Projet Réseaux Biologiques en Python

Emmanuelle Becker, Olivier Dameron

Semestre automne-hiver 2020

Table des matières

Ce projet est dédié à l'étude des graphes d'interactions entre protéines. Vous allez être amené·e à écrire des outils permettant de manipuler ces graphes, et à introduire au sein de ces graphes une notion d'interaction entre domaines protéiques.

Chapitre 1

Semaine 1 : Lecture d'un graphe d'interactions entre protéines

1.1 Préambule

Les graphes d'interactions qui vous seront fournis se trouvent dans des fichiers dont le format est le suivant : la première ligne indique le nombre d'interactions (le nombre de lignes du fichier -1), et chaque ligne suivante décrit une interaction. Ces interactions n'ont pas d'orientation ; le graphe d'interaction est donc un graphe non orienté.

1.1.1 Question de compréhension

Dessinez un petit graphe d'interactions (une dizaine), et écrivez le fichier qui y sera associé. Échangez votre dessin avec vos collègues, écrivez le fichier correspondant à leur graphe et déterminez si vous avez compris la même chose.

1.2 Structure de données pour stocker le graphe

Cette section a pour objectif de comparer différentes stratégies pour représenter un graphe d'interactions.

Une première manière de stocker ce graphe est d'utiliser un dictionnaire où les clés sont les sommets et les valeurs associées aux clés sont les voisins des sommets. On peut aussi imaginer de stocker ce graphe dans une liste de couples (X, Y) qui représentent toutes les interactions.

1.2.1 Question préliminaire

Écrire une fonction qui lise un graphe d'interactions entre protéines dans un fichier tabulé et le stocke dans un dictionnaire. Le nom de la fonction sera `read_interaction_file_dict` et la fonction prendra en unique argument le nom du fichier à lire. Le dictionnaire créé sera retourné par la fonction.

1.2.2 Question préliminaire

Écrire une fonction qui lise un graphe d'interactions entre protéines dans un fichier et le stocke dans une liste de couples. Le nom de la fonction sera `read_interaction_file_list` et la fonction prendra en unique argument le nom du fichier à lire. La liste créée sera retournée par la fonction.

1.2.3 Question bonus

Écrire une fonction qui lise un graphe d'interactions entre protéines dans un fichier et le stocke dans une matrice d'adjacence (vous pouvez utiliser `numpy` pour les matrices en python, et wikipédia pour la définition de matrice d'adjacence).
Le nom de la fonction sera `read_interaction_file_mat` et la fonction prendra en unique argument le nom du fichier à lire. La matrice créée sera retournée par la fonction.

1.2.4 Question préliminaire

Écrire une fonction nommée `read_interaction_file`, qui a partir d'un fichier d'interactions, retourne un couple `(d_int, l_int)` dont le premier élément est le dictionnaire représentant le graphe, et le second élément est la liste d'interactions représentant le même graphe.

1.2.5 Question bonus

Pour un gros graphe d'interactions, quelle(s) stratégie(s) adopteriez-vous pour ne pas trop dégrader les performances de la fonction `read_interaction_file` ?

1.2.6 Question test

Il est fondamental de tester le bon comportement de vos fonctions de lecture avant de poursuivre plus avant dans l'étude des réseaux d'interactions protéine-protéine. Vos fonctions ne sont pas utilisables si nous n'avons pas moyen de nous assurer qu'elles font effectivement ce pour quoi elles ont été conçues.
Dans un fichier à part, pour lequel vous choisirez un nom explicite, préparez toute une série de tests pour vérifier que vos fonctions ont le comportement que nous attendons d'elles.
Pour les courageu·ses·x, c'est l'occasion de creuser la notion de test unitaire [1].

1.2.7 Question test

Écrire une fonction nommée `is_interaction_file` dont l'objectif est de vérifier que le fichier est bien au format attendu pour être lu correctement. Cette fonction prend en argument un fichier d'interaction, et renvoie `true` is le format est correct et `false` sinon. Travaillez à partir de plusieurs fichiers tests, certains respectant les spécifications du format demandés, d'autres non. Par exemple :

1. `https://docs.python.org/fr/3/library/unittest.html`

1. fichier ne comportant pas la première ligne qui compte le nombre d'interactions ;
2. fichier vide ;
3. fichier dont la première ligne contient un nombre qui n'est pas le nombre d'interactions ;
4. fichier contenant une ligne qui ne comporte pas le bon nombre de colonnes...

