Gabriel Cretin
M2 BIB

Assignation et détection des parties transmembranaires d'une protéine

Introduction

Les protéines transmembranaires (TMPs) sont situées au sein des bicouches lipidiques des membranes plasmiques ou de membranes des organelles (Langó et al. 2017). Elles jouent un rôle crucial dans le transport d'ions, la signalisation ou la communication intracellulaire, ainsi que le maintien d'un équilibre entre le milieu intracellulaire et extracellulaire. Cela en fait des cibles privilégiées pour l'industrie pharmaceutique par exemple, en démontre les 55% des médicaments approuvés par la FDA ("Food and Drug Administration") ciblant des TMPs (Langó et al. 2017; Uhlén et al. 2015). Cependant, malgré les progrès continues et exponentiels des techniques expérimentales ces dernières années en détermination des structures 3D des protéines, il reste excessivement difficile d'obtenir une résolution assez élevée des structures de TMPs. Cela explique la faible proportion de structures de TMPs disponible dans les bases de données (< 3%).

Pour la faible proportion des structures disponibles identifiées comme transmembranaires, pour la plupart, la localisation précise de la bicouche n'est pas précisée dû aux techniques expérimentales qui cristallisent les protéines sans leurs bicouches. C'est pourquoi des techniques algorithmiques ont été développées pour tenter de déterminer la position la plus probable de la membrane et ainsi de pouvoir différencier les protéines globulaires des protéines transmembranaires.

C'est ce dont il est question dans ce projet basé sur un article (Tusnády, Dosztányi, and Simon 2004) qui a implémenté une nouvelle approche géométrique afin de discriminer les TMPs des protéines globulaires à l'aide d'informations uniquement structurales et en localisant la position la plus probable de la bicouche. Nous allons chercher à réimplémenter leur méthode en ne s'attachant qu'à la partie géométrique et basée sur l'hydrophobicité. Nous ne nous intéresserons pas au facteur structural de la fonction objective de l'article.

Méthodes

L'algorithme qui a été implémenté récupère en entrée un fichier PDB (.pdb) contenant la structure d'une protéine. Il va ensuite faire tourner le programme Naccess dessus pour calculer la surface accessible au solvant. La sortie de Naccess donne l'accessibilité relative au solvant sur tous les atomes. On récupère alors uniquement les résidus accessibles dont la valeur est supérieure à 30 (valeur arbitraire par rapport aux valeurs de la littérature mais assez discriminante). Cela permet par la suite d'extraire dans le fichier PDB les coordonnées des atomes C_{alphas} de ces résidus *uniquement*, ce qui optimise le coût en mémoire par rapport à une mise en mémoire de tous les résidus. Pour optimiser les I/O de fichier et ne lire qu'une seule fois le fichier PDB, le centre de masse est calculé simultanément en faisant attention à ne pas prendre en compte les "HETATOM" qui apparaissent dans les fichiers de structures de la base de donnée OPM ("Orientations of

Proteins in Membranes") qui incluent des "HETATOM" de type "DUM" pour représenter les membranes. Toutes les informations utiles pour chaque résidu sont répertoriées dans une structure de dictionnaire car elles seront utilisées dans des étapes ultérieures.

Le programme s'attèle ensuite au coeur du problème, qui est de pouvoir quadriller la protéine avec des droites qui passent par le centre de masse. Pour cela, un plusieurs points (dont le nombre est défini par l'utilisateur, ce qui détermine la finesse du quadrillage) sont générés de manière uniforme (pas dans le sens de la loi uniforme, mais équidistants les uns aux autres) sur la surface d'une hémisphère englobant la protéine comme le montre la figure 1.

