GMQL Examples (Draft)

Introduction

This documents contains notable examples and more complex biological examples based on GMQL query language. Note that it is currently in preparation therefore it might contain instructions that do not work as intended yet.

The document is structured in two parts:

A. NOTABLE EXAMPLES: this section contains a list of query examples used to perform non-basic operations that cannot be done using single GMQL statements only, or may not be obvious at first glance from the simple operator definitions. These GMQL snippets are interesting for their use inside longer, more structured GMQL queries;

B. BIOLOGICAL EXAMPLES: this section collects several examples where GMQL is used to answer practical questions/tasks of biological and clinical interest. For each example, after an initial textual statement describing the question/task to be answered, the GMQL query that answers it is reported together with a detailed commented description of the query and its results.

A. NOTABLE EXAMPLES

This section contains a list of GMQL query examples used to perform non-basic operations that cannot be done using single GMQL statements only, or may not be obvious at first glance from the simple operator definitions. These GMQL snippets are interesting for their use inside longer, more structured GMQL queries.

Note: in each of the following subsections, template queries are given for each example. In each template, dataset, metadata and region attribute names are for illustration purposes only and might not correspond with any value found in actual public datasets contained in the GMQL repository.

1) Dataset renaming

```
NEW_DS_NAME = SELECT() OLD_DS_NAME;
```

This very simple select statement allows renaming an existing dataset with a more convenient name.1 It is particularly useful when the old dataset is a result of a previous GMQL query and is saved in the repository with a cumbersome name: using this procedure one can rename it while loading it from the repository.

<u>Example</u>:

```
CONSENSUS = SELECT()
job_consensus_query_user_20000101_000000_JUN_consensus;
```

2) Attribute duplication for processing

```
OUTPUT_DS = PROJECT(region_update: att_name_copy AS att_name) INPUT_DS;
```

Sometimes it may be useful to store the content of a region attribute (coordinate or feature) into a different region attribute in order to manipulate it without modifying the original values. In GMQL this can be done by using the PROJECT command as shown, where *att_name* is the name of the original attribute and *att_name_copy* is the name of the new attribute with the values to be edited.

Example:

```
GEN_SEG = SELECT(dataType == 'DnaseSeq' AND cell == 'H1-hESC')
                                                  HG19_ENCODE_BED;
GEN_SEG_NS = PROJECT(region_update: newScore AS score) GEN_SEG;
```

This example shows a possible application of the attribute duplication option. A user might need to save the *score* region attribute in a new attribute (named *newScore*) where its values can be modified without altering the values in the original attribute. The PROJECT operation allows to do this. The new dataset GEN_SEG_NS contains the new attribute and safely stores also the original *score* attribute.

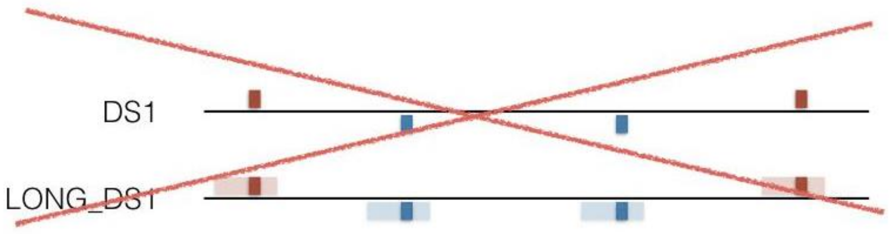
3) Stranded region extension

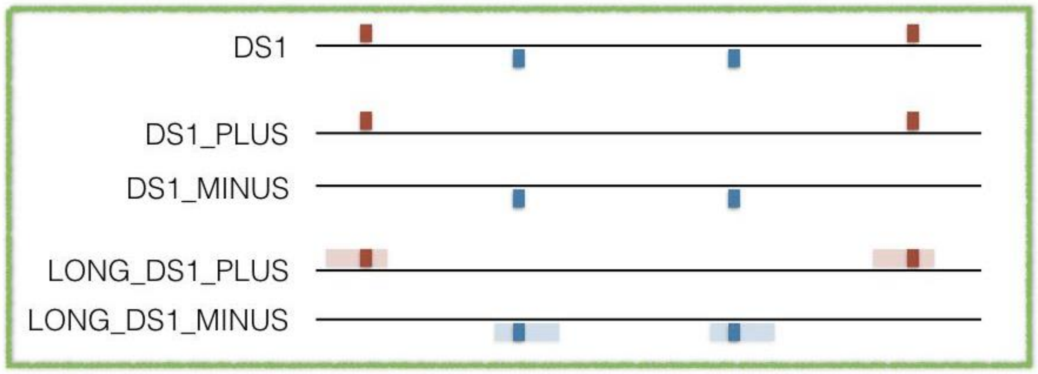
```
DS1_PLUS = SELECT(region: strand == +) DS1;
DS1_MINUS = SELECT(region: strand == -) DS1;

LONG_DS1_PLUS = PROJECT(region_update: left AS left - 100000) DS1_PLUS;
# Do whatever on LONG_DS1_PLUS here
# ….
LONG_DS1_MINUS = PROJECT(region_update: right AS right + 50000) DS1_MINUS;
# Do whatever on LONG_DS1_MINUS here
# ….

# If needed
DS1_SPLIT = UNION() LONG_DS1_PLUS LONG_DS1_MINUS;
DS1_EXTENDED = MERGE(groupby: ID) DS1_SPLIT;
```

Suppose that a certain dataset DS1 is composed of multiple samples with regions that lay on both the plus and minus strand of the genome, and suppose that a particular application requires to spatially extend in a given direction (upstream/downstream) the regions in the samples of such dataset. In the current GMQL version, a single PROJECT command can produce unwanted results (see upper part of the figure below, where such case is presented).





Thus, this snippet of code guarantees that the upstream/downstream region extension is performed correctly on both strands. Additionally, dividing a dataset in this way allows to perform different operations on the two different strands; if needed, one can collapse the two datasets back to the original dataset using a UNION() and a MERGE(groupby: ID) command, using the groupby option applied on sample ID (where ID is a metadata attribute that univocally identifies the samples in a dataset) to recompose the original samples after region extension when the original dataset contains multiple samples.
Note: due to the current UNION() functioning, the final obtained DS1_EXTENDED dataset will include duplicated metadata with respect to the original input DS1 dataset, half metadata with attribute names prefixed with *LONG_DS1_PLUS* and half with *LONG_DS1_MINUS*.

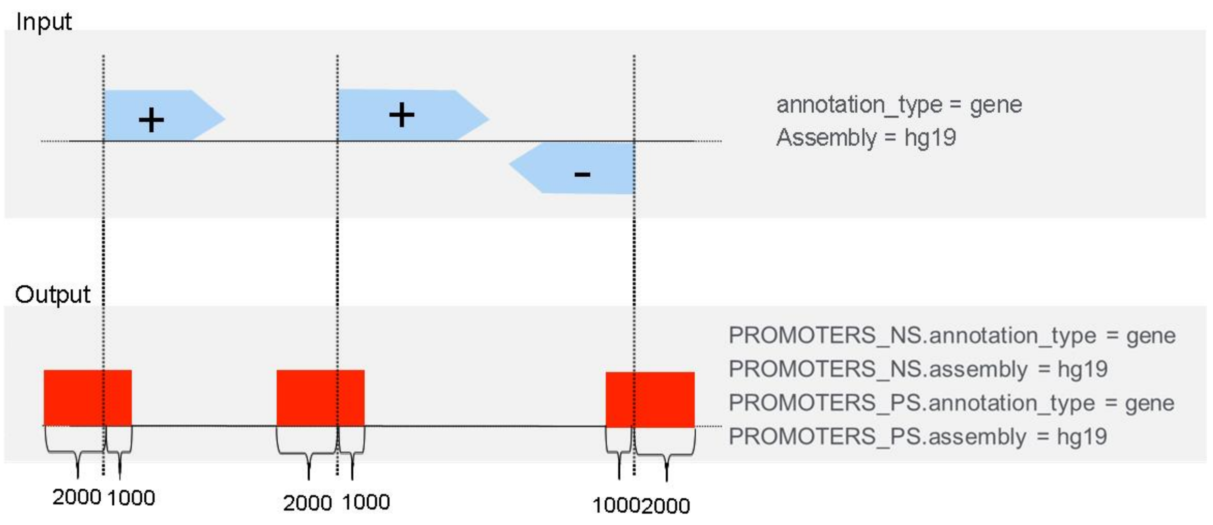
Example 1:
```
TSS_PLUS = SELECT(annotation_type == 'TSS'; region: strand == +)
                                                      HG19_BED_ANNOTATION;
TSS_MINUS = SELECT(annotation_type == 'TSS'; region: strand == -)
                                                      HG19_BED_ANNOTATION;
LONG_TSS_PLUS = PROJECT(region_update: left AS left - 100000) TSS_PLUS;
LONG_TSS_MINUS = PROJECT(region_update: right AS right + 100000) TSS_MINUS;
LONG_TSS = UNION() LONG_TSS_PLUS LONG_TSS_MINUS;
MERGED_LONG_TSS = MERGE() LONG_TSS;
```

This example is the direct application of the scenario presented in this notable example section. Notice that from the used input dataset only a single sample is extracted, which contains Transcription Start Sites (TSSs): they represent the first single DNA base of gene transcripts, which is conventionally used as the starting point of gene transcription. The PROJECT operator allows extending the TSSs 100k bp upstream (i.e., on their left for the positive stranded TSSs, and on their right for the negative stranded TSSs). The output datasets LONG_TSS_PLUS and LONG_TSS_MINUS contain the extended TSSs. UNION and MERGE operations let obtaining a final result dataset with extended positive and negative stranded TSSs all included in a single sample, as in the original dataset.

