Discovering Most Classificatory Patterns for Very Expressive Pattern Classes

Masayuki Takeda[1,2], Shunsuke Inenaga[3], Hideo Bannai[4], Ayumi Shinohara[1,2], and Setsuo Arikawa[1]

[1]Department of Informatics, Kyushu University
[2]Japan Science Technology Corporation Agency
[3]Department of Computer Science, University of Helsinki
[4]Human Genome Center, University of Tokyo

Background and Motivation

Distinguish two given string datasets

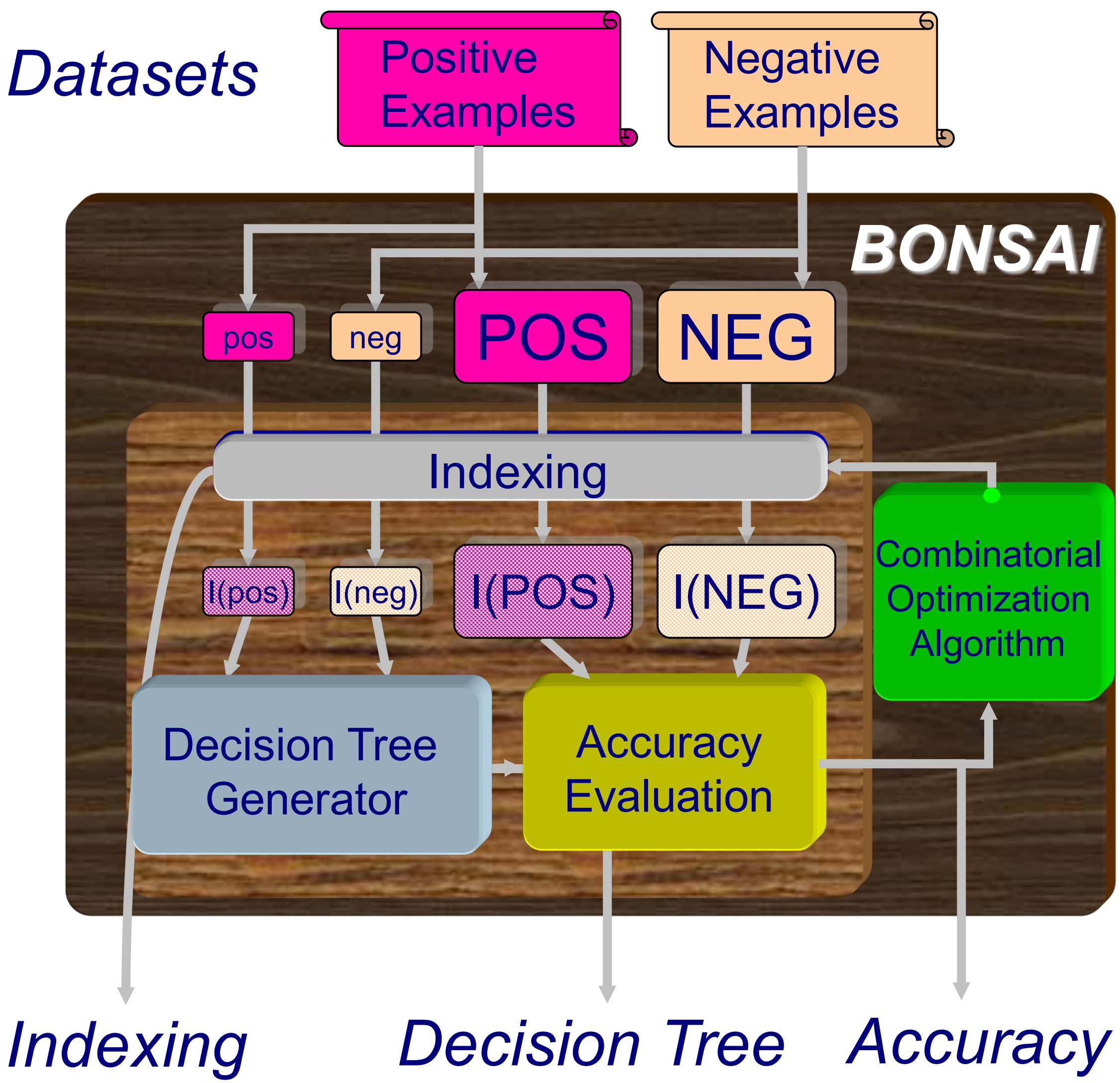
- to obtain a good ***rule*** *and/or useful* ***knowledge***

Grade up ***BONSAI*** *system*

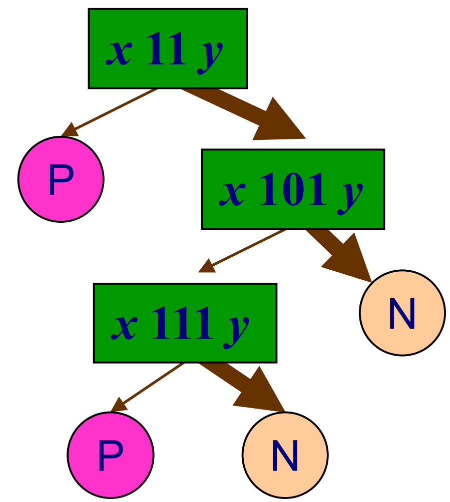
- so that it can deal with more ***expressive pattern classes***

Machine Discovery System BONSAI

[*Shimozono et. al 1994*]



ABCDEFGHIJKLMNOPQRSTUVWXY
0011001010001110000011010



Pattern Discovery from Datasets

Find a pattern string that occurs ***in all strings of A*** *and* ***in no strings of B****.*

