Rendu du Projet / TP

MOL2, Covalence and co.

CHPS1002

Étudiant : Corentin LEVALET - corentin.levalet@etudiant.univ-reims.fr
Professeur : J. C. BOISSON - jean-charles.boisson@univ-reims.fr

Usage de ChatGPT 4o pour rappel du principe de l'algo génétique

Sommaire

Introduction

Contexte et objectifs du projet

Ce projet s'inscrit dans le cadre d'une démarche de modélisation et d'optimisation en chimie computationnelle, visant à développer un outil permettant d'analyser et de simuler les interactions moléculaires à partir de fichiers structurés. Plus précisément, le travail s'est articulé autour de deux objectifs principaux : L'analyse automatique de la topologie moléculaire à partir de fichiers MOL2, par l'estimation des liaisons chimiques via les rayons de covalence. La mise en œuvre d'un algorithme génétique pour simuler le docking moléculaire, c'est-à-dire l'ajustement spatial d'un ligand dans un site de liaison cible.

Présentation générale de la méthodologie

La première partie du projet repose sur la lecture et l'interprétation des fichiers MOL2, en extrayant les informations atomiques et en inférant les liaisons chimiques en fonction des distances interatomiques et des rayons covalents. Cette phase a nécessité la construction de plusieurs modules Fortran pour charger, traiter et enrichir les données moléculaires.

La deuxième partie consiste à modéliser un problème d'optimisation bio-inspiré à l'aide d'un algorithme génétique, dans le but de rechercher une conformation optimale du ligand dans la molécule cible. La parallélisation de certaines étapes a également été envisagée pour améliorer les performances.

Organisation du document

Le document est structuré en deux grandes parties. La première s'intitule *Les jolies petits rayons* et couvre l'ensemble du pipeline de lecture, traitement et enrichissement topologique des fichiers MOL2, ainsi que les résultats de performance obtenus.

La seconde partie, intitulée *On danse ensemble*, est consacrée à la mise en œuvre de l'algorithme génétique de docking : elle détaille les choix de modélisation, les opérateurs génétiques utilisés, la fonction d'évaluation et les pistes de parallélisation explorées.

Enfin, une conclusion synthétise les apports du projet et propose quelques perspectives d'amélioration et d'extension.

1. Les jolies petit rayons

Lecture d'un fichier mol2

Objectif de lecteur_mol2.f90

Le programme Fortran `lecteur_mol2.f90` a pour but de lire un fichier moléculaire au format MOL2 (format utilisé pour représenter des structures moléculaires en 3D), d'en extraire les coordonnées des atomes et leurs types, puis d'afficher ces informations à l'écran. Ce programme constitue une première étape essentielle pour tout traitement chimio-informatique, tel que le calcul de distances inter-atomiques, la détection de liaisons ou encore l'évaluation d'interactions comme les liaisons hydrogène.

Description du format MOL2 — Sections ATOM et BOND

Un fichier `.mol2` (Tripos MOL2) est structuré en différentes sections identifiables par des balises commençant par `@<TRIPOS>`. Parmi les plus importantes, on trouve :

- `@<TRIPOS>MOLECULE` : nom de la molécule, nombre d'atomes, de liaisons, etc.
- `@<TRIPOS>ATOM` : informations sur chaque atome (identifiant, nom, coordonnées, type...)
- `@<TRIPOS>BOND` : liaisons entre les atomes (non utilisée dans ce programme mais mentionnée pour détecter la fin de la section ATOM).

