User Manual of ChIP-GSM (R)

1. Introduction

We develop a novel ChIP-seq based Gibbs Sampling approach for cis-regulatory Module inference (ChIP-GSM). ChIP-GSM iteratively samples read tags of multiple TFs and their possible combinations under a Gibbs sampling framework. If read counts are from bindings at gene promoter regions, ChIP-GSM will identify promoter-focused TF modules; or if from enhancer regions, ChIP-GSM will provide a list of modules functional at enhancers. In our study, we do find difference between TF combinations at promoter and enhancer regions. For a general use, if there is no region preference, all binding signals from the whole genome can be jointly studied by ChIP-GSM.

The regulatory effect of TF modules (instead of individual TFs) was directly modeled by ChIP-GSM. Specifically, we use a weight-based read tag tossing approach to identify binding events for multiple TFs simultaneously, where any binding event with a relatively low read count can be well captured by assigning a weight much higher than that of a background region. Such 'weak' bindings will help identify TF associations across multiple regions and further highlight their cooperative action in a cell type-specific manner.

2. Programming environment

ChIP-GSM is written in R language and has been tested under R 3.2.2 and Ubuntu 12.04 64 bit system. HOMER package should be also properly installed to pre-process SAM-formatted ChIP-seq data.

3. Work flow of ChIP-GSM

A workflow of ChIP-GSM is shown in **Fig. 1**. We use the HOMER (http://homer.ucsd.edu/homer/) to generate raw read counts from ChIP-seq data at candidate regions (at promoters, enhancers or the whole genome) and provide an R script to achieve major functions of ChIP-GSM. In this manual, we provide a demo case for module reference from gene promoter regions of K562 cells.

