Création d'une interface graphique capable d'analyser des séquences protéiques

Ammar Fatma & Laghrissi Hiba & Mariault Lee

Projet de programmation en PYTHON

Sciences de la vie parcours Biologie-informatique-mathématiques
Université de Nice côté d'Azur
Nice,France
02/05

Contents

1 Introduction

La bio-informatique est la mise en oeuvre de méthodes, de concepts ou d'algorithmes éprouvés pour résoudre des problèmes posés par la biologie. Elle nous permet la comparaison de séquences, l'accélération d'acquisition des informations sur les différentes séquences et par conséquent l'accélération de la production de la connaissance.

Notre projet consiste justement à construire une interface graphique utilisateur (GUI) dans le but de répondre à des besoins biologiques de la façon la plus efficace possible.

Notre programme repose sur le langage de programmation multiplateformes PYTHON. Ce dernier dispose de plusieurs outils de développement d'interface graphique comme **Tkinter**, **WxPython** et **JPython**. Notre projet tourne autour de tkinter qui est considéré comme une librairie standard de Python et son utilisation simple rend facile la création d'interfaces graphiques.
Les algorithmes d'alignement de séquences que nous utilisons reposent sur le concept de la programmation dynamique, qui contrairement aux algorithmes heuristiques, donnent le meilleur résultat possible. Les deux algorithmes que nous allons voir sont ceux de Needlman-Wunch pour l'alignement global et Smith-Waterman pour l'alignement local.

Par conséquent, nous nous retrouvons avec un défi à relever, celui de faire un script qui est capable, à la fois, de gérer la complexité des informations qu'il reçoit en entrée et d'avoir une sortie d'information efficace à la fois sur le plan fonctionnel et ergonomique.

2 Expression du besoin

Nous devions mettre en place une interface graphique capable de répondre aux besoins d'un utilisateur (étudiant, technicien, ingénieur ou chercheur) dans sa recherche d'informations sur une séquence protéique donnée. Pour cela, le programme utilise soit internet pour récupérer les informations sur la protéine du site UNIPROT soit un fichier multifasta importé.

L'outil sera donc capable de proposer les services suivants:

- Récupérer la séquence protéique de la séquence (online ou offline)
- Analyser la séquence (Taille, composition en acides aminées, analyse du profil d'hydrophobicité)
- Alignement global/local 2 à 2 en utilisant la matrice BLOSUM62
- Analyse des fiches UniprotKB et en soutirer les informations qui intéressent l'utilisateur avec une possibilité de trier et filtrer les informations.

3 Fonctionnalités

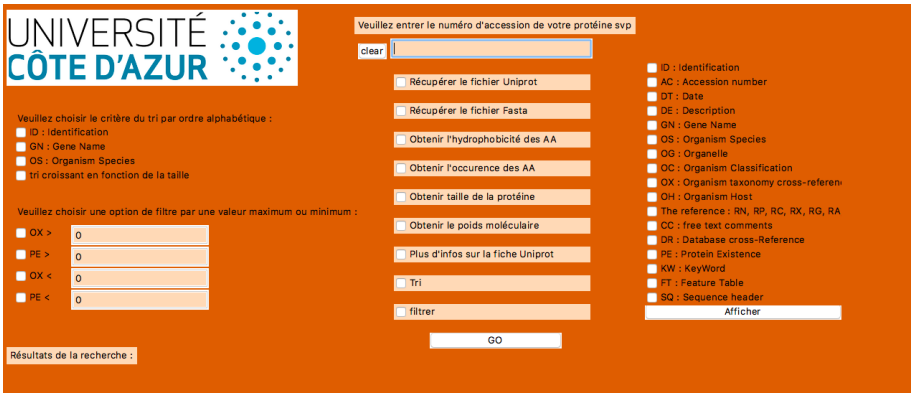

Figure 1: Aperçu de l'interface graphique

3.1 Récupération de la fiche UNIPROT/FASTA à partir d'internet

Les librairies utilisées pour cette tâche sont **Urllib** et **requests** qui utilisent toutes les deux HTTP/1.1. La raison pour laquelle nous avons choisi les deux modules est à cause d'une erreur dans la partie des classes (pour récupérer les informations d'Uniprot) qu'on n'a pas pu résoudre avec requests uniquement. D'où le fait qu'on a aussi utilisé Urllib.

