

## RAPPORT DE STAGE - LICENCE INFORMATIQUE

---

# Visualisation des clones V(D)J en fonction des distances d'édition, et suivi de la Leucémie Aigüe Lymphoblastique

---

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE  
ÉQUIPE DE RECHERCHE "BONSAI"

*Tuteur Entreprise :*  
*Auteur :* Dr. Mathieu GIRAUD  
Antonin CARETTE *Tuteur Universitaire :*  
Dr. Samy MEFTALI



# Table des matières

<b>Introduction</b>	<b>3</b>
0.1 Remerciements . . . . .	3
0.2 Présentation de l'entreprise et de l'équipe . . . . .	3
0.2.1 Le <i>LIFL</i> . . . . .	3
0.2.2 L'équipe <i>Bonsai</i> . . . . .	3
<b>1 Contexte du stage et objectifs</b>	<b>5</b>
1.1 La Leucémie Aigüe Lymphoblastique . . . . .	5
1.2 Le projet Vidjil . . . . .	6
1.3 Objectifs de stage . . . . .	8
<b>2 Préparation</b>	<b>9</b>
2.1 L' <i>interface</i> . . . . .	9
2.1.1 Un travail d'un an... . . . . .	9
2.1.2 Langages et frameworks . . . . .	9
2.1.3 Composition de l' <i>afficheur</i> . . . . .	10
2.2 Étude et intégration . . . . .	11
2.2.1 La documentation . . . . .	11
2.2.2 L'intégration au projet . . . . .	11
2.2.3 Le domaine biologique . . . . .	11
<b>3 Le graphe de distances d'édition</b>	<b>12</b>
3.1 Ajout d'une nouvelle distribution . . . . .	12
3.2 Création du graphe . . . . .	13
3.3 Problème et résolution des arêtes . . . . .	13
3.3.1 " <i>Il était une fois, un premier algorithme...</i> " . . . . .	14
3.3.2 " <i>L'algorithme contre-attaque</i> " . . . . .	15
3.3.3 " <i>Le carré est un triangle qui a réussi, ou une circonférence qui a mal tourné...</i> " . . . . .	15
3.4 Résultats . . . . .	16
<b>4 L'algorithme de clusterisation DBSCAN</b>	<b>18</b>

4.1	Présentation . . . . .	18
4.2	Implantation de DBSCAN . . . . .	20
4.2.1	Choix des paramètres . . . . .	21
4.2.2	Visualisation . . . . .	21
4.3	Résultats . . . . .	24
<b>5</b>	<b>Recherche et calcul des distances d'édition</b>	<b>25</b>
5.1	Utilisation des <i>windows</i> . . . . .	25
5.1.1	Qu'est-ce qu'une <i>window</i> . . . . .	25
5.1.2	Pourquoi les utiliser ? . . . . .	26
5.2	Les différents coûts utilisés . . . . .	26
5.2.1	Les coûts étudiés . . . . .	26
5.3	Conclusion . . . . .	28
<b>6</b>	<b>Le travail en recherche</b>	<b>29</b>
6.1	La vie est un long fleuve tranquille... . . . . .	29
6.2	...ou pas! . . . . .	30
<b>Conclusion</b>		<b>32</b>
<b>Bibliographie - Webographie</b>		<b>33</b>

# Introduction

## 0.1 Remerciements

Je remercie tout d'abord mon tuteur d'Université : **Samy Meftali**, ainsi que mon tuteur en entreprise : **Mathieu Giraud**, pour toute l'attention donnée quant au travail sur le projet, ainsi que sur ce rapport.

Je remercie ensuite l'équipe *Bonsai* ainsi que les membres du projet *Vidjil* pour leur accueil ainsi que leur soutien, recueillis durant ces 3 mois de stage.

## 0.2 Présentation de l'entreprise et de l'équipe

### 0.2.1 Le *LIFL*

Le *LIFL* (Laboratoire d'Informatique Fondamentale de Lille) est un laboratoire Français fondé en 1983, présent sur le campus de l'Université Lille1, rattaché à l'Institut des Sciences de l'Information et de leurs Interactions (*INS2I*) du *CNRS* (Centre National de la Recherche Scientifique).

Il est maintenant dirigé par **Sophie Tison** comprend 10 équipes-projets et équipes du centre de recherche *INRIA* Lille - Nord Europe (Institut national de recherche en informatique et en automatique), de recherches diverses et variées (axe Mathématique, Réalité Virtuelle, BioInformatique, Sécurité et Réseau, Systèmes Multi-agents, etc...) ; aujourd'hui, plus de 300 personnes y travaillent (enseignants-chercheurs, ingénieurs, techniciens, administratifs et doctorants).

### 0.2.2 L'équipe *Bonsai*

*Bonsai* est une équipe du *LIFL*, auparavant nommée *Sequoia*. Cette jeune équipe dépend du *LIFL*, de l'*INRIA* ainsi que du *CNRS*, et est dirigée par **Hélène Touzet**. Les travaux des membres de l'équipe (on en compte actuellement 22) sont tous réunis dans un seul grand domaine : la Biologie. En effet, l'équipe *Bonsai* est très réputée dans la bio-informatique ; on dénombre aujourd'hui plus de 30 applications/logiciels

utilisables gratuitement<sup>1</sup>, créés par les membres de l'équipe.

Cette équipe m'a donc accueillie pour mon stage de fin de licence, s'étant déroulé du 1er Avril au 30 Juin 2014, mes encadrants étant **Mathieu Giraud** (mon tuteur en entreprise), **Mikaël Salson** ainsi que **Marc Duez**, ingénieur du projet *Vidjil*.

---

1. Ces logiciels sont consultables sur le site internet de l'équipe : <http://bioinfo.lifl.fr>

# Chapitre 1

## Contexte du stage et objectifs

### 1.1 La Leucémie Aigüe Lymphoblastique

Chaque Humain et Vertébré détient un système de recombinaison de l'ADN (que l'on appelle **recombinaison V(D)J**<sup>1</sup>), mécanisme permettant de créer une grande diversité de récepteurs d'antigènes (nécessaires à la reconnaissance de corps étrangers), et donc de lymphocytes. Chaque **lymphocyte** (faisant partie des "globules blancs", ou Leucocyte) possède un réarrangement V(D)J (c'est-à-dire un réarrangement de l'ADN du lymphocyte - voir **Figure 1**)<sup>2</sup> unique, et le dote d'un récepteur lui permettant de reconnaître un antigène bien spécifique. Ce lymphocyte pourra plus tard se multiplier (si il y a contamination par des antigènes spécifiques reconnus), afin de mieux permettre de contrer l'agresseur, et formera alors dans ce cas un **clone**<sup>3</sup>.

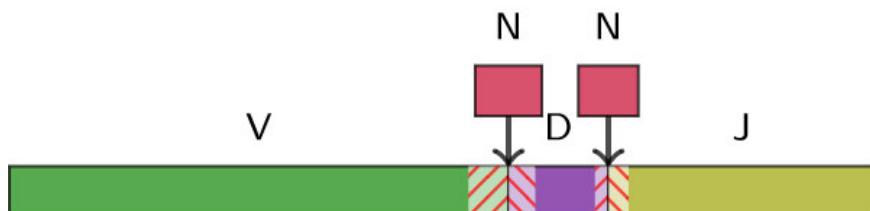


FIGURE 1 – Exemple d'un réarrangement V(D)J sur un lymphocyte. La partie "N" correspond à des nucléotides pouvant être insérés, entre les gènes réarrangés. Schéma repris de l'article *Fast multiclonal clusterization of V(D)J recombinations from high-throughput sequencing*, M. Giraud, M. Salson (bibliographie).

La **Leucémie Aigüe Lymphoblastique** (L.A.L.) est un cancer liquide<sup>4</sup> affectant

1. Phénomène biologique impliquant des processus par lesquels une molécule d'ADN (ou d'ARN) est coupée, puis jointe à une autre.
2. Les régions V, D et J sont des parties d'un anticorps - voir **Figure 2**.
3. Un clone V(D)J est donc un ensemble de lymphocytes ayant le même réarrangement V(D)J.
4. Le cancer liquide, encore appelé cancer sanguin, est un ensemble composé des leucémies (cancers du sang et de la moëlle épinière) et des lymphomes (cancers du système lymphatique).

majoritairement les enfants. Il s'attaque à un lymphocyte en le forçant à se multiplier afin de pouvoir créer un clone de lymphocytes cancéreux, ce qui inhibe les actions du reste du système immunitaire.

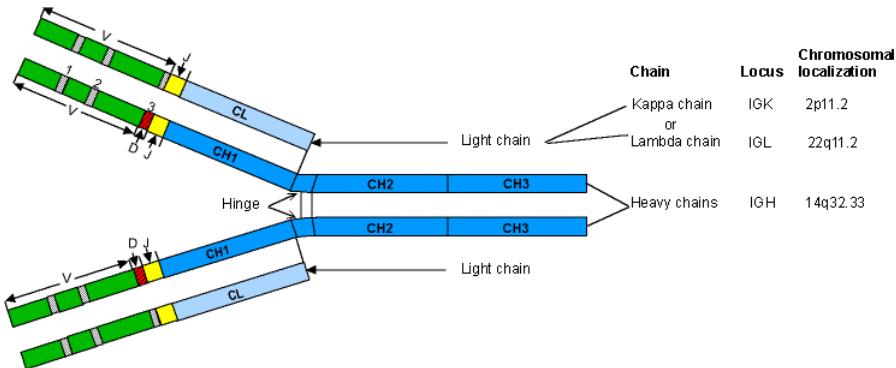


FIGURE 2 – Aperçu d'un lymphocyte, et des régions V(D)J sur le site actif. Figure reprise via [imgt.org](http://imgt.org).

