BMC Bioinformatics

SOFTWARE

Open Access

# BIOMAPP::CHIP: large-scale motif analysis

Jader M. Caldonazzo Garbelini[1*], Danilo S. Sanches[2†] and Aurora T. Ramirez Pozo[1†]

†Danilo S. Sanches and Aurora T. Ramirez Pozo have contributed equally to this work.

*Correspondence: jmcgarbelini@inf.ufpr.br

[1] Department of Informatics, Federal University of Parana, XV de Novembro Street, Curitiba, Parana 80060000, Brazil
[2] Department of Informatics, Federal University of Technology, Alberto Carazzai Avenue, Cornelio Procopio, Parana 86300000, Brazil

## Abstract

**Background:** Discovery biological motifs plays a fundamental role in understanding regulatory mechanisms. Computationally, they can be efficiently represented as *kmers*, making the counting of these elements a critical aspect for ensuring not only the accuracy but also the efficiency of the analytical process. This is particularly useful in scenarios involving large data volumes, such as those generated by the *ChIP-seq* protocol. Against this backdrop, we introduce BIOMAPP::CHIP, a tool specifically designed to optimize the discovery of biological motifs in large data volumes.

**Results:** We conducted a comprehensive set of comparative tests with state-of-the-art algorithms. Our analyses revealed that BIOMAPP::CHIP outperforms existing approaches in various metrics, excelling both in terms of performance and accuracy. The tests demonstrated a higher detection rate of significant motifs and also greater agility in the execution of the algorithm. Furthermore, the SMT component played a vital role in the system's efficiency, proving to be both agile and accurate in *kmer* counting, which in turn improved the overall efficacy of our tool.

**Conclusion:** BIOMAPP::CHIP represent real advancements in the discovery of biological motifs, particularly in large data volume scenarios, offering a relevant alternative for the analysis of *ChIP-seq* data and have the potential to boost future research in the field. This software can be found at the following address: (https://github.com/jadermcg/biomapp-chip).

**Keywords:** Motif discovery, Chip-seq, Kmer counting, Optimization

## Background

*Motifs* are subsequences of length $k$ that occur with high frequency in a set of sequences of DNA, RNA, or proteins. Formally, consider an alphabet $\Sigma$, which can be $\{A, C, G, T\}$ for DNA, $\{A, C, G, U\}$ for RNA, or a set of amino acids for proteins. A sequence $s$ is an ordered list of symbols from $\Sigma$ and a *motif* $m$ is a subsequence of $s$ such that $m \in \Sigma^k$, where $k$ is the length of the *motif*. The discovery of MOTIFS involves identifying these subsequences that frequently occur in a set of sequences, possibly with some variations [4, 5].

It is important to note that *motifs* can be represented through *kmers* (contiguous subsequences of a given size k), making the efficient counting of these structures highly relevant. The representation of *motifs* as *kmers* allows for the simplification of complex

computational problems, enabling efficient counting algorithms to be applied as a preliminary step in identifying biologically significant sequences [18].

In *motif* discovery, there are two main approaches: enumerative techniques and probabilistic techniques. Both have their own advantages and disadvantages, and the choice between them depends on the specific needs of the analysis at hand. Enumerative techniques, as the name suggests, aim to identify *motifs* by exhaustively enumerating all possible subsequences within a set of sequences. These methods are generally more accurate as they consider all possibilities. However, they are computationally intensive and may not be feasible for large volumes of data or for *motifs* of significant length [10].

On the other hand, probabilistic techniques such as *Gibbs Sampling* (GS) and *Expectation Maximization* (EM) employ statistical models to estimate the presence of *motifs*. These methods are generally faster in handling large volumes of data; however, they are sensitive to initial conditions and model parameters, which may compromise accuracy. Both approaches have their merits: while enumerative techniques are typically more accurate, probabilistic techniques are more time-efficient. Nevertheless, it is possible to integrate these two approaches to leverage the advantages of both. For example, one could use an enumerative approach to filter candidates and a probabilistic approach for final optimization [15].

This interconnection between *kmer* counting and *motif* discovery highlights the importance of efficient algorithms in both domains for the acceleration and enhancement of bioinformatics analyses. In this context, algorithms for *kmer* counting serve as initial tools in the processing chain for discovering biological *motifs*, especially in analyses that involve large volumes of data, such as *ChIP-seq* (Chromatin Immunoprecipitation followed by Sequencing) data.

The *ChIP-seq* (Chromatin Immunoprecipitation followed by Sequencing) protocol is a modern technology that allows the identification of DNA-protein interactions on a genomic scale. This method has become an indispensable tool in the discovery of biological *motifs*, as it provides a comprehensive mapping of protein-binding regions throughout the genome. One of the major challenges of *ChIP-seq* is the substantial volume of data generated. The handling, storage, and analysis of such data require robust computational infrastructure and efficient algorithms, as traditional *motif* discovery techniques may not be suitable for such a magnitude of data [9].

In addition to the data volume challenge, complexity and the presence of noise are also significant factors. The *ChIP-seq* protocol is susceptible to various types of artifacts and noise that can introduce errors in *motif* identification. Therefore, robust preprocessing and filtering methods are required before the *motif* discovery stage itself. Many current *motif* discovery algorithms were not designed to handle the challenges imposed by *ChIP-seq*, such as large data volumes and complexity in sequence structure. This often results in a trade-off between accuracy and computational efficiency [12].

In this context, BIOMAPP::CHIP (*Biological Application for ChIP-seq data*) is designed to address these challenges. Our algorithm adopts a two-step approach for *motif*

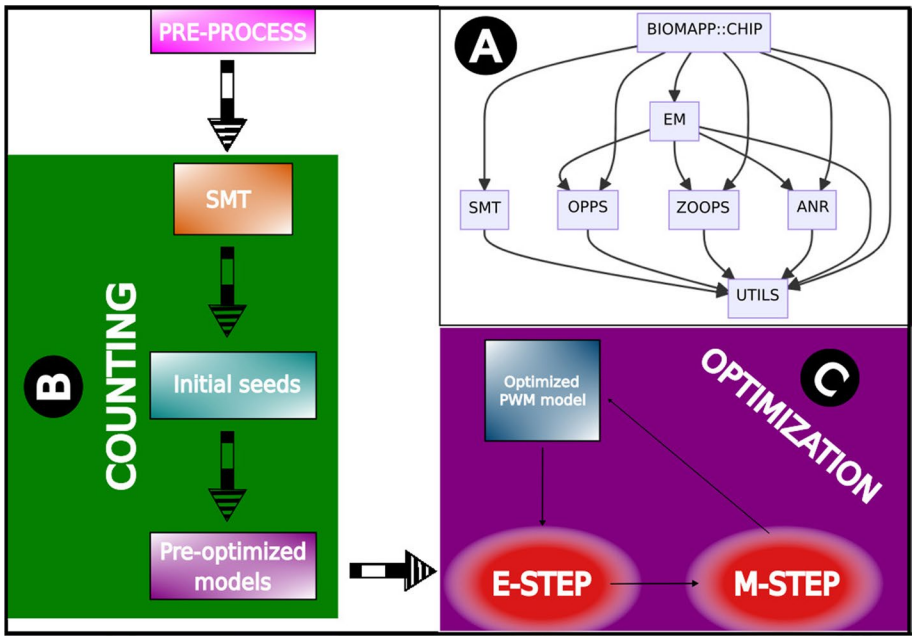


**Fig. 1** BIOMAPP::CHIP framework pipeline. (A) The main workflow is structured in various interconnected modules. Each module is responsible for a specific part of the overall functionality of the package. Preprocessing involves identifying and normalizing peaks in the data. (B) In the counting step, the dataset is loaded, a background model is generated, followed by the construction of the SMT. Enriched *kmers* are extracted and initial models are created. (C) Optimization occurs by refining the seeds using the FAST-EM algorithm

discovery: counting and optimization. In the counting phase, the SMT (*Sparse Motif Tree*) is employed for efficient *kmer* counting, enabling rapid and precise analysis. For the optimization stage, BIOMAPP::CHIP employs an enhanced version of the EM algorithm, aimed at improving accuracy in *motif* identification.

## Implementation

Implemented in C++ and R, BIOMAPP::CHIP combines analytical and numerical methods optimized for in-depth data treatment, particularly data derived from *ChIP-seq* experiments. The objective of this approach is to build effective solutions that assist researchers and experts in the field of molecular biology, more specifically in the study of conserved sequences, thereby facilitating complex investigations that involve sequence motif analysis. Through the appropriate combination of methods, some already established and others specifically created for our framework, BIOMAPP::CHIP offers a robust and adaptable tool for the scenario of functional genomics. Figure 1 illustrates the general pipeline, all the framework modules, and their respective interactions.

As illustrated in Fig. 1, the framework initiates its process with the pre-processing of the input sequences. These are then forwarded to the SMT, responsible for the efficient counting of *kmers*, generating seeds that have already gone through an initial stage of optimization as a result. These seeds are subsequently refined by the FAST-EM algorithm.

The architecture is modular and composed of seven main modules: UTILS, SMT, OOPS, ZOOPS, ANR, EM, and BIOMAPP::CHIP. The UTILS module acts as the backbone of the system, providing essential functionalities used by all the other modules.

The SMT module is developed on the foundation provided by the UTILS, and the OOPS, ZOOPS, ANR, and EM modules are likewise constructed on this foundation. In addition to being supported by the UTILS, the EM module has interdependencies with the OOPS, ZOOPS, and ANR modules. The BIOMAPP::CHIP module, in turn, represents the highest level of complexity as it integrates the functionalities of all the previous modules to execute its operations. Parallelization is supported in all modules through the OPENMP (https://www.openmp.org/) and THREAD BUILDING BLOCKS (https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html) libraries. The source code of BIOMAPP::CHIP is available at: https://github.com/jadermcg/biomapp-chip.

### Preprocessing

Before delving into the method description, it is important to understand how the data are acquired. They can be obtained from various public and private sources. For more information on the *ChIP-seq* databases used in this work, please refer to "Results and discussion" Section. The data acquisition process can be summarized in the following steps:

1. Experiment execution: the proteins of interest are immunoprecipitated along with the associated DNA fragments.
2. Raw data processing: the raw sequencer data are processed to obtain short DNA sequences, called READS.
3. Peak identification: specific programs, such as MACS [21] or SICER [20], are used to identify regions of READS enrichment, called peaks.
4. Pre-processing: removal of peaks in non-specific regions, peak filtering based on quality criteria, data normalization, and removal of spurious peaks.
5. Analysis: performing the analysis of interest, which may include: MOTIF extraction, functional annotation, regulatory network analysis, conservation analysis, among others.

