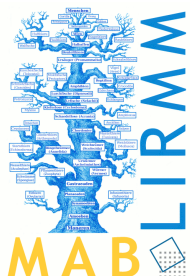
Algorithms for searching dinucleotidic Position Weight Matrices (di-PWM)

M. Mille, J. Ripoll, B. Cazaux, E. Rivals

November 25, 2021



Outline

Introduction

Enumeration based strategy

Results

Conclusion and future work

Introduction

Binding sites and probabilistic motifs

- proteins bind to DNA or RNA at different locations to control transcription, translation, etc.
- these binding sites share sequence similarity
  $\Rightarrow$ **multiple alignment**
- consensus sequences or regular expressions are inappropriate
  $\Rightarrow$ **probabilistic motifs**
  (PWM, di-PWM, HMM, Covariant Matrices)
- used to reveal new candidate binding sites
- PWMs: most frequent, but disregard dependencies between nuc. positions
  $\Rightarrow$ di-PWM

State of the art and objectives

Existing:

- Database for Transcription Factor binding sites: HOCOMOCO
  [Kulakovskiy et al., NAR, 2013]

- Tool for searching di-PWM occurrences in a target sequence: SPRY-SARUS
  Straightforward yet Powerful Rapid SuperAlphabet Representation Utilized for motif Search
  Algorithm [Pizzi et al. 2007]

State of the art and objectives

Existing:

- Database for Transcription Factor binding sites: HOCOMOCO
  [Kulakovskiy et al., NAR, 2013]

- Tool for searching di-PWM occurrences in a target sequence: SPRY-SARUS
  Straightforward yet Powerful Rapid SuperAlphabet Representation Utilized for motif Search
  Algorithm [Pizzi et al. 2007]

Objectives:

- new search algorithms

- running time and memory evaluation

- python module

What is a di-PWM ?

di-PWM

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16

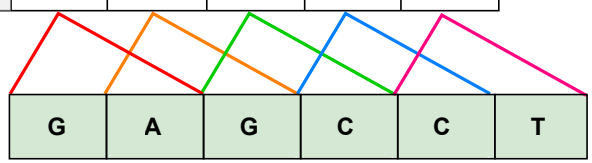
- Matrix P

What is a di-PWM ?

di-PWM

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16

Word

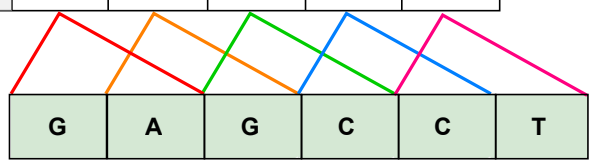


- Matrix P
- Overlap

What is a di-PWM ?

di-PWM

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16



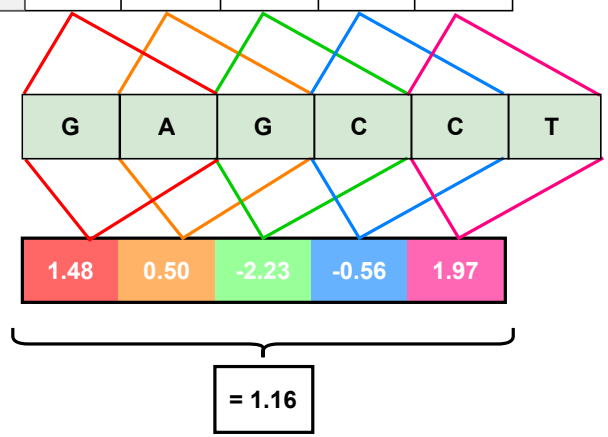
- Matrix P
- Overlap
- Weight

$$log\left(\frac{\text{obs. freq.}}{\text{exp. freq.}}\right)$$

What is a di-PWM ?

di-PWM

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16



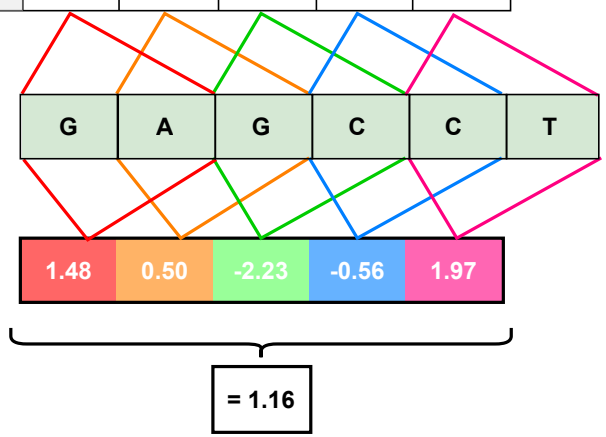
- Matrix P
- Overlap
- Weight
- Score

$$score(u) = \sum_{i \in \{0, m-2\}} P[\, u[i, i+1], i]$$

What is a di-PWM ?

