Discovering DNA Motifs Using Particle Swarm Optimization with Sequence Preprocessing

- CSC 530: Group 2 Project Report -

Grant Billings and Karthik Sanka

November 2022

Abstract

Sequence motifs arise from evolutionary processes and are known to have many biological functions, like transcription factor-binding sites. Motif discovery is an important problem in bioinformatics that helps understand the structure and function of molecules. Finding motifs given a set of DNA sequences is challenging due to the large search space and imperfect conservation of nucleotides at each position of the motif. Combinatorial optimization approaches have a very high time complexity making them unfeasible for very long sequences. Our method uses Particle Swarm Optimization[1], a nature inspired heuristic approach to solve non linear optimization problems. Our method initializes particles which correspond to potential motifs. These particles are randomly sampled from different parts of a sequence. Each particle's personal best score and the global best are stored. Particles are updated as a weighted function of the current character, character in global best and personal best. The local neighbourhood of the particle at every step is explored to detect partial alignments and overlaps. This is repeated for multiple iterations. We perform extensive experiments on artificial data to study the performance of our algorithm. Our results on both artificial sequences as well as real-world cotton fiber sequences [2] show that our method is able to detect motifs by outperforming other existing probabilistic methods such as MEME [3] in terms of speed, albeit at the cost of accuracy.

1 Introduction

In the context of bioinformatics, a motif is a unique subsequence on a protein or DNA sequence. Motifs help understand the structure and function of molecules making Motif discovery a very important problem in biology. Traditional algorithms using Position weight Matrix (PWM) although accurate have an exponential time complexity in the order of the length of sequences. Sequences in biology can get complex and very long, sometimes in the order of billions. This makes it impractical to use combinatorial algorithms solely based on PWMs. This led to increasing research in heuristic algorithms (MEME, GIBBS Sampling, etc) which are very fast. Our method uses Particle Swarm Optimization (PSO) to deal with this non linear optimization problem, which is to find the motif which maximizes the information criterion derived from the position weight matrix.

PSO works by generating a set of particles corresponding to candidate solutions. Then, it iteratively updates these particles based on the candidate's current state, its personal best and global best of all the candidate solutions. This process is run until convergence or a fixed number of iterations decided by the user.

Our method formulates Motif Finding by initializing particles (candidate solutions) to random start positions of one of the DNA sequences. We maintain a personal best parameter for each particle and global best parameter for each random initialization. We then update each character of the candidate motifs, using a weighted probability function that takes in the character in the current motif, personal best motif, global best motif and a character generated randomly as proposed in [4]. We iteratively update the motifs following the above update rule for 100 iterations. We then repeat the process for different random initialization of our particles. We have used multiple branch and bound optimization and statistical tricks in our implementation which makes it extremely fast also while keeping it reliable, although it begins to fail at longer sequences with less-conserved motifs.

Our implementation also uses a look-up table with precomputed scores for each combination of characters (discussed in section 2.3.1). These optimizations makes the algorithm to significantly faster which offer potentials to further improve accuracy with more restarts.

2 Materials and Methods

All software described here was developed interactively in `Jupyter Notebook` using `Julia 1.8`. The `MEME` program was run using a `Windows Subsystem for Linux` installation, and results were saved to text files for analysis in `Julia`.

2.1 Datasets and Sources

Implanted motif sequences were generated using our `GenerateTestData_ld` function. Briefly, the program generates **n** nucleotide sequences of length **l** from the DNA alphabet with the help of the `Random.randstring` function. A consensus motif of length **k** is generated, and each is mutated **d** times. The sequences and the correct consensus motifs are stored as type `Vector{Char}`. Simulated data was used in order to have a ground-truth of how well our algorithm is performing, and to be able to isolate a single variable at a time for testing. Sequences were stored as `Vector{Char}` rather than `Vector{String}` since this allowed us to take advantage of stack memory rather than heap, given the fixed size of the type `Char` in Julia.

