MOL2, Covalence and co.

Simao CORTINHAL

2024-2025

—

CHPS1002

Dans le cadre du projet de CHPS1002, on a écrit des codes en langage Fortran 90, afin de reconstruire et remplir un fichier moléculaire mol2, afin d'ensuite exploiter cette représentation pour réaliser un amarrage moléculaire par algorithme génétique. Voici l'ordre dans lequel nous avons écrit les différents codes que nous allons voir dans ce court rapport

1. **Lecture et affichage des coordonnées atomiques**
   Développement de lecteur_mol2.f90, qui extrait d'un fichier mol2 la liste des atomes avec leur seul symbole chimique et leurs coordonnées 3D.
2. **Chargement des rayons de covalence**
   chargeur_covalence.f90 pour lire un tableau de rayons atomiques et interroger la valeur associée à un élément donné.
3. **Détection de la topologie**
   Implémentation de affiche_topologie.f90, qui avec les rayons de covalence et un critère de tolérance variable (delta croissant de 10 % à 35 %), propose pour chaque paire d'atomes un type de liaison (simple, double ou triple).
4. **Génération d'un mol2 complet**
   complete_mol2.f90, capable d'insérer automatiquement dans une section BOND les liaisons, et d'écrire un nouveau fichier mol2 sans écraser l'original.
5. **Optimisation et parallélisation**
   Parallélisation de l'étape de détection de topologie, et proposition d'une version OpenMP pour accélérer.
6. **Amarrage moléculaire par algorithme génétique**
   Algorithme génétique appliqué au docking ligand–site

Lecteur_mol2

lecteur_mol2.f90 se décompose en cinq étapes principales :

1. **Initialisation**
   - Définition du type dérivé info_atome (symbole chimique + coordonnées 3D).
   - Nom de fichier mol2 passé en argument avec get_command_argument.
2. **Ouverture et recherche des sections**
   - Ouverture du fichier en lecture avec contrôle d'erreur (iostat=statut_IO).
   - Parcours des lignes : détection de @<TRIPOS>MOLECULE pour lire le nombre d'atomes, puis arrêt à @<TRIPOS>ATOM.
3. **Allocation dynamique**
   - Après avoir obtenu nb_atomes, on alloue un tableau atomes(nb_atomes) de info_atome, ce qui garantit une empreinte mémoire adaptée.
4. **Lecture et parsing**
   - Boucle de 1 à nb_atomes :
     - Lecture d'une ligne, puis read(ligne,*) pour extraire l'identifiant, un nom temporaire, les coordonnées x,y,z, un champ type_atome_tmp ("C.3") et deux variables ignorées.
     - Suppression du suffixe (tout ce qui suit ".") dans type_atome_tmp pour conserver que le symbole atomique.
     - Remplissage de atomes(idx)%element et des trois champs de coordonnées.
5. **Affichage et nettoyage**
   - Fermeture du fichier, puis affichage du nombre total d'atomes et, pour chacun, son index, son symbole et ses coordonnées formatées.
   - Désallocation du tableau.

Gestion des erreurs : à chaque ouverture ou lecture on vérifie statut_IO et on interrompt le programme en cas de problème.

Commande :

./lecteur_mol2 <fichier mol2>

chargeur_covalence

chargeur_covalence.f90 doit lire un fichier de rayons de covalence, extraire pour chaque élément chimique ses valeurs de liaison simple, double et triple, et laisser l'utilisateur'interroger trois symboles d'éléments et afficher les rayons (en angströms).

En premier, on définit un type dérivé info_covalence capable de stocker, en double précision, le symbole (2 caractères) et trois réels pour les rayons. Après avoir récupéré en argument le nom du fichier, on ouvre en lecture et vérifie systématiquement statut_IO pour intercepter toute erreur d'ouverture.

La lecture se fait en deux étapes:

1. **Compte** les lignes « exploitables » (non vides, non commentées et suffisamment longues) pour déterminer nb_elements.
2. **Parsing** ligne à ligne, avec :
    - extraction du symbole aux colonnes 4–5,
    - découpage des sous-chaînes 6–8, 9–11 et 12–14 pour les rayons simple, double et triple,
    - conversion en réel (avec contrôle d'iostat), division par 100 pour passer de picomètres à angströms,
    - initialisation d'un tableau allocatable covalences(nb_elements) de info_covalence.

Une fois le fichier refermé, on entre dans une boucle de trois requêtes utilisateur : pour chaque saisie d'un symbole d'élément, on parcourt linéairement le tableau pour, si trouvé, afficher les trois rayons formatés à deux décimales, ou signaler « Élément inconnu ». Enfin, on libère la mémoire allouée.