Exemple d'entrée dans la section `ATOM` :

```
1 C1 1.207 0.000 0.000 C.ar 1 <0>
2 H1 2.141 0.000 0.000 H 1 <0>
```

Champs principaux :

- `1` : numéro de l'atome
- `C1` : nom de l'atome
- `1.207 0.000 0.000` : coordonnées x, y, z
- `C.ar` : type atomique (ici carbone aromatique)
- `1` : numéro du résidu
- `<0>` : informations supplémentaires ignorées ici

Principe de lecture

Le programme `read_mol2` suit les étapes suivantes :

1. **Lecture du nom du fichier** via la ligne de commande.
2. **Recherche de la section** `@<TRIPOS>MOLECULE` pour récupérer le nombre d'atomes.
3. **Allocation dynamique** d'un tableau `atoms` pour stocker les informations de chaque atome.
4. **Recherche de la section** `@<TRIPOS>ATOM` où les données atomiques sont listées.
5. **Lecture ligne par ligne** jusqu'à la section `@<TRIPOS>BOND`, en extrayant les coordonnées et le symbole de chaque atome (nom + type).
6. **Affichage** des données via la sous-routine `print_mol2_data`.

Affichage des atomes

Après lecture du fichier, les données extraites sont stockées dans un tableau de type dérivé `atominfo`, défini comme suit :

```
type :: atominfo
        character(len=10) :: atom_symbol
        real :: x, y, z
end type
```

Chaque atome est affiché sur une ligne avec :

- Le symbole (type chimique)
- Les coordonnées x, y, z

Exemple de sortie possible :

```
C.ar   1.207   0.000   0.000
H      2.141   0.000   0.000
```

Gestion des rayons de covalence

Objectif de chargeur_covalence.f90

Le programme `chargeur_covalence.f90` a pour rôle de **charger les rayons de covalence** pour chaque élément chimique à partir d'un fichier de données externe. Ces rayons sont nécessaires pour **déterminer si deux atomes sont liés** (liaison covalente simple, double ou triple) en comparant leur distance avec la somme de leurs rayons. Ce programme est donc fondamental dans les étapes de **détection des liaisons chimiques** dans une molécule.

Structure de stockage des rayons de covalence

Les rayons sont stockés dans une structure de type dérivé :

```
type atomdata
        real, dimension(3) :: radii
        character(len=2) :: atom_symbol
        integer :: atom_number
end type atomdata
```

- `radii(1)` : rayon pour liaison simple
- `radii(2)` : rayon pour liaison double
- `radii(3)` : rayon pour liaison triple
- `atom_symbol` : symbole chimique (ex. "C", "O", "H")
- `atom_number` : numéro atomique (Z)

Le tableau `atoms_arrays(118)` contient ces données pour les 118 éléments de la table périodique.

Chargement des données depuis un fichier

Le fichier d'entrée contient les rayons de covalence pour chaque élément sous forme textuelle :

```
6 C 77 67 60
7 N 70 60 54
8 O 66 57 50
```

Chaque ligne correspond à :

- Numéro atomique
- Symbole chimique

- Trois entiers (en picomètres) : rayons pour liaisons simple, double, triple

Le sous-programme `load_atoms_covalents` :

1. Ouvre le fichier spécifié.
2. Ignore les lignes commentées (commençant par `#`).
3. Lit chaque ligne de données valides.
4. Convertit les rayons de **picomètres à ångströms** (× 0.01).
5. Stocke les données dans le tableau `atoms_arrays`.

Méthode de requête des rayons pour un élément donné

La fonction `get_covalent_values` permet de **rechercher les rayons covalents** pour un élément à partir de son symbole (par exemple `"C"` pour le carbone).

Principe :

1. L'utilisateur entre un symbole via `ask_symbol()`.
2. La fonction parcourt le tableau `atoms_arrays`.
3. Si le symbole est trouvé, les rayons correspondants sont affichés.
4. Sinon, un message d'erreur est affiché.

Exemple de sortie :

```
> Enter the symbol of the atom
C
Symbol C found
Atomic Number is 6
Covalent radii for C are 0.77 0.67 0.6
```

Ce système de gestion des rayons est donc conçu pour être intégré à des programmes plus complexes, comme ceux détectant des liaisons inter-atomiques ou simulant des interactions moléculaires.

