Motif search using genetic algorithms

Mohamed Darkaoui, Viktor Hura, Mounir Madmar

June 2022

Contents

1 Introduction

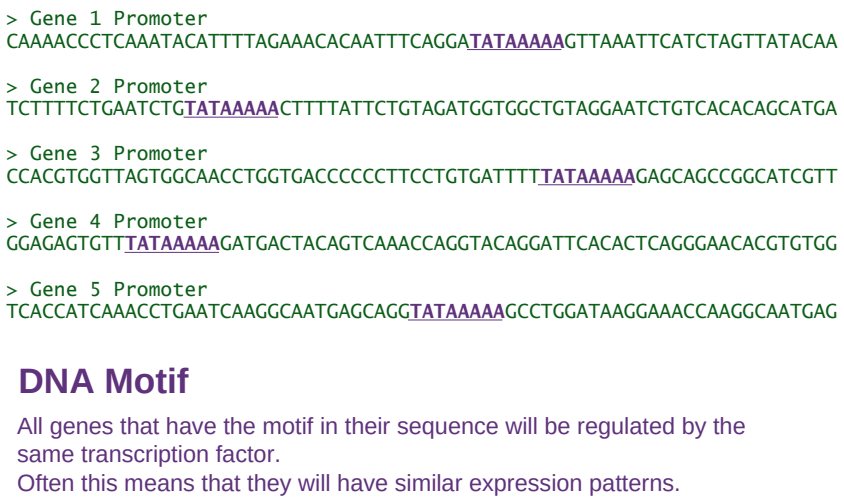


Figure 1: DNA Motifs

All organisms need hereditary information to function and develop properly. This information, called Deoxyribonucleic acid or DNA, comes in the form of large molecules centered in the nucleus of every cell in that organism. Its molecular structure is built up out of four different bases, which are represented by the letters A, C, G and T. A DNA sequence is represented by a combination of these letters. Genes are segments of the DNA, which contain instructions to produce certain proteins in the body.
Genes can be regulated by different motifs contained within the DNA promoter regions. Those are specific sequences that signal when a gene needs to be turned on. The goal of our project is to develop a new algorithm using genetic algorithms to find these sequences.

But it's not as simple as Figure 1 makes it out to be. In many cases, the DNA signal is not absolute due to degeneracy, but some error tolerance should be allowed. Thus we can't simply look for a reoccurring sub-string.
In the guest lecture by Dr. Pieter Meysman, we saw commonly used algorithms for this problem. In particular we have recreated the Gibbs motif sampling algorithm, to use as a benchmark for our new algorithm which will be described in the next section.

2 Genetic Algorithm

To find motifs in DNA sequences we will make use of a genetic algorithm.
As seen in the Natural Computation lecture by Kris Laukens, we know that a genetic algorithm is an optimisation algorithm inspired by Darwin's theory of evolution. This means that we must first define the solutions to our problem as organisms, and secondly define a fitness function to evaluate these solutions.
Our organism will be the "consensus instance", which is a motif instance of a fixed length. We will then look in each DNA sequence for lookalikes of this instance and assume that they are other instances of this motif. Our fitness function is a score of how much on average these instances look like the consensus instance.

