Figmop tutorial

Version 1.1
Written by Dave Curran, July 30, 2015
curran.dave.m@gmail.com

Table of Contents

Quick start

Run Figmop on protein sequences:
```
figmop MEME_FILE.txt --prot PROT_FILE.fa -o OUTPUT_NAME
```

Generate pattern file:
```
figmop -g LENGTH -f PATTERN_FILE.py
```

Run Figmop on a genome:
```
figmop PATTERN_FILE.py GENOME_FILE.fa GENOME_MAST_DIR -p PADDING
-o OUTPUT_NAME
```

Figmop overview

Figmop was originally written to deal with a case of bad gene models. We had recently published the genome of a parasitic worm, and I was looking for cytochrome P450 (Cyp) genes; comparisons led me to expect ~40 members. Unfortunately the gene models only contained one that appeared to be near full length, along with a handful of fragmented sequences. Modern gene finders like AUGUSTUS get most gene models mostly correct, but they are not perfect. They did a good job on the rest of the genes, but for some unknown reason failed utterly on the Cyps. I then tried collecting known Cyps from related species and searching the raw genome using tBLASTn. This found a fair number of hits, but the majority of potential genes only contained one or two short (~10 amino acid) hits. This would have made assembling the coding sequences very difficult and time-consuming, and as it turned out BLAST missed a number of the likely genes. Figmop handles this situation much better, as it abstracts sequences more than BLAST does and so is better suited to detect very divergent sequences. It also is able to handle large introns much more effectively, which was a major issue with our parasite.

Figmop expects the user to gather together a set of known protein sequences, and run the very excellent program MEME to find a set of robust sequence motifs. This is the first layer of sequence abstraction, as motifs generally do a much better job of identifying divergent protein sequences. Figmop then identifies all instances of these motifs in the genome file via MAST, and finally parses this sequence of motifs to find fuzzy matches (allowing insertions, deletions, and substitutions) to the specified pattern. This is the second layer of sequence abstraction, essentially resulting in a search for a fuzzy sequence of fuzzy sequences.

With a good set of motifs Figmop can exhibit very high sensitivity and specificity. Besides this, the pattern of motifs breaks up each of your genes into a number of separate blocks, which makes the MAST output very informative. As described later in this document, it makes introns quite obvious, highlights sequencing errors, and can even provide insight into the evolutionary history of your gene family.

Installation

Figmop works by searching for a pattern of sequence motifs in nucleotide or protein sequences. It was designed to work with the MEME suite of software, so as the user you need the program MEME to generate a set of motifs, and Figmop makes use of MAST as part of its running process. The pattern searching is carried out using the Python package patternHmm.

patternHmm

This Python package can be installed from https://github.com/dave-the-scientist/patternHmm. The easiest way is to use the 'Download ZIP' feature, extract it, and then install using the command (see the README for more details): `sudo python setup.py install`

MEME / MAST

Figmop was built using MEME versions 4.5.0 and 4.10.1, but barring major option changes it should work with other versions as well. Check the Installation Guide at http://meme-suite.org for prerequisite programs and directions to get the software installed on your system. Only the command-line version of the software is required.

NOTE: Ensure that you edit your path so that the commands `meme` and `mast` can be run from anywhere on your system.

Finding a motif pattern

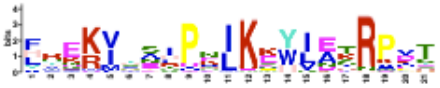
Generating a good motif pattern is essential if you hope to get good results. To do this you must select a positive set of protein sequences that give a good representation of their gene family of interest. Depending on the nature of your genes, the discovered motifs may be general enough to detect surprisingly divergent genes (a pattern generated from nematode cytochrome P450s was able to detect all of the relevant genes in human and drosophila). This part of the process must be done manually, and will likely require some experimentation on the part of the user when applying it to new data sets.