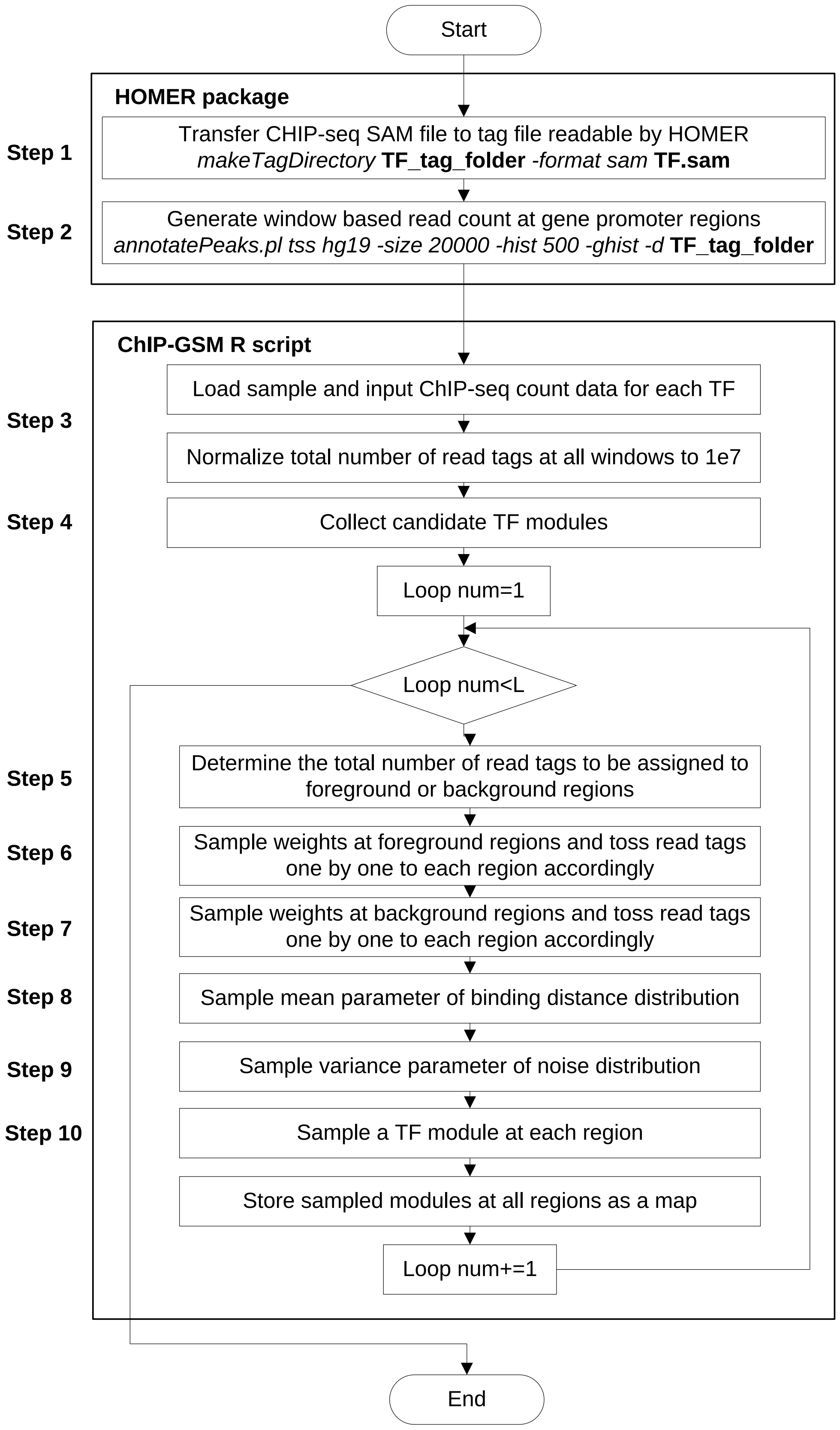

Start
HOMER package
Step 1
Transfer CHIP-seq SAM file to tag file readable by HOMER
makeTagDirectory TF_tag_folder -format sam TF.sam
Step 2
Generate window based read count at gene promoter regions
annotatePeaks.pl tss hg19 -size 20000 -hist 500 -ghist -d TF_tag_folder
ChIP-GSM R script
Step 3
Load sample and input ChIP-seq count data for each TF
Normalize total number of read tags at all windows to 1e7
Step 4
Collect candidate TF modules
Loop num=1
Loop num<L
Step 5
Determine the total number of read tags to be assigned to foreground or background regions
Step 6
Sample weights at foreground regions and toss read tags one by one to each region accordingly
Step 7
Sample weights at background regions and toss read tags one by one to each region accordingly
Step 8
Sample mean parameter of binding distance distribution
Step 9
Sample variance parameter of noise distribution
Step 10
Sample a TF module at each region
Store sampled modules at all regions as a map
Loop num+=1
End

Fig. 1. Workflow of ChIP-GSM.

3.1 ChIP-seq data pre-processing

Step 1: We use HOMER function `makeTagDirectory` to transfer any SAM-formatted ChIP-seq profile into tag files, where 5' start locations as well as directions of read tags in each chromosome are stored in an individual file. Input data is used in ChIP-GSM algorithm so both `TF.sam` and `input.sam` are needed.

```
makeTagDirectory TF_tag_folder -format sam TF.sam
```

Step 2: We use HOMER function `annotatePeaks.pl` to count read tags falling each binned genomic segment (`-hist 500`). HOMER can directly partition gene promoter regions given the human reference genome using an option `tss hg19`. In the demo, gene promoter regions are defined as ±10k bps domain around transcription starting sites (`-size 20000`).

```
annotatePeaks.pl tss hg19 -size 20000 -hist 500 -ghist -d TF_tag_folder
```

If users want to study TF modulization at enhancers or across the whole genome, a list of candidate regions in BED format should be provided to HOMER (i.e., `candidate_region.bed`). Using the following command, we can extend each region to ±1k around its midpoint (`-size 2000`) and count read tags in each bin. If the option is not specified, the original length of each region will be used.

```
annotatePeaks.pl region.bed hg19 -size 2000 -hist 500 -ghist -d TF_tag_folder
```

More details about HOMER functions used in Steps 1 and 2 can be found from the HOMER package (http://homer.ucsd.edu/homer/).

