PROJET D'APPLICATION N°13

INTEGRATION DES ANALYSES DE REDUCTION ET COLORATION DE GRAPHES DANS UN OUTIL DE VISUALISATION DE GRAPHES

Réalisateurs : Pierre Le Jeune et Jules Paris

Encadrants : Carito Guziolowski et Bertrand Miannay

Remerciements

Nous tenons à remercier Mme Carito Guziolowski et M. Bertrand Miannay pour nous avoir accompagné pendant toute la durée de ce projet. Nous tenons également à remercier l'ensemble du corps enseignants qui s'est toujours montré disponible et réactif pour nous conseiller.

TABLE DES MATIERES

1. PRESENTATION DU PROJET :

1.1 Contexte

Lors de ses travaux de doctorant, Bertrand Miannay a travaillé, entre autres, sur une méthode d'identification des sous-réseaux par colorations dites parfaites de graphes. Cette méthode permet, à partir d'un réseau modélisé sous forme de graphe, d'identifier les ensembles de nœuds dont les signes seront corrélés entre eux et les assemble en sous graphes qui sont appelés composants. Dans sa thèse, il démontre l'efficacité de cette méthode pour identifier différents profils de patients atteints du myélome multiple. Dans ce modèle, les nœuds représentent des protéines, des gènes, des complexes ou autres phénomènes biologiques et les arcs représentent les interactions qu'il peut y avoir entre eux (ie l'activation ou l'inhibition). Cet algorithme permettant d'identifier ces composants a été baptisé IGGY-poc.

Cet algorithme d'identification des colorations de graphes pouvant demander un temps de calcul important, il a également mis au point un algorithme de réduction de graphe permettant de diminuer considérablement le temps d'exécution nécessaire afin d'obtenir un résultat exploitable avec IGGY-poc. En effet, le temps d'exécution de l'algorithme d'identification des composants passe de 1h30 à 10 min une fois le graphe réduit sur l'exemple utilisé par M. Miannay. Ce dernier quant à lui est composé de plusieurs scripts permettant à la fois d'identifier les colorations mais aussi de pouvoir reconstruire le graphe à partir du graphe réduit et des colorations trouvées.

Lors de ses travaux, il a donc réalisé plusieurs scripts écrits en Python et ASP (un langage de programmation logique) afin de produire ses résultats. Cependant, ses travaux étant destinés à des personnes ne sachant pas forcément utiliser Linux et encore moins les lignes de commandes, cet enchaînement de tâches n'était pas forcément très aisé ou intuitif. Ainsi, il a souhaité rendre son travail plus accessible en créant une Interface Homme-Machine (IHM). Cette interface aura pour but de pouvoir exécuter chacun des scripts sur les graphes de l'utilisateur ainsi que d'afficher ces graphes et leurs colorations. Cette tâche nous a donc été confiée dans le cadre de ce projet d'application.

Notre objectif sera donc de réaliser cette IHM de manière à ce qu'elle soit facile d'utilisation ainsi qu'un guide permettant à l'utilisateur de réaliser les tâches qu'il désire. Ces tâches seront d'afficher un graphe, de réduire un graphe ou bien d'identifier les colorations dans le but de pouvoir les afficher.

1.2 Cytoscape, un outil de visualisation de graphes

Afin de visualiser les graphes, Mme Guziolowski nous a suggéré d'utiliser Cytoscape qui est un outil permettant de visualiser des graphes contenus dans des fichiers de divers formats. M. Miannay ayant travaillé dans des fichiers au format .sif lors de son doctorat, Cytoscape était adapté puisqu'il peut lire ces fichiers. De plus, il existe de nombreux modules communautaires permettant de faire communiquer un script avec cette application. L'un d'entre eux, appelé CyREST, contient une bibliothèque permettant de lier Python à Cytoscape – sobrement appelée py2cytoscape – tout à fait adaptée aux tâches à réaliser.

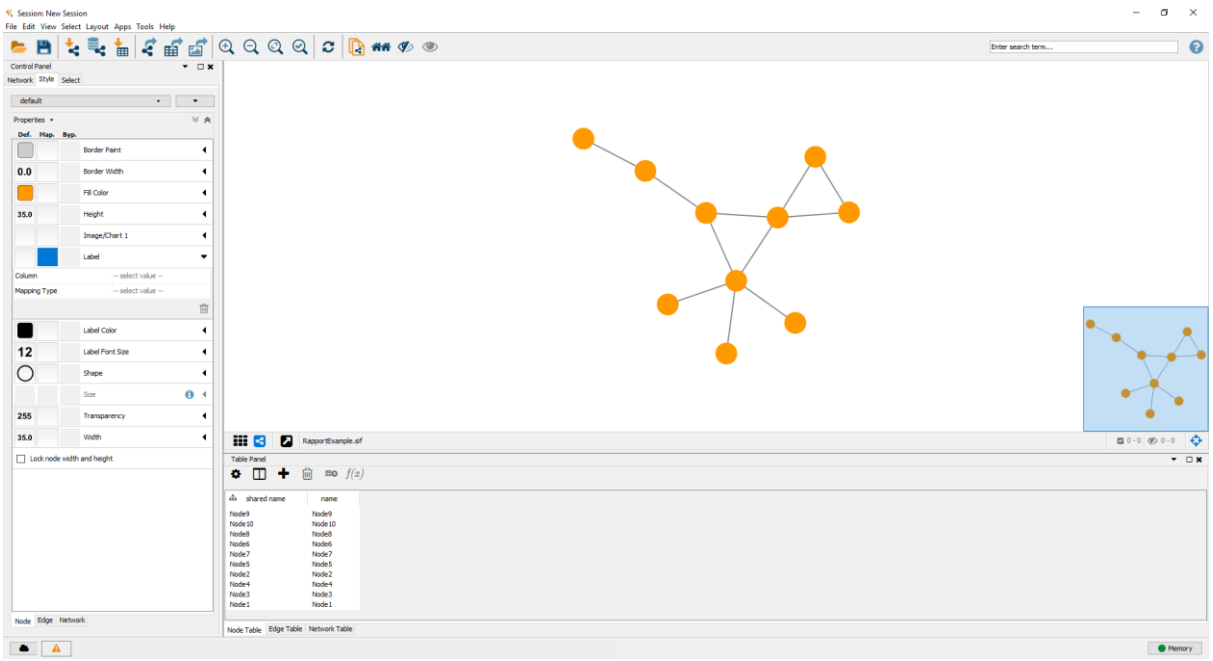

Figure 1 - Cytoscape avec un graphe affiché

L'un des objectifs du projet a donc été de maitriser cette bibliothèque afin de pouvoir retranscrire au mieux cet affichage de graphe.

1.3 Iggy-POC

Iggy-POC désigne l'ensemble des algorithmes développés par M Miannay afin d'effectuer cette identification de composants. Ces scripts s'articulent selon le schéma suivant :

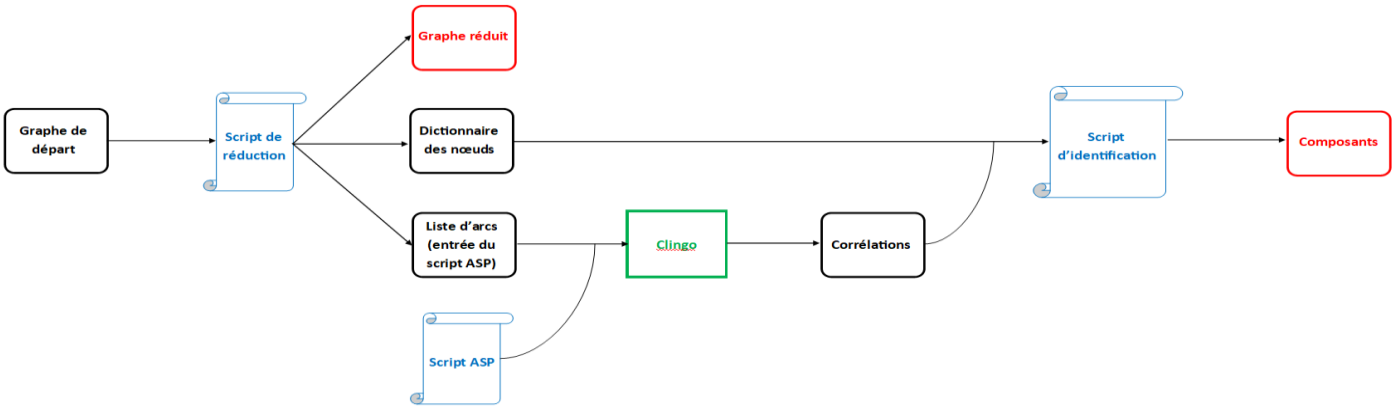

Figure 2 : Schéma récapitulatif du fonctionnement de Iggy-POC

1.3.1 Le script de réduction

La réduction est la première étape du processus. Elle permet de diminuer la taille du graphe sans perdre de l'information. Cette réduction s'effectue en trois passages correspondants aux trois règles utilisées pour la réduction (cf. Annexe 1). A chaque passage, on regarde dans le graphe où est observée la règle correspondante et on simplifie selon cette règle. A l'issue des trois passages, on obtient le graphe simplifié (ie avec moins de nœuds et moins d'arcs). Ce graphe réduit est ensuite traduit dans un deuxième fichier en liste d'arcs sous la forme : (nodeX,nodeY,i)

avec X et Y deux entiers numérotant les nœuds et i $\in \{+1,-1\}$.