Pour chaque fonction qui fait appel à Internet, nous avons mis en place 4 gestions d'erreur:

- Si le site renvoie un code HTTP entre 200 et 300 non inclus, cela veut dire que tout va bien et que le programme peut continuer
- Si le site renvoie un code HTTP entre 300 et 400 non inclus, cela veut dire que le problème est au niveau de la redirection.
- Si le site renvoie un code HTTP entre 400 et 500, cela veut dire que l'erreur vient de l'utilisateur, par exemple un numéro d'accession erroné.
- Si le site renvoie un code HTTP à partir de 500, alors le problème viendrait forcément au niveau du serveur.
- Enfin, nous avons aussi anticipé l'absence de connexion internet avec l'exception OSError.

4 Analyse de la séquence protéique

4.1 Récupération de la séquence

La récupération de la séquence protéique s'est faite grâce à la fonction **readFasta(file)**. Cette dernière est la fonction la plus importante de notre script, car elle récupère la fiche Fasta de la protéine dont les liens sont tous sous forme : **https://www.uniprot.org/uniprot/***.fasta**. Ensuite, elle récupère la séquence et le header pour que les autres fonctions puissent les utiliser.

ReadFasta(file) peut être exécutée de deux façons différentes :
Si un code d'accession est directement saisi dans le champ destiné à cet effet, alors la variable booléenne globale **offline_status** qui fait office de 'tag', reste False. Ceci amène le programme à chercher les informations dont il a besoin d'internet directement. Si l'utilisateur décide d'importer un fichier multifasta, la

fonction new_file() s'exécute et le programme va lui donner en sortie une liste de codes d'accessions trouvés. L'exécution de newfile() a aussi un autre rôle capital : elle change la variable offline_Statuts de False à True. Par conséquent, quand la fonction readFasta(file) est exécutée, elle va chercher les informations localement et n'ira sur internet que si l'utilisateur a besoin de sa fiche Uniprot.

4.2 Taille de la séquence

La taille de la séquence protéique est récupérée grâce à la fonction Tk_taille(). Cette dernière utilise la séquence que la fonction readFasta(file) a récupéré et calcule simplement le nombre de caractères qui la compose.

Nous y avons inclus une gestion d'erreur dans le cas où la séquence contiendrait des lettres qui ne représentent aucun acide aminée. Dans ce cas, le programme va informer l'utilisateur de leurs présences mais va les compter quand même, parce que ces caractères peuvent y être volontairement (un "X" par exemple). Ainsi, libre à l'utilisateur de modifier la séquence.

4.3 Composition en acides aminées:

La composition en acides aminées est donnée grâce à la fonction occurence_aa. Elle renvoie un dictionnaire dont les clés sont les acides aminées composant la séquence.
La fonction bouton_occ permet de générer deux boutons :

- Un bouton 'SAVE' qui appelle la fonction save_occ. Cette dernière, à son tour, appelle la fonction occurence_aa puis enregistre le résultat dans le répertoire courant.

- Un bouton "graphique": si l'utilisateur clique dessus, elle appelle la fonction graph_occ qui nous donne le graphique généré par Matplotlib. Nous avons choisi ce module parce que, en plus de représenter l'occurrence, il dispose d'un bouton sous forme de disquette qui permet à l'utilisateur d'enregistrer le graphique s'il le souhaite. Ainsi, cela nous permet à gagner quelques lignes de code dans le cas où nous devions demander à l'utilisateur avec une fenêtre tkinter, etc...

4.4 Le poids moléculaire:

Le poids moléculaire est calculé grâce à la fonction PM_peptide(). Cette dernière utilise la séquence récupérée par la fonction readFasta() et un dictionnaire contenant le poids moléculaire de chaque acide aminée. Dans le cas où il y aurait des caractères qui ne sont pas des acides aminées, ils ne seront pas comptés car ils n'appartiendront pas au dictionnaire contenant les valeurs du poids moléculaire mais l'utilisateur sera informé de leur présence.