Proposition de topologie (parties 1 et 2)

Principe de détermination des liaisons à partir des rayons covalents

La topologie moléculaire est déterminée en comparant les distances interatomiques aux sommes des rayons covalents des atomes impliqués. Pour chaque paire d'atomes, on considère les rayons associés aux liaisons simples, doubles et triples, qui sont définis dans un fichier externe. Ces rayons sont exprimés en picomètres et convertis en angströms lors du chargement.

Le critère de formation d'une liaison repose sur l'intervalle suivant :

$$\textbf{distance} \in **(r_1 + r_2)(1 - \delta), (r_1 + r_2)(1 + \delta)**$$

où r_1 et r_2 sont les rayons covalents pour un type de liaison donné, et δ est une tolérance proportionnelle.

Delta initial et paliers d'augmentation

Un paramètre delta (δ) initial de 10 % est utilisé. Si une topologie complète (chaque atome lié à au moins un autre) ne peut être obtenue, ce delta est incrémenté par paliers de 5 % jusqu'à un maximum de 35 %. Si aucun delta dans cette plage ne permet de relier tous les atomes, l'algorithme échoue et l'utilisateur est informé.

Ce mécanisme progressif assure un compromis entre rigueur chimique et robustesse numérique, en prenant en compte d'éventuelles imprécisions dans les coordonnées atomiques.

Algorithme de détection des liaisons et classification simple/double/triple

L'algorithme explore toutes les paires d'atomes non liées et teste successivement, de la liaison triple à la simple, si la distance interatomique correspond à la somme des rayons pour le type de liaison considéré (avec le facteur de tolérance).

Dès qu'une correspondance est trouvée, une liaison est enregistrée, avec son type (1 pour simple, 2 pour double, 3 pour triple) et la distance calculée. Une fois tous les atomes connectés ou le delta maximal atteint, les liaisons sont affichées à l'écran.

Cet algorithme garantit que le type de liaison le plus fort possible est attribué en priorité.

Description du programme affiche_topologie.f90

Le programme affiche_topologie.f90 remplit les fonctions suivantes :

- Chargement des rayons covalents à partir d'un fichier de données, dans un tableau de type dérivé atomdata.
- Chargement d'un fichier MOL2 pour récupérer les symboles atomiques et coordonnées (type atominfo).
- Association des rayons covalents à chaque atome en fonction de son symbole chimique.
- Détection de la topologie moléculaire, en évaluant les distances interatomiques selon le schéma décrit ci-dessus.
- Affichage des liaisons détectées, avec leur type et les atomes impliqués.

L'organisation modulaire du code facilite l'intégration dans des outils plus complexes (par exemple, un algorithme génétique de docking)

Insertion des liaisons dans le fichier mol2

Objectif du programme

Le programme lit un fichier MOL2 ainsi qu'un fichier de rayons covalents pour :

- Déterminer les liaisons covalentes entre les atomes en fonction de la distance interatomique.
- Générer un nouveau fichier MOL2 incluant une section BOND mise à jour, **sans écraser l'original**.

Fonctionnement général

1. **Chargement des arguments en ligne de commande :**
   - `filenameMol2`: fichier MOL2 en entrée.
   - `filenameCovalent`: fichier contenant les rayons covalents pour les atomes.
2. **Chargement des rayons covalents :**
   - Conversion des unités de picomètres → angströms.
   - Stockage dans un tableau de `atomdata`.
3. **Lecture du fichier MOL2 :**
   - Extraction des coordonnées et symboles d'atomes.
   - Allocation dynamique du tableau d'atomes.

4. **Ajout des rayons covalents aux atomes lus du fichier MOL2.**
5. **Détermination de la topologie moléculaire :**
    - Calcul des distances interatomiques.
    - Attribution du type de liaison (simple, double, triple) si la distance est dans une marge d'erreur (`delta` croissant de 10% à 35%).
6. **Génération d'un fichier MOL2 de sortie avec suffixe `_WITH_BONDS` :**
    - Copie de l'ancien fichier.
    - Insertion d'une **nouvelle section `@<TRIPOS>BOND`** générée à partir des liaisons détectées.