Running MEME

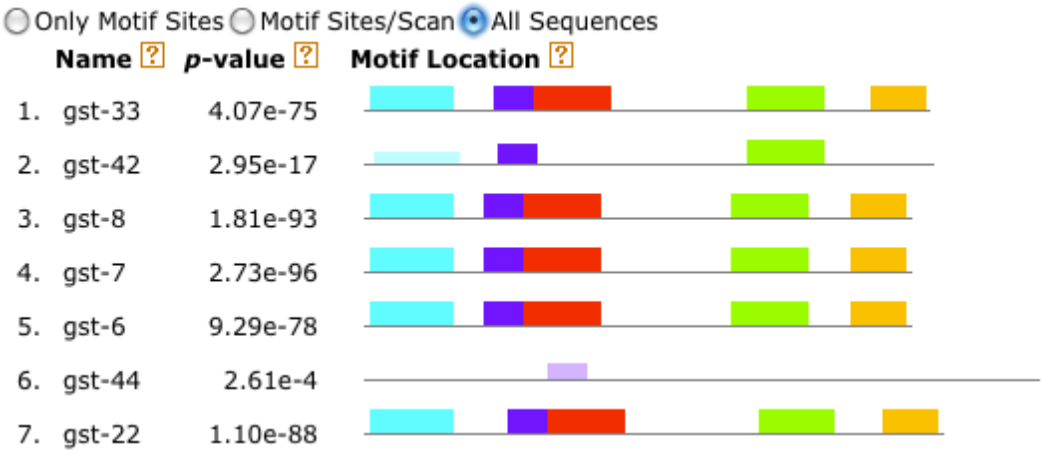
After extracting the compressed program, you will see a directory called `tutorial` alongside this document. Move into it, and you will see a file called `test_prots.fasta`. This file contains a positive set of 139 Gst (Glutathione-S-transferase) or Gst-like protein sequences from various nematodes. We are first going to look for up to 5 conserved motifs using the command:

```
meme test_prots.fasta -protein -nmotifs 5 -o meme_out_5
```

Note that you can use the option `-p 4` to use 4 processes (or however many CPU cores you desire) to speed up any of the calls to `meme`. After this completes (took ~3 minutes on my computer), you will see a folder called `meme_out_5`. Open the file `meme.html` that you find inside. The first things you will see are the sequence logos of the 5 motifs that MEME was able to find in these sequences, a measure of significance for each (how likely it is to have been found purely by chance), the number of times each motif was found, and its width. Each column in the sequence logo indicates which amino acids (coloured by similarity) are found at that position in the motif, with the height indicating the frequency. In motif 1, column 1 indicates that the first amino acid is either an isoleucine or a leucine with approximately equal prevalence. The giant Q in column 3 indicates that the third amino acid is always a glutamine.

	Logo	E-value	Sites	Width
1.		5.3e-1636	93	29
2.		6.4e-1736	94	32
3.		2.6e-1397	108	29
4.		1.0e-749	112	15
5.		3.5e-803	100	21

Next, scroll down to ‘Motif Locations’ and click ‘All Sequences’ to see how these motifs are distributed in your sequences. In this section of the results, each line represents one of the 139 sequences, the bars represent the motifs (mouse over to see which colour is which), with the height representing how well that motif matches at the given location.



