BLIM vs. BNDM Family on DNA



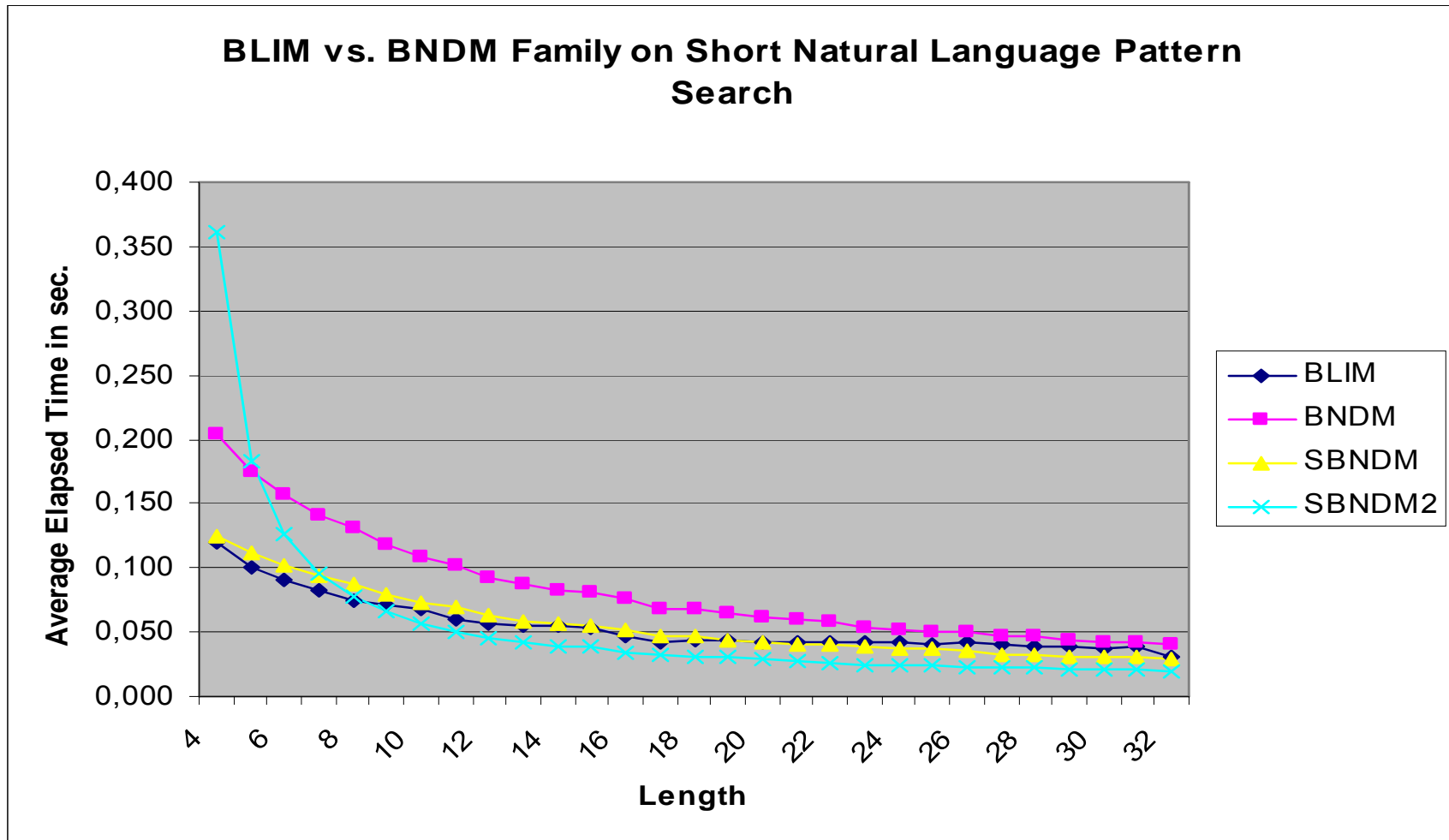
Overall Performance