## 1.2 Le projet Vidjil

*Vidjil* est un projet intra (pour le développement des programmes) et extra (pour toute la partie échantillonage) Universitaire, créé par un petit groupe de chercheurs et d'ingénieurs de l'équipe *Bonsai* et du *Laboratoire d'Hématologie de Lille2*.

Ce projet a pris naissance en 2011. Peu d'outils fiables et complets étaient disponibles pour le séquençage à haut-débit<sup>5</sup> demandé par l'équipe d'Hématologie de Lille2 pour leurs analyses concernant principalement la L.A.L., menée par **Claude Preudhomme** - requérant beaucoup de travail en algorithmique et programmation informatique, ainsi que dans la création et l'adaptation à un modèle biologique spécifique<sup>6</sup>. Après en avoir parlé avec **Martin Figeac**, cette dernière a de suite émis l'idée d'en discuter avec des informaticiens intéressés par la biologie, issus notamment de l'équipe *Bonsai* du *LIFL* - ce qu'il a fait en parlant du projet à M. **Giraud** et M. **Salson**. Les deux équipes Universitaire se sont donc rencontrées début de l'année 2011 pour en parler. Ce n'est qu'en 2012 que le premier programme a été écrit, et que le projet *Vidjil* a réellement débuté.

Le projet *Vidjil* est principalement conçu pour les médecins et les chercheurs.

Il consiste à la mise en place d'un programme permettant de séquencer à haut-débit les clones V(D)J, issus de données d'un format FASTA ou FASTQ<sup>7</sup>, à partir d'un échantillon de sang d'un patient atteint de L.A.L. (voir **Figure 3** de la partie **Annexe**). Le résultat retourné par le programme est donc une analyse de l'échantillon,

5. Méthode de séquençage massif de l'ADN, avec clonage et amplification moléculaire - créé en 2005

6. Voir le premier article écrit par **Mathieu Giraud** et **Mikaël Salson**, sur l'*algorithme*.

7. Les formats FASTA et FASTQ sont des formats de fichier texte utilisés pour stocker des séquences biologiques de nature nucléique ou protéique.

stockée dans un fichier au format DATA, en utilisant JSON<sup>8</sup>). Ce programme est écrit en C++, dû à la gestion de la mémoire héritée du langage C, et est dénommé *Vidjil*. Aussi, il consiste en une interface Web (composée majoritairement de JavaScript), appelée *interface*, dont le but est d'afficher toutes les informations nécessaires quant aux résultats de séquençage recueillis dans le fichier DATA, et d'en faire une sortie papier au format PDF. L'interface a été décidé comme site internet afin de permettre aux utilisateurs d'avoir le même résultat sur leur ordinateur, et ne pas avoir à installer des packages ou plugins pour l'utiliser à bon escient, qu'importe le système d'exploitation utilisé, mais aussi de pouvoir créer et exploiter plus facilement un accès serveur pour sauvegarder et/ou retrouver ses données analysées.

```
>clone-039-window
TACTGTGCAAAGATATTCTAAATCACTTAAGCAGCAGCTGGCCACCCGAACCTGGTCG
>clone-039-representative - VDJ          0 54 70 81 90 137      IGHV3-9*01 0/TTCTAAATCACTA/6 IGHD6-13*01 3
/CACCCCG/2 IGHJ5*02
ACCTGAGGAGACGGTGACCAGGGTCCCTGGCAGGGTCAACCCAGTTGGCAGCTGCTGTTAAAGTGA
GTGCTCTCAGCTCTCAGACTGTTCATTG
#### Clone #040 -      300 reads - 0.100% - 0.100% 1.000 (similar to Clone #001)
>clone-040-window
TCTGTGACTCCCAGGGAGTGCTACAGTAACCCCTTGACTGCCAGGGAACCTGGT
>clone-040-representative - VDJ          0 256 260 269 273 315      IGHV6-1*01 27/GTG/3 IGHD4-11*01 3/CCC/4 IGHJ
4*02
ACCTGAGGAGACGGTGACCAGGGTCCCTGGCAGTAGCAAAGGGGTTACTGTAGCAC
ACTCCTGGAGTCACAGAGTTAGCTGCAGGGAGAAGTGGTCTGGATG
TGTCTGGTTGATGGTTATTCGACTTTACAGATACTGCATAATCATTATA
ACCACTGGACCTGTAGTAGTGTCTTCCAGGCCACTCAAGGCCTCTCGATGGGACT
GCCTGATCCAGTTCCAAGCAGCACTGTTGCTAGAGACACTGTCCCCGGAGATGGCACAGGTGAGTGAGAGGGTCTGCAGGGCTTAC
CAGTCCTGGACCAC
#### Clone #041 -      300 reads - 0.100% - 0.100% 1.000 (similar to Clone #001)
>clone-041-window
ACTCTGTGGCTCCGAGGGAGTGCTACAGTAACCCCTTGACTACTGCCAGGGAACCTGGT
>clone-041-representative - VDJ          0 260 264 273 277 319      IGHV6-1*01 27/GTG/3 IGHD4-11*01 3/CCC/4 IGHJ
4*02
ACCTGAGGAGACGGTGACCAGGGTCCCTGGCAGTAGCAAAGGGGTTACTGTAGCAC
ACTCCTGGAGGCCACAGAGTTAGCTGCAGGGAGAAGTGGTCTGGATG
TGTCTGGTTGATGGTTATTCGACTTTACAGATACTGCATAATCATTATA
ACCACTGGACCTGTAGTAGTGTCTTCCAGGCCACTCAAGGCCTCTCGATGGGACT
GCCTGATCCAGTTCCAAGCAGCACTGTTGCTAGAGACACTGTCCCCGGAGATGGCACAGGTGAGTGAGAGGGTCTGCAGGGCTTAC
CAGTCCTGGACCAC
#### Clone #042 -      287 reads - 0.096% - 0.096% 1.000 (similar to Clone #001)
>clone-042-window
TCCGTGACTCCCAGGGAGTGCTACAGTAACCCCTTGACTACTGCCAGGGAACCTGGT
>clone-042-representative - VDJ          0 261 265 274 278 320      IGHV6-1*01 27/GTG/3 IGHD4-11*01 3/CCC/4 IGHJ
4*02
ACCTGAGGAGACGGTGACCAGGGTCCCTGGCAGTAGCAAAGGGGTTACTGTAGCAC
ACTCCTGGAGTCACGGAGTTAGCTGCAGGGAGAAGTGGTCTGGATG
TGTCTGGTTGATGGTTATTCGACTTTACAGATACTGCATAATCATTATA
ACCACTGGACCTGTAGTAGTGTCTTCCAGGCCACTCAAGGCCTCTCGATGGGACT
GCCTGATCCAGTTCCAAGCAGCACTGTTGCTAGAGACACTGTCCCCGGAGATGGCACAGGTGAGTGAGAGGGTCTGCAGGGCTTAC
CAGTCCTGGACCAC
#### Clone #043 -      284 reads - 0.095% - 0.095% 1.000 []

```

FIGURE 3 – Aperçu d'un séquençage (en cours sur ce *screenshot*), avec *Vidjil*

L'idée sur laquelle repose le projet *Vidjil* est que toutes les informations, quant aux réarrangements V(D)J concernés par la maladie et issus du prélèvement de sang, pourront être étudiées, comptées et analysées plus facilement par les utilisateurs du logiciel, afin de pouvoir suivre la LAL, mesurer son étendue, mais aussi adapter les traitements médicaux au mieux et prédire les rechutes.

8. JSON est un format de données textuelles et générique, permettant de représenter de l'information structurée. Ce format a été créé par M. **Douglas Crockford** en 2005.

*Vidjil* est donc un projet créé en partenariat avec le *Laboratoire d'Hématologie de Lille2*, par **Mathieu Giraud**, **Mikaël Salson** et **Claude Preudhomme**. Les deux équipes travaillent aujourd’hui avec *EuroClonality NGS*<sup>9</sup>, ainsi que différents laboratoires Français - comme l’hôpital Necker à Paris ou l’hôpital de Rennes - mais aussi d’Angleterre.

### 1.3 Objectifs de stage

Mes objectifs m’ont été donnés par **Mathieu Giraud** et **Mikaël Salson** lors d’un entretien téléphonique et oral en Décembre 2013/Janvier 2014. La problématique était de permettre à l’utilisateur de l’*interface* de visualiser de façon interactive les clones V(D)J selon leurs distances d’édition, de façon à suivre la progression des mutations des lymphocytes cancéreux dans le temps.

Cette problématique devait être résolu *via* deux objectifs, présentés ci-dessous.

#### L’*interface*

Dans un premier temps, je devais intégrer dans l’*interface* un graphe permettant de visualiser physiquement les distances d’édition des clones, ce qui permettrait de percevoir directement la similarité et la disparité des clones.

Ce premier travail devait s’effectuer sur une durée de deux mois, avec une programmation en HTML5, CSS3 et le langage Less, le langage orienté objet Javascript, Ajax, et en utilisant les frameworks<sup>10</sup> Javascript D3JS et JQuery - tous ces langages et frameworks étant déjà intégrés à l’*interface*, depuis sa conception par **Marc Duez**.

#### DBSCAN

Deuxièmement, le travail devait déborder sur une partie beaucoup plus algorithmique, avec l’utilisation de DBSCAN<sup>11</sup> afin de partitionner les clones V(D)J en fonction de deux paramètres : leur distance d’édition, ainsi que leur nombre de voisins.

L’algorithme était à planter en Javascript, sur une durée de 1 mois.

---

9. *EuroClonality NGS* est un Consentium Européen contenant plusieurs laboratoires, ce qui est utile au projet quant à l’échantillonage des données.

10. Un *framework* est un ensemble d’outils et de composants logiciels organisés conformément à un plan d’architecture et des patterns, l’ensemble formant ou promouvant un "squelette" de programme - il est conçu en vue d’aider les programmeurs dans leur travail.

11. DBSCAN est un algorithme de partitionnement de données proposé en 1996 par **Martin Ester**, **Hans-Peter Kriegel**, **Jörg Sander** et **Xiaowei Xu**.