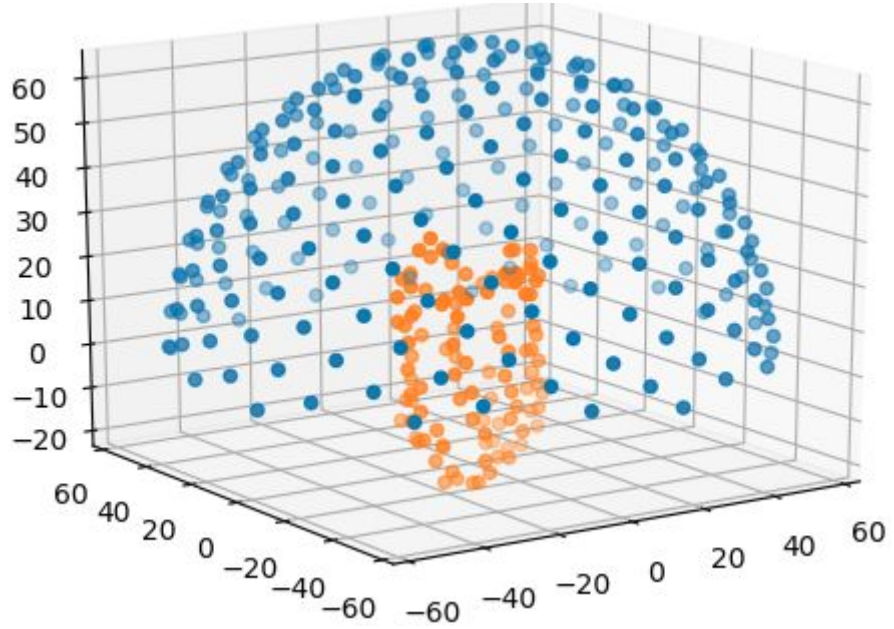

Figure 1. 250 points (bleu) répartis sur une hémisphère de rayon 70 qui englobe les C_{alphas} dont la valeur relative de surface accessible au solvant est supérieure à 30, de la protéine 1uaz (points oranges).

Pour simplifier les calculs de vecteurs et les produits scalaires, toutes les coordonnées des C_{alphas} sont transformées pour qu'elles soient dans un système centré sur l'origine (0, 0, 0).
Vient ensuite la partie parallélisée du programme. Les calculs qui suivent se font pour chaque droite passant par le centre de masse et un point de la sphère. Pour chaque point de la sphère, et donc chaque droite, l'algorithme va créer un plan assez loin de tous les résidus (le facteur multiplicateur est de 500 angströms, assurant d'être assez loin pour n'importe quelle protéine).
Une distance sera calculée entre le plan et tous les C_{alphas} afin de déterminer lequel est le plus proche du plan. Le plan est alors ramené au résidu le plus proche. L'étape d'après consiste à quadriller la droite par des tranches d'une épaisseur modifiable par l'utilisateur (par défaut 15 A comme dans l'article) qui va glisser le long de la droite avec un certain pas (aussi déterminable par l'utilisateur, par défaut 5 A). Une hydrophobicité relative est calculée pour chaque tranche sur les résidus accessibles au solvant. Et un facteur est calculé pour l'hydrophobicité globale de la droite qui a été calculée (dans l'article il s'agit de la *Q_value*). Il ne s'agit pas d'une moyenne mais d'un facteur tel que:

$$hydrophobicité\ de\ la\ droite\ =\ (\sum hydrophobicité\ des\ tranches)\ *\ nombre\ de\ tranches\ **\ 2$$

Pour que la valeur soit assez discriminante malgré l'absence du facteur structurale dans notre programme, il faut multiplier par le nombre de tranches au carré.
Toutes les valeurs d'hydrophobicité sont gardées en mémoire pour pouvoir retrouver par la suite la droite ayant la meilleure hydrophobicité ainsi que les tranches qui maximisent l'hydrophobicité associée.
Afin de déterminer la normale aux membranes les plus probables, il faut trouver la droite dont la valeur d'hydrophobicité est la plus élevée, et à l'intérieur de cette droite, retrouver la tranche, ou l'espacement entre lequel l'hydrophobicité est maximale, c'est-à-dire la "sous-tranche" qui maximise la somme d'hydrophobicité. Pour ce problème, l'algorithme de Kadane a été implémenté. Il est connu pour résoudre ce problème de somme maximale de sous-tableau en temps O(n) et mémoire O(1). Il retourne les indices min a et max b des tranches ainsi que la somme maximale c possible tel que $\sum tranches[a\ :b] = c\ avec\ c\ maximal$.