1.3 Chapitre 1 : Structurez, commentez et déposez votre travail

1.3.1 Charge de travail chapitre 1

Pour cette première semaine de projet, nous estimons la charge de travail à entre 2h et 4h de programmation (selon votre aisance), et 2h de nettoyage de code et commentaires. La création du projet git et son dépôt pourront vous prendre un quart d'heure de plus.

1.3.2 Objectifs

1. Vos fonctions devront être écrites propremment.
2. Vos noms de variables doivent être cohérents entre eux (pas de mélange de langue français - anglais, et comme le code est commencé en anglais, tous les noms de variables sont en anglais)
3. Vos noms de fonctions et de variables ont la même syntaxe (ici tout en minuscule avec des tirets du bas pour séparer les mots).
4. Vos noms de variables doivent contenir un suffixe indiquant leur type.
5. Vos noms de variables doivent faire moins de 20 caractères de long.
6. Toutes vos fonctions doivent être préfixées avec des commentaires qui indiquent le but de la fonction et sa signature. C'est sans doute le bon moment pour enfin lire la documentation sur les docstrings [2] (l'équivalent des javadoc pour python).
7. Vos fonctions ne doivent pas faire plus de 30 lignes (et les lignes doivent faire une taille raisonnable).
8. Votre code doit être déposé sous git. Idéalement pas en une fois à la fin, mais de façon incrémentale au fur et à mesure que vous ajoutez de nouvelles fonctionnalités ou que vous corrigez des bugs.
9. Vos enseignants doivent avoir accès à votre projet sous git.

1.3.3 Progresser en gestion de projet

Cette semaine, vous devrez vous familiariser avec : `git`, avec la documentation de fonction `docstring`, et pour les plus téméraires avec les tests unitaires.

2. `https://www.python.org/dev/peps/pep-0257/`

Chapitre 2

Semaine 2 : Exploration du graphe d'interactions protéine-protéine

Dans les questions suivantes, vous aurez besoin de lire, voire d'écrire, des graphes d'interactions entre protéines. Il ne vous sera jamais précisé quelle structure de données utiliser. Vous devrez trouver par vous-même, en fonction de la question posée, laquelle sera la plus adaptée.

2.1 Exploration du graphe global

2.1.1 Question exploration

Écrire une fonction `count_vertices(file)` qui compte le nombre de sommets d'un graphe.

2.1.2 Question exploration

Écrire une fonction `count_edges(file)` qui compte le nombre d'arêtes d'un graphe.

2.1.3 Question nettoyage

Écrire une fonction `clean_interactome(file)` qui lit un fichier contenant un graphe d'interactions protéine-protéine et y enlève (i) toutes les interactions redondantes, et (ii) tous les homo-dimères. Le graphe obtenu sera écrit dans un nouveau fichier au même format que le format de départ.

2.1.4 Question test

Il est fondamental de tester le bon comportement de vos fonctions avant de poursuivre plus avant dans l'étude des réseaux d'interactions protéine-protéine. Vos fonctions ne sont pas utilisables si nous n'avons pas moyen de nous assurer qu'elles font effectivement ce pour quoi elles ont été concues. Préparez toute une série de tests pour vérifier que vos fonctions ont le comportement que nous attendons d'elles.

2.2 Travail autour du degré des sommets

On nomme degré d'un sommet le nombre d'arêtes incidentes au sommet. Par exemple, dans le graphe de départ, si A est connecté à B et F, le degré du sommet A est 2.

2.2.1 Question degré

Écrire une fonction `get_degree(file, prot)` qui prend en argument un fichier contenant un graphe d'interactions protéine-protéine et le nom d'une proteine, et qui renvoie le degré de cette protéine dans le graphe.

2.2.2 Question degré

Écrire une fonction `get_max_degree(file)` qui renvoie le nom de la protéine de degré maximal ainsi que le degré de cette protéine.

2.2.3 Question degré

Écrire une fonction `get_ave_degree(file)` qui calcule le degré moyen des protéines du graphe.

2.2.4 Question degré

Écrire une fonction `count_degree(file, deg)` qui calcule le nombre de protéines du graphe dont le degré est exactement égal à `deg`.