# Chapitre 2

## Préparation

Avant de me lancer dans le travail tout de suite, il m'a fallu étudier l'*interface*, ses différentes parties afin de savoir le rôle de chacune, mais aussi les langages qui compossait le site internet afin de pouvoir au mieux les exploiter.

### 2.1 L'*interface*

#### 2.1.1 Un travail d'un an...

Toute la conception et l'écriture du code de l'*interface* ont été effectués par l'ingénieur de recherche du projet *Vidjil* : **Marc Duez**.

M. **Duez** a commencé à participer au projet *Vidjil* dès Mars/Avril 2013, lors son projet de fin d'année pour le Master MoCAD. Son ambition était de créer un logiciel répondant aux besoins des médecins et des chercheurs, avec le plus de facilité possible, tout en respectant les principes de "Programmation Orientée Objet"<sup>1</sup> et de "Modèle-Vue-Contrôleur"<sup>2</sup>.

À la fin de son Master, il a été embauché dans l'équipe *Bonsai* afin de pouvoir continuer à travailler sur cette même *interface*.

#### 2.1.2 Langages et frameworks

Le programme utilise les langages les plus utilisés du Web : HTML5, CSS3, JavaScript et Ajax.

Tout le côté "Administration serveur" a été réalisé grâce au langage Python (version 2.7), ainsi qu'au framework *open source* web2py.

Côté conception, il y avait déjà plus de 20 classes Javascript écrites (à raison de

---

1. Paradigme de programmation informatique.

2. Modèle destiné à répondre aux besoins des applications interactives en séparant les problématiques liées aux différents composants au sein de leur architecture respective.

1000 lignes de code par classe), utilisant 2 *frameworks* JavaScript (re)connus (D3JS et JQuery), ainsi que le langage Less<sup>3</sup>.

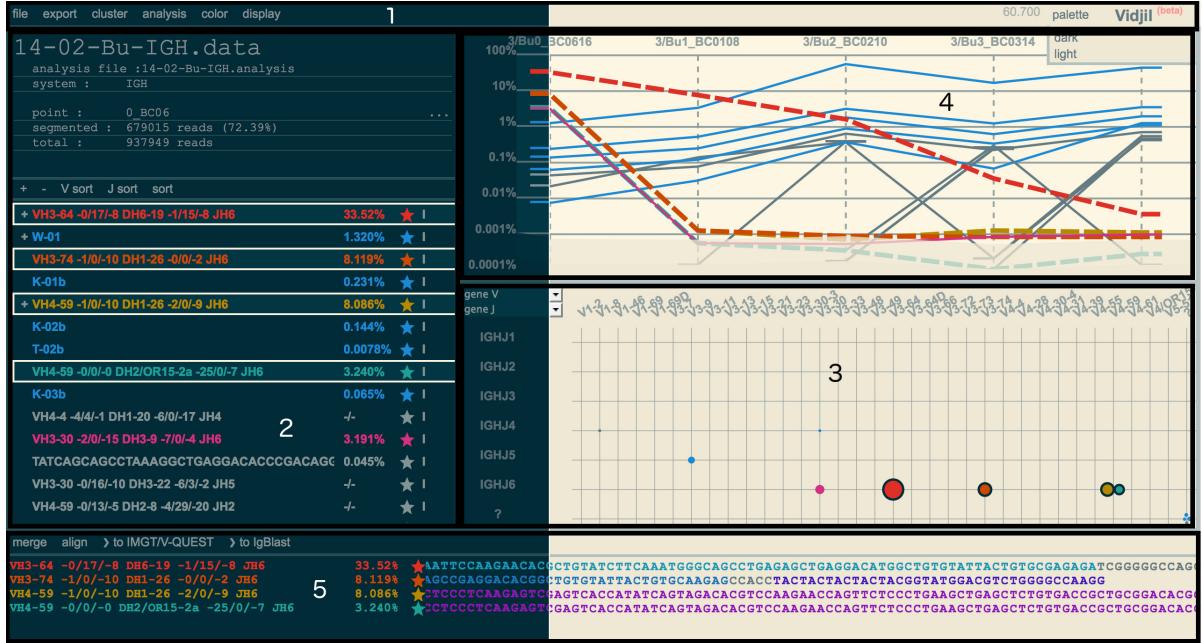


FIGURE 4 – Aperçu général de l’interface - montage via *Gimp*, avec les palettes CSS “Dark” (à gauche) et “Light” (à droite).

- 1 : menu haut,
- 2 : menu gauche - affichage des informations,
- 3 : *scatterplot* - visualisation des clones,
- 4 : affichage graphique,
- 5 : aligneur.

### 2.1.3 Composition de l’afficheur

L’interface se présente sur différentes parties, toutes dépendantes les unes des autres<sup>4</sup>. Je ne me suis intéressé, pour mon stage, qu’à trois grandes parties composant celui-ci (voir **Figure 4**) :

- le **menu haut** - pour l’implémentation des distributions, de la clusterisation et de la colorisation DBSCAN,
- l’affichage des informations (**menu gauche**) - pour l’implémentation de la clusterisation par DBSCAN,
- et enfin le ***scatterplot*** - partie au format *SVG*<sup>5</sup> permettant la visualisation des clones selon la distribution, colorisation et/ou clusterisation demandée(s).

3. Less est un langage dynamique de génération de feuilles de style, conçu par **M. Sellier Alexis**, très utile quant au changement dynamique de feuille de style afin de mieux visualiser les données calculées pour un échantillonage préparé, et permettant beaucoup plus de souplesse que le langage CSS, permettant de définir la présentation du document HTML.

4. Se rapporter à l’annexe, pour une visualisation de l’afficheur et ses différentes parties

5. SVG est un format de données conçu pour décrire des ensembles de graphiques vectoriels.

## 2.2 Étude et intégration

Cette phase a duré 2 semaines.

### 2.2.1 La documentation

Ne connaissant pas les *frameworks* D3JS et JQuery, les langages Ajax ou encore Less, il a donc fallu que je me documente à leur sujet afin de pouvoir exploiter toutes les fonctionnalités implémentées, mais aussi comprendre quand et comment les intégrer au projet.

### 2.2.2 L'intégration au projet

Durant les deux premières semaines, il m'a fallu étudier tout le code du projet utile à mes objectifs, ainsi que recourir à la création de la documentation pour le développeur (obsolète ou inexistante). L'étude du projet a été pour moi la plus grosse difficulté dans ce stage, n'ayant jamais effectué ceci auparavant, ni obtenu une ou plusieurs méthode(s) pour me permettre d'être le plus efficace possible pour ceci. Aussi, j'ai profité de ce temps afin de corriger plusieurs erreurs mineurs dans l'*interface*, afin de me permettre de mieux discerner les grandes parties de celle-ci et leurs buts respectifs.

De plus, il était très important pour moi de respecter les règles de l'équipe en matière d'ingénierie logicielle, ne serait-ce par l'écriture du code, de la documentation, mais aussi par rapport au respect des *commits*<sup>6</sup>/*push*<sup>7</sup> sur le serveur, via le logiciel de gestion de versions décentralisé *Git*<sup>8</sup>.

### 2.2.3 Le domaine biologique

Enfin, je me suis remis à étudier par moi-même la biologie moléculaire, le principe de recombinaison V(D)J, ainsi que les techniques de séquençage et de clusterisation. Tout celà afin de me permettre de comprendre et de poursuivre les discussions lors des réunions dans l'équipe, et avec le *Laboratoire d'Hématologie de Lille2*.

---

6. Un *commit* est une action permettant d'envoyer ses modifications locales vers le référentiel central

7. Un *push* est une action permettant de transférer des commits d'un dépôt local à un dépôt distant

8. *Git* est un logiciel libre créé par **Linus Torvalds**, permettant de gérer très facilement les codes sources de projets.

# Chapitre 3

## Le graphe de distances d'édition

Le projet contenait, à mon arrivée, 5 visualisations des données, pour une utilisation spécifique (nous appelons cette visualisation une **distribution**), dans le *scatterplot* (voir **Figure 5**) :

- V/J génique,
- V/J allélique,
- longueur des régions V/N,
- abondance des régions V et J,
- histogramme d'abondance des régions V.

analysis	color	display
V / J (genes)		
V / J (alleles)		
V / N length		
V / abundance at selected timepoint		
V distribution		

FIGURE 5 – Sous-menu Analysis de l'*interface*

Toutes ces distributions sont nécessaires à l'utilisateur, afin de visualiser directement la disparité et le réarrangement des clones, en fonction du jeu de données. Or, il n'y avait pas encore de travaux effectués sur une distribution des clones en fonction de leurs distances d'édition, afin de pouvoir visualiser leur rapprochement en fonction de leurs similitudes nucléotidiques, et donc de pouvoir observer le nombre de différences nucléotidiques (et donc de mutations) qu'il y a eu, au cours du temps, entre chaque clone.

### 3.1 Ajout d'une nouvelle distribution

L'ajout d'une nouvelle distribution n'était pas aisée, pour différentes raisons techniques.

Il a d'abord fallu me plonger dans le code du *scatterplot* afin de prévisualiser où implanter ma distribution, et quelles modifications seraient à effectuer sur cette partie de l'*interface*. Le moteur de D3JS, initialement, ne pouvait que positionner

les clones à une place donnée, et se charger de tous mouvements associés à ceux-ci. Pour cette distribution, il fallait forcer le moteur physique de D3JS afin qu'il puisse positionner les clones **lui-même, le plus précisément possible** (c'est à dire en respectant au maximum toutes les distances définies entre chaque clone), ne pouvant prédestiner la position des clones et de ses voisins sans distances - celà n'aurai alors aucune utilité d'implanter cette distribution.

## 3.2 Création du graphe

D3JS contient absolument toutes les fonctions nécessaires afin de construire proprement un graphe - un soucis concerne cependant le respect exact des informations données, et de la transformation effectuée par son moteur physique, ce qui a été précédemment évoqué.