A real dataset was sourced from Ando et. al. 2021 [2], a collection of cotton fiber gene expression data. We extracted the genes that were preferentially expressed only in Upland cotton in the day of pollination, which is the time period where cotton fiber initials form. This dataset was used in order to determine if biological inferences would be different if our algorithm was used as opposed to a standard motif discovery program. The positions of the genes were extracted from the `.gtf` file, and 1000 bp upstream of the transcription start site was extracted using the following commands:

```
for i in $(cat Ando_2021_fiber.txt); do fgrep "$i" Ghirsutum_527_v2.1.gene.gff3 | fgrep "longest=1"
     | fgrep "mRNA" >> Ando_genes.gff3; done
~/tools/bedtools getfasta -fi ../assembly/Ghirsutum_527_v2.0.fa -bed promoters.gff3 -s -nameOnly |
    fold -w 80 > promoters.fasta
```

Real results from MEME were collected using the following code, limiting the results to a single motif of the desired length. Benchmarking was performed for only a single experiment (described below) against MEME due to the cumbersome nature of moving results to and from the command line and `Julia`.

```
for i in $(ls /mnt/c/Users/grant/OneDrive/Documents/GitHub/CSC530_project/benchmarking/meme_test |
    sort -V); do /usr/bin/time -ao time.log -f "%e %M" ~/meme/bin/meme /mnt/c/Users/grant/OneDrive/
    Documents/GitHub/CSC530_project/benchmarking/meme_test/"$i" -dna -oc . -nostatus -time 14400 -
    mod oops -nmotifs 1 -minw 14 -maxw 14 -objfun classic -markov_order 0 -text | grep "^MOTIF " |
    awk '{print $2}' >> motifs; done
```

2.2 Initial Implementation Attempt

The initial implementation, which was not chosen for in-depth benchmarking due to very poor performance, will be briefly described here since we spent so much time on it. Our original proposal suggested the particles would represent starting positions of the motif in each DNA sequence, and Hamiltonian Monte Carlo (HMC) was to be used for local search. We were able to get the program working, including our own creation of a discrete version of HMC. However, this formulation of the problem is extremely slow and inaccurate because the search space is huge, and calculating the first degree approximation of the derivative (gradient) in **N** directions was prohibitively computationally expensive. Therefore, an alternative formulation, where particles represent consensus motifs (the approach described in our template paper) was used instead.

As a final note on this implementation - HMC may have some utility in motif discovery since it so effectively samples positions in the search space. A very creative approach taking much more time, and having some mathematical or theoretical basis, would be needed to adapt HMC to the motif-oriented formulation of the problem.

2.3 Final Implementation

Our PSO implementation is broken up into three key parts: preprocessing, initialization, and the main loop. The pre-processing steps are, to our knowledge, our own inventions. The formulae used for the match scores and position weight matrix originate from our template paper.

2.3.1 Preprocessing

Precomputing Scores Two scoring approaches are used, one that is "cheap" during the scanning phase and one that is "expensive" during the finding consensus phase. The scanning phase is the most time consuming step in the algorithm, so special care was taken to make this step very fast.

For the scanning phase, each position in the particle motif sequence is compared to a subsequence of the DNA sequence; this process is repeated $l - k + 1$ times for each of the places the motif could potentially be located in the DNA sequence. At each position the score is calculated as the sum across all the nucleotides i in the motif and j in a substring of the DNA sequence:

$$\sum_{i=1}^{k} \left[I(i=j) \times \left(1 + \log_4\left(\frac{0.25}{bg(i)}\right)\right) + I(i \neq j) \times \left(\log_4\left(\frac{0.25}{\sqrt{(bg(i) \times bg(j))}}\right)\right)\right]$$

Where bg is the background frequecy of the nucleotide across all the sequences and I is 1 if the statement evaluated to true and 0 otherwise. These logs are expensive to compute repeatedly, so we precomputed a lookup table that stores these values for every combination of i and j. This calculation is very similar to the information content (based on entropy), but tries to correct for differences in nucleotide frequency for matches and mismatches

Sequence Preprocessing Before the algorithm is run, each sequence is processed to find all the starting positions in the sequence that have an A at position 1, C at position 1, G at position 1, T at position 1, A at position 2, ... T at position **l**. The sequence itself and the desired motif length are used during the sequence pre-processing step. Take the following sequence for example `ACATTGA`. This information is stored in a dictionary where, for example, $Dict[1]['A'] => [1, 3, 7]$ represents an entry pointing to a vector listing all of the motif starting positions (as integers) that have an $'A'$ at their 1st position.