The steps 1 and 2 are part of the biological experiment and are generally conducted by a biologist. Step 3, called enrichment analysis, aims to identify the peak regions, ranging from 100 to 300 bp[1], where there is a high probability that the fragments of interest are present. Step 4 is critical as it involves the removal of repetitive and low-complexity sequences. The final step consists of the analysis of the pre-processed data.

The BIOMAPP::CHIP starts operating at the end of step 3, where it receives as input the peak regions properly extracted from the genome. In step 4, specialized algorithms are used for pre-processing, and step 5 is fully implemented. The BIOMAPP::CHIP consists of three main phases: pre-processing, initialization, and optimization.

[1] The resolution of the READS heavily depends on the sequencing technology employed.

The pre-processing phase involves adapting the dataset for the execution of experiments, such as standardizing the size of the sequences and converting all characters to uppercase. However, the main goal of this stage is to identify low-complexity sequences, transposable elements such as ALUS (*Arthrobacter luteus*) and SINES (*Short Interspersed Nuclear Elements*), *E. coli* insertions, or any other components that could lead to inaccurate conclusions in subsequent steps. This is carried out with the help of specialized algorithms like DUST [19] and REPEAT MASKER [17]. This phase involves the elimination of redundant sequences and error correction, ensuring the quality and integrity of the data.

#### Counting

The counting phase takes as input pre-processed peak regions, where it is presumed that there are fragments from the distribution of interest surrounded by background distribution elements. The purpose of the initialization algorithm is to efficiently identify, count, and group enriched *kmers*, as well as to perform hypothesis tests to determine their statistical significance. The core of this stage consists of two components specifically developed for this purpose: i) SMT: a scalable data structure based on suffix trees responsible for storing all *kmers* from the main dataset; ii) KDIVE: an efficient algorithm whose objective is to search for fragments with up to $d$ degrees of mutation.

The counting phase follows a set of steps to identify enriched *kmers* in the main dataset and generate initial models based on this information. First, the algorithm loads the pre-processed data, which contains the nucleotide sequences of interest. Next, if available, control sequences are loaded. Otherwise, they are generated by shuffling the main dataset using the MARKOV method [6] or the EULER method [1]. A background probabilistic model is then generated from the control data, which will serve as a reference for identifying enriched *kmers*. The value of `k` is then estimated, or a pre-defined value is used, depending on the approach adopted. Based on the peak regions, the SMT data structure is built, allowing the efficient extraction of enriched *kmers*.

##### *SMT*

SMT is represented by a two-dimensional data structure $M^{\nu \times 6}$, where $\nu$ is the number of nodes, implemented from fixed-width text fragments. It is lightly inspired by ROOM squares theory [2], in which each element $M_{i,j}$ is either empty or has a value set between 1 and $\nu$. Its main goal is to efficiently store all *kmers* belonging to the main dataset, along with their representation and count, allowing for rapid searches. Thus, each row of $M$ represents a node $\{1, 2, 3, \ldots, \nu\}$, columns 1 to 4 represent the nucleotides, and the last two columns, 5 and 6, represent the number of times a fragment appeared in the dataset and its respective address or numeric representation. Figure 2 graphically shows how this process occurs.

In panel (a), we observe the extraction of six overlapping k-mers from the `ACGTAC GAT` DNA sequence, where each *kmer* is visually distinguished by a specific color. This color overlay method makes it easy to follow how the *kmers* are interspersed throughout

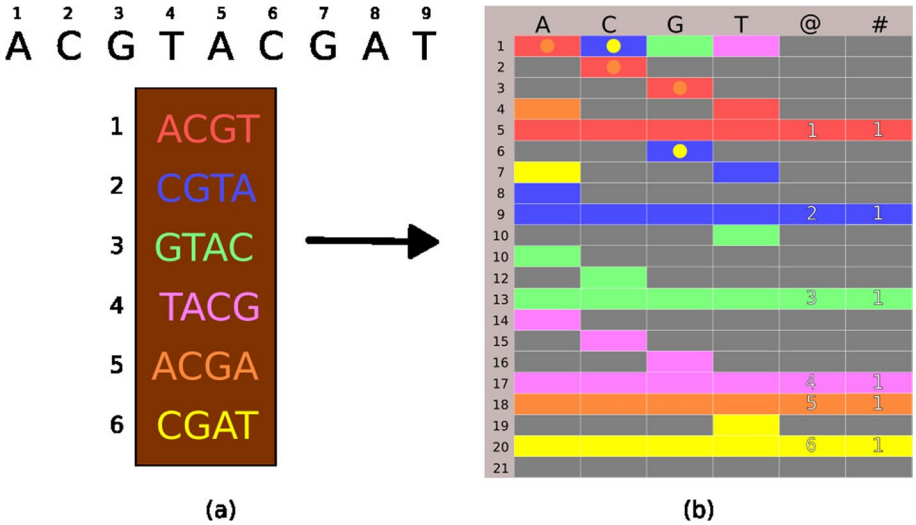


**Fig. 2** Example of the SMT functionality. **a** demonstrates the extraction of overlapping *kmers* from the DNA sequence ACGTACGAT, where each *kmer* is represented by a unique color, facilitating the visualization of the overlap. **b** Illustrates the subsequent addition of the *kmers* to the SMT, with the colored cells representing the individual fragments. The columns labeled with @ indicate the addresses of each *kmer* in the original sequence, while the columns marked with # display the frequency. Fully colored lines denote the totalization nodes, where the sums of the frequencies and the compilation of the addresses are calculated. Circles inside cells denote that they have been visited more than once

the sequence. Moving forward to panel (b), these *kmers* are integrated into the SMT. The structure of SMT is represented by a matrix, in which *kmers* are positioned according to their corresponding nucleotide sequences. Each cell in the matrix is color-coded according to the *kmer* it represents, creating a visual mapping that reflects the composition and frequency in the original DNA sequence. The columns in the matrix marked with the @ symbol record the addresses, indicating the specific location of each *kmer* in the DNA sequence from which they were extracted. Conversely, columns marked with the # symbol account for the frequency, providing a quantitative means to assess the recurrence of each sequence within the dataset. Notably, the fully colored lines within the matrix denote the so-called totalization nodes of the SMT. These nodes are fundamental to the tree structure, as they aggregate counts and consolidate their respective positioning information. Through this mechanism, SMT captures the diversity and distribution of *kmers* in the sequence and also provides a cumulative summary that is essential for subsequent analyses.

For example, if $M_{1,T} = 10$, then the symbol T is present between nodes 1 and 10. Its construction has time complexity $O(nk)$, where $k$ is the fixed size and $n$ is the number of fragments. However, $k$ does not scale with $n$ and has a well-defined limit falling between $5 \leq k \leq 35$, so we can simplify the analysis to $O(n)$ which is linear in $n$. Furthermore, it is possible to run several algorithms on SMT in linear time, such as exact search and approximate search, which, for example, allow quick recovery of the most abundant *kmers* in the dataset. The approximate search is performed by the KDIVE algorithm that will be presented later. Algorithm 1 shows how the SMT is constructed.

**Algorithm 1** CREATE-SMT

**Require:** $S = \{s_1, s_2, s_3, \ldots, s_n\}$, the set of kmers
**Require:** $k$, the width of the kmers
1: $n \leftarrow$ number of nodes
2: $M \leftarrow \text{zeros}(n, 4)$ ▷ Creates an empty matrix with $n$ rows and 4 columns.
3: root $\leftarrow 1$ ▷ The root is set to 1 and the new node to 2.
4: new_node $\leftarrow 2$
5: **for** $s \in S$ **do** ▷ Processes all sequences $\{s_1, s_2, s_3, \ldots, s_n\}$ starting from the root, each with width `k`.
6: node = root
7: **for** $i = 1$ to $k$ **do**
8: symbol = $s[i]$
9: **if** $M[\text{node}, \text{symbol}] = 0$ **then** ▷ Checks if `M[node,symbol]` equals 0.
10: $M[\text{node}, \text{symbol}] = \text{new\_node}$ ▷ A new node needs to be added at this position, as carried out in lines 10 and 11.
11: new_node = new_node + 1
12: node = new_node ▷ The variable `node` takes the value of the new node and is incremented in line 12.
13: **else**
14: node = $M[\text{node}, \text{symbol}]$ ▷ If `M[node,symbol]` is not 0, the node already exists and the variable `node` is simply updated with the value represented by $M[\text{node, symbol}]$.
15: **end if**
16: $M[\text{node}, 5] += 1$
17: $M[\text{node}, 6] += i$
18: **end for**
19: **end for**
20: **return** $M$

In line 3, the root (`root`) is set to the value 1 and the new node (`new_node`) as 2. In line 5, all the sequences $\{s_1, s_2, s_3, \ldots, s_n\}$ are processed starting from the root, each of which has a width of `k`. In line 9, the algorithm checks if `M[node, symbol]` is equal to 0. If this occurs, it means that there is no edge with the symbol defined by the variable `symbol` leaving from the variable `node` and going towards another node. In other words, it is necessary to add a new node at this position, which is done in lines 10 and 11. Finally, the variable `node` takes the value of the new node (`new_node`) and in line 12, the new node is incremented. In line 13, if `M[node, symbol]` is different from 0, then the node already exists, and in line 14, the variable `node` is simply updated with the value represented by $M[\text{node, symbol}]$.

The space complexity of SMT in the worst case is $O(\sigma v)$, where $\sigma$ is the number of symbols in the alphabet, in this case 4, and $v$ is the number of nodes in the tree. The value of $v$ directly depends on the size and variance of the fragments. Thus, the higher the variance, the greater the number of nodes. An inherent characteristic of SMT is that most elements in `M` remain empty. This behavior is recurrent and can be calculated. In particular, we can compute the expected occupancy level of `M`. To do so, consider that the maximum and minimum number of children a node can have is 4 and 0, respectively. The average would then be $\frac{4+0}{2} = 2$ children per node.

A tree with depth `k` and with each node having an average of 2 children, will have an average of $2^k$ leaves. The average number of nodes in the tree will be

$\nu = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1$. In this case, we will need a matrix `M` with $2^{k+1} - 1$ rows and 4 columns (two more columns for metadata). However, each line has 4 columns and on average only 2 of them will be occupied. With this, we can then calculate the occupancy level using $\omega = \frac{2\nu}{4\nu} = 0.5$. In practice, this number will be even smaller, as the average number of children decreases as the depth of the tree increases. This way, the average space complexity of `M` is $O(\omega\nu)$, as $\omega < \sigma$, then $O(\omega\nu) < O(\sigma\nu)$. Thus, we can expect that only half of the capacity of `M` will be utilized, which characterizes a significant waste of space. For this reason, the create-smt algorithm was implemented using a high-performance sparse matrix through the armadillo linear algebra library [16], employing the C++ programming language.