2.2.5 Question degré

Écrire une fonction `histogram_degree(file, dmin, dmax)` qui calcule, pour tous les degrés `d` compris entre `dmin` et `dmax`, le nombre de protéines ayant un degré `d`. Si vous êtes courageux, essayer d'afficher le résultat sous la forme d'un histogramme en comme par exemple :

```
1 **
2 **
3 **
```

Que constatez-vous ? Quelle analyse pouvez-vous faire au vu de cette distribution ?

2.3 Semaine 2 : Structurez, commentez et déposez votre travail

2.3.1 Charge de travail semaine 2

Pour cette semaine de projet, nous estimons la charge de travail à entre 2h et 4h de programmation (selon votre aisance), et 2h de nettoyage de code et commentaires. La

compréhension des notions nouvelles dans le projet (sommets, arêtes, degrés) pourrait vous predre une heure de lecture en plus (maximum).

2.3.2 Objectifs de la semaine 2

1. Vos fonctions devront être écrites propremment.
2. Vos noms de variables doivent être cohénrents entre eux (pas de mélange de langue français - anglais, et comme le code est commencé en anglais, tous les noms de variables sont en anglais)
3. Vos noms de fonctions et de variables ont la même syntaxe (ici tout en minuscule avec des tirets du bas pour séparer les mots).
4. Vos noms de variables doivent contenir un suffixe indiquant leur type.
5. Vos noms de variables doivent faire moins de 20 caractères de long.
6. Toutes vos fonctions doivent être préfixées avec des commentaires qui indiquent le but de la fonction et sa signature.
7. Auncune fonction ne doit faire plus de 30 lignes.
8. Votre code doit être déposé sous git.
9. Vos enseignants doivent avoir accès à votre projet sous git.
10. Votre mise à jour du code doit être visible sur l'architecture du projet git.

Chapitre 3

Semaine 3 : Modification des spécifications, python orienté objet

3.1 Problème actuel

Vous aurez peut-être noté que notre organisation n'est pas optimale. En effet, le graphe d'interaction protéine-protéine peut-être lu sous forme de liste, de dictionnaire, voire de matrice. Vous avez aussi vu que certaines structures de données sont plus adaptées à certaines fonctions d'exploration que d'autres. C'est la raison pour laquelle, jusqu'à présent, dans chaque fonction d'exploration, on commence par lire le fichier avec la méthode de stockage adaptée. Mais ce n'est pas optimal, car du coup, on passe son temps à relire le même fichier...

3.1.1 Question de compréhension

Lors de l'execution de la fonction `histogram_degree`, combien de fois a t-on lu le fichier d'interactions ?

3.2 Créons un objet interactome

Nous proposons dans cette section de créer un objet (comme en programmation orientée objet l'année dernière). Cet objet représentera un réseau d'interactions protéine-protéine (appelé interactome), et les attributs de la classe seront la liste des interactions, le dictionnaire des interactions, et la matrice d'adjacence des interactions. De cette façon, on pourra lire le fichier un nombre raisonnable de fois pour construire l'objet, et ensuite manipuler la structure de données qui nous convient sans avoir à relire le fichier à chaque fois...

3.2.1 Lecture...

Allez lire le chapitre 19 du livre Python pour les sciences de la vie.

3.2.2 Modification

Créez une nouvelle branche dans votre dépôt git. Dans cette nouvelle branche, créez une classe Interactome :

- avec un attribut `int_list` qui stockera l'interactome sous la forme d'une liste d'interaction ;
- avec un attribut `int_dict` qui stockera l'interactome sous la forme d'un disctionnaire d'adjacence
- avec un attribut `proteins` qui stockera la liste des interactants.

3.2.3 Constructeur

Ajoutez un constructeur, qui prend en entrée un fichier d'interactions, et remplit les différents attributs.

3.2.4 Méthodes...

Transformez toutes les méthodes que vous avez écrites précédemment en membres de la classe `Interactome`.

Chapitre 4

Chapitre 4 : Calcul des composantes connexes d'un graphe d'interactions entre protéines

4.1 Travail sur les composantes connexes

On dit qu'un graphe est connexe si quels que soient les sommets X et Y, il existe un chemin qui relie X à Y. Malheureusement, les graphes d'interactions entre protéines sont rarement connexes. On cherche alors à savoir combien de composantes connexes composent ces graphes... Les questions suivantes listent des méthodes à écrire et leur spécification. Soyez rusés, lisez le sujet jusqu'au bout et réfléchissez à l'ordre optimal pour implémenter vos méthodes...