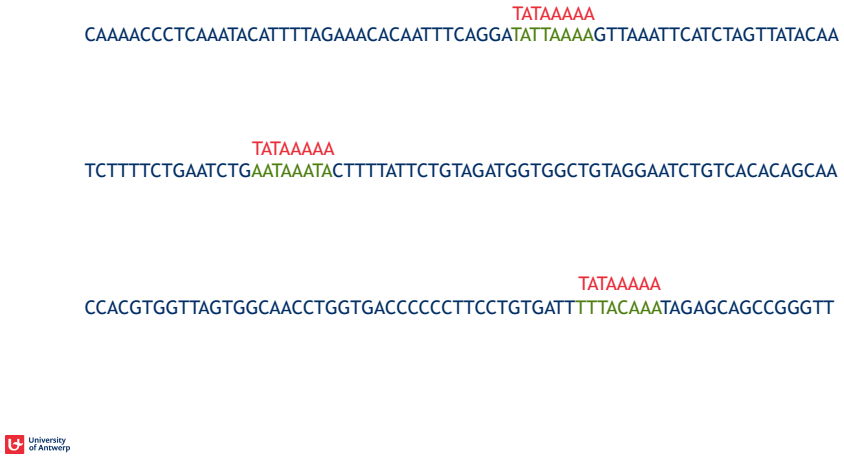


Figure 2: An organism and it's possible instances.

This is better illustrated in the above figure, where you can see that our organism is "TATAAAAA" and in green we found the best corresponding instances of the organism for each given DNA sequence. For each of these instances we can simply compare them to our organism and count how many letters match out of the total motif length. Then we take an average of all these normalised counts as our fitness function.
Now that we have defined an organism, which is in essence a string of fixed length, we need to define operations on this organism.

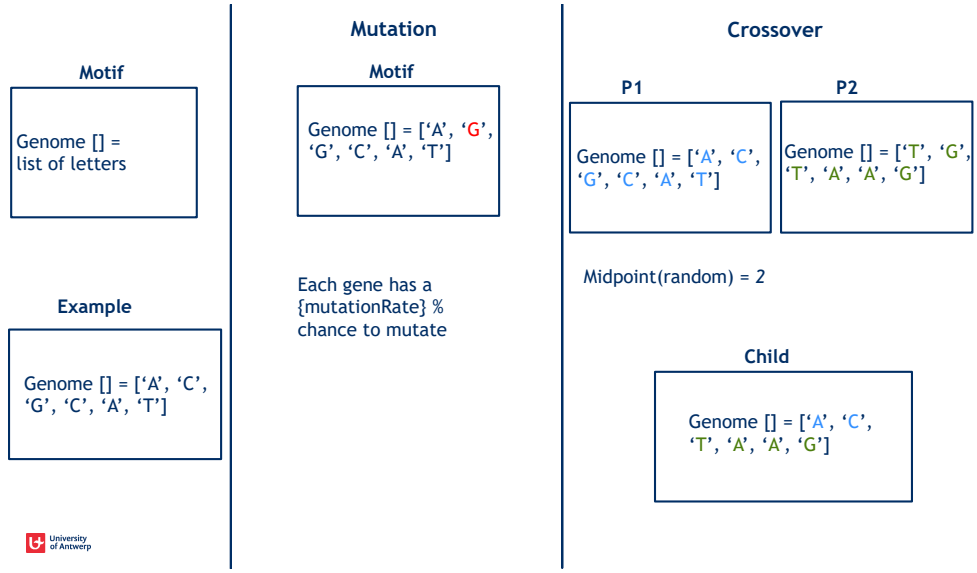


Figure 3: Operations on the organism.

In the figure above we define an organism as a list of genes, where each gene is a letter of this string. Each letter will have a certain percentage chance to mutate, which we define as the mutation rate. When a letter mutates, it changes to a different random letter.

When we have two organism, we need to apply crossover to create a child from them. We make use of single point crossover, where we choose one point along the child's genome and say that genes left of the point come from one parent, and those right of it are inherited from the other parent.