A	B
AKEBONO MUSASHIMARU	WAKANOHANA TAKANOHANA
CONTRIBUTIONS OF AI	CONTRIBUTIONS OF UN
BEYOND MESSY LEARNING	TRADITIONAL APPROACHES
BASED ON LOCAL SEARCH ALGORITHMS	GENETIC ALGORITHMS
BOOLEAN CLASSIFICATION	PROBABILISTIC RULE
SYMBOLIC TRANSFORMATION	NUMERIC TRANSFORMATION
BACON SANDWICH	PLAIN OMELETTE
PUBLICATION OF DISSERTATION	TOY EXAMPLES

Answer: **BONSAI**

Optimization Problem

- *Input:* *Two sets* S, T *of strings*
- *Output: A pattern* p *that maximizes the score function* $f(x_p, y_p, |S|, |T|)$.

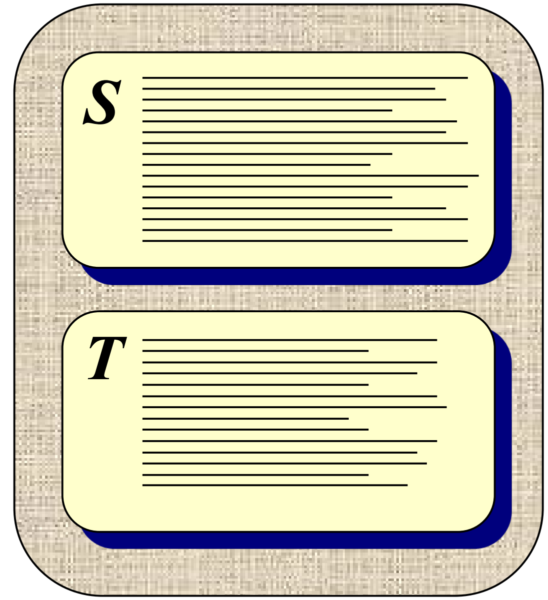
x_p : The num. of strings in S that p matches.

y_p : The num. of strings in T that p matches.

Score function f *expresses the* ***goodness*** *of* p *in terms of separating the two sets* S *and* T*.*

Process of Computation

INPUT



computing the *"goodness"*
for all possible patterns

OUTPUT

the pattern of best score

as fast as possible!!

Previous Work

- *BONSAI*
  (discovering best ***Substring*** *pattern), Shimozono et al., 1994*
- *Discovering best* ***Subsequence*** *pattern, Hirao et al., 2000*
- *Discovering best* ***Episode*** *pattern, Hirao et al., 2001*
- *Discovering best* ***VLDC*** *pattern, Inenaga et al., 2002*
- *Discovering best* ***Window Accumulated VLDC*** *pattern,*
  Inenaga et al., 2002

This Work

We present efficient algorithms to discover:

- *the best **Fixed/Variable Length Don't Care** Pattern*
- *the best **Approximate FVLDC** Pattern*

*The aim is to apply more **expressive** pattern classes to BONSAI*

- *the best **Window Accumulated FVLDC** Pattern*
- *the best **Window Accumulated Approx. FVLDC** Pattern*