4.1.1 Question préliminaire

Recherchez sur internet la définition de « composante connexe ».

4.1.2 Question composantes connexes

Écrivez une fonction `count_CC()` qui calcule le nombre de composantes connexes d'un graphe, et donne pour chacune d'elle sa taille (c'est à dire son nombre de protéines).

4.1.3 Question composantes connexes

Écrivez une fonction `write_CC()` qui écrive dans un fichier les différentes composantes connexes d'un graphe. Vous utiliserez le format suivant :

1. une ligne par composante connexe
2. le premier élément de la ligne est la taille de la composante connexe,
3. ensuite vous ajouterez la liste des sommets qui composent cette composante connexe.

4.1.4 Question composantes connexes

Écrivez une fonction `extract_CC(prot)` qui renvoie tous les sommets de la composante connexe de `prot`.

4.1.5 Question composantes connexes

Écrivez une fonction `compute_CC()` qui renvoie une liste `lcc` dont chaque élément `lcc[i]` correspond au numéro de composante connexe de la protéine à la position `i` dans la liste des protéines du graphe (qui est un attribut de notre classe).

4.2 Travail sur la densité

4.2.1 Question densité

Dans le cas d'un graphe non orienté simple $G = (V, E)$, la densité est le rapport :

$$D = \frac{2\ |E|}{|V| \cdot (|V| - 1)}$$

Il représente le nombre d'arêtes présentes, par rapport au nombre d'arêtes total qu'il pourrait théoriquement y avoir. Écrivez une fonction `density()` qui renvoie la densité du graphe.

4.2.2 Question coefficient de clustering

En théorie des graphes et en analyse des réseaux sociaux, le coefficient de clustering d'un graphe (aussi appelé coefficient d'agglomération, de connexion, de regroupement, d'agrégation ou de transitivité), est une mesure du regroupement des nœuds dans un réseau. Plus précisément, ce coefficient est la probabilité que deux nœuds soient connectés sachant qu'ils ont un voisin un commun. C'est l'un des paramètres étudiés dans les réseaux sociaux (par exemple) : les amis de mes amis sont-ils mes amis ?

Le coefficient de clustering local d'un nœud `i` est défini comme :

$$C_i = \frac{|\text{triangles de sommet } i|}{|\text{paires de voisins distincts de } i|}$$

C'est la fraction de ses paires de voisins connectés, égale à 0 si $d_i \leq 1$ par convention. Écrivez une fonction `clustering(prot)` qui renvoie le coefficient de clustering du sommet `prot` au sein du graphe.

4.3 Testez vos méthodes

Dans un second fichier, écrivez un programme principal où vous testez vos nouvelles méthodes. Pensez à bien travailler vos tests : ils doivent nous convaincre que votre code fonctionne...

4.3.1 Charge de travail semaine 4

Pour cette semaine de projet, nous estimons la charge de travail à entre 4h et 8h de programmation (selon votre aisance), et 2h de nettoyage de code et commentaires.

4.3.2 Objectifs de la semaine 4

1. Continuez sur votre lancée : on reste en python orienté objet.
2. Vos fonctions devront être écrites propremment.
3. Vos noms de variables doivent être cohénrents entre eux (pas de mélange de langue français - anglais, et comme le code est commencé en anglais, tous les noms de variables sont en anglais)
4. Vos noms de fonctions et de variables ont la même syntaxe (ici tout en minuscule avec des tirets du bas pour séparer les mots).
5. Vos noms de variables doivent contenir un suffixe indiquant leur type.
6. Vos noms de variables doivent faire moins de 20 caractères de long.
7. Toutes vos fonctions doivent être préfixées avec des commentaires qui indiquent le but de la fonction et sa signature.
8. Auncune fonction ne doit faire plus de 30 lignes.
9. Votre code doit être déposé sous git.
10. Votre mise à jour du code doit être visible sur l'architecture du projet git.

Chapitre 5

Chapitre 5 : Recherche des domaines qui composent les protéines

5.1 Extension de la structure de données

Vous allez maintenant ajouter une information de « composition en domaines » au sein de votre graphe d'interactions entre protéines. **Comment stocker cette nouvelle information ?**

La manière la plus simple est sans doute d'étendre le dictionnaire qui servait à représenter les graphes d'interactions entre protéines. En effet, auparavant le format était le suivant :

```
dict_graphe = {}
dict_graphe['protA'] = ['protC','protB']
```

Ce format peut s'étendre :

```
dict_graphe = {}
dict_graphe['protA'] = {}
dict_graphe['protA']['domaines'] = ['domaineA1','domaineA2',
                                    'domaineA3', 'domaineA2']
dict_graphe['protA']['voisins'] = ['protC','protB']
```

NB : remarquez qu'une protéine peut avoir plusieurs fois le même domaine ! Par exemple, le récepteur de l'insuline `INSR_HUMAN` a deux fois le domaine « Recep L domain ».