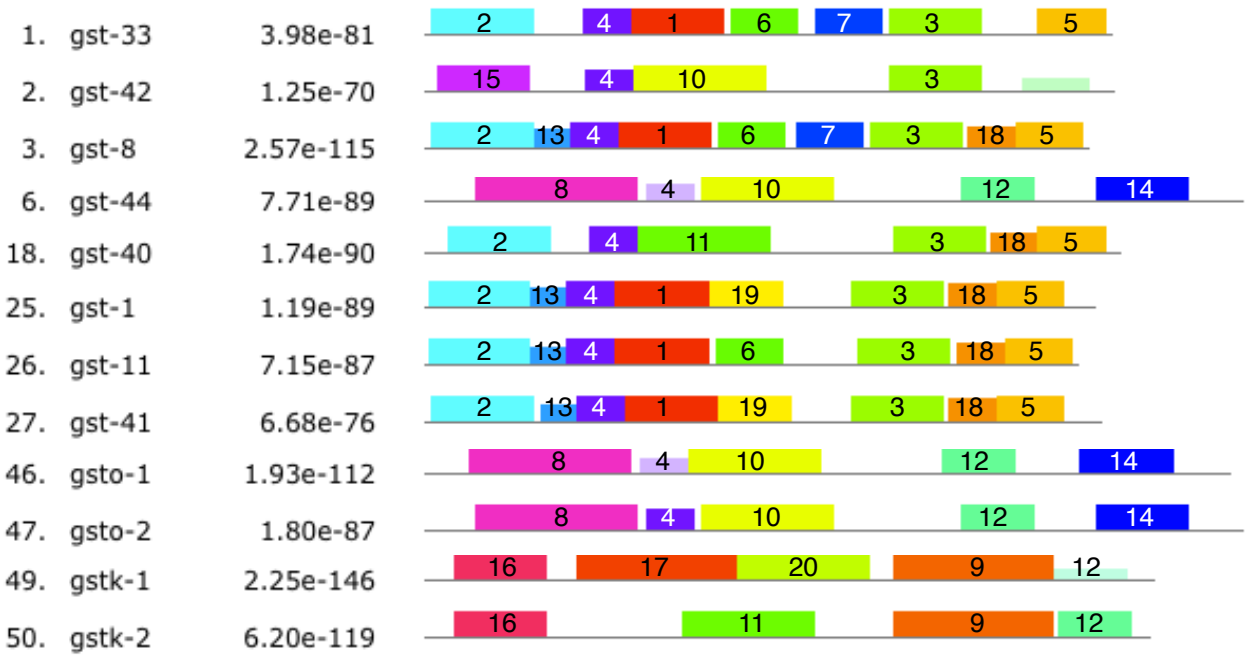
Just looking at these top few sequences, you can see a definite motif pattern emerging: 2, 4, 1, 3, 5. However Gst-42 is missing motif 1 (not a huge problem in itself), and Gst-44 only contains motif 4 (and very weakly at that). If you scroll down further you will see that this is also the case for all of the Gsto sequences, in that there is only 1 or 2 weak motifs present. Further, none of the Gstk proteins contain any of these 5 motifs. This means that Gsts appear to be different from the Gsto and Gstk sequences, and could mean that we need to remove them from the analysis. However experimental information tells us that they function in a very similar manner, so instead we will expand our motif search.

Refining the pattern with another MEME run

To try and get a more robust pattern, we will run meme again:

```
meme test_prots.fasta -protein -nmotifs 20 -o meme_out_20
```

This took ~7 minutes on my computer. Check the motif locations and you can see that the original pattern has been expanded to motifs: 2, 13, 4, 1, 6, 7, 3, 18, 5. You can also see several variation on this pattern, which may hint at the evolutionary history of these proteins. There are several proteins (Gst-1, 41, 23, 10, etc) that resemble the standard pattern but replace motifs 6 and 7 with motif 19. All of the known Gst-40 proteins have replaced motifs 1, 6, 7 with motif 11. Gst-42 no longer appears to be simply one motif away from the standard pattern, and instead now shares the pattern 15, 4, 10, 3 with all other Gst-42 and Gst-43 sequences. Gst-44 now has a distinct pattern shared with all of the Gsto proteins (8, 4, 10, 12, 14), which could imply that it has been classified incorrectly, and should actually be a Gsto protein instead. Gstk-1 (16, 17, 20, 9, 12) and Gstk-2 (16, 11, 9, 12) are also now present and contain their own patterns.



Before we begin describing these patterns to Figmop, let's examine motif 11. It is found with high confidence in the Gst-40 and Gstk-2 sequences, though they share nothing else. Find it in the Discovered Motifs section and click the down arrow to reveal more information. The sequence logo shows that the majority of the positions contain two amino acids with equal probability, while only a few have a dominant choice. This split distribution is somewhat suspicious, especially because the two amino acids rarely share properties (both being hydrophobic, positively charged, etc). In the list of sequences that define this motif you can see that all of the Gst-40 proteins share the top sequence in the logo, while all of the Gstk-2 proteins share the bottom one. It may be the case that these two sequences do share some similar biological function or evolutionary history, or it may be that the four shared residues have caused MEME to incorrectly group them together. Regardless, it is important to note that the presence of motif 11 in two proteins may not signify the same degree of homology as would the presence of some other motif.

