



# Développement d'un alphabet structural intégrant la flexibilité des structures protéiques

Ikram Allam Sekhi

## ► To cite this version:

Ikram Allam Sekhi. Développement d'un alphabet structural intégrant la flexibilité des structures protéiques. Bio-informatique [q-bio.QM]. Université Sorbonne Paris Cité, 2018. Français. NNT : 2018USPCC084 . tel-02296605

**HAL Id: tel-02296605**

**<https://theses.hal.science/tel-02296605>**

Submitted on 25 Sep 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Université Paris Diderot

Sorbonne Université

Université Sorbonne Paris Cité

École doctorale Médicament, Toxicologie, Chimie, Imageries

# THÈSE DE DOCTORAT

Discipline : Bioinformatique

Présentée par

**Ikram ALLAM**

---

## Développement d'un alphabet structural intégrant la flexibilité des structures protéiques

---

Dirigée par Anne-Claude CAMPROUX et Grégory NUEL

Soutenue le 29 Janvier 2018 devant le jury composé de :

Pr. Alessandra CARBONE	Sorbonne Université	Présidente et rapporteure
Pr. Dominique BARTH	Université de Versailles-St-Quentin-en-Yvelines	Rapporteur
MC. Delphine FLATTERS	Université Paris Diderot	Examinatrice
Dr. François ANDRÉ	I2BC-CEA CNRS Université Paris Saclay	Examineur
Pr. Anne-Claude CAMPROUX	Université Paris Diderot	Directrice
Pr. Grégory NUEL	Sorbonne Université	Co-directeur

**Université Paris Diderot**

**Sorbonne Université**

**Université Sorbonne Paris Cité**

**École doctorale Médicament, Toxicologie, Chimie, Imageries  
(MTCI ED 563)**

# **THÈSE DE DOCTORAT**

**Discipline : Bioinformatique**

Présentée par

**Ikram ALLAM**

Pour obtenir le grade de

**DOCTEUR de l'UNIVERSITÉ PARIS DIDEROT**

---

## **Développement d'un alphabet structural intégrant la flexibilité des structures protéiques**

---

Dirigée par Anne-Claude CAMPROUX et Grégory NUEL

Soutenue le 29 Janvier 2018 devant le jury composé de :

Pr. Alessandra CARBONE	Sorbonne Université	Présidente et rapporteure
Pr. Dominique BARTH	Université de Versailles-St-Quentin-en-Yvelines	Rapporteur
MC. Delphine FLATTERS	Université Paris Diderot	Examinatrice
Dr. François ANDRÉ	I2BC-CEA CNRS Université Paris Saclay	Examineur
Pr. Anne-Claude CAMPROUX	Université Paris Diderot	Directrice
Pr. Grégory NUEL	Sorbonne Université	Co-directeur

## Résumé

**Mots clés :** Structure tridimensionnelle d'une protéine, Alphabet Structural (AS), les chaînes de Markov cachées (HMM), Lettre structurale, fichier au format PDB (*Protein Data Bank*), *parser* PDB, encodage structural, *Maximum a Posteriori* (MAP), *marginal posterior distribution* (POST), *entropie d'encodage*, *effective number of structural letters* (NEFF), prise en compte des données manquantes, prise en compte de l'incertitude des données, variabilité des conformations. Flexibilité des protéines.

L'objectif de cette thèse est de proposer un Alphabet Structural (AS) permettant une caractérisation fine et précise des structures tridimensionnelles (3D) des protéines, à l'aide des chaînes de Markov cachées (HMM) qui permettent de prendre en compte la logique issue de l'enchaînement des fragments structuraux en intégrant l'augmentation des conformations 3D des structures protéiques désormais disponibles dans la banque de données de la *Protein Data Bank* (PDB).

Nous proposons dans cette thèse un nouvel alphabet, améliorant l'alphabet structural HMM-SA27, appelé SAFlex (*Structural Alphabet Flexibility*), dans le but de prendre en compte l'incertitude des données (données manquantes dans les fichiers PDB) et la redondance des structures protéiques. Le nouvel alphabet structural SAFlex obtenu propose donc un nouveau modèle d'encodage rigoureux et robuste. Cet encodage permet de prendre en compte l'incertitude des données en proposant trois options d'encodages : le *Maximum a posteriori* (MAP), la distribution marginale *a posteriori* (POST) et le nombre effectif de lettres à chaque position donnée (NEFF). SAFlex fournit également un encodage consensus à partir de différentes réplifications (chaînes multiples, monomères et homomères) d'une même protéine. Il permet ainsi la détection de la variabilité structurale entre celles-ci. Les avancées méthodologiques ainsi que l'obtention de l'alphabet SAFlex constituent les contributions principales de ce travail de thèse. Nous présentons aussi le nouveau *parser* de la PDB (SAFlex-PDB) et nous démontrons que notre *parser* a un intérêt aussi bien sur le plan qualitatif (détection de diverses erreurs) que quantitatif (rapidité et parallélisation) en le comparant avec deux autres *parsers* très connus dans le domaine (Biopython et BioJava).

Nous proposons également à la communauté scientifique un site web mettant en ligne ce nouvel alphabet structural SAFlex. Ce site web représente la contribution concrète de cette thèse alors que le *parser* SAFlex-PDB représente une contribution importante pour le fonctionnement du site web proposé. Cette caractérisation précise des conformations 3D et la prise en compte de la redondance des informations 3D disponibles, fournies par SAFlex, a en effet un impact très important pour la modélisation de la conformation et de la variabilité des structures 3D, des boucles protéiques et des régions d'interface avec différents partenaires, impliqués dans la fonction des protéines.



## Abstract

**Keywords :** protein three-dimensional structure, Structural Alphabet (SA), Hidden Markov Model PDB (Protein Data Bank), PDB File Format, PDB file parser, Structural protein encoding, *Maximum a Posteriori* (MAP), *marginal posterior distribution* (POST), encoding entropy , *effective number of structural letters* (NEFF), missing data in protein 3D structure, encoding data uncertainty, protein variability, protein structural flexibility.

The purpose of this PhD is to provide a Structural Alphabet (SA) for more accurate characterization of protein three-dimensional (3D) structures as well as integrating the increasing protein 3D structure information currently available in the Protein Data Bank (PDB). The SA also takes into consideration the logic behind the structural fragments sequence by using the hidden Markov Model (HMM).

In this PhD, we describe a new structural alphabet, improving the existing HMM-SA27 structural alphabet, called SAFlex (Structural Alphabet Flexibility), in order to take into account the uncertainty of data (missing data in PDB files) and the redundancy of protein structures. The new SAFlex structural alphabet obtained therefore offers a new, rigorous and robust encoding model. This encoding takes into account the encoding uncertainty by providing three encoding options : the maximum *a posteriori* (MAP), the marginal posterior distribution (POST), and the effective number of letters at each given position (NEFF). SAFlex also provides and builds a consensus encoding from different replicates (multiple chains, monomers and several homomers) of a single protein. It thus allows the detection of structural variability between different chains. The methodological advances and the achievement of the SAFlex alphabet are the main contributions of this PhD. We also present the new PDB parser (SAFlex-PDB) and we demonstrate that our parser is therefore interesting both qualitative (detection of various errors) and quantitative terms (program optimization and parallelization) by comparing it with two other parsers well-known in the area of Bioinformatics (Biopython and BioJava).

The SAFlex structural alphabet is being made available to the scientific community by providing a website. The SAFlex web server represents the concrete contribution of this PhD while the SAFlex-PDB parser represents an important contribution to the proper function of the proposed website. Here, we describe the functions and the interfaces of the SAFlex web server. The SAFlex can be used in various fashions for a protein tertiary structure of a given PDB format file ; it can be used for encoding the 3D structure, identifying and predicting missing data. Hence, It is the only alphabet able to encode and predict the missing data in a 3D protein structure to date. Finally, these improvements are promising to explore increasing protein redundancy data and obtain useful quantification of their flexibility.

## Remerciements

“On ne grandit pas sans souffrances, et on n’apprend pas sans faire d’erreurs, et on ne réussit pas sans échec. Ainsi va la vie, mais il ne faut pas perdre espoir après un premier échec. Lorsque Dieu ferme une porte, il en ouvre toujours une autre”

Proverbe arabe

Louanges, à Dieu qui m’a guidé et permis de trouver les bonnes clés pour ouvrir les grandes portes en vue de finaliser cette thèse. La route sur laquelle je me suis engagée lorsque j’ai commencé cette thèse a été longue et semée d’obstacles mais extrêmement enrichissante au niveau humain et connaissances.

Je tiens tout d’abord à remercier mon encadrante Madame Anne Claude Camproux pour son aide, sa patience et ses conseils avisés dans le cadre de la préparation de cette thèse qui sans son aide n’aurait pu arriver à terme, je tiens particulièrement à la remercier pour son aide en bio-informatique et sa collaboration pour reprendre et compléter son précédent travail. Travailler sous sa direction m’a permis de progresser plus rapidement et avec méthode.

Je tiens également à remercier mon encadrant Monsieur Gregory Nuel pour son aide, sa patience et ses encouragements. Cette thèse n’aurait pas pu arriver à son terme sans ses conseils avisés et son aide en probabilités et statistiques ainsi que ses compétences de chercheur et enseignant. Son aide m’a permis de franchir des obstacles que je pensais insurmontables qu’il en soit grandement remercié.

Je remercie Mesdames et Messieurs les membres du jury pour avoir accepté de juger mon travail et pour leurs remarques extrêmement constructives.

Je tiens à remercier « Initiative D’EXcellence (IDEX) qui a financé le projet de cette thèse *SAFlex Structural Alphabet Flexibility* de l’université Sorbonne Paris Cité (SPC).

Je remercie l’ensemble de l’équipe du laboratoire Inserm MTi (Molécules Thérapeutiques In silico, équipe *Computational approaches applied to pharmacological profiling*) à l’université Paris-Diderot et le laboratoire CNRS LPMA (Laboratoire de Probabilités et Modèles Aléatoires, équipe *Probability Statistique and Biology*) à l’université Pierre et Marie Curie pour leurs précieuses collaborations. En effet, l’équipe du MTi a joué un rôle important en ce qui concerne la connaissance des structures protéiques et *Drug Design* et les AS (Alphabets Structuraux) et l’équipe du LPMA avec sa connaissance du HMM (les chaînes de Markov cachées) ainsi que la méthodologie mathématique pour atteindre les objectifs de cette thèse. Je remercie d’une manière particulière pour l’aide et la collaboration qu’ils m’ont apportée et plus spécialement : Inés, Dhoha, Soukaina, Alexandre, Mélaine, Stéphanie, Baptiste, Michel, Delphine, Jérôme, Alban, Natacha, Géraldine et Olivier Spérandio, Gautier, Dehbia, Marouane et Vincent Delos, pour le temps et la patience qu’ils m’ont accordés. De manière générale je remercie tous ces responsables pour le temps et l’aide qu’ils ont consenti à m’accorder et je remercie également tous mes collègues qui m’ont soutenu et aidé durant tout ce travail.

Enfin un immense merci n’est pas suffisant pour remercier toutes les personnes que j’aime autant : mes parents, mes sœurs et mon mari pour leur soutien continu, leur aide et leur patience. Merci grand mère pour toutes tes prières de réussite. Merci Papa et Massinissa pour tous vos encouragements. Un grand merci également à maman pour son aide et son soutien dans une période pénible.

# Sommaire

<b>Résumé</b>	<b>iii</b>
<b>Remerciements</b>	<b>v</b>
<b>Table des matières</b>	<b>x</b>
<b>Table des figures</b>	<b>xii</b>
<b>Liste des tableaux</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Contexte scientifique . . . . .	1
1.2 Applications des AS . . . . .	4
1.3 Limites des AS . . . . .	7
1.4 Objectifs et plan de la thèse . . . . .	8
<b>2 Un nouveau parser de fichiers PDB</b>	<b>11</b>
2.1 Introduction . . . . .	11
2.2 Les fichiers de données de la banque PDB . . . . .	12
2.3 Les parsers PDB existants . . . . .	17
2.3.1 PDBlib (1994) . . . . .	17
2.3.2 Biopython (2003) . . . . .	20
2.3.3 BioJava (2008) . . . . .	21
2.3.4 BiopLib (2015) . . . . .	22
2.3.5 Les avantages et inconvénients des parsers PDB . . . . .	22
2.4 Le nouveau parser SAFlex-PDB . . . . .	24
2.4.1 Contexte et fonctionnalités . . . . .	24
2.4.2 Modélisation et implémentation . . . . .	24
2.4.3 Optimisation et parallélisation du parser PDB . . . . .	28
2.5 Tests du parser SAFlex-PDB . . . . .	31
2.5.1 Jeu de données . . . . .	31



2.5.2	Préparation des tests . . . . .	32
2.5.3	Présentation des résultats . . . . .	33
2.6	Qualité du parser SAFlex-PDB . . . . .	36
2.7	Conclusion . . . . .	38
<b>3</b>	<b>Les avancées méthodologiques</b>	<b>39</b>
3.1	L'alphabet structural existant HMM-SA27 . . . . .	39
3.1.1	Sélection des modèles . . . . .	40
3.1.2	Encodage des protéines . . . . .	41
3.2	Démarche de développement d'un AS . . . . .	43
3.3	Décomposition des structures 3D en fragments . . . . .	46
3.3.1	Décomposition des structures 3D en fragments . . . . .	46
3.3.2	Jeu de données . . . . .	47
3.4	Description du fragment structural . . . . .	48
3.5	Modélisation des structures 3D par HMM . . . . .	49
3.5.1	Encodage des structures protéiques . . . . .	50
3.5.2	Gestion des données manquantes . . . . .	52
3.5.3	Gestion des chaînes multiples . . . . .	52
3.6	Implémentation du nouvel AS . . . . .	53
3.6.1	Analyse, conception et modélisation des besoins . . . . .	54
3.6.2	Décomposition en modules du programme . . . . .	55
3.6.3	Implémentation de l'algorithme EM . . . . .	58
3.6.4	Implémentation en échelle logarithmique . . . . .	59
3.7	Optimisation du code . . . . .	59
3.7.1	Tests de performance . . . . .	59
3.7.2	Analyse des modules . . . . .	60
3.7.3	Profil de performance . . . . .	61
3.7.4	Corrections et optimisations . . . . .	61
3.7.5	Résultats de l'optimisation . . . . .	62
3.8	Le nouvel alphabet structural SAFlex . . . . .	64
3.8.1	Un encodage structural robuste aux données manquantes . . . . .	66
3.8.2	Un encodage qui gère les chaînes multiples . . . . .	70
3.9	Preuve de concept . . . . .	76
3.10	SAFlex et HMM-SA27 : Comparaison méthodologique . . . . .	80
3.11	Conclusion . . . . .	81
<b>4</b>	<b>Un nouveau site web pour SAFlex</b>	<b>83</b>
4.1	Introduction . . . . .	83
4.2	Conception et implémentation du site web . . . . .	84
4.2.1	Implementation de l'interface graphique . . . . .	84
4.2.2	Architecture du site web . . . . .	84

4.2.3	Les versions de l'alphabet structural : SAFlex.V1 et SAFlex. $\beta$	87
4.2.4	Performance du site web	87
4.3	Description des fonctionnalités du site web	88
4.3.1	Les entrées du site web	88
4.3.2	Les résultats du site web	89
4.4	Conclusion	96
<b>5</b>	<b>Conclusion et perspectives</b>	<b>97</b>
<b>Annexe A</b>	<b>Estimateur du maximum de vraisemblance</b>	<b>103</b>
A.1	Méthode de maximum de vraisemblance	103
A.1.1	Calcul de l'estimateur de la fonction maximum de vraisemblance pour une loi normale à une dimension	104
A.1.2	Calcul l'estimateur de la fonction maximum de vraisemblance pour une loi normale multivariée à $d$ dimensions	106
<b>Annexe B</b>	<b>L'algorithme EM et les HMM</b>	<b>109</b>
B.1	L'algorithme EM	109
B.2	HMM	110
B.3	Algorithme de Forward et Backward	111
<b>Annexe C</b>	<b>Optimisation des programmes</b>	<b>117</b>
C.1	Optimisation et tuning de performance	117
C.1.1	Méthodologie et approche	117
C.2	Granularités et types de parallélismes	119
C.2.1	Parallélisme d'instructions	119
C.2.2	Le parallélisme de mémoire et défaut de caches	120
C.2.3	Parallélisme de données	120
C.2.4	Parallélisme de threads	121
C.2.5	Parallélisme de tâches	121
C.3	Conclusion	122
<b>Annexe D</b>	<b>Tableaux des résultats de tests de performances</b>	<b>123</b>
<b>Annexe E</b>	<b>Tableaux du parser SAFlex-PDB</b>	<b>129</b>
E.1	Les statistiques sur PDBSelect	129
E.2	Tableaux tests comparatifs	129
<b>Annexe F</b>	<b>Tableau récapitulatif des AS</b>	<b>131</b>
<b>Annexe G</b>	<b>Résultats de SAFlex</b>	<b>137</b>
G.1	Résultats de SAFlex.V1	137
G.2	Résultats de SAFlex. $\beta$	142

<b>Annexe H Article soumis à PLOS One</b>	<b>147</b>
<b>Bibliographie</b>	<b>167</b>

# Table des figures

2.1	Classification globale des classes PDBlib et les relations entre elles . . . . .	18
2.2	Représentation conceptuelle de la relation entre les classes PDBlib . . . . .	19
2.3	Diagramme UML de la structure de données « SMCRA ». . . . .	20
2.4	Modélisation UML par un diagramme de classes de données . . . . .	26
2.5	Comparaison de temps d'exécution de SAFlex-PDB avant et après optimisation . . . .	28
2.6	Comparaison de temps d'exécution entre SAFlexPDB optimisé et parallélisé . . . . .	31
2.7	Détection du manque des C $\alpha$ par les parsers . . . . .	35
2.8	Détection des trous à l'intérieur des protéines . . . . .	36
2.9	Comparaison de temps d'exécution entre SAFlexPDB, BioPython et Biojava . . . . .	37
3.1	Géométrie des 27 lettres structurales de l'alphabet structural HMM-SA27 . . . . .	40
3.2	Évolution du BIC en fonction du nombre d'états . . . . .	41
3.3	Encodage structurale de la chaîne 4ll3A . . . . .	42
3.4	Démarche méthodologique du développement d'un alphabet structural . . . . .	45
3.5	Décomposition d'une partie de la conformation tridimensionnelle de la chaîne 4ll3A en fragments structuraux . . . . .	46
3.6	Représentation détaillée d'un fragment structural . . . . .	47
3.7	Les descripteurs du fragment structural . . . . .	48
3.8	Le modèle HMM-SA27 . . . . .	50
3.9	Le modèle SAFlex avec des chaînes multiples. . . . .	53
3.10	Les modules de développement de SAFlex . . . . .	56
3.11	Le profiling du module 4 montrant les dix premières fonctions consommatrices du CPU avant optimisation, obtenu par l'outil <i>callgrid</i> . . . . .	60
3.12	Comparaison du temps d'exécution du module de génération de l'alphabet structural avant et après optimisation . . . . .	63
3.13	Les lettres de l'alphabet structural SAFlex.V1 . . . . .	65
3.14	Les options d'encodage structural de la structure protéique 2hba. . . . .	66
3.15	Comparaison entre l'encodage des alphabets structuraux . . . . .	67
3.16	La corrélation entre les fragments structuraux manquants et l'entropie d'encodage . .	68

3.17	Détection des conformations des fragments manquants pour la structure de la PDB 3h8z	69
3.18	Les encodages structuraux associés aux quatre monomères pour le fichier PDB 1gme	71
3.19	Les valeurs NEFF associées aux quatre monomères vec la chaîne consensus générée par SAFlex pour le fichier PDB 1gme	73
3.20	Comparaison entre les quatre monomères du PDB 1gme	74
3.21	Correspondance entre le B facteur et les variations de l'encodage structural pour les quatre monomères de HSP	75
3.22	Prédiction de la protéine HSP16.9 du fichier PDB 1gme.	76
3.23	Les lettres structurales de la version SAFlex. $\beta$	77
3.24	A1 vue de dessus	78
3.25	A2 vue de dessus	78
3.26	A3 vue de dessus	79
4.1	Architecture du site web SAFlex	86
4.2	Les caractéristiques structurales des lettres de l'alphabet structural SAFlex. $\beta$ sur la page web.	87
4.3	Le formulaire « SAFlex-Encoder » du site web SAFlex.	88
4.4	Les modes de visualisation 3D associés aux structures proposés par SAFlex	90
4.5	Les résultats de l'encodage structural de la chaîne 'A' du PDB '2hba' affichés par SAFlex.	92
4.6	L'entropie de l'encodage structural de la chain 'A' du PDB '2hba'.	93
4.7	L'encodage POST de la chaîne 2hbaA du PDB '2hba'	94
4.8	L'entropie de l'encodage structural de la chaîne consensus associée aux deux monomères 'A' et 'B' du PDB '2hba'.	95
B.1	Le modèle HMM	111

# Liste des tableaux

2.1	Statistiques sur le jeu de données PDBSelect . . . . .	32
2.2	Détection des erreurs et manques de données par les trois parsers . . . . .	34
2.3	Temps du traitement des données de PDBSelect par les trois parsers . . . . .	36
3.1	Tableau de correspondance entre les lettres structurales de HMM-SA27 et SAFlexV1 .	64
3.2	Le RMSD entre les coordonnées des monomères de 1GME . . . . .	72
D.1	Tests de performance du parser SAFlexPDB . . . . .	124
D.2	Tests de performances du module 2 . . . . .	125
D.3	Tests de performances du module 3 . . . . .	126
D.4	Tests de performances du module 4 . . . . .	127
D.5	Temps d'exécution du module générer AS . . . . .	128
E.1	Statistiques sur le jeu de données PDBSelect . . . . .	129
E.2	Tableau de tests comparatifs entre les <i>parsers</i> : BioPython, BioJava et SAFlex-PDB .	129
F.1	Récapitulatif des différentes librairies et des alphabets structuraux . . . . .	135
G.1	La matrice de transition de SAFlex.V1 . . . . .	138
G.2	Les paramètres des lettres structurales de l'alphabet SAFlex.V1 . . . . .	141
G.3	Tableau récapitulatif des moyennes et des occurrences des lettres structurales . . . . .	143
G.4	Matrice de transitions entre les lettres structurales . . . . .	144
G.5	Tableau des probabilités de sortie et de provenance des lettres structurales . . . . .	145
G.6	Matrice de provenance des lettres structurales . . . . .	146



# Introduction

## 1.1 Contexte scientifique

Les protéines sont des macromolécules biologiques impliquées dans les processus centraux de la vie. Elles interviennent, à titre d'exemple, dans la régulation du métabolisme, dans le transport des molécules, et dans la défense de l'organisme. La protéine ne peut réaliser sa fonction que si elle adopte le bon repliement dans l'espace tridimensionnel (3D), qui correspond, en général, au minimum d'énergie libre de la protéine. Les protéines sont tout d'abord classiquement décrites par leur séquence primaire, c'est-à-dire leur séquence en acides aminés<sup>1</sup> La formation de liaison d'hydrogène entre les résidus proches dans la séquence permet de créer des repliements locaux au sein de la protéine. Ceux-ci correspondent ainsi à la structure secondaire de la protéine. Deux types de structures secondaires régulières sont identifiés, les hélices alpha et les brins bêta [134]. Elles sont retrouvées très fréquemment dans les structures protéiques et sont décrites précisément. Elles sont reliées entre elles par les boucles qui sont des régions plus variables. Des bibliothèques ou librairies de fragments structuraux ont été développées pour décrire plus finement les boucles des structures protéiques.

Dans ce qui suit, nous expliquons brièvement le principe d'une librairie de fragments et quelques exemples de librairies existantes et leurs objectifs. Historiquement, l'idée d'utiliser des fragments a été initiée par Jones et Thirup en 1986 : En effet, leur reconstitution de la protéine de liaison au rétinol à partir des fragments représente la première utilisation de petits motifs structuraux tridimensionnels<sup>2</sup> [92]. Ces librairies de fragments structuraux sont constituées d'une collection limitée de fragments spécifiques des protéines. Ainsi, plusieurs équipes se sont penchées sur le développement de ces librairies : *Building Blocks* [177], [39], *Recurrent local structural motifs* [152], *Substructures* [139], [108], *Local Structural Motifs* [159], I-Sites [24], HMMSTR [25], *Oligons* [123], *Fragment Libraries* [100, 106], *Structural Representative* [155], *Centroids* [88, 102], *Building Blocks Library* [56, 57], *BriX data base* [5], *FragBag* [23], *Dynamic Fragment Library* [94], *Flib Libraries* [51].

La construction de ces librairies de fragments structuraux visait plusieurs objectifs principaux, qui ont évolué durant ces dernières décennies. Durant les années 80 jusqu'au milieu des années 90,

1. Chaque acide aminé possède à la fois une fonction acide carboxylique (COOH) et une fonction amine (NH<sub>2</sub>).

2. Les motifs structuraux tridimensionnels sont issus des chaînes principales de trois protéines ayant peu d'homologie de séquence.



les différentes équipes s'intéressaient principalement à *la reconstitution des protéines* ou des chaînes principales, à partir d'assemblage de motifs structuraux en utilisant principalement des méthodes expérimentales [39, 92, 139, 177]. À partir des années 90 à nos jours, les objectifs se sont diversifiés. Ainsi, certains auteurs cherchent à avoir un nombre important de fragments représentatifs par *une description précise des structures 3D* [100, 123, 152], alors que d'autres auteurs cherchent un nombre plus limité de fragments protéiques, dédiés à *la prédiction des structures locales* à partir de la séquence primaire [152, 153].

Ces bibliothèques apportent une description plus précise des structures protéiques que les structures secondaires classiques, mais ne prennent pas en compte leurs organisations structurales. Or l'agrégation des fragments structuraux obéit à des règles logiques non prises en compte par ces bibliothèques. Ce qui a conduit à introduire le concept de l'Alphabet Structural (AS). Les AS suscitent un grand intérêt auprès des chercheurs en bioinformatique structurale. Dans la littérature, différentes définitions d'AS ont été proposées, nous citons celle proposée en 2014 par Jianzhu et Sheng : « les AS sont constitués d'un ensemble discret de représentants de structures locales, ils sont généralement construits à partir de la classification de fragments locaux issus des structures protéiques, en se basant sur une variété de mesures géométriques et différentes longueurs de fragments » [90].

D'une manière plus précise dans cette thèse, nous définissons un AS, comme étant un ensemble de conformations géométriques représentatif des fragments protéiques, appelés « lettres structurales (LS) », et destinées à simplifier la structure 3D d'une macromolécule biologique. Ces LS suivent des règles logiques qui permettent de les associer entre elles. En d'autres termes, un AS est une bibliothèque de LS représentatives associées entre elles en suivant des règles logiques. Ainsi, ces AS sont des outils très puissants pour la simplification des structures 3D des protéines en séquences de LS. Les précurseurs de la notion d'AS sont Zhang et ses collaborateurs en 1993 [186]. Ils *mirent en évidence les caractéristiques intrinsèques des structures macromoléculaires* en développant, à partir de 74 protéines, un premier AS, appelé *Structural Building Blocks*. Cet AS contenait six fragments de sept résidus et avait comme objectif de prédire et reconstruire la structure 3D des protéines globulaires [186]. Depuis, plusieurs AS ont été développés, nous citons quelques exemples des AS pertinents existants : *Structural Building blocks* [64, 186], *Short Structural Building Blocks* [31], HMM-SA27 [28], *Protein Blocks* [48, 63], SADB<sup>3</sup> [172], PFSC<sup>4</sup> [183], *Conformational Letter* [180], M32K25 [132], USA<sup>5</sup> [173]. Ces AS diffèrent par leurs objectifs de construction et d'applications, les méthodes et données utilisées dans leur apprentissage. Nous récapitulons donc dans le tableau (F.1) les principaux AS ainsi que leurs méthodes d'élaboration et leurs contenus.

Plusieurs désignations ont été attribuées aux fragments structuraux représentatifs, que nous nommons LS dans cette thèse : *Building Bloc* (BB) par [177], *SubStructure* par [139], *Recurrent local structural motif* par [152], *Structural Building Block* par [186], *Local structural motif* par [159], *Structural Building block* [64], *Short structural Building Block* par [31], *Oligon* par [123], *Protein Block* (PB) par [44], *Centroid* par [88], *Structural Letter* (SL) par [28], *Local Protein Structure* (LPS) par [10], *Structural Representative* (SR) par [155], *Protein Folding Shape Code* (PFSC) par [183], *Centroid*

---

3. Structural Alphabet Data Base

4. Protein Folding Shape Code

5. Units of Structural Alphabet

par [102], *Conformational Letter* (CLe) par [180], *Conformational Attractors* par [132], et *Unit of Structural Alphabet* (USA) par [173].

La taille de ces fragments structuraux représentatifs varie selon les AS et leurs objectifs. Ils peuvent être issus uniquement des chaînes principales (les chaînes polypeptidiques entre les carbones alpha ( $C\alpha$ ), comme ils peuvent inclure les chaînes latérales. D'autre part, les fragments structuraux peuvent être chevauchants [31, 25, 48, 28, 63, 45, 132, 94] ou non [100]. Cette idée d'utiliser des fragments chevauchants revient à Claessens et ses collaborateurs en 1989 [39]<sup>6</sup>. Il convient de noter que ce chevauchement permet la prise en compte des relations de dépendances spatiales existantes entre les fragments structuraux successifs parmi les protéines. Le choix de la taille d'un fragment varie en fonction de l'objectif recherché : si l'objectif visé est la prédiction des structures locales à partir de la séquence primaire, ceci justifie le fait de choisir des fragments protéiques longs avec un nombre plus limité de fragments représentatifs [48, 152]. Si par ailleurs c'est la caractérisation ou la description fine et précise des structures tridimensionnelles qui est le but, des fragments protéiques courts en nombre relativement important seront plus intéressants à adopter [31, 123, 100, 28, 132]. Tous ces éléments expliquent que les librairies et AS développés possèdent un nombre différent de fragments représentatifs, de taille variable. Enfin, différents descripteurs ont été utilisés pour la caractérisation de ces fragments protéiques. Les premiers descripteurs étaient basés sur les coordonnées cartésiennes des atomes de  $C\alpha$  qui constituent le fragment structural [123, 177]. Par la suite [48, 132, 172] ont utilisé des descripteurs basés sur des coordonnées internes (les angles dièdres ( $\phi/\psi$ ), ou les angles  $\alpha$ , ou encore angle  $\kappa$ ). Ainsi, [132] ont préféré caractériser leurs fragments par deux pseudo-angles de liaison ( $\phi_1, \phi_2$ ) et un angle de torsion ( $\theta$ ) calculés entre les atomes  $C\alpha$ . Une autre façon de décrire les fragments structuraux protéiques choisis par [31, 28, 155, 183], repose sur les distances calculées à partir des positions des atomes  $C\alpha$ . En effet, Camproux et ses collaborateurs ont décrit leur fragment structural  $i$  à l'aide d'un vecteur de distances entre les coordonnées cartésiennes des  $C\alpha$  non successifs  $X_i = (X_i^1, X_i^2, X_i^3, X_i^4)$ , présentés dans la section 3.4 [31, 28]. D'autre part, [64] ont caractérisé les fragments en se servant des distances entre  $C\alpha$  et un angle. Enfin les séquences en acides aminés ont été suggérées par [24, 26, 56]

Pour finir avec les différents AS, différentes méthodes ont été utilisées pour leur apprentissage. Ces méthodes classifiaient les fragments protéiques en plusieurs groupes qui se ressemblent du point de vue de leurs géométries tridimensionnelles. Le choix entre ces méthodes et ces modèles joue un rôle crucial dans le développement d'un AS robuste et rigoureux. Ces méthodes d'apprentissage utilisent des algorithmes de classification automatique dans les cas des AS suivants : [64, 172, 173, 183, 186]. Elles sont basées par exemple sur les *k-means* [111], la classification hiérarchique ascendante [89], les réseaux de neurones formels [122], la méthode des *K-NN* (*K-Nearest Neighbor*) [41, 65]. Un AS a été obtenu à l'aide des chaînes de Markov cachées (HMM) tels que [28, 31]. Les chaînes de Markov cachées [166] ont prouvé leur utilité dans la biologie moléculaire, notamment dans les domaines de la biologie structurale. Ainsi, ils ont servi à modéliser les familles de protéines, à déterminer les domaines protéiques dans une séquence de requêtes, à aligner les séquences multiples ou à prédire la topologie des protéines bêta transmembranaires [101, 61, 6, 133]. L'utilisation des HMM dans le développement

---

6. Leurs travaux consistait à reconstituer la chaîne principale d'une protéine à partir de motifs récurrents chevauchants et de tailles variables issus de 66 structures cristallographiques

des AS, a permis la prise en compte de l'enchaînement et de la dépendance spatiale existante entre les fragments structuraux, car ces AS supposèrent que ceux-ci suivent des règles logiques entre eux, ce qui impliquait l'emploi de fragments chevauchants.

## 1.2 Applications des AS

Les AS ont de multiples applications pour l'analyse des structures protéiques avec deux objectifs principaux : la description fine et détaillée des structures ainsi que la reconstruction de celles-ci. Plusieurs applications existent actuellement visant à améliorer la reconnaissance des conformations protéiques [74], à prédire des structures 3D [46,63,57,56], à analyser leur dynamique moléculaire [131,114,130], à extraire de motifs de boucles fonctionnels [146], à identifier la signature de site de liaison [59], et à détecter les motifs conservés<sup>7 8</sup> [59,148,103,164].

D'autres types d'applications ont pour objectif de reconstruire certaines structures 3D des protéines. Ainsi [49, 46, 50, 10, 20] parvinrent à reconstruire des longs fragments des structures 3D, en utilisant l'AS *Protein Blocks* [48]<sup>9</sup>. Quant à [67,175], ils réussirent à reconstruire des courtes boucles. Enfin certaines équipes [170, 120, 169,105] se sont penchées sur la reconstruction des peptides en utilisant différents AS. À titre d'exemple [121,120,169] employèrent l'encodage structural issu de l'AS HMM-SA27 pour reconstruire des peptides dont la taille varie de 10 à 49 résidus. Récemment, les AS ont été exploités dans des applications très importantes telles que la prédiction de la flexibilité des protéines [21, 47,131,42, 55], et l'analyse de la variabilité structurale d'une cible thérapeutique [143].

Certaines de ces applications ont été mises à la disposition de la communauté scientifique à travers la conception de sites web. La plupart ont été proposées pour la comparaison et la recherche des similitudes entre les structures 3D (SA-Search [79], PB-align<sup>10</sup> [176], 3D-BLAST<sup>11</sup> [184,172], fastSCOP<sup>12</sup> [174], CLePAPS [180], CleFAPS [179], iPBA [73], mulPBA [107], BLOMAPS [180, 187], PAR-3D-BLAST [154]). Récemment un outil très performant 3DCOMB [150] montre son efficacité pour les alignements multiples des structures protéiques avec un score élevé et une bonne qualité d'alignement. Il est basé sur l'AS dénommé *CLe* [180]<sup>13</sup>. D'autres sites web concernent l'analyse des chaînes latérales : SCit [72] basé sur HMM-SA27, proposé en service<sup>14</sup> afin d'analyser les chaînes latérales et leurs positionnements et d'identifier les conformations peu probables de celles-ci. D'autres services sont également proposés tels que : la génération, la sélection et l'évaluation des sous-ensembles de base de données [129], la détection des domaines structuraux [174], l'extraction des positions des motifs structuraux récurrents et conservés au sein des boucles protéiques [147] et la reconstruction de la trace des C $\alpha$  [121].

7. [59] découvrirent un ensemble de motifs structuraux caractérisant les sites de liaison liés au ligand Mg<sup>2+</sup> sur 70 structures protéiques

8. Ces protéines partagent moins de 30% d'identité de séquence, en se basant sur l'AS « centroids » [102]. Ces motifs ont été nommés donc, SA-motifbase [59].

9. Celui-ci comprend 16 fragments protéiques représentatifs, appelés PBs, de longueur 5

10. [http://www.bo-protscience.fr/pbe/?page\\_id=12](http://www.bo-protscience.fr/pbe/?page_id=12)

11. <http://3d-blast.life.nctu.edu.tw>

12. <http://fastSCOP.life.nctu.edu.tw>

13. l'alphabet « CLe » constitué de 17 lettres conformationnelles de 4 résidus

14. (<http://bioserv.rpbs.jussieu.fr/SCit>)

Parmi les alphabets existants, nous avons choisi dans ce travail d'améliorer l'AS existant HMM-SA27. En effet, il permet de décrire finement les structures tridimensionnelles des protéines en prenant en compte la logique de l'enchaînement entre les fragments structuraux. HMM-SA27 a été développé par Camproux et ses collaborateurs, nommé, en 1999, *Short structural Building Blocks* et qui contenait 12 SBBs [31]. Ce travail a été poursuivi en 2004, pour aboutir à un AS disposant de 27 LS [28]. Cet alphabet se base sur l'emploi des chaînes de Markov cachées et modélise les fragments structuraux par un processus Markovien afin de considérer l'enchaînement entre les fragments structuraux dans le processus d'apprentissage de ceux-ci. De ce fait les fragments représentatifs sont appris en tenant compte de leurs géométries et de leurs transitions. Il sera décrit plus précisément dans le chapitre 3. Cet AS a montré son intérêt à travers ses différentes applications dont quelques-unes ont été citées dans le paragraphe précédent. Dans ce qui suit, nous présentons les différentes applications de l'AS HMM-SA27 existantes, puis certaines de ses limites. Ainsi les différentes applications de HMMSA27 permettent :

1. *la recherche et l'analyse des similitudes structurales* entre les protéines grâce à l'alignement structural à l'aide de l'outil SA-Search [79], en réduisant la recherche 3D à des problèmes d'alignement de séquence 1D tout en comparant les protéines ainsi encodées. SA-Search est un serveur web, utilisant une approche identique à celle de Blast [3], mais appliquée sur des séquences de LS et non sur des acides aminés, ce qui le rend beaucoup plus rapide. Cette détection et analyse des similarités structurales peut fournir une meilleure précision sur les mécanismes fonctionnels ou les relations entre les protéines.
2. *l'étude des conformations des chaînes latérales* au sein des structures protéiques à l'aide de SCit [72], un outil proposé en service à travers le site web (<http://bioserv.rpbs.jussieu.fr/SCit>), pour l'analyse des chaînes latérales, leurs positionnements et l'identification des conformations peu probables de chaînes latérales. En effet, celles-ci jouent un rôle essentiel dans le repliement de la protéine et la stabilisation de sa structure à cause des interactions spécifiques entre les régions distinctes de la chaîne polypeptidique.
3. *la prédiction ab initio locale simplifiée* des structures protéiques dans l'espace de l'AS à partir de la séquence primaire [29], ce travail permet de souligner la dépendance considérable présente entre la séquence primaire et les LS de HMM-SA27 et d'affirmer que le nombre élevé de celles-ci est adapté à une prédiction plus fine.
4. *l'analyse des types d'acides aminés privilégiés ou non dans les LS diverses* [29]. HMMS-SA27 dévoile que les LS désignées comme hélices alpha {a,A,V,W} préfèrent les acides aminés hydrophobes (méthionine, leucine, alanine) et chargés (arginine et acide glutamique) et aussi la glutamine. Par contre celles associées au brin bêta {X,L,M,N,T} favorisent les acides aminés : valine, tyrosine, isoleucine, phénylalanine et thréonine. Par ailleurs les lettres restantes associées aux boucles sont toutes distinguées entre elles. Ceci est en harmonie avec les préférences classiques des structures secondaires [92, 108, 139, 152, 177].
5. *l'analyse des boucles*, qui représentent les régions structurellement les plus variables et flexibles dans les protéines [27, 145]. En effet, elles sont importantes par leurs rôles majeurs pour la stabilité et la fonction de la protéine. HMM-SA27 fournit alors 18 LS pour décrire ces boucles

et extraire les motifs récurrents fortement conservés au cœur de ces régions [142].

6. *l'analyse des contacts entre protéines* [118]. L'étude montre que la description des contacts entre les protéines par les résidus de structures locales (décrit par HMM-SA27) impliqué dans ces contacts est plus précise que celle fournie par les types d'acides aminés impliqués.
7. *l'analyse des déformations lors des interactions entre les protéines* [119]. En effet l'étude détermine les régions qui subissent les interactions avant et après celles-ci. Les auteurs montrent que HMM-SA27 est capable de détecter les motifs structuraux communs, de la souche, impliqués dans l'interaction des deux protéines.
8. *l'identification d'un AS spécifique aux protéines porines*<sup>15</sup>. À l'aide de cet AS une description fine et détaillée de la composition des tonneaux bêta formés par ces protéines a été fournie par [117], afin de caractériser leurs séquences et leurs structures.
9. *l'amélioration de la reconnaissance des repliements des structures protéiques* à partir de séquences d'acides aminés grâce à l'utilisation des HMM [54].
10. *l'extraction des motifs structuraux* à partir des séquences 1D des structures 3D des boucles, grâce à un autre serveur web SA-Mot (*Structural Alphabet Motif*) [147] destiné à extraire la position des motifs structuraux récurrents et conservés au sein des boucles protéiques.
11. *l'identification des patterns structuraux fonctionnels* [148]. Cette étude a montré que les patterns détectés aident à prédire les sites de liaisons S-adenosyl-méthionine (SAM) et S-adenosyl-homocysteine (SAH)<sup>16</sup> des protéines. Elle permet aussi de prédire les bêta turns.
12. *le Déchiffrement de la structure 3D et de la déformation des structures secondaires grâce à l'analyse de conformation locale* [9]. L'étude démontre qu'il est pertinent d'explorer la déformation de la chaîne principale impliquée dans les interactions protéines-protéines. Ainsi, la conformation des structures secondaires a été analysée et détaillée grâce à HMM-SA27, ce qui a permis une meilleure description de la surface, du noyau et de l'interface des protéines en termes de forme et de déformation des structures secondaires.<sup>17</sup>
13. *la modélisation et la reconstruction tridimensionnelle des peptides* grâce à l'outil PEP-FOLD [120, 169]. Celui-ci est basé sur l'AS HMM-SA27 et est capable de modéliser et de prédire jusqu'à 56 résidus par peptide et de les rassembler entre eux pour proposer des conformations de plus basse énergie du peptide.
14. *l'analyse de la variabilité structurale d'une cible thérapeutique* [143]. L'outil « SA-conf » permet de déterminer la redondance d'information structurale dans les données des fichiers de la banque *Protein Data Bank* (PDB, [16]). Il a démontré son aptitude à détecter la variabilité liée à la fixation d'un ligand sur la protéine, ce qui permet d'identifier les résidus d'intérêt.
15. *l'analyse fine de l'asymétrie de la protéase de type II du Virus de l'immunodéficience humaine* [171].

---

15. Les porines sont des protéines membranaires qui constituent des canaux (tonneaux) donnant lieu à la diffusion de petites molécules hydrophiles via la membrane des cellules.

16. SAM et SAH sont des molécules associées à certains processus de méthylation et sont étudiées dans la recherche sur les médicaments antiviraux.

17. Les tendances de modification induites décrites dans cette étude devraient être des informations précieuses pour identifier et caractériser les régions soumises à de fortes contraintes structurales pour des raisons fonctionnelles.

En conclusion, HMM-SA27 fournit une description très précise des structures protéiques, en particulier des régions en boucles [146], qui jouent un rôle important dans la fonction protéique. Il a également été démontré qu'il était pertinent d'explorer la déformation de la chaîne principale impliquée dans les interactions protéines-protéines [9, 119] et de générer des conformations peptidiques 3D [104, 105].

### 1.3 Limites des AS

Bien que les AS existants à l'heure actuelle aient démontré leurs intérêts à travers de nombreuses applications importantes, ces alphabets n'ont pas été construits pour la prise en compte de l'incertitudes des structures protéiques (par exemple : données manquantes) ainsi que de la richesse des données et de leur flexibilité (les chaînes d'homomères multiples). En effet, le nombre des structures 3D ne cesse de s'accroître de façon exponentielle au niveau de la banque PDB. Celle-ci fournit donc un accès, en 2017, à environ 133 000 macromolécules biologiques (protéines, acides nucléiques et des glucides), parmi lesquelles certaines sont complexées avec des petits composants chimiques (molécules de solvant, des ions, des cofacteurs, des inhibiteurs et des médicaments). Ces données PDB contiennent une redondance considérable à la fois pour la séquence et la structure, par exemple, plus de la moitié partagent une identité de séquence d'au moins 95%. Même si cette redondance est considérée comme utile dans l'étude des familles de séquences homologues [72, 79], l'approche dominante pour l'exploration de données de la PDB est de considérer que la redondance est non informative [129]. Ce qui entraîne une réduction de l'exploitation de la variabilité des structures 3D. Pourtant, l'analyse de la redondance des protéines est d'une importance majeure pour comprendre la flexibilité des protéines. En effet, les protéines sont des macromolécules très flexibles, suite à leurs repliements 3D, et leurs propriétés dynamiques sont indispensables pour accomplir leurs fonctions. Ainsi, l'intégration de l'information de redondance de la PDB est nécessaire dans l'analyse des structures protéiques afin d'améliorer la compréhension de leurs flexibilités [115]. De nombreux fichiers PDB contiennent plusieurs chaînes. Par exemple, en 2015, les structures cristallisées non redondantes de protéines contiennent 7 972 monomères (correspondant à une chaîne de protéine individuelle qui ne fait pas partie d'un complexe), 9 206 homomères correspond à un complexe protéique formé à partir de l'autoassemblage d'un seul type de la sous-unité ou répliquations et 2 677 hétéromères [116]. Il est donc primordial de tenir en compte de cette redondance et de modéliser ces données à chaînes multiples. Le travail réalisé au cours de la présente thèse tient en compte de cette redondance et modélise les chaînes multiples.

Une autre difficulté, non gérée par les AS actuels, concerne les régions manquantes rencontrées lors de l'analyse des fichiers PDB. En effet, une étude faite par Gall et ses collaborateurs en 2007, sur la comparaison des séquences de structure 3D, déterminées par cristallographie aux rayons X, avec leurs séquences *fasta* correspondantes de la base de référence Uniprot, a montré que parmi 16 370 structures étudiées, 10% des fichiers PDB contiennent des régions de résidus consécutifs manquants de plus de 30 acides aminés, et 40% des protéines ont des régions entre 10 et 30 acides aminés de résidus manquants [70]. Ces régions manquantes ont un impact important sur la détermination d'un repliement précis des protéines. Cependant, ces régions manquantes n'ont pas été explicitement modélisées par différentes approches de AS existants.

## 1.4 Objectifs et plan de la thèse

Bien que de nombreuses applications et outils se sont développés autour de HMMSA-27 permettant une amélioration intéressante de nos connaissances sur l'organisation des structures tridimensionnelles protéiques. Cet AS présente des limites principalement liées à la prise en compte de l'incertitude des données de la PDB, HMM-SA27 ne donnant la possibilité de n'encoder qu'une structure tridimensionnelle complète (sans résidus manquants ou mal déterminés). Le but de cette thèse est d'améliorer HMM-SA27 pour construire un modèle capable de la prise en compte des régions manquantes dans les fichiers PDB, de prendre en compte l'incertitude et la richesse des données de la PDB due à leurs redondances.

Ce sujet de thèse a été financé par le projet « Initiative D'EXcellence (IDEX) » : *SAFlex Structural Alphabet Flexibility* de l'université Sorbonne Paris Cité (SPC). Il est à la croisée du *Drug Design* et de la modélisation mathématique. Il s'inscrit dans le contexte d'un projet très interdisciplinaire (biologie structurale, mathématiques et informatique). Ce projet de thèse a été codirigé par le Pr Anne-Claude Camproux du laboratoire Inserm MTi (Molécules Thérapeutique In silico, équipe *Computational approaches applied to pharmacological profiling*) à l'université Paris-Diderot et le Pr Grégory Nuel du laboratoire CNRS LPMA (Laboratoire de Probabilités et Modèles Aléatoires, équipe *Probability Statistical and Biology*) à l'université Pierre et Marie Curie. En effet, l'équipe du MTi a joué un rôle important en ce qui concerne la connaissance des structures protéiques et *Drug Design* et les AS et l'équipe du LPMA avec sa connaissance du HMM ainsi que la méthodologie mathématique pour atteindre les objectifs de cette thèse.