Example 2:

```
GENE_PS = SELECT(annotation_type == 'gene'; region: strand == +)
                                                        HG19_BED_ANNOTATION;
GENE_NS = SELECT(annotation_type == 'gene'; region: strand == -)
                                                        HG19_BED_ANNOTATION;
PROMOTER_PS = PROJECT(region_update: right AS left + 1000, left AS left - 2000)
                                                        GENE_PS;
PROMOTER_NS = PROJECT(region_update: left AS right - 1000, right AS right + 2000)
                                                        GENE_NS;
PROMOTER_ALL = UNION() PROMOTER_PS PROMOTER_NS;
PROMOTER = MERGE() PROMOTER_ALL;
```

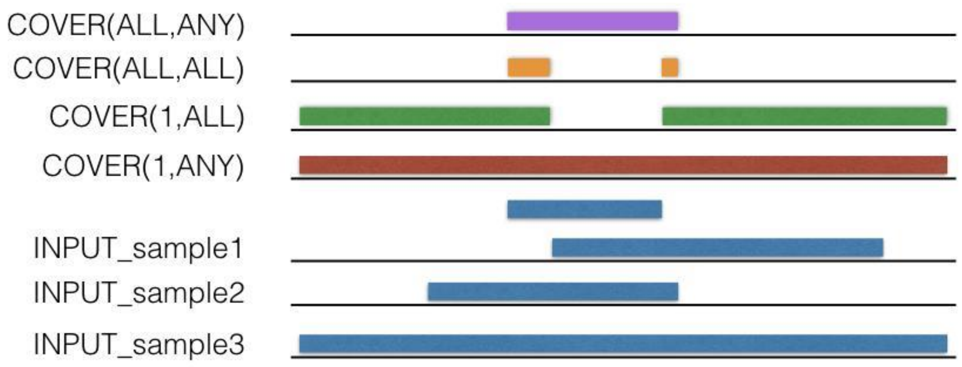
This example is built with the same structure of the previous one, but it serves a different purpose: in this case the input dataset is made of genes instead of Transcription Start Sites.
The initial SELECT statements consider an input dataset of annotations and extract those included samples (usually one) regarding genes, dividing the latter ones into two different output datasets: GENE_PS containing only gene regions on the positive strand and GENE_NS with only gene regions on the negative strand.
To define a promotorial region, it is necessary to start from a TSS (a single base of DNA, at the beginning of a gene transcript, conventionally taken as the starting point for the gene transcription) and extend it upstream and downstream by a number of given bases. When no TSS annotation data are available, also gene annotation data can be used to extract the main gene TSS as the first gene base (conventionally considered as a TSS); notice however that a gene can have multiple TSSs, one for each of its transcripts.
As an example, here, the values of the coordinate attributes *left* and *right* of a gene region are redefined by shifting the region ends such that the start (beginning) of the new region is set 2k bp before the start of the gene region, and the stop (end) of the new region is set 1k bp after the start of the gene region (where the start of a gene region is a TSS); thus, the new *left* and *right* coordinates of the region represent the ends of a promotorial region. Notice that, since the beginning and end of a region depend on the reading direction of the region (i.e. on the region strand), this needs to be performed separately for each strand. The first PROJECT statement modifies the coordinates of the regions on the positive strand (in the GENE_PS dataset), while the second PROJECT statement deals with the regions on the negative strand (in the GENE_NS dataset). Since in the latter ones the region start and stop positions are inverted, the statement applies a shift of 2k bp to the *right* coordinate and a shift of 1k bp to the *left* coordinate of the first base of a gene, i.e. of a gene TSS, (always in opposite directions, to enlarge the TSS region).
Then, the UNION statement unifies the resulting samples in a unique dataset and the MERGE statement merges all samples in a single one.


Input
+
+
-
annotation_type = gene
Assembly = hg19
Output
PROMOTERS_NS.annotation_type = gene
PROMOTERS_NS.assembly = hg19
PROMOTERS_PS.annotation_type = gene
PROMOTERS_PS.assembly = hg19
2000 1000
2000 1000
1000 2000

4) Maximum consensus

```
DS1_TILES = HISTOGRAM(ALL, ANY) DS1;
DS1_TILES_0 = EXTEND(maxCount AS MAX(AccIndex)) DS1_TILES;
DS1_MAX_CONS = SELECT(region: AccIndex == META(maxCount)) DS1_TILES_0;
```

Given a dataset DS1, one can extract the maximum consensus among its samples, i.e. those regions (or parts of) that present the local maximum amount of overlap between all dataset sample regions. This can be tricky due to the fact that a single sample can include overlapping regions; these can make results from SUMMIT and HISTOGRAM operation inaccurate (see picture below).



The blue regions are from 3 ATF3 transcription factor samples of ChIP-seq experiments in cell line K562, contained in DS1 dataset.

- Red regions are from the result of the command COVER(1, ANY) DS1;
- Green regions are from the result of the command COVER(1, ALL) DS1;
- Orange regions are from the result of the command COVER(ALL, ALL) DS1;
- Violet regions are from the result of the command COVER(ALL, ANY) DS1;

COVER(1, ALL) (and therefore SUMMIT(1, ALL)), where in the above example ALL = 3, excludes the middle region of COVER(1, ANY). This is working as intended, but it is counterintuitive to its definition (the middle region is, strictly speaking, a part of all samples);

a similar argument can be made for the orange region, i.e. COVER(ALL, ALL). The pink region (COVER or SUMMIT(ALL, ANY)) is the only one that successfully captures the four overlapping regions in the middle.

The here proposed GMQL code snippet extracts the maximum consensus by using the HISTOGRAM command, which generates a region attribute named *AccIndex*, that stores the number of locally overlapping regions. By storing the maximum of this value across all regions, one can then correctly perform the required operation of extracting the maximum consensus among all the input samples also when one or more of such samples include overlapping regions.

Example:

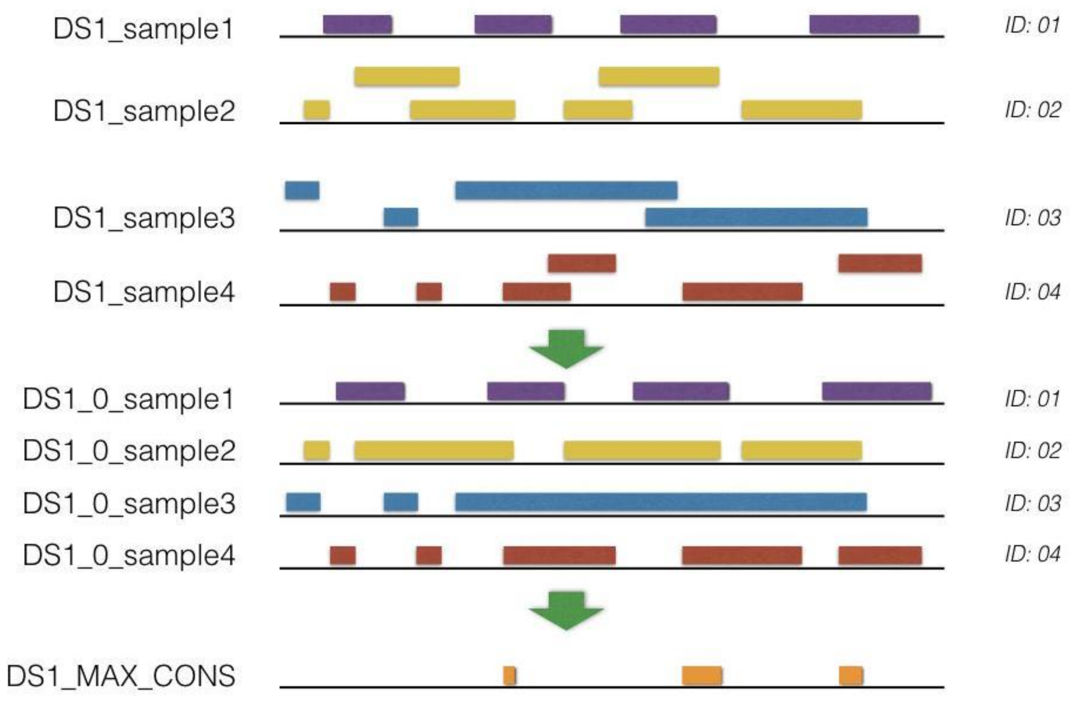
```
K562_JUN = SELECT(dataType == 'ChipSeq' AND cell == 'K562' AND
                                          antibody_target == 'JUN') HG19_ENCODE_NARROW;
K562_JUN_TILES = HISTOGRAM(ALL,ANY) K562_JUN;
K562_JUN_TILES_0 = EXTEND(maxCount AS MAX(AccIndex)) K562_JUN_TILES;
K562_JUN_MAX_CONS = SELECT(region: AccIndex == META(maxCount))
K562_JUN_TILES_0;
```

This example extracts all ENCODE samples (in narrowPeak format) containing ChIP-seq data for the transcription factor JUN taken from tumoral cell line K562. Then, using the HISTOGRAM command, it produces the zones with the highest accumulation values of the regions of all input samples (see description of the HISTOGRAM command). Using the EXTEND command on the HISTOGRAM output, it stores in the metadata (as value of the *maxCount* attribute) the maximum of the accumulation values, and then SELECTs all zones with the maximum value.

