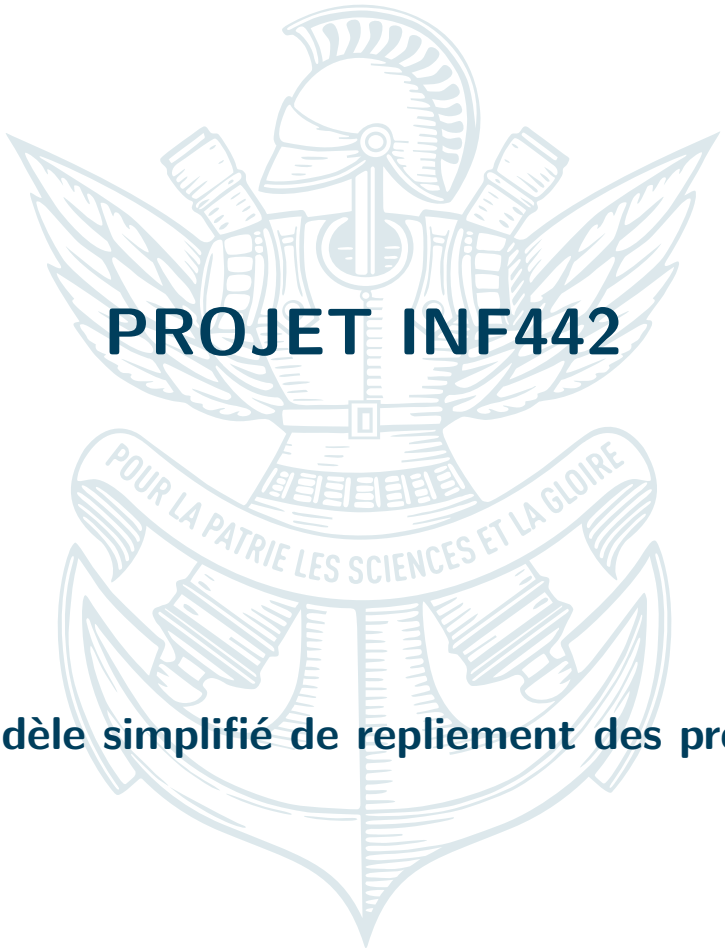

PROJET INF442

Un modèle simplifié de repliement des protéines

Hua-ting YAO et Qi SUN

1 CONTEXTE ET PROBLÉMATIQUE

Dans ce projet, étant donnée une séquence d'acides aminés, on cherche à déterminer la plus stable structure de la proteine constituée de cette séquence d'acides aminés. Pour simplifier, les acides aminés sont seulement divisés en deux groupes par selon leurs caractéristiques chimiques et biologiques : polaire et hydrophobe. Deux acides aminés hydrophobes peuvent former une liaison entre eux afin de renforcer la stabilité de la structure de la proteine. Le but est donc d'établir une structure qui contient le plus possible de liasons H-H. De plus, en réalisant cette objective, on arrive à mettre les acides aminés de type H à l'intérieur de ceux de type P. Comme les acides aminés polaires sont peuvent former des liaisons avec l'eau, cela peut donc encore améliorer la stabilité de la structure.

2 STRUCTURE DES DONNÉES

2.1 Acide aminé

Avant de définir la structure d'une protéine, on a tout d'abord définit une classe **AcideAmine** qui contient quatre champs :

-> Un champs **int indice** qui présente l'indice de l'acide aminé dans la séquence d'une protéine ;
-> Puis, deux champs **int x** et **int y** qui présentent respectivement les coordonées des abscisses et des ordonées de la position de l'acide aminé ;
-> Enfin, un champs **char valeur** qui porte information sur le type de l'acide aminé (polaire ou hydrophobe).

2.2 Proteine

La classe **Proteine** s'appuie sur celle de l'**AcideAmine** et comporte dix champs en total :

-> **std : :string sequence** qui correspond à la séquence d'acides aminés de la protéine ;
-> **std : :vector<AcideAmine*> proteine** qui permet de stocke la structure de la protéine dans un tableau de pointeurs d'acides aminés ;
-> Un champs **int l** qui donne information sur la longueur de la protéine ;
-> Un champs **std : :vector<AcideAmine*> hydrophobes** qui consiste à stocker tous les acides aminés hydrophobes dans la séquence de la protéine ;

Ces quatres champs sont intrinsèques à la séquence de la protéine et ne dépendent pas de sa structure. Mais nous avons aussi construit des champs qui permettent de rendre la recherche des solutions approchées ou exhaustives plus facile.