Commande :

./chargeur_covalence CoV_radii

Affiche_topologie

affiche_topologie.f90 combine les fichiers mol2 et des rayons de covalence pour proposer une topologie de liaisons moléculaires en trois grandes étapes.

D'abord, on lit et parse le fichier de rayons via read_covalence, qui compte les lignes valides, alloue un tableau de structures covalence_data (symbole + rayons simple, double, triple en double précision) puis remplit chaque entrée en extrayant des colonnes fixes et en convertissant les valeurs de picomètres en angströms.

Ensuite, read_mol2 ouvre le fichier mol2, recherche les sections @<TRIPOS>MOLECULE> (pour connaître nb_atomes) et @<TRIPOS>ATOM, puis alloue un tableau d'atom_info (symbole + coordonnées 3D). Un contrôle check_elements garantit qu'aucun atome n'échappe à la table de covalence.

La détection de la topologie se fait en analysant toutes les paires d'atomes : pour chaque delta de tolérance de 10 % à 35 % par pas de 5 %, on calcule la distance euclidienne (calculate_distance) et on essaie successivement les types de liaison triple, double et simple via la fonction try_bond_type, qui compare la distance aux bornes définies par la somme des rayons ± delta. Dès que tous les atomes sont connectés (check_connectivity), on affiche la liste des liaisons et on termine, sinon, on augmente le delta ou, passé 35 %, on signale alors l'échec.

Commande :

./affiche_topologie 1QSN_NO_BOND.mol2 CoV_radii

Complete_mol2

complete_mol2.f90 assemble les codes précedent pour faire un fichier mol2 complet, incluant la section BOND, sans écraser l'original. Il s'articule en trois grandes phases :

1. **Lecture et calcul de la topologie**.
   On utilise les éléments précèdent

2. **Insertion de la section BOND**
   – Le fichier mol2 d'origine est relu intégralement ligne par ligne dans un tableau de chaînes (count_file_lines + read_file_lines).
   – On repère la ligne de début de la section ATOM pour savoir où s'arrête la partie atomique, puis on recopie tout le bloc ATOM dans le fichier de sortie.
   – On écrit ensuite la balise @<TRIPOS>BOND suivie de toutes les liaisons trouvées (identifiant, indices des atomes, type "1", "2" ou "3"), avant de recopier le reste du fichier initial.

Commande :

./complete_mol2 1QSN_NO_BOND.mol2 CoV_radii

Parallélisation

Para_complete_mol2, la détection des liaisons est rendue parallèle grâce à OpenMP. Dès le début, on démarre un chronomètre via omp_get_wtime pour mesurer le gain de performance. Puis, on ajoute une directive !$OMP PARALLEL DO avant la double boucle qui parcourt toutes les paires d'atomes. Cette directive transforme la boucle en région parallèle : chaque thread récupère dynamiquement un lot d'itérations à traiter grâce à l'option de planning dynamic, et dispose de ses propres copies des variables de boucle et des temporaires (i,j, distance, symboles d'atomes) déclarées en PRIVATE, tandis que le tableau global des liaisons et le compteur restent partagés.

À l'intérieur de la fonction try_bond_type_omp, tous les calculs (récupération des rayons, comparaison de la distance aux bornes) sont effectués de manière indépendante par chaque thread. Seule l'étape où l'on enregistre une liaison détectée dans le tableau partagé est placée dans une section critique (!$OMP CRITICAL) afin d'éviter qu'au même instant plusieurs threads n'incrémentent simultanément le compteur et n'écrivent sur la même position.

Cette approche tire parti de tous les cœurs disponibles : chaque thread effectue en parallèle les calculs lourds de distance et de test de liaison, et ne se synchronise que pour l'écriture des résultats, ce qui autorise un gain de temps notable dès qu'on travaille sur des molécules de taille significative.

Commande :

./para_complete_mol2 1QSN_NO_BOND.mol2 CoV_radii

Algo génétique

genetic_docking.f90 réalise un docking d'un ligand sur un site cible, avec suivi des performances et génération automatique de tous les fichiers de sortie. Il se décompose en cinq grandes parties :

1. **Définition des types et I/O (modules mol2_types et mol2_io)**
    - On définit d'abord, dans mol2_types, les types dérivés utilisés partout :
        - covalence_data pour stocker les rayons de covalence (simple, double, triple),
        - atom_info pour un atome (symbole + coordonnées 3D),
        - bond pour une liaison (indices et type).
    - Dans mol2_io, on rassemble toutes les routines de lecture/écriture de fichiers Mol2 et de fichiers de rayons :
        - read_covalence et read_mol2 pour charger en mémoire la table des rayons et la liste d'atomes,
        - write_mol2 pour écrire un fichier Mol2 complet (avec sections ATOM et BOND),

- une fonction uppercase et un utilitaire itoa pour les manipulations de chaînes.

