Bees? DNA Motif Discovery with Alternating Global-Local Search

- CSC 530: Group 2 Project Proposal -

Grant Billings and Karthik Sanka

09/30/2022

Abbreviations

(l, d): a planted motif of length l with d random changes; **DNA**: Deoxyribonucleic Acid; **HMC**: Hamiltonian Monte Carlo; **MEME**: Multiple Expectation Maximization for Motif Elicitation; **PSO**: Particle Swarm Optimization;

1 Executive Summary

Living organisms have genomes that evolve randomly over time, with natural selection working to increase the frequency of functionally beneficial sequences over generations. Motifs are non-random nucleotide sequences that in many cases have been shown to have biological function in gene regulation. Detection of motifs in sets of sequences is challenging because random mutations make exact matching of sequences ineffective, and brute force methods are very slow. The most popular software package for motif discovery is MEME, which works well but slows down significantly if many query sequences are provided. Motif discovery across large data sets has become an important step in genome analysis. *Software that can efficiently mine these sequences for motifs is needed.*

Nature-inspired algorithms have promise for DNA motif discovery since they broadly allow for efficient exploration of potential motifs while allowing good solutions to learn from each other. We propose use of Particle Swarm Optimization with Hamiltonian Monte Carlo (PSO-HMC) in alternating cycles of global and local search to quickly find motifs. Our algorithm will be tuned using implanted motifs in simulated data, tested on previously characterized benchmarking data, and finally applied to discover new sequence motifs in a cotton promoter sequence dataset. Precision, recall, F1 score, memory usage, and running time will be used to compare performance between our software and other widely used alternatives. At the end of the semester, we will present our findings in comparison to other available software in a poster and submit a project report. We will also create an animation showing the algorithm running and share it upload it to Wikipedia so others can gain a visual intuition for how PSO-HMC works. *Our work will contribute to the rapid characterization of large genomic datasets.*

2 Abstract

Biologists are interested in detecting motifs from DNA sequencing data because of their role in gene expression and chromatin architecture. The (l, d) planted motif problem is NP-complete, so heuristics are usually employed to find motifs. Non-probabilistic scoring functions for potential motifs and their positions in sequences are discrete, making the non-convex, non-smooth solution space very difficult to work with using traditional optimization techniques. Nature-inspired algorithms tend to excel in problems of this type due to the ability for the algorithm to exchange information on potential solutions. Here, we propose a novel method for motif discovery using 1) alternating rounds of Particle Swarm Optimization for efficient global exploration of the solution space; and 2) Hamiltonian Monte Carlo for detailed local search to avoid poor outcomes due to local optima. We will implement our algorithm in Julia, and benchmark on synthetic and real datasets. Key deliverables include a poster presentation and project report, as well as release

of a graphical representation of the algorithm and code into the public domain. We hope the speed and quality of the predicted motifs will help researchers generate hypotheses for motif sequences that can then be functionally validated through wet lab experiments.

3 Prior Work

Nature-inspired algorithms are elegant heuristic solutions to many of the most challenging computational problems in the sciences [1]. Particle swarm optimization (PSO) is one such algorithm, inspired by the behavior of cooperative populations of bees or birds that "fly together". The method is reviewed well by Banks et al 2007 [2]. The main use case of PSO is for finding globally optimal or near-optimal solutions for problems with a non-convex score function, where there may be many local optima throughout the search space.

Hamiltonian Monte Carlo is a sampling technique that has been widely adopted in the area of Bayesian statistics. The ideas of HMC were reviewed by [3]. The basic idea is to simulate a number of particles moving throughout the space. Typically some parameter is desired to be known across the space, such as the log-likelihood of the posterior distribution in statistics, or in the case of DNA motif discovery the score of different motifs beginning at specific starting positions in each sequence.

PSO has been used multiple times to find solutions for the motif discovery problem [4, 5, 6, 7]. For this proposal, Lei and Ruan, 2010 [5], served as a **template paper** inspiring our current work. Their main contribution was in implementing PSO for the motif finding problem by modifying the standard algorithm to take discrete inputs, as well as using both consensus sequences and position weight matrices for scoring. Since PSO does not invoke a gradient calculation (or even require the function be continuous or differentiable), it is highly flexible, but makes individual particles unable to explore the local search space efficiently without combining PSO with an additional algorithm. Lei and Ruan note that a main weakness of PSO is the inability to escape from local optima, which they circumvent by occasionally shifting the motif start sites to see if a better scoring solution is nearby. In other works such as Hardin and Rouchka, 2005 [4], an expectation maximization step is used to search close to the particles to see if a better score can be discovered before the swarm step.

