dipwmsearch: a python package for searching di-PWM motifs

Marie Mille, Julie Ripoll, Bastien Cazaux, Eric Rivals

▶ **To cite this version:**

Marie Mille, Julie Ripoll, Bastien Cazaux, Eric Rivals. dipwmsearch: a python package for searching di-PWM motifs. 2022. lirmm-03834008

HAL Id: lirmm-03834008

https://hal-lirmm.ccsd.cnrs.fr/lirmm-03834008

Preprint submitted on 28 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons `Attribution - NonCommercial - ShareAlike| 4.0 International License`

dipwmsearch: a python package for searching di-PWM motifs

Marie Mille, Julie Ripoll, Bastien Cazaux, Eric Rivals

October 28, 2022

Affiliations: Laboratory of Informatics, Robotics and Microelectronics (LIRMM), Université Montpellier, CNRS, Montpellier, France, Contact author: E. Rivals, *Email*: `rivals@lirmm.fr`.

Abstract
Motivation Seeking probabilistic motifs in a sequence is a common task to annotate putative transcription factor binding sites (TFBS). Useful motif representations include Position Weight Matrices (PWMs), dinucleotidic PWMs (di-PWMs), and Hidden Markov Models (HMMs). Dinucleotidic PWMs combine the simplicity of PWMs – a matrix form and a cumulative scoring function –, but also incoporate dependency between adjacent positions in the motif (unlike PWMs which disregard any dependency). For instance, to represent binding sites, the HOCOMOCO database provides di-PWM motifs derived from experimental data. Currently, two programs, SPRy-SARUS and MOODS, can search for di-PWMs in sequences.
Results We propose a Python package, *dipwmsearch*, which provides an original and efficient algorithm for this task (it first enumerates matching words for the di-PWM, and then search them at once in the sequence even if it contains IUPAC codes). The user benefits from an easy installation via *Pypi* or *conda*, a documented Python interface, and reusable example scripts that smooth the use of di-PWMs.
Availability and Implementation: *dipwmsearch* is available at https://pypi.org/project/dipwmsearch/ under Cecill license.

1 Introduction

Protein binding sites on nucleic acids (DNA or RNA) share similar, but not identical sequences. The collection of sequences of such binding sites, which in practice is a set of sequences (of identical length), are summarized and represented as a probabilistic motif. Often only a few positions within such sequences are conserved across a majority of binding sites. Even at a conserved position, when the collection is large enough, alternative nucleotides occur. Hence, for each position of the binding site, it is convenient to summarize its variability as the probability of each nucleotide to occur at this position. The probabilities are estimated from the frequencies of nucleotides at that position in the collection. This explains why the first and most popular probabilistic motif representation is the Position Weight Matrix (PWM) [8]. A PWM is a matrix containing the weight or score of each nucleotide at each position of the sequence alignment: the weights that are log-odd scores of the nucleotide probabilities at each position. Numerous search algorithms are available for PWMs [3, 2, 7]. However, in a PWM positions are entirely independent one of another; but in reality neighboring positions are constrained for they influence the shape of DNA, or the propensity to undergo epigenetic modifications, and hence the binding of the protein. Hence, a more complex representation for probabilistic motifs that accounts for local position dependencies was proposed: dinucleotidic PWM (di-PWM) [5]. At each position, one records the frequency of all 16 possible dinucleotides (instead of 4 nucleotides for PWM). A di-PWM and score computation of a word is illustrated Figure 1a.
For instance, the *HOCOMOCO* database (v11) stores di-PWMs of binding motifs of Human and Mouse transcription factors (292 and 257, respectively), which were directly computed from experimental ChIP-Seq data [6]. For detecting new binding sites, it was shown that di-PWMs provide enhance sensitivity compared to classical PWMs [5]. Results of DREAM-ENCODE challenge from 2017 tend to confirm this observation.
To our knowledge, only *SPRy-SARUS* and *MOODS* can find occurrences of di-PWMs in long sequences. *SPRy-SARUS* is an efficient stand alone Java program to search for di-PWMs, which is coupled with *MoLoTool*,

a webtool allowing visual inspection of occurrences in short sequences [6]. *MOODS*, a tool to search for PSSM/PWM, can also handle di-PWMs [4] (v3 available as C++ code and python package). Both adopt a window scanning strategy, while dipwmsearch uses an enumeration strategy.
Hence, we provide a Python package for searching di-PWM in sequences: it can be easily installed via *conda* and offers several functions that can be used in Python programs. We designed a novel search algorithm that differs from previous approaches. Running time comparisons demonstrate that our algorithm is on par with SPRy-SARUS in practice.

