Rapport de Projet

2I013 - Projet
Identification de motifs structuraux dans les protéines

Licence d'informatique et de mathématiques appliquées

Année universitaire 2018 - 2019

Encadrante : **Pr Mathilde Carpentier**
Groupe : Mr Vincent Fu
Mr Chengyu Yang
Mr Yuhao Liu

Table des matières

Première partie

Introduction

I.1 Protéines et Structures

Les protéines sont des molécules essentielles du vivant. Elles assurent une multitude de fonctions au sein de la cellule vivante notamment sur des fonctions enzymatiques, de régulation (modulation de l'activité), de signalisation (capteurs), de défense (anticorps), de stockage ainsi que du maintien de la structure et de la mobilité.

Les protéines sont formées à partir de plusieurs monomères de base. Ce sont ces fameux acides aminés qu'on appellera par la suite : les résidus. Au nombre de 20 pour les protéines, les différents enchaînements possibles de ces derniers dans la chaîne protéique permet de définir la protéine et donc en l'occurrence sa fonction à travers des propriétés physiques et chimiques. Une protéine naturelle contient entre 50 et 2000 résidus (Stryer, 1994).

Les 20 acides aminés de base sont quant à eux des molécules qui ne diffèrent uniquement à leur chaîne latérale où est placé un des 20 groupements chimiques. La partie commune de ces résidus est composée d'**un seul atome de carbone central** (C à 4 liaisons simples) lié à un groupe aminé ($-NH_2$), à un groupe carboxylique ($-COOH$) et à un atome d'hydrogène ($-H$) (voir FIGURE *1*).

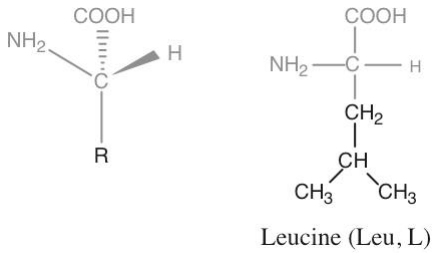

FIGURE *1* – *Gauche : Structure générique d'un acide aminé avec pour -R la chaîne latérale. Droite : Formule chimique semi-développée de la Leucine*

De plus, dans la chaîne protéique, les acides aminés sont liés 2 à 2 par le biais d'une liaison peptidique amide en reliant le groupe carboxylique du résidu en question et le groupe aminé du résidu suivant (voir FIGURE *2*).

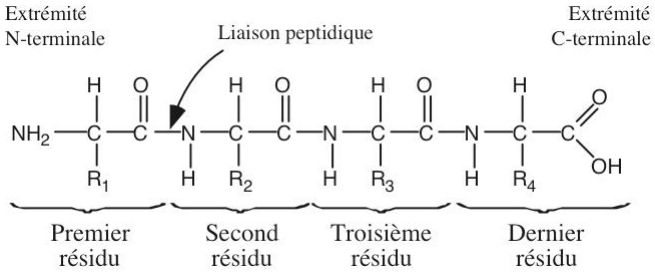

FIGURE *2* – *Un polypeptide formé de 4 acides aminés*

En raison de la rigidité[1] de la liaison peptidique amide, cette dernière forme une géométrie plane. Or, les différents agencements dans l'espace des protéines sont régies par les **rotations** possibles de l'atome de carbone central des résidus, par l'encombrement stérique des groupes d'atomes ainsi que par les interactions d'hydrogène, électrostatiques et de Van der Valls.

Tous ces phénomènes imposent des conformations plus ou moins stables aux résidus mais il faut savoir que la plupart des protéines adoptent une forme stable et unique puisque les chaînes latérales interagissent entre eux et également avec l'eau présent en grande quantité dans l'organisme chez la majorité des êtres vivants.

Ainsi, en se plaçant dans un espace euclidien, il est possible de calculer des **distances internes**[2] et de calculer l'**angle orienté** α associé à un atome de carbone central[3] (voir FIGURE *3*).

1. issue de la double liaison CO
2. distance entre 2 atomes centraux de carbone, ce sont des données dites relationnelles
3. calculé à partir de ce dernier, du C central du résidu précédent et des C centraux des 2 résidus suivants, ce sont des données dites non relationnelles

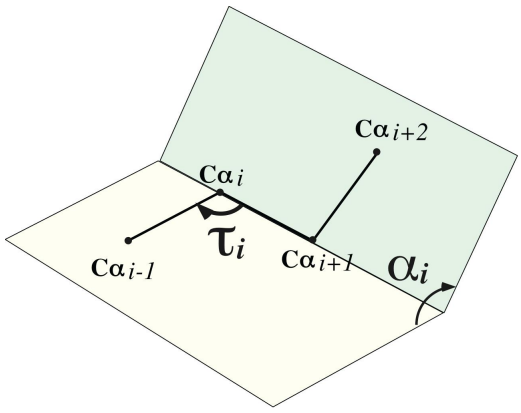

FIGURE 3 – *L'angle orienté* α

Ces données sont très importantes dans la suite du projet car ce sont elles qui permettent l'identification des motifs.

En collectant toutes ces données, il est alors possible d'appliquer l'algorithme KMRC en vue de trouver les similitudes des motifs et de les classifier.

I.2 Algorithme KMRC

L'algortihme KMRC développé par H. Soldano et coll. puis par M.-F. Sagot et coll. (Soldano et al., 1995 ; Sagot et al., et Jean et al., 1997) est issu de l'algorithme de recherche de motifs répétés identiques KMR (Karp, Miller, Rosenberg) (Karp et al., 1972) ayant pour but cette fois-ci de trouver des motifs répétés similaires. Décrivons cette méthode.

Sachant qu'une protéine est décrite par l'enchaînement des résidus, il est possible de décrire la protéine non plus avec les résidus mais avec une suite finie d'angles orientés α. Cette chaîne est alors appelé le « **texte** ». En classifiant ces valeurs dans des intervalles discrétisés [4] (exemple : de -180° à 180°, on découpe en plusieurs intervalles avec une **discrétisation** de 10°. 45° appartient alors à [40°, 50°[, -63° à [-70°, -60°[et etc...), il est alors possible d'effectuer des recherches de motifs.

L'ensemble des données (ou symboles [5]) est appelé l'**alphabet**. Les symboles sont classifiés dans des ensembles appelés **pavés**. La **dégénérescence symbolique** est le nombre maximal de pavés auquel peut appartenir un **symbole**. Contrairement à l'algorithme KMR où on cherche des motifs répétés strictement identiques, l'algorithme KMRC cherche des motifs répétés similaires en fonction des appartenances des symboles aux pavés. Par exemple, avec pour alphabet : $\{a, b, c\}$ et pour pavés : $\{a, c\}$, $\{a, b\}$, le 2-motif « aa » est similaire au 2-motif « ab » mais non identique. « ab » est alors une **instance** du motif « aa » et inversement. Dans tous les cas, « aa » et « ab » représentent ce 2-motif en question alors que « ba » et « ca » forment 2 2-motifs différents puisque b et c ne sont pas dans le même pavé.

Ainsi, un k-**motif** ne représente pas qu'une seule suite unique de symboles mais plusieurs et par suite est entièrement défini par sa longueur k ainsi que son **extension**. Un extension est l'ensemble des positions des instances du motif en question. Un motif est dit **maximal** si son extension n'est incluse dans aucune autre extension d'un motif différent de celui-ci et est dit répété si le cardinal de l'extension est strictement supérieur à 1. Le **quorum** est le nombre minimal de structures [6] que doit appartenir un motif.