Voici les transformations effectuées sur le moteur physique :

- ajout d'une durée d'animation élevée,
- ajout d'une gravité,
- ajout d'arêtes via un tableau Javascript (dans le fichier au format DATA) ayant comme entrées :
  - un point source,
  - un point cible,
  - une longueur entre ces deux pointset de distances spécifiques pour chacune,
- d'une élasticité - ce paramètre sera détaillé dans la sous-partie consacrée aux **Résultats**
- ajout d'une liaison entre un point et un début d'arête, afin de pouvoir bouger les arêtes en même temps que le point.

## 3.3 Problème et résolution des arêtes

Sur un graphe à 100 clones, chacun était lié aux 99 autres par une arête, il faudra donc créer un graphe complet à 10 000 arêtes. Je me suis alors posé la question suivante : "Le moteur physique sera-t-il capable de prendre en compte un nombre aussi élevé d'arêtes, ainsi que le calcul nécessaire pour la mouvance du graphe complet, en temps réel ?" Après plusieurs tests, la réponse est **non** : il n'a pas été créé et optimisé pour une tâche aussi lourde.

Il a donc fallu trouver un moyen d'optimiser au maximum la composition du graphe (c'est à dire garder un nombre minimal d'arêtes), afin de le représenter le plus rapidement possible, avec un maximum de concordance et de respect par rapport aux données fournies. Les 3 solutions présentées ci-dessous implémentent cette idée

de réduire drastiquement le nombre d'arêtes, sans diminuer la qualité du graphe.

### 3.3.1 "Il était une fois, un premier algorithme..."

Le premier algorithme qui m'est venu à l'esprit est de supprimer toutes les arêtes ayant une longueur supérieure à une distance implantée en dur dans le programme - cela permettra donc d'alléger le tableau d'arêtes Javascript pris en compte par le moteur, et de moins le faire souffrir lors des calculs.

Ce procédé a été un semi-échec : sur plusieurs jeux de données, nous avons supprimé en moyenne un peu plus de 50% des arêtes disponibles (nous étions passé de 10 000 à 4 970 arêtes sur un jeu de données, pour citer un exemple) - malheureusement, le moteur peinait à calculer toutes les positions des noeuds et arêtes par *frame* une nouvelle fois (sur une machine à processeur corei5 à 8Go de RAM, GPU Intel HD 4 000 intégré). Il était donc nécessaire de chercher un nouvel algorithme qui pourrait, au moins, diviser de moitié le nombre d'arêtes qui seront à prendre en compte par le moteur. Cependant, l'idée de modifier la distance maximale d'arêtes à afficher a été validé et gardé dans le projet - elle a été implémenté sous la forme d'un *slider*, présent dans le sous-menu "Display" du **menu haut** (voir **Figure 6**).



FIGURE 6 – Sous-menu "Display" du menu haut avec, de haut en bas : le *slider* "Display nodes" permettant d'afficher ou non un nombre plus ou moins grand de clones, les deux *sliders* permettant de changer les paramètres de DBSCAN, et enfin le *slider* pour modifier la distance maximale d'affichage des arêtes (pour le graphe concernant les distances d'édition)

### 3.3.2 "L'algorithme contre-attaque"

Le deuxième algorithme a été proposé par **Marc Duez**; beaucoup plus brutal, il permet de dispenser une grande partie d'arêtes, considérées comme inutiles lors de la construction du graphe - toujours en fonction des distances.

Il s'agit de :

1. trouver les 3 plus grandes arêtes du graphe ayant les caractéristiques suivantes :
  - chaque arête ne doit pas avoir de point commun avec les autres,
  - chaque point à une distance minimale d'une extrémité d'une arête trouvée ne doit plus être prise en compte lors du calcul,
  - chaque arête doit être le moins parallèle possible des autres,
2. calculer toutes les arêtes possibles entre une extrémité d'arête trouvée,
3. calculer 4 ou 5 arêtes possibles, dans une distance minimale, des points trouvés lors de l'étape 2.

Le résultat a été concluant - en effet, en déroulant l'algorithme, nous pouvons voir que :

- l'étape 1 se fera avec un maximum de 3 arêtes,
- l'étape 2 se fera avec un maximum de 600 arêtes (au total :  $600 + 3$  arêtes),
- l'étape 3 se fera avec un maximum de  $100 * 5$  arêtes (au total : 1103 arêtes maximum).

Ainsi, nous avons pu enlever un peu moins de 90% des arêtes totales pour un jeu de données quelconque, en moyenne ; ce qui nous est très bénéfique par rapport au temps de calcul du moteur pour chaque *frame*. Malheureusement, malgré un bénéfice de mémoire et calcul, le respect de la structure du graphe n'était pas optimal.

### 3.3.3 "Le carré est un triangle qui a réussi, ou une circonference qui a mal tourné..."

<sup>1</sup> Le troisième algorithme a été proposé par M. **Giraud Mathieu**. C'est cet algorithme qui a été retenu par mon équipe et moi-même, car respectant aussi bien la structure, que consommant moins de mémoire et de temps de calcul.

Cet algorithme modifie seulement la 1ère partie de l'algorithme précédent, de sorte à ne prendre que les 3 plus grandes arêtes formant un triangle, dont la somme des côtés sera la longueur maximale de toutes les autres représentations triangulaires possibles, selon les données initialement recueillies. Ainsi, chaque noeud du triangle est une source d'une arête, mais aussi une cible d'une autre, ce qui nous assurerera d'avoir une structure triangulaire : un "squelette" du graphe, pour l'aider le moteur à mieux modéliser la "chair".

---

1. Citation de **Pierre Dac**

## 3.4 Résultats

Le résultat final montre bien les similarités entre les différents clones calculés, très rapidement. Cependant, les résultats ne sont pas à la hauteur de nos espérances, encore à cause d'un soucis de représentation (voir **Figure 7**). En effet, celle-ci ne préserve pas de façon optimale les distances données en entrée. Étant donné qu'il place par lui-même chaque clone, il le fera au fur-et-à-mesure de la lecture (par un ordre quelconque) des distances d'édition du tableau Javascript, et les prendra en compte de moins en moins.

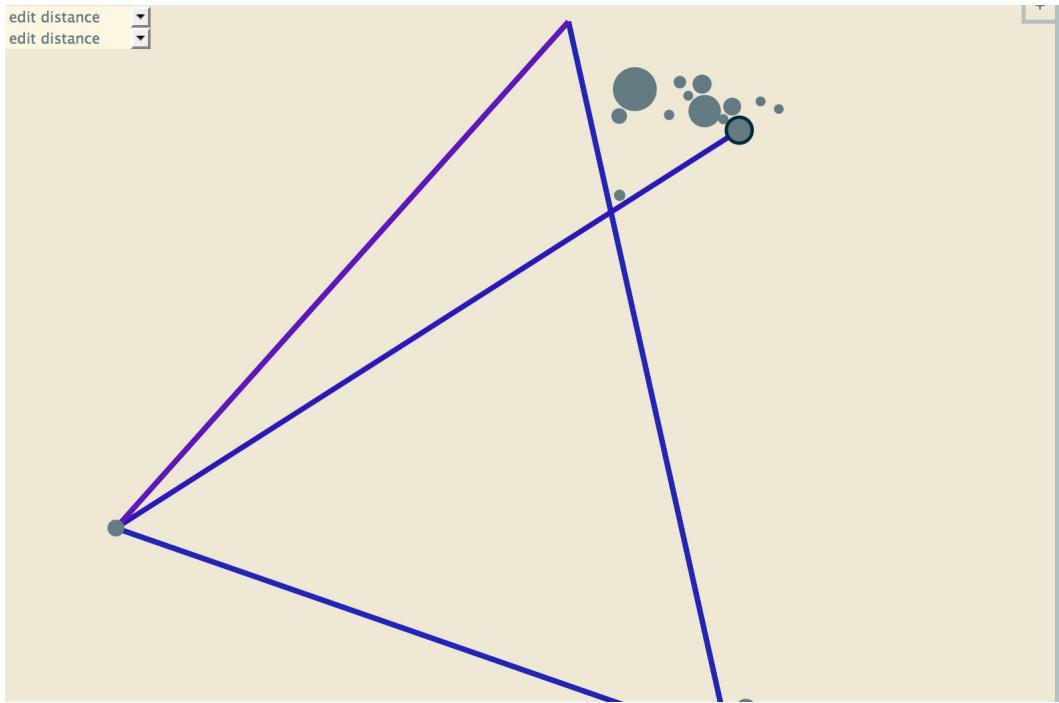


FIGURE 7 – *Interface* avec la distribution de distances d'édition - vous pouvez bien voir la disposition du triangle initial, ainsi que l'exploitation partielle visuellement du graphe.

Nous nous intéressons plus particulièrement à la similarité des clones qu'à leur disparité. Il était donc extrêmement important de garder les arêtes les plus petites (signe que la similarité est élevée) - ainsi, les plus petites arêtes doivent être plus respectées que les plus grandes, sauf pour les arêtes constituant le triangle initial bien entendu. La dernière solution à apporter était de donner une élasticité aux arêtes. Après les tests, il se trouve que les similarités sont bien respectées dans le graphe, mais accompagne un nouveau problème. Celui-ci, découlant de la précédente solution, est que les plus grandes distances ne sont plus respectées. Ainsi, deux points ayant une distance d'édition très grande peuvent se retrouver à proximité dans le graphe vu que le moteur attribue par lui-même les positions des clones, dans le *scatterplot*. Ce problème étant, à cause de ce problème par le moteur physique, insoluble, le déploiement du graphe de distances d'édition est un semi-échec.