5.2 Base(s) de données pour obtenir les domaines des protéines

5.2.1 Question préliminaire 1

Visitez la banque de données Pfam [1]. Quel genre d'information contient-elle ? Vous pourrez par exemple vous intéresser à la protéine *Insulin receptor* (`INSR_HUMAN`).

1. `https://pfam.xfam.org/`

5.2.2 Question préliminaire 2

Comme il y a une grande hétérogénéité des noms de gènes ou de protéines (même avec la nomenclature officielle, qui contient des synonymes, et dont la syntaxe n'est pas toujours bien suivie), il est en général plus prudent d'utiliser un identifiant par ex dans Uniprot. Quel est l'identifiant Uniprot de `INSR_HUMAN` [2] ?

5.2.3 Question préliminaire 3

Dans pfam, quels sont les domaines de `INSR_HUMAN` ? Néanmoins, l'information de la base pfam n'est pas très pratique à obtenir par un programme. Est-ce que vous retrouvez cette information dans la page Uniprot ?

5.3 Récupération systématique des identifiants des protéines

Pour la question 4, choisissez la version que vous souhaitez.

5.3.1 Question 4 - version 1 (au choix)

Allez sur la page Uniprot permettant de convertir des noms de gènes en identifiants Uniprot, `https://www.uniprot.org/uploadlists/`. Avec votre navigateur, donnez en entrée tous les noms de gènes de votre fichier, et demandez les correspondances en Uniprot ID. Téléchargez le fichier obtenu et intégrez le comme fichier de données à votre projet.

5.3.2 Question 4 - version 2 (au choix)

Allez sur le site d'Uniprot. Cherchez sur le site un fichier à télécharger dans lequel vous pourrez par la suite extraire une correspondance entre nom de gène et identifiant Uniprot. Téléchargez le fichier obtenu et intégrez le comme fichier de données à votre projet.

5.3.3 Question 5

Vous allez définir une nouvelle classe `Proteome`. La classe protéome aura trois attributs :

- la liste des noms de protéines du proteome (exemple de nom de protéine : `INSR_HUMAN`)
- un dictionnaire qui a comme clé les nom de protéines et comme valeur l'identifiant Uniprot ID (un identifiant Uniprot ressemble à `P06213`)
- un dictionnaire qui a comme clé l'identifiant UniprotID et comme valeur le nom de protéine associé.

La classse protéome contiendra un constructeur qui prend en paramètre votre fichier de la question 4, et contruit les dictionnaires et liste demandés.

2. `https://www.uniprot.org/uniprot/P06213`

5.3.4 Question 6

Dans votre classe `Interactome`, écrivez une fonction `xlink_Uniprot`, qui pour chaque protéine de l'interactome, étende le dictionnaire stockant l'interactome de la façon suivante :

```
dict_graphe = {}
dict_graphe['protA'] = {}
dict_graphe['protA']['UniprotID'] = "P06213"
dict_graphe['protA']['voisins'] = ['protC','protB']
```

5.4 Récupération des domaines des protéines

5.4.1 Question 7 préliminaire

Pour extraire de l'information de la page Uniprot d'une protéine, il sera plus simple de passer par la version textuelle d'Uniprot plutôt que par les pages HTML.

1. à partir de la page HTML de `P06213` [3], repérez les domaines pfam ;
2. en ajoutant `.txt` à l'URL de la page d'une protéine, vous obtenez une version traitable bien plus facilement [4]. Où sont les domaines pfam que vous aviez identifiés au point précédent ? Comment savoir combien de fois chaque domaine est présent dans la protéine ?

5.4.2 Question 8

Écrivez une fonction `get_protein_domains(p)` qui permette de (i) rechercher la page Uniprot correspondant à cette entrée p et (ii) renvoie le nom du ou des domaines contenus par cette protéine p.
Vous aurez bien entendu besoin des modules `urllib` [5] (et sans doute aussi de tempfile [6] pour créer des fichiers temporaires) pour (i) et `re` [7] pour (ii).