	Name	Start	p-value	Sites
18.	gst-40	67	1.08e-48	LPFIEIDGGK TTLCQTVSICRYLAKSIQPDRWFGGATKTDSAKVDMMADGF ADLFQLAAMA
74.	Cbr-gst-40	67	2.17e-46	VPFIEIDGGK VTLCQTVSICRYLAKSVHPDRWLGGATKTDSAKVDMMADGF ADLFQLAAMA
123.	Cre-gst-40	67	1.55e-44	VPFIEIDGGK TTLCQTVSICRYLAKSVQPDKWFGGATKTDSAKVDMMADGF ADIYQLAMFG
55.	Cbn-gst-40	66	2.98e-43	LPFMEIDGGK ITLCQTVSICRYLGKSVHPERWLGGATKADSAKVDMMADGF ADLYQMAMIA
62.	Cbn32277	81	5.61e-40	LFSAQYWGIP FRLPKDYTNMMLTTSSIIPQRILVAAQLRDNALMEDVARTL WHRFYGYGKP
134.	Cre-gstk-2	81	9.78e-40	LFSAQYWGIP FRLPKDYTNMMLTTSSIIPQRILVAAQLRDNALMEDVARAL WHRFYAYGKP
61.	Cbn-gstk-2	81	1.09e-39	LFSAQYWGIP FRLPKNYTNMMLTTSSIIPQRILVAAQLRDNALMEDVARTL WHRFYGYGKP
92.	CBG05816	81	1.61e-38	LFSAQYWGIP FRLPKDYTNVMLNTSSIVPQRILVAAQLRDNALMEDVARAL WHRFYAYGKP
50.	gstk-2	81	3.01e-38	LFSAQYWGIP FRLPKDYTNMMLNTSSIVPQRILVASQLRDNVLMEDVARGL WHRFYAYGKP

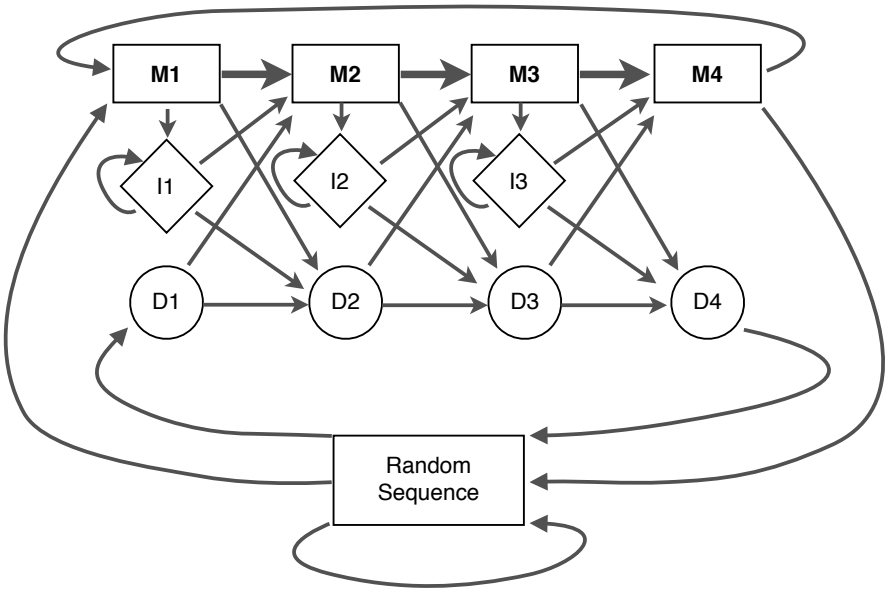
Pattern recognition with a pHMM

These observed motif patterns will allow us to use this set of motifs with Figmop to detect the *gst* genes in genomic sequences. However, as we've seen these patterns are not perfect; they can be slightly different between proteins, with some motifs absent or substituted with others. This problem becomes much harder when we look for these patterns in a genome because by chance introns are very likely to contain a few low-scoring motifs, which has the effect of inserting random motifs within the true pattern.

Figmop uses a profile Hidden Markov Model (pHMM) to detect regions containing the true pattern. This is a probabilistic model consisting of several different kinds of

connected states. The model can be in one state at a time; at each time step a state emits a symbol (or not, depending on the state), and then moves onto the next state.

