Identification of motifs in biological sequences using genetic programming.

Àlex Velasco (Computer Engineering Student UAB)

Abstract — Current tools for motif discovery search patterns that are over-represented in DNA sequences but do not use DNA curvature or cofactors associated with the protein bind. We developed a tool that searches for motifs with a variable gap between patterns. The search is done using a genetic programming algorithm that searches for possible models that could be the motif and tries to fit them in a set of positive sequences with the motif against a control dataset. To evaluate the fitness of the organisms we have created an energy model for each component of the regulated bacterial promoters. The final genetic algorithm is able to find hidden motifs in synthetic sequences and real biological sequences.

Index Terms— Algorithm, binding site, biological motif, complexity control, framework, genetic programming, mutation operator, organism, placement, population, PSSM recognizer, sequences, tree structure.

1 Introduction

In biology, the process of creating a protein begins when DNA is read by a set of proteins (the RNA-polymerase holoenzyme), which transforms the double helix of DNA into a single RNA sequence[1]. Information codified on RNA is then read by ribosomes and translated into proteins. The initiation of transcription from DNA to RNA is regulated by specific proteins called transcription factors (TFs)[2]. This kind of proteins bind to specific DNA regions, also known as promoter regions of the transcription.

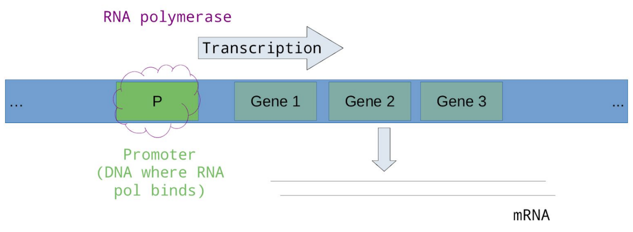
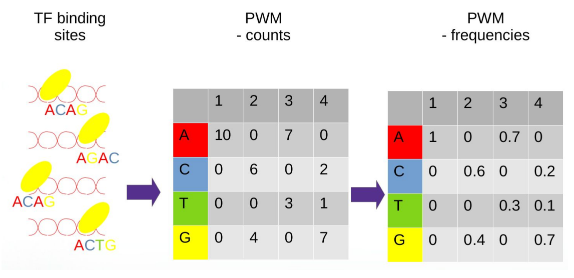


Fig. 1. Global structure of gene transcription from DNA to mRNA.

Transcription factor binding is governed by the recognition of specific patterns, with every TF recognizing a specific combination of DNA nucleobases. These patterns are known as biological sequence motifs.
The existence of TF-binding motifs were first reported in the late 1960's. Since then, scientists have created models to represent the interaction of TF and DNA, predominantly using position-specific weight matrices (PSWM)[3]. In the PSWM model, we consider DNA as a linear string of letters where the TF targets a specific pattern or motif within a long string representing the genome. This model assumes a linear sequential representation, as well positional independence. Even though these assumptions work well for many TFs, they are not intrinsically granted, since they ignore DNA structure, co-dependencies between positions and among TFs and co-factors.



	1	2	3	4
A	10	0	7	0
C	0	6	0	2
T	0	0	3	1
G	0	4	0	7

	1	2	3	4
A	1	0	0.7	0
C	0	0.6	0	0.2
T	0	0	0.3	0.1
G	0	0.4	0	0.7

Fig. 2. Process of creating a PWM from the TF's binding sites.

Motifs can be identified using experimental methods *in vitro* (e.g. EMSA) or *in vivo* (e.g. ChIP-Seq). ChIP-Seq is a type of massively-parallel chromatin immunoprecipitation assay that provides approximate TF-binding data across the entire genome[4].
Electrophoretic Mobility Shift Assays (EMSA) discriminate TF-bound from TF-unbound DNA using electrophoretic gel migration[5]. Both methods can identify relatively short sequences (i.e. ~200 bp) that are bound to a TF, but they cannot precisely outline the TF-binding site.
When it comes to analyzing sets of sequences suspected of harboring TF-binding sites, MEME is the most popular tool. MEME is an efficient tool for PSSM-based motif discovery[6]. However, many TFs do not target well-defined sequence motifs. Instead, they rely on recognition of the DNA curvature, internal co-dependencies and binding to associated cofactors. The lack of tools for appropriately modeling these TFs has led to a lag in their study, in spite of their biological importance. Hence, having the ability to model and discover more flexible TFs would open up many research lines in the future.
In this context, it must be stressed that the current capstone project is defined as an open-ended research project that incorporates Genetic Programming (GP). Evolutionary algorithms provide a flexible platform for the discovery of unconventional biological sequence motifs, as they can identify global solutions that incorporate both sequence and structure derived elements in their recognition[7].

2 OBJECTIVES

The main objective of this work is to develop and provide a proof-of-concept that a genetic programming (GP) platform can be adapted for discovery of TF binding motifs and, more generally, of the complex interplay between the components that make up regulated bacterial promoters.
This requires defining the problem in terms of an algorithmic solution, translating the biological problem to a computational framework and understanding the required biology and biochemistry background to generate an accurate implementation of the algorithm.
The development of such a GP framework requires that we incorporate basic elements of the known biology and biochemistry pertaining to the binding of TFs to bacterial promoters and to our current knowledge of bacterial promoter architecture. This development necessarily implies the definition of the overall GP framework in terms of basic workflow and Object-Oriented Programming (OOP) class structure, as well as adapting GP operators, including crossover and mutation, to reflect our knowledge of the biological entities involved .
The most critical part of an evolutionary algorithm is the definition of fitness function, since this will define the evolution of the population towards an optimal solution. To assess the effectiveness of proposed fitness functions and placement strategies, we will test the GP framework on both synthetic and real biological datasets.