on DNA



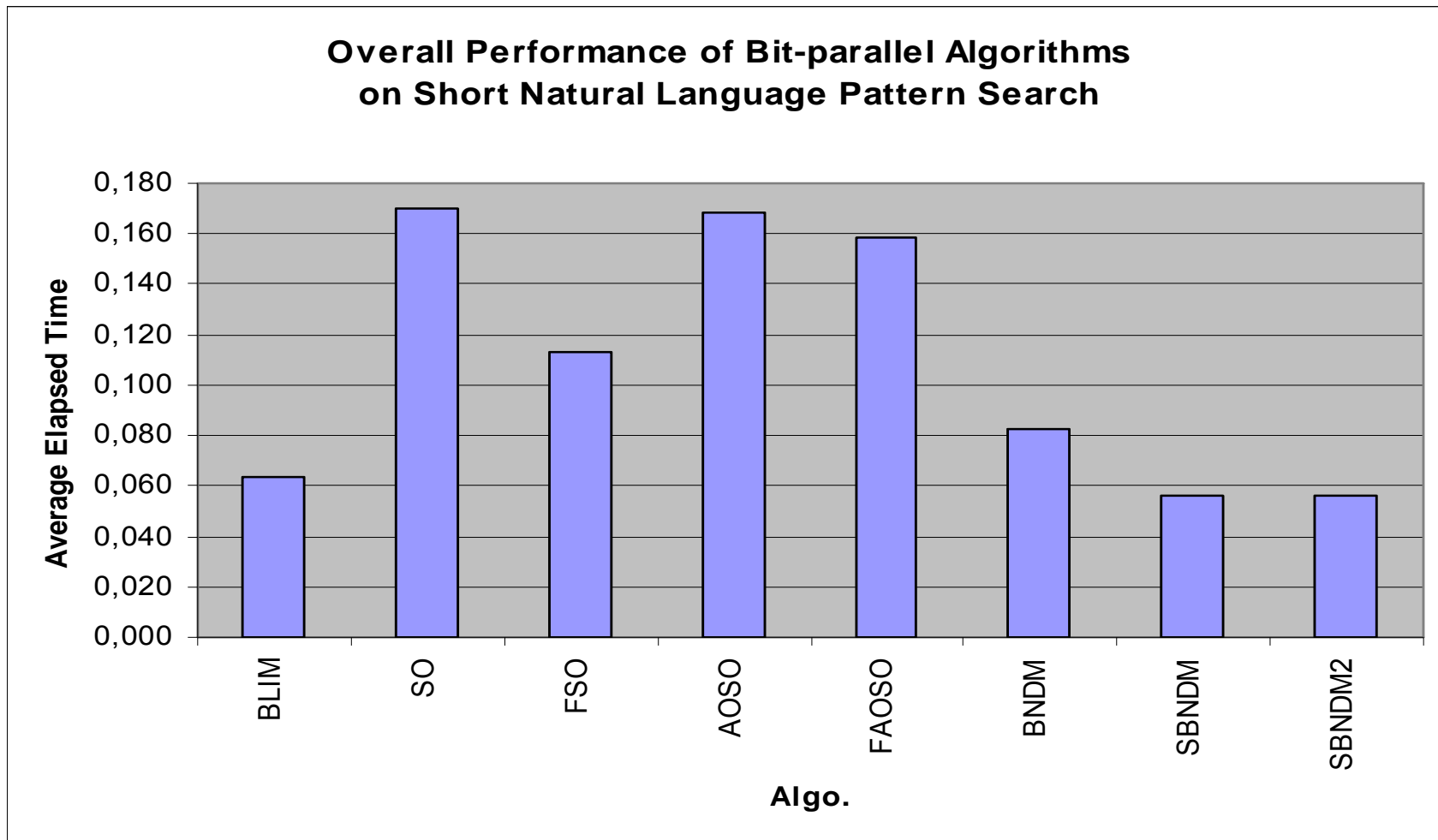
BLIM vs. SO Family on Nat.Lan.



