Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière informatique

Projet de semestre d'autome

2014-2015

Visualisation des interactions entre gènes humains

Rapport

Auteur :
Maria Sisto

Responsable interne :
M. Pierre Kuonen
M. Rudolf Scheurer
M. Beat Wolf

Responsable externe :
M. David Atlan
(PhenoSystems SA)

Rendu le
29 janvier 2015

0 Table des matières

1 Introduction **1**

1.1 Contexte 1

1.2 Glossaire 1

1.3 But 2

1.4 Structure du rapport 2

2 Analyse **3**

2.1 Interactome 3

2.1.1 Représentation 3

2.1.2 Graphe 4

2.2 Interactome annoté 4

2.2.1 Représentation 4

2.2.2 Graphe 5

2.3 Interactome augmenté 5

2.3.1 Représentation 5

2.3.2 Graphe 6

2.4 Recherche dans le graphe 6

2.4.1 Recherche à implémenter 6

2.5 Etat de l'art 6

2.5.1 Logiciels généraux 7

2.5.2 Logiciels spécialisés 7

2.6 Outils de visualisation 7

2.7 Génération du fichier de sortie 8

2.7.1 Format 8

2.7.2 Quels sont les formats disponibles ? 8

2.7.3 Description des formats 9

2.7.4 Choix du format 10

3 Modélisation **11**

3.1 Génération du graphe de l'interactome . . . 11

3.1.1 Parseur . . . 12

3.2 Annotations des gènes . . . 13

3.3 Annotations du patient . . . 14

3.4 Génération du graphe de sortie . . . 15

3.5 Diagramme de classe . . . 16

3.6 Tests . . . 18

3.6.1 Test unitaires . . . 18

3.6.2 Tests d'intégrité . . . 18

3.7 Méthodes de recherche . . . 18

3.7.1 Outil . . . 18

3.7.2 Plugin Gephi : fonctionnement général . . . 18

3.7.3 Plugin Gephi : recherche concrète . . . 20

3.7.4 Format et attributs attendus en entrée : . . . 21

3.7.5 Affichage . . . 21

4 Implémentation **23**

4.1 Outils utilisés . . . 23

4.1.1 Génération du fichier du graphe . . . 23

4.1.2 Visualisation . . . 23

4.1.3 Recherche dans le graphe . . . 23

4.2 Problèmes rencontrés . . . 23

4.2.1 Général . . . 23

4.2.2 Plugin filtre gephi . . . 24

4.3 Résultats . . . 24

4.3.1 Visualisation de l'interactome avec le layout OpenOrd . . . 24

4.3.2 Application sur un sous-graphe . . . 25

5 Conclusion **26**

5.1 Travail réalisé . . . 26

5.2 Perspectives . . . 26

5.2.1 Améliorations . . . 26

5.2.2 Développement futur . . . 26

5.3 Déclaration sur l'honneur . . . 27

6 Annexes **29**

6.1 Glossaire 29

6.2 Mode d'emploi : Comment créer un plugin Gephi de type filtre 29

6.2.1 Mise en place de l'evironnement de développement 29

6.2.2 Création d'un plugin 29

6.2.3 Création d'un filtre 30

6.2.4 Ajout d'un paramètre : 30

6.3 Installation 30

6.3.1 Installation du générateur de ficher de graphe 30

6.3.2 Installation du plugin Gephi 30

6.4 Contenu du CD 31

1 Introduction

1.1 Contexte

L'étude des gènes et de l'ADN humain est un domaine de recherche très actuel, particulièrement dans le domaine de la recherche bio-médicale.

Les maladies génétiques sont des maladies dues à une ou des anomalies dans les séquences ADN. Afin de pouvoir mettre en évidence les variations de code génétique chez un individu, les scientifiques ont procédé au séquençage du génome humain, c'est-à-dire à la mise en évidence des différents gènes présents dans le code génétique. Une fois cela fait, il a été possible d'établir une liste de gènes communs aux être humains en bonne santé. En effet, bien que l'on dise, à juste titre, que le code génétique d'un individu est unique, nous partageons 99.99% de notre code génétique avec les autres humains.

Une fois le génome séquencé, il a été possible d'identifier les interactions entre lesdits gènes. Ceci est un travail encore en cours sur lequel plusieurs équipes de recherche travaillent actuellement. Les données trouvées sont appelées « *interactome* ».

Toutes ces découvertes permettent de nouvelles recherches sur le lien entre les maladies et les gènes variants chez un individu. Il n'est cependant pas évident de trouver ces liens lorsque l'on a un patient avec des symptômes et une liste de gènes reconnus comme variants, mais non référencés comme responsables desdits symptômes.

Le but de ce projet est de permettre une visualisation graphique de ces gènes et de leurs interactions, afin de permettre une mise en rapport entre les gènes variants d'un patient et sa maladie.

1.2 Glossaire

— Gène variant : Gène présentant une variation par rapport au modèle établi.
— Interactome : Contient les interactions entre les gènes (commun à tous les humains).
— Interactome annoté : Contient l'interactome avec des annotations concernant les gènes (génotypes)
— Interactome augmenté : Contient l'interactome annoté augmenté des données du patient.

1.3 But

Comme dit précédemment, le but de ce projet est de permettre une meilleure compréhension de l'interaction entre les gènes, et en particulier de mettre en lien des variations sur les gènes avec certaines maladies.

De manière plus concrète, il s'agit de représenter l'interactome augmenté sous la forme d'un graphe à partir de fichiers de données fournis. Sur ce graphe, il devra être possible de mettre en évidence les gènes variants de patients et de permettre un certain nombre de requêtes sur ce graphe.

1.4 Structure du rapport

Nous allons tout d'abord faire une analyse du problème de base ainsi que des données dont nous disposons (données et outils).

Nous allons ensuite parler plus concrètement de la structure du programme et de comment le réaliser.

Finalement, nous allons aborder le sujet des problèmes rencontrés et des résultats, ainsi que des perspectives et améliorations pour ce projet.

2 Analyse

2.1 Interactome

L'interactome est une liste d'interactions entre deux gènes.

Un gène est en fait une portion du code génétique. Nous pouvons le repérer grâce à des séquences génériques indiquant le début et la fin du gène. La seule manière pour le gène de communiquer est d'être lu. Sa lecture provoquera par exemple la production d'une protéine, qui pourra ordonner la lecture d'un autre gène. Il n'est cependant pas facile de déterminer quels sont les critères d'activation d'un gène, c'est pourquoi nous possédons pour l'instant plusieurs fichiers contenant les interactions, issus de différentes études ou contenant des relations plus ou moins sûres.

2.1.1 Représentation

Concrètement, nous avons deux gènes interagissants l'un avec l'autre, représentés de la manière suivante :

entrez_gene_ida	symbol_a	entrez_gene_idb	symbol_b
9454	HOMER3	170680	PSORS1C2
337959	KRTAP13-2	340359	KLHL38
340419	RSPO2	386675	KRTAP10-7

TABLE 1 – Exemple concret de ce qui peut être trouvé dans le fichier de l'interactome (ici CCSB).

Chaque gène est identifié de manière unique par une suite de lettres et de chiffres (appelons cela le "nom" du gène). Ce nom est le point commun entre la plupart des recherches sur les gènes. Nous voyons cependant que dans notre exemple chaque gène a un code numérique associé. L'utiliser faciliterait grandement le traitement, malheureusement il ne semble pas y avoir de convention sur cette numérotation et presque chaque projet de référencement de l'interactome en utilise une différente. Nous sommes donc contraints d'utiliser le nom, moins pratique mais communs aux différentes sources que l'on est sur le point d'utiliser.