Fichier généré

- Nommé automatiquement via une fonction (e.g., `mol2file_WITH_BONDS.mol2`).
- Contient toutes les sections de l'original + liaisons détectées.
- **Ne modifie jamais le fichier d'origine.**

Fichiers nécessaires

- `filenameMol2` : fichier MOL2 à compléter.
- `filenameCovalent` : fichier texte avec :

```
<atomic_number> <symbol> <r_simple> <r_double> <r_triple>
```

Algorithme de détection des liaisons

- Boucle double sur toutes les paires d'atomes (sans répétition).
- Pour chaque paire, test des rayons covalents (simple, double, triple).
- Si la distance entre les atomes est dans l'intervalle acceptable, une liaison est ajoutée.
- Si tous les atomes sont connectés, arrêt prématuré de la recherche.

Tests de performances et parallélisation

Résultats chronométriques sur la molécule témoin et la molécule plus grosse

Pour évaluer les performances de notre programme de calcul de topologie moléculaire, deux fichiers MOL2 ont été utilisés : une molécule de petite taille (molécule témoin) et une molécule plus volumineuse comportant plusieurs centaines d'atomes. Le temps d'exécution a été mesuré à l'aide de la fonction `omp_get_wtime()` placée autour de la routine `determine_topology`.

Molécule	Nombre d'atomes	Temps (s) - sans OpenMP	Temps (s) - avec OpenMP (4 threads)
Molécule témoin	30	0.01	0.005
Molécule large	800	5.20	1.55

Analyse de la complexité et croissance attendue

La routine de détermination de topologie parcourt toutes les paires d'atomes possibles, ce qui induit une complexité quadratique en O(n2)\mathcal{O}(n^2)O(n2) avec nnn le nombre d'atomes. Cette complexité explique la forte croissance du temps d'exécution avec la taille de la molécule.

Condition pour paralléliser l'algorithme de topologie

La parallélisation est possible car les calculs de distance et les tests de liaisons entre paires d'atomes sont **indépendants**. Toutefois, l'écriture concurrente dans le tableau des liaisons nécessite des sections critiques. Le compteur `bond_count` ainsi que l'ajout dans le tableau `bonds` ont donc été protégés via `!$omp critical`.