3 STATE OF THE ART

3.1 MEME

Currently there are tools that allow motif discovery, like MEME, under the assumption of a rigid model (the size of the motif is predefined) and positional independence. This tool is able to discover overrepresented motifs in a set of sequences of DNA. It is important to highlight that when a bipartite motifs cannot be directly detected by MEME, which treats them as independent motifs and therefore imposes heftier restrictions on the relevance of each individual submotif. In other words, does not accept patterns with variable distance spacers[8].

3.2 Motif discovery methods

MEME is part of an extensive family of algorithms based on a target optimization criteria, like the information content of the motif or the potential binding energy to the DNA of the inferred model.
There have been multiple approaches based on different optimization methods:
MEME uses expectation maximization (EM) on a model of a fixed width and uses an initial estimate of the number of sites to search motifs. It then, sorts the possible sites by probabilities according to EM. Meme calculates the expected values (E-values) of the first *n* sites for different values of *n*. This procedure is repeated for different values of width and initial estimates of the number of sites. To finish, it outputs the motif with the lowest E-value [9].
Other methods use greedy algorithms to search for motifs. These methods first perform an alignment of n-mers and translate it into an alignment matrix. Then, from the matrix they directly extract the consensus sequence (here is the greedy method). From the alignment matrix build a weight matrix using a logarithmic ratio[10].
Gibbs sampling methods use 2 evolving data structures. A pattern description in the form of a probabilistic model of the background frequencies and a constituting alignment for the common pattern within the sequences. This pattern is obtained by locating the alignment that maximizes the ratio of the corresponding pattern to background probability[11].
GLAM (Gapless Local Alignment of Multiple sequences) is a C++ program that uses Gibbs sampling to detect and align similar regions of biological sequences and optimize the alignment using simulated annealing. It added several enhancements to the Gibbs sampling alignment method like the detection of the alignment width and the calculation of statistical significance[12].
ANN-Spec (Artificial Neural Network - Specificity) is a machine learning algorithm and can be applied for discovering un-gapped patterns in DNA sequence. ANN-Spec searches for parameters for a weight matrix that will maximize the specificity for binding sequences of a positive dataset compared with a background sequences dataset[13].

There have been other more recent approaches like qPMS9, that uses a tree to pass through all possible polymers using a pruning criteria to optimize the search and discard branches[14].

3.3 GP for motif discovery

There have been studies comparing linear and tree-based representations for unaligned protein sequences. This studies had been executed over existing genetic programming systems (LilGP) with very simple mutational operators [15].
This project has features that make it unique, the possibility of including shape recognition in the model architecture, which tries to model the motifs using nucleobases and the DNA structure[16]. It is also important that the problem is redefined as a promoter architecture inference, and not a single motif discovery, so the model also includes the interaction between single motif patterns.

4 Methodology

4.1 Dataset for the analysis

Datasets are written in FASTA format, a text format used to represent the sequences of nucleobases[17]. For every sequence, it consists of a line that includes metadata about the sequence, and a second line with the DNA sequence.
For every execution we will have 2 datasets, the positive dataset is the one that is supposed to include the motif we want to detect. The negative dataset or control dataset is a randomly generated dataset that does not have any motif inside, and will be used to check that the binding on the positive dataset is providing a positive benefit the negative dataset can not provide.
For this project we used 2 types of datasets, synthetic datasets were used mainly during the development to improve the algorithm, using motifs we created and knowing the exact solution. Biological datasets were used in the final stage of the project, to evaluate the GP algorithm with real data.
To generate the first synthetic positive datasets we generated a random mononucleotide raw sequence (200 bp). Then, on fixed positions 80 -100 inserted the following motif:

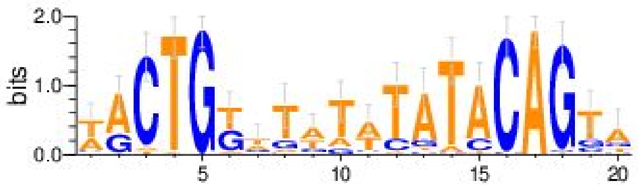


Fig. 3. Sequence logo[18] corresponding to the main fixed motif used on synthetic data.

This motif is clearly visible in the sequence logo. We also inserted a second motif at positions over 150, but with a variable spacing (0-9 bp) for each sequence:

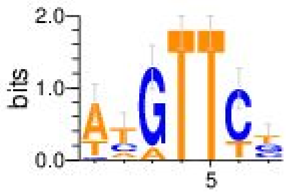


Fig. 4. Second motif inserted with a variable spacing.

The negative dataset was generated by generating pseudoreplicates of the positive dataset (sampling with replacement) using a given window size (w=2) . The goal is to maintain statistical parameters of the input of sequences at the dinucleotide level without recapitulating the sites of interest.

4.2 Coding

The project is hosted on GitHub, on a collaborative repository called ErillLab:
https://github.com/ErillLab/TF_GA
The code was developed using Python 3.7.3 on a Unix-like system. To manage all the libraries we used the virtual environment manager conda, but also virtalenv can be used.
There are 2 main dependences in the program:

- Biopython on version 1.76
- Numpy on version 1.11.3

The composition of this repository is a README.md with basic information about the project and the installation of dependencies, a docs folder including all the documentation for the future researchers and a src directory including the source code of the algorithm.

4.3 GP framework

The main component of the GA is the population it is used to find motifs along a set of DNA sequences. The subjects of the population are a data structure we will call organisms.

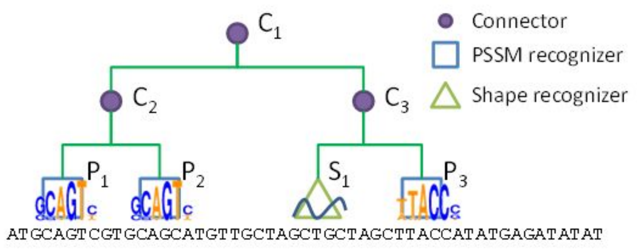


Fig. 5. Example of an organism with 3 connectors, 3 PSSM recognizers and 1 shape recognizer.