5.4.3 Question 9

Écrivez une fonction `xlink_domains(p)` qui permette, pour chaque protéine de votre interactome, d'étendre votre structure de donnée dictionnaire pour ajouter les domaines des protéines.
Attention ! Essayez de faire en sorte que votre fonction ne recherche pas plusieurs fois la composition en domaines de la même protéine.

3. `https://www.uniprot.org/uniprot/P06213`
4. `https://www.uniprot.org/uniprot/P06213.txt`
5. `https://docs.python.org/3/howto/urllib2.html`
6. `https://docs.python.org/3/library/tempfile.html`
7. `https://docs.python.org/3/library/re.html`

Chapitre 6

Semaine 6 : Analyse de la composition en domaines de l'interactome

6.1 Notes aux étudiant·e·s

Vous entrez dans la partie où, au delà d'implémenter de nouvelles fonctionalités, vous allez être en mesure de présenter des résultats. Soyez attenti·f·ve·s : différents points sont abordés (on vous demande des courbes, des figures, des résultats), et nous nous attendons à ce que vous nous les présentiez le jour de la soutenance orale.

Cette semaine 6 propose succéssivement différentes pistes à explorer. Différentes questions sont succéssivement posées :

— sur la co-occurrence des domaines dans les protéines
— sur la co-occurrence des domaines dans les protéines en interaction directe
— sur la représentativité de ces résultats (aspects statistiques)

Choisissez les pistes que vous souhaitez explorer (toutes, ou pas), et allez le plus loin possible.

Enfin, vous entrez dans une section plus « à la carte ». Il devient donc crucial, pour que nous puissions suivre :

— que vous commentiez correctement vos fonctions (pensez docstring ; d'ailleurs, faites plus qu'y penser, utilisez-les)
— que votre programme principal soit clair sur vos différentes analyses.

6.2 Quels sont le domaines qui apparaissent fréquemment dans les mêmes protéines ?

6.2.1 Question liste des protéines

Dans la classe `Interactome`, écrivez une fonction `ls_proteins()` qui renvoie la liste non redondante de toutes les protéines que l'on trouve le graphe d'interactions protéine-protéine.

6.2.2 Question liste des domaines

Dans la classe `Interactome`, écrivez une fonction `ls_domains()` qui renvoie la liste non-redondante de tous les domaines que l'on trouve dans les protéines qui composent le graphe d'interactions protéine-protéine.

6.2.3 Question liste des domaines

Dans la classe `Interactome`, écrivez une fonction `ls_domains_n(n)` qui renvoie la liste non-redondante de tous les domaines que l'on trouve au moins n fois dans les protéines qui composent le graphe d'interactions protéine-protéine.

6.2.4 Question distribution du nombre de domaines par protéine (optionnelle)

Dans la classe `Interactome`, écrivez une fonction `nbDomainsByProteinDistribution()` qui renvoie un dictionnaire dont les clés correspondent aux nombres de domaines que l'on trouve dans les protéines, et la valeur associée à chaque clé est le nombre de protéines comportant ce nombre de domaines.
Bonus : affichez l'histogramme correspondant (il y a y protéines qui comportent exactement x domaines).

6.2.5 Question distribution du nombre de protéines par domaine (optionnelle)

Dans la classe `Interactome`, écrivez une fonction `nbProteinsByDomainDistribution()` qui renvoie un dictionnaire dont les clés correspondent aux nombres de protéines associées à un domaine, et la valeur associée à chaque clé est le nombre de domaines présents dans ce nombre de protéines.
Bonus : affichez l'histogramme correspondant (il y a y domaines qui sont présents dans x protéines).

6.2.6 Question co-occurrence des domaines (1)

Dans la classe `Interactome`, écrivez une fonction `co_occurrence(dom_x,dom_y)` qui calcule et renvoie le nombre de co-occurrences des domaines x et y dans les proteines de notre interactome.

6.2.7 Question co-occurrence des domaines (2)

Dans la classe `Interactome`, on souhaite maintenant écrire une fonction qui va calculer un tout nouveau réseau/graphe : le graphe des co-occurrences entre domaines. C'est un graphe dont les sommets sont les domaines. Il existe une relation entre les domaines x et y si et seulement si il existe au moins une protéine au sein de laquelle on trouve les domaines x et y ensemble.

