Genomic Computing Evaluation

Assignment 2: GMQL

Fabrizio Frasca

April 13, 2017

1 Activator and Repressive epigenetic signals

Among the different epigenetic signals, H3K27ac and H3K27me3 are generally associated with active and repressed chromatin regions, respectively. Considering H3K4me1 in cell line A549 and the aforementioned signals (H3K27ac and H3K27me3) in broadPeak format, under ethanol treatment (EtOH):

1.1

Using GMQL, select the required ENCODE ChIP-seq data for the reference human genome hg19 (originally from the @UCSC website), i.e. the H3K4me1, H3K27ac and H3K27me3 under ethanol treatment (EtOH) in broadPeak format, and the promoter region annotation for the same reference human genome, by writing the required GMQL statements.

```
H3K4me1_0 = SELECT(treatment == 'EtOH_0.02pct' AND dataType == 'ChipSeq'
                 AND type == 'broadPeak' AND antibody_target == 'H3K4me1'
                     AND cell == 'A549') HG19_ENCODE_BROAD;

PROMOTER_0 = SELECT(annotation_type == 'promoter') HG19_BED_ANNOTATION;

H3K27me3_0 = SELECT(treatment == 'EtOH_0.02pct'
                 AND dataType == 'ChipSeq' AND type == 'broadPeak'
                     AND antibody_target == 'H3K27me3') HG19_ENCODE_BROAD;

H3K27ac_0 = SELECT(treatment == 'EtOH_0.02pct'
                 AND dataType == 'ChipSeq' AND type == 'broadPeak'
                     AND antibody_target == 'H3K27ac') HG19_ENCODE_BROAD;

H3K27ac = COVER(1,ANY; aggregate: BAG(name)) H3K27ac_0;
H3K27me3 = COVER(1,ANY; aggregate: BAG(name)) H3K27me3_0;
H3K4me1 = COVER(1,ANY; aggregate: BAG(name)) H3K4me1_0;
PROMOTER = COVER(1,ANY; aggregate: BAG(name)) PROMOTER_0;
```

Here I took the promoters by just enforcing the specific annotation_type. If we wanted to take the promoters explicitly from RefSeq then we would have to correctly set the provider, to extract the TSS of the genes and then to manually update the start and stop attributes in order to construct the promoter regions. This is because RefSeq does not provide promoters directly.

The COVER operations were performed in order to 'collapse' several experiments and overlapping regions into one single sample per dataset, with no overlapping regions. This is to ease the computation and to avoid replicates.

Then, compute the following:

a. Active H3K4me1 regions (i.e. overlapping with H3K27ac regions)

```
H3K4me1_H3K27ac = JOIN(distance<0; output: INT) H3K4me1 H3K27ac;
```

Overlapping regions are the ones characterized by having a negative genometric distance. Then, we impose INT as output in order to take the portions of the regions which actually overlap, and only those ones.

b. Repressed H3K4me1 regions (i.e. overlapping with H3K27me3 regions)

```
H3K4me1_H3K27me3 = JOIN(distance<0; output: INT) H3K4me1 H3K27me3;
```

c. Poised H3K4me1 regions (i.e. being simultaneously active and repressed)

```
H3K4me1_H3K27ac_H3K27me3 = JOIN(distance<0; output: INT)
            H3K4me1_H3K27ac H3K4me1_H3K27me3;
```

This join highlights the portion of regions which are simultaneously present in all the three starting datasets.

d. Active H3K4me1 regions in promoters

```
H3K4me1_H3K27ac_PROMOTER = JOIN(distance<0; output: INT)
            H3K4me1_H3K27ac PROMOTER;
```

The needed operation is the same as those above, except that we want to search for overlaps with promoters.

e. For each region in (d) find the closest H3K4me1 region farther than 10 kb

```
RES_0 = JOIN(distance > 10000, mindistance(1); output: RIGHT)
H3K4me1_H3K27ac_PROMOTER H3K4me1;
RES = EXTEND(regionCount AS COUNT()) RES_0;
```

Here it is important to write the join genometric predicates in the correct order, since the filtering on the minimum distance has to be applied before picking the closest region.

f. Store the last result (e)

```
MATERIALIZE RES INTO res;
```

g. Run the created GMQL query and report: running time, obtained number of samples and number of regions in each sample. Can the result include replicated regions? Why?