Au travers d'un dictionnaire créé, on garde le nom des nœuds de départ et leurs numéros qui correspondent.

1.3.2 Le script ASP

L'ASP est un langage logique qui est très efficace pour résoudre des problèmes sur les graphes. Il est donc utilisé dans ce contexte pour accélérer grandement le calcul des colorations. En effet, ce calcul demande plusieurs parcours du graphe et est donc très lourd. Le but est de déterminer des composants (ensembles de nœuds) du graphe étant indépendants les uns des autres. C'est-à-dire que les modifications (i.e. activer ou inhiber un nœud) apportées dans un composant ne doivent pas se répercuter sur les autres composants. Ce script s'exécute sur la liste d'arcs et nous donne les composants sous la forme d'ensembles de nœuds du graphe réduit.

1.3.3 Le script d'identification

Finalement, il nous faut mettre en relation les colorations obtenues sur le graphe réduit et le graphe initial (celui dont on souhaite extraire les composants). Pour ce faire, on réutilise le dictionnaire créé lors de l'algorithme de réduction puis on traduit chacun des nœuds en n'oubliant pas de repartir du graphe de départ et non du graphe réduit.

2. PREMIERS PAS, PLANIFICATION ET CONCEPTION

Notre premier contact avec le sujet a été une réunion avec Mme Guziolowski et M. Miannay pendant laquelle nous avons eu une présentation du sujet et de son contexte. C'est également au cours de cette réunion que Mme Guziolowski nous a parlé de Cytoscape. L'objectif établi à la fin de la réunion était d'afficher des graphes dans Cytoscape ainsi que d'ajouter à cela une IHM permettant de faciliter l'utilisation pour des personnes s'y connaissant très peu en programmation. Notre premier travail a alors été de faire un cahier des charges comportant la planification ainsi que les attentes de nos encadrants (cf. Annexe 2) ainsi que des diagrammes UML pour définir les fonctionnalités de l'application (cf. Annexe 3).

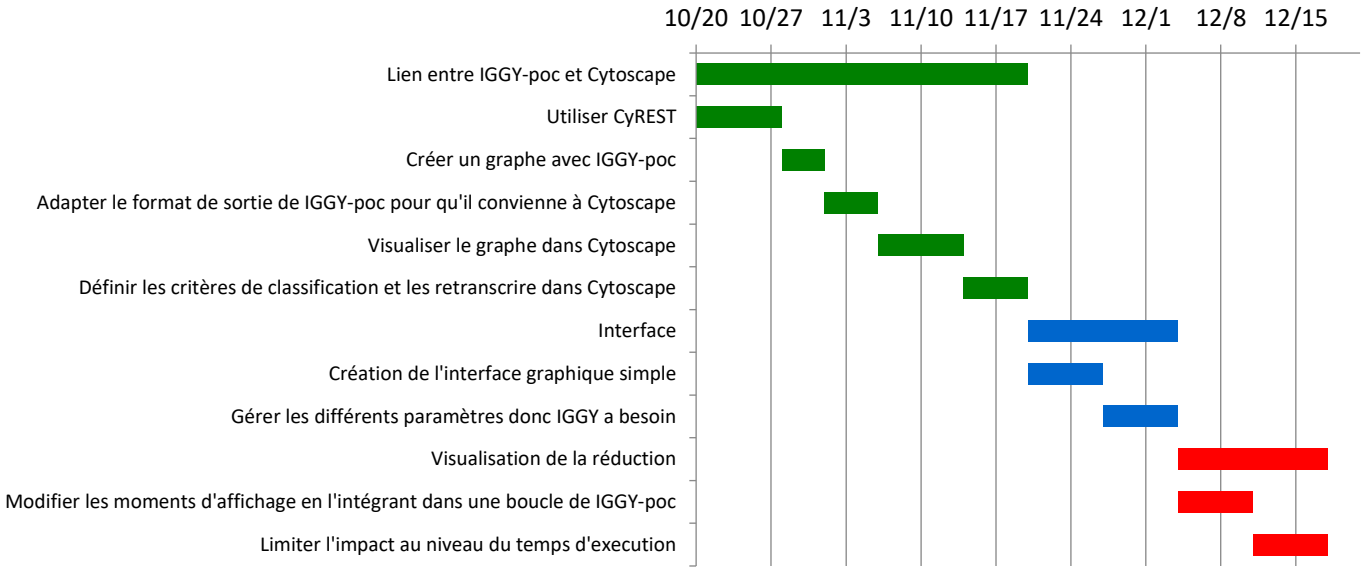

Figure 3 - Planification

Du fait que nous ne pouvions estimer que difficilement le temps que nous prendrait la liaison entre les scripts de M. Miannay et Cytoscape, notre planification a laissé beaucoup de temps pour cette tâche. Pour le reste, étant donné que nous étions tous les deux à l'aise en python et que nous connaissions bien le langage, les durées ont été estimées plus précisément selon ce que nous nous faisions comme idée.

3. ESSAIS ET PREMIERE IHM

3.1 Installation de l'environnement

Une fois la conception terminée, nous avons pu commencer la réalisation de l'application. La première chose à faire était de choisir un environnement Python. Nous avons choisi de travailler avec Python 3.6 et avec l'IDE Spyder. Ensuite nous avons installé les bibliothèques nécessaires pour relier Python et Cytoscape. Pour cela nous avons utilisé le gestionnaire de paquets Python "pip" qui permet d'installer automatiquement des bibliothèques. Nous avons choisi d'installer "py2cytoscape" pour utiliser Cytoscape depuis Python. Cette bibliothèque fait partie d'un ensemble d'outils appelé CyREST qui permet d'automatiser les fonctionnalités de Cytoscape via différents langages de programmation. Cependant l'installation s'est révélée plus complexe que prévue. En effet, pour fonctionner, py2cytoscape a besoin de plusieurs autres bibliothèques Python (cf. Annexe 4)

C'est cette étape qui a posé de nombreuses difficultés. En effet, toutes les bibliothèques ne s'installent pas aussi facilement, notamment parce qu'il faut trouver les versions qui peuvent travailler les unes avec les autres alors que pip installe en priorité les versions les plus récentes. Pour remédier à cela, nous avons donc installé les bibliothèques manuellement et en faisant attention aux versions.

Ensuite il a fallu installer Cytoscape (version 3.5.1). Cela fut plus simple puisqu'il suffit de télécharger un exécutable et de le lancer. La seule chose à laquelle il faut faire attention est que pour fonctionner, Cytoscape a besoin d'une version de Java. En revanche, nous avons dû utiliser la version 8 de Java, afin d'utiliser Cytoscape. La version 9 n'étant pas compatible. Ce problème fut assez vite résolu puisque ces recommandations sont écrites juste en dessous du lien de téléchargement sur leur site. Entre temps une nouvelle version de Cytoscape est sortie (3.6.0) mais celle-ci non plus n'est pas compatible avec Java 9.

3.2 Premiers essais

Après l'installation des logiciels nécessaires, nous avons pu commencer à utiliser Cytoscape via Python. Nous avons d'abord parcouru la documentation (plutôt succincte) en ligne, sur py2cytoscape et nous avons appliqué tout cela pour apprendre à créer, afficher et modifier des graphes. Ensuite nous avons testé l'affichage des graphes à partir des fichiers .sif que M. Miannay nous a fourni. Nous avons aussi fait attention au style du graphe, notamment pour prendre en compte les deux types d'arcs qui cohabitent dans ces graphes. Des arc "+1", activateurs et des arc "-1", inhibiteurs qui modélisent les deux types d'interactions possibles entre les protéines (cf. partie 1). Pour faire cela, il a fallu modifier les feuilles de styles de Cytoscape. Cette étape a été plus délicate. En effet, la documentation fournie ne donnant que des exemples incomplets, il a fallu procéder par tâtonnement jusqu'à trouver le nom et les arguments de la fonction pour modifier ces feuilles de style.

3.3 IHM et premières fonctionnalités

Une fois py2cytoscape maitrisé nous avons attaqué la réalisation d'une interface graphique qui permettrait de pouvoir charger et visualiser les graphes. Pour cela, nous avons choisi la bibliothèque graphique PyQt, notamment parce qu'elle était assez simple à prendre en main. De plus, l'IHM que nous voulions faire n'avait pas besoin de fonctionnalités très élaborées et pouvait être réalisée avec n'importe quelle bibliothèque graphique.

3.3.1 L'IHM

Comme nous l'avons expliqué plus tôt, l'IHM était très simple à la base, seulement quelques boutons pour les différentes fonctionnalités. L'interface graphique présentée sur la figure 4 n'est pas la version originale de l'IHM, mais une version un peu plus avancée, présentant la fonction de recherche des colorations. Nous n'avons pas conservé la toute première version de l'IHM qui ne présentait pas cette fonctionnalité, mais le design était identique.