[5](our template paper) makes a slight modification to the update rule which is usually defined in a continuous domain, as a summation of vectors. The modification is applied by using a combination of scaling factor, a weight function as well as a random function. This combination balances the algorithm and allows it to reach the optimum as well as explore the space. These updates are done iteratively till a fixed number of iterations are reached or the fitness value remains the same. PSO being a meta-heuristic algorithm suffers from the problem of bad initialization, so the algorithms is restarted from a random initialization several times to increase the probability of convergence. The algorithm also makes another important contribution to the motif finding problem using PSO, they make available a choice to the user to enter motif length and gap length to customize the algorithm to find gapped motifs.

4 Project Description

4.1 Data

Planted motifs of size $l \in [5, 15]$ and $d \in [1, \lfloor \frac{l}{2} \rfloor]$, corresponding to at most half of the nucleotides in the motif being mutated, will be simulated using Julia code (we have already developed the code for simulation). Motifs will be planted into 3 1000bp-long sequences during development, but the program will be evaluated on sets of more sequences (see **Computational Experiments**. Planted motifs will be used for tuning the hyper-parameters in PSO and HMC.

The program will also be benchmarked using manually curated motif binding sites from the online database resource `footprintDB` [8].

The program will be applied to find motifs in the 1000 bp upstream of 314 Upland cotton fiber-specific discovered in Ando et al, 2021 [9]. Genome sequences 1000 bp upstream of the start codon will be extracted from the v2.1 Upland cotton genome assembly [10] using `samtools faidx` [11].

4.2 The Algorithm

We propose to use this method for efficient local search between rounds of "swarming". Particles will search nearby using HMC, generating an approximate score density in its neighborhood. Then, the density will be updated so that new particle positions are likely to be drawn in the direction of the global best particle. A new position and velocity is drawn for each particle, and the cycle repeats until finishing criteria are met. Additional detail is presented in the Appendix.

4.3 Implementation

The PSO-HMC algorithm for motif discovery will be implemented using `Julia 1.8` [12] within a `Jupyter Notebook` [13]. Five main functions will need to be implemented:

- `Score`: returns the score for a set of sequences, motif length, and the proposed starting positions of the motif in each sequence.
- `FindMotifs`: takes DNA sequences and the motif length as inputs. Returns the ten best consensus motif sequences and the motif start sites in each DNA sequence. Iterates through HMC and PSO until stopping criteria is met.
- `HMC`: searches the space nearby the particle's current position using Hamiltonian dynamics. Utilizes a discrete estimate of a line integral (based on a discrete version of finite differences) rather than the gradient to determine the trajectory across the solution space. Returns an estimate of the score density in the region surrounding the particle.
- `PSO`: augments the score density near each particle given the output of `HMC` for the particle and the position of the best particle. Works by adding more weight to the density in the direction of the highest scoring particle.
- `UpdateParticle!`: draws a new position for each particle. Sets each particle's new velocity by composing its current velocity with a vector in the direction of highest scoring particle. Must be fine-tuned based on inertia and social attraction hyper-parameters.

Some packages like `StatsBase, Random` and `Distances` will be used for basic functionality not included in base Julia, but not for the PSO or HMC implementation. Additionally, the `WebLogo` available at their website (`http://weblogo.threeplusone.com/`) will be used for visualizing detected motifs.

4.4 Computational Experiments

Computational experiments will be conducted to answer three questions. In each case, the performance of our program will be compared to MEME and the PSO implementation in our template paper.

Sets of simulated input data will be generated according to all of the following combinations: sequence length `100, 1,000:1,000:10,000`; number of input sequences `20:20:200`; motif length `5:1:15`; and number of mutations in the planted motif `1:1:`$\lfloor \frac{l}{2} \rfloor$, resulting in a total of $11 \times 10 \times 11 \times 7 = 8,470$ combinations of input variables. At first, a set of 100 instances will be used, and if results are highly variable between runs of the algorithm on the same input data or different data (evaluated using boxplots of performance on individual instances), additional instances will be used.

We are proposing to use all combinations of the variables controlling the input motifs/sequences since it is possible that the algorithm will perform in unpredictable ways at different combinations of inputs. For example, maybe longer sequences and and shorter motif lengths will be harder for the algorithm to process, but having both of those conditions will increase the difficulty non-additively. Doing all combinations will help us understand what these interacting effects may do.

4.4.1 Question 1: How does PSO-HMC CPU and memory usage empirically scale to length and number of input sequences?

CPU and memory usage will be determined using the `@time` macro from the algorithm's performance on the input datasets varying in length and number of input sequences.

4.4.2 Question 2: How many mismatches can PSO-HMC tolerate in detecting motifs?

The precision, recall, and F1 score will be computed across the number of mutations and motif length.

