Computational identification of specific mutations in DGRP and wild drosophila lines

Internship report

Prepared by
VERBRUGGHE Maëwenn and
HOLGUIN Santiago
Bachelor's Degree in Biology and Computer Science

Laboratory: NeuroPsi
Team: Daniel VASILIAUSKAS - Drosophila Neurogenetics
Co-supervisor: Guillaume POSTIC

Année : 2023-2024

Abstract

Genomics research has long faced major challenges in accurately identifying SNPs and other genetic polymorphisms.

The Drosophila Genetic Reference Panel (DGRP) (Mackay *et al.*, 2012, Huang, Massouras, *et al.*, 2014), comprising 205 inbred lines of wild-caught Drosophila, constitutes a very important dataset for the study of Drosophila genetic diversity. However, despite its usefulness for studies of associations between variants and their characteristics (GWAS), the understanding and above all the precise detection of certain specific polymorphisms is insufficient.

Currently, 7 mutations causing specific phenotypes in Rhodopsin expression in the adult Drosophila retina have been identified in these lines. Detection of these mutations is therefore crucial for their study, as well as for the study of affected populations. However, existing computational methods are unable to identify and retrieve these genetic variants within Drosophila lines with complete accuracy.

The aim of this study is therefore to develop a computerized tool for searching genomic sequencing reads of Drosophila lines for variants responsible for known phenotypes.

The algorithm we have developed, consisting of 3,262 lines of code, enables the 7 mutations to be detected very precisely in their reference line. It also revealed the presence of the *Rh5[Q365]* substitution in RAL509, the DGRP line for the *melt[RAL509]* deletion, which had not previously been detected by DGRP algorithms. This underlines the accuracy of our algorithm in detecting these 7 mutations.

In the future, our algorithm could be applied to the analysis of these mutations in wild lines from Europe and around the world. Initially, this research will make it possible to calculate prevalence according to the geographical location of the mutations, and eventually to trace the evolutionary history of these mutations while studying their possible impact within populations.

Table of contents

List of figures and tables

Figures :

Tables :

Thanks

First of all, we'd like to express our deepest gratitude to all those who have contributed to the smooth running of this internship and project.

We would like to thank our internship manager, Daniel VASILIAUSKAS, for welcoming us to his team, and for his guidance and invaluable advice throughout the internship.

We would also like to thank all the team members for their welcome, help and cooperation during our internship.

A big thank you to Guillaume POSTIC, our co-worker on this project and head of the course, for supporting us throughout our studies and once again during our internship.

Last but not least, thanks to Carène Rizzon, our sector manager, who has been with us for several years and helped us with invaluable advice.

VERBRUGGHE Maewenn and HOLGUIN Santiago.

I. Introduction

Identifying SNPs (Single Nucleotide Polymorphisms), and other sequence polymorphisms, from genomic data has been a major challenge in bioinformatics and genetics over recent decades. Powerful algorithms developed by researchers now make it possible to study natural genetic variations at population level, and their impact on individuals, both in humans and in other species. These approaches attempt to identify actual sequence differences between the sequenced genome and the reference genome, while rejecting differences resulting from sequencing errors and artifacts. Consequently, this process does not identify all genetic variants. For example, it does not reliably identify clues. It is also possible to identify natural genetic variants from a phenotype using a mixture of classic and modern mapping techniques. However, these variants are often not among those identified purely by computational methods. Our aim, as described below, is to identify experimentally-discovered Drosophila variants from sequenced genomes.

The Drosophila Genetic Reference Panel (DGRP) is a resource created to study quantitative traits in Drosophila (Mackay *et al.*, 2012, Huang, Massouras, *et al.*, 2014). It is a collection of 205 inbred lines derived from individual mated females, wild-caught in Raleigh, North Carolina, USA. After 20 inbred generations, these lines were fully sequenced and ~4.5 million variants present in these lines were identified and annotated using computational tools. Over 140 articles have been published on DGRP lines. Typically, a quantitative phenotype of interest is measured and used to perform a genome-wide association study (GWAS). Genes containing phenotype-associated variants are tested by RNAi knockdown. In general, the actual variants, i.e. the specific genetic changes in the DGRP lines that result in the phenotypes of interest, are not identified or studied.

The Neurogenetics of Drosophila team in the NeuroPSI laboratory is interested in the natural genetic variants that affect rhodopsin gene expression patterns in color photoreceptor neurons in Drosophila (Rister et al., 2013). Insects like Drosophila possess panoramic vision of the world around them, mainly because they have three types o f eyes: compound eyes, ocelli and

eyelets. Compound eyes are made up of 800 individual eye units called ommatidia. Each ommatidia contains eight photoreceptor (PR) neurons that express 5 rhodopsins (G-protein-coupled receptors that detect photons) with different spectral sensitivities. Together, they capture and detect light information from a wide range of wavelengths. Each ommatidium contains 8 PRs of 3 types. 6 PRs (R1- R6) are analogous to human rods. They express Rh1, sensitive to a wide range of wavelengths, and are responsible for motion detection and low-light vision. The "inner" PRs, R7 and R8, are analogous to human cones. They are located one above the other and collect visual information from the same point in space. R7 and R8 are responsible for color vision. The Rhs expressed by the PRs R7 and R8 define two subtypes of ommatidia: p and y, normally present in a 1:2 ratio and stochastically distributed in the eye. pR7 and pR8 express UV-sensitive Rh3 and blue-sensitive Rh5 respectively. The yR7 and yR8 types express a different UV-sensitive rhodopsin, Rh4, and a green-sensitive Rh6, respectively.

This Rh expression results from a stochastic p vs. y decision in the R7 PRs that is then communicated to the R8 PR, ensuring coordinated rhodopsin expression (Rister et al., 2013). Although this differentiation logic is well understood, many aspects of the molecular pathways involved in the specification and maintenance of Rh expression are not known. To answer this question and, at the same time, understand the impact of natural genetic variation on Rh expression pattern, our team set out to target the 205 DGRP lines for Rh5/Rh6 expression phenotypes. Rh5/Rh6 were used because their expression lies at the end of the differentiation cascade, and their visualization can reveal perturbations in every event of RP differentiation and maintenance. The study uncovered a surprisingly large number of lines with unusual Rh expression. Subsequently, 7 genetic variants causing some of these phenotypes were identified through candidate testing, recombination mapping and other techniques (Table 1). Other variants are currently being mapped.

Of these 7 identified variants, 3 were not correctly identified by the variant search algorithms. And, at present, we have not been able to identify a tool that can directly search for variant sequences in genomic sequencing reads.

This would enable us, for example, to analyze these mutations in wild Drosophila populations. It would also enable us to find traces o f evolution and

of adaptation within different populations, highlighting the impact of these mutations on populations.

Here, we focus on identifying reads carrying specific variants within DGRP lines, using a computational approach. We will then use the same methods to study wild Drosophila lines from different regions of Europe and the world.

We have therefore developed an algorithm capable of identifying specific mutations in a high-throughput sequenced genome. First, we demonstrate the validity of this algorithm using lines carrying the identified variants. Next, we will discuss the results of our algorithm applied to other DGRP lines, in order to identify mutations that may have escaped computational analysis. Finally, we will apply our algorithm to unknown Drosophila lines from different geographical regions.

II. Materials and methods

High-throughput sequencing

The genomes of the DGRP lines we will be using are the result of high-throughput sequencing. This technique involves fragmenting purified DNA into pieces of around 100 nucleotides, amplifying them and sequencing them using different methods. The result is a list of reads, each corresponding to a fragment. Then, using alignment algorithms, the reads are aligned with the reference sequence of the species studied, enabling the reconstruction of most of the sequence of the sequenced genome.

NCBI tools

Using DGRP and NCBI tools, we were able to access files containing sequencing reads from all 205 DGRP lines. What's more, the interface allows you to view all reads aligned to the reference sequence, enabling you to identify and verify the presence of mutations within each line.

Natural variants

To date, 7 genetic variants causing abnormal Rh5/Rh6 expression have already been identified, 6 in DGRP lines and 1 from another source. Ongoing work aims t o identify 2 or 3 additional variants over the next few years. Collaboration with other teams will also enable us to identify variants affecting other phenotypes, such as behavior. The mutations already characterized are listed in Table 1. This table contains the most important data on each mutation/variant, i.e. gene, name, description, sequence, notes, coordinates, line where the mutation was identified, variant name given by DGRP, and DGRP notes.