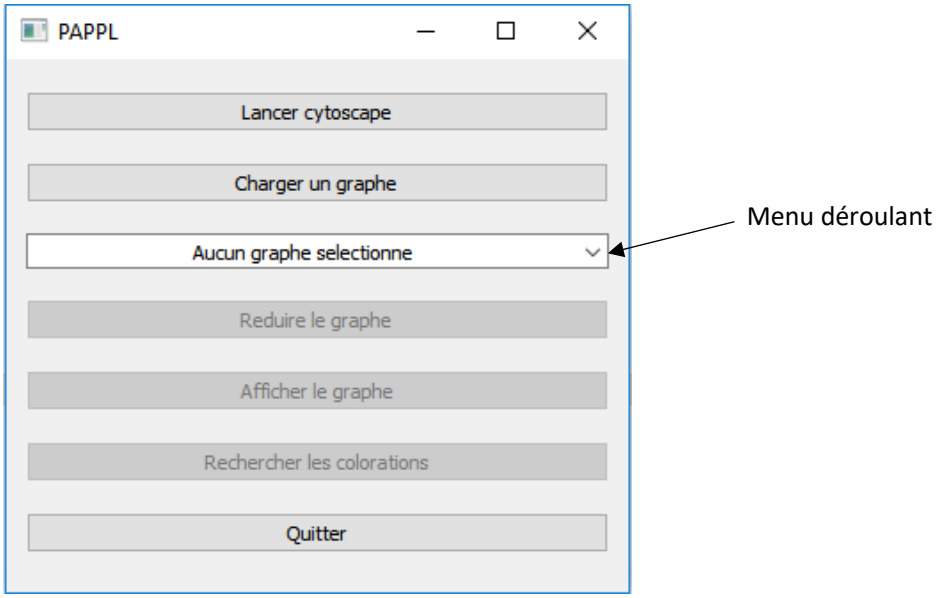

Figure 4 - Première IHM

Voici les différentes fonctionnalités de l'IHM présentée ci-dessus :

- Le premier bouton permet de lancer Cytoscape car il faut qu'une session Cytoscape soit ouverte pour que l'on puisse afficher des graphes.
- Un bouton pour charger les graphes qui ouvre un explorateur de fichiers. Une fois sélectionnés, les graphes se retrouvent dans un menu déroulant (cf. Figure 4).
- Ensuite un bouton par fonctionnalité, afficher, réduire, rechercher les colorations (à noter que cette dernière fonctionnalité n'était pas présente dans la version initiale).

Pour l'affichage du graphe, il suffit simplement de lancer la bonne fonction de py2cytoscape en récupérant le bon nom de fichier (correspondant bien au graphe que l'on souhaite afficher) et son emplacement.

3.3.2 Réduction

La réduction a été plus difficile à mettre en place. En effet, la réduction se base sur l'algorithme de réduction de graphe de M. Miannay, IGGY-poc, qui prend un fichier .sif en entrée et qui renvoie le graphe réduit ainsi que deux autres fichiers qui seront nécessaires plus tard : une table de hachage qui associe l'ensemble des nœuds du graphe initial au nœud correspondant dans le graphe réduit et l'entrée du programme logique. L'intégration du script de M. Miannay a nécessité quelques ajustements. En effet, il l'utilisait en l'exécutant depuis une console alors que nous voulions en faire une fonction d'un programme de sorte à pouvoir l'appeler à chaque pression du bouton

correspondant. Cela changeait notamment la manière dont le script récupérait les fichiers d'entrées ce qui a engendré quelques modifications.

Voici un exemple de réduction réalisée à partir de l'IHM et d'un graphe d'exemple.

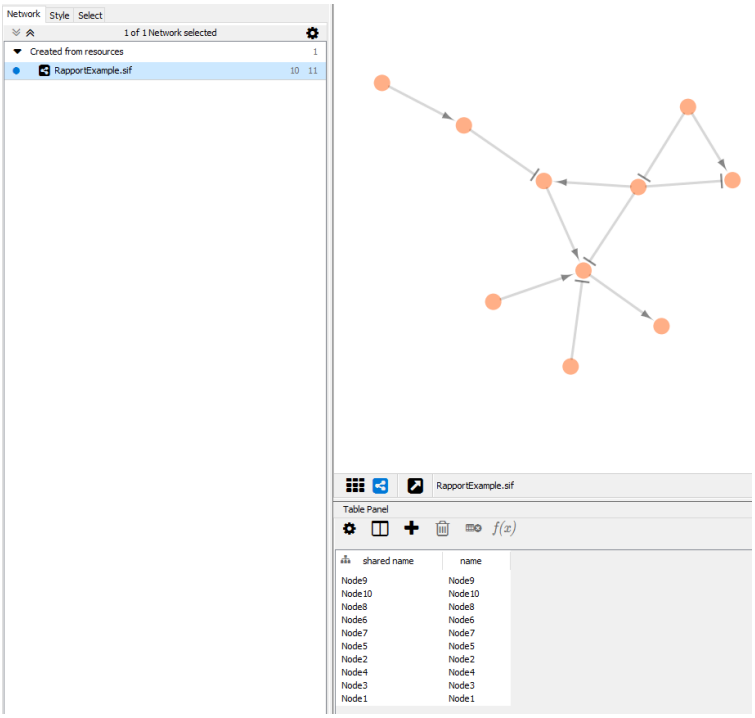

Figure 5 - Graphe d'exemple (GE)

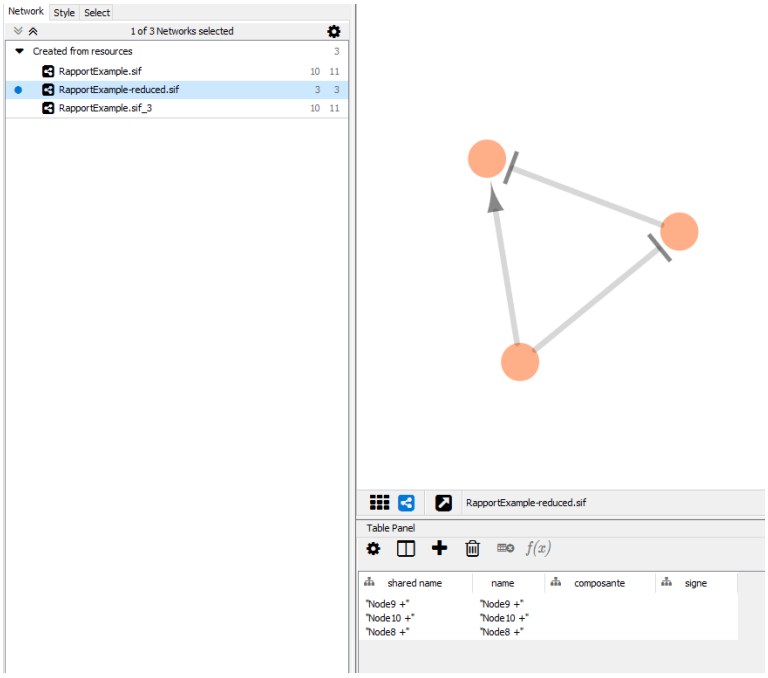

Figure 6 - GE réduit

4. REFONTE DE L'IHM ET DEVELOPPEMENT DE NOUVELLES FONCTIONNALITES

4.1 Présentation de la nouvelle IHM

Lors des réunions avec nos encadrants, notre première version de l'IHM ne leur a pas paru très intuitive. En effet, toutes les fonctionnalités étant présentes sur une seule petite fenêtre et la démarche à suivre pour réaliser une tâche n'était pas forcément claire pour eux. De plus, cet aspect très minimaliste n'était pas très séduisant pour l'utilisateur. Nous avons donc eu comme objectif de créer une nouvelle IHM qui soit à la fois plus simple d'utilisation mais aussi plus agréable à utiliser.

Ainsi, nous avons pensé à un système d'onglets permettant de guider l'utilisateur pour chacune des tâches qu'il peut effectuer. Cela permet de compartimenter les actions de l'utilisateur et de diminuer le nombre de boutons présents à l'écran rendant l'interface plus épurée.

Pour faciliter notre développement et pour rendre les futures modifications plus aisées, nous avons décidé de travailler sur QtDesigner, un logiciel permettant d'éditer une interface graphique facilement en plaçant boutons et autres objets de manière manuelle, sans avoir à écrire directement de code.