*The aim is to add a more **classificatory** power to the pattern classes*

Score Function

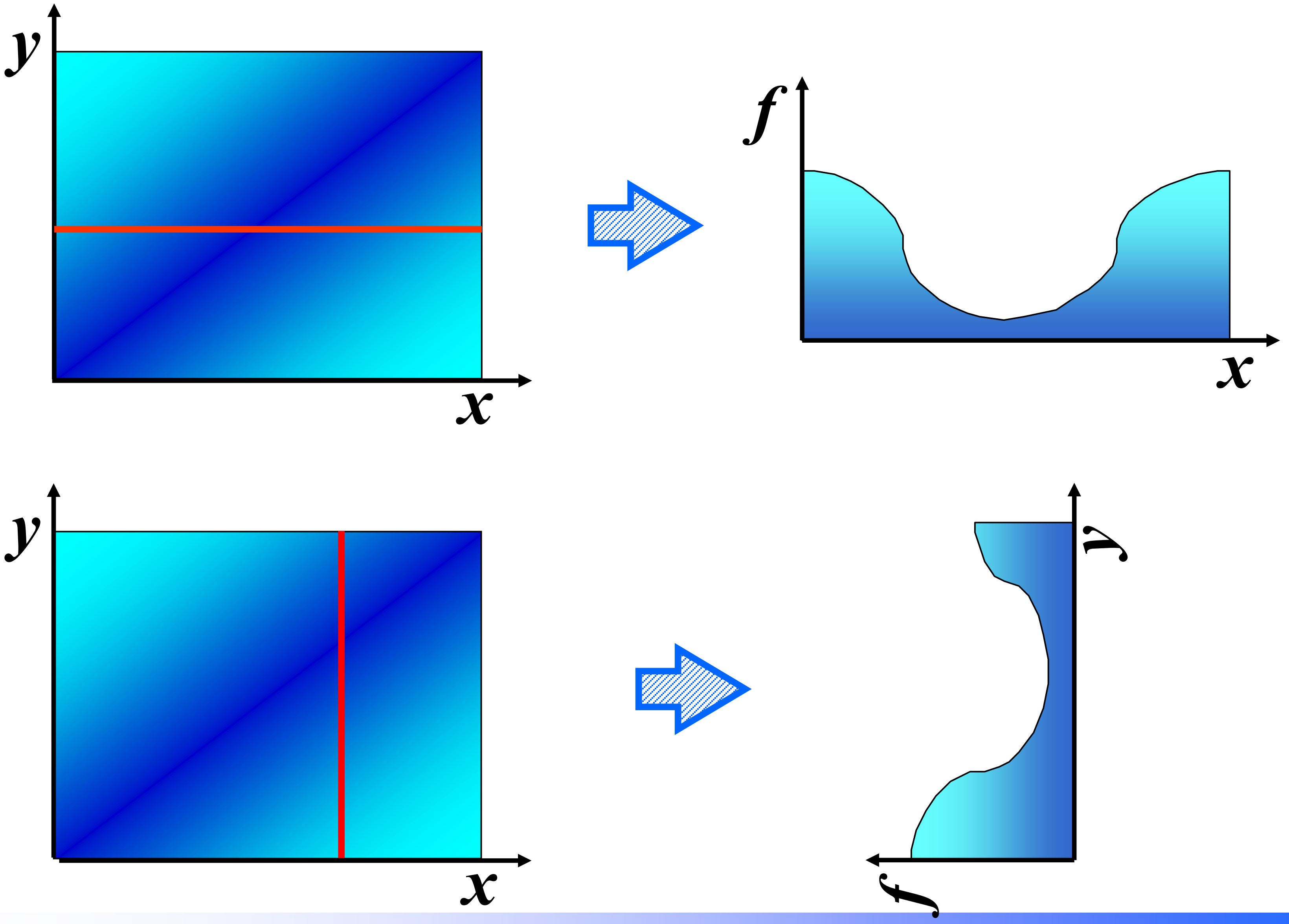
The *goodness* of pattern p

$$good(p, S, T) = f(x_p, y_p, |S|, |T|)$$

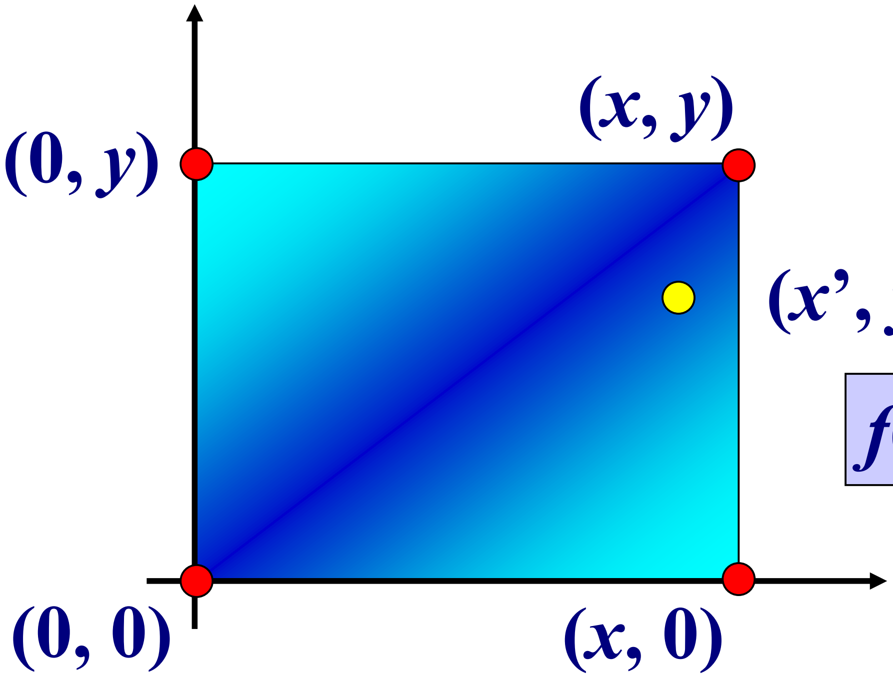
S, T : two given sets of strings
x_p : num. of strings in S that p matches
y_p : num. of strings in T that p matches

If score function f is ***conic****, then we can apply an efficient pruning technique for speeding up the computation.*

