Sorbonne Université , M2 PSF, 2024 dirk.stratmann@upmc.fr

TP protein-protein docking avec Rosetta

1 Introduction

Ce TP est la suite du TP protein-protein docking avec zdock et co., c'est donc un prérequis de l'avoir fait avant.

Pour utiliser Rosetta, on n'utilisera pas la machine virtuelle, comme Rosetta a été installé sur chaque machine de la salle de TP via l'image docker officiel de Rosetta. Docker est une sorte de machine virtuelle, mais qui est plus intégré avec le système hôte, on ne peut alors pas installer cet image docker sous Windows par exemple, mais uniquement sous Linux :

https://hub.docker.com/r/rosettacommons/rosetta

Cette image est actualisée très fréquemment et surtout ne demande que 2 Go au téléchargement, 4,5 Go installé, contrairement à la variante des binaries Ubuntu: 20 Go au téléchargement et 56 Go installé !

https://downloads.rosettacommons.org/downloads/academic/3.14/

Mais ce lien permet de télécharger le code source (5 Go) avec surtout le dossier des « demos » qui permettent d'apprendre les nombreux protocoles implémentés dans Rosetta. (ne pas faire en salle de TP svp!). Il y aussi une version pour Mac (14 Go à télécharger).

Pour installer Rosetta chez soi, il faut disposer d'un environnement Linux ou Mac OS X avec docker installé. Voir ici pour installer docker sous Ubuntu :

https://docs.docker.com/engine/install/ubuntu/

Il peut être utile d'ajouter l'utilisateur au groupe docker, afin d'éviter de devoir taper « sudo » à chaque commande docker (ce qui a déjà été fait dans les salles de TP, comme les utilisateurs n'y peuvent pas utiliser sudo):

https://docs.docker.com/engine/install/linux-postinstall/

Pour lancer une commande de Rosetta il faut utiliser le script fournit avec le TP : **runRosetta.sh**
en copiant ce script dans chaque dossier de travail où on souhaite lancer une commande de Rosetta. Ce script contient un appel à docker pour charger l'image de Rosetta tout en montant le dossier actuel sur le dossier /tmp de l'image docker et en spécifiant comme dossier de travail /tmp, i.e. le dossier actuel au final :

```
docker run -v .:/tmp -w /tmp rosettacommons/rosetta:serial $@
```

1.1 Premier test de Rosetta

1. Faites un test avec le l'exemple « testRosetta.tar » fournit (cet archive contient également le script runRosetta.sh) :
   tar xvf testRosetta.tar
   cd testRosetta
   ./runTest.sh
   Cet exemple calcule l'énergie de Rosetta pour le fichier PDB fournit en entrée (1qys.pdb ici) en appelant le programme « score_jd2 » de la suite de Rosetta. La ligne de commande qui a été

sauvegardée dans runTest.sh est :

```
./runRosetta.sh score_jd2 -in:file:s 1qys.pdb
```

Le fichier d'entrée est spécifié avec l'option « -in:file:s ».
Vous devriez avoir sur la fenêtre Terminal un retour en temps réel des étapes de calcul de Rosetta, terminant avec entre autres cette phrase :

```
protocols.jd2.JobDistributor: 1qys_0001 reported success in 0 seconds
```

De plus un nouveau fichier devrait être crée par Rosetta : « score.sc », qui contient les valeurs des composantes de l'énergie et de l'énergie totale du fichier PDB fournit en entrée.
Si cela n'a pas marché, vérifier que vous vous êtes loggé sur votre PC avec « etu_numéroEtudiant » et pas uniquement le numéro d'étudiant.

2 Protein-Protein docking protocol

2.1 Demo de Rosetta

La documentation de Rosetta est disponible online ici :

https://docs.rosettacommons.org

Le dossier « demos » du code source de Rosetta (voir plus haut) contient les données des tutoriels :

https://docs.rosettacommons.org/demos/latest/Home#tutorials

Ici on va étudier d'abord le tutoriel « protein-protein docking » :

https://docs.rosettacommons.org/demos/latest/tutorials/Protein-Protein-Docking/Protein-Protein-Docking

En parallèle on peut regarder le détail de cette « application » de Rosetta :

https://docs.rosettacommons.org/docs/latest/application_documentation/docking/docking-protocol

1. Faire les étapes du tutoriel « protein-protein docking » en remplaçant
   `$ROSETTA3/main/source/bin/docking_protocol.linuxgccrelease`
   par
   ./runRosetta.sh docking_protocol
   A la fin de la ligne de commande on peut rediriger la sortie qui s'affiche dans la fenêtre Terminal dans un fichier « log », comme ceci :
   ./runRosetta.sh docking_protocol @flag_local_docking > local.log
   Cela permet de vérifier le bon déroulement du protocole, surtout que Rosetta produit rapidement beaucoup de lignes de texte dans la fenêtre Terminal sinon.
   Les données du tutoriel peuvent être téléchargées depuis mon dropSU :
   https://dropsu.sorbonne-universite.fr/s/ry7dwNzC4kpk6cH
   dossier « demos », puis « Protein-Protein-Docking.tar.gz »
2. Inspecter avec pymol les fichiers PDB d'entrée (« input_files ») et de sortie (« output_files) (peut être lancé directement depuis un Terminal de la machine hôte, pas besoin de la machine

virtuelle). La commande **align** de pymol peut être utile ici pour superposer les structures d'entrée ou la structure de référence du complexe avec celles qui ont été générées par Rosetta.