2 Search algorithm

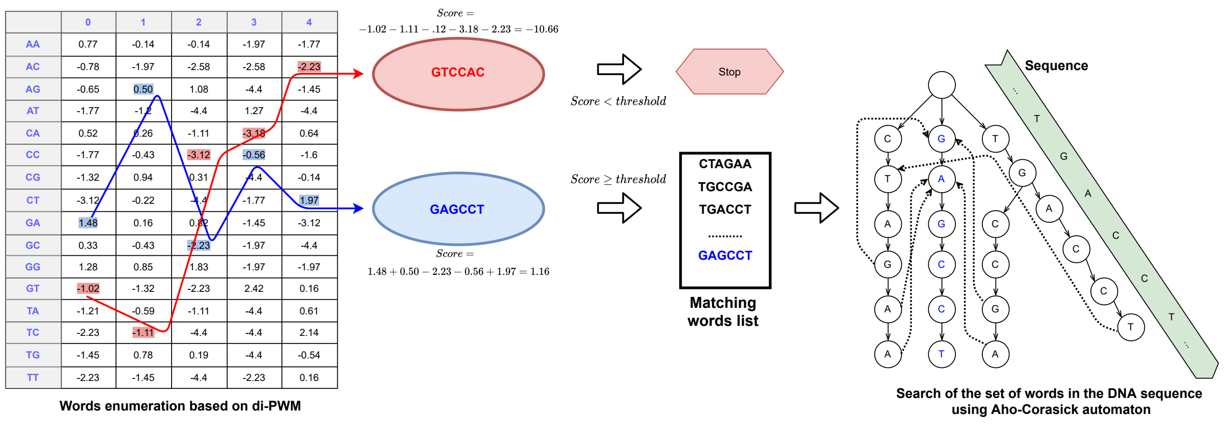
Our package provides distinct search algorithms: an optimized scanning algorithm (OS), an enumeration based algorithm for full di-PWM (FE), and core enumeration based algorithm (CE). For place sake, we described only CE algorithm for it is the most efficient. Algorithm viewpoint, we aim at proposing new approaches capable of searching for matches in long sequences, with limited amount of memory. Before explaining the algorithm, we give some rationale for our approach. Let us consider that in a text T, we seek a di-PWM P of size $\sigma^2 \times (m-1)$ (for a motif of length m) and with a score threshold t. An entry $P[\alpha\beta, i]$ gives the score of dinucleotide $\alpha\beta$ at position i in the motif. Remark that for any interval $[i..j]$ with $1 \leq i < j < m$, the restriction of P to this interval of positions, which we denote by $P[][i..j]$, is a smaller di-PWM of length $j-i+1$.

Traditional scanning algorithm and the enumeration strategy

In a traditional scanning algorithm, one considers each possible window of length m in T and computes its score according to P. It takes $O(m \times |T|)$ time, which is quadratic. The scanning approach implies redundant computation (for instance when processing identical or similar substrings whose score are too low) and often is inefficient. A classical speedup trick uses the LookAheadTable to stop the score computation after viewing only a prefix of the current window [2]; it does not improve the worst case complexity.
An alternative is to first enumerate all words of length m that match P with a score $> t$, which we call *valid words*, and then to search the valid words in T using an Aho-Corasick (AC) automaton [1] (or any other algorithm that solves the Set Pattern Matching problem). We implemented this in the enumeration based algorithm for full di-PWM (FE). We call this global idea the *enumeration strategy*; it concentrates the complexity in the enumeration phase, and makes the scanning efficient because it seeks `only exact` matches of the valid words – the scanning phase does not compute any score. Nicely, building the Aho-Corasick automaton takes linear time in cumulated length of valid words, and scanning T with it takes linear time in $|T|$.