Step 3: We load all read tags output from HOMER into the workspace of R and used a R script `ChIP_GSM_main.R` to perform the ChIP-GSM functions. We first normalize the total number of read tags in each ChIP-seq profile to 10 million, for either sample or input ChIP-seq data. This step is important to ensure that each TF is equally treated.

```
multi_TFs<-loading_multi_TF_read_counts(TF_path, hg19_RefSeq)
```

Step 4: ChIP-GSM will automatically search a list of candidate modules. If needed, a list of candidate modules based on prior biology knowledge can also be loaded from external recourses.

```
Sampling_input_data_frame<-Candidate_cluster_searching(multi_TFs, Num_gene,
segment_num, Num_TF)
```

3.2 TF module inference using ChIP-GSM

Step 5: Based on an initial assignment of TF modules to each region or from previous round of sampling, we can get the status of each region as binding (a module-bound region) or non-binding (a background region). Then, the proportion of reads to be respectively assigned to bound or background regions will be estimated. After that, we will process module-bound and background regions separately since we assume their read counts follow distinct distributions.

```
Read_count_total<-sampling_TX_TI_cluster(Sampling_input_data_frame, Fold_change_facto
r, alpha_I, beta_I, Binding_pattern, Num_TFs)
```

Step 6: For each TF, we sample a weight for each of its bound regions according to a Power-Law distribution. Here, the parameters of Power-Law distribution are TF specific and can be fitted to read count distribution in each TF ChIP-seq profile.

```
est_powerlaw<- power.law.fit(Sampling_input_data_frame$Sampling_window_Sample_tags[,
t], impelementation = "plfit")
```

Then, we toss read tags one by one to TF-bound regions (determined in **Step 5)** according to their weights.

```
X <- sampling_X_cluster_distance(Sampling_input_data_frame, Binding_pattern_old, TX,
Xmin, Nsigma, Xupper, Pdfx_PowerLaw, Binding_prior, Num_windows, Num_TFs)
```

Step 7: For background regions, we sample another weight for each according to a Gamma distribution. Here, the parameters of Gamma distribution are obtained through fitting the read count distribution in each input ChIP-seq profile.

```
mean_Gamma<-mean(Sampling_input_data_frame$Sampling_window_Input_tags[,t])
variance_Gamma<-var(Sampling_input_data_frame$Sampling_window_Input_tags[,t])
alpha_I[t]=max(c(mean_Gamma^2/variance_Gamma, 1))
beta_I[t]=min(c(variance_Gamma/mean_Gamma+1, 8))
```

We toss the remaining read tags one by one in current TF ChIP-seq profile to background regions according to their weights.

```
I <- sampling_I_cluster_distance(Sampling_input_data_frame, Binding_pattern_old, TI, N
sigma, Xupper, Pdfi_Gamma, Background_prior, Num_windows, Num_TFs)
```

Step 8 (optional for promoter study): We will need to sample a mean parameter of Exponential distribution on binding locations of each TF referring to the nearest TSS and calculate a probability as a prior for each TF-region binding event.

```
Binding_prior<-sampling_lambda_distance(Sampling_input_data_frame, Binding_pattern, al
pha_lambda, beta_lambda, Num_windows, Num_TFs, window_size, promoter_size)
```

Step 9: We sample a variance parameter of the residues between observed read counts and tossed read counts across all regions for all TFs.

```
Nsigma <- sampling_Nsigma_cluster(Sampling_input_data_frame, X, I, alpha_N, beta_N,
Num_windows, Num_TFs)
```

Step 10: For each region, we calculate a posterior probability on each candidate module. And according to the posterior probability distribution across all candidate modules, we probabilistically sample a module for current region.

```
Sampling_temp_results<-sampling_binding_cluster_distance(Sampling_input_data_frame,
Binding_pattern_old, X, Xmin, Xupper, I, Nsigma, Pdfx_PowerLaw, Pdfi_Gamma,
Binding_prior, Background_prior, Num_windows, Num_TFs, Num_clusters)
```