Pour créer ce nouveau graphe, on pourrait être tenté·e·s de se re-servir de notre classe `Interactome`, mais ça ne serait pas super. En effet, les fonctions comme `xlink_domains` etc n'auraient plus aucun sens sur un réseau de domaines...

Donc, au final, nous allons vous demander de créer une nouvelle structure, `DomainGraph`, qui sera faite pour les graphes de domaines. Vous y insererez tous les champs et toutes les méthodes que vous jugerez nécessaires. [1]

Écrivez une fonction `generate_cooccurrence_graph` qui crée une instance et renvoie un nouvel objet `DomainGraph`, dont les sommets sont les domaines. Il doit exister une interaction entre les sommets x et y uniquement lorsque ces deux domaines sont co-occurrents dans au moins une protéine du graphe.

6.2.8 Question topologie (1)

Quelle est la densité de ce graphe des domaines ? Ajoutez une fonction pour répondre à cette question si besoin...

6.2.9 Question topologie (2)

Quels sont les 10 domaines ayant le plus grand nombre de voisins ? Ajoutez une fonction pour répondre à cette question si besoin.

6.2.10 Question topologie (3)

Quelle est les 10 domaines ayant le plus petit nombre de voisins ? Ajoutez une fonction pour répondre à cette question si besoin. Êtes-vous bloqué·e par cette borne à 10 domaines ?

6.2.11 Question topologie (4)

Les domaines ayant le plus grand nombre de voisins sont-ils tout simplement ceux qui sont présents le plus grand nombre de fois dans les protéines de départ ?

6.2.12 Question topologie (5)

Combien de domaines sont en co-occurrence avec eux-mêmes ? Ajoutez une fonction pour répondre à cette question si besoin.

6.2.13 Question co-occurrence des domaines (3)

Dans notre graphe des co-occurrences, on ne distingue pas les co-occurrences fréquentes de celles qui le sont moins. Or c'est une information importante.

1. On remaquera que DomainGraph est une structure proche de Interactome, et on pourrait se poser la question de la possibilité d'avoir une classe plus générale `Graph` dont hériteraient à la fois `Interactome` et `DomainGraph`. Nous n'avons pas prévu de traiter cette question dans ce projet, mais rien ne vous empêche de le faire de votre côté.

Écrivez une fonction `generate_cooccurrence_graph_np(n)` qui crée une instance et renvoie un nouvel objet `DomainGraph`, dont les sommets sont les domaines. Il doit exister une interaction entre les sommets x et y uniquement lorsque ces deux domaines sont co-occurrents dans au moins n protéines du graphe.

6.2.14 Question lien entre topologie et seuil choisi (1)

Dans la fonction `generate_cooccurrence_graph_np`, plus le seuil n choisi est grand, plus le sous-graphe généré est petit (et donc la densité diminue). Représentez de façon graphique le lien entre n et la densité du graphe.

Votre graphe devra avoir en abcisse n et en ordonnée la densité du sous-graphe obtenu. Vous utilisrez le module `matplotlib.pyplot` pour faire ce graphique.

6.2.15 Question co-occurrence des domaines (4)

Parfois, des protéines contiennent plusieurs fois le même domaine. C'est par exemple le cas de la protéine `RAD53` de la levure, qui contient deux domaines `FHA` et un domaine `Kinase`. Leur agencement dans la séquence est `(FHA, Kinase, FHA)`. On notera, mais c'est hors sujet dans notre projet, que l'orthologue de `RAD53` chez l'homme, la protéine `CHK2`, ne contient qu'un seul domaine `FHA` associé au domaine `Kinase`...

Vous noterez que dans la fonction `generate_cooccurrence_graph_np`, si une protéine contient les domaines (a, b, a), on ne compte qu'une seule co-occurrence de a et b. En effet, dans le fonction `generate_cooccurrence_graph_np`, on s'intéresse aux nombre de protéines (d'où le np) qui contiennent les deux domaines.

Écrivez une nouvelle fonction `generate_cooccurrence_graph_n` qui crée une instance et renvoie un nouvel objet `DomainGraph`, dont les sommets sont les domaines. Il doit exister une interaction entre les sommets x et y uniquement lorsque ces deux domaines sont co-occurrents dans au moins n fois dans les protéines du graphe (donc dans l'exemple de `RAD53`, il y aurait 2 associations entre `FHA` et `Kinases`).