4.4.3 Question 3: What is PSO-HMC performance when there are zero instances of the motif in some sequences?

Input data will be generated where different fractions `0%:10%:50%` of the sequences completely lack the planted motif. Precision, recall, and F1 score will be determined based on the fraction of input sequences with the planted motif.

4.4.4 Question 4: How does PSO-HMC perform on benchmarked, real transcription factor binding site datasets?

Manually annotated transcription factor binding site data from `footprintDB` will be tested against our implementation for precision, recall and F1 score to see how the algorithm works on real data.

4.4.5 Question 5: Does PSO-HMC detect anything interesting or convincing on new, real data? How do these results compare to other software?

The cotton fiber gene promoter dataset will be run through our program, MEME, and the PSO implementation in our template paper. Logo plots for the highest scoring motif in each sequence for the programs will be compared. The algorithm will be run for multiple motif lengths, and results will be manually inspected to determine which motif length should be displayed in the logo plots. It may be different motif lengths in different programs, since each will likely find different consensus motifs.

4.5 Evaluation and Statistics

We will evaluate and compare the results obtained from our algorithm using the following metrics and synthetic and annotated datasets.

- **Precision**: can be calculated by computing the ratio number of correctly predicted sites to the number of predicted sites.
- **Recall**: calculated by computing the number of correctly predicted sites to the binding sites present in the sequence.
- **F1 Score**: It is the harmonic mean of precision and recall and informs us about the balance between the two.
- **Running Time**: A run-time comparison will be plotted for our implementation compared to MEME and PSO from the template paper, as well as in response to changes in the input data size.
- **Memory Usage** The extra memory used to run the algorithm and its scalability to larger input sequences will be discussed in our analysis.

4.6 Deliverables

A project report and poster presentation to CSC 530 classmates will be finalized at the end of the project. The poster will include a diagram showing how the PSO-HMC algorithm work; plots showing running time vs motif length, number of mismatches in, input sequence number, input sequence length for the simulated dataset; sensitivity and specificy for the motifs in the benchmarked data set from the database; and example sequence logos for motifs detected from the cotton dataset.

Code will be made available on a public GitHub repo (`https://github.com/gtbil/CSC530_project`) at the project's close.

As a final deliverable, an animated gif showing how the algorithm works for two input sequences will be made and shared on a relevant Wikipedia page (like `https://en.wikipedia.org/wiki/Particle_swarm_optimization/`).

4.7 Anticipated Problems and Solutions

- *Stuck in local optima*: Restart randomly and at random initial positions. If this is ineffective, a heuristic will be used to find motif starting positions rather than random initialization.
- *Poor model convergence due to bad sequence initialization*: To be handled by starting with a random subsequence of the input sequence.
- *Hard to find hyper-parameters that work well in many situations*: Make hyper-parameters adaptive so they change based on the rate at which the best score changes.
- *Difficultly in applying or using HMC*: Implement a simpler Monte Carlo method, like Metropolis-Hastings.
- *Running time is too slow for all combinations of variables in computational experiments*: The experiments will be modified to only change on variable at a time. Additionally, use of the `Henry2` computational cluster will be employed if more compute is needed to complete the experiments. A memory or running time profiler will be used if the problems appear to be with our implementation/code.

5 Timeline

Major goals for completing parts of the project are given below.

- Oct 3—Nov 18: Writing of the project report.
- Oct 3—14: Implement the basic code of the algorithm in Julia.
- Oct 17—21: Unit testing and debugging.
- Oct 24—28: Hyper-parameter tuning on simulated data.
- Oct 31—Nov 4: Conduct computational experiments on simulated and manually curated datasets.
- Nov 7—11: Conduct computational experiment on actual promoter sequence data from cotton.
- Nov 15—18: Initial drafting of poster images and text. Statistical analysis of results.
- Nov 21—25: Arrangement of poster and revision. Printing on Friday Nov 25. Final editing of project report.
- Nov 29: Present poster findings to class.

6 Appendix

In PSO, each particle is initialized with some random position and velocity, corresponding to a proposed solution to the problem and the direction of the next proposed solution to the problem if the particle were left unperturbed. The score function is evaluated at each particle, and the particle with the best current score is noted. Then, a second set of velocity vectors between each particle and the best particle are computed. The initial velocity vector and this second vector are then composed to determine the next position of each particle. This is accomplished by weighting the hyper-parameters "inertia" (how much each particle wants to keep going in its current direction) and "social attraction" (how much each particle wants to head towards the best particle). The theoretical idea is that the algorithm will end up spending more time near the global optimum, converging quickly and exploring the solution space little if the social attraction is weighted highly, and the opposite if inertia is weighted highly. PSO does not involved calculating the gradient or any other parameters for score of nearby solutions.