2.1.2 Graphe

Nous avons choisi de représenter l'interactome sous forme de graphe, les gènes étant les nœuds , et les arêtes les interactions. L'avantage des graphe est la facilité de recherche par proximité. En effet, nous nous intéressons particulièrement à trouver des gènes à une certaine distance d'autres gènes.

gène	gène
Gene1	Gene2
Gene1	Gene3

TABLE 2 – Exemple générique

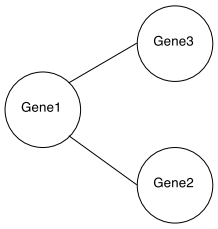


FIGURE 1 – Exemple générique

2.2 Interactome annoté

Le séquençage des gènes à permis d'établir, pour chaque gène, un "modèle" qui représente ledit gène chez un individu sain. Lorsqu'un gène diffère d'avec le modèle établi, nous appelons ce gène « *gène variant* ».

Des recherches sont actuellement en cours afin d'identifier certains variants comme responsables de certains symptômes chez un humain, ces symptômes sont appelés « *phénotype* » du gène. Nous voulons garder trace de ces phénotypes de manière à pouvoir facilement les retrouver lors du traitement des données (dans le cas d'une recherche par exemple.

2.2.1 Représentation

HPO-ID	HPO-Name	Gene-ID	Gene-Name
HP :0001459	1-3 toe syndactyly	2737	GLI3
HP :0006088	1-5 finger complete cutaneous syndactyly	64327	LMBR1
HP :0010708	1-5 finger syndactyly	64327	LMBR1
HP :0010713	1-5 toe syndactyly	2737	GLI3

TABLE 3 – Exemple concret de ce qui peut être trouvé dans le fichier des phénotypes des gènes

Nous utilisons le nom des gènes pour l'identification, noms qui doivent être identiques à ceux utilisés dans l'interactome . En effet, comme pour les interactions, les numéros de classification des gènes et des phénotypes ne sont pas identiques selon les recherches, nous ne pouvons donc nous fier qu'au nom du gène, généralement constant dans les différentes recherches, et au nom du phénotype, qui lui est constant à l'intérieur d'une recherche (donc d'un fichier).

2.2.2 Graphe

gène	phénotype
Gene1	Phénotype1
Gene2	Phénotype2
Gene31	Phénotype3

TABLE 4 – Exemple générique

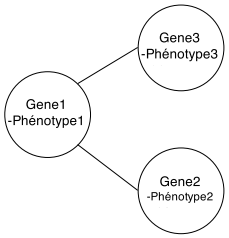


FIGURE 2 – Exemple générique

2.3 Interactome augmenté

Pour chaque gène d'un individu (disons un patient), il est possible de le comparer avec le modèle de gène établi afin de détecter une éventuelle variation. Après avoir passé tous les gènes en revue, nous obtenons une liste de gène variants pour un patient.

Un gène variant peut avoir plusieurs manières de varier. Ces différentes manières sont appelés « *génotypes* » et ont chacun une gravité (plus le chiffre est haut, plus la gravité est haute).

2.3.1 Représentation

Le fichier du patient contient, en plus du nom du patient, les gènes variants ainsi que tous les génotypes connus sur ce gène.

Nom_Gene	ID_Gene	génotypes
PNPLA3	609567	8,Missense ;15,Intronic variation
LONRF1		8,Missense ;10,Synonymous ;15,Intronic variation
IQCE		10,Synonymous ;9,Splice site ;15,Intronic variation
TST	180370	10,Synonymous

TABLE 5 – Exemple d'une liste de gène variant pour un patient, incluant une liste de génotype pour chaque gène

2.3.2 Graphe

Exemple pour un patient :

gène	génotypes
Gene1	Génotype1

TABLE 6 – Exemple générique

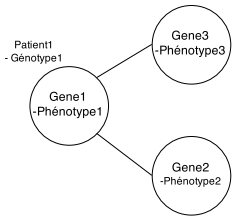


FIGURE 3 – Exemple générique

2.4 Recherche dans le graphe

Notre but est de mettre en évidence les relations entre les gènes identifiés comme variants chez un patient et ses symptômes. Plus particulièrement, trouver les gènes variants du patients et les gènes porteurs d'un certain phénotype à une distance maximale fixée. Il faut aussi réussir à identifier les nœuds intermédiaires permettant un chemin entre les gènes variants et les gènes porteurs du phénotype.

La structure d'arbre se prête particulièrement bien à ce genre de recherche.

2.4.1 Recherche à implémenter

La recherche que nous avons choisie est la suivante : Nous souhaitons afficher les gènes d'un patient donné se trouvant à une distance d ou moins d'un gène porteur d'un certain phénotype. Il faut aussi afficher les gènes faisant partie des chemins entre ces gènes.

2.5 Etat de l'art

Le nombre de logiciels permettant une visualisation d'interactions entre les gènes est important. Nous nous intéressons ici à ceux permettant une visualisation sous forme de graphe. Nous nous retrouvons donc avec deux catégories de logiciel :

— Les logiciels généraux de traitement de graphes
— Les logiciels spécialisés pour la visualisation de gènes

2.5.1 Logiciels généraux

Deux logiciels généraux permettent le traitement de gros graphes et possèdent des plugins permettant le traitement de graphes d'interactome . Voici les deux logiciels principaux trouvés :

- Cytoscape
- Gephi

Malheureusement, pour Gephi, il n'y a pas de plugin proposé spécifiquement pour traiter notre problème, et les plugins proposés pour Cytoscape se sont tous révélés associés à une base de données d'interactions, ne permettant ni l'ajout d'annotations sur les nœuds ni l'utilisation de sources différentes.

Exemples :

- BioGRID plugin
- MiMi plugin
- Reactome FI

2.5.2 Logiciels spécialisés

Ces logiciels sont pointus dans leur domaine, mais ne permettent pas de liberté. Voici quelques exemples :

- EINVis : `http://www.ybwu.net/einvis/`
- Navigator : `http://ophid.utoronto.ca/navigator/`
- IGV : `http://www.broadinstitute.org/igv/`
- ...

Mais ces outils ne sont pas très souples. Il permettent la pluspart du temps un seul layout et des recherches prédéfinies. Il n'y a donc pas beaucoup de liberté dans l'utilisation de ces outils.

2.6 Outils de visualisation

Critères désirés pour l'outil de visualisation :

- Traitement de gros graphes
- Changement de layout (manière d'afficher le graphe)
- Definition de recherches personnalisées
- Annotations personnalisées sur les nœuds et les arêtes
- Coloration possible des nœuds

Deux logiciels ont été retenus comme satisfaisant ces critères : Gephi et Cytoscape. Ils permettent tous les deux de créer des plugins (option qui sera finalement choisie pour la recherche).