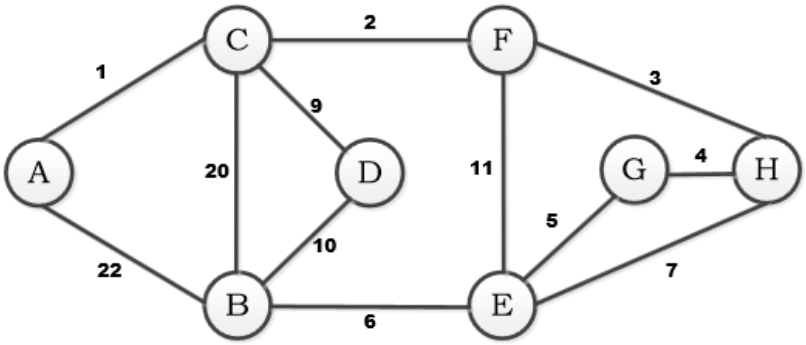
Attention, cette question n'est pas triviale. Il faut notamment vous poser la question de ce que vous faites si il y a une protéine (a, a, b, b), ou avec (a, a, a, a).

6.2.16 Question co-occurrence des domaines (5)

Vous avez développé des fonctions permettant de calculer la co-occurrence des domaines, mais cette information n'est pas stockée dans `DomainGraph`. En effet, pour l'instant, notre structure `DomainGraph` nous permet simplement de savoir s'il y a plus de n

co-occurrences des domaines, mais on ne connait pas ce nombre. C'est dommage...

Transformez votre classe `DomainGraph` en une classe permettant de stocker un graphe pondéré. On appelle graphe pondéré un graphe avec des sommets et des arêtes, mais cette fois-ci les arêtes ont chacune un poids (strictement positif).



Enfin, dans la classe `Interactome`, créez une fonction `weighted_cooccurrence_graph` qui crée une instance et renvoie un nouvel objet `DomainGraph`, maintenant pondéré. Les sommets sont les domaines. Il doit exister une interaction entre les sommets x et y uniquement lorsque ces deux domaines sont co-occurrents. Le poids de l'interaction entre x et y doit être égal au nombre de co-occurrences de ces deux domaines dans les protéines du graphe.

6.2.17 Export et visualisation

Exportez votre graphe des domaines. Le format utilisé sera semblable au format d'export des interactomes. Téléchargez le logiciel `Cytoscape` [2] et faites une jolie visualisation graphique de votre graphe des domaines.

6.2.18 Graphe seuil

Dans votre graphe, combien d'associations de domaines sont soutenues par un poids inférieur à 10 ? Ajoutez une fonction `graph_threshold(k)` qui élimine toutes les arêtes de poids inférieur à k.

Appliquez votre fonction pour obtenir un nouveau graphe des domaines avec un seuil à 10. Exportez votre nouveau graphe des domaines. Le format utilisé sera semblable au format d'export des interactomes. Avec `Cytoscape`, faites une jolie visualisation graphique de votre graphe des domaines.

2. http ://manual.cytoscape.org/en/stable/

6.3 Les domaines sont-ils distribués aléatoirement dans les protéines ?

6.3.1 Probabilité de co-occurrence de deux domaines (dans une protéine qui contient deux domaines)

À partir des différents paramètres calculés précédemment, quelle est la formule donnant la probabilité qu'une protéine contenant exactement deux domaines contienne à la fois $domA$ et $domB$, en faisant l'hypothèse d'indépendance ?

6.3.2 Co-occurrences de domaines atypiques (dans une protéine qui contient deux domaines)

Quel test statistique utiliseriez-vous pour comparer les fréquences observées de co-occurrences de domaines et les fréquences théoriques ? Posez H_0 et H_1.

Parmi les protéines de votre interactome contenant exactement deux domaines, quelles sont les paires de domaines dont la fréquence de co-occurrence diffère de la fréquence théorique ?

6.3.3 Probabilité de co-occurrence de deux domaines (dans une protéine qui contient plus de deux domaines)

Généralisez la formule de la question 6.3.1 pour donner la probabilité qu'une protéine contenant exactement trois domaines contienne à la fois $domA$ et $domB$, en faisant l'hypothèse d'indépendance ?

Essayez ensuite de généraliser pour une protéine contenant exactement n domaines ($n \geq 2$).

Quelles sont les paires de domaines dont la fréquence de co-occurrence diffère de la fréquence théorique ?