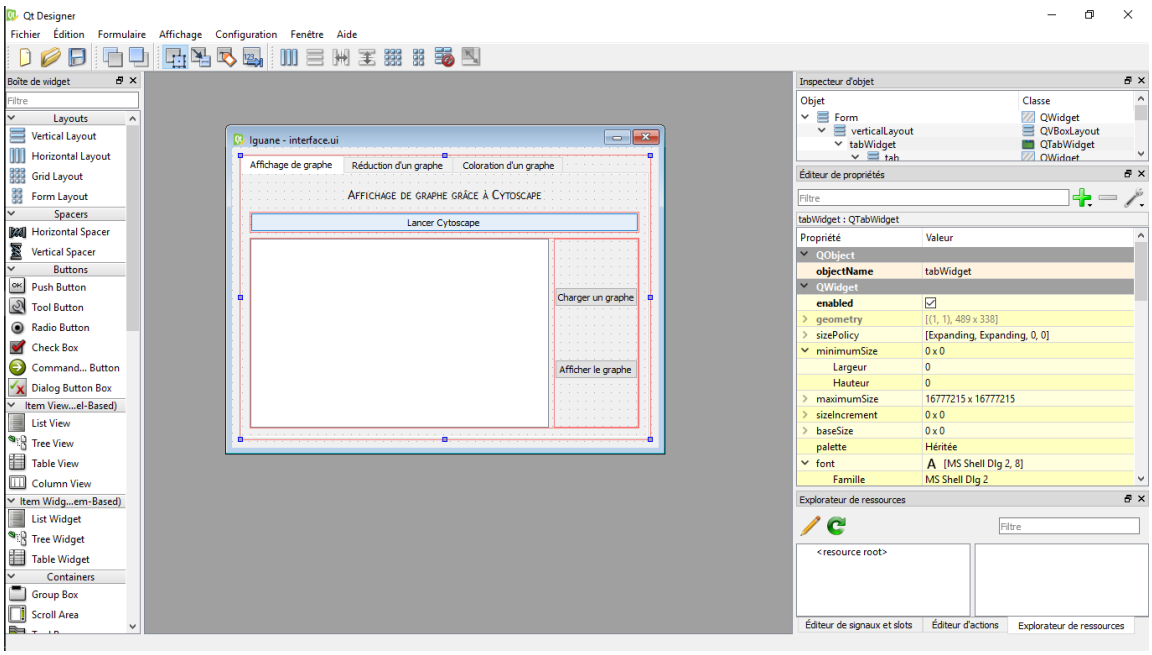

Figure 7 - QtDesigner

Il faut ensuite exécuter des lignes de commandes afin de pouvoir créer un fichier python à partir de ce fichier .ui grâce à la bibliothèque pyuic5 – puisque nous utilisons pyQt5.

Voici les deux commandes que nous utilisons pour générer le .py (remplacer « Session » par le nom de nos sessions Windows) :

- `cd C:\Users\"Session"\Desktop\Interface_Build_Iguane`
- `D:\WinPython-64bit-3.5.3.1Qt5\python-3.5.3.amd64\Scripts\pyuic5 -x interface.ui -o interface_ui.py`

Le fichier généré – interface_ui.py – peut alors être importé dans notre script principal afin d'être utilisé. Ensuite, à chaque fois que nous désirons modifier l'interface il nous suffit de régénérer le ficher .py puis de remplacer l'ancien par le nouveau. Tant que les noms d'objet des boutons restent les mêmes, ceux-ci seront toujours utilisables dans notre script principal et donc aucun changement ne sera à prévoir.

Notre nouvelle interface ainsi créée se décompose en trois onglets (cf. Annexe 5) :

- Le premier permettant d'ouvrir Cytoscape et d'afficher un graphe
- Le deuxième permettant d'effectuer la réduction
- Le troisième la recherche des colorations et l'affichage des colorations

Lors des semaines qui ont suivies, d'autres modifications ont été faites, notamment l'ajout d'un bouton pour l'export des N composants ainsi que la possibilité de choisir le temps alloué pour l'identification des colorations.

4.2 Identification des colorations du graphe

Pour l'identification des colorations, on a utilisé clingo qui est un solveur de scripts ASP. M. Miannay a également écrit un script permettant de trouver les composants d'un graphe, nous n'avions qu'à lancer clingo avec ce script et le troisième fichier produit lors de la réduction du graphe. Ce fichier est une liste d'arcs dans un format bien spécifique. La difficulté ici a été de faire fonctionner clingo avec les options adéquates, de manière à garantir que la solution trouvée est bien un optimum global et non local. Pour cela, M. Miannay nous a conseillé d'ajouter deux options (**--opt=optN** pour avoir l'optimum et **--enum-mode=cautious** pour avoir l'intersection de tous les ensembles de solutions).

De manière à améliorer la vitesse d'exécution, nous avons également ajouté l'option **--parallel-mode=2** afin d'utiliser plusieurs cœurs (deux en l'occurrence) du processeur pour pouvoir parcourir plus vite l'espace des solutions.

Ensuite il suffisait d'exécuter la commande via Python et d'écrire le résultat dans un fichier texte. Cependant le résultat est le résultat brut, sorti de la console, il faut donc retravailler ce fichier pour pouvoir récupérer uniquement les informations nécessaires à l'identification des composants.

4.3 Coloration du graphe

L'objectif de cette fonction est d'afficher le graphe de départ mais d'attribuer des couleurs différentes aux nœuds du graphe en fonction du composant auquel ils appartiennent. Pour cela, il faut d'abord créer un nouvel attribut dans Cytoscape, cela équivaut à ajouter une colonne à la table des nœuds de Cytoscape et d'attribuer le numéro du composant pour chaque nœud.

Une fois les colorations identifiées, on applique le deuxième script de M. Miannay, qui prend en entrée la table de hachage (créée lors de la réduction du graphe) ainsi que le résultat de l'identification des colorations (script ASP) afin de construire la liste des nœuds composants chaque coloration. Nous avons, à partir de cela, créé une table qui contient pour chaque nœud : son nom, le numéro de son composant et son signe (selon l'action activatrice ou inhibitrice qu'il a sur l'ensemble du composant). Le fichier ainsi créé est un fichier .csv, lisible avec n'importe quel tableur. On importe ensuite ce fichier dans Cytoscape, pour mettre à jour la table des nœuds.

Ensuite il suffit de modifier le style du graphe et d'attribuer des couleurs à chaque composant. Pour cela, Cytoscape possède une fonctionnalité (le continuous mapping), très efficace qui prend deux couleurs et va attribuer des couleurs régulièrement espacées en fonction d'un attribut (ici le numéro de la composant) de manière automatique.

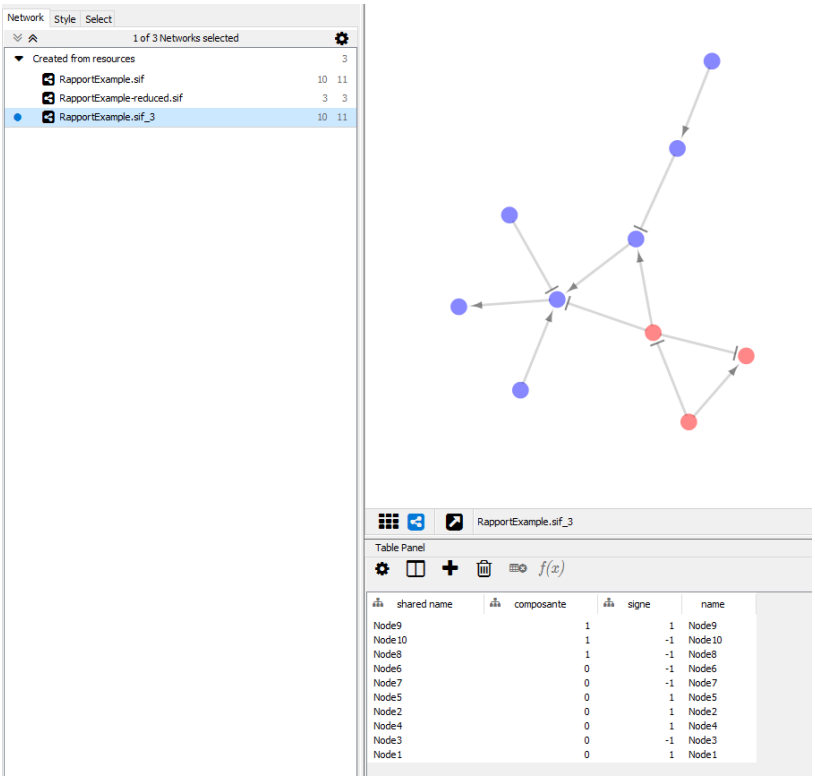

Figure 8 – GE coloré. Il y a deux couleurs, une pour chaque composant.
On colore les graphes avec des nuances entre le bleu et le rouge en fonction du nombre de composants.
Ici, il y en a deux donc les deux couleurs sont le bleu et le rouge.

4.4 Export des n composants

L'implémentation de cette fonctionnalité a demandé plus de temps que les précédentes. En effet, pour chaque composant, il faut sélectionner les arcs du graphe de départ dont les deux extrémités sont dans le composant. Cela nécessite le parcours du graphe autant de fois qu'il y a de composants et peut donc prendre un peu de temps. Ensuite il suffit de créer un fichier par composant. Il faut néanmoins faire attention aux composants qui ne comportent qu'un seul nœud. En effet, en premier lieu nous n'en avons pas tenu compte et les fichiers des composants ne comportant qu'un seul nœud étaient vides.

5. REALISATION D'UN EXECUTABLE WINDOWS

5.1 Réalisation de l'exécutable

Comme expliqué en introduction, ce logiciel a vocation à être le plus simple possible et n'est pas destiné à des personnes sachant déjà programmer. Ainsi, il nous a semblé indispensable de créer un exécutable ne nécessitant aucune installation de Python ni de bibliothèque. Tout cela afin de rendre l'utilisation très simple. Pour ce faire, nous nous sommes documentés sur internet et avons finalement choisi d'utiliser la bibliothèque cx_Freeze (sous sa version : 5.0.1). Son utilisation est simple :

- On crée un fichier setup.py qui sera utilisé pour compiler l'exécutable à placer dans le même dossier que le fichier principal
- Dans le fichier setup.py on écrit la base du code contenant notamment le fichier à compiler en .exe, les fichiers complémentaires et bibliothèques à inclure ou bien un icone pour le .exe.
- En dernier lieux on lance la commande « python setup.py build » à partir d'un terminal Windows pour lancer la compilation

Une fois ces étapes réalisées, on obtient un dossier « Build » contenant tous les fichiers nécessaires à l'exécutable et l'exécutable lui-même.