Efficiency conditions for the enumeration strategy

To be efficient, the enumeration strategy needs 1/ a fast enumeration algorithm, 2/ a set of valid words that is small enough for the AC automaton to fit in memory (i.e., remain fast to build). Below, we exhibit a enumeration algorithm takes linear time in the output size, which resolves the first condition. However, the number of valid words depends on the selectivity of the di-PWM P with threshold t. The least selective position is when the scores of all 16 dinucleotides are equal. It turns out that some di-PWMs from HOCOMOCO contain positions that are not selective, i.e., in which the scores are almost equally distributed. A closer examination shows in such di-PWMs non selective positions often occur in intervals of successive positions (see Figure 2 in Supplemental Material). For such an interval of say f positions, if we consider P' the restriction of P to this interval, almost any word of length $f+1$ is a valid word for the di-PWM P'. This may lead to an explosion of valid words for the full di-PWM P.
We propose to identify selective and non-selective positions by considering the standard deviation of their scores: a large deviation means a selective position. To avoid cases with huge set of valid words, we propose to restrict P to an interval of selective positions, which we term the *core*. We proceed as follows: first we compute the standard deviation of scores for all positions, then we select, by exhaustive search among all possible intervals of length at least 10, the interval with the largest average standard deviation. This interval determines the core (which is a smaller di-PWM).



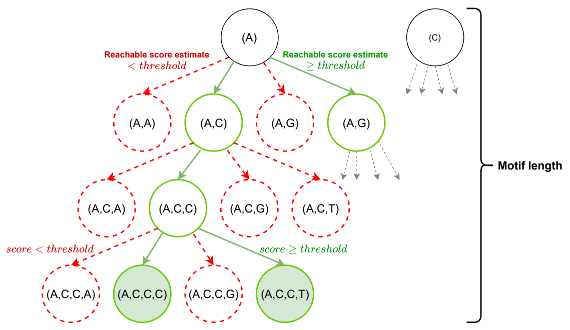
	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-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

Words enumeration based on di-PWM

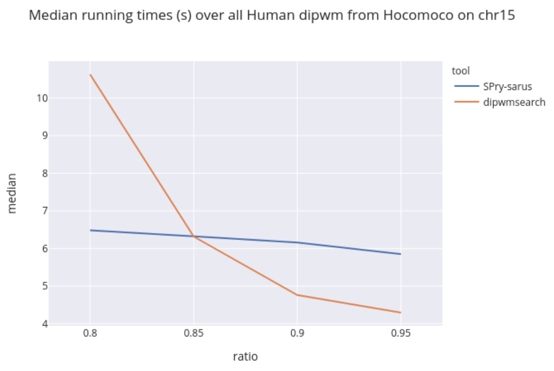
$Score =$
$-1.02 - 1.11 - .12 - 3.18 - 2.23 = -10.66$

$Score =$
$1.48 + 0.50 - 2.23 - 0.56 + 1.97 = 1.16$

(a) Enumeration and scanning strategy for a di-PWM



(b) Enumeration of valid words



(c) Running time comparisons

Figure 1: (**a**) Enumeration and scanning strategy for a di-PWM. Left part shows how the score of two words are computated by summing the score of their five dinucleotides. If the score lies above the threshold, the word is a valid word and is added to the list for later search. Right part: we build an Aho-Corasick automaton with all valid words in the list, then use the automaton to scan the sequence.
(**b**) Illustration of the branch and bound strategy for the enumeration procedure. We build a trie for words starting with letter *A*, and explore it in Depth-First manner. As soon as a prefix cannot give rise to a valid word, which is determined using the LookAheadMatrix (LAM), we cut the corresponding branch. Only valid words generate a leaf in the trie. (**c**) Comparison of SPRy-SARUS and dipwmsearch for searching all Human di-PWMs from HOCOMOCO on Human chromosome 15. The plot shows the median running time over all di-PWMs.