Dans l'algorithme KMR, la construction des motifs répétés identiques est progressive. Autrement dit, les k-motifs permettent de construire des $2k$-motifs et ainsi de suite. Pour ce faire, il est nécessaire que le k-motif répété d'extension $\{i, j\}$ se relie à un second k-motif répété d'extension $\{i + k, j + k\}$ afin de former un $2k$-motif répété d'extension $\{i, j\}$. Si le second motif répété a une extension $\{i + k, l\}$ avec $l \neq i + k$, on obtient alors un $2k$-motif d'extension $\{i\}$ mais non répété. Il sera alors éliminé de l'algorithme où seuls les motifs répétés sont conservés. Ce

4. les voir comme des ensembles ayant un critère d'appartenance propre à chacun d'eux
5. les symboles sont des angles orientés dans notre cas
6. les structures sont des protéines dans notre cas

principe de construction est également utilisé dans l'algorithme KMRC. Cependant, il arrive que dans la recherche de motifs, on trouve des motifs non maximaux ou/et des motifs ne respectant pas la condition du quorum (exemple : un k-motif répété d'extension $\{i, j\}$ n'est pas maximal s'il existe un autre k-motif répété d'extension $\{i, j, l\}$). Il est alors nécessaire d'ajouter des étapes de filtrages dans l'algorithme puisque seuls les motifs maximaux suffisent à construire tous les motifs maximaux de taille supérieure.

Avec les méthodes de programmation et toutes les considérations ci-dessus, on arrive à trouver tous les motifs répétés similaires maximaux de longueur maximale avec en théorie une complexité en $\mathcal{O}(g^{2k_{max}} n k_{max} log k_{max})$ [7] avec g la dégénérescence symbolique, k_{max} la longueur maximale des motifs répétés et n la longueur du « texte ».

Sachant que le paramètre k_{max} dépend des structures étudiées et donc difficilement prédictible, seule la dégénérescence symbolique g paramétrable par l'utilisateur lors de la construction des pavés permet d'obtenir un temps d'exécution raisonnable de l'algorithme. En effet, les pavés sont à construire puisqu'ils sont fabriqués à partir des symboles. La façon dont les pavés sont construits n'impactera pas la capacité de l'algorithme à rechercher des motifs répétés similaires. Seulement, les motifs trouvés avec ces pavés seront différents des motifs trouvés à l'aide d'autres pavés. On adoptera alors la méthode de découpage (citée plus haut) avec une valeur de discrétisation pour la construction des pavés.

L'implémentation de l'algorithme étape par étape sera expliqué en détail par la suite.

I.3 Mise en pratique

Notre travail consiste à implémenter une version dérivée de l'algorithme KMRC qui est l'algorithme de Triades (thèse de M. Carpentier, atelier de bioinformatique de l'UFR des Sciences de la Vie de Sorbonne Université (ex-UPMC), 2005) en ajoutant en plus des symboles d'angles, des symboles de distances internes afin de construire des motifs suivant la méthode de construction progressive non plus de k-motifs à $2k$-motifs mais de k-motifs à k+1-motifs.

Nous avons également pour but d'implémenter les différentes fonctions d'extraction des données sur les protéines, de calcul pour obtenir les alphabets [8] ainsi que des fonctions de filtrages afin d'optimiser le temps d'exécution et d'obtenir que des motifs répétés maximaux. Enfin, il est possible qu'une **occurrence** [9] appartient à plusieurs extensions de motifs maximaux. Ces motifs représentent alors en réalité un seul et même motif. Il est alors également nécessaire de regrouper ces motifs en un seul motif en regroupant ces extensions en une seule extension à l'aide de fonctions de fusion.

Le langage de programmation utilisé est Python pour sa simplicité de programmation.

Les informations concernant les protéines notamment sur leur nom, l'enchaînement des atomes, des résidus ou encore les coordonnées en trois dimensions des atomes et etc... sont toutes stockés dans des fichiers .pdb qui n'est ni plus ni moins un fichier au format texte (.txt) (voir FIGURE *4*).

```
ATOM    497  N   ARG A  72      44.737  30.632   3.997  1.00 12.38           N
ATOM    498  CA  ARG A  72      44.288  31.427   2.872  1.00 13.76           C
ATOM    499  C   ARG A  72      42.793  31.282   2.665  1.00 14.77           C
ATOM    500  O   ARG A  72      42.096  32.314   2.682  1.00 16.05           O
ATOM    501  CB  ARG A  72      44.996  31.131   1.573  1.00 16.84           C
ATOM    502  CG  ARG A  72      46.416  31.680   1.485  1.00 18.10           C
ATOM    503  CD  ARG A  72      46.745  31.912   0.029  1.00 21.16           C
ATOM    504  NE  ARG A  72      47.992  32.620  -0.147  1.00 24.17           N
ATOM    505  CZ  ARG A  72      49.211  32.152   0.137  1.00 25.42           C
ATOM    506  NH1 ARG A  72      49.377  30.877   0.485  1.00 25.03           N
ATOM    507  NH2 ARG A  72      50.217  33.034   0.133  1.00 24.76           N
ATOM    508  N   GLU A  73      42.355  30.040   2.652  1.00 14.83           N
ATOM    509  CA  GLU A  73      40.947  29.681   2.503  1.00 15.19           C
ATOM    510  C   GLU A  73      40.055  30.230   3.588  1.00 13.93           C
ATOM    511  O   GLU A  73      38.980  30.765   3.307  1.00 15.93           O
ATOM    512  CB  GLU A  73      40.733  28.148   2.535  1.00 16.42           C
ATOM    513  CG  GLU A  73      40.968  27.451   1.215  1.00 20.74           C
ATOM    514  CD  GLU A  73      40.898  25.950   1.258  1.00 25.13           C
ATOM    515  OE1 GLU A  73      40.573  25.307   2.242  1.00 23.84           O
ATOM    516  OE2 GLU A  73      41.360  25.471   0.168  1.00 27.95           O
ATOM    517  N   ALA A  74      40.465  30.062   4.801  1.00 14.17           N
ATOM    518  CA  ALA A  74      39.778  30.512   6.001  1.00 13.62           C
ATOM    519  C   ALA A  74      39.605  32.024   5.967  1.00 14.79           C
ATOM    520  O   ALA A  74      38.502  32.518   6.292  1.00 15.88           O
ATOM    521  CB  ALA A  74      40.435  29.989   7.223  1.00 11.01           C
```

7. si on cherche des motifs de taille k, alors la complexité est en $\theta(g^{2k} n k log k)$
8. ensemble d'angles et ensemble de distances internes
9. position d'une instance

FIGURE 4 – *Une partie d'un fichier .pdb (une protéine) (ouvrable en .txt) où les données (atomes, résidus, positions et etc...) doivent être collectés*

Toutes les protéines existantes peuvent être mises sous format .pdb et ces derniers sont enregistrés dans une banque de données appelé **Protein Data Bank** mise à disposition sur internet (`http://www.rcsb.org`). Il est également possible de visualiser en trois dimensions la protéine sous format .pdb (voir FIGURE 5) ou même encore un motif répété (voir FIGURE 6) en respectant les règles de format .pdb [10] à l'aide du logiciel **PyMOL** (téléchargeable gratuitement sur `https://pymol.org`).

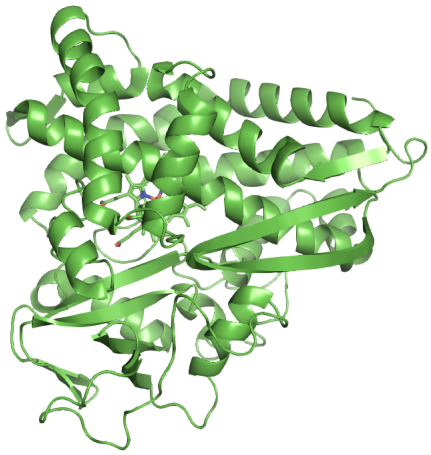

FIGURE 5 – *La représentation de la protéine 3CPP sur le logiciel PyMOL*

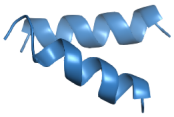

FIGURE 6 – *La représentation d'un motif d'extension de taille 2 issu de la protéine 3CPP sur le logiciel PyMOL*

10. qu'on détaillera par la suite

Avant de commencer la partie suivante, voici les définitions importantes pour la compréhension de l'algorithme :

symbole : une donnée étudiée (dans le cas relationnel, c'est une distance interne. Sinon, c'est un angle orienté)
alphabet : l'ensemble des symboles
« texte » : chaîne d'éléments issus de l'alphabet
instance : version d'un motif
motif : chaîne de symboles issue du « texte » dont il est entièrement définie par sa taille et son extension
extension : l'ensemble des occurrences
occurrence : position d'une instance
quorum : nombre minimal de structures (protéines dans notre cas) que doit appartenir un motif
pavé : ensemble de symboles classifiés selon des conditions
discrétisation : pas d'un pavé
dégénérescence symbolique : nombre maximal de pavés que peut appartenir un symbole
maximal : condition de non inclusion d'un extension à un autre extension

Deuxième partie

Conception et Analyse

II.4 Implémentation

Algorithme de Triades

Avant d'appliquer l'algorithme de Triades, il est nécessaire d'obtenir le « texte », c'est-à-dire la suite d'angles orientés α associés aux atomes centraux des résidus de la protéine étudiée, ainsi que les pavés.