In contrast, HMC classically uses the gradient to determine where particles should go next. In a broad sense, HMC starts by initializing the search with some proposed position in the search space. Then, the particle is pushed with some random velocity, and it "rolls" throughout the search space. The speed it rolls is determined by the gradient, such that the particle will end up exploring more of the space if it continues finding better and better potential solutions. The speed is determined by calculating the gradient (exactly if a closed form is available, otherwise some form of automatic differentiation is used to estimate the gradient) as the particle rolls, and the particles trajectory and speed are updated. Eventually, the particle stops moving, and the position and score are recorded. This process is repeated dozens are hundreds of times, and as the algorithm learns where good scores are (and are not) located, the proposed particle starting positions will be occur more frequently in areas of good scores. In this way, HMC is often able to efficiently explore the search space and approximate the density of the desired function without computing any quantities precisely. Since the formulation of the motif discovery problem

Algorithm 1 Motif Detection with PSO-HMC

for all motif lengths $k \in k_{\min}..k_{\max}$ **do** ▷ repeat algorithm for each plausible motif length
  Initialize a set M of particle position vectors and velocities m containing p particles in $\mathbb{Z}^n$
  Initialize a dictionary for the 10 best motif starting positions M_{best} and their scores
  Initialize a vector V for storing the scoring distribution information near each m
  $i \leftarrow 1$
  while not converged or $i <$ iteration limit **do** ▷ search until all particles are very close
    for all particles $m_i \in 1..p$ **do** ▷ do local search near each particle
      Evaluate the current score with $Score(m_i)$ ▷ score is hamming dist. against consensus
      if $Score(m_i) > \min(M_{\text{best}})$ **then**
        Add the score and position $M_{\text{best}}[m_i] \leftarrow Score(m_i)$ ▷ store for update step and output
      end if
      Initialize a dictionary O for the q motif starting positions and their scores
      for all sampled particles $o_j \in 1..q$ **do** ▷ local search step
        Allow the particle to roll in the solution space near m_i
        Add the resulting motif starting positions and scores $O[o_j] \leftarrow Score(o_j)$
        if $Score(o_i) > \min(M_{\text{best}})$ **then**
          Add the score and position $M_{\text{best}}[o_i] \leftarrow Score(o_i)$ ▷ store for update step and output
        end if
      end for
      $V[i] \leftarrow O$ ▷ store local search results for update step
    end for
    for all particles $m_i \in 1..p$ **do** ▷ global search step to pull particles towards best particle
      Use $V[i]$ and argmax(M_{best}) to propose a new position and direction for m_i
      $M_i \leftarrow m_i$
    end for
    $i \leftarrow i + 1$
  end while
  return M_{best}
end for

References

[1] Iztok Fister Jr et al. "A brief review of nature-inspired algorithms for optimization". In: *arXiv preprint arXiv:1307.4186* (2013).

[2] Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. "A review of particle swarm optimization. Part I: background and development". In: *Natural Computing* 6.4 (2007), pp. 467–484.

[3] Michael Betancourt. "A conceptual introduction to Hamiltonian Monte Carlo". In: *arXiv preprint arXiv:1701.02434* (2017).

[4] C Timothy Hardin and Eric C Rouchka. "DNA motif detection using particle swarm optimization and expectation-maximization". In: *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* IEEE. 2005, pp. 181–184.

[5] Chengwei Lei and Jianhua Ruan. "A particle swarm optimization-based algorithm for finding gapped motifs". In: *BioData mining* 3.1 (2010), pp. 1–12.

[6] U Srinivasulu Reddy, Michael Arock, and AV Reddy. "Planted (l, d)-motif finding using particle swarm optimization". In: *IJCA Special Issue ECQT* 2 (2010), pp. 51–56.

[7] Hongwei Ge et al. "Discovery of DNA motif utilising an integrated strategy based on random projection and particle swarm optimization". In: *Mathematical Problems in Engineering* 2019 (2019).

[8] Alvaro Sebastian and Bruno Contreras-Moreira. "footprintDB: a database of transcription factors with annotated cis elements and binding interfaces". In: *Bioinformatics* 30.2 (2014), pp. 258–265.

[9] Atsumi Ando et al. "LCM and RNA-seq analyses revealed roles of cell cycle and translational regulation and homoeolog expression bias in cotton fiber cell initiation". In: *BMC genomics* 22.1 (2021), pp. 1–16.

[10] Z Jeffrey Chen et al. "Genomic diversifications of five Gossypium allopolyploid species and their impact on cotton improvement". In: *Nature genetics* 52.5 (2020), pp. 525–533.

[11] Heng Li et al. "The sequence alignment/map format and SAMtools". In: *Bioinformatics* 25.16 (2009), pp. 2078–2079.

[12] Jeff Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: https://epubs.siam.org/doi/10.1137/141000671.

[13] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.