Enumeration of valid words: B&B approach and LAM

For a full di-PWM, we propose an algorithm that explores a trie data structure of valid words using a Branch-and-Bound approach (see Figure 1b). We build a trie that spells out prefixes of potential valid words, one letter at a time. After each letter, assume the current prefix has length k, we compute the partial score for this prefix. Then, we check the score for the best possible suffix of length $m-k$ in an additional matrix called the LAM. If the sum of prefix and suffix scores does not reach the threshold t, then extensions of the current branch of the trie are unnecessary. The LookAheadMatrix (or LAM for short) is a precomputed $\sigma \times (m-1)$ matrix that depends only on P. For a position i in P and a symbol α, the $LAM[\alpha, i]$ stores the best score for a suffix starting with symbol α at position i. Algorithm 1 computes the LAM in $O(\sigma^2 \times (m-1))$ time. The LAM has a crucial property: for any stored score value in the LAM, there exists a word that realises this score. This ensures that only branches of the trie corresponding to valid words are fully built by the enumeration algorithm. Moreover, the amount of computation spent between two successive valid words is bounded by $2m$, which implies that our algorithm takes linear time in the output size.
Note that a pendant matrix to the LAM can be built symmetrically to compute the best scores of prefixes of P. We call this matrix, the LookBackMatrix or LBM.
After enumeration, in the search step, the set of valid words for P are searched for in T using an Aho-Corasick automaton [1].

Adapting the enumeration strategy and search with the core

The enumeration algorithm and search phase must be adapted to use the core instead of P. Assume the core, denoted by Q, starts at position $k+1$ in the motif and has length $h-1$. We must enumerate words of length h for Q that are substrings of valid words of length m for P. We run the branch & bound algorithm described above to spell out words of length h according to Q, but we cannot select them on their own score (which is a sum only over $h-1$ positions!). We must use the score of a prefix of P, not a prefix of Q. Assume the current prefix w starts with letter α at position $k+1$ and ends with letter β; as score, we use score(xwy), where x is a highest scoring valid prefix for P of length k ending with letter α, and y a highest scoring suffix of length $m-k-|w|+1$ starting with letter β, and such that $|xwy| = m$. The idea behind is that xwy is the best possible word of length m with substring w (at positions $[k+1, k+|w|+1]$). The constraints on the letters are implied by the fact that successive positions of a di-PWM score overlapping dinucleotides. We use the LAM to get the contribution of y to this score (without knowing y) and we use the LBM to get that of x (without knowing x). The algorithm outputs the set of all words w of length h that occur in at least one valid word for P (at position $[k+1, k+h+1]$).
The search phase builds an AC automaton with this set and scan T with it. Each time a match is found, say at position i, it computes the score of the window of T between positions $(i-k+1)$ and $(i-k+m)$. If the scores reaches the threshold t it reports a match of P at position $(i-k+1)$ in T, and its score.

3 Results

We compare the core motif algorithm to SPRy-SARUS in terms of efficiency by searching each Human di-PWM from HOCOMOCO on Human chromosomes 3 and 15 for four different score ratios (0.8, 0.85, 0.90, and 0.95). First, this confirms that core algorithm is able to search any di-PWM with reasonable score ratios. Figure 1c displays the median search time over all di-PWMs for both tools and shows first, that dipwmsearch offers affordable runtimes whatever the ratio, and second that dipwmsearch takes longer times than SPRy-SARUS for ratio 0.80, while it is as efficient or faster for larger ratios.

4 Conclusion

Our Python package, *dipwmsearch*, provides an easy use of an efficient procedure to search di-PWM in nucleotidic sequences, through a set of documented snippets. It offers practical advantages compared to an existing solution (like processing IUPAC codes, or an adaptable output – see Supplementary Materials) and can be enhanced by combining it with other Python packages (e.g., for processing compressed sequence files). Most

of all, installation is straighforward using *pypi* or *conda*. In addition, we presented an original, enumeration based search algorithm that can handle di-PWMs even if they contain non selective positions – which seems novel. Coping with non selective positions was necessary to make search effective for some di-PWMs, which questions their information content, and in turn their construction process. Examining the set of valid words and their occurrences could help determining if a di-PWM model is truely well suited.
Several perspectives come to mind. First, once enumerated, the set of valid words can stored in a file and reused for other searches. Second, the search phase can be streamlined by using a precomputed index of the search sequence to find valid words, which would enable a web application to support numerous di-PWM searches.