*Example* To add the k-mer `ACGT` to SMT, we start at the root of the matrix, which represents the starting point for all *kmers*. Each node in the matrix, or row, can be considered a decision point, which directs to the next node based on the next nucleotide. We insert the first nucleotide, `A`, checking the corresponding cell in the root row. If there is no previous entry (indicated by a zero), this means that `A` is a new path and a new node is created to represent it, with its index recorded in the cell `A` of the root.

Now, moving to the new node, we repeat the process for the second nucleotide, `C`. Likewise, the absence of a node for `C` indicates that we must add a new node, connecting it to the path starting with `A`. This addition process is sequential and continues for `G` and `T`, creating a unique path within SMT for the `ACGT` fragment. When we reach the end of *kmer*, we update the count and address at the end node to reflect the new addition. When two or more fragments are identical or share a common prefix, SMT uses this shared structure to save space and facilitate parsing. For example, if we already have *kmer* `ACGT` in SMT and we come across it again in the sequence, we simply increment the count at the final node, without the need to add new nodes to this fragment repeated.

In the case of *kmers* that share a prefix, such as `ACGA` and `ACGT`, SMT takes advantage of common nodes and diverges only in the last step, creating new paths for distinct nucleotides finals. This is particularly efficient because the paths that are shared do not need to be duplicated. Each *kmer* has its own count register and address, allowing SMT to accumulate information in a compact and structured way. Totalization nodes aggregate the counts of all k-mers that pass through that point, providing a cumulative count along different paths.

#### *KDIVE*

While KDIVE stands as the most important algorithm for analyzing conserved sequences in the context of SMT, it is not the only option available. Other algorithms that operate on the SMT include KSEARCH, IUPACSEARCH, and KHMAP. Each of these methods offers distinct advantages and functionalities, and they are discussed in detail in the *supplementary data*.

KDIVE was created with the objective of performing efficient text fragment searches in the smt, even if these present up to `d` mutations. In other words, kdive should return true for the search even if the query string contains up to `d` mismatches. It is important to highlight that the algorithm assumes as a premise that the probability of a mutation occurring is the same for any position in the sequence. For example, if

the probability of a match is equal to $\frac{1}{4}$, then that of a mismatch is $1 - \frac{1}{4} = \frac{3}{4}$. Therefore, the occurrence of a mutation is three times more likely than that of a base conservation. This characteristic is important because it alters the expected number of computations carried out and consequently modifies the algorithm's complexity. Algorithm 2 shows the basic functioning of kdive.

**Algorithm 2** KDIVE

**Require:** $M$, the SMT data structure
**Require:** $s$, the search sequence
**Require:** $d_{\max}$, the maximum number of allowed mutations
**Require:** $d$, the current mutation count (initially set to 0)
**Require:** $i$, the current position in the search sequence (initially set to 1)
**Require:** $k$, the kmer or search sequence size
**Require:** $next_{\text{node}}$, the index of the next node in the SMT (initially set to the root)
**Require:** $resp$, the response variable (initially set to 0)
1: **if** $k > d_{\max}$ **then**
2: **return** ▷ The search string exceeded $d_{\max}$ mutations and the tree will be pruned.
3: **end if**
4: **if** $i \geq k$ **then**
5: $resp = 1$ ▷ Found the search string with up to $d$ mutations.
6: **end if**
7: $symbol = s[i]$
8: **for** $j = 1$ to 4 **do**
9: $next_{\text{node}} = M[\text{node}, j]$
10: **if** $next_{\text{node}} \neq 0$ and $symbol = j$ **then**
11: kdive$(M, s, d_{\max}, d, i + 1, k, next_{\text{node}}, resp)$ ▷ Match!
12: **else**
13: kdive$(M, s, d_{\max}, d + 1, i + 1, k, next_{\text{node}}, resp)$ ▷ Mismatch!
14: **end if**
15: **if** $resp = 1$ **then**
16: **Break**
17: **end if**
18: **end for**

Algorithm 2 was implemented recursively and takes as its main parameters the smt matrix, the fragment `s`, and the total number of allowed mutations `d`. If $s \in M$ with at most `d` mutations, the algorithm returns the boolean value true, indicating the presence of the fragment with up to `d` mutations. The algorithm performs its task by traversing the SMT to identify matches between the search STRING and the stored *kmers*, taking into account the number of allowed mutations. It is worth noting that this algorithm can be easily adapted to return the complete set of mutated elements, instead of just a boolean value (TRUE or FALSE). This modification can be useful in certain applications, as it allows for more detailed information about the matches found. For example, it is possible to identify which *kmers* in the SMT correspond to the search fragment, considering the allowed mutations.

The use of a recursive algorithm in the implementation of KDIVE comes from several strategic and technical considerations. Tree data structures, such as SMT, benefit from the recursive approach because of how naturally recursion allows you to navigate nodes and

branches. Code simplicity is another significant advantage, where recursion offers a more readable and concise implementation compared to iterative methods that would require explicit stack or queue management. Furthermore, recursion aligns perfectly with the divide and conquer paradigm, dividing the problem into manageable sub-problems and handling them individually in an efficient manner.

This approach simplifies implementation and also improves code clarity and maintainability. The divide and conquer nature of recursion is particularly useful in scenarios with multiple choices or branches, common in structures like SMT where each node can lead to multiple paths. Efficiency in dealing with these multiple branches is one of the fundamental reasons for choosing a recursive method. Furthermore, recursion facilitates backtracking, allowing the algorithm to explore alternative paths in an orderly and efficient manner, essential in search and optimization processes such as the one described by KDIVE.

Although recursion carries the risk of stack overflow and overhead associated with function calls, these concerns are mitigated by the fact that the problem is well bounded and the depth of recursion is controlled by the size of the *kmer*, imposing a natural limit. Thus, the advantages of recursion, especially in terms of simplicity, readability and suitability for tree structures, justify its choice for KDIVE, providing a robust and effective method for analyzing k-mers in SMT.

The parameters $d_{\max}$, `d`, `i`, and `k` allow for the control of searching for exact or approximate matches. The recursion enables the algorithm to traverse the tree in depth and check all possible paths until it finds a match or reaches the maximum depth defined by the parameter $d_{\max}$. A possible call to Algorithm 2 could be `kdive(M, s,` $d_{\max} = 2$, `d = 0, i = 1, k = 10, node = root, resp =` FALSE`)`. In this case, the algorithm expects to find a match even if up to $d = 2$ mutations are detected in sequences of size $k = 10$. Algorithm 2 has two base cases, shown on lines 1 and 4. The first checks if the number of mutations has exceeded the limit $d_{\max}$, that is, if $d \geq d_{\max}$. If this condition is met, a pruning will occur in the search. The second base case checks if the algorithm has completely analyzed the query string, which will happen if $i \geq k$. If this condition is true, then the pattern has been found and the *resp* variable is updated to the value TRUE.

The time complexity of KDIVE can be measured through the Negative Binomial distribution. Consider a SMT with $\nu$ nodes, $d$ mutations, and fragments of width $k$. Also consider the random variable $X \sim NB(d, p)$, which counts the number of comparisons that the KDIVE algorithm performs. It's easy to verify that $p = \frac{3}{4}$, since if the probability of a MATCH is $\frac{1}{4}$, the probability of a MISMATCH will be $1 - \frac{1}{4} = \frac{3}{4}$, therefore $X \sim NB(d, \frac{3}{4})$. The PMF $P_x(d, p)$ is given by Eq. 1 and the expectation by Eq. 2.

$$P_x(d,p) = \binom{n-1}{d-1} p^d (1-p)^{n-d} \tag{1}$$

$$E(X) = \sum_{x \in X} x P_x(d,p) \tag{2}$$

The Negative Binomial distribution is a generalization of the geometric distribution, which is defined as $X \sim GEOM(p) = NB(1, p)$. Therefore, the geometric distribution is the Negative Binomial distribution with $d = 1$, making it possible to use it for

calculating the expected value. For example, consider $X \sim NB(1,p)$, $Y \sim NB(1,p)$, and $Z = X + Y$. If $X$ and $Y$ are independent, then $Z = X + Y \sim NB(2,p)$. Therefore, we can calculate $E(Z) = E(X) + E(Y) = \frac{1}{p} + \frac{1}{p} = \frac{2}{p}$. In this way, $X \sim NB(d,p)$ can be written as $Z = X_1 + X_2 + X_3 + \cdots + X_d$, where $X_i \sim NB(1,p)$. Calculating $E(Z) = E(X_1) + E(X_2) + E(X_3) + \cdots + E(X_d) = \frac{1}{p} + \cdots + \frac{1}{p} = \frac{d}{p}$.

The time complexity in the worst case will still be $O(k)$, but on average $\frac{d\times 4}{3}$ comparisons will be executed. With this result, we can write that the average time complexity of KDIVE is $O\left(\frac{d\times 4}{3}\right) = O(d)$. It is easy to verify that the larger the size of $d$, the more operations will be necessary. It is important to highlight that MOTIFS rarely exceed a width of 20 nucleotides, and the number of mutations is generally no more than 20% of their size. Considering this, we can expect to find an average of $d = 5$ mutations, which results in $\frac{5\times 4}{3} \approx 6.666$ comparisons per fragment.

We can calculate the complexity of the KDIVE algorithm in relation to the size of the input sequences. Consider a dataset with $n$ sequences of width $t$. We have a total of $m = t - k + 1$ fragments of size $k$ in each sequence of size $t$. Let $X$ be the number of comparisons made for each fragment. We know that $X$ follows a negative binomial distribution with parameters $p = \frac{3}{4}$ and $d$, and that $E(X) = \frac{d}{p} = \frac{4d}{3}$. Therefore, the number of comparisons per sequence will be $\frac{4dm}{3}$ resulting in an average asymptotic behavior $O(dm)$.

#### *Optimization*

The aim of this stage is to optimize the initial seeds obtained in the previous phase through the EM algorithm. The use of this algorithm allowed refining the parameters associated with the seeds, making them more accurate representations. This optimization process is important for the quality of the resulting models and for the success of subsequent analyses.

In this phase, pre-optimized seeds obtained from the SMT are used as a starting point for running the EM algorithm. The EM algorithm is responsible for refining these seeds, adjusting their parameters in order to maximize the likelihood of the given observations. As a result of this phase, each seed is transformed into a PWM model, which represents a more accurate and adjusted description of the motif in question. This PWM model can then be employed on new data to find patterns that have not yet been labeled.