5) “One from each” consensus

```
DS1_0 = COVER(1, ANY; groupby: ID) DS1;
DS1_MAX_CONS = COVER(ALL, ALL) DS1_0;
```

Similarly to the above example, this snippet of GMQL code is used to extract maximum consensus regions from all samples in a given dataset. However, this time we take a more “democratic” approach: we require the maximum consensus region to overlap a region in each sample of the original dataset. To avoid overlapping regions in a single sample, we prepare the original dataset with a COVER grouping by sample ID: since each sample ID is unique, this guarantees that each sample is COVERed separately. This in turn makes COVER(ALL, ALL) extract the consensus region that we are looking for.
Note that the COVER removes all the region attributes in the original dataset, unless their values are aggregated over overlapping regions.



Example:

```
K562_POLR2A = SELECT(dataType == 'ChipSeq' AND cell == 'K562' AND
                                        antibody_target == 'POLR2A') HG19_ENCODE_NARROW;
K562_POLR2A_0 = COVER(1, ANY; groupby: ID) K562_POLR2A;
K562_POLR2A_MAX_CONS = COVER(ALL, ALL) K562_POLR2A_0;
```

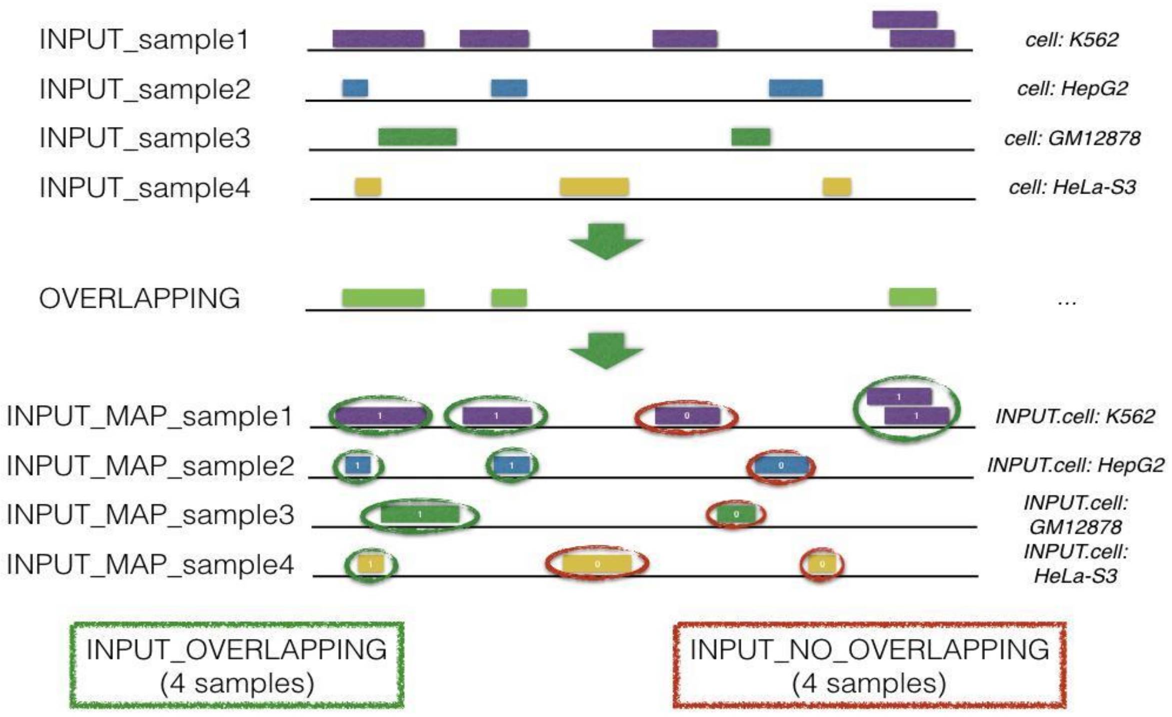
This example extracts all ENCODE samples (in narrowPeak format) containing ChIP-seq data for transcription factor POLR2A taken from tumoral cell line K562. After collapsing together regions that overlap in a single sample (COVER(1, ANY; groupby: ID)), it extracts all genomic zones that contain exactly one region in each sample.

6) Extraction of overlapping / no-overlapping regions / parts of regions (in one / multiple samples)

PART A

```
OVERLAPPING = COVER(2, ANY) INPUT;
INPUT_MAP = MAP() INPUT OVERLAPPING;
# Overlapping regions
INPUT_OVERLAPPING = SELECT(region: count_INPUT_OVERLAPPING > 0)
INPUT_MAP;
# No-overlapping regions
INPUT_NO_OVERLAPPING = SELECT(count_INPUT_OVERLAPPING == 0) INPUT_MAP;
```

This snippet of code allows to extract, for each sample of an input dataset (say, INPUT), in the INPUT_OVERLAPPING output dataset, the regions of the sample that overlap with at least one other region from the same dataset (in the same or other samples of the dataset), whereas in the INPUT_NO_OVERLAPPING dataset those regions of the sample that do not overlap with any other region from the same dataset. Examples of application of the overlapping region extraction include consensus search in biological replicates (i.e. the search for those regions with the highest combined evidence given the input dataset), whereas examples of the not overlapping region extraction include the search for those regions that are specific of a sample (such as cell line specific binding sites of a certain transcription factor). An illustration of results can be found in the following pictures, where green circled regions are only in the INPUT_OVERLAPPING dataset, whereas red circled regions are only in the INPUT_NO_OVERLAPPING dataset (and both datasets do not include samples without any region):



<u>Note</u>: Each sample in output from the MAP-SELECT combination contains all metadata from the corresponding INPUT sample (with their attribute names prefixed with '*INPUT.*'), as well as all the distinct metadata from all samples in INPUT (with their attribute names prefixed with *'OVERLAPPING.*') as a consequence of the MAP operation itself.

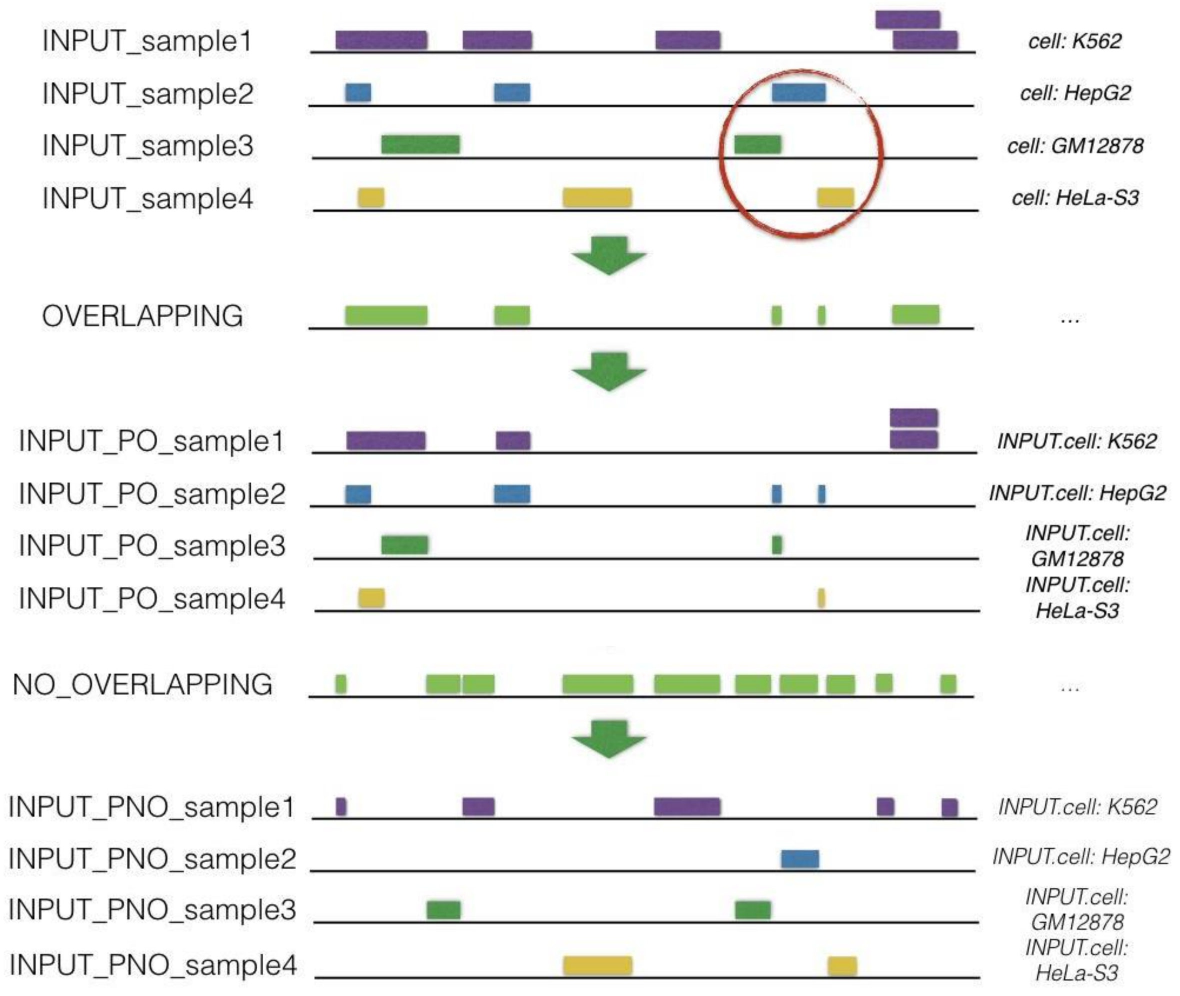
<u>PART B</u>

In certain cases, it is required to determine which parts of a given region intersect with at least one other, or with no other, region in a dataset. This gives a much higher level of accuracy for results in all cases where consensus, or exclusiveness, among regions is required. Overlapping (and no-overlapping) parts of regions can be obtained with a JOIN command with the INT output option:

```
# Overlapping region parts
OVERLAPPING = COVER(2, ANY) INPUT;
INPUT_PART_OVERLAPPING = JOIN(DLE(-1); output: INT) INPUT OVERLAPPING;
# No-overlapping region parts
```

```
NO_OVERLAPPING = COVER(1, 1) INPUT;
INPUT_PART_NO_OVERLAPPING = JOIN(DLE(-1); output: INT) INPUT
                                                           NO_OVERLAPPING;
```