References

[1] A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[2] M. Beckstette, R. Homann, R. Giegerich, and S. Kurtz. Fast index based algorithms and software for matching position specific scoring matrices. *BMC Bioinformatics*, 7(1), Aug 2006.

[3] J. Korhonen, P. Martinmäki, C. Pizzi, P. Rastas, and E. Ukkonen. MOODS: fast search for position weight matrix matches in DNA sequences. *Bioinformatics*, 25(23):3181–3182, 2009.

[4] J. H. Korhonen, K. Palin, J. Taipale, and E. Ukkonen. Fast motif matching revisited: high-order PWMs, SNPs and indels. *Bioinformatics*, 33(4):514–521, Dec 2017.

[5] I. Kulakovskiy, V. Levitsky, D. Oshchepkov, L. Bryzgalov, I. Vorontsov, and V. Makeev. From binding motifs in chip-seq data to improved models of transcription factor binding sites. *Journal of Bioinformatics and Computational Biology*, 11(01):1340004, Feb 2013.

[6] I. V. Kulakovskiy, I. E. Vorontsov, I. S. Yevshin, R. N. Sharipov, A. D. Fedorova, E. I. Rumynskiy, Y. A. Medvedeva, A. Magana-Mora, V. B. Bajic, D. A. Papatsenko, and et al. HOCOMOCO: towards a complete collection of transcription factor binding models for human and mouse via large-scale ChIP-Seq analysis. *Nucleic Acids Research*, 46(D1):D252–D259, Nov 2018.

[7] D. Martin, V. Maillol, and E. Rivals. Fast and accurate genome-scale identification of dna-binding sites. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 201–205, 2018.

[8] G. D. Stormo. DNA binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, Jan 2000.

5 Supplementary Material

Access to the packages, the source code and the documentation.

1. Python package: `https://pypi.org/project/dipwmsearch/`
2. Documentation: `https://rivals.lirmm.net/dipwmsearch/`
3. Conda package: `https://anaconda.org/atgc-montpellier/dipwmsearch`
4. Source code: `https://gite.lirmm.fr/rivals/dipwmsearch`
5. Contact: dipwm@lirmm.fr

Practical comparison of SPRy-SARUS, MOODS, and dipwmsearch

We summarize some features and practical differences between the tools, SPRy-SARUS, MOODS, and dipwmsearch, in the table below.

Feature	SPRy-SARUS	MOODS v3	dipwmsearch
conda installation	n	y	y
pypi installation	n	y	y
IUPAC code (except N)	blocking	y	y
API programming library	n	y	y
output matching word	n	y	y
compressed sequence file	n	y	y (via gzip/bzip python packages)
multi FASTA	y	y	y (example)

Simplicity of use: an example of python code

To illustrate how easily one can use dipwmsearch in python, we provide an example of code for searching one di-PWM motif in a FASTA sequence for a given threshold, exactly as SPRy-SARUS would perform it. This script takes three arguments on the command line: first, the file containing the di-PWM, second, the file containing the FASTA formatted sequence, third the threshold ratio as a real value.

```python
import sys, os
import dipwmsearch as ds
from Bio import SeqIO
from Bio.Seq import Seq

pathSeq = sys.argv[1]
pathDiPwm = sys.argv[2]
threshold = float(sys.argv[3])


# read diPWM, sequence
diP = ds.create_diPwm(pathDiPwm)
file = open(pathSeq)
seqRecord = SeqIO.read(file, "fasta")

mySeq = str(seqRecord.seq.upper())
seq = mySeq.translate(mySeq.maketrans("NMRWSYKVHDB","GGGGGGGGGGG"))

# print(threshold)

# Block optimized search
number_matches_block_opt = 0
# 1st loop to search valid words on Watson strand
for i, word, score in ds.search_block_optimized(diP, seq, threshold):
	number_matches_block_opt += 1

# reverse complement sequence
seq_rev = seq.translate(seq.maketrans("ACGT","TGCA"))
seq_rev = seq_rev[::-1]

# 2nd loop to search valid words on Crick strand
for i, word, score in ds.search_block_optimized(diP, seq_rev, threshold):
	number_matches_block_opt += 1

dipwm_name = os.path.split(pathDiPwm)[1].split(".")[0]
print(f'{dipwm_name}\t{threshold}\t{number_matches_block_opt:,}')
```