To select the appropriate variant of the EM algorithm for *ChIP-seq* data, it's important to consider the protocol's nuances, which include the presence of noise and non-specific signals. While *ChIP-seq* aims to capture sequences with the motif of interest, not all peaks may contain it due to various factors. Therefore, the ZOOPS model is generally the most fitting for motif analysis in *ChIP-seq* data, allowing for the motif's occasional absence. Depending on the biological context, ANR or OOPS models may also be relevant.

#### *FAST-EM*

The FAST-EM represents a significant optimization over traditional OOPS and ZOOPS models. This algorithm was inspired by the implementation provided by [7, 8], whose modification not only speeds up the execution time but also enhances efficiency in

handling larger data sets. In other words, FAST-EM enhances the capabilities of the original models, making them more adaptable and robust when faced with large volumes of information.

The calculation of marginal probabilities represents one of the most computationally demanding phases in the EM algorithm. This stage requires computing the probability of a *kmer* being found at each of the $n \times (t - k + 1)$ valid positions, given the $n$ input sequences. This process can become a significant bottleneck, especially in *ChIP-seq* experiments where a large volume of sequences is often dealt with, in turn making the value of $n$ considerably high. Although it is possible to mitigate this computational challenge by using only a subset of the sequences as a sample, this approach compromises the predictive power of the model.

To understand its workings, we need to recap some concepts. The BAYES' rule for the EM algorithm states: $P(p_j|s_i) = \frac{P(s_i|p_j) \times P(p_j)}{P(s_i)}$, which can be interpreted as: "the probability of a MOTIF existing at position $p_j$ given we are observing sequence $i$, divided by the marginal probability of $s_i$." The marginal probability can be written as the weighted sum: $p(s_i) = P(s_i|p_1)P(p_1) + \cdots + P(s_i|p_m)P(p_m)$. If we consider that a MOTIF can be in any position with equal likelihood, then this sum simplifies to: $p(s_i) = P(s_i|p_1) + \cdots + P(s_i|p_m)$ .

To compute each term of the marginal probability, it is necessary to employ both the positive model ($\alpha$) and the negative model ($\beta$) through the following equation: $P(s_i|p_u) = \prod_{j=1}^{u} P(s_{ij}|\beta) \prod_{j=u+1}^{u+k} P(s_{ij}|\alpha) \prod_{j=u+k+1}^{m} P(s_{ij}|\beta)$, for $1 \leq u \leq m$. In other words, this equation must be run for all $m$ valid positions in each sequence. The FAST-EM algorithm does this by computing $P(s_i|\beta) = \prod_{j=1}^{m} P(s_{ij}|\beta)$ just once. Then, for each $1 \leq u \leq m$, this value is divided by $P(p_u|\beta) = \prod_{j=u}^{u+k} P(s_{ij}|\beta)$ and multiplied by $P(p_u|\alpha) = \prod_{j=u}^{u+k} P(s_{ij}|\alpha)$. To work on a logarithmic scale, simply replace multiplications with additions and divisions with subtractions. This simple optimization, when combined with MULTITHREADING techniques, makes the FAST-EM algorithm significantly faster and, therefore, better suited for handling large volumes of data.

To more intuitively understand the optimization provided by FAST-EM, we can consider the calculation of marginal probabilities as the most expensive optimization operation. In essence, the algorithm seeks to understand the probability of each *kmer* in all possible positions, a considerable computational challenge when dealing with a large number of sequences. The efficiency of FAST-EM lies in its ability to simplify this process without compromising the integrity of the results. Instead of repeatedly calculating complex probabilities for each position and *kmer*, FAST-EM makes use of a positive model and a negative model to pre-calculate and reuse results, significantly reducing the number of operations required.

By focusing on pre-calculating the negative model for the entire sequence and adjusting it as needed for each position, the algorithm avoids redundancy and speeds up the process. This optimization, although simple in concept, has a profound impact on the efficiency of the algorithm, allowing it to operate faster and handle significantly larger volumes of data. Furthermore, by operating on a logarithmic scale, replacing multiplications with additions and divisions with subtractions, FAST-EM further improves its performance.

## Results and discussion

In this section, we will show and analyze the results obtained by the BIOMAPP-CHIP framework in comparison with state-of-the-art algorithms. The BIOMAPP-CHIP was rigorously compared with state-of-the-art algorithms, including MEME [3], PROSAMPLER [13], and HOMER [11]. In the experiments conducted, two distinct types of data were employed to evaluate the effectiveness of our method. Synthetic data were used for load tests, allowing a comparative analysis of time consumption and RAM memory usage between the different approaches. On the other hand, real data were employed to assess accuracy, thus ensuring a more realistic evaluation of its applicability in practical scenarios.

In addition to comparisons with state-of-the-art methods, we also conducted a direct evaluation between the SMT and JELLYFISH [14], a widely-used tool for *kmer* counting. The aim was to understand the efficacy of the SMT algorithm in relation to established methods in the literature. Details on this comparison, including performance metrics, are provided in *supplementary data*. This additional analysis reinforces the robustness of our approach and offers further insights into the performance of the SMT algorithm.

It is important to highlight that all performance metrics presented in this study were rigorously evaluated. Load tests, which include measurements of execution time and memory consumption, have been standardized and quantified. The execution time of each algorithm was measured in seconds, while memory consumption was recorded in megabytes. In addition to performance metrics, accuracy was evaluated using several distance and correlation measures as a reference. These included EUCLIDEAN distance, MANHATTAN distance, HELLINGER distance, PEARSON correlation, BHATTACHARYYA coefficient, and SANDELIN-WASSERMAN similarity. These measurements were used to evaluate the models found in relation to the reference models.

### Synthetic data

The generation of synthetic data was carefully planned to allow a comprehensive comparison between BIOMAPP-CHIP framework in relation to other motif discovery algorithms. For the comparison, 50 datasets were generated, with sizes ranging from $1 \times 10^5$ to $2 \times 10^7$ bases. Each of these datasets was submitted to all algorithms with K-MER sizes ($k$) ranging from 5 to 30 bases. This experimental design allowed for a rigorous evaluation of performance and efficiency in K-MER counting under different load and complexity conditions.

### Real data

Real data were extracted from the JASPAR database version 2022 along with the genomic position files (.BED). These experiments form a fundamental step in validating our approach. By using real data, we can evaluate the performance of the algorithms in practical applications, increasing the reliability of the results and the robustness of our conclusions. Table 1 presents a detailed analysis of the data volume (measured by the number of peak regions) generated by different types of experiments available in the JASPAR 2022 database.

**Table 1** Number of peak regions extracted from various experiments available in the JASPAR 2022 database

| | TYPE | MIN | MAX | AVG |
|---|---|---|---|---|
| 1 | CHIP-SEQ | 22 | 322803 | 17538.78 |
| 2 | CAP-SELEX | 206 | 22850 | 9878.26 |
| 3 | HT-SELEX | 55 | 91919 | 9856.47 |
| 4 | NA | 84 | 105198 | 8801.62 |
| 5 | NCAP-SELEX | 2249 | 13498 | 6792.00 |
| 6 | PBM | 98 | 100001 | 4424.22 |
| 7 | COMPILED | 6 | 28379 | 3661.81 |
| 8 | DAP-SEQ | 22 | 19758 | 1222.74 |
| 9 | SMILE-SEQ | 998 | 1001 | 999.97 |
| 10 | CHIP-EXO | 48 | 1000 | 568.80 |
| 11 | SELEX | 9 | 1001 | 561.11 |
| 12 | CHIP-CHIP | 71 | 4737 | 475.80 |
| 13 | SELEX-SEQ | 384 | 384 | 384.00 |
| 14 | Universal protein binding microarray | 99 | 101 | 100.00 |
| 15 | PBM, CSA and/or DIP-CHIP | 8 | 102 | 99.12 |
| 16 | EMSA | 29 | 100 | 64.50 |
| 17 | Bacterial 1-hybrid | 13 | 60 | 23.13 |
| 18 | DNaseI footprinting | 10 | 41 | 16.33 |

It is clearly noted that the *ChIP-seq* protocol is responsible for generating the most significant volume of data

The *ChIP-seq* protocol stands out as the main source of data, generating the largest volume of information, with an average number of peak regions of 17,538.78. The data volume generated by this protocol varies considerably, ranging from a minimum of 22 to a maximum of 322,803 peak regions. Other significant protocols in terms of data volume include CAP-SELEX and HT-SELEX, which generated an average of 9,878.26 and 9,856.47 peak regions, respectively. Although these protocols generate considerable data volumes, they are still significantly below *ChIP-seq*.

In the experiments involving real data, datasets containing more than 10,000 sequences were selected. This criterion resulted in an initial set of 148 distinct datasets. However, 17 of these datasets were subsequently excluded from the analysis, as they were still in the process of validation. Therefore, the final set for evaluation consisted of 131 validated datasets. The choice of this selection criterion aims to ensure a sufficiently large and varied sample to rigorously and comprehensively evaluate the performance of the algorithms. This approach allows not only to test the efficacy of the algorithms on real data, but also provides an in-depth analysis of their applicability in scenarios closer to the experimental conditions frequently encountered in research in the area of biological MOTIF discovery.

### Parameters

In order to ensure the reproducibility of experiments, we established the parameters for each algorithm according to specific guidelines. The standardization of these settings is important to guarantee the integrity and comparability of the results obtained. Furthermore, all experiments were run on Intel Xeon CPU E5-2673 v4 2.30GHz servers with

**Table 2** Parameters configured for the algorithms used in the research

| ALGORITHMS | PARAMETERS |
|---|---|
| BIOMAPP::CHIP | `-k <size of kmer> -n 1 -d 2 -c zoops -r 1000 -f 0.001` |
| MEME | `-w <size of kmer> -dna -mod zoops -nmotifs 1` |
| PROSAMPLER | `-k <size of kmer> -m 1 -l 0` |
| HOMER | `-len <size of kmer> -S 1` |

Each algorithm is listed alongside its respective parameters to ensure the reproducibility of the experiments

8Gb of Ram memory with Linux/Ubuntu 22.04 operating system. The parameters used in the experiments followed those displayed in Table 2.

It is essential to clarify some aspects regarding the choice of parameters. All methods were adjusted to identify only the most effective model, as evidenced by the options `-n 1`, `-nmotifs 1`, `-m 1`, and `-S 1` for the SMT, MEME, PROSAMPLER, and HOMER algorithms, respectively. The ZOOPS model was employed in the BIOMAPP::CHIP and MEME algorithms, while HOMER uses the ZOOPS model intrinsically. The PROSAMPLER algorithm, on the other hand, does not offer the option of selecting this model. Additionally, the parameters `-r 1000` and `-f 0.001` are used to govern the convergence of BIOMAPP::CHIP. The iterative process will be halted if the increment in the score difference between two successive iterations is below `-f 0.001`. Otherwise, the algorithm will continue until it reaches a limit of `-r 1000` iterations.