After this step, the state of each region is updated as either a bound region for any TFs in the sampled module or a background region for TFs not in the selected module. States of all regions will be brought back to **Step 5** to start a new round of sampling.

```
Binding_pattern_old=Sampling_temp_results$Binding_pattern
```

3.3 Results output

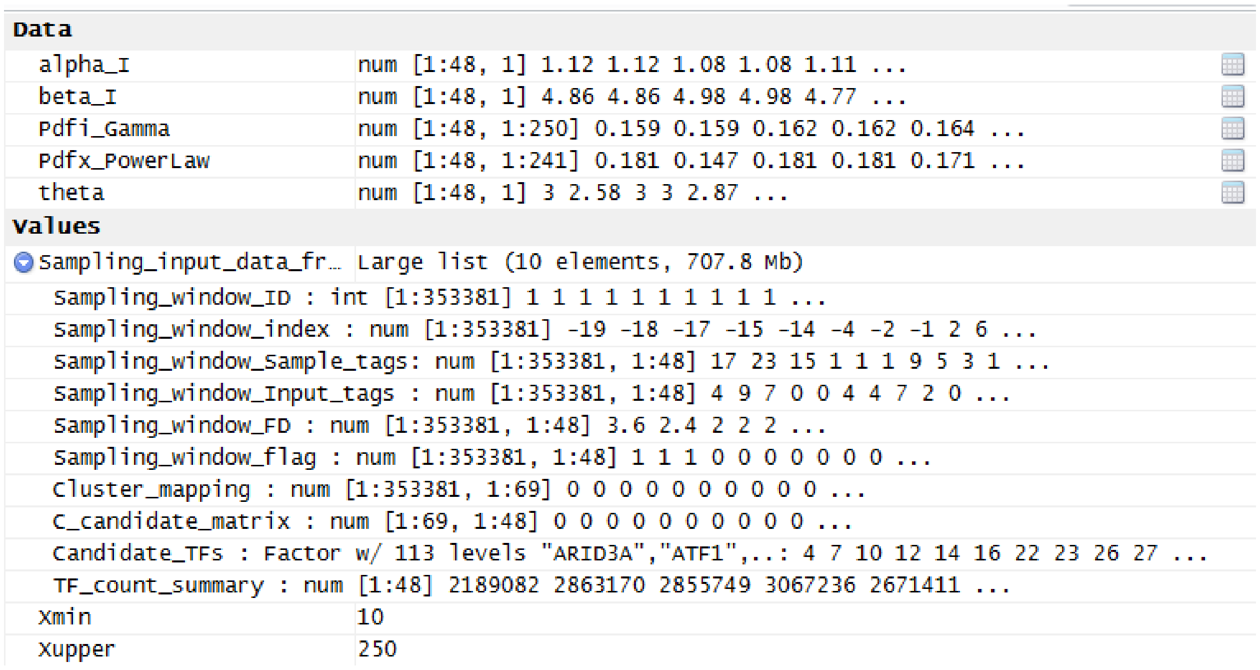
After enough rounds of sampling, the sampling frequency for module-region is proportional to the posterior probability of regulation.

```
Summition_binding=Summition_binding+Sampling_temp_results$Binding_pattern
Summition_cluster=Summition_cluster+Sampling_temp_results$Cluster_pattern
```