5.2 Rédaction d'un guide d'utilisation

Pour aller avec notre exécutable, Mme Guziolowski nous a demandé de rédiger un guide d'utilisation expliquant comme installer cette application et comment l'utiliser. Dans ce guide d'utilisation, nous avons expliqué chaque étape du processus au travers d'une brève explication, toujours accompagnée d'une capture d'écran afin de rendre cela le plus clair possible.

Nous avons également indiqué en particulier quelle version de Cytoscape installer ainsi que quelques conseils d'utilisation permettant de faciliter l'utilisation de cette application. Vous pouvez trouver l'ensemble des fichiers de l'application ainsi que le guide d'utilisation sur un dépôt git (cf. Annexe 6).

6. ORGANISATION DU PROJET

6.1 Organisation du travail

Sur ce projet nous avons travaillé avec une méthode agile. Pour cela, nous avons convenu de faire des réunions hebdomadaires avec les clients pour leur présenter notre état d'avancement. Chaque semaine nous avons travaillé pour présenter un prototype fonctionnel lors de la réunion, un peu à la manière de Scrum. Lors de ces réunions, nous définissions également les avancements à réaliser pendant la semaine suivante, nous parlions des difficultés rencontrées et comment nous y faisions face. De plus, étant dans la même option disciplinaire, nous pouvions discuter tous les jours de l'avancement des développements et ainsi ajuster la répartition des tâches au fur et à mesure de l'avancement. La bonne communication au sein du groupe et avec les clients nous a permis d'avancer plus vite que prévu puisque lorsqu'un de nous était indisponible, l'autre prenait le relai. Nous avons ainsi dépassé les objectifs premiers du projet, à savoir réaliser une interface pour la réduction de graphe. Les autres fonctionnalités n'étaient pas prévues pour être implémentées à la base, mais au vu de notre avancement, nous avons redéfinis les objectifs avec les clients.

En général nous séparions les tâches de manière à pouvoir avancer en parallèle, mais nous avons toujours conservé un regard sur ce que faisait l'autre de sorte à pouvoir s'entraider sans perdre trop de temps à rentrer dans le travail de l'autre le cas échéant.

Nous avons également utilisé Git pour partager notre code, mais uniquement en tant que dépôt, car nous n'avions pas encore suivi les cours de MEDEV où nous avons appris à utiliser les principales fonctionnalités de Git.

6.2 Difficultés rencontrées

Lorsque nous avons implémenté l'identification des composants, nous pensions que tout fonctionnait correctement, nous n'avions pas d'erreur d'exécution et on obtenait des colorations qui semblaient cohérentes. Cependant lors d'une de nos réunions hebdomadaires, M. Miannay a remarqué que le nombre de composants que nous obtenions ne correspondait pas avec le sien. Nous avons donc cherché ce qui pouvait être à l'origine de cette différence. Nous avons tout d'abord trouvé un problème assez vicieux. En effet, python est langage de programmation qui l'indentation pour séparer les différents blocs d'un programme. Or, il se trouve que lorsque l'on a récupéré le script de réduction de M. Miannay, les indentations n'ont pas toutes été rétablies à l'identique par rapport au

script original. Cela entrainait un problème dans la réduction du graphe. En l'occurrence, certaines instructions n'étaient plus dans leurs boucles et ne s'exécutaient pas le bon nombre de fois.

```
a=0

for i in range(1,10):
    print(a)
    a=i

>>> (executing lines 1 to 5 of "<tmp 1>")
0
1
2
3
4
5
6
7
8
```

Figure 9 : Exemple d'une boucle python

```
a=0

for i in range(1,10):
    print(a)
a=i

>>> (executing lines 1 to 5 of "<tmp 1>")
0
0
0
0
0
0
0
0
0
```

Figure 10 : Exemple d'une boucle Python

Dans cet exemple on voit que selon l'indentation de la dernière ligne, celle-ci est ou n'est pas exécutée à l'intérieur de la boucle. Ainsi le programme ne donne pas le même résultat.

Nous ne comprenons toujours pas pourquoi ce problème est survenu, en effet lorsque nous faisons la même opération sous Linux, nous n'avons aucun problème, les indentations sont correctes. En revanche tous les essais que nous avons réalisé sous Windows présentaient les mêmes erreurs d'indentation.

Plus étrange encore, une fois les problèmes d'indentation résolus, nous n'obtenons pas les mêmes résultats en exécutant le script (le même fichier) sous Linux et sous Windows. Sous Linux nous obtenons les mêmes résultats que M. Miannay alors que sur Windows nous obtenons des résultats proches mais pas identiques. Le graphe réduit a le même nombre de nœuds et d'arcs mais les nœuds du graphe réduits ne correspondent pas toujours. En effet lors de la réduction du graphe, l'algorithme sélectionne des nœuds qui dépendent les uns des autres et les regroupe en un seul nœud du graphe réduit (cf. 1.3.1). Ces regroupements de nœuds ne sont pas tous les mêmes selon que l'on exécute le script sous Linux ou sous Windows. De plus, même sous Windows, selon la version de Python que nous utilisons, nous n'obtenons pas les mêmes résultats.

Nous pensons qu'il s'agit d'une différence dans le module Networkx (cf. Annexe 4) puisque les versions de cette bibliothèque varient selon le système d'exploitation et la version de Python utilisée. Networkx est la seule bibliothèque nécessaire pour l'exécution du script de réduction et c'est elles qui permet de construire et de manipuler les graphes. Nous pensons donc que certaines fonctions utilisées lors que la réduction, ont des comportements différents selon l'environnement logiciel utilisé. Cette erreur nous est apparue assez tardivement et nous n'avons pas eu le temps d'investiguer assez pour trouver une solution. M. Miannay recherche lui aussi quelle pourrait être la cause de ces différences pour que nous puissions corriger ce problème et rendre l'application totalement opérationnelle.

Cette erreur est apparue à la fin du projet alors que le problème provient d'une des premières fonctionnalités que nous avons implémentées. Cela s'explique notamment par le fait que les fichiers qui sont produits lors de la réduction du graphe sont particulièrement illisibles (cf. Annexe 7). Il était donc nécessaire que les autres fonctionnalités soient implémentées pour pouvoir se rendre compte des erreurs. Néanmoins, nous aurions dû être plus vigilants quant aux résultats que nous obtenions à ce stade du développement. En effet lorsque nous avons présenté les résultats, ceux-ci avaient l'air tout à fait cohérents, notamment puisque les graphes avaient exactement le même nombre de nœuds et d'arcs. C'est une erreur de notre part, cependant si nous avions détecté ce problème à ce moment-là, nous n'aurions certainement pas eu le temps de développer les autres fonctionnalités d'Iguana. Or ces fonctionnalités ont à présent été testé à l'aide des fichiers intermédiaires (produit par le script de réduction) de M. Miannay et elles sont correctes, nous obtenons les bons composants à la fin du processus.

7. CONCLUSION

Ce projet nous a permis d'aborder la notion de travail de groupe sur un travail de moyen terme (3 mois). En effet, nous n'avions pas encore eu l'occasion de travailler avec des clients attendant un résultat sur une telle période. Nous avons dû écrire un cahier des charges, comportant un ensemble de tâches à réaliser ainsi qu'une planification, prendre en mains les scripts de M. Miannay ainsi que s'approprier les notions scientifiques sous-jacentes et nous organiser avec nos encadrants pour faire des réunions régulières permettant de faire le point sur notre avancement et les objectifs à réaliser par la suite.

D'un point de vu technique, nous avons vu comment créer une application en Python mêlant une bibliothèque graphique, l'utilisation d'une application tierce ainsi que la transposition des scripts Python en exécutable. Notre obstacle principal a été l'installation des différentes bibliothèques nécessaires puisqu'il fallait faire en sorte que les versions soient toutes compatibles entre elles.

Enfin, nos échanges hebdomadaires avec Mme Guziolowski et M. Miannay nous ont également apporté beaucoup de choses, surtout au sujet de la biologie qui est un domaine dans lequel nous n'avons que peu de connaissances. On a pu également aborder le sujet de la recherche puisque ce travail fait suite au travail de thèse de M. Miannay. Ce projet pose les bases de l'implémentation et la mise en pratique des recherches de M. Miannay et a donc un potentiel énorme. La classification des patients est une partie indispensable du diagnostic médical et permet la mise en place de traitements beaucoup plus adaptés aux différentes maladies en particulier dans le cadre de myélomes multiples.

La prochaine étape de ce projet est d'implémenter un algorithme de classification en insérant des données expérimentales dans les graphes en tenant compte des colorations et composants obtenus avec les algorithmes de M. Miannay. Cela impliquera l'utilisation de techniques de classification de Machine Learning et fera l'objet d'un projet de groupe au prochain semestre.