D'après le Protein Data Bank Contents Guide (`http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html#ATOM`), les coordonnées des atomes se situent à la 7^e, 8^e et 9^e colonnes et les atomes de carbone centraux sont signalés avec des `CA` à la 3^e colonne (voir FIGURE 7).

```
ATOM    497  N   ARG A  72      44.737  30.632   3.997  1.00 12.38           N
ATOM    498  CA  ARG A  72      44.288  31.427   2.872  1.00 13.76           C  ****
ATOM    499  C   ARG A  72      42.793  31.282   2.665  1.00 14.77           C
ATOM    500  O   ARG A  72      42.096  32.314   2.682  1.00 16.05           O
ATOM    501  CB  ARG A  72      44.996  31.131   1.573  1.00 16.84           C
ATOM    502  CG  ARG A  72      46.416  31.680   1.485  1.00 18.10           C
ATOM    503  CD  ARG A  72      46.745  31.912   0.029  1.00 21.16           C
ATOM    504  NE  ARG A  72      47.992  32.620  -0.147  1.00 24.17           N
ATOM    505  CZ  ARG A  72      49.211  32.152   0.137  1.00 25.42           C
ATOM    506  NH1 ARG A  72      49.377  30.877   0.485  1.00 25.03           N
ATOM    507  NH2 ARG A  72      50.217  33.034   0.133  1.00 24.76           N
ATOM    508  N   GLU A  73      42.355  30.040   2.652  1.00 14.83           N
ATOM    509  CA  GLU A  73      40.947  29.681   2.503  1.00 15.19           C  ****
ATOM    510  C   GLU A  73      40.055  30.230   3.588  1.00 13.93           C
ATOM    511  O   GLU A  73      38.980  30.765   3.307  1.00 15.93           O
ATOM    512  CB  GLU A  73      40.733  28.148   2.535  1.00 16.42           C
ATOM    513  CG  GLU A  73      40.968  27.451   1.215  1.00 20.74           C
ATOM    514  CD  GLU A  73      40.898  25.950   1.258  1.00 25.13           C
ATOM    515  OE1 GLU A  73      40.573  25.307   2.242  1.00 23.84           O
ATOM    516  OE2 GLU A  73      41.360  25.471   0.168  1.00 27.95           O
ATOM    517  N   ALA A  74      40.465  30.062   4.801  1.00 14.17           N
ATOM    518  CA  ALA A  74      39.778  30.512   6.001  1.00 13.62           C  ****
ATOM    519  C   ALA A  74      39.605  32.024   5.967  1.00 14.79           C
ATOM    520  O   ALA A  74      38.502  32.518   6.292  1.00 15.88           O
ATOM    521  CB  ALA A  74      40.435  29.989   7.223  1.00 11.01           C
```

FIGURE 7 – *La* FIGURE *4 avec les lignes importantes mises en valeur par* ****

Ainsi, à l'aide de fonctions de lecture de fichier, on peut extraire les coordonnées des atomes de carbone centraux afin de calculer la suite d'angles orientés α.

Pour calculer l'angle orienté α_i associé à C_i à partir de 4 atomes de carbone centraux C_{i-1}, C_i, C_{i+1} et C_{i+2}, on doit d'abord calculer les vecteurs normaux $\overrightarrow{n_i}$ du plan formé par les vecteurs $\overrightarrow{C_iC_{i-1}}$ et $\overrightarrow{C_iC_{i+1}}$. Puis, on utilise la relation du produit scalaire :

$$\overrightarrow{n_i} \cdot \overrightarrow{n_{i+1}} = \|\overrightarrow{n_i}\|\|\overrightarrow{n_{i+1}}\| \cos \alpha_i \tag{1}$$

Attention, on rappelle qu'on cherche à calculer un angle **orienté**. En effet, (1) ne donne qu'un angle non orienté compris entre 0 et $\frac{\pi}{2}$. Si on considère que l'espace où on étudie est en base orthonormée directe, alors le sens de l'angle qu'on assigne arbitrairement est du même signe (ou du signe opposé, il n'y a pas d'importance sur le choix mais dès que le choix est fait, il faut alors conserver cette convention) que le $\det(\overrightarrow{n_i}, \overrightarrow{n_{i+1}}, \overrightarrow{OC_i})$ (ou du produit mixte $(\overrightarrow{n_i} \wedge \overrightarrow{n_{i+1}}) \cdot \overrightarrow{OC_i}$) avec $O(0, 0, 0)$ pour origine. On obtient alors notre « texte » en faisant l'opération sur l'ensemble des atomes de carbone centraux. **Attention, si une protéine a n résidus** [11] **, alors sa suite d'angles orientés α est de taille n - 3** [12].

Pour la construction des pavés, on se fixe une discrétisation et on applique la méthode de découpage citée précédemment sur $[s_{min}, s_{max}]$ (du symbole de valeur minimale au symbole de valeur maximale) en classifiant les angles dans les ensembles qui leurs correspondent. Cependant, il faut aussi prendre en compte la dégénérescence symbolique qu'on donne en entrée également. En effet, après avoir construit les pavés avec seulement la méthode de découpage, ces pavés sont en principe par défaut de dégénérescence 1. Il faut alors regrouper ces pavés circulairement en de nouveaux pavés beaucoup plus gros selon la dégénérescence fixée (exemple : si on se donne une dégénérescence 2 et on a 4 pavés A, B, C et D alors, on obtient comme nouveaux pavés : $A \cup B$, $B \cup C$, $C \cup D$ et $D \cup A$. Si la dégénérescence valait 3, on aurait eu comme pavés : $A \cup B \cup C$, $B \cup C \cup D$, $C \cup D \cup A$ et $D \cup A \cup B$).

La façon dont on a regroupé les pavés n'est en réalité pas importante. On aurait pu prendre qu'un seul symbole et de l'intégrer dans plusieurs pavés pour respecter la dégénérescence ou bien uniquement former qu'une seule réunion

11. chaque résidu n'a un seul atome de carbone central
12. calculs impossibles sur le premier résidu et les 2 derniers résidus : voir FIGURE *3*

de pavés et de laisser les autres pavés tels quels mais nous avons pris cette manière de regroupement pour plus de généralité. Ainsi, pour obtenir plusieurs pavés, **il est préférable de choisir une discrétisation comprise entre 10 et 40 pour les angles orientés** [13].
Puis, afin d'obtenir un temps d'exécution raisonnable, nous nous imposons **des dégénérescences inférieures ou égales à 5**. Passons à présent à l'algorithme.