A view of the results of these operations can be found in the next pictures (where INPUT_PO refers to INPUT_PART_OVERLAPPING, INPUT_PNO to INPUT_PART_NO_OVERLAPPING, and the three circled regions are pairwise limitedly overlapping). Due to how COVER and JOIN interact, some problems can arise when regions are exactly adjacent, i.e. when one region ends when the other starts:



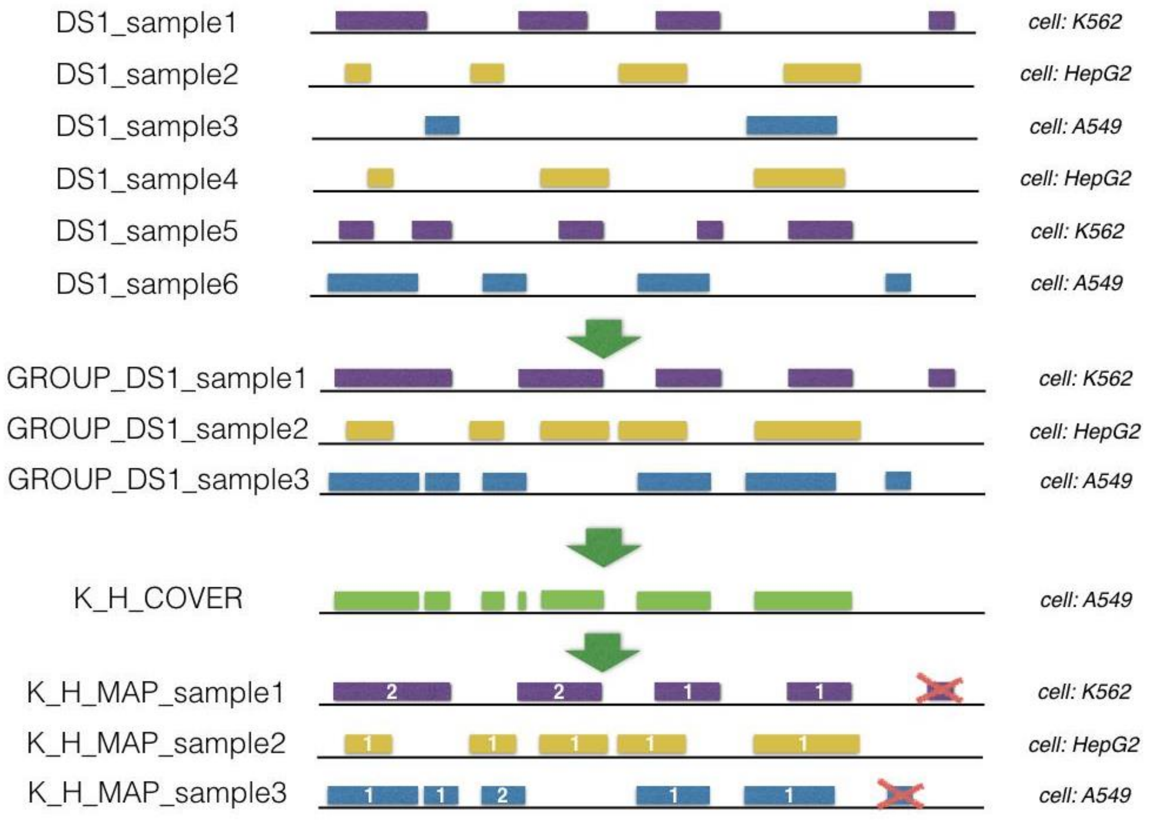
Metadata behaviour in the output sample is the same as in previous PART A. Also note that resulting region attributes contain the *JaccardIntersect* and *JaccardResult* values from the original COVERed INPUT region. If one would need to remove them, a *PROJECT(ALLBUT JaccardIntersect, JaccardResult)* can be used.

7) K-H overlappings from grouped dataset

```
GROUP_DS1 = COVER(1, ANY; groupby: attr_name) DS1;
K_H_COVER = COVER(K, H) GROUP_DS1;
K_H_MAP = MAP() GROUP_DS1 K_H_COVER;
```

```
RES = SELECT(region: count_GROUP_DS1_K_H_COVER > 0) K_H_MAP;
```

Suppose you have a dataset DS1 that includes samples which can be grouped based on a metadata attribute *attr_name*., and that you like to extract all those contiguous (non-overlapping) regions from the grouped samples of DS1 which overlap at least K-1 and at most H-1 other regions in at least K-1 and at most H-1 other samples of the same DS1 dataset.
This snippet of code allows doing that (as each region to be extracted from each sample is included in the accumulation counts of the *COVER(K ,H)* (which considers regions having all different values of the grouping attribute *attr_name* due to the previous *COVER(1, ANY; groupby: attr_name)*).
The initial COVER(1, ANY) is necessary to merge and flatten (make non-overlapping) regions from DS1 samples with the same grouping metadata attribute *attr_name*, and to prevent confusion when SELECTing on the MAP count attribute (it grants that regions with count > 0 have *minAcc >= K* and *maxAcc <= H* regarding other sample regions).
The following pictures provide a visual example in the case when the grouping metadata *attr_name* is equal to *cell*, H = 2 and K = 3 (values within regions of K_H_MAP samples represent MAP count values).



<u>Example</u>:

```
HMN_RAD21 = SELECT(dataType == 'ChipSeq'
                    AND antibody_target == 'RAD21') HG19_ENCODE_NARROW;
GROUP_RAD21 = COVER(1, ANY; groupby: cell) HMN_RAD21;
2_5_COVER = COVER(2, 5) GROUP_RAD21;
2_5_MAP = MAP() GROUP_RAD21 2_5_COVER;
```

```
CONSENSUS = SELECT(region: count_GROUP_RAD21_2_5_COVER > 0) 2_5_MAP;
MATERIALIZE CONSENSUS INTO consensus;
```

This example extracts all samples from the ENCODE narrowPeak dataset, aligned to human 19 reference genome, which pertain to ChIP-seq experiments investigating genome bindings of the RAD21 transcription factor in human (HMN_RAD21), and groups them according to their cell value. Assuming for example that only 15 samples remain (one per cell line), the query then extracts only those regions that are found in 2 to 5 samples (or better said, only those regions which have at least a subpart of them found in 2 to 5 samples).

B. BIOLOGICAL EXAMPLES

This section collects several examples where GMQL is used to answer practical questions/tasks of biological and clinical interest. For each example, after an initial textual statement describing the question/task to be answered, the GMQL query that answers it is reported together with a detailed commented description of the query and its results.

1) Find ChIP-seq peaks in promoter regions

"*In each enriched region (often called 'peak') sample from ENCODE human ChIP-seq experiments, find how many peaks co-occur in each promoter region.*"

```
HM_TF = SELECT(dataType == 'ChipSeq'
                                AND view == 'Peaks') HG19_ENCODE_NARROW;
TSS = SELECT(annotation_type == 'TSS' AND provider == 'UCSC';
                                region: score >= 1000) HG19_BED_ANNOTATION;
PROM = PROJECT(region_update: left AS left - 2000, right AS right + 1000) TSS;
PROM_HM_TF = MAP() PROM HM_TF;
MATERIALIZE PROM_HM_TF INTO PROM_HM_TF;
```

This GMQL query starts by selecting all the ChIP-seq peak samples from the dataset *HG19_PEAK*, which can represent, e.g., all the ENCODE peak samples called from human NGS experiments aligned to the human reference genome assembly 19 (https://genome.ucsc.edu/ENCODE/dataMatrix/encodeChipMatrixHuman.html). From an annotation base for the same human genome assembly (represented by the dataset *HG19_ANN)*, it also selects the transcription start sites (TSS), provided by the UCSC database (https://genome.ucsc.edu/cgi-bin/hgTables) from SwitchGear Genomics (http://switchgeargenomics.com/) and defines the promoter regions as the ones extending from 2000 bases upstream a TSS to 1000 bases downstream of the same TSS (using only TSS with high confidence score based on existing experimental evidence). Finally, it maps the peaks of each sample to the promoter regions and, for each promoter, counts how many of such peaks co-localize with the promoter, saving the obtained results.
Although very simple, this example shows the power of the Map operation, which easily relates experimental data to known annotations. Note that a Select-Project-Map sequence of GMQL operations is comparable with the basic Select-From-Where sequence of SQL statements. When applied to the entire ENCODE data collection, at the time of writing this GMQL query selects 2,423 samples including a total of 83,899,526 peaks, which are mapped to 131,780 promoters, saving as result 29 GB of data files in standard GTF format. Executed in a Hadhoop framework on a single server equipped with a Dual Intel Xeon ES-2650 processor, 64 GB of RAM and 16 TB of disks (RAID 5, 5 disks), it required 18 min and 33 s.