Precomputing Match Scores This data structure is then used, together with the precomputed scores, to make vectors that can be added together to score all starting positions in a DNA sequence. This is the data structure that is actually used for finding the subsequence with the highest score to a particle (proposed consensus motif). Using the same example `ACATTGA`, the sequence is represented by a dictionary with the following structure $Dict[1]['A'] => [1, 0, 1, 0, 0], Dict[2]['A'] => [1, 0, 0, 0, 0], Dict[3]['A'] => [0, 0, 0, 0, 1]$.

2.3.2 Initialization

The program is, by default, restarted 5 times. This is to avoid the program from entering a local optima and not outputting the consensus motif resulting in the highest score. 100 particles are initialized, and each one is updated 100 times while the program runs. Each particle is initialized to a $Vector\{Char\}$ of length k. The 100 sequences are chosen by picking random substrings, each from a random DNA sequence, of the input sequences being searched for motifs.

In particle swarm optimization, the score is also called the fitness. Each particles personal best fitness and the global best fitness (and the corresponding motif consensus sequences) are set to $-\infty$ and blank, respectively.

2.3.3 Main Loop

The main loop follows the structure suggested by our template paper[4].

Scanning The motif sequence stored in the particle is searched against each DNA sequence in a "scanning" step, which returns a $Vector\{Vector\{Char\}\}$, which has n sequences of length k. During each iteration in the scanning phase, these precomputed match score vectors are added together to get a match score for each position in the sequence given the particle (representing a motif), and the position with the highest score in this vector (the vector addition of k precomputed match score vectors) is used for the consensus step. This process is repeated for each of the DNA sequences, where the scan of each DNA sequence returns a potential motif in that sequence and its start site. The particle is revised at this step if the nucleotide consensus from these sequence differs from the motif sequence stored in the particle used to identify them.

We attempted to speed this step up by terminating the scanning step if the particle could not possible beat its current best after searching m sequences. This did not work due to the expensive matrix addition required, but this possibility will be further discussed later.

Finding Consensus A position weight matrix is generated using the results from the "scanning" step. The consensus score is calculated using the following formula, where $motif[i]$ is the nucleotide at position i and $profile[i, j]$ is the profile probability of the nucleotide j at the ith position in the sequences from "scanning":

$$\sum_{i=1}^{k} \sum_{j}^{j\in[A,C,G,T]} \left[profile[i,j] \times \log_2 \left(\frac{profile[i,j]}{bg[j} \right) \right]$$

The resulting sum is the fitness of the particle, representing the scores of the best set of subsequences resulting from searching using the consensus motif stored in the particle. If this score is better than the personal best and/or global best, the corresponding fitness and motif sequence are stored.

Check-Shift It is possible that the particle represents a consensus motif that is partially overlapping the one with the highest score. Therefore, every ten iterations, a check shift step is performed, where the vector of starting positions in the DNA sequences best matching the particle consensus motif are moved up or down $-5 : -1$ and $1 : 5$ steps. The consensus score is recomputed and if it exceeds the personal or global best, these terms are updated.

Updating The authors of our template paper proposed a novel update rule for communicating between particles, also known as "swarm intelligence". The update is a combination of the current motif sequence, the personal best, the global best, and a random nucleotide sequence. The probability of selecting a nucleotide at each position is proportional to the frequency of each nucleotide among these four sequences. In this way, particles will slowly become more similar to their personal best and the global best, although some of the nucleotides in the motif will likely remain the same as there is a contribution from the current sequence and the random sequence.

After updating, the loop begins again, with each of the particles being scanned against all the sequences, a consensus is found, the check-shift is sometimes performed, and finally the particles are updated through swarm intelligence.

2.4 Program Debugging and Unit Testing

Unit tests were developed for each of the functions in our program, using the `Test.@test` macro. Memory leaks or unnecessary allocations were found using memory profiling with `Profile.Allocs.@profile`. Likewise, functions using too much run time were found with `StatProfilerHTML.@profilehtml`. Flame graphs were produced and manually inspected. In most cases, memory and CPU slowdowns were resolved by pre-allocating arrays or using arrays of fixed sizes with the `StaticArrays` package.