Voici l'algorithme expliqué schématiquement à quelques détails près en plusieurs étapes (l'alphabet relationnel représente l'ensemble des distances internes et son utilisation sera expliqué ci-dessous à l'étape 2) :

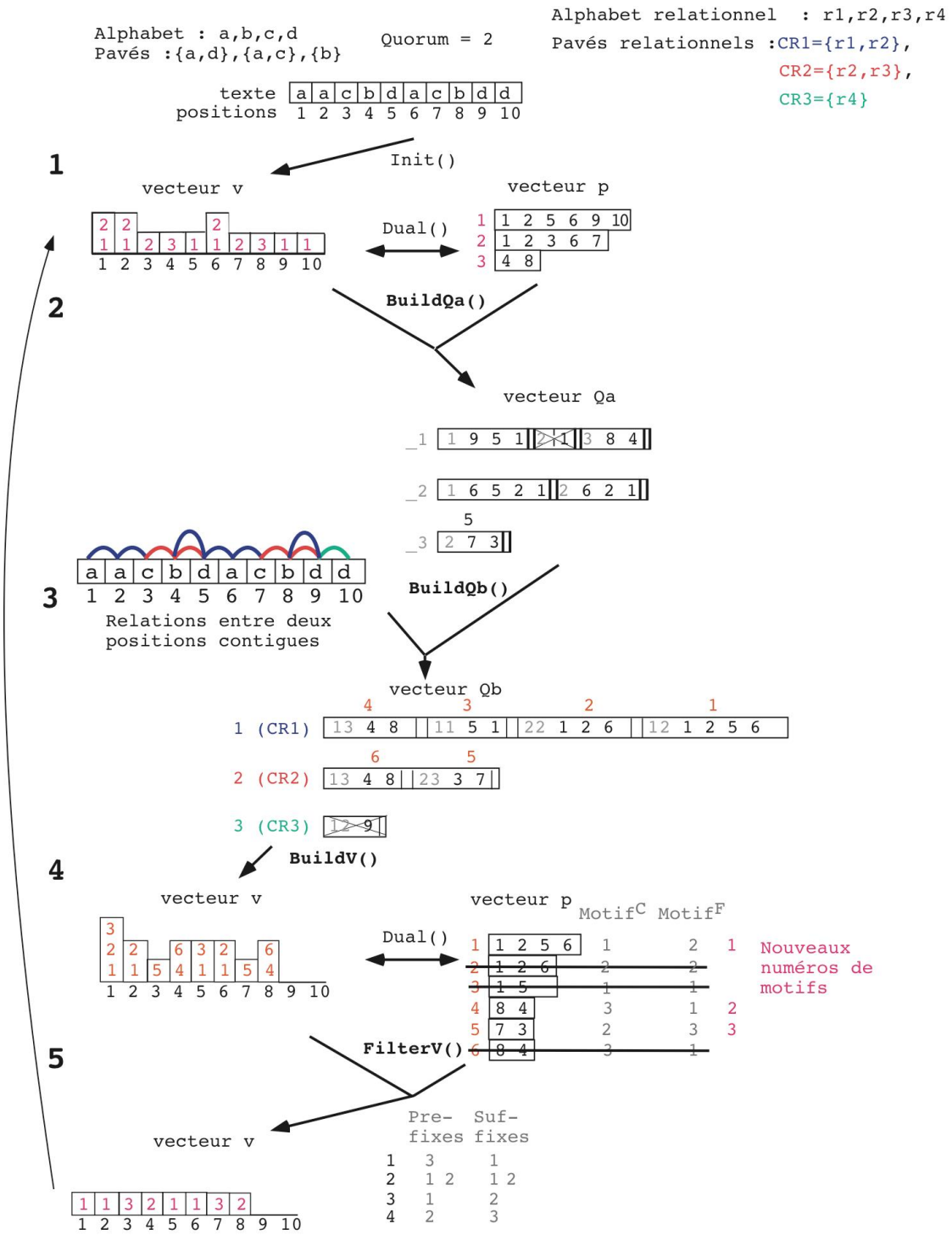

FIGURE 8 – *L'algorithme de Triades avec dans notre cas* `a`, `b`, `c` *et* `d` *des angles et* `r1`, `r2`, `r3` *et* `r4` *des distances internes*

13. la valeur de discrétisation est arbitraire

Etape 1 : On construit le vecteur v avec la fonction `Init(texte, paves)`.
La fonction `Init` parcourt le texte de gauche à droite et pour chaque symbole détermine les positions des pavés qui contient ce symbole puis les placent dans un ensemble et ces ensembles sont ensuite placés dans une liste. Par suite, on utilise le vecteur v pour construire son dual p sous forme de liste de listes (liste de piles en réalité [14]). La taille de p vaut en principe le nombre de pavés.

Etape 2 : On construit le vecteur Qa qui est également une liste de piles similaire à v avec la fonction `BuildQa(v, p, pas)`.
La fonction `BuildQa` va parcourir chaque pile de p. Pour les éléments p_i de chaque pile, on cherche l'ensemble $v[p_i + 1]$ puis on ajoute p_i aux $v[p_i + 1]$ ème piles de Qa (exemple : l'élément 5 dans p est placé dans les piles 1 et 2 de Qa car à la position $5 + 1$ de v, il y a les indices 1 et 2. L'élément 10 quant à lui est éliminé puisqu'il n'y a pas de position 11). A chaque fin de parcours d'une pile, on ajoute un séparateur [15] (un nombre négatif -1 par exemple) dans toutes les piles de Qa pour finir la construction d'un motif dans chaque pile.
En effet, en fonction de la pile de provenance de p_i dans p et dans quelles piles de Qa est inséré ce dernier, on peut connaître le motif ayant p_i dans son extension (exemple : l'élément 5 provient de la pile 1 de p et est inséré dans les piles 1 et 2 de Qa, on a alors les motifs « 11 » et « 12 » [16].
Le $+ 1$ correspond en fait au `pas` de construction de motifs. Au 1^e tour (la grande flèche sur la FIGURE précédente indique une boucle), ce pas vaut 1 car au commencement de l'algorithme, nous avons seulement des 1-motifs [17] et pour construire les 2-motifs, on va alors relier les 1-motifs répétés de positions (d'extension) $\{i, j, k, ...\}$ [18] aux 1-motifs répétés d'extension $\{i+1, j+1, k+1, ...\}$ d'où le $+ 1$. Si on avait déjà effectué un tour de boucle, le vecteur p représenterait alors tous les 2-motifs répétés et il faudrait alors relier ces motifs répétés d'extension $\{i, j, k, ...\}$ aux motifs répétés d'extension $\{i + 2, j + 2, k + 2, ...\}$ avec un pas de 2 pour former des 4-motifs répétés. Puis le tour suivant, les 8-motifs répétés sont fabriqués avec un pas de 4 et ainsi de suite. Ainsi, le pas correspond à une puissance de 2 dans le cas de la méthode de construction de k à $2k$-motifs.
Or, dans la section **I.3 Mise en pratique**, nous avons dit qu'on allons utiliser une méthode de construction de k à k+1-motif. En effet, en réalité, dans l'algorithme de Triades, on utilise en plus de l'alphabet initial un autre type d'alphabet qui est l'**alphabet relationnel**. Cela implique l'étude d'un second « texte » dit **relationnel** [19] (c'est une suite de distances internes). Il faut donc encore faire d'autres opérations notamment calculer l'appartenance des symboles relationnels dans les **pavés relationnels** et d'intégrer ces caractéristiques dans les motifs où pour l'instant seuls les angles orientés sont intégrés (d'où la fonction `BuildQb` expliqué à l'**étape 3** où son fonctionnement est dans le même qu'esprit que `BuildQa`).
Or, si on utilise la méthode de construction k à $2k$-motif, cela impliquerait de calculer les relations de distances entre chaque position du premier k-motif et chaque position du second k-motif, soit k^2 relations à calculer. La fonction `BuildQb` serait alors plus compliquée. Pour optimiser l'algorithme et éviter autant de calculs, on procède la construction de motif par chevauchement de 1 (la méthode de construction de k à k+1-motif) (voir FIGURE *9*).