GENE	CG	Variant name (what we call it)	Variant description	Variant sequence (lower case) in its sequence context (upper case)	notes	Release 6 coordinates (by BLAST)	Release 5 coordinates (obtained by BLAST in R6 and converted by me) - I don't trust the precision of DGRP calls	Identified in	DGRP name of the variant	DGRP annotation
melted	CG8624 FBgn0023001	melt[RAL509]	214 bp deletion	GCCAGCTCTTTTCATTTCATGTTTTTTTTTTA GCG[illegible] ACGGAAAACAGAgccGCCGAATCATTTTCGCA GAATTTCTGGCCGGCCATACGGTGCAGATTC TT	the 3 bp deletion (gcc) that follows the 214 bp deletion is present in many lines and is likely irrelevant	2L:7138228..7138441	2L:7,131,328..7,131,541	DGRP line 509	not called	not annotated
Rh5	CG5279 FBgn0014019	Rh5[delta26]	A 26-bp deletion in the first exon within the open reading frame introduces a frame shift that results in termination of the ORF after 27 codons	CGCCCACTGGCGGGGATTCCGTGAGGCACC TATCTACt[illegible]TATCGTCC TCATGCTGTCGAGTATCTTTGGAAATGGA		2L:12008599..12008624	2L:12,008,599..12,008,624	DGRP lines 559, 646, 8[illegible]	2L_12333832_DEL *note: DGRP only called this for RAL 832, but not the other 3 lines, and even for RAL 832 did not identify the variant sequence correctly*	FBgn0014019|Rh5|FRAME_SHIFT|0
Rh5	CG5279 FBgn0014019	Rh5[Q365*]	C to T resulting in Q365*	(reference: c / variant: t) followed by AGGAGAGTGTGGCCAGTGTCAGCGGCGATA CTTTGGCAC	the polymorphic base is preceded by another C in the reference genome which is commonly (and silently!) mutated to T among DGRP lines	2L:12009649	2L:12,009,649	DGRP lines 397, 589, 759	2L_12009649_SNP	FBgn0014019|Rh5|STOP_GAINED|0
Rh6	CG5192 FBgn0019940	Rh6[Edn]	A deletion removes ATGTTCACC resulting in three amino acid deletion in the 2nd TM domain	GCACGCCGTCCAACATGTTTGTGGTCAATCT GGCCTTCTCGGACTTCATGATGatgttcaccATG TTCCCGCCGGTGGTGCTGAATGGCTTCTAT GGCACCTGGATCATGGGTCCCTTCC	Alan Bergland (2015): "I looked very quickly for this deletion but don't see it in our datasets that have indel calls. These sets are mostly focused on North American populations and it is possible it is a European or Scottish specific polymorphism. Or it is exceedingly rare."	3R:15163584..15163592	3R:11,309,306..11,309,314	Identified among flies derived from Edinburgh, Scotland from an outbreed stock donated by Darren Obbard to San Diego stock center	not DGRP	NA
Rh6	CG5192 FBgn0019940	Rh6[RAL551]	23 bp deletion that removes, resulting in a frame shift after Trp16 which then continues for 115 aa until a STOP codon.	TCTCAGCCTGGCGGAGAGTGTGCCCGCCGA GATTATGCACATGGTGGATCCGTACTGGTAC CAGTGGCCTCCATTGGAGCCCATGTGGTtcgg [illegible]CCATTCTGGGCACCATGT CGCTGGCGGGCAACTTCATTGTGATGTACAT CTTTACCTCGTCGAAGGGACTGCGCACGCC GTCCAACA	is clearly in many sequence reads of the original sequencing of the line, i.e. present at the time the line was sequenced and remained stably segregating	3R:15163435..15163457	3R:11,309,157..11,309,179	Identified as a segregating variant in RAL551 line. Very approximately a quarter of flies showed no Rh6 immunoreactivity.	not called	NA
CG9150	CG9150 FBgn0031775	nja1	A deletion removes CAAC resulting in an immediate STOP	CCTCGCGGGTACACCAAATCACGCCCATTAC gttgGTGTCTATGACCTCCTTAAGCTTCTGCG TA	The sequence here is sense for genomic, but inverted+ antisense for the gene	2L:6068232..6068235	2L:6,068,232..6,068,235	DGRP lines 738, 892	2L_6068230_DEL	FBgn0031775|CG9150|FRAME_SHIFT|0
CG8757	CG8757 FBgn0036360	nja2	Two deletions 22bp and 37bp separated by 6 bp of preserved sequence.	CCAGACCCACCACTATCATTCCTGCTCCGAT [illegible]GCTCC G[illegible]ACCAACGCTCCATGTT ACCGTTCCAC	The sequence here is sense for genomic, but inverted+ antisense for the gene	3L:13842121..13842157, 13842164..13842165	3L:13,835,221..13,835,257, 13,835,264..13,835,265	DGRP lines 93, 176, 908	3L_13835220_DEL & 3L_13835263_DEL	FBgn0036360|CG8757|FRAME_SHIFT|0 FBgn0036360|CG8757|FRAME_SHIFT|0

Table 1: Natural Variant Table. Table of the 7 identified mutations with the lines that have been identified as carrying the mutation.

Of the seven mutations, only one is an SNP (single nucleotide substitution); the others are deletions ranging from 4 to 214 bp. Future mutations may also include insertions, inversions and translocations. Our algorithm will therefore be able to handle all these types of genetic variants.

Of the seven mutations identified among DGRP lines by our team, three were not identified by the variant identification algorithm used by the DGRP consortium (Table 1). For example, the *melted[RAL509]* and *Rh5[delta26]* mutations are either not recognized or misrecognized, respectively. This represents the fundamental problem of this project: how to find known variants among genomic sequences read *de novo*, without relying on the lists of variants generated by the variant identification algorithm.

Spring web

To meet this need, we decided to develop our own algorithm. We chose to introduce the algorithm within a web server, so that it could be used in different environments. We therefore used Spring Web, a Java framework capable of creating robust and simple web servers. This Framework will introduce each request received into a complex pipeline that returns reads containing mutations, and reads containing the wild sequence without mutation.

API USCS

In order to limit what the user has to type, avoid errors and maintain a similar structure between our mutation classes, we have chosen to recreate the mutant and wild-type sequences within our code, using the mutation positions. This required us to find a reliable source of reference sequences. Several options were available, but we chose to use the "sequence" endpoint of the USCS API. An endpoint is a specific URL in an API that can be accessed via the Internet and represents a particular resource. Here's an example of the API request we used : https://api.genome.ucsc.edu/getData/sequence?genome=dm3;chrom=chr3L;start=7131328;end=7131541. This query provides the sequence from position 7.131.328 to 7.131.451, located on chromosome 3 of Drosophila melanogaster.

JPA, Hibernate and databases

Calculating the prevalence of mutations requires taking into account all existing results in order to calculate the distribution of mutations. This is why we found it useful to develop a database. This enabled us to group together the different alignments we had carried out and to save the data more easily. The biggest problem in creating a database is persistence and transforming data efficiently, accurately and flexibly to avoid losing information. This prompted us to use Hibernate, a solution that enables direct transformation from Java classes to SQL data. Hibernate is capable of creating and populating databases, as well as obtaining all kinds of data. This is ideal when the data is complex, and needs to retain good logic and structure.

To represent our mutation objects, Hibernate has created 5 tables - Insertion, Inversion, Deletion, Multiple Mutation, Substitution - which you can see in the following entity-association diagram (Figure 1).

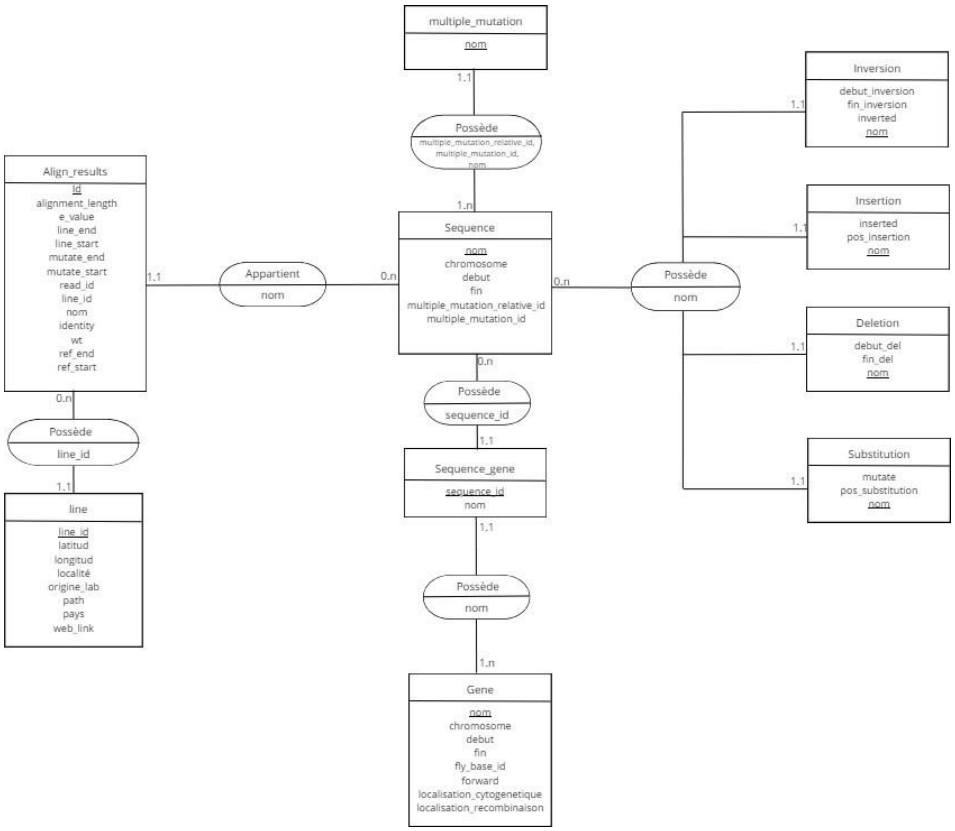


Figure 1: Database entity-association diagram.

With Hibernate, each class has a repository, which uses the class itself and a unique primary key to make the connection with SQL. This repository is then implemented in a Spring Service (MutationsServices, LinesServices and ResultServices), and can be used by our pipelines to add and retrieve data at any time. Our server acts as an intermediary interface between the database and the user.

To express the biological problem of mutations, we declared a data structure. To represent the chosen structure, we have drawn up a class diagram. This structure enables us to deal with several mutations at the same time, although there are of course certain differences between the different types of mutation.

React and interface

As you can see, this program requires a lot of information and has several types of data input. On the other hand, this computer tool is intended for biologists who may not be familiar with programming. Therefore, to make the use of the program more attractive and intuitive, we thought it would be useful to develop a simple graphical interface (see interface screen in figure 2).