#### *Biomapp-chip on synthetic data*

The aim of these experiments was to measure the performance of the BIOMAPP::CHIP framework on a diverse set of synthetic datasets. These data were generated with variable parameters to simulate different scenarios and complexities associated with the real world. The use of synthetic data offers a controlled platform that allows isolating and evaluating the performance and efficacy of algorithms under well-defined conditions. This analysis procedure is essential for the initial validation of the adopted methodological approach, serving as a preliminary step before handling real data.

These trials were designed to collect basic data on the time consumption and RAM memory usage by the BIOMAPP::CHIP algorithm. This information was compared with time and space metrics obtained from other established algorithms in the field, such as MEME, PROSAMPLER, and HOMER. The purpose of this comparison was to understand how BIOMAPP::CHIP stands in relation to other solutions and identify the strengths and weaknesses of each approach. To carry out the experiments, the *k* parameter, representing the size of the *kmers*, was systematically varied within a range that covers values from 5 to 30, thus keeping in line with the criteria established in the SMT load tests. In this way, 5200 experiments were performed, 1300 for each algorithm. The capture of relevant metrics, which include both time consumption and RAM memory allocation, was conducted using the UNIX/LINUX **/usr/bin/time -v** command, ensuring consistent and comparable data collection across all algorithms under study.

Figure 3 illustrates the global averages of the tests for each algorithm, considering both time and space consumption. BIOMAPP::CHIP proved superior in terms of time efficiency, closely followed by MEME. On the other hand, PROSAMPLER and HOMER showed

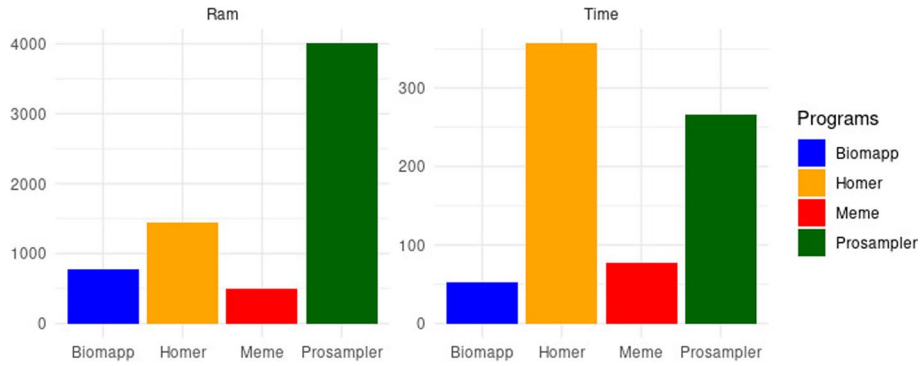


**Fig. 3** Performance comparison between BIOMAPP-CHIP, MEME, HOMER and PROSAMPLER, with emphasis on time (measured in seconds) and memory consumption (measured in MBYTES). The values presented are calculated as the average performance across all analyzed datasets and for all sizes of *k*. While MEME stood out for its efficient use of memory, BIOMAPP-CHIP proved to be the fastest in terms of execution time

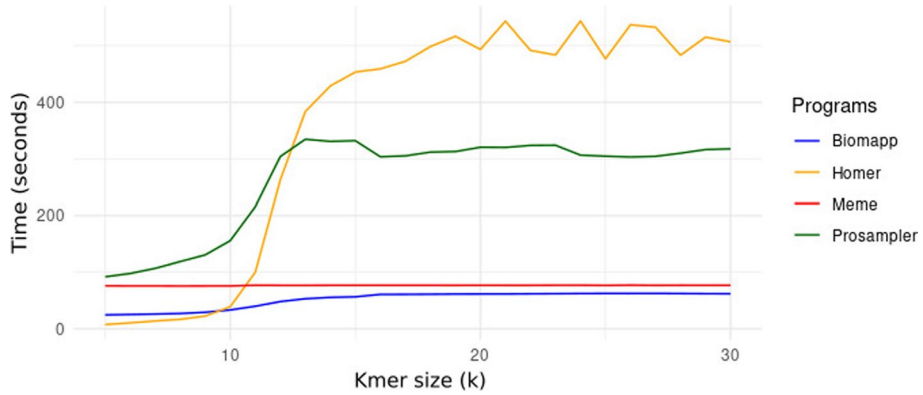


**Fig. 4** Runtime variation in relation to k for the algorithms BIOMAPP::CHIP, MEME, HOMER, and PROSAMPLER. It is observed that the BIOMAPP::CHIP algorithm exhibits an advantage in terms of time efficiency, followed by the MEME algorithm

less efficacy in this aspect, requiring longer periods for execution. Regarding memory usage, MEME led in performance but was closely followed by BIOMAPP::CHIP. The latter employs the SMT data structure to initialize its models, which demonstrates the efficiency of this structure in terms of memory usage. As observed in the time dimension, HOMER and PROSAMPLER occupied lower positions in memory consumption.

The Fig. 4 analyzes the execution time of the four algorithms – BIOMAPP::CHIP, MEME, HOMER, and PROSAMPLER – as a function of the k size. It is noticeable that BIOMAPP::CHIP is the most time-efficient algorithm, closely followed by MEME, both maintaining an average execution time below 100 s for all evaluated cases. In contrast, the average execution times for PROSAMPLER and HOMER were substantially higher, with PROSAMPLER exceeding 200 s and HOMER surpassing 400 s. Additionally, the analysis also reveals that the execution time for all evaluated algorithms increases as the size of k grows.

The Fig. 5 presents a line graph illustrating the RAM memory consumption for each algorithm. It is a unified graph, allowing for direct comparison between all of them. Similar to what was observed in execution time, BIOMAPP::CHIP and MEME were close in

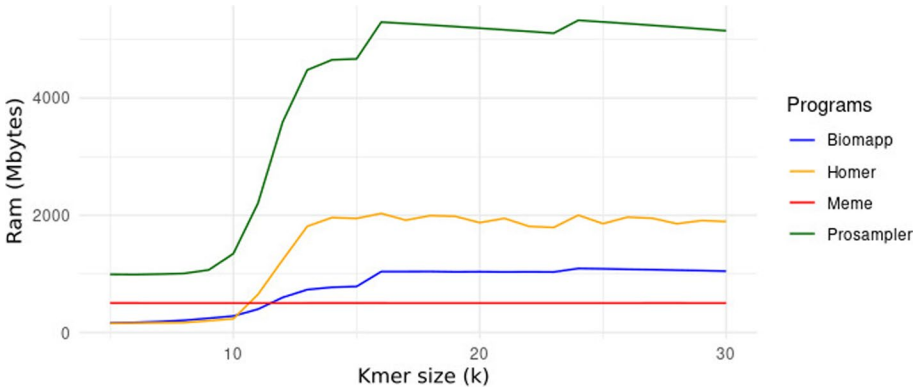


**Fig. 5** Variation of RAM memory consumption in relation to k for the algorithms BIOMAPP::CHIP, MEME, HOMER, and PROSAMPLER. It is observed that the MEME algorithm has an advantage in terms of spatial efficiency, followed by the BIOMAPP::CHIP algorithm

memory efficiency, although MEME had a slight advantage, consuming just under 500 Mbytes in the worst case. BIOMAPP::CHIP showed an average memory consumption slightly below 1000 Mbytes. In contrast, the HOMER and PROSAMPLER algorithms displayed much higher memory usage, reaching just over 2000 Mbytes and exceeding 4000 Mbytes in the worst scenario, respectively.

#### *Biomapp::chip on real data*

Although experiments in simulated environments are useful for initial understanding and fine-tuning of algorithms, it is the trials with real data that provide the true validity test for any computational method in bioinformatics. They offer a much more complex and variable scenario, which more accurately plays out the conditions that algorithms will face in real-world applications. Therefore, this section is the most critical and provides more precise information about the feasibility and robustness of the evaluated algorithms.

In this test set, all algorithms were executed on a sample of 131 datasets, obtained from the JASPAR repository. These data were carefully selected based on two specific criteria: (i) all are from *ChIP-seq* experiments and (ii) all have more than 10 thousands sequences. The main focus of this test was the comparison of accuracy between the approaches. For this reason, unlike the tests performed on synthetic data, where the value of k was varied systematically, in this scenario the value of k adopted was that suggested by scientific literature. Thus, each algorithm was tested with a unique and specific value of k, according to the most recent academic guidelines.

*Performance in relation to time and space consumption* Figure 6 reveals important nuances in the performance of the four examined algorithms in terms of execution time and RAM memory consumption. BIOMAPP:CHIP stands out for displaying the shortest average execution time, only 15.9 s, making it the most agile option. On the other hand, HOMER proved to be the slowest, with an average time of 624 s and an average RAM consumption of 1436.16 MB, which could be a limitation in scenarios that demand quick responses.

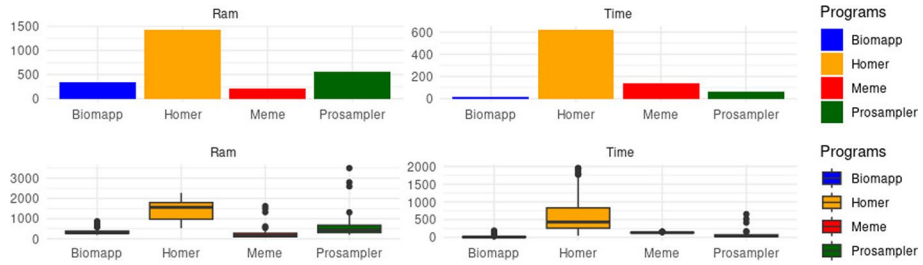


**Fig. 6** Comparison of the overall average RAM memory (measured in Mbytes) and execution time consumption (measured in seconds) among different algorithms applied to real datasets. It is observed that BIOMAPP::CHIP leads in time consumption and occupies the second position in RAM consumption. MEME shows the lowest RAM consumption, while PROSAMPLER, despite its inferior performance in synthetic tests, shows a significant improvement, coming in second place in memory consumption. The HOMER algorithm does not stand out in any of the criteria

**Table 3** Comparison of memory consumption (measured in Mbytes) and time (measured in seconds) between the algorithms BIOMAPP::CHIP, HOMER, MEME and PROSAMPLER

| ALGORITHM | AVG TIME | AVG RAM | MAX TIME | MAX RAM | MIN TIME | MIN RAM |
|---|---|---|---|---|---|---|
| Biomapp | 15.86 | 335.57 | 192.79 | 868.92 | 1.41 | 190.08 |
| Homer | 624.45 | 1436.16 | 1952.69 | 2270.09 | 52.47 | 524.70 |
| Meme | 138.36 | 216.94 | 159.70 | 1618.04 | 118.93 | 69.32 |
| Prosampler | 61.83 | 548.17 | 655.92 | 3500.79 | 14.26 | 199.07 |