A pHMM is shown below that is capable of producing many noisy variations of the pattern ‘ABCD’. The states labeled M1 - M4 are the match states, and they produce the symbols we are interested in; M1 emits ‘A’, M2 emits ‘B’, and so on. The states labeled I1 - I3 are the insert states that allow random noise to be present within our pattern; each can emit any letter and they allow the model to detect the true pattern within the sequence ‘ABCeD’. The states labeled D1 - D4 are the delete states, which allow some of the match states to be skipped; they do not emit anything, and allow the model to detect ‘ABD’ as containing a valid pattern. The ‘Random Sequence’ state accounts for all of the other sequence between valid pattern matches; it can emit any symbol and can be considered the background state.



Several given sequences are shown below, along with the path of states through the model that could be used to generate them. The state ‘rs’ indicates Random Sequence, and the true patterns are highlighted with boxes. Notice that there are two instances of the pattern in the last sequence.

```
Given:    q   e   s   [A   B   C   D]  f   e   e
States:   rs->rs->rs->M1->M2->M3->M4->rs->rs->rs

Given:    q   [A   z   o   B   C   a   D]  f   e
States:   rs->M1->I1->I1->M2->M3->I3->M4->rs->rs

Given:    d   e   [B   C   D   A   B   z   D]  f
States:   rs->rs  M2->M3->M4->M1->M2->I2  M4->rs
                ↳D1↑                    ↳D3↑
```

Though this sort of model is generative, the well-known Viterbi algorithm can be used to predict the most likely path of states that would generate a given sequence. Figmop identifies all instances of the motifs in a given genome using MAST, and then searches for instances of your pattern in the resulting sequences of motifs using the pHMM you define in the pattern file.

Preparing the pattern file

Generating the pattern file

After identifying a pattern of motifs using MEME, we need to describe this pattern to Figmop. The main Gst pattern we found was motif 2, 13, 4, 1, 6, 7, 3, 18, 5. This is nine motifs long, so we will generate a pattern file named GST_pattern.py using:

```
figmop --generate_model_file 9 --model_filepath GST_pattern.py
```

Open this using your favourite code editor, and you will first see three options at the top. We will set `min_matches` to 4, indicating that at least 4 of the 9 motifs must be present to qualify as a true pattern. We will keep `max_genome_region` at 40,000, which indicates the upper limit on the size of a genomic region containing a pattern, including all introns. Finally, set `meme_file` to the full path of the file describing the motifs that MEME found.

```
min_matches = 4
max_genome_region = 40000
meme_file = '/home/dave/Desktop/figmop/test/meme_out_20/meme.txt'
```

Defining the pHMM

The next object is a dictionary describing the emission probabilities of the 9 match states in our pHMM. This is where you describe the sequence of the pattern, as well as allow for ambiguities (described a little later). Fill out the dictionary using string values to identify the motif numbers:

```
matchEmissions = {
    'M1': {'2': 1.0},
    'M2': {'13': 1.0},
    'M3': {'4': 1.0},
    'M4': {'1': 1.0},
    'M5': {'6': 1.0},
    'M6': {'7': 1.0},
    'M7': {'3': 1.0},
    'M8': {'18': 1.0},
    'M9': {'5': 1.0},
    }
```

The next object is a dictionary representing the transition probabilities of our pHMM, which describes the probability of moving to the other states starting in a given state. Every edge in the model pHMM figure above can be modified here. While this

may look complex, pHMMs are quite robust with respect to their parameters and the default values are generally sufficient. We will leave them for now.