3. Grâce à cette comparaison avec pymol, est-ce qu'il s'agit d'un exemple de bound-bound ou de unbound-unbound docking ?
4. Les options de chaque run de Rosetta sont assemblées dans des fichiers « flag ». Chercher dans la documentation de Rosetta ce que veulent dire les options qui ne sont pas détaillés dans le tutoriel et noter ici ce que vous avez trouvé (vous pouvez copier&coller les explications en anglais de la documentation directement en citant vos sources):
5. En étudiant la dernière partie du tutoriel « Analyzing Docked Structures », comment est-ce que les poses sont évaluées ici ? Copier ici les valeurs que vous avez obtenu. Combien d'étoiles est-ce que vous avez obtenu ?

2.2 Appliquer sur E9-IM9

On va maintenant appliquer le même protocole de « local docking » de Rosetta sur les meilleurs poses obtenues avec zdock lors du dernier TP.

1. Pourquoi est-ce que cela fait du sens ? Quelles sont les différences entre la méthode de docking de zdock et celle de Rosetta ?
2. Prendre la meilleure pose obtenue avec zdock sur E9-IM9 (dernier TP) et copier-coller la dans un nouveau dossier. Pour utiliser le protocole de « local docking » de Rosetta, il faut séparer les deux monomères de 10 Å environ. On peut utiliser pymol pour cela :
   a) Ouvrir la meilleure pose de zdock avec pymol.
   b) Clique-droit sur un des deux monomères. Un menu local s'ouvre, y choisir « chain » puis « drag ». Rester ensuite sur la touche « maj », puis appuyer sur la molette (3ème bouton de la souris) tout en bougeant la souris. Le monomère choisit devrait bouger sur un axe. Vous pouvez revenir en arrière avec Ctrl+Z.
      Appuyer sur « Done » quand vous avez séparé les deux monomères dans l'espace.
   c) Enregistrer ces nouvelles coordonnées des deux monomères sous format PDB (classique) via le menu pymol « File → Export Molecule »
   d) Reinitialiser pymol avec la commande pymol **reinitialize**
   e) Charger le fichier PDB généré à l'instant pour vérifier que tout va bien.
3. Copier le fichier flag_local_docking du tutoriel dans votre dossier actuel. Modifier le pour prendre en entrée votre fichier PDB généré dans la question précédente. Pour ce premier test, on ne donne pas la structure de référence (1EMV.pdb) à Rosetta, car cela demande un peu de travail d'adaptation du fichier PDB. Il suffit simplement de commenter avec un # la ligne de l'option en question dans le fichier flag_local_docking : # -in:file:native …
4. Faire un premier run en laissant l'option « partners » inchangé, i.e. « A_B ». Qu'est-ce que spécifie cette option ? Analyser avec pymol la pose obtenue. Renommer le dossier « output_files » en « output_files01 », puis créer un nouveau dossier « output_files ».
5. Faire un deuxième run en permutant l'ordre des chaînes dans « partners », i.e. « B_A ». Qu'est-ce qu'on observe ?
6. Pour pouvoir inclure le fichier PDB 1EMV en tant que structure de référence, il faut d'abord harmoniser le fichier 1EMV.pdb avec le fichier de la pose obtenue avec zdock. Avant cela il y a

une étape d'installation des « protein_tools » de Rosetta :

Créer d'abord un nouvel environnement « conda » pour installer python 2.7 :

```
conda init bash (redémarrer la fenêtre Terminal ensuite)
conda create -n py27 python=2.7
conda activate py27
conda install biopython
conda install matplotlib
```

Ensuite il faut télécharger le fichier « protein_tools.tar.gz » depuis le dropSU (voir au début de l'énoncé) dans le dossier « demos ». Décompresser cet archive, puis aller dans le dossier « protein_tools » (tout depuis le Terminal svp). Lancer l'installation en indiquant le dossier d'installation :

```
python setup.py install --install-scripts=/home/user/rosetta/protein_tools
```

Ensuite il faudra éditer le fichier PDB de la pose zdock en y enlevant les résidus supplémentaires en début et fin de chaîne, qui n'existent pas dans 1EMV.pdb. Utiliser un simple éditeur de texte pour cela (kate, gedit, etc). De plus il faudra avec l'éditeur de texte mettre les deux chaînes dans l'ordre : d'abord A, puis B.

En dernière étape on utilise l'outil « clean_pdb.py » de Rosetta

```
python  /home/user/rosetta/protein_tools/clean_pdb.py   complex.1.pdb  AB
python  /home/user/rosetta/protein_tools/clean_pdb.py   1EMV.pdb  AB
```

Utiliser les deux fichiers PDB ainsi générés pour un troisième run avec un fichier flag_local_docking actualisé. Dans le fichier de sortie « score_local_dock.sc » on devrait voir les mesures de CAPRI de la pose par rapport à la structure de référence.

3 Meiler-lab tutorials : Protein-Protein docking protocol

Le laboratoire de Jens Meiler organise tous les ans des workshops (en états-unis) autour de Rosetta, le matériel est disponible ici :

https://meilerlab.org/tutorials/

Celui de 2023 contient un tutoriel plus avancé sur le docking protéine-protéine :

https://meilerlab.org/rosetta-workshop-2023/

Ce tutoriel est une mise-à-jour du tutoriel « advanced » sur rosettacommons qui date de 2017 :

https://docs.rosettacommons.org/demos/latest/tutorials/advanced_protein-protein_docking/advanced_protein-protein_docking_tutorial

1. Suivre le tutoriel de 2023
2. Essayer un autre tutoriel des workshops proposés par le Meiler-lab.