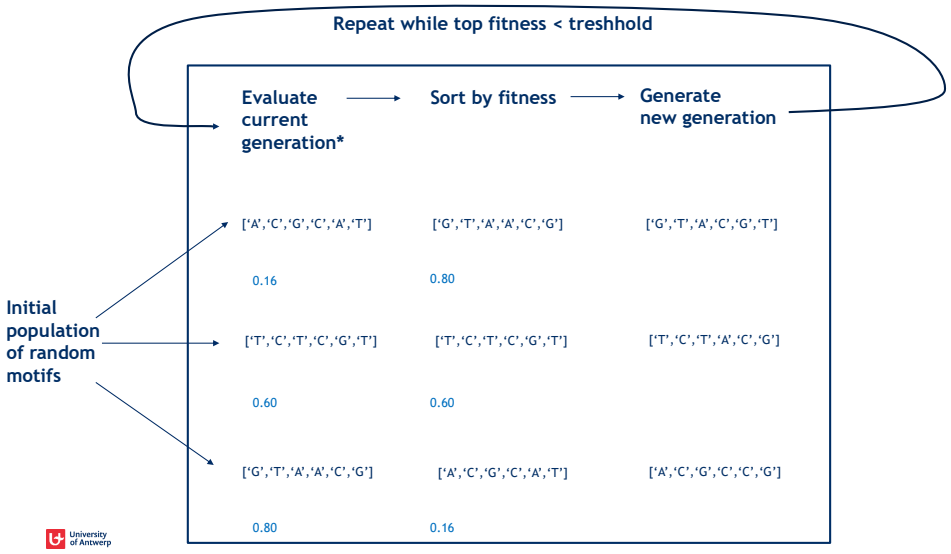


Figure 4: Pipeline

Once we have defined organisms and operations upon them, we can simulate evolution. The above figure shows the pipeline for such a genetic algorithm.

We begin with an initial population of organisms which are random strings in this case.
Next we apply our fitness function on each of our organism to give them their fitness scores by which they are sorted.
Finally we create a new generation from the current one which in turn can be evaluated again. This process will continue until the score of the best organism of a generation does not improve anymore or until this score has reached a desired threshold.

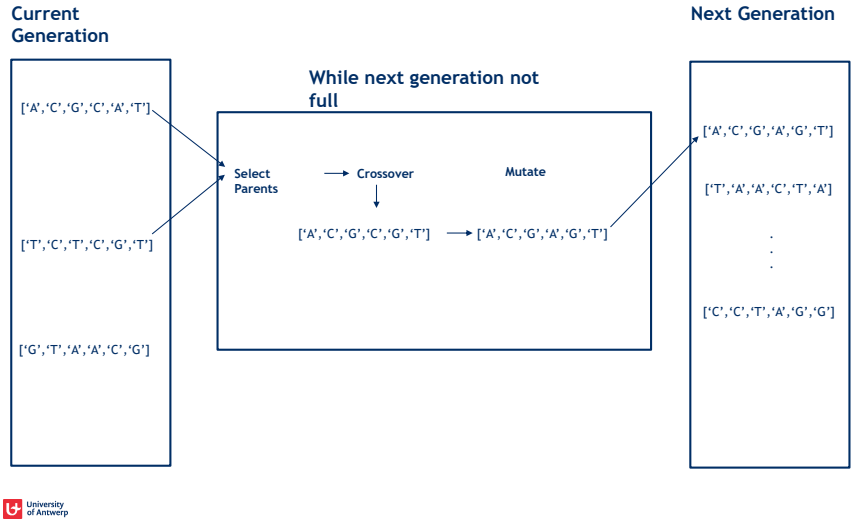


Figure 5: Pipeline

Generating a new generation happens in a few steps as seen in the figure above. First parents must be selected. The chance of a parent being selected should be proportional to their fitness score. We use tournament selection for our algorithm as the tournament parameter lets us fine-tune the balance between selection based on merit and random selection.

Next crossover is applied as defined before to create a child upon which a mutation might be induced before this child is added to the next generation of organisms.

We can repeat these steps to fill up the next generation. An exception is that we account for "elite" organisms, which are the top-n best organisms of their generation. These elite organisms get to copy themselves onto the next generation on top of just procreating. Usually there is only one elite organism, so that we don't lose our best found solution if their children happen to get worse fitness.

In the end we get an optimal consensus instance of a motif. But to be able to compare this algorithm to Gibbs motif sampling, in the end we take all the instances of this motif out of each given sequence, and use those to calculate the PSSM and the scores for each instance. We used pseudo-frequencies of 0.1 in all unseen bases of the PSSM.