Downloading a test genome

Now that we have found a set of motifs and defined a pattern, we need a genome to run them on. We will use *Caenorhabditis brenneri*, a nematode related to *C. elegans*. Download the genome from ftp://ftp.wormbase.org/pub/wormbase/releases/WS249/species/c_brenneri/PRJNA20035/c_brenneri.PRJNA20035.WS249.genomic.fa.gz and extract it into the `tutorial` directory using a command like:

```
gzip -d c_brenneri.PRJNA20035.WS249.genomic.fa.gz
```

Running Figmop

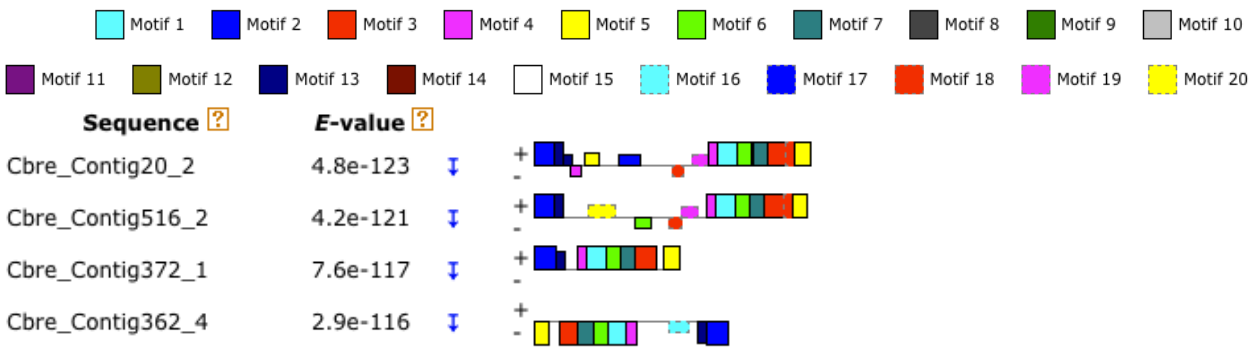
We now have everything set up to run Figmop, which we will do using the command:

```
figmop GST_pattern.py c_brenneri.PRJNA20035.WS249.genomic.fa
figmop_genome_mast/gst -o gst_run_1
```

The first option is the pattern file we just made, the second option is the sequence file of the genome, the third is the location that we want figmop to store some intermediate files, and finally `-o gst_run_1` indicates the name we are giving to the output. This took approximately 17 minutes on my computer.

Understanding the output

Once it's finished open up the `mast.html` file created in the `gst_run_1` directory. The first thing you will see is a similarity score between every pair of motifs, which will highlight in red any pair of motifs that are too similar. None of our motifs should cross this threshold. Scroll down to the Search Results, and here you will see every predicted pattern found in the *C. brenneri* genome. You will likely notice that MAST is using different colours than MEME to differentiate each motif, though at least it provides a legend at the top.



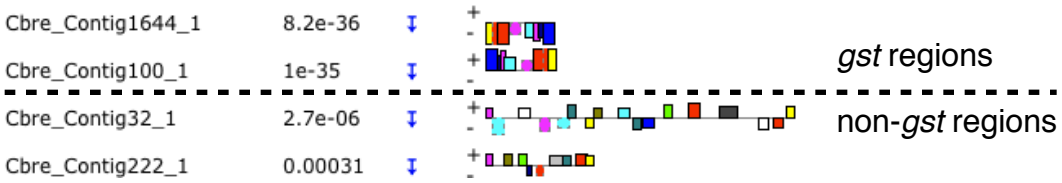
The hits here are displayed in order of their E-value, though it is important to note that this value is calculated by MAST, and relates only to the strength of the motifs found and the size of the region; MAST doesn't use our pattern and cares nothing for the order of the motifs. That being said, in practice it tends to work well for our purposes. The name of the first hit indicates it was found on the sequence named Cbre_Contig20 from the given genome file, and that it was the second region identified on this sequence. The order relates to the position in the sequence, so the first hit could be very poor and may not show up in the results. However in this case it is just found lower down at an E-value of 1.5e-92.

If you remember the pattern we found using the protein sequences, it was essentially continuous from start to end. However the first two hits here both have a large gap between motif 13 and motif 4, where several weak motifs can be found. While it's possible that this is true coding sequence, it is almost always evidence of an intron. With 20 motifs, a stretch of random sequence is going to look similar to some of them just by chance. You will also notice that some of those motifs are upside-down, as is the entire pattern in the 4th hit. This is because we are scanning DNA sequence, and so motifs may also appear in the inverse complement. This indicates that there is likely a gene on the negative strand of the sequence in Cbre_Contig362. If you scroll down to hit Cbre_Contig362_3, you will see that it appears to contain 2 full patterns separated by a short distance. This is perfectly normal; if two patterns are found very close together, the regions may be merged into one.