2.5 Experiments

Five experiments were conducted to learn how our algorithm's run time and accuracy responds to different input instances, how our algorithm compares to MEME, and what sorts of information can be learned from our tool using real data. The program was run on a desktop PC with an Intel i5 10400F and 48GB of RAM. The exact combinations of variables chosen for the first two questions is described in Table **2**.

2.5.1 How does PSO's CPU and memory usage empirically scale with inputs?

To determine how program run time is impacted by sequence length, number of sequences, and motif length, the program was different levels of each variable. Default values were chosen as sequence length 500, number of sequences 100, and motif length 14 with 2 mutations because these settings provided decent accuracy (50% of consensus motifs correctly identified). We ran these tests because we expected the program's run time to be linear to these three parameters individually (approximately $\mathcal{O}(n \times l \times k)$).

2.5.2 How does accuracy of PSO change with inputs?

We hypothesized that PSO accuracy would be insensitive to motif length, but that it would go down if the sequences were longer or there were more sequences due to challenges associated with initial conditions. We also expected accuracy to be lower the number of mutations per implanted motif was high since the fitness of the correct motif would be low compared to other subsequences that happened to be similar across many of the DNA sequences.

2.5.3 What is PSO's performance when there are zero instances of the motif in some sequences?

In many biological datasets, not all sequences examined necessarily contain the motif. Sequences are aggregated based on some experiments, and experimental error or confounding can make identifying a set of sequences that definitely contains a motif can be very difficult. We wanted to see how our algorithm would work in such a scenario. Sequences were generated similarly to before, except in this case motifs of length 14 were implanted with 0 mutations in 100 sequences of length 500. The experiment was run for 0% : 10% : 90% of sequences lacking the implanted motif.

2.5.4 How does PSO performance compare against MEME?

MEME is the most widely used motif discovery software, so we expected it to be more accurate than our program. However, we hypothesized that potentially our program would be more time and memory efficient, at the expense of accuracy. For the same 100 instances of 100 sequences with length 500 and implanted motifs of length 14 with 2 mutations, we ran the command line version of MEME (using the aforementioned code in Section 3.1).

2.5.5 Does PSO detect anything interesting or convincing on new, real data?

The true test of new software is whether or not it reveals anything interesting on real data from living organisms. Using the cotton fiber promoter dataset, we searched for the highest scoring motifs of length 6, 8, 10, and 14. For the motif of length 6, we also used MEME to compare our results. WebLogo was used to make logo plots of the results for PSO and MEME so that we could comapre how different the results were. We then searched for promoter motifs in the `footprintDB` [5] database to see if there were any known transcription factor binding motifs with plausible biological interpretations for our results.

2.6 Evaluation Metrics

Accuracy was calculated as the proportion of correctly identified consensus motifs. Run time and memory were averaged over 100 instances for each set of input parameters. We chose not to use formal statistics since we were more interested in the qualitative trends in the data rather than getting exact numerical relationships between the input parameters and performance metrics.

3 Results

Full results of the computational experiments are given in **Table 2**.