4. Demo case running

We include a demo in our package to guide users through the running process of ChIP-GSM, using ChIP-seq data of K562 cells, also a major case study in the ChIP-GSM paper. After applying the first four steps to all 114 TFs, we stored processed data in a R data file as `K562_R_demo.Rdata.`

```
load('K562_R_demo.Rdata')
```



```
Data
  alpha_I              num [1:48, 1] 1.12 1.12 1.08 1.08 1.11 ...
  beta_I               num [1:48, 1] 4.86 4.86 4.98 4.98 4.77 ...
  Pdfi_Gamma           num [1:48, 1:250] 0.159 0.159 0.162 0.162 0.164 ...
  Pdfx_PowerLaw        num [1:48, 1:241] 0.181 0.147 0.181 0.181 0.171 ...
  theta                num [1:48, 1] 3 2.58 3 3 2.87 ...
Values
  Sampling_input_data_fr…  Large list (10 elements, 707.8 Mb)
    Sampling_window_ID : int [1:353381] 1 1 1 1 1 1 1 1 1 1 ...
    Sampling_window_index : num [1:353381] -19 -18 -17 -15 -14 -4 -2 -1 2 6 ...
    Sampling_window_Sample_tags: num [1:353381, 1:48] 17 23 15 1 1 1 9 5 3 1 ...
    Sampling_window_Input_tags : num [1:353381, 1:48] 4 9 7 0 0 4 4 7 2 0 ...
    Sampling_window_FD : num [1:353381, 1:48] 3.6 2.4 2 2 2 ...
    Sampling_window_flag : num [1:353381, 1:48] 1 1 1 0 0 0 0 0 0 0 ...
    Cluster_mapping : num [1:353381, 1:69] 0 0 0 0 0 0 0 0 0 0 ...
    C_candidate_matrix : num [1:69, 1:48] 0 0 0 0 0 0 0 0 0 0 ...
    Candidate_TFs : Factor w/ 113 levels "ARID3A","ATF1",..: 4 7 10 12 14 16 22 23 26 27 ...
    TF_count_summary : num [1:48] 2189082 2863170 2855749 3067236 2671411 ...
  xmin                 10
  xupper               250
```

Fig. 2. K562 cells ChIP-seq data for TF module inference at gene promoters.

In total, we generated 353,381 candidate regions (500bps long, partitioned from gene promoter regions); 69 candidate TF modules (68 modules with at least two TFs in each and one all-zero module for background region modeling), covering 48 TFs with fitted parameters of Power-Law (`xmin=10` and `theta`) or Gamma distributions (`alpha_I` and `beta_I`). We limit the maximum read count at each region to 250 (`xupper=250`).

We set the initial state of each candidate region as follows:

```
for (k in 1:Num_windows){
   candidate_cluster=which(Sampling_input_data_frame$Cluster_mapping[k,]>0)
   rand_index=sample(length(candidate_cluster), 1)
   Binding_pattern[k,]=Sampling_input_data_frame$C_candidate_matrix[candidate_cluster
[rand_index[1]],]
   Cluster_pattern[k,candidate_cluster[rand_index[1]]]=1
}
```

```
for (t in 1:Num_TFs){
   binding_index=which(Binding_pattern[,t]>0)
   Background_sampling_weight = rgamma(Num_windows, shape=alpha_I[t], scale=beta_I[t])
   Forground_sampling_weight = Background_sampling_weight
   Forground_sampling_weight[binding_index] = Fold_change_factor*Forground_sampling_we
ight[binding_index]
   Forground_sampling_weight = Forground_sampling_weight/(sum(Forground_sampling_weigh
t))
   TX[t]=floor(sum(Forground_sampling_weight[binding_index])*Sampling_input_data_frame
$TF_count_summary[t])
   TI[t]=Sampling_input_data_frame$TF_count_summary[t]-TX[t]
 }


alpha_N=2
beta_N=5
Nsigma=1/rgamma(1, shape=alpha_N, scale=1/beta_N)

Summition_binding=Binding_pattern
Summition_cluster=Cluster_pattern
Binding_pattern_old=Binding_pattern


window_size=500
promoter_size=10000
Background_prior=matrix(1, nrow=nrow(Binding_pattern), ncol=ncol(Binding_pattern))
alpha_lambda=1
beta_lambda=1
Binding_prior<-sampling_lambda_distance(Sampling_input_data_frame, Binding_pattern,
alpha_lambda, beta_lambda, Num_windows, Num_TFs, window_size, promoter_size)
```