4.5 Hydrophobicité :

L'échelle d'hydrophobicité de Kyte and Doolittle, J.Mol.Biol.157 (1982) est une des échelles utilisées pour faire des calculs d'hydrophobicité sur des protéines. En effet, on a un dictionnaire contenant la valeur correspondante de chaque acide aminé qui est repris par la fonction **hydrophobicite()**, celle-ci utilise la composition de la séquence prise à l'aide de readFasta() et prend une fenêtre de lecture glissante de 9 acides aminés qui glisse de 1 en 1 pour passer en revue toute la séquence afin de calculer une moyenne d'hydrophobicité. On peut ensuite afficher le graphique en appuyant sur le bouton correspondant après avoir enregistré l'hydrophobicité dans un fichier texte.
Pour l'affichage du graphique de ce dernier on a utilisé MatPlotlib qui ouvre une fenêtre à part et permet aussi d'enregistrer directement le graphique dans l'ordinateur.

Pour anticiper la présence de caractères qui ne sont pas des acides aminées, nous avons utilisé la même logique qu'avec le poids moléculaire.

5 Alignement des séquences

5.1 Alignement 2 à 2

L'alignement entre deux séquences similaires permet d'observer leur degré d'apparenté et de poser une hypothèse d'homologie si le pourcentage d'identité le permet. Ceci nous donne l'opportunité de tracer l'histoire de séquences inconnues. L'alignement optimal est calculé en essayant de faire coïncider les deux séquences, par l'insertion/délétion d'acides aminés ou d'ajout de gaps. Dans notre cas nous avons été amenés à travailler avec la matrice BLOSUM62. Cette dernière est utilisée avec les séquences qui ont des pourcentages d'identité supérieurs à 62%.

Une matrice de substitution associe un score à chaque paire de résidus. Ce score indique la vraisemblance qu'a cette paire de résidus de se retrouver dans un alignement de séquences homologues. Lors d'un alignement, le score est calculé en cumulant les coûts de chacune des substitutions, le nombre de gaps et leur longueur.

Dans ce projet, deux types d'alignement utilisant la programmation dynamique nous intéressent :

- le premier est l'alignement global 2 à 2 issu de l'algorithme de Needleman and Wunsch, et qui recouvre les séquences alignées sur l'ensemble de leur longueur. Dans ce type d'alignement, si les longueurs des séquences sont différentes, alors des insertions sont faites dans la séquence la plus petite pour arriver à aligner les deux séquences d'une extrémité à l'autre, ce qui peut induire des pénalités très importantes et réduire le score.
- Le second est l'alignement local issu de l'algorithme de Smith and Waterman, conçu pour rechercher dans la première séquence des régions semblables à la deuxième séquence (ou à des parties de la deuxième séquence).

5.2 Programmation dynamique

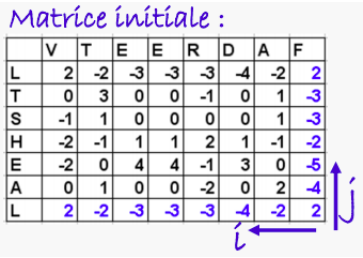

Figure 2: Matrice de comparaison (m,n)

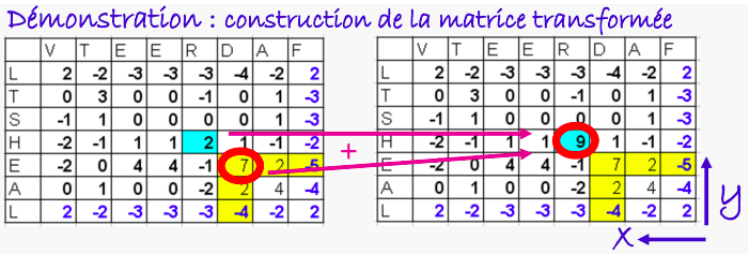

Figure 3: Transformation de la matrice initiale

La programmation dynamique est une méthode de résolution de problèmes qui passe par la résolution des sous-problèmes. De façon plus spécifique, le meilleur score de chaque paire de résidus est calculé pour ensuite obtenir un score qui est l'addition de tous ces scores optimaux.