Score Function to be Conic



Conic Function Property

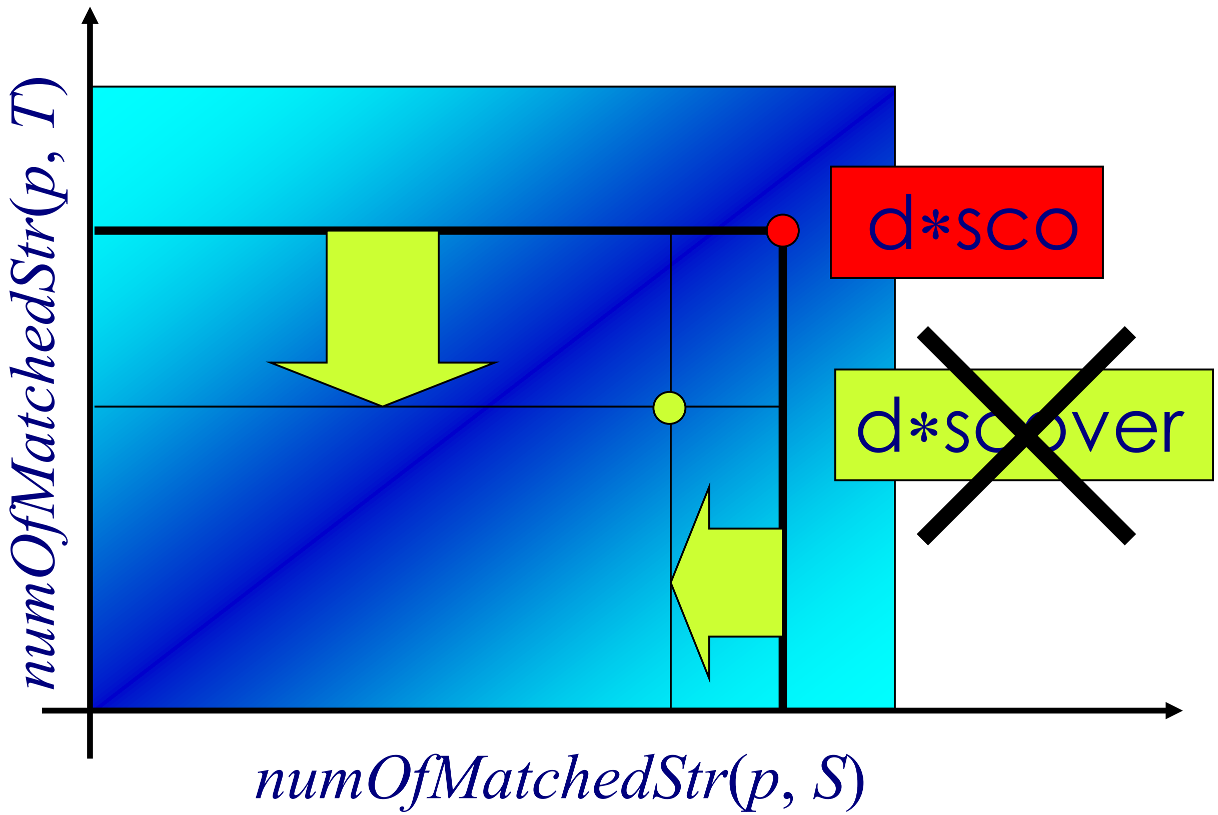


$f(x', y') \leq upperBound(x, y)$

$upperBound(x, y)$: the max value on the square

$$= max\{f(0, 0), f(x, 0), f(0, y), f(x, y)\}$$

Pruning Technique



The *goodness* of d*scover $\leq$ The *upperBound* of d*sco $<$ The current best score

FVLDC Pattern

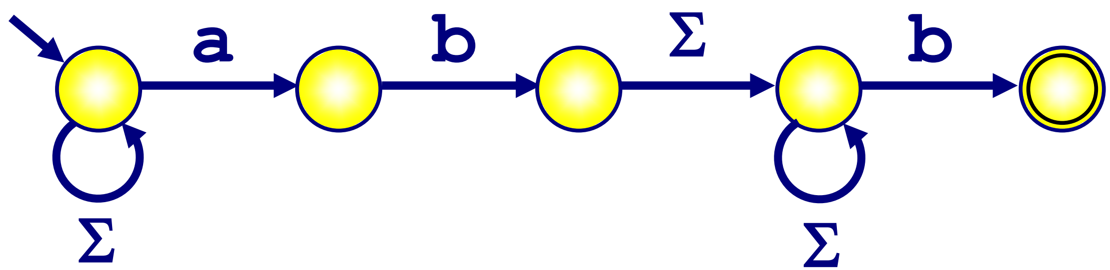
A ***Fixed/Variable Length Don't Care Pattern*** *is an element of* $\Pi = (\Sigma \cup \{\circ, \star\})^*$, *where* $\circ$ *matches any character and* $\star$ *matches any string*.

e.g. FVLDC pattern **ab○a○★b** *matches* **abbaabbb**.

abbaa bb

We use an ***NFA*** *that recognizes the language of a given FVLDC pattern* p*. The num. of states is* $m+1$*, where* m *is the num. of constants and* ○*'s in* p*.*

$p =$ ★**ab**○★**b**



Using the ***bit-parallel technique****, we can do matching for* p *in* $O(m|\Sigma|)$ *preprocessing time and* $O(n)$ *running time .*

Approximate FVLDC Pattern

An Approximate ***FVLDC Pattern*** *is an element of* $\Pi \times N$, *where* N *is the set of non-negative integers.*

Approx. FVLDC pattern <p, k> *is said to match a string* w *within distance* k *if the Hamming Distance between* p *and* w *is within* k.

e.g. Approx. FVLDC pattern <**ab○a○★b, 1**> *matches* **abbaabba**.

ab baa bb a

Approx. FVLDC Pattern Matching

*We use an **NFA** that recognizes the language of a given approx. FVLDC pattern* $\langle \boldsymbol{p}, \boldsymbol{k} \rangle$.