di-PWM

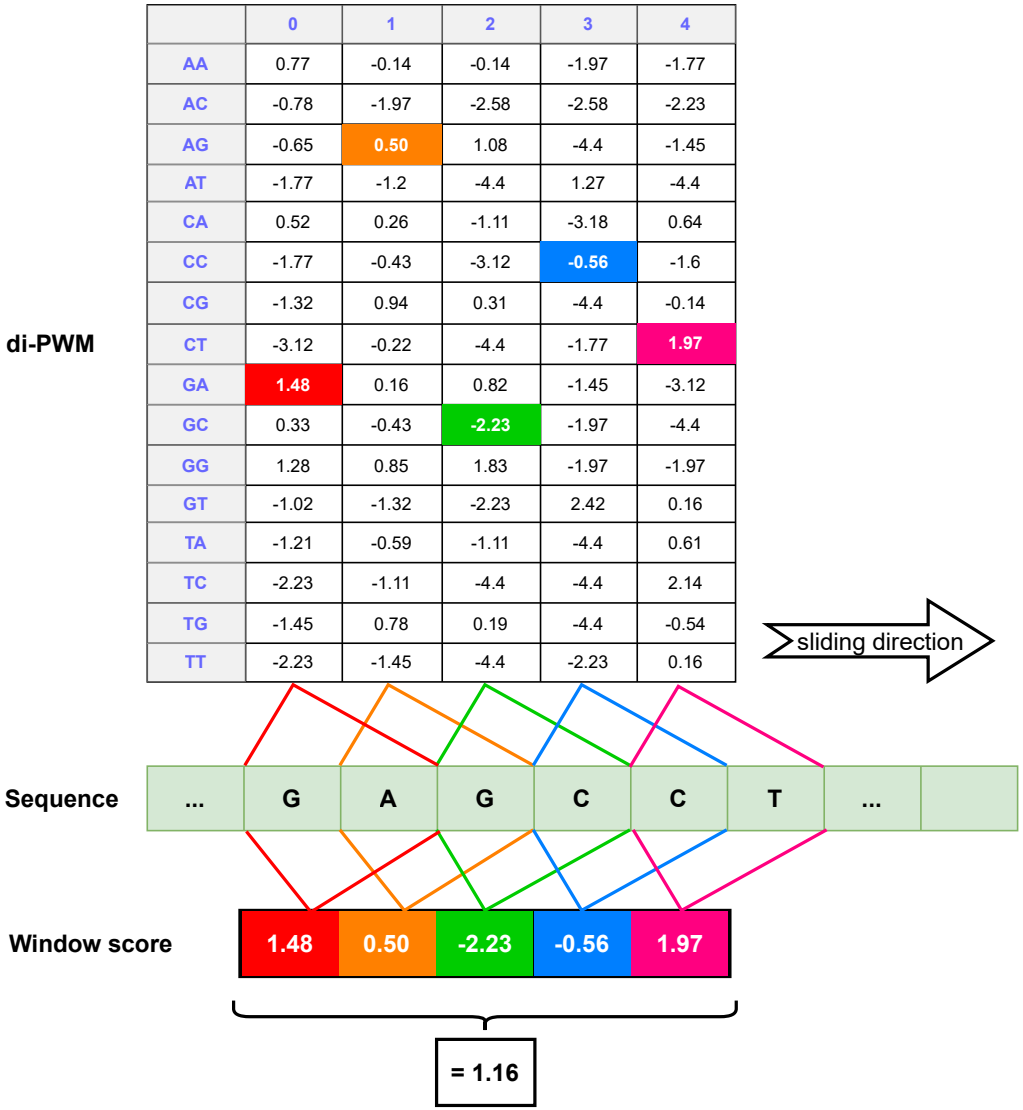
	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16



- Matrix P
- Overlap
- Weight
- Score
- Threshold

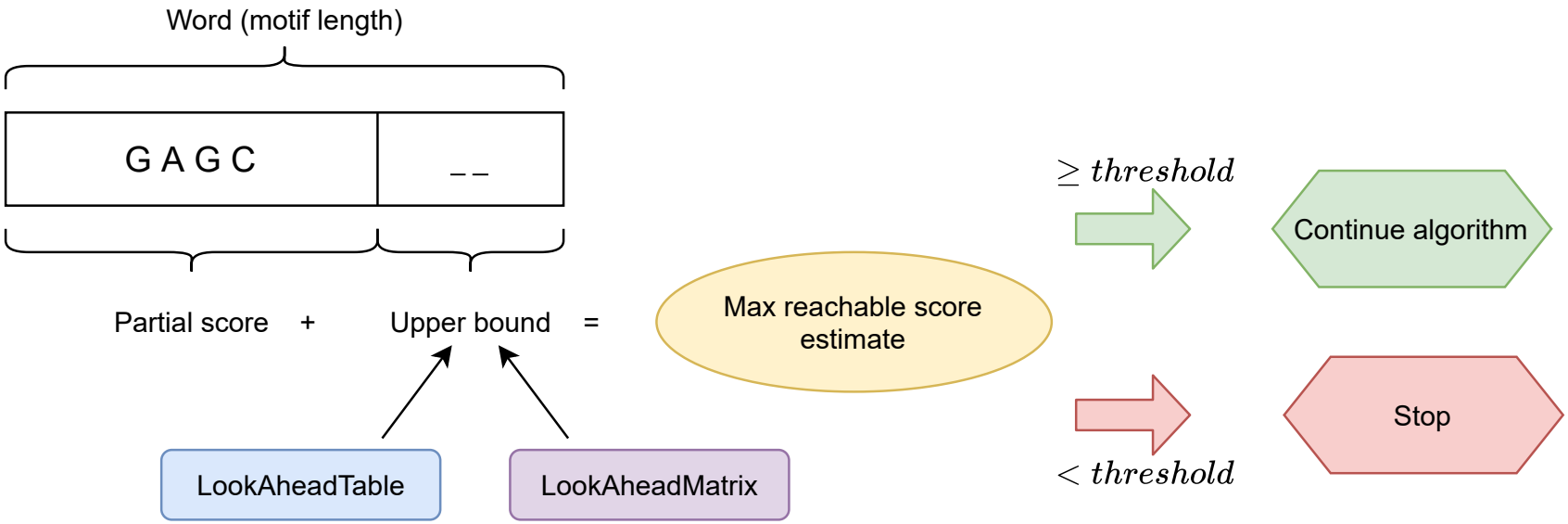
$$\theta = (score_{max} - score_{min}) * ratio + score_{min}$$

What is a di-PWM ?