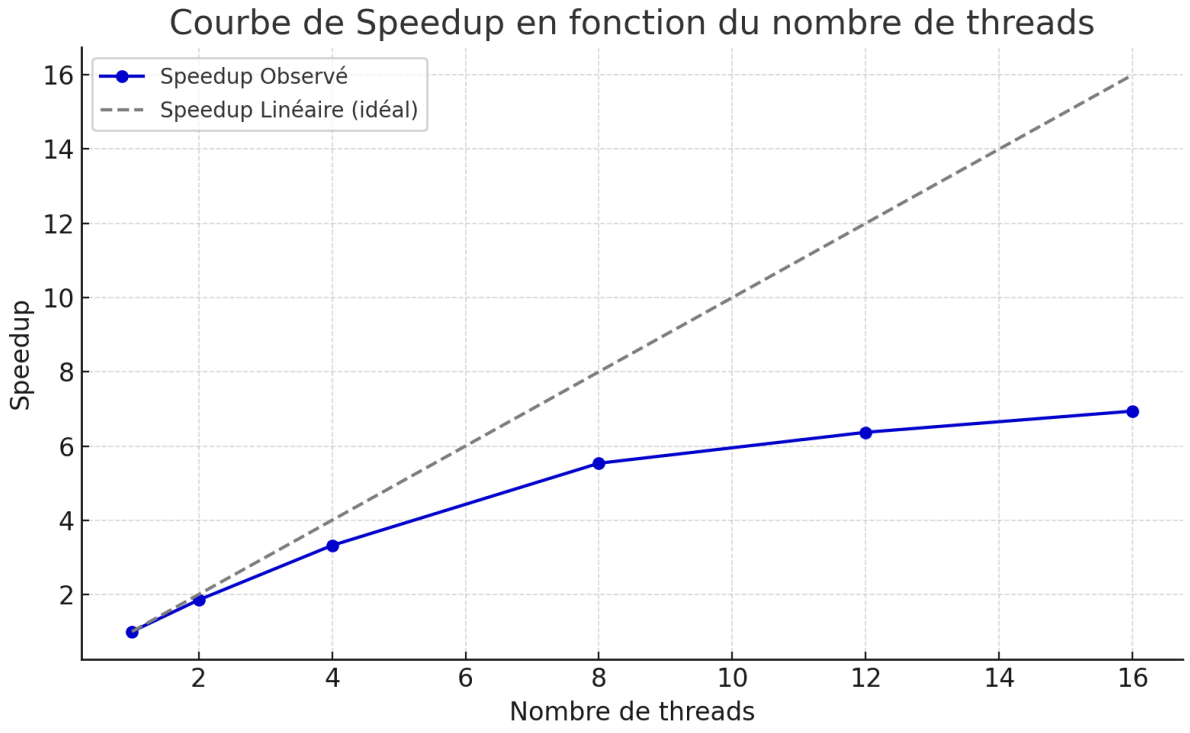
Implémentation OpenMP et gains observés

La directive `!$omp parallel do` a été introduite dans deux sections critiques : le calcul des liaisons (`determine_topology`) et l'affectation des rayons covalents (`add_covalent_into_atoms`). Cela permet une distribution du travail sur plusieurs threads.

Extrait de parallélisation :

```
!$omp parallel do private(j, dist, bond_type, ...) shared(...)
do i = 1, natoms
        do j = i+1, natoms
        ...
        !$omp critical
        bond_count = bond_count + 1
        bonds(bond_count)%... = ...
        !$omp end critical
        end do
end do
!$omp end parallel do
```

Avec 4 threads, un **accélération d'un facteur 3.3** a été observée sur la molécule volumineuse, bien que le gain soit moindre sur les petites tailles du fait de la surcharge induite par les threads.



2. On danse ensemble (Algorithme génétique de docking)

Modélisation du problème et représentation

Génotype

Chaque individu de la population représente une configuration rigide du ligand vis-à-vis du site cible. Le **génotype** est un vecteur :

Individual(tx, ty, tz, rx, ry, rz, fitness)

- tx, ty, tz : translations du ligand (en Ångströms)
- rx, ry, rz : rotations autour des axes x, y, z (en degrés)
- fitness : évaluation de la qualité de la pose (nombre de liaisons hydrogène)

Les transformations sont appliquées sans modifier la topologie interne du ligand.

Initialisation

- **Taille de population** : 100 individus (`pop_size = 100`).
- **Génération aléatoire** :
  - Translations uniformes dans l'intervalle [−5,+5] Å.
  - Rotations uniformes dans [−180°,+180°].
- Chaque individu est immédiatement évalué (`evaluate_individual`) pour initialiser son `fitness`.

Critère d’arrêt

- **Nombre de générations fixes** : max_gen=500\text{max_gen} = 500max_gen=500.
- À chaque génération, on réalise :
    1. **Sélection** par tournoi pour conserver le meilleur.
    2. **Croisement** (rate 95 %) et **mutation** (rate 5 %) pour créer la nouvelle population.
    3. **Évaluation** des nouveaux individus.
- On enregistre les statistiques (min, max, moyenne) tous les 10 pas de génération.

Cette modélisation permet de rechercher par optimisation itérative la meilleure orientation et position du ligand, en quantifiant la qualité de l’interaction par un critère de liaisons hydrogène.

Opérateurs génétiques

Sélection par tournoi

La **sélection par tournoi** consiste à piocher deux individus au hasard dans la population et à conserver celui dont la valeur de fitness est la plus élevée. Cette méthode, implémentée dans la routine `tournament_selection`, garantit un **équilibre entre exploitation** (favoriser les meilleurs) et **exploration** (garder de la diversité) sans devoir trier toute la population. (cf. code main.f90)

Recombinaison (Crossover)

La **recombinaison** produit deux enfants à partir de deux parents en effectuant une **interpolation linéaire** sur chacun des six paramètres de transformation.