Il existe cependant 2 solutions à apporter, pour un prochain sujet :

- aborder le souci comme non pas un seul graphe, mais un ensemble de sous-graphes dont le seul paramètre de variation sera la distance d'édition à prendre en compte - variable grâce au *slider* déjà implanté dans l'*interface* pour cette utilisation,
  - ne pas développer cet objectif comme un graphe mais plutôt comme un arbre, permettant toujours de répondre à la principale problématique - comme un arbre phylogénétique. Ce modèle pourra être implanté avec le *framework* D3JS (voir **Figure 8**).



FIGURE 8 – Exemple d'implantation d'un arbre phylogénétique circulaire, avec D3JS. Figure reprise via <http://bl.ocks.org/mbostock/4063550>.

# Chapitre 4

## L'algorithme de clusterisation DBSCAN

Ce chapitre conduira mon travail et mes réflexions sur la deuxième partie du stage que je devais effectué : l'intégration de l'algorithme de clusterisation DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*).

Ce travail a été réalisé en deux semaines, de fin Mai à la deuxième semaine de Juin.

### 4.1 Présentation

DBSCAN est un algorithme permettant la clusterisation de données, proposé en 1996 par Messieurs **Ester Martin, Kriegel Hans-Peter, Sander Jörg et Xu Xiaowei**, s'appuyant sur la densité estimée des clusters à partitionner. Cet algorithme est très simple à implémenter.

Tout d'abord, il faut choisir deux paramètres, essentiels à la clusterisation :

- **une distance** ( $\epsilon$ ), représentant un rayon de cercle et qui nous aidera à considérer qu'un point situé à une distance (la distance d'édition ici) inférieure ou égale à  $\epsilon$  d'un autre point est représentatif d'un seul cluster,
- **un nombre minimum de points** ( $M_p$ ), que l'on utilisera pour considérer le point pris en compte comme un **point central** (*core point* - le point détient un nombre de voisins plus élevé que  $M_p$ , dans le rayon  $\epsilon$ ), un **point de bordure** (*border point* - le point détient un nombre de voisins moindre que  $M_p$ , mais est voisin d'un **point central**), ou encore un **point de bruit** (*noise point* - tout point n'étant pas l'un des deux précédemment évoqué).

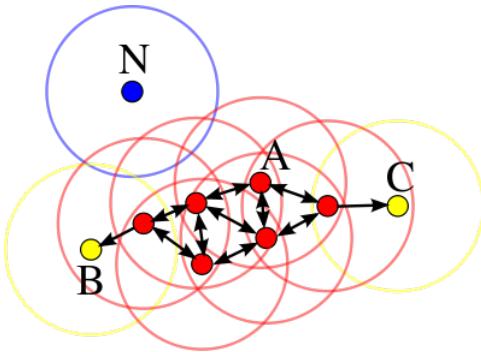


FIGURE 9 – Exemple de clusterisation, avec DBSCAN. Figure reprise via <http://fr.wikipedia.org/wiki/DBSCAN>

Cet algorithme est intéressant car il nous permet de parcourir chacun des points d'un graphe, son voisinage, et de clusteriser chaque point en fonction des deux paramètres énoncés précédemment - la densité revient donc au nombre de points ( $M_p$ ) compris dans un rayon bien spécifié ( $\epsilon$ ).

**Remarque :** Il est intéressant de constater que DBSCAN partitionne les données selon une certaine densité, donnée en paramètre via  $\epsilon$  et  $M_p$ ; cet algorithme ne pourra donc pas gérer les clusters de densités différentes en même temps (ce qui ne nous intéresse pas ici, fort-heureusement).

L'énorme apport de l'implémentation de DBSCAN est de **regrouper les clones similaires par rapport aux distances d'édition, et son voisinage** (voir **Figure 9**), ce qui n'était pas possible avec les précédentes distributions. Ainsi, cette implémentation de l'algorithme dans l'*interface* permettra à l'utilisateur de pouvoir visualiser directement quels sont les clones les plus similaire, par le rassemblement de ceux-ci, et ainsi de pouvoir déceler les clones cancéreux ayant subi une mutation au cours du temps, et ainsi de marquer l'évolution de la maladie (les distances d'édition étant calculées par rapport aux *windows*, marqueur d'évolution temporel - voir le **chapitre 5**).

L'algorithme est représenté ci-dessous (issu de Wikipedia<sup>1</sup>).

**Algorithm 4.1.1:** DBSCAN( $D, \text{eps}, \text{MinPts}$ )

**comment:** Base de l'algorithme

$C=0$

Pour chaque point  $P$  non visité des données  $D$   
marquer  $P$  comme visité

$PtsVoisins = \text{EPSILONVOISINAGE}(P, \text{eps})$

**if**  $\text{tailleDe}(PtsVoisins) < \text{MinPts}$

**then** marquer  $P$  comme "bruit"

**else**  $\begin{cases} C++ \\ \text{EXTENSIONCLUSTER}(D, P, PtsVoisins, C, \text{eps}, \text{MinPts}) \end{cases}$

**comment:** Procédure permettant d'inclure un point dans un cluster - extension d'un cluster

**procedure** EXTENSIONCLUSTER( $D, P, PtsVoisins, C, \text{eps}, \text{MinPts}$ )

    Ajouter  $P$  au cluster  $C$

**for each**  $P' \in PtsVoisins$

**do if**  $P'$  n'a pas été visité

**then**  $\begin{cases} \text{Marquer } P' \text{ comme visité} \\ PtsVoisins' = \text{EPSILONVOISINAGE}(D, P', \text{eps}) \\ \text{if } \text{tailleDe}(PtsVoisins') \geq \text{MinPts} \\ \quad \text{then } PtsVoisins = PtsVoisins \cup PtsVoisins' \end{cases}$

**if**  $P'$  n'est membre d'aucun cluster

**then** Ajouter  $P'$  au cluster  $C$

**comment:** Procedure retournant tous les points de  $D$  qui sont à une distance inférieure à  $\text{eps}$  de  $P$

**procedure** EPSILONVOISINAGE( $D, P, \text{eps}$ )

    Retourner tous les points de  $D$  qui sont à une distance inférieure à  $\text{eps}$  de  $P$

## 4.2 Implantation de DBSCAN

DBSCAN étant réutilisable à souhait dans le projet, son implémentation aura sa propre classe dans l'*interface*, communiquant avec le Modèle du "*MVC*"<sup>2</sup>. À l'aide

1. <http://fr.wikipedia.org/wiki/DBSCAN>

2. "Modèle-Vue-Contrôleur"

de QUnit<sup>3</sup>, j'ai pu créer une série de tests unitaires<sup>4</sup> (voir **Figure 10**) permettant de démontrer la fonctionnalité totale de l'algorithme, sur un jeu de tests (écrit à la main) permettant de tester tous les cas possibles issus d'un jeu de données biologiques.

#### 4.2.1 Choix des paramètres

Les paramètres pris par défaut sont de 0 pour  $\epsilon$  et  $Mp$ . Ainsi, au départ, étant donné que la distance d'édition maximale pour clusteriser est nulle, chaque clone a son propre cluster.

Le meilleur moyen pour l'utilisateur de réaliser, pour un jeu de données, la meilleure clusterisation est de faire varier les paramètres par nos propres moyens. J'ai donc ajouté, pour cet usage, deux sliders dans l'*interface* (voir **Figure 6**). Ces sliders créent, à chaque modification apportée, un nouvel objet DBSCAN selon les paramètres voulus - ce qui permet de visualiser, en une fraction de seconde, les modifications apportées manuellement.

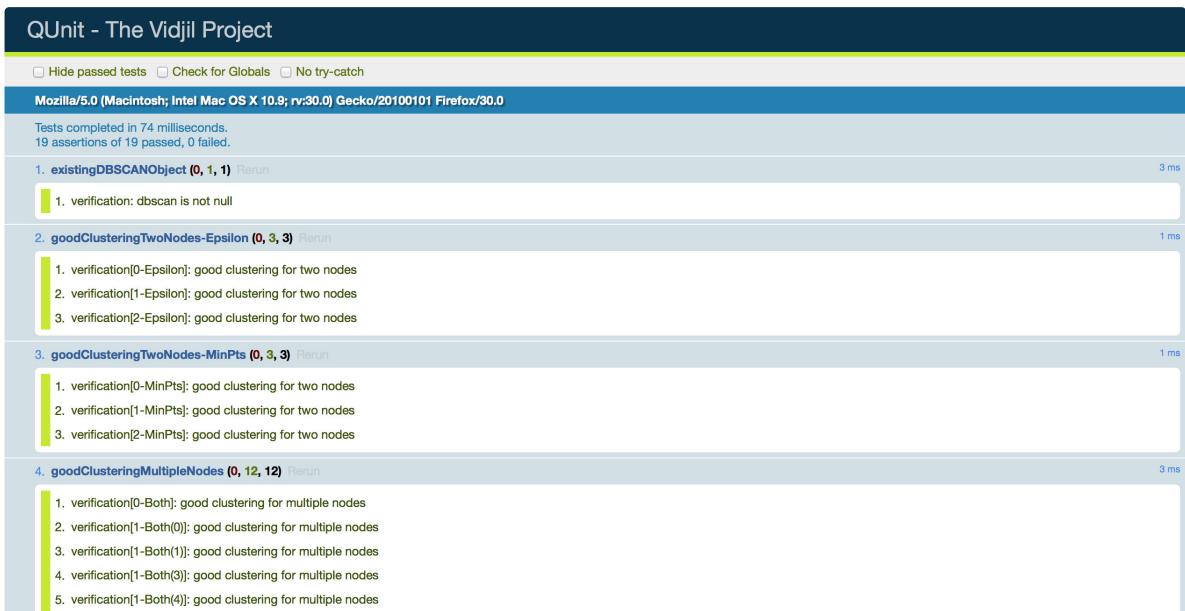


FIGURE 10 – Interface de QUnit, les tests étant passés avec succès

#### 4.2.2 Visualisation

La visualisation des groupes de clones est primordiale - elle en est même la base du sujet de stage. J'ai donc travaillé, une fois l'implémentation terminée de l'algorithme, sur la visualisation du résultat retourné par l'algorithme. Les clusters sont

3. QUnit est un *framework* permettant d'implémenter des tests unitaires, développé par l'équipe ayant créé le *framework* JQuery.