2. **Géométrie et topologie (module geometry)**
   - calculate_distance calcule la distance euclidienne entre deux atomes,
   - build_topology génère la topologie du ligand ou du site à partir des rayons de covalence et d'un critère unique (rayon simple × 1,35)
   - apply_transform applique à chaque individu du GA une rotation (trois angles Euler) et une translation (trois déplacements) au ligand.
3. **Évaluation des ponts hydrogène (module hbonds)**
   - La fonction evaluate_hbonds parcourt tous les atomes H du ligand et tous les atomes O/N du site
   - Pour chaque paire candidate, elle vérifie la distance (2,2 Å–4 Å) et l'angle H–donneur–accepteur (90°–150°) en recherchant dans la topologie du site l'atome lié au donneur.
4. **Opérateurs génétiques (module ga_ops)**
   - init_population initialise aléatoirement la population (6 paramètres : 3 rotations en [0,2π], 3 translations en [–10, +10] Å)
   - tournament_selection réalise une sélection par tournoi pour construire la pool de parents
   - one_point_crossover et mutate appliquent un croisement en un point et une mutation ponctuelle (petits écarts sur chaque gène) avec probabilités paramétrables.

5. **Programme principal**
   - On lit les arguments : nom du ligand, du site et répertoire de sortie,
   - On crée les dossiers de résultats (un sous-répertoire par exécution, 10 exécutions au total),
   - Lecture des données statiques : rayons de covalence, atomes et liaisons du site,
   - Pour chaque run (1 à 10) :

1. Initialisation de la population (taille 100),
2. Boucle principale sur 500 générations :
   - Pour chaque individu, on recharge le ligand, on calcule sa topologie, on lui applique la transformation définie par ses gènes,
   - On évalue son score par nombre de ponts hydrogène,
   - On collecte, tous les 10 générations, les valeurs min, max et moyenne des scores,
   - On réalise la sélection, le crossover et la mutation pour produire la nouvelle population.
3. À la fin, on écrit pour chaque individu un fichier Mol2 (ligand transformé + topologie) et un fichier CSV des statistiques.

Commande :

./genetic_docking ligand_NO_BOND.mol2 site.mol2

Para algo génétique

Dans **genetic_docking.f90**, la partie la plus coûteuse est l'évaluation du score de chaque individu (rechargement du ligand, reconstruction de sa topologie, application de la transformation puis comptage des ponts hydrogène) est parallélisée grâce à OpenMP.
On utilise la directive : !$OMP PARALLEL DO DEFAULT(shared) PRIVATE(i, lig_atoms, n_lig, lig_bonds, n_lig_b) SHARED(pop, scores, lig_in, site_atoms, n_site, covs, n_cov, site_bonds, n_site_b)avant la boucle.

Chaque thread prend en charge un sous-ensemble d'indices i, recharge et transforme son propre ligand, calcule son score et écrit dans scores(i) sans interférence (variables locales privées). À la fin de la région parallèle (!$OMP END PARALLEL DO), tous les scores sont disponibles pour la sélection.

Commande :

./para_genetic_docking ligand_NO_BOND.mol2 site.mol2

Questions

1. **Performance sur une molécule plus grosse**

   Lorsque l'on passe du petit fichier test à une molécule plus volumineuse, le temps de calcul de la détection de topologie (paires d'atomes × calcul de distance × tests de liaison) augmente rapidement. En effet, la complexité est en $O(N^2)$ pour NNN atomes, car on doit examiner chaque paire. Sur un exemple 5 × plus lourd (en nombre d'atomes), on constate un temps multiplié par plus de 20 : c'est donc tout à fait attendu, étant donné que $(5^2)=25$ paires à tester au lieu de 1, et quelques surcoûts liés à la gestion mémoire.

2. **Possibilité de paralléliser la recherche de topologie**

   Oui, la phase de détection de liaisons est entièrement parallélisable : chaque paire (i,j) est traitée indépendamment (calcul de distance, comparaison aux bornes). Les seules dépendances sont lors de l'enregistrement de la liaison validée (incrément d'un compteur et écriture dans un tableau partagé). On peut donc :

- distribuer les itérations de la double boucle for i=1…N-1; for j=i+1…N sur plusieurs threads (OpenMP, MPI, etc.),
- protéger la mise à jour du compteur/tableau par une section critique