Nous proposons donc dans cette thèse un nouvel alphabet, améliorant l'alphabet HMM-SA27 [28], appelé SAFlex (*Structural Alphabet Flexibility*), dans le but de prendre en compte l'incertitude des données (données manquantes dans les fichiers de la banque PDB et la redondance des structures protéiques). L'objectif de cette thèse est de proposer d'une part un AS permettant une caractérisation fine et précise des structures 3D des protéines, à l'aide des HMM qui permettent de prendre en compte la logique issue de l'enchaînement des fragments structuraux, d'autre part, un AS qui intègre l'augmentation des conformations 3D des structures protéiques désormais disponibles dans la PDB. Cette caractérisation précise des conformations 3D et la prise en compte de la redondance des informations 3D disponibles a en effet un impact très important pour la modélisation de la conformation et de la variabilité des structures 3D, des boucles protéiques et des régions d'interface avec différents partenaires, impliqués dans la fonction des protéines.

Le manuscrit de thèse se présente sous la forme de cinq chapitres principaux et d'annexes. Le chapitre introductif présente la notion de AS et un bref état de l'art sur les différents AS existants, leurs applications et leurs limites notamment de l'AS préexistant HMM-SA27 [28].

Nous présentons dans le chapitre 2 le nouveau *parser* (analyseur) de la PDB (SAFlex-PDB), qui représente une contribution importante pour le fonctionnement du site web (chapitre 4). Ce chapitre présente tout d'abord les problèmes liés au format PDB, les *parsers* de PDB existants les plus connus, ainsi que leurs limites. Ensuite, nous expliquons les étapes de développement, d'implémentation, d'optimisation et de parallélisation du nouveau *parser*. Enfin, nous démontrons que notre *parser* a un intérêt sur le plan qualitatif (détection de diverses erreurs) et le plan quantitatif (rapidité et

parallélisation) en le comparant avec deux autres *parsers* très connus dans le domaine (Biopython et BioJava).

Le chapitre 3 présente les avancées méthodologiques du nouvel alphabet structural SAFlex obtenu en proposant un nouveau modèle d’encodage rigoureux et robuste, qui prend en compte la redondance des données et leurs incertitudes (données manquantes dans les fichiers PDB). Ces développements permettent en outre de prendre en compte l’incertitude des données en proposant trois options : le Maximum *a posteriori* (MAP), la distribution marginale *a posteriori* (POST) et le nombre effectif de lettres à chaque position donnée (NEFF). Enfin, le nouvel alphabet SAFlex fournit un encodage consensus à partir de différentes réplifications (chaînes multiples, monomères et homomères) d’une même protéine. Il permet ainsi la détection de la variabilité structurale de la protéine. Ce chapitre décrit ensuite le nouvel alphabet structural SAFlex et illustre certaines applications. Les avancées méthodologiques ainsi que l’obtention de l’alphabet SAFlex constituent la contribution principale de ce travail de thèse.

Le chapitre 4 présente la contribution concrète de cette thèse. Nous proposons aussi à la communauté scientifique un site web mettant en ligne ce nouvel alphabet structural SAFlex. Il présente la conception et l’implémentation de l’interface graphique du site web. Dans ce chapitre, les entrées et les résultats ainsi que les principales fonctionnalités du site sont explicités en détail en utilisant deux versions de l’alphabet structural SAFlex. Ce site web rend l’alphabet SAFlex et ses enjeux méthodologiques accessibles à la communauté scientifique.

Le dernier chapitre de conclusion clôture cette thèse et résume les avancées méthodologiques proposées et ouvre des perspectives sur le nouvel alphabet structural en cours de développement. Enfin, pour ne pas trop alourdir le contenu de cette thèse et faciliter sa lecture, les notions de statistiques et d’informatique essentielles utilisées dans ce mémoire ont été mises en annexe.





# Un nouveau parser de fichiers PDB

## Résumé

L'objectif principal de ce chapitre est de présenter le *parser* SAFlex-PDB, qui représente une contribution importante mise en œuvre au niveau du site web. Il contribue aussi à l'obtention de l'alphabet structural SAFlex.

Ce chapitre présente tout d'abord les problèmes du format PDB, les outils qui permettent d'analyser les fichiers PDB (*parsers* PDB) ainsi que leurs avantages et inconvénients. Compte tenu des inconvénients rencontrés avec les *parsers* existants (par exemple : un traitement inefficace des données manquantes dans les fichiers PDB), le développement d'un nouveau *parser* nous a semblé indispensable. Nous exposons plus particulièrement l'analyse et la modélisation informatique pour la conception de SAFlex-PDB. Nous développons ensuite minutieusement l'implémentation de l'outil. Puis nous expliquons la démarche de son optimisation et de sa parallélisation. En outre ce *parser* a été soumis à des tests comparatifs avec deux autres *parsers* très connus dans le domaine (Biopython et BioJava) en utilisant le jeu de données représentatif « PDBSelect ».

Les comparaisons présentées démontrent que l'outil a un intérêt aussi bien sur le plan qualitatif (détection de diverses erreurs) que quantitatif (rapidité et parallélisation).

## 2.1 Introduction

L'idée de créer l'archive PDB revient à un groupe de cristallographes, dont *Edgar Meyer*, *Gerson Cohen* et *Helen Miriam Berman* au début des années 1970 [12]. Leurs principaux objectifs étaient de collecter, normaliser et de partager les coordonnées atomiques des structures tridimensionnelles résolues par cristallographie [16]. En automne 1971, la PDB a été créée par une collaboration entre le laboratoire national de Brookhaven (BNL) et le centre de données cristallographiques de Cambridge (CCDC) avec uniquement sept structures résolues [98].

La banque PDB contient les structures de la plupart des protéines et des acides nucléiques impliqués dans les processus centraux de la vie, des ribosomes, des oncogènes, des cibles de médicaments, et même des virus entiers. Elle contient souvent plusieurs structures pour une molécule donnée, ou des structures

partielles, ou des structures qui ont été modifiées ou inactivées de leur forme native. Cette banque est gérée par une collaboration internationale de centres de données régionaux, appelée *Worldwide Protein Data Bank* (wwPDB), répartis sur les Etats-Unis, l'Europe et le Japon, formée en 2003, afin de veiller à ce que la PDB reste une archive mondiale, accessible au public, et uniforme [11].

Au fur et à mesure que la banque évolue, l'archive PDB, intègre d'autres éléments de données afin de mieux décrire la détermination de la structure en utilisant d'autres méthodes de résolution des macromolécules biologiques [13]. La plupart des structures biologiques résolues dans l'archive PDB sont déterminées par cristallographie aux rayons X. Chaque méthode de détermination des structures 3D a ses avantages et ses inconvénients, ainsi la cristallographie aux rayons X fournit plus de détails sur l'information atomique concernant tous les atomes présents dans le crystal. Néanmoins la cristallographie demeure difficile à réaliser et impose des limitations sur le type de protéines à étudier. En revanche la résonance magnétique nucléaire (RMN) semble la meilleure méthode pour étudier les structures atomiques des protéines flexibles, car elle propose un ensemble de modèles atomiques pour la macromolécule biologique. Les modèles de structures dans cet ensemble sont très similaires dans les régions rigides, par contre elles sont légèrement différentes dans les régions flexibles. Cependant cette méthode est limitée uniquement sur des protéines de petites ou moyennes tailles. La microscopie électronique (EM), remédie aux inconvénients de la cristallographie et de la RMN, en effet elle permet de déterminer les structures de macromolécules complexes de grande taille, et donc elle est très intéressante pour les complexes de ribosomes et les tRNA. Malgré cela, les expériences de la microscopie électronique ne permettent pas de voir chaque atome de la structure, alors pour en déterminer le modèle atomique, une combinaison des informations de cristallographie aux rayons X ou par RMN est nécessaire pour régler les détails atomiques.

Ce chapitre aborde les notions de base concernant les formats des fichiers de données de la banque PDB et plus particulièrement le format PDB. Il présente également quelques *parsers* PDB existants ainsi que leurs avantages et inconvénients. Cette analyse conduit à la nécessité de développer un nouveau *parser* plus adapté au format PDB et performant. Le chapitre reprend aussi l'ensemble des phases du développement de notre *parser* PDB. La phase d'implémentation du *parser*, son optimisation, et sa parallélisation sont détaillées. Enfin il présente les tests comparatifs effectués avec les *parsers* Biopython et Biojava et leurs résultats.

## 2.2 Les fichiers de données de la banque PDB

Toutes les structures tridimensionnelles (3D) de la banque PDB sont décrites dans des fichiers de données de différents formats. Ainsi cette banque propose actuellement trois formats : le fichier PDB, le fichier mmCIF et le fichier XML. Le premier format proposé en 1971 était le format PDB, conçu pour des biologistes de la branche structurale, son contenu était simple, lisible par l'humain et adapté pour être stocké sur des cartes perforées [15]. Chaque fichier PDB contient à la base des coordonnées atomiques et un ensemble d'informations complémentaires, qui décrivent la macromolécule biologique ainsi que des informations spécifiques liées à la méthode de résolution utilisée (pour déterminer ses coordonnées atomiques dans l'espace tridimensionnel). On trouve alors le nom des molécules, des informations sur la structure primaire et secondaire, des détails sur la collecte des données et les

citations bibliographiques, ..., etc. Un fichier de format PDB n'est rien d'autre qu'un fichier texte dans lequel les informations contenues sont regroupées par sections. Pour plus de détails sur la structure du fichier PDB et son format nous vous invitons à consulter le lien suivant : [https://www.rcsb.org/pdb/static.do?p=file\\_formats/pdb/index.html](https://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html).

Bien que le format était très populaire et utilisé par un grand public, il posa pas mal de problèmes aux chercheurs, car la description de la structure était ambiguë<sup>1</sup>, et le format de carte perforée limitait chaque ligne à 80 caractères, limitant ainsi le nombre et les noms des atomes, des résidus et des chaînes. En effet, un fichier de format PDB peut prendre en charge au maximum 99 999 atomes et 62 chaînes ce qui nécessite plusieurs fichiers PDB pour décrire les structures importantes et complexes tels que les ribosomes afin de représenter leurs nombreux atomes et les chaînes [13]. Malgré les efforts faits par la wwPDB pour le contrôle, la vérification et la validation des structures déposées au niveau de la banque PDB, on trouve plusieurs erreurs à l'intérieur des fichiers PDB qui seront détaillées dans la suite. Nous nous intéressons aux problèmes liés au format PDB, nous citons donc quelques uns et des erreurs liées au format PDB, par ailleurs, la démarche pour identifier les erreurs de détermination des structures 3D a été décrite dans le chapitre 10 du livre [37].

Un problème qui revient souvent est celui du manque des résidus et/ou atomes dans la séquence résolue, qui ne sont pas souvent déclarés, par les auteurs, sachant que les enregistrements « REMARK 465 » et « REMARK 470 » permettent de lister les résidus et les atomes manquants dans une chaîne macromoléculaire. À titre d'exemple, le fichier PDB « 1OX1 » présente un trou de deux résidus « 35 » et « 36 » qui ne sont pas mentionnés dans l'enregistrement « REMARK 465 » par les auteurs comme illustré dans le listing ci-dessous :

---

```
REMARK 465
REMARK 465 MISSING RESIDUES
REMARK 465 THE FOLLOWING RESIDUES WERE NOT LOCATED IN THE
REMARK 465 EXPERIMENT. (M=MODEL NUMBER; RES=RESIDUE NAME; C=CHAIN
REMARK 465 IDENTIFIER; SSSEQ=SEQUENCE NUMBER; I=INSERTION CODE.)
REMARK 465
REMARK 465      M RES C SSSEQI
REMARK 465      TYR B      11
REMARK 500
REMARK 500 GEOMETRY AND STEREOCHEMISTRY
```

DBREF	1OX1	A	16	245	UNP	P00760	TRY1_BOVIN	21	243
-------	------	---	----	-----	-----	--------	------------	----	-----

ATOM	127	CD1	LEU	A	33	15.651	-9.945	33.599	1.00	17.92	C
ATOM	128	CD2	LEU	A	33	13.479	-11.191	33.559	1.00	19.50	C
ATOM	129	N	ASN	A	34	11.469	-8.956	37.613	1.00	18.38	N
ATOM	130	CA	ASN	A	34	11.001	-8.234	38.788	1.00	19.40	C
ATOM	131	C	ASN	A	34	11.648	-8.661	40.099	1.00	19.63	C

---

1. la description qui était faite par des biologistes ne respecte pas les différents bases de l'informatique : par exemple plusieurs identifiants pour le même atome.

ATOM	132	O	ASN	A	34	11.828	-9.849	40.351	1.00	20.79	O
ATOM	133	CB	ASN	A	34	9.469	-8.436	38.828	1.00	19.60	C
ATOM	134	CG	ASN	A	34	8.784	-7.691	39.963	1.00	21.76	C
ATOM	135	OD1	ASN	A	34	8.929	-8.042	41.135	1.00	20.72	O
ATOM	136	ND2	ASN	A	34	8.011	-6.664	39.611	1.00	21.08	N
ATOM	137	N	SER	A	37	12.003	-7.691	40.933	1.00	19.94	N
ATOM	138	CA	SER	A	37	12.593	-7.977	42.248	1.00	21.17	C
ATOM	139	C	SER	A	37	11.913	-7.025	43.222	1.00	21.97	C
ATOM	140	O	SER	A	37	12.572	-6.355	43.996	1.00	23.28	O
ATOM	141	CB	SER	A	37	14.123	-7.730	42.229	1.00	22.86	C
ATOM	142	OG	SER	A	37	14.429	-6.364	41.882	1.00	26.05	O

**Listing 2.1** – Exemple tiré du fichier PDB 1OX1

En outre, une étude faite par Gall et ses collaborateurs en 2007, sur la comparaison des séquences de structure 3D, déterminées par cristallographie aux rayons X, avec leurs séquences fasta correspondantes de la base de référence Uniprot, a montré que parmi 16 370 structures étudiées, 10% des fichiers PDB contiennent des régions de résidus consécutifs ambigus manquants de plus de 30 acides aminés, et 40% des protéines ont des régions entre 10 et 30 acides aminés de résidus ambigus ou manquants [70].

De plus la numérotation des résidus de la structure résolue est généralement arbitraire et ne correspond pas à la numérotation de la séquence dans la banque de référence Uniprot car certains auteurs préfèrent attribuer le numéro « 1 » au premier résidu résolu de la structure même s'il ne correspond pas au premier résidu de la séquence de référence fasta, tel est le cas pour les fichiers PDB « 1ARB » et « 7PTI », où les premiers résidus résolus notés « 1 » sont associés au résidus « 206 » et « 36 » de la séquence fasta respectivement.

DBREF	1PFK	A	0	319	UNP	P0A796	K6PF1_ECOLI	1	320
DBREF	1PFK	B	0	319	UNP	P0A796	K6PF1_ECOLI	1	320

ATOM	1	N	MET	A	0	29.153	15.048	31.593	1.00	81.05	N
ATOM	2	CA	MET	A	0	27.731	15.334	31.887	1.00	80.28	C
ATOM	3	C	MET	A	0	26.935	14.627	30.789	1.00	74.98	C
ATOM	4	O	MET	A	0	27.544	13.914	29.973	1.00	74.90	O
ATOM	5	CB	MET	A	0	27.478	16.830	32.004	1.00	84.76	C
ATOM	6	CG	MET	A	0	26.332	17.138	32.933	1.00	89.29	C
ATOM	7	SD	MET	A	0	26.340	18.916	33.376	1.00	93.60	S
ATOM	8	CE	MET	A	0	27.977	19.068	34.125	1.00	92.98	C
ATOM	9	N	ILE	A	1	25.651	14.847	30.782	1.00	68.62	N
ATOM	10	CA	ILE	A	1	24.699	14.268	29.831	1.00	63.20	C

**Listing 2.2** – Exemple tiré du fichier PDB 1PFK

D'autres auteurs attribuent le numéro « 0 » au premier résidu résolu de la chaîne comme dans le fichier PDB « 1PFK » illustré par le listing (2.2).

On trouve même des auteurs qui attribuent un numéro aléatoire au premier résidu résolu de la chaîne, car ce numéro dépend du contexte si on veut comprendre la motivation des auteurs. Ce qui n'est pas possible à gérer si on souhaite automatiser la description de la structure 3D. À titre d'exemple les fichiers « 1BET » et « 9WGA » qui donnent le numéro « 10 » et « 2 » au résidu « 131 » et « 29 » respectivement de leurs chaînes de référence fasta. Le listing (2.3) ci dessous, montre que les auteurs ont associé « 10 » au résidu « 131 » de la séquence fasta de la chaîne « A » du fichier « 1BET » :

---

DBREF	1BET	A	10	116	UNP	P01139	NGF_MOUSE	131	237
<hr/>									
ATOM	1	N	GLY	A	10	-12.055	21.656	13.213	1.00 65.66 N
ATOM	2	CA	GLY	A	10	-10.982	20.890	13.825	1.00 61.16 C
ATOM	3	C	GLY	A	10	-9.955	21.950	14.122	1.00 58.46 C
ATOM	4	O	GLY	A	10	-10.350	23.123	14.067	1.00 61.24 O
ATOM	5	N	GLU	A	11	-8.716	21.587	14.391	1.00 55.21 N
ATOM	6	CA	GLU	A	11	-7.716	22.567	14.735	1.00 52.18 C
ATOM	7	C	GLU	A	11	-6.847	22.942	13.538	1.00 49.81 C
ATOM	8	O	GLU	A	11	-6.435	22.075	12.765	1.00 50.12 O
ATOM	9	CB	GLU	A	11	-6.926	21.975	15.854	1.00 53.84 C
ATOM	10	CG	GLU	A	11	-7.898	21.734	17.002	1.00 57.31 C

---

**Listing 2.3** – Exemple tiré du fichier PDB 1BET

Aussi, parmi les erreurs, commises par les auteurs des fichiers PDB, qui sont très répandues :

- l'oubli de l'enregistrement « TER » qui marque la fin de la chaîne moléculaire ce qui génère des chaînes très longues erronées, par exemple dans le fichier 4OE8 il y a une absence de « TER » entre la chaîne 'B' et 'C'.
- l'utilisation de l'enregistrement « ATOM » au lieu de « HETATM », pour des composés qui ne forment pas les chaînes moléculaires tels que : les molécules d'eau ou l'hème ;
- l'alignement incorrect des noms d'atomes dans les chaînes moléculaires : chaque nom d'atome est composé d'un symbole atomique, par exemple « O », justifié à droite dans les colonnes 13-14, suivi par un caractère identifiant, tel que « E », justifié à gauche dans les colonnes 15-16. L'exemple ci-dessous montre clairement cette erreur<sup>2</sup>, se trouvant à la ligne 418 pour l'atome n° 37 ;

---

<u>411</u>	ATOM	30	CA	GLU	B	6	9.719	10.844	17.662	1.00 37.17	C
<u>412</u>	ATOM	31	C	GLU	B	6	8.639	11.496	16.812	1.00 35.58	C
<u>413</u>	ATOM	32	O	GLU	B	6	7.462	11.172	16.967	1.00 35.95	O
<u>414</u>	ATOM	33	CB	GLU	B	6	10.031	11.707	18.891	1.00 39.30	C
<u>415</u>	ATOM	34	CG	GLU	B	6	8.840	12.000	19.794	1.00 40.33	C
<u>416</u>	ATOM	35	CD	GLU	B	6	8.288	10.778	20.517	1.00 42.45	C
<u>417</u>	ATOM	36	OE1	GLU	B	6	8.979	9.745	20.628	1.00 43.65	O
<u>418</u>	ATOM	37	OE2	GLU	B	6	7.137	10.857	20.992	1.00 47.45	O

---

2. Les lignes de l'exemple sont tirées du fichier PDB 5E80, mais l'erreur qu'on voit sur le listing (2.4) n'existe pas dans ce fichier, c'était fait uniquement pour montrer le type d'erreur

---

<u>419</u>	ATOM	38	N	ARG B	7	9.026	12.415	15.926	1.00	32.61	N
<u>420</u>	ATOM	39	CA	ARG B	7	8.055	13.030	15.032	1.00	30.53	C

---

**Listing 2.4** – Exemple tiré du fichier PDB 5E80, qui montre qu'à la ligne 417, il y a un alignement incorrect du nom atome oxygène OE1

- duplication de l'atome dans un résidu en lui attribuant deux numéros différents, dans l'exemple suivant deux atomes « 30 » et « 33 » dans le résidu « GLU » sont nommés « CA » :

---

<u>411</u>	ATOM	30	CA	GLU B	6	9.719	10.844	17.662	1.00	37.17	C
<u>412</u>	ATOM	31	C	GLU B	6	8.639	11.496	16.812	1.00	35.58	C
<u>413</u>	ATOM	32	O	GLU B	6	7.462	11.172	16.967	1.00	35.95	O
<u>414</u>	ATOM	33	CA	GLU B	6	10.031	11.707	18.891	1.00	39.30	C
<u>415</u>	ATOM	34	CG	GLU B	6	8.840	12.000	19.794	1.00	40.33	C
<u>416</u>	ATOM	35	CD	GLU B	6	8.288	10.778	20.517	1.00	42.45	C
<u>417</u>	ATOM	36	OE1	GLU B	6	8.979	9.745	20.628	1.00	43.65	O
<u>418</u>	ATOM	37	OE2	GLU B	6	7.137	10.857	20.992	1.00	47.45	O
<u>419</u>	ATOM	38	N	ARG B	7	9.026	12.415	15.926	1.00	32.61	N
<u>420</u>	ATOM	39	CA	ARG B	7	8.055	13.030	15.032	1.00	30.53	C

---

**Listing 2.5** – Exemple tiré du fichier PDB 5E80, qui montre qu'à la ligne 411 et 414, les deux atomes portent le même nom CA.

Compte tenu de toutes ces erreurs sur le format PDB, sachant que l'on n'a cité que quelques unes, il est important de noter que le traitement informatique des fichiers PDB est une tâche ardue, et donc la modélisation des structures 3D et leurs analyses s'annoncent plus complexe que prévu.

Notre travail dans cette thèse consiste, entre autre, à modéliser les données des fichiers PDB, puis à les traiter afin de les analyser, ce qui permettra de tirer des conclusions intéressantes à propos des protéines. Afin de remédier à ces lacunes (expliquées en détail à la section 2.2 à la page 12), *the International Union for Crystallography* (IUCr) a chargé un comité dirigé par Paula Fitzgerald, en 1990, de trouver une meilleure représentation des structures macromoléculaires ce qui a permis, en 1996, à un nouveau format appelé *the Macromolecular Crystallographic Information File* (mmCIF) de voir le jour. Ainsi le premier dictionnaire de données du mmCIF contenait plus de 3 000 termes utilisés pour décrire une structure macromoléculaire biologique et ses détails expérimentaux [13, 80]. Le format mmCIF est donc lisible par machine où chaque élément de données est complètement défini sans ambiguïté ainsi que les relations entre eux, en plus il n'y a pas de restrictions sur la taille de la macromolécule et il peut être utilisé pour créer une base de données relationnelle [13]. Depuis le wwPDB a continué à élargir le dictionnaire de mmCIF pour inclure des termes pour la description des structures déterminées par RMN et 3DEM (*3D Electron Microscopy*) ce qui a donné lieu au dictionnaire étendu PDBx. Grâce à ce dernier de grandes structures peuvent être décrites sans aucune difficulté car il a la même syntaxe que mmCIF mais contient toutes les définitions nécessaires pour traiter les données qui font maintenant partie de la banque PDB [181].

Néanmoins le format mmCIF n'est pas compatible avec les technologies du web sémantique, alors pour combler cette lacune, la wwPDB a fourni également d'autres formats de données, tels que

PDBML/XML [181], qui représente une conversion du fichier mmCIF en fichier XML<sup>3</sup> (*Extensible Markup Language*), et *Resource Description Framework* (PDB/RDF), qui représente à son tour la conversion du fichier PDBML dans le format RDF<sup>4</sup> [13, 97]. Ce dernier devient un format standard pour le web sémantique qui facilite l'échange et l'intégration des données provenant de diverses sources sur le web [97].

## 2.3 Les parsers PDB existants

Les premiers *parsers* PDB ont commencé à voir le jour dans les années 1990, permettant ainsi d'ouvrir de nouvelles perspectives d'évolution de la bioinformatique structurale. Ainsi plusieurs *parsers* PDB ont été développés en utilisant différents langages de programmation. La plupart d'entre eux ont été implémentés en langage C [95] ou C++ [167], tels que : PDBLib [35], BALL [83, 99], Protein Library (<http://protlib.uchicago.edu/index.html>), EMBOSS [149], Victor [85], OpenStructure [17], MMDB [112, 113], BiopLib [136], ESTBL [110] et CCP4 [137]. D'autres *parsers* PDB, ont été développés en utilisant des interpréteurs tels que les langages Python [168], Perl [14] voire Ruby [126], à titre d'exemple, BioPython [36], BioPerl [165] et BioRuby [77]. En revanche d'autres développeurs ont préféré utiliser un langage multiplate-forme tel que le Java [76], avec lequel a été implémenté BioJava [140]. Les tous premiers *parsers* ont été développés sous forme de fonctions, mais la majorité d'entre eux sont proposés soit en librairie ou framework ou sous forme de modules.

Dans la suite de cette section, je vais présenter un certain nombre de *parsers* représentatifs en fonction de leurs langages de programmation et par ordre chronologique de leurs apparitions.

### 2.3.1 PDBLib (1994)

PDBLib (C++ *Macromolecular Class Library*) [35] est une librairie extensible implémentée en langage C++ orienté objet, pour représenter la structure tridimensionnelle des macromolécules biologiques. Elle contient 129 classes qui décrivent les caractéristiques structurales des protéines, de l'ARN, de l'ADN et des complexes biologiques.

Les classes fournies par PDBLib sont organisées en quatre catégories :

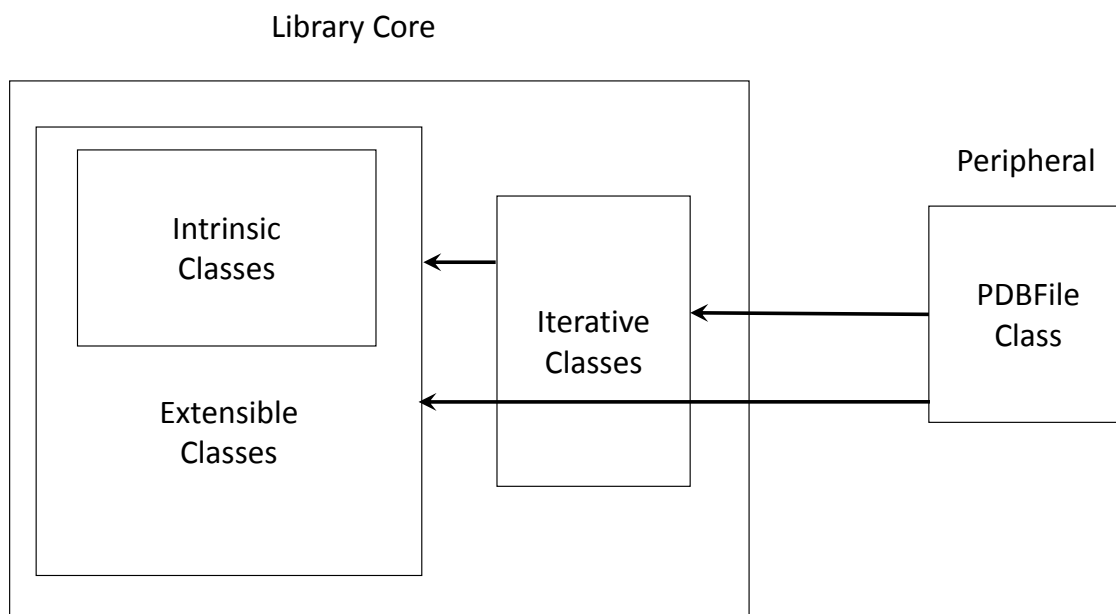
1. les classes intrinsèques qui modélisent la macromolécule, elles contiennent la partie essentielle de la mise en œuvre de la librairie ;
2. les classes extensibles qui facilitent l'extensibilité de la librairie ;
3. les classes itératives qui fournissent un accès et un déplacement facile entre les objets qui permettent de représenter la structure macromoléculaire ;
4. la classe « PDBFile » qui charge un fichier PDB dans la mémoire pour une représentation orientée objet de la structure macromoléculaire.

---

3. Le langage de balisage extensible est un langage de balisage générique qui permet de structurer des données afin qu'elles soient lisibles aussi bien par les humains que par des programmes de toutes sortes.

4. Le Resource Description Framework (RDF) est un langage pour représenter des informations sur les ressources du web.



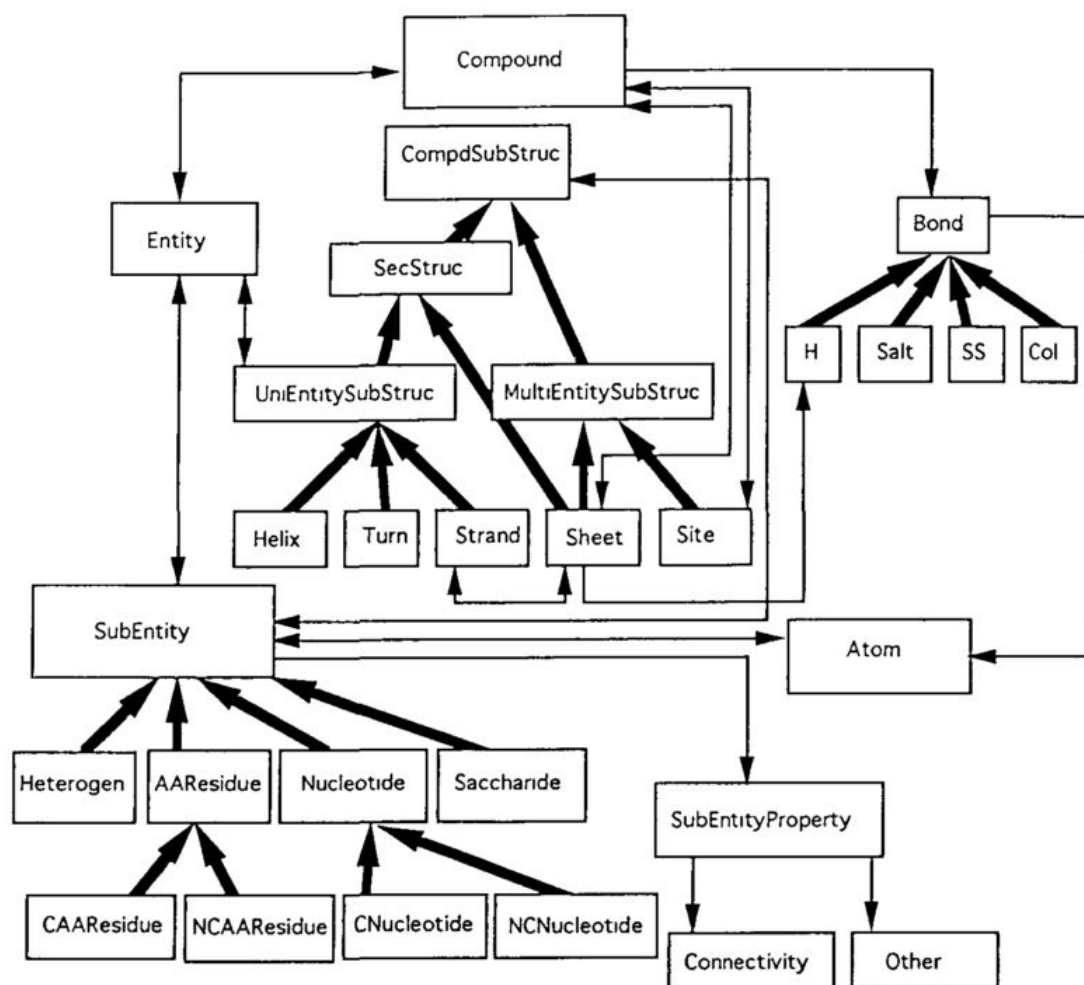


**Figure 2.1** – Classification globale des classes PDBlib et des relations entre elles  
Schéma tiré de [35].

Les trois premières classes forment le noyau de la librairie, alors que la quatrième classe « PDBFile » ne fait pas partie du noyau de la librairie, mais représente une interface entre les utilisateurs et le noyau (voir la figure (2.1)). Elle a été développée dans le but de faciliter aux utilisateurs de la librairie, qui souhaitent faire des applications biologiques, la lecture des fichiers PDB.

La classe « PDBFile » charge un fichier PDB selon la structure de données définie par les classes PDBlib. Elle se compose d'un ensemble de fonctions qui interprètent le fichier PDB et génèrent la séquence des sous-entités appropriées. Une structure primaire d'une macromolécule biologique est construite sur la base des enregistrements PDB « SEQRES », puis des informations des enregistrements PDB « ATOM » et « HETATM » afin de conserver autant d'informations que possible.

Les classes *intrinsèques* et *extensibles* contenues dans PDBlib ont été développées pour représenter et manipuler des composants macromoléculaires, comme les chaînes polypeptidiques ou les brins d'ADN, les structures secondaires, les sites fonctionnels, les résidus, les nucléotides, et les atomes.



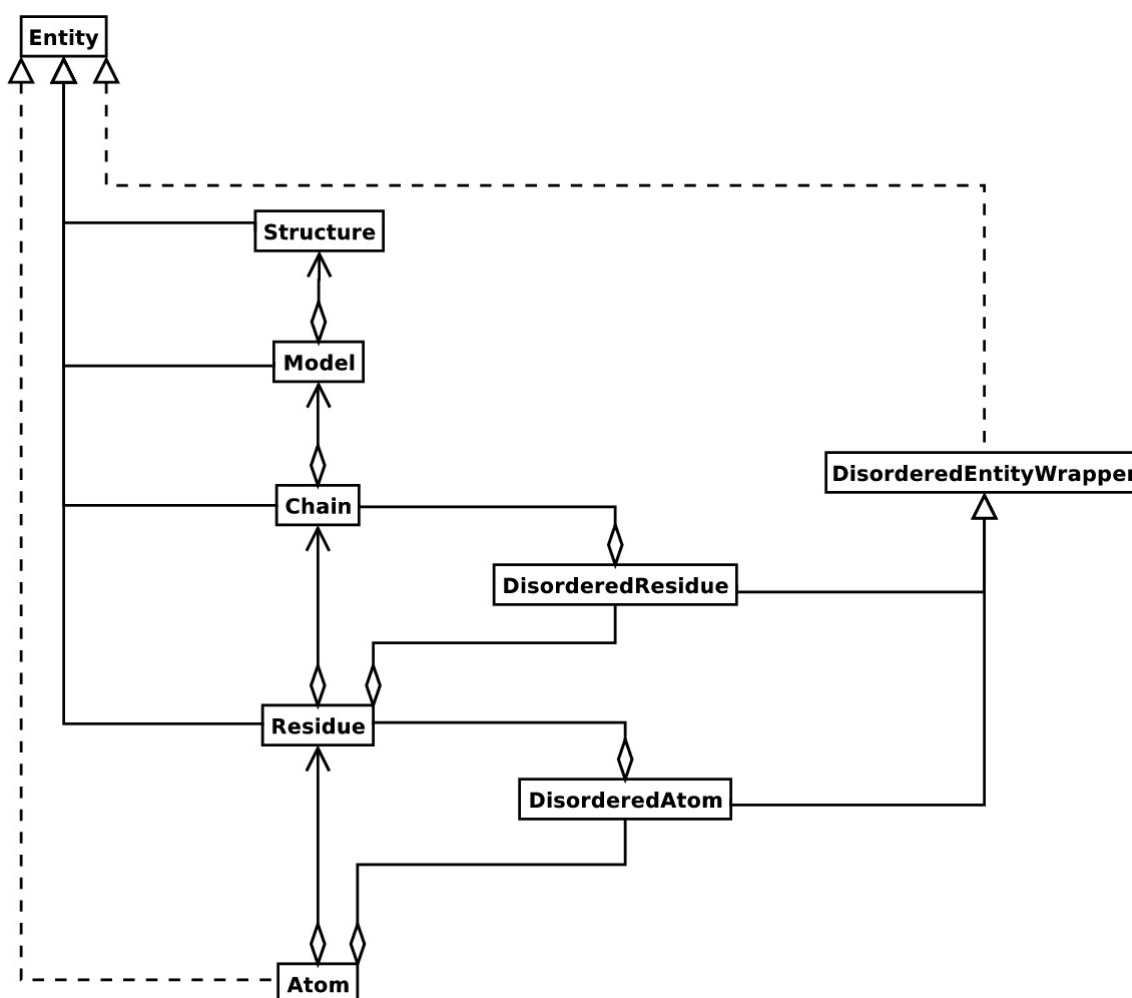
**Figure 2.2 – Représentation conceptuelle de la relation entre les classes PDBlib**  
Schéma tiré de [35].

La figure (2.2) illustre la représentation conceptuelle des relations entre les classes PDBlib d'une structure macromoléculaire. Les objets et les classes sont représentés par des rectangles et les flèches représentent les relations. Une flèche mince représente une relation d'appartenance alors qu'une flèche épaisse représente une relation d'héritage. Deux flèches minces bidirectionnelles indiquent que la relation peut être établie dans les deux sens.

Une structure macromoléculaire est représentée par la classe « compound », constituée d'autres composants de type classe : *entity*, *sheet*, *site*, *bond* ainsi que d'autres composants. La classe *entity* contient des informations telles que la liste des sous-entités (les résidus, les nucléotides, les hétérogènes) et la liste des structures secondaires. Les informations concernant les atomes, comme les coordonnées cartésiennes et le B facteur, sont des données de l'objet « Atom ». Chaque objet « Atom » a une relation avec chaque objet des sous-entités.

### 2.3.2 Biopython (2003)

Le projet Biopython (*Python tools for computational biology*) [36], créé en 1999, pour développer des outils bioinformatiques orientés objets, écrits en langage Python, à libre accès pour le public. Il propose en 2003 un ensemble de modules réutilisables, qui traitent et gèrent les données des structures macromoléculaires biologiques [81].



**Figure 2.3 – Diagramme UML de la structure de données « SMCRA ».**

Les lignes complètes avec des flèches triangulaires indiquent l'hérédité, les lignes complètes avec des diamants indiquent l'agrégation et les lignes en pointillés indiquent l'interface réalisation. Schéma tiré de [81].

Biopython, fournit la classe « Structure », représentée à l'aide d'une structure de données hiérarchique appelée « SMCRA » (Structure / Model / Chain / Residue / Atom), illustrée par la figure (2.3)<sup>5 6</sup>. Cette structure de données lui permet un accès facile et pratique aux coordonnées atomiques dans un fichier PDB. En effet, un objet enfant (Atom, Residue, Chain ou Model) peut tou-

5. l'agrégation permet de définir une entité comme étant liée à plusieurs entités de classe différentes.

6. une relation de réalisation entre deux éléments, est une relation dans laquelle un élément (le client) réalise (ou exécute) le comportement que l'autre élément (le fournisseur) spécifie.

jours être extrait de son parent (Residue, Chain, Model, Structure) en utilisant un identifiant comme une clé, ces identifiants sont : « model », « chain », « residue » et « atom ».

Les modules proposés pour analyser les fichiers PDB, tiennent compte de certaines ambiguïtés ou non respect de la syntaxe du format PDB présents dans les fichiers. Ainsi ces problèmes communs sont identifiés et provoquent une exception, d'où l'intérêt d'utiliser le drapeau « PERMISSIVE » (« PERMISSIVE » est initialisé à 0, valeur par défaut). Les erreurs peuvent être des résidus multiples ou désordonnés, ainsi que des atomes qui ont le même identifiant ou qui sont désordonnés. Le *parser* Biopython a été évalué par l'analyse de 5 405 fichiers PDB de la base de données « PDBSelect » [86], qui contient la liste des structures de protéines représentatives de la banque PDB. Bien que l'analyse de tous les fichiers a pris 3 heures et 50 minutes sur un ordinateur de 1.6 GHz, en moyenne environ 2.5 secondes par structure, le drapeau « PERMISSIVE » de l'analyseur a été fixé à « 1 », cela révèle que quelques cas intéressants et non triviaux ont généré une exception, accompagnée d'un avertissement.

### 2.3.3 BioJava (2008)

BioJava [87, 140] a été conçu en 1999 par Thomas Down et Matthew Pocock comme une "*Application Programming Interface*" (API) pour simplifier le développement de logiciels de bioinformatiques en utilisant le langage de programmation Java, lui permettant ainsi de fonctionner sur toutes les plates-formes (Windows, linux, Mac OS X, ..., etc.). Il a depuis évolué pour devenir un framework composé de plusieurs modules indépendants pour effectuer de nombreuses tâches bioinformatiques. L'objectif du projet BioJava est de faciliter la réutilisation du code et de fournir des implémentations qui sont faciles à relier à des scripts et des applications externes. Il est hébergé par la Fondation *Open Bioinformatics* (OBF, <http://www.open-bio.org>).

Au cours des dernières années, une grande partie du code source a été réécrite, donnant ainsi naissance à BioJava 3 [140]. Cette librairie Biojava 3 fournit désormais plusieurs modules indépendants :

1. le module de base ;
2. les modules des structures protéiques ;
3. les modules génome et séquençage ;
4. le module alignement ;
5. le module « ModFinder » ;
6. le module des propriétés des acides aminés ;
7. le module du désordre protéique ;
8. le module d'accès web service ;

Ces modules permettent d'analyser l'ADN, l'ARN, les protéines, et les propriétés d'acides aminés, ils prédisent des régions désordonnées dans les protéines, comme ils fournissent aussi des outils pour la comparaison structurale des protéines, et pour l'alignement multiple des séquences appariées. Ils offrent aussi des *parsers* pour des fichiers de formats courants (PDB, mmCIF) et des interfaces légères à d'autres projets qui se spécialisent dans des outils de visualisation des macromolécules biologiques en 3D.

Le module de base fournit des classes pour modéliser les nucléotides et les acides aminés. Afin d'améliorer la convivialité du framework pour les biologistes, différentes classes pour les types de séquences, telles que l'ADN et les protéines ont été intégrées dans le module de base. Les modules de structures de protéines fournissent des outils pour représenter et manipuler des structures biomoléculaires 3D, pour la comparaison de la structure des protéines et des *parsers* pour les fichiers de formats PDB et mmCIF qui permettent le chargement des données de structure dans un modèle de données réutilisable. Le module « ModFinder » fournit de nouvelles méthodes pour identifier et classer les modifications des protéines au sein des structures 3D des protéines. Le module propriétés des acides aminés a pour but de fournir une gamme de propriétés physico-chimiques précises pour les protéines.

BioJava inclut aussi un module de prédiction RONN (*Regional Order Neural Network*) [185] pour prédire les régions désordonnées de protéines.

### 2.3.4 BiopLib (2015)

« BiopLib » [136] est une librairie implémentée en C, son noyau représente un *parser* PDB fiable capable de charger et traiter des fichiers PDB et PDBML. Elle permet aux utilisateurs de gérer les données PDB comme une simple liste d'atomes, ou sous une forme structurée en utilisant des chaînes, des résidus et des atomes. L'implémentation de la librairie « BiopLib » a commencé à la fin des années 1980 et a continué de s'améliorer au cours des 25 dernières années en termes de normalisation, de documentation du code et de manipulation des structures macromoléculaires ainsi que la prise en compte des fichiers de format PDBML.

BiopLib propose de nombreuses fonctions importantes pour :

- lire et écrire les fichiers PDB et les fichiers PDBML ;
- tenir en compte les structures 3D proposées avec plusieurs facteurs « occupancy » ;
- manipuler et modifier la liste des coordonnées atomiques ;
- rechercher les résidus et les atomes ;
- calculer les distances entre deux points, les angles, RMSD, ... etc.

Cette librairie est intégrée à l'outil BiopTools [136], qui propose plusieurs fonctionnalités telles que :

- extraction d'une séquence du fichier PDB ;
- sélection des atomes en fonction de leurs types ;
- sélection des résidus ;
- extraction d'une chaîne bien déterminée ;
- ajout ou suppression des atomes hydrogènes ;
- rotations et translations ;
- calcul des distances entre les résidus ou atomes ;
- calcul des angles de torsion du squelette des protéines ;

### 2.3.5 Les avantages et inconvénients des parsers PDB

La majorité des *parsers* PDB qui ont été proposés sont fournis soit sous forme de librairies ou de modules, ce qui les rend extensibles, plus facile à installer et à réutiliser par d'autres applications ou

logiciels. Cependant ces *parsers* PDB, héritent des avantages ainsi que des inconvénients des langages de programmation avec lesquels ils ont été codés. La plupart des *parsers* PDB implémentés en C ou C++, sont très performants car les instructions du C sont toutes proches de celles du processeur. Ces *parsers* sont extensibles et disponibles sous formes de bibliothèques compilées et bien maintenues tels que : Protein Library, Victor, OpenStructure et BALL, par contre ils sont limités dans leur portabilité et leur souplesse, en effet pour les rendre souples il sera nécessaire de paramétrer sans cesse les critères utilisés. Les *parsers* PDB récents codés principalement en langage Python, ou Perl ou Ruby, présentent l'avantage d'un code plus facile à utiliser mais relativement lent en exécution en raison de l'utilisation d'un interpréteur de commandes. Le plus souvent, le langage Perl est utilisé pour écrire des programmes de petite taille (<500 lignes), développés par un petit groupe ou un individu. En revanche, Python présente une clarté et une concision qui attire les débutants, et leurs permet de développer aisément des codes plus importants dans le domaine bioinformatique. Les *parsers* PDB implémentés avec le langage Java tels que BioJava et STRAP [75], offrent l'avantage d'être multi-plate-formes, ce qui les rend très facilement portables sur plusieurs systèmes d'exploitation. En effet le langage Java reprend la syntaxe du langage C++ mais plus améliorée ce qui lui permet d'être très utilisé par les développeurs, notamment par ceux qui veulent entreprendre une carrière en bioinformatique.

Indépendamment du langage de programmation avec lesquels les *parsers* ont été développés, on constate que certains d'entre eux ont été proposés il y a un certain temps, ce qui ne leur a pas permis de profiter des nouvelles fonctionnalités du langage en terme de performance, de souplesse et de généricité<sup>7</sup>. Ainsi les *parsers* tels que BTL [135] et PDBlib n'ont pas été maintenus à jour. On trouve aussi des *parsers* PDB tels que MMDB (développé par RCSB) et CCP4, qui sont très spécifiques, ce qui les rend plus difficiles à utiliser par d'autres applications et logiciels du fait de leurs complexités. Les *parsers* BioPerl, BioPython et la boîte à outils MMTK [84] sont des outils puissants et librement disponibles. Ils fournissent des modules pour charger des structures PDB mais n'offrent pas de possibilités pour la traitement et l'analyse des données. Une lacune principale de BioPython est le fait que ses modules ne traitent que les coordonnées atomiques des macromolécules sans prendre en compte les informations concernant les macromolécules biologiques telles que celles présentes dans l'enregistrement « header » du fichier PDB. Quant au *parser* BioRuby, il fournit une structure de données pour le format PDB et un certain nombre de fonctions pour l'identification des enregistrements, mais peu pour l'analyse des données. Par ailleurs, le *parser* PDB BALL nécessite plusieurs composants externes. De même pour PDBLib, qui se base sur l'utilisation d'une base de données orientée objet commerciale pour la réalisation des structures macromoléculaires. En outre, EMBOSS, met l'accent beaucoup plus sur les séquences biologiques mais il ne fournit pas assez de fonctionnalités pour le traitement des structures 3D macromoléculaires. Quant au ESBTL, il est léger et ne nécessite pas des bibliothèques externes, pourtant il est conçu beaucoup plus pour les calculs géométriques sur les structures 3D.

Enfin le *parser* PDB BioJava présente une structure modulaire capable de charger des fichiers PDB et mmCIF et de les manipuler, il assure aussi la gestion des dépendances externes, permettant ainsi l'utilisation de bibliothèques externes sans compliquer excessivement la procédure d'installation pour

---

7. la généricité consiste à définir des algorithmes identiques opérant sur des données de types différents.

les utilisateurs, néanmoins il ne traite et ne corrige pas les erreurs détectées dans les fichiers PDB, il se contente d’afficher des remarques aux utilisateurs une fois l’erreur détectée.

En ce qui nous concerne, aucun de ces *parsers* existants ne répond à notre problématique à propos de la gestion des données manquantes et de la redondance des chaînes. Compte tenu des inconvénients rencontrés avec les *parsers* existants, le développement d’un nouveau *parser* nous a semblé donc indispensable, surtout pour améliorer la traitement des structures tridimensionnelles des macromolécules en utilisant des fonctionnalités et données récentes et permettre la correction de certaines erreurs des fichiers PDB, sans avoir recours à des composants ou des bibliothèques externes, tout en optimisant les performances du *parser*.