Pour les départager, voici une liste de critères utilisés :

1. Facilité de développement d'un plugin (documentation et communauté)
2. Variété des formats supportés en entrée et en sortie
3. Variété de plugins existants
4. Facilité d'utilisation de logiciel de manière générale

5. Rapidité de traitement des gros graphes

	Cytoscape	Gephi
Facilité de développement d'une extension	-	+
Variété des formats supportés	+	+
Varié des plugins existants	+	+
Facilité d'utilisation de logiciel	-	+
Rapidité de traitement	-	-

2.7 Génération du fichier de sortie

2.7.1 Format

2.7.2 Quels sont les formats disponibles ?

Dans le monde de la visualisation de graphes, de multiples formats existent déjà, plus ou moins spécifiques à un domaine et permettant plus ou moins de libertés. Comme nous l'avons déjà vu, seul deux logiciels ont été retenus comme candidats pour l'utilisation dans ce projet.

Voici donc les formats d'entrée pris en charge par deux visualiseurs de graphes : Gephi[2] et Cytoscape[3]. L'idéal étant de trouver un format qui :

1. soit supporté par les deux logiciels
2. permette un moyen facile de les parcourir (par exemple sur une structure xml)

Le premier critère est important afin de pouvoir laisser le choix à l'utilisateur du programme à utiliser, mais aussi pour nous permettre de repousser le choix du visualiseur à une étape plus avancée du projet.

Le deuxième critère se révèle finalement de peu d'intérêt, la recherche dans le graphe ayant finalement été exécutée sous forme de plugin dans un programme de visualisation.

Format	Gephi	Cytoscape	structure xml
CSV	x	-	-
DL Ulcinet	x	-	-
Dot Graphviz	x	-	-
GDF	x	-	-
GEXF	x	-	x
GML	x	x	-
GraphML	x	(x)*	x
NETPajek	x	-	-
TLP Tulip	x	-	-
VNA Netdraw	x	-	-
SFI	-	x	-
XGMML	-	x	x
SBML	-	x	x
BioPax	-	x	-
PSI-MI	-	x	-

*Pas officiellement supporté, mais fonctionne « *de facto* »

Nous voyons malheureusement qu'aucun format ne remplit ces trois critères (sauf GraphML, mais qui n'est pas supporté officiellement par Cytoscape au moment du projet). J'ai donc décidé d'examiner les formats remplissant deux de ces critères et gardant à l'esprit que le format choisi devra permettre des annotations de tous types sur les nœuds .

2.7.3 Description des formats

GML[4] : Seul format lisible par les deux logiciels. N'est pas basé sur une structure xml mais permet de mettre des attributs sur les noeuds.

```
graph [
        comment "This is a sample graph"
        directed 0
        id 42
        label "Hello, I am a graph"
        node [
                id 1
                label "node 1"
                thisIsASampleAttribute name1
        ]
        node [
                id 2
                label "node 2"
                thisIsASampleAttribute name2
        ]
        edge [
                source 1
                target 2
                label "Edge from node 1 to node 2"
        ]
]
```

GraphML[5] : Format à structure xml, lisible par Gephi. Permet des annotations sur les nœuds et sur les arêtes.

```
<?xml version="1.0" encoding="UTF-8"?>
 <graphml xmlns="http://graphml.graphdrawing.org/xmlns"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
 http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

 <graph id="Sample Graph" edgedefault="undirected">

 <key id="thisIsASampleAttribute" for="node" attr.name="thisIsASampleAttribute"
     attr.type="string"/>

 <node id="1">
   <data key="thisIsASampleAttribute">name1</data>
 </node>
```

```
  <node id="2">
    <data key="thisIsASampleAttribute">name1</data>
  </node>

  <edge id="12" source="1" target="2"/>

  </graph>
</graphml>
```

Malgré le fait que le format ne soit pas dans la liste des formats supportés par Cytoscape, il reste possible d'ouvrir un graphe graphml avec ce logiciel.

GEXF[6] : Format à structure xml, lisible par Gephi. Cela semble être un format "inventé" par Gephi, ce qui me pousse à l'écarter.

XGMML[7] : Format à structure xml, lisible par Cytoscape. Il s'agit d'une évolution de GML et permet des annotations sur les noeuds et sur les arêtes.

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <graph label="Sample Graph"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cy="http://www.cytoscape.org"
    xmlns="http://www.cs.rpi.edu/XGMML"
    directed="0">

  <node label="node 1" id="1">
    <att name="thisIsASampleAttribute" type="string" value="name1"/>
  </node>
  <node label="node 2" id="2">
    <att name="thisIsASampleAttribute" type="string" value="name2"/>
  </node>