Une fois que sont déterminées la meilleure droite ainsi que les tranches, il suffit d'écrire le résultat sous formes d'une droite et de deux plans pour pouvoir les visualiser avec la protéine elle-même. C'est pourquoi un nouveau fichier PDB est généré auquel ont été ajoutées les coordonnées des points des plans qui vont représenter les deux membranes. De même qu'un fichier PyMol (.pml) est généré pour dessiner la "meilleure" droite, c'est-à-dire celle qui est normale aux membranes.

Résultats et Discussions

Le programme s'exécute la plupart du temps en moins d'une seconde, cela oscille entre 0.7 et 2 secondes, pour 1000 points générés sur la demi sphère, ce qui représente donc 1000 droites qui quadrillent la protéine. Sans parallélisation, le programme met environ 4 fois plus de temps à tourner, ce qui reste raisonnable. Le test a été tourné pour la protéine transmembranaire 1uaz (structure de la archaerhodopsin-1), et la protéine globulaire 1uw3 (prion du mouton). Le programme retrouve bien la droite normale aux membranes pour la protéine transmembranaire, mais pas pour la protéine globulaire. L'algorithme n'implémente pas encore la possibilité d'arrêter le programme si un seuil d'hydrophobicité n'a pas été dépassé.

De plus il y a un problèmes de différenciation entre les protéines globulaires et transmembranaires car l'implémentation actuelle ne prend en charge que l'aspect géométrique de la fonction discriminante, et non pas l'aspect structural. L'aspect structural pourrait permettre d'apporter des informations discriminantes entre une protéine traversant une membrane et ayant alors une surface accessible au solvant moindre qu'une protéine globulaire, généralement plus hydrophile à la surface.

Egalement, l'algorithme de Kadane ne peut pas être assez efficace pour retrouver l'espacement entre les deux membranes avec des valeurs d'hydrophobicité relative à 1 parfois. Cette valeur s'explique par des tranches dans lesquelles il y a seul 1 résidu, qui est hydrophobe. Il faudrait donc pondérer les hydrophobicités des tranches ou de la droite par le nombre de tranches de la droite qui en possède le plus par exemple.

Néanmoins, le programme ne parvient pas à dessiner le plan des membranes. En effet, les coordonnées des plans ne sont pas perpendiculaires à la droite normale aux membranes.

Conclusion

Le programme tourne de manière efficace et optimisée, et parvient à trouver la droite normale aux membranes dans quelques cas, mais il est encore trop délicat de pouvoir déterminer à coup sûr la droite et les membranes sans l'apport de l'aspect structural. De plus, par manque de temps, les membranes ne sont pas correctement représentées sur PyMol.

Références

Tusnády GE, Dosztányi Z, Simon I. Transmembrane proteins in the Protein Data Bank: identification and classification. Bioinformatics. 2004 Nov 22;20(17):2964-72. Epub 2004 Jun 4. PubMed PMID: 15180935.

Langó, Tamás, Gergely Róna, Éva Hunyadi-Gulyás, Lilla Turiák, Julia Varga, László Dobson, György Várady, et al. 2017. "Identification of Extracellular Segments by Mass Spectrometry Improves Topology Prediction of Transmembrane Proteins." *Scientific Reports* 7 (February): 42610.

Tusnády, Gábor E., Zsuzsanna Dosztányi, and István Simon. 2004. "Transmembrane Proteins in the Protein Data Bank: Identification and Classification." *Bioinformatics* 20 (17): 2964–72.

Uhlén, Mathias, Linn Fagerberg, Björn M. Hallström, Cecilia Lindskog, Per Oksvold, Adil Mardinoglu, Åsa Sivertsson, et al. 2015. "Proteomics. Tissue-Based Map of the Human Proteome." *Science* 347 (6220): 1260419.