- Matrix *P*
- Overlap
- Weight
- Score
- Threshold
- Scanning window algorithm

Maximum reachable score for a suffix



Useful for

- stopping the score computation of the current window
- cutting a branch during enumeration.

LookAheadTable(LAT) et LookAheadMatrix(LAM)

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16

di-PWM

LookAheadTable

Suffix start position	0	1	2	3	4
Max reachable score estimate	8.81	7.33	6.39	4.56	2.14

LookAheadMatrix

Start position of suffix starting with	0	1	2	3	4
A	7.66	6.89	5.64	3.41	-1.45
C	7.41	7.33	4.87	1.41	1.97
G	8.52	7.24	6.39	4.56	0.16
T	5.79	7.17	4.75	-0.09	2.14

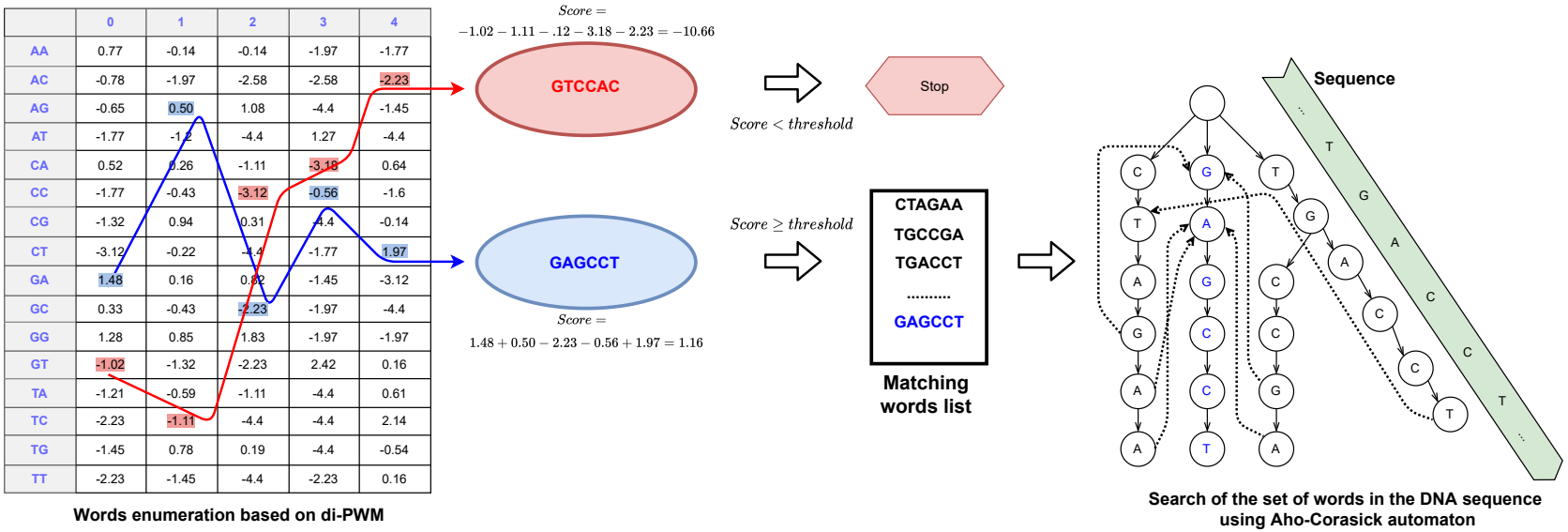
Properties

- *LAT*: Upper bound
- *LAM*: Greatest element

Ex:

$$LAM[G,3] = 4.56 = 2.42 + 2.14 = score(GT) + score(TC)$$

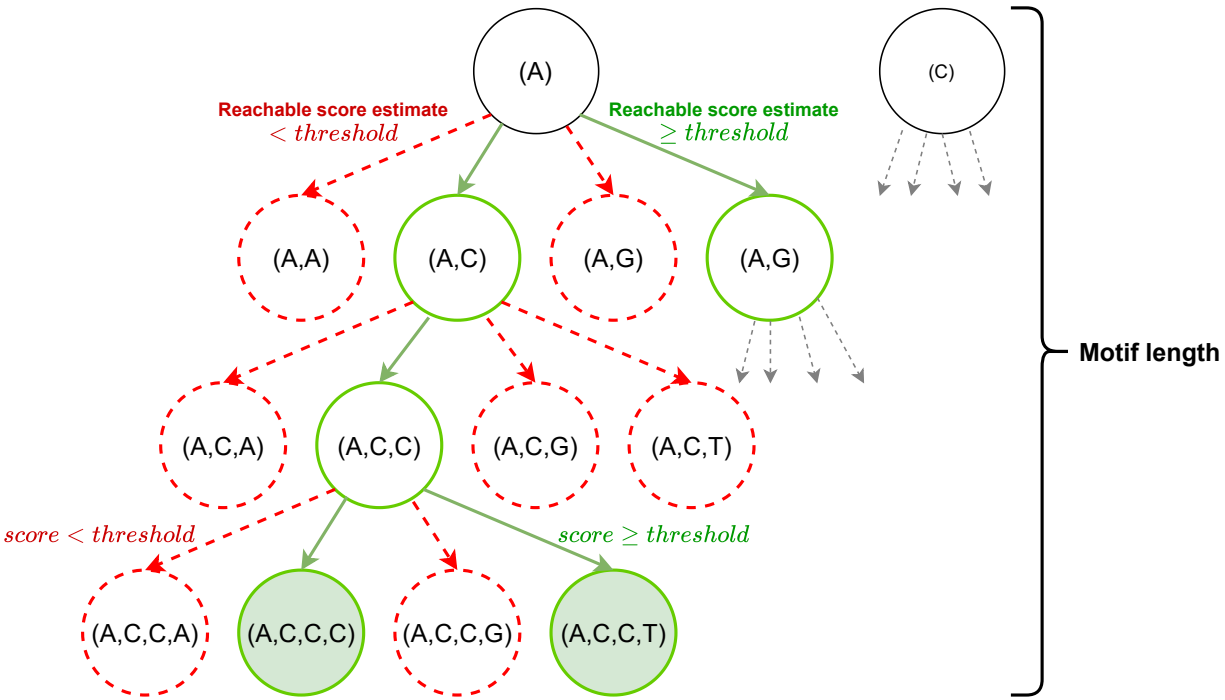
Enumeration based strategy: overview



Strategy:

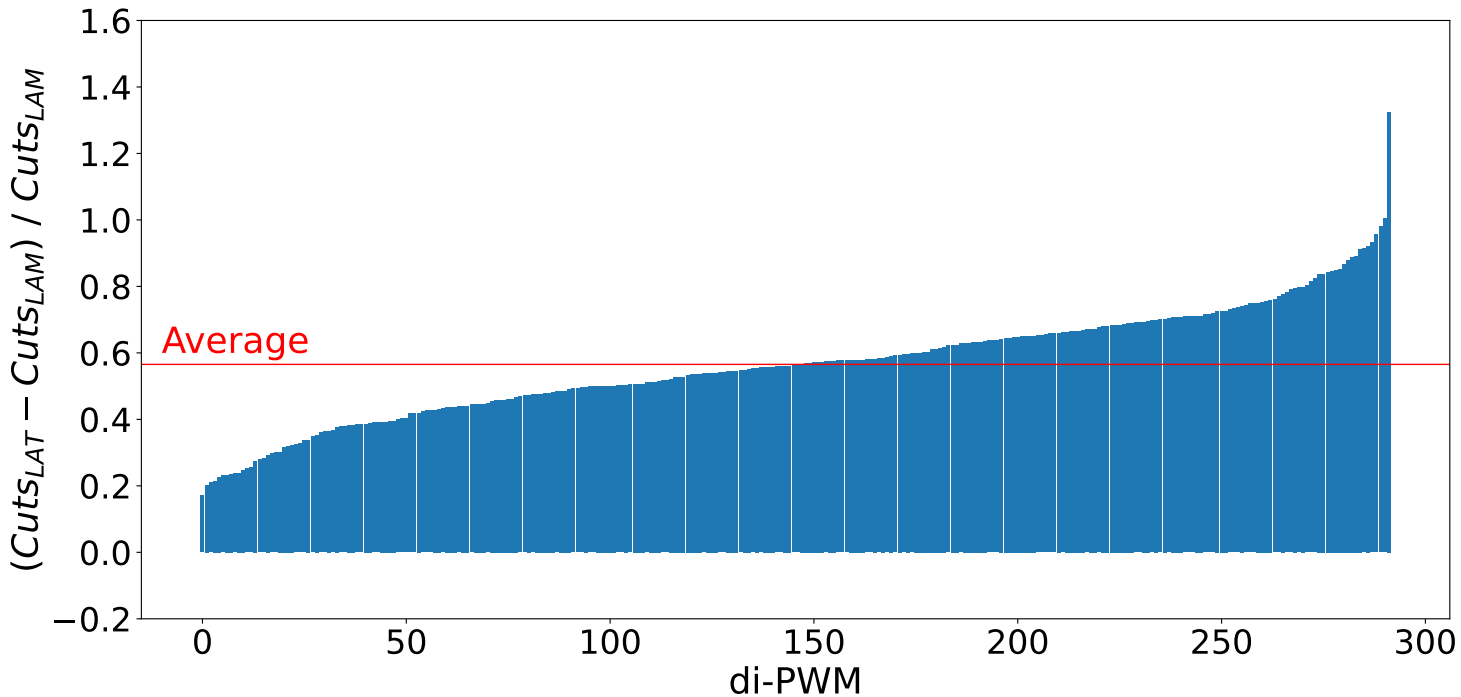
1. enumerate all valid words for threshold
2. use *exact set pattern matching* algorithm to search them in T
   archetypal solution: Aho-Corasick automaton [Aho & Corasick, 1975]

Enumeration of valid words



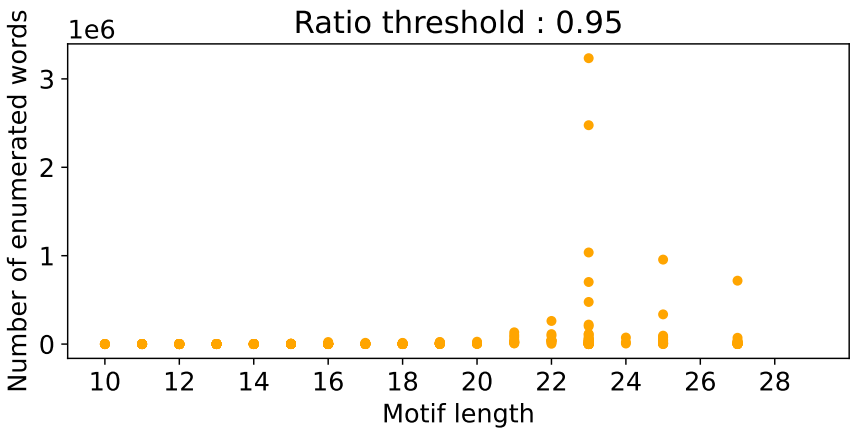
- Build a prefix trie that spells out words
- Depth first construction to limit memory usage
- Branch & Bound using LAT or LAM