  <edge label="1-2" source="1" target="2"/>
</graph>
```

SBML[8] : Format à structure xml, lisible par Cytoscape. Format très spécialisé dans la biologie et les interactions chimiques. Trop spécialisé pour l'utilisation que l'on voudrait en faire.

2.7.4 Choix du format

Après cette analyse, il nous reste trois formats : GML, XGMML et GraphML.
Le format basé xml a été favorisé dans la perspective de recherches futures sur le fichier du graphe généré. Cependant, ces recherches pourraient se faire sur une base de donnée de graphes (Graph database) ou alors avec les données chargées dans le programme. Le format le plus simple et lisible par les deux logiciels a finalement été choisi : le format GML.

3 Modélisation

3.1 Génération du graphe de l'interactome

La génération du graphe se fait en 7 étapes :

1. Parser le fichier contenant les interactions
2. Traiter et stocker les interactions et les gènes associés
3. Parser le fichier contenant les gènes et ses phénotype
4. Traiter les données et annoter les gènes existants
5. Parser le fichier contenant les données du patient
6. Traiter et stocker les données du patient
7. Ecrire ces données dans un fichier sous forme de graphe

La classe interactome contient les informations récupérées depuis les différents fichiers (interactions, gènes et patients). Il permet la consultation et la modifications de ces informations, mais aussi des fonctions de test de cohérence des données (integrity check). La classe interactome contient donc trois collections : une de gènes, une d'interactions et une de patients. Les deux collections remplies à cette étape sont les gènes et les interactions. Les gènes seront ensuite mis à jour lors du parsage des fichiers suivants.

La classe Gene contient le nom du gène (utilisé comme identifiant unique), une liste de phénotype et une liste de patients (ces deux dernières seront mises à jour plus tard). La classe Interaction ne contient que deux gènes. Deux interactions contenant les même gènes sont considérées comme égales (relation non-orientée).

La récupération des données depuis le fichier contenant les interactions se fait de la manière suivante :

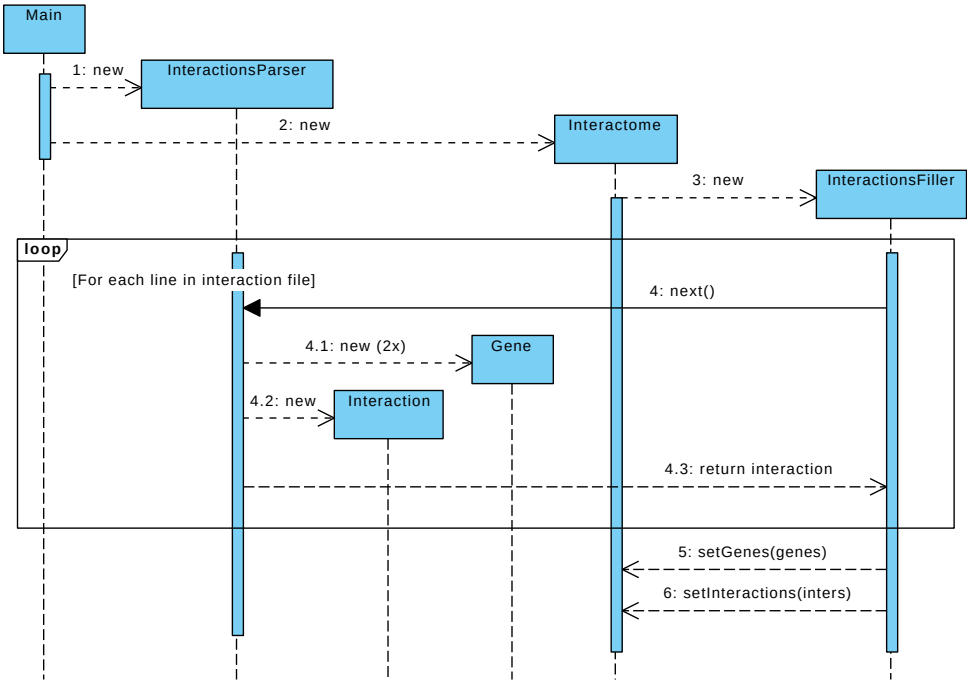


FIGURE 4 – Diagramme de séquence représentant la création de base de l'interactome

3.1.1 Parseur

La création du bon parseur se fait dans le main. Il est de la responsabilité de l'utilisateur d'utiliser le bon parseur ou de choisir d'en implémenter un nouveau afin de correspondre au format donné en entrée.

Le parseur n'est pas trop compliqué, car nos fichiers d'entrée sont au format .tsv[1], c'est à dire que chaque ligne contient une nouvelle interaction et que les identifiants des gènes sont séparés par des tabulations.

Le parseur est appelé par un "remplisseur" (InteractionsFiller) qui se charge de traiter les données et des mettre à jour l'interactome .