14. on aurait pu mettre sous forme d'une liste d'ensembles puisque les positions vont composer les extensions mais cela n'a pas d'importance
15. dans le schéma de l'algorithme, les séparateurs sont représentés par ||
16. voir les informations en grise dans le schéma de l'algorithme au niveau de `Qa`
17. les symboles sont des 1-motifs non répétés et les piles de p sont en fait des 1-motifs répétés
18. positions dans le « texte »
19. car une distance est calculée à partir de 2 atomes de carbone centraux

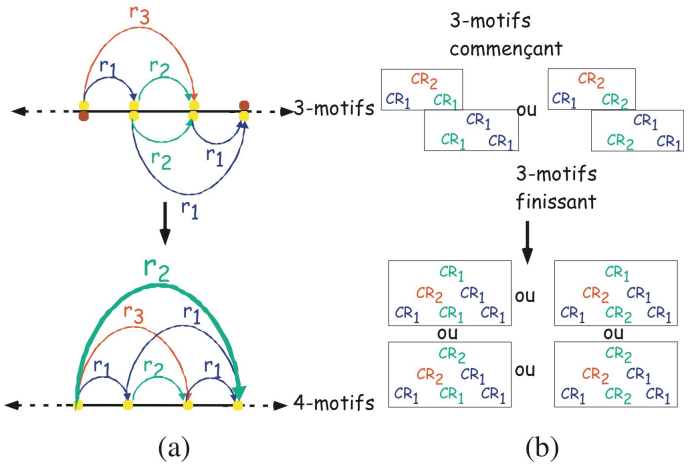

FIGURE *9 – Construction de motif par chevauchement de 1*

De cette manière, il suffit uniquement de calculer la distance (la relation) entre la première position et la dernière position des différents motifs dans la suite d'atome de carbone centraux (voir FIGURE *9*) sauf pour les 1-motifs où il suffit de calculer les relations entre les positions et les positions suivantes (voir FIGURE *10*).

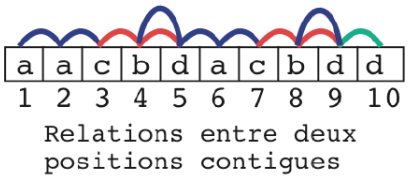

FIGURE *10 – Les relations au premier tour de l'algorithme*

Le pas alors, n'avancera plus en une puissance de 2 mais par incrémentation de 1 à chaque tour de boucle (puisqu'on utilise la construction par chevauchement de 1).
Dans le schéma de l'algorithme, on observe que le motif « 21 » est éliminé de la pile car il n'est pas répété. Il suffit alors d'ajouter un compteur pour chaque pile de Qa et puis, pour chaque fin de parcours de piles de v, avant de placer les séparateurs, on vérifie si chaque partie des piles après séparateur précédent est de taille strictement supérieur à 1. Sinon, on dépile les parties non respectés [20]. Après cela, on remet les compteurs à 0 et on reprend la suite.
En outre, un motif ne respectant pas le quorum est également éliminé. En effet, depuis le début, on ne parle que de « la » protéine étudiée et de son « texte » mais en réalité, on peut trouver des motifs similaires sur plusieurs protéines différentes. D'où le critère de quorum qui fixe le nombre minimum de protéines différentes que doit apparaître un motif. Pour cela, on crée tous les « textes » associés à chaque protéine puis on les relie bout à bout en un seul long « texte » avec des marqueurs de fin afin de délimiter les protéines. Il suffit alors ensuite de regarder les éléments d'extension pour vérifier la condition du quorum (là aussi, on utilise des compteurs). Cependant, nous avons décidé d'implémenter la condition du quorum pas ici mais après la fin de l'algorithme de Triades car il n'y a pas de sens de parler du quorum si on a qu'une seule protéine à étudier et c'était le cas au début de notre projet.

Etape 2 bis 1 : Avant de passer à l'étape 3, on construit un vecteur v' similaire au vecteur v (une liste d'ensemble de positions des pavés) mais cette fois-ci, elle est formée non plus avec `(texte, pavés)` mais avec le « **texte** » **relationnel** ainsi qu'avec les **pavés relationnels**. Les pavés relationnels sont construits sur $[d_{min}, d_{max}]$ (de la distance interne minimale au distance interne maximale) de la même manière qu'avec les angles orientés sauf qu'ici ce sont des distances internes qui ne fluctuent que très peu (au dixième près). **Nous nous fixions alors une discrétisation relationnelle comprise entre 0,05 et 0,5 pour les distances internes** afin d'obtenir pas mal de pavés.
Le « texte » relationnel correspond à la suite des distances euclidiennes entre les atomes centraux $C_i(x_i, y_i, z_i)$ et les atomes centraux $C_{i+pas}(x_{i+pas}, y_{i+pas}, z_{i+pas})$ (le pas correspond à l'argument de l'**étape 2** toujours en raison de la construction de k à k+1-motifs). Les coordonnées quant à eux sont obtenues tout

20. le motif est barré sur le schéma de l'algorithme

au début, avant l'application de l'algorithme, lorsque nous avons parlé d'extraction de données des fichiers .pdb avec les fonctions de lecture [21]. Dans l'exemple de la FIGURE *10*, le vecteur v' vaudrait alors `[{1}, {1}, {2}, {1, 2}, {1}, {1}, {2}, {1, 2}, {3}]` [22].

Etape 2 bis 2 : Lors de nos tests, nous avons remarqué que les piles obtenues à l'**étape 3** étaient beaucoup trop grandes si la dégénérescence était élevée. Cela impactait assez grandement le temps d'exécution de l'algorithme et parfois pouvait créer des bugs [23]. En réalité, le vecteur Qa contenait beaucoup trop de motifs non maximaux. Il fallait les éliminer puisqu'ils sont ne sont pas nécessaires à la construction des motifs maximaux. Nous avons donc programmé une fonction nommée `FiltreQa(Qa)`.
`FiltreQa` va d'abord transformer les piles de Qa en piles d'ensembles de positions [24]. Par suite, grâce aux manipulations possibles sur les ensembles (notamment la vérification d'inclusion), on élimine les extensions de chaque pile, qui sont incluses dans au moins une extension de cette même pile (exemple : la pile d'extensions `[{2, 1}, {1, 2, 3}, {1, 3, 2, 4}]`, en filtrant donne `[{1, 3, 2, 4}]`). Puis, on élimine les extensions qui sont incluses dans au moins une extension des autres piles. (exemple : dans l'exemple précédent, supposons que c'était la pile 1, si j'ai une pile 2 qui vaut `[{3, 7}, {1, 3, 2, 4, 5}]`, alors après deuxième étape de filtrage, la pile 1 vaut `[]` et la pile 2 vaut `[{3, 7}, {1, 3, 2, 4, 5}]`). Après cela, il suffit alors de retransformer ces piles sous la même forme initiale que Qa, c'est-à-dire en des piles de positions avec des séparateurs (la pile 2 redevient : `[3, 7, -1, 1, 3, 2, 4, 5]`). En outre, si on avait plusieurs extensions identiques, il suffit d'en garder une seule extension de n'importe quelle pile.

Etape 3 : On construit le vecteur Qb avec la fonction `BuildQb(`v'`, Qa)`.
La fonction `BuildQb` fonctionne exactement comme `BuildQa` sauf que ses arguments sont différents et qu'il n'y a pas de `pas`. En effet, on regarde directement les ensembles $v'[p_i]$ pour savoir où placer p_i dans les pavés. Cependant, lorsque p_i est un séparateur, on ajoute le séparateur à toutes les piles. Les motifs non répétés sont également éliminés.
Ici, on pourrait obtenir des motifs non maximaux et donc on pouvait potentiellement filtrer encore une fois avec la même fonction de filtrage `FiltreQa`. On a préféré de ne pas filtrer car le filtrage de Qa à l'étape précédente était suffisant pour réduire drastiquement les motifs non maximaux qu'on pouvait avoir ici. On estime une élimination en moyenne de 95 % des motifs non maximaux grâce à l'étape précédente. Le reste des 5 % sera alors éliminé à l'**étape 5**.

Etape 4 : Dans le schéma de l'algorithme, le vecteur v est fabriqué à partir de Qb pour ensuite construire le vecteur p, son dual [25]. Or, il n'était pas nécessaire de passer v pour construire p. On peut directement calculer p à partir de Qb.
On a alors implémenté une fonction `BuildP(Qb)` qui renvoie p. Elle parcourt chaque pile de Qb. Les éléments de Qb sont dépilés et placés dans une pile de p et lorsqu'on rencontre un séparateur, on crée une nouvelle pile et les éléments suivants sont à placés dans cette nouvelle pile et ainsi de suite.