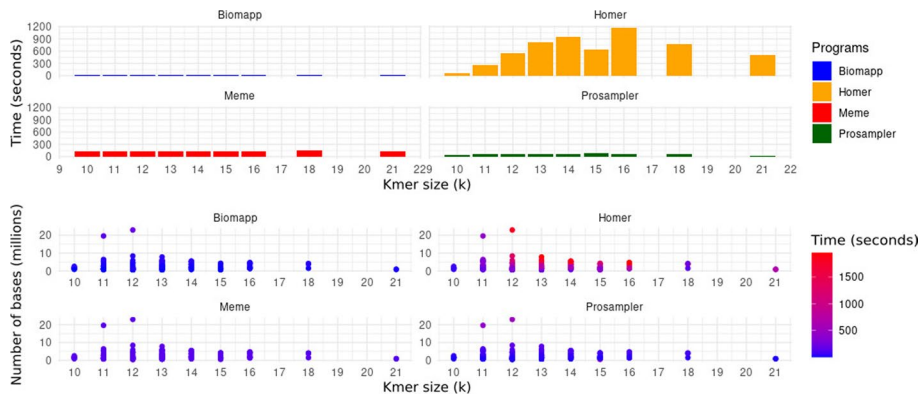


**Fig. 7** Comparison between the average performance of the algorithms in relation to time consumption grouped by value of k. It is possible to verify that BIOMAPP::CHIP, PROSAMPLER and MEME exhibit the shortest times while HOMER presented significantly higher execution times

In terms of RAM consumption, MEME was the most efficient with an average of 217 MBYTES, while HOMER exhibited the highest consumption, with 1436 MBYTES. PROSAMPLER, which had moderate performance in time, registered the highest peak in RAM usage, reaching 3501 MBYTES. In addition, PROSAMPLER displayed the greatest variation in both time and RAM consumption, suggesting that its performance may be highly sensitive to the characteristics of the analyzed dataset. Overall, the tests indicated that BIOMAPP::CHIP and MEME proved to be more robust, making them more predictable in their performance, as shown in Table 3.

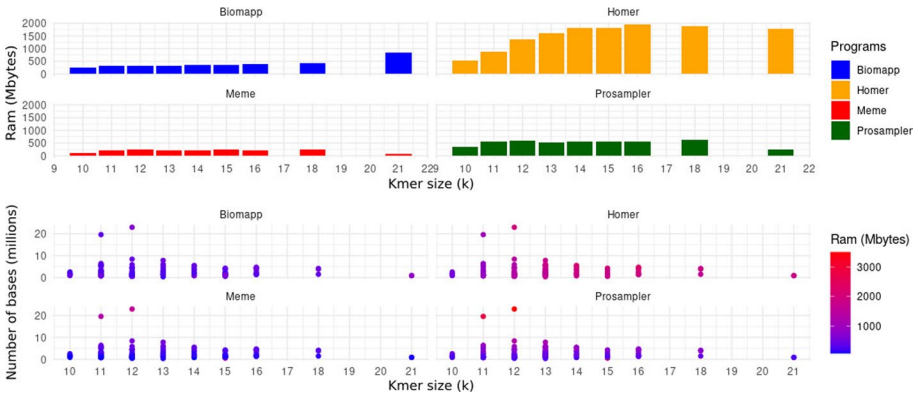


**Fig. 8** Comparison between the average performance of the algorithms in relation to memory consumption RAM grouped by k value. It is possible to verify that BIOMAPP::CHIP, PROSAMPLER and MEME exhibit the shortest times while HOMER presented significantly higher execution times

The analysis of the data presented in Figs. 7 and 8 provides important insights into the performance and efficiency of the four algorithms evaluated in various scenarios, parameterized by the value of k. First, it is possible to observe that the BIOMAPP::CHIP algorithm displays, on average, the lowest consumption of time and RAM memory in almost all scenarios. Its average execution time ranges from 9.79 to 23.00 s, while the average RAM usage lies between 251.50 and 838.70 MBYTES. The algorithm is also consistent in maintaining a relatively low maximum and minimum execution time, as shown by the data in Table 4.

On the other hand, the HOMER algorithm presents the highest consumption of both time and RAM. It is interesting to note the increase in average execution time and average RAM usage as k increases, reaching a peak of 1174.47 s and 1950.41 MBYTES for $k = 16$.

The MEME algorithm shows relative stability in average execution time, ranging from 134.64 to 143.33 s, but with peaks of RAM usage reaching 1618.04 MBYTES for $k = 12$. The PROSAMPLER displays intermediate performance in terms of time efficiency and RAM usage, with a large variation in maximum and minimum metrics, particularly for RAM usage, which reaches up to 3500.79 MBYTES for $k = 12$.

*Comparison with reference models* In this subsection, we will address the evaluation of the models generated by each algorithm, contrasting them with reference models extracted from the JASPAR database version 2022. The goal of this comparison is to quantify how well the algorithms were able to approximate the PWM matrices found by each approach to their respective reference matrices. For this, specific metrics were employed to measure the distance between the matrices, thus providing a comprehensive analysis of the accuracy of the methods in finding reliable and precise representations of the investigated biological MOTIFS.

According to Fig. 9 and Table 5, the BIOMAPP algorithm showed superiority in almost all metrics analyzed. In the BHA metric, the BIOMAPP algorithm recorded an average of 0.993 and a very low standard deviation of 0.0308, signaling consistency in the results.

**Table 4** Comparison of memory consumption (measured in Mbytes) and time (measured in seconds) between the algorithms BIOMAPP::CHIP, HOMER, MEME and PROSAMPLER grouped by k

| ALGORITHM | K | AVG TIME | AVG RAM | MAX TIME | MAX RAM | MIN TIME | MIN RAM |
|---|---|---|---|---|---|---|---|
| Biomapp | 10 | 19.34 | 251.50 | 95.17 | 422.70 | 4.69 | 208.02 |
| | 11 | 15.83 | 307.61 | 150.88 | 489.64 | 2.62 | 221.99 |
| | 12 | 21.53 | 326.33 | 192.79 | 664.13 | 2.36 | 235.98 |
| | 13 | 13.25 | 310.64 | 40.45 | 514.13 | 1.41 | 190.08 |
| | 14 | 15.99 | 370.37 | 34.08 | 576.42 | 3.68 | 262.31 |
| | 15 | 10.43 | 343.74 | 28.53 | 683.56 | 4.05 | 271.53 |
| | 16 | 15.84 | 381.98 | 24.34 | 527.36 | 7.01 | 281.87 |
| | 18 | 23.00 | 420.45 | 34.17 | 478.62 | 11.25 | 374.58 |
| | 21 | 9.79 | 838.70 | 15.10 | 868.92 | 4.47 | 808.49 |
| Homer | 10 | 65.81 | 542.56 | 85.76 | 575.83 | 52.47 | 524.70 |
| | 11 | 258.96 | 867.76 | 421.93 | 1054.18 | 133.48 | 684.94 |
| | 12 | 549.22 | 1368.86 | 1802.54 | 1944.38 | 152.97 | 1156.77 |
| | 13 | 806.39 | 1613.70 | 1951.66 | 1796.67 | 171.83 | 1467.87 |
| | 14 | 952.28 | 1825.39 | 1875.14 | 2103.16 | 213.74 | 1565.54 |
| | 15 | 645.16 | 1826.03 | 1584.17 | 2062.00 | 326.13 | 1717.78 |
| | 16 | 1174.47 | 1950.41 | 1952.69 | 2195.72 | 184.05 | 1592.81 |
| | 18 | 763.16 | 1887.06 | 1931.68 | 2270.09 | 142.80 | 1595.04 |
| | 21 | 497.58 | 1765.16 | 709.77 | 1824.43 | 285.39 | 1705.90 |
| Meme | 10 | 139.84 | 117.86 | 152.63 | 166.72 | 133.59 | 73.30 |
| | 11 | 138.23 | 225.86 | 159.70 | 1319.03 | 122.78 | 72.28 |
| | 12 | 137.20 | 242.56 | 155.45 | 1618.04 | 126.23 | 69.32 |
| | 13 | 136.74 | 202.81 | 153.11 | 538.06 | 118.93 | 72.25 |
| | 14 | 140.46 | 223.61 | 156.88 | 409.22 | 129.15 | 73.24 |
| | 15 | 139.28 | 240.67 | 152.55 | 1491.57 | 130.57 | 75.79 |
| | 16 | 139.92 | 218.37 | 152.61 | 338.57 | 129.52 | 102.36 |
| | 18 | 143.33 | 248.87 | 152.39 | 328.58 | 136.32 | 135.78 |
| | 21 | 134.64 | 86.45 | 143.97 | 93.61 | 125.32 | 79.28 |
| Prosampler | 10 | 31.15 | 350.59 | 47.54 | 482.50 | 16.71 | 232.54 |
| | 11 | 62.76 | 563.89 | 418.14 | 2796.34 | 15.60 | 199.07 |
| | 12 | 69.60 | 603.25 | 518.37 | 3500.79 | 14.26 | 213.74 |
| | 13 | 55.79 | 526.55 | 155.40 | 1166.94 | 16.35 | 235.40 |
| | 14 | 61.46 | 571.55 | 112.23 | 909.96 | 17.21 | 226.32 |
| | 15 | 81.32 | 566.79 | 655.92 | 2596.81 | 16.70 | 236.42 |
| | 16 | 58.93 | 558.32 | 97.11 | 817.34 | 25.89 | 335.26 |
| | 18 | 69.08 | 626.60 | 91.61 | 794.47 | 34.94 | 400.59 |
| | 21 | 20.13 | 259.94 | 21.58 | 265.49 | 18.68 | 254.39 |

However, it is interesting to note that PROSAMPLER had the lowest minimum in this metric (0.685), which could be a point of concern in terms of consistency.