Cette méthode consiste à construire une matrice de comparaison (m,n) entre 2 séquences qu'on transforme par addition des scores (initilisation de (m,o) et (o,n) puis addition des scores), ceci selon l'équation suivante:

$$S(i,j) = Se(i,j) + max(S(x,y)) \quad \text{avec} < x \quad \text{et} \quad y = j+1 \quad \text{ou} \quad x = i+1 \text{et} j < y \tag{1}$$

s(i,j) étant le score (i,j) dans la matrice transformée et Se(i,j) étant le score élémentaire de la case d'indice i et j de la matrice initiale [1].

	V	T	E	E	R	D	A	F
L	14	7	6	6	4	0	0	**2**
T	10	12	9	9	6	4	3	**-3**
S	8	10	9	9	7	4	3	**-3**
H	6	7	9	8	9	5	1	**-2**
E	2	4	8	8	8	7	2	**-5**
A	2	3	2	2	0	2	4	**-4**
L	**2**	**-2**	**-3**	**-3**	**-3**	**-4**	**-2**	**2**

Figure 4: Matrice de scores finaux calculées

	V	T	E	E	R	D	A	F
L	↘							
T		↘						
S		↓						
H			↘					
E				↘	→	→		
A							↘	
L								↘

Figure 5: Chemin optimal

Une fois les scores calculés dans chaque case, le programme cherche le chemin qui donne le meilleur score d'alignement. Au cours du chemin, le passage diagonal d'une case à l'autre représente une **substitution** (valeur calculée à partir de BLOSSUM62), le passage vertical ou horizontal d'une cellule à l'autre illustre un INDEL soit à la séquence i soit à la séquence j. La pénalité d'une ouverture de gap est de **-10.5** tandis que celle de son extension est de **-0.5** car il est moins probable d'avoir une extension que des ouvertures à plusieurs endroits.

Figure 6: substitution vs INDELS

Dans le cas, d'un alignement local, on reste sur le même principe de calcul. Cependant, n'importe quel point dans la matrice de comparaison peut être un point de départ pour le calcul des scores. De plus, l'alignement s'arrête quand le score devient négatif et l'alignement peut s'initialiser et repartir d'un point à 0. L'avantage de cet alignement est qu'il peut révéler des similitudes que l'alignement global ne détecte pas à cause des pénalités sur toute la séquence.

Les implémentations des alignements 2 à 2 ont été trouvées sur **Github**. Des changements ont bien sûr été apportés pour que cela soit utilisable par tkinter.

Pour l'alignement global [2]: Nous avons rendu les variables E, F et G globales et modifié le format de sortie. Le fichier résultat affichera donc le score avec les séquences après l'alignement.

Pour l'alignement local [3]: Nous avons changé la manière dont le script s'exécutait. Au début, il fallait exécuter le fichier en rentrant dans le terminal la commande **[python3 hw1.py -i input.txt -s blosum62.txt]**, on devait donc mettre 2 arguments : input.txt était le nom du fichier contenant les 2 séquences à aligner et blosum62.txt celui de la matrice.
Cependant, on a changé le script de sorte à ce que le fichier BLOSUM62 soit directement importé à partir du répertoire courant et que le script prenne deux séquences à partir des champs de saisie dans l'interface graphique au lieu de les prendre d'un fichier importé.

6 Analyse des fiches UNIPROT

6.1 Récupération des informations

En vue de la quantité importante d'informations que nous pouvons avoir à partir d'une fiche UNIPROT, nous avons décidé d'utiliser les classes pour cette partie du programme.