Algorithm 1: Filling the interactome data sets

Data: Interactome object and appropriate parser
Result: Interactome data sets (interactions and genes) filled with information from parsed file
new empty interaction set
new empty gene set
while *parser not at the end of the file* **do**
  get interaction Object from parser **if** *the two genes are different* **then**
    add interaction to interaction set
    create two new Gene Objects
    add genes to gene set
  end
  update interactome sets with local sets
end

3.2 Annotations des gènes

La récupération des phénotypes des gènes se fait de manière sensiblement pareille à la récupération des interactions. Pour chaque phénotype parsé, on met à jour la liste des phénotypes du gène concerné dans la liste des gènes. La liste des phénotypes est simplement une liste de chaînes de caractères.

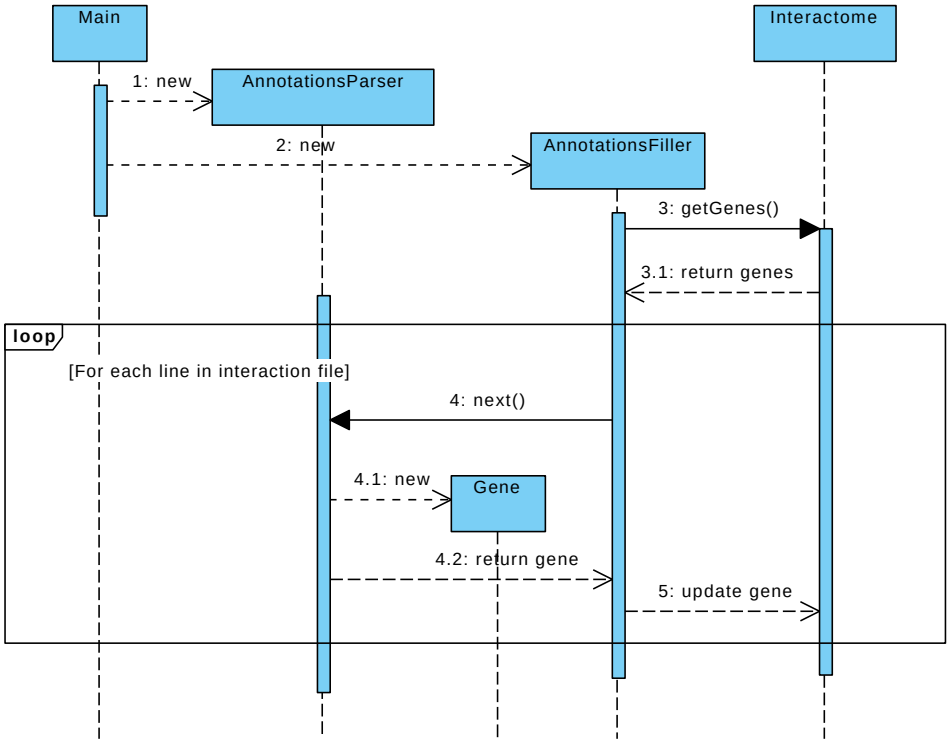


FIGURE 5 – Diagramme de séquence représentant l'ajout des phénotypes sur les gènes

3.3 Annotations du patient

La classe Genotype contient juste un nom et une gravité. Elle est utilisée sous forme de liste dans la classe Variant, qui est elle caractérisée par un gène et une liste de génotype.

Chaque fichier parsé représente un patient. Un patient est identifié par un nom et possède une liste de variants, qui est remplie lors dudit parsage. Il est donc facile, si on a le patient, de retrouver les gènes variants. Afin de faciliter l'opération inverse (retrouver les patients ayant une variation sur un gène en ne possédant que le gène), une liste de patient est aussi stockée dans chaque gène et mise à jour à chaque variant du patient.

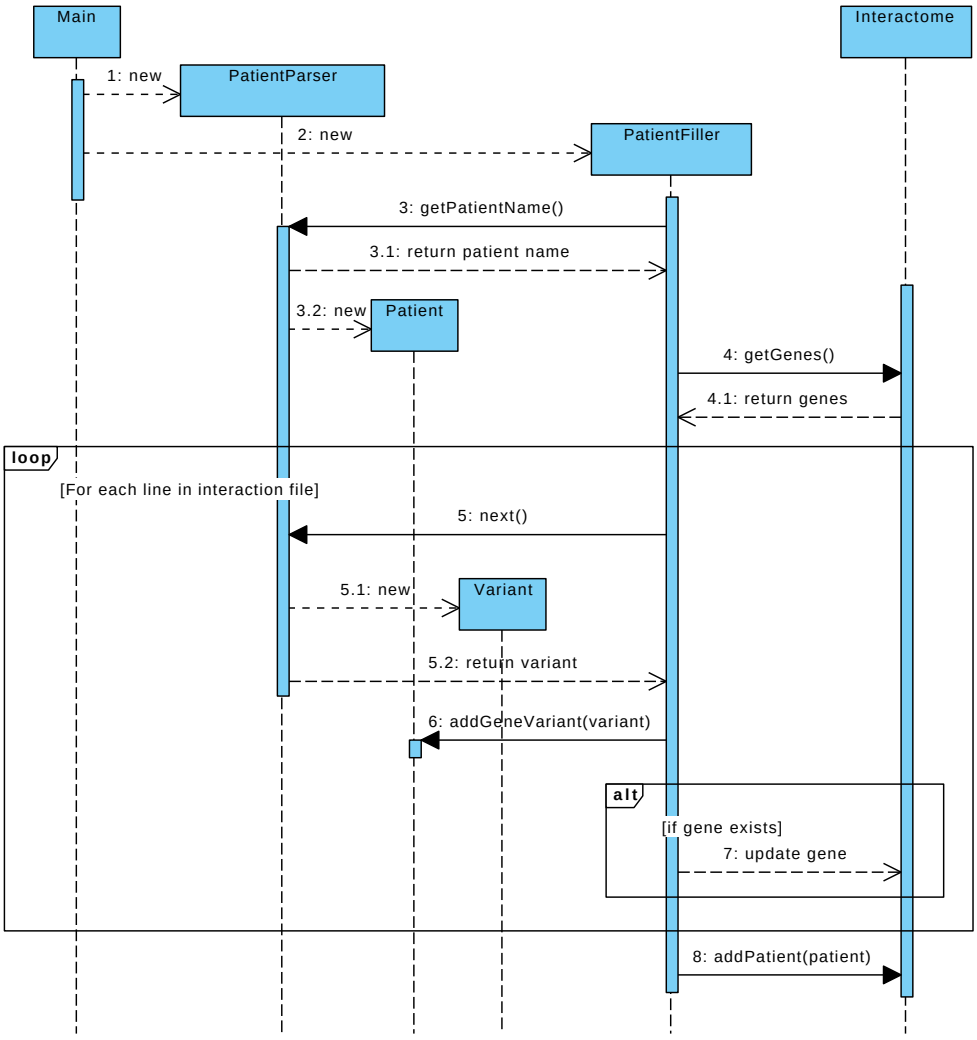


FIGURE 6 – Diagramme de séquence représentant l'ajout des informations du patient

3.4 Génération du graphe de sortie

La génération du graphe est très simple. Une fois le format choisi, il suffit d'écrire tous les éléments précédemment stockés

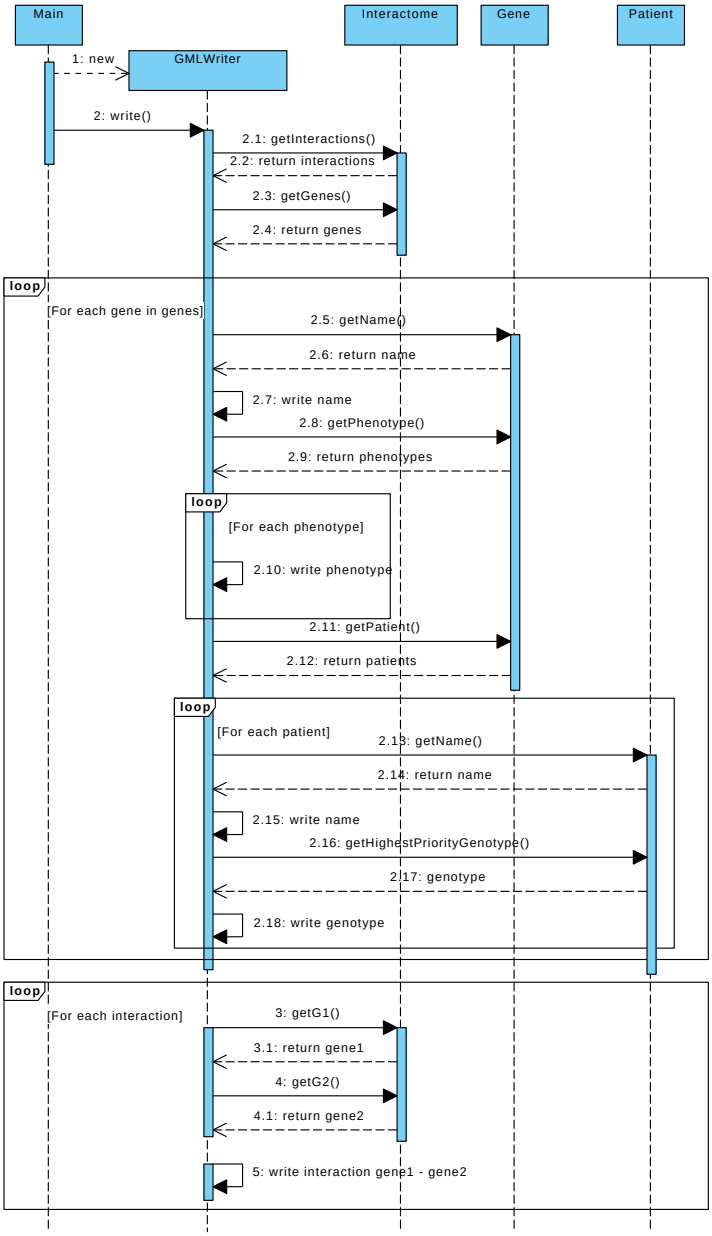
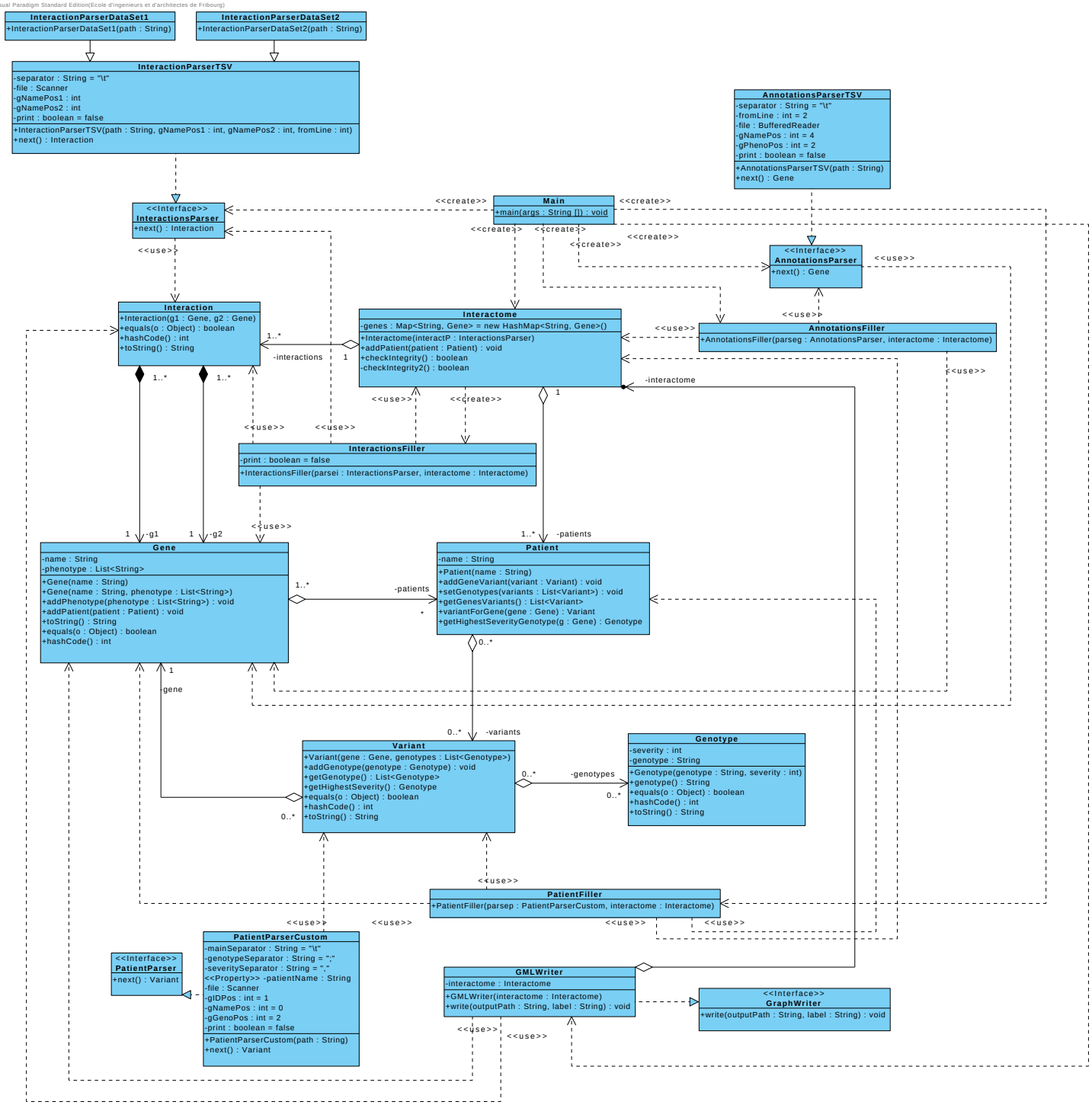


FIGURE 7 – Diagramme de séquence représentant la génération du graphe de sortie

3.5 Diagramme de classe

Voir page suivante :



3.6 Tests

Pour cette première partie, deux sortes de tests on été mis en place :
- Des tests unitaires
- Des tests d'intégrité des données

3.6.1 Test unitaires

Deux fichiers de tests on été crées : Un pour les parsers et un pour les fillers. Chaque fichier contient un test pour les interactions et un tests pour les annotations (phénotypes). Ce sont des tests très simple se basant sur des fichiers dont le contenu est connu. Il est donc facile de tester si le programme a le comportement attendu ou non.

3.6.2 Tests d'intégrité