Perhaps the most useful feature of the MAST output is the ability to evaluate the identified regions at a glance. If you scroll down to hit Cbre_Contig126_2, you will notice that it is different from the other hits as it contains motif 11 in the middle of the sequence. If you recall from our MEME analysis, this was the sub-pattern seen with the Gst-40 sequences. So even though our pattern did not include this as an option, Figmop was able to find it. This is also useful with a hit like Cbre_Contig165_4. This region is exceptionally long, with a ~8kb stretch containing no motifs at all. This is almost certainly a long region of Ns in the genome file, as it is very unlikely there would be no weak matches in a sequence of that length. You can also see why this was included, as there is the first half of the Gst pattern at the far right, followed by a weak match to motif 18 at the far left. Figmop disregards the strength of a motif match and the distance between motifs (unless it exceeds the maximum specified in the pattern file), and as this spurious motif fits the pattern the entire region was included as a match. The conclusion from this sort of a hit is that there is likely a true *gst* at this location in the genome, but the sequencing effort was not able to recover the second half of the locus.

Filtering the sequences

The last thing we have to do is decide on a cut-off for these patterns, as the bottom half of the page is filled with hits that clearly do not fit the Gst pattern. There is no well-defined or systematic way to do this, but in practice it is usually quite obvious. The E-value jumps by over 1e-19 between sequences Cbre_Contig100_1 and Cbre_Contig32_1, and you can see that there are no compact instances of the pattern or even any high-scoring motifs after this point.



One of the other outputs from Figmop is a file called `hits.fa` found in the same location as the html file. This contains the sequence for each of the hits, listed in the same order as the `mast.html` file. This lets us easily keep only the hits we are interested in.

Simply search the file for the sequence named Cbre_Contig32_1, and delete it and every sequence below it. Next search for sequence Cbre_Contig165_4. As we suspected by looking at the motifs the majority of this sequence is Ns, which we should delete because it is useless to us. But which end of the sequence do we keep? From the MAST output we know that this pattern was found on the negative strand (it was upside down in the html file), and this is confirmed in the sequence name (the start position is later than the end position). When extracting the sequence for a hit on the negative strand Figmop automatically converts it to its inverse complement, so the Gst pattern is found in the first part of the sequence before the Ns. This means we should delete all of the sequence from the first N to the end.

Refining the pattern

Though this basic pattern may be sufficient for finding most of the *gst* genes in a genome, we can improve it by including the information about several of the sub-patterns that we have seen into the pattern file. If you recall from the MEME file the main pattern was 2, 13, 4, 1, 6, 7, 3, 18, 5; but several of the sequences had motif 19 instead of 6 and 7. We can add this information by modifying the emission probabilities of match state 5, allowing it emit both motif 6 and motif 19. We will set this probability to 0.8 instead of 1.0, as it occurs less frequently than motif 6. Now the model can generate this sub-pattern via the path: M1->M2->M3->M4->M5->**D6**->M7->M8->M9. As this alternate path now proceeds through a delete state, we can improve its likelihood by increasing the weight of the edge from M5 to D6 in the transition probabilities dictionary. We will increase it from 0.1 to 0.2, and to ensure the M5 entry sums to 1.0 we must reduce the M5->I5 and M5->M6 edges to 0.25 and 0.55, respectively.

In the same way we can also add in the sub-pattern for Gst-40, which replaces motifs 1, 6, 7 with motif 11. We will allow match state 4 to emit both motif 1 and motif 11 (with probability 0.5 this time), and increase the transition probability of edge M4->D5. To add the sub-pattern for Gst-42 and Gst-43, we will allow match state 1 to emit motif 15, allow match state 4 to also emit motif 10, and increase the transition probabilities for edges M1->D2 and M4->D5 (which we just did). There are also a few smaller adjustments we could make, as it is fairly common for a protein to be missing motifs 13, 7, or 18. To accommodate this we can increase the transition probabilities for edges M1->D2 (which we already have), M5->D6 (which we already have), and M7->D8. The pattern file after all of these additions has been included as `GST_pattern_refined.py`.