The organism is a data structure that stores all the components in a tree-like structure where we can find different types of nodes. The leaves of the tree are recognizers and the rest of the nodes are connectors.

Recognizers are connected by connectors, but connectors can also be connected through them.
PSSM-type recognizers are nodes used to recognize a specific sequence in the DNA using a PWM/PSSM matrix of a certain dimension (i.e. 4bp).

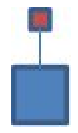


Fig. 6. PSSM recognizer visual representation.

Shape-type recognizers are nodes used to recognize a specific shape in the DNA based on certain parameters (e.g. twist, roll). We will not consider these except in acknowledging that in the future they could be available.
Connectors are nodes used to connect 2 nodes at a certain distance with a mean connector distance (**μ**) and a standard deviation (**σ**) between nodes.

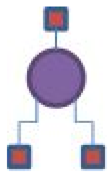


Fig. 7. Connector visual representation.

4.4 GP workflow

The basic execution of the GA is defined by the following flowchart:

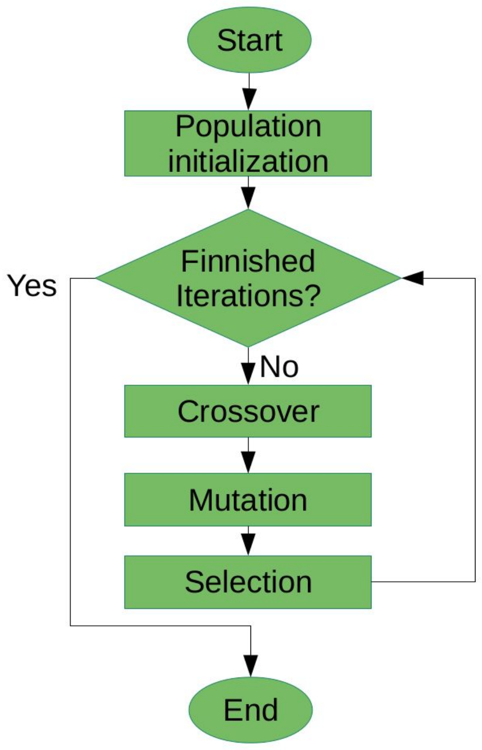


Fig. 8. Genetic programming basic workflow execution.

The population is firstly initialized using random organisms as subjects or reading organisms from the input file as a template to start the execution. If the input file does not contain enough organisms to fill the population, it can be automatically filled with random organisms or a copy of that organism.
Then, a predefined number of iterations are executed. The condition to finish the loop can be changed to other custom methods (i.e. when it reaches a certain fitness score or when new generations do not imply a considerable improvement).
On every iteration, organisms are randomly paired to run a deterministic crowding on the GP algorithm[19].
Crossover is done by selecting 2 random nodes from the paired parent organisms and creating a descendants swapping these 2 nodes. The crossover is not always done so we can find local solutions only by mutating existing parents. When the crossover is done, and childs are generated, we also compute the proximity from every child to both parents based on the number of nodes taken from each one.
Mutation consists in modifying children parameters on some nodes (i.e. modify the mean distance in a connector, modify columns of the PSSM recognizer) or changing the structure of the organism by adding or removing nodes.
To finish the iteration we compute the energy of every child and its closest parent and return the organism with the highest energy to the population.

4.5 Mutation and recombination operators

Mutation operators are executed over different parts of the organism. Mutation that affects the entire organism is usually used to stabilize the organism's complexity. There are 3 different general organism mutations
PSSM substitution mutation consists in replacing any node in the organism for a random PSSM recognizer. This allows the organism to reduce complexity if the node selected is a connector or maintain complexity if it is another recognizer.

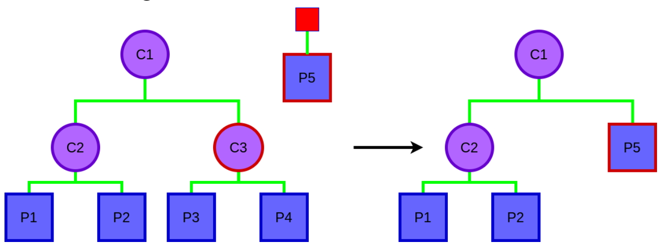


Fig. 9. Example of the PSSM substitution mutator for an organism.

Rise child mutation consists in selecting a node and moving it to its parent location, always reducing the organism complexity due to a node elimination (and the hanging tree associated).

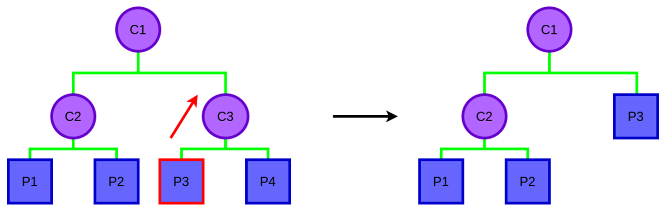


Fig. 10. Example of the rise child mutator for an organism.

Sunk node mutation consists in creating a connection and inserting it in the tree structure, connecting the current nodes to one random side of the new connection.

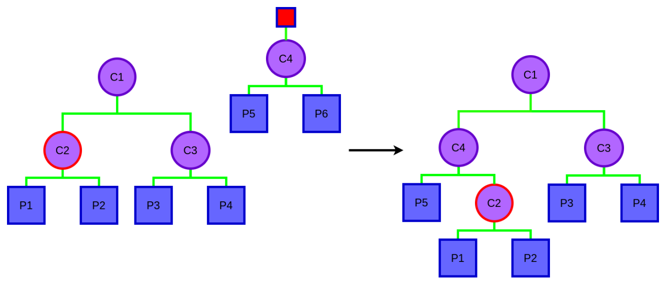


Fig. 11. Example of the sunk child mutator for an organism.