BLIM vs. BNDM Family on Nat.Lan.



Overall performance on Nat. Lan.



Multi-pattern Case

- Bit-parallel approaches suffer more in multi-pattern case, as the total length is more likely to exceed the word size.
- BLIM serves a good basis for that case with its ability to search **up to W patterns** of any length in a common bit-parallel fashion.

Multi-pattern BLIM

	0	1	2	3	4	5	6
0	a	b	a	a			
1		a	b	a	a		
2			a	b	a	a	
3				a	b	a	a
4	b	b	a				
5		b	b	a			
6			b	b	a		
7				b	b	a	

$$\text{pivot} = \min\{R-1+|P_i|, P_i \in P\} = 6$$

$$P = \{abaa, bba\}$$

$$R = \lceil W / |P| \rceil = 8/2 = 4$$

$$\text{ws} = \max\{R-1+|P_i|, P_i \in P\} = 7$$

Multi-pattern BLIM

- Bitmask creation
 - straight forward as before.
- ScanOrder
 - Let $s = \min\{ P_i, P_i \in P \}$
 - $S1 = \{ s-i, 2s-i, \dots, ks-i \}, i = 1, 2, \dots, s$
 - $(ks-i) < \text{pivot}$

$\text{ScanOrder} = S1 \cup \{\text{pivot}, \text{pivot}+1, \dots, \text{ws}-1\}$

Multi-BLIM Scan Order

	0	1	2	3	4	5	6
0	a	b	a	a	a	a	a
1		a	b	a	a	a	
2			a	b	a	a	
3				a	b	a	a
4	b	b	a	a	b	a	
5		b	b	b	a	a	
6				b	b		
7					b	a	