## 2.4 Le nouveau parser SAFlex-PDB

### 2.4.1 Contexte et fonctionnalités

Le *parser* SAFlex-PDB est à la base le module 1 implémenté pour la génération de l’alphabet structural, mais il a été amélioré en librairie séparée des modules statistiques qui contribuent à l’obtention de l’alphabet. De ce fait, il peut être intégré dans n’importe quel autre travail cherchant à *parser* un ou plusieurs fichiers PDB.

SAFlex-PDB est en premier lieu destiné aux biologistes mais peut être utilisé par des statisticiens et des informaticiens. Son objectif principal est d’analyser les fichiers au format PDB. Le présent travail prend en compte tous les types de macromolécules biologiques (ADN, ARN, protéine) mais ne traite que les chaînes protéiques.

De nombreuses fonctionnalités sont intégrées et proposées dans le *parser* lui permettant de :

- calculer les descripteurs des fragments structuraux,
- calculer les distances euclidiennes entre les  $C\alpha$  successifs,
- détecter des données manquantes dans les chaînes protéiques,
- détecter les trous dans les protéines<sup>8</sup>,
- étudier la variabilité des fragments structuraux<sup>9</sup>.

Cette section a exposé le contexte et les fonctionnalités du *parser*, la section suivante explique les différentes étapes de la modélisation<sup>10</sup> de la macromolécule et détaille l’implémentation de notre *parser* SAFlex-PDB.

### 2.4.2 Modélisation et implémentation

Dans le but d’implémenter le *parser* PDB, la modélisation informatique d’une structure 3D d’une macromolécule biologique revient tout d’abord à analyser le contenu du fichier au format PDB. Dans cette analyse la modélisation des différentes composantes de la macromolécule est cruciale afin de répondre aux besoins de mener à bien l’implémentation du *parser* et de faciliter ultérieurement la

---

8. se fait à partir des distances euclidiennes calculées entre les  $C\alpha$  successifs

9. l’étude de variabilité des fragments se fait en calculant les moyennes, la variance et l’écart type entre leurs descripteurs

10. La modélisation des données d’une manière informatique consiste à la description des données et besoins fonctionnelles décrits par les futurs utilisateurs du projet logiciel à réaliser.

maintenance du *parser*. Ainsi, nous considérons « ATOM » et « RESIDU » comme les objets élémentaires de notre programme. Ces objets forment ensemble l'objet « CHAIN ». En effet, cette chaîne est une séquence de résidus, or ces derniers sont à leur tour constitués d'une suite d'atomes. Nous définissons également l'objet « SECONDARY\_STRUCTURE » qui regroupe les différents objets : « HELIX », « SHEET » et « COIL » obtenus à partir du fichier PDB. Ensuite, nous considérons aussi l'objet « 3D\_STRUCTURE » qui représente le repliement 3D de la macromolécule biologique. Notre modélisation suggère d'avoir un nouveau type élémentaire : « FRAGMENT », qui représente notre fragment structural obtenu après une décomposition de la chaîne protéique en quatre C $\alpha$ . Et enfin l'objet « PROTEIN » qui regroupe tout ce qu'on a cité ci-dessus comme objets élémentaires. Il convient de noter qu'il faut pas confondre la protéine avec l'objet « PROTEIN », car cet objet modélise la macromolécule qui peut contenir une ou plusieurs chaînes de protéines, de l'ADN ou de l'ARN. À chaque « PROTEIN », nous associons un objet « 3D\_STRUCTURE », formé des coordonnées 3D des atomes constituant la macromolécule.

Le *parser* SAFlex-PDB proposé au cours de ce travail a été implémenté en langage C++, en utilisant l'environnement de développement Code::Blocks [124]. L'implémentation tire donc profit de l'architecture orientée objet. Ainsi, tous les objets cités dans le paragraphe précédent, sont implémentés à l'aide des classes, qui représentent un des concepts fondamentaux de la programmation orientée objet.

Le diagramme UML de classes présenté par la figure 2.4 montre l'architecture globale du *parser* SAFlex-PDB. Il permet de décrire les différentes classes ainsi que les relations établies entre elles. Ces relations sont représentées par un arc dans le diagramme de classe. Quand aux petits chiffres sur un arc entre la classe A et B, ils sont appelés cardinalités et représentent le nombre de B associés à A. Par exemple les cardinalités 1 et \* entre la classe « FRAGMENT » et la classe « CHAIN » exprime qu'il faut au moins 1 objet ou plusieurs de la classe « FRAGMENT » pour composer un objet de la classe « CHAIN ».



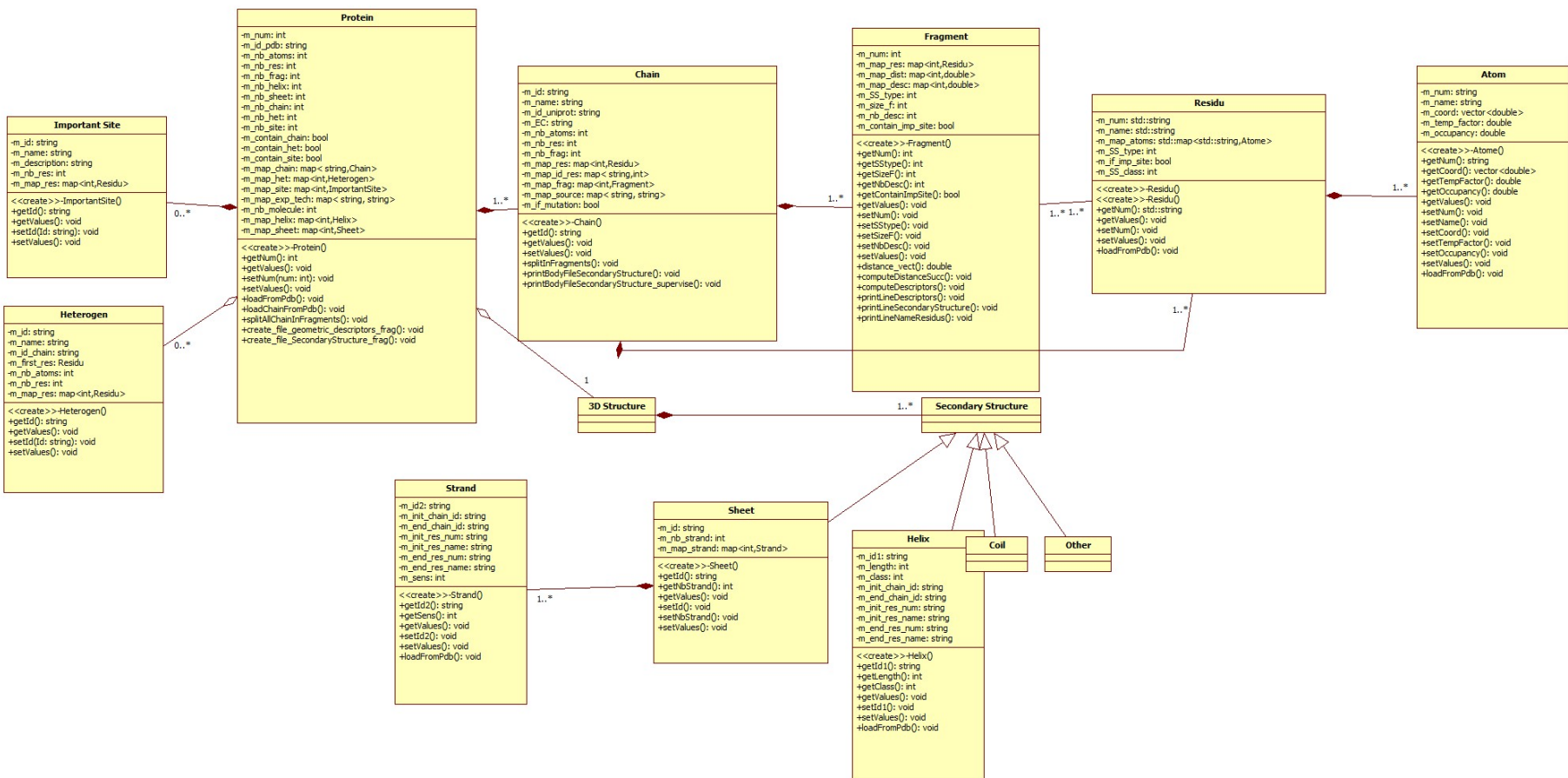


Figure 2.4 – Modélisation UML de la structure du parser PDB par un diagramme de classes de données.

Cette importante tâche d'implémentation a permis maintenant non seulement de récupérer les informations fournies sous forme de texte (alphanumérique) mais aussi de créer les liens entre ces données. Parmi les informations récupérées du fichier PDB nous citons :

- la liste des atomes,
- la liste des résidus,
- la liste des chaînes,
- la liste des protéines,
- la liste des sites importants,
- la liste des ligands,
- la liste des hélices,
- la liste des feuillets,

Le fait de *parser* consiste à récupérer ces listes en format texte (chaîne de caractères) et à les transformer en format « OBJET », puis à les traiter et à les analyser. Par suite le programme est implémenté de telle façon à afficher les résultats sur écran, à les sauvegarder automatiquement et à générer les fichiers en fonction des données en entrée. Les fichiers résultants sont :

1. le fichier des descripteurs géométriques des fragments calculés à partir des atomes  $C\alpha$ .
2. le fichier des structures secondaires associées à chaque résidu du fragment.
3. le fichier des sites importants des chaînes protéiques résolues dans le fichier PDB.
4. le fichier des noms des résidus contenus dans chaque fragment.
5. le fichier des distances successives entre les atomes  $C\alpha$ .
6. le fichier des séquences de résidus (structures primaires) des chaînes protéiques.

Le *parser* SAFlex-PDB fournit également pour chaque chaîne les informations suivantes :

- son identifiant, son nom, sa longueur (nombre de résidus), son n° d'accèsion.
- le nombre de fragments, la taille de celui-ci et le nombre de descripteurs.
- la présence ou non d'un site important, les résidus qui appartiennent à celui-ci, son type et sa méthode de détection.
- la présence ou non d'un ligand ou d'un partenaire (HETATM).
- les résidus manquants ou présents dans la résolution de la chaîne.
- la moyenne et l'écart type de chaque descripteur associées à chaque chaîne.
- les outliers associés à chaque descripteur.

L'implémentation du *parser* SAFlex-PDB a nécessité près de 9 000 lignes de code C++, pourtant il ne présente qu'une étape préparatoire mais cruciale pour la mise en place de l'alphabet structural pour la communauté scientifique.

Cette section a présenté les différentes étapes de modélisation des macromolécules et l'implémentation du *parser* mais pour plus d'efficacité, il est nécessaire d'optimiser celui-ci et de le paralléliser c'est ce qui fait l'objet de la section suivante.

### 2.4.3 Optimisation et parallélisation du parser PDB

#### 2.4.3.1 Profil de performance

Le *parser* SAFlex-PDB est une composante implémentée en bibliothèque (interface programmeur) et en outil en ligne de commande (interface utilisateur). L'objectif de ce module est d'analyser les informations contenues dans un fichier PDB, les compléter éventuellement, avant de les traduire en format exploitable par les autres modules (de génération de l'alphabet structural en l'occurrence). Nous avons voulu que notre implémentation soit la plus rapide possible pour pouvoir traiter un très grand nombre de fichiers PDB en un temps raisonnable. Nous allons exposer le caractère hautement parallèle de ce module. En effet, les fichiers PDB ne contiennent pas d'informations dépendantes qui rendent le traitement d'un fichier dépendant du résultat d'un autre. Le *parser* est alors un candidat naturel pour une parallélisation par threads. Nous avons respecté la méthodologie introduite dans la section C.1.1 de l'annexe C. À savoir, exploiter toutes les granularités fines du parallélisme avant toute tentative de mise-en-oeuvre d'un parallélisme supérieur.

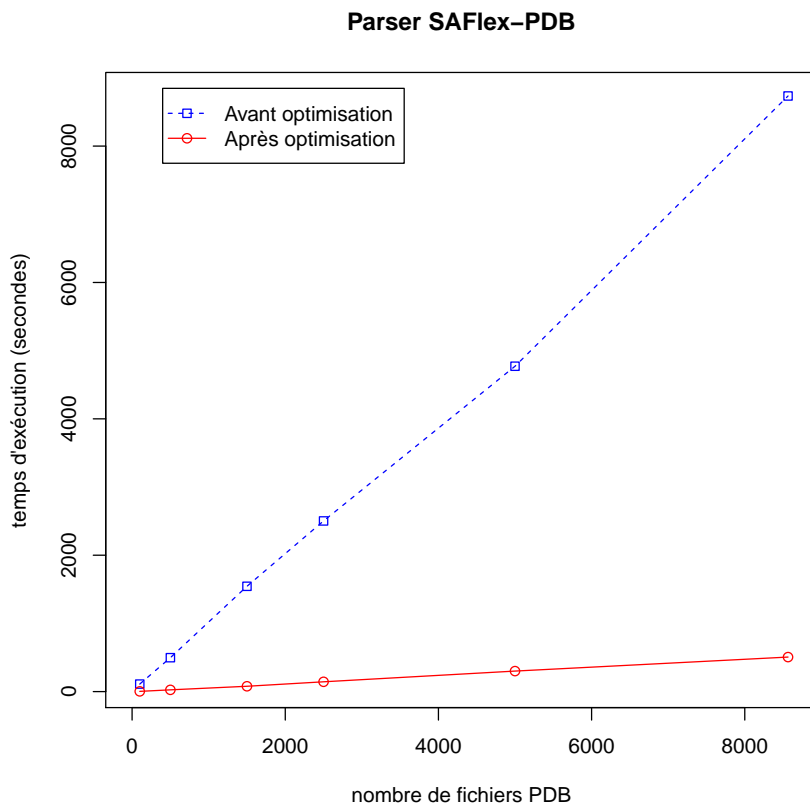


Figure 2.5 – Comparaison de temps d'exécution de SAFlex-PDB avant et après optimisation.

### 2.4.3.2 Exploitation du parallélisme fin (d'instructions et de données)

À la base le code du *parser* de fichiers PDB partage les mêmes bibliothèques utilisées pour le programme génération de l'alphabet structural. Le *parser* souffrait donc des mêmes lacunes : des copies excessives d'objets en mémoire. Nous avons donc appliqué les mêmes méthodes pour les résoudre. En l'occurrence, le transfert des ressources détenues par des objets en fin de vie grâce au mécanisme de déplacement introduit dans le standard C++11 (voir la section C.2.2 de l'annexe C). Nous avons également maximisé l'utilisation de mémoires caches en rassemblant des portions de codes qui utilisent les mêmes objets. Ou en séparant des portions de code qui utilisent trop de données qui ne tiennent pas dans les différents niveaux des mémoires caches [109].

La figure 2.5 montre que l'accélération globale que nous avons obtenue avec ces modifications est de l'ordre de 8x. Autrement dit, le code optimisé jusqu'ici est 8 fois plus rapide que le code original.

### 2.4.3.3 Parallélisation

Nous avons évoqué, dans la section précédente, le caractère hautement parallèle du *parser*. En effet, hormis les opérations d'écritures dans le fichier de sortie, il n'y a aucune dépendance entre le traitement de deux fichiers PDB. Ceci est reflété par le code de la boucle principale de l'utilitaire présenté par le listing 2.6 suivant :

```
void ProcessPDBFiles(
    vector<string> const& inputFileArray
    , ostream& outputFile)
{
    for(size_t i; i<inputFileArray.size(); ++i)
    {
        Protein protein;
        protein.LoadFromPDBFile(inputFileArray[i]);
        protein.WriteDescriptors(outputFile);
    }
}
```

**Listing 2.6** – Exemple simplifié de la boucle principale du code du *parser* SAFlexPDB avant parallélisation

Nous avons parallélisé ce code en utilisant la technologie OpenMP [33] (décrite dans la section C.2.4) de la façon la plus intuitive illustrée par le listing 2.7 suivant:

```
void ProcessPDBFiles(
    vector<string> const& inputFileArray
    , ostream& outputFile)
{
    #pragma omp parallel for schedule(dynamic,256) shared(outputFile,
        inputFileArray)
    for(size_t i; i<inputFileArray.size(); ++i)
    {
```

```
Protein protein;
protein.LoadFromPDBFile(inputFileArray[i]);

#pragma omp critical
{
    protein.WriteDescriptors(outputFile);
}
}
```

**Listing 2.7** – Exemple simplifié de la boucle principale du code du *parser* SAFlexPDB après parallélisation

Ici, le code C++ d'origine a été maintenu dans la version parallélisée. Seules les deux directives de compilation (`#pragma`) ont été rajoutées pour orienter la génération du code parallèle. La première directive demande au compilateur (et à la bibliothèque OpenMP) de paralléliser la boucle `for` entre les threads préalablement créés<sup>11</sup>. Chaque thread se voit attribuer progressivement un *morceau*<sup>12</sup> de 256 itérations uniques et aura la responsabilité d'exécuter les calculs associés. L'ordonnancement de ces morceaux sur les threads est déterminé d'une façon dynamique. Autrement dit, les morceaux de boucle non encore exécutés seront confiés au premier thread disponible.

La deuxième directive délimite une section critique pour protéger le fichier de sortie. Ceci garantira l'intégrité du fichier de sortie en s'assurant que seul un thread aura accès au fichier à un moment donné.

L'ordonnancement dynamique permet de faire du partage équitable de charge de calcul (*load-balancing*) entre les threads. Ceci évite que certains threads se voient attribuer une forte charge de travail par rapport à d'autres qui auront une charge plus légère. La taille des morceaux de boucle 256 (itérations) a été empiriquement déterminée pour garantir une charge de calcul suffisante pour chacun des threads (environ 200 millisecondes). Une charge de calcul plus faible peut compromettre la rentabilité de la parallélisation. Le risque est de voir le mécanisme d'ordonnancement devenir plus coûteux que le calcul lui même.

#### 2.4.3.4 Résultat de la parallélisation

La figure 2.9, ci-dessous, montre que nous avons obtenu une accélération de 3,2 avec 4 threads pour traiter plus de 8 500 fichiers PDB. Nous avons réussi à diviser par 3 le temps nécessaire pour traiter l'ensemble des fichiers. L'accélération théorique maximale espérée est de 4 (avec une parallélisation avec 4 threads). Il aurait été possible de *tuner* davantage la parallélisation pour augmenter son efficacité. Mais nous avons préféré en rester là car le résultat obtenu est très satisfaisant.

Cette section a développé les différentes phases de l'optimisation et de la parallélisation du *parser* pour bien mettre en évidence les gains obtenus, il était nécessaire de faire des tests comparatifs avec des *parsers* équivalents, une synthèse de ceux-ci est l'objet de la section suivante.

11. Bien que légers, la création des threads a un coût non négligeable. Pour y remédier, OpenMP crée l'ensemble des threads au moment du chargement des binaires. Des portions de calculs leurs seront affectées par la suite.

12. OpenMP découpe les boucles en morceaux (loop chunks). Chaque morceau est un bloc d'une ou de plusieurs itérations de la boucle originale.

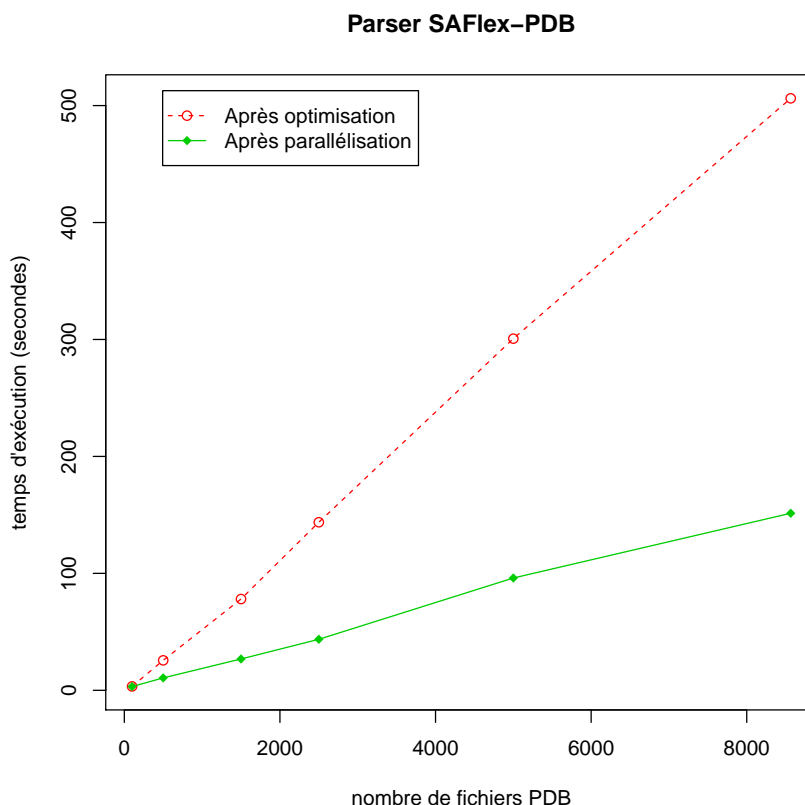


Figure 2.6 – Comparaison de temps d'exécution entre SAFlexPDB optimisé et parallélisé.

## 2.5 Synthèse des tests comparatifs parser SAFlex-PDB, Biopython et Biojava

En vue d'évaluer les performances du nouveau *parser* SAFlex-PDB, des tests d'évaluation comparatifs ont été entrepris, ils ont porté sur la qualité du traitement et analyse des données et sur les performances (temps d'exécution) du programme. Ainsi, notre *parser* SAFlex-PDB a été comparé aux *parsers* Biopython et BioJava. Ces deux derniers ont été présentés et décrits au début de ce chapitre (les sections 2.3.2 et 2.3.3).

Dans cette section, nous commençons par une brève présentation du jeu de données utilisé pour les tests puis nous expliquons ce qui a été mis en place pour ces tests. Enfin, nous récapitulons les principaux résultats obtenus.

### 2.5.1 Jeu de données

Nous avons utilisé au cours de ces tests le jeu de données représentatif appelé « PDBSelect » [78, 86] qui comporte actuellement 8 565 fichiers PDB et plus de 21 800 chaînes protéiques. Celles-ci ont été choisies avec moins de 30% d'identité de séquence. Ce jeu est surtout utilisé quand on est confronté à des analyses statistiques car il représente une simplification représentative du contenu de la banque

PDB [163]. Afin de mener à bien ces tests comparatifs entre les *parsers*, nous avons fait des statistiques sur le jeu choisi, nous les présentons dans la table 2.1 suivante :

	Contenant RES Manq	contenant des Atom C $\alpha$ Manq	prot Contenant RES Manq	contenant des Trous
Nb PDB dans le jeu	6393	6424	6189	7705
Pourcentage	74,64%	75,00%	72,25%	89,96%
Exemples PDB	1A26, 1A1W, 1A26,...	1B62, 1CY5, 1GU4,...	5F2K, 8A3H, 6RLX,...	3FL2, 3WWL, 4QHQ,...

**Table 2.1** – Statistiques sur le jeu de données PDBSelect

### 2.5.2 Préparation des tests

Le *parser* SAFlex-PDB étant implémenté en langage C++. Il était nécessaire de coder deux autres programmes pour les *parsers* BioJava et Biopython. L'implémentation des programmes porte sur l'analyse des fichiers PDB du jeu de données choisi pour nos tests.

Pour cet implémentation, nous avons choisi comme environnement de développement intégré Eclipse [71], en effet celui-ci est construit principalement pour le développement Java mais il a l'avantage d'être extensible pour intégrer des *plugins*<sup>13</sup> d'autres langages tel que Python, C++, PHP, ..., etc. Afin d'utiliser le *parser* BioJava, des notions nécessaires sur le langage Java nous ont permis ainsi de développer le programme d'analyse des fichiers PDB utilisant celui-ci. De même nous avons eu à implémenter un autre programme en langage python pour utiliser le *parser* BioPython.

Il est à noter qu'il est nécessaire d'apporter des configurations au niveau d'Eclipse pour intégrer les *plugins* de BioPython et BioJava. Ainsi, pour BioPython, une installation du *plugin* Python et du *package* BioPython est essentielle, car le *plugin* permet de développer des programmes Python sous Eclipse. Nous avons eu besoin pour le développement du programme de l'importation des modules suivants :

- Bio.PDB.PDBParser,
- Bio.PDB.Polypeptide,
- PDBParser,
- is\_aa,
- Bio,
- NumPy,
- math,
- time.

Pour ce qui concerne BioJava, il suffit d'importer les librairies Biojava suivantes :

- biojava-alignment-4.2.0.jar,
- biojava-core-4.2.0.jar,
- biojava-structure-4.2.0.jar,
- logback-classic-1.1.3.jar,
- logback-core-1.1.3.jar,
- slf4j-api-1.7.21.jar,

---

13. un *plugin* est un module d'extension ou un paquet qui complète un logiciel afin de lui apporter de nouvelles fonctionnalités.

– vecmath-1.5.1.jar.

Une fois les configurations apportées et les programmes utilisant BioJava et BioPython développés, nous avons réalisé nos tests, nous présentons dans ce qui suit les résultats de ces tests.

### 2.5.3 Présentation des résultats

#### 2.5.3.1 Au niveau qualité de l'analyse et traitement des données

Les trois *parsers* SAFlex-PDB, BioPython et BioJava traitent tous les fichiers PDB du jeu de données, à savoir 8 565 PDB, en signalant des messages « warning » en cas d'erreur détectée au niveau des fichiers traités.

##### 2.5.3.1.1 Détection des erreurs dans les fichiers PDB par les *parsers*

BioPython signale des warnings avec précision en identifiant la chaîne et la ligne concernées mais n'indique pas le nom du fichier PDB, qui est très important à connaître. Ainsi, Biopython affiche 21 771 « warnings » de type « PDB Construction warning », ce qui fait 8 539 / 8 565 PDB, il est à noter que Biopython associe le même message, présenté par le listing 2.9 pour plusieurs types d'erreurs comme suit :

```
/usr/lib/python2.7/dist-packages/Bio/PDB/StructureBuilder.py:85:  
PDBConstructionWarning: WARNING: Chain A is discontinuous at line  
3459.
```

**Listing 2.8** – Exemple du warning « PDBConstructionWarning » affiché par Biopython

Il signale aussi 59 « warnings » de type erreur d'assignement des noms de résidus comme présenté dans le listing suivant :

```
/usr/lib/python2.7/dist-packages/Bio/PDB/Atom.py:99:  
PDBConstructionWarning: Could not assign element 'UNK' for Atom (  
name=UNK) with given element 'X' warnings.warn(msg,  
PDBConstructionWarning)
```

**Listing 2.9** – Exemple d'un message affichant une erreur au niveau du nom de résidu par Biopython

Pour BioJava, il affiche tellement de détail sur la structure récupérée au moment du chargement fichier PDB, que la console de l'environnement Eclipse ne peut sauvegarder tous ceux-ci et qu'un manque de mémoire (*Overflow memory*) est constaté. J'ai essayé donc de forcer la sauvegarde de ces détails pour avoir une idée sur le nombre et le type de message mais le programme s'arrêtait au bout d'un moment avec une erreur de *overflow* de mémoire. Au moment de la rédaction du présent manuscrit de thèse, toutes les pistes explorées pour récupérer ces détails affichés n'ont pas abouti.

Concernant le *parser* SAFlex-PDB développé durant cette thèse, il signale des « warnings » en cas d'erreurs dans le format du fichier en précisant le type de celles-ci avec la ligne exacte de l'erreur et le nom du fichier PDB. Quelques exemples de fichiers PDB qui contiennent des données manquantes dans le jeu représentatif « PDBSelect » sont cités dans le tableau 2.1. Nous avons déjà détaillé ce



sujet au cours de la section 2.2 de ce chapitre. Dans cette section, nous proposons un nouveau *parser* performant et robuste aux manques de données au niveau des fichiers PDB. Celui-ci détecte et signale un certain nombre d'erreurs et de problèmes au niveau des fichiers. Quelques exemples des fichiers PDB qui ont des erreurs détectées par notre *parser* ont été cités dans le tableau 2.1.

Aussi, le *parser* SAFlex-PDB signale cinq « warnings » graves sur le format PDB en précisant la ligne et le nom du fichier. Ces derniers :

- soit ne contenaient pas des records « DBREF » (énonce les différentes chaînes dans le fichier PDB : cas de 1q2j),
- soit il n'y a pas de signalement de fin de la chaîne (pas de record « TER » : fichier 4oe8 absence de mot clé TER entre la chaîne B et C),
- soit non respect du format du record « COMPND » (énonce les différentes molécules contenues dans les fichiers 4go6, 5cj1, 3unb).

Parmi ces erreurs abordées ci-dessus par notre parser SAFlex-PDB, le listing 2.10 montre l'affichage d'une erreur lors du traitement du fichier n° 8 195 concernant le PDB 4go6. L'erreur est survenue à la ligne 8 du fichier PDB, celle-ci ne respecte pas le format exigé pour le record « COMPND ».

```
8195.File being processed is : 4go6.pdb
*****
4go6.pdb :line #8: Error 'basic_string' occurred while processing the
following line 'COMPND 6 HCF N-TERMINAL CHAIN 2, HCF N-TERMINAL
CHAIN 3, HCF N-TERMINAL CHAIN '. Error in format of COMPND record.
```

**Listing 2.10** – Exemple d'un message affichant une erreur au niveau du format du record « COMPND » par SAFlex-PDB

### 2.5.3.1.2 Détection du manque de données et des trous par le parser

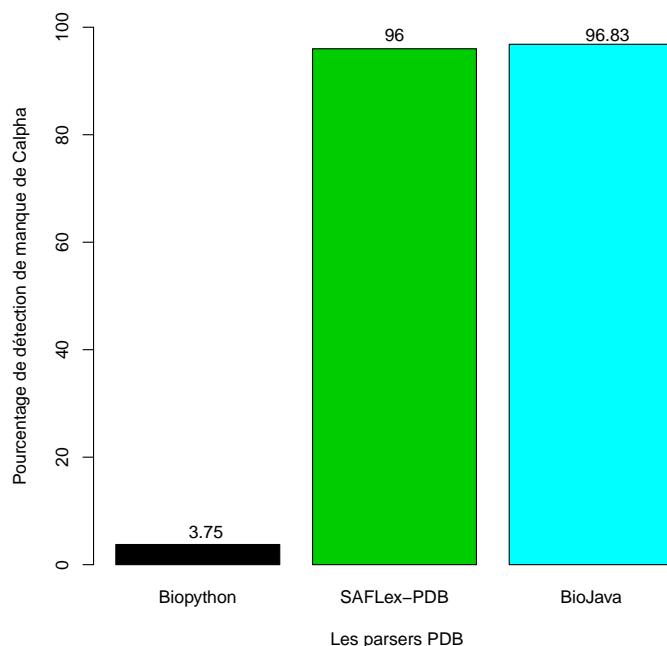
Le *parser* SAFlex-PDB détecte aussi bien que BioJava le nombre des résidus et les C $\alpha$  manquants (voir le tableau 2.2). En effet, SAFlex-PDB et BioJava signalent les manques de résidus et de C $\alpha$ , ils insèrent également les résidus manquants aux chaînes appropriées en signalant le nom du fichier PDB sur lequel le manque est constaté. Ceci n'est pas le cas de biopython telle qu'illustré par la

	Nb.PDB.TRAITES	Nb.PDB.MANQUE.C $\alpha$	POURC detec manq C $\alpha$	Nb PDB.contenant.TROU	Détecte.TROUS
SAFlex-PDB	8565	5942	96.00%	7705 (89,96%)	OUI
BioJava	8565	5993	96,83%	/	NON
BioPython	8565	232	3,75%	/	NON

**Table 2.2** – Détection des erreurs et manques de données par les trois parsers

figure 2.7. Ceci influence le calcul des descripteurs (décrits dans la section 3.4 du chapitre 3) surtout pour les chaînes qui ont un manque de résidus au milieu de la structure tels que les fichiers PDB : 4xmr (plusieurs résidus manquants au milieu de la chaîne), 2woz (plusieurs résidus manquants à 2 endroits différents au milieu de la chaîne), 3h8z (plusieurs résidus manquants à trois endroits différents au milieu de la chaîne). De ce fait Biopython n'est pas du tout fiable dans ces cas.

De plus, pour chaque fichier PDB le *parser* SAFlex-PDB vérifie en premier lieu la cohérence des numéros d'atomes attribués, puis la cohérence des numéros de résidus attribués c'est à dire vérifier s'il



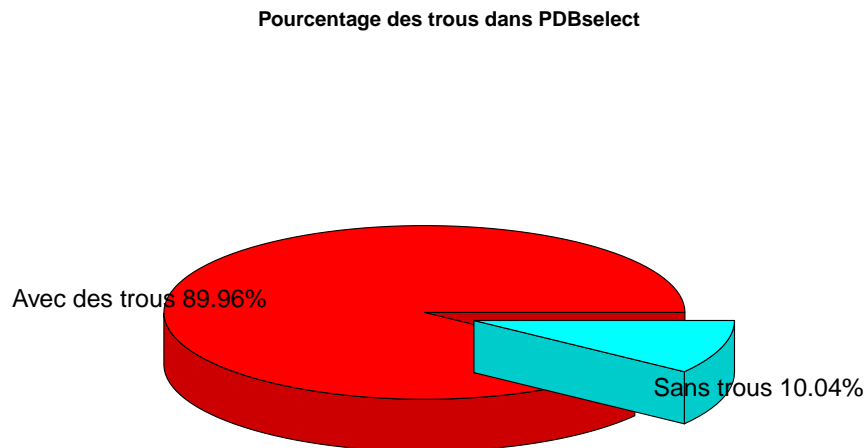
**Figure 2.7 – Pourcentage de détection du manque des C $\alpha$  par les *parsers*.**

y a des données manquantes. Si tous les numéros sont cohérents, il calcule ainsi la distance Euclidienne entre les C $\alpha$  des résidus consécutifs. Ceci lui permettra de détecter les trous (les manques) non signalés par les auteurs des fichiers PDB. Ainsi nous avons calculé le nombre des fichiers ayant des trous au sein du jeu de données choisi, les résultats sont donc présentés dans le tableau 2.2. La figure 2.8 montre que presque 90% des fichiers PDB contiennent des chaînes protéiques ayant des trous, c-à-d ayant un manque de résidus non déclarés dans le fichier PDB.

Pour la suite de notre travail de thèse, il est très important de savoir gérer les trous et les données manquantes au niveau des fichiers PDB afin de répondre aux objectifs de la thèse qui envisage de tenir compte de l'incertitude de ces données.

### 2.5.3.2 Au niveau performances

La figure 2.9 montre une comparaison du temps d'exécution des trois *parsers* sur le même jeu de données « PDBSelect ». Ces trois *parsers* ont été exécutés sur une MACHINE Intel (Xeon) CPU E5-1603 2,80 GHZ sur Ubuntu. Les résultats montrent que le *parser* SAFlex-PDB est sept (07) fois plus rapide que le *parser* Biopython et une fois et demi (1,5) plus rapide que BioJava et ce sans parallélisation. Après parallélisation avec seulement quatre (04) threads, le *parser* SAFlex-PDB est vingt trois fois (23) plus rapide que Biopython et cinq fois (05) plus rapide que BioJava. Pour plus de détails, les tableaux de résultats de cette partie des tests sont présentés à l'annexe E. Cependant le tableau 2.3 ci-dessous montre les temps de traitement des données de « PDBSelect » par chaque *parser* (SAFlex-PDB, BioPython et BioJava)



**Figure 2.8** – Pourcentage de détection des trous à l’intérieur des protéines dans le jeu de données représentatif « PDBSelect » contenant 8565 fichiers PDB.

	Les.parsers			
Temps exécution (secondes)	Biopython	Biojava	SAFlex-PDB optimisé	SAFlex-PDB parallélisé
	3521.342	763.408	506.226	151.404

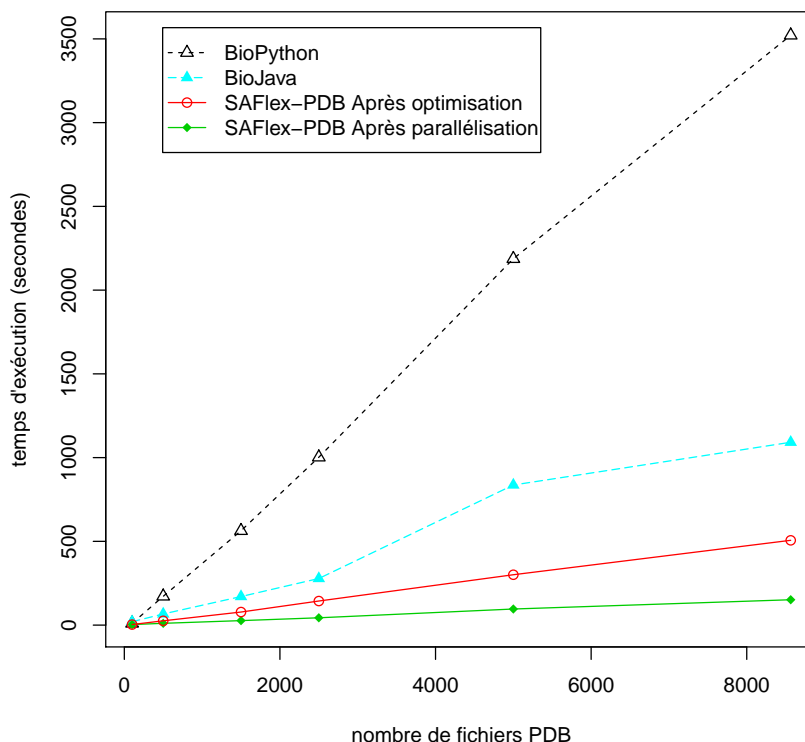
**Table 2.3** – Temps du traitement des données de PDBSelect par les trois parsers

Les comparaisons que nous présentons semblent intéressantes, démontrant bien que le *parser* SAFlex-PDB a un intérêt aussi bien sur le plan qualitatif (détection de diverses erreurs) que quantitatif (rapidité et parallélisation).

Cette section a exposé les résultats des tests comparatifs de notre *parser* SAFlex-PDB avec les *parsers* Biopython et Biojava, les principaux aspects qualitatifs sont mis en évidence dans la section suivante.

## 2.6 Qualité du parser SAFlex-PDB

L’implémentation de SAFlex-PDB a respecté les différents aspects qui définissent la qualité d’un programme informatique (maintenabilité, la fiabilité, portabilité et performance). Afin de vérifier la *fiabilité* du *parser* que nous proposons, des analyses statiques et dynamiques en utilisant Cppcheck [128] et valgrind [125] ont été réalisées. Ces analyses attestent de la robustesse de SAFlex-PDB, en effet aucune fuite de mémoire, de fuites de ressources et l’utilisation des fonctions est valide selon la



**Figure 2.9 – Comparaison de temps d'exécution entre SAFlexPDB, BioPython et Biojava.**

STL (*Standard Template Library*) [151]<sup>14</sup>. En d'autres termes, les allocations de ressources ne se font que lorsque c'est utile (par exemple fichier PDB), puis libérées après utilisation.

Concernant la *qualité* du *parser*, elle est garantie par un affichage à l'écran de messages d'erreurs souvent détectées lors des traitements des fichiers PDB, ou lors d'une mauvaise utilisation de l'outil par l'utilisateur. Ainsi, ces messages sont destinés à aider l'utilisateur à comprendre et à rechercher l'erreur dans le fichier PDB, ou à le guider à la bonne utilisation du *parser*. Nous avons expliqué ces messages en détail dans la section 2.5.3.1 de ce chapitre. Celle-ci a été consacrée à la détection des erreurs lors de l'analyse des fichiers PDB.

Pour ce qui est de la *portabilité* du *parser*, elle est assurée par un fonctionnement sur tous les systèmes Linux (POSIX) et Windows. De plus, les résultats du traitement et analyse de SAFlex-PDB sont affichés sur écran et sauvegardés automatiquement dans des fichiers. Quant à la maintenabilité du programme, elle est garantie par un code source souple se basant sur une architecture orientée objet. Celle-ci permet d'éviter d'éventuelles erreurs faciles et rapides lors des recherches et des corrections des bugs.

Enfin les *performances* du *parser* sont attestées par une meilleure gestion des appels aux fonctions systèmes (minimisation de ces appels par exemple : fonctions d'ouverture et de fermeture des fichiers PDB) et un accès réduit aux moyens de stockage de masse (disque dur) en utilisant par exemple les

14. La STL est une bibliothèque C++ normalisée.

tableaux associatifs tels que les classes `std::MAP` et `std::multimap`. Ceci améliore les performances de SAFlex-PDB et rend son temps d'exécution moyen de l'ordre d'une seconde par fichier PDB.

Cependant les performances du programme ont été considérablement améliorées après son optimisation et sa parallélisation. La méthodologie utilisée pour cette optimisation était détaillée dans la section 2.4.3 de ce chapitre.

De plus, les tests comparatifs avec les *parsers* très connus dans le domaine bioinformatique (Bio-Python et BioJava) démontrent bien que l'outil à un intérêt sur le plan qualitatif et quantitatif.

## 2.7 Conclusion

Au cours de ce chapitre nous avons présenté brièvement la banque PDB, les données qu'elle contient en expliquant leurs structures et leurs méthodes de résolution. Nous avons expliqué également les différents *parsers* existants, leurs avantages et inconvénients. Cette étude sur les *parsers* nous a convaincu de la nécessité de développer un nouveau *parser* PDB plus performant. Ce chapitre a présenté donc essentiellement les étapes de développement de ce nouveau *parser* « SAFlex-PDB » en détaillant son implémentation étape par étape puis en mettant en évidence sa nécessaire optimisation, la méthode de parallélisation retenue et son intérêt. Les performances obtenues après parallélisation sont mises en évidence. Enfin des tests comparatifs avec les *parsers* Biopython et Biojava sont exposés et les résultats obtenus sont commentés. Ceux ci permettent de constater que le nouveau *parser* SAFlex-PDB proposé a des performances meilleures tant au niveau qualitatif que vitesse ce qui était recherché car celui ci représente un des outils principaux du site web proposé qui fait l'objet du chapitre 4 et constitue également le résultat concret principal de cette thèse.

# Les avancées méthodologiques

## Résumé

Ce chapitre présente une nouvelle méthodologie de développement du nouvel alphabet structural SAFlex offrant plusieurs innovations. Cette méthodologie a permis la conception d'un nouveau modèle d'encodage rigoureux et robuste. Cet encodage permet en outre de prendre en compte l'incertitude des données en proposant trois options d'encodages : le Maximum *a posteriori* (MAP), la distribution marginale *a posteriori* (POST) et le nombre effectif de lettres à chaque position donnée (NEFF). Il permet également de gérer et de traiter les données manquantes dans les fichiers PDB (plus de 75% des fichiers PDB contiennent des données manquantes). Enfin, le nouveau alphabet SAFlex fournit un encodage consensus à partir de différentes réplifications (chaînes multiples, monomères et homomères) d'une même protéine. Il permet ainsi la détection de la variabilité structurale entre celles-ci. Ce chapitre décrit ensuite le nouvel alphabet structural SAFlex et détaille ses applications sur des protéines. Les avancées méthodologiques ainsi que l'obtention de l'alphabet SAFlex constituent la contribution principale de ce travail de thèse.

Dans ce chapitre, la section 3.1 présentera l'alphabet existant HMM-SA27, puis la section 3.2 détaille la démarche du développement d'un AS. Les sections 3.3 et 3.4 présentent la description et la caractérisation des structures tridimensionnelles. La section 3.5 détaille le nouveau modèle de développement de l'alphabet structural employé dans le cadre de cette thèse. Cette section explique les améliorations apportées par le nouveau modèle. Puis, le chapitre reprend dans les sections 3.6 et 3.7 les différentes phases d'implémentation et d'optimisation de cet alphabet. Ensuite, la section 3.8 expose le nouvel alphabet structural SAFlex, ses améliorations méthodologiques et à titre illustratif quelques exemples d'applications sur des protéines. Le chapitre présente également dans la section 3.10 une comparaison entre SAFlex et HMM-SA27. Enfin la section 3.11 conclut ce chapitre.

## 3.1 L'alphabet structural existant HMM-SA27

L'alphabet structural HMM-SA27 développé par [28] se base sur l'emploi des chaînes de Markov cachées, car les auteurs proposent que les fragments structuraux sont régis par un processus Markovien

afin de considérer l'enchaînement entre les fragments structuraux dans le processus d'apprentissage de ceux-ci. De ce fait les fragments représentatifs sont appris en tenant compte de leurs géométries et de leurs transitions. L'intérêt de HMM-SA27 est de décrire finement les structures tridimensionnelles des protéines afin de rechercher, simplifier et analyser les similarités structurales des motifs récurrents.

Dans cette section j'explique comment l'alphabet structural HMM-SA27 a été obtenu, ensuite je présente la notion de l'encodage structural.

### 3.1.1 Sélection des modèles

Camproux et ses collaborateurs ont utilisé le critère BIC (*Bayesian Information Criterion* [160]) pour déterminer le nombre optimal des lettres structurales du modèle HMM-SA27, qui est fonction du logarithme de vraisemblance, du nombre de paramètres et de la taille de l'alphabet [28]. Ainsi plusieurs alphabets structuraux de taille différente  $m$  ont été appris, par un modèle Markovien, et comparés, sur les deux jeux de données cités dans la section 3.3.2 à la page 47.

$$\text{BIC} = \mathcal{L}_{\max} - 0.5 \times m \times \log(n)$$

où :  $\mathcal{L}_{\max}$  est le log de vraisemblance maximisée,  $m$  est le nombre de paramètres du modèle et  $n$  le nombre d'observations.

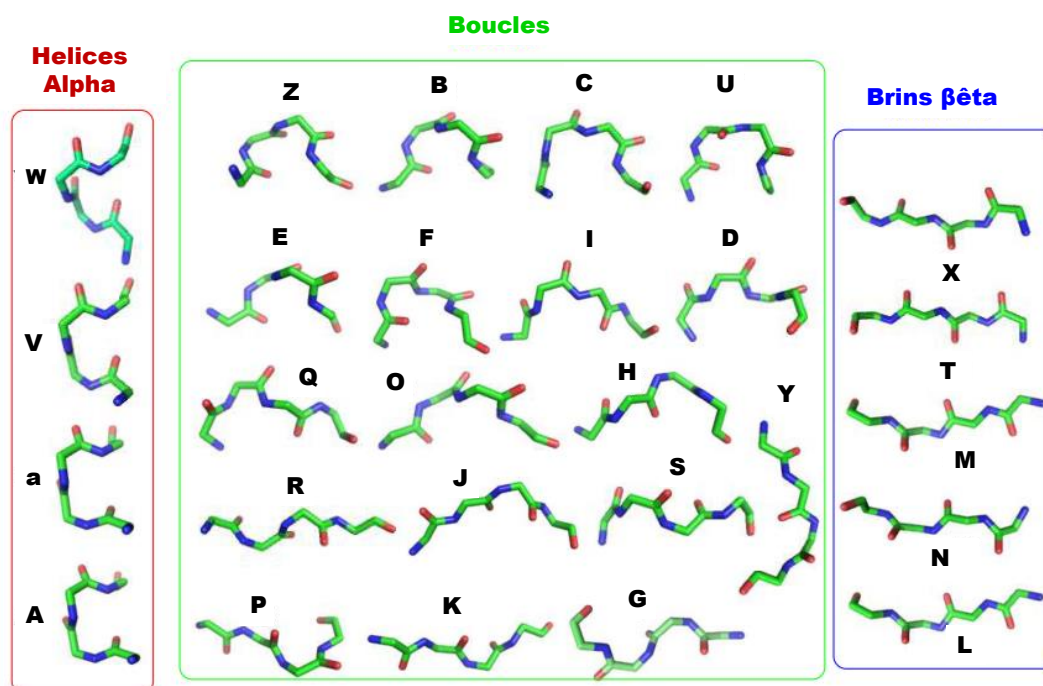
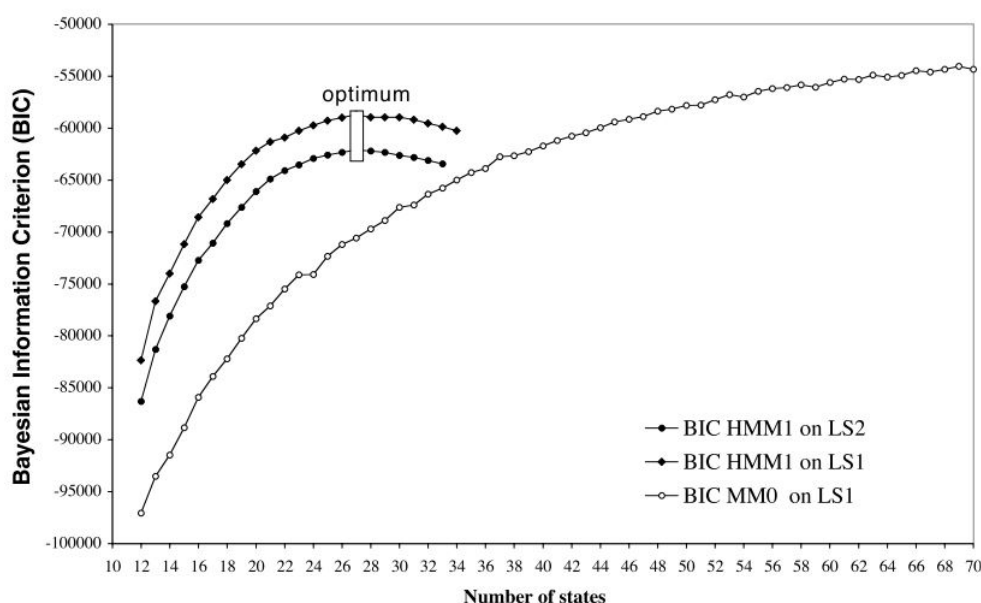


Figure 3.1 – Géométrie des 27 lettres structurales de l'alphabet structural HMM-SA27 (figure tirée de [142]).