Running Figmop again

Run Figmop again using the command below:

```
figmop GST_pattern_refined.py
c_brenneri.PRJNA20035.WS249.genomic.fa figmop_genome_mast/gst -o
gst_run_2
```

You will notice that this completes much faster than our last run (~1.5 minutes on my computer), but it has nothing to do with our edited pattern file. Running MAST on the full genome is by far the slowest step in Figmop, and so it stores these results as the intermediate files (`figmop_genome_mast/gst` in this example). If the specified location is empty then Figmop will start the slow step, but it will use existing MAST output if it is present. You can change the pattern file as much as you like and still use the same intermediate files, but you will have to recreate them if you change the genome file or use a set of motifs that are not defined in your current MEME file.

Open the `mast.html` file you just created. The first 32 regions look much the same with the exception of Cbre_Contig392_1. This region has become much larger, including 5 motifs past the end of the obvious pattern. In fact these low-scoring motifs form a good approximation of the Gst pattern, and Figmop has decided that it represents another match. Though it is often be helpful to refine the pattern as we have above, the danger in relaxing alternate paths through the model is that you may find more false-positive regions. The final 17 regions are also the same as those in our first run, except for the new hits Cbre_Contig302_1 and Cbre_Contig882_1. We postulated that Cbre_Contig126_2 may be a *gst-40* gene, so these two new hits may be related to it. On the other hand they both contain a weak resemblance to motif 9 (which is why they weren't found in the first run), so they may not be *gst* genes at all (there were no examples of sequences beginning this way in our defining set).

So in conclusion Figmop was used to find 49 putative standard *gst* genes (though we can only recover the sequence for the front half of one), 1 putative *gst-40* gene, and 2 others that may or may not be related to *gst-40*, in the *C. brenneri* genome assembly (version WS249).

Finding other genes

A good second exercise would be to generate a pattern file and identify the putative *gsto* genes in this genome. I will leave the details to the user, but an example pattern file can be found as `GSTO_pattern.py`. Remember that when you run it you can use the same intermediate files as in the previous runs, so it should complete quite quickly. You should be able to conclude that there are three *gsto* genes in this version of the *C. brenneri* assembly, Cbre_Contig5_1, Cbre_Contig35_1, and Cbre_Contig7_4.

Searching protein sequences with these motifs

Figmop can also be used for the much simpler task of classifying unknown protein sequences using our set of motifs. It would be possible to define the various patterns to Figmop and check each sequence, but in practice this was found to be unnecessary. Protein sequences are short enough to be human-readable and do not possess any of the extra complexity introduced by nucleotide sequences, so the presence of strong motifs is sufficient as a search criterion. When run in this way Figmop simply runs MAST on the given sequences, and then extracts the hits into a new file with the same order as the MAST results.

We will again use *C. brenneri* as our test case. Download the protein sequences from ftp://ftp.wormbase.org/pub/wormbase/releases/WS249/species/c_brenneri/PRJNA20035/c_brenneri.PRJNA20035.WS249.protein.fa.gz and extract it into the `tutorial` directory, and then run Figmop using the commands:

```
gzip -d c_brenneri.PRJNA20035.WS249.protein.fa.gz
```

```
figmop meme_out_20/meme.txt --prot
c_brenneri.PRJNA20035.WS249.protein.fa -o gst_prot
```

Once finished (~15 seconds) open the `mast.html` results file. Recall the Gsto pattern of 8, 4, 10, 12, 14, and that above we identified three genomic regions containing putative *gsto* genes. You can see all three present here as hits CBN12846, CBN31196, and CBN12387. If you scroll down near the bottom to hit CBN28166, you can see that it is very short, but contains the first two Gst motifs with high confidence. You may also notice that hit CBN28100 appears to be missing the first two motifs of the Gst pattern. This might be an example of a fragmented gene model, where the software mistakenly identified two genes where in truth there is only one. This idea is strengthened if you check the genomic locations of these sequences at http://www.wormbase.org/species/c_brenneri/gene/WBGene00190131, and scroll down to the ‘Sequence features’ section where you can see that the two genes are adjacent and on the same strand.