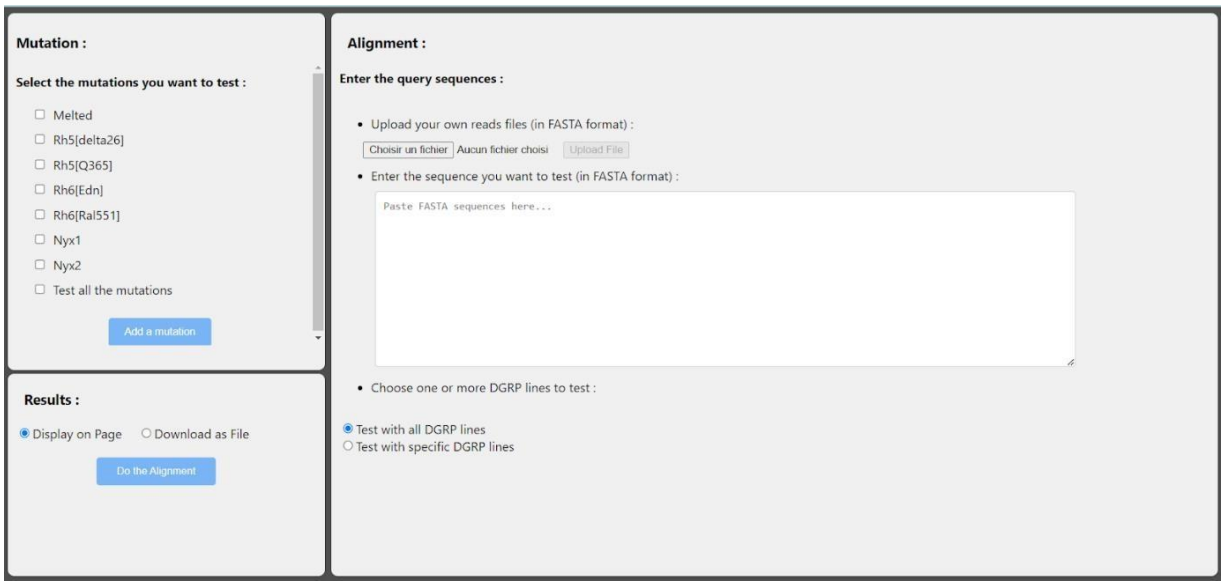


Figure 2: Graphical interface.

This interface was developed using react, a JavaScript library used to create web interfaces. It allows users to interact with several web components in a simple, visual way, using text fields, checkboxes, etc., making the program easy to use and accessible to biologists.

This interface is therefore composed of a "mutation" section allowing the user to select the mutations to be tested. The default setting is the 7 mutations already found. The user can then select the mutations he wishes to test from among those available, or add new mutations by supplying the required information.

Once the mutations to be searched for have been selected, the user must indicate the lines or sequences with which he wishes to perform the alignment. The user can download a reads file in FASTA format from his directory and add this file to the database by simply pressing the "Add to DataBase" button. He can also simply paste a small sequence in FASTA format into the space provided. Finally, he

can choose to test one or more DGRP lines, which are already installed, by specifying the line name or choosing to align with all DGRP lines.

For the results section, the user can choose to view the alignment results directly in the interface, or to receive them as a file, with the option of specifying the output file name.

Mutant research models

Our problem was largely based on the treatment of mutations. In our code, a mutation cannot simply be represented by a sequence of nucleotides. In order to process each mutation, we had to separate them according to type, and add important characteristics for each one. We therefore decided to create the Sequence class, which represents the general characteristics of a mutation.

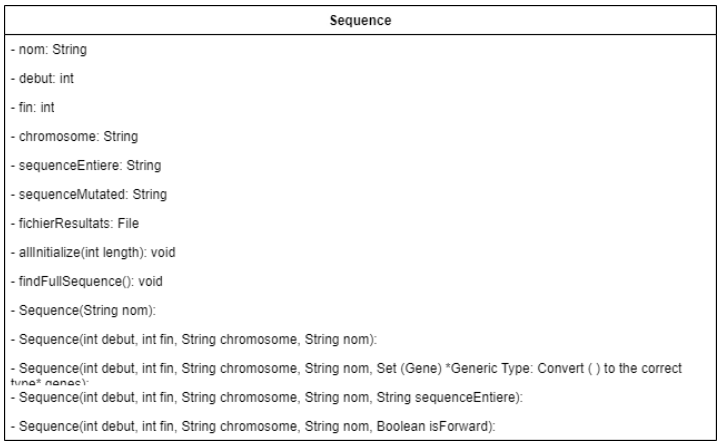


Figure 3: UML diagram of the Sequence class

Once we had defined this abstract class, we defined the 5 types of mutations (Deletion, Insertion, Inversion, Substitution and MultipleMutation), with their specific characteristics. Each mutation is built according to certain constraints: for example, deletions have 3 reference sequences, while the rest of the mutations we deal with have only 2. However, each mutation features which you can see in Figure 4.

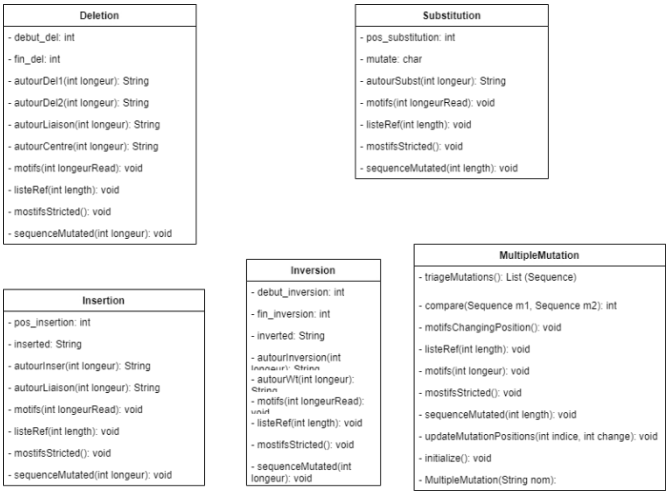


Figure 4: UML diagram of mutation classes

Multiple mutations correspond to a sequence of mutations leading to a particular sequence. For example, the nyx2 mutation is a sequence of 2 deletions separated by 6 bp (see Table 1). In our algorithm, a multiple mutation is composed of a list of mutations that must be given in order.

We have also defined the Line class, allowing us to access all line information in a single object, Figure 5. Of the same type, we have defined the Gene class, which will allow us to know which genes are affected by a certain mutation, Figure 5. These classes allow us, in part, to save information concerning the provenance of mutated lines, as well as the impact of mutations within the Drosophila genome.

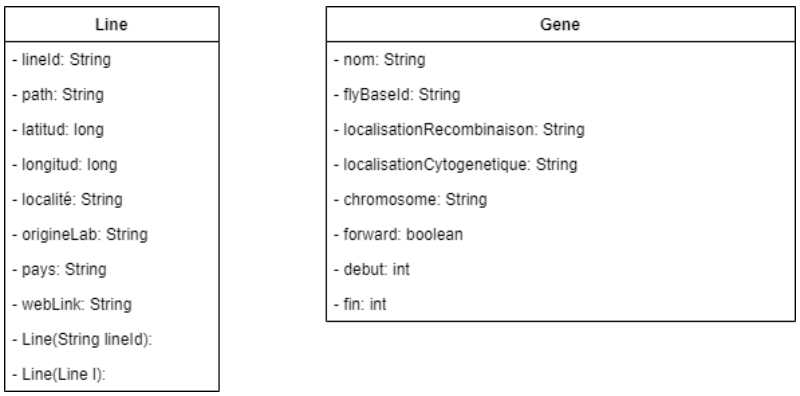
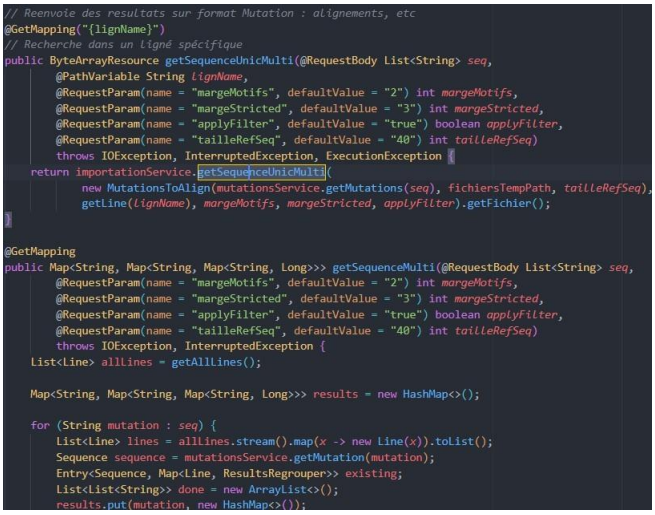


Figure 5: UML diagram of Line and Gene classes

Mutant research controller (query reception)

The role of a controller is to receive a request and redirect it into the right pipeline. First, we have all the information and communication requests between the user and the database: add a lineage and its file, view all lines, view all mutations, view details of a given mutation, view details of a lineage and view all mutations, all detailed.