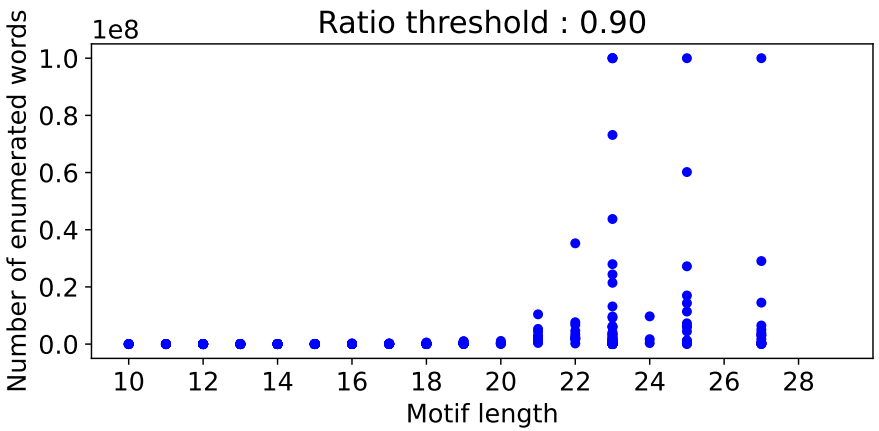
Efficiency of *LAT* vs *LAM* for enumeration

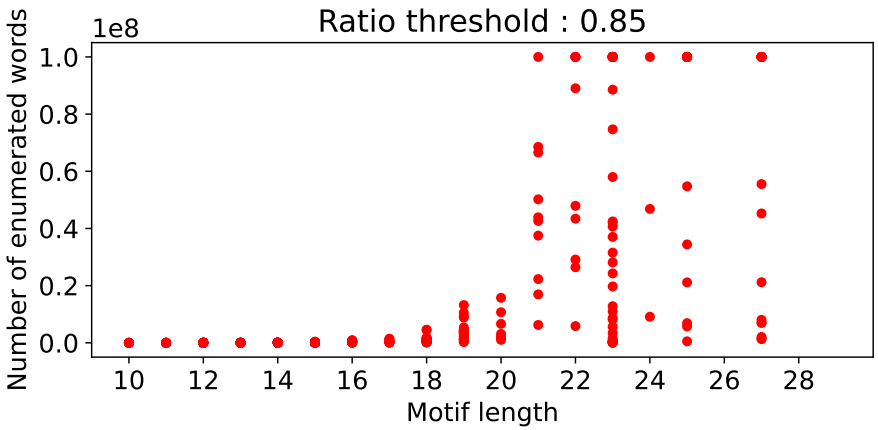


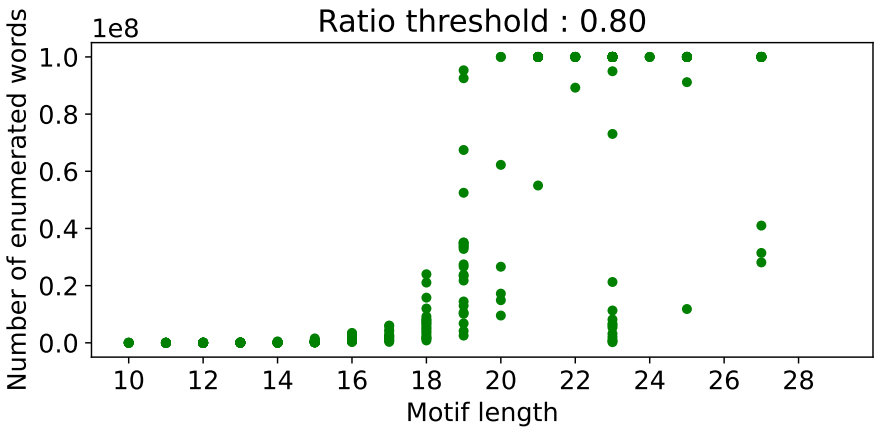
- Computed over all Human di-PWMs of HOCOMOCO – ratio 90%
- difference in nb of cuts is 56% of nb of cuts with LAM
- **less cuts means higher cuts** $\Rightarrow$ LAM more efficient

Number of valid words for each di-PWM of HOCOMOCO









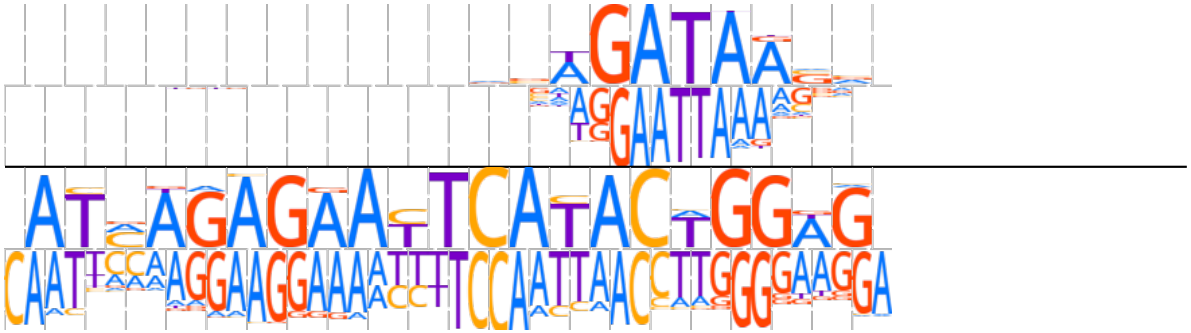
- Impact of ratio threshold and motif length on the number of valid words
- # of valid words exploses for lower ratios for some di-PWMs. Why?