3 Test results

The genetic algorithm is not guaranteed to always find an optimal solution as this depends on the search space and population size chosen. Thus we ran the genetic algorithm with different population sizes as shown in Table 1 to see if more optimal solutions existed. We also changed the tournament sizes to keep them in proportion with the population.

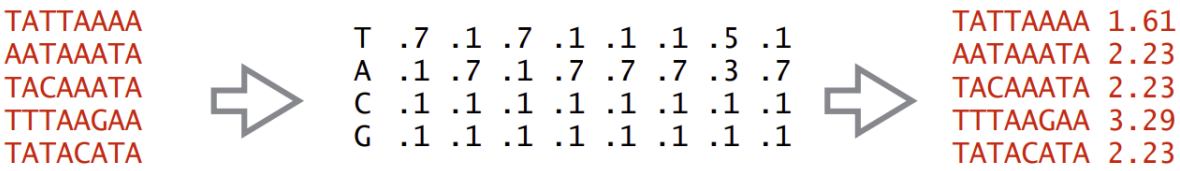
Table 1: Genetic algorithm parameters used.

Genetic Algorithm	population	elite	tournament	mutation rate
#1	16	1	2	2%
#2	32	1	2	2%
#3	64	1	3	2%
#4	128	1	3	2%
#5	256	1	6	2%

For the first test we generated 10 DNA sequences of 50 letters randomly. Then we ran the algorithms 50 times to find motif instances of length 10 and noted their execution times and motif scores.

Motif scores are calculated by taking all the found instances and creating a PSSM out of them. Then these instances are again fed through this PSSM to calculate a score for each of them. Finally we take the average of these instance scores to get our motif score.
You can see an example below, here the motif score would the average of all the scores on the right hand side.

Note that we use the negative logarithm of the output of the PSSM when we feed the instance through it. So the lower the score, the better.



Below you can see the results of this first test.

Table 2: Results with less sequences and lower sequence length.

Sequences:	10
Sequence length:	50
Motif length:	10
Tests:	50

Algorithm	Average motif score	Average search time (s)
Gibbs sampling	9.04288	0.01282
Genetic Algorithm #1	9.32464	0.28475
Genetic Algorithm #2	9.20012	0.50256
Genetic Algorithm #3	9.07553	0.77744
Genetic Algorithm #4	9.10533	1.42294
Genetic Algorithm #5	8.99614	1.77420

It is clear that for this simple search, we see that our genetic algorithms find motifs with similar scores as those found by Gibbs sampling. This means that our algorithm does actually work, but we can see that it is clearly slower than Gibbs sampling.
We can't say much yet about increasing the population size of genetic algorithms, as for this simple example, even a population size of 16 results in a close to optimal result. We can only say that it increases computation time, which was already clear given that we have to process more organisms.

The next test was performed with 50 sequences, each 400 letters long, which were also randomly generated. We again ran both algorithms to look for motifs of length 10.
We ran Gibbs and genetic algorithm 1 through 3, 50 times for this test. *But genetic algorithm 4 and 5 were ran only 10 times because their runs took much longer to execute.
Below are the results:

Table 3: Results with more sequences and higher sequence length.

Sequences:	50
Sequence length:	400
Motif length:	10
Tests:	50*

Algorithm	**Average motif score**	**Average search time (s)**
Gibbs sampling	5.14713	0.42961
Genetic Algorithm #1	4.37700	9.53665
Genetic Algorithm #2	4.29795	13.16963
Genetic Algorithm #3	4.17668	16.94488
Genetic Algorithm #4	4.09512	40.12148
Genetic Algorithm #5	3.89320	50.84550

In this test we can see again that the Gibbs algorithm is orders of magnitude faster.
But here we can see that the genetic algorithms find more optimal motifs. Keeping in mind that these are logarithmic scores, those are not insignificant improvements.
We also see that larger population sizes allow the algorithm to get closer to the global optima, at the cost of increased execution time.

