SAT - protein folding

Becker Robin Gilles - Bourgeois Noé

November 2022

Contents

1 Introduction

Ce projet vise à résoudre le problème de "protein folding", dans un contexte symbolique en 2 dimensions au moyen d'un solveur SAT.

cf énoncé: Trouver la structure d'une protéine est un problème fondamental en biologie. Dans ce projet, nous utilisons un modèle simple de protéines, qui groupe les acides aminés (AA) selon qu'ils sont hydrophobiques ou hydrophiles. Les acides aminés hydrophobiques s'attire. Une protéine est alors simplement représentée par une séquence de 1 (pour hydrophobique) et de 0 (pour hydrophile).

Le but est alors de trouver un agencement des 1 et 0 dans N^2 qui maximise le nombre d'appariements (des 1 qui sont voisins dans l'espace choisi). Une séquence d'AA peut être plongée dans N^2 de diverses manières. Le score d'un plongement est donné par le nombre de paires de points voisins (horizontalement ou verticalement, mais pas en diagonal), qui sont étiquetés par 1.
Il faut donc trouver une manière de plonger la séquence qui maximise ce score.

2 Tests de satisfiabilité

2.1 Données générales

- Soit s un caractère avec $s \in \{'1', '0'\}$.
- Soit S une séquence de caractères de longueur n. Chaque caractère $s \in S$ est présenté au solveur par son index i dans S.
- Soit M la matrice de plongement qui contiendra le résultat final.
- Soit j un emplacement dans la matrice M, défini par deux coordonnées x, y, respectivement le numéro de la colone et de la ligne correspondantes dans M.

2.2 Clauses relatives au placement légal d'un élément

Nous avons 4 clauses générales définissant la manière dont les éléments de S peuvent être disposés dans M.

2.2.1 La séquence est entièrement utilisée :

Chaque chaînon doit se retrouver dans la réponse, c'est à dire à une coordonée $j \in M$.
La formule CNF correspondante est celle-ci :

$$\bigwedge_{0 \leq i \leq n-1} \bigvee_{1 \leq j \leq |G|} X_{i,j}$$

2.2.2 Un élément maximum à chaque emplacement :

A chaque emplacement ne peut être assignée qu'une seule valeur, soit :

$$\bigwedge_{\substack{0 \leq i \leq n-1 \\ 0 \leq i' \leq n-1 \\ i \neq i' \\ 1 \leq j \leq |G|}} (\overline{X_{i,j}} \vee \overline{X_{i',j}})$$

2.2.3 Un emplacement maximum par élément :

Chaque chaînon ne peut se retrouver qu'à un seul emplacement $j \in M$, soit :

$$\bigwedge_{\substack{0 \leq i \leq n-1 \\ 1 \leq j,j' \leq |G|, \\ j \neq j'}} (\overline{X_{i,j}} \vee \overline{X_{i,j'}})$$

2.2.4 Maintient de l'ordre de la séquence :

Deux chaînons adjacents dans S donc d'index $i, i+1$ doivent se retrouver à deux emplacements $j, j' \in M$ adjacents, soit :

$$\bigwedge_{\substack{0 \leq i \leq n-1 \\ 1 \leq j,j' \leq |G| \\ j \neq j' \\ , j \; et \; j' \; sont \; adjacents}} (\overline{X_{i,j}} \vee X_{i+1,j'})$$

2.3 Clauses de cardinalité

cf énoncé: les méthodes de cardinalité card :
$classmethod\_atleast(lits, bound = 1, top_id = None, vpool = None, encoding = 1)$
$classmethod\_equals(lits, bound = 1, top_id = None, vpool = None, encoding = 1)$

permettent, étant donné une liste de litéraux 'lits' et une borne 'bound', de créer une formule en CNF (autrement dit, un ensemble de clauses) qui sera satisfaisable uniquement par des valuations telles que respectivement 'au moins' et 'exactement' 'bound' litéraux de lits sont vrais.

2.3.1 Un emplacement par élément exactement

Pour chaque élément, exactement un emplacement doit y être assigné. Après une conversation avec Anton Romanova, nous avons décidé de remplacer, dans le code, les clauses "Un élément maximum à chaque emplacement" et "La séquence est entièrement utilisée" par cette clause de cardinalité plus simple et plus efficace. Sa fomule CNF est la même que dans "La séquence est entièrement utilisée" à laquelle on ajoute la contrainte de cardinalité d'une seule variable à True dans chaque disjonction.

2.3.2 Comptage des contacts entre deux 1

Au moins 'bound' éléments '1' doivent être en contact.