4. Un test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme.

à eux-seuls de tous petits graphes complets dans le cadre *SVG*, ce qui le différencie bien du premier sujet, qui était à lui-seul un seul et même graphe. Ici, nous ne tenons pas compte de la restitution des distances d'édition sur les clusters, étant donné que nous souhaitons juste rassembler les clones et non pas les comparer ! J'ai ainsi établi un nouveau paramétrage du moteur pour cette distribution, ne gérant pas la gravité, la restitution des exactes distances d'édition, ainsi que l'élasticité.

Je me suis alors demandé quelle était la meilleure visualisation possible afin de percevoir les différents clusters - sans aucun doute, la couleur. Chaque clone appartenant à un cluster calculé lors du calcul par l'algorithme, chaque cluster pourra donc avoir une couleur bien spécifique qui pourra donc la distinguer des autres clusters.

La colorisation (voir **Figure 11**) est beaucoup plus bénéfique visuellement et résiduelle dans le temps (il est en effet possible de colorer les clones selon les résultats de DBSCAN, tout en étudiant une distribution quelconque).

Aussi, il a fallu faire attention à plusieurs choses :

- la couleur choisie pour chaque cluster est comprise entre 0 et 270 selon *RGB*<sup>5</sup>
  - afin d'aller d'une teinte rouge (0) vers le violet (270), des couleurs déjà utilisées dans le projet, et permettant aussi de respecter scrupuleusement la légende des couleurs déjà établie par l'ingénieur de recherche du projet,
- il a fallu faire attention à ce que les noeuds se ressemblant n'aient pas une teinte beaucoup trop proche l'une de l'autre, afin de permettre une visualisation non-faussée.

Pour palier à ce problème, j'ai donc créé un objet tableau à N entrées (100 ici, pour utiliser au maximum 100 clusters, soit 1 clone par cluster) représentants chaque cluster et une couleur, que j'ai mélangé via l'**algorithme de Fisher-Yates** (encore appelé **mélange de Knuth**), très utilisé dans ce genre de situation.

---

5. *RGB* : Red, Green, Blue

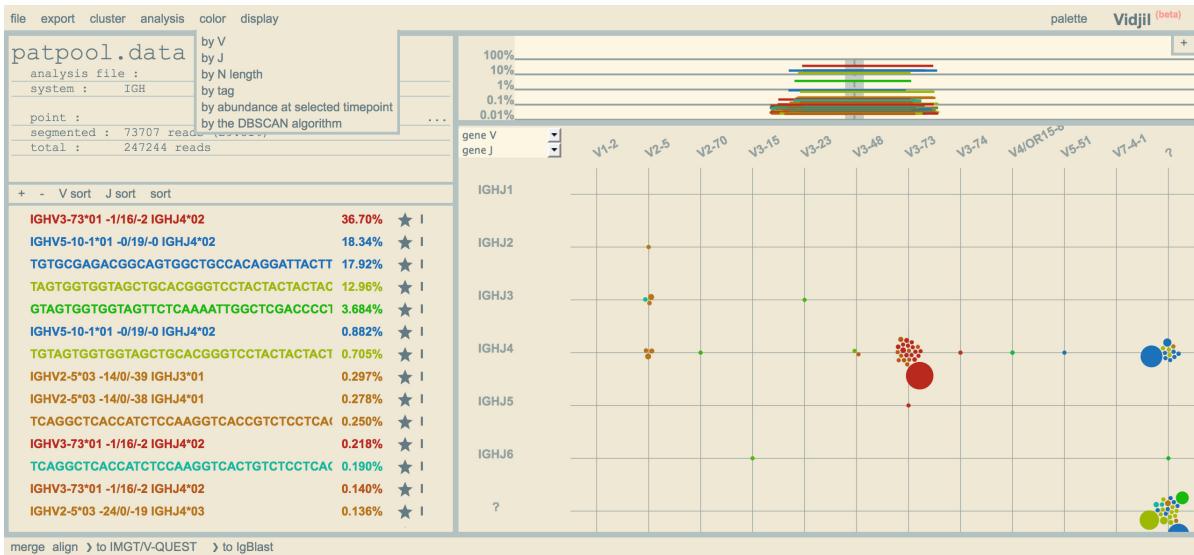


FIGURE 11 – Interface avec la colorisation DBSCAN

Après cela, j'ai tenté une nouvelle approche en voulant permettre de visualiser dans le **menu gauche** non pas les clones, mais les clusters issus de DBSCAN (voir **Figure 12**).

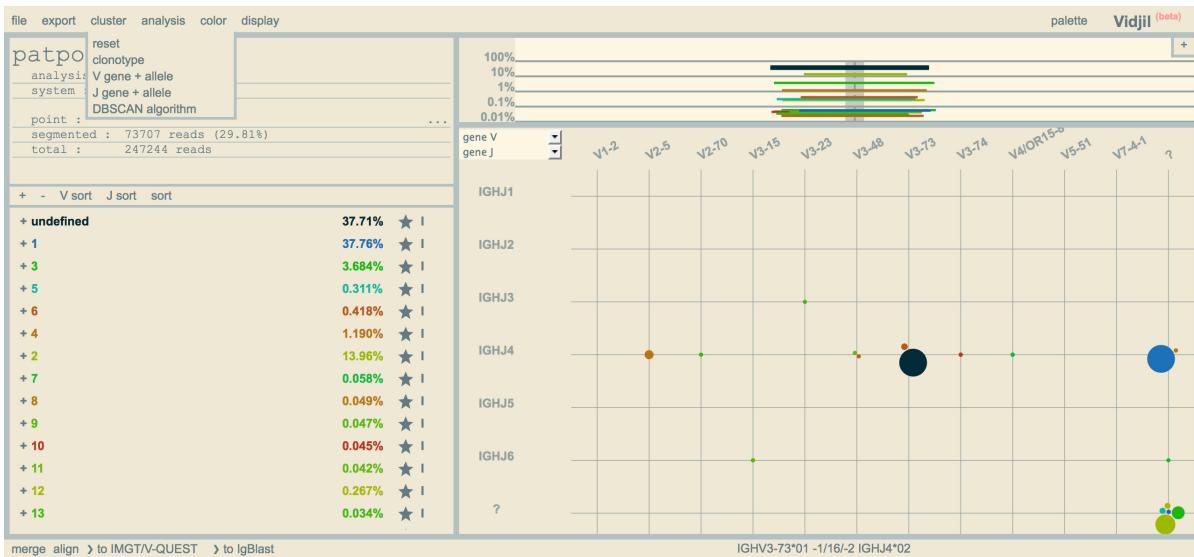


FIGURE 12 – Interface avec la clusterisation DBSCAN

Enfin, en testant plusieurs fois l'algorithme sur des données, je me suis rendu compte qu'il pourra être bénéfique que les clusters se repoussent mutuellement afin de laisser un maximum de liberté à l'utilisateur, et non pas l'inciter à bouger par lui-même les clusters si les clones issus de ceux-ci sont mélangés, mais aussi que les points **points centraux** soient au centre de ceux-ci.

## 4.3 Résultats

L'implémentation de cet algorithme est un vrai succès. En effet, comme nous pouvons le voir sur la **Figure 13**, la distribution restitue bien une visualisation respectueuse du partitionnement effectué en fonction des données, et des deux paramètres précédemment évoqués. Cette seconde partie appartient bien à la visualisation interactive des clones, et est aussi une suite au graphe des distances d'édition, car elle utilise elle aussi ces distances pour fournir un résultat. Aussi, cette distribution permet d'affirmer que la représentation d'un système permettant de visualiser les similarités et différences entre clones V(D)J est bien mieux respectée par l'implantation de plusieurs graphes distincts, présents dans une seule fenêtre *SVG*.