Deux méthodes de test d'intégrité des données sont fournis dans la classe Interactome, permettant de s'assurer qu'une cohérence est maintenue entre les structures (interactions et gènes). En effet, un gène faisant partie d'un interaction doit impérativement apparaître dans la liste des gènes, sous peine de générer un fichier de graphe inconsistant.

3.7 Méthodes de recherche

3.7.1 Outil

Nous avons choisi de créer un plugin pour le logiciel Gephi permettant une recherche dans le graphe. Le logiciel est codé en Java et fournit une documentation pour les développeurs.

3.7.2 Plugin Gephi : fonctionnement général

La structure d'appel n'est pas bien documentée dans les ressources Gephi. Après des tests personnels, voici ce qui a été constaté.

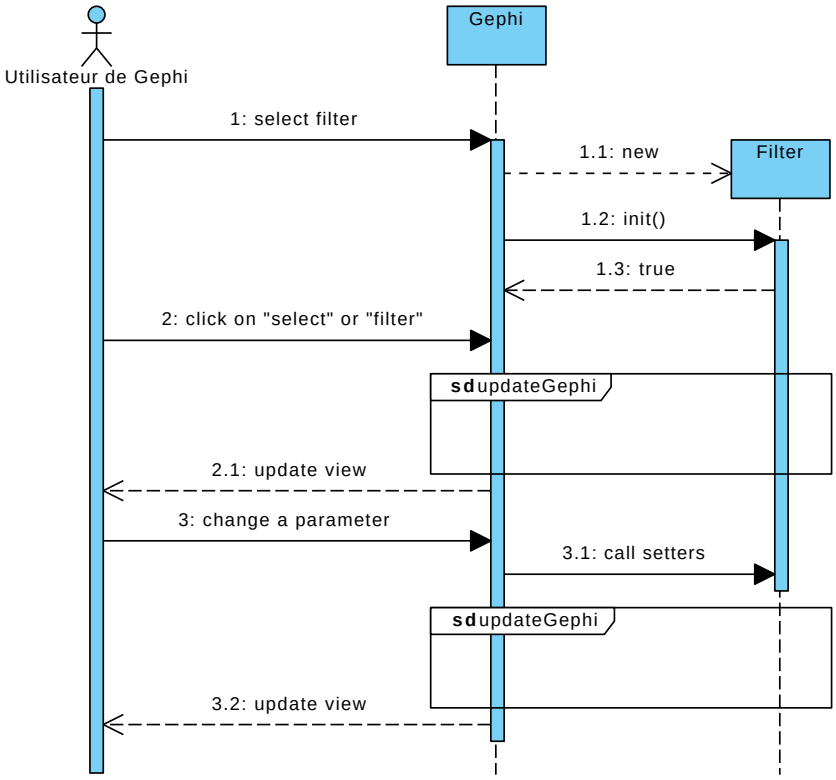


FIGURE 8 – Appel à un filtre dans Gephi

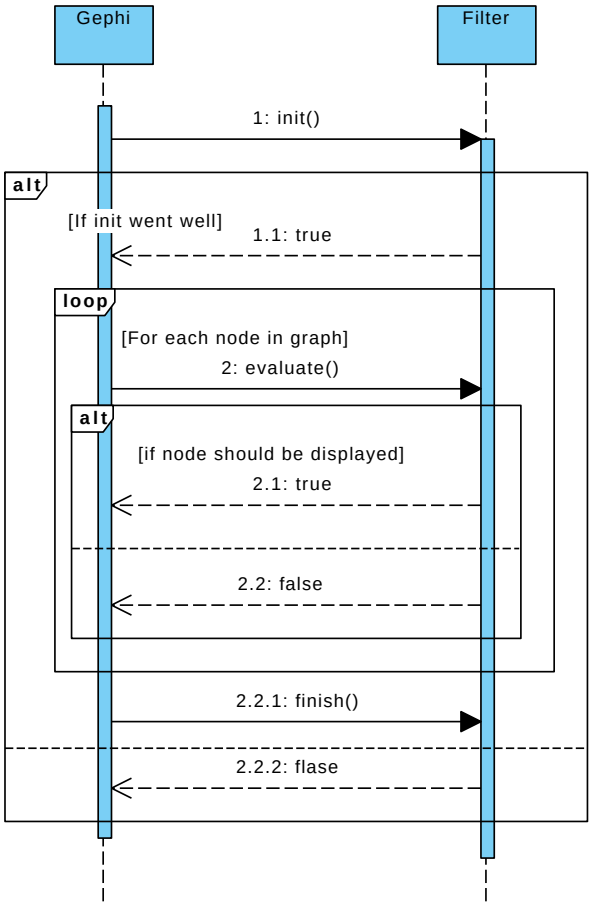


FIGURE 9 – Appel à un filtre dans Gephi : updateGephi

On voit que le init est appelé à chaque fois, mais que le evaluate est appelé à chaque changement sur chaque nœud . En fait, la seule manière d'afficher un nœud est de retourner "true" quand la méthode evaluate est appelée sur le nœud . Il est donc impossible de mettre de la vraie logique de traitement dans cette méthode (on ne peut pas espérer influer l'affichage ou non d'un autre nœud à partir de là). La logique se fait donc dans le init. Dans notre cas, nous avons décidé de remplir une liste contenant les nœuds devant être affichés.

3.7.3 Plugin Gephi : recherche concrète

Le filtrage des nœuds se fait en plusieurs étapes distinctes :

1. Sélectionner les nœuds des patients et les stocker
2. Sélectionner les nœuds ayant le phénotype et les stocker
3. Faire un plus court chemin entre les nœuds porteurs du phénotype et les autres
4. Stocker les chemins ayant une longueur plus petite ou égale à celle recherchée

3.7.4 Format et attributs attendus en entrée :

Le fichier de graphe fourni en entrée doit, pour chaque nœud , renseigner les attributs suivants :

nom	rôle	unique
id	arbitraire	oui
name	nom du gène	oui
phenotypeNb	nombre d'attributs de type phénotype ce nœud contient	non
phenotypeX	un phénotype	non
patientNb	nombre d'attributs patients qu'a ce nœud	non
patientX	nom du patient	non
genotypeX	génotype le plus grave du patient X	non

TABLE 7 – Attributs obligatoires d'un nœud

3.7.5 Affichage

Afin de mettre en évidence les nœuds qui nous interressent, la coloration suivante a été choisie pour les nœuds :

- Rouge : nœuds du patient à une distance d d'un noeud porteur du phénotype recherché
- Vert : nœuds porteur du phénotype à une distance d d'un noeud du patient
- Bleu : nœud porteur du phénotype et variant chez le patient (satisfaisant donc les deux conditions sus-mentionnées).

Initialisation des tableaux de distances
Pour l'algorithme du plus court chemin, l'algorithme de Bellman-Ford à été choisi car, dans le cas d'un développement futur, il permet une détection des cycles et autorise des nœuds à poids négatifs.

Algorithm 2: Algorithme de Bellman-Ford