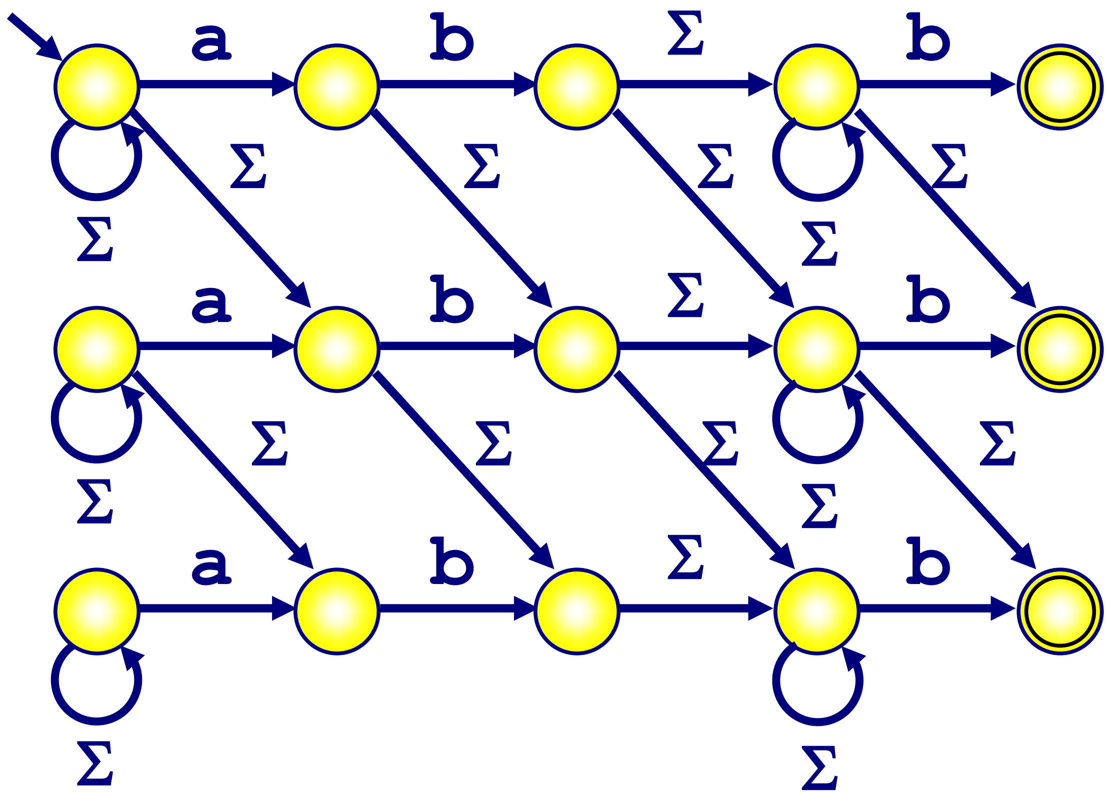
The NFA has $(\boldsymbol{m}+1)(\boldsymbol{k}+1)$ *states, but* $(\boldsymbol{m}-\boldsymbol{k}+1)(\boldsymbol{k}+1)$ *bits are actually enough.*

If $(\boldsymbol{m}-\boldsymbol{k}+1)(\boldsymbol{k}+1)$ *is not larger than the computer word length, our bit-parallel algorithm runs in* $\boldsymbol{O}(|\boldsymbol{n}|)$ *time after* $\boldsymbol{O}(\boldsymbol{m}|\Sigma|)$*-time preprocessing for* $\boldsymbol{p}$.