8. ANNEXES

Annexe 1 : Article 4.2.4 de la thèse de M. Bertrand Miannay

Annexe 2 : Cahier des charges

Annexe 3 : Diagrammes UML

Annexe 4 : Liste des bibliothèques et logiciels utilisés

Annexe 5 : Présentation de l'IHM

Annexe 6 : Lien vers l'application et le guide d'utilisation

Annexe 7 : Exemples fichiers obtenus lors de la réduction

Annexe 1 : Article 4.2.4 de la thèse de M. Bertrand Miannay

4.2.4 Réduction de l'espace des solutions

Du fait de notre méthode de génération des solutions candidates, l'espace des solutions pour un graphe de n nœuds est de l'ordre de 2n. En raison de notre modèle de coloration à 2 signes avec des règles symétriques, nous pouvons observer qu'une coloration de graphe et son inverse (nœuds signés "up" sont signés "down" et inversement) ont le même score vis à vis des contraintes de minimisation de l'incohérence, des colorations imparfaites et des régulateurs pondérés imparfaits. Il est alors possible d'instancier un nœud avec une coloration fixée afin de diviser par 2 la taille de l'espace des solutions. Ainsi, par exemple avec la ligne 28, nous fixons le nœud node0 avec la coloration "down".

28 coloring(node0,down).

Pour améliorer cette réduction d'espace des solutions, nous proposons 3 méthodes de réduction du graphe (figure 4.5) qui peuvent être appliquées successivement sur le réseau de régulation avant de chercher les colorations parfaites. Ces méthodes identifient des nœuds du graphe qui feront partie du même composant. Ces nœuds seront fusionnés en un sous-composant. Un sous-composant peut être vu comme un sous-ensemble de nœuds appartenant au même composant, autrement dit si les nœuds d'un sous-composant font partie d'un seul composant, tous les nœuds de ce composant ne feront pas obligatoirement partie du sous-composant. La première et seconde réductions identifient des sous-composants. La fusion de nœuds en sous-composants permet de réduire le nombre de nœuds dans le graphe et donc l'espace des solutions à explorer. La troisième méthode réduit le nombre d'arcs et peut détecter des composants isolés du reste du graphe.

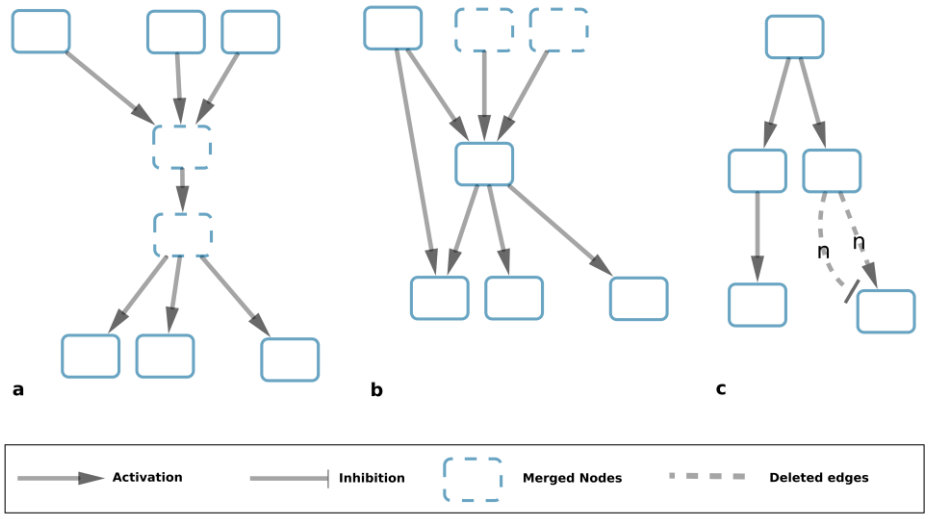

FIGURE 4.5 – Motifs recherchés par les 3 méthodes de réductions présentées dans ce chapitre. a : nœuds avec colorations corrélées dans les solutions cohérentes. b : nœuds avec colorations corrélées partageant les mêmes cibles. c : arcs avec les mêmes poids, source, cible et des signes opposés.

Annexe 2 : Cahier des charges

Voici le cahier des charges original, que nous avons rédigé au début du projet. On peut voir que nous avons revu les objectifs au fur et à mesure de notre avancement. En effet, la dernière partie ne paraissait pas fondamentale et comme il restait du temps nous avons ajouté deux fonctionnalités la coloration du graphe et l'export des composants. Celle-ci n'apparaissent donc pas dans la planification.

Liste des tâches à réaliser :

- Créer un lien entre le script Python (IGGY-poc) et CytoScape afin d'afficher les graphes colorés.
    - Utiliser la bibliothèque CyREST pour lier python et CytoScape.
    - Créer un graphe à partir des données avec IGGY-poc.
        - Identifier les commandes nécessaires à la création du graphe.
        - Les exécuter.
    - Adapter le format de la sortie de IGGY pour qu'elle soit acceptée par CytoScape.
    - Lancer la visualisation du graphe dans CytoScape à partir de CyREST.
    - Définir les critères de classification des nœuds dans CytoScape, en accord avec la méthode utilisée, pour réaliser la coloration.

- Création d'une IHM pour faciliter la construction des graphes.
    - Créer une interface graphique simple.
    - Gérer les différents paramètres dont IGGY a besoin pour fonctionner.
        - Ouverture de boite de dialogue pour aller chercher des fichiers.
        - Champs modifiables pour entrer des paramètres numériques.
        - Sélection de mode.

- Visualisation de la réduction des graphes étapes par étapes.
    - Modifier le moment de l'affichage en l'insérant dans la boucle d'exécution de IGGY.
    - Limiter l'impact au niveau du temps d'exécution.
        - Choisir une période d'actualisation ou un nombre d'étape entre chaque affichage.

Annexe 3 : Diagrammes UML

Voici les diagrammes que nous avons jugés utile à la conception de notre application. Ceux-ci permettent de comprendre à quels besoins répond Iguana ainsi que son fonctionnement interne. Une fonctionnalité ne fait néanmoins pas partie de ces diagrammes (l'export des composants) car nous avons décidé de l'ajouter une fois le projet bien avancé.