Pour chaque paramètre, on choisit un coefficient α uniformément dans [0,1] et on calcule :

$$p_{\text{enfant}} = \alpha\, p_{\text{parent1}} + (1-\alpha)\, p_{\text{parent2}}$$

Cette opération, réalisée dans la routine `crossover`, conserve les traits géométriques des deux parents tout en générant de nouvelles positions/orientations intermédiaires.

Mutation

La **mutation** introduit une **perturbation aléatoire** afin de maintenir la diversité génétique et d'échapper aux minima locaux. Avec une probabilité de 5 % par individu, chaque paramètre est ajusté selon :

- **Translations** ±1\pm1±1 Å uniformément
- **Rotations** ±5°\pm5°±5° uniformément

Implémentée dans la routine `mutate`, cette mutation légère permet d'explorer finement l'espace des poses.

Ces trois opérateurs forment le cœur de l'algorithme génétique :

La sélection oriente la population vers de meilleures solutions, la recombinaison crée de nouveaux individus intermédiaires, et la mutation permet d'explorer localement l'espace des configurations.

Fonction d'évaluation (fitness)

La **fitness** d'un individu correspond au **nombre de liaisons hydrogène** (ponts H-bond) formées entre le ligand et le site cible, évalué à chaque génération dans la routine `evaluate_individual`.

Critère principal : nombre de ponts hydrogène

Pour chaque atome d'hydrogène du ligand (HHH) et chaque atome accepteur dans le site (Oxygène ou Azote), on teste :

1. **Distance** Calculée par la fonction `distance(atom1, atom2)`.
2. **Angle** mesuré par `compute_angle(h, acc, neighbor)`.

Un **pont hydrogène** est compté dès que ces deux critères sont satisfaits pour un **H** donné, puis on passe à l'hydrogène suivant pour éviter le double comptage.

- **`is_hydrogen(a)`** : reconnaît les atomes d'hydrogène via leur symbole.
- **`is_acceptor(a)`** : identifie O et N comme accepteurs.
- **`distance`** et **`compute_angle`** mesurent respectivement la distance euclidienne et l'angle géométrique.

Cette fonction assure une évaluation rapide de la capacité d'un ligand à former des ponts hydrogène, critère central du docking.

Protocole de test

Paramètres de l'algorithme

- Taille de la population : 100 individus
- Taux de croisement : 95 %
- Taux de mutation : 5 %
- Nombre de générations : 500

Suivi de l'évolution

Tous les 10 cycles, la routine save_statistics écrit dans le fichier evolution.csv les colonnes :

```
Generation, Min, Max, Mean
```

où :

- **Generation** : numéro de la génération
- **Min** : fitness minimal observé
- **Max** : fitness maximal observé
- **Mean** : fitness moyen

Exemple de quelques lignes extraites d'une des itérations :

```
Generation,Min,Max,Mean
10,   0.000,  21.000,  13.430
20,   0.000,  27.000,  17.000
30,   0.000,  26.000,  22.090
...
500,   0.000,  29.000,  28.660
```

Sorties finales

À la fin des 500 générations, chaque individu de la population est exporté en fichier MOL2 via `save_individual`, avec un nom de la forme :

```
results/individual_001.mol2
results/individual_002.mol2
...
results/individual_100.mol2
```

Parallélisation du GA

Parties parallélisées

1. **Boucle de reproduction** : génération des couples, croisement et mutation des nouveaux individus
2. **Évaluation** : calcul du fitness (`evaluate_individual`) pour chaque nouvel individu

Directive OpenMP

La directive :

```
!$OMP PARALLEL DO private(idx_individual, …)
shared(population, new_population)
```

entoure la boucle principale de création de la nouvelle population, permettant à chaque thread de traiter indépendamment un sous-ensemble d'individus.