Approx. FVLDC Pattern Matching

p = <★ab○★b, 2>

m=4
k=2

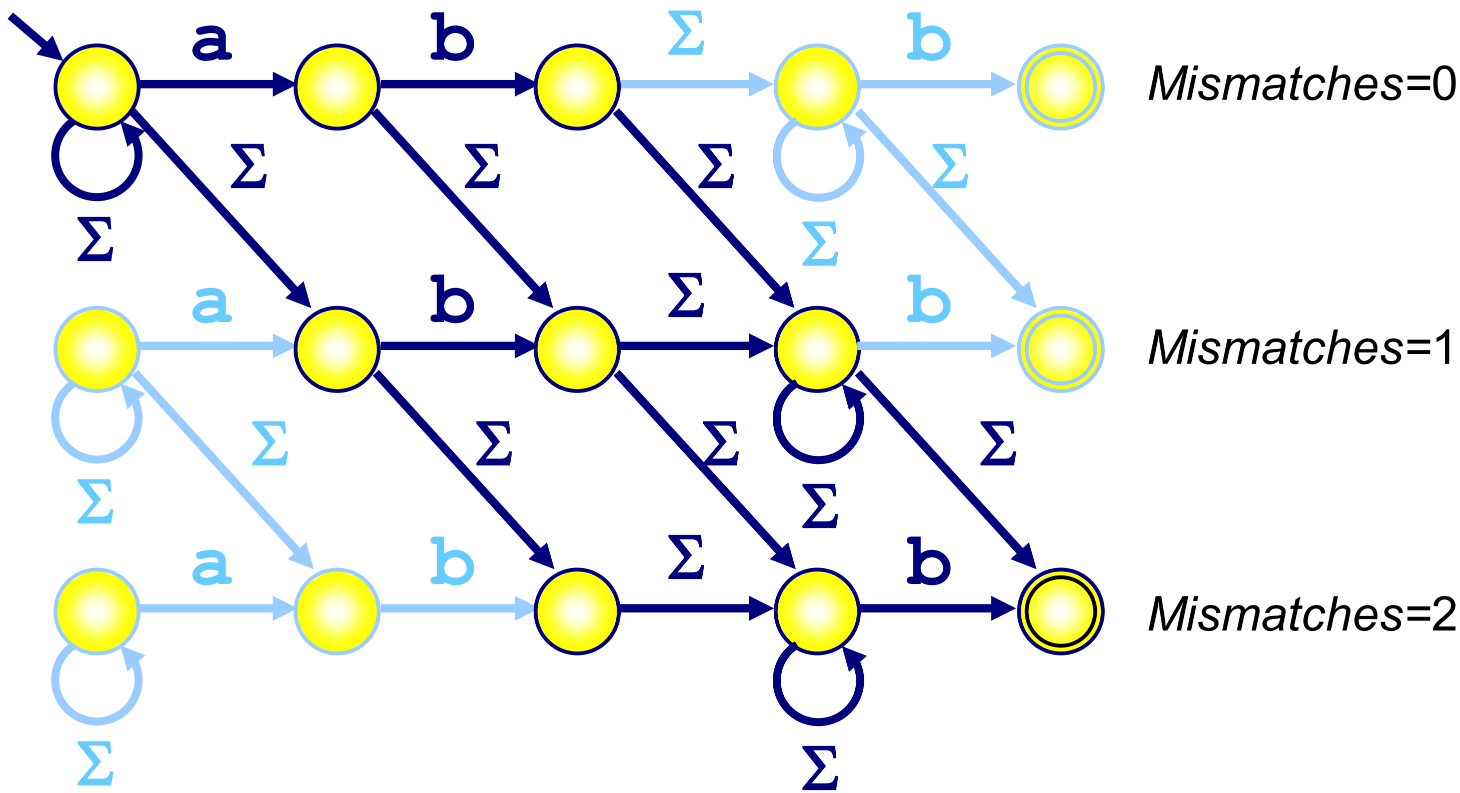


The NFA has $(m+1)(k+1)$ *states.*

Approx. FVLDC Pattern Matching

p = <★ab○★b, 2>

m=4
k=2



Only $(m\text{-}k+1)(k+1)$ *states are necessary.*

More Classificatory Pattern Class

$p = \star\mathtt{d}\bigcirc\star\mathtt{sc}\bigcirc\star\mathtt{very}\star$

any pattern similar
to "**discovery**"?

$w =$ **fhdihertlhglehglioogfrg**
xawpolmkhhjqirvnbotuhxxxxr
ylnvhbtriscovbgneinmvgerig
eooitrnrnvevroigreintnnvoi
woireohirlneroiveryniritro
eitruijnnbrymxbairive

They're far apart!!

Window Accumulation

Bound the length of occurrence of p *by a* ***window size*** h.

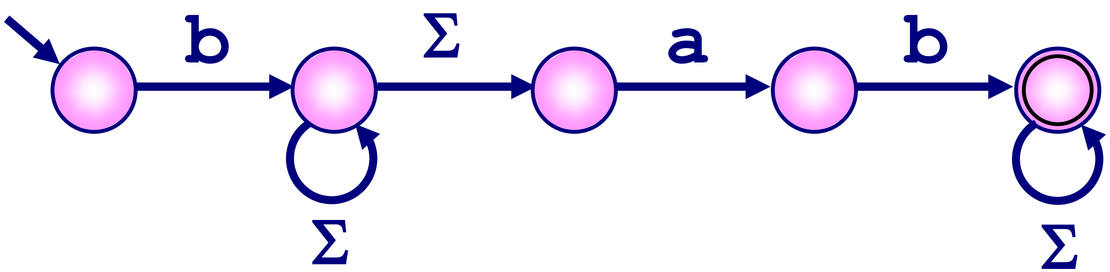
$$p = \star\mathtt{d}\bigcirc\underbrace{\star\mathtt{sc}\bigcirc\star\mathtt{very}}_{h}\star$$

This way we can get rid of redundant matches, and obtain better classification!

Window Accumulated Pattern Matching

*We use two **NFAs** each recognizes the language of either a given FVLDC pattern p or its reversal.*

$$p^{\mathrm{rev}} = \mathtt{b}\bigstar\bigcirc\mathtt{ab}\bigstar$$



*Using the **bit-parallel technique**, we can do pattern matching for $<p, h>$ in $O(m|\Sigma|)$ preprocessing time and in $O(n^2)$ running time .*

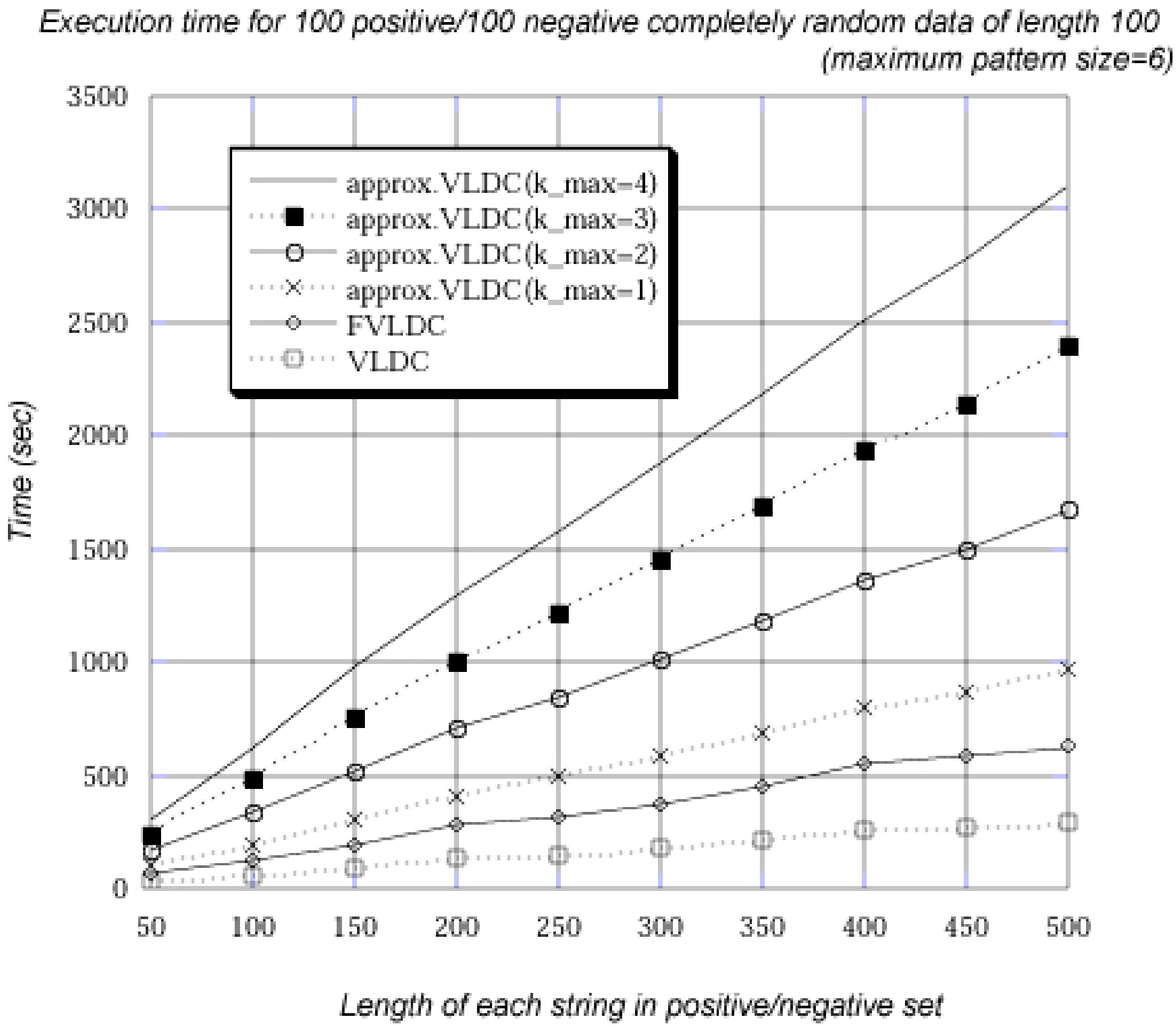
Same for Win-Acc. approx. FVLDC patterns.