Etape 5 : Il ne reste plus qu'à éliminer les motifs non maximaux de p à l'aide de la fonction de filtrage `FilterP(p)` (sur le schéma de l'algorithme, c'est `FilterV`).
La fonction va d'abord transformer les piles de p en plusieurs ensembles puis vérifier les inclusions de chaque ensemble en éliminant ceux qui sont inclus dans au moins un autre ensemble. En éliminant les motifs non maximaux dans p, les indices des piles changent [26].

Etape « grande flèche » : Enfin, on recommence un tour de boucle (on refait encore toutes les étapes depuis le début) pour reconstruire des motifs répétés agrandis de 1 symbole. Le vecteur v de l'**étape 1** est alors construit à partir du dual de p de l'**étape 5** [27]. On effectue autant de fois moins 1 tours de boucle que de la taille des motifs souhaitée [28].

21. voir tout au début de la section **II.4.1 Algorithme de Triades**
22. Le vecteur est de taille 9 puisque le pas vaut 1. Si le pas vaut 2, alors le vecteur serait de taille 8 et etc...
23. débordement de mémoire
24. afin d'éliminer les séparateurs et de repérer plus facilement les extension
25. c'est le résultat final en sortie d'algorithme si on est au dernier tour de boucle après filtrage de l'**étape 5**, les piles sont les extensions des motifs
26. sur le schéma de l'algorithme, les réindexations sont en rouge à droite
27. ce vecteur p correspond aussi au vecteur p de l'**étape 1** suivant l'**étape 5**
28. obtenir un k-motif nécessite k - 1 tours de boucle

Etape de fin : Comme dit précédemment, après la fin de toutes les étapes précédentes et uniquement dans le cas où on travaille sur plusieurs protéines, on vérifie la condition du quorum.
Le « texte » (non relationnel) est dans ce cas une chaîne formée par les « textes » de chaque protéine avec entre eux des marqueurs de fin (exemple : on a 3 protéines de texte A de taille 3, de texte B de taille 4 et de texte C de taille 5 respectivement alors le texte étudié au début de l'algorithme de Triades est A|B|C) (les | représentent des marqueurs de fin).
En observant les occurrences [29], on peut alors vérifier la condition du quorum. (exemple : si le quorum vaut 2, alors il faut qu'il existe une occurrence dans au moins 2 textes de protéines (A et B ou A et C ou B et C ou A, B et C). Dans l'exemple précédent, un 2-motif d'extension $\{2, 4\}$ vérifie le quorum. Un 2-motif d'extension $\{4, 5\}$ ne vérifie pas le quorum. Par contre, un 2-motif d'extension $\{4, 5, 1\}$ vérifie à nouveau le quorum).
Cependant, un autre problème survient. Lors de l'application de l'algorithme de Triades, ce dernier ne prend pas en compte les marqueurs de fin dans la recherche des occurrences. Cela arrive alors qu'une instance déborde sur 2 protéines différentes (Dans l'application de Triades et avec les 2 exemples précédents, on pourrait obtenir un 4-motif d'extension {2, 4}. Or, ce dernier en réalité ne vérifie pas le quorum puisque l'instance en position 2 du motif déborde sur 2 textes).
Il faut alors filtrer encore une fois en éliminant les occurrences qui débordent sur 2 protéines avant de vérifier la condition du quorum (Le 4-motif était en fait d'extension {4} et donc devenu non répété. Il faut en plus l'éliminer).
Nous avons alors programmé une fonction qui filtre les débordements, élimine les motifs non répétés puis vérifie le quorum (les 3 à fois) nommée `cut(p, marqueurs, k, quorum)` avec `marqueurs` la liste des positions des marqueurs.

Ainsi, on a réussi à trouver les extensions des motifs répétés maximaux sur un seul et même protéine ou sur plusieurs protéines [30]. Il ne reste plus qu'à présent à visualiser les résultats en 3 dimensions.

Fusion des motifs

Nous allons à présent parler d'une dernière fonction importante que nous avons nommé `fusion_motifs_similaires`. En effet, nous avons parlé des motifs non maximaux, c'est-à-dire des motifs où leurs extensions sont incluses dans un autre extension d'un motif.

Cependant, il arrive aussi qu'il y est des inclusions partielles des extensions. Autrement dit, **lorsque l'intersection d'un extension et d'un autre extension est non vide**. Dans ce cas, il faut alors regrouper ces extensions dans un seul motif ou pas selon si un critère est vérifié. En effet, selon l'inclusion partielle plus où moins importante, on peut décider de fusionner ou pas. Le critère qu'on va utiliser est la **distance de Jaccard** $J_{A,B}$ [31], une valeur comprise entre 0 et 1 et est définie par la formule suivante :

$$J_{A,B} = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

Plus cette valeur est proche de 0, plus cela signifie qu'il y a une inclusion partielle importante entre A et B et inversement. Ainsi, en fixant un `seuil`, on décide des fusions (exemple : soient s le seuil qu'on définit, un extension A et un extension B, si $J_{A,B} \leq s$, on fusionne A et B. Sinon, on les laisse tels quels). La fonction de fusion prend donc en argument `p` et `seuil` et est appliquée après l'**étape de fin**.

Visualisation des résultats

Pour pouvoir visualiser les résultats en 3 dimensions, il faut obtenir nos motifs au format .pdb.
Sachant que l'algorithme précédent ne donne que les positions des instances [32] dans les protéines, il faut alors utiliser les fonctions d'écritures pour écrire nos fichiers .pdb. Or, avant d'obtenir notre texte avant l'application de l'algorithme de Triades, nous avons préalablement mis en mémoire le contenu des fichiers .pdb des protéines dans **des dictionnaires** (voir FIGURE *11*).

29. une occurrence est une position d'une instance
30. selon le cas étudié
31. A et B sont des ensembles
32. positions des atomes de carbone centraux

```
{62: ['ATOM    497  N   ARG A  72      44.737  30.632   3.997  1.00 12.38           N  \n',
  'ATOM    498  CA  ARG A  72      44.288  31.427   2.872  1.00 13.76           C  \n',
  'ATOM    499  C   ARG A  72      42.793  31.282   2.665  1.00 14.77           C  \n',
  'ATOM    500  O   ARG A  72      42.096  32.314   2.682  1.00 16.05           O  \n',
  'ATOM    501  CB  ARG A  72      44.996  31.131   1.573  1.00 16.84           C  \n',
  'ATOM    502  CG  ARG A  72      46.416  31.680   1.485  1.00 18.10           C  \n',
  'ATOM    503  CD  ARG A  72      46.745  31.912   0.029  1.00 21.16           C  \n',
  'ATOM    504  NE  ARG A  72      47.992  32.620  -0.147  1.00 24.17           N  \n',
  'ATOM    505  CZ  ARG A  72      49.211  32.152   0.137  1.00 25.42           C  \n',
  'ATOM    506  NH1 ARG A  72      49.377  30.877   0.485  1.00 25.03           N  \n',
  'ATOM    507  NH2 ARG A  72      50.217  33.034   0.133  1.00 24.76           N  \n'],
 63: ['ATOM    508  N   GLU A  73      42.355  30.040   2.652  1.00 14.83           N  \n',
  'ATOM    509  CA  GLU A  73      40.947  29.681   2.503  1.00 15.19           C  \n',
  'ATOM    510  C   GLU A  73      40.055  30.230   3.588  1.00 13.93           C  \n',
  'ATOM    511  O   GLU A  73      38.980  30.765   3.307  1.00 15.93           O  \n',
  'ATOM    512  CB  GLU A  73      40.733  28.148   2.535  1.00 16.42           C  \n',
  'ATOM    513  CG  GLU A  73      40.968  27.451   1.215  1.00 20.74           C  \n',
  'ATOM    514  CD  GLU A  73      40.898  25.950   1.258  1.00 25.13           C  \n',
  'ATOM    515  OE1 GLU A  73      40.573  25.307   2.242  1.00 23.84           O  \n',
  'ATOM    516  OE2 GLU A  73      41.360  25.471   0.168  1.00 27.95           O  \n'],
 64: ['ATOM    517  N   ALA A  74      40.465  30.062   4.801  1.00 14.17           N  \n',
  'ATOM    518  CA  ALA A  74      39.778  30.512   6.001  1.00 13.62           C  \n',
  'ATOM    519  C   ALA A  74      39.605  32.024   5.967  1.00 14.79           C  \n',
  'ATOM    520  O   ALA A  74      38.502  32.518   6.292  1.00 15.88           O  \n',
  'ATOM    521  CB  ALA A  74      40.435  29.989   7.223  1.00 11.01           C  \n']}
```

FIGURE *11* – *La* FIGURE *4 en dictionnaire (62ème, 63ème et 64ème d'atome de carbone central)*

A partir des résultats de la section précédente, il suffit alors de sélectionner (de *pick*) dans l'ordre croissant[33] les portions qui nous intéressent et de les mettre dans un nouveau fichier .pdb. Attention, les motifs sont calculés à partir des atomes de carbone centraux dans l'algorithme de Triades mais ici, **il faut prendre tout le résidu associé à l'atome de carbone central en question pour bien visualiser les motifs**.