We decided to create two types of data processing requests. After receiving a list of sequences and a specific line id, the first of these requests sends a file in the form of an array of bytes. The second sends a HashMap containing the number of mutated read matches and the number of wt read matches, for any lineage, after receiving a list of mutations that will be aligned to all existing lineages (Figure 6).



```
// Reenvoie des resultats sur format Mutation : alignements, etc
@GetMapping("{lignName}")
// Recherche dans un ligné spécifique
public ByteArrayResource getSequenceUnicMulti(@RequestBody List<String> seq,
        @PathVariable String lignName,
        @RequestParam(name = "margeMotifs", defaultValue = "2") int margeMotifs,
        @RequestParam(name = "margeStricted", defaultValue = "3") int margeStricted,
        @RequestParam(name = "applyFilter", defaultValue = "true") boolean applyFilter,
        @RequestParam(name = "tailleRefSeq", defaultValue = "40") int tailleRefSeq)
        throws IOException, InterruptedException, ExecutionException {
    return importationService.getSequenceUnicMulti(
            new MutationsToAlign(mutationsService.getMutations(seq), fichiersTempPath, tailleRefSeq),
            getLine(lignName), margeMotifs, margeStricted, applyFilter).getFichier();
}

@GetMapping
public Map<String, Map<String, Map<String, Long>>> getSequenceMulti(@RequestBody List<String> seq,
        @RequestParam(name = "margeMotifs", defaultValue = "2") int margeMotifs,
        @RequestParam(name = "margeStricted", defaultValue = "3") int margeStricted,
        @RequestParam(name = "applyFilter", defaultValue = "true") boolean applyFilter,
        @RequestParam(name = "tailleRefSeq", defaultValue = "40") int tailleRefSeq)
        throws IOException, InterruptedException {
    List<Line> alllines = getAlllines();

    Map<String, Map<String, Map<String, Long>>> results = new HashMap<>();

    for (String mutation : seq) {
        List<Line> lines = alllines.stream().map(x -> new Line(x)).toList();
        Sequence sequence = mutationsService.getMutation(mutation);
        Entry<Sequence, Map<Line, ResultsRegrouper>> existing;
        List<List<String>> done = new ArrayList<>();
        results.put(mutation, new HashMap<>());
```

We've represented the pipeline with a sequence diagram in Figure 7. This diagram is just a representation and doesn't give an exact idea of the algorithm, but it does give a first impression of how it works. Indeed, it describes how and in what order several objects work together

The combination of technologies we chose enabled us to draw on the strengths of each language: the performance of C to process a large amount of data in a very short time, and the efficiency of Java to make our code robust and facilitate the management of https requests.

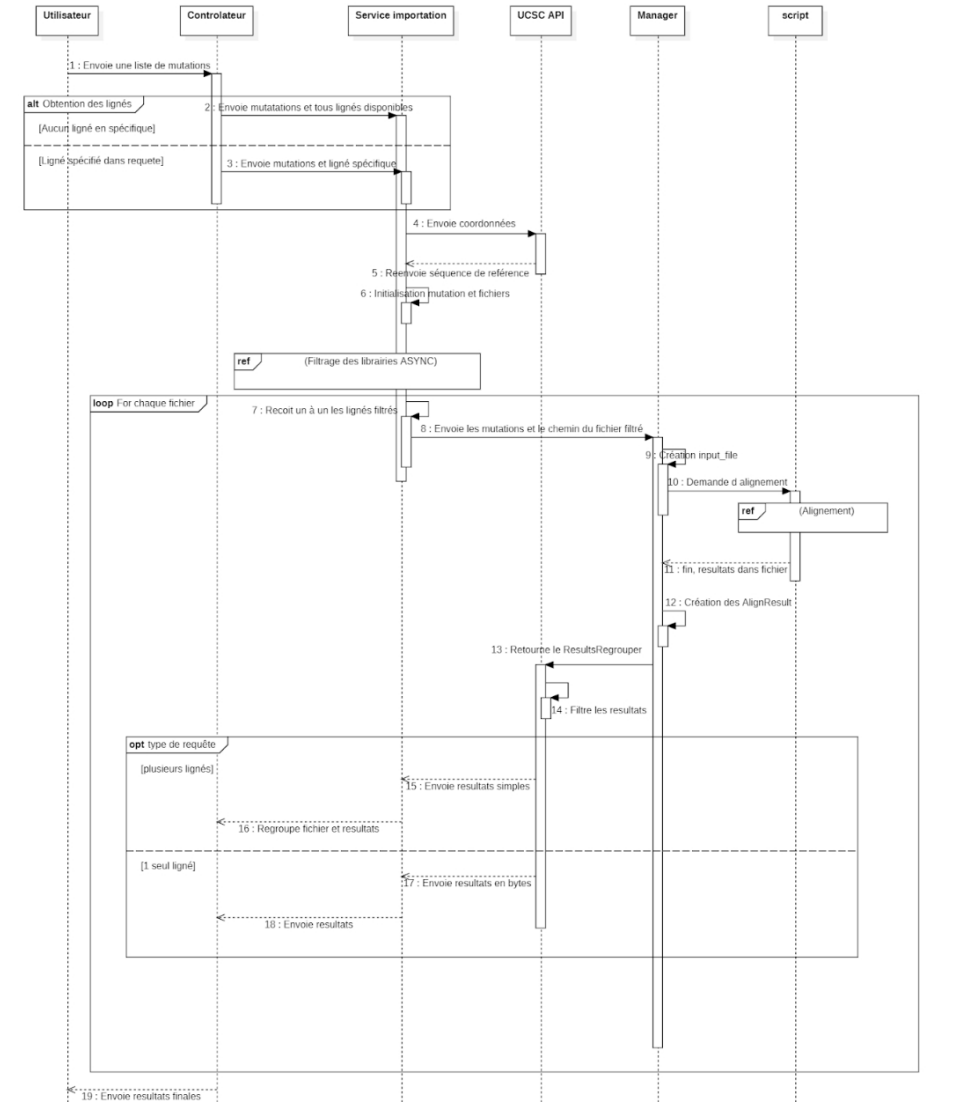


Figure 7: Sequence diagram.

Obtaining reference sequences

The first step in our code is to obtain the reference sequences, i.e. the sequences with which our reads will be aligned. Referential sequences are obtained as a function of mutation positions. Figure 8 shows an example of this mechanism, with a schematic representation of our deletion strategy.

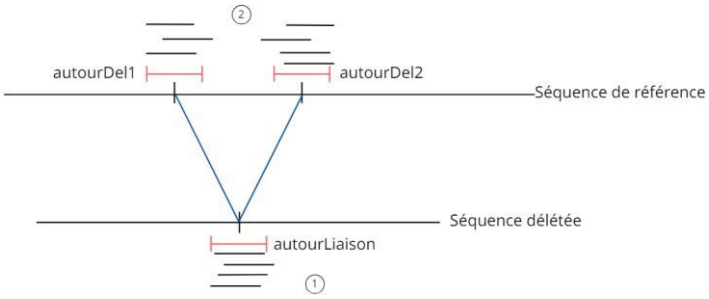


Figure 8: Schematic representation of deletion reference sequences.

As you can see, we have 3 referential sequences, 2 at reference sequence level, i.e. 1 referential sequence for the start position of the deletion and one for the end position of the deletion. The third referential sequence is is located at the level of the mutated sequence, and more precisely at the position of the deletion, we have named it aroundLiaison.

For other types of mutation, we don't necessarily always need several wt reference sequences, e.g. for substitutions, inversions and insertions, we only need 2, one wt and one mutated. In all cases, the mechanism is the same, i.e. the reference sequences are determined according to the position of the mutations (see diagram in appendix x).

Service import and filtering

Spring allows us to create Services that encapsulate the core logic of the application, providing methods to perform specific tasks, such as computation, data manipulation or transaction coordination. The main role of a service is to process data received from controllers, communicate with repositories or other services, and return the necessary results. Centralizing business logic in services promotes reusability, testability and separation of concerns, making code more organized and maintainable. Here we've created the Import service, whose role will be to import the reference data mentioned later, and then launch a series of operations to retrieve the mutations.

Having obtained the referential sequences, we can, in theory, align all reads with each of these sequences. However, given that files can contain several million lines to align, depending on the number of referential sequences, this process is extremely time-consuming. What's more, not all reads are necessarily relevant to the study, as most of them correspond to other genomic regions that are not relevant to the study.

are not part of the mutation. To optimize this step as much as possible, we set up a filtering system consisting of two searches for specific motifs of interest.

For deletions, there are a total of 6 so-called Stricted motifs (3 forwards and 3 inverted-complementary). For the three forwards motifs, one of the strict motifs corresponds to a sequence of 10 nucleotides surrounding the position of the deletion in the mutated sequence. This sequence comprises 5 nucleotides on either side of the deletion position (see Figure 9). The other two strict motifs are centered on the start and end positions of the deletion in the reference sequence. Each of these motifs also corresponds to a sequence of 10 nucleotides, i.e. 5 nucleotides on either side of the start and end positions of the deletion. These three strict motifs are then transformed into their complementary, inverted sequence, creating three new strict motifs.

Reverse and complementary Stricted motifs were added at the end after analysis of the results without filtering. Indeed, with filtering, we found far fewer mutated reads. Then we noticed, thanks to the NCBI alignment, that some reads were inverted and complementary. This is why we added this type of pattern.

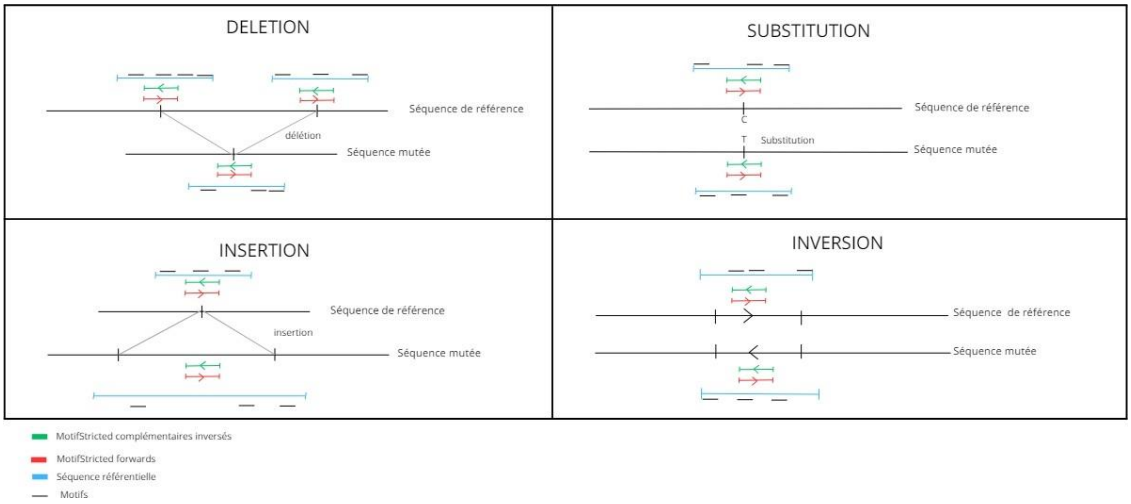


Figure G: Diagram of referential sequences, motifs and motifStricted. Schematic representation of referential sequences, random motifs and motifSticted as a function of the type studied. (blue: motifs, red: motifStricted, black: random motifs, green: motifStricted reversed and complementary.