Experimental Environment

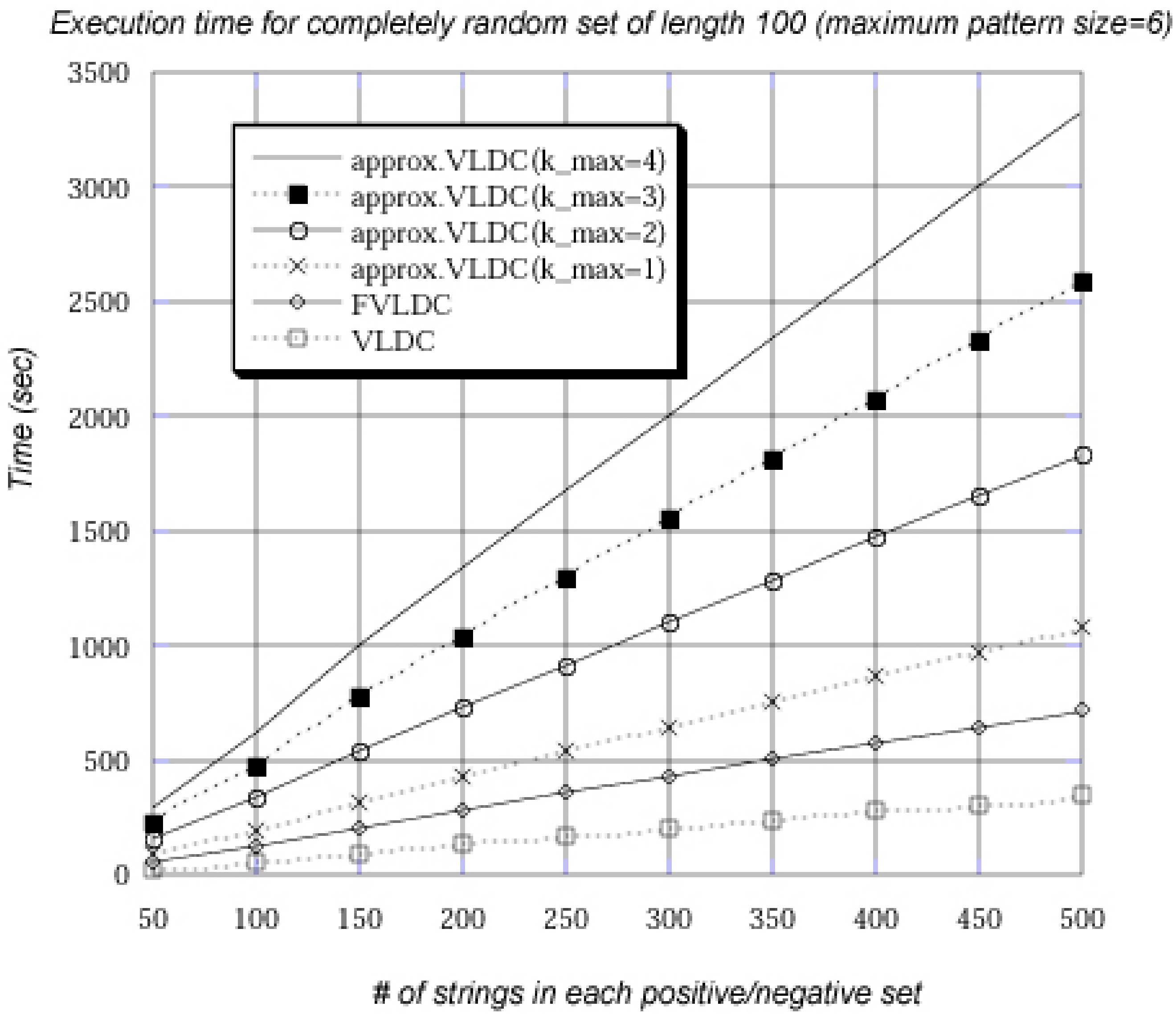
Machine: Alpha Station XP1000
CPU: Alpha21264 processor of 667MHz
OS: Tru64 Unix OS V4.0F