Nodes itself can also mutate, modifying internal parameters. Connections have 3 different mutations:
Sigma mutation consists in modifying the standard deviation associated with connector stiffness.
Mu mutation consists in modifying the mean distance, adjusting the distance between nodes.

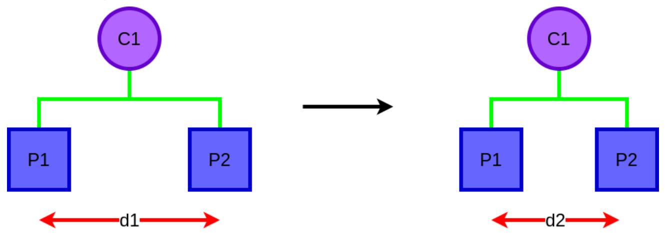


Fig. 12. Example of the mu mutator for a connector node.

Swap node mutation consists in swapping the location of the nodes of a single connector.

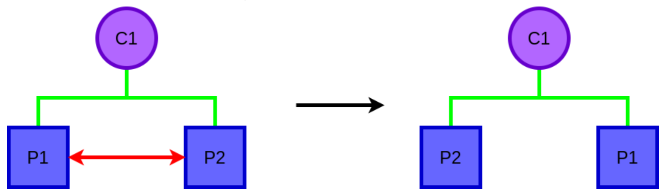


Fig. 13. Example of the swap node mutator for a connector node.

PSSM recognizers have a total of 5 mutator operators that act mainly on the PWM. To see an example see FIGURA DEL PRINCIPI.
Random column mutation consists in generating a new column on the pwm (a probability for each base) and substituting the new random column on the PWM.
Flip column mutation consists in selecting 2 random comuns in the PWM and swapping the values for all the nucleobases.
Flip rows mutation consists in selecting 2 random bases and swapping the values between bases in all the columns of the PWM.
There are two more mutators related to shifting the PWM to both sides. Shift mutation consists in moving 1 position all columns to one side modulus the number of columns.

The recombination operator is the main operator of the GA, that creates child organisms with a mixture of the parents. The operator selects 2 random nodes from the parents and swaps the nodes with the whole tree structure under it.

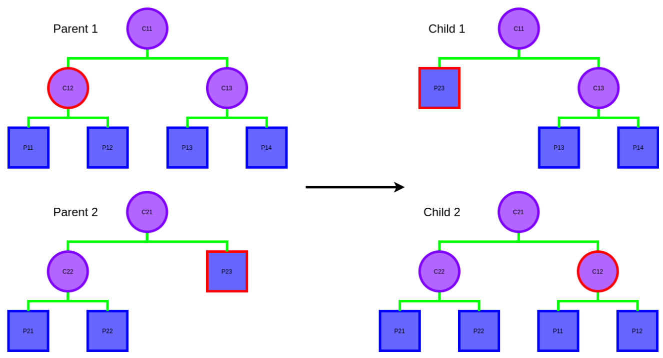


Fig. 14. Example of a crossover of 2 organisms. Parents do not disappear, they will continue if they are better than the children.

4.6 Definition of the organism energy model

One of the most important sections in the GP algorithm is the function we use to evaluate the fitness of each organism.
The fitness function is a composite function resulting from the evaluation of a positive dataset versus a control negative dataset. The fitness of each dataset is evaluated as the mean energy of all sequences contained.

The assignment of an energy value to an organism-sequence pair is the result of the evaluation of a particular placement of the organism on the sequence, and the overall energy of the resulting configuration. The energy is propagated upwards, from the PSWM recognizers that directly interact with the sequence, through their interconnection node, and all the way up to the root node that defines the organism.

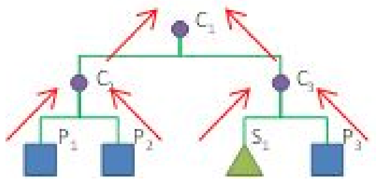


Fig. 15. Representation of the energy computation.

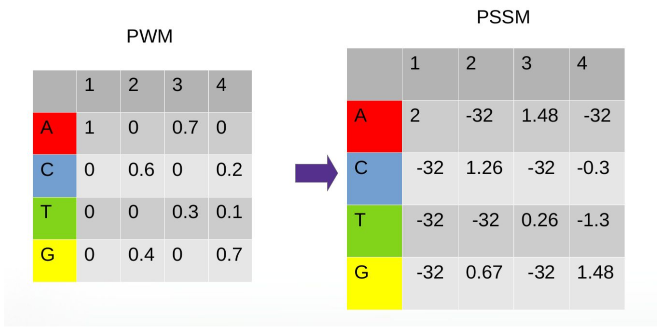
4.6.1 PSSM recognizer energy

PSSM recognizers first do a conversion of the PWM to a Position Specific Scoring Matrix (PSSM) to create a score based on the PWM values.
Each value in the matrix follows this conversion:

$$\mathrm{M}_{PSSM} = \log_2\left(\frac{M_{PWM}}{0.25}\right)$$

Fig. 16. Equation used to create PSSM from the PWM[3].

We divide the PWM value with the random probability of that base, in this case we have equal probability for every base. To deal with negative infinities, we add a pseudo-count to every value.



PWM

	1	2	3	4
A	1	0	0.7	0
C	0	0.6	0	0.2
T	0	0	0.3	0.1
G	0	0.4	0	0.7

PSSM

	1	2	3	4
A	2	-32	1.48	-32
C	-32	1.26	-32	-0.3
T	-32	-32	0.26	-1.3
G	-32	0.67	-32	1.48

Fig. 17. Example of a PSSM computation from the PWM.

Once we have placed the PSSM recognizer we evaluate it as the position-wise sum of the PSSM scores given the sequence.

