UE NGS - Polymorphism

Thibault Latrille, Carine Rey & Marie Sémon

December 16, 2019

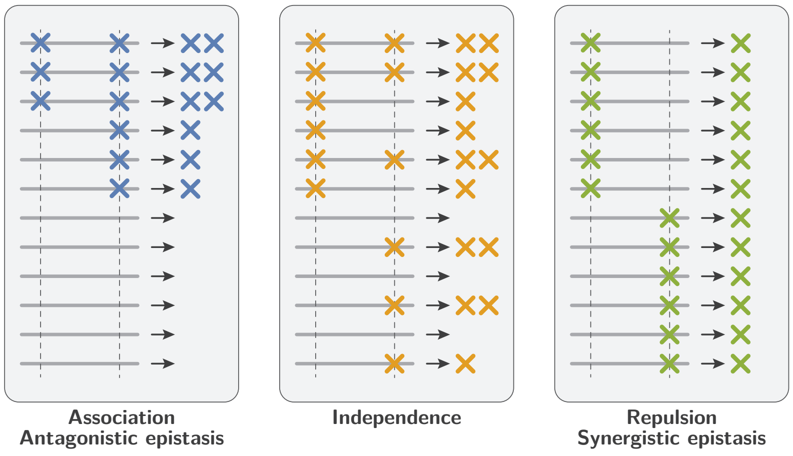
Contents

Introduction

The materials are available at `https://github.com/ThibaultLatrille/tp-polymorphism`

Negative, or purifying selection prevents the unlimited accumulation of deleterious mutations and establishes a mutation-selection equilibrium. Because of the difficulty of ascribing precise selection coefficient to different mutations, the selective coefficient ascribed to one individual can be approximated by the total number of deleterious mutations encountered in this individual. Under the null hypothesis of no epistasis, selection on different mutations acts independently, so that each additional mutation causes the same decline in fitness. By contrast, if synergistic (or repulsive) epistasis between deleterious alleles is present, each additional mutation causes a larger decrease in relative fitness, and we expect not to see many individuals carrying multiple mutations. On the other hand, if antagonistic (or associative) epistasis between deleterious alleles is present, each additional mutation causes a lower decrease in relative fitness, and we expect to see many individuals carrying multiple mutations.

In this TP, our goal is to investigate if we can detect epistasis at the whole genome scale in humans. In other word, we want to know if the negative effect of a deleterious mutation is dependent on whether one already has other deleterious mutations ? Can we detect and quantify this effect of dependence in humans using NGS dataset ?



Step by step, we first need to find a dataset containing mutations of individuals belonging to a human population (section 1), then we need to filter out any mutations that is not inside coding sequences (section 2). Since each mutation don't have the same fitness effect, we need to filter the mutations depending on their functional effect (section 3). Finally, we will do a little of mathematics to compute epistasis at the genome level with our filtered dataset and investigate whether the null hypothesis of no epistasis is valid (section 4).

1 Polymorphism dataset

In this first section, we will download a polymorphism dataset and display it.
Required data: a variant call file (VCF)
Required program: *IGV*

Genetic polymorphism is the occurrence in the same population of two or more alleles at one locus, each with appreciable frequency. One project that investigated human genetic diversity and polymorphism is the 1000 Genomes project. We will use their public dataset for our analysis, which can be downloaded from the project website (`http://www.internationalgenome.org/`). The dataset we want to download must contain information about the genotype of all individuals, not solely the frequency in the population. Moreover we will use the GRch38 human assembly to be compatible with other independent dataset we will use afterward. So the first task is to find which VCF file(s) to download that satisfies this constrains.

To understand and visualize the dataset, we first need to uncompress the file(s) (vcf.gz). Then we can use the program IGV already installed to display the VCF, and we can use a text editor (geany or less) to view the VCF file.

2 Filtering out non-coding polymorphism

In this section, we will filter the variant call file (VCF) to discard all the polymorphisms that are not inside coding sequences.
Required data: a variant call file (VCF) and gene annotation file (GTF)
Required program: *bedtools*, *gtf_to_bed.py* and *IGV*

To filter polymorphism, we can use their position in the genome and assert if they fall inside a coding sequence. We thus need to find a file containing the start and stop position of all coding sequence, which data can be found in the Ensembl database (`https://www.ensembl.org/`) in form of a gene annotation file (GTF). Since we downloaded the VCF (section 1) aligned to the GRch38 human assembly, we must download the GTF accordingly. Once downloaded and uncompressed, the file can be displayed with a text editor.

One issue is that this file contains many more information than we need, and we need to extract only the relevant information (chromosome, start, stop and transcriptId). We can then use the custom script *gtf_to_bed.py* to extract only tracks corresponding to consensus coding sequences (CCDS), the scripts output the result in a BED format.