Pour rappel : “La programmation orientée objet (=POO) permet de rédiger du code plus compact et mieux ré-utilisable. L'utilisation de classes évite l'utilisation de variables globales en créant ce qu'on appelle un espace de noms propres à chaque objet permettant d'y encapsuler des attributs et des méthodes. De plus, la POO amène de nouveaux concepts tels que le polymorphisme (capacité à redéfinir le comportement des opérateurs), ou bien encore l'héritage (capacité à définir une classe à partir d'une classe préexistante et d'y ajouter de nouvelles fonctionnalités)”[4].
Ainsi, nous avons choisi d'utiliser les classes afin de permettre une meilleure gestion des données et rendre le script plus lisible et facile à comprendre. Ceci a aussi permis d'économiser le nombre de variables utilisées dans le cas où on aurait utilisé des dictionnaires ou des listes.

6.2 Tri des séquences:

Le tri des séquences peut représenter plusieurs intérêts intrinsèques dans le cas où nous avons un fichier multifasta avec un nombre élevé de séquences et donc de descriptions (en particulier des numéros d'Accession UniprotKB). Afin d'améliorer la visibilité et optimiser les recherches, nous avons décidé de classer le fichier multifasta par ordre alphabétique en fonction du **numéro d'accession**, du **Gene Name(GN)**, et enfin de **l'Organism Species(OS)**. Pour cela nous avons utilisé un module appelé **"Biopython"** qui prend en entrée toutes les séquences du fichier multi-fasta.

6.3 Filtre des séquences:

Le filtre consiste à améliorer la recherche d'informations, en fonction de critères spécifiques. Pour effectuer le filtre d'ordre, nous avons utilisé le même module appelé **"biopython"** qui prend en entrée toutes les séquences du fichier multifasta. Les filtres les plus utilisés de ce module sont les filtres minimum et maximum qui vont donner en sortie, un autre fichier multifasta filtré en fonction de **l'Organism taxonomy cross-reference(OX)** ou **Protein Existence (PE)**.
Pour cela, l'utilisateur doit choisir une valeur maximum ou minimum de OX ou de PE. Ainsi, le programme se charge de lui créer un nouveau fichier multi-fasta en mettant uniquement les séquences qui sont inférieures ou supérieures aux valeurs choisies.

6.4 Génération d'un fichier de sortie sous forme tabulé:

Les formats CSV et TAB sont des formats de texte simples qui sont utilisés dans de nombreux contextes et surtout quand de grandes quantités de données doivent être fusionnées sans être directement connectées les unes aux autres. L'extension de ces types de fichiers sont .csv et .tab, et ils peuvent être utilisés par différents outils informatiques et bases de données, lorsque l'on souhaite déployer le contenu d'une base de données sur un tableau dans un format Excel ou Open Office. Le sigle CSV signifie Comma-Separated Values et TAB signifie tabulé. Ainsi, ces derniers nous permettent d'avoir des fichiers qui présentent les informations de manière plus simple et efficace.

Dans notre projet, nous avons utilisé un module appelé tabulate. Il fournit une seule fonction, **“tabulate”** qui prend une liste de listes comme premier élément qui est structuré en colonne. Les deuxièmes éléments sont disposés en ligne, ce qui génère un tableau en texte clair bien formaté. Par ailleurs, nous avons laissé à l'utilisateur la possibilité de choisir le format qu'il souhaite .tab ou .csv pour pouvoir afficher notre fichier résultat. En effet, pour faire ceci l'utilisateur doit cocher les cases des informations qu'il a besoin d'avoir sur son fichier tabulé, ceci correspond aux données retrouvées dans le fichier uniprot. Pour extraire les informations de chaque ligne nous avons utilisé les classes vues précédemment.

7 Conclusion

L'élaboration d'une interface graphique qui propose plusieurs fonctionnalités n'était pas une tâche facile au début. Dans le sens où nous avions tellement d'idées pour procéder mais pas de plan spécifique. Nous avons dû nous organiser, filtrer nos idées et nous mettre à la place de l'utilisateur pour pouvoir mettre en place une interface capable de gérer des informations complexes mais dont l'utilisation reste agréable, simple à comprendre.