Gabriel Cretin
M2 BIB

ANNEXES

Plusieurs choix d'implémentation ont été faits. Etant donné que le programme se base sur une seule partie de la fonction principale qui est décrite dans l'article, à savoir la partie géométrique qui utilise l'hydrophobicité (zones accessibles au solvant), il est prévisible qu'il n'ai pas le même taux de réussite pour trouver les emplacements de la membrane ni la même capacité discriminante entre protéines transmembranaires et globulaires. C'est pourquoi plusieurs paramètres ont été rendu modifiables lors du lancement du programme en ligne de commande, dans l'optique de pouvoir jouer dessus selon la protéine que l'on veut étudier. En effet des tests ont montrés que le programme parvenait à déterminer la norme à la membrane pour une protéine avec des valeurs de paramètres donnés, mais n'y parvenait plus pour une autre. C'est pourquoi des paramètres jouant sur la résolution et la précision du quadrillage de la protéine sont modifiables. Ces paramètres seront décrits ultérieurement.

Le programme est construit de façon à privilégier la rapidité d'exécution en premier, en utilisant le module Numpy de Python notamment, qui permet d'utiliser des tableaux à N-dimensions et des produits matriciels, et qui est très rapide. Quasiment tous les calculs scientifiques sont réalisé en utilisant Numpy. La boucle principale du programme est également parallélisée sur tous les CPUs disponibles, ce qui accélère encore plus les calculs.

La mémoire que prend le programme n'est pas négligée pour autant car la parallélisation se fait sur la base de la fonction `itertools.imap()` et non `map()`. La différence tient sur le fait que `map()` utilise l'itérable de la boucle parallélisée en le convertissant en une liste, le divise en fragments ("chunks"), puis les envoie aux processus du `Pool`. Casser l'itérable en fragments est plus performant que de passer chaque itératif un par un aux processus entre eux, plus particulièrement lorsque l'itérable est grand. Cependant, transformer l'itérable en liste demande de garder en mémoire toute la liste, ce qui a un coup en mémoire. Tandis que la fonction `imap()` ne fragmente pas ni ne transforme l'itérable en liste, il passe chaque élément de l'itérable un par un à un processus, ce qui sauvegarde beaucoup de mémoire, mais qui est moins performant pour un grand itérable. De plus, de nombreux calculs sont effectués sur les coordonnées des C_{alpha} avec des valeurs qui doivent être ré-utilisées ultérieurement. C'est pourquoi des structures imbriquées de type dictionnaire ont été générées, ce qui génère nécessairement un certain coût en mémoire mais les dictionnaires font partie des structures les plus optimisées en python grâce notamment aux *hash*, c'est pourquoi le temps d'accès est assez réduit. Le coût en mémoire est moins préoccupant au vue des capacités des machines d'aujourd'hui, car il est question de l'ordre d'une centaine de mégaoctets seulement, avec des temps de calculs d'une seule seconde en moyenne ($0\ s < T_{calcul} < 2\ s$), voire moins pour certaines protéines.

Le programme utilise la programmation orientée objet autant que faire se peut. La structure du projet et du programme ne se prêtant pas idéalement à l'orienté objet, seule une classe Vecteur a été créée, étant définie comme un *Numpy array* refermant trois coordonnées cartésiennes. Un objet est construit tel que `Vector(x, y, z)` génère un tableau de type `ndarray` contenant les trois coordonnées cartésiennes soit d'un point dans l'espace soit d'un vecteur, qui informatiquement sont représentés de façon identique. Cela nous permettra d'accéder à chaque coordonnée de façon plus lisible dans les autres modules, tout en pouvant utiliser la puissance des tableaux de type `ndarray`.

De grandes difficultés ont été rencontrées pour les calculs trigonométriques des coordonnées, par exemple des plans (qui n'a pas pu être aboutie). De manière générale, la partie complexe de ce projet était de générer des points dans l'espace pour visualiser correctement nos résultats. Par manque de temps, la fonction de visualisation n'est pas aboutie et le programme n'identifie pas toujours la meilleure droite.