LAM: definition and algorithm

Definition 5.1 (*LookAheadMatrix* for di-PWM) *The* LookAheadMatrix *M of a di-PWM P of a motif of length m on an alphabet Σ of size σ is a matrix of size taille $\sigma \times (m-1)$, where for any i such that $0 \leq i \leq m-2$ and for any $d \in \Sigma$:*

$$M[d,i] := \begin{cases} \max_{b\in\Sigma} P[db,i], & \text{if } i = m-2 \\ \max_{b\in\Sigma} (P[db,i] + M[b,i+1]), & \text{if } 0 \leq i < m-2 \end{cases}$$

Algorithm 1: MakeLookAheadMatrix: compute the LookAheadMatrix of a di-PWM P.

```
Input: Alphabet $\Sigma$ of size $\sigma$,di-PWM matrix $P$ of size $\sigma^2 \times (m-1)$, for a motif of length $m$
Output: LookAheadMatrix $M$ of size $\sigma \times (m-1)$
1  $M \leftarrow$ initialized with $-\infty$
2  for $d \in \{0,\dots,\sigma-1\}$ do
3      $max \leftarrow -\infty$
4      for $b \in \{0,\dots,\sigma-1\}$ do
5          $score \leftarrow P[db, m-2]$
6          if $score > max$ then $max \leftarrow score$
7      $M[d, m-2] \leftarrow max$
8  for $i \in \{m-3,\dots,0\}$ do
9      for $d \in \{0,\dots,\sigma-1\}$ do
10         $max \leftarrow -\infty$
11         for $b \in \{0,\dots,\sigma-1\}$ do
12             $score \leftarrow P[db, i] + M[b, i+1]$
13             if $score > max$ then $max \leftarrow score$
14         $M[d, i] \leftarrow max$
15 return $M$
```

Non selective positions and core

An example of di-PWM with non selective positions, for GATA2 transcription factor, is shown Figure 2. The height of the letters at each motif position in the LOGO above the line, indicate the information content of the corresponding nucleotide at this position. The letters first 12 positions are nearly invisible compared to those between position 13 until 22.

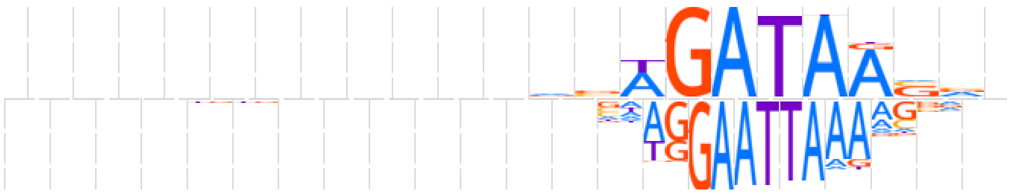


Figure 2: LOGO representation of the Human di-PWM for the binding site motif of transcription factor GATA2. The motif 23 bp long and its consensus sequence is *nnnnnnnnnnnnnvvWGATAASvn*. The interval with the first 12 positions at 5' end contains non selective positions (with little information content), as is the last 3' end position. Hence, the core is restriction to interval [13, 22] of positions.