2) Find distal bindings in transcription regulatory regions

"*Find all enriched regions (peaks) in CTCF transcription factor (TF) ChIP-seq samples from different human cell lines which are the nearest regions farther than 100 kb from a transcription start site (TSS). For the same cell lines, find also all peaks for the H3K4me1 histone modifications (HM) which are also the nearest regions farther than 100 kb from a TSS. Then, out of the TF and HM peaks found in the same cell line, return all TF peaks that overlap with both HM peaks and known enhancer (EN) regions.*"

```
TF = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
                        AND antibody_target == 'CTCF') HG19_ENCODE_NARROW;
HM = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
                        AND antibody_target == 'H3K4me1') HG19_ENCODE_BROAD;
TSS = SELECT(annotation_type == 'TSS' AND provider == 'UCSC')
                        HG19_BED_ANNOTATION;
EN = SELECT(annotation_type == 'enhancer' AND provider == 'UCSC')
                        HG19_BED_ANNOTATION;
TF1 = JOIN(distance > 100000, mindistance(1); output: right) TSS TF;
HM1 = JOIN(distance > 100000, mindistance(1); output: right) TSS HM;
HM2 = JOIN(distance < 0; output: int) EN HM1;
TF_res_0 = MAP(joinby: cell) TF1 HM2;
TF_res = SELECT(count_HM2_TF1 > 0) TF_res_0;
MATERIALIZE TF_res INTO TF_res;
```

This second example, whose context is illustrated in Figure 1, shows that GMQL is a powerful expressive language to answer frontier epigenomics questions.

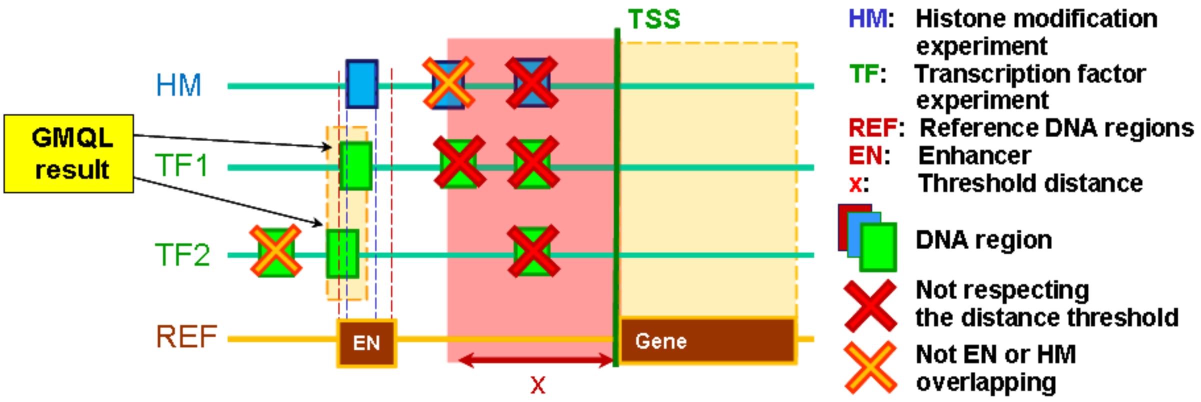

TSS
HM
GMQL result
TF1
TF2
REF
EN
Gene
x
HM: Histone modification experiment
TF: Transcription factor experiment
REF: Reference DNA regions
EN: Enhancer
x: Threshold distance
DNA region
Not respecting the distance threshold
Not EN or HM overlapping

Figure 1. Representation of the histone modification (HM) and transcription factor (TF) binding site enriched regions ('peaks'), known reference DNA regions and their distance relationships involved in Example 2

The GMQL query selects TF and HM enriched region samples from *HG19_PEAK*, a collection of experimental data files of signal enrichment called regions from NGS human samples aligned to the human genome assembly 19, e.g. provided by ENCODE (https://genome.ucsc.edu/ENCODE/dataMatrix/encodeChipMatrixHuman.html). It also selects TSS and EN known human annotation regions from *HG19_ANN*, e.g. provided by the UCSC database (https://genome.ucsc.edu/cgi-bin/hgTables) from SwitchGear Genomics (http://switchgeargenomics.com/) and Vista Enhancer (http://enhancer.lbl.gov/), respectively.

Then, for each sample, it computes in *TF1* the TF regions that are at minimal distance from a TSS, provided that such distance is greater than 100,000 bases, and in *HM1* the HM regions that are also at minimal distance from a TSS, with the same constraint. Note that, in addition to the distal condition, the Join parameter also indicates that the result must include only the right sample matching regions, for *TF1* and *HM1* respectively. Next, in *HM2* the query computes those *HM1* regions that intersect with enhancer regions in *EN*, by taking the intersection of resulting regions. Finally, in *TF_res*, which is then saved as a result, for each cell line, it computes the TF regions in *TF1* that intersect with regions in *HM2*, also this time taking only the right sample matching regions. At the time of writing, all enriched region samples from ENCODE human ChIP-seq amount to 2,423 samples from 120 cell lines, which become 987 samples after merging sample replicates. When applied to such merged replicate samples, initially this example query selects 90 CTCF and 21 different H3K4me1 samples from 93 cell lines, including a total of 3,317,361 peaks; finally, as the result it finds 42 CTCF peaks in 11 samples from 11 cell lines.

The Map operation (in Examples 1, 3 and 4) as well as the Join operation with *distance* and *first after distance* functions (in Example 2) highlight the power of GMQL in performing genometric evaluations in batch on multiple samples at a time; they are normally performed by executing data manipulation scripts, developed by individual researchers in different programming languages.

3) Associating transcriptomics and epigenomics

"*In RNA-seq experiments of human cancer cell line HeLa-S3, find the average expression of each exon. Then, in ChIP-seq experiments of the same cell line, find the average signal of H3K4me3 histone modifications within each exon.*"

```
Exon = SELECT(annotation_type == 'exons'
                        AND provider == 'RefSeq') HG19_BED_ANNOTATION;
HM = SELECT(dataType == 'ChipSeq' AND view == 'Peaks' AND cell == 'HeLa-S3'
                        AND antibody_target == 'H3K4me3') HG19_ENCODE_NARROW;
RNA = SELECT(dataType == 'RnaSeq' AND view == 'ExonsDeNovo'
                        AND cell == 'HeLa-S3'; region: IDR < 0.05) HG_RNA_GTF;
RNA2 = PROJECT(region_update: signal AS FPKM1 / 2 + FPKM2 / 2) RNA;
Exp = UNION() RNA2 HM;
Genome_space = MAP(signal_avg AS AVG(signal)) Exon Exp;
MATERIALIZE Genome_space INTO Genome_space;
```

This third example shows that datasets produced by different experiment types, such as RNA-seq and ChIP-seq, can be used in the same GMQL query, thanks to the interoperability provided by GDM; the *UNION* operator takes care of automatically unifying their different region data schemas. For each RNA-seq sample, the signal in exons with reproducible expression, i.e. with Irreproducibility Discovery Rate (IDR) < 0.05, is averaged (in the new attribute *signal*) over two replicates (ENCODE provides RNA-seq data files with expression data quantified for two replicates separately as FPKM, i.e. Fragments Per Kilobase of exon per Million fragments mapped). Then, the Map operation averages, within each exon, the signal in each RNA-seq or ChIP-seq sample and constructs a query result, called *Genome Space*, which is the ideal starting point for subsequent data analysis steps; they are not covered by GMQL, but can be performed using classical tools, which are immediately applicable to GMQL output. For instance, these analyses may include building lists of top *k* genes with most similar expression to that of a given gene, or identifying groups of genes and their associated histone modifications and transcription factors with similar signal patterns, e.g. by using bi-clustering.