4.6.2 Connector energy

Energy from connectors use an additive model that is composed of the energy of the elements they are connecting plus a term that provides energy based on the agreement between observed and connector distance.

$$E_{C_2} = E_{P_1} + E_{P_2} + \tau_1 \frac{1}{\log_{10}(10+\sigma^2)} e^{-\frac{(\mu - d(C_1,C_2))^2}{(1+2\sigma^2)}}$$

Fig. 18. Formula used to compute the energy of a connector.

And on the following figure we can see the energy contribution:

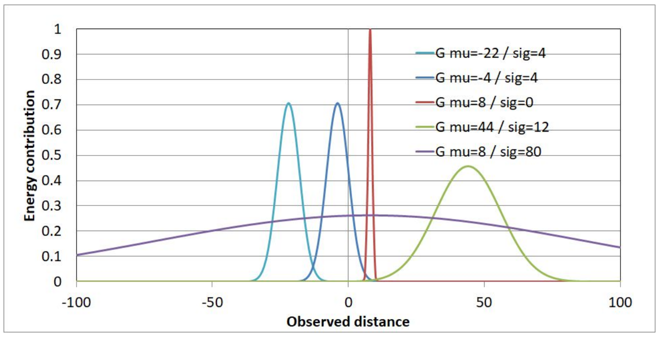


Fig. 19. Values of the energy contribution of the connector based on μ and σ values and the distance between nodes.

The term that provides energy for the connection includes the mean distance between nodes, standard deviation and a τ parameter to regulate energy provided by the connection.
The first placement strategy used was called Best All. This strategy places first all the recognizers in their best possible position in the sequence independently for all the recognizers. Once positions for each PSWM recognizer have been set, the energy is propagated upwards following the model seen above (Fig. XXX, Eq. XXX). This means that for every DNA sequence, every recognizer will have his maximum score, regardless of whether this optimizes the overall placement energy for the organism .

4.7 Complexity control strategies

A common issue in GP algorithms is the ability of organisms to improve their fitness by increasing in complexity. In the case of our relative (positive-to-negative) energy measurement, there is no direct selective pressure to increase complexity, but there is also no reason for organisms to optimize their size when attempting to maximize the fitness function over the sequence datasets. To control the dimension of organisms in terms of nodes we used implicit and explicit methods.
Implicit methods involve introduction of mutational operators that tend to reduce the organism complexity. It is the case of the PSSM substitution mutation and rise child mutation. In order to be able to adjust the mutational pressure, we also introduced a sunk node operator that always adds complexity.
Furthermore, we added an explicit complexity factor that directly controls the complexity of the organism by modulating the fitness function. We compute the mean fitness per node in the population at each generation, and then multiply that value times the number of nodes of the organism. So, if the organism has less nodes than the average for the population, it will receive a lower penalty. Lastly, we have introduced upper and lower bounds for

organism size that can be specified by the user, and which are implemented as hefty additions to the complexity factor of an organism if a bound is surpassed.
We call effective fitness to the fitness function with the explicit penalty applied.

5 Results and discussion

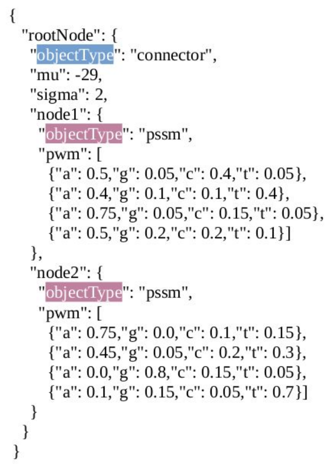
5.1 Operational GP framework

The program requires the execution of a basic Genetic Algorithm workflow and, at the beginning of the project, the idea of using an actual framework from the Python Library to simplify the development of the program was considered. Pyeasyga is an example of an easy to use library consulted before the development .However, using a GP library would require additional work to adapt our specific problem into a generic framework algorithm. There is also a need for us to have a total control every phase, so we can customize every step, applying necessary changes and trying to improve execution performance in terms of efficiency finding motifs in the dataset.

5.1.1 Issues in development

The first version of the program only executed the GA framework, finding huge organisms (with a massive number of nodes) that did not match the motif we created. At that point we were able only to execute the framework and after a few iterations see the result of the best organism (in terms of basic fitness).
During the development of the GA, we noticed that there was no way for us to extract conclusions about the fitness function due to the amount of random factors actively affecting the algorithm. This led us to the development of an alternative tool to evaluate any given organism outside the GP framework, and showing deterministic results.

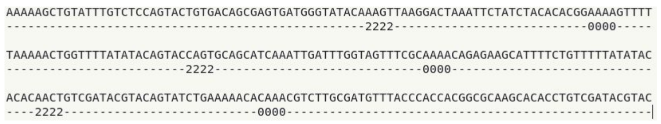
The implementation of this tool required a simple way to save and load organisms, allowing us to move organisms generated from the GA framework and use it on the test tool to evaluate the fitness function. A simple data text format to store organisms is JSON, that allows us to understand an organism's structure and add some specific custom organisms, in case we need it.



```
{
  "rootNode": {
    "objectType": "connector",
    "mu": -29,
    "sigma": 2,
    "node1": {
      "objectType": "pssm",
      "pwm": [
        {"a": 0.5,"g": 0.05,"c": 0.4,"t": 0.05},
        {"a": 0.4,"g": 0.1,"c": 0.1,"t": 0.4},
        {"a": 0.75,"g": 0.05,"c": 0.15,"t": 0.05},
        {"a": 0.5,"g": 0.2,"c": 0.2,"t": 0.1}]
    },
    "node2": {
      "objectType": "pssm",
      "pwm": [
        {"a": 0.75,"g": 0.0,"c": 0.1,"t": 0.15},
        {"a": 0.45,"g": 0.05,"c": 0.2,"t": 0.3},
        {"a": 0.0,"g": 0.8,"c": 0.15,"t": 0.05},
        {"a": 0.1,"g": 0.15,"c": 0.05,"t": 0.7}]
    }
  }
}
```

Fig. 20. Example of an organism stored in JSON format.