Datasets:
(1) completely random data
(2) VLDC pattern embedded data
(3) FVLDC pattern embedded data
(4) 2-approx. VLDC pattern embedded data
(5) window-accumulated 2-approx.
VLDC pattern embedded data

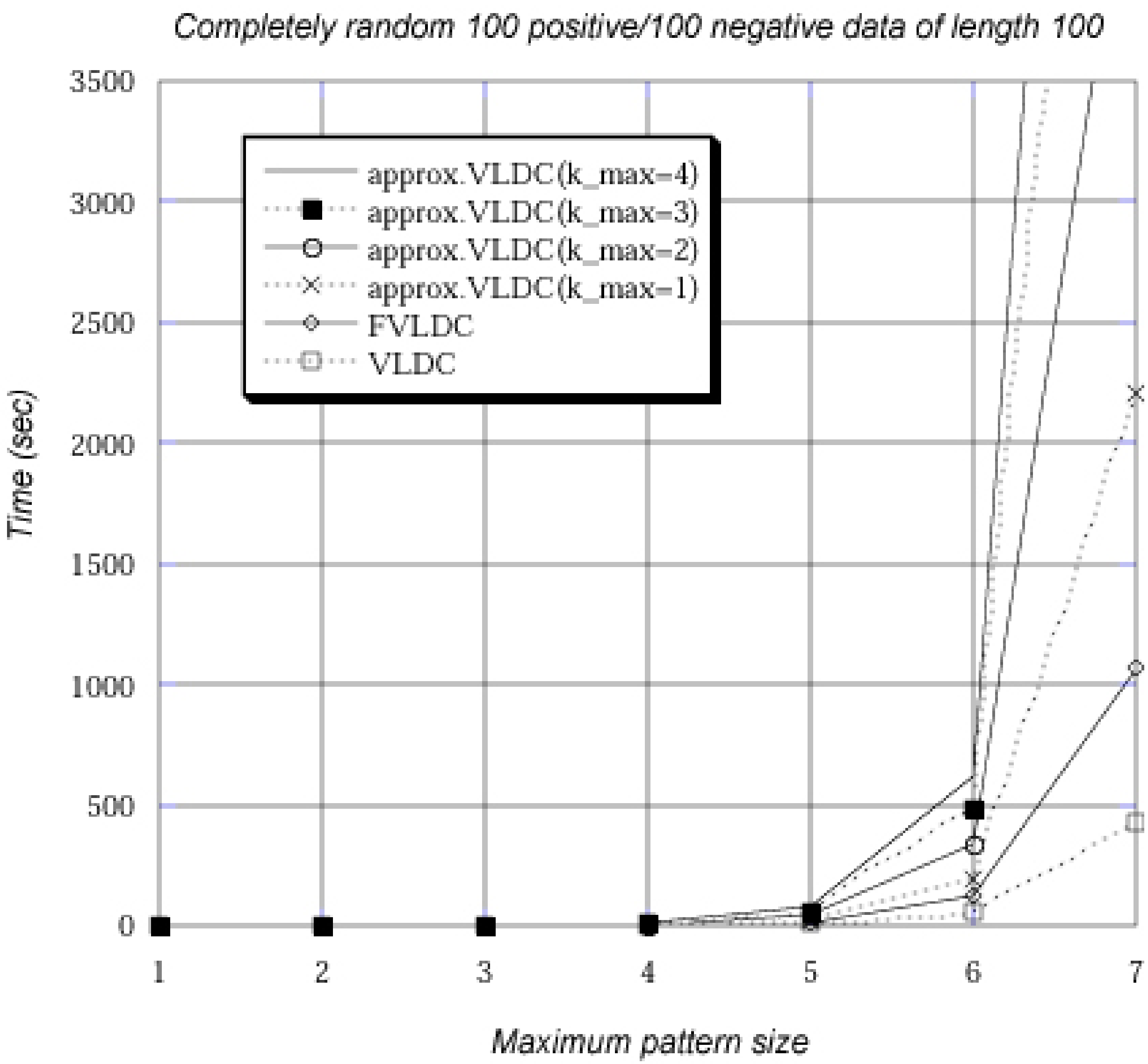
Experimental Result 1



Experimental Result 2



Experimental Result 3



Experimental Result 4

pattern class	*dataset*					
	(1)	*(2)*	*(3)*	*(4)*	*(5)*	*(5)*
VLDC	*423*	*109*	*236*	*182*	*224*	*(554)*
FVLDC	*1068*	*331*	*645*	*514*	*623*	*(1579)*
approx. VLDC ($k_{max}=1$)	*2203*	*725*	*1088*	*853*	*1026*	*(1820)*
approx. VLDC ($k_{max}=2$)	*4569*	*1660*	*2185*	*1790*	*2035*	*(3558)*
approx. VLDC ($k_{max}=3$)	*6973*	*2739*	*3324*	*2868*	*3146*	*(5679)*
approx. VLDC ($k_{max}=4$)	*9396*	*3880*	*4492*	*4008*	*4304*	*(8377)*

Execution times (in seconds) for different pattern classes:
The maximum pattern length was set to 7.
Execution time for each window-accumulated version with dataset (5) is shown in parentheses.