Enumeration for two di-PWM motifs (of length 23)

Logos of di-PWMs of two TF

GATA2 →

ZNF274 →



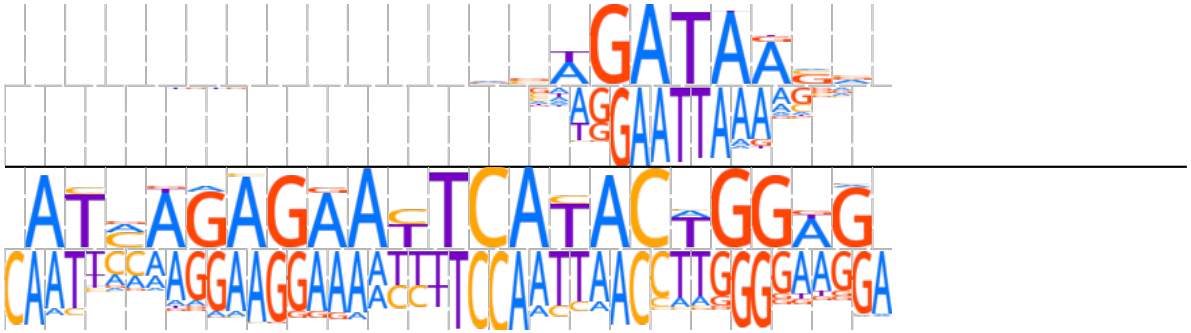
- Same length, but very different information content!
- Empty columns generate all possible substrings.

Enumeration for two di-PWM motifs (of length 23)