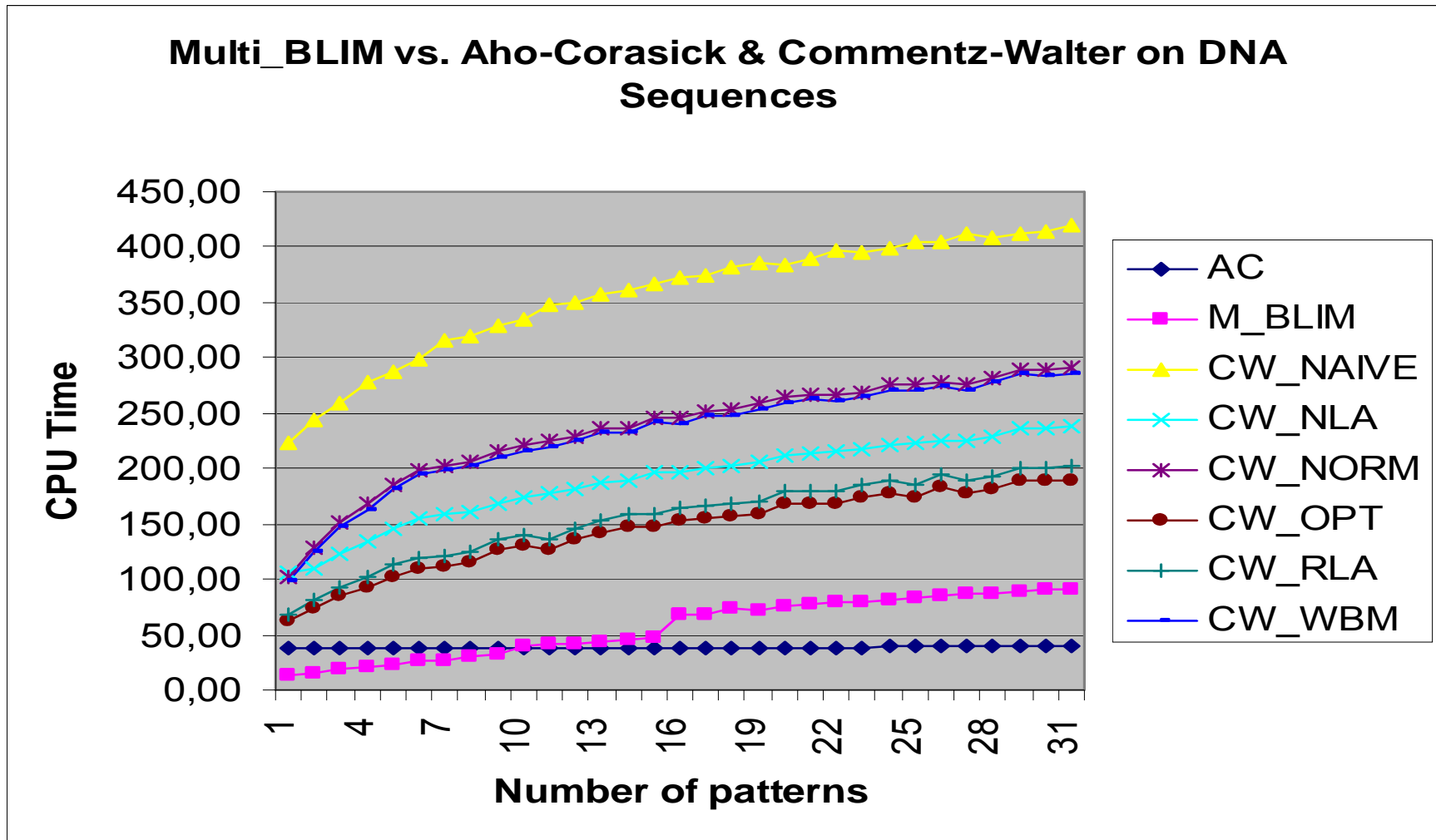
$$s = \min \{4, 3\} = 3$$

ScanOrder= 2, 5, 1, 4, 0, 3, **6**

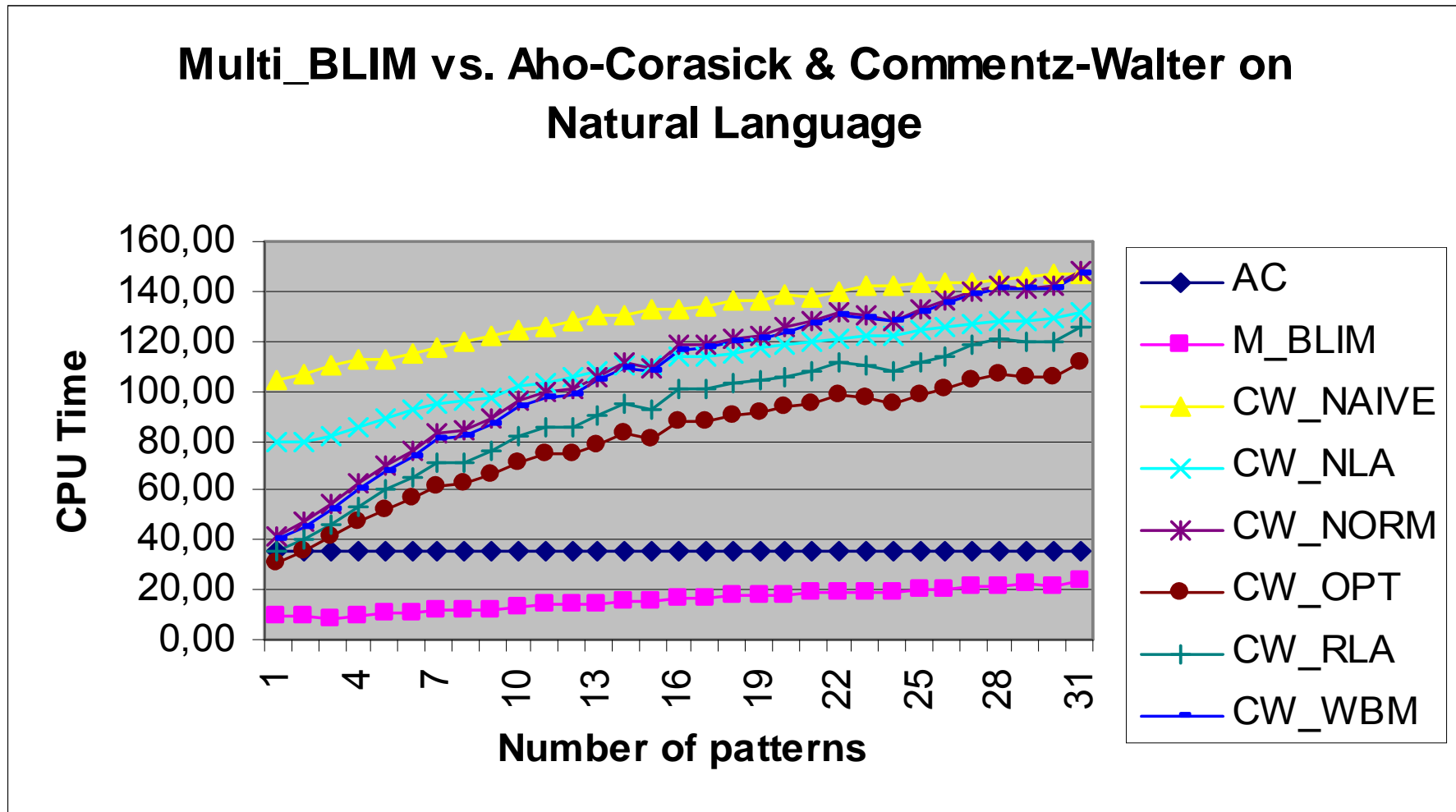
Experimental Results on Multi_BLIM

- Multi_BLIM is compared with Aho&Corasick and Commentz&Walter algorithms via the SPARE Parts 2003 toolkit.
- DNA pattern lengths in between 4 to 30.
- NL pattern lengths in between 2 to 20.
- Up to 32 patterns randomly collected for each test.
- Intel Xeon 2.4GHz, 3GB Memory
- Manzini's DNA corpus & enwik8.txt

Multi_BLIM Performance on DNA



Multi_BLIM Performance on Nat. Lan.



About q-gram utilization?

- Instead of reading one character at a time, read more by the help of the recent advances in CPU architecture
 - *Fredriksson, 'Shift-or string matching with super alphabets', 2003*
 - *Durian et al., 'Tuning BNDM with q-grams', 2009*
- Unfortunately, not so much gain because of BLIM's random access structure

About q-gram utilization

Mainly 2 reasons for low gain when using q-grams in BLIM:

1. BLIM does not pass over the text sequentially, but instead performs distant reads on the investigation window.
2. Mask is of size $|\Sigma|*(W+m-1)$. As Σ grows with q-gram usage, Mask becomes large that is not fitting into the first level cache.

Conclusion

- An initial attempt to solve computer word size limitation in bit-parallel pattern matching
- The speed is in range of SBNDM, and SBNDM2, *with an additional advantage that it does not require to do something special when input pattern length is longer than W .*

Conclusion

- It must be noted that, in general, it is slower than Lecroq's new algorithm, and also for lengths longer than 100, backward (suffix)oracle matching is a better alternative (*applies to all bit-parallel algorithms also*).
- Multi pattern BLIM shows good performance and maybe a strong alternative for classical multi pattern search algorithms.

Acknowledgement

Thanks to

- **Jorma Tarhio,**
- **Kimmo Fredriksson,**
- **Thierry Lecroq,**

for sharing their codes and comments.

Thank you!

any question?