4) Find somatic mutations in exons

“*Consider mutation data samples of human breast cancer cases. For each sample, quantify the mutations in each exon and select the exons with at least one mutation. Return the list of samples ordered by the number of such exons.*”

```
Mut = SELECT(manually_curated__dataType == 'dnaseq' AND
                              clinical_patient__tumor_tissue_site == 'breast') HG19_TCGA_dnaseq;
Exon = SELECT(annotation_type == 'exons' AND original_provider == 'RefSeq')
                              HG19_BED_ANNOTATION;
Exon1 = MAP() Exon Mut;
Exon2 = SELECT(count_Exon_Mut >= 1) Exon1;
Exon3 = EXTEND(exon_count AS COUNT()) Exon2;
Exon_res = ORDER(exon_count DESC) Exon3;
MATERIALIZE Exon_res INTO Exon_res;
```

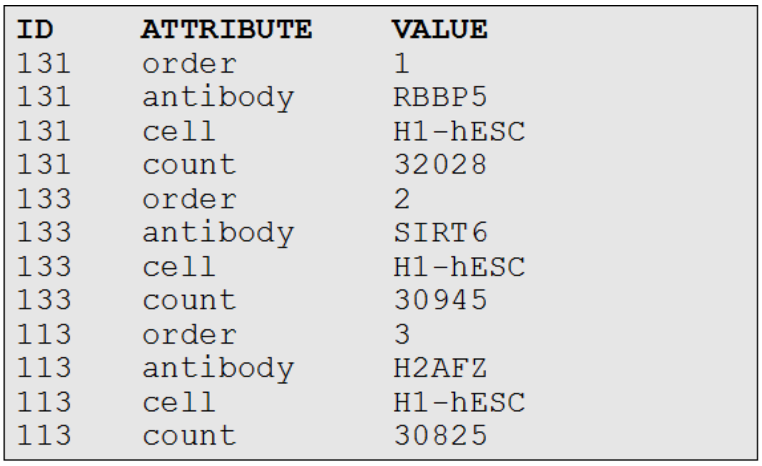
This fourth example shows that GMQL is very effective at counting, in batch on multiple samples, mutations that are mapped upon known regions (in this case exons) and extracting those regions having more mutations than a given threshold, ordering samples according to their number of regions extracted. Known human protein-coding and non-protein-coding exon regions are selected from *HG19_ANN*; such regions are provided by the UCSC database (https://genome.ucsc.edu/cgi-bin/hgTables), after retrieving them from the NCBI RNA reference sequences collection (RefSeq). Count of data sample items is performed by the Map and Aggregate operations. The Map counts mutations in each sample within each exon while mapping the mutations to the exon regions; after removal of the exons in each sample that do not contain mutations, the Extend counts how many exons remain in each sample and stores the result in the sample metadata as a new attribute–value pair. Note that, also in this example, the query is applied in batch on multiple data samples, i.e. all those samples selected from the *HG19_TCGA_Dnaseq* collection, which can be very many (in the case of the publicly available TCGA data, at the time of writing the available breast cancer patient samples were 963 for a total of 87,131 mutations). By applying this GMQL query to the curated somatic mutation data publicly available in TCGA from breast cancer patients and considering all 482,313 exon regions of 46,949 human protein-coding and non-protein-coding genes provided by RefSeq, at the time of writing it extracted 963 breast cancer patient samples with mutations involving 54,688 distinct exons of 41,364 genes.

5) Find top k samples

“*In ENCODE ChIP-seq sample, select gene promoters that intersect at least one ChIP-seq peak and extract the top 3 samples with the highest number of such promoters.*”

```
HM_TF = SELECT(dataType == 'ChipSeq') HG19_ENCODE_NARROW;
PROM = SELECT(annotation_type == 'promoter') HG19_BED_ANNOTATION;
PROM1 = MAP() PROM HM_TF;
PROM2 = SELECT(count_PROM_HM_TF >= 1) PROM1;
PROM3 = EXTEND(prom_count AS COUNT()) PROM2;
RES = ORDER(prom_count DESC; meta_top: 3) PROM3;
MATERIALIZE RES INTO RES;
```

This example uses a MAP operation to count the peak regions in each ENCODE ChIP-seq sample that intersect with a gene promoter (i.e., proximal regulatory region); then, in each sample it projects over (i.e., filters) the promoters with at least one intersecting peak, and counts these promoters. Finally, it extracts the top 3 samples with the highest number of such promoters.
The GMQL query was executed over 2,423 samples including a total of 83,899,526 peaks, which first were mapped to 131,780 promoters within the ANN annotation dataset, producing as result 29 GB of data; next, promoters with intersecting peaks were counted, and the 3 samples with more of such promoters were selected, having between 30K and 32K promoters each. Processing required 11 minutes and 50 seconds.
The RES result dataset includes both regions and metadata; the former ones indicate interesting promoter regions (that can be further inspected using viewers, e.g., genome browsers), the latter ones allow tracing provenance of resulting samples. Figure 2 shows 4 metadata attributes of the 3 resulting samples: the *order* of the sample, the *antibody* and *cell* type considered in the ChIP-seq experiment, and the promoter region *count*.



ID	ATTRIBUTE	VALUE
131	order	1
131	antibody	RBBP5
131	cell	H1-hESC
131	count	32028
133	order	2
133	antibody	SIRT6
133	cell	H1-hESC
133	count	30945
113	order	3
113	antibody	H2AFZ
113	cell	H1-hESC
113	count	30825

Figure 2. Metadata excerpt of the resulting samples.

6) Combining multiple replicate samples in different data formats

"*For all antibody targets of the K562 chronic myelogenous leukemia cell line in ENCODE, merge broad and narrow peaks in ChIP-seq replicate samples and calculate the average enrichment (signal) for each obtained peak.*"

```
HM_TF_rep_broad = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
                AND setType == 'exp' AND cell == 'K562') HG19_ENCODE_BROAD;
HM_TF_rep_narrow = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
                AND setType == 'exp' AND cell == 'K562') HG19_ENCODE_NARROW;
HM_TF_rep = UNION() HM_TF_rep_broad HM_TF_rep_narrow;
HM_TF = COVER(1, ANY; groupby: cell, antibody_target;
                aggregate: AVG_signal AS AVG(signal)) HM_TF_rep;
MATERIALIZE HM_TF INTO HM_TF;
```

Considering NGS experimental variability, replicates are usually performed and have to be taken into account in result evaluation, which can be done in multiple ways with different stringency. Thanks to the use of GDM and GMQL, this example illustrates how it can be easily done even when replicate samples are in different formats. All ChIP-seq peak samples in BROADPEAK or NARROWPEAK format from the ENCODE data collection that regard the K562 chronic myelogenous leukemia cell line are selected and included in a single unifying dataset. Then, multiple replicate samples, in case existing for an antibody target of the K562 cell line, are combined in a single sample including the disjoined DNA regions where at least one peak in the replicates exists. The average enrichment of the peaks in the replicates that contribute to each obtained region is calculated and assigned to such region.
When this example query was executed over 1,970 HG19_ENCODE_BROAD and 1,999 HG19_ENCODE_NARROW datasets, 130 BROADPEAK and 130 NARROWPEAK samples regarding 75 and 78 antibody targets, respectively, were selected, including a total of 4,566,008 and 4,426,212 peaks, respectively. After combining the replicates, 136 samples were obtained, containing a total of 5,121,711 regions regarding 136 antibody targets of the K562 cell line. Processing required 5.5 min.

7) Combining ChIP-seq and DNase-seq data in different formats and sources

“*Extract broad peaks of ChIP-seq transcription factor binding sites and histone modifications from ENCODE samples that intersect DNase-seq open chromatin regions from Roadmap Epigenomics in normal H1 embryonic stem cells.*”

```
CHIPSEQ = SELECT(dataType == 'ChipSeq' AND view == 'Peaks' AND setType == 'exp'
                                        AND cell == 'H1-hESC') HG19_ENCODE_BROAD;
DNASESEQ = SELECT(assay == 'DNase.hotspot.broad'
                                        AND Standardized_Epigenome_name == 'H1 Cells')
                                        HG19_ROADMAP_EPIGENOMICS_BED;
DNASESEQ1 = COVER(1, ANY) DNASESEQ;
CHIPSEQ_IN_DNASESEQ = JOIN(distance < 0; output: right) DNASESEQ1 CHIPSEQ;
MATERIALIZE CHIPSEQ_IN_DNASESEQ INTO CHIPSEQ_IN_DNASESEQ;
```

Combining data available, but in different formats and sources, this example shows how to improve the quality of ChiP-seq called peaks by filtering out those peaks that are not in open chromatin regions, where only they can be present biologically. For the same tissue, available ChIP-seq broad peaks from the ENCODE data collection, and DNase-seq open chromatin regions from the Roadmap Epigenomics Project, are selected. Multiple DNase-seq replicate samples in case existing are first combined in a single sample including all identified open chromatin regions, which are then joined with ChIP-seq peaks; only the peaks that at least partially overlap any of these regions are finally extracted. The join is performed for each of the selected ChIP-seq samples individually, so that each resulting sample is a selected ENCODE ChIP-seq sample, but including only the peaks that intersect open chromatin regions.

By executing this GMQL example query on 1,970 HG19_ENCODE_BROAD and 78 HG19_ROADMAP_EPIGENOMICS_BED input datasets, 90 ChiP-seq samples regarding 54 antibody targets and 1 DNase-seq sample were initially selected, including a total of 3,071,136 peaks and 412,042 regions, respectively. ChiP-seq called peaks finally obtained were 2,097,289 in total, regarding 54 different antibody targets. Processing was performed in 4.5 min.

