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 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. 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/`