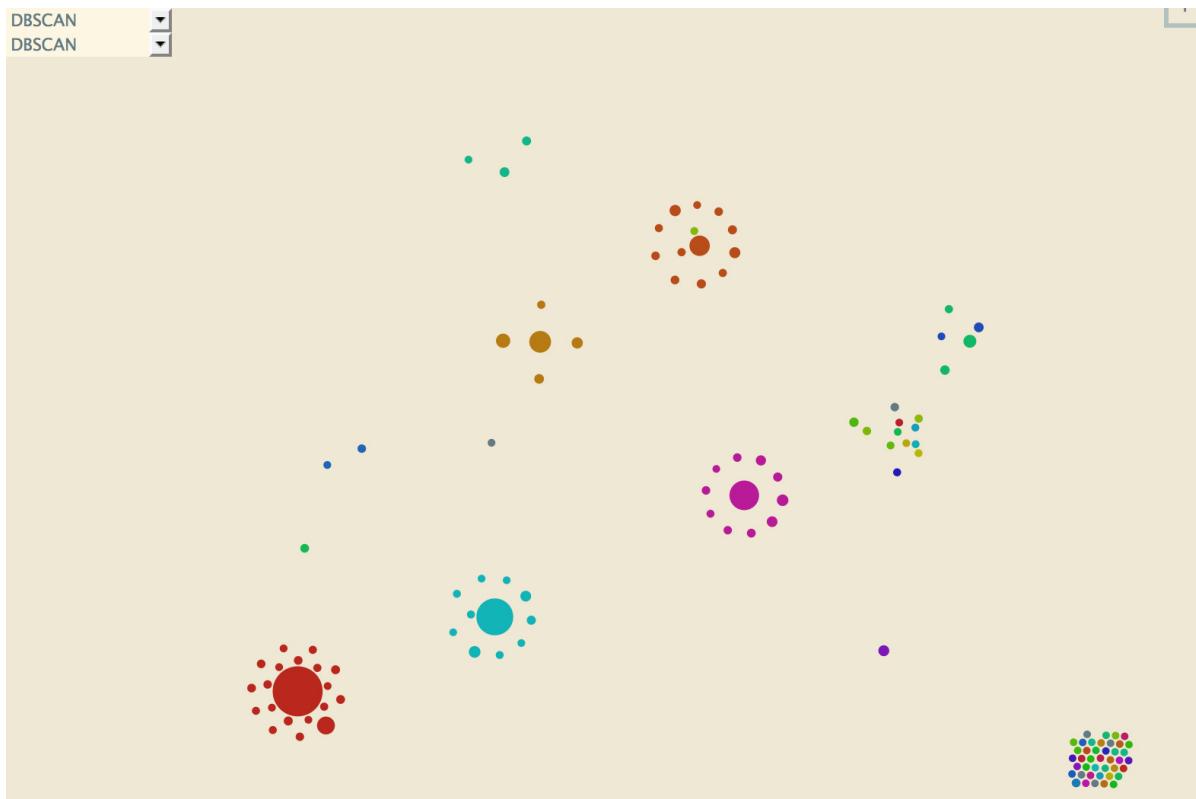


FIGURE 13 – *Interface* avec la distribution DBSCAN

# Chapitre 5

## Recherche et calcul des distances d'édition

Les distances d'édition sont essentielles quant à la résolution de ma problématique. Ce sont elles qui régissent la structure globale des graphes, et ce sont aussi les principales données des deux distributions que j'ai intégré au projet.

Auparavant, nous utilisions comme distances des pourcentages d'identité, correspondants à une différence entre deux séquences représentatives appartenant à deux clones quelconques (calculé dans le programme C++) - grosse approximation basée sur des séquences de nucléotides de différentes tailles, et fabriquées à partir des séquences V(D)J et de séquences prises entre la région V et J (appelées *windows*). Cette approximation n'étant pas des plus respectables quant à tout le travail précédemment effectué, il a donc fallu obtenir des distances d'édition beaucoup plus précises. J'ai eu alors l'idée de comparer les *windows* de chaque clone V(D)J dans le programme C++, et de retourner le résultat dans le fichier JSON qu'il fournit en sortie.

### 5.1 Utilisation des *windows*

#### 5.1.1 Qu'est-ce qu'une *window*

Les *windows* sont des séquences de 40 nucléotides, trouvées directement entre la fin de la région V, et le début de la région J après le séquençage. Ces séquences ont été alignées de façon globale<sup>1</sup>, et non pas locale<sup>2</sup>, comme étaient calculées les pourcentages d'identité.

**Remarque :** Il se peut donc tout à fait que la partie D se retrouve, ou non, dans cette séquence.

---

1. Alignement global : alignement entre deux séquences sur toute leur longueur.

2. Alignement local : alignement entre une séquence et une partie de l'autre séquence

### 5.1.2 Pourquoi les utiliser ?

L'avantage d'une *window* est qu'elle servira, tout au cours du temps, à marquer spécifiquement un et un seul clone. L'on pourra ainsi faire le suivi de la L.A.L. via l'étude des *windows*, spécifiant des zones bien spécifiques (voir **Figure 14**) pouvant être contraintes à évoluer en fonction des mutations infligées, ce qui n'était pas possible avec les séquences représentatives.

```
#### Clone #040 -      300 reads - 0.100% - 0.100% 1.000 (similar to Clone #001)
>clone-040-window
TCTGTGACTCCCGAGGAGTGCTACAGTAACCCCCTTGGCTACTGCCAGGGAACCTGGT
>clone-040-representative - VDJ      0 256 260 269 273 315  IGHV6-1*01 27/GTG/3 IGHD4-11*01 3/CCC/4 IGHJ
4*02
ACCTGAGGAGACGGTGACCAGGGTCCCTGGCAGTAGCCAAAGGGGTTACTGTAGCACTCCTCGGGAGTCACAGAGTTAGCTGCAGGGAGAACTGGTTCTGGATG
TGTCTGGGTGATGGTTATTCGACTTTACAGATACTGCATAATCATTATACCACTGGACCTGTAGTATGTCCTTCCAGCCACTCAAGGCCTCTCGATGGGGACT
GCCGTGATCCAGTTCCAAGCAGCACTGTTGCTAGAGACACTGTCCCCGGAGATGGCACAGGTGAGTGAGGGTCTGCGAGGGCTTCACCAAGTCCTGGACCC
```

FIGURE 14 – La *window* du 40ème clone, dans le programme C++, est ici représentée par 'clone-040-window'

## 5.2 Les différents coûts utilisés

Les distances d'édition sont calculées grâce à la programmation dynamique<sup>3</sup>, ce qui nous permettra d'obtenir, pour deux mots *x* et *y*, le nombre minimal d'opérations d'édition<sup>4</sup> pour transformer *x* en *y* - ce nombre minimal étant la distance d'édition recherchée.

### 5.2.1 Les coûts étudiés

#### Sur quoi est basé un coût ?

Chaque calcul effectué se fait sur un coût, constitué de 5 opérations :

- le *match* : la lettre de la séquence 1 sur la position actuelle correspond bien à celle de la séquence 2,
- le *mismatch* (ou substitution) : la lettre de la séquence 1 sur la position actuelle ne correspond pas à celle de la séquence 2,
- l'insertion,
- la délétion,
- la détection d'un homopolymère : motif de répétition, le plus souvent attribué par une erreur dans le séquenceur.

3. La programmation dynamique est une technique algorithmique pour optimiser des sommes de fonctions monotones croissantes sous contrainte.

4. Il y a 3 opérations possibles : la **substitution** (le remplacement d'une lettre par une autre - nous utiliseront aussi le mot *mismatch* pour cette opération), l'**insertion** (l'ajout d'une nouvelle lettre), et la **délétion** (la suppression d'une lettre)

## Quels sont ces coûts ?

Pour ces opérations est attribué un poids (n'importe quel entier positif, négatif ou nul), permettant de régir et de calculer une distance, spécifique en fonction de ce poids.

Ainsi, deux coûts ayant des poids sur ces opérations différentes ne devraient pouvoir obtenir le même résultat (qui sera une distance) à la fin du calcul.

Il est important de calculer de bons coûts, car ils influeront directement sur les distributions précédemment mises en place. Nous en avons donc sélectionner 4 :

- le coût de **Levenshtein** : (0, -1, -1) : 0 en *match*, -1 en *mismatch*, et -1 en insertion et délétion - la détection d'homopolymère(s) est considéré comme "gratuit" ici, donc ne pèse rien,
- le coût **DNA** : (+5, -4, -10) : 5 en *match*, -4 en *mismatch*, et -10 en insertion et délétion - la détection d'homopolymère(s) est aussi considéré comme "gratuit",
- le coût **VDJ** : (+4, -6, -10, -1, -2) : 4 en *match*, -6 en *mismatch*, -10 en insertion et délétion, -1 pour les délétions de fin de séquence<sup>5</sup>, -2 pour la détection d'homopolymère(s),
- le coût **Identity** : (+1, -1, -1, 0, 0) : 1 en *match*, -1 en *mismatch*, -1 en insertion et délétion, rien pour la détection d'homopolymère(s).

Sur ces 4 coûts, il a fallu choisir celui ayant les plus respectueuses distances par rapport aux similarités entre séquences, afin de perfectionner les distributions.

	<b>G</b>	<b>A</b>	<b>C</b>	<b>G</b>	<b>A</b>	<b>T</b>	<b>T</b>	<b>A</b>
<b>G</b>	0	-1	-1	0	-1	-1	-1	-1
<b>A</b>	-1	0	-1	-2	0	-1	-2	-1
<b>T</b>	-1	<b>-1</b>	-1	-2	-1	0	0	-1
<b>C</b>	-1	-2	<b>-1</b>	-2	-2	-1	-1	-1
<b>G</b>	0	-2	-2	<b>-1</b>	-2	-2	-2	-2
<b>A</b>	-1	0	-1	-2	<b>-1</b>	-2	-3	-2
<b>T</b>	-1	-1	-1	-2	-2	<b>-1</b>	-1	-2
<b>T</b>	-1	-2	-2	-2	-3	-1	<b>-1</b>	<b>-2</b>

TABLE 1 – Calcul de distance d'édition entre 2 séquences ADN (avec **Levenshtein**). La distance est entourée - alignement GA\_CGATTA et GATCGATT\_ (le chemin permettant de trouver ces résultats est marqué en **gras**)

5. Imaginons que, pour un alignement de deux séquences de longueurs différentes, si la partie de l'une est beaucoup plus grande que l'autre, nous pouvons compter une délétion sur la partie grande (et non-alignée) avec un poids différent.

## Comment effectuer les calculs de coût ?

Les coûts étaient déjà implantés à mon arrivée dans l'entreprise, et le programme en C++ contenait déjà un objet (DynProg) prenant en paramètre les deux séquences ADN, l'alignement ainsi que le coût utilisé lors du calcul, et nous renvoyant une matrice de similarité que j'ai transformé en matrice de distances.

Ainsi, pour étudier les objets *windows* entre eux, il a juste fallu modifier l'alignement local (pris par défaut) par l'alignement global, et le coût par un de notre choix - étant à tester. La matrice est à calculer ou non, en fonction du poids choisi lors du calcul dans la programmation dynamique.

## 5.3 Conclusion

Les coûts *VDJ*, *Identity* et *Levenshtein* sont les plus probants pour les graphes implémentés. En effet, ce sont eux qui, lors des tests, permettent d'avoir des distances aussi variées que respectueuses des similarités entre clones - ce qui a été vérifié avec un objet Javascript "Stats" (voir **Figure 15**), implanté directement dans le *scatter-plot* en mode *console*, permettant d'obtenir des statistiques sur les arêtes enregistrées par cette vue (comme la moyenne, la distribution ou la médiane par exemple).