```
Data: Graph g, Node source
Result: Shortest path from source to all other nodes
Map distance
Map predecessors
for all nodes do
    if node is source then
        distance(source) = 0
    else
        distance(node) = infinity
    end
    predecessor(node, null)
end
for 1 to nb of nodes do
    for all edges do
        if distance(edge.source) + 1 < distance(edge.target) then
            distance(edge.target) = distance(edge.source) + 1 ; predecessor(edge.target) =
            edge.source ;
        else
            distance(edge.source) = distance(edge.target) + 1 ; predecessor(edge.source) =
            edge.target ;
        end
    end
end
```

4 Implémentation

4.1 Outils utilisés

4.1.1 Génération du fichier du graphe

Cette partie a été entièrement réalisée en Java avec Eclipse.

4.1.2 Visualisation

Gephi est un logiciel riche qui fourni plusieurs types de plugins dont principalement les layouts et les filtres.

Layout : Pour la visualisation du graphe de l'interactome au complet, OpenOrd[9] est un algorithme fournissant une vue assez satisfaisante du graphe. C'est un algorithme fait spécialement pour les grands graphes, et se prête donc assez bien à notre cas, que ce soit du point de vue de la vitesse d'exécution que du rendu visuel.

4.1.3 Recherche dans le graphe

L'outil choisi pour la visualisation est Gephi. Le plugin de type filtre a donc été réalisé pour ce logiciel (voir mode d'emploi dans les annexes).

4.2 Problèmes rencontrés

4.2.1 Général

La taille des données à été une vraie difficulté pour ce projet. En effet, les tests ainsi que l'affichage prennent vite beaucoup de temps à s'exécuter, et bien que l'on puisse dans un premier temps travailler sur un graphe plus petit, il est indispensable de tester les données réelles.

4.2.2 Plugin filtre gephi

Bien que la création de plugin de type filtre soit documentée par Gephi, comprendre la logique des appels cachée derrière les classes n'est pas évident.

Il ne m'a pas non plus été possible de tester le filtre sur le graphe de l'interactome complet pour des raisons de mémoire. Il aurait fallu optimiser un peu le code, mais le temps à disposition ne l'a finalement pas permis.

4.3 Résultats

4.3.1 Visualisation de l'interactome avec le layout OpenOrd

La visualisation de l'interactome généré est possible et relativement rapide. Voici ce que cela donne avec un layout OpenOrd :

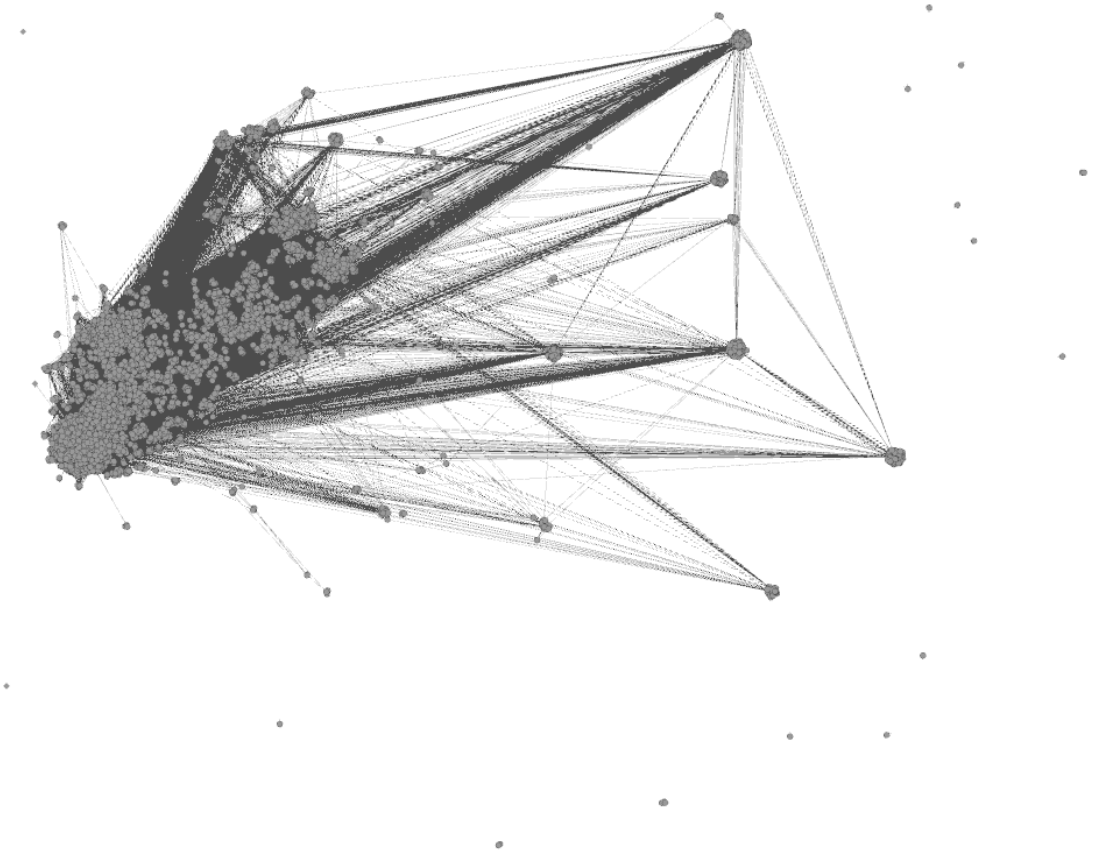


FIGURE 10 – Visualisation sous Gephi d'un graphe généré à partir du programme Java de ce projet : layout OpenOrd

4.3.2 Application sur un sous-graphe

Pour ce qui est de l'application du filtre, il n'a pas été possible de le faire sur le graphe entier, pour cause de problèmes de mémoire. Un sous-grapne a donc été généré à partir des données de l'interactome, mais ne contenant que 101 nœuds et 62 arètes. Après un layout de type Fruchterman Reingold (OpenOrd n'est pas adapté pour les petits graphes), voici ce que l'on peut obsever.

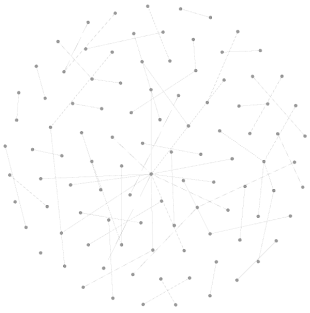


FIGURE 11 – Graphe avant l'application du filtre

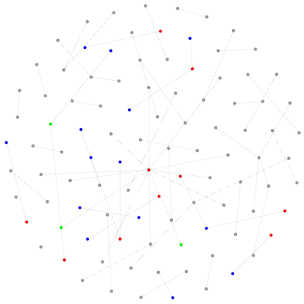


FIGURE 12 – Graphe après l'application du filtre (fonction filtre plus activée

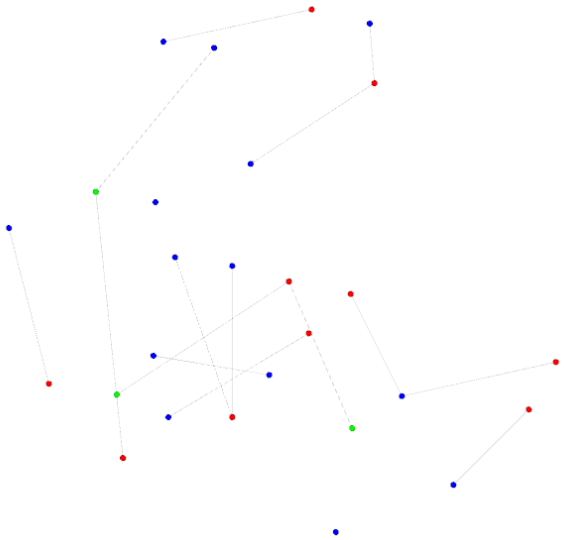


FIGURE 13 – Graphe après l'application du filtre (fonction filtre encore activée

5 Conclusion

5.1 Travail réalisé

Finalement, ce projet c'est déroulé en deux parties :

- Une première partie consistant en un programme Java permettant d'agréger les différentes informations sur les gènes en un fichier de graphe fourni en sortie.
- Une deuxième partie consistant en un plugin de filtre pour Gephi permettant une recherche des nœuds satisfaisant certaines contraintes

La première partie a été totalement réalisée, bien que quelques améliorations soient souhaitables.

La deuxième partie elle est largement perfectible. Beaucoup de temps a été perdu à la compréhension de l'outil ainsi qu'à la pleine compréhension de la recherche désirée.

Nous pouvons cependant dire que les objectif de départ ont été atteint. Le seul objectif n'ayant pas eu besoin d'être implémenté est celui de la génération d'un fichier de sortie contenant le résultat de la recherche. En effet, l'utilisation d'un outil externe (Gephi) rend cela automatique.

5.2 Perspectives

5.2.1 Améliorations

Le plugin Gephi est très perfectible mais son problème principal est le fait d'utiliser beaucoup trop de ressources. Le choix d'un meilleur algorithme de calcul du plus court chemin ou le stockage de moins d'éléments pourrait peut être améliorer la situation. Ce problème reste ouvert et mériterait d'être examiné.