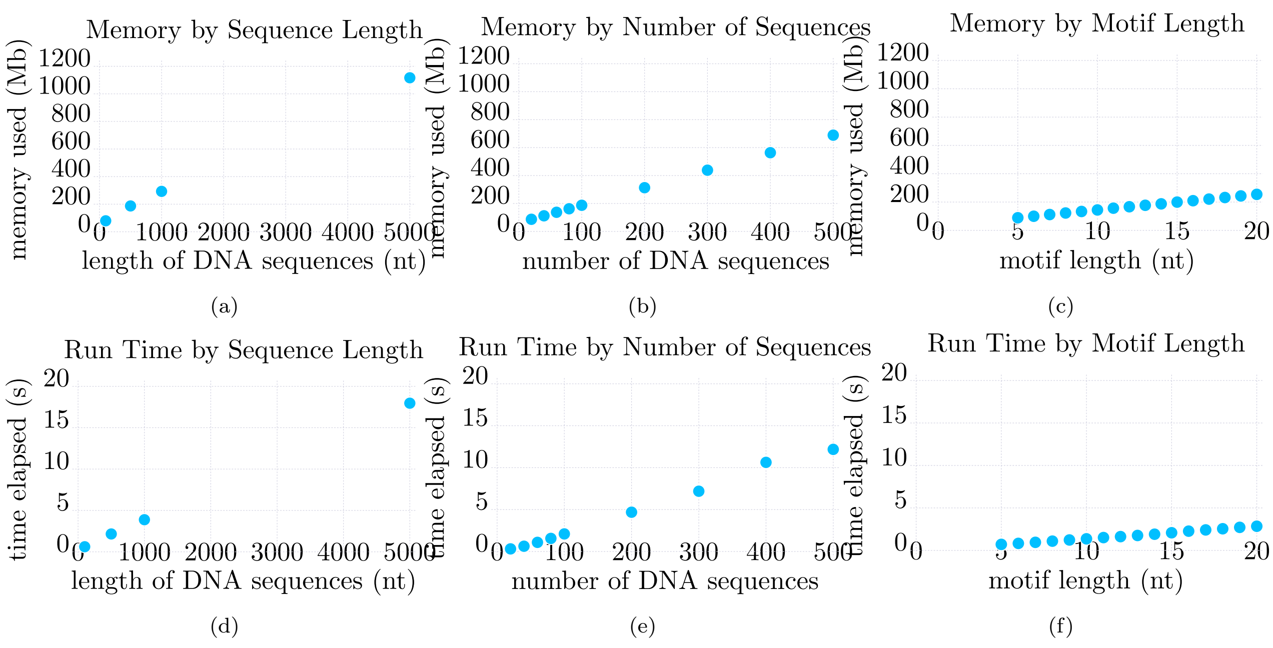


Figure 1: Memory usage (a-c) and run time (d-f) of PSO.

How does PSO's CPU and memory usage empirically scale with inputs? Our PSO implementation had run time and memory usage that scaled linearly with sequence length, number of sequences, and motif length (**Figure 1**). This is in line with our expectations, since the scanning step is the most time consuming, and the amount of work it must do is proportional to the product of all three of these characteristics of the instance. For a "normal dataset", run times of 3 seconds with 100 Mb of memory is expected.

We also checked how the run time changed in response to the number of restarts, number of particles, and number of updates per particle. The results are not shown here, but run time also increased linearly in response to each of these variables, which makes sense given how the program works.

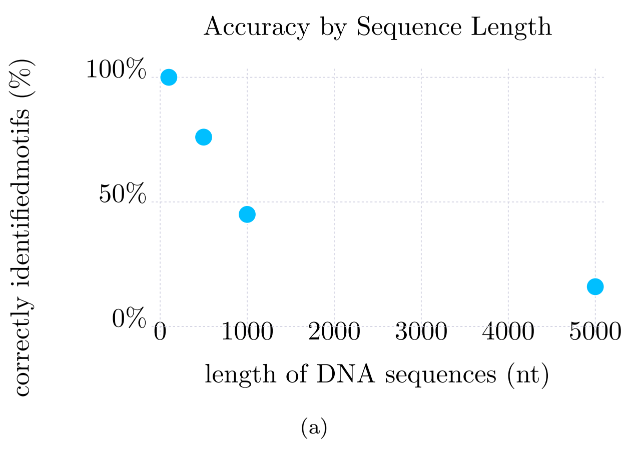
How does accuracy of PSO change with inputs? The accuracy did not change in response to the motif length or number of sequences (**Figure 2b** and **2c**). However, there was a major drop off in the accuracy in motif detection for sequence length at greater than 1000 nucleotides per sequences (**Figure 2a**) and more than three mutations (**Figure 2d**), at which point the accuracy got very close to zero.

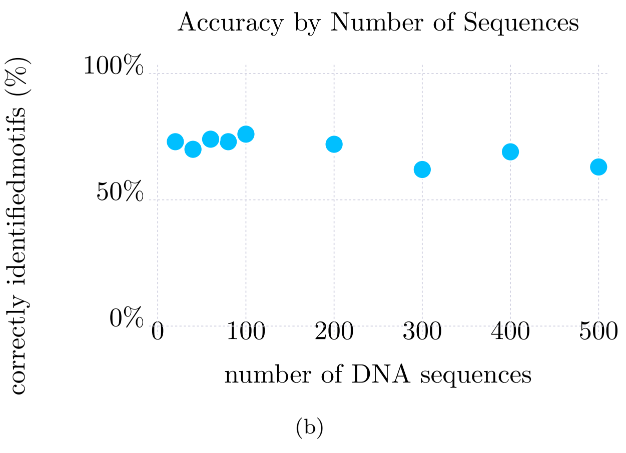
What is PSO's performance when there are zero instances of the motif in some sequences? PSO was robust to most of the sequences lacking an implanted motif (**Figure 3**). There was minimal effect on the algorithm's accuracy until it exceed 60% of sequences not having a motif, at which point the accuracy dropped to below 50% of motifs correctly identified.