Another problem was the visualization of the binding of an organism to a sequence. This feature allows us to see how every organism's recognizer binds to DNA, and put it in an easy and readable format. This decision made the program much easier, because just by having a look at that file, we could conclude if the organism was correctly binding as we were expecting. See the format:

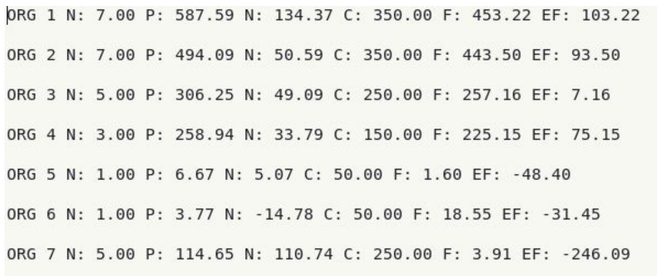


```
AAAAAGCTGTATTTGTCTCCAGTACTGTGACAGCGAGTGATGGGTATACAAAGTTAAGGACTAAATTCTATCTACACACGGAAAAGTTTT
--------------------------------------------------2222---------------------------0000-----

TAAAAACTGGTTTTATATACAGTACCAGTGCAGCATCAAATTGATTTGGTAGTTTCGCAAAACAGAGAAGCATTTTCTGTTTTTATATAC
-------------------------2222-----------------------------0000----------------------------

ACACAACTGTCGATACGTACAGTATCTGAAAAACACAAACGTCTTGCGATGTTTACCCACCACGGCGCAAGCACACCTGTCGATACGTAC
----2222---------------------------0000---------------------------------------------------|
```

Fig. 21. Example of PSSM recognizers binding to some sequences.

5.1.2 Test datasets

The tool used for evaluating organisms was created for the isolated evaluation of the organisms through all the datasets. It loads the objects' structure to import custom organisms and then executes a single evaluation of the organism to show the results on screen as seen here:



```
ORG 1 N: 7.00 P: 587.59 N: 134.37 C: 350.00 F: 453.22 EF: 103.22

ORG 2 N: 7.00 P: 494.09 N: 50.59 C: 350.00 F: 443.50 EF: 93.50

ORG 3 N: 5.00 P: 306.25 N: 49.09 C: 250.00 F: 257.16 EF: 7.16

ORG 4 N: 3.00 P: 258.94 N: 33.79 C: 150.00 F: 225.15 EF: 75.15

ORG 5 N: 1.00 P: 6.67 N: 5.07 C: 50.00 F: 1.60 EF: -48.40

ORG 6 N: 1.00 P: 3.77 N: -14.78 C: 50.00 F: 18.55 EF: -31.45

ORG 7 N: 5.00 P: 114.65 N: 110.74 C: 250.00 F: 3.91 EF: -246.09
```

Fig. 22. Example of 7 different organisms evaluated.

In the output we can see the execution of 7 organisms and some parameters that represent the values of some parameters of the execution related to: Number of nodes, evaluation on the positive dataset, evaluation on the negativa dataset, complexity applied, fitness of the organism and effective fitness.

5.1.3 Runtime analysis

Every run depends on some parameters of the GP algorithm. It includes the size of the population, the sampling in the positive and negative dataset and the number of iterations. Most of the program execution time is dedicated to the placement and those are the parameters that affect. It is also important the computer executing the program and the CPU power.
There are a few examples of the execution and parameters:

Population size	Sampling size	Iterations	Time (minutes)
50	200	50	25
50	100	50	10
100	100	50	14

Table 1. Examples with the time execution with a certain population, sampling size and iterations.

5.1.5 Parallelization

Output of every runtime analysis was initially stored in a single directory, so every run was saved using the same directory name. That led us to have sequential runs with a single CPU usage per execution. Also every run should be moved manually before starting another run. Parallelization consisted in creating multiple folders based on the start time of the execution so you can execute multiple runs saving every run a different directory. Is important to notice that every run should be executed with a second delay, but it allows us to optimize the time to recollect results about multiple parallel runs .

5.2 Complexity control

Mostly in the beginning of the project, we had a problem with the complexity of the solutions in the GP algorithm, that were organisms with an intractable number of nodes.

Our first approach of addressing this problem was trying a multiplicative model for the connectors, so the connector modulates the whole energy of its nodes.

In this context, the standard approach is to assume some form of additive contribution of the connector to the pair. That is, we assume that both PSSMs bind with some energy to their respective sites, and then the connector provides a boost to their binding. This is interpreted as the fact that the dimer, as a whole, has a higher binding energy (i.e. higher binding specificity). It can therefore search more efficiently for its binding sites in the genome (this is the 'pre-recruitment model'; alternatively, one can postulate that when one of the monomers is bound to DNA, it facilitates binding of the second monomer recruitment model).

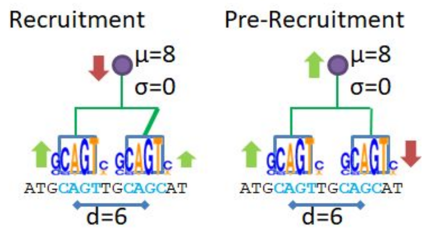


Fig. 23. Comparison between recruitment and pre-recruitment energy flow.

In other words, the binding energy of the two individual monomers (PSSMs) should remain intact, but be provided with an additive boost when the connector is present.