In the next test we used the same sequences but searched for motifs of double the length (20). We tested only on genetic algorithm 1 and 2, but given the result of the previous test, we can expect better scores when increasing the population size.
Below are the results:

Table 4: Results with bigger motif length.

Sequences:	50
Sequence length:	400
Motif length:	20
Tests:	10

Algorithm	**Average motif score**	**Average search time (s)**
Gibbs sampling	11,86501005	0,49003258
Genetic Algorithm #1	10,84175872	35,8983025
Genetic Algorithm #2	9,726592314	51,09821664

Again as with the previous test, we can see that the genetic algorithms take much longer to execute.
But again we confirm that even our simple genetic algorithms are able to find more optimal motifs.

For our final test we decided to try the algorithms on a more 'organic' dataset. We used data from the University of California, Santa Cruz Genome Institute (UCSC)
In particular we used a subset of their cat genome sequences dataset labelled "ICGSC Felis_catus_8.0".

We took the first 50 sequences, and shortened them to a fixed length of 5000 letters. We ran Gibbs and genetic algorithm 2, 10 times each to find a motif of length 20.
Here are the results:

Table 5: Results with UCSC Cat sequences.

UCSC-Cat

Sequences:	50
Sequence length	5000
Motif length	20
Tests	10

	Average motif score	**Average search time (s)**
Gibbs sampling	13.20885	10.62160
Genetic Algorithm #2	14.39913	823.04195

Here we see that our genetic algorithm took almost 100 times longer and got worse motif scores. This is because algorithm 2 has small population size and the search space of this problem is much bigger than the ones that we saw before.
If one used a much larger population size, we would expect it to yield better results, perhaps better than Gibbs. But it is clear that the execution time for this algorithm would be many orders of magnitude longer.