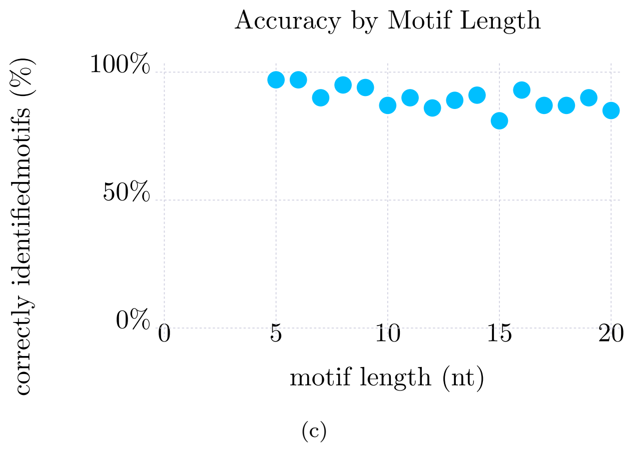
How does PSO performance compare against MEME? MEME was far more accurate, identifying 100% of the implanted motifs with 7% the memory of PSO (**Table 1**). However, PSO ran for less than half the amount of time.

Table 1: Comparison of performance of PSO vs MEME.

Software Used	Accuracy (%)	Time (s)	Memory (MB)
PSO	76%	2.2	187
MEME	100%	4.9	13







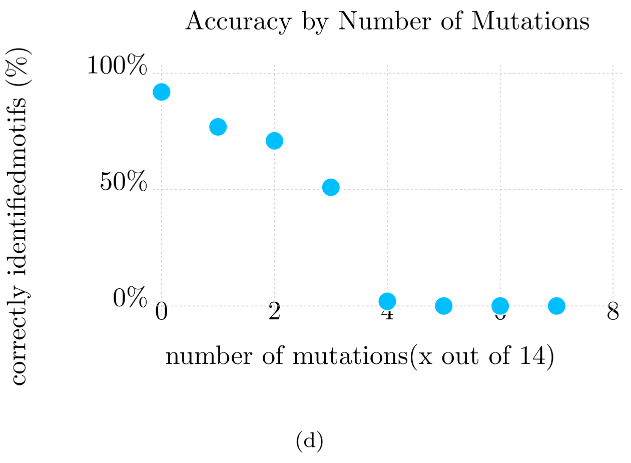


Figure 2: Accuracy of motif detection by PSO.

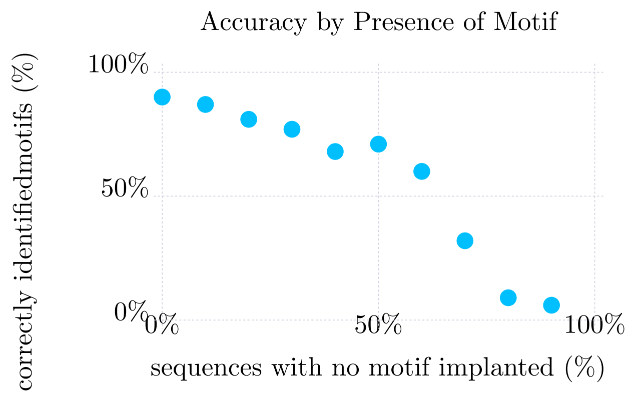
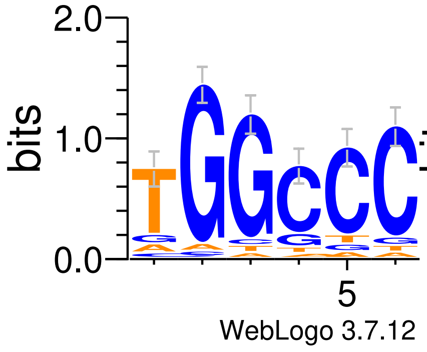
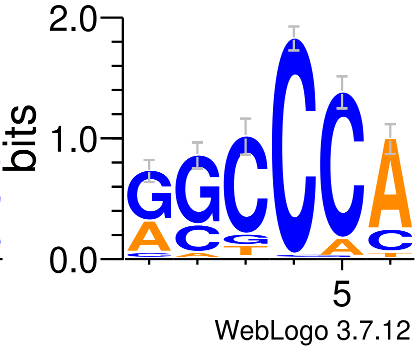


Figure 3: Accuracy of motif detection when some sequences do not have an implanted motif.