We performed several trials and found that a combination of mutational pressure and explicit fitness penalty kept the organism's complexity under control while allowing effectivent exploration of the search space.
On the following figure we can observe the average nodes per iterations and the nodes with the maximum fitness organism without using any placement enhancement:

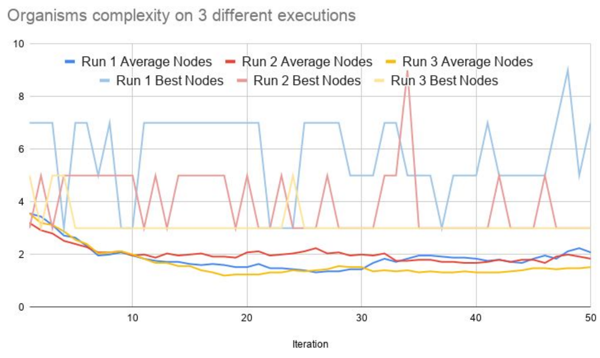


Fig. 24. Average nodes and nodes of the best organism of every iteration.

Complexity remains low (at about 2.0 nodes on average), but the winning organisms are less sharp and don't seem to locate the motif very efficiently. They are also more bloated than in the previous iterations. This suggests that selection is weaker, which sort of makes sense, in that the organisms are not selectively constrained to grow. Given this, the organisms are free to explore solutions that are not extremely efficient (in terms of discriminating well with few nodes). They are simply buffeted against by a strong downgrade mutational wind, but as long as they keep an edge, they are good to go. In other words, solutions that are minimally better by incorporating an extra connector will do okay, even though the mutational push will be to downgrade. They will therefore not be forced to identify the best possible configurations.

5.3 Placement enhancement

The program was based on a PSSM-centred model that firstly inserted the PSSM recognizer on the best binding site according to the PSSM. This is an issue due to the connector not providing much of an energy contribution in the case recognizers were not “well” placed on the maximum binding energy locations. To circumvent this issue, we changed the placement strategy so that each PSSM would propose several “good” binding positions, and the connnector would then chose the ones that satisfied it best.

To address the issue of organisms exploiting multiple connection energies from PSSMs that are, essentially, binding at the same location, we implement effective blocking of the entire sequence length covered by each PSSM. This is done both at the connector and PSSM levels, because the connector is the one that chooses the best configuration of PSSMs and establishes those positions as blocked.

To check the enhancement, we executed multiple tests of a solution organism with increasing values of placement options(1, 3, 9, 18, 27):

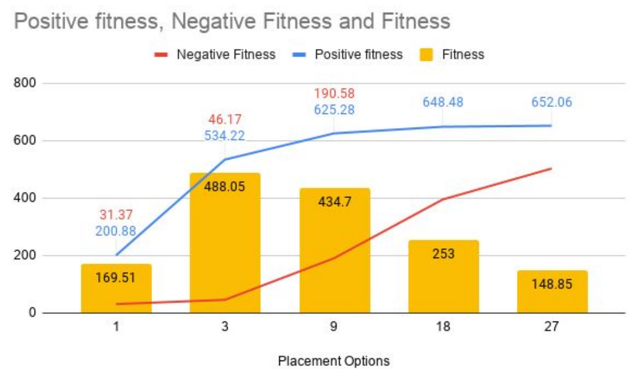
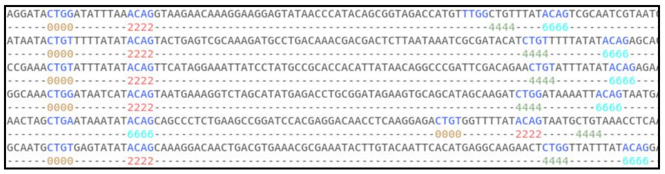


Fig. 25. Results of a “good” organism’s fitness in positive and negative datasets. Columns show the total fitness of the organism.

Something noticeable in these results is the fact that as placement options (PO) increase, the relative fitness of “good” solutions goes down. This is largely driven by the fact that there is no way to improve placement in the positive set with higher PO, but increasing the PO allows the system to find better ways of fitting negative set sequences.

However, the placement in the positive dataset is improved, finding the motif on almost every sequence.



```
AGGATACTGGATATTTAAACAGGTAAGAACAAAGGAAGGAGTATAACCCATACAGCGGTAGACCATGTTTGGCTGTTTATACAGTCGCAATCGTAAT
------0000--------2222--------------------------------------------------4444----6666-------------
ATAATACTGTTTTTATATACAGTACTGAGTCGCAAAGATGCCTGACAAACGACGACTCTTAATAAATCGCGATACATCTGTTTTTATATACAGAGCA
------0000--------2222-------------------------------------------------------4444--------6666----
CCGAAACTGTATTTATATACAGTTCATAGGAAATTATCCTATGCCGCACCACATTATAACAGGCCCGATTCGACAGAACTGTATTTATATACAGAGA
------0000--------2222--------------------------------------------------------4444--------6666---
GGCAAACTGGATAATCATACAGTAATGAAAGGTCTAGCATATGAGACCTGCGGATAGAAGTGCAGCATAGCAAGATCTGGATAAAATTACAGTAATGA
------0000--------2222------------------------------------------------------4444--------6666-----
AACTAGCTGAATAAATATACAGCAGCCCTCTGAAGCCGGATCCACGAGGACAACCTCAAGGAGACTGTGGTTTTATACAGTAATGCTGTAAACCTCAA
------------------6666------------------------------------------0000--------2222----4444--------
GCAATGCTGTGAGTATATACAGCAAAGGACAACTGACGTGAAACGCGAAATACTTGTACAATTCACATGAGGCAAGAACTCTGGTTATTTATACAGGA
------0000--------2222----------------------------------------------------------4444--------6666-
```

Fig. 25. Example of sequences and recognizers binding sites with 3 placement options. In blue is represented the motif to find and in yellow, red, green and cyan the recognizers binding sites.

5.4 Results on synthetic and biological data

After establishing an enhancement placement and a complexity control, we execute the GP algorithm with synthetic data and conclude that with enough iterations, it is able to find the motif with a good placement of the recognizers.

However, sometimes the solution gets stuck in a local optima that includes a part of the motif, but not the complete motif.