Avec $(j_x, j_y), (j'_x, j'_y) \in N^2$ voisins si $(|j_x - j'_x|, |j_y - j'_y|) \in (0,1), (1,0)$.
et adjacency id l'adjacence des 2 éléments présentée sous la forme d'un identifiant entier au solveur,

$$\bigwedge_{\substack{0 \leq i \leq n-1 \\ 1 \leq j,j' \leq |G|, \\ j \neq j' \\ (|j_x - j'_x|, |j_y - j'_y|) \in (0,1),(1,0)}} (adjacency\_id \vee \overline{X_{i,j}} \vee \overline{X_{i,j'}}) \wedge (adjacency\_id \vee \overline{X_{i,j}}) \wedge (adjacency\_id \vee \overline{X_{i,j'}})$$

3 Code

Le code écrit en Python 3 pour répondre aux questions de l'énoncé est dans le fichier nommé `folder.py`.

3.1 Quantité de contacts estimée

get_contact_quantity_min_and_max(*seq*) calcule les scores minimum et maximum estimé d'une séquence. Dans la sequence non pliée, le nombre d'adjascences de uns détermine le score minimum de celle-ci quelle que soit la structure de son futur pliage. Le score maximum estimé est quant à lui calculé de 2 manière:

- comme deux '1' ne peuvent etre mis en contact par pliage que si ils sont séparés par des éléments en quantité paire, un scan de la séquence depuis chaque '1' avec un saut de 3 jusqu'à la fin de celle-ci permet de compter le nombre de contacts potentiels. À ce score, on ajoute le score minimum et on obtient un score maximum
- si la séquence est retranchée de ses zéros et spiralée, on obtient une compression optimale et une estimation rapide gràce à la formule suivante: avec n la quantité uns dans la séquence:

$$a(n) = 2 * n - \lceil (2 * \sqrt{n}) \rceil \tag{1}$$

  Que nous avons décider d'utiliser lors d'une conversation avec Anton Romanova car cette méthode donne souvent un score moindre que celui du scan. On prend alors le minimum des deux.

3.2 Dimensions de matrice de plongement optimal

get_matrix_dimensions retourne les dimensions de matrice pour le plongement optimal d'une séquence.

Nous n'avons pas encore trouvé de pattern permettant de les déterminer à l'avance, sans effectuer les tests empiriques. Nous avons donc décider de stocker chaque séquence testée dans une liste regroupant les séquences avec plongement optimal dans une matrice de mêmes dimensions, jusqu'à trouver un pattern.

3.3 Résultats

Notre programme a été capable de réaliser tous les tests sans erreurs ou mauvaises réponses mais avec quelques timeouts.

Instances avec solutions correctement repondues: 54 sur 54 tests realises
Nombre de timeouts: 0 — Nombre d'exceptions: 0

Instances sans solution correctement repondues: 44 sur 54 tests realises
Nombre de timeouts: 10 — Nombre d'exceptions: 0

Meilleurs scores correctement calcules: 44 sur 54 tests realises
Nombre de timeouts: 10 — Nombre d'exceptions: 0

3.4 Librairies utilisées

Les librairies utilisées dans les différents programmes sont les suivantes:

- pysat.solvers (Minisat22)
- pysat.card
- optparse
- numpy
- func_timeout

3.5 Recherche du meilleur score :

Le meilleur score pour une séquence est défini par le nombre maximum de contacts qu'il est possible de former avec. Nous devions implémenter la recherche du meilleur score de deux façons : incrémentale et dichotomique.
Comme les 4 premières clause ne dépendent pas de la borne, celles-ci ne sont calculées qu'une fois.

3.5.1 Recherche incrémentale

Nous définissons $lower\_bound = 0$ comme la valeur initiale à partir de laquelle nous allons chercher le score maximum. S'il n'est pas possible de trouver une solution au problème avec $lower\_bound$ contacts, la requête est déclarée impossible et nous renvoyons $None$. Sinon, il suffit de tester itérativement s'il est possible de former x contacts avec la séquence en appelant $get\_solution(S, x)$. A chaque itération, le $lower\_bound$ initial est incrémenté de 1 jusqu'à ce qu'il ne soit plus possible de trouver une solution.

3.5.2 Recherche dichotomique

Le principe est similaire à la recherche incrémentale sauf que x n'est pas incrémenté de 1 à chaque essai. A la place, on teste $get\_solution(S, x)$ où $x = (high\_bound + lower\_bound)//2$. Si une solution est trouvée, cela signifie que le score maximum est plus grand ou égal à x et nous recommençons le test avec $lower\_bound = x$. Sinon, le score maximum est inférieur à x et nous recommençons avec $high\_bound = x$.
Nous arrêtons ce procédé quand $high\_bound - lower\_bound = 1$ car nous savons alors que le score maximum est $lower\_bound$.

3.6 Addition d'options :

Les options de tests (-t), bound (-b), affichage de la solution (-p) et recherche incrémentale à la place de recherche dichotomique (-i) ont toutes été implémentées et sont fonctionnelles.

3.7 Github repository

https://github.com/nobourge/protein-structure-satisfiability

3.8 Sources

https://core.ac.uk/download/pdf/59219558.pdf p4

3.8.1 code repositories

https://github.com/hannah-aught/prototein-problem
https://github.com/angary/protein-folding-sat

3.8.2 sequence analysis

https://oeis.org/A123663 from Anton Romanova