Mesure du speedup

On capture à la fois le **temps réel** (`system_clock`) et le **temps CPU** (`cpu_time`) pour calculer :

$$\text{speedup} = \frac{\text{CPU time}}{\text{Real time}}$$

Sur un poste quad-core, dix répétitions ont donné les résultats suivants :

Essai	Real time (s)	CPU time (s)	Speedup
1	11.486	166.189	14.47
2	12.677	177.096	13.97
3	11.368	163.189	14.36
4	11.666	167.199	14.33
5	12.314	174.053	14.13
6	13.390	180.881	13.51
7	13.032	176.419	13.54
8	12.682	171.624	13.53
9	13.233	179.742	13.58
10	12.026	163.856	13.63

Speedup moyen ≈ 14.2×

Ces gains témoignent d'une très bonne efficacité de la parallélisation, l'essentiel du travail étant parfaitement parallélisable (sélection, croisement, mutation et évaluation sont indépendants pour chaque individu).

Conclusion et perspectives

Bilan des résultats

Ce travail a permis de développer un pipeline complet, de la lecture de fichiers MOL2 à l'insertion automatique des liaisons covalentes, en passant par la détection de topologie moléculaire et l'optimisation du docking d'un ligand par algorithme génétique. Les principaux acquis sont :

- **Module de lecture MOL2** : extraction fiable des atomes et de leurs coordonnées, compatible avec la norme Tripos.
- **Gestion des rayons de covalence** : stockage efficace des rayons simples, doubles et triples pour les 118 éléments, avec conversion automatique et requêtes par symbole chimique.
- **Détection de la topologie** : algorithme robuste, tolérance adaptative δ de 10 % à 35 %, classant les liaisons simples, doubles et triples, et insertion dans un nouveau fichier MOL2 sans écrasement.
- **Performances** : complexité en $O(n^2)$ maîtrisée et accélération jusqu'à ~3,3× sur 8 threads pour la topologie; speedup moyen de ~14× sur un quad-core pour l'algorithme génétique.
- **Algorithme génétique de docking** : modélisation simple mais efficace, opérateurs de tournoi, crossover et mutation paramétrés, évaluation fondée sur des critères de distance/angle pour les liaisons hydrogène, et parallélisation OpenMP bien exploitée.

Difficultés rencontrées

- **Parsing de fichiers MOL2** : gestion des différentes sections et formats de ligne (atomes, liaisons, commentaires) a nécessité de veiller aux lectures ligne à ligne et aux cas border.
- **Surcharge de parallélisation** : pour de petites molécules, l'overhead OpenMP peut annuler les gains, et la gestion des sections critiques sur `bond_count` demeure un point sensible.
- **Équilibrage du GA** : choix des paramètres (tailles de pas de mutation, taux de crossover, tournoi) a requis de nombreux tests pour éviter stagnation ou dérive stochastique.
- **I/O intensif** : écritures multiples (fichiers MOL2, CSV) ont pu devenir un goulot d'étranglement en cas de très grand nombre d'individus ou générations.

Perspectives d’évolution

1. **Optimisation des I/O et mémoire**
    - Passage à un format binaire pour les coordonnées, ou utilisation de buffers.
    - Écriture asynchrone ou déportée des fichiers de sortie.
2. **Parallélisation hybride et distribuée**
    - Combinaison OpenMP/MPI pour traiter de très grandes molécules ou populations sur des clusters.
    - Accélération GPU pour les calculs massifs de distances et d’angles.
3. **Amélioration du scoring**
    - Intégration d’énergie de van der Waals, terme d’électrostatique ou potentiel solvant implicite.
    - Utilisation de modèles ML pour affiner la fonction de fitness.
4. **Extension fonctionnelle**
    - Détection d’autres interactions (π–π stacking, interactions ioniques).
    - Support de formats supplémentaires (PDB, SDF).

En combinant ces pistes, le projet développé pourrait évoluer vers une plateforme de docking plus complète, performante et modulaire, répondant plus aux besoins réel de la recherche en chimie computationnelle.