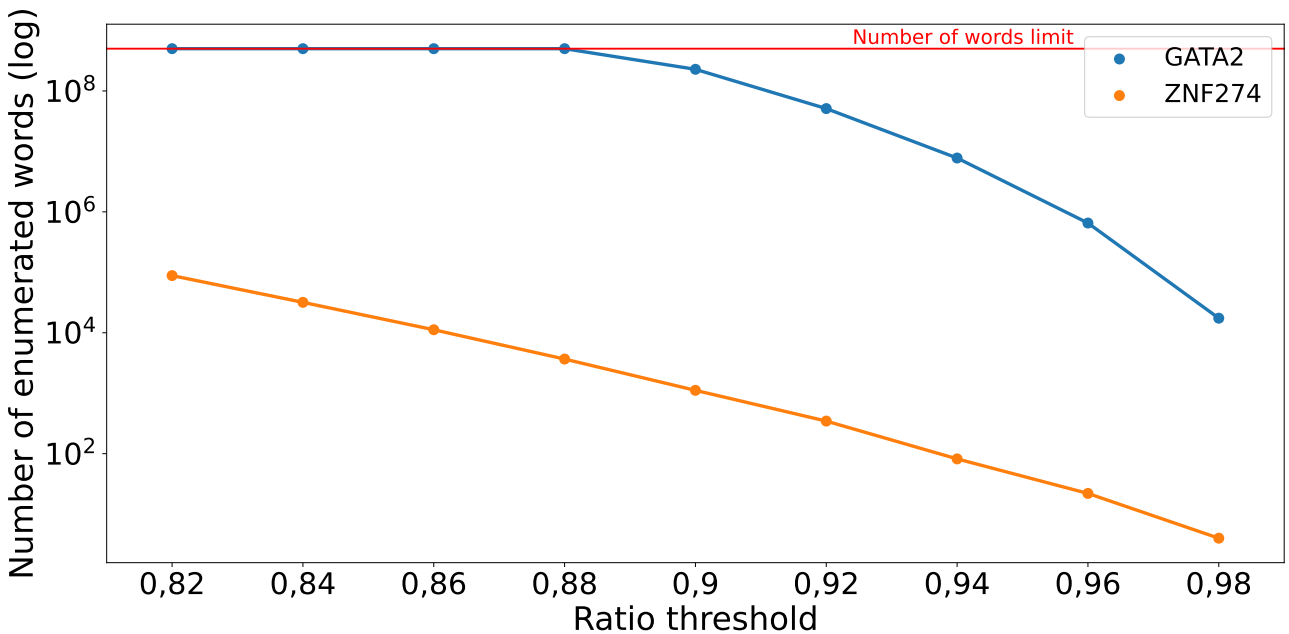
Logos of di-PWMs of two TF

GATA2 →

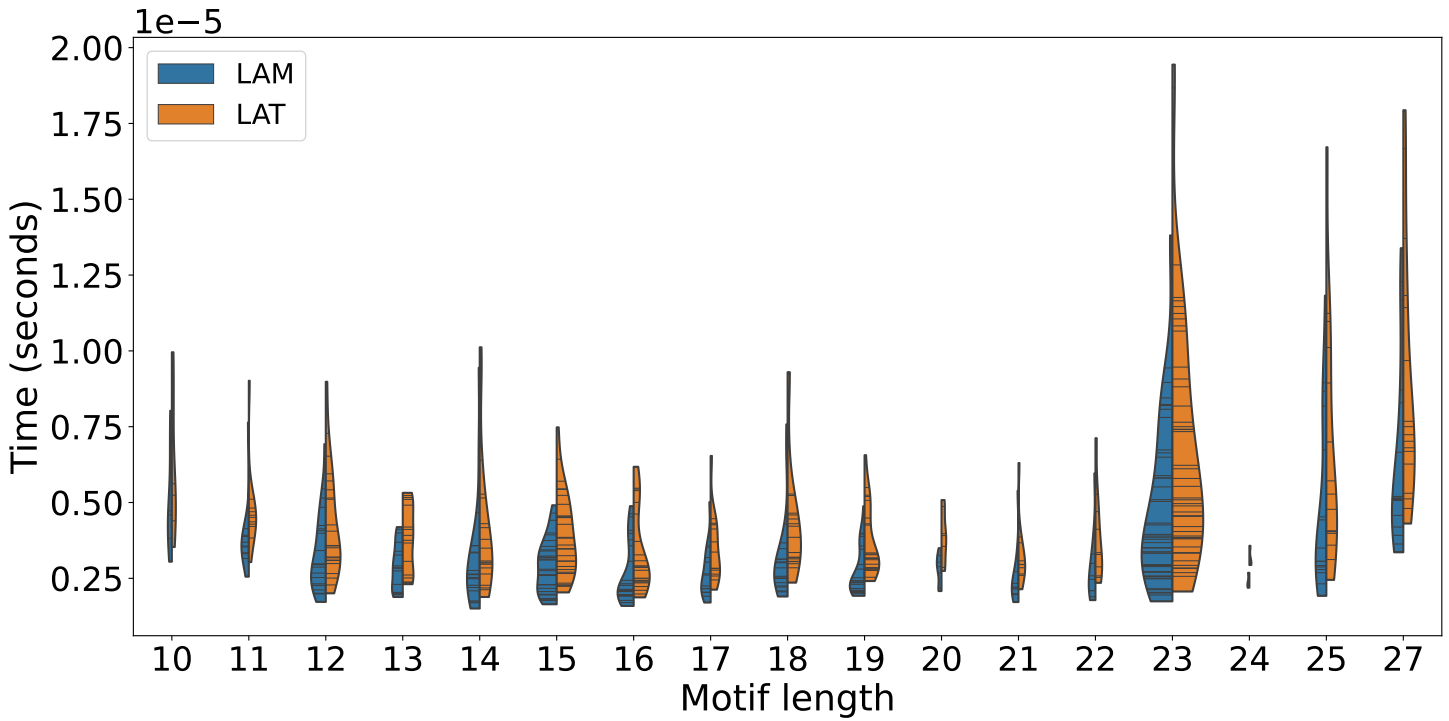
ZNF274 →



- Same length, but very different information content!
- Empty columns generate all possible substrings.



Average enumeration time for one valid word



- Enumeration: time complexity is linear in output size.

Comparing running times: enumeration vs semi-naive

- Search in a 50 million nuc. sequence for ratio: 0.9 using LAM
- Scanning window vs enumeration strategy

di-PWM	motif length	Semi-naïve Search time	Enumeration + search times
COE1	23	221 s	68 + 27 = 96 s
CREB1	23	148 s	35 + 8 = 43 s
EGR1	21	117s	6 + 3 = 9 s
GATA1	21	140 s	20 + 7 = 27 s
ATF2	14	125 s	0.05 + 1.4 = 1.45 s
DUX4	14	79 s	0.0008 + 1 = 1.1s

Enumeration + search: efficient compared to naive search

Comparing running times: SPry-SARUS vs dipwmsearch

- We propose a more involved enumeration+search algorithm: **dipwmsearch**
- Java tool: SPRY-SARUS based on SuperAlphabet Representation
- All HOCOMOCO di-PWMs, four ratios: 0.8, 0.85, 0.9, 0.95

Conclusion and future work

What do we get?

- python module with API
- installation via conda
- IUPAC code allowed
- threshold as ratio
- other tricks for more speed

Conclusion and future work

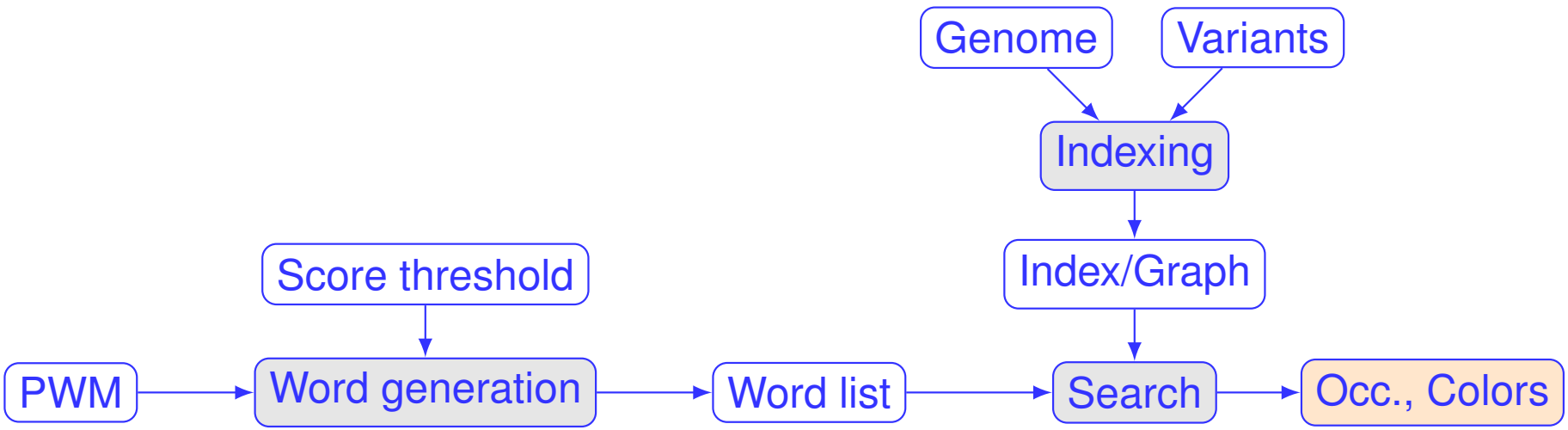
What do we get?