There are also 5-nucleotide motifs obtained from the previously obtained reference sequences (see Figure 9). 6 motifs are selected for insertions, inversions and substitutions, 10 for deletions and 8 for multiple mutations (such as Nyx2, Table 1). We have chosen to allow a small margin of error to account for

small, silent mutations that may occur. If the margin of error is exceeded for all patterns, the read is not saved. Filtering is performed first with strict patterns, then with random patterns. The margin of error for the first filter is 2 mismatches and for the second only one.

Ssearch36

Our algorithm relies on the use of two key tools to operate: a reads filter and an alignment algorithm such as Smith-Waterman. Initially, we opted for a Java implementation of all our code, including the filter and the Smith-Waterman algorithm, which we found in open source at the following address
: github.com/DulanDias/SequenceAlignment.java.
Although we got good results with this implementation, it took between 30 and 60 minutes to run when we tested a mutation on a file, which was far too long.

So, to improve the performance of our algorithm, we decided to switch to a compiled language like C, an imperative programming language that allows complex tasks to be done in a short time. We therefore used Pearson's Smith-Waterman implementation, accessible via the following link: fasta.bioch.virginia.edu/wrpearson/fasta/fasta36, where we used the latest version, i.e.
v36.3.8.tar.gz, which gave us access to the ssearch36 executable file.

Ssearch36 is a high-performance implementation of the Smith-Waterman algorithm. Unlike global alignment (such as the Needleman-Wunsch algorithm), which aligns the entire length of the sequence, Smith-Waterman, which is a local alignment, aligns regions with strong similarities to each other. This method is very useful for identifying small conserved regions in sequences.

Alignment

Alignment is performed by Aligner, a class that is created prior to filtering thanks to parameters: mutations to be aligned, temporary folder path, and a mutation service. Aligner features the getSimpleResults method, which receives a Line (reads file) and where the first step is to copy the file with the filtered reads into a new file called output_file.fasta. This file is then aligned with the file containing the reference sequences, thanks to Ssearch36, which gives the results in tabulated form in the file results.tsv. This latter file will be read, and with each line of it we'll create an instance of AlignResults, the whole of which will be returned as a ResultsRegrouper.

Mutant research models: processing results

In order to group the mutations chosen by the user and maintain the same logic for each mutation, we have created the Mutations To Align class. This class will initialize and register the mutations to be tested, saving the data in files containing the reference sequences for each mutation, as well as the pattern files required for filtering.

To represent alignment results, we've chosen the AlignResults class and entity, which will be built from the tabulated results returned by SSearch 36 (option
-m 8). This class implements the Comparable interface, which enables a comparison to be made between different result lines. This comparison is generally made between two reads that work, one wt and one mutated, then it always favors the result with the higher score. And, in the case where the alignments have the same score, we study the number of gaps, then the number of mismatches. Then, if all these are equal between the two alignments, we retain the result where the read is aligned with a mutated referential sequence.

This class also has a score() method, which will also provide a score result calculated as follows*: score = identity * (alignmentlength / alignmentmaximumlength) - gaps*, i.e. the percentage of normalized identity in relation to the alignment length minus the number of gaps (not taken into account in the % identity calculation). AlignResults also features a method that determines whether the alignment contains a zone of interest: a zone where a mutation has occurred.

Mutant research models: filtering results

The class that groups all alignment results for a given line and Sequence is called ResultsRegroup. The first role of this class is to separate the results into two: those that are significant in relation to the other results and have a good score (or are in an interesting zone); and those that don't respect any of these characteristics. Once separated, the significant results are added to a "matching" area. The other "waste" results are added to the "not matching" area. In the case of an existing "matching" result (an alignment with wt and one with mutated for the same read), the two alignments are compared as explained above, the worst result of the two is added to "waste" and the other is kept or added to "matching".

Conclusion on the pipeline

This section describes and details all the components of the mutanteReasearch program. All parts of the code have been tested both independently and within the program. In Figure 10, you can see a summary of the pipeline stages in the form of an activity diagram.

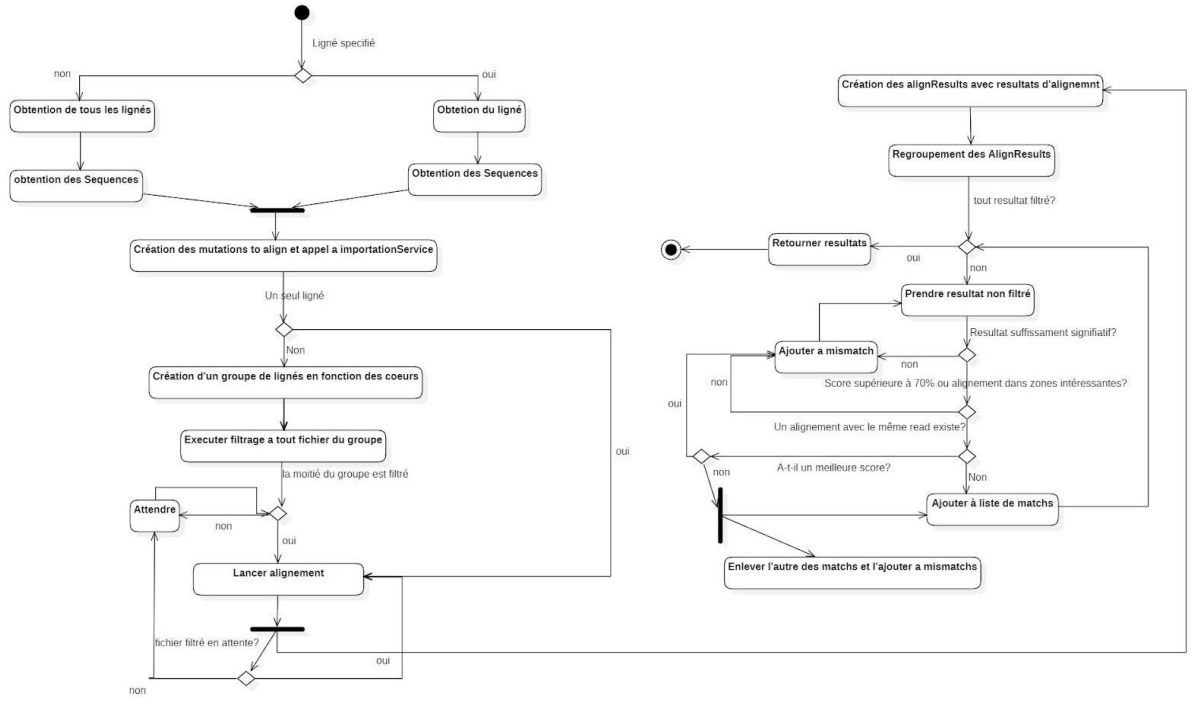


Figure 10: Activity diagram.

III. Results and discussion

The central topic is the analysis of the results we have obtained with our algorithm. We'll also be making comparisons with other algorithms we've developed that were less successful.

In this first part, we will discuss the implementation of our code for each type of mutation. We'll also show how accurate our algorithm is, and how these tests have enabled us to verify its validity through validation by alignment analysis provided by NCBI.

Solution implemented: Deletions

The melted mutation is a 204 bp deletion. To successfully identify the deletions, we came up with the following strategy (Figure 11):

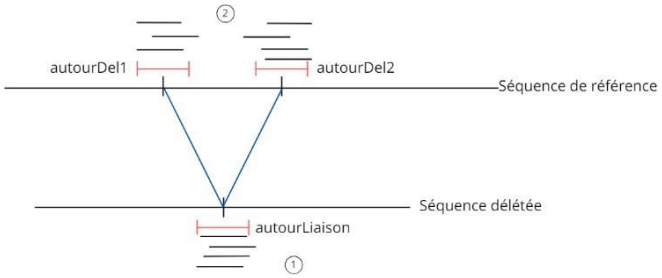


Figure 11: Schematic representation of the strategy used for deletions.

If we are in case 1, i.e. we find reads aligned with the reference sequence aroundLaison, the reads are mutated, meaning that the line has the deletion. Conversely, if we are in case 2 and obtain reads wt, i . e. aligned with the reference sequences aroundDel1 and/or aroundDel2, this means that the reads are not mutated and that the line does not have the deletion.

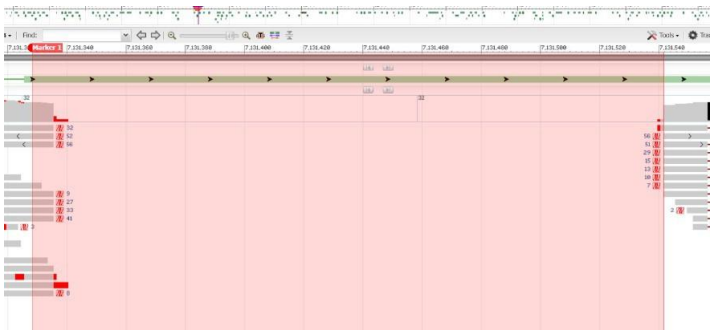


Figure 12: RAL50G reads aligned to the reference sequence for the melt[RAL50G] deletion positions. Screen capture from NCBI.