Exemple d'exécution du programme sur la protéine 1uaz.

```
$ ./main.py data/1uaz_tm.pdb --points 1000 && pymol
src/pymol_visualize.pml
```

Sortie:

```
###################################
```

Best line/direction is between the center of mass and the following point:
Point of the sphere: [42.508, 30.603, 497.222]
Center of mass: [0.299, -0.394, -0.028]

Highest hydrophobicity factor: 39.7001

Program runtime: 0:00:01.227251

```
###################################
```

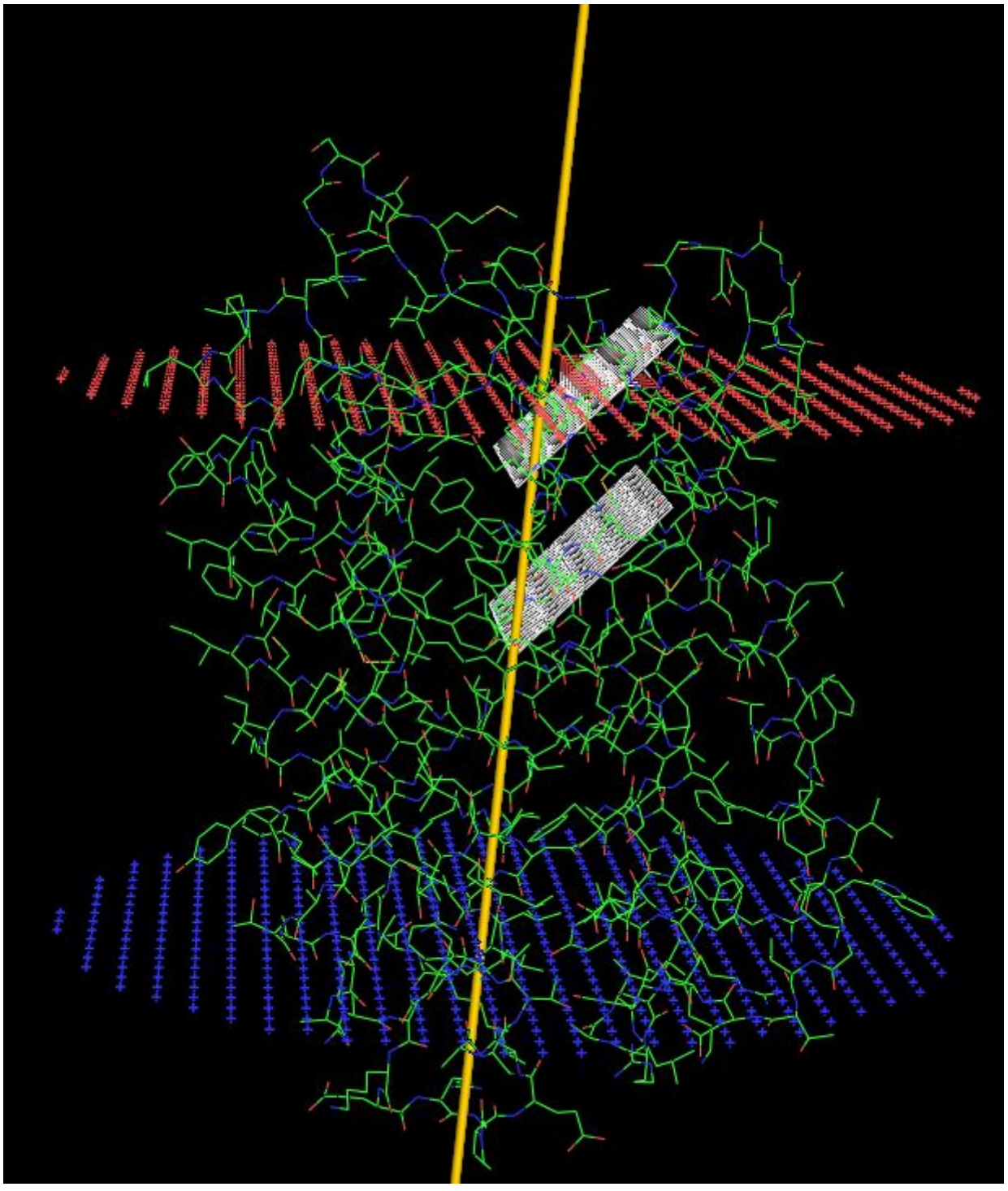

Figure 2. Figure tirée de PyMol. Protéine 1uaz (en sticks) traversée par la droite normale aux membranes la plus probable, et en blanc les deux plans théoriques de la membrane. Les plans ne sont pas perpendiculaires à la droite, il y a encore une erreur dans la fonction qui génère les coordonnées de ces plans.