In one round of sampling, **Steps 5** to **10** will be carried out according to the order. Variables `TX` (total number of read tags to be assigned to foreground regions of each TF), `TI` (total number of read tags to be assigned to background regions of each TF), `X` (read count at each foreground region of each TF), `I` (read count at each background region of each TF), `Binding_prior` (binding prior of region location to TSS), `Nsigma` (residue variance), `Binding_pattern` (TF-region regulatory map) and `Cluster_pattern` (Module-region regulatory map) will be sampled sequentially.

```
X <- sampling_X_cluster_distance(Sampling_input_data_frame, Binding_pattern_old, TX,
Xmin, Nsigma, Xupper, Pdfx_PowerLaw, Binding_prior, Num_windows, Num_TFs)


I <- sampling_I_cluster_distance(Sampling_input_data_frame, Binding_pattern_old, TI,
Nsigma, Xupper, Pdfi_Gamma, Background_prior, Num_windows, Num_TFs)
```

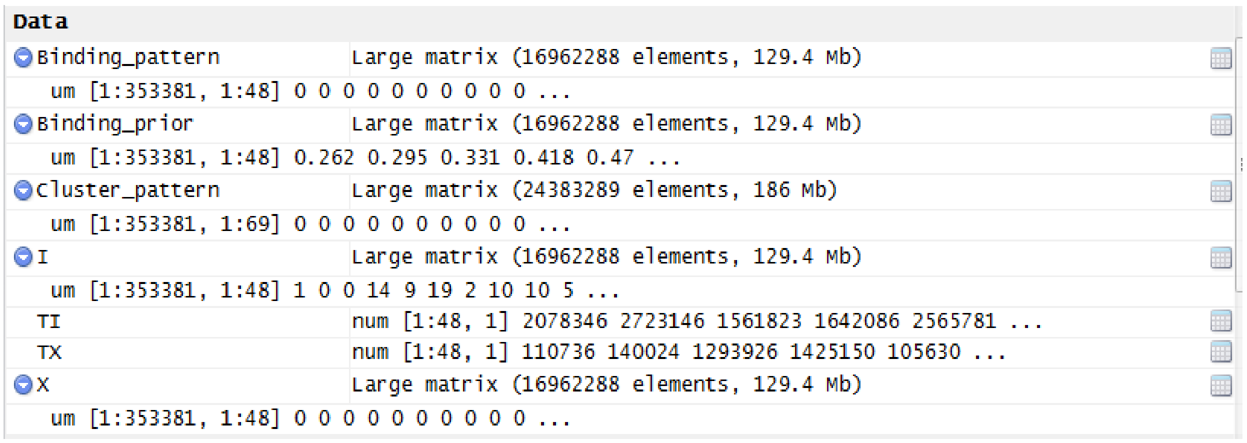
```
Nsigma <- sampling_Nsigma_cluster(Sampling_input_data_frame, X, I, alpha_N, beta_N,
Num_windows, Num_TFs)


Sampling_temp_results <- sampling_binding_cluster_distance(Sampling_input_data_frame,
Binding_pattern_old, X, Xmin, Xupper, I, Nsigma, Pdfx_PowerLaw, Pdfi_Gamma,
Binding_prior, Background_prior, Num_windows, Num_TFs, Num_clusters)


Binding_prior<-sampling_lambda_distance(Sampling_input_data_frame, Sampling_temp_resul
ts$Binding_pattern, alpha_lambda, beta_lambda, Num_windows, Num_TFs, window_size, prom
oter_size)


Read_count_total<-sampling_TX_TI_cluster(Sampling_input_data_frame, Fold_change_facto
r, alpha_I, beta_I, Sampling_temp_results$Binding_pattern, Num_TFs)

TX<-Read_count_total$TX
TI<-Read_count_total$TI
```