Figure 12 shows the alignment of reads from the RAL509 line with the reference sequence at the deletion positions (red marker). As we can see, despite the fact that it's very easy to see the deletion visually, the DGRP algorithms don't match it. detect it in this way, even though it's there.

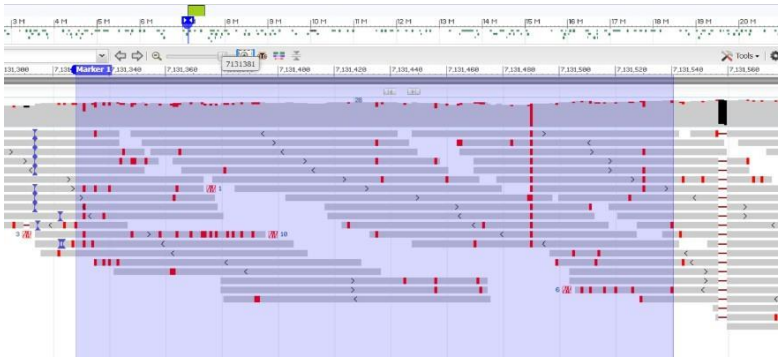


Figure 13: Reads of RAL7G0 aligned to the reference sequence for the positions of the melt deletion [RAL50G]. NCBI screenshot.

This time, in Figure 13, we observe the RAL790 line at the position of the *melt[RAL509]* deletion. As you can see, there are no deleted reads. We therefore expect to obtain only wt reads and no matches for the mutation.

For the RAL509 line, we succeeded in obtaining 16 mutated reads, i.e. 16 matches for the mutation and 0 wt. This means that the reads were aligned with the sequence aroundLiaison (case 1, Figure 11) and that RAL509 is indeed mutated. Conversely, for the RAL790 line, we obtain 24 wt and 0 matches, meaning that the reads have been aligned with the sequence aroundDel1 or aroundDel2 (case 2, Figure 11).

We then have small deletions, ranging from 4 bp to 26 bp, such as *nyx1*, *Rh5[delta26]*, *Rh6[RAL551]*, *Rh6[Edn]*, using the same strategy as for the *melt[RAL509]* mutation.

Figure 14: RAL8G2 reads aligned to reference sequence for nyx1 deletion positions. Screen capture from NCBI.

In this section, we'll take a closer look at *nyx1*, the most extreme of the small deletions we're working on (4 bp). In Figure 14, you can see the reads of RAL892, the reference line for this mutation, aligned at the *nyx1* deletion. As before, even if the deletion is visually present for us, for other software tools, the deletion is not considered a mutation.

Using our algorithm, we obtained the following results: 13 mutated reads and 0 wt. Indeed, visually, we can see that more than 13 reads have the deletion. In

Comparing the IDs of the 13 reads found by our code with those observed, we confirmed the accuracy of our algorithm. Despite this, there is a loss of information in the reads that should be found as matches.

Figure 15: RALG3 reads aligned to reference sequence for nyx1 deletion positions. Screen capture from NCBI.

that no read is mutated.

For the RAL93 line, the reference line for the *nyx2* mutation, we obtained 0 matches for the *nyx1* mutation and 16 wt. We can therefore conclude that these are good results, since we know that this line does not possess the mutation. Furthermore, Figure 15, which shows the alignment of reads from this line at the *nyx1* mutation level, confirms that this is a good line.

Solution implemented: Substitutions

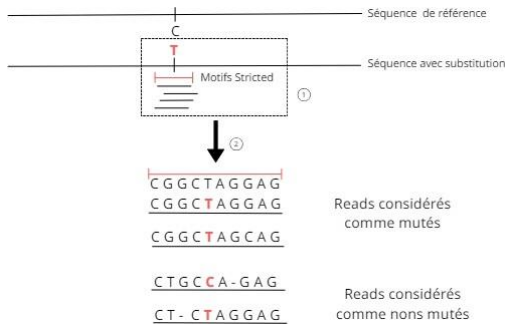


Figure 16: Schematic representation of the substitution strategy.

For substitutions (Figure 16), we have 2 reference sequences (see **Obtaining reference sequences**, Materials and methods section). After the reads have been filtered and aligned, in-depth checks of the alignment results are carried out, enabling us to verify that, at the substitution position, we have a T and not a C. Reads with a T are considered mutated if they

comply with the various score thresholds.

For the reference line for this mutation, RAL790, we obtained the following results

Figure 17: RAL7G0 reads aligned to reference sequence for Rh5[q365] deletion positions. Screenshot from NCBI.

8 matches for the mutation and 0 wt. Figure 17 shows the *Rh5[q365]* mutation for the RAL790 line. As you can see, there are generally 2 substitutions in a row, the first

substitution is silent and does not influence the phenotype, unlike the second substitution, which is the one we're looking for.

For the other lines tested, which are not expected to possess this mutation, we obtained good results, with only wt and no match for the mutation. However, one result in particular caught our attention. For line RAL509, we obtained

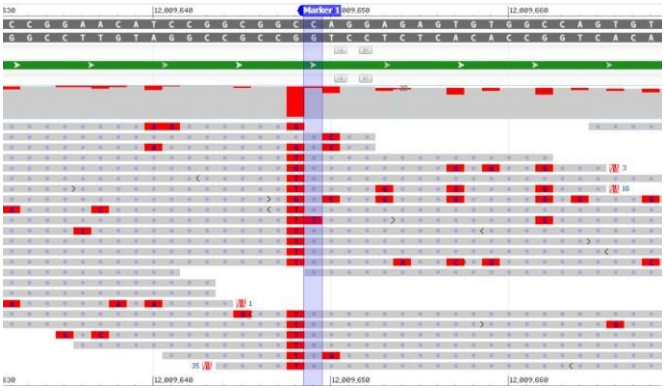


Figure 18: RAL50G reads aligned to reference sequence for Rh5[q365] deletion positions. Screenshot from NCBI.

we obtain a match for the mutation, meaning that a read possesses the substitution. We then checked with the alignment provided by NCBI and obtained Figure 18, which shows that a read of RAL509 is mutated for *Rh5[q365]*.

This last result is proof of two things: the mutantResearch program is capable of finding both large and small polymorphisms. And, above all, it is capable of giving more precise and accurate results than the polymorphism detection algorithm used by DGRP. For them, line 509 was not a mutated line for *Rh5[q365]*, but we were able to show that there is indeed a read with this mutation.

Solution implemented: Multiple mutations

We now turn to the implementation used for multiple mutations. Each mutation involved in the multiple mutation is treated independently according to its type.

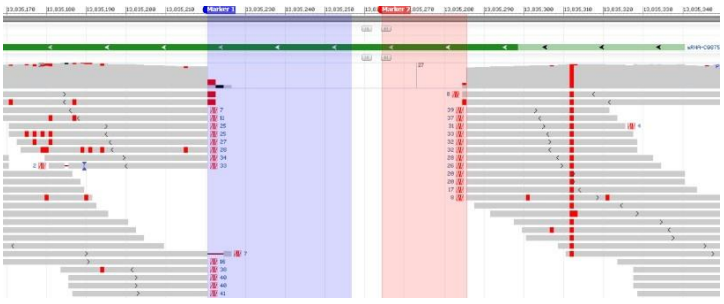


Figure 1G: RALG3 reads aligned to reference sequence for nyx2 deletion positions. Screenshot from NCBI.

Figure 19 shows the deleterious part of the RAL93 line, the reference sequence for the *nyx2* mutation, with the 37 bp deletion in blue and the 22 bp deletion in red. Our algorithm's multipleMutation class enables us to correctly

build the mutated referential sequence, using only the coordinates and type of each "small" mutation.

The results provided by our algorithm show that the 6 other lines tested do not have the *nyx2* mutation. In particular, for line RAL790 (Figure 20), we found 16 wt and 0 mutations. Conversely, for line RAL93, the reference sequence for this mutation we obtain 12 mutates and 0 wt. This result is perfectly in line with expectations and demonstrates that the algorithm is also capable of finding mutations. multiple.

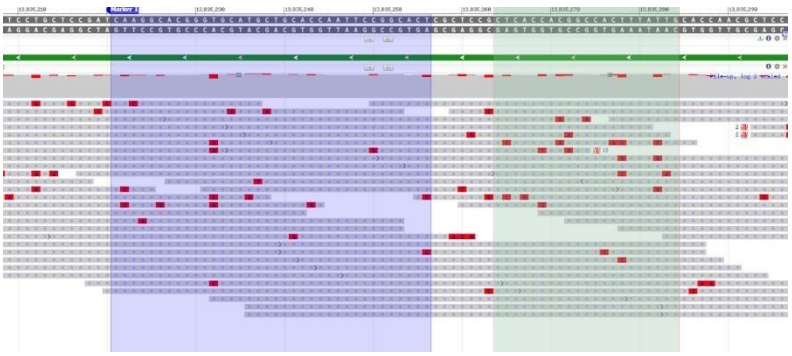


Figure 20: Reads of RAL7G0 aligned with the reference sequence for the position of the nyx2 deletion. NCBI screenshot.

Despite the satisfactory results, we note that in some cases we have a slight loss of information. For example, for the nyx1 deletion we can see in figure 14 that 15 reads have my mutation, but our algorithm returns only 13. Or for RAL790, we find 8 matches whereas we see on NCBI that there are more.

At first, therefore, we thought that filtering might result in a loss of information. But after checking, we found that the results, with and without filtering, were very similar (see section **Filtering comparison**).