(a) Motif logo from PSO, IC=9.4.



(b) Motif logo from MEME, IC=9.9.

Figure 4: Comparison of motifs detected by PSO (a) and MEME (b).

Does PSO's detect anything interesting or convincing on new, real data? PSO and MEME detected very similar motifs in the cotton dataset, overlapping at five of six nucleotides (**Figure 4**). However, the information content of the MEME motif was 9.9, but it was only 9.4 from PSO. Additional motif sizes were attempted, but most of the motifs were only composed of A's and T's, which are bases common in gene promoters in plants but not commonly indicative of biologically relevant function (beyond the classic TATA box).

4 Discussion

On simulated data, our PSO implementation was fast, memory efficient, and usually produced the expected output. Testing showed that 5 reinitializations of 100 particles, each with 100 updates, had a good balance of speed and accuracy. Lei and Ruan did not have publicly available code for their implementation of the algorithm in C, so we were unable to compare our implementation to theirs. Lei and Ruan also suggested to increase the number of particles with the length of the sequences in the hope that at least one of the particles is partially overlapping with the true motif. This strategy would work well, but would essentially make the algorithm run in quadratic time with the length of the sequences (assuming the number of particles scaled linearly with the sequence length). Note again the algorithm runs with the following complexity:

$$\mathcal{O}(n \times l \times k \times (\texttt{restarts}) \times (\texttt{particles}) \times (\texttt{updates per particle}))$$

Examination of the memory and run time plots in **Figure 1** shows that **n**, the number of sequences, has the largest effect on run time. This is due to the fact that the "scanning" and "consensus" steps both slow down substantially as more and more sequences are examined. Future research could focus on only using a subset of the sequences in each step to avoid this limitation.

The most surprising finding was that PSO continued to work well, even if most sequences lacked the implanted motif (**Figure 3**). This suggests the applicability of the algorithm to real datasets, where all provided sequences may not necessarily contain the motif of interests. However, care would nee dot be taken to avoid entering sequences that are too long or searching for motifs with low conservation, since our algorithm performed poorly on these tests (**Figure 2**).

We also found substantial speed up from our preprocessing steps. The most time consuming step is the "scanning" step, where the motif is aligned against every potential starting position to find a best matching subsequence for each DNA sequence. We sped this step up by about 10× by precomputing the expensive logs and turning this step into basically vector addition followed by finding the maximum score in $\mathcal{O}(n)$ time.

Another speed up we implemented was terminating the scanning step if the particle could not possibly beat its current best. This can be enabled through the `optimistic_search`$(...; vanilla = false)$ optional argument, which we ended up disabling for testing. In preliminary testing, the matrix addition needed to find the optimistic best score (in a sort of branch-and-bound style) was just too slow compared to the vector addition performed for each sequence. Further mathematical decomposition of these formulas could likely be performed to reduce these operations to something faster than matrix addition.

The biggest shortcoming of our implementation was the lack of accuracy compared to MEME **Table 1**. The lack of accuracy could be due to the algorithm not running long enough, non-optimal weights being used in the update step, or not enough particles being initialized.

We also tested our implementation of PSO against the standard package MEME on real cotton promoter data (**Figure 4**). We found a DNA sequence motif TGGCCC, which was very similar to the GGGCCA motif found by meme. A search for the motif sequence against a promoter motif database found TCP23 was a match in Arabidopsis. The TCP transcription factors are a biologically plausible candidate for regulating the differentiation of cotton seed hairs into spinnable fibers. Further work would need to be done using bench lab and bioinformatic approaches to determine which program's output represents the biologically active motif.

Besides the results presented here, additional testing was done on changing the weights parameters of the algorithm. No improvement in model accuracy was observed by changing these parameters from their default value of 1.