8) Counting distinct DNA mutations in patient groups

"*Group patients by tumor type and ethnicity, and count the distinct DNA somatic mutations in each group.*"

```
MUTATION = SELECT(manually_curated__dataType == 'dnaseq'
                        AND manually_curated__tumor_tag == 'kirc') HG19_TCGA_dnaseq;
MUTATION_BY_RACE = COVER(1, ANY;
                        groupby: manually_curated__tumor_tag, clinical_patient__race;
                        aggregate: overlap_count AS COUNT(),
                        barcodes AS BAG(tumor_sample_barcode)) MUTATION;
MUTATION_COUNT = EXTEND(mutation_count AS COUNT()) MUTATION_BY_RACE;
MATERIALIZE MUTATION_COUNT INTO MUTATION_COUNT;
```

This example of GMQL query takes into account DNA-seq data of TCGA patients, groups samples by their tumor type and patient ethnicity, and for each ethnic group of every tumor type it extracts and counts its distinct DNA somatic mutations, counting for each of them the overlaps among the different samples (each sample is identified by its TCGA barcode). It is worth noting that, the COVER operator permits to extract the genomic regions with certain features (e.g., DNA mutations) in the considered samples, and for each extracted region the BAG operator collects the barcodes of the samples with genomic features in that region. Conversely, the EXTEND operator counts (through its COUNT() aggregate function) the number of distinct mutations in each resulting sample and stores it in the sample metadata; finally, the MATERIALIZE operator returns the obtained result. In particular, with the COVER operator extracts a sample for each tumor type and kind of patient race; the regions in the result samples are non-overlapping and are formed as contiguous intersections of at least one and at most any number of regions (i.e., somatic mutations) in the grouped input samples. For each result region, the COUNT aggregate function in the COVER operator computes the number of feature regions (i.e., mutations) that contribute to create the result region, and the BAG aggregate function gathers the TCGA barcode (identifier) of the sample of each contributing region to keep track of them. The metadata of each final resulting

sample are the union of the metadata of the samples in the input data set that regard the same tumor type and patient race, and are enhanced with the number of distinct mutations computed for the tumor type and patient race the sample is referring to.
For example, at the time of writing the number of TCGA DNA-seq data samples regarding the Kidney Renal Clear Cell Carcinoma (KIRC) was 235, and the result data set included three samples, one for each ethnic group represented in the KIRC TCGA data; the total numbers of overall DNA somatic mutations in the input samples were 1,971, 4,913, and 30,940 for the Asian, black or African American, and white race, respectively, and the number of samples for the three races were 7, 20, and 209, respectively, whereas the corresponding numbers of distinct somatic mutations in the result samples were 1,049, 1,070, and 3,227, respectively.

9) Combining different data types

"*Match DNA copy number variation (CNV) and microRNA (miRNA) data samples regarding the same biospecimen and extract the CNVs occurring within expressed miRNA genes in the paired samples.*"

```
CNV = SELECT(manually_curated__dataType == 'cnv'
             AND manually_curated__tumor_tag == 'luad') HG19_TCGA_cnv;
MIRNA_GENE = SELECT(manually_curated__dataType == 'mirnaseq'
             AND manually_curated__tumor_tag == 'luad') HG19_TCGA_mirnaseq_mirna;
CNV_GENE_0 = MAP(mirna_genes AS BAG(mirna_id);
             joinby: biospecimen_sample__bcr_sample_barcode) CNV MIRNA_GENE;
CNV_GENE = SELECT(region: count_CNV_MIRNA_GENE > 0) CNV_GENE_0;
MATERIALIZE CNV_GENE INTO CNV_GENE;
```

This example GMQL query searches and combines pairs of TCGA samples of Copy Number Variation (CNV) and miRNA-seq data types that regard the same biospecimen, and returns the DNA copy number variations in each CNV sample that are within expressed microRNA (miRNA) genes in the paired miRNA-seq sample. In particular, the MAP operator on CNV and miRNA-seq data sets first joins samples based on the equivalence of their metadata *biospecimen_sample__bcr_sample_barcode* attribute (the identifier for TCGA biospecimens); then, in each pair of samples the COUNT aggregate function calculates the number of miRNA genes overlapping each DNA copy number variation, and the BAG aggregate function collects the miRBase (http://www.mirbase.org/) IDs of such genes. Finally, the PROJECT operator selects only those copy number variations of the paired samples that overlap at least one expressed miRNA gene, and the MATERIALIZE operator returns the result. The resulting data set contains only those CNV samples, with their metadata, that have a matching miRNA-seq sample, and containing only their DNA copy number variations (at least one) that occur within an expressed miRNA gene in the matched miRNA-seq sample.
For example, at the time of writing the TCGA CNV and miRNA-seq data samples of Lung Adenocarcinoma (LUAD) patients were 1,141 and 504, respectively. The pairs of samples found regarding the same biospecimen were 496; 442 of them contained DNA copy number

variations within expressed miRNA genes of the same sample, with an average number of 146 copy number variations per sample.

10) Combining and comprehensive processing of patients' heterogeneous omics data

"*In TCGA data of Head and Neck Squamous Cell Carcinoma (HNSC) patients, find the DNA somatic mutations within the first 2000 bp outside of the genes that are both expressed and methylated in at least one biospecimen of these patients, and extract these mutations of the top three patients with the highest number of such mutations and their somatic mutations.*"

```
EXPRESSED_GENE = SELECT(manually_curated__dataType == 'rnaseqv2'
                AND manually_curated__tumor_tag == 'hnsc') HG19_TCGA_rnaseqv2_gene;
METHYLATION = SELECT(manually_curated__dataType == 'dnamethylation'
                AND manually_curated__tumor_tag == 'hnsc') HG19_TCGA_dnamethylation;
MUTATION = SELECT(manually_curated__data_type == 'dnaseq'
                AND manually_curated__seqPlatform == 'humanmethylation450'
                AND manually_curated__tumor_tag == 'hnsc') HG19_TCGA_dnaseq;
GENE_METHYL_0 = MAP(joinby: biospecimen_sample__bcr_sample_barcode)
                EXPRESSED_GENE METHYLATION;
GENE_METHYL = SELECT(region: count_EXPRESSED_GENE_METHYLATION > 0)
                GENE_METHYL_0;
GENE_METHYL1 = COVER(1, ANY) GENE_METHYL;
MUTATION_GENE = JOIN(distance < 2000, distance > 0; output: left)
                MUTATION GENE_METHYL1;
MUTATION_GENE_count = EXTEND(mutation_count AS COUNT()) MUTATION_GENE;
MUTATION_GENE_top = ORDER(mutation_count DESC; meta_top: 3)
                MUTATION_GENE_count;
MATERIALIZE MUTATION_GENE_top INTO MUTATION_GENE_top;
```

This example shows a GMQL query over numerous samples of multiple heterogeneous genomic features from the TCGA repository, seamlessly combined and comprehensively cross-evaluated, together with their clinical and biospecimen metadata, thanks to their availability in BED format (as provided by the TCGA2BED software - http://bioinf.iasi.cnr.it/tcga2bed/) and to the data model that GMQL uses. Comprehensively considering genomic, epigenomic and transcriptomic data of cancer patients can provide a better view of the patients' complex biomolecular system, which may lead to interesting findings. Leveraging on GDM and GMQL, this example presents how to do it focusing on expressed genes, DNA methylations (which generally repress gene expression), and the DNA somatic mutations close to methylated expressed genes. From the TCGA data collection in BED format, first all expressed gene, DNA methylation and DNA somatic mutation data of patients affected by Head and Neck Squamous Cell Carcinoma (HNSC) are selected. Then, by joining these heterogeneous data, the expressed genes with at least a DNA methylation are identified, and the DNA somatic mutations close to these genes are extracted. Thus, this query applies on RNA-seq, DNA-methylation and DNA-seq data of TCGA HNSC patients to find the DNA somatic mutations occurring within the first 2,000 bp

outside of the genes that are both expressed and methylated in at least one of these patients, and extracts these mutations of the top three patients with the highest number of such somatic mutations.
Specifically, the first JOIN operator applies on RNA-seq gene and DNA-methylation data sets. It first combines samples based on the equivalence of their metadata *bcr_sample_barcode* attribute (the TCGA biospecimen identifier); then, from every pair of samples of each biospecimen, it extracts the expressed gene regions that overlap at least a methylation site in the paired DNA methylation sample. Through the COVER operator all these gene regions are then merged in a single sample, which includes the genes both expressed and methylated in at least one of the TCGA HNSC patients. The second JOIN operator applies on this single sample data set and on the entire HNSC DNA-seq data set, and in each sample of the latter one it finds the DNA somatic mutations occurring within the first 2,000 bp upstream or downstream of any of the expressed methylated genes extracted. Then, the EXTEND operator uses the COUNT() aggregate function to determine the number of these mutations in each sample, the ORDER operator ranks the samples according to such number and extracts the top three samples with the highest number of these somatic mutations, and finally the MATERIALIZE operator returns the result.
At the time of writing the RNA-seq, DNA-methylation and DNA-seq samples of TCGA Head and Neck Squamous Cell Carcinoma (HNSC) patients were 294, 598, and 279, respectively. Applied on these samples, the described GMQL query found 271 DNA-seq samples with DNA somatic mutations close (within 2,000 bp) to the 11,307 genes which were identified as both expressed and methylated. These somatic mutations found in the three samples with the highest number of such mutations were 108, 45, and 36, respectively. The top three samples selected regarded white patients, who were current or current reformed smoker for 15 or less years, with age at the initial pathologic diagnosis of 69, 67 and 68 years, respectively, and presenting 95%, 87% and 95% of tumor cells, respectively.