As far as the running time is concerned, the query took 15 sec in pending state and then about 2 min and 20 sec for running.

The query output is made up of just one single sample as expected. This is because we performed a COVER operation for all the input datasets in order to 'collapse' several experiments and overlapping regions into one single sample per dataset, with no overlapping regions. The next operations we applied are not supposed to generate more than one sample, explaining the presence of one single output sample.

The number of regions in the output sample can be read in the metadata of the output sample, stored as "regionCount". Unfortunately this number is not really significant because the join operator may sometimes unwillingly generate regions replicated 4 times, a bug that has been reported.

Even if we had managed carefully the possibility to have replicates, it is still possible, in principle, to have replicates because several active regions may share the same H3K4me1 region satisfying the conditions required (being the closest that is farther than 10 bases). These replicates are actually meaningful because they highlight the importance of one particular H3K4me1 region.

2 Differential binding

Consider the JUN Transcription Factor (TF) in cell line K562 in two different conditions: treatment with interferon alpha 30 minutes (IFNa30) and untreated. Considering the untreated sample as the baseline:

2.1

Using GMQL, select the required ENCODE ChIP-seq data for the reference human genome hg19 (originally from the @UCSC website), i.e. for JUN antibody_target with none, or IFNa30 treatment in narrowPeak format, and the RefSeq promoter region annotation for the same reference human genome, by writing the required GMQL statements. Combine replicas if needed.

```
JUN_BASELINE_0 = SELECT(cell == 'K562' AND antibody_target == 'JUN'
            AND dataType == 'ChipSeq' AND treatment == 'None')
                  HG19_ENCODE_NARROW;
JUN_BASELINE = COVER(1, ANY; aggregate: BAG(name)) JUN_BASELINE_0;

JUN_IFNa30_0 = SELECT(cell == 'K562' AND antibody_target == 'JUN'
           AND dataType == 'ChipSeq' AND treatment == 'IFNa30')
                 HG19_ENCODE_NARROW;
JUN_IFNa30 = COVER(1, ANY; aggregate: BAG(name)) JUN_IFNa30_0;

GENE = SELECT(provider == 'RefSeq' AND annotation_type == 'gene')
                 HG19_BED_ANNOTATION;

TSS_0 = PROJECT(region_update: stop AS start+1) GENE;
TSS = COVER(1,ANY; aggregate: BAG(name)) TSS_0;

PROMOTER = PROJECT(region_update: start AS start - 2000,
                 stop AS start + 1000) TSS;
```

Considering that we are required to work with RefSeq annotations, we manually construct promoter regions starting from the gene annotations, since the ones for the promoters are not provided by RefSeq.

COVER operations are performed, with the aim described in exercise 1.

Then, compute the following:

a. DNA regions in common with the baseline and the treatment

```
JUN_BASELINE_IFNa30 = JOIN(distance<0; output: INT) JUN_BASELINE JUN_IFNa30;
```

b. Considering the regions identified in (a), find the promoters in which they are present

Promoters satisfying the wanted condition (with some overlapping (a) regions) can be extracted by first executing a MAP operation with PROMOTER as reference dataset and then by filtering out all the regions in the MAP result with overlapping count set to 0. These correspond to promoters with no (a) regions present.

```
RES_0 = MAP() PROMOTER JUN_BASELINE_IFNa30;
RES = SELECT(region: count_PROMOTER_JUN_BASELINE_IFNa30>0) RES_0;
```

c. Store the last result (b)

```
MATERIALIZE RES INTO res;
```