When we observe distance metrics such as EUC, HELL, and MAN, BIOMAPP once again stands out for having the lowest averages. Here, HOMER's performance is notably inferior; for example, in the MAN metric, the average is 0.264 with a standard deviation of 0.194, both values being the highest among the algorithms analyzed. This could point to a generally inferior performance of HOMER in multidimensional comparisons. Regarding the PCC metric, all algorithms showed high average values, indicating a good degree

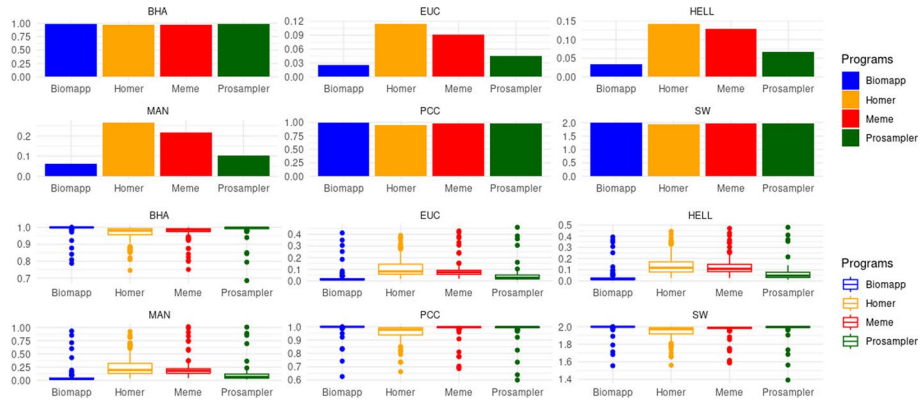


**Fig. 9** General comparative analysis between BIOMAPP::CHIP, MEME, PROSAMPLER and HOMER algorithms, employing a variety of distance metrics and correlation coefficients. Metrics used for comparison include EUCLIDEAN distance, MANHATTAN distance, HELLINGER distance, PEARSON correlation coefficient, SANDELIN-WASSERMAN coefficient, and BHATTACHARYYA COEFFICIENT . The values presented are the global average of all metrics, calculated over all datasets and for all sizes of *k*

**Table 5** Comparative analysis between algorithms in various distance and correlation metrics

| METRICS | PROGRAM | MEAN | SD | MIN | MAX |
|---|---|---|---|---|---|
| | **Biomapp** | **0.993** | **0.031** | **0.787** | **1.000** |
| BHA | Homer | 0.963 | 0.045 | 0.746 | 0.999 |
| | Meme | 0.972 | 0.040 | 0.751 | 0.999 |
| | Prosampler | 0.988 | 0.037 | 0.685 | 1.000 |
| | **Biomapp** | **0.026** | **0.058** | **0.002** | **0.412** |
| EUC | Homer | 0.115 | 0.085 | 0.016 | 0.391 |
| | Meme | 0.091 | 0.071 | 0.019 | 0.429 |
| | Prosampler | 0.045 | 0.065 | 0.008 | 0.460 |
| | **Biomapp** | **0.033** | **0.063** | **0.003** | **0.393** |
| HELL | Homer | 0.143 | 0.091 | 0.026 | 0.443 |
| | Meme | 0.129 | 0.077 | 0.027 | 0.470 |
| | Prosampler | 0.067 | 0.069 | 0.011 | 0.479 |
| | **Biomapp** | **0.061** | **0.137** | **0.005** | **0.935** |
| MAN | Homer | 0.264 | 0.194 | 0.039 | 0.925 |
| | Meme | 0.218 | 0.167 | 0.044 | 1.016 |
| | Prosampler | 0.104 | 0.147 | 0.019 | 1.008 |
| | **Biomapp** | **0.991** | **0.045** | **0.626** | **1.000** |
| PCC | Homer | 0.955 | 0.059 | 0.663 | 0.999 |
| | Meme | 0.983 | 0.055 | 0.686 | 1.000 |
| | Prosampler | 0.985 | 0.054 | 0.600 | 1.000 |
| | **Biomapp** | **1.988** | **0.057** | **1.554** | **2.000** |
| SW | Homer | 1.940 | 0.079 | 1.561 | 1.999 |
| | Meme | 1.969 | 0.070 | 1.586 | 1.999 |
| | Prosampler | 1.982 | 0.074 | 1.391 | 2.000 |

The bold highlights the winner of each metric in terms of average

of correlation. However, it is interesting to note that PROSAMPLER had the lowest minimum value (0.600) in this metric. Such a minimum value is significantly lower than the others, which may indicate instability in specific cases. Figure 10 displays the evolution of each algorithm as a function of the variable k.

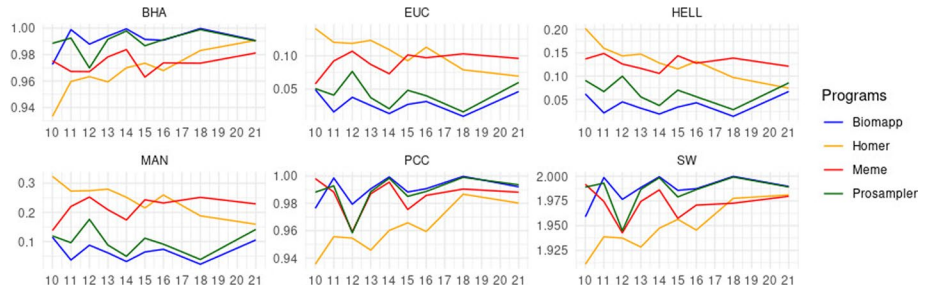


**Fig. 10** Performance comparison of the BIOMAPP, HOMER, MEME and PROSAMPLER in different motif evaluation metrics. Metrics include Bhattacharyya coefficient (BHA), Euclidean distance (EUC), Hellinger distance (HELL), Manhattan distance (MAN), Pearson correlation coefficient (PCC), and Sanderlin-Wasserman coefficient (SW), as a function of the size of the *kmer* (represented on the x-axis)

It is possible to observe in this figure that BIOMAPP in the EUCLIDEAN, HELLINGER, and MANHATTAN distances always stayed below the other algorithms, for all values of k. In the BHATTACHARYYA coefficient, PROSAMPLER obtained the best result (0.988) for $k = 10$, followed by MEME (0.975) and BIOMAPP::CHIP (0.972). For other values of k, BIOMAPP::CHIP achieved better values in this metric. Something similar occurred with the PEARSON correlation and SANDELIN-WASSERMAN similarity, in which BIOMAPP::CHIP had the lowest score for $k = 10$. The SW metric showed a closer competition among the algorithms, with all averages approaching 2 and relatively low standard deviations. However, PROSAMPLER here showed the lowest minimum value of 1.39, which once again raises questions about its consistency compared to the other algorithms. While BIOMAPP exhibited a generally superior performance in all metrics evaluated, the other algorithms have weak points that deserve attention. The inferior performance of HOMER in distance metrics and the variability of PROSAMPLER in metrics like BHA and PCC are points that require additional investigations.

Analyzing Fig. 10 focusing on BIOMAPP and considering the points where it is surpassed by other algorithms, an interesting pattern can be seen in the behavior in relation to the different metrics and *k* values. In the Euclidean (EUC), Hellinger (HELL) and Manhattan (MAN) distance metrics, BIOMAPP demonstrate to be superior in all *k* values, which implies significant consistency and efficiency in capturing differences between motifs, regardless of the k-mer size. However, when evaluating the BHA, PCC and SW metrics, we observed that BIOMAPP is not the leader for $k = 10$. This could be a reflection of intrinsic characteristics of the data or the nature of the shorter motifs, which may not be as well captured by the algorithm as those of larger size. In small *kmers*, measurement precision may be more susceptible to variations due to the limited amount of information available for analysis. When considering *k* values greater than 10, BIOMAPP outperforms other algorithms in most cases, indicating optimization for intermediate to large *kmers*. From $k = 15$ onwards, BIOMAPP's performance is quite similar to that of PROSAMPLER, suggesting that both algorithms are efficient in capturing the relevant features of the motifs for these sizes of *kmers*, but BIOMAPP has a slight advantage. The reason BIOMAPP exhibits an improvement may be related to its ability to integrate and interpret more complex information that is more evident in larger sequences. As *kmer* grows, there is more context for analysis, allowing BIOMAPP's strengths in terms of algorithms and statistical modeling to come to the

**Table 6** Results of FRIEDMAN and NEMENYI tests for the evaluation of MOTIFS discovery algorithms

| | $\chi^2$ | P-VALUE | HOMER | MEME | PROSAMPLER | RESULT |
|---|---|---|---|---|---|---|
| EUC | 280.46 | 1.69e−60 | 0.00e+00 | 0.00e+00 | 4.01e−09 | + |
| MAN | 281.35 | 1.08e−60 | 0.00e+00 | 0.00e+00 | 8.58e−10 | + |
| PCC | 243.61 | 1.57e−52 | 0.00e+00 | 2.94e−14 | 7.42e−13 | + |
| HELL | 275.29 | 2.21e−59 | 0.00e+00 | 0.00e+00 | 1.50e−12 | + |
| SW | 261.30 | 2.35e−56 | 0.00e+00 | 0.00e+00 | 9.85e−09 | + |
| BHA | 271.19 | 1.71e−58 | 0.00e+00 | 0.00e+00 | 3.32e−11 | + |

Each line represents an evaluated metric. The columns $\chi^2$ and $p-$ value correspond to the results of FRIEDMAN's test. The columns relating to HOMER, MEME and PROSAMPLER display the P-VALUES obtained by the NEMENYI test, all compared to the BIOMAPP::chip program. The last column, called RESULT, indicates whether BIOMAPP performed better (+), lower (-) compared to the other approaches or (=) to not statistically significant

fore. This may be a direct consequence of the method used to model motifs or the way BIOMAPP weights different aspects of sequences when calculating similarity and correlation measures.

*Statistical analysis* In the performance analysis of algorithms, it is imperative to employ rigorous statistical techniques to determine whether the differences observed in various metrics are truly significant. The application of statistical tests provides a means to validate comparisons between different approaches or settings, ensuring that the conclusions drawn are robust and reliable.

In this subsection, use is made of the FRIEDMAN test, a non-parametric method employed to identify significant variations in medians among multiple paired groups. This statistical test is especially relevant when the conditions of normality and homoscedasticity, which are prerequisites for the application of ANOVA, are not verified, as observed in the present study. After the main tests, POST HOC tests will be performed for multiple comparison analyses. The aim is to identify which groups are statistically different from each other. The results of the POST HOC tests are essential for establishing concrete conclusions about the evaluated metrics.

The data displayed in Table 6 show the results of the FRIEDMAN and NEMENYI statistical tests applied to various metrics to evaluate the performance of the motif discovery algorithms. The metrics evaluated include EUC (EUCLIDEAN distance), MAN (MANHATTAN distance), PCC (PEARSON correlation), HELL (HELLINGER distance), SW (SANDELIN-WASSERMAN coefficient), and BHA (BHATTACHARYYA distance).