Ainsi, nous avons été capables de proposer un programme qui permet d'interagir avec l'utilisateur et de lui laisser un maximum de liberté (choix du nom et du répertoire d'enregistrement, changement du mode de traitement offline/online, capacité à réinitialiser la page, etc.). Nous avons aussi pensé au côté ergonomique en soignant les positions des boutons et la couleur d'affichage, ainsi qu'en anticipant au maximum les erreurs possibles que peut rencontrer l'utilisateur.

Notre groupe s'est organisé pour que chaque personne fasse des fonctions séparément pour ensuite fusionner le tout dans un seul script. Cela nous a non seulement permis de gagner en efficacité, et a obligé chaque personne à comprendre l'intégralité du code afin qu'elle puisse faire fonctionner leur partie dans le script commun.
Bien sûr, nous avons rencontré plusieurs difficultés pendant l'élaboration de notre projet, parmi elles :

- **Le choix entre les fonctions grid/pack/place :** au début, nous utilisions pack() mais lorsque l'on a commencé à ajouté plusieurs widgets, la fonction n'était plus suffisamment précise. Pour grid(), si 2 labels de tailles différentes se trouvaient dans la même ligne, cela perturbait l'organisation de toute la fenêtre. Place() nous a évité ces deux problèmes et rassemble à la fois précision et flexibilité.

- **L'exécution répétée des fonctions:** effectivement, plus l'utilisateur appuyait sur le bouton "GO" et plus les résultats s'accumulaient sans être réinitialisés. Ceci était dû au fait que les fonctions se répétaient alors que les variables avaient toujours les résultats de la recherche précédente. La solution que l'on a trouvé était les variables booléennes **tag_1time**. Ainsi, une fonction ne s'exécute qu'une seule fois dans une même session.

- **Les résultats du fichier tabulé :** En effet, nous ne réussissions pas à faire un tableau et le délimiter avec le '', donc les résultats s'affichent en désordre. Pour contrer ce problème, nous avons utilisé un module appelé 'tabulate' pour organiser au mieux possible le fichier de sortie.

- **la fonction readFasta() et le multifasta :** c'est le problème qui nous avait pris le plus de temps, car readFasta() est la fonction clé de notre script étant donné que c'est elle qui récupère la séquence et le titre des protéines.
  Nous avons fini par régler le problème grâce à l'utilisation du tag offline_status comme expliqué ci-dessus.

- **Le bouton RESET** : Pour améliorer notre code sur le plan ergonomique, nous avons jugé utile de donner à l'utilisateur la possibilité de réinitialiser la fenêtre sans la quitter pour faire une nouvelle recherche avec une nouvelle protéine. Au début, nous avions bloqué sur la taille de la protéine car c'était la seule fonction qui ne marchait plus après qu'on ait remis tout à 0 (les tags, les variables, etc.). La solution que l'on a trouvée était la fermeture puis réouverture automatique de la fenêtre.

Nous avons aussi pensé à plusieurs idées qu'on aurait voulu ajouter au programme mais malheureusement, nous n'avons pas eu le temps de les ajouter. Comme par exemple :

- L'envoi par mail des résultats de la recherche : Nous sommes arrivés à envoyer un mail automatique à l'utilisateur mais les résultats étaient imprimés. Nous n'avons pas réussi à joindre les fichiers plutôt que les écrire.

- Améliorer les résultats de l'alignement global pour avoir la matrice de score comme avec l'alignement local.

References

[1] Françoise Giroud. *$TD3_Bio24a_AlignementSequencesScore$.*. Laboratoire TIMC-IMAG/Equipe DyC-TiM. URL: http://membres-timc.imag.fr/Francoise.Giroud/Bio24a/pdf/TD3_Bio24a_AlignementSequencesScore.pdf.

[2] raha1. *BLOSUM62*. Version 1.0. June 2018. URL: https://github.com/raha1/BLOSUM62.

[3] slyyy312. *$local_alignment$*. Version 1.0. Feb. 2019. URL: https://github.com/slyyy312/local_alignment.

[4] Patrick Fuchs Pierre Poulain. *Avoir la classe avec les objets*. Cours de Python. URL: https://python.sdv.univ-paris-diderot.fr/19_avoir_la_classe_avec_les_objets/.