Real data is tested on promoter regions of the *lexA* gene from several Gammaproteobacteria and Betaproteobacteria species. Many of these promoters have two LexA-binding sites, but this is not a general rule (some have one, some three). We performed the first solid runs and the last one was the only targeting an instance of the motif. However, several of the runs seem to have hit at some point the correct solution.

Several runs later, after modifying some parameters about the population size and sample size of the sequences analyzed, the solutions looks in the same direction. The motifs are identified, but the solutions get stuck in a local optima. Some organisms recognize shifted versions of the motif, and it does not seem easy for the solutions to identify the proper solution.

GP algorithms are good at finding solutions, but maybe not that good at optimizing them. In the future it can be a good idea to execute an independent program only for optimizing the local solution to get the exact motif. Adding shape recognizers and improving the placement will be something that possibly will improve the overall in the motif finding.

6 Conclusion

In biology, transcription from DNA to mRNA is a complex process that involves specific proteins binding the promoter regions of DNA’s double helix. TFs not only bind to well-defined sequences motifs but also recognize DNA curvature, internal co-dependencies and binding of associated cofactors. Currently there is a lack of tools to model these dependencies that has led to a lag in the study of TFs with more flexible binding profiles.

In this project we have developed a genetic algorithm that allowed us to find motifs by modeling the components that make up regulated bacterial promoters. We have also developed a test tool that allowed us the possibility of evaluating independent organisms and extracting conclusions without executing the genetic algorithm. Development has shown concepts that could must be addressed in future research (i.e. complexity control, placement of the recognizers and the energy model used in organisms).

Local optimization is an issue that implies that final organisms do not model the exact motif but a close approximation to it.

However, we achieved successful results in both real and synthetic data, even though some features should be improved in order to have a more optimal result and a better performance of the GP algorithm.

Acknowledgements

I would like thank Joan Serra-Sagrista and Ivan Erill for all the discussion and support on the current project. I

also wish to thank my family and friends for their support in time of confinement.

7 BIBLIOGRAPHY

[1] "Overview: Gene regulation in bacteria (article)," *Khan Academy*. https://www.khanacademy.org/science/biology/gene-regulation/gene-regulation-in-bacteria/a/overview-gene-regulation-in-bacteria (accessed Jun. 08, 2020).

[2] "Transcription factor," *Wikipedia*. May 30, 2020, Accessed: Jun. 09, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Transcription_factor&oldid=959782582.

[3] "Position weight matrix," *Wikipedia*. Apr. 18, 2020, Accessed: Jun. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Position_weight_matrix&oldid=951611110.

[4] "ChIP sequencing," *Wikipedia*. Jun. 21, 2020, Accessed: Jun. 27, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ChIP_sequencing&oldid=963650023.

[5] "Electrophoretic mobility shift assay," *Wikipedia*. Apr. 07, 2020, Accessed: Jun. 27, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Electrophoretic_mobility_shift_assay&oldid=949641213.

[6] F. Zambelli, G. Pesole, and G. Pavesi, "Motif discovery and transcription factor binding sites before and after the next-generation sequencing era," *Brief. Bioinform.*, vol. 14, no. 2, pp. 225–237, Mar. 2013, doi: 10.1093/bib/bbs016.

[7] R. Chauhan and P. Agarwal, "A Review: Applying Genetic Algorithms for Motif Discovery," *Int. J. Comput. Technol. Appl.*, vol. 03, Jul. 2012.

[8] "Multiple EM for Motif Elicitation," *Wikipedia*. Oct. 15, 2018, Accessed: Jun. 20, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Multiple_EM_for_Motif_Elicitation&oldid=864227879.

[9] "MEME - MEME Suite." http://web.mit.edu/meme_v4.11.4/share/doc/meme.html (accessed Jun. 20, 2020).

[10] G. Z. Hertz and G. D. Stormo, "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences," *Bioinforma. Oxf. Engl.*, vol. 15, no. 7–8, pp. 563–577, Aug. 1999, doi: 10.1093/bioinformatics/15.7.563.

[11] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment," *Science*, vol. 262, no. 5131, pp. 208–214, Oct. 1993, doi: 10.1126/science.8211139.

[12] M. C. Frith, U. Hansen, J. L. Spouge, and Z. Weng, "Finding functional sequence elements by multiple local alignment," *Nucleic Acids Res.*, vol. 32, no. 1, pp. 189–200, 2004, doi: 10.1093/nar/gkh169.

[13] C. T. Workman and G. D. Stormo, "ANN-Spec: a method for discovering transcription factor binding sites with improved specificity," *Pac. Symp. Biocomput. Pac. Symp. Biocomput.*, pp. 467–478, 2000, doi: 10.1142/9789814447331_0044.

[14] M. Nicolae and S. Rajasekaran, "qPMS9: An Efficient Algorithm for Quorum Planted Motif Search," *Sci. Rep.*, vol. 5, no. 1, Art. no. 1, Jan. 2015, doi: 10.1038/srep07813.

[15] R. Seehuus, A. Tveit, and O. Edsberg, "Discovering biological motifs with genetic programming," in *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, Washington DC, USA, 2005, p. 401, doi: 10.1145/1068009.1068074.

[16] M. A. H. Samee, B. G. Bruneau, and K. S. Pollard, "A De Novo Shape Motif Discovery Algorithm Reveals Preferences of Transcription Factors for DNA Shape Beyond Sequence Motifs," *Cell Syst.*, vol. 8, no. 1, pp. 27-42.e6, 23 2019, doi: 10.1016/j.cels.2018.12.001.

[17] "FASTA format," *Wikipedia*. May 11, 2020, Accessed: Jun. 27, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=FASTA_format&oldid=956134292.

[18] "Sequence logo," *Wikipedia*. May 11, 2020, Accessed: Jun. 27, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Sequence_logo&oldid=956012026.

[19] S. F. Galan and O. J. Mengshoel, "Generalized crowding for genetic algorithms," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, Portland, Oregon, USA, 2010, p. 775, doi: 10.1145/1830483.1830620.