Invoking the gibbs.py program:

./gibbs.py chip.dat 1000 10 5 output\_dir

here chip.dat are the shape-profile under positive sequences,

Description of the outputs from gibbs.py (the train.instance files):

A train.instance file contains two types of lines: (i) header lines (i.e., those that start with #) and (ii) those that contain shape-data

A line starting with #, for example, #1,10, is a header line for a Gibbs alignment. The first number (i.e., 1) means that this is the first attempt to find an alignment. The second number (i.e., 10) means that we are trying to find an alignment of length 10. Thus, if we list all the lines that start with #, we will see the first number is between 1 and the fourth argument to gibbs.py; and the second number is between the third argument to gibbs.py, say *L*, and *L+4* (since we try to find alignments of length *L, L+1, … , L+4*). So, if Gibbs.py was run with the following command:

./gibbs.py chip.dat 1000 10 5 output\_dir,

After each header line, we should see *N* number of lines where the *i*-th line is a window from the *i*-th ChIP peak in the input data file. For example, for the above command, we should see 1000 lines after every header line. If the header line is #5,11, then it means that the windows will be of length 11.

Description of the second module, motif\_detector.py:

The second module takes the following inputs:

1. Positive training shape-data (corresponding to positive training ChIP peaks)
2. Negative training shape-data (corresponding to negative training ChIP peaks)
3. Positive validation shape-date (corresponding to positive validation ChIP peaks)
4. Negative validation shape-data (corresponding to negative validation ChIP peaks)
5. Output prefix (a prefix for the various files to be generated by motif\_detector.py)
6. Name of the feature (HelT, MGW, ProT, Roll, etc.)
7. The minimum threshold for negative log transformed hypergeometric p-value (motifs below this threshold will not be considered)
8. The train.instance file(s) that are output from gibbs.py, the training and validation datasets (shape-date; both positive and negative), and an output directory.

In the specified output directory, motif\_detector.py will output:

1. A summary file giving the p-values, F1/3 scores, TPRs, and FPRs all motifs
2. The motifs (specified as ranges per position)
3. Shape-data of the occurrences of the motifs in training and validation positive data
4. bed coordinates of the occurrences of the motifs in training and validation positive data
5. Distances of the occurrences of the motifs from the 5’ end of the ChIP peak
6. Shape-data +/- 50 bps flanking the occurrences of the motifs in training and validation positive data
7. bed coordinates of +/- 50 bps flanking the occurrences of the motifs in training and validation positive data

An example:

Let us assume we are working to find the HelT motif for the protein JUNB. The training and validation positive and negative shape-data are in a directory called junb-data and the four files are named train.chip.dat, train.ctrl.dat, test.chip.dat, and test.ctrl.dat. Typically, we run gibbs.py to find motifs of several lengths. So we would create a directory junb-analysis and several subdirectories within junb-analysis where gibbs.py will output the train.instance files. To find motifs of length between 5 and 30, would then first run the following commands:

1. ./gibbs.py junb-data/train.chip.dat 1000 5 5 junb-analysis/5
2. ./gibbs.py junb-data/train.chip.dat 1000 10 5 junb-analysis/10
3. ./gibbs.py junb-data/train.chip.dat 1000 15 5 junb-analysis/15
4. ./gibbs.py junb-data/train.chip.dat 1000 20 5 junb-analysis/20
5. ./gibbs.py junb-data/train.chip.dat 1000 25 5 junb-analysis/25

Once the above commands are done, we then run:

./motif\_detector.py junb-data/train.chip.dat junb-data/train.ctrl.dat junb-data/test.chip.dat junb-data/test.ctrl.dat junb-analysis/results HelT 8 junb-analysis/\*/train.instance

This command will find shape-motifs by optimizing F1/3 scores and computing p-values. It will only consider those motifs that have a hypergeometric p-value < 10^-8 on the validation set, and output the non-redundant motifs (please check our manuscript). The first output file will be junb-analysis/results.p\_val\_len\_summary. For each motif, there will be a separate line and the motifs are sorted according to their FPR (false positive rate) in the validation data. The *i*-th line provides the following information for the *i*-th motif (in this order; we later use awk in various downstream programs to parse this file):

1. Hypergeometric p-value on the training data
2. Hypergeometric p-value on the validation data
3. F1/3 score on the training data
4. F1/3 score on the validation data
5. Motif length
6. The number of standard deviations that this motif extends from its mean shape-profile
7. TPR on training data
8. FPR on training data
9. TPR on validation data
10. FPR on validation data

The various information related to the *i*-th motif (the motif itself, bed coordinates of its occurrences, etc.) are output in the files that have the prefix junb-analysis/results\_$i

For example, the first motif will be given in a file named junb-analysis/results\_1.motif\_as\_range and it will give the ranges of shape-data per position of the shape-motif. If the shape-motif is of length 5, then this file can show:

32.276496 35.899984

34.262414 36.361846

31.002831 33.501329

34.135928 35.608732

34.290139 35.927121

Here the *i*-th line shows the range of HelT values that the shape-motif allows at the *i*-th position of the binding site.

Generating sequence-logos from sequences underlying shape-motifs:

For the above example, we will have .bed files that gives us bed coordinates of shape-motif occurrences in the training and validation positive sets. For example, the file junb-analysis/results\_1.bed will give this information for the first motif. We can use bedtools to get the fasta sequences from this bed file. Assuming that the fasta file is junb-analysis/results\_1.fasta, we will then use our other program find\_consensus\_seqs.py and the program weblogo (to be downloaded from here: <http://weblogo.threeplusone.com/manual.html#download>) as follows to compute the sequence logo:

./find\_consensus\_seqs.py junb-analysis/results\_1.fasta junb-analysis/results\_1.opt\_align junb-analysis/results\_1.H

and then:

weblogo --format png\_print --composition 'none' --revcomp --color-scheme 'classic' --fineprint '' < junb-analysis/results\_1.opt\_align > junb-analysis/results\_1\_fasta\_rev.png

weblogo --format png\_print --composition 'none' --color-scheme 'classic' --fineprint '' < junb-analysis/results\_1.opt\_align > junb-analysis/results\_1\_fasta.png

Notes:

1. The above weblogo commands will generate the sequence logo in both forward and reverse-complement orientation.
2. There is an empty string (‘’) being passed as the argument paired with --fineprint. Please check weblogo --help for the details.
3. In the program find\_consensus\_seqs.py, we need to specify a genomic background. Currently we are using {'A': 0.295, 'C': 0.205, 'G': 0.205, 'T': 0.295} that reflects the GC-content of the human genome. You might want to change this to match the GC-content of your organism at hand.