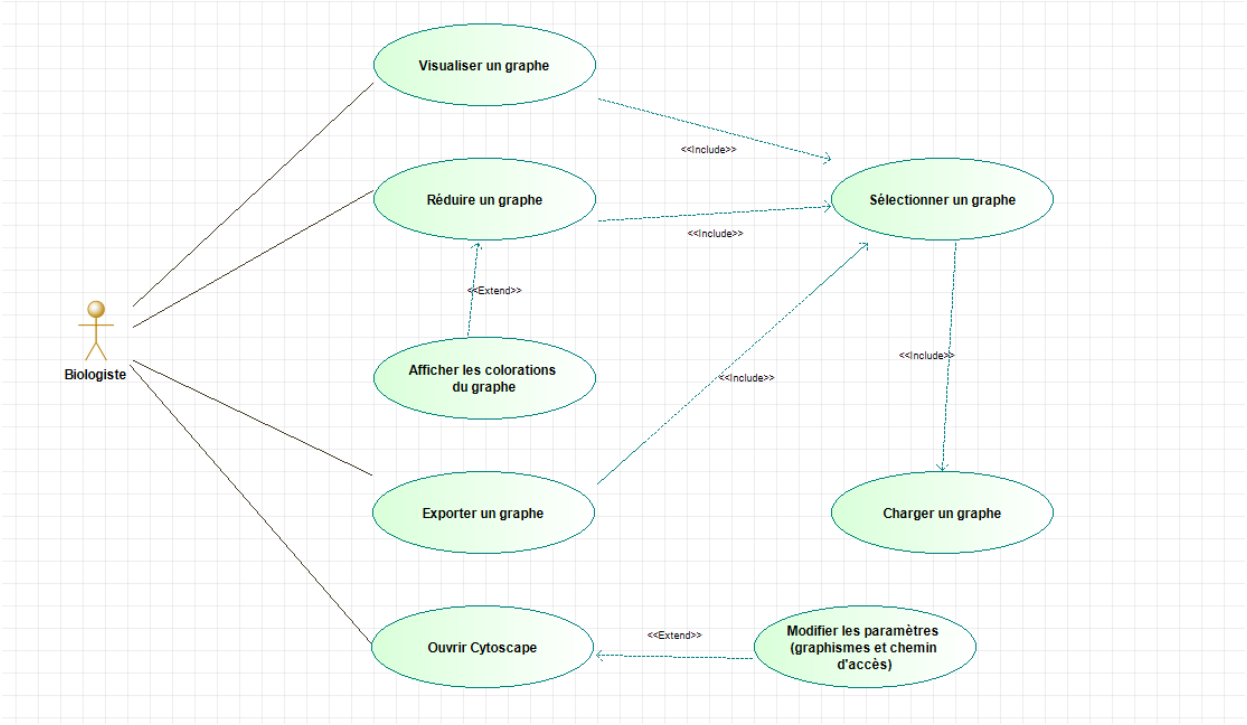

Figure 11 : Diagramme des cas d'utilisation

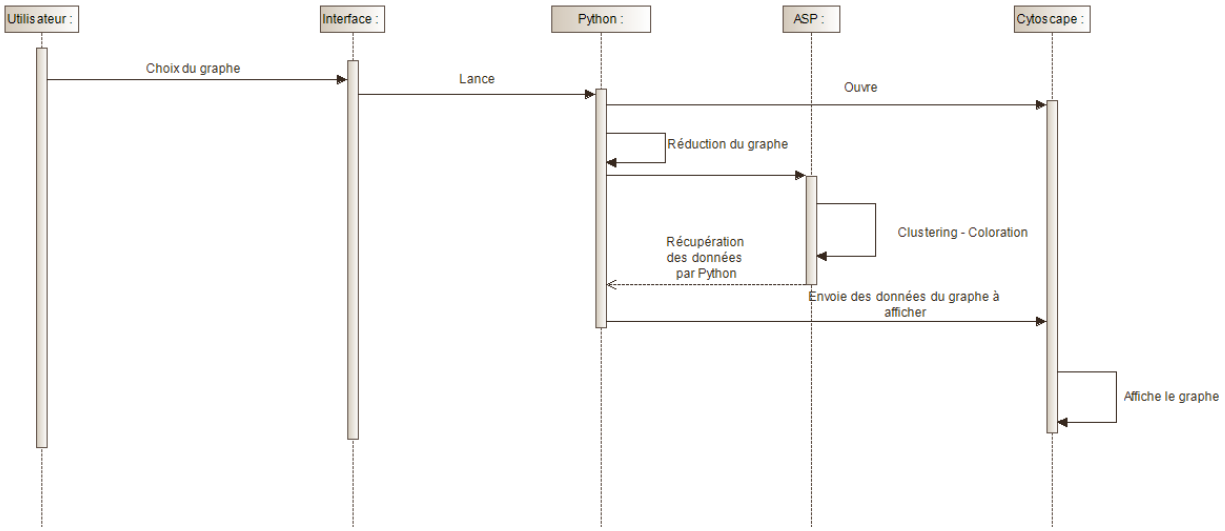

Figure 12 : Diagramme des séquences

Annexe 4 : Liste des bibliothèques et logiciels utilisés

Bibliothèques Python :

- networkx (1.10)
- numpy (1.13.3)
- pandas (0.21.1)
- psutil (5.4.2)
- py2cytoscape (0.6.2)
- pydot (1.2.3)
- python-igraph (0.7.1.post6)
- python-qt5 (0.1.10)
- scipy (1.0.0)
- cx_Freeze (5.0.1)

Logiciels :

- QtDesigner
- Cytoscape 3.5.1
- Clingo

Annexe 5 : Présentation de l'IHM

Iguana est composé de trois interfaces distinctes permettant chacune d'effectuer des tâches spécifiques : affichage du graphe, réduction, recherche des colorations et export des composants.