Par contre, il y a quelques règles d'écriture à respecter. Un fichier .pdb commence toujours par un `HEADER` (titre, nom de la protéine)[34] et se termine toujours par un `END`. Les instances doivent également être séparés par des `TER`. En respectant ces règles, on devrait obtenir à un fichier .pdb de la forme du FIGURE *11*.

```
HEADER    OXIDOREDUCTASE(OXYGENASE)                05-JUL-89   3CPP
ATOM    497  N   ARG A  72      44.737  30.632   3.997  1.00 12.38           N
ATOM    498  CA  ARG A  72      44.288  31.427   2.872  1.00 13.76           C
ATOM    499  C   ARG A  72      42.793  31.282   2.665  1.00 14.77           C
ATOM    500  O   ARG A  72      42.096  32.314   2.682  1.00 16.05           O
ATOM    501  CB  ARG A  72      44.996  31.131   1.573  1.00 16.84           C
ATOM    502  CG  ARG A  72      46.416  31.680   1.485  1.00 18.10           C
ATOM    503  CD  ARG A  72      46.745  31.912   0.029  1.00 21.16           C
ATOM    504  NE  ARG A  72      47.992  32.620  -0.147  1.00 24.17           N
ATOM    505  CZ  ARG A  72      49.211  32.152   0.137  1.00 25.42           C
ATOM    506  NH1 ARG A  72      49.377  30.877   0.485  1.00 25.03           N
ATOM    507  NH2 ARG A  72      50.217  33.034   0.133  1.00 24.76           N
ATOM    508  N   GLU A  73      42.355  30.040   2.652  1.00 14.83           N
ATOM    509  CA  GLU A  73      40.947  29.681   2.503  1.00 15.19           C
ATOM    510  C   GLU A  73      40.055  30.230   3.588  1.00 13.93           C
ATOM    511  O   GLU A  73      38.980  30.765   3.307  1.00 15.93           O
ATOM    512  CB  GLU A  73      40.733  28.148   2.535  1.00 16.42           C
ATOM    513  CG  GLU A  73      40.968  27.451   1.215  1.00 20.74           C
ATOM    514  CD  GLU A  73      40.898  25.950   1.258  1.00 25.13           C
ATOM    515  OE1 GLU A  73      40.573  25.307   2.242  1.00 23.84           O
ATOM    516  OE2 GLU A  73      41.360  25.471   0.168  1.00 27.95           O
TER
ATOM    508  N   GLU A  73      42.355  30.040   2.652  1.00 14.83           N
ATOM    509  CA  GLU A  73      40.947  29.681   2.503  1.00 15.19           C
ATOM    510  C   GLU A  73      40.055  30.230   3.588  1.00 13.93           C
ATOM    511  O   GLU A  73      38.980  30.765   3.307  1.00 15.93           O
ATOM    512  CB  GLU A  73      40.733  28.148   2.535  1.00 16.42           C
ATOM    513  CG  GLU A  73      40.968  27.451   1.215  1.00 20.74           C
ATOM    514  CD  GLU A  73      40.898  25.950   1.258  1.00 25.13           C
ATOM    515  OE1 GLU A  73      40.573  25.307   2.242  1.00 23.84           O
ATOM    516  OE2 GLU A  73      41.360  25.471   0.168  1.00 27.95           O
ATOM    517  N   ALA A  74      40.465  30.062   4.801  1.00 14.17           N
ATOM    518  CA  ALA A  74      39.778  30.512   6.001  1.00 13.62           C
ATOM    519  C   ALA A  74      39.605  32.024   5.967  1.00 14.79           C
ATOM    520  O   ALA A  74      38.502  32.518   6.292  1.00 15.88           O
ATOM    521  CB  ALA A  74      40.435  29.989   7.223  1.00 11.01           C
END
```

FIGURE *12* – *Un 2-motif d'extension* $\{62, 63\}$ *issu de la protéine 3CPP*

33. pour éviter des bugs car il y a des numérotages croissants sur les atomes et qu'il faut respecter
34. un motif ayant des instances dans plusieurs protéines n'est pas obligé d'avoir un `HEADER`

II.5 Résultats et Analyses

Maintenant, nous avons nos motifs sous format .pdb, on peut alors les visualiser sur **PyMOL**.
Voici les résultats sur l'étude d'une protéine (voir FIGURE *5*, FIGURE *6* et FIGURE *13*) :

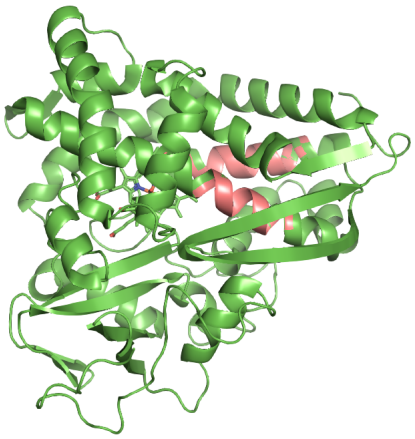

FIGURE *13 – Superposition de des figures 5 et 6*

Voici les résultats sur l'étude de plusieurs protéines :

FIGURE *14 – Représentation de 4 protéines 1GJM, 3CPP, 3ZWU et 5QRV*

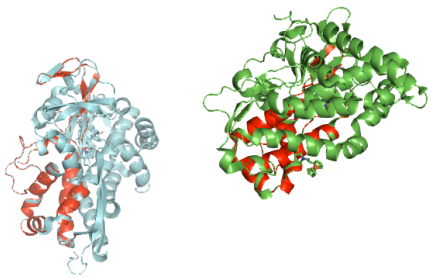

FIGURE *15 – Superposition d'un motif vérifiant le quorum à 1 et le quorum à 2*

Afin de vérifier l'efficacité de la fonction de fusion, nous avons implémenté des fonctions de traçage de courbe. Cette courbe représente alors le nombre de motifs en fonction du seuil de distance de Jaccard. Voici les courbes pour les protéines *3CPP*, *3ZWU*, *1GJM* avec une dégénérescence de 2, une discrétisation des angles de 10° et une discrétisation des distances de 0.2 :