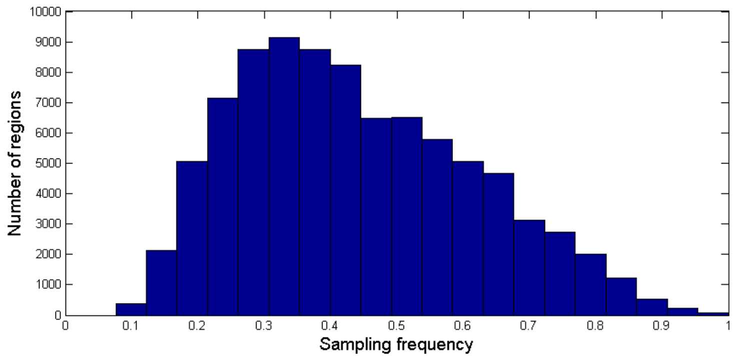


```
Data
Binding_pattern          Large matrix (16962288 elements, 129.4 Mb)
   um [1:353381, 1:48] 0 0 0 0 0 0 0 0 0 0 ...
Binding_prior            Large matrix (16962288 elements, 129.4 Mb)
   um [1:353381, 1:48] 0.262 0.295 0.331 0.418 0.47 ...
Cluster_pattern          Large matrix (24383289 elements, 186 Mb)
   um [1:353381, 1:69] 0 0 0 0 0 0 0 0 0 0 ...
I                        Large matrix (16962288 elements, 129.4 Mb)
   um [1:353381, 1:48] 1 0 0 14 9 19 2 10 10 5 ...
TI                       num [1:48, 1] 2078346 2723146 1561823 1642086 2565781 ...
TX                       num [1:48, 1] 110736 140024 1293926 1425150 105630 ...
X                        Large matrix (16962288 elements, 129.4 Mb)
   um [1:353381, 1:48] 0 0 0 0 0 0 0 0 0 0 ...
```

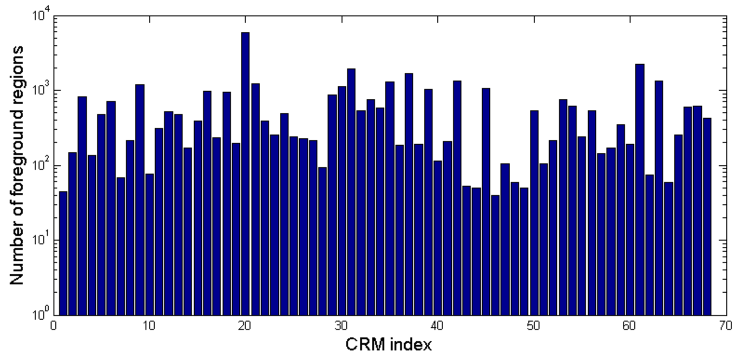
Fig. 3. Intermediate results of one round of sampling.

We run 1000 rounds of sampling and accumulate TF-region and Module-region regulatory maps as follows:

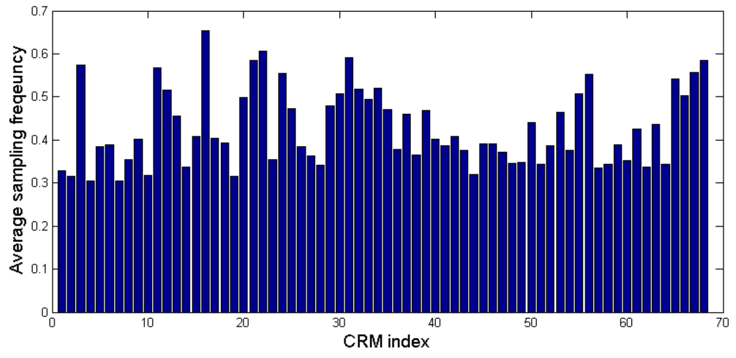
```
Summition_binding=Summition_binding+Sampling_temp_results$Binding_pattern
Summition_cluster=Summition_cluster+Sampling_temp_results$Cluster_pattern
```



(a) Histogram of module-region sampling frequency



(b) Number of bound regions for each module



(c) Abondance of each module

Fig. 4. Sampling results of 68 candidate modules of K562 cell line.