Dû à sa grande popularité, mon choix du coût pour le calcul des distances d'édition entre les *windows* s'est porté vers le coût *Levenshtein*, nous permettant aussi d'obtenir directement des distances d'édition lors du calcul, après les avoir transformées en valeur absolue.

```
stats.usage()
undefined
/**printStats(): "
"      Fonction permettant d'afficher dans la sortie standard la moyenne des longueurs des arêtes du jeu de données
"
/**printLayout(bool, gap): "
"      Permet d'afficher en mode console une représentation de la distribution des arêtes"
"          bool -> True: Affiche seulement les valeurs dont la distribution est supérieure à 0"
"          gap: écart (10 par défaut)
"

/**printMedian(): "
"      Permet d'afficher en mode console la médiane du jeu de données étant analysé
"
stats.printLayout(true)
undefined
"[0,10]: 0.07717171717171717 %"
"[10,20]: 0.3597979797979798 %"
"[20,30]: 0.2222222222222222 %"
"[30,40]: 0.21313131313131314 %"
"[40,50]: 0.12767676767676767 %"
```

FIGURE 15 – Aperçu de l'objet "Stats", sur un jeu de données avec les distances d'édition de *Levenshtein*

# Chapitre 6

## Le travail en recherche

Je voulais effectuer mon premier stage Universitaire dans une structure de recherche, étant motivé par l'idée de participer aux solutions et innovations de demain. Ayant effectué un DEUST de Biologie avant de postuler pour une licence d'Informatique à l'Université de Lille1, et étant toujours intéressé par ce domaine de la Science, j'étais très curieux et intéressé de pouvoir effectuer mes premiers pas dans une équipe de bio-informaticiens

### 6.1 La vie est un long fleuve tranquille...

Mon stage dans l'équipe de recherche *Bonsai* a été un véritable plaisir.

Eternel curieux, je suis à la recherche perpétuelle d'informations de tous genres pouvant me permettre de progresser dans la vie, mais aussi de m'instruire et me cultiver. L'intérêt et la curiosité de chaque membre de l'équipe envers la Science (à proprement parlé) est imparable, chacun ayant une certaine spécificité en fonction de son parcours et ses intérêts, et qui en fait quelqu'un d'unique dans l'équipe, lui permettant d'échanger avec ses collègues sur ses intérêts dans sa recherche de résultats ou ceux des autres.

J'ai pu ressentir cela par :

- **les conférences/séminaires du Mardi matin**

Chaque mardi matin, généralement entre 11H et 12H, était programmé un séminaire sur un sujet déterminé par le conférencier : un membre de l'équipe *Bonsai* ou un intervenant extérieur.

Nous avons donc pu avoir le plaisir de recevoir des chercheurs de Pennsylvanie et d'un autre État de l'Amérique Centrale, venus nous parler de nouvelles méthodes de recherche, d'attrait à celle-ci, ou des dernières innovations et recherches effectuées sur le sujet donné (*the International Mouse Phenotype Database*)

*ping Consortium*<sup>1</sup>, les graphes de Bruijn<sup>2</sup> ou encore la détection des variants complexes<sup>3</sup>) - mais pas seulement !

Cela a aussi permis à certains membres de l'équipe de pouvoir exposer ses travaux finis ou en cours, afin de pouvoir expliquer son cheminement, sa méthode de travail et ses résultats, comme par exemple le travail d'un membre de l'équipe réalisé pendant 6 mois, en 2013.

#### — **les entraides perpétuelles**

Chaque membre de l'équipe peut compter sur les autres s'il a un problème - en effet, l'entraide est omniprésente dans l'équipe, que ce soit pour une aide informatique, algorithmique ou encore de compréhension biologique. Un bref exemple à donner seraient ceux m'ayant permis de rassembler plus de 5 algorithmes différents, de 5 personnes différentes, pour mon problème de représentation respectueuse d'un graphe à 1000 arêtes.

#### — **la pause déjeuner**

Lors de la pause déjeuner, l'heure est à la discussion scientifique la plupart du temps, et les débats font rage, chacun exposant sa thèse, annonçant et dénonçant un ou plusieurs argument(s) contraire(s) à sa propre vision, etc...

Tout ceci m'a permis de me créer une belle idée de ce qu'est le monde de la recherche sur le plan relationnel et scientifique, et de me sentir beaucoup plus à l'aise dès la première semaine de stage.

## 6.2 ...ou pas !

Ce stage m'a permis aussi de me rendre compte de tous les problèmes liés à la recherche, que ce soit pour l'accréditation des sujets de recherche, les problèmes liés aux actions en dehors de sa thèse (le temps attribué à l'éducation, ou encore aux services rendus à l'Université par exemple), ou encore plus directement aux problèmes touchant à son sujet de thèse (la publication du sujet en est un exemple valable).

Je me suis rendu compte qu'un travail de recherche représentait des heures incalculables à rechercher, programmer, analyser et rédiger - il faut en moyenne plus d'un an (si tout se passe correctement) pour arriver à des résultats valant la peine de les publier.

La problématique de la thèse mérite un temps considérablement long à être développée, approfondie et traitée, afin de pouvoir enfin réellement commencer à effectuer

---

1. Entreprise Internationale scientifique créant et caractérisant le phénotype de 20 000 *knock-out* (ou "invalidation génétique" en Français) chez la souris

2. Graphe orienté qui permet de représenter les chevauchements de longueur ( $n-1$ ) entre tous les mots de longueur  $n$  sur un alphabet donné - nom donné par le mathématicien les ayant décrit en 1946 : Nicolaas Govert de Bruijn

3. Variants cytogénétiques constituant une majeure partie des cas de Leucémie myéloïde chronique

ses propres recherches - un peu plus longtemps qu'un nouveau sujet de recherche en général, avec l'expérience, les compétences et la culture en plus.

Tout ceci est lié à un rythme de publication assez rapide et drastique, qui ne permet pas réellement aux chercheurs de pouvoir travailler dans de bonnes conditions.

La vigilance est aussi de mise quant à la publication des recherches de scientifiques, sur un sujet similaire de celui sur lequel la personne travaille. En effet, un article paraissant à une période très proche d'un autre traitant du même sujet, mais déjà publié, pourrait être refusé en publication pour ce motif.

# Conclusion

## Résolution de la problématique

Concernant le graphe de distances d'édition, nous ne pouvions pas savoir si le moteur allait réagir correctement par rapport à des distances précises, mon stage étant un test par rapport à son implémentation. N'ayant pas réussi à obtenir le résultat voulu pour cette distribution, il serait préférable de penser au graphe de distances d'édition comme un arbre phylogénétique, ou comme un ensemble de plusieurs graphes comme la distribution DBSCAN. Quant à cette dernière, son implantation a été réalisé avec succès, et démontre bien les ressemblances et disparités entre les clones V(D)J séquencés. Par rapport au projet *Vidjil*, l'équipe M. Giraud et M. Salson sont en pleine rédaction du nouvel article, M. Duez perfectionnant quant à lui le serveur sur l'*interface*.

## Bilan personnel

J'ai énormément appris durant ce stage, notamment quant à la conception logicielle - ayant fait de l'ingénierie logicielle durant plusieurs semaines. C'est ce qui m'a appris à me confronter à un projet déjà existant et pouvoir rentrer au coeur du sujet rapidement (la plus grosse difficulté du stage pour moi), à apprendre de nouveaux langages (comme C++) et *frameworks* (D3JS, JQuery) informatiques, ainsi que confronter mes idées avec celles des autres et optimiser au possible mon travail, s'il y a nécessité. De plus, ce stage m'a beaucoup apporté quant au travail en groupe (le respect du travail de mes collègues, les réunions à prévoir, le respect des dead-lines, l'utilisation d'outils spécifiques comme *Git*, etc...), dans mon organisation personnelle, et m'a permis de confronter le milieu scolaire et le milieu professionnel. Mais ce qui est le plus important à mes yeux, c'est que celui-ci m'a bien confirmé mon idée de poursuivre dans le monde de la recherche, un milieu extrêmement riche composé par le travail, la réflexion, l'envie d'apprendre et d'échanger.

# Bibliographie - Webographie

- *Fast multiclonal clusterization of V(D)J recombinations from high-throughput sequencing*, M. Giraud, M. Salson. *BMC Genomics* 2014. 28 Mai 2014.  
Consultable sur <http://www.biomedcentral.com/1471-2164/15/409>
- *La recombinaison V(D)J illégitime, et le développement de Leucémies Aigües Lymphoblastiques*, B. Montpellier, B. Nadel. Thèse de doctorat - Aix Marseille 2. 2008.  
Consultable sur <http://www.worldcat.org/title/recombinaison-vdj-illegitime-et-de-oclc/495056914> - consulté le 10 Avril 2014.
- *Clustering, Part IV*, Dr. Sanjay Ranka, Université de Floride, Gainesville.  
Consultable sur <http://www.cise.ufl.edu/class/cis4930sp09dm/notes/dm5part4.pdf> - consulté le 22 Mai 2014.
- *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, M.Ester, H-P Kriegel, J. Sander, X. Xu. 1996.  
Consultable sur <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.4045&rep=rep1&type=pdf> - consulté le 21 Mai 2014.
- *Recombinaison VDJ*  
Consultable sur **Wikipédia** [http://fr.wikipedia.org/wiki/Recombinaison\\_V%28D%29J](http://fr.wikipedia.org/wiki/Recombinaison_V%28D%29J) - consulté le 2 Avril 2014.
- *DBSCAN*  
Consultable sur **Wikipédia** <http://fr.wikipedia.org/wiki/DBSCAN> - consulté le 21 Mai 2014.
- *Les distances de Levenshtein*  
Consultable sur **Wikipédia** [http://fr.wikipedia.org/wiki/Distance\\_de\\_Levenshtein](http://fr.wikipedia.org/wiki/Distance_de_Levenshtein) - consulté le 5 Juin 2014.