What's more, there are sometimes good alignments between certain reads and all types of referential sequences, both mutated and wt. That's why we've added filtering to the results. This filtering allows us to keep only the very good alignments: those with a good alignment score and being significantly correct. Although filtering is performed for all types of mutations, we found that it was particularly essential for small mutations, such as deletions (*nyx1*) or substitutions (*Rh5[Q365]).* Indeed, without filtering, some reads were both aligned to wt referential sequences and mutated, resulting in a loss of information, in both cases, in terms of number of matches and wt. Thanks to the result filtering we've implemented, even if reads are aligned several times, there's no risk of losing results and missing a mutated read.

or, conversely, to falsely consider a read as mutated when a better wt alignment exists (see **Mutant research models,** Materials and methods section**).**

Thanks to this filtering, we were able to obtain good results for all types of small deletions, such as RAL646, the reference line for *Rh5[delta26]*, or line 892, the reference for nyx1. Filtering also enabled us to highlight the presence of the *Rh5[q365]* substitution in line RAL509.

Even if these results are correct and rather satisfactory, the fact of not being able to identify mutations in inverted reads may be limiting. It would therefore be interesting to implement additional processing to handle both inverted and forwards reads at the same time.

We think it would also be possible to use the strategy we use for substitutions to deal with small deletions. In particular, this could reduce alignment and filtering time. The number of referential sequences would be reduced from 3 to 2, and the number of motifs used from 10 to 6. In extreme examples, this would help avoid bad matches: too many SNPs at the very beginning and very end of alignments with reads can influence the score and therefore the result for small deletions.

We have not been able to test the performance of this algorithm for insertions and inversions, as no mutations of this type have yet been identified. However, we use a strategy similar to those presented above (see Figure 21). The red sequences represent the reference sequences to which the reads will be aligned. In each of the mutations, if the reads are aligned as in case 1, there is no mutation. Conversely, if the results returned imply that the reads are aligned as in case 2, the mutation is present.

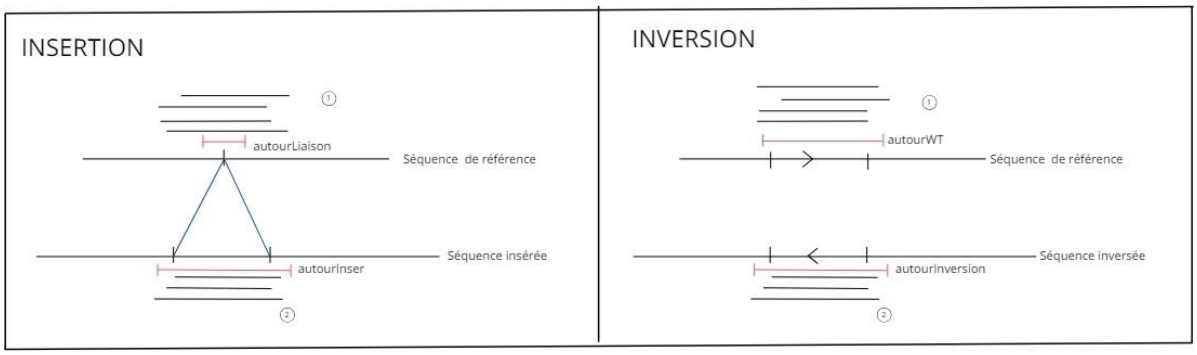


Figure 21: Schematic representation of the strategy used for inversions and insertions.

Filtering comparison

As we said earlier, our final code includes a filtering phase that allows us to save time on alignment by avoiding aligning reads located in irrelevant regions. However, this method entails the risk of losing information in the saved reads. It was therefore crucial for us to check the results obtained in both cases, i.e. without the filtering phase and with a filtering phase, in order to compare them, a step that was complicated for us by the large increase in execution time without line filtering.

We therefore first ran our code with the DGRP lines, with the files of the lines that are mutated, for each mutation. Table 2 shows the results for 7 files, together with the execution time. As you can see, with filtering we don't exceed 14 minutes execution time for a mutation searched in 7 files. However, without filtering, when we test a mutation in the same 7 files, we get much higher times, up to around 71 minutes. As you can see, in both cases we get a wide range of times. For example, for nyx2, the double deletion, the execution time is much higher. This is due to the fact that there are twice as many motifs to process and also more reference sequences, which considerably increases execution time. On the contrary, for Q365(substitution) we obtain in both cases shorter execution times than the others. Indeed, in the case of substitution, fewer patterns, patternsStricted and reference sequences are used, so filtering and alignment are much shorter.

Mutation testée	Nombre de fichiers testés	Temps d'éxécution (min)	Filtrage ?
melt[RAL509)	7	6	Oui
Rh5[delta26]	7	6,49	Oui
Rh5[Q365]	7	4,47	Oui
Rh6[Edn]	7	6,29	Oui
Rh6[RAL551]	7	7,1	Oui
nyx1	7	8,15	Oui
nyx2	7	14	Oui
melt[RAL509)	7	38,15	Non
Rh5[delta26]	7	39,45	Non
Rh5[Q365]	7	29,1	Non
Rh6[Edn]	7	35	Non
Rh6[RAL551]	7	40,35	Non
nyx1	7	45,5	Non
nyx2	7	71,24	Non

Table 2: Execution times with and without filtering. Table representing the execution time with or without filtering, obtained for each mutation on 7 read files of DGRP lines.

It's also important to note that the power of the machine on which we run the program also plays a role in the algorithm's performance. The results you see here represent the performance we achieved with a

with 16 cores and 24 gigabytes of RAM. On another computer, with only 4 cores, the algorithm took much longer, sometimes up to 25 minutes. So, with a more powerful machine, with more cores, the performance of our algorithm could really be improved.

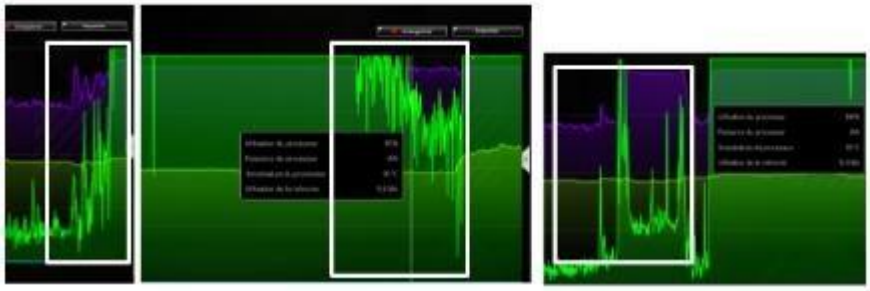


Figure 22: Evolution of machine resource usage during mutantResearch execution.

Figure 22 shows the moment when the query is received and filtering for 16 lines is launched. We can see that CPU utilization at 100%. This utilization remains constant until the filtering starts to finish. The n/2 (n = cores) are gradually released, reducing the load on the CPU. Once half is released, utilization explodes again to trigger alignment. Utilization remains at 100% until the end of the alignment. The last image shows the case of a self-running filter and its alignment.

We can therefore say that, even if certain improvements can be made to maintain 100% utilization for longer, the algorithm is very efficient, and uses all the resources necessary for its operation. We also note that RAM is not heavily used for either alignment or filtering. It would therefore be essential, in the next stage of deployment, to find more powerful processors with more cores.

We also noticed that the programming language used to implement the algorithm also had a major influence on execution speed. Indeed, as we said earlier, before using SSearch36, we had used a JAVA implementation of Smith-Waterman. With this JAVA version, for similar tests (looking for a mutation on 7 files with filtering), we had to wait around 5 hours before obtaining results. In comparison, the C implementation we chose with SSearch36 proved far more powerful and efficient.

The importance of comparing code with and without filtering also lies in checking that no important information is lost, i.e. that we don't miss any mutated reads or lose any wt reads. We'll now show you some examples to illustrate that filtering doesn't cause us to lose any information. Figure 25 shows the results obtained for the melt[RAL509] mutation in 6 line files. As you can see, we don't lose any mutated reads for the RAL509 line with filtering. And for the other lines we don't win or lose any matches, with or without filtering. Conversely, for wt, we can see that thanks to filtering extra wt alignments are found. For the *nyx1* mutation (Figure 24), with or without filtering, there is no difference in results for either matches or wt. Similarly, for the *Rh5[q365]* mutation (Figure 23), there is no noticeable difference between match results, with or without filtering. But we can note that in some cases, we find more wt alignment with filtering.

Line	Mutation	filtrage	Matchs	Wt
509	q365	OUI	1	9
509	q365	NON	1	9
790	q365	OUI	8	0
790	q365	NON	8	0
551	q365	OUI	0	14
551	q365	NON	0	14
93	q365	OUI	0	12
93	q365	NON	0	10
559	q365	OUI	0	8
559	q365	NON	0	8
892	q365	OUI	0	8
892	q365	NON	0	8
646	q365	OUI	0	15
646	q365	NON	0	14
			0	3

Figure 25: Comparison of Rh5[q365] mutation for 7 files with and without filtering.

Line	Mutation	filtrage	Matchs	Wt
509	nyx1	OUI	0	15
509	nyx1	NON	0	15
790	nyx1	OUI	0	6
790	nyx1	NON	0	6
551	nyx1	OUI	0	11
551	nyx1	NON	0	11
93	nyx1	OUI	0	16
93	nyx1	NON	0	16
559	nyx1	OUI	0	4
559	nyx1	NON	0	4
892	nyx1	OUI	13	0
892	nyx1	NON	13	0
646	nyx1	OUI	0	11
646	nyx1	NON	0	11
			0	0

Figure 23: Comparison of the melt[RAL50G] mutation for 7 files with and without filtering.

Line	Mutation	filtrage	Matchs	Wt
509	melted	OUI	16	0
509	melted	NON	16	0
790	melted	OUI	0	24
790	melted	NON	0	21
551	melted	OUI	0	0
551	melted	NON	0	0
93	melted	OUI	0	28
93	melted	NON	0	28
559	melted	OUI	0	0
559	melted	NON	0	0
892	melted	OUI	0	20
892	melted	NON	0	17
646	melted	OUI	0	18
646	melted	NON	0	16
			0	8

Figure 24: Comparison of nyx1 mutation for 7 files with and without filtering.