The same GMQL query can be equally applied on other types of patients, for example Breast Invasive Carcinoma (BRCA) patients of the TCGA collection (just by changing the SELECT command parameter *manually_curated__tumor_tag* to be equal to *'brca'*), and to finally select a different amount of patients with the highest number of the identified mutations, e.g. the top five patients (just by setting the ORDER command parameter *meta_top* to *5*). In this case the execution of the GMQL query, over all the 9,217 HG19_TCGA_RnaSeqV2_Gene, 1,384 HG19_TCGA_Dnamethylation, and 6,361 HG19_TCGA_DnaSeq samples available in TCGA, initially selected 1,218 samples of gene expression data, 11 of DNA methylation data, and 993 of DNA somatic mutations of TCGA BRCA patients, containing a total of 24,986,052 expressed genes, 4,024,460 methylation sites, and 90,490 DNA mutations, respectively. The combination (through a GMQL Join operation) of each patient’s gene expression and DNA methylation data, modeled with GDM, identified 10 breast cancer patients presenting methylated expressed genes, with an average of 11,481 of such genes for each identified patient; these patients presented an average *age at diagnosis* of 57.80, an average *percent stromal cells* of 13.90%, an average *percent tumor nuclei* of 69.0%, and an average *tumor necrosis percent* of 0.11%, versus the average of 58.0%, 22.42%, 72.34%, and 6.62%, respectively, of all the initially considered patients with expressed gene data, based on the patients’ clinical data reported in the available sample metadata managed by GDM. Then, the query takes into account all and only the expressed genes methylated in at least one of the considered patients and, for each TCGA breast cancer patient with DNA somatic mutation data, extracts the mutations occurring within the first 2,000 bp outside of these genes (801 patients were found with such mutations). Finally, these mutations in each patient are counted (their average number per patient was 5.5) and the top 5 patients with the highest number of such mutations are selected. The table below reports an excerpt of the six most relevant metadata attributes and of their values associated with the five finally selected patients: the patient's *mutation count* and *order* within the patients with the highest number of mutations, the *age* of the patient *at* her breast cancer *diagnosis*, and the *percentage* of *stromal cells*, *tumor nuclei* and *tumor necrosis percent* in the evaluated patient's histological sample.

Metadata excerpt of the top five patients finally selected.

patient id	mutation count	order	age at diagnosis	percent stromal cells	percent tumor nuclei	tumor necrosis percent
a046	210	1	68	10	80	0
a23h	199	2	90	25	85	0
a0a6	150	3	64	24	75	20
a18g	89	4	81	5	90	0
a0t5	75	5	39	25	75	0

Leveraging on GMQL and TCGA data in BED format, this example query shows how to easily combine heterogeneous datasets to answer complex biomedical questions, such as to select DNA somatic mutations potentially relevant in altering the regulation of gene expression, which is generally repressed by DNA methylation. Thanks to both the availability in easy-to-use tab-delimited attribute-value text format also of the TCGA clinical and biospecimen metadata associated with the genomic data in BED format, as provided by the TCGA2BED software, and to the unique and innovative seamless management provided by GDM, and their seamless combined processing that GMQL uniquely and innovatively performs, the MUTATION_GENE_top result dataset includes both genomic somatic mutations and clinical metadata of the finally selected patients. The former ones indicate interesting mutations that could be associated with breast cancer (which can be further inspected using viewers, e.g., genome browsers); the latter ones allow tracking the provenance of resulting samples and ease the biomedical interpretation of the results, facilitating also result sample stratification and further evaluations. This association between processed genomic data and their biological/clinical metadata is not supported by any other system currently available, and represents one of the new relevant aspects of GDM and GMQL.

11) Mapping pairs of genes on Topologically Associating Domains (TADs)

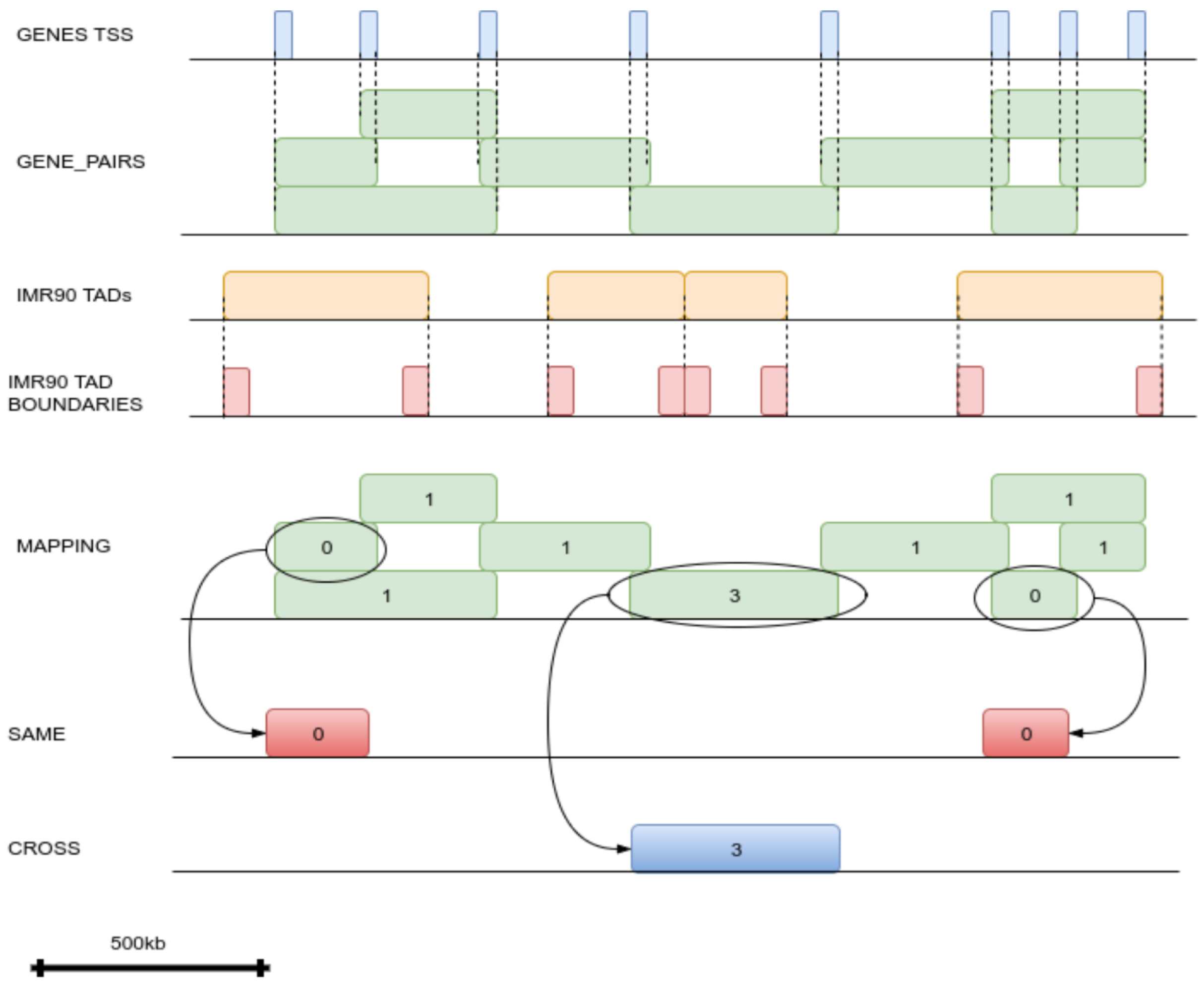
We are given a dataset of genomic regions TADS (Topologically Associating Domains). These represent regions of the genome in which physical interactions are stronger than

outside of them. We are also given a set of genes with their coordinates (TSS) in the dataset GENES.
The two required output datasets represent the following:

- The SAME set will contain all the pairs of genes that belong to the same TAD;
- The CROSS set will contain all the pairs of genes that belong to two different TADs.

The maximum distance for the computation of the gene pairs is set to 500kb.

```
IMR90_TADs = SELECT(cell==”imr90”) TADS;
GENE_PAIRS = JOIN(DLE(500000); output:CONTIG) GENES GENES;
IM90_TADs_boundaries_1 = PROJECT(stop AS start + 1) IMR90_TADs;
IM90_TADs_boundaries_2 = PROJECT(start AS stop - 1) IMR90_TADs;
IM90_TADs_boundaries = UNION() IM90_TADs_boundaries_1 IM90_TADs_boundaries_2;
IM90_TADs_boundaries = MERGE() IM90_TADs_boundaries;
MAPPING = MAP() GENE_PAIRS IMR90_TADs;
SAME = SELECT(left.gene_name < right.gene_name AND
                                        count_GENE_PAIRS_IMR90_TADs == 0);
CROSS = SELECT(left.gene_name < right.gene_name AND
                                        count_GENE_PAIRS_IMR90_TADs > 1);
MATERIALIZE SAME INTO same;
MATERIALIZE CROSS INTO cross;
```



The query works as follows:

1. We extract from the TADs dataset the sample regarding the cell imr90.
2. We build the new dataset GENE_PAIRS by joining the TSS coordinates using the CONTIG clause. We select only the gene pairs having a distance less than 500kb.
3. We extract the boundaries of the TADs, which are correspond to the beginning and ending base of the regions. This can be done with a sequence of two PROJECT statements. We finally put together all the boundaries using a UNION and a MERGE (this last operation is used for compacting the dataset to only one sample).
4. We map the TADs boundaries to the gene pairs regions and we count how many times a TAD boundary is present between every gene pair
5. The SAME dataset is obtained by filtering those gene pairs which do not intersect any TAD boundary (i.e., they belong to the same TAD)
6. The CROSS dataset is obtained by filtering those gene pairs which intersect at least 2 TAD boundaries (i.e., they belong to two different TADs.

This query enables us to get the SAME and CROSS sets with also the names of the genes involved in the couples. This is useful when we want to use expression datasets like GTEx or TCGA for calculating the correlation between the genes inside and outside the topological domains.