What constrain on the length of coding sequences must be satisfied by all tracks ? Can we implement a a filter inside *gtf_to_bed.py* that assert that this constrain is satisfied for all CCDS.

At this step, we can use *bedtools intersect* to filter the VCF file by using the BED file. But we want to keep the information about the transcript identifiers that each polymorphism falls in. The problem is that the BED file contains transcript tracks that might be overlapping, and we won't know on which transcript falls each polymorphism, since a polymorphism can fall in several transcripts at the same time. To solve this conflict, we can use the *bedtools merge* command to merge overlaps in a pre-sorted BED file before running the *bedtools intersect* command on the merged BED file to filter the VCF.

3 Synonymous, Non-Synonymous and Stop polymorphism

In this first section, we will determine if a polymorphism is synonymous, non-synonymous or stop.
Required data: a variant call file (VCF), a gene annotation file (GTF) and sequences files (FASTA)
Required program: *vcf_coding_polymorphism.py*

To assert if a polymorphism is synonymous (no change of amino-acid sequence), non-synonymous (change of amino-acid sequence) or stop (introducing a stop codon inside the sequence), we need to know the reference codon and the reference encoded amino-acid, and compare it to the amino-acid encoded by the mutated codon once we apply the mutation. To this aim we need the DNA sequences of all the transcripts to be able to retrieve the reference codon. Moreover, we need the gene annotation of all transcripts to convert absolute position (in the genome) to relative position (in the sequence) of a polymorphism. We already downloaded in the first part of the TP the gene annotation in Ensembl, and we need to download the transcript sequences (FASTA) in Ensembl (`https://www.ensembl.org/`).

4 Analysis

In this section, we will analyze the synonymous, non-synonymous and stop dataset with two simple statistics V_A and σ^2 in the different populations.
Required data: synonymous, non-synonymous or stop variant call files (.vcf). File linking each individual to its population (.panel file).
Required program: *vcf_meta_analysis.py*

$V_A = \sum_i Var(X_i)$ and $\sigma^2 = Var(\sum_i X_i)$, where X_i is a discrete variable that represents the number of derived alleles present at locus i and can take values 0 (not present), 1 (heterozygous) or 2 (homozygous).

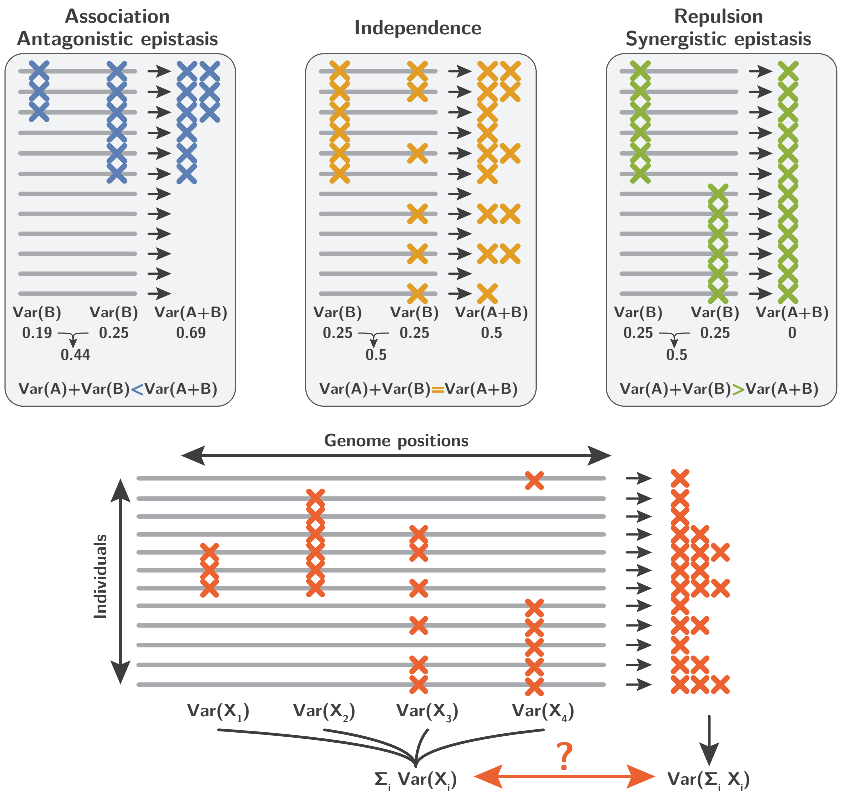
- If $\sigma^2 = V_A$, the deleterious mutations are independent of one another.
- If $\sigma^2 < V_A$, the deleterious mutations are synergistic (repulsive).
- If $\sigma^2 > V_A$, the deleterious mutations are antagonistic (associative).