Finally to illustrate the working of both algorithms, we took a subset of our cat sequences and used Gibbs and genetic algorithm 4 to find a motif of length 20. You can see the results below:

```
AGAGACTCCAAAATTGGACCCACAAAAGTATGGCCAACTAATCTTTGACAAAGCAGGAAAGAATATCCAATGGAAAAAAGACAGTCTCTTTTACAAATGG
TGCTGGGAGAACTGGACAGCAACATGCAGAAGGTTGAAACTAGACCACTTTCTCACACCATTCACAAAAATAAACTCAAAATGGATAAAGGACCTGAATG
CCATTTCTCATGAGTTGGGCTCATTCTTTCTCTTTGAAAATAACCAAAGATTTTTTTTTTTCTTTTTTAATGTGTATTTCTTTTGAGAGAGGGGAAGAGGG
AAAGAAAAAAGTAAATTAAAAAAATAATGTGTGATCAGTGGAGAGCCAGAAGTAAATCTCGAGTTGTATTAATATTTGGGTTAGCTATATAAATGTATAT
GGGAGAGAGCTTGGAACAAGCAGGAAATGGGGGTGGGGCAGCGAGACAGCGGTTTGACTGATGCCTTAATTAAGCTGGATTCTGTGACCCTGTCTCTGGT
AGATGCAAAAGTGGGAATTCCCTAAACCTTTATATGCCTATTAAAAATAATACCTCCACAGATGAGGAGTGTCCAATCAAATCTCCAATAAGCTCCAATG
GACGGAGGCAGTGGACAGCACGACGTACCCTGGCCCACAAGTGCCACGGGGTGGGCCAGCCCTGCTGGCCACTGCCTGCTGCACTGGTAGGGCGCCTTGG
ACACACACACACACACACACACAAACACACACACACGTGTGGTATAATTCCATATTTTTAAAATTAAAATCAAAACCAAACAGAAAATAAATTGCATATA
AAAAAAATTAAAAAAATATTAAATACAGGAAAATGATAATTTAATGCTGAATGACAGCTGCAAGAAACAAAACAATACACATAGAGTGATTTCGAGTATT
GTTGCTTCTAGAGGCGGCTGGCCCCCAGGCTGGACAGGGCGAGGGGAATGTGCTCAGATTGTACCTAAAGGACCGCAGTGTGGCAATGCTTCAAGTGAAA
CATTGACAATTGAGTGATAGTGTGGTTGGTTATGGATCTAGATTTCACCTTTCTTCAGAATTTTTAAGGCATTTCTGATTTTATTTTTAGCTTCTATTGT
TTACACACACCACACACACACACACACACACACACACACACACACACACACACACAACACACACACACGTGTGGTATAATTCCATATTTTTAAAATTAAAATCAA
CTCTTGCACATACACTCTAGAAGCTTTTCAAACTGTTGTTTCTGTGCTGTGTCTCAGGCTGAGTTAGTCAGTATATGGGCCCTTAAAGGGTGGGGTCTCC
CAAATACACACTAAGCACATGAAAAGATGTTCAATGTCACTAAATACTACAGAAATGACAATGAAGACTGTAATGAGATACGTCTTCATACCCATTAGGA
TTTTGAGAGAGAGAGAGAGAGAGAGAGAGAAGCATGAGCGAGGGGAGGGGCAGAGAGAGAGGGAGACACAGAATCTGAAGCCGACTCCAGGCTCTGAGCT
TTTGGTTATAGCTTAAGCAGTTGTCTGATTTGGGAACGCCTAGGTGGCTGTTTGTGACTGGCTGTCCTCAGGGTTCATTTTCTTGGATTCAAATGCACTG
TGTTGATAGTGCTGTTATAAACATTAGGGTCCATGTGCCTCTTCGAATCATTACTCCTGTATCCTTTGGATAAATACCTAGTAGTGCAATTGCTGGGTCA
TAATTTGTTCCAGAGTGAAGAGAGTGGAAGGAAAAAATAGAGTTGATAGAAGCGGTTGGGTATATGGAGAAAGAAACGGCAAAGACTCAATGATGGCTTC
CAGAAGTACCTGGGCAGCCGCGCGCTAGCGGCGCAGCGGACCTGGGCGCGCTTCATCCCCGGCCGCGTGGAGTTTTTTTCCAGTGAGCAGCCCCCCAACG
GAGAGAGAGAGAGAATTCCAAGCAGGCTCCAAGCCCAGTGTGGAGTCCGATGCAGAGCTCGATCTCAGCACCTTGAGATCATGACCTGAGCCAAAATCAA
```

Figure 6: Gibbs algorithm on UCSC Cat subset

Figure 7: Genetic algorithm on UCSC Cat subset

Genetic algorithm output includes the "consensus" instance that it evolved, shown in red. Both these motifs have similar score, which again confirms that our genetic algorithm does indeed work for this problem.

4 Conclusion

The genetic algorithm approach to finding hidden motifs in DNA sequences, does work given certain constraints.

Genetic algorithms in general tend to suffer from premature convergence. This is due to the fact that we have a fixed population size and eventually the 'biodiversity' becomes low as most organisms start to resemble each other and we are stuck in a local optimal solution.

This can be mitigated by using a more complex genetic algorithm, of which many exist. As an example one could use a genetic algorithm that models the organisms into species.

The other way to get closer to the optimal solution is to increase the population size, proportional to the search space of the problem. But comes at a cost of computation time.

This is why our simple algorithm seems to work well with smaller problem sequences and shorter motifs. But it would be very computationally expensive to scale it to bigger and more realistic motif searches.

Compared to Gibbs motif sampling algorithm we can also conclude that our approach is much slower. But the testing suggests that a genetic algorithm approach could lead to better motifs being found.

This would require more testing at a larger scale with more complex genetic algorithms, which could be an interesting research project to build upon the testing we have done.

References

[1] Dr. Pieter Meysman (2022) Computational biology, Deciphering the patterns hidden in DNA

[2] Professor Kris Laukens (2022) An introduction to Natural Computation

[3] The MEME Suite - Motif-based sequence analysis tools