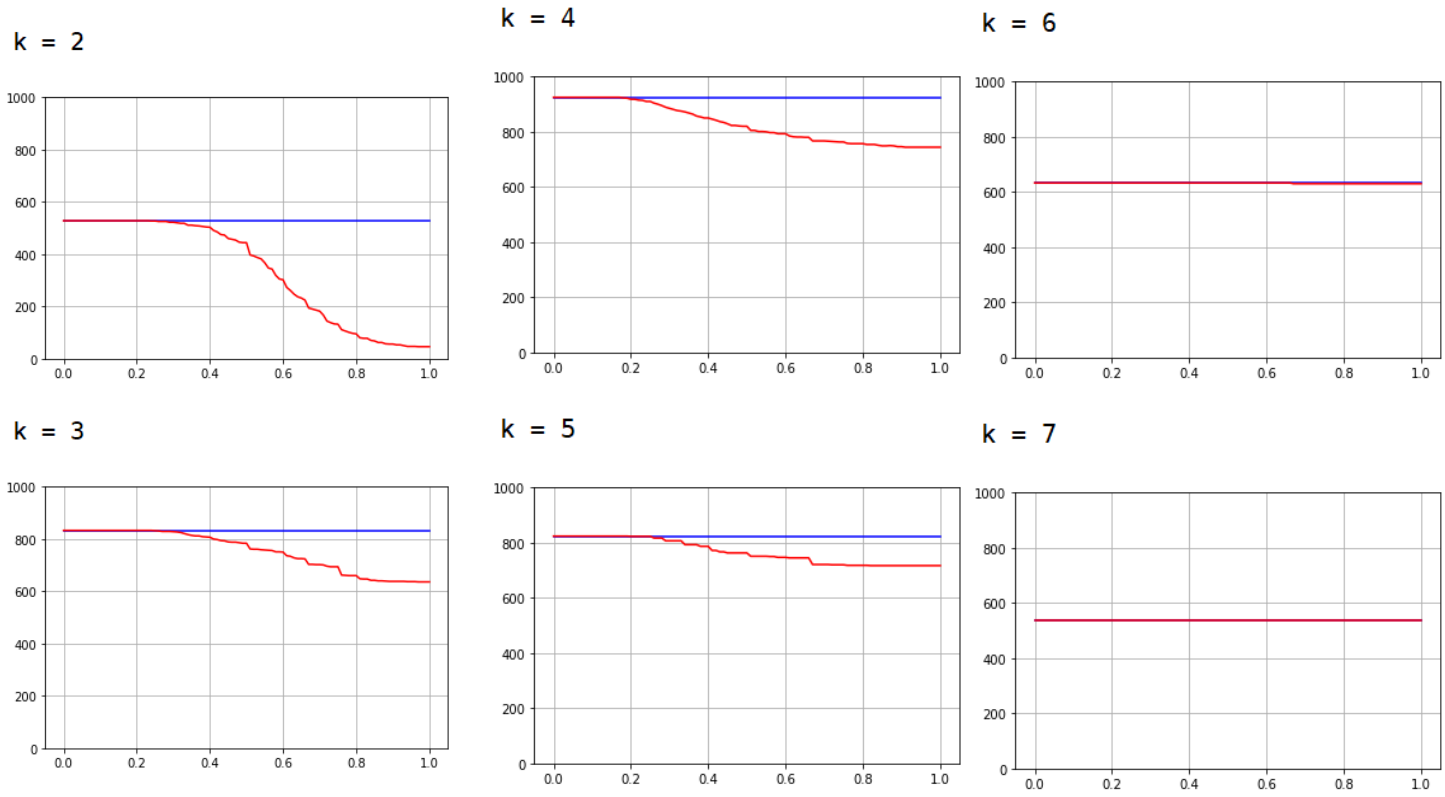

FIGURE *16 – évolution des courbes du nombre de motifs en fonction du seuil avec pour étude les protéines 3CPP, 3ZWU, 1GJM et à un quorum à 1*

Sur la FIGURE *16*, on observe qu'il y a un très gros écart entre la courbe fixe et la courbe liée aux fusions pour les motifs de petite taille. Cependant au fur et à mesure, qu'on essaye de construire des motifs de taille plus grande, l'écart se resserre. On en déduit donc que les fusions sont beaucoup plus nombreux sur les motifs de petite taille.

Il est aussi intéressant de noter que les fusions commencent à un niveau de seuil d'environ 0,3 et qu'au fur et à mesure que la taille des motifs augmente, le niveau de seuil reste toujours à peu près au même niveau de seuil (fluctue très peu). Attention, la conjecture sur le seuil n'est valable que pour l'exemple des 3 protéines citées précédemment avec les paramètres fixés. En réalité, si on avait des dégénérescences élevées [35], alors le niveau de seuil fluctuerait grandement [36] peu importe des protéines étudiées et de leur nombre. En effet, comme les dégénérescences sont élevés, nous obtenons beaucoup plus de motifs sans pour autant agrandir la taille du « texte » étudiée et donc, on peut potentiellement obtenir plus de motifs ayant une inclusion partielle importante de leurs extensions.

Concernant la **complexité expérimentale** de l'algorithme de Triades, on obtient en calculant sur nos implémentations une complexité en $\mathcal{O}(d_a^2 d_d n k_{max})$ avec d_a la dégénérescence pour les angles, d_d la dégénérescence pour les distances, n la longueur du « texte » et k_{max} la longueur maximale des motifs qu'on peut fabriquer. En effet, la convention que nous avons adopté pour la construction des pavés (non relationnels et relationnels) nous permet d'en déduire de manière assez fiable cette complexité.

Cependant, il faut nuancer cette conjecture car il faut prendre en compte les fonctions de filtrage de l'**étape 3** et **5**. Ces fonctions réduisent les tailles des piles, parfois de manière non négligeable. On peut alors encore en déduire une meilleure complexité.

35. nous les avons testé
36. d'abord en diminuant très rapidement et se rapprochant vers 0 pour les petits motifs puis en augmentant en dépassant le premier niveau de seuil calculé (niveau de seuil des 2-motifs) pour les motifs de taille élevé

Troisième partie

Conclusion

Récapitulatif et perspectives

Pour conclure, lors de notre projet sur 6 mois (un semestre), nous avons pas à pas parvenu à ce travail (ce rapport étant l'aboutissement de nos recherches). D'abord, en étudiant la structure d'une protéine notamment sur les caractéristiques des résidus. Puis, en essayant de comprendre et d'assimiler les définitions spécifiques à l'algorithme de Triades. On a alors pu commencer à implémenter l'algorithme KMRC (l'algorithme de Triades en plus simple, c'est-à-dire sans l'**étape 3** avec les données relationnelles).

Par la suite, pour vérifier que notre implémentation était correcte, nous avons essayé de visualiser les motifs sur PyMOL. Il nous était alors demandé d'implémenter les fonctions d'extraction de données des fichiers .pdb à l'aide des fonctions de lecture/écriture. Nous avons ensuite implémenté l'algorithme de Triades en peaufinant l'algorithme KMRC. Cependant, quelques problèmes étaient survenus lors de nos tests de l'algorithme de Triades notamment sur le temps d'exécution de l'algorithme. Nous avons découvert la présence de motifs non maximaux. Il était donc nécessaire d'implémenter une fonction de filtrage et de savoir où placer ce filtre. L'**étape 3** était alors le résultat de nos réflexions. Enfin, notre projet s'était achevé sur l'implémentation de la fonction de fusion.

En réfléchissant un peu plus sur les différences de l'algorithme KMRC et l'algorithme de Triades, on pouvait non pas de travailler sur 2 types de données (angles et distances) et de les considérer comme 2 entités à part mais plutôt les voir comme un couple de données en une entité. Dans cette perspective, ce couple correspondra alors à un symbole. En définissant les conditions sur ces couples, on obtient qu'un seul type de pavés au lieu de 2. Cela permet donc d'éviter d'effectuer l'**étape 3** et de revenir à un cas de l'algorithme KMRC.

Remerciements

Nous remercions Mme Carpentier de nous avoir guidé tout au long du projet et d'avoir été patiente avec nous. Ce projet a été une bonne découverte de la bioinforatique et nous sommes assez contents des résultats obtenus notamment avec l'affichage en 3 dimensions des motifs sur pyMOL.

Annexe : Bibliographie

Voici les liens utilisés fournies par notre encadrante :
— https://bioinfo.mnhn.fr/abi/people/mathilde/download/TheseCarpentier.pdf
— https://www.researchgate.net/publication/15403482_Finding_flexible_patterns_in_a_text_-_An_application_to_3D_matching
— http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html#ATOM

Voici les liens que nous avons utilisés pour nos recherches :
— https://fr.wikiversity.org/wiki/Outils_math%C3%A9matiques_pour_la_physique_(PCSI)/Produit_scalaire,_produit_vectoriel_et_produit_mixte
— https://bioinfo.mnhn.fr/abi/public/Escan/These_VincentEscalier_97.pdf
— https://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard

Voici les logiciels utilisées lors de notre projet :
— PyMOL : https://pymol.org
— Spyder (Python 3.7) : https://www.spyder-ide.org/