For each metric, the FRIEDMAN test produced highly significant P-VALUES, pointing to a considerable difference between the approaches. All the P-VALUES are very close to zero, well below the adopted significance level of $\alpha = 0.05$, indicating that the differences are statistically significant. The WINNER column indicates that the BIOMAPP program outperformed all other approaches for all the tested metrics, as denoted by the symbol (+). The results of the NEMENYI test reinforce this conclusion, with P-VALUES very close to zero. These data can be graphically visualized through Fig. 11.

Analyzing this figure, we can verify that the results obtained from the NEMENYI test point to statistically significant differences between the BIOMAPP::CHIP program and the other evaluated methodologies, covering all the metrics in question. It is relevant to highlight that MEME was statistically similar to HOMER in all distance metrics. However,

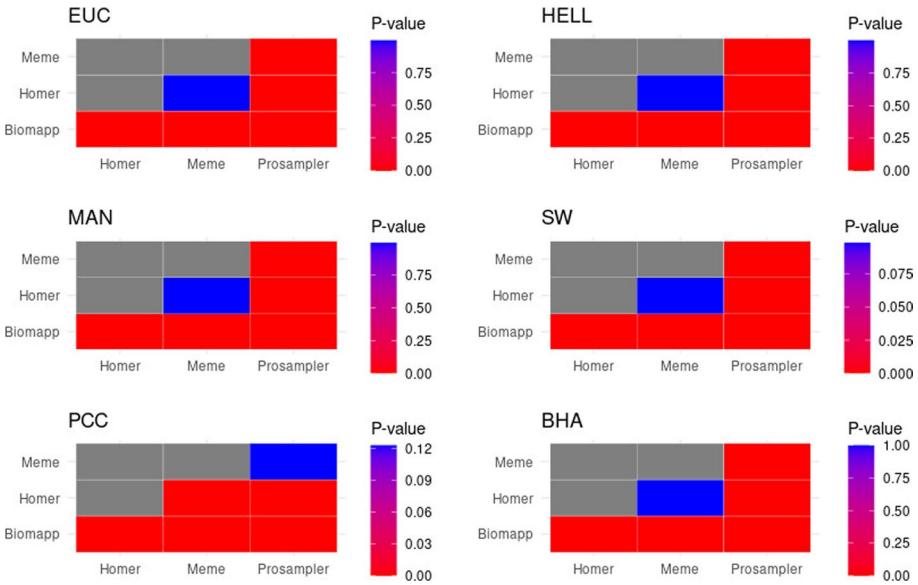


**Fig. 11** Results of P-VALUES derived from the NEMENYI test indicate the presence of significant statistical differences between the BIOMAPP::CHIP program and the other approaches tested in all metrics evaluated. Additionally, it is noted that MEME does not present significant statistical differences in relation to HOMER except in the PCC metric. In the latter, both MEME and PROSAMPLER were statistically similar

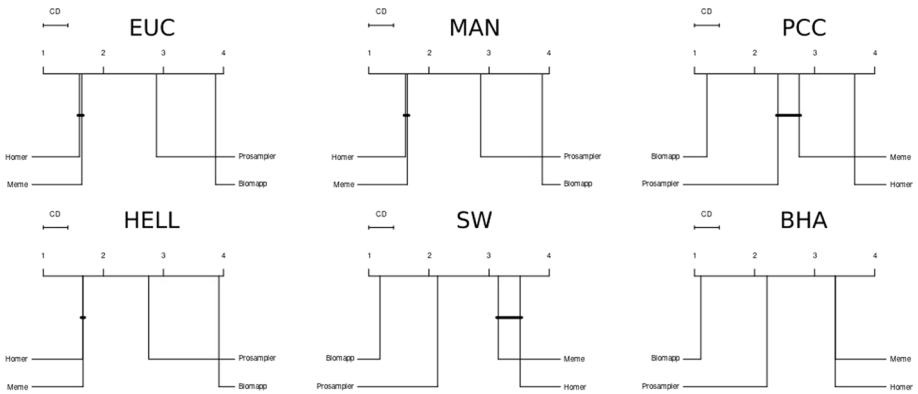


**Fig. 12** Comparison of the critical distance between different approaches. The analysis reveals that the BIOMAPP::CHIP algorithm stands out, presenting superior performance in relation to the other methodologies evaluated

in the PCC metric, the MEME program was more similar to PROSAMPLER, showing no statistically significant differences between them in this regard. The similarity between MEME and HOMER in distance metrics and the closeness between MEME and PROSAMPLER in the PCC metric can be attributed to various factors. These may include the nature of the algorithms, sensitivity to outliers, data peculiarities, or even optimization for different loss functions. Each metric may be highlighting different aspects of the relationships between the approaches, which may contribute to this behavior.

Lastly, Fig. 12 presents the critical distance graphs between the different algorithms, providing an intuitive visualization of the relative positions of each method in terms of performance. The critical distance is a key value obtained from statistical tests such

as the NEMENYI post-hoc test, and serves as a threshold for determining whether performance differences between multiple algorithms are statistically significant. This measure is important because it provides an objective criterion for evaluating and comparing the effectiveness of different methods. If the difference in performance between two algorithms exceeds the critical distance, we can conclude that one algorithm is significantly better than the other with a certain level of confidence. It is clearly observed that the BIOMAPP::CHIP algorithm achieved the best performance, demonstrating statistical superiority compared to all other approaches. A close relationship is also noted between the MEME and HOMER algorithms in various metrics, especially those related to distance, corroborating the analysis performed on Fig. 12. Similarly, it is also possible to see in this figure that, in the PCC metric, the MEME algorithm displayed a high proximity to the PROSAMPLER algorithm, indicating that there are no statistically significant differences between them in this specific metric.

## Conclusions

This study addressed the problem of biological motif discovery, a field of extreme relevance for understanding regulatory mechanisms in genomes and with implications in various areas of biology. Aiming to overcome the current limitations of available algorithms, we introduced BIOMAPP::CHIP, a tool designed to optimize both accuracy and efficiency in the analysis of large volumes of data, such as those generated by *ChIP-seq* protocols. BIOMAPP::CHIP sets itself apart with its two-step approach: the first is dedicated to efficient k-mer counting through the SMT algorithm, and the second focuses on optimization via an enhanced version of the EM algorithm. Our comparative analyses with state-of-the-art methods, such as MEME, PROSAMPLER, and HOMER, demonstrated that BIOMAPP::CHIP offers a strong balance between accuracy and efficiency. The success of BIOMAPP::CHIP in the experiments suggests that the tool can be an important resource for researchers looking for reliable and effective approaches to motif discovery in large data sets. Future research may explore additional optimizations and the application of the tool in different biological scenarios.

## Availability and requirements

Project name: BIOMAPP::CHIP; Project home page: https://github.com/jadermcg/biomapp-chip; Operating system(s): Linux; Programming language: C++ and R; Other requirements: Threading Building Blocks https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html; License: Apache License 2.0.

**Acknowledgements**
The authors would like to thank Coordenação de Aperfeiçoamento de Pessoal de NÃvel Superior—Brasil (CAPES)—Finance Code 001—for the financial support given to this research.

**Author contributions**
JMCG conceived and designed the approach. ATPR and DSS oversaw and coordinated the project. JMCG developed, implemented, realized the experiments and analyzed the results. JMCG, ATPR and DSS tested the algorithm and wrote this paper. All authors approved the final version of this manuscript.

**Funding**
This work has been supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES) under Finance Code 001.

**Availability of data and materials**
All datasets used in the paper can be downloaded in: https://github.com/jadermcg/biomapp-chip/tree/main/datasets.

## Declarations

**Ethics approval and consent to participate**
Not applicable.

**Consent for publication**
Not applicable.

**Competing interests**
The authors declare that they have no competing interests.

Received: 29 September 2023 Accepted: 18 March 2024
Published online: 26 March 2024

**References**

1. Altschul SF, Erickson BW. Significance of nucleotide sequence alignments: a method for random sequence permutation that preserves dinucleotide and codon usage. Mol Biol Evol. 1985;2(6):526–38.
2. Archbold J, Johnson N. A construction for room's squares and an application in experimental design. Ann Math Stat. 1958;29(1):219–25.
3. Bailey TL, Elkan C et al. Fitting a mixture model by expectation maximization to discover motifs in bipolymers. UCSD Technical Report CS94-351; 1994
4. D'haeseleer P. How does DNA sequence motif discovery work? Nat Biotechnol. 2006;24(8):959–61.
5. D'haeseleer P. What are DNA sequence motifs? Nat Biotechnol. 2006;24(4):423–25.
6. Fitch WM. Random sequences. J Mol Biol. 1983;163(2):171–6.
7. Garbelini JMC, Sanches DS, Pozo ATR. Expectation maximization based algorithm applied to DNA sequence motif finder. In: 2022 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2022; pp 1–8
8. Garbelini JMC, Sanches DS, Pozo ATR. Towards a better understanding of heuristic approaches applied to the biological motif discovery. In: Brazilian conference on intelligent systems. Springer, 2022; pp 180–194
9. Hashim FA, Mabrouk MS, Al-Atabany W. Review of different sequence motif finding algorithms. Avicenna J Med Biotechnol. 2019;11(2):130.
10. He Y, Shen Z, Zhang Q, et al. A survey on deep learning in DNA/RNA motif mining. Brief Bioinform. 2021;22(4):bbaa229
11. Heinz S, Benner C, Spann N, et al. Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and b cell identities. Mol Cell. 2010;38(4):576–89.
12. Kumar A, Hu MY, Mei Y, et al. CSSQ: a chip-seq signal quantifier pipeline. Front Cell Dev Biol. 2023;11:1167111.
13. Li Y, Ni P, Zhang S, et al. ProSampler: an ultrafast and accurate motif finder in large chip-seq datasets for combinatory motif discovery. Bioinformatics. 2019;35(22):4632–9.
14. Marçais G, Kingsford C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics. 2011;27(6):764–70.
15. Pevzner PA, Sze SH, et al. Combinatorial approaches to finding subtle signals in DNA sequences. In: ISMB, 2000; pp. 269–278
16. Sanderson C, Curtin R. Armadillo: a template-based c++ library for linear algebra. J Open Source Softw. 2016;1(2):26.
17. Smit AF, Hubley R, Green P. Repeatmasker 1996.
18. Stormo GD. DNA binding sites: representation and discovery. Bioinformatics. 2000;16(1):16–23.
19. Tatusov R, Lipman D. Dust, in the NCBI. Toolkit available at 1996; http://blast.wustl.edu/pub/dust
20. Zang C, Schones DE, Zeng C, et al. A clustering approach for identification of enriched domains from histone modification chip-seq data. Bioinformatics. 2009;25(15):1952–8.
21. Zhang Y, Liu T, Meyer CA, et al. Model-based analysis of chip-seq (MACS). Genome Biol. 2008;9(9):1–9.

## Publisher's Note