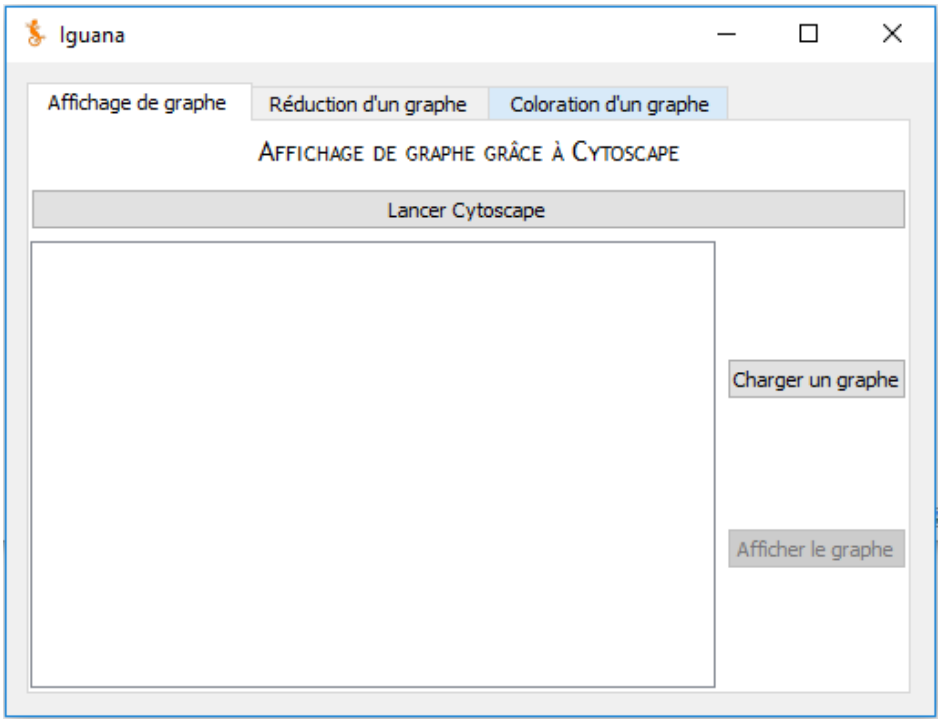

Figure 13 : Interface d'affichage

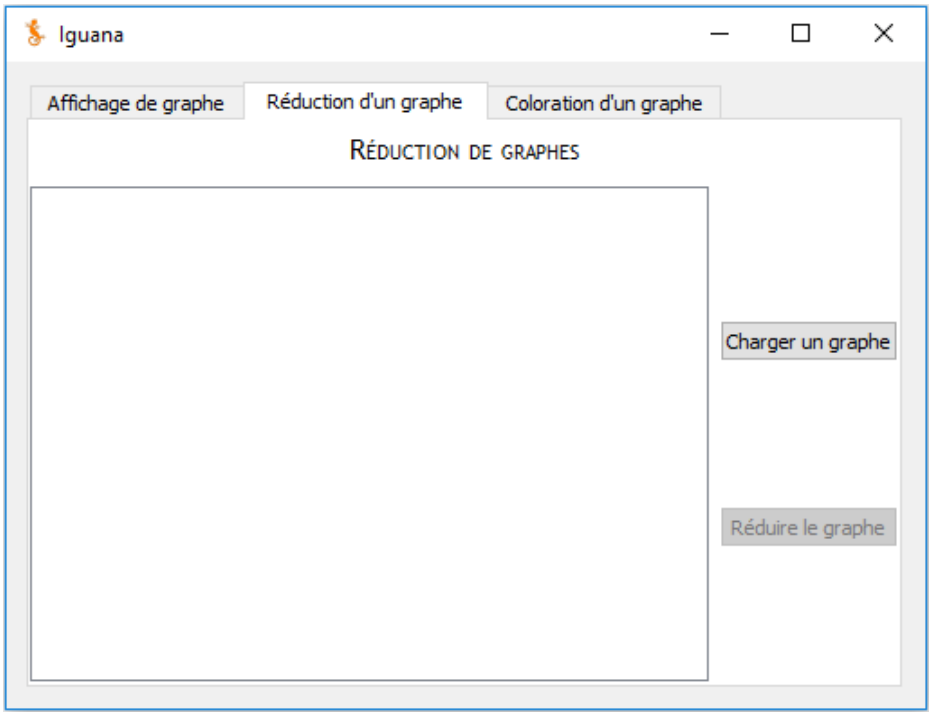

Figure 14 : Interface de réduction

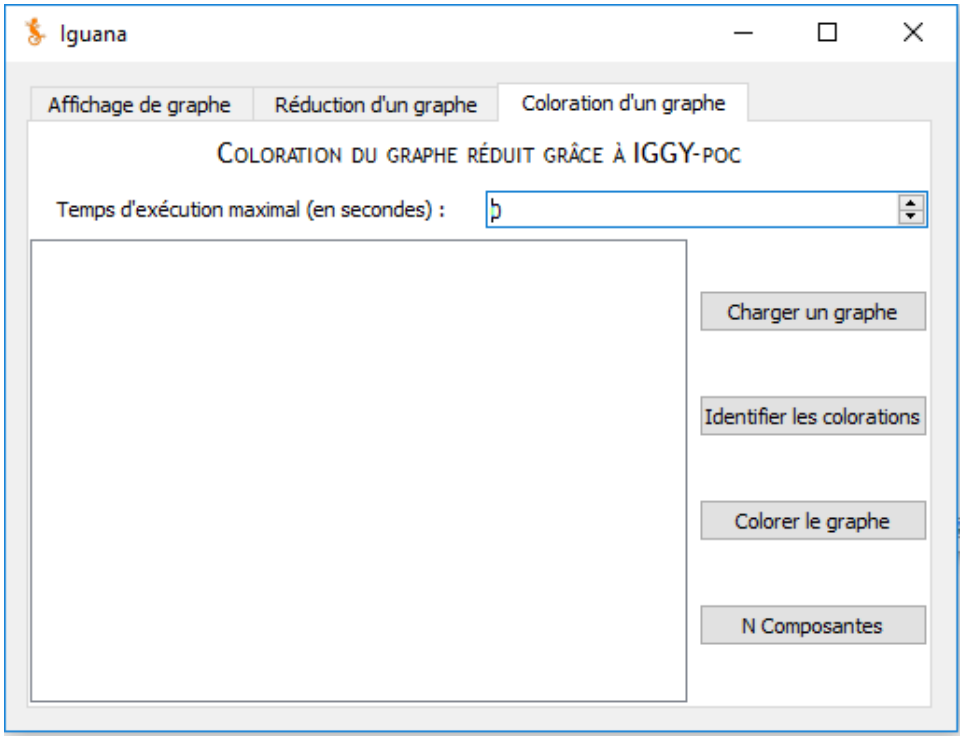

Figure 15 : Interface de coloration

Annexe 6 : Lien vers l'application et le guide d'utilisation

https://github.com/ipeter50/Iguana.git

Annexe 7 : Exemples fichiers obtenus lors de la réduction

graphe_example-reduced.sif

```
"pid_i_208152 +,LEFTPARP01100AROBASEnucleusTWOPOINTP05412AROBASEnucleusRIGHTPARAROBASEnucleus +,pid_i_204233 -" -1  "O14746 +,pid_i_204209 +"
"LEFTPARP30281TWOPOINTTWOPOINTP30279TWOPOINTTWOPOINTP24385TWOPOINTRIGHTPAR +" 1 "LEFTPARpid_m_200090TWOPOINTpid_m_200089TWOPOINTP42771RIGHTPAR +"
"LEFTPARP30281TWOPOINTTWOPOINTP30279TWOPOINTTWOPOINTP24385TWOPOINTRIGHTPAR +" 1 "LEFTPARpid_m_200090TWOPOINTpid_m_200089RIGHTPAR +,pid_i_210970 +,O43463 +"
"pid_i_201175 +,LEFTPARP05112TWOPOINTP24394BACKSLASHMod2404TWOPOINTP23458BACKSLASHMod153BACKSLASHactiveTWOPOINTP78552TWOPOINTO60674BACKSLASHMod544BACKSLASHa
+,pid_i_201208 +,P35372 +,pid_i_201209 +,P01833 +,pid_i_201200 +,Q9BQ08 +,pid_i_201203 +,P05106 +,pid_i_201204 +,P16109 +,pid_i_201207 +,Q14627 +,P24394 +"
"pid_i_207896 +,LEFTPARQ14790RIGHTPARBACKSLASHactive +,pid_i_207908 +,pid_i_207907 +" 1 "P55212BACKSLASHactive +,pid_i_208559 +,pid_i_208570 +,Q01826 +"
"pid_i_207896 +,LEFTPARQ14790RIGHTPARBACKSLASHactive +,pid_i_207908 +,pid_i_207907 +" 1 "P42574BACKSLASHactive +,pid_i_208558 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211146 +,O60469 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211151 +,P39060 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211148 +,P00441 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211153 +,Q9UG56 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211145 +,P17861 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211150 +,P56181 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211152 +,Q10567 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211141 +,P04155 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211147 +,P48552 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "pid_i_211154 +,P18859 +"
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" 1 "P55317AROBASEnucleus +,chromatin_remodeling +,pid_i_211129 +,P33260 +
+,pid_i_204258 +,P55851 +,pid_i_204260 +,P30613 +,pid_i_204274 +,Q16836 +,pid_i_204255 +,Q01581 +,pid_i_204275 +,P52945 +,pid_i_204279 +,P00734 +,pid_i_2042
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
+,LEFTPARP05412TWOPOINTP17535RIGHTPARAROBASEnucleus +,pid_i_207990 +,LEFTPARpid_m_201877TWOPOINTP03372TWOPOINTP05412RIGHTPARBACKSLASHactiveAROBASEnucleus +,
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
+,pid_i_207884 +,LEFTPARQ9NPF7TWOPOINTP29460RIGHTPARAROBASEextracellular_region +,pid_i_207844 +,LEFTPARQ9NPF7TWOPOINTP29460TWOPOINTQ5VWK5BACKSLASHMod2182TW
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
+,LEFTPARP01579RIGHTPAR +,pid_i_206256 +,LEFTPARP15260BACKSLASHMod540TWOPOINTP23458BACKSLASHMod153TWOPOINTO60674BACKSLASHMod544TWOPOINTP01579RIGHTPARBACKSLA
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
```

Figure 16: Fichier .sif d'un graphe réduit

graphe_example-reduced.sif | graphe_example-reduced-hash.txt

```
"pid_i_209095 +,Q9BZS1 +" : node95
"P00749 +" : node0
"pid_i_208152 +,LEFTPARP01100AROBASEnucleusTWOPOINTP05412AROBASEnucleusRIGHTPARAROBASEnucleus +,pid_i_204233 -" : node1
"P41134 +" : node14
"LEFTPARP30281TWOPOINTTWOPOINTP30279TWOPOINTTWOPOINTP24385TWOPOINTRIGHTPAR +" : node2
"pid_i_201175 +,LEFTPARP05112TWOPOINTP24394BACKSLASHMod2404TWOPOINTP23458BACKSLASHMod153BACKSLASHactiveTWOPOINTP78552TWOPOINTO60674BACKSLASHMod544BACKSLASHa
"pid_i_207896 +,LEFTPARQ14790RIGHTPARBACKSLASHactive +,pid_i_207908 +,pid_i_207907 +" : node3
"P03372 +,LEFTPARP03372TWOPOINTpid_m_201877RIGHTPARAROBASEnucleus +,pid_i_211143 +" : node4
"LEFTPARP05412BACKSLASHMod68TWOPOINTP15336BACKSLASHMod560RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206619 +,pid_i_206648 +,P05089 +,pid_i_206653 +,Q1446
"P06748 +" : node100
"pid_i_204682 +,P31749BACKSLASHMod2689BACKSLASHactive +,pid_i_204503 -,Q16822 -,pid_i_205188 +,P98177BACKSLASHMod239AROBASEnucleus +" : node6
"pid_i_201504 +" : node101
"P53350BACKSLASHMod432BACKSLASHactive +,pid_i_205084 +,Q5FBB7 +" : node102
"JNK_cascade +,pid_i_200639 +" : node7
"LEFTPARP42336TWOPOINTP27986RIGHTPARBACKSLASHactiveAROBASEplasma_membrane +,pid_i_202064 +,pid_i_203834 +,pid_i_210686 +,pid_i_208395 +,LEFTPARpid_m_200216T
+,pid_i_203511 +,pid_i_209041 +,LEFTPARP62330TWOPOINTpid_m_200576RIGHTPARBACKSLASHinactive +,pid_i_203524 +,pid_i_203521 +,pid_i_203501 +,pid_i_203498 +,pid
+,LEFTPARP12830-P1TWOPOINTP35222TWOPOINTP35221RIGHTPARAROBASEbasolateral_plasma_membrane +,pid_i_202068 +,pid_i_210926 +,Q13153BACKSLASHMod4555BACKSLASHacti
+,Q02750BACKSLASHMod277BACKSLASHactiveAROBASEcytoplasm +,pid_i_202065 +,pid_i_203507 +,pid_i_205524 +,pid_i_203523 +,pid_i_203503 +,pid_i_204897 +,O15530BAC
"P01343 +,pid_i_204886 +,pid_i_206222 +,pid_i_201750 +,LEFTPARP08069AROBASEextracellular_regionTWOPOINTP08069-P1AROBASEtransmembraneTWOPOINTP01343TWOPOINTP3
"pid_i_204765 +" : node10
"pid_i_210971 +" : node11
"P55317AROBASEnucleus +,chromatin_remodeling +,pid_i_211129 +,P33260 +,pid_i_211157 +,P04004 +,pid_i_206210 +,LEFTPARQ9Y624TWOPOINTP06756TWOPOINTP05106RIGHT
+,Q01581 +,pid_i_204275 +,P52945 +,pid_i_204279 +,P00734 +,pid_i_204267 +,P80370 +,pid_i_204248 +,P02768 +,pid_i_211123 +,P01275 +,pid_i_204257 +,P49748 +,Q
"pid_i_202791 +,O60716 +,P35221 +" : node13
"P05412 +,LEFTPARP05412TWOPOINTP17535RIGHTPARAROBASEnucleus +,pid_i_207990 +,LEFTPARpid_m_201877TWOPOINTP03372TWOPOINTP05412RIGHTPARBACKSLASHactiveAROBASEnu
"pid_i_206588 +" : node104
"LEFTPARP42224BACKSLASHMod1077RIGHTPARBACKSLASHactiveAROBASEnucleus +,pid_i_206305 +,P29466 +" : node96
"P52564BACKSLASHMod3903BACKSLASHactive +,pid_i_203069 +,pid_i_204397 +,pid_i_209376 +,pid_i_205851 +,pid_i_208231 +,P53778BACKSLASHMod1776BACKSLASHactive +,
"pid_i_200400 +" : node16
"P42574BACKSLASHactive +,pid_i_208558 +" : node105
"pid_i_202908 +" : node106
"pid_i_200709 +,LEFTPARP06213-P1AROBASEextracellular_regionTWOPOINTP06213-P2BACKSLASHMod26BACKSLASHactiveAROBASEplasma_membraneTWOPOINTP01308AROBASEextracel
+,LEFTPARP00533BACKSLASHMod11TWOPOINTP01133TWOPOINTP62993TWOPOINTQ07889RIGHTPARBACKSLASHactiveAROBASElysosome +,pid_i_208445 +,Q07889 +" : node107
```

Figure 17: Table de hachage du graphe réduit