Par example, dans la solution approximative, les deux deux tableaux **v** et **vInv** de couple d'entiers dont le couple (a_i, b_i) de chaque case correspond respectivement aux nombres d'acides aminés hydrophobes d'indice impair et ceux d'indice pair avant et après celui d'indice i :

-> **std : :vector<std : :vector<int> > v** ;

-> **std : :vector<std : :vector<int> > vInv** ;

Avec ces deux tableaux, on arrive à calculer l'indice k qui donne la valeur maximale de n_{ref}. En effet, plusieur valeurs de k sont possibles et on prend la médiane de ces valeurs car le minimum (k_{min}) et le maximum (k_{max}) d'entre eux correspondent forcément à des acides aminés hyprophobes d'indice de parité différente. Le fait de plier à la médiane permet donc de déjà former une liaison entre les deux acides aminés d'indice resp. k_{min} et k_{max}. Ensuite, avec ce k bien choisi, on détermine les acides aminés à gauche et à droite qui se formeront des liaisons entre eux et on les stocke dans les deux tableaux de pointeur d'acides aminés suivante :

-> **std : :vector<AcideAmine*> typePL** ;

-> **std : :vector<AcideAmine*> typePR** ;

Nous avons choisi de construire des tableaux de pointeurs au lieu de ceux d'objets car avec les pointeurs si on modifie la position d'un acide aminé, ce sera automatiquement modifié dans les tableaux correspondants et cela simplifie la complexité des programmes.

Enfin, on a deux entiers qui représentent les résultats de la recherche exhaustive :

-> **int neff** qui correspond au nombre le maximal possible de liaisons qui peuvent être formées dans la protéine ;

-> **int nbOpt** qui correspond au nombre de solutions optimale.

3 SOLUTION APPROCHÉE

Pour chercher la solution approchée décrite dans l'énoncé, on a d'abord écrit quelques programmes auxiliaires qui permettent de mettre à jour les tableaux **v, vInv, typeHR, typeHG** décrits précédemment et de calculer la valeur de k également.

Avec ceci, on connaît déjà une structure de la protéine qui réalise au moins $n_{ref} - 1$ contacts. Pour mettre une telle structure en évidence, on commence par deux programmes auxiliares qui consistent à ranger une sous-séquence d'acides aminés, étant données les coordonnées des positions de la tête et la fin de cette sous-séquence, respectivement dans la colonne de gauche et dans celle de droite.

Grâce à ces deux programmes auxilaires **RangerAutoRight(AcideAminé* a, AcideAminé* b)** et **RangerAutoLeft(AcideAminé* a, AcideAimné* b)**, et les deux tableaux **typeHR** et **typeHG** dans lesquels on a stocké les acides aminés qui se formeront des contacts entre eux ; on arrive à mettre à jour les positions de tous les acides aminés de la protéine et à tracer sa structure dans un fichier svg.

Cependant, dans ce programme, on se contente d'avoir au moins $n_{ref} - 1$ contacts et on n'a pas cherché à maximiser le nombre de contacts possibles en tenant compte des contacts supplémentaires, comme ceux de parité inversée. Par exemple, avec la séquence correspondant à la première protéine de la figure 4, nous obtenons lun résultat avec seulement 2 contacts au lieu de 3 contacts.

4 SOLUTION EXHAUSTIVE

Pour la solution exhaustive, on a tout d'abord commencé par un programme récursif ; puis, comme on a pensé qu'un programme itératif pourrait être plus efficace en terme de temps, on a donc aussi écrit un programme itératif et a effectué des amélioration sur ce programme.

4.1 Programmation récursive

Le programme récursif consiste à chercher les solutions optimales de façon exhaustive comme décrite dans la section **2.2.1 Principe général** de l'énoncé.