These results allow us to confirm that there is a significant absence of loss of mutated reads, and that there is better detection of wt reads. This confirms that filtering is beneficial overall. It optimizes the quality of alignments without compromising mutation detection, and enables us to optimize time very significantly: time is reduced by around 60%.

All these results led us to validate the use of filtering. However, we feel that we should still be cautious about its use, and that it would be interesting to

regularly run tests without filtering to see if there is any loss of information for other lines, for example. That's why we've added several options for the queries: apply filtering or not, change the margin of error, and so on. After several results, it might be interesting to analyze which margins are the most interesting to use.

200 DGRP lines

After making our code as efficient as possible and checking that no information was lost, we were able to run our code on a larger scale. We searched for each of the 7 mutations in all the DGRP lines. Table x shows the results we obtained for the 7 mutations in 16 lines.

Line	Melted mutated	Melted wt	Delta26 mutated	Delta26 Wt	Q365 mutated	Q365 wt	Edn mutated	Edn wt	RAL551 mutated	RAL 551 Wt	nyx1 mutated	nyx1 wt	nyx2 mutated	nyx2 wt
21	0	9	0	23	0	4	0	6	0	9	0	8	0	0
26	0	15	0	33	0	18	0	18	0	26	0	22	0	6
28	0	17	0	31	0	16	0	25	0	28	0	18	0	8
31	0	51	0	60	0	16	0	0	0	0	0	27	0	14
32	0	48	0	62	0	16	0	0	0	0	0	29	0	3
38	0	27	0	41	0	8	0	18	0	32	0	19	0	2
40	0	25	0	40	0	12	0	12	0	30	0	28	0	4
41	0	19	0	41	0	10	0	22	0	32	0	19	0	3
42	0	17	0	26	1	5	0	9	0	11	0	16	0	2
45	0	14	0	42	0	11	0	24	0	27	0	20	0	4
509	16	0	0	31	1	9	0	18	0	16	0	16	0	4
551	0	0	0	45	0	14	-	-	0	0	0	11	0	0
892	0	22	0	25	0	8	-	-	0	0	13	0	0	0
790	0	23	0	32	8	0	0	11	0	13	0	6	0	16
93	0	28	0	27	0	12	0	19	0	21	0	16	12	0
646	0	17	21	0	0	15	0	9	0	22	0	12	0	8

Figure 26: Table of results for 16 DGRP lines according to the 7 mutations. (lines indicate that no significant results were found for these alignments).

The table you see here (Figure 26) is just an overview of the lines we tested. The lines in some of the boxes represent lines for which no alignment was carried out. By following this link results_stage you can access the full table of results for each mutation and each line. As you can see, so far we've only been able to test x lines out of 205. Our aim is to finish testing all the data so that we can present complete results at the viva.

Returning to the results, we again found the Rh5[q365] mutation in a new line, 42. As you can see, in the results shown in this table, we have not discovered any new lines carrying one of the mutations.

Another interesting finding is the lack of results for line 551. Even the RAL551 mutation, specific to this line, is not found. This could be explained by possible errors in the file, since there's no indication that either the algorithm or the filtering was wrong.

to lose some data. This file is particularly small and has no similarity to what we see on the results and what we see on the NCBI interface.

IV. Conclusions and outlook

Our study is based on the detection of specific mutations using a search algorithm we have developed. This algorithm is based on a pipeline including a smith Waterman alignment and a series of filtering steps, post and pre-alignment, aimed at improving the algorithm's performance and accuracy. We're talking mainly about deletions and substitutions, as these are currently the only types of mutation that have been identified as responsible for certain Drosophila rhodopsin phenotypes. However, our algorithm has also been developed to recognize other types of mutation, such as inversions and insertions.

Thanks to this algorithm, we were first able to identify in-silico each of the mutations in their reference lines. Examples include melt[RAL509] and Rh5[q365], where melted was not recognized at all by DGRP and q365 was only recognized in a few lines. For these mutations, we were able to broaden the number of DGRP lines found to be mutated compared with the DGRP results and the natural variants table. These results were verified visually using the alignments proposed by NCBI. This demonstrates that our algorithm is capable of identifying small mutations precisely, and that it detects mutations that could not be detected before.

Large deletions, small deletions, multiple mutations and substitutions were correctly identified, thanks to a precise, high-performance pipeline. With regard to substitutions, we also obtained the results expected for the reference line. We were also able to highlight the presence of a read mutation, for Q365, in the RAL509 and 41 lines, which could not be identified by the alignment algorithms used by DGRP.

We were also able to show that filtering reads saved us a considerable amount of time, from 71 minutes to 14 minutes for a mutation searched in 7 files, and avoided irrelevant alignments. And we can now confirm that no information is lost during this stage. Indeed, in a comparative analysis between the results found with and without filtering for 3 mutations, we have shown that there is no significant loss of information for mutated reads, thus ensuring that important mutations are not missed. On the contrary, filtering improves the detection of wt reads.

We've also managed to ensure that the vast majority of CPU utilization throughout the process is 100%, optimizing the algorithm. Indeed, even if we obtain satisfactory execution times with a better machine or exhaustive calculation of the filter order, this performance could be improved.

Implementation of the database will be essential for mutation prevalence studies, as well as for maintaining mutation, gene and lineage data. The graphical interface has yet to be tested and connected to the rest of the code, but this remains an upcoming objective.

These studies and tools will complement the experimental work of teams working on Drosophila, and perhaps other species in the future. This could provide phylogenetic and evolutionary information on certain rare mutations for which researchers are still seeking explanations.

The results of the search for the 7 mutations in the DGRP lines revealed that the Rh5[q365] mutation affects lines that had not previously been identified as mutated by DGRP. In addition to lines 397, 589 and 790, already identified for this mutation (Figure 1), we were able to identify the presence of this mutation in lines 509 and 41. On the other hand, to date we have not detected any mutations in lines other than the one we have identified.

Due to time constraints, we were unable to test all 205 DGRP lines for each mutation. Before the defense, we hope to test all the lines in order to give global results on the whole panel.

The next step in this project will be to study Drosophila lines from different geographical regions. This part of the project will require the collection of sequencing data for these lines. To do this, we plan to use the DrosEu database, a European consortium of researchers working on genetic variations in Drosophila. To study these genetic variations, the researchers on this project have already carried out detailed sequencing, which we can use to verify the presence of the mutations we're working on. We will therefore need to contact the researchers associated with DrosEu to obtain access to the sequencing files of the Drosophila lines. In parallel, we can also explore other databases, such as FlyBase, an online resource dedicated to providing genetic and genomic data on Drosophila.

Once we have collected enough data, we can proceed with the analysis to identify the mutations present in the new Drosophila lines. We will then need to classify the mutations according to their geographical location. This will enable us to calculate the prevalence of mutations and determine which Drosophila lines are most likely to be affected by each mutation. The prevalence calculation corresponds to the proportion of individuals in a population with a particular phenotype ((nb of individuals with the phenotype / nb of total individuals) * 100.

This would enable us to understand how mutations are distributed geographically, and whether phenotypes are associated with a geographic region.

In future versions, it would be interesting to carry out pre-filtering calculations in order to prevent moments of thread stagnation, when filtering is carried out, threads remain available but no alignment takes place. This could be done by predicting alignment time as a function of file size, and predicting alignment order.

V. References

1. **Vasiliauskas Group Report**:

   Vasiliauskas, D. *Master Vasiliauskas Group*. NeuroPSI, 2021, https://neuropsi.cnrs.fr/wp-content/uploads/2021/06/master-vasiliauskas-group-v2-1.pdf.

2. **DrosEU**:

   "About Us." *DrosEU*, https://droseu.net/about-us/.

3. **DGRP**:

   Drosophila Genetic Reference Panel. North Carolina State University, http://dgrp2.gnets.ncsu.edu/.

4. *Trost, B., Loureiro, L. O., & Scherer, S. W. (2021). Discovery of genomic variation across a generation.* Human Molecular Genetics, *30(R2), R174-R186. https://doi.org/10.1093/hmg/ddab209*

5. Mackay et al., 2012

   Mackay, T. F. C., Richards, S., Stone, E. A., Barbadilla, A., Ayroles, J. F., Zhu, D., ... & Gibbs, R. A. (2012). The Drosophila melanogaster Genetic Reference Panel. *Nature*, 482(7384), 173-178. https://doi.org/10.1038/nature10811

6. Huang, Massouras, et al, 2014

   Huang, W., Massouras, A., Inoue, Y., Peiffer, J., Ràmia, M., Tarone, A. M., ... & Mackay, T. F. C. (2014). Natural variation in genome architecture among 205 Drosophila melanogaster Genetic Reference Panel lines. *Genome Research*, 24(7), 1193- 1208. https://doi.org/10.1101/gr.171546.113

7. Cook, R.K., Christensen, S.J., Deal, J.A. *et al.* The generation of chromosomal deletions to provide extensive coverage and subdivision of the *Drosophila melanogaster* genome. *Genome Biol* 13, R21 (2012).

8. Mitchell, Chandani & Becker, Vada & DeLoach, Jordan & Nestore, Erica & Bolterstein, Elyse & Kohl, Kathryn. (2022). The Drosophila Mutagen-Sensitivity Gene mus109 Encodes DmDNA2. Genes. 13. 312. 10.3390/genes13020312.

VI. Appendices

1. Example of results output for the melt[RAL509] mutation and some files


```
"melted": {
    "RAL38": {
      "matches": 0,
      "mismatches: 27,
      "ratio: 0
    },
    "RAL28": {
      "matches": 0,
      "mismatches: 17,
      "ratio: 0
    },
    "RAL26": {
      "matches": 0,
      "mismatches: 15,
      "ratio: 0
    },
```