To perform this analysis in the VCF file, we need to code the computation of V_A and σ^2 in the script. The script use a VCF file that is converted to an array, where the first dimension is the number of polymorphism and the second dimension is the number of individuals. From this array, we need to compute V_A and σ^2, this can be done in python using the conjunction of the function *var()* and *sum()* provided by the package *numpy*.

A panmictic population is one where all individuals are potential partners. This assumes that there are no mating restrictions, neither genetic nor behavioural, upon the population, and that therefore all recombination is possible. The human population in its whole might not be panmictic, how does that will affect the analysis? Will we tend to see more association or repulsion of mutations if the population is structured in sub-population? Which human sub-population could we use that might be closer to panmixia?

Instead of filtering the VCF file by population, we will use the VCF of the whole human population on the custom script *vcf_meta_analysis.py* that will compute V_A and σ^2 for each sub-population. In the 1000 Genomes project website (http://www.internationalgenome.org/), we thus need to find the file (.panel extension) that link each identifier (individual) to its population.

What can you conclude from this analysis, is there more association or repulsion? Is epistasis detectable ?



5 Data Format

5.1 VCF

Reference: http://www.internationalgenome.org/wiki/Analysis/vcf4.0
The Variant Call Format (VCF) is a text file format (most likely stored in a compressed manner). It contains meta-information lines, a header line, and then data lines each containing information about a position in the genome. There is an option whether to contain genotype information on samples for each position or not.

5.2 GTF

Reference: https://www.ensembl.org/info/website/upload/gff.html
The Gene transfer format (GTF) is a text file format used to hold information about gene structure. It is a tab-delimited text format based on the general feature format (GFF), but contains some additional conventions specific to gene information.

5.3 BED

Reference: https://www.ensembl.org/info/website/upload/bed.html
The Browser Extensible Data (BED) file is a tab-delimited text file used to hold information about genomic regions. The lines of a BED file that describe a genomic region have three required fields (chromosome, start, end). Optionally, columns containing a name for the region, a score and the strand orientation (+/-) can be added.

5.4 Fasta

Reference: https://zhanglab.ccmb.med.umich.edu/FASTA/
FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column.

6 Tools and scripts

6.1 IGV

Reference: http://software.broadinstitute.org/software/igv/book/export/html/6
The Integrative Genomics Viewer (IGV) is a high-performance visualization tool for interactive exploration of large, integrated genomic datasets. It supports a wide variety of data types, including array-based and next-generation sequence data, and genomic annotations.

6.2 bedtools

Reference: http://quinlanlab.org/tutorials/bedtools/bedtools.html
bedtools is a swiss-army knife of tools for a wide-range of genomics analysis tasks. The most widely-used tools enable genome arithmetic: that is, set theory on the genome. For example, bedtools allows one to intersect, merge, count, complement, and shuffle genomic intervals from multiple files in widely-used genomic file formats such as BED and VCF. While each individual tool is designed to do a relatively simple task (e.g., intersect two interval files), quite sophisticated analyses can be conducted by combining multiple bedtools operations on the UNIX command line.

6.3 gtf_to_bed.py

gtf_to_bed.py is a custom script in python3 that convert a GTF file into a BED file, extracting only tracks corresponding to consensus coding sequences (CCDS). The script also filter out CCDS that are badly annotated (termination not confirmed, exons consisting of one nucleotide). Moreover, the script provide a simple framework to implement easily new filters.
Example usage:

```
$ python3 gtf_to_bed.py --help
$ python3 gtf_to_bed.py --gtf <gtf_file>
```

6.4 vcf_coding_polymorphism.py

vcf_coding_polymorphism.py is a custom script in python3 that compute the effect of polymorphisms on coding sequence and takes as input a VCF file, a FASTA file and a GTF file. Each polymorphism can be either synonymous, non-synonymous or stop. The script split the input VCF file into 3 files, respectively Syn.filename.vcf, NonSyn.filename.vcf and Stop.filename.vcf.
Example usage:

```
$ python3 vcf_coding_polymorphism.py --help
$ python3 vcf_coding_polymorphism.py --gtf <gtf_file> --vcf <vcf_file> --fasta <fasta_file>
```

6.5 vcf_meta_analysis.py

vcf_meta_analysis.py is a custom script in python3 that takes as input a VCF file containing genotype data and the panel file assigning identifier (individual) to their population. For each population, the script compute $V_A = \sum_i Var\left(X_i\right)$ and $\sigma^2 = Var\left(\sum_i X_i\right)$, where X_i is a discrete variable that represents the number of derived alleles present at locus i and can take values 0 (not present), 1 (heterozygous) or 2 (homozygous).
Example usage:

```
$ python3 vcf_meta_analysis.py --help
$ python3 vcf_meta_analysis.py --stop <vcf_file> --syn <vcf_file> --nonsyn <vcf_file> --panel <panel_file> --cutoff <f>
```