Protocol for comparing running times

Currently, it is not possible to search for di-PWM using MOODS v3: MOODS cannot read the format for di-PWM matrices containing scores (an issue was sent on the github repository on Sep 22, 2022). This is why MOODS was not used in the comparisons.
Comparisons between dipwmsearch and SPRy-SARUS were performed as follows, using two bash scripts on a Linux system. Technical information are summarised below.
For both chromosome sequences (15 and 3), for all Human di-PWM motifs of HOCOMOCO database, for four ratios (0.8, 0.85, 0.9, 0.95), we recorded the running time of dipwmsearch and SPRy-SARUS using the `time` command. Each search for a di-PWM was performed in a distinct execution of the tools, implying that each time the target sequence and the di-PWM were loaded, the enumeration of valid words and then the scanning of the sequence performed, before storing the results in an output file on the disk. We did not take advantage of

the possibility for dip to search for a multi FASTA file containing both sequences at once (which would have avoided to redo the enumeration step).
For dipwmsearch, we used a python script that calls the procedure `search_block_optimized` for searching the di-PWM first on the Watson strand and then on the Crick strand. The time was measured using the `time` command from Python package `time`. For SPRy-SARUS, we recorded the time taken by program in user mode, using the `/usr/bin/time` command with option `-f "%U"`. Both tools were run using a single thread; despite this, it occurs that the JAVA machine running SPRy-SARUS used more than 100% of the CPU.

- Technical information The last version of SPRy-SARUS (release: 2.0.2) was obtained from its github repository `https://github.com/autosome-ru/sarus`.

  Linux version

  Linux 5.11.0-40-generic #44~20.04.2-Ubuntu SMP Tue Oct 26 18:07:44 UTC 2021 $x86_{64}$ $x86_{64}$ $x86_{64}$ GNU/Linux

  Java version

  - openjdk version "10.0.2" 2018-07-17
  - OpenJDK Runtime Environment Zulu10.3+5 (build 10.0.2+13)
  - OpenJDK 64-Bit Server VM Zulu10.3+5 (build 10.0.2+13, mixed mode)

Variation of running time with respect to the HOCOMOCO di-PWM

The summary of results reported in the article gives the median running times for all HOCOMOCO di-PWM motifs. As mentioned above, the di-PWMs of HOCOMOCO vary in information content and thus in selectivity. This implies that the number of valid words strongly depends on the motif and the ratio, and hence, so do the enumeration and overall running times.
The Figures 3 provides a view on the variability of the running times in function of the di-PWM and ratio, for both SPRy-SARUS (left plot) and dipwmsearch (right plot). We report the running times for searching on Human chromosomes 15 and 3. Note that because SPRy-SARUS can only process sequences containing A, C, G, T (but no other IUPAC codes), the sequence of chromosome 3 was slightly modified to replace undetermined positions by one standard nucleotide (before feeding SPRy-SARUS). For dipwmsearch, we used the normal chromosome 3 sequence with IUPAC codes.
The variation of running times in comparison to the median running time is higher for dipwmsearch than for SPRy-SARUS. For dipwmsearch, it depends on the matrix and ratio, while SPRy-SARUS times depend mostly on the sequence length. However, the overal running times of dipwmsearch remain fast enough for practical uses.

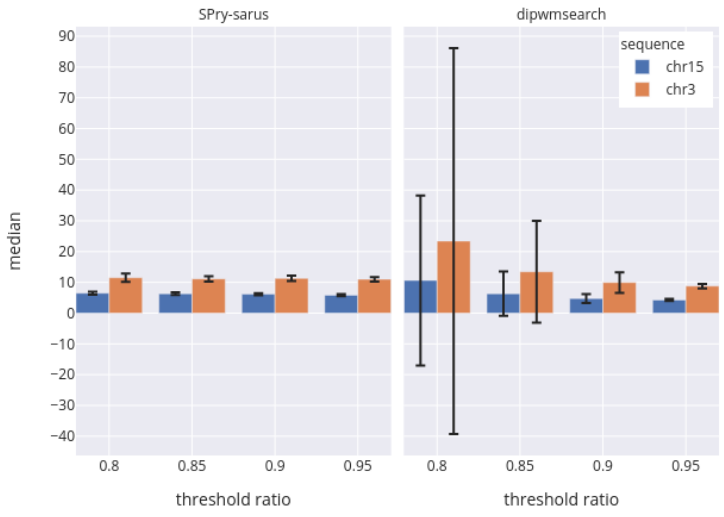


Figure 3: Median running times and standard deviations of both dipwmsearch and SPRy-SARUS for searching each Human di-PWMs from HOCOMOCO on Human chromosomes 15 and 3.