- python module with API
- installation via conda
- IUPAC code allowed
- threshold as ratio
- other tricks for more speed

What's next?

- Advanced online search: MPSCAN algorithm [Rivals et al. 2009]
- Offline search in indexed genomes or pan-genomes
- Combined search for multiple motifs
- Score distribution for di-PWMs
- Refinement of di-PWM matrices

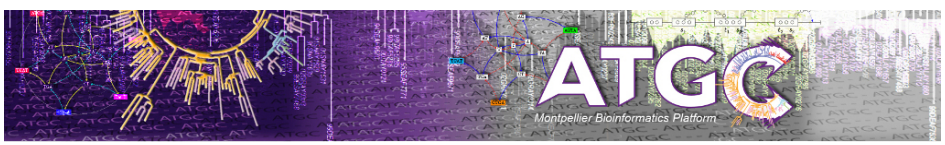
Indexed probabilistic motif search in pan-genomes



Funding and acknowledgments

Thanks for your attention

Questions?



MPSCAN: efficient and scalable set pattern matching

MPSCAN: Scalability & performance

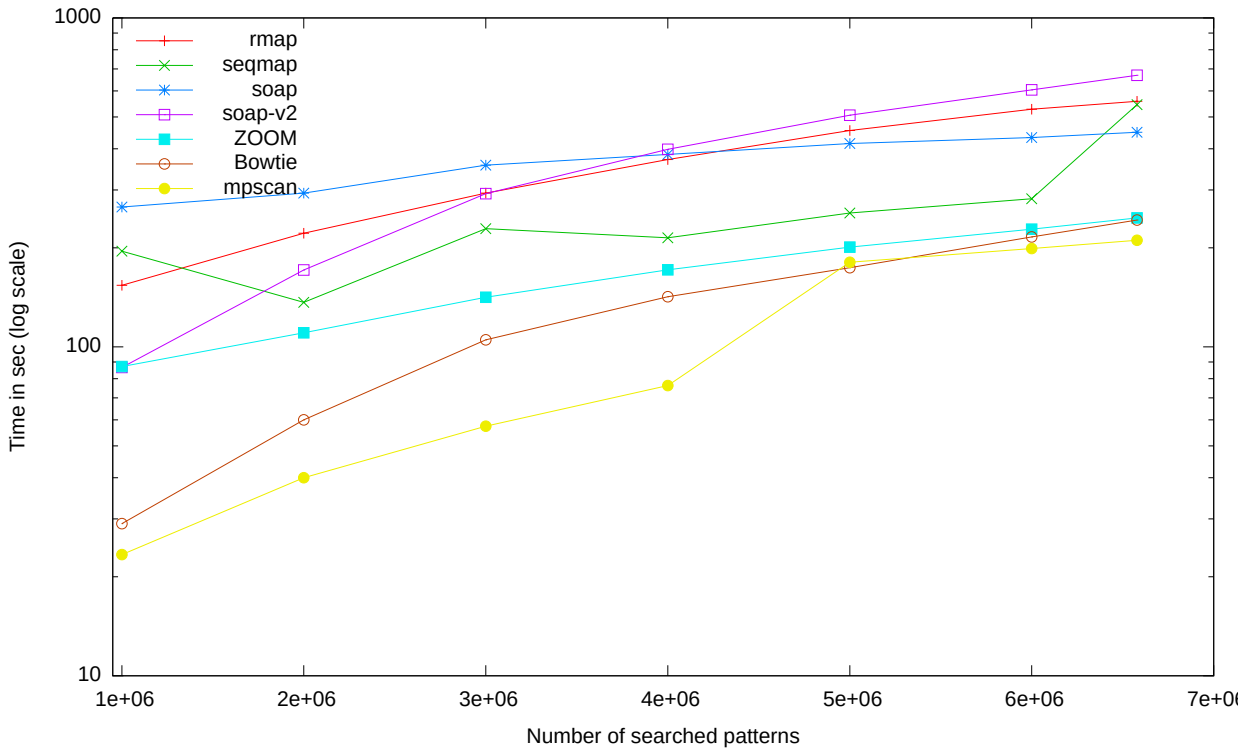


Figure: Exact mapping of 27 bp *ChIP-seq* reads on human chromosome 1

Average time complexity

Theorem:

The average time complexity of MPSCAN for searching r patterns of size l in a text of length n over an alphabet of size c is:

$$O(n \log_c(rl)/l) \qquad \text{if} \qquad q = \Theta(\log_c(rl)).$$

This average complexity is optimal (cf. [Navarro & Fredriksson, 04])