5.2.2 Développement futur

Il serait intéressant de réussir à mettre les deux parties de ce projet dans un seul outil : soit de rapatrier la partie recherche dans un programme java, soit intégrer la partie génération du graphe dans un plugin Gephi. Les deux solutions semblent possible, bien que la première implique l'utilisation d'un moteur de visualisation de graphe dans le programme Java, rendant peut être le

travail plus difficile. Cependant, il faut aussi prendre en compte qu'il avait été discuté au début du projet de l'éventuelle intégration future de celui-ci dans un logiciel Java tiers. Dans cette perspective, il serait donc plus opportun d'opter pour la première solution.

L'utilisation d'une base de donnée pour le stockage des informations parsées pourrait aussi être une bonne idée.

5.3 Déclaration sur l'honneur

Je, soussigné, Maria Sisto, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d'informations utilisées et les citations d'auteur ont été clairement mentionnées.

5 Bibliographie

[1] Work with tab-delimited (tsv) format : https://support.google.com/docs/answer/63377?rd=1

[2] Supported formats for Gephi : http://gephi.github.io/users/supported-graph-formats/

[3] Supported formats for Cytoscape http://wiki.cytoscape.org/Cytoscape_User_Manual/Network_Formats

[4] GML documentation : http://www.fim.uni-passau.de/index.php?id=17297&L=1

[5] The GraphML File Format documentation : http://graphml.graphdrawing.org/

[6] GexRef documentation : http://gexf.net/format/

[7] XGMML documentation : http://cgi5.cs.rpi.edu/research/groups/pb/punin/public_html/XGMML/

[8] SBMLRef documentation : http://sbml.org/Documents

[9] OpenOrd plugin for Gephi : https ://marketplace.gephi.org/plugin/openord-layout/

6 Annexes

6.1 Glossaire

— Gène variant : Gène présentant une variation par rapport au modèle établi.
— Phénotype : Symptôme présent chez un patient
— Génotype : Type de variation sur un gène
— Interactome : Contient les interactions entre les gènes (commun à tous les humains).
— Interactome annoté : Contient l'interactome avec des annotations concernant les gènes (génotypes)
— Interactome augmenté : Contient l'interactome annoté augmenté des données du patient.

6.2 Mode d'emploi : Comment créer un plugin Gephi de type filtre

6.2.1 Mise en place de l'evironnement de développement

http ://gephi.github.io/developers/intro/

1. Installer git
2. Récupérer les source sur Github : https ://github.com/gephi/gephi
3. Télécharger et installer Netbeans IDE : https ://netbeans.org/
4. Ouvrir le projet git précédamment récupéré dans Netbeans

6.2.2 Création d'un plugin

https ://wiki.gephi.org/index.php/Plugin_Quick_Start_%285_minutes%29

Dans Netbeans :

1. Dérouler le projet Gephi Plugins
2. Click droit sur Modules > Add New...
3. Entrer un nom ainsi qu'un codebase et clicker sur Finish

6.2.3 Création d'un filtre

https ://wiki.gephi.org/index.php/HowTo_write_a_filter

1. Créer une classe implémentant FilterBuilder et ayant l'annotation suivante : @ServiceProvider(service = FilterBuilder.class)
2. Implémenter les méthodes de la classe parente, notamment getFilter() qui devra renvoyer notre filtre personnalisé et getPanel() qui devra renvoyer l'interface du filtre
3. Créer une classe implémentant Nodefilter
4. Créer une classe étendant JPanel représentant l'interface du fitre

6.2.4 Ajout d'un paramètre :

1. Dans la classe filtre, ajouter un attribut correspondant au nouveu paramètre
2. Ajouter les getters et setters dans la classe fitre
3. Dans la méthode getProperties(), ajouter un Filter.createProperty(...) au FilterProperty renvoyé
4. Dans l'interface, ajouter un champ de changement du paramètre avec un listener

6.3 Installation

6.3.1 Installation du générateur de ficher de graphe

Il n'y a pas besoin d'Installation.

6.3.2 Installation du plugin Gephi

1. Télécharger Gephi : http ://gephi.github.io/users/download/
2. Démarrer Gephi
3. Avant de charger le graphe, choisir Tools > Plugins dans le menu
4. Sélectionner l'onglet Downloaded
5. Clicker sur Add Plugin et sélectionner le plugin eiafr-visudna-patient.nbm
6. Accepter les conditions et terminer l'Installation

6.4 Contenu du CD

— Code java pour la partie génération du fichier
— Code du plugin Gephi
— Format installable du plugin Gephi
— Cahier des charges
— Mode d'emploi (installation et utilisation)