Une fonction auxiliaire **bool notOverlap(int i)** qui renvoie **true** si, supposé les acides aminés d'indice de 0 à $i-1$ rangé, la position que l'on choisit pour i est différente des positions de tous les acides aminés qui le précède ; et une autre fonction auxilaire **int calculeNeff()** qui permet, étant donnée la structure d'une protéine, de calculer le nombre de contacts dans cette structure ; ces deux programmes serviront à écrire le fonction finale de méthode récursive : **int RangerRecursif(int i, Proteine p)** qui renvoie un entier compris entre 0 et 4 qui signifie combien de places autour du acide aminé d'indice $i-1$ ont été prises et renvoie -1 lorsque l'on a déjà placer tous les acides aminés. La **Proteine p** nous sert d'un compteur, que l'on modifie à chaque essaie de placement. Néanmoins, on modifie la protéine considérée en **p** seulement si la valeur de n_{eff} de **p** est plus grand que celui de la protéine considérée. La fonction met également à jour le nombre de solutions optimales et la valeur maximale de n_{eff} à chaque fois où on considère une nouvelle structure ou modifie la protéine.

Etant donnée une séquence d'acides aminés **s**, on fixe les positions des acides aminés d'indice 0 et 1. Puis, il suffit d'exécuter **RangerRecursif(2,p)** avec **p** une protéine construite à partir de la même séquence **s**, pour calculer le maximum de n_{eff} et le nombre de solutions optimales.

4.2 Programmation itérative

Nous implémentons aussi un programme itérative. Tout d'abord, nous définissons un nouveau champs **vector<int> pos** dans la classe **Proteine.pos[i]** qui enregistre la position de i-ième acide aminé par rapport le $(i-1)$-ième. 0,1,2,3 présentent respectivement à droite, au dessus, à gauche et en dessous. Nous posons pos[0] = 0. Chaque structure peut ainsi être présentée par un nombre en base 4. Par exemple, 000000 présente un structure linéaire d'une chaîne d'acide aminé de taille 6.

Supposons que la taille de la chaîne d'acide aminé est 6. L'idée du programme est de parcourir sur toutes les structures à partir de 000000 (Tout acide aminé est à droite du précédent). Dans chaque itération, on teste la structure "+1" par rapport le précédent. i.e. 000000 -> 000001, 000233->000300 etc. Le parcours termine quand la structure est égale à la structure de terminaison, par exemple 010000 pour le programme séquentiel. Ceci présente la structure dans laquelle l'acide aminé a_1 est au dessus de a_0. Le parcours termine en raison de la rotation. Si une structure a un nombre de contacts plus que celui stocké. Nous remplaçons la structure stockée par ceci.

Cet algorithme naif est coûteux parce que le parcours est réalisé sur toutes les structures. Nous ajoutons une fonction **shift()** qui renvoie la prochaine structure possible en sautant toutes celles ayant au moins deux acides aminés qui occupent la même position ou dont le premier déplacement vertical n'est pas 1 (au dessus de la précédent) pour éviter de construire des solutions symétriques.

Chaque acide aminé de type H a deux contacts possibles avec les acides aminés de type H, trois si c'est en extrémité. Supposons que nous avons construit une structure jusqu'à acide aminé a_i et qu'il y a k acides aminés de type H parmi la suite. Nous n'avons plus besoin de continuer cette branche si le nombre de contacts jusqu'à a_i + 2*k (+1, si un des k est en extrémité) est inférieur au meilleur nombre de contact à l'instant. Nous ajoutons cette vérification dans la fonction **shift()** et utilisons la solution approchée comme le seuil initial.

L'algorithme est :
seuil = solutionApproche();
s = le structure présenté par une chaine de 0;
nbOpt = 0 ;
sMax = s;
while *structure courrant est différent que la terminaison* **do**
  if *le nombre de contact de s > celui de sMax* **then**
    sMax = s;
    nbOpt = 0;
    seuil = le nombre de contact de sMax;
  end
  else if *le nombre de contact de s == celui de sMax* **then**
    nbOpt++;
  end
  s = shift();
end
return *sMax et nbOpt*

4.3 Mise en place de MPI

Etant donné une puissance de trois de processeurs, par exemple 3^k, on envoie les 3^k différentes structures pour les $(k+2)$ premiers acides aminés dans la séquence aux processeurs, en utilisant $\mathrm{MPI}_Scatterv.Chaqueprocesseurcalculeunevaleurden_{eff}$ et une valeur du nombre de solutions pour ce n_{eff} locales. Ensuite, on utilise $\mathrm{MPI}_Reducepourcalculelavaleurmaximaleden_{eff}$ et la somme des nombres de solutions optimales pour ce n_{eff} maximal.