PSO is an effective approach for identifying DNA sequence motifs. Our introduction of a pre-processing step is a significant advance in DNA motif discovery which could be used to make many heuristics that require identification of best matches run considerably faster. Future research in this area could focus on:

(1) Identifying weights that improve convergence rate without sacrificing accuracy;

(2) Using the pre-processed data structures directly for choosing better initialization states;

(3) Doing additional, in-depth comparison with other heuristic algorithms; and

(4) Adding the ability to parallelize particle simulations across multiple threads or cores simultaneously.

References

1. Kennedy, J. & Eberhart, R. *Particle swarm optimization* in *Proceedings of ICNN'95-international conference on neural networks* **4** (1995), 1942–1948.
2. Ando, A., Kirkbride, R. C., Jones, D. C., Grimwood, J. & Chen, Z. J. LCM and RNA-seq analyses revealed roles of cell cycle and translational regulation and homoeolog expression bias in cotton fiber cell initiation. *BMC genomics* **22,** 1–16 (2021).
3. Bailey, T. L. *et al.* MEME SUITE: tools for motif discovery and searching. *Nucleic acids research* **37,** W202–W208 (2009).
4. Lei, C. & Ruan, J. A particle swarm optimization-based algorithm for finding gapped motifs. *BioData mining* **3,** 1–12 (2010).
5. Sebastian, A. & Contreras-Moreira, B. footprintDB: a database of transcription factors with annotated cis elements and binding interfaces. *Bioinformatics* **30,** 258–265 (2014).

5 Additional Data

Table 2: Results from benchmarking.

Test Name	Number of Trials	Number of Sequences	Length of Sequences	Number of Mutations	Motif Length	Percent Motifs Correctly Identified	Run Time (s)	Memory (Mb)
seqlen	100	100	100	2	14	100%	0.614754	79.5418
seqlen	100	100	500	2	14	76%	2.15834	187.272
seqlen	100	100	1000	2	14	45%	3.88067	292.772
seqlen	100	100	5000	2	14	16%	17.9598	1116.64
seqnum	100	20	500	2	14	73%	0.311298	86.8951
seqnum	100	40	500	2	14	70%	0.627607	112.022
seqnum	100	60	500	2	14	74%	1.08752	137.001
seqnum	100	80	500	2	14	73%	1.56004	162.22
seqnum	100	100	500	2	14	76%	2.09558	187.27
seqnum	100	200	500	2	14	72%	4.68545	313.159
seqnum	100	300	500	2	14	62%	7.1868	438.278
seqnum	100	400	500	2	14	69%	10.6286	563.497
seqnum	100	500	500	2	14	63%	12.182	688.985
motiflen	100	100	500	0	5	97%	0.710067	88.9482
motiflen	100	100	500	0	6	97%	0.827825	99.8085
motiflen	100	100	500	0	7	90%	0.952198	111.811
motiflen	100	100	500	0	8	95%	1.0914	122.721
motiflen	100	100	500	0	9	94%	1.23055	133.32
motiflen	100	100	500	0	10	87%	1.36186	144.104
motiflen	100	100	500	0	11	90%	1.51059	156.76
motiflen	100	100	500	0	12	86%	1.62413	166.757
motiflen	100	100	500	0	13	89%	1.75795	177.084
motiflen	100	100	500	0	14	91%	1.90813	187.239
motiflen	100	100	500	0	15	81%	2.07742	199.544
motiflen	100	100	500	0	16	93%	2.26798	209.578
motiflen	100	100	500	0	17	87%	2.41162	220.811
motiflen	100	100	500	0	18	87%	2.55032	231.916
motiflen	100	100	500	0	19	90%	2.71399	243.097
motiflen	100	100	500	0	20	85%	2.84687	254.296
mutations	100	100	500	0	14	92%	1.9548	187.242
mutations	100	100	500	1	14	77%	1.96398	187.313
mutations	100	100	500	2	14	71%	1.96555	187.282
mutations	100	100	500	3	14	51%	1.97545	187.172
mutations	100	100	500	4	14	2%	1.98412	187.154
mutations	100	100	500	5	14	0%	1.96087	187.176
mutations	100	100	500	6	14	0%	1.96202	187.221
mutations	100	100	500	7	14	0%	1.98032	187.271