En outre l'alphabet structural optimum obtenu pour les deux apprentissages (par HMM d'ordre 1) sur les deux jeux de données indépendants est celui de 27 lettres structurales nommées  $a, A, B, \dots, Y, Z$ . Elles sont illustrées par la figure 3.1 et classées en trois groupes, selon leurs géométries qui se distinguent d'une lettre à l'autre en suivant différentes règles logiques. Les trois lettres  $A, a, V, W$  apparaissent uniquement dans les hélices  $\alpha$  (plus de 92% des fragments protéiques adoptent la conformation d'une hélice  $\alpha$ ), tandis que les cinq lettres  $L, N, M, T, X$  sont associées aux structures étendues (entre 47% et 78% des fragments sont associés à des brins  $\beta$ ). Les 18 qui restent sont assignées aux structures boucles. L'alphabet structural HMM-SA27 identifié est représenté non seulement par l'ensemble des 27 lettres structurales mais aussi par leur matrice de transition, celle-ci indique les probabilités qu'une lettre soit suivie par une autre.

Par ailleurs, un apprentissage en utilisant les mélanges Gaussiens (HMM d'ordre 0) a été aussi effectué sur les deux jeux de données, celui-ci n'admet pas un optimum (un nombre supérieur à 40 classes atteint et toujours pas d'optimum) comme illustré par la figure 3.2. Ceci sous-entend que la prise en compte de la dépendance spatiale a une considération majeure dans l'apprentissage des structures tridimensionnelles protéiques.



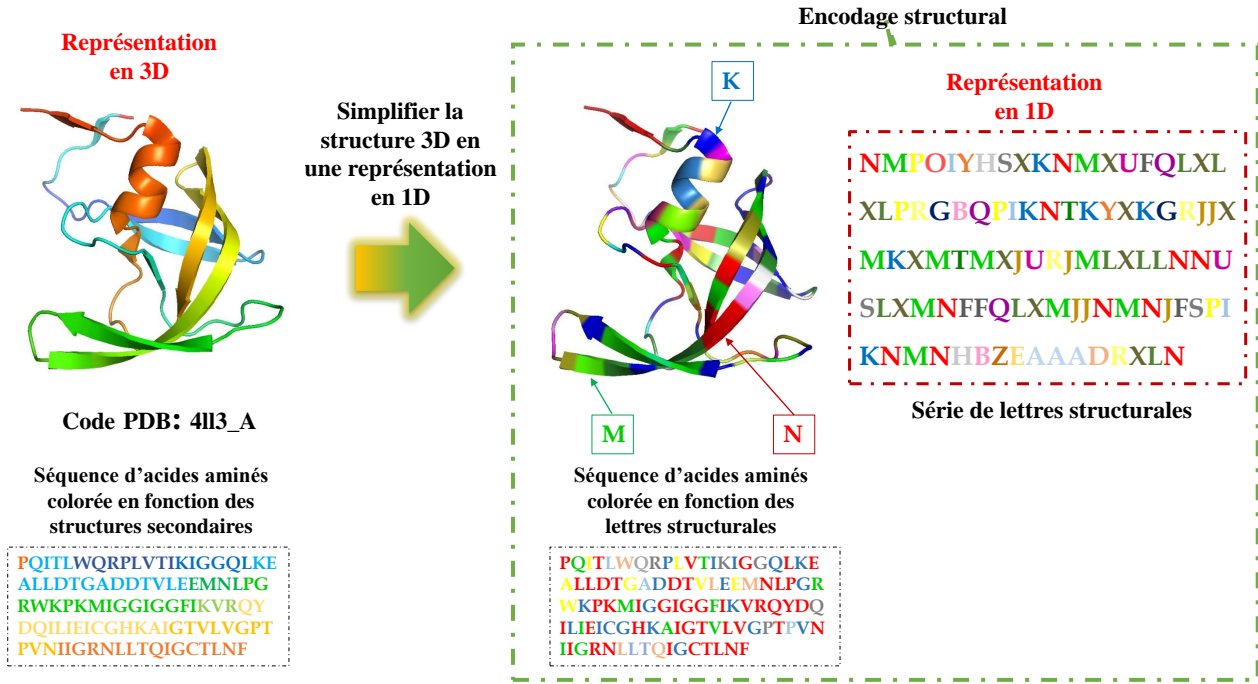
**Figure 3.2 – Identification de l'alphabet structural optimum en fonction du nombre d'états (lettres structurales) en utilisant le critère BIC.** Identification réalisée sur les deux jeux de données LS1 et LS2 par apprentissage (HMM1) et (HMM0) (figure tirée de [28]).

### 3.1.2 Encodage des protéines

Le premier objectif principal derrière le développement de l'alphabet structural est l'obtention de la séquence, non observée, des lettres structurales concernant les chaînes polypeptidiques, à partir des vecteurs de descripteurs des fragments protéiques. Les lettres structurales de cette séquence sont considérées comme des états cachés, alors que les descripteurs représentent les états observés de la chaîne de Markov cachée.



Le second objectif est de fournir une classification des fragments structuraux successifs en  $m$  lettres structurales. Ainsi, l'encodage consiste à trouver une séquence d'états optimaux pour un ensemble de conformations 3D donné, parmi toutes les séquences d'états possibles dans  $\mathcal{S}^n = \{1, 2, \dots, m\}^n$  en sélectionnant un modèle, tout en fixant le nombre de lettres structurales  $m$  et en estimant ses paramètres par un algorithme basé sur un processus Markovien.



**Figure 3.3 – Encodage structurale de la chaîne 4ll3A**

Simplification de la structure 3D de la chaîne 4ll3A (Code PDB) en une série de lettres structurales par HMM-SA27. À gauche est représentée la structure 3D de la chaîne 4ll3A. À droite la structure 3D colorée en fonction des lettres structurales et la séquence de lettres structurales qui simplifie cette chaîne en 1D.

En vue d'obtenir l'encodage structural, Camproux et ses collaborateurs [28] ont fait appel à la notion du maximum *a posteriori* (MAP) à l'aide de l'algorithme Viterbi [141] et l'algorithme *Forward-Backward*. Ce dernier sera expliqué en détail dans la section B.3 de l'annexe B à la page 109 :

$$\hat{S}^{\text{MAP}} = \arg \max_s \mathcal{L}(\hat{\theta} | X = x, S = s)$$

Cette démarche permet de décrire chaque structure comme une séquence de Lettres Structurales (LS). La figure 3.3 illustre l'encodage structural permettant la simplification de la structure 3D de la protéine en une séquence de lettres structurales issues de l'alphabet structural HMM-SA27 en utilisant les chaînes de Markov cachées. Chaque LS est décrite par une forme géométrique, représentée dans la figure par la même couleur que celle-ci.

## 3.2 Démarche de développement d'un AS

La démarche méthodologique du développement de l'alphabet structural est décrite sur la figure 3.4. Celle-ci est pluridisciplinaire car elle exige des efforts considérables de la part de différents groupes d'acteurs : les mathématiciens, les biologistes et les informaticiens du MTi (Molécules Thérapeutique In silico) et du LPMA (Laboratoire des Probabilités et Modèles Aléatoires, UPMC). En effet chacun de ces groupes a joué un rôle important pour atteindre nos objectifs.

Le développement a été découpé en neuf phases réparties sur trois ans et demi :

**Phase 1:** Phase de formation en probabilités et statistiques.

**Phase 2:** Étude et choix des jeux de données PDB.

**Phase 3:** Modélisation mathématique des données.

**Phase 4:** Implémentation en C++ du modèle.

**Phase 5:** Constitution et validation des nouveaux jeux de données.

**Phase 6:** Optimisation du code de génération d'AS.

**Phase 7:** Parallélisation du code.

**Phase 8:** Sélection du modèle optimum associé à l'alphabet structural.

**Phase 9:** Utilisation de l'alphabet structural et mise en place du site web.

Afin de faciliter la lecture de ce manuscrit, certaines de ces étapes seront développées au cours de ce chapitre, d'autres seront abordés dans le chapitre 5. Il est à noter que le début de ce travail de thèse était vraiment complexe vue mon profil d'informaticienne et c'est pourquoi nous avons procédé comme première phase de ce processus à ma remise à niveau en probabilités et statistiques. Celle-ci était nécessaire afin de bien appréhender la démarche statistique du développement d'un nouvel alphabet structural.

Cette première phase a été réalisée avec l'aide des mathématiciens et a apporté les notions essentielles nécessaires à la troisième phase sur la modélisation des structures tridimensionnelles. Ces notions statistiques portaient surtout sur la loi normale gaussienne (voir la section A.1.2), le maximum de vraisemblance (voir la section A.1) et les modes d'apprentissage automatiques (supervisée, semi-supervisée et non supervisée). En ce qui concerne la troisième phase, elle a consisté à modéliser les structures des chaînes protéiques à l'aide des chaînes de Markov cachées. Ce qui permet d'observer et d'étudier les structures tridimensionnelles dans le but de mieux les caractériser.

Quant aux biologistes, ils interviennent à la deuxième et cinquième phase. Leur travail a consisté à :

- choisir les données (les fichiers des structures tridimensionnelles);
- vérifier ces données, dans un troisième temps;
- réfléchir aux informations pertinentes existantes dans les structures 3D contenues dans les fichiers PDB;
- étudier leurs représentativités;
- constituer de nouveaux jeux de données et valider ceux-ci;
- valider biologiquement le modèle.

Pour ce qui est de la partie informatique, elle a été traitée au cours des phases 4, 6, 7, 8 et 9, celle-ci avait pour objet principal l'implémentation de l'alphabet structural. Ce travail consiste à :

- recueillir les informations,
- modéliser informatiquement les différents objets intervenant dans cette implémentation (les structures 3D, protéines, résidus, atomes, les structures secondaires, ..., etc.);
- implémenter un premier module ce qui permet de *parser* un fichier PDB, de lire son contenu et de récupérer les informations nécessaires à ce travail;
- implémenter un second module ce qui aboutit au calcul des probabilités d'évidences correspondantes à nos données;
- implémenter un troisième module de calcul des Log *Forward* et log *Backward* des données;
- enfin implémenter un dernier module entraînant l'initialisation des paramètres et leur estimation à chaque étape.

Tout ce travail a fait l'objet d'un poster à la conférence ISMB/ECCB [2]. Quant aux phases 6 et 7, elles consistaient à l'optimisation de mes modules, ainsi que leurs parallélisations. Ces phases sont réalisées en séquence mais parallèlement à la phase 5 (constitution de nouveaux jeux de données). Finalement une sélection des modèles aboutissant à la génération de plusieurs AS de différentes tailles ce qui permettra de garder le meilleur qui sera optimisé (critère BIC).

Seules deux phases sont des étapes encore en cours : l'étape de parallélisation de la phase 7 et l'alignement structural de la phase 9. Les autres étapes sont terminées et ont permis de répondre aux trois objectifs de la thèse : l'apprentissage d'un nouveau AS, l'encodage structural et la mise à disposition d'un site web pour la communauté scientifique ce qui permettra l'exploitation du nouvel alphabet structural.

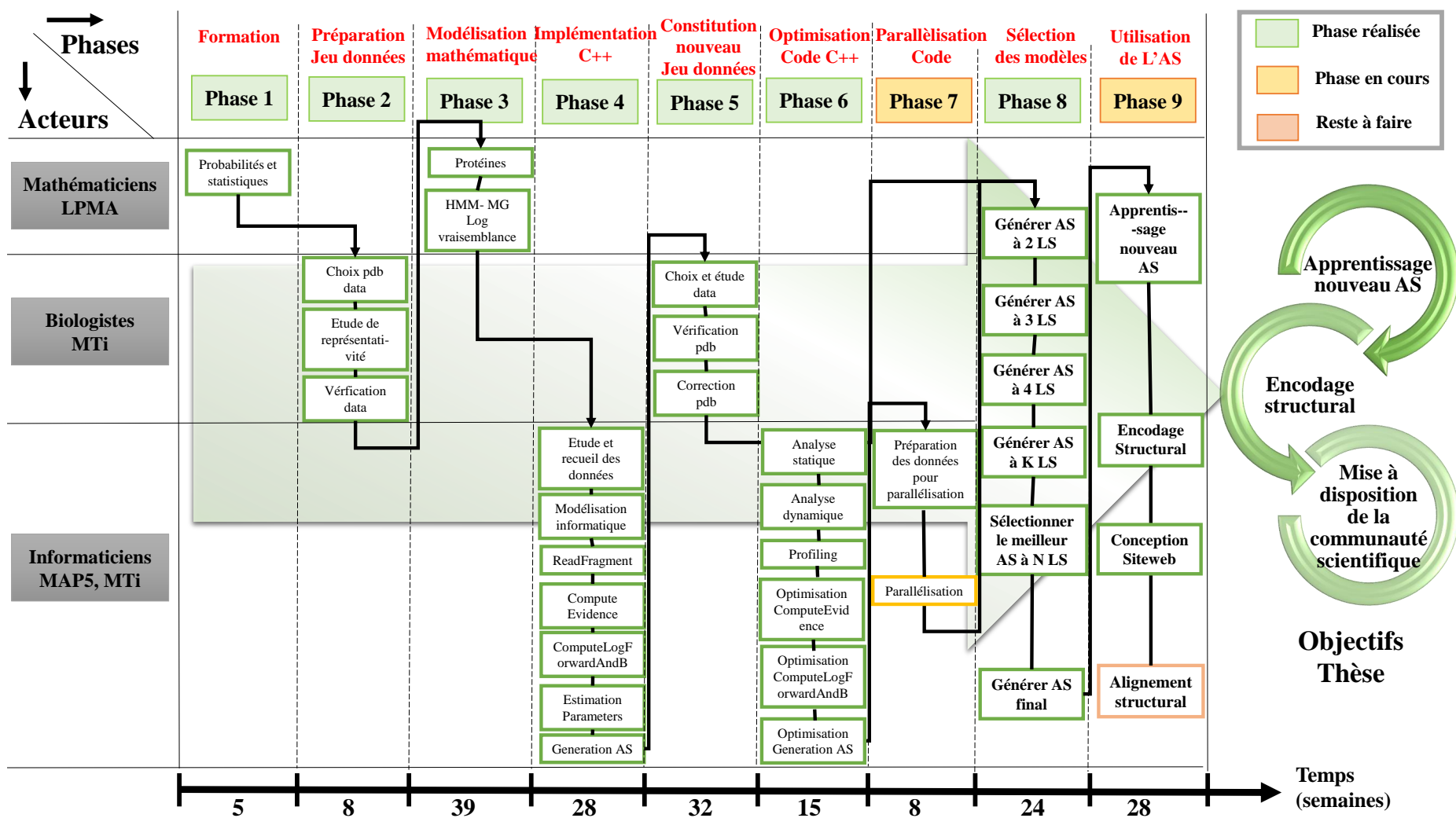
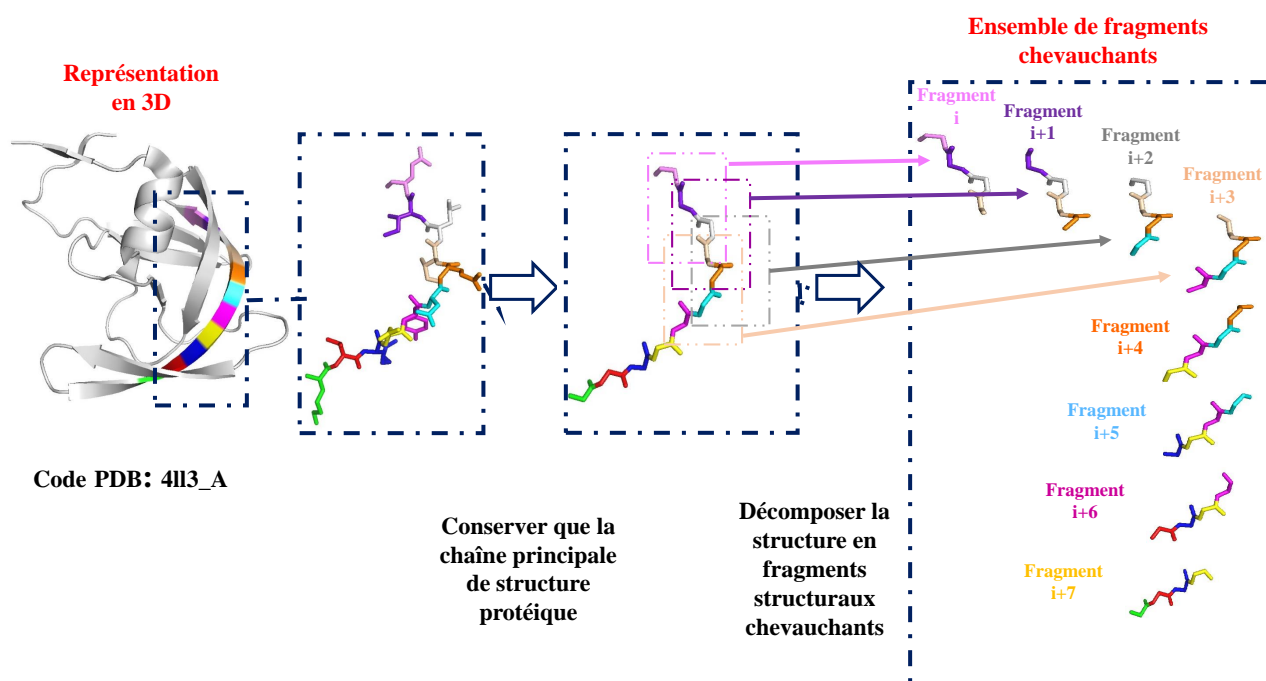


Figure 3.4 – Démarche méthodologique pluridisciplinaire du développement d'un alphabet structural.

### 3.3 Décomposition des structures tridimensionnelles en fragments structuraux

#### 3.3.1 Décomposition des structures 3D en fragments

La description de la structure tridimensionnelle des protéines est primordiale, mais cette structure s'avère très complexe à décrire. C'est la raison pour laquelle nous commençons tout d'abord par une décomposition de la structure, qui consiste à isoler sa chaîne principale. Celle-ci sera fractionnée ensuite en fragments structuraux chevauchants par trois résidus. Chaque fragment structural comporte quatre résidus, ainsi nous obtenons  $n$  fragments structuraux pour une structure d'une chaîne protéique de longueur  $(n + 3)$ . À titre illustratif, la figure 3.5 montre la décomposition d'un brin  $\beta$  de onze résidus, auquel on extrait la chaîne principale, celle-ci est ensuite découpée en fragments structuraux. Ceux-ci vont ensuite donner les huit fragments structuraux chevauchants de longueur 4 résidus.

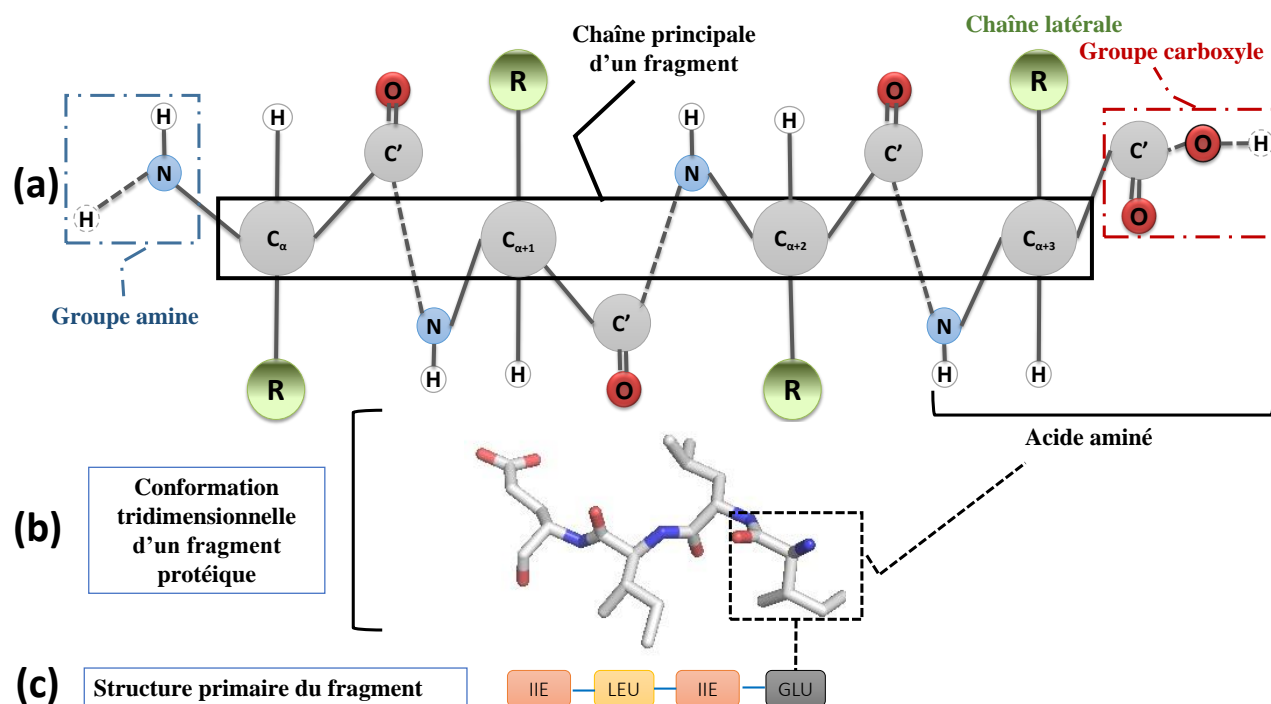


**Figure 3.5 – Décomposition d'une partie de la conformation tridimensionnelle de la chaîne 4ll3A en fragments structuraux.**

À gauche est représentée la structure tridimensionnelle de la chaîne 4ll3A (Code PDB) avec encadrement de la partie qui sera décomposée en fragments de longueur quatre. À droite l'ensemble des conformations tridimensionnelles des fragments structuraux chevauchants obtenus. Chaque résidu du fragment est colorié en une couleur différente.

La figure 3.6 décrit deux niveaux d'organisation d'un fragment structural : la structure primaire et tridimensionnelle. À titre illustratif, la partie (c) du schéma montre la séquence des quatre acides aminés présents au niveau de ce fragment, il s'agit donc de l'isoleucine (ILE), la leucine (LEU), l'isoleucine (ILE) et l'acide glutamique (GLU). Ce qui représente la structure primaire du fragment structural. Quant à la partie (b) du schéma, elle représente la conformation tridimensionnelle adoptée par le

fragment protéique dans l'espace. Celle-ci est associée à une structure biochimique décrite dans la partie (a). Elle présente une vue schématique détaillée de quatre résidus du fragment structural, choisis de la structure primaire de celui-ci. Un fragment structural est formé donc d'une chaîne principale constituée d'un squelette de quatre carbone alpha ( $C_\alpha$ ) auxquels sont greffés les groupes carboxyle ( $\text{COOH}$ ) et amine ( $\text{NH}_2$ ).



**Figure 3.6 – Représentation détaillée d'un fragment structural.**

La conformation tridimensionnelle du fragment structural est représentée sous forme de sticks par Pymol. Les atomes d'oxygènes sont colorés en rouge, les atomes d'azote en blue et les carbones en gris.

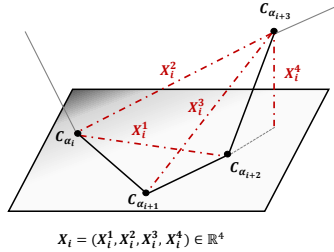
### 3.3.2 Jeu de données

Le jeu de données de structures protéiques tridimensionnelles choisi pour l'obtention de l'alphabet structural HMM-SA27, par [28], est non redondant, avec 30% d'identité. Il ne contient que des protéines d'une longueur de plus de 30 résidus. Celles-ci sont résolues par diffraction aux rayons X supérieure à 2,5Å. Ce jeu comporte 1 429 structures protéiques *sans manque de résidus* dans leurs chaînes. L'ensemble de celles-ci donne 332 493 fragments structuraux de longueur quatre chevauchant par trois. Parmi ceux-ci, deux jeux de données distincts ont été prélevés (jeu d'apprentissage et jeu de test), composés chacun d'une manière respective de 56 167 et 57 544 fragments protéiques, afin d'effectuer l'apprentissage des paramètres du HMM de HMM-SA27.

### 3.4 Description et caractérisation d'un fragment structural

Je rappelle que chaque structure de chaîne protéique peut être divisée en fragments structuraux chevauchants. La conformation d'une chaîne polypeptidique A d'une structure protéique, de longueur  $(n + 3)$  où  $n$  représente le nombre de fragments, est décrite de façon adéquate par l'utilisation des distances entre les coordonnées cartésiennes des atomes de C $\alpha$  de sa chaîne principale. Celle-ci est représentée sous la forme  $(C\alpha_1, C\alpha_2, C\alpha_3, \dots, C\alpha_{(n+3)})$ .

Ces fragments sont décrits par des descripteurs. Dans ce travail, la description du fragment structural prend en compte seulement les C $\alpha$  du fragment qui sont encadrés en noir dans le schéma (3.6). On définit un fragment structural  $i$  décrit selon [28] par un vecteur de distances entre les coordonnées cartésiennes des carbones alpha non successifs  $X_i = (X_i^1, X_i^2, X_i^3, X_i^4)$ , qui sont appelées des descripteurs calculés selon les formules suivantes.



$$X_i^1 = \|\overrightarrow{C\alpha_i C\alpha_{i+2}}\|$$

$$X_i^2 = \|\overrightarrow{C\alpha_i C\alpha_{i+3}}\|$$

$$X_i^3 = \|\overrightarrow{C\alpha_{i+1} C\alpha_{i+3}}\|$$

**Figure 3.7** – Les quatre descripteurs  $X_i = (X_i^1, X_i^2, X_i^3, X_i^4)$  du  $i^{\text{ème}}$  fragment structural de  $C\alpha_{i+1}$  jusqu'à  $C\alpha_{i+3}$ .

$$X_i^4 = \frac{\overrightarrow{C\alpha_i C\alpha_{i+2}} \wedge \overrightarrow{C\alpha_{i+2} C\alpha_{i+3}}}{\|\overrightarrow{C\alpha_i C\alpha_{i+2}} \wedge \overrightarrow{C\alpha_{i+2} C\alpha_{i+3}}\|} \times \overrightarrow{C\alpha_{i+3} C\alpha_{i+4}}$$

La longueur de quatre C $\alpha$  pour chaque fragment structural est choisie, d'une part pour être assez grande pour décrire les caractéristiques des éléments structuraux, c'est à dire un tour d'hélice, et d'autre part assez petite pour contenir des fragments uniformes de structure protéique, c'est à dire seulement des bouts d'hélices ou de feuillets.

La figure 3.7 montre une représentation du fragment structural et la définition de ses descripteurs. Selon [31], les quatre distances appelées descripteurs, donnent une représentation unique de la conformation des fragments structuraux. Ces descripteurs permettent donc de caractériser un fragment structural. Ainsi, les valeurs  $X_i^1, X_i^2, X_i^3$  désignent les distances Euclidiennes entre les coordonnées cartésiennes des C $\alpha$  non consécutifs du fragment. Le quatrième descripteur  $X_i^4$ , nommé « Produit Mixte », est associé à la position du quatrième C $\alpha$ , celui-ci est proportionnel au plan construit par les trois premiers C $\alpha$ . Chaque distance représente une signification,  $X_i^1$  décrit si l'entrée dans le fragment structural est étendue ou courte,  $X_i^2$  désigne la longueur globale de celui-ci, plus le fragment est étendu, plus sa valeur de  $X_i^2$  est grande et vice versa. Quant à  $X_i^3$ , elle représente l'étendue de la sortie du fragment structural, enfin  $X_i^4$  indique la direction du repliement polypeptidique et attribue un volume à la conformation du fragment structural.

### 3.5 Modélisation des structures tridimensionnelles par HMM

L'idée de base du développement d'un alphabet structural est de constituer une collection finie de fragments structuraux génériques, nommés lettres structurales, tout en supposant que cette collection soit satisfaisante pour représenter l'ensemble des fragments structuraux protéiques. Afin de construire cet alphabet, [28] supposent que chaque lettre structurale représente un ensemble de fragments structuraux, dont les descripteurs  $X_i$  sont générés par une loi multi-normale de dimension quatre, comme suit :

$$\mathcal{L}(X_i|S_i = s) \sim \mathcal{N}(\mu_s, \Sigma_s) \quad \forall s \in \mathcal{S}$$

Avec :

$m$  : nombre de lettres structurales

$\mathcal{S} = \{1, 2, \dots, m\}$  représente l'ensemble des états cachés ou lettres structurales.

$\mathcal{N}(\mu_s, \Sigma_s)$  représente la densité de la loi multi-normale de dimension quatre avec les paramètres  $(\mu_s, \Sigma_s)$  qui décrivent la moyenne des descripteurs et la variabilité de chaque descripteur ainsi que la covariance entre ceux-ci.

où :

$$\mu_s = (\mu_s^1, \mu_s^2, \mu_s^3, \mu_s^4) \quad \text{et} \quad \Sigma_s = (\Sigma_s^{i,j}) \quad i, j \in \{1, \dots, 4\}$$

Concernant l'organisation séquentielle des fragments structuraux, HMMSA-27 prend en compte les dépendances spatiales existantes entre ceux-ci en supposant qu'ils suivent un processus Markovien et donc en utilisant une chaîne de Markov cachée HMM d'ordre 1. Pour plus de détails sur les HMM, nous vous invitons à consulter la section B.2 de l'annexe B.

Dans cette section, nous présentons le modèle utilisé dans le cadre de la présente thèse. Le modèle que nous proposons apporte de nouvelles améliorations méthodologiques pour un meilleur et rigoureux encodage structural des chaînes protéiques en une séquence de lettres structurales. Nous commençons donc par la description des fragments structuraux puis nous expliquons notre modèle statistique qui se base sur les chaînes de Markov Cachées.

Nous modélisons donc la séquence  $X_{1:n} \in \mathbb{R}^{n \times 4}$  de  $n$  fragments structuraux par une chaîne de Markov cachée (HMM), où les états observés représentent les descripteurs des fragments et les états cachés sont les lettres structurales  $S_{1:n} \in \{1, 2, \dots, m\}^n$ . Chacune de ces lettres étant la structure 3D du fragment représentatif.

Il s'ensuit donc que la distribution jointe entre  $X$  et  $S$  du modèle résultant, illustrée par la figure 3.8, s'écrit de la façon suivante :

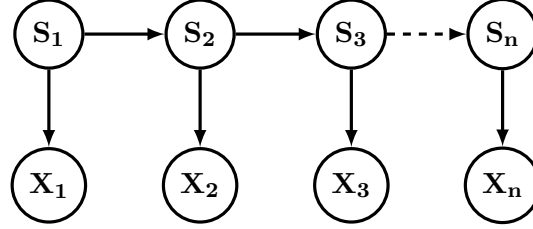
$$\mathbb{P}(X_{1:n}, S_{1:n}) = \underbrace{\mathbb{P}(S_1) \prod_{i=2}^n \mathbb{P}(S_i|S_{i-1})}_{\text{Partie Markov}} \times \underbrace{\prod_{i=1}^n \mathbb{P}(X_i|S_i)}_{\text{Partie Emission}} \quad (3.1)$$

Sachant que la chaîne de Markov a une distribution initiale uniforme et une matrice de transition homogène  $\pi \in \mathbb{R}^{m \times m}$  nous notons  $e_i(S_i) = \mathbb{P}(X_i|S_i)$ . Ainsi, nous obtenons l'équation suivante



simplifiée :

$$\mathbb{P}(X_{1:n}, S_{1:n}) = \frac{1}{m} \prod_{i=2}^n \pi(S_{i-1}, S_i) \prod_{i=1}^n e_i(S_i) \quad (3.2)$$



**Figure 3.8 – Le modèle HMM-SA27.**

$X_i \in \mathbb{R}^{n \times 4}$  représentent les descripteurs des fragments et  $S_i \in \{1, 2, \dots, 27\}^n$  sont les lettres structurales.

En ce qui concerne la distribution d'émission du HMM, l'équation est la suivante :

$$\log e_i(s) = \text{cste.} - \frac{1}{2} \log \det \Sigma_s - \frac{1}{2} (X_i - \mu_s)^T \Sigma_s^{-1} (X_i - \mu_s) \quad (3.3)$$

### 3.5.1 Encodage des structures protéiques

L'assignation des lettres structurales aux fragments structuraux faisant partie d'une chaîne protéique est nommée *l'encodage structural* de celle-ci. Cet encodage structural prend en considération la forme géométrique des fragments structuraux grâce aux descripteurs géométriques qui lui sont associés, l'enchaînement spatial entre ces fragments ainsi que les données manquantes qui sont souvent rencontrées dans les fichiers au format PDB.

#### 3.5.1.1 Maximum *a posteriori*

L'encodage d'une structure 3D (séquence de fragments) en une séquence structurale peut être obtenu en calculant le Maximum *a posteriori* (MAP) [18] défini par :

$$\text{MAP}_{1:n} = \arg \max_{S_{1:n}} \mathbb{P}(S_{1:n} | X_{1:n}) = \arg \max_{S_{1:n}} \mathbb{P}(X_{1:n}, S_{1:n}) \quad (3.4)$$

Pour calculer cette quantité nous introduisons les quantités de *max-forward* et *max-backward*, qui sont présentées par l'annexe B par les définitions B.3.3 et B.3.4. Ainsi nous avons :

$$F_1^{\max}(s) = e_1(s)/m \quad F_i^{\max}(s) = \max_{S_{1:i-1}} \mathbb{P}(X_{1:i}, S_{1:i-1}, S_i = s) \quad \text{pour } i = 2 \dots n \quad (3.5)$$

$$B_n^{\max}(s) = 1 \quad B_{i-1}^{\max}(s) = \max_{S_{i:n}} \mathbb{P}(X_{1:i}, S_{i:n}, S_i = s) \quad \text{pour } i = n \dots 2 \quad (3.6)$$

En prenant en considération le corollaire B.3.2 présenté avec sa preuve B.3.3 dans l'annexe B, les quantités *forward* et *backward* peuvent être calculées donc récursivement pour  $i \in \{2, \dots, n\}$  par :

$$F_i^{\max}(s) = \max_r F_{i-1}(r) \times \pi(r, s) \times e_i(s) \quad \text{et} \quad B_{i-1}^{\max}(r) = \max_s \pi(r, s) \times e_i(s) \times B_i(s) \quad (3.7)$$

En tenant compte du théorème B.3.1 et de sa preuve B.3.1 nous avons finalement :

$$\text{MAP}_i = \arg \max_s F_i^{\max}(s) B_i^{\max}(s) \quad \text{pour tout } i = 1 \dots n \quad (3.8)$$

### 3.5.1.2 Distribution marginale *a posteriori*

Alternativement, en considérant le corollaire B.3.3 de l'annexe B, nous pouvons nous focaliser sur la distribution marginale *a posteriori* plutôt que sur le maximum *a posteriori* :

$$\text{POST}_i(s) = \mathbb{P}(S_i = s | X_{1:n}) = \frac{\mathbb{P}(S_i = s, X_{1:n})}{\mathbb{P}(X_{1:n})} \quad (3.9)$$

Ces quantités peuvent être calculées en utilisant la notion de MAP abordée dans la section précédente tout simplement en remplaçant toutes les occurrences de 'max' par des 'sum' dans les équations : 3.5, 3.6 et 3.7 et en tenant compte du théorème B.3.1, les corollaires B.3.1 et B.3.3. Une fois les quantités des *sum-Forward* et *sum-Backward* calculées, il est possible d'obtenir immédiatement la distribution marginale *a posteriori* :

$$\text{POST}_i(s) \propto F_i^{\text{sum}}(s) B_i^{\text{sum}}(s) = \frac{F_i^{\text{sum}}(s) B_i^{\text{sum}}(s)}{\sum_r F_i^{\text{sum}}(r) B_i^{\text{sum}}(r)} \quad (3.10)$$

Il est à noter que la quantité  $\arg \max_s \text{POST}_i(s)$  qui est une distribution marginale *a posteriori*,  $\arg \max_s \text{POST}_i(s)$  n'est pas nécessairement égale au Maximum *a posteriori* (même si c'est souvent le cas quand la distribution *a posteriori* marginale est assez précise).

La distribution marginale *a posteriori* POST peut aussi être utilisée pour quantifier le niveau d'incertitude de l'encodage de la lettre structurale en calculant l'entropie marginale *a posteriori* et la taille effective.

$$\text{ENT}_i = - \sum_s \text{POST}_i(s) \log \text{POST}_i(s) \quad \text{et} \quad \text{NEFF}_i = \exp(\text{ENT}_i) \quad (3.11)$$

$\text{ENT}_i$  est une mesure du désordre pour la lettre structurale associée au fragment  $i$ , tirée de l'entropie de Shannon [162]. Si la distribution *a posteriori* est Dirac<sup>1</sup>, l'entropie  $\text{ENT}_i = 0$  est minimale, si l'incertitude d'encodage est maximale avec  $\text{POST}_i(s) = 1/m$ , l'entropie  $\text{ENT}_i = \log m$  est maximale.

Quant à la taille effective  $\text{NEFF}_i \in [1, m]$ , elle donne une simple interprétation de l'entropie comme le nombre effectif de lettres structurales acceptables pour le fragment  $i$ . En d'autres termes, cette taille effective mesure le nombre équivalent d'états de sorties possibles de l'état  $i$ . S'il est égal à 1 alors seulement un état de transition est possible. S'il est égal au nombre de lettres donc tous les états sont des transitions équiprobables.

---

1. La distribution de Dirac est une distribution qui prend une valeur infinie en 1, et la valeur zéro dans tous les autres cas

### 3.5.2 Gestion des données manquantes

Lors des traitements de structures 3D, Il est assez fréquent que des positions de C $\alpha$  soient manquantes. Par conséquent, plusieurs descripteurs de fragments seront manquants. Il est donc nécessaire de traiter efficacement ces données manquantes, ce qui est assez simple sous l'hypothèse gaussienne.

Soit  $J \subset \{1, 2, 3, 4\}$  le sous ensemble des descripteurs non manquants du fragment  $i$ ,  $J = \emptyset$  si tous les descripteurs sont manquants et  $J = \{1, 2, 3, 4\}$  si aucun n'est manquant. Alors la probabilité d'émission du fragment structural  $i$  pour la lettre structurale  $s$  est obtenue en considérant la restriction  $X_i[J] \sim \mathcal{N}(\mu_s[J], \Sigma_s[J, J])$ . Par convention  $e_i(s) = 1$  pour tout  $s$  si  $J = \emptyset$ , ce qui signifie que la totalité du fragment manquant  $i$  est totalement non informative. À titre d'illustration nous présentons quelques cas de gestion de manque de données :

- si aucun résidu n'est manquant au fragment  $i$  alors  $J = \{1, 2, 3, 4\}$  et le vecteur des descripteurs  $X_i = (X_i^1, X_i^2, X_i^3, X_i^4)$  donc la probabilité d'émission  $e_i(s)$  du fragment  $i$  pour la lettre  $s$  sera calculée par la formule 3.3 sans aucun changement dans les paramètres;
- si le premier résidu du fragment  $i$  est manquant alors  $J = \{2, 3, 4\}$  et le vecteur des descripteurs  $X_i = (\text{NA}, X_i^2, X_i^3, X_i^4)$  donc la probabilité d'émission  $e_i(s)$  du fragment  $i$  pour la lettre  $s$  sera calculée par la restriction  $X_i[J] \sim \mathcal{N}(\mu_s[J], \Sigma_s[J, J])$ . Ceci revient à appliquer la formule 3.3 avec les nouveaux  $\mu_s$  et  $\Sigma_s$  tels que :

$$\mu_s = (\mu_2, \mu_3, \mu_4) \text{ et } \Sigma_s = (\Sigma_s^{i,j}) \quad i, j \in J \text{ donc } \Sigma_s = \begin{pmatrix} \Sigma_{22} & \Sigma_{23} & \Sigma_{24} \\ \Sigma_{32} & \Sigma_{33} & \Sigma_{34} \\ \Sigma_{42} & \Sigma_{43} & \Sigma_{44} \end{pmatrix}$$

- dans le cas où les deux premiers résidus sont manquants au fragment  $i$ , alors  $J = \{3, 4\}$  et  $X_i = (\text{NA}, \text{NA}, X_i^3, X_i^4)$ ,  $\mu_s = (\mu_3, \mu_4)$  et  $\Sigma_s = (\Sigma_s^{i,j}) \quad i, j \in J$  donc

$$\Sigma_s = \begin{pmatrix} \Sigma_{33} & \Sigma_{34} \\ \Sigma_{43} & \Sigma_{44} \end{pmatrix}$$

- dans le cas où les trois premiers résidus sont manquants au fragment  $i$ , alors  $J = \{4\}$  et  $X_i = (\text{NA}, \text{NA}, \text{NA}, X_i^4)$ ,  $\mu_s = (\mu_4)$  et  $\Sigma_s = (\Sigma_s^{i,j}) \quad i, j \in J$  donc  $\Sigma_s = \Sigma_{44}$ ;
- si tous les résidus sont manquants alors  $J = \emptyset$  et  $e_i(s) = 1$ .

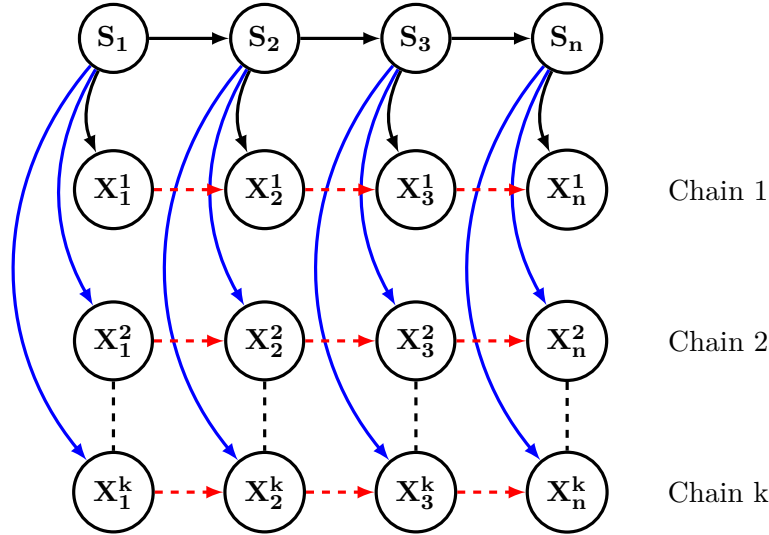
### 3.5.3 Gestion des chaînes multiples

Dans un fichier PDB, certaines protéines peuvent être représentées plusieurs fois. Cela peut être de simples répliques ou des structures tridimensionnelles obtenues dans différentes conditions (le fait qu'une protéine soit dans un état lié ou pas, par exemple<sup>2</sup>). Il est évidemment possible d'effectuer un encodage pour chacune de ces répliques mais il y a aussi une autre possibilité : construire un encodage consensus à partir de l'ensemble des répliques de la protéine. Ceci est rendu possible en considérant le modèle de la figure 3.9 qui résulte de la loi jointe suivante :

---

2. Une protéine peut être associée ou non avec différents composés (ligands, ARN, ADN)

$$\mathbb{P}(X_{1:n}^{1:k}, S_{1:n}) = \underbrace{\mathbb{P}(S_1) \prod_{i=2}^n \mathbb{P}(S_i | S_{i-1})}_{\text{encodage consensus}} \times \prod_{j=1}^k \underbrace{\prod_{i=1}^n \mathbb{P}(X_i^j | S_i)}_{\text{Emission de la chaîne } j} \quad (3.12)$$



**Figure 3.9 – Le modèle SAFlex avec des chaînes multiples.**

Les flèches en bleu représentent les différentes chaînes multiples pour une protéine. Les flèches en rouge montrent les dépendances entre les descripteurs des fragments.

L'algorithme d'encodage précédent peut être facilement adapté au nouveau contexte en changeant simplement les expressions de  $e_i(s)$  comme suit :

$$\log e_i(s) = \text{cste.} - \sum_{j=1}^k \frac{1}{2} \left( \log \det \Sigma_s + (X_i^j - \mu_s)^T \Sigma_s^{-1} (X_i^j - \mu_s) \right) \quad (3.13)$$

Il convient de noter que les données manquantes peuvent aussi être prises en compte et gérées dans ce contexte tel expliqué dans la section 3.5.2. Ce qui n'était pas pris en compte par l'ancien modèle. Le modèle que nous proposons prend donc en compte plusieurs chaînes possibles (conformations) pour les protéines, ce qui représente un des atouts de sa flexibilité. Par contre s'il s'agit d'une seule conformation associée à une protéine, il suffit d'appliquer les formules du modèle précédent mis en place par notre modèle aussi. Cependant, l'encodage consensus ne peut être réalisé que s'il y a au moins deux chaînes (répliques) de la même protéine. Celles-ci doivent avoir la même longueur.

### 3.6 Implémentation du nouvel alphabet structural

L'implémentation du nouvel alphabet structural consiste à la mise en pratique du modèle statistique proposé dans la section précédente. Cette phase est donc primordiale afin de concrétiser notre réflexion sur le développement de cet alphabet. Cette implémentation n'a pas été très simple et un certain nombre de difficultés ont été rencontrées. En effet il s'agit premièrement de traiter des fichiers PDB présentant fréquemment un certain nombre d'anomalies (qui ont été déjà détaillées dans la sec-

tion "Les fichiers de données de la banque PDB" du chapitre "Un nouveau parser de fichiers PDB" à la page 12). Deuxièmement, l'implémentation du calcul des densités gaussiennes a représenté un travail fastidieux, celle-ci sera développée ultérieurement. Troisièmement, l'implémentation des calculs des probabilités (évidence et marginale) qui ont des valeurs extrêmement faibles a rendu nécessaire le passage à l'échelle logarithmique. Une dernière difficulté principale statistique concerne l'implémentation de l'algorithme EM (voir la section B.1), l'estimation de ses paramètres, les notions *forward* et *backward* ainsi que les chaînes de Markov cachées (HMM), qui ont exigé un travail lent et ardu.

Dans cette section nous expliquons la démarche logicielle suivie pour l'implémentation du programme de génération de l'AS en détaillant sa modélisation, sa conception et ce en tenant compte des besoins. Ce qui induit sa décomposition en modules. Enfin, nous développons l'implémentation de l'algorithme EM.

L'alphabet structural SAFlex a été implémenté en langage C++ [167]. En fait, les performances liées au temps d'exécution étaient le facteur principal dans le choix du langage C++. En effet, il est très rapide car il est considéré proche du bas niveau. De plus, Il est multi paradigme car il offre plusieurs styles de programmation tels que : la programmation orientée objets, la programmation procédurale et la programmation générique grâce aux Templates. D'autant plus qu'il est écrit en un anglais simple et facile à comprendre par un humain. Enfin, le langage C++ est portable puisqu'un code bien écrit peut être compilé sur plusieurs plateformes (Windows, mac et Linux).

Aussi, il est particulièrement bien documenté avec un accès plus facile à l'aide et offre de nombreuses bibliothèques disponibles gratuitement. Nous en avons utilisé quelques unes telles que : STL, Boost C++, ..., etc. Notre choix s'est porté sur l'environnement de développement intégré (IDE) Code::Blocks [124] pour l'implémentation de nos programmes en raison de sa facilité prise en main et de ses fonctionnalités très avancées pour une utilisation plus poussée. De plus, il est dédié principalement pour les langages C et C++, mais il peut supporter d'autres langages. Il est libre et multiplate-forme. Nous l'avons donc utilisé sur deux systèmes d'exploitation : Windows 7 et Linux Ubuntu.

### 3.6.1 Analyse, conception et modélisation des besoins

Ce programme est destiné à différents acteurs (biologistes, statisticiens et informaticiens). Son objectif principal est de générer un alphabet structural. Cela revient à estimer les paramètres optimaux des lettres structurales composant l'alphabet. En effet, il faut générer plusieurs alphabets structuraux de paramètres différents et choisir le plus optimal. Par conséquent, l'architecture logicielle du programme doit donc répondre aux choix stratégiques qui définissent la qualité du programme (performance, maintenabilité, adaptabilité et fiabilité). Ce sont les raisons pour lesquelles nous avons choisi alors une modélisation orientée objet de nos besoins en utilisant le langage C++.

Cette approche objet permet de modéliser les informations ainsi que les actions d'une entité. De cette façon, notre démarche consiste à décrire les entités principales de notre programme. Il s'agit donc de la protéine, de la chaîne protéique, du fragment structural et de la lettre structurale. Nous définissons alors les objets : « PROTEIN », « CHAIN », « FRAGMENT » et « SL » respectivement. Puis nous décrivons les relations entre ces objets. Ainsi, notre objet élémentaire, dans ce programme

est le fragment structural. En effet l'objet « CHAIN » est constitué d'une séquence de fragments structuraux et l'objet « SL », qui n'est qu'un fragment structural représentatif.

### 3.6.2 Décomposition en modules du programme du nouvel alphabet structural

La nouvelle architecture du code source de l'alphabet structural est modulaire ce qui facilite son implémentation et sa réutilisabilité ainsi que l'avancement progressif sur le code. Cette façon d'implémenter aide à structurer et à organiser le code source en unités logiques. Chacune de ces unités représente un module et comporte un ensemble de routines (fonctions).

L'architecture générale de l'implémentation de chaque module est organisée en trois types de fichiers : le fichier d'application (main.cpp), le(s) fichier(s) d'interface « Header (.hpp) » et le(s) fichier(s) d'implémentation « Source (.cpp) ». Le fichier *Application* contient une fonction importante et spéciale du programme dénommée *main*, en effet celle-ci marque le début du programme. Le fichier *Header* comprend les attributs et les prototypes des méthodes des classes de notre modèle. Quand au fichier *Source*, il comporte la définition et l'implémentation des méthodes. De plus cette nouvelle implémentation est orientée objet, ce qui permet de tirer profit des avantages de ce style de programmation proposé par le langage C++.

Le développement du nouvel alphabet structural SAFlex a nécessité l'implémentation de quatre grands modules différents développés indépendamment. La figure 3.10 illustre bien la complexité du programme de développement de l'alphabet. Chaque module est représenté par une couleur et chaque fichier de sortie prend la même couleur que le module dont il est issu. Chaque fichier de sortie suit le format FASTA, celui-ci rend le traitement et la lecture des résultats du fichier plus aisé. Pour bien illustrer les différents fichiers de sortie des modules, nous avons attribué à chaque fichier un numéro dans le schéma (voir figure 3.10). L'ensemble des modules génèrent donc, les fichiers de sortie suivants :

- (1). fichier des descripteurs géométriques des fragments protéiques (sortie du module 1);
- (2). fichier des probabilités d'émissions des fragments contenant les  $\log(P(X_i|S_i))$  pour chaque fragment  $i$  de la chaîne protéique (sortie du module 2);
- (3). fichier des log des probabilités *forward*  $F_i^{\text{sum}}$  (sortie du module 3);
- (4). fichier des log des probabilités *backward*  $B_i^{\text{sum}}$  (sortie du module 3);
- (5). fichier des log des probabilités max *forward*  $F_i^{\text{max}}$  (sortie du module 3);
- (6). fichier des log des probabilités max *backward*  $B_i^{\text{max}}$  (sortie du module 3);
- (7). fichier des probabilités *a posteriori* des fragments structuraux  $\log(P(S_i|X_i))$  (sortie du module 4);
- (8). fichier modèle comprenant le nombre des lettres structurales et la matrice de transition ( $\pi$ ) entre les lettres structurales (sortie du module 4);
- (9). fichier des paramètres des lettres structurales, contenant la moyenne ( $\mu_{\text{LS}}$ ), la matrice de covariance ( $\Sigma_{\text{LS}}$ ), l'inverse de la matrice de covariance ( $\Sigma^{-1}$ ), et  $\log(|\Sigma_{\text{LS}}|)$  (sortie du module 4), Où LS : représente la lettre structurale.

Il est primordial de signaler que tous les résultats de sortie de nos modules sont en logarithme. Les raisons de ce choix d'implémentation sont détaillées dans la section 3.6.4.

Dans ce qui suit, nous expliquons le fonctionnement de chacun des modules.

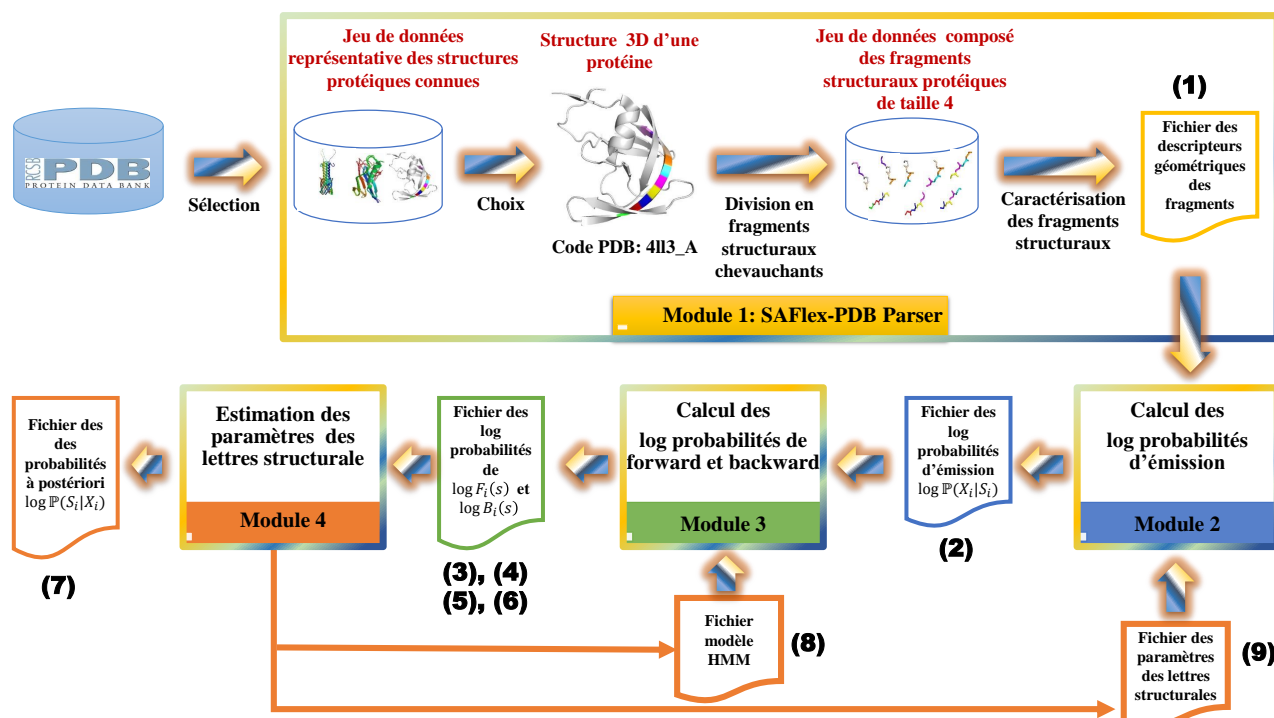


Figure 3.10 – L'architecture modulaire du code source du nouvel alphabet structural SAFlex.

### 3.6.2.1 Module 1

Le premier module est celui de prétraitement et lecture des fichiers PDB (*SAFlex-PDB parser*), il est destiné à l'analyse des structures tridimensionnelles et à l'obtention des fichiers de données concernant les fragments structuraux protéiques (la taille de fragment, le nombre de descripteurs géométriques ainsi que ces derniers, les résidus composant chaque fragment). Ce module traite toutes les chaînes contenues dans les fichiers PDB mais ne découpe que les chaînes protéiques en fragments structuraux. Les chaînes tels que : ADN, ARN ne sont pas divisées ni pris en compte pour le développement du nouveau alphabet structural. Le fichier résultat principal de ce module est le fichier des descripteurs géométriques des fragments protéiques (fichier N° (1) dans la figure 3.10). Il convient de noter que ce premier module gère et traite les données manquantes (résidus et atomes  $C\alpha$ ) au niveau des chaînes protéiques, comme il est capable de détecter et signaler les trous dans les chaînes. Le fonctionnement de ce module ne sera pas expliqué durant cette section car il est détaillé au niveau du chapitre "Un nouveau parser de fichiers PDB" à la page 11, en effet, il fait partie de nos contributions principales de cette thèse de doctorat.

### 3.6.2.2 Module 2

Le deuxième module est alimenté par les données de sorties du module 1 (fichier de descripteurs géométriques) afin de générer le fichier (3) des log probabilités d'émissions des fragments structuraux.

Ce module nécessite le fichier (9) des paramètres des lettres structurales. En effet les probabilités d'émission suivent une loi normale multivariée, ce qui revient au calcul des densités gaussiennes.

#### 3.6.2.2.1 Implémentation du calcul des densités gaussiennes

L'implémentation du calcul des densités gaussiennes a nécessité l'emploi et la maîtrise de la librairie *Boost C++* [156]. En effet il s'agit d'implémenter la formule complexe de la densité de la loi normale à quatre dimensions. Celle-ci a été développée au niveau de la section A.1.2 de l'annexe "Estimateur du maximum de vraisemblance" à la page 103. Le travail a consisté surtout à traiter des matrices tridimensionnelles et à calculer les matrices de covariance inverses de dimension quatre ainsi que leurs déterminants. Il est à noter que la densité gaussienne du fragment structural, qui représente la probabilité d'émission de celui-ci, est calculée dans les deux cas suivants :

- a) cas où le fragment n'a pas de manque de descripteurs (les quatre descripteurs géométriques sont présents),
- b) cas où le fragment a un manque d'un ou plusieurs descripteurs, au pire les quatre sont manquants.

Le calcul de ces deux cas a été déjà expliqué dans la section "Gestion des données manquantes" à la page 52.

#### 3.6.2.3 Module 3

L'objectif du troisième module est de réaliser un traitement statistique sur les données protéiques, à partir des fichiers émissions et paramètres du modèle proposé (contenant le nombre de lettres structurales, la matrice de transition entre celles-ci, les moyennes et les covariances associées à chacune des lettres structurales). Ce traitement a pour but de générer les fichiers concernant les *backward* et *forward* (pour chaque fragment structural). Ainsi les fichiers de sortie (3) et (4) nous serviront pour calculer les distributions marginales *a posteriori* (POST) et les fichiers (5) et (6) nous serviront pour le calcul des *Maximum a posteriori* (MAP), autrement dit nous les utilisons pour l'encodage structural le plus probable des fragments structuraux.

#### 3.6.2.4 Module 4

Le quatrième module estime et met-à-jour les paramètres des lettres structurales qui suivent une loi normale de dimension quatre (les moyennes et les covariances associées à chacune des lettres structurales). Ce module estime aussi les probabilités de transition entre les lettres structurales de la matrice de transition. Il utilise les fichiers (3) et (4) de sortie du module 3 pour calculer les probabilités *a posteriori* des lettres structurales. De plus, je souligne aussi que cette estimation des paramètres prend en compte et gère les données manquantes. En effet, il calcule pour chaque cas de manque de descripteur, la moyenne et la matrice de covariance associées à celui-ci. Les formules utilisées pour chaque cas ont été déjà expliquées dans la section 3.5.2. Aussi, il permet de prendre en compte lors de l'estimation les chaînes multiples d'une même protéine. Celle-ci a été détaillée dans la section 3.5.3 de ce chapitre.



### 3.6.3 Implémentation de l'algorithme EM

L'implémentation de l'algorithme EM, décrit dans la section B.1 de l'annexe B, n'était pas aisée, car la complexité de celui-ci réside dans le fait d'implémenter les formules mathématiques nécessaires pour la mise en œuvre des étapes E et M de l'algorithme. Chaque étape est implémentée par une ou plusieurs fonctions qui nécessitent un test, une vérification et une validation pour chacune. En effet pour nous assurer que les résultats obtenus sont corrects il faut tester et déboguer chaque étape de l'algorithme séparément. Dans notre cas nous avons codé l'étape E, tester et vérifier les sorties de chaque fonction faisant partie de celle-ci. Puis nous avons implémenté et tester les fonctions de l'étape M. Il est primordial que les tests et les vérifications faites ne soient pas manuelles mais automatiques. Effectivement, nous avons vérifié les résultats de ces formules qui opèrent sur des matrices multidimensionnelles, en les programmant non seulement en C++ mais en R [43]. Celui-ci facilite les calculs algébriques et les opérations matricielles, ce qui permet une vérification et une validation du résultat. Aussi, nous avons procédé à reimplémenter de nouvelles formules mathématiques pour confirmer que les résultats obtenus par les étapes sont fiables.

En outre, lancer cet algorithme revient à lancer tout d'abord la fonction d'initialisation des paramètres des lettres structurales. Celle-ci traite deux cas distincts :

**cas 1 :** initialisation des paramètres pour un alphabet de deux lettres,

**cas 2 :** initialisation des paramètres de l'alphabet structural de  $m$  lettres en prenant en compte les paramètres et les résultats<sup>3</sup> de l'AS de  $(m - 1)$  lettres structurales construit auparavant.

De plus, l'implémentation de l'algorithme EM fait appel à tous les modules cités dans la section 3.6.2 de ce chapitre. Il s'agit donc de lancer l'exécution des fonctions de :

- a) calcul des : log probabilités d'émission,
- b) calcul des : log probabilités *forward* et log probabilités *backward*,
- c) calcul des : log probabilités max *forward* et log probabilités max *backward*,
- d) l'estimation des paramètres : moyenne, matrice de covariance des lettres ainsi que la matrice de transition entre celles-ci.

L'algorithme EM est itératif, ce qui implique qu'il nécessite un critère d'arrêt, ce dernier représente l'erreur relative calculée entre les anciens paramètres et les nouveaux paramètres estimés. Si cette erreur est inférieure à  $10^{-5}$  alors l'algorithme converge sinon il ne converge pas. Il est à noter que la construction d'un alphabet structural de  $m$  lettres nécessite le lancement de l'algorithme EM plusieurs fois jusqu'à la convergence de celui-ci. La démarche suivie a été déjà expliquée lors de la section 3.1.1.

L'algorithme EM fournit également la log vraisemblance, la dimension et le critère BIC du modèle statistique implémenté. Il permet aussi d'ordonner les lettres structurales selon l'ordre croissant des occurrences associées à celles-ci et de permuter la matrice de transition.

---

3. il s'agit de prendre en compte les résultats du fichier des probabilités *a posteriori* pour tous les fragments structuraux obtenu au moment de la génération de l'alphabet de taille  $(m - 1)$  de LS.

### 3.6.4 Implémentation en échelle logarithmique

Pour éviter les problèmes *underflow*<sup>4</sup>, toutes les formules de calcul ont été implémentées en échelle logarithmique. Ce passage à l'échelle logarithmique étant nécessaire car le nombre des fragments structuraux à traiter est assez important, ce qui génère des probabilités très faibles (proches du zéro). Par conséquent, cet échelle permet une meilleure lisibilité des formules de calcul en rendant les opérations de multiplication additives.

## 3.7 Optimisation du code du nouvel alphabet

Au cours de notre implémentation des différents modules techniques, associés au développement du nouvel alphabet structural, nous avons été confrontés aux problématiques de performance et de lenteur d'exécution. Cette lenteur est due à trois problèmes majeurs spécifiques à notre contribution scientifique :

1. les modèles mathématiques utilisés sont gourmands en temps de calcul (opérations algébriques, transformations logarithmiques, algorithme EM, ..., etc.);
2. l'obtention de l'alphabet structural a nécessité le traitement d'une quantité massive de données (traitement de plusieurs jeux de données pour l'AS et analyse de plus de 8000 fichiers pdb pour le *parser* par exemple);
3. le découpage en plusieurs modules a présenté un temps d'exécution très lent ce qui réduisait beaucoup les performances des modules.

Ceci nous a amené à réfléchir à une optimisation des programmes codés. Je me suis contentée alors de présenter une approche d'optimisation en plusieurs étapes. La première étape consiste à dresser le profil de performance de l'application et d'identifier les parties gourmandes en temps CPU. Une fois ces codes identifiés, on pourrait analyser leurs codes et déterminer les types de parallélismes qu'on pourrait exploiter pour les accélérer. La mise en œuvre de ces optimisations est alors déléguée aux outils appropriés. Nous pourrions, ensuite, réitérer les opérations jusqu'à ce qu'on obtienne un résultat satisfaisant. C'est la méthodologie que nous avons utilisé dans le cadre de cette thèse et qui sera détaillée dans ce qui suit. Je rappelle que les différents aspects techniques de cette méthodologie d'optimisation ont fait l'objet de l'annexe C. Il convient de noter que dans ce travail, nous nous intéressons uniquement à l'optimisation de la vitesse d'exécution des modules développés dans la section précédente.

### 3.7.1 Tests de performance

Afin de dresser le profil de performance, il est nécessaire de préparer un jeu de tests pour chaque module, qui consiste à choisir les données significatives concernant les paramètres d'entrées des modules. Ce jeu de tests de performances a été réalisé sur une machine ayant une mémoire vive RAM de taille 8 Go avec comme modèle de processeur « Intel(R) Xeon(R) CPU E5430 @ 2,66 GHz ».

---

4. les calculs de probabilités donnent un nombre très petit pour être stocké dans le type de données considéré. Ceci provoque un bug d'exécution qui a pour conséquence de quitter le programme

Les tableaux des résultats obtenus lors du jeu de tests de performances avant optimisation des modules sont présentés dans l'annexe D (tableaux D.1, D.2, D.3 et D.4).

### 3.7.2 Analyse des modules

#### 3.7.2.1 Analyse statique et dynamique des modules

L'analyse statique et dynamique de tous les modules montre que le développement de ceux-ci ne présente pas d'erreurs statiques, ni dynamiques en d'autres termes, il n'y a pas d'erreurs de syntaxe sur les notions utilisées dans la programmation (classes, map, vector, tableau, ..., etc.), pas de fuites de mémoire (exemple : allocations de mémoire), de fuites de ressources (par exemple les fichiers qu'on utilise), et l'utilisation des fonctions est valide selon la STL.

#### 3.7.2.2 Profiling des modules

Le profiling donne une idée très claire sur les fonctions des modules consommatrices du CPU (les fonctions qui prennent le plus de temps au niveau CPU). La figure 3.11 présente donc les dix fonctions les plus consommatrices du temps CPU du module 4. Il est réalisé grâce à l'outil *callgrid* [125], où les fonctions sont triées par la colonne *Self*.

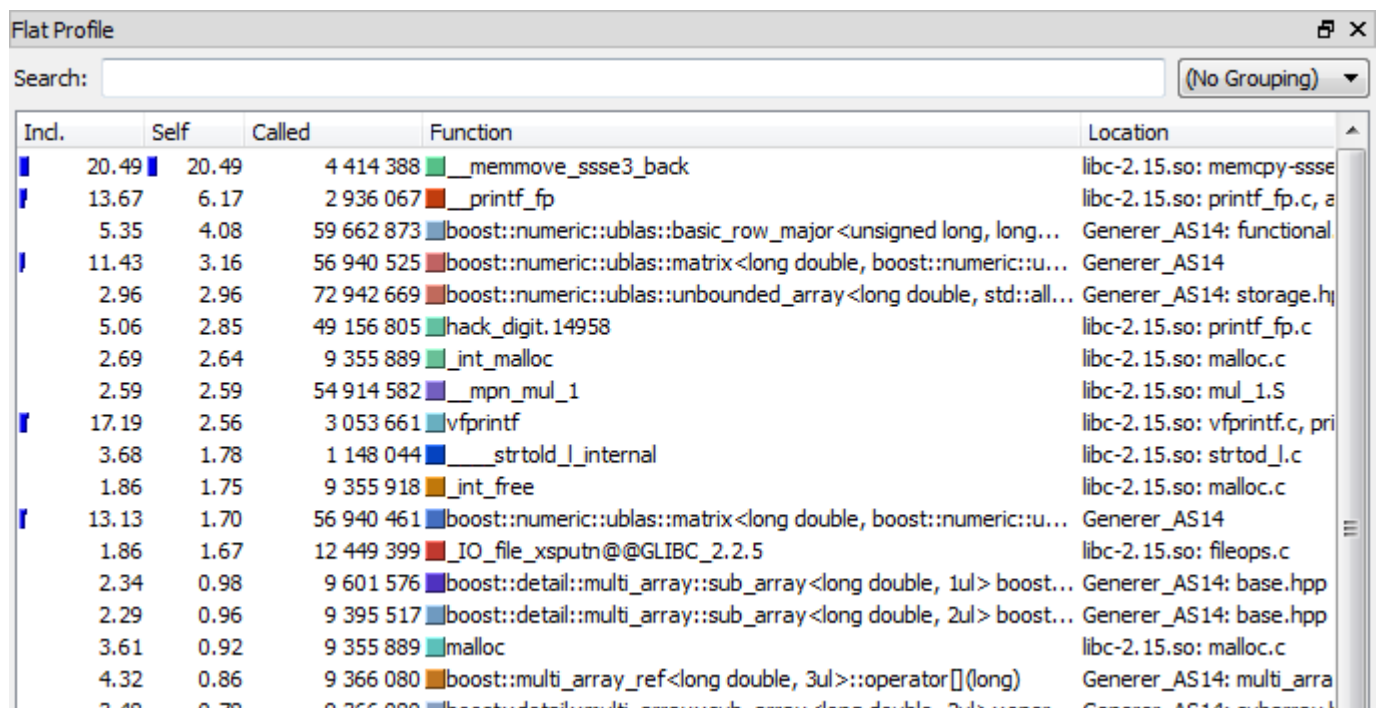


Figure 3.11 – Le profiling du module 4 montrant les dix premières fonctions consommatrices du CPU avant optimisation, obtenu par l'outil *callgrid*

Le profiling du module 4 montre que 30% du temps d'exécution de celui-ci est occupé à faire des copies entre les mémoires. Ceci est dû au passage par copie des paramètres des fonctions (matrices, tables, maps, etc.). La fonction système *memmove* en est la preuve. De plus, la fonction la plus

consommatrice est celle qui lance l'algorithme EM (*lancerAlgoEM()*), appelée par le module 4, donc c'est celle-ci qui doit être optimisée en priorité, ce qui revient à améliorer le temps d'exécution des fonctions et les modules 2 et 3 appelées par celle-ci. En conclusion une réécriture des prototypes des fonctions prenant comme paramètres des références des objets et non pas leurs copies devrait permettre un gain de temps d'exécution d'au moins de 30% au module 4.

### 3.7.3 Profil de performance

Le profil de performance de notre implémentation (voir la figure 3.11) montre que le programme réalise des opérations de copie et de relocation de mémoires d'une façon excessive. Ceci est couplé à une mauvaise exploitation des mémoires caches causant une lenteur additionnelle liée aux nombreux défauts de caches déclenchés par les opérations de copie.

Cette analyse des modules nous a conduit à adopter, en vue de l'optimisation des codes sources des modules, les décisions suivantes :

- la création d'une librairie unique issue de la fusion des modules 2, 3 et 4, ce qui évitera l'inefficacité du passage par des fichiers dans l'enchaînement des modules.
- la réalisation d'un unique module exploitant la librairie créée, ce qui permettra l'atteinte d'un temps optimum.
- l'optimisation du module 1, traitant les fichiers PDB, sera réalisée après celles des autres modules.

Il est à noter que le module 1 et son optimisation ne seront pas décrits dans cette section car ils font l'objet de la section 2.4.3 du chapitre 2.

### 3.7.4 Corrections et optimisations

Nous avons cherché toutes les sources potentielles des opérations de copie indispensables. Ceci a nécessité une réécriture d'une partie du code source. En commençant par s'assurer que tous les paramètres de fonctions sont passés par référence au lieu de par valeur. En effet, le passage des paramètres par valeur augmente le temps d'exécution du module. Ce passage implique la création de deux objets temporaires lors de chaque appel de la fonction pour chacun de ses paramètres. Ce qui a conduit à choisir le passage des paramètres par références constantes au lieu du passage par valeurs. En conséquence un changement des signatures des fonctions a été effectué, en remplaçant tous les passages des paramètres par valeurs, par des passages par références constantes, ce qui induira la copie de la référence de l'objet passé en paramètre et pas de tout l'objet. Ci-dessous, le listing 3.1 présente un exemple de signature d'une fonction simple en C++ :

```
void LoadFromPdb3D(int NAtom, std::vector<double> coord3D);
```

**Listing 3.1** – Exemple simple d'un prototype d'une fonction en C++

Dans cette fonction (*LoadFromPdb3D*), les deux paramètres sont passés par copie, le paramètre 1 (*NAtom*) est un type de base (entier) ceci n'influence pas beaucoup les performances du module, mais cela risque d'être très gênant pour le paramètre 2 (*coord3D*) car une copie complète de ce vecteur sera créée à chaque appel de la fonction ce qui rend l'exécution lente. Alors pour optimiser cette fonction

on remplace la signature de celle-ci présentée par le listing 3.1, par la signature présentée par le listing 3.2 suivant :

```
void LoadFromPdb3D(int NAtom, std::vector<double> const & coord3D);
```

**Listing 3.2** – Exemple d’un prototype d’une fonction en C++ après optimisation

D’autre part, des modifications ont été apportées sur les pointeurs (comme les itérateurs dans les boucles là où l’ordre n’était pas vraiment important), ce qui a permis une réduction considérable du temps d’exécution.

Nous avons également profité de la logique de déplacement introduite dans le standard C++11 pour transférer les ressources détenues par des objets en fin de vie<sup>5</sup>. Aussi, nous avons changé les implémentations de certaines fonctions afin de maximiser la réutilisation des objets au lieu d’en créer des nouveaux. Ceci permet d’augmenter la localité spatiale de ces fonctions et donc d’optimiser l’utilisation des mémoires caches [109]. Dans le même souci, nous avons également regroupé des portions de calculs qui utilisent les mêmes données.

Le modèle mathématique, adopté par notre solution EM, utilise d’une façon très intensive le calcul algébrique et matriciel. Son implémentation est donc naturellement candidate à une optimisation par vectorisation. Cette optimisation a été déléguée au compilateur et a été activée par des options de compilation. L’activation de la vectorisation a eu des effets collatéraux positifs. Les unités de calculs vectoriels sont sollicitées par le compilateur pour faire des calculs scalaires habituellements exécutés sur les pipelines du microprocesseur [82]. Aussi les opérations de copies de mémoires utilisent les registres MMX (256/512bits) au lieu des registres généraux (32/64bits). Ceci a permis d’accélérer davantage toutes les opérations de copie et de relocation de mémoires.

Sur le plan algorithmique, nous avons réduit les opérations d’écritures systématiques dans les fichiers de sorties. Les écritures des fichiers sont désormais déclenchées à des événements bien précis (erreur, fin d’une étape, fin du programme, etc).

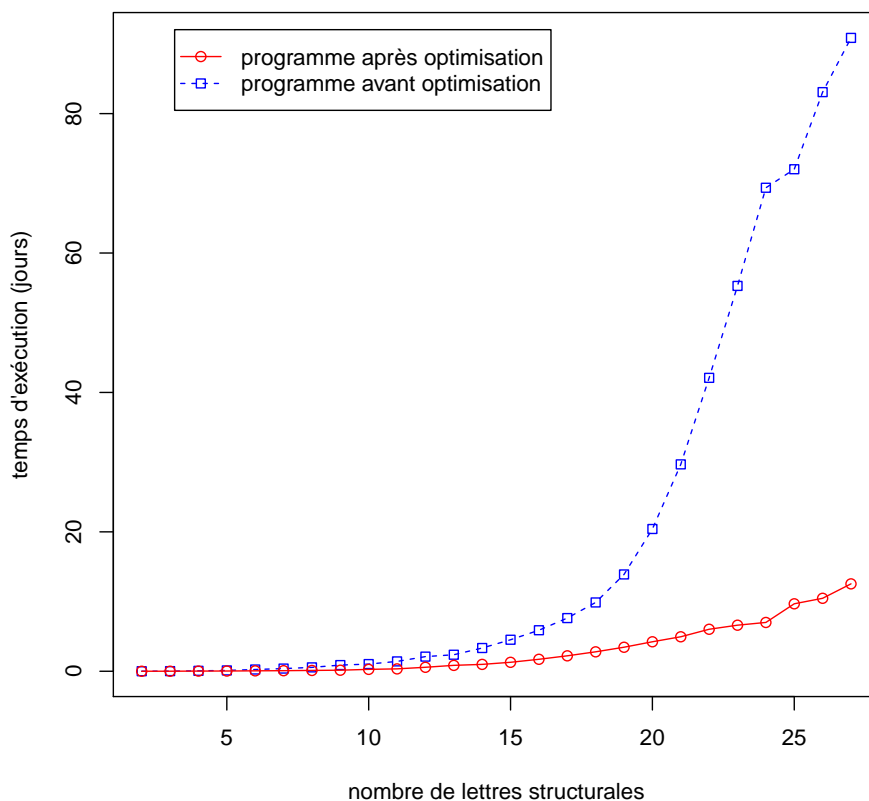
### 3.7.5 Résultats de l’optimisation

La méthodologie d’optimisation suivie a permis une réduction des copies qui se faisaient entre les registres du CPU à travers un *memmove* de moitié en passant les paramètres des fonctions par référence au lieu d’un passage par valeur. Néanmoins les transferts entre les registres du CPU concernant les retours des résultats des fonctions (*return*), qui nécessitent une copie vers la mémoire ne peuvent pas être réduits

Ce mode d’optimisation a permis au module 2 qui calcule les probabilités d’émission des chaînes, une exécution de 3,6 fois plus rapide que la version non optimisée. Il a donné lieu également à l’accélération du temps d’exécution du module 3 (chargé du calcul des log *forward*, *backward* et des log *max-forward* et *log max-backward*) de 3 fois, et celui du module 4 (chargé de la génération de l’alphabet structural) de 4,5 fois.

---

5. Il s’agit d’entités rvalues, des variables temporaires créées par le compilateur pour décomposer une expression complexe. Ou des variables locales créées par l’utilisateur qui atteignent leurs fins de vies (a la sortie d’une fonction par exemple). L’utilisateur peut explicitement déclarer qu’une variable arrive en fin de vie en utilisant la fonction `std::move()`.



**Figure 3.12 – Comparaison du temps d'exécution du module de génération de l'alphabet structural avant et après optimisation**

Pour une optimisation plus complète du module 4, il convient de réduire l'appel aux fonctions de lectures et écritures dans les fichiers, en effet les écritures dans les fichiers nécessitent 4 319 119 780 appels aux fonctions concernées, pour 100 fichiers PDB et pour une seule itération de l'algorithme EM, ce qui ralentit beaucoup les performances du module, donc l'échange par fichiers entre les modules 2, 3 et 4 n'était pas la solution optimale. Les tableaux de résultats de tests de performances après optimisation sont présentés dans l'annexe D (tableaux D.2, D.3 et D.4).

Ainsi, les modules 2, 3 et 4 ont été regroupés en un seul module (module générer AS). Par conséquent, ces transformations nous ont conduit donc à une accélération très importante de l'ordre de 20x. le binaire ainsi optimisé est 20 fois plus rapide que le binaire d'origine<sup>6</sup>. De plus ce nouveau module est structuré d'une manière qui facilitera sa parallélisation plus tard.

La figure 3.12 montre l'évolution du temps d'exécution du nouveau module en fonction du nombre de lettres structurales avant et après optimisation. Le temps nécessaire pour générer l'alphabet structural qui augmentait de façon exponentielle avant optimisation a été réduit d'une manière considérable.

6. Une seule itération de l'algorithme EM prenait 38 secondes pour traiter 100 fichiers PDB avant l'optimisation, alors qu'après l'optimisation elle est passée à 2,156 secondes

En effet, l'obtention d'un alphabet structural de 27 lettres structurales prenait 90 jours et 20 heures avant optimisation des modules (voir le tableau D.5 de l'annexe D), alors qu'après optimisation la génération d'un alphabet structural de même taille prend seulement 12 jours et 12 heures d'exécution du programme. Il est à noter que cet apprentissage de l'alphabet a été réalisé sur un jeu de 100 fichiers PDB seulement alors que notre but final était de générer un AS appris sur plusieurs milliers de fichiers PDB. Ce qui nécessite un temps important pour la génération de ce nouveau alphabet.

### 3.8 Le nouvel alphabet structural SAFlex

Nous proposons dans cette thèse une extension d'une approche basée sur l'alphabet structural (HMM-SA27) tenant compte de la redondance des protéines et des données manquantes. Nous avons donc développé un nouvel alphabet structural, appelé SAFlex, en utilisant le modèle de Markov caché développé antérieurement, contenant  $m = 27$  lettres structurales (LS) avec une faible variabilité géométrique [28, 30, 144].

Afin d'améliorer l'interprétation des 27 LS, nous proposons, pour SAFlex, une nouvelle nomenclature basée sur le type de structure secondaire. Nous avons donc assigné les lettres structurales de SAFlex selon qu'elles soient hélices ou brins ou boucles (identifiées à l'aide du logiciel STRIDE [69]). Ainsi, nous utilisons seulement trois lettres ('A', 'B', 'C') qui font référence à la classe de LS (*Helices* ou *Strands* ou *Coils*) respectivement comme indiqué dans le tableau 3.1. Chaque assignement de la lettre est suivi d'un nombre indiquant son occurrence dans le jeu de données après un apprentissage non supervisé. Par conséquent, les lettres structurales sont classées par ordre décroissant selon leurs fréquences, par exemple : la lettre «A1» est plus fréquente que la lettre «A2».

	États																										
	hélices				Boucles																		Brins				
SAFlex.V1	A1	A2	A3	A4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	B1	B2	B3	B4	B5
HMM-SA27	A	V	W	a	B	Z	P	K	Q	G	S	I	H	D	E	J	U	Y	F	C	R	O	M	L	N	X	T
Frequency %	12.6	5.6	5.3	2.6	4.7	4.5	4.4	4.1	4.1	3.4	3.2	2.9	2.7	2.0	2.0	2.0	2.0	2.0	1.9	1.8	1.7	1.5	5.3	5.1	4.9	4.7	3.0
n_eff_output	2.4	3.6	3.7	2.8	9.6	11.7	11.8	10.4	10.7	12.6	10.4	9.3	9.6	4.3	12.4	11.7	8.5	15.5	8.9	11.6	11.8	13.0	7.9	10.8	9.6	9.2	7.6
n_eff_input	2.3	4.5	3.4	3.0	9.0	12.4	13.3	10.5	12.2	17.5	8.3	7.1	9.7	9.6	11.2	8.8	12.5	10.8	13.5	6.2	12.4	9.8	8.0	10.8	7.0	10.4	7.7

**Table 3.1 – Tableau de correspondance entre les lettres structurales de HMM-SA27 et SAFlexV1.**

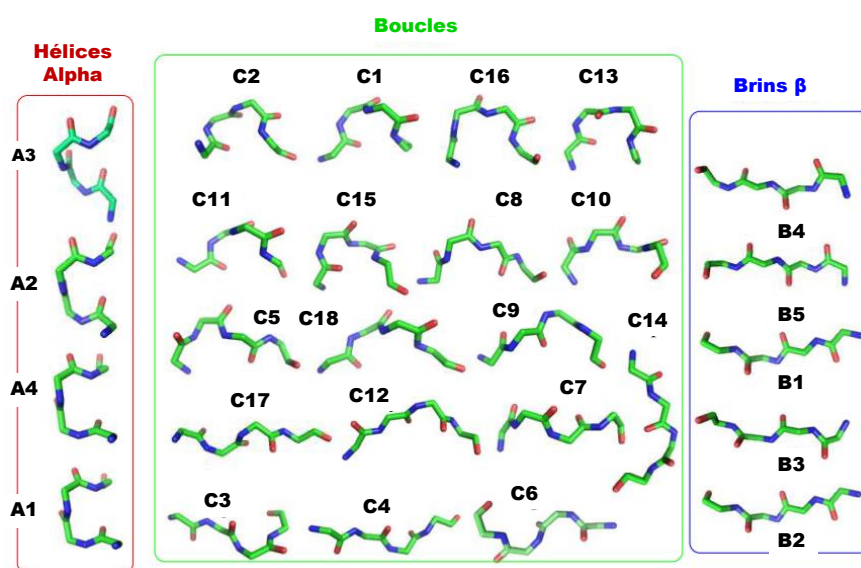
La nomenclature des lettres structurales a été mise à jour ainsi que les nombres effectifs correspondants aux LS possibles en entrée ou en sortie sont indiqués. La correspondance avec la nomenclature précédente de HMM-SA27 est également fournie dans le tableau 3.1. Nous fournissons aussi la matrice de transition (voir le tableau G.1) précédemment publiée et les paramètres d'émission des lettres structurales (voir le tableau G.2). Chaque lettre structurale correspond à une forme tridimensionnelle représentée par la figure 3.13.

Nous avons déjà expliqué dans la section 3.5, que notre modèle fournit trois différentes options d'encodage pour chaque chaîne :

- 1) **Le MAP** : le Maximum *a posteriori* offre la séquence des LS la plus probable en tenant compte de la dépendance complexe entre la structure des Lettres structurales et les probabilités de transition. Ce type d'encodage est souvent utilisé par les expérimentateurs pour l'analyse et la

comparaison de la structure.

- 2) **Le POST** : la distribution marginale *a posteriori* indique l'incertitude d'encodage. Il fournit la redistribution pondérée de la lettre structurale possible à chaque position donnée. Cette information est particulièrement utile pour les régions structurales où l'encodage est difficile et pourrait être plus représentative de la structure 3D que le MAP.
- 3) **Le NEFF** : le nombre effectif de lettres structurales à chaque position donnée résultant également de l'entropie de POST. Il permet une meilleure interprétation en fournissant donc une mesure pratique et commode pour la certitude de l'encodage. Le NEFF est aux environs de 1 lorsque l'encodage est très certain et peut atteindre 27 pour des positions totalement incertaines.



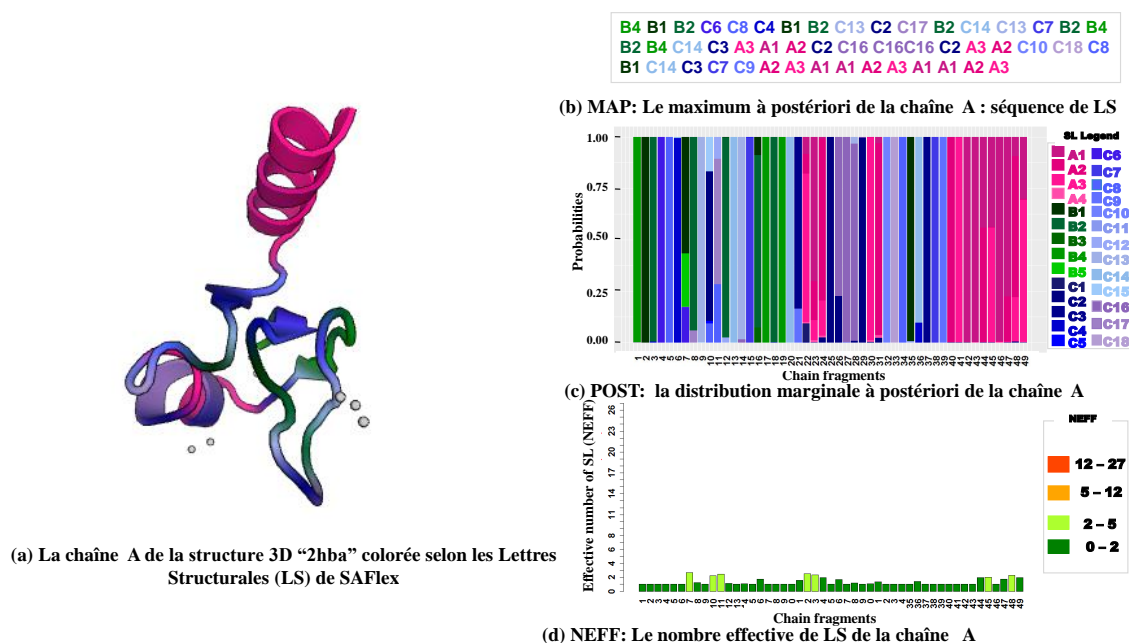
**Figure 3.13 – Les lettres de l'alphabet structural SAFlex.V1.**

Les LS sont groupées en fonction du type de structures secondaires en trois groupes : Hélices, boucles et feuilletts. Les hélices sont décrites par 4 LS différentes, les feuilletts sont décrits par 5 LS et 16 LS pour décrire les différents motifs des boucles.

La figure 3.14 montre les trois différents encodages de SAFlex pour la chaîne « A » du fichier PDB 2hba. Cette structure correspond au domaine N-terminal de la protéine ribosomale L9 (NTL9). Le repliement est mixte de type alpha-bêta et cette structure comporte 52 résidus. Elle a été largement utilisée comme un modèle pour les études expérimentales et computationnelles du repliement des protéines [38]. La structure 3D de la chaîne « A » de cette protéine est représentée à gauche de la figure 3.14. À droite sont représentés les différents encodages structuraux : MAP, POST et NEFF. L'encodage « MAP » correspond clairement à un mélange de alpha-bêta reliés avec peu de boucles dans la structure protéique. Dans l'encodage « POST », nous observons que la plupart des positions de l'encodage sont très sûres avec peu de positions incertaines. Cet encodage met en évidence donc trois régions d'incertitude relative aux fragments compris entre les positions : 7-11, 21-24 et la région finale 44-49 de la protéine. Par exemple, la distribution *a posteriori* du fragment 10 met en évidence



trois lettres de boucles possibles associées à ce fragment : C2 (prob = 0.726), C15 (prob = 0.167) et C8 (prob = 0.093). Enfin, l'option NEFF de la chaîne « A » fournit rapidement un aperçu de l'incertitude de l'encodage à chaque position (ex: NEFF = 2.244 pour le fragment 10), ce qui correspond à un nombre faible.



**Figure 3.14 – Les différentes options d'encodage structural de la structure protéique 2hba.**

À gauche, la structure de 2hba correspondant au domaine N-Terminal de la protéine ribosomale L9 (NTL9). À droite sont représentés les résultats de sortie de SAFlex: MAP, POST et NEFF.

### 3.8.1 Un encodage structural robuste aux données manquantes

La banque PDB contient de nombreuses structures ayant des régions manquantes. Ce genre de problème peut avoir une incidence importante s'il s'agit d'un manque de chaînes latérales, des régions de boucles entières ou un domaine entier. Ce problème a été détaillé dans la section 2.2 du chapitre 2. En effet, notre étude sur le jeu de données représentatif « PDBselect » [78] qui contient plus de 8 500 fichiers PDB a montré que dans trois quart des cas on est face à un problème de données manquantes, il peut s'agir des coordonnées des résidus complets manquants ou de ceux des atomes de  $C\alpha$ . La présence des coordonnées des atomes de  $C\alpha$ , permettant de se connecter au squelette peptidique, est une donnée nécessaire pour notre étude. En fait, l'absence de celles-ci pose un sérieux problème car elle empêche effectivement d'assigner la structure secondaire et d'avoir une idée du repliement 3D.

Ces parties manquantes se rapportent souvent aux extrémités de la protéine ou des boucles qui sont les régions les plus flexibles des protéines. Celles-ci sont impliquées dans les interactions entre protéines et leurs fonctions. Ainsi, la détection et la modélisation de ces données manquantes pourraient avoir un impact très intéressant pour l'analyse de la structure 3D des protéines.

Cependant, les alphabets structuraux existants et plusieurs études portant sur des structures 3D telles que les alignements locaux et multiples de structures n'ont pas pris en compte ces régions

manquantes. En effet, les méthodes d'encodage basées sur les alphabets structuraux reposent sur la numérotation séquentielle des atomes de la section *Coordinate* du fichier PDB<sup>7</sup> sans tenir compte des régions manquantes à l'intérieur de celui-ci bien que les résidus manquants y sont annotés.

N° Frag	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
CLE	R	R	K	G	N	G	C	E	D	D	E	C	F	A	J	O	G	C	E	C	E	D	C	B	L	D	E	A	K	L	D	D	
PB	Z	Z	G	O	I	A	C	D	D	D	D	F	K	O	P	A	C	C	D	D	D	D	F	B	D	C	F	K	B	C	C	D	
M32K25	W	Q	X	Q	F	E	I	D	I	D	D	K	V	P	R	I	A	I	E	G	I	M	N	G	B	K	W	N	G	I	E	I	F
SAFlex	C2	C7	C13	C7	B2	B4	B2	B3	B2	B3	C9	C1	C10	C7	B2	B3	B2	B4	B2	B2	C6	C8	B1	B5	C14	C16	C8	B2	B2	B4	B2	B2	
AA	A	Y	F	Q	G	L	P	V	E	V	R	G	S	N	G	A	F	Y	K	G	F	V	K	D	V	H	E	D	S	V	T	I	
N° Frag	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	
CLE	D	C	C	N	G	P	M	L	E	E	D	E	C	A	J	L	H	G	E	C	C	C	C	R				R	R	E	C	N	
PB	D	E	H	I	A	K	B	C	C	C	D	D	D	F	K	L	G	C	C	D	D	D	X	X			X	X	E	H	I	F	
M32K25	H	X	T	M	V	N	C	D	G	B	I	K	V	U	R	B	I	F	H	H	H	H	E	B			E	H	X	Q			
SAFlex	C14	C13	C5	C6	C2	C5	B1	B3	B1	B3	B2	C9	C1	C1	C5	B5	C4	C4	C14	C4	B3	B5	B5	B5	B5	B5	B1	B4	C14	C13	C7	B2	
AA	F	F	E	N	N	W	Q	S	E	R	Q	I	P	F	G	D	V	R	L	P	P	P	A	D	Y	N	K	E	I	T	E	G	
N° Frag	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	
CLE	A	C	E	D	E	E	E	D	R				R	R	C	E	C	E	C	E	D	E	P	L	E	E	Q	N	G	E	E	D	
PB	G	C	D	D	D	D	D	X					X	X	D	D	D	D	D	D	D	F	B	D	C	H	J	A	C	C	D	D	
M32K25	E	I	D	I	Q	G	S	N	X				H	E	H	A	F	E	G	I	M	N	B	B	H	Y	R	A	C	E	G	I	
SAFlex	B4	B2	B3	B2	B4	B2	C9	A4	A4	A4	C10	C7	C4	B4	C4	B3	C4	B4	B2	C3	C15	C15	C15	C15	C5	B5	B1	B4	B1	B2	C3	C1	
AA	D	E	V	E	V	Y	S	R	A	N	E	Q	E	P	C	G	W	W	L	A	R	V	R	M	M	K	G	D	F	Y	V	I	
N° Frag	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123						
CLE	D	C	A	J	R					R	R	D	E	D	A	J	J	L	E	C	C	C	P	G	F	C	R						
PB	D	F	K	X	X					X	X	D	D	D	F	K	L	P	C	C	D	D	F	B	D	C	Z	Z					
M32K25	J	U	H	I	B					I	E	I	K	V	U	N	B	H	H	H	L	R	D	H									
SAFlex	A1	A1	A1	A1	A1	A2	C5	B2	B4	B2	C9	C1	C1	C5	B5	C4	C14	C4	C3	C5	B2	C14	C9	A1	A1	A1	A1						
AA	E	Y	A	A	C	D	A	T	Y	N	E	I	V	T	L	E	R	L	R	P	V	N	P	N	P	L	A	T	K	G	S	F	

Figure 3.15 – Comparaison entre l'encodage des alphabets structuraux CLe, PBs, M32K25 et SAFlex.

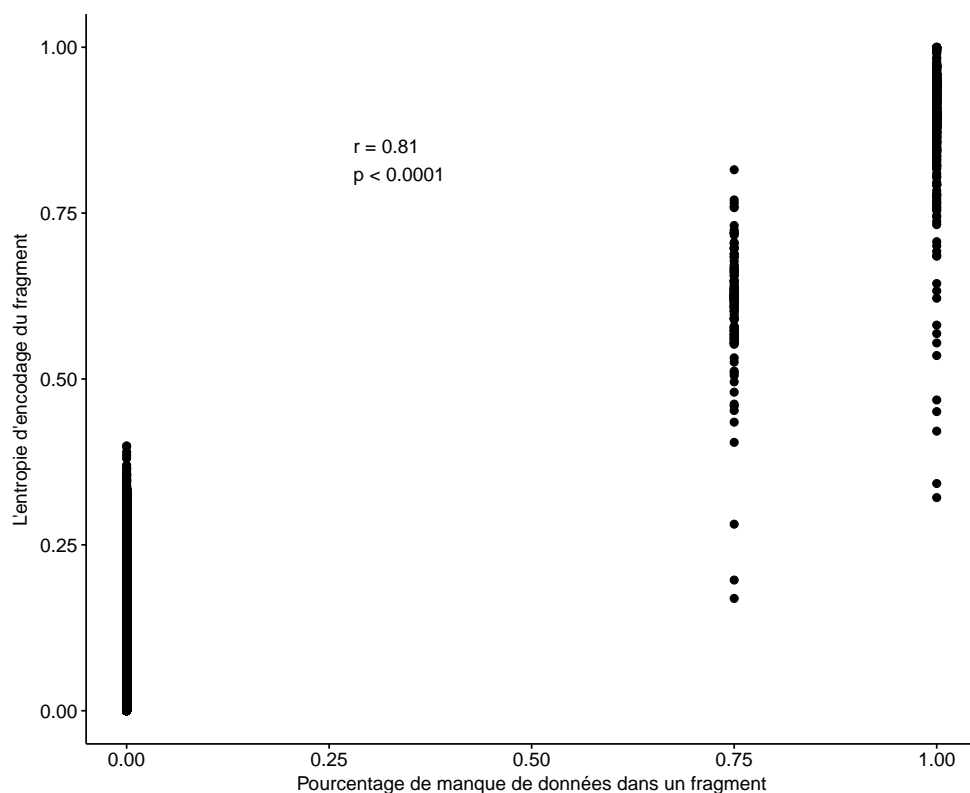
À titre d'exemple, la figure 3.15 montre un alignement des encodages structuraux de la protéine du syndrome du retardement mental chez l'humain issue du fichier PDB 3h8z en utilisant les alphabets structuraux : CLe [180], M32K25 [132], PBs [44] et SAFlex. La première ligne représente les numéros de fragments de la chaîne protéique. Ensuite, chaque ligne est associée à l'encodage structural avec un AS : (*Conformational Letters* (CLe), *Protein Blocks* (PBs), *Conformational Attractors* (M32K25) et notre AS SAFlex. Enfin la dernière ligne représente la séquence *fasta* des acides aminés (AA). Tous les encodages sont alignés en prenant comme référence la séquence des AA. Les régions non résolues de la séquence *fasta* sont considérées donc comme des régions manquantes. Celles-ci sont encadrées en rouge. Nous constatons alors trois régions manquantes au milieu de la séquence et une dernière à la fin de celle-ci.

Les trois alphabets (CLe, PB, M32K25) encodent la structure sans détecter les différentes régions manquantes (encadrées en rouge). Dans la séquence encodée, ils enchaînent toutes les lettres des régions résolues sans tenir compte de ce manque, ce qui donne un encodage faux de la protéine. Par contre SAFlex, détecte, identifie les régions manquantes non résolues et encode celles-ci. Les lettres de ces régions sont ainsi colorées en rouge orangé.

7. La section *Coordinate* indique aussi ses coordonnées x, y et z, son B facteur (*Temperature Factor*) et son *occupancy* facteur. Pour plus de détails voir [https://www.rcsb.org/pdb/static.do?p=file\\_formats/pdb/index.html](https://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html)

Nous avons déjà expliqué dans la section 3.5.2 que notre nouveau modèle prend rigoureusement en compte ces informations manquantes par des calculs probabilistes en fonction des descripteurs correspondants aux positions manquantes. Ainsi, lorsque les quatre descripteurs du fragment sont indisponibles, la position de celui-ci est totalement informelle (la vraisemblance locale pour toute lettre structurale est égale à 1,0), mais le contexte de la position est encore pris en compte grâce à la dépendance Markovienne du modèle HMM à travers la matrice de transition. Lorsque seulement quelques descripteurs sont manquants, la probabilité locale peut encore être calculée en utilisant les probabilités marginales d'émission gaussienne. Donc, cette information partielle peut être combinée avec le contexte local pour fournir des estimations fiables. Néanmoins, lorsque des motifs manquants répétés apparaissent dans une région donnée de la structure 3D, l'encodage résultant reste très incertain, ce qui est généralement souligné par des valeurs élevées de NEFF.

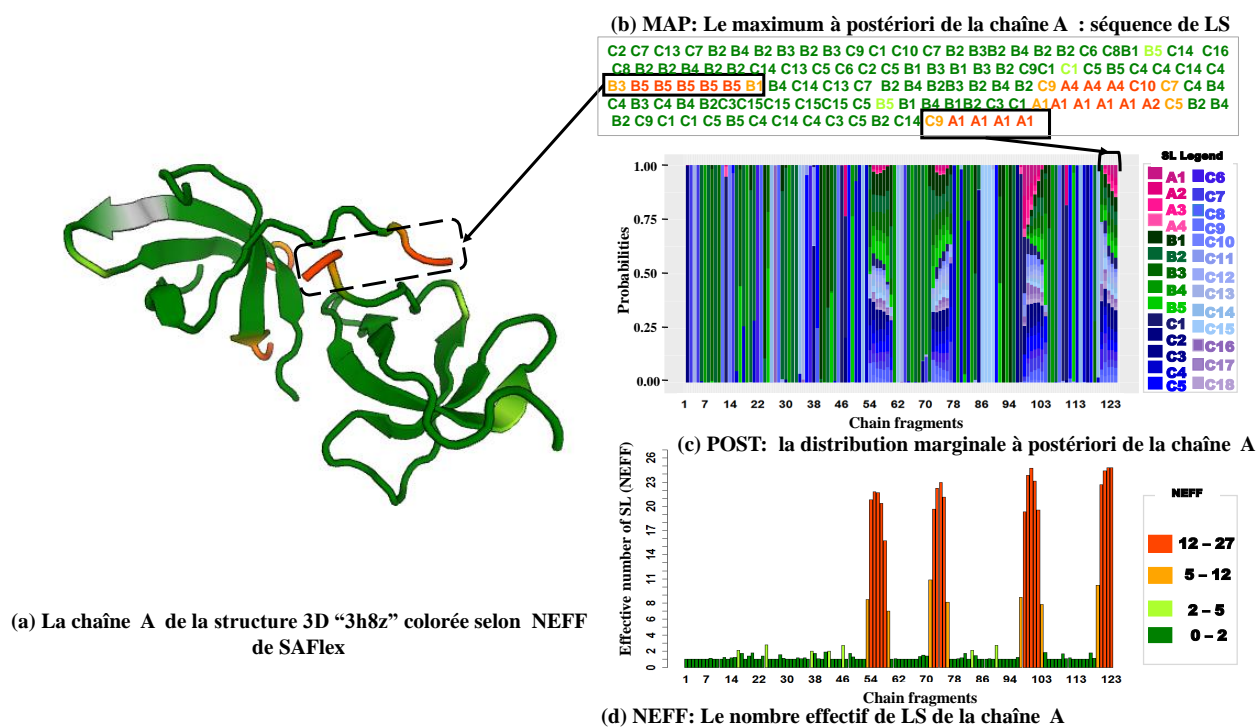
En conséquence, SAFlex prédit rigoureusement la séquence de la structure 3D complète. La méthode vise particulièrement à identifier les fragments manquants en exploitant les transitions HMM pour encoder leurs conformations. Cette identification et intégration de résidus manquants ne sont pas si simples. En effet, ils nécessitent un traitement du fichier de structures biologiques, tout en déterminant les positions des résidus manquants dans la chaîne protéique.



**Figure 3.16 – La corrélation entre les fragments structuraux manquants et l'entropie d'encodage.**

Par ailleurs, SAFlex fournit deux mesures d'incertitude d'encodage structural : l'entropie d'encodage et le NEFF. Celles-ci permettent aussi de détecter les régions manquantes d'une manière efficace. La figure 3.16 ci-dessus montre la corrélation entre l'entropie d'encodage et le pourcentage de don-

nées manquantes dans les fragments structuraux. Cette figure présente en abscisse le pourcentage de données manquantes au niveau des fragments et en ordonnée l'entropie d'encodage des fragments structuraux. Chaque fragment est caractérisé par ses quatre descripteurs et constitué de quatre résidus. L'absence d'au moins un des résidus du fragment est considérée comme des données manquantes, ce qui entraîne une indisponibilité de la valeur de descripteurs. Lorsque les coordonnées des quatre résidus du fragment sont présentes, le pourcentage de données manquantes de celui-ci est nul (tous les descripteurs ont donc des valeurs). Si un seul résidu du fragment est manquant, ceci implique l'indisponibilité des valeurs de trois descripteurs sur quatre, ce qui donne 75% de données manquantes. Donc, Il est tout à fait normal que nous n'ayons pas des pourcentages de données manquantes égaux à 25% et 50% car les coordonnées de chaque résidu du fragment servent pour calculer toujours trois descripteurs du même fragment. Lorsque deux résidus ou plus sont manquants au niveau du fragment, les valeurs des quatre descripteurs liés au fragment sont donc indisponibles impliquant ainsi un pourcentage de données manquantes égal à 1. Cette figure montre que les données manquantes dans les fragments et leurs entropies d'encodages sont corrélés positivement. En effet, lorsque le pourcentage de résidus manquants croît, l'entropie associée au fragment croît également. La valeur de corrélation obtenue est de 0,8, ce qui représente une corrélation très positive.



**Figure 3.17 – Détection des conformations des fragments manquants pour la structure de la PDB 3h8z.**

À gauche, la structure de 3h8z correspondant à la protéine associée au gène FXR2 relié au syndrome de retardement mental X fragile de l'espèce *homo sapiens*. À droite sont représentés les résultats de sortie de SAFlex: MAP, POST et NEFF.

À titre d'illustration, nous considérons le fichier PDB 3h8z correspondant à la protéine du syndrome du retardement mental de l'espèce *homo sapiens* associée au gène FXR2. Les protéines FMRP, FXR1

et FXR2 représentent une petite famille de protéines bien conservées. Elles sont importantes dans la régulation de la traduction, en particulier dans les cellules neuronales [1]. La figure 3.17 montre à gauche la structure de 3h8z et à droite les différents encodages de SAFlex : MAP, POST et NEFF. Cette structure 3D correspond à 123 fragments structuraux chevauchants colorés selon leurs valeurs de NEFF. Nous observons dans 3.17.(a) et 3.17.(b), qu’une grande partie de la protéine correspond à des lettres structurales de type bêta liées par des régions de LS de courtes boucles. Ces régions sont associées à de faibles incertitudes tel que montré par (3.17).(c) et 3.17.(d). Ceci approuve le fait que le domaine « Tud1 » forme une barre canonique de tudor qui présente cinq brins bêta antiparallèles très tordus.

Néanmoins, nous observons clairement la présence de quatre régions à forte incertitude (NEFF aux environs de 27). Ces quatre régions correspondent aux 16 résidus dont les coordonnées 3D sont manquantes dans le fichier PDB. Parmi celles-ci la première région (53-59) est prédite comme désordonnée en utilisant le site psipred [22, 91]. En dépit de cette forte incertitude, il faut noter que le MAP suggère des encodages pour ces quatre régions. Cependant, ces positions sont associées à un NEFF d’environ 27 qui correspondent à une entropie d’encodage proche de 1, ce qui suggère une grande incertitude de cet encodage. Ceci illustre davantage l’intérêt des encodages multiples de SAFlex.

### 3.8.2 Un encodage qui gère les chaînes multiples et détecte la variabilité entre les conformations des monomères d’une protéine

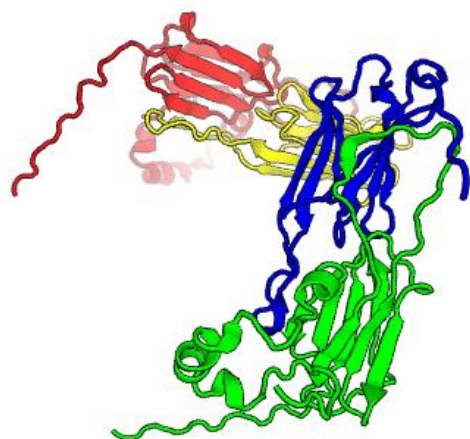
Durant les dernières décennies, de nombreuses structures complexes de protéines ont été déterminées contenant de multiples chaînes : homomères ou hétéromères ou différents fichiers PDB correspondant à une même protéine résolue dans différentes conditions. Si les hétéromères sont encodés en différentes séquences structurales, les homomères peuvent être considérés comme des répliques de la même structure sous-jacente. SAFlex propose d’encoder les homomères soit en tant que structures indépendantes (une séquence structurale par chaîne) soit en une seule séquence consensus de telle façon qu’une seule séquence structurale cachée est partagée par toutes les chaînes monomères. L’encodage consensus résultant représente donc la variabilité des monomères à travers les chaînes, cette variabilité étant due à l’incertitude de mesure ou à la flexibilité intrinsèque.

À titre illustratif, la présente étude est basée sur les petites protéines de chocs thermiques (sHSPs), qui jouent un rôle essentiel dans la prise en charge des protéines abimées pour qu’elles ne forment pas d’agrégats. Ainsi, elles participent à la protection, au maintien et à la régulation des fonctions spécifiques de la protéine [138]. Le fichier PDB 1gme comprend HSP16.9, un membre des petites protéines de choc thermique (les sHSPs), qui s’assemble en un double disque dodecamère<sup>8</sup>. La structure 3D de ce fichier montrée par la figure 3.18.(a) contient uniquement un tétramère<sup>9</sup> qui permet la reconstruction du dodecamère par des opérations de symétrie [178]. Les quatre monomères de 1gme ont une structure commune très globale, appelée *alpha crystallin domain signature* [138] avec des différences dans certaines régions.

---

8. Un dodecamère est une structure quaternaire composée de 12 sous-unités protéiques dans un complexe

9. Un tétramère est un complexe protéique constitué de quatre sous-unités



(a) La structure 1gme comportant 4 chaînes protéiques A,B,C et D

1:A	C7	B2	C6	C8	C14	C4	B3	B2	B3	C3	A2	C7	C9	C11	C1	C11	C16	C6	A3	A1	A1	A1	A1	C2	C16	C11	A1	A2	C11	C1	C5	C12	C12	B1	C3	C11	A1	A1	C1	39:A	
1:B	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	39:B
1:C	C7	B2	C6	C8	C4	C4	B5	B2	B5	C3	C2	C7	C9	C1	C1	C11	C16	C6	A3	A1	A1	A2	A3	C2	C2	C11	C1	C2	C18	C2	C16	C5	C4	B3	C3	C11	A1	A1	C1	39:C	
1:D	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	A4	C10	39:D	
1:Cons	C4	B2	C6	C8	C14	C4	B5	B2	B5	C3	C2	C7	C9	C11	C1	C11	C16	C6	A3	A1	A1	A2	A3	C2	C2	C11	C1	C2	C15	C15	C15	C15	C17	B3	C3	C11	A1	A1	C1	39:Cons	
40:A	C1	C1	C16	C6	C17	B1	B1	B5	B1	B5	C9	C2	C8	C4	B3	B5	B5	B5	C3	C6	B3	C13	C7	B2	C9	C1	C2	C5	B5	B1	B4	B1	B3	C13	C15	C2	C5	B1	B3	78:A	
40:B	A4	C10	C7	B2	B3	B1	B1	B4	B1	B5	C9	C2	C8	C4	B3	B5	B5	B5	B2	C6	C17	C13	C7	B2	C9	C1	A2	C5	B5	B1	B4	B2	B3	C13	C15	C16	C5	B1	B3	78:B	
40:C	C1	C1	C16	C6	B4	B1	B1	B5	B1	B5	C9	C2	C8	C4	B3	B5	B5	B5	B2	C6	C17	C13	C7	B3	C9	C1	A2	C5	B5	B1	B4	B2	B5	C13	C15	C2	C5	B1	B3	78:C	
40:D	C7	B3	B1	B2	B3	B1	B1	B4	B1	B5	C9	C2	C8	C4	B3	B5	B5	B5	B2	C6	C17	C13	C7	B2	C9	C1	A2	C5	B5	B1	B4	B1	B3	C13	C15	C16	C5	B1	B3	78:D	
40:Cons	C1	C11	C15	C17	B4	B1	B1	B4	B1	B5	C9	C2	C8	C4	B3	B5	B5	B5	B2	C6	B3	C13	C7	B3	C9	C1	A2	C5	B5	B1	B4	B2	B3	C13	C15	C2	C5	B1	B3	78:Cons	
79:A	B5	B4	B4	B2	B2	B2	C4	C4	C3	C7	C12	C3	C2	C7	C4	B2	C6	C8	B5	C15	C11	C7	B4	B4	C12	B4	B1	B5	B5	B5	B2	B2	C3	C8	C14	C17	C3	C3	C1	A2	118:A
79:B	B5	B4	C12	C12	C3	C17	B2	C3	C6	C17	C12	C9	C11	C5	B1	B1	C6	C8	B5	C15	C5	B2	B5	C12	C12	B5	B5	B5	B5	B2	B3	C14	C9	C2	C7	C9	C6	A2	C2	118:B	
79:C	B5	B4	B4	B2	B2	C4	C4	C4	C14	B3	B5	C3	A2	C7	B3	B1	C6	C8	B1	C15	C5	B4	B5	B4	C14	B4	B1	B5	B5	B5	B2	B2	C9	C8	C14	C4	C3	C6	A3	A2	118:C
79:D	B1	B4	C12	C12	B2	B3	B2	C4	C4	C4	C3	C15	C11	C5	B3	B2	C6	C8	B5	C15	C5	B3	B5	C12	C12	B5	B5	B5	B5	B2	B2	C4	C9	C1	C5	C3	C6	C1	C2	118:D	
79:Cons	B5	B4	C12	C12	C3	C17	C15	C15	C15	C17	C15	C15	C15	C15	C17	B1	C6	C8	B5	C15	C15	C17	B5	C12	C12	B4	B5	B5	B5	B2	B2	C15	C15	C15	C15	C15	C6	C1	C2	118:Cons	
119:A	C5	B3	B5	B5	B1	B4	C13	C15	C5	B1	B3	B5	B5	B5	B2	B3	C4	C4	B4	C3	C15	C10	C17	C4	B2	B3	B3	B2	B2	B3										148:A	
119:B	C7	B3	B5	B5	B1	B4	C13	C15	C5	B1	B3	B5	B1	B1	B2	B2	C9	C8	C4	B4	B2	C4	C4	B4	B1	B3	B3	C4	B3	C12										148:B	
119:C	C5	B3	B5	B5	B1	B4	C13	C15	C5	B1	B3	B5	B5	B1	B2	B2	C4	B1	C9	C17	C9	C17	C4	B2	B2	B3	B2	B2	B3											148:C	
119:D	C7	B3	B5	B5	B1	B4	C13	C15	C5	B1	B3	B5	B5	B5	B2	B2	C9	C8	C4	B4	B2	C4	C9	C7	B1	B2	B1	B2	B2	C3										148:D	
119:Cons	C7	B3	B5	B5	B1	B4	C13	C15	C5	B1	B3	B5	B5	B1	B2	B2	C15	C15	C15	C15	C15	C15	C15	C17	B1	B2	B1	B2	B2	C15										148:Cons	

(b) Alignement multiple de l'encodage structural (MAP) des 4 chaînes protéiques : A, B C et D avec l'encodage de la chaîne consensus.

Figure 3.18 – Les encodages structuraux associés aux quatre monomères pour le fichier PDB 1gme.

Un monomère de ce tétramère est de longueur 151 ce qui correspond à 148 fragments structuraux chevauchants. Cependant, tous les monomères de 1gme ont moins de 151 résidus parce qu'il leur manque des résidus. En effet, les 42 résidus N-terminaux sont manquants dans les deux monomères B et D, tandis que les N-terminaux dans les monomères A et C sont entièrement résolus (sauf pour le premier résidu) et composés d'hélices liées à des boucles [178].

La figure 3.19 illustre les valeurs de NEFF pour les quatre chaînes et pour l'encodage de la chaîne consensus (encadrée en rouge). Les deux régions manquantes des chaînes B et D sont clairement mises en évidence (NEFF au environ de 27) et l'incertitude globale d'encodage le long des quatre chaînes est assez grande avec une valeur moyenne de NEFF  $\simeq 4.4$  et de NEFF  $\simeq 1.3$  en excluant les régions manquantes. Quand à l'encodage consensus, il a une incertitude beaucoup plus faible avec un NEFF  $\simeq 1.1$ . Ceci illustre l'intérêt de l'approche consensus qui permet non seulement de suggérer un encodage pour la protéine complète malgré les motifs manquants des chaînes B et D, mais aussi de profiter des répliquations (redondance) pour affiner l'encodage structural.

La comparaison de l'encodage structural des conformations des quatre monomères du complexe, à travers l'alignement multiple du MAP, révèle une asymétrie dans le tétramère HSP16.9, pourtant il s'agit de la même séquence d'acides aminés pour les quatre monomères étudiés de la protéine. La figure 3.18.(b) montre l'alignement multiple du MAP pour les quatre chaînes et indique les régions manquantes dans les chaînes B et D. Celles-ci apparaissent clairement dans l'encodage MAP avec de longues séries de LS « A4 »; ce qui est normal en l'absence de toute information supplémentaire puisque cette LS a la plus forte probabilité de bouclage (voir la matrice de transition G.1 à la page 138).

Cependant, les chaînes A et C fournissent des MAP informatifs pour les positions manquantes correspondantes et le résultat est clairement conforme à la MAP du consensus. Pour la plupart des positions restantes (les fragments non manquants), nous observons un fort consensus entre tous les encodages. Ce qui reflète ainsi la faible variabilité des structures. Il est toutefois intéressant de noter que dans certaines régions, il existe beaucoup de variabilité dans leurs encodages MAP des quatre chaînes, indiquée par des LS différentes. En effet, la mesure du RMSD confirme cette variabilité entre les monomères (voir le tableau 3.2). D'après ce tableau, les RMSD calculés sont tous inférieurs à 1 Å, ils varient entre 0,39 et 0,81 Å. Les quatre monomères ont des structures globales similaires avec des différences dans certaines régions.

	Chain A	Chain B	Chain C	Chain D
Chain A	0.00	0.81	0.42	0.75
Chain B	0.81	0.00	0.77	0.39
Chain C	0.42	0.77	0.00	0.74
Chain D	0.75	0.39	0.74	0.00

**Table 3.2** – Le RMSD (Root mean square deviations) entre les coordonnées des monomères de 1GME.

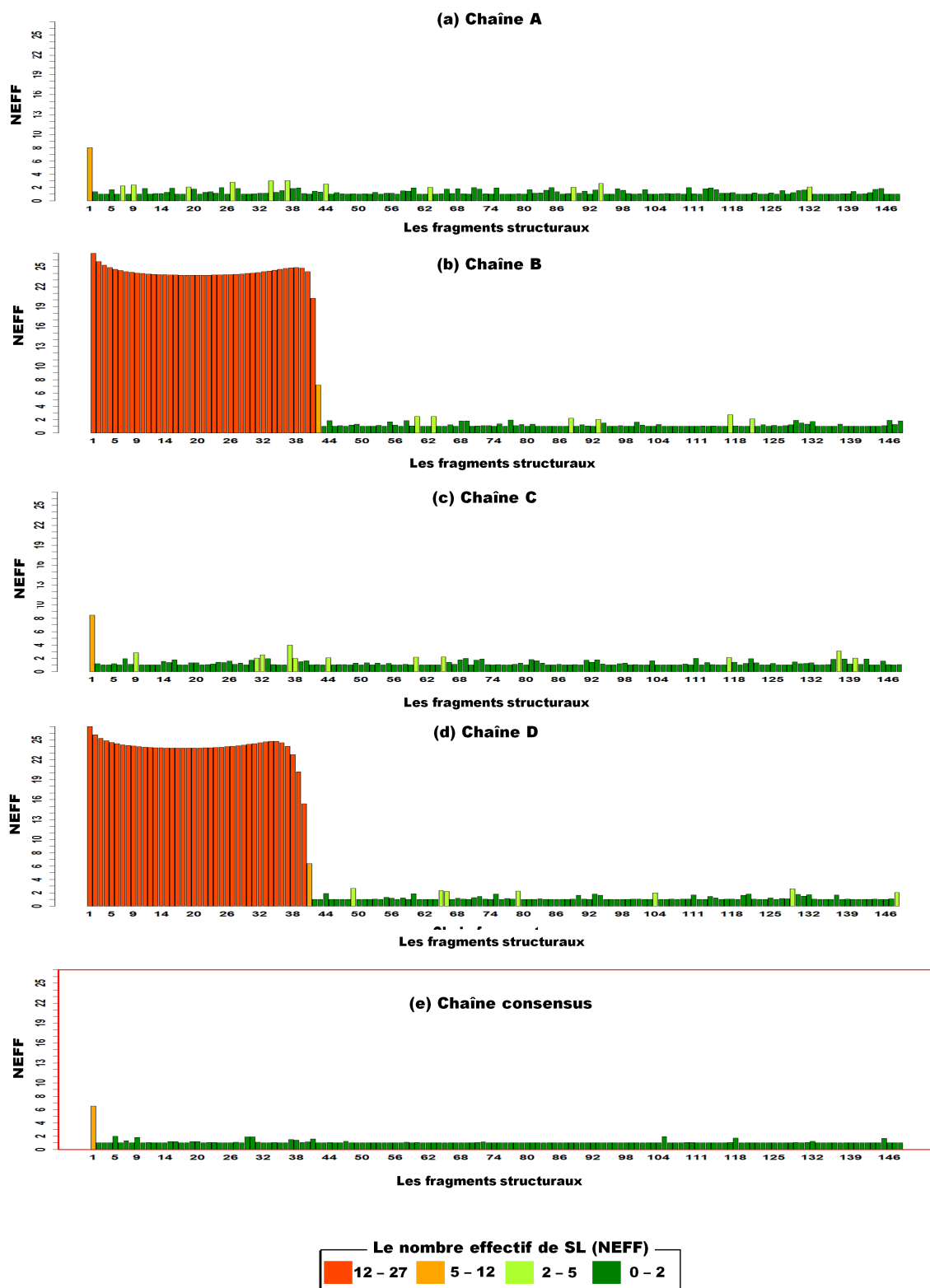
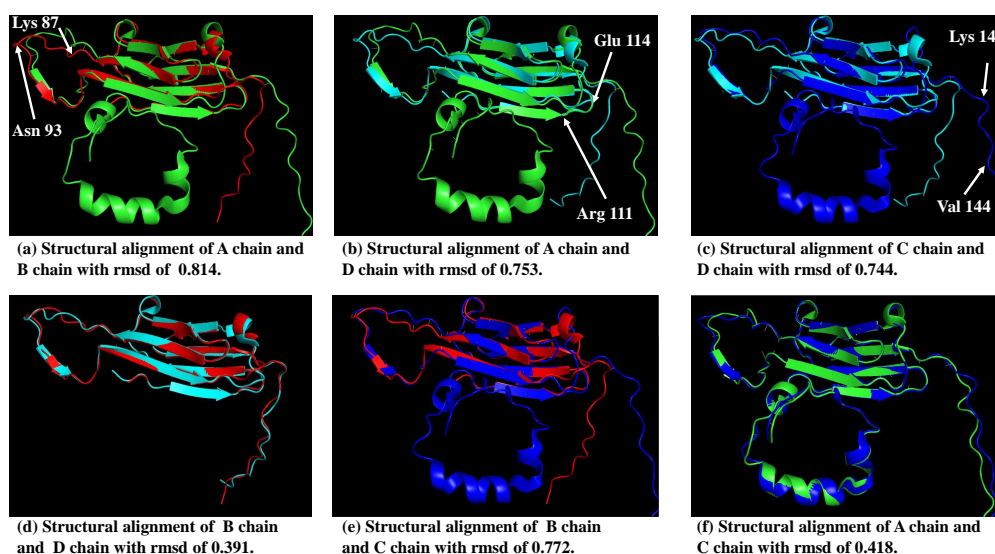


Figure 3.19 – Les valeurs (NEFF) associées aux quatre monomères avec la chaîne consensus générée par SAFlex pour le fichier PDB 1gme. La figure montre les valeurs de NEFF pour l’encodage des quatre chaînes et la chaîne consensus. L’encodage de la chaîne consensus est encadré en rouge.



Aussi pour identifier les différences dans ces structures, il est nécessaire de superposer l'alignement par paire en utilisant l'outil Pymol [157, 158]. La figure 3.20 montre les alignements de structures des quatre monomères du tétramère HSP16.9. Les différences conformationnelles sont apparentes au niveau de plusieurs zones de ces monomères comme le montre la figure 3.20 ci-dessous. Il convient de noter d'abord que la structure des monomères A et C commence par un acide aminé Ser2 tandis que celle des monomères B et D commence respectivement par des acides aminés Asn 43 et Ala 42. Par conséquent, le premier acide aminé est manquant pour A et C et les acides aminés de 1 à 42 et de 1 à 41 sont manquants aussi pour B et D. Dans ce cas, des hélices liées aléatoirement à des boucles peuvent être perdues dans les chaînes B et D.

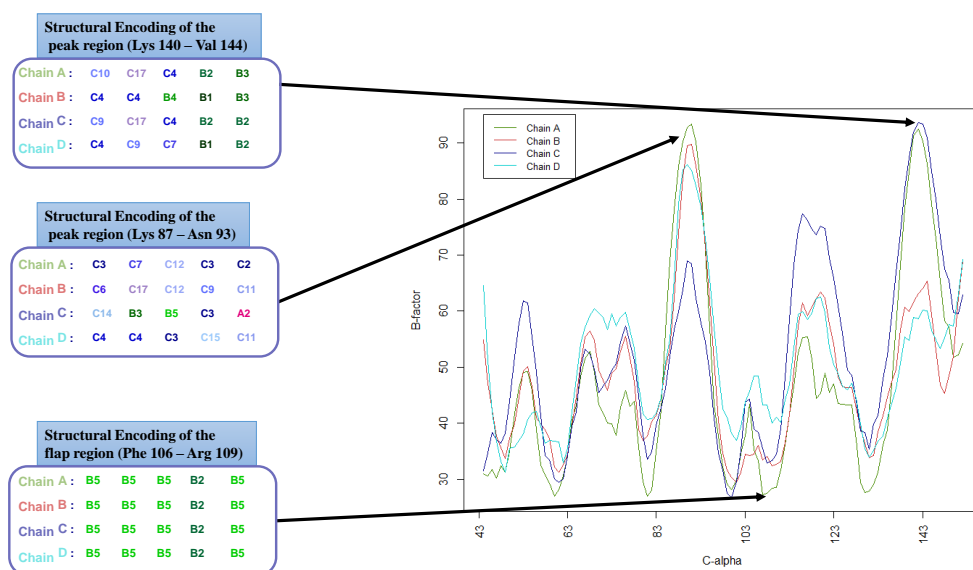


**Figure 3.20 – Comparaisons entre les quatre monomères du PDB 1gme.** Les chaînes protéiques sont en style cartoon, avec la chaîne A en vert, la chaîne B en rouge, la chaîne C en bleu et la chaîne D en cyan. les quatres conformations sont superposées

Les alignements de structures des quatre monomères montrent clairement que les conformations des chaînes A, B, C et D sont proches mais pas identiques. Ceci explique la grande ressemblance entre leurs encodages (encodage de A et C et celui de B et D). Par contre les boucles constituées par les résidus (Lys 87 - Asn 93, Arg 111 - Glu 114 and Lys 139 - Val 151) varient énormément géométriquement. Ceci suggère que certaines des positions de ces régions pourraient correspondre à des positions de chaînes flexibles intrinsèques ou à des incertitudes de résolution. En effet, cette variabilité entre les monomères se manifeste également par la fluctuation de la valeur du facteur de température (*B factor*)<sup>10</sup> pour les atomes de C $\alpha$  des quatre monomères, comme le montre la figure 3.21. Il convient surtout de noter que les courbes n'ont pas été tracées pour les parties manquantes des résidus de 1 à 42 pour tous les monomères. Les résidus des fragments des régions (Phe 57 -Leu 61, Leu 79 - Val 81, Arg 98 -Glu 100, Phe 106 -Arg 109 et Gly 128 - Thr 133) ont les valeurs les plus faibles du facteur de température, ce qui suggère une légère différence conformationnelle de ces régions. Ces régions ont des conformations rigides pour toutes les chaînes. Cela est évident car elles coïncident avec les brins bêta de HSP16.9. Par

10. Ce facteur indique la mobilité dynamique des atomes d'une molécule

contre, la plus grande partie des différences conformationnelles apparait au niveau des résidus (Lys 87 - Asn93 , Arg111-Glu 114 et Lys 139 - Val 151), les deux premières régions coïncident avec des régions très flexibles tandis que les résidus de la dernière région sont flexibles parce qu'ils correspondent à l'extension N-terminal de tous les monomères. Dans ce contexte, l'encodage consensus tente de trouver la lettre structurale commune la plus adéquate pour refléter cette variabilité et sélectionne la lettre structurale C15 (ancien F dans HMM-SA27) qui correspond à l'état de boucle flou de l'alphabet [144]. Dans ce cas, les encodages indépendants de la chaîne peuvent être explorés avec soin pour détecter les positions variables et les changements potentiellement de LS et ce en raison de la flexibilité intrinsèque, de la liaison avec un partenaire ou des effets de mutation [143].



**Figure 3.21 – Correspondance entre le B-facteur et les variations de l'encodage structural pour les quatre monomères de HSP.** Les valeurs du facteur de température (B facteur) des  $C\alpha$  ( $\text{\AA}^2$ ) pour les quatre monomères HSP du fichier PDB 1GME. La chaîne A en vert, la chaîne B en rouge, la chaîne C en bleu et la chaîne D en cyan.

Par ailleurs, la prédiction de la séquence protéique HSP16.9 par l'outil *PsiPred* [22, 91] indique que les régions manquantes sont des régions désordonnées et prédites en tant que boucles (voir Figure 3.22.a). Ceci est évident parce que ces régions n'ont pas pu être résolues d'un point de vue structural. Celles-ci sont représentées par SAFlex, comme des régions fortement ambiguës en terme de lettres structurales associées à une distribution marginale *a posteriori* très élevée (valeur proche de 1). Donc il y a une sorte de connexion entre toutes ces informations qui convergent toutes dans le même sens. Il est important de noter que, *Psipred* utilise les informations de séquences en se basant sur les acides aminés pour prédire les conformations associées à une structure. Cependant SAFlex n'utilise pas l'information de séquence pour prédire les conformations des fragments structuraux et pourtant nous arrivons à des résultats cohérents avec les chaînes résolues entièrement.

En conclusion, un NEFF associé à la chaîne consensus aurait plusieurs interprétations. Un NEFF plus élevé peut être expliqué par plusieurs facteurs :

- la possibilité d'être dans une région dite désordonnée,

- la possibilité d’être dans un région dite flexible,
- Enfin la possibilité d’être dans une région manquante.

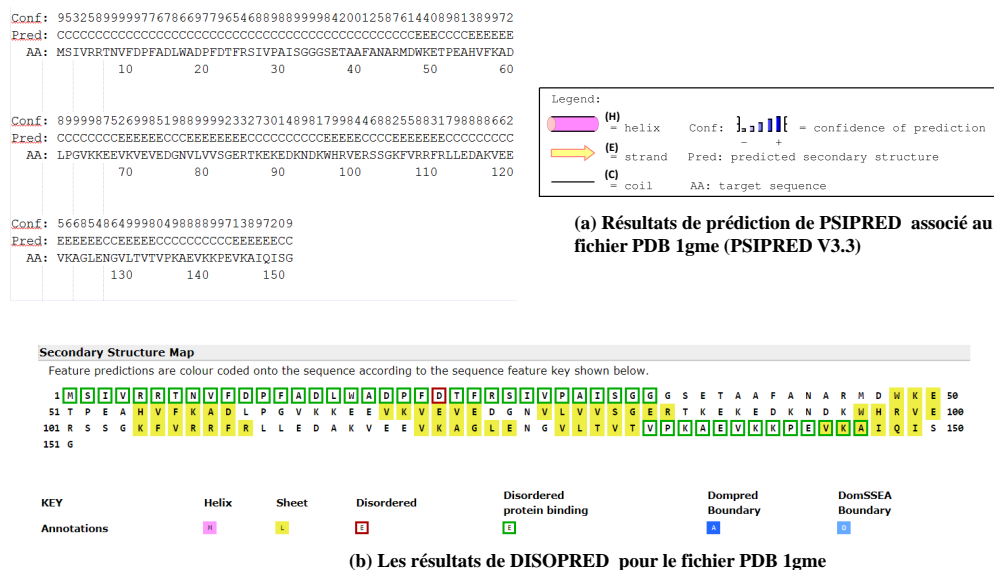


Figure 3.22 – Prédiction de la protéine HSP16.9 du fichier PDB 1gme en utilisant *PsiPred* et *DISOPRED*.

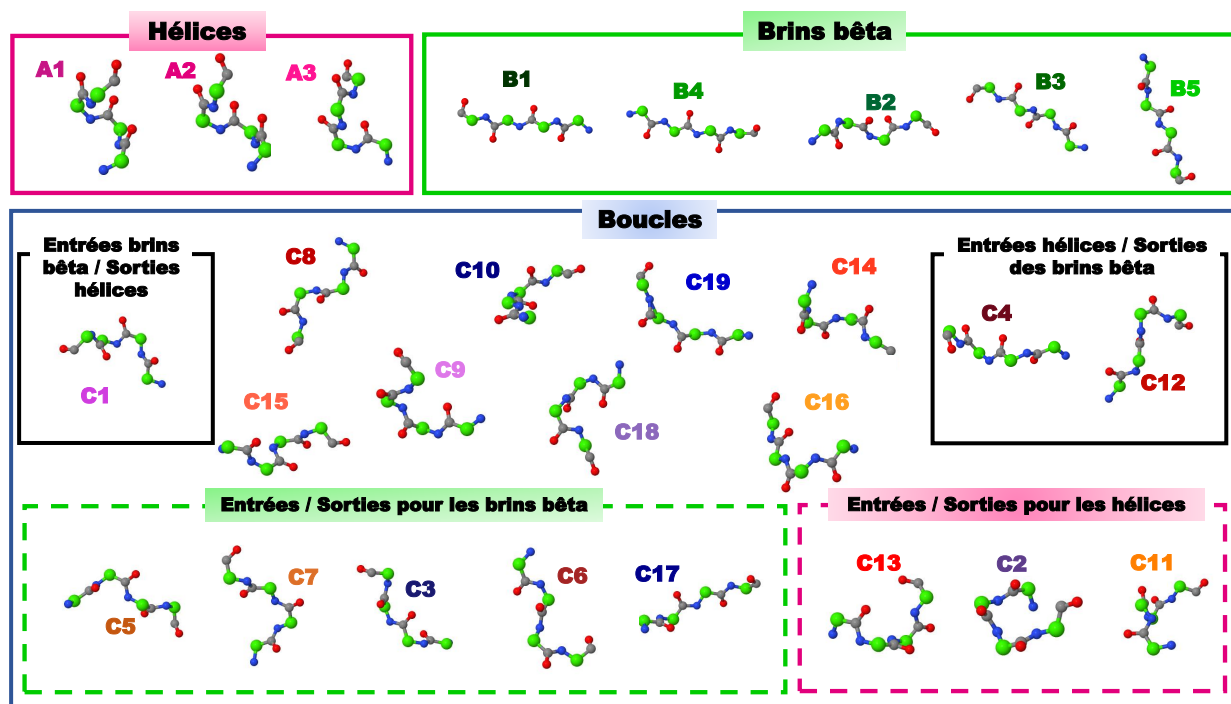
Cette étude montre donc une cohérence entre des régions que nous n’arrivons pas à prédire hélice, brin et boucle (qui sont aussi prédites comme désordonnées par *DISOPRED* [22, 91]. Ces régions sont signalées aussi par SAFlex par des lettres incertaines associées à une entropie et un NEFF élevés. Donc, l’encodage structural des conformations des monomères de SAFlex peut éventuellement indiquer un changement au niveau du B facteur .

D’une autre façon, notre étude propose le NEFF comme degré de confiance associé à chaque position de l’encodage structural avec une meilleure précision sur les incertitudes d’encodages indiquées par le POST. En outre le MAP permet un encodage d’une structure 3D en fournissant la séquence la plus probable. Celle-ci est utile pour l’analyse et la comparaison des structures. Cette étude a montré sur un exemple la détection de la variabilité entre les monomères, mais la complexité de ce travail nécessite des analyses supplémentaires sur d’autres exemples de protéines.

### 3.9 Preuve de concept : Développement de la version SAFlex. $\beta$

Afin de montrer que les améliorations méthodologiques proposées dans les sections précédentes sont faisables nous avons commencé à réestimer un nouvel alphabet avec ces améliorations. La version SAFlex. $\beta$  obtenue après estimation présentée dans ce chapitre est une preuve de concept que la démarche suivie est innovante et fonctionne bien puisque nous avons commencé de le faire sur un jeu de 100 PDB. L'alphabet structural obtenu et ses fréquences sont cohérents, ceci est dû à l'utilisation des chaînes de Markov cachées (HMM) car celles-ci permettent d'éviter les états quasi vides (occurrences faibles), ce qui présente un avantage par rapport aux autres approches utilisées pour développer un alphabet structural.

Pour une meilleure précision d'assignation des lettres structurales, nous avons eu recours à l'analyse des descripteurs de chaque LS ainsi qu'à l'exploitation des probabilités de transition entre ces lettres structurales nous avons donc réorganisé la matrice de transition (voir le tableau G.4) en fonction du descripteur  $X^2$  en classant du plus petit au plus grand. Dans les vingt-sept lettres celles qui ont un  $X^2$  le moins court représentent des hélices, celles qui ont un  $X^2$  le plus grand désignent les feuillets<sup>11</sup>. Nous arrivons d'après notre analyse à trois classes de bouts d'hélices, cinq classes de bouts de brins bêta et 19 classes de bouts de boucles (voir le tableau G.3). La figure 3.23 montre les 27 représentants moyens de ces lettres structurales de la version SAFlex. $\beta$  obtenus.



**Figure 3.23 – Les lettres structurales de la version SAFlex. $\beta$**

Les lettres structurales sont groupées en fonction du type de structure secondaires en trois grands groupes : hélices, feuillets et boucles. Les hélices sont décrites par 3 LS différentes, les feuillets sont décrits par 5 LS. En ce qui concerne les boucles décrites par 19 LS, elles sont classées en plusieurs sous groupes : des boucles spécifiques hélices, des boucles spécifiques feuillets et d'autres spécifiques pour hélices et feuillets ensemble et enfin des boucles non spécifiques.

L'analyse de la matrice de transition entre les lettres structurales de SAFlex, montre qu'il y a une forte dépendance entre les états. La matrice de transition entre les lettres structurales est très importante pour compléter nos informations sur la cohérence de l'alphabet structural obtenu. Par souci de lisibilité de cette matrice, nous avons donc remplacer par des « . » les transitions de très faible probabilité (typiquement  $< 10^{-10}$ ), et organiser les lettres par classes : A1, ..., A3, B1, ..., B5, C1, ..., C19. Ce sont bien les transitions qui nous guideront pour décider si celles-ci sont des lettres de sorties pures, ou encore intra-structure. Par ailleurs la matrice de transition ne nous fournit que les lettres de sorties de chaque lettre structurale mais pas les lettres de provenances vers une lettre.

11.  $X_i^2$  désigne la longueur globale de celui-ci, plus le fragment est étendu, plus sa valeur de  $X_i^2$  est grande et vice versa

Ainsi pour identifier ces lettres de provenance de chaque lettre structurale, il faut calculer la matrice de provenance des lettres ( $\pi^{\text{prov}}$ ) à partir de la matrice de transition. Pour ce faire j'ai calculé le vecteur propre ( $\nu$ ) associé à la valeur « 1 » qui nous permet d'avoir une loi stationnaire de la matrice de transition. Puis j'ai fait appel à la formule suivante :

$$\pi^{\text{prov}}(k, j) = \frac{\pi(j, k) \times \nu(j)}{\nu(k)} \quad (3.14)$$

Avec :

$\pi \in \mathbb{R}^{m \times m}$  est la matrice de transition entre les lettres structurales.

$\pi(j, k)$  est la probabilité de transition de la lettre structurale  $j$  vers la lettre  $k$ .

$\pi^{\text{prov}} \in \mathbb{R}^{m \times m}$  : représente la matrice de provenance des lettres structurales.

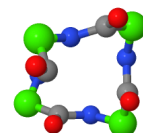
$\pi(k, j)^{\text{prov}}$  est la probabilité de provenance de la lettre structurale  $k$  vers la lettre structurale  $j$ .

$\nu \in \mathbb{R}^m$  est le vecteur propre de la matrice de transition  $\pi$ .

L'analyse des deux matrices de transition et de provenance (voir le tableau G.6) nous a permis d'identifier les entrées et les sorties des lettres structurales de notre alphabet. Le tableau (G.5) récapitule ces informations. Nous constatons d'après notre analyse des lettres structurales que :

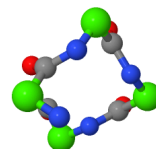
#### pour les états hélices :

**A1** : cette lettre est la plus fréquente parmi toutes les lettres structurales avec 17%. Les acides aminés composant celle-ci sont organisés dans une structure hélicoïdale. Elle a l'air d'être un pas d'une structure hélice  $\alpha$  puisque vue de dessus elle a une forme carrée (voir la figure 3.24). Cette lettre ne communique presque pas avec les autres lettres des autres classes (B et C) car ses sorties sont de structure hélice **A2** et **A3** bienque sa sortie principale est la lettre **A2** avec 12%. Ceux-ci permet d'affirmer que **A1** représente un motif de cœur d'hélice.



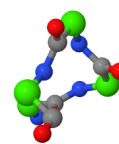
**Figure 3.24** – A1 vue de dessus

**A2** : cette lettre est la deuxième plus fréquente après **A1** de toutes les lettres structurales avec 12%. Elle représente un bord d'hélice étendu, elle peut être une entrée ou une sortie pour des structures secondaires de type hélice  $\alpha$ . Les acides aminés composant celle-ci sont organisés aussi dans une structure hélicoïdale. Elle a l'air d'être aussi un pas entrant ou sortant d'une structure hélice  $\alpha$  (vue de dessus elle a une forme carrée tel que montré par la figure 3.25) puisqu'elle communique beaucoup plus avec les structures boucle avec 27%. Sa sortie et son entrée principale sont la lettre **A1** avec 15% et 17% respectivement.



**Figure 3.25** – A2 vue de dessus

**A3** : Les acides aminés composant cette lettre sont organisés, dans une structure hélicoïdale. Elle a l'air de former une extrémité d'une hélice  $\alpha$ . Il s'agit d'un tour d'une hélice droite qui contient trois résidus d'acides aminés. Elle ressemble à un pas d'une hélice  $3_{10}$  en effet vue de dessus elle a une forme triangulaire (voir la figure 3.26). Elle communique beaucoup plus avec les structures boucle avec 46%. Sa sortie principale est la lettre **A2** avec 17%, alors que ses entrées hélices **A1** et **A2** sont équiprobables à 8% .



**Figure 3.26** – A3 vue de dessus

#### pour les états brins

**B1, B3, B4 ,B5** : Ces lettres représentent des motifs brins qui forment ensemble un feuillet. Les lettres **B1** et **B4** ont le moins de communication avec les motifs de type 'boucle', et les probabilités les plus élevées de transition à des motifs de type 'brin' (85% et 72% respectivement). C'est pour cela qu'elles laissent croire qu'on les classe comme des cœurs de brins. Par ailleurs les lettres structurales **B3** et **B5** représentent des bords entrants et sortants de brins.

**B2**: représente une extrémité des brins reliée directement avec des motifs de type boucles.

#### pour les états boucles

**C8, C9, C10, C14, C15, C16, C18, C19** : représentent des différents motifs pour des boucles. Ces lettres communiquent le plus souvent entre elles. **C18** est la plus courte des boucles après **C2**.

**C2, C11, C13** : ces lettres représentent des motifs boucles pour début et fin d'hélice. La lettre **C2** est un exemple de motif de boucle courte qui détermine la fin d'une hélice à 59% pour passer à une boucle. Une autre lettre **C11** marque la fin des hélices avec 59%, sauf que celle-ci est un motif de boucle longue. Pour la lettre **C13** restante elle annonce l'entrée à une hélice à 45%.

**C3, C5, C6, C7, C17** : Ces lettres sont des motifs de boucles longues. Les lettres **C5, C7** marquent un vrai début d'un brin d'un feuillet  $\beta$  avec 57% et 52% respectivement. Par contre les lettres **C3, C6, C17** annoncent la fin d'un brin avec 46%, 47% et 54% respectivement. Ces lettres entament ensuite des boucles avec les probabilités de 62%, 64% et 88% respectivement.

**C1**: Cette lettre est spéciale car elle marque la fin d'une hélice avec 40% et le début d'un brin avec 54%

**C4, C12** : ces deux dernières lettres sont aussi spécifiques car elles désignent la fin des brins avec 44% et 64% respectivement alors qu'elles annoncent le début d'une hélice avec 50% et 75% respectivement, faisant ainsi **C12** une entrée d'hélice par excellence.

#### Les transitions sont impossibles entre les lettres hélices et les lettres brins :

On remarque aussi qu'il n'y a pas de transitions entre les lettres des hélices et les lettres des brins, ce qui est tout à fait normal car il n'y a quasiment pas de sortie ou d'entrée de la part des motifs brins vers les hélices. Les sorties des motifs hélices vers les motifs brins ne se font qu'à travers **C2** et **C18**. Alors que les entrées des motifs brins se font de **B3** et **B5** uniquement.

De plus on note que les motifs boucles qui se répètent dans tous les entrées de motifs hélices sont : C6, C10, C14 et C15. Les motifs C13, C2, C11, C15 et C16 ne sont que des motifs de sorties pour des motifs hélices jamais pour des motifs brins. Il y a aussi C5, C8 et C14 qui sont des sorties des motifs hélices mais qui peuvent également concerner des motifs brins et boucles. D'autre part, les motifs C2, C3, C5, et C7 ne sont que des entrées pour des motifs brins, jamais pour des motifs hélices.

### 3.10 SAFlex et HMM-SA27 : Comparaison méthodologique

Cette section présente une comparaison méthodologique entre les deux mises au point de HMM-SA27 (voir la figure 3.8) et SAFlex (voir la figure 3.9). Comme notre nouvelle méthode s'inspire de celle de HMM-SA27, précisément dans l'utilisation des mêmes formules de description du fragment structural ainsi que dans l'utilisation de HMM, il est donc très important de comparer qualitativement les deux méthodes et modèles de développement de ces deux alphabets structuraux.

Les points suivants résument les différences au niveau *encodage structural* entre l'encodage initial utilisé par HMM-SA27 et le nouvel encodage de SAFlex :

- 1) HMM-SA27 proposait un seul encodage structural *Viterbi* des chaînes d'une structure tridimensionnelle, alors que SAFlex présente trois différents encodages structuraux : *MAP*, un nouvel encodage marginal *POST* et le nombre effectif de lettres à chaque position donnée *NEFF*.
- 2) L'ancien encodage ne permettait qu'un encodage par chaîne alors que le nouveau alphabet SAFlex présente en plus un encodage consensus pour les chaînes homomères de la même protéine.
- 3) HMM-SA27 ne prenait pas en compte le manque de résidus ce qui engendrait une perte de lettres et donc un décalage, ce problème n'existe plus avec le nouvel encodage SAFlex car il exploite l'information de transition entre les lettres structurales en complétant et comblant les vides entre les résidus de la chaîne.

En ce qui concerne *l'apprentissage* des alphabets structuraux HMM-SA27 et SAFlex, nous décrivons dans les points suivants les principales différences entre l'apprentissage HMM-SA27 et le nouvel apprentissage de SAFlex :

- 4) La loi initiale de notre nouveau modèle est uniforme, ce qui améliore l'apprentissage en évitant le problème de surajustement, tandis que l'ancien modèle nécessitait l'adaptation des valeurs initiales pour chaque lettre au début de l'apprentissage.
- 5) Le nouvel apprentissage de SAFlex est entièrement automatique, alors que l'apprentissage de HMM-SA27 nécessitait une intervention manuelle notamment dans les choix des valeurs initiales pour les lettres.
- 6) Le nouveau modèle SAFlex permet en outre la gestion des descripteurs manquants même quand les quatre manquent, ce qui permet d'exploiter les informations partielles avec manque de résidus au moment de l'apprentissage. Ce qui représente un atout par rapport à l'ancien modèle de HMM-SA-27 qui négligeait cet aspect. SAFlex permet donc de gérer ce manque d'une manière correcte et juste.



- 7) Les paramètres de SAFlex sont appris sur des données récentes. Par contre ceux de HMM-SA27 ont été appris sur des données anciennes.
- 8) Le premier modèle ne tenait pas compte en outre de la redondance entre les chaînes (les chaînes multi-conformationnelles, homomères), le nouveau modèle prend celle-ci en compte lors de l'apprentissage des paramètres, ce qui permet d'apprendre la variabilité entre les différentes conformations d'une même protéine.
- 9) L'apprentissage de SAFlex est simplifié à l'aide de l'algorithme EM. Ce qui n'est pas le cas pour l'ancien modèle en utilisant l'algorithme SAEM [52].
- 10) Le nouveau apprentissage est parcimonieux car le modèle n'apprend pas les lois initiales mais suppose qu'elles sont uniformes.

Quant à *l'implémentation* des deux méthodes, HMM-SA27 a été implémenté en langage C alors que le nouvel alphabet structural SAFlex est implémenté en langage C++, celui-ci simplifie la programmation et la rend plus lisible grâce au concept de l'orienté objet. Cette architecture permet de diviser le programme en plusieurs objets interagissant entre eux. D'autant plus que l'implémentation de SAFlex utilise la bibliothèque STL du C++, ce qui permet un accès très souple à la mémoire sans perdre en efficacité. Elle a comme avantages principaux par rapport à l'implémentation initiale (de HMM-SA27) une meilleure gestion des données manquantes et des chaînes multi-conformationnelles. En outre, la nouvelle implémentation est écrite en échelle logarithmique afin de permettre le traitement d'un nombre important de données. Le code source de l'implémentation de SAFlex a été optimisé contrairement à l'ancienne implémentation. Cette optimisation a fait que le temps d'apprentissage d'un alphabet de taille 27 est sept fois plus rapide que la nouvelle version implémentée avant optimisation.

Cette implémentation est innovante en permettant l'apprentissage de la variabilité des chaînes tout en prenant en compte la redondance des chaînes et en évitant la perte des données.

### 3.11 Conclusion

Dans ce chapitre, nous avons exposé la méthodologie de développement du nouvel alphabet structural SAFlex basée sur plusieurs innovations. Nous avons ensuite détaillé son implémentation et son optimisation. Nous avons présenté enfin SAFlex et nous l'avons utilisé sur un certain nombre de protéines, ce qui a permis de mettre en évidence ses nouveautés.

Ce travail est innovant, d'une part au niveau méthode, ce qui a permis la mise au point d'un nouveau modèle d'encodage robuste et rigoureux doté de plusieurs options : MAP, POST et NEFF. Le Maximum *a posteriori* (MAP) fournit la séquence de LS la plus probable d'encodage de la structure 3D d'une chaîne protéique. La distribution marginale *a posteriori* définit l'incertitude d'encodage. Quant au nombre effectif de LS à chaque position donnée de l'encodage structural (NEFF), il fournit une mesure pratique pour la certitude de l'encodage et permet de mieux cerner les données manquantes dans les structures PDB.

D'autre part, au niveau implémentation, en utilisant une architecture orientée objet par le langage C++. Cette nouvelle implémentation est écrite en échelle logarithmique ce qui permet un traitement d'un nombre plus important de données. De plus, le code source de SAFlex a été optimisé. Cette



optimisation a permis un gain de temps important, ce qui rend l'apprentissage de l'alphabet sept fois plus rapide qu'avant optimisation. En outre, cette implémentation permet la prise en compte de la redondance des chaînes et évite la perte des données à travers un algorithme robuste gérant et traitant les données manquantes dans les structures 3D.

Ce travail de développement d'un alphabet structural offre aux biologistes un outil corrigé plus performant que HMM-SA27, un nouvel encodage meilleur du point de vue méthodologique, tenant compte de la redondance des données et qui gère les données manquantes au niveau des structures PDB. Ce travail a donné lieu à l'élaboration d'un article ayant pour titre « *SAFlex: A structural alphabet extension to integrate protein structural flexibility and missing data information* » en révision pour publication à la revue *PLOS ONE* (mis en annexe H). Cette méthodologie a été concrétisée par un outil mis à la disposition de la communauté scientifique via un site web qui fait l'objet du chapitre suivant.

# Un nouveau site web mis à la disposition de la communauté scientifique

## Résumé

Ce chapitre présente la contribution concrète de cette thèse. Il permet la mise à la disposition du travail effectué au cours de cette thèse à la communauté scientifique. Il présente la conception et l'implémentation du site web. Il détaille surtout les moyens techniques utilisés pour la réalisation, la mise en œuvre du site SAFlex, l'architecture du site web et ses performances. L'implémentation de l'interface graphique et son architecture sont également détaillées. Les deux versions utilisables de l'AS : SAFlex.V1 et SAFlex. $\beta$  sont présentées. Dans ce chapitre, les entrées et les résultats du site sont explicités en détail. Les principales fonctionnalités sont décrites. Chacune de ces fonctionnalités est représentée sous forme textuelle et graphique et est illustrée par une application sur protéines de la PDB.

## 4.1 Introduction

Le site web SAFlex (<http://saflex.rpbs.univ-paris-diderot.fr/>), développé dans le cadre de cette thèse de doctorat, met à la disposition de la communauté scientifique des outils d'encodage et de simplification des structures protéiques : SAFlex.V1 et SAFlex. $\beta$  utilisés pour décrire finement les structures protéiques 3D et simplifier leurs analyses. Il permettra à terme de faciliter la comparaison des structures protéiques afin de mieux comprendre la fonction de celles-ci.

Ce site web s'adresse aux chercheurs intéressés par l'analyse et la caractérisation des structures protéiques dans les domaines de la biologie structurale et de la recherche pharmaceutique. Il est hébergé sur un serveur Linux situé à l'université Paris-Diderot en France. Antérieurement à notre site web une page web proposait l'outil SA-search publié dans l'article : [79]. Celui-ci a été décrit dans la section 1.2 sur "Applications des AS" du chapitre "Introduction" à la page 4. SAFlex s'inspire du modèle de l'AS HMM-SA27 utilisé par l'outil SA-Search, mais propose des améliorations importantes au niveau du modèle de l'AS. Ce qui présente l'avantage de mettre à la disposition des chercheurs un

encodage robuste aux données manquantes. De plus, il a été développé pour fournir aux utilisateurs des représentations graphiques de leurs structures protéiques sous plusieurs modes (encodage, entropie, NEFF et POST). Ce chapitre présente les différentes fonctionnalités offertes aux visiteurs.

## 4.2 Conception et implémentation du site web

La création et la conception du site web SAFlex fait appel à plusieurs compétences concernant l'architecture de l'information, l'ergonomie, le référencement et la rédaction du contenu des pages. Elle nécessite une réflexion sur l'objectif du site, sa cible et les services attendus. Cette conception consiste aussi à établir la structure des pages HTML, à définir l'arborescence et la navigation dans le site web et enfin à élaborer le graphisme.

Ce travail a donc nécessité une bonne connaissance des aspects techniques liés à la programmation web tels que : les spécificités des différents langages, l'accessibilité et la portabilité du site web. Nous expliquons les différents moyens techniques utilisés pour la réalisation et la mise en œuvre de SAFlex. Ensuite nous détaillons l'architecture du site web et pour finir nous évoquons les performances de SAFlex.

### 4.2.1 Implementation de l'interface graphique

SAFlex propose des fonctionnalités dynamiques évoluant en fonction de la requête du visiteur du site. Pour réaliser celles-ci, le site web utilise le serveur *Apache HTTP* et des langages d'interface utilisateur tels que : HTML5, CSS personnalisé, JQuery (version 3.1.1), bootstrap (version 3.3.7) et Javascript. Ceux-ci permettent la rédaction du contenu statique des pages (HTML5), la mise en forme du texte (police d'écriture, couleur, taille) (CSS) et ajoute du dynamisme au navigateur (JavaScript). Par ailleurs, les programmes qui tournent du côté serveur emploient d'autres langages qui peuvent créer dynamiquement les pages, en analysant les requêtes des visiteurs tout en adaptant les résultats affichés. Ainsi, les requêtes des formulaires sont écrites en PHP (version 7.0), les résultats quant à eux sont générés par les programmes implémentés en C++ comme déjà expliqué dans le chapitre 3.

De plus, SAFlex emploie les logiciels libres « JSmol » (JavaScript-Based Molecular Viewer From Jmol <https://sourceforge.net/projects/jsmol/>) et « PV » (JavaScript Protein Viewer <http://biasmv.github.io/pv/>) pour la visualisation de structures protéiques en 3D directement dans les navigateurs web. De même notre site web fait appel à la bibliothèque graphique JavaScript « Chart.js » (<http://www.chartjs.org/>) pour afficher les résultats sous forme graphique et la bibliothèque JavaScript cross-browser « MathJax » (<https://www.mathjax.org/>) pour intégrer les notations mathématiques dans les pages du site.

### 4.2.2 Architecture du site web

Le diagramme présenté par la figure 4.1, décrit la structure hiérarchique des pages web du site SAFlex. La page principale (*home page*) de SAFlex se distingue des autres pages du site par les différentes rubriques de l'organisation du site web ainsi que la présentation de SAFlex. Ce qui permet non seulement une navigation facile pour les visiteurs mais aussi une découverte de l'organisation du

site ainsi que les différentes applications offertes par SAFlex. SAFlex est organisé autour de *quatre principales rubriques* : « *Structural Alphabet* », « *Applications* », « *Help* » et « *Contact* ».

Le contenu de la première rubrique *Structural Alphabet* vise à présenter les deux versions de l'AS SAFlex, développées dans cette thèse : SAFlex.V1 et SAFlex. $\beta$ . Ces versions seront décrites dans la section suivante.

La deuxième rubrique *Applications* présente des services très pratiques basés sur le concept de l'AS. Cette rubrique propose aux visiteurs huit applications d'intérêts : « *3D-View encoding* », « *SAFlex-Encoder* », « *3D encoding Entropy* », « *Consensus 3D encoding* », « *NEFF Encoding* », « *POST Encoding* », « *Consensus NEFF Encoding* » et « *Consensus POST Encoding* ».

Chaque application fait appel à une page formulaire web, où les visiteurs sont invités à entrer un identifiant de fichier PDB ou à télécharger un fichier de format PDB. Toutes les applications du site offrent également aux visiteurs deux différentes versions de l'AS au choix. À terme, de nouvelles versions de l'AS, avec différents nombre des LS pourront être introduites.

Une fois le formulaire transmis par le visiteur, un gestionnaire est chargé de traiter les macromolécules biologiques entrées, ensuite la requête est envoyée au serveur afin de faire tourner, en arrière plan, les programmes implémentés en C++ : le parser SAFlex-PDB (présenté dans le chapitre 2) et le programme SAFlex (décrit dans la section 3.6 du chapitre 3). Les résultats du programme sont ensuite récupérés et affichés dans la page web *Résultats*. Par la suite, les fichiers de résultats sont générés sur le serveur et peuvent être téléchargés par les utilisateurs. Les résultats dépendent du choix de la version de l'alphabet et seront adaptés en fonction de ce choix.

Chaque page résultats présente un résumé sur les structures protéiques contenues dans le fichier PDB, et met en évidence leurs séquences structurales coloriées. De plus, il permet la visualisation de ces chaînes en utilisant les programmes PV (JavaScript Protein Viewer <http://biasmv.github.io/pv/>) et JSmol (JavaScript-Based Molecular Viewer From Jmol <https://sourceforge.net/projects/jsmol/>)<sup>1</sup>. Par ailleurs, il est à noter que les résultats proposent aussi pour chaque chaîne protéique, une entropie d'encodage structural ainsi que différents encodages (MAP, NEFF et POST) et une entropie consensus (NEFF et POST consensus) pour les chaînes de la même protéine. Ces résultats seront expliqués en détail dans la section 4.3.2

La troisième section *Help* explique aux visiteurs comment utiliser les services proposés par notre site web. Enfin, les coordonnées des personnes à contacter sont disponibles sous la rubrique *Contact*.

---

1. Jsmol et PV sont des programmes permettant la visualisation de structures protéiques en 3D directement dans les navigateurs web.

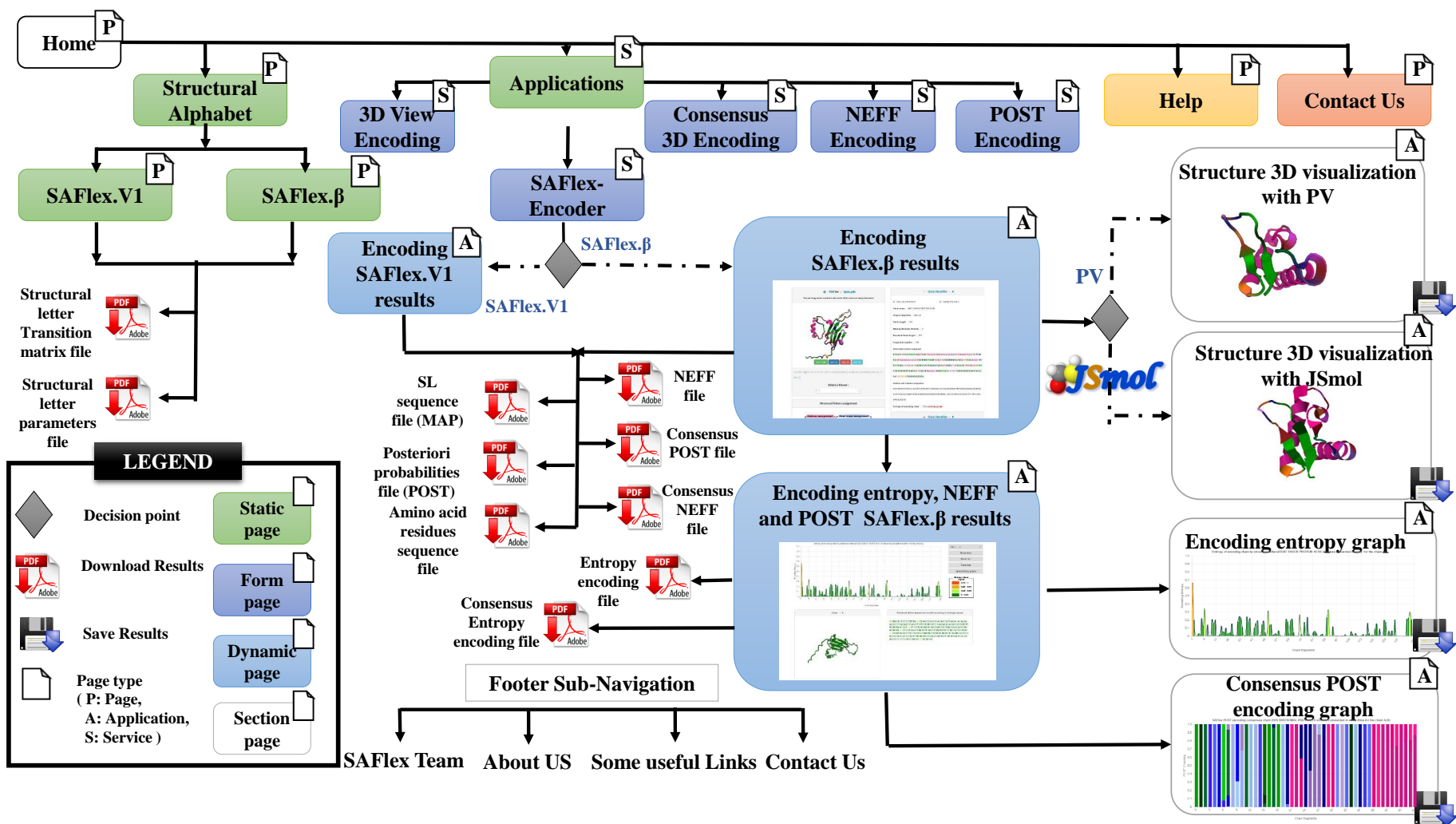


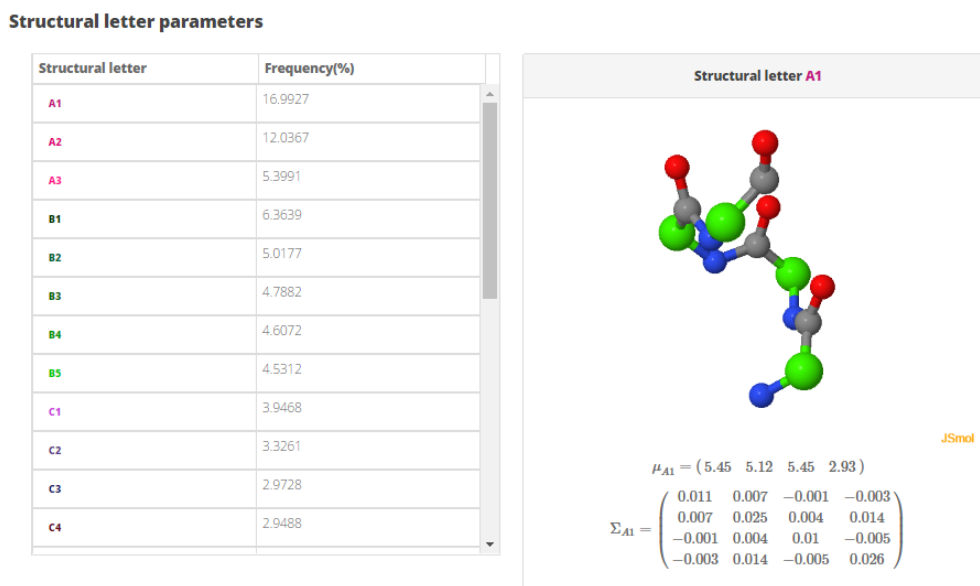
Figure 4.1 – Architecture du site web SAFlex

### 4.2.3 Les versions de l'alphabet structural : SAFlex.V1 et SAFlex. $\beta$

Le principe et l'intérêt des AS sont développés dans la page principale du site web. SAFlex présente aux visiteurs du site web un accès aux informations concernant l'AS SAFlex.

Ainsi, SAFlex fournit deux versions de l'alphabet : SAFlex.V1 et SAFlex. $\beta$ . Il décrit leurs LS et les paramètres de celles-ci (moyenne de la lettre et sa matrice de covariance). Il met à disposition les représentations graphiques des représentants moyens des LS. Il offre également la possibilité de télécharger leurs matrices de transition ainsi que les paramètres des LS. Nous avons déjà présenté la version SAFlex.V1 dans la section 3.8 au cours du chapitre 3 à la page 64 et nous avons exposé brièvement la version SAFlex. $\beta$  dans la section 3.9 du chapitre 3.

En outre, la visualisation en 3D de chaque représentant d'une lettre structurale est mise en ligne à travers notre site web sur l'AS. À titre d'exemple, la figure 4.2 montre les paramètres de la lettre "A1" et sa conformation 3D affichée par JSmol pour la version SAFlex. $\beta$ . Ces conformations 3D des LS peuvent être sauvegardées à partir de la rubrique *Structural Alphabet*.



**Figure 4.2 – Les caractéristiques structurales des lettres de l'alphabet structural SAFlex. $\beta$  sur la page web.** La figure indique la lettre structurale 'A1' affichée dans une fenêtre JSmol suivie de sa moyenne  $\mu_s$  et de sa covariance entre les descripteurs  $\Sigma_s$ .

### 4.2.4 Performance du site web

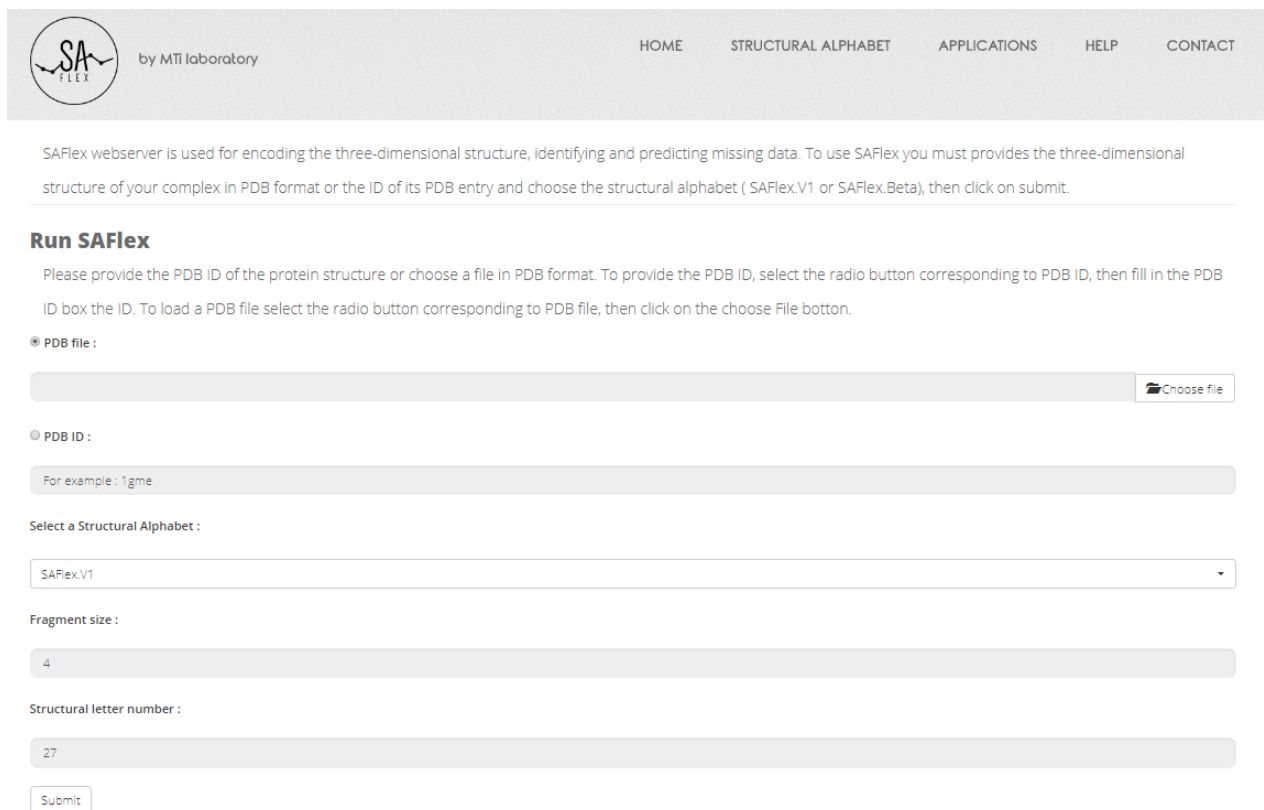
SAFlex présente un atout majeur par rapport aux sites existants sur les AS car il est le seul qui présente un outil robuste aux données manquantes au niveau des structures 3D et propose un encodage pour les régions manquantes. Il fournit un traitement et un encodage rapide et efficace d'une structure PDB (moins d'une seconde par chaîne protéique). Ceci est possible par l'utilisation du langage C++, considéré comme *un langage de bas niveau*. Ce qui lui permet d'être efficace, puissant et extrêmement rapide grâce à la manipulation explicite des registres et des adresses mémoires. De plus, SAFlex a été testé avec succès sur Chrome, Firefox et Safari.

## 4.3 Description des fonctionnalités du site web

### 4.3.1 Les entrées du site web

Les entrées du site web sont un fichier de structure 3D au format PDB et un AS à choisir dans les deux versions de l'alphabet : SAFlex.V1 ou SAFlex. $\beta$  actuellement proposées. Chacune de ces entrées a été développée et a fait l'objet de plusieurs sections des chapitres 2 et 3 dans ce manuscrit. Le visiteur peut fournir la structure 3D de la macromolécule biologique de deux différentes manières : en fournissant un identifiant PDB ou en téléchargeant un fichier de format PDB (voir figure 4.3). En effet il est très pratique de pouvoir donner un simple identifiant de fichier PDB en entrée plutôt que d'utiliser un fichier PDB. Cependant cette dernière option pourrait permettre à certains chercheurs de charger le PDB de leurs choix même s'il a été modifié.

Ces choix sont obligatoires pour encoder les chaînes de protéines présentes au niveau du fichier PDB requis. Si le visiteur tente de soumettre le formulaire sans fichier PDB ou sans sélectionner un AS, un message d'erreur indiquant au visiteur de remplir les données manquantes sera affiché. Les données transmises par le formulaire sont traitées par des programmes C++ qui produisent plusieurs résultats de fichier « *Fasta* ». Ces différents résultats obtenus sont expliqués dans la section suivante.



The screenshot displays the SAFlex webserver interface. At the top, there is a header with the SAFlex logo (a stylized 'SA' with 'FLEX' below it) and the text 'by MTI laboratory'. To the right of the logo are navigation links: HOME, STRUCTURAL ALPHABET, APPLICATIONS, HELP, and CONTACT. Below the header, a paragraph explains the webserver's purpose: 'SAFlex webserver is used for encoding the three-dimensional structure, identifying and predicting missing data. To use SAFlex you must provide the three-dimensional structure of your complex in PDB format or the ID of its PDB entry and choose the structural alphabet (SAFlex.V1 or SAFlex.Beta), then click on submit.' Below this text is the 'Run SAFlex' section. It contains instructions: 'Please provide the PDB ID of the protein structure or choose a file in PDB format. To provide the PDB ID, select the radio button corresponding to PDB ID, then fill in the PDB ID box the ID. To load a PDB file select the radio button corresponding to PDB file, then click on the choose File button.' There are two radio buttons: 'PDB file:' (selected) and 'PDB ID:'. Below the 'PDB file:' radio button is a text input field with a 'Choose file' button to its right. Below the 'PDB ID:' radio button is a text input field with the placeholder text 'For example: 1gme'. Below these fields is a section titled 'Select a Structural Alphabet:' with a dropdown menu currently showing 'SAFlex.V1'. Below the dropdown is a 'Fragment size:' label followed by a text input field containing the number '4'. Below that is a 'Structural letter number:' label followed by a text input field containing the number '27'. At the bottom of the form is a 'Submit' button.

Figure 4.3 – Le formulaire « SA-Encoder » du site web SAFlex.

### 4.3.2 Les résultats du site web

Les résultats sont organisés en cinq principales fonctionnalités: « *3D view results* », « *Protein structure details* », « *3D encoding results* », « *Consensus 3D encoding results* » et « *Download outputs* ». Ces résultats sont basés sur l'AS SAFlex présenté et développé déjà dans le chapitre "Les avancées méthodologiques" à la page 39. Ils utilisent également la même nomenclature et légende que les LS présentées dans celui-ci.

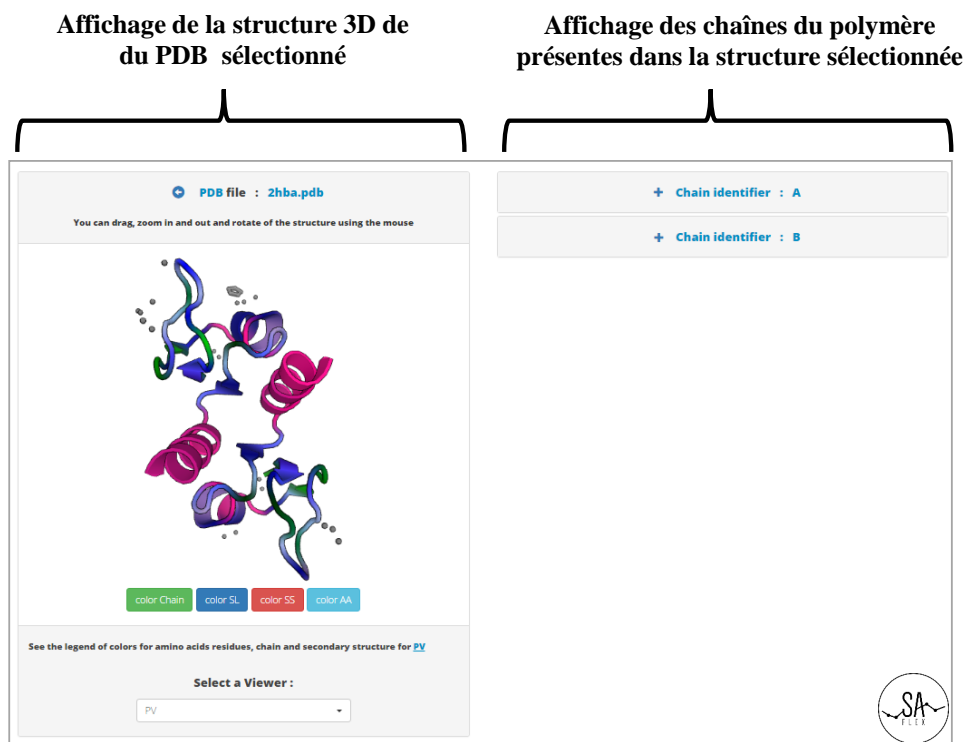
#### 4.3.2.1 « 3D view results »: Plusieurs modes de visualisation des chaînes protéiques

La figure (4.4.a) illustre, en utilisant la visualisation PV, la structure 3D présente dans le fichier PDB "2hba" entrée. Cette structure correspond au domaine N-terminal de la protéine ribosomale L9, (NTL9), une petite protéine alpha-bêta de la protéine mixte à 52 résidus. NTL9 a été largement utilisée comme système modèle pour des études expérimentales et computationnelles du repliement des protéines [38]. Ainsi, les utilisateurs du site ont le choix entre deux visualisateurs PV et JSmol, ils peuvent également déplacer la structure 3D de la protéine, pivoter, zoomer sur différentes parties de la structure, et peuvent finalement enregistrer une image haute résolution de la pause personnalisée obtenue après manipulation de la chaîne ou protéine.

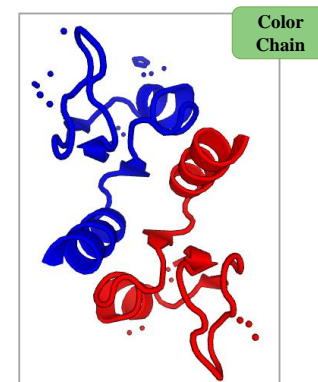
Quatre modes d'affichage distincts sont disponibles pour afficher les structures dans différents styles et couleurs. Les utilisateurs peuvent choisir parmi trois modes de style prédéfinis et un nouveau style personnalisé. Le premier mode « *Chain* » considéré par défaut colorie chaque chaîne par une couleur différente (voir figure 4.4.b). Le deuxième mode « *SS* » (*Secondary Structure*) (voir figure 4.4.c) colorie les chaînes en fonction de la structure secondaire (DSSP [93]) et le troisième mode « *AA* » (*Amino Acids*) (voir figure 4.4.d) colorie la structure en fonction des 20 résidus. Quant au mode personnalisé « *SL* », il colorie les fragments structuraux en fonction des LS choisies. Ce mode est primordial dans notre travail car il permet de montrer de façon concrète à quoi ressemble la structure 3D d'une lettre structurale. Ce qui permet de lier la séquence des lettres structurales avec la visualisation 3D des chaînes protéiques. Tous les modes d'affichage visualisent la structure en style *Cartoon*.

En outre, SAFlex fournit trois autres options pour explorer une chaîne sélectionnée dans un polymère. L'option par défaut permet de visualiser en 3D toutes les chaînes du polymère, mais seule la chaîne sélectionnée par l'utilisateur est coloriée en fonction des lettres structurales, toutes les autres chaînes sont grisées, comme le montre la figure 4.5.a. Dans cette figure, l'utilisateur a sélectionné la chaîne 'A' du fichier PDB, ce qui fait que la structure 3D de celle-ci s'est coloriée en fonction des LS alors que la structure 3D de la chaîne 'B' est coloriée en gris. Les utilisateurs peuvent également afficher uniquement cette chaîne « *Show only this chain* » ou se focaliser sur celle-ci « *Center this chain* » ou de préférence utiliser les deux options ensemble. Ce sont des options très utiles dans le cas où la structure sélectionnée comporte de nombreuses chaînes, tels que : des protéines, de l'ADN, de l'ARN et des ligands. Le fichier PDB '3LK4' est un exemple concret d'un tel cas qui comprend 36 chaînes. Donc la combinaison de ces deux options indépendantes : « *Show only this chain* » et « *Center this chain* » représente une solution très pratique pour examiner chaque chaîne à part.

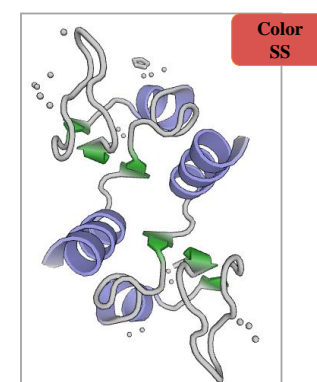




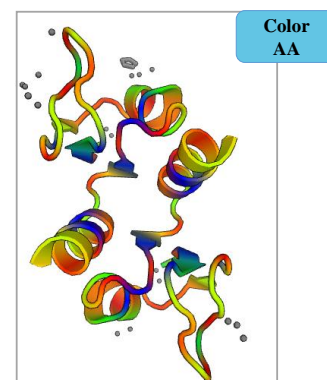
**(a) Exemple de la structure 3D de la page de résultats du site web SAFlex.** Le domaine N-terminal de la protéine ribosomale L9 (NTL9) (fichier PDB '2hba' ) est représenté dans cet exemple.



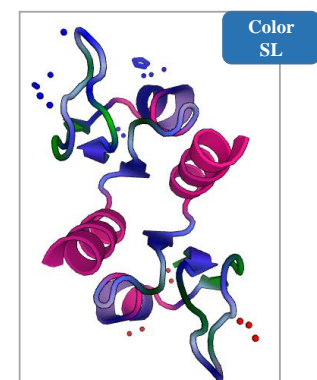
**(b) Chain mode**  
Colorie chaque chaîne protéique par une couleur différente.



**(c) SecondaryStructure (SS) mode**  
Colorie la structure 3D par structure secondaire.



**(d) Amino Acid (AA) mode**  
Colorie la structure 3D par résidus.



**(e) Structural letter (SL) mode**  
Colorie la structure 3D par Lettres Structurales.

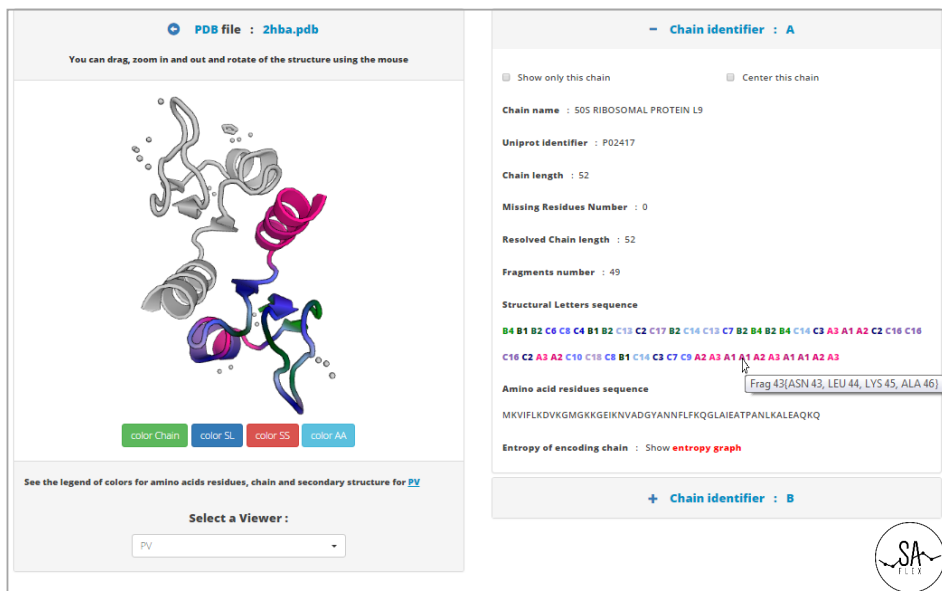
Figure 4.4 – Les modes de visualisation 3D associés aux structures proposés par SAFlex.

#### 4.3.2.2 « Protein structure details » : Les informations relatives aux chaînes protéiques

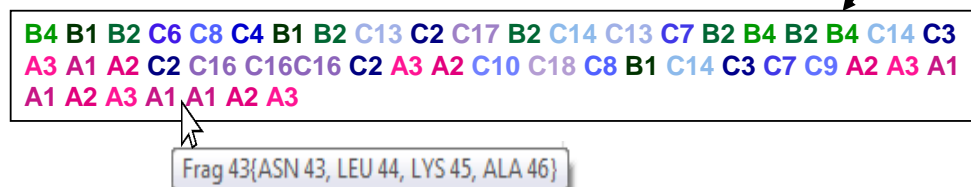
Pour chaque chaîne protéique de la structure entrée, les informations importantes relatives à cette chaîne sont affichées dans une section comme illustré par la figure 4.5.b. Les informations affichées incluent le nom de la protéine, sa référence Uniprot, sa longueur totale, sa longueur résolue, le nombre de ses résidus manquants et sa séquence de résidus. Cette section comporte également des informations intéressantes concernant l'encodage structural de la chaîne protéique tels que le nombre des fragments structuraux ainsi que la séquence des LS. Quant à l'encodage structural de la chaîne, il peut être visualisé par JSmol ou PV dans la page web comme indiqué sur la figure 4.5.a. Cette même section renvoie aussi vers une page des résultats concernant l'entropie, NEFF et POST de l'encodage structural de la chaîne sélectionnée. Ces résultats sont abordés dans la section suivante.

En ce qui concerne l'encodage structural (l'encodage MAP) de la chaîne, il correspond à une séquence de LS où chaque lettre est représentée par une couleur spécifique (voir figure 4.5.c). Cette lettre structurale définie au cours des chapitres précédents, est associée à un fragment structural, ce dernier est visualisé graphiquement par PV ou JSmol en conservant la même couleur que cette lettre. Ainsi, cette représentation fait la liaison entre la visualisation 3D et l'encodage du fragment structural de la chaîne sélectionnée.

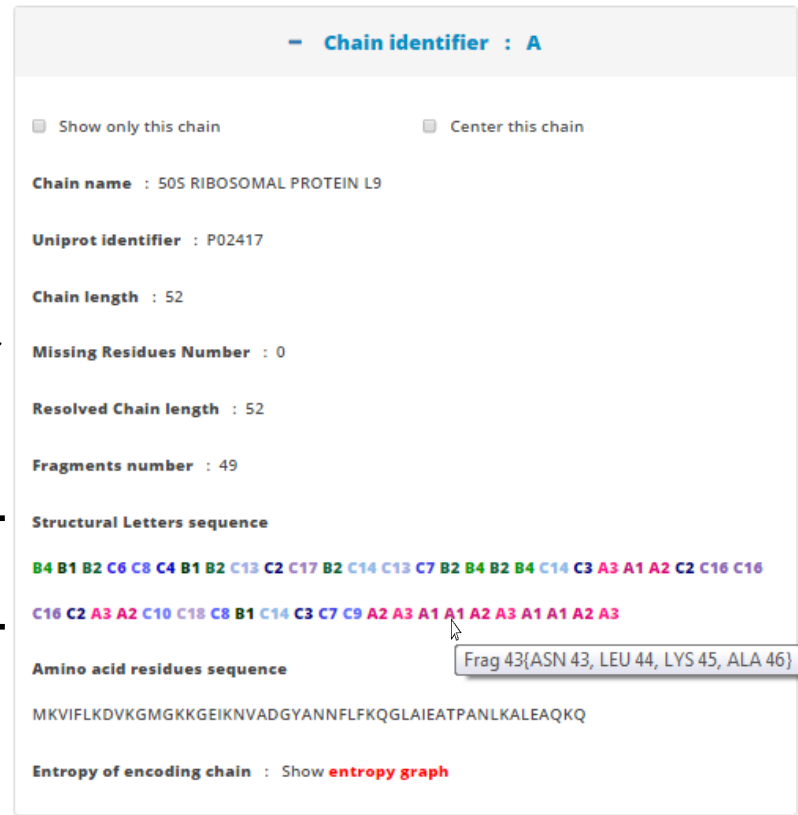
De plus, l'utilisateur du site dispose par une infobulle des informations concernant les résidus constituant chaque fragment structural et ce en pointant la souris sur une lettre de la séquence structurale. En outre, en cliquant sur le lien en haut de la fenêtre comportant la visualisation 3D, l'utilisateur peut accéder à tous les détails, présentés par la banque PDB, sur la structure entrée.



(a) Un exemple d'encodage structural de la chaîne A du PDB 2hba présenté dans la page résultats du site SAFlex.



(c) La séquence de Lettres Structurales associée pour la chaîne 2hbaA avec un infobulle sur la lettre A1



(b) Section des informations de la chaîne 2hbaA

Figure 4.5 – Les résultats de l'encodage structural de la chaîne "A" du PDB '2hba' affichés par SAFlex. L'encodage utilise la version SAFlex.V1.

### 4.3.2.3 « 3D encoding results » : l'entropie, NEFF et POST sont des indicateurs de qualité de l'encodage structural

SAFlex fournit un moyen efficace pour vérifier et s'assurer de la qualité de l'encodage structural en utilisant l'entropie, comme cela a été décrit dans le chapitre 3. Il interprète les résultats d'entropie par des représentations graphiques associées à un encodage structural. Ainsi, trois modes de représentation visuelle sont disponibles pour chaque chaîne protéique. Le premier mode affiche un graphique dans le coin supérieur gauche de la page de résultats d'entropie comme illustré par la figure 4.6. Les abscisses de celui-ci représentent les fragments structuraux de la chaîne et les ordonnées représentent les pourcentages de désordre. Ce désordre est obtenu par le calcul de l'entropie de Shannon [162] correspondante aux différentes nuances d'encodages associées à chaque fragment structural (voir le chapitre 3).



Figure 4.6 – L'entropie de l'encodage structural de la chaîne "A" du PDB '2hba'.

En ce qui concerne le « NEFF encoding », le nombre effectif de LS à chaque position donnée résultant de l'entropie d'encodage, il fournit une meilleure interprétation pour la certitude de l'encodage. En effet, lorsque le NEFF est proche de 1, l'encodage est très certain, alors que lorsqu'il est proche de 27 (le nombre de LS) il est incertain. Cependant, les résultats de NEFF obéissent à la même logique et raisonnement de présentation des résultats de l'entropie avec les trois modes de représentation visuelles et graphiques pour chaque chaîne protéique.

Le site web offre aussi aux utilisateurs trois affichages optionnels : *bar plot* ou *line plot* ou *both plot*, cette dernière représente une combinaison des deux premières options. Il permet également d'enregistrer le graphique avec l'option requise. La qualité de l'encodage d'un fragment structural est évaluée

en fonction de la valeur d'entropie et/ou NEFF associée à celui-ci. Une légende de code couleurs a donc été établie par SAFlex afin d'aider l'utilisateur à interpréter cette qualité d'encodage. De cette façon une couleur verte du graphique indique que l'encodage du fragment est certain. Par contre, une couleur rouge ne communique pas d'informations sur la certitude de l'encodage. Cette couleur reste cependant un indicateur très utile pour détecter les positions des régions manquantes au niveau de la chaîne protéique.

Le deuxième mode affiche la séquence des lettres structurales de la chaîne considérée coloriée en fonction du NEFF ou du pourcentage de désordre pour l'entropie. Enfin le dernier mode montre une visualisation 3D de celle-ci en coloriant différemment les fragments structuraux selon la couleur associée à l'entropie. Cette vue 3D peut être sauvegardée très facilement par les utilisateurs du site.

Par ailleurs, SAFlex offre la possibilité aux visiteurs de choisir les résultats de NEFF et de l'entropie d'encodage d'une chaîne tout simplement en sélectionnant la chaîne désirée dans une liste déroulante tel que montré dans la figure (4.6).

En ce qui concerne le « *POST encoding* », il représente la distribution marginale *a posteriori* indiquant l'incertitude d'encodage qui peut aider l'utilisateur du site web à détecter les régions structurales où l'encodage est difficile. La figure 4.7 montre la redistribution pondérée de la lettre structurale possible à chaque position donnée. Les résultats de « *POST encoding* » sont disponibles sous forme des trois modes de représentation graphiques associées à l'encodage structural pour chaque chaîne de la même manière qu'ils ont été présentés pour NEFF et l'entropie.

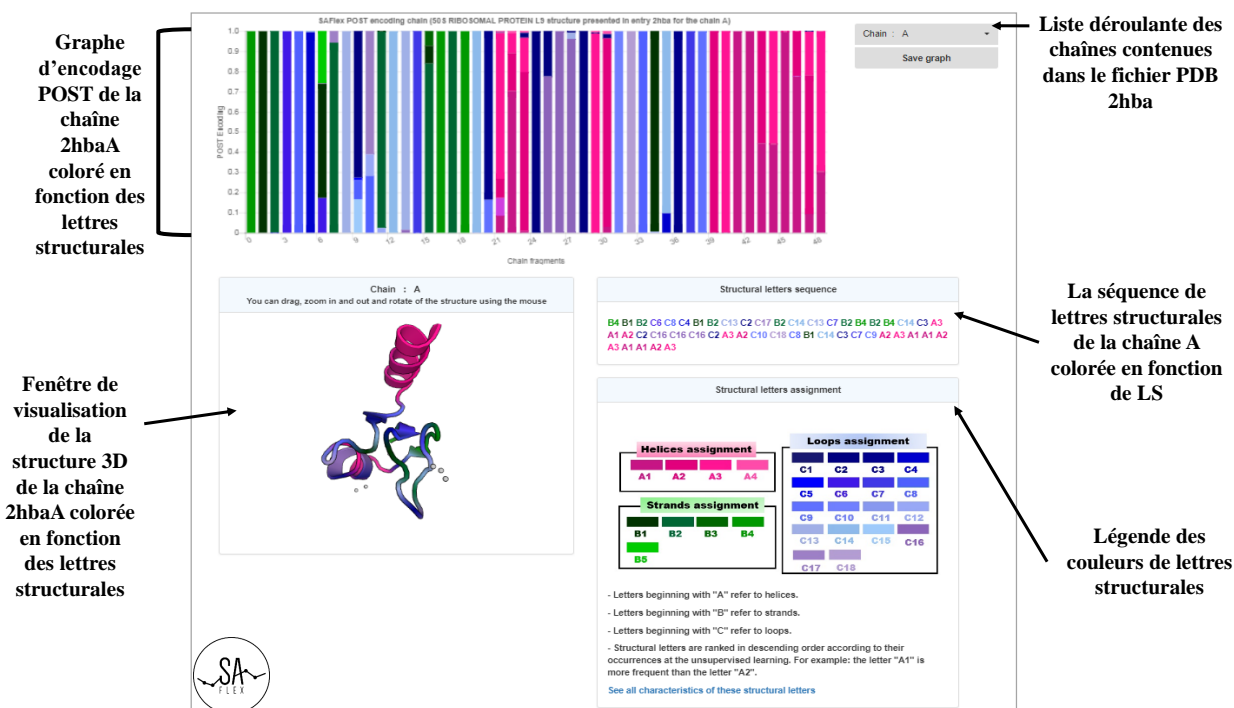


Figure 4.7 – L'encodage POST de la chaîne 2hbaA du PDB '2hba'. Cet encodage utilise la version SAFlex.V1.

#### 4.3.2.4 « Consensus 3D Encoding results » : Détection de la variabilité entre les conformations ou monomères d'une même protéine

SAFlex est capable de détecter toutes les protéines qui existent dans un fichier PDB entré même si elles sont différentes. Il propose un encodage consensus pour toutes ces protéines si pour chaque protéine il y a au moins deux chaînes qui la représentent et si celles-ci ont la même longueur. Ainsi la figure 4.8 illustre un encodage consensus des chaînes A et B de la protéine ribosomal (NTL9) K12M du fichier PDB "2hba" entrée.

De plus l'encodage consensus propose une description conformationnelle pour les parties manquantes de la protéine. Quant à l'entropie de cet encodage consensus, elle peut avoir plusieurs interprétations. Une entropie plus élevée s'expliquerait par plusieurs facteurs : la possibilité d'être dans une région dite désordonnée, la possibilité d'être dans une région dite flexible et enfin la possibilité d'être dans une région manquante. D'une autre façon, l'entropie consensus proposée par SAFlex permet de repérer les zones non prédictibles, dites désordonnées. Celles-ci sont signalées aussi par SAFlex par des LS incertaines associées à une entropie élevée et un NEFF élevé. Ces régions apparaissent clairement dans l'encodage POST de la chaîne consensus.



Figure 4.8 – L'entropie de l'encodage structural de la chaîne consensus associée aux deux monomères "A" et "B" du PDB '2hba'. Cet encodage utilise la version SAFlex.V1.

#### 4.3.2.5 « Download outputs » : Fichiers résultats à télécharger

Le site web SAFlex présente tous les résultats de traitement d'un fichier PDB sous forme de fichiers « Fasta ». Ainsi la section *Download result Files* met à la disposition des utilisateurs les fichiers suivants :

- fichier des séquences de lettres structurales (l’encodage MAP);
- fichier de l’entropie d’encodage des chaînes protéiques;
- fichier du nombre effectif de LS à chaque position des chaînes (l’encodage NEFF);
- fichier des probabilités *a posteriori* marginales des chaînes (l’encodage POST );
- fichier des séquences d’acides aminés;
- fichier des séquences de lettres structurales associées aux chaînes consensus;
- fichier de l’entropie d’encodage des chaînes consensus;
- fichier des probabilités *a posteriori* marginales des chaînes consensus;
- fichier du nombre effectif de LS à chaque position des chaînes consensus.

Ces fichiers peuvent être visualisés par n’importe quel éditeur de texte. Il est à noter que ces fichiers téléchargés contiennent les résultats de toutes les chaînes protéiques de l’entrée PDB. À titre d’exemple, le fichier PDB "2hba" illustré dans toutes les figures de ce chapitre contient deux chaînes protéiques, d’où le fichier résultat de séquences de LS comprend deux séquences chacune d’entre elles concerne une chaîne.

De plus, comme mentionné précédemment, les visiteurs du site peuvent sauvegarder toutes les vues de structure personnalisées selon différents modes (couleur par 'SL', couleur par 'SS', couleur par 'chaîne', couleur par 'AA', et couleur selon NEFF et l’entropie). Ils peuvent également enregistrer le graphique de l’entropie d’encodage ou NEFF des chaînes avec différentes options en cliquant sur le bouton *Save entropy graph*.

## 4.4 Conclusion

Au cours de ce chapitre, nous avons détaillé la conception et l’implémentation du site web SAFlex en décrivant également son architecture. Puis nous avons développé les fonctionnalités offertes aux utilisateurs. SAFlex dispose d’une interface graphique permettant ainsi plusieurs modes de visualisation des structures 3D en tirant au mieux profit des outils actuels (PV et JSmol). Il présente ses résultats d’une manière visuelle et graphique et sous forme de fichiers téléchargeables. Il permet également d’avoir accès à différentes informations concernant les versions SAFlex.V1 et SAFlex. $\beta$ .

Le site web SAFlex offre un outil puissant d’encodage des structures 3D permettant une simplification utile de celles-ci en une série de LS. Il offre l’opportunité d’utiliser différents AS, deux versions de l’AS, actuellement, pour effectuer cet encodage structural. De plus, il propose différents encodages et entropie et apporte une information sur la qualité de l’encodage (entropie, MAP, NEFF et POST), il permet la détection et la prédiction des régions manquantes. Nous avons donc proposé un outil performant tant du point de vue rapidité d’exécution que qualité d’encodage et présentation des résultats.

## Conclusion et perspectives

### Conclusion générale

Au cours de ces dernières années, les AS ont pris une importance considérable dans la recherche en biologie structurale. En effet, ceux-ci ont permis une simplification de la structure tridimensionnelle des macromolécules biologiques grâce à l'encodage structural. Celui-ci facilite l'analyse et la comparaison des conformations protéiques. Ces AS ont permis une amélioration considérable de nos connaissances sur l'organisation des structures 3D des protéines à travers le développement de multiples applications et outils très intéressants autour de ces AS.

L'objectif principal de notre travail était de remédier aux limites des AS qui doivent être améliorés pour mieux traiter les données manquantes, et la flexibilité intrinsèque observée grâce aux différentes structures disponibles dans la banque PDB. Ceci a un impact sur nos connaissances des fonctions des protéines et de leurs régions désordonnées, qui contribuent à la capacité des protéines à établir des interactions avec différents partenaires. Dans ce cadre, cette thèse a pour but d'améliorer l'AS HMM-SA27 existant mis au point par Camproux et ses collaborateurs afin de prendre en considération ces limites [28]. En effet, le modèle mathématique utilisé par HMM-SA27 ne prenait pas en compte ni les données manquantes ni la redondance de celles-ci (les chaînes d'homomères multiples) dans les fichiers PDB. Ce qui a amené à développer le nouvel AS SAFlex avec mise à la disposition de la communauté scientifique à travers la conception et l'implémentation d'un site web.

Par ailleurs, les *parsers* PDB existants présentaient des lacunes, à savoir un traitement inefficace des données manquantes dans les fichiers PDB, ce qui nous a conduit à développer un nouveau *parser* PDB plus adapté aux données manquantes. Le chapitre 2 présente notre nouveau *parser* SAFlex-PDB, qui est à la base un des modules qui contribuent à l'obtention de l'AS SAFlex. Ce module a été amélioré en librairie séparé de tous les autres modules statistiques qui génèrent l'AS. SAFlex-PDB permet d'analyser les données contenues dans les fichiers au format PDB. Il extrait donc les chaînes protéiques, les découpe en fragments structuraux et calcule les descripteurs associés à chaque fragment. Il détecte aussi les données manquantes et les trous dans les chaînes protéiques. Le *parser* SAFlex-PDB est d'une grande utilité pour le fonctionnement du site web de SAFlex que nous avons implémenté durant cette thèse. En effet, le site web l'utilise pour afficher toutes les informations sur



les chaînes, les protéines, les fragments structuraux, les résidus, les atomes, ..., etc. De plus, les descripteurs calculés par le *parser* sont utilisés pour l'encodage structural des chaînes protéiques au niveau du site web. Les résultats des tests comparatifs de notre *parser* SAFlex-PDB avec les *parsers* Biopython et Biojava démontrent que SAFlex-PDB a un intérêt d'une part sur le plan qualitatif en détectant les diverses erreurs liées au format PDB, d'une autre part dans sa rapidité d'analyse et traitement. En effet, SAFlex-PDB est 7 fois plus rapide que Biopython et 1,5 plus rapide que BioJava et ce sans parallélisation. Après parallélisation, le *parser* devient 23 fois plus rapide que BioPython et 5 fois plus rapide que BioJava. c'est ce qui était recherché et qui permet de contribuer à une meilleure performance du site web.

Dans le chapitre 3, nous présentons notre contribution principale en proposant des améliorations méthodologiques pour HMM-SA27 avec un modèle tenant compte d'une part, de la prise en compte de l'incertitude et manque au niveau des structures protéiques de la PDB et d'autre part, de la richesse des données et de la flexibilité. Nous avons présenté donc un nouvel alphabet structural, SAFlex, amélioré à partir de HMM-SA27, avec des descripteurs de fragment similaires, un nombre de lettres de 27, une distribution gaussienne par lettre et une matrice de transition. SAFlex a les trois principales nouveautés : (1) SAFlex permet un encodage structural plus rigoureux tenant compte de l'incertitude d'encodage en proposant trois sorties possibles : MAP, POST et NEFF; (2) une nouvelle implémentation robuste pour tout modèle de données manquants dans le fichier PDB; (3) SAFlex propose en plus d'un encodage par chaîne un nouveau modèle qui prend en compte les homomères et qui permet un nouvel encodage consensus pour ces homomères. Le nouvel encodage de SAFlex permet de fournir un indicateur de confiance renseignant sur les données manquantes au niveau de la structure des chaînes protéiques, le désordre et la variabilité entre les conformations des chaînes homomères d'une protéine.

Enfin, le chapitre 4 est destiné à la présentation du site web développé dans le cadre de cette thèse qui constitue la contribution concrète de ce travail à la communauté scientifique, les différentes fonctionnalités de celui-ci sont nombreuses. Ce site web public et gratuit montre que SAFlex est un outil puissant permettant un encodage structural plus rigoureux, robuste aux données manquantes et capable de détecter la variabilité entre les conformations d'une même protéine en proposant les différents encodages : MAP, NEFF et POST. Il présente les résultats d'une manière visuelle et graphique à travers plusieurs modes de visualisation des structures 3D en tirant profit des outils actuels (Jsmol et PV). Il est pourvu d'un outil d'évaluation de la qualité de l'encodage (NEFF) et de l'incertitude sur les données (POST), comme il permet de détecter les régions manquantes dans les protéines. Il met à la disposition de la communauté scientifique deux versions de l'AS : SAFlex.V1 et SAFlex. $\beta$ . Ce site web est un outil très performant tant du point de vue rapidité d'exécution que de la qualité d'encodage et de la présentation des résultats.

## Perspectives

Maintenant qu'une version basique de SAFlex a été implémentée, notre prochaine étape consisterait à utiliser des données à grande échelle afin d'apprendre un nouvel AS en prenant en compte la quantité des données dans la banque PDB. L'idée est d'étudier et de choisir un jeu de données beaucoup plus

grands, ce qui donne un AS pertinent plus complet et plus fin, ceci permet également un meilleur apprentissage. Néanmoins, un certain nombre de problèmes persistent encore, le modèle présente une instabilité à cause de la prise en compte de la redondance des données (chaînes multiples ou homomères). Ceci a nécessité le développement de la version SAFlex. $\beta$ , comme preuve de concept (une version mise en place de taille 27) qui est raisonnable, mais doit être validée. Le développement de la version SAFlex. $\beta$  montre que les avancées méthodologiques présentées dans cette thèse que nous avons développé sur un jeu de données de 100 PDB sont faisables. Cette version a été estimée en prenant en compte les conformations multiples des protéines et elle fonctionne bien. Cependant, notre travail d'analyse de cette version SAFlex. $\beta$  n'est pas fini au moment de la rédaction de ce manuscrit. Nous pensons que la version SAFlex. $\beta$  proposée comme preuve de concept au cours de cette thèse doit être améliorée en mettant en place une version finale d'un alphabet plus stable sur un large jeu de données PDB, car l'une des faiblesses de ce travail réside dans le côté peu stable de l'apprentissage de la version SAFlex. $\beta$  (plusieurs alphabets possibles avec le même jeu de données). Cette instabilité est due notamment à la prise en compte de la redondance des chaînes (chaînes homomères ou multiples) dans le jeu de données choisi. Cette redondance est nécessaire pour permettre la détection de la variabilité et l'apprentissage de la flexibilité entre les homomères des protéines. Ceci n'est pas le cas de la version SAFlex.V1 qui utilise des paramètres validés et qui a rendu service à la communauté scientifique pendant longtemps. Enfin, l'obtention d'une version finale d'un alphabet stable qui tient en compte la redondance des chaînes protéiques dans la banque PDB est un travail rigoureux et patient en cours de finalisation.

La présente thèse ouvre de multiples perspectives, qui peuvent s'organiser en trois grands volets : le modèle de l'alphabet structural, l'apprentissage d'un nouvel AS optimisé sur un large jeu de données PDB et l'exploitation de SAFlex.

### Vers un modèle meilleur

Le premier volet concerne le modèle utilisé pour développer l'AS et plus précisément, la description des fragments structuraux. En effet, afin de représenter graphiquement les lettres structurales de notre alphabet il faut identifier les coordonnées tridimensionnelles de celles-ci. Cette identification nécessite de résoudre un système linéaire de quatre équations utilisées auparavant pour la description d'un fragment structural durant le développement de notre AS.

Sachant qu'on a déjà obtenu les quatre descripteurs de chaque lettre structurale à la fin de l'apprentissage de l'alphabet, nous voulons alors les utiliser pour obtenir les coordonnées cartésiennes des carbones alpha ( $C\alpha$ ) formant celle-ci. Nous nous retrouvons donc face à un problème, car nous n'avons que quatre équations pour six inconnues. Nous ne pourrions pas retrouver les vraies coordonnées 3D des  $C\alpha$  formant chaque lettre structurale. Nous avons donc adopté, dans le présent travail, la notion du représentant moyen de chaque LS. Pour ce faire, nous proposons de chercher dans notre jeu d'apprentissage les fragments structuraux qui se rapprochent au mieux des descripteurs de nos lettres structurales. Cette démarche revient à prendre le fragment structural qui a la plus grande vraisemblance pour chaque lettre, tout simplement cela revient à comparer les log-vraisemblance des fragments.

Par la suite, pour que notre modèle soit complet, nous devons donc passer à un modèle plus complexe sachant que deux fragments consécutifs dans une chaîne protéique partagent des descripteurs communs (le troisième descripteur d'un fragment représente le premier descripteur du fragment suivant). Dans ce cas nous ne négligeons plus les dépendances entre les fragments structuraux, ce qui ajoute plus de liens entre les fragments dans notre modèle HMM. Nous visons aussi un modèle plus parcimonieux avec distribution de départ uniforme, plus de descripteurs pour assurer une bijection entre la conformation du fragment 3D et l'espace descripteur. Enfin, des extensions des modèles devraient permettre les multi-conformations (hétéromères).

En outre, il faudrait inclure dans l'approche l'information sur la séquence (les résidus contenus dans la structure 3D) afin de mieux prédire la structure 3D des fragments structuraux. En effet, si le motif est en réalité 'A1' notre approche peut le prédire comme étant 'A3' parce que nous ne prenons pas en compte l'information de séquence. Si nous prenons en compte cette information, ce problème sera corrigé.

## **Vers un apprentissage d'un alphabet plus stable**

Le deuxième volet concerne l'apprentissage de l'alphabet structural, pour compléter le travail effectué dans cette thèse, il serait utile de régler le problème d'instabilité de l'alphabet SAFlex constaté dû à la redondance des données (chaînes multiples des protéines). Si cela est concrétisé, nous pourrions avoir un apprentissage sur plusieurs jeux de données de taille plus importante (de l'ordre de milliers de fichiers PDB) et nous obtiendrions un AS plus fin. Par ailleurs, l'obtention d'un alphabet structural final stable nécessite un apprentissage sur plusieurs jeux de données ce qui représente un travail lent et fastidieux. Par conséquent, il y a en perspective un travail très rigoureux et patient de validation de l'AS sur plusieurs jeux de données beaucoup plus grands, qui est en cours de finalisation.

## **Vers une exploitation de l'alphabet SAFlex**

Enfin le troisième volet de recherche pourrait alors être l'exploitation de l'alphabet pour construire un outil d'alignement structural qui permettra l'alignement des chaînes protéiques robuste aux données manquantes. Cet alignement permettra une comparaison rapide entre les structures tridimensionnelles qui sera utile à la découverte de nouveaux motifs ayant un intérêt thérapeutique. Une dernière perspective consiste à reconstruire les fragments manquants en 3D, une fois que les coordonnées 3D des C $\alpha$  des LS seront identifiées.

En conclusion, nous pouvons dire que le nouvel alphabet SAFlex que nous proposons offre un encodage meilleur du point de vue méthodologique et le nouvel apprentissage de la nouvelle version stable sera parcimonieux, tiendra en compte la redondance des données et gère les données manquantes et leurs incertitudes au niveau des structures PDB. Le nouvel SAFlex permettra la mise en place d'une nouvelle nomenclature. De plus la méthode sera appliquée sur des données récentes.

# Annexes



# Estimateur du maximum de vraisemblance d'une loi Gaussienne multidimensionnelle

## A.1 Méthode de maximum de vraisemblance

Selon [182] « Le terme *maximum de vraisemblance* est une méthode d'estimation de paramètres d'une population à partir d'un échantillon aléatoire. Elle s'applique quand on connaît la forme générale de distribution de la population mais quand un ou plusieurs paramètres de cette distribution sont inconnus. Elle consiste à choisir comme estimateurs des paramètres inconnus, les valeurs qui maximisent la probabilité d'obtenir l'échantillon observé. »

On considère un échantillon de  $n$  variables aléatoires  $X_1, X_2, \dots, X_n$ , où chaque variable aléatoire est distribuée selon une loi de probabilité ayant une fonction de densité  $f(x, \theta)$ , avec  $\theta \in \Omega$ , où  $\Omega$  est l'espace des paramètres.

On définit aussi une fonction  $L(\theta; x_1, x_2, \dots, x_n)$  qui est, pour la valeur  $\theta$  du paramètre, la *probabilité de la loi jointe* de l'échantillon  $(X_1, X_2, \dots, X_n)$  dans le cas discret, ou sa *densité jointe* dans le cas continu.

Pour  $x$  fixé, cette fonction est une fonction de  $\theta$  et est appelée alors fonction de « vraisemblance  $L$  »:

$$L(\theta; x_1, x_2, \dots, x_n) = f(x_1, \theta) \times f(x_2, \theta) \dots f(x_n, \theta)$$

On appelle *estimateur* du maximum de vraisemblance tout estimateur  $\hat{\theta}$  qui maximise la fonction de vraisemblance.

On note que cette fonction  $L$  peut être maximisée en annulant les premières dérivées partielles par rapport à  $\theta$  et en résolvant les équations ainsi obtenues.

On explique dans la section suivante, comment trouver l'estimateur de la fonction du maximum de vraisemblance pour une loi normale à une dimension en détail, puis on cite brièvement le calcul pour une loi normale multivariée, car le calcul utilise des notions avancées de l'algèbre linéaire, de plus ceci n'est pas l'objet de notre thèse.

### A.1.1 Calcul de l'estimateur de la fonction maximum de vraisemblance pour une loi normale à une dimension

Selon [182] la *variable aléatoire*  $X$  est distribuée selon une loi normale si elle a une *fonction de densité* de la forme :

$$\begin{aligned} \text{dnorm}(x, \mu, \sigma) &= \text{dnorm}(x, \text{mean} = \mu, \text{sd} = \sigma) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right), \quad (\sigma > 0) \\ &= f_{\mu, \sigma^2}(x) \end{aligned} \quad (\text{A.1})$$

Nous dirons donc que  $X$  suit une loi normale (appelée aussi loi des possibilités ou courbe des possibilités ou distribution gaussienne) de moyenne  $\mu$  et de variance  $\sigma^2$ . La loi normale est une *loi de probabilité continue*.

Soit la fonction  $L$  suivante:

$$L(\theta; x_1, x_2, \dots, x_n) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2}\right), \quad (\sigma > 0) \quad (\text{A.2})$$

Soit le paramètre  $\hat{\theta} = (\hat{\mu}, \hat{\sigma}^2)$  est l'estimateur du maximum de vraisemblance qui maximise la fonction  $L$ .

Il est souvent plus facile de résoudre la  $(\log L)$  au lieu de  $L$ , de plus, chacune des fonctions  $(L)$  et  $(\log L)$  étant maximales pour la même valeur de  $\theta$ .

Donc, on résout l'équation :

$$\frac{\partial \log L(\theta; x_1, x_2, \dots, x_n)}{\partial \theta} = 0$$

On commence par calculer la fonction  $\text{Log } L$ :

$$\begin{aligned} \log L &= \log \left[ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2}\right) \right] \\ &= \log(\sigma\sqrt{2\pi})^{-n} + \log \left[ \exp\left(\frac{-\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2}\right) \right] \\ &= \log(\sigma)^{-n} + \log(2\pi)^{-n/2} - \left( \frac{\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2} \right) \\ &= -n \cdot \log(\sigma) - \frac{n}{2} \cdot \log(2\pi) - \frac{\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2} \end{aligned}$$

Puis on calcule les dérivées partielles  $\frac{\partial \log L}{\partial \mu}$  et  $\frac{\partial \log L}{\partial \sigma^2}$  :

$$\begin{aligned}
\frac{\partial \log L}{\partial \mu} &= \left( -\frac{\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2} \right)' \\
&= \frac{-[2 \times (-1) (\sum_{k=1}^n (x_k - \mu))] \times (2\sigma^2)}{4\sigma^4} \\
&= \frac{4\sigma^2 \cdot (\sum_{k=1}^n (x_k - \mu))}{4\sigma^4} \\
&= \frac{1}{\sigma^2} \cdot \sum_{k=1}^n (x_k - \mu)
\end{aligned}$$

Et

$$\begin{aligned}
\frac{\partial \log L}{\partial \sigma^2} &= \left( -n \cdot \log(\sigma) - \frac{\sum_{k=1}^n (x_k - \mu)^2}{2\sigma^2} \right)' \\
&= (-n) \cdot \frac{1}{\sigma} - \left[ \frac{-4 \cdot \sigma (\sum_{k=1}^n (x_k - \mu)^2)}{(2\sigma^2)^2} \right] \\
&= -\frac{n}{\sigma} + \frac{1}{\sigma \cdot \sigma^2} \cdot \sum_{k=1}^n (x_k - \mu)^2
\end{aligned}$$

En annulant les dérivées partielles  $\frac{\partial \log L}{\partial \mu}$  et  $\frac{\partial \log L}{\partial \sigma^2}$ , on obtient les équations suivantes :

$$\frac{\partial \log L}{\partial \mu} = \frac{1}{\sigma^2} \cdot \sum_{k=1}^n (x_k - \mu) = 0 \quad (\text{A.3})$$

$$\frac{\partial \log L}{\partial \sigma^2} = -\frac{n}{\sigma} + \frac{1}{\sigma \cdot \sigma^2} \cdot \sum_{k=1}^n (x_k - \mu)^2 = 0 \quad (\text{A.4})$$

Enfin, on résout les équations (A.3) et (A.4):

De l'équation (A.3) on déduit que pour la loi normale, la moyenne arithmétique est l'estimateur du maximum de vraisemblance, voici la preuve :

$$\begin{aligned}
\frac{\partial \log L}{\partial \mu} &= 0 \\
\frac{1}{\sigma^2} \cdot \sum_{k=1}^n (x_k - \mu) &= 0 \\
\sum_{k=1}^n (x_k - \mu) &= 0 \\
\sum_{k=1}^n (x_k) - n \cdot \mu &= 0 \\
\hat{\mu} &= \frac{1}{n} \cdot \sum_{k=1}^n x_k \\
\hat{\mu} &= \bar{x}
\end{aligned}$$

Et de l'équation (A.4) on trouve que pour la loi normale, l'estimateur du maximum de vraisem-



blance de la variance est la variance empirique de l'échantillon , la preuve est donnée ci- dessous:

$$\begin{aligned} -\frac{n}{\sigma} + \frac{1}{\sigma \cdot \sigma^2} \cdot \sum_{k=1}^n (x_k - \mu)^2 &= 0 \\ \frac{1}{\sigma \cdot \sigma^2} \cdot \sum_{k=1}^n (x_k - \mu)^2 &= \frac{n}{\sigma} \\ \hat{\sigma}^2 &= \frac{1}{n} \sum_{k=1}^n (x_k - \mu)^2 \end{aligned}$$

On trouve donc que, pour la loi normale, la moyenne et la variance de l'échantillon sont les estimateurs du maximum de vraisemblance de la moyenne et de la variance de la *population* [182].

### A.1.2 Calcul l'estimateur de la fonction maximum de vraisemblance pour une loi normale multivariée à d dimensions

Soit  $X = (X_1, X_2, \dots, X_n)$  un vecteur aléatoire distribué selon une loi normale multivariée gaussienne de vecteur moyenne  $\mu$  et une matrice de covariance  $\Sigma$ , c'est-à- dire que la densité  $f_{\mu, \Sigma}$  de  $X \sim \mathcal{N}(\mu, \Sigma)$ .

La fonction du maximum de vraisemblance  $L_{\mu, \Sigma}(\theta; X_1, X_2, \dots, X_n)$  s'écrit de la manière suivante :

$$L_{\mu, \Sigma}(\theta; X_1, X_2, \dots, X_n) = \frac{1}{\sqrt{2\pi^{n \cdot d} |\Sigma|}} \exp \left( \frac{-1}{2} \sum_{k=1}^n (X_k - \mu)^T \Sigma^{-1} (X_k - \mu) \right) \quad (\text{A.5})$$

Pour trouver l'estimateur  $\hat{\theta} = (\hat{\mu}, \hat{\Sigma})$ , qui maximise la fonction log de vraisemblance ( $\log L$ ), il faut résoudre les équations obtenues en annulant les gradients de la  $\log L$  par rapport à  $\theta$ , car ça revient à trouver un vecteur  $\hat{\mu}$  et une matrice  $\hat{\Sigma}$  et non des valeurs numériques simples comme pour le cas de la loi normale à une dimension. De l'équation A.5, voici la log de vraisemblance qu'on obtient après le calcul:

$$\log L_{(\mu, \Sigma)} = \text{cste.} - \frac{n}{2} \cdot \log |\Sigma| - \frac{1}{2} \cdot \text{tr}(\Sigma^{-1} \cdot S) - \frac{1}{2} \cdot n \cdot (\bar{X} - \mu)^T \cdot \Sigma^{-1} \cdot (\bar{X} - \mu)$$

Où:

cste. : est une constante qui ne dépend pas des paramètres.

$|\Sigma|$  : le déterminant de la matrice de covariance  $\Sigma$ .

$\text{tr}(\Sigma^{-1} \cdot S)$ : la trace de la matrice  $\Sigma^{-1} S$ .

$$\text{Avec : } \bar{X} = \frac{1}{n} \sum_{k=1}^n X_k$$

$$\text{Et } S = \sum_{k=1}^n (X_k - \bar{X})(X_k - \bar{X})^T$$

Sans citer les détails pour les gradients par rapport à  $\theta$ , voici l'estimateur du maximum de vraisemblance  $\hat{\theta}$  :

$$\hat{\mu} = \bar{X}$$

$$\hat{\Sigma} = \frac{1}{n} \cdot S$$

Pour de plus amples informations sur les preuves d'obtention des dérivées et de l'estimateur, on vous incite à consulter l'article [62].

Finalement, les estimateurs du maximum de vraisemblance ne sont pas forcément uniques et cette méthode générale fournit des estimateurs qui sont très souvent les meilleurs [68, 182].



## L'algorithme EM et les HMM

Le but de cet annexe est d'introduire et de présenter les outils indispensables (EM et HMM) à la modélisation mathématique des structures tridimensionnelles, et qui sont mentionnés dans le chapitre 3. En effet, notre travail fait appel au HMM afin de modéliser les chaînes protéiques. Celles-ci sont supposées suivre un processus markovien de paramètres inconnus. Quant à l'algorithme EM, il apporte une aide précieuse pour déterminer ces paramètres inconnus du HMM. Il est employé dans des modèles appliqués à des données incomplètes. C'est ce qui motive notre choix pour celui-ci dans le présent travail.

### B.1 L'algorithme EM

L'algorithme EM (Expectation - Maximisation) proposé par [53] en 1977, est un algorithme numérique, itératif dont le but est de maximiser la vraisemblance en provenance de données incomplètes. Il est utilisé pour l'estimation des paramètres inconnus dans divers modèles tels que : les mélanges gaussiens, les chaînes de Markov cachées, les modèles mixtes, et les réseaux bayésiens. D'où son application dans différents domaines comme la segmentation d'images, la reconnaissance automatique de paroles, la génétique et données longitudinales. Son utilisation nécessite le découpage des données en deux groupes  $\mathbf{X}$  et  $\mathbf{S}$ , avec l'idée que  $\mathbf{X}$  représente les variables observées et  $\mathbf{S}$  les variables cachées (non observées). Ces données dépendent d'un vecteur de paramètres inconnu  $\Theta$  et ont une fonction de vraisemblance  $L(\Theta|\mathbf{X}, \mathbf{S}) = \mathbb{P}(\mathbf{X}, \mathbf{S}|\Theta)$ . L'estimateur optimal des paramètres *Maximum Likelihood Estimator* (MLE) de  $\Theta$  est calculé en maximisant la fonction de vraisemblance marginale des données observées. Nous pouvons donc écrire cette fonction comme suit :

$$L(\Theta|\mathbf{X}) = \mathbb{P}(\mathbf{X}|\Theta) = \sum_{\mathbf{S}} \mathbb{P}(\mathbf{X}, \mathbf{S}|\Theta) \quad (\text{B.1})$$

d'où :

$$\hat{\Theta}_{\text{MLE}} = \arg \max_{\Theta} L(\Theta|\mathbf{X})$$

La maximisation de la fonction  $L(\Theta|\mathbf{X})$  n'est pas une tâche facile, car cela revient à sommer sur toutes les configurations ( $\mathbf{S}$ ), ce qui est en général impossible. L'algorithme EM propose alors de

surmonter ce problème en estimant le paramètre  $\Theta$  optimum de la log vraisemblance des données incomplètes  $\log \mathbb{P}(S|\Theta)$  en maximisant, itérativement, l'espérance de la log vraisemblance des données complètes  $\log \mathbb{P}(X, S|\Theta)$ . Il définit donc une nouvelle fonction  $Q$  dépendante du paramètre courant  $\Theta^{old}$  et le nouveau paramètre  $\Theta$  défini par :

$$\begin{aligned} Q(\Theta|\Theta^{old}) &= \mathbb{E}_{S|X, \Theta^{old}} [\log L(\Theta|X, S)] \\ &= \sum_S \mathbb{P}(S|X, \Theta^{old}) \log \mathbb{P}(X, S|\Theta) \end{aligned} \quad (\text{B.2})$$

La mise en place de l'algorithme EM s'effectue en deux phases, tout d'abord une phase dite *d'initialisation*, celle ci est la plus délicate, elle consiste à calculer les valeurs initiales des paramètres du modèle. Ensuite vient une phase dite de *calcul*, qui se fait d'une manière itérative. Chaque itération  $i$  de ce calcul garantit une augmentation de la log vraisemblance  $\log L(\Theta^i|X)$  et une convergence du vecteur de paramètres  $\Theta^i$  vers  $\hat{\Theta}_{MLE}$ . Cette itération comporte deux étapes fondamentales :

- L'étape Expectation (E): consiste à calculer la fonction  $Q(\Theta|\Theta^{old})$ , ce qui revient à calculer les espérances conditionnelles des données non observées à partir des données observées et l'estimateur courant  $\Theta^{old}$ . En d'autres termes cette étape permet le calcul des poids des données.
- l'étape Maximization (M): maximise la fonction  $Q(\Theta|\Theta^{old})$ , ce qui revient à maximiser les espérance conditionnelles obtenues au cours de l'étape E, autrement dit ceci revient à exploiter les poids des données. Ce qui permet de résoudre le problème d'optimisation

$$\Theta^{(i)} = \arg \max_{\Theta} Q(\Theta|\Theta^{(i-1)})$$

Avec l'estimateur courant  $\Theta^{(i-1)}$  calculé durant l'itération  $(i-1)$  et  $\Theta^{(i)}$  est le nouveau estimateur calculé dans l'itération courante  $(i)$ .

La condition initiale de tout calcul est le choix du critère d'arrêt. Celui-ci va décroître au fur et à mesure des itérations successives de l'algorithme de plus en plus lentement. Le temps d'exécution de l'algorithme sera donc conditionné par le choix de ce seuil.

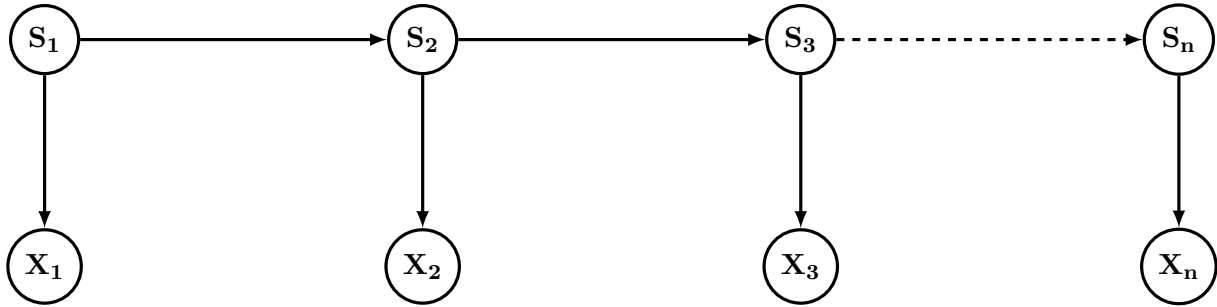
À la fin de l'exécution de l'algorithme, EM fournit un ensemble de valeurs pour des paramètres  $\Theta$  du modèle statistique.

Par ailleurs, l'utilisation de l'algorithme EM aboutit à la convergence vers un optimum (le plus souvent le plus proche local) mais peut aboutir parfois à des maxima locaux. Les résultats de l'algorithme sont toujours fonction de son initialisation. Enfin, EM est d'autant plus précis que le nombre de données inconnues est plus élevé bien qu'il converge très lentement.

## B.2 HMM

Les Chaînes de Markov Cachées (HMM), sont des outil de modélisation de séquences de données observées d'un phénomène associé à une loi de probabilité. HMM se présente sous la forme de deux séquences de variables aléatoires, l'une observable et l'autre cachée. La première séquence représente les données observées quant à la seconde séquence, elle correspond aux états cachés. HMM suppose

que toute observation à un instant  $t$  est générée par un processus possédant un état  $S_t$  caché. C'est un modèle statistique où la distribution des états cachés est supposée suivre un processus Markovien de paramètres inconnus. Lors de l'estimation de ces paramètres du HMM, les états sont traités en tant que des données manquantes. Les observations sont indépendantes les unes des autres conditionnellement aux états cachés, par contre chaque état est *dépendant* de l'état précédent, représenté par la figure B.1 suivante :



**Figure B.1 – Formalisation du problème en HMM**

$\{X_1, X_2, X_3, \dots, X_n\}$  représentent les données observées et  $\{S_1, S_2, S_3, \dots, S_n\}$  sont les états cachés.

Soient  $X_{1:n} = (X_1, X_2, X_3, \dots, X_n)$  représentant la séquence des variables observées et  $S_{1:n} = (S_1, S_2, S_3, \dots, S_n)$  la séquence des états cachés, où :

$n$  : représente la taille de l'échantillon des données observées.

Les paramètres du HMM sont les probabilités d'émission et les probabilités de transitions. La probabilité d'émission  $\mathbb{P}(X_i = x | S_i = s) = p_s(x_i)$  représente la distribution de la variable observée  $X$  sachant l'état caché  $S$ , et  $\mathbb{P}(S_i = s | S_{i-1} = r) = \pi(r, s)$  correspond aux transitions entre les états et enfin la probabilité de l'état initial  $\mu_1(s) = \mathbb{P}(S_1 = s)$ . où  $s_i, x_i$  sont pris dans les ensembles de tous les résultats possibles de  $S_i$  et  $X_i$ .

Il s'ensuit donc que la distribution jointe entre  $X$  et  $S$  du modèle illustré par la figure B.1, s'écrit de la façon suivante::

$$\mathbb{P}(X_{1:n}, S_{1:n} | \Theta) = \underbrace{\mu_1(s) \prod_{i=2}^n \pi(r, s)}_{\text{Partie Markov}} \times \underbrace{\prod_{i=1}^n \mathbb{P}(X_i | S_i)}_{\text{Partie Emission}} \quad (\text{B.3})$$

### B.3 Algorithme de Forward et Backward

Selon [96] l'algorithme forward et backward fut découvert par [34] mais généralement assigné à [7, 8]. Il constitue la base de l'estimation des paramètres HMM. Il permet ainsi de calculer efficacement les densités de probabilité afin de mettre à jour les paramètres du HMM.

Le calcul des probabilités forward et backward apporte une aide précieuse lors de l'application de l'algorithme EM. Ces probabilités sont définies comme suit :

**Définition B.3.1** La probabilité forward  $F_t(s)$  est la probabilité d'émission du HMM des variables  $X_{1:t}$  qui aboutissent à l'état  $s$  à l'instant  $t$ .

$$F_t(s) = \mathbb{P}(X_{1:t} = x_{1:t}, S_t = s) \quad \forall s \in [1; n] \quad \forall t \in [1; T]$$

**Définition B.3.2** La probabilité backward  $B_t(s)$  est la probabilité d'émission des variables  $X_{(t+1):T}$  finissant à l'état final du HMM, à partir de l'état  $s$  à l'instant  $t$ .

$$B_t(s) = \mathbb{P}(X_{(t+1):T} = x_{(t+1):T} | S_t = s) \quad \forall s \in [1; n] \quad \forall t \in [1; T]$$

$$B_T(s) = 1$$

**Théorème B.3.1** Soient la séquence d'états  $\{S_t : t \in [1; T]\}$  et la séquence de variables observées  $\{X_t : t \in [1; T]\}$ . La probabilité que  $X$  visite l'état  $s$  à l'instant  $t$  est donnée par:

$$\mathbb{P}(X = x, S_t = s) = F_t(s)B_t(s) \tag{B.4}$$

**Preuve B.3.1**

$$\begin{aligned} \mathbb{P}(X_{1:T}, S_t = s) &= \mathbb{P}(X_{1:t}, X_{(t+1):T}, S_t = s) \\ &= \mathbb{P}(X_{1:t}, X_{(t+1):T} | S_t = s) \mathbb{P}(S_t = s) \\ &= \mathbb{P}(X_{1:t} | S_t = s) \mathbb{P}(X_{(t+1):T} | S_t = s) \mathbb{P}(S_t = s) \\ &= \mathbb{P}(X_{1:t}, S_t = s) \mathbb{P}(X_{(t+1):T} | S_t = s) \\ &= F_t(s)B_t(s) \end{aligned}$$

**Corollaire B.3.1** En sommant les  $n$  états du HMM de la formule B.6, nous obtenons l'équation suivante :

$$\sum_{s=1}^n F_t(s)B_t(s) = \mathbb{P}(X = x)$$

**Théorème B.3.2** La probabilité que  $X$  a visité l'état  $r$  à l'instant  $t - 1$  et entre à l'état  $s$  à l'instant  $t$  est donnée par:

$$P(X = x, S_{t-1} = s, S_t = r) = F_{(t-1)}(s)p_r(x_t)\pi(s, r)B_t(r) \tag{B.5}$$

**Preuve B.3.2**

$$\begin{aligned}
\mathbb{P}(X_{1:T} = x_{1:T}, S_{t-1} = s, S_t = r) &= \mathbb{P}(X_{1:(t-1)}, X_{t:T}, S_{t-1} = s, S_t = r) \\
&= \mathbb{P}(X_{1:(t-1)}, S_{(t-1)} = s) \mathbb{P}(X_{t:T}, S_t = r | X_{1:(t-1)}, S_{(t-1)} = s) \\
&= \mathbb{P}(X_{1:(t-1)}, S_{(t-1)} = s) \mathbb{P}(X_{t:T}, S_t = r | S_{(t-1)} = s) \\
&= \mathbb{P}(X_{1:(t-1)}, S_{(t-1)} = s) \mathbb{P}(X_t, X_{(t+1):T}, S_t = r | S_{(t-1)} = s) \\
&= \mathbb{P}(X_{1:(t-1)}, S_{(t-1)} = s) \mathbb{P}(X_{(t+1):T} | X_t, S_t = r, S_{(t-1)} = s) \\
&\quad \mathbb{P}(X_t, S_t = r | S_{(t-1)} = s) \\
&= \mathbb{P}(X_{1:(t-1)}, S_{(t-1)} = s) \mathbb{P}(X_{(t+1):T} | S_t = r) \\
&\quad \mathbb{P}(X_t, S_t = r | S_{(t-1)} = s) \\
&= \mathbb{P}(X_{1:(t-1)}, S_{(t-1)} = s) \mathbb{P}(X_{(t+1):T} | S_t = r) \\
&\quad \mathbb{P}(S_t = r | S_{(t-1)} = s) \mathbb{P}(X_t = x_t | S_t = r) \\
&= F_{(t-1)}(s) B_t(r) \pi(s, r) p_r(x_t)
\end{aligned}$$

**Corollaire B.3.2** *Les quantités forward  $F_t(s)$  et backward  $B_t(s)$  sont calculées par récursion dans le temps par les équations suivantes:*

$$\begin{aligned}
F_t(s) &= \sum_{r=1}^n F_{(t-1)} \pi(r, s) p_s(x_t) \\
B_t(s) &= \sum_{r=1}^n B_{(t+1)}(s) \pi(s, r) p_r(x_{t+1})
\end{aligned}$$

**Preuve B.3.3** *De l'équation B.6 et l'équation B.5 nous pouvons déduire ce qui suit:*

$$\begin{aligned}
\mathbb{P}(X_{1:T}, S_t = r) &= \sum_{s=1}^n \mathbb{P}(X_{1:T}, S_{(t-1)} = s, S_t = r) \\
F_t(r) B_t(r) &= \sum_{s=1}^n F_{(t-1)}(s) B_t(r) \pi(s, r) p_r(x_t) \\
F_t(r) &= \sum_{s=1}^n F_{(t-1)}(s) \pi(s, r) p_r(x_t)
\end{aligned}$$

*De la même manière nous déduisons la quantité backward  $B_t(s)$  comme suite :*



$$\begin{aligned}
\mathbb{P}(X_{1:T}, S_t = s) &= \sum_{r=1}^n \mathbb{P}(X_{1:T}, S_t = s, S_{(t+1)} = r) \\
F_t(s)B_t(s) &= \sum_{r=1}^n F_t(s)B_{(t+1)}(r)\pi(s, r)p_r(x_{(t+1)}) \\
B_t(s) &= \sum_{r=1}^n B_{(t+1)}(r)\pi(s, r)p_r(x_{(t+1)})
\end{aligned}$$

**Corollaire B.3.3** *La probabilité que le système visite l'état  $s$  à l'instant  $t$  compte tenu de la séquence observée est :*

$$\begin{aligned}
POST_t(s) &= \mathbb{P}(S_t = s | X = x) \\
&= \frac{\mathbb{P}(X = x, S_t = s)}{\mathbb{P}(X = x)} \\
&= \frac{F_t(s)B_t(s)}{\mathbb{P}(X = x)}
\end{aligned}$$

**Corollaire B.3.4** *La probabilité que le système quitte l'état  $s$  à l'instant  $t - 1$  et entre à l'état  $r$  à l'instant  $t$  compte tenu de la séquence observée est :*

$$\begin{aligned}
\pi(s, r)(t) &= \mathbb{P}(S_{(t-1)} = s, S_t = r | X = x) \\
&= \frac{\mathbb{P}(S_{(t-1)} = s, S_t = r, X = x)}{\mathbb{P}(X = x)} \\
&= \frac{F_{(t-1)}(s)B_t(r)\pi(s, r)p_r(x_t)}{\mathbb{P}(X = x)}
\end{aligned}$$

Les quantités max-forward ( $F^{\max}$ ) et max-backward ( $B^{\max}$ ) sont définies comme qui suite :

**Définition B.3.3** *La probabilité forward  $F^{\max}_t(s)$  est la probabilité d'émission la plus probable (maximale) du HMM des variables  $X_{1:t}$  qui aboutissent à l'état  $s$  à l'instant  $t$ .*

$$F_t^{\max}(s) = \max_{S_1:S_{t-1}} \mathbb{P}(X_{1:t} = x_{1:t}, S_t = s) \quad \forall s \in [1; n] \quad \forall t \in [1; T]$$

**Définition B.3.4** *La probabilité backward  $B_t^{\max}(s)$  est la probabilité d'émission la plus probable (maximale) des variables  $X_{(t+1):T}$  finissant à l'état final du HMM, à partir de l'état  $s$  à l'instant  $t$ .*

$$B_t^{\max}(s) = \max_{S_{(t+1):T}} \mathbb{P}(X_{(t+1):T} = x_{(t+1):T} | S_t = s) \quad \forall s \in [1; n] \quad \forall t \in [1; T]$$

$$B_T(s) = 1$$

**Théorème B.3.3** Soient la séquence d'états  $\{S_t : t \in [1; T]\}$  et la séquence de variables observées  $\{X_t : t \in [1; T]\}$ . La probabilité maximale que  $X$  visite l'état  $s$  à l'instant  $t$  est donnée par:

$$\mathbb{P}(X = x, S_t = s) = F_t^{\max}(s)B_t^{\max}(s) \quad (\text{B.6})$$

Les quantités max-forward ( $F^{\max}$ ) et max-backward ( $B^{\max}$ ) ont exactement les mêmes preuves comme celles du *Forward* et *backward* définies dans cet annexe en remplaçant toutes les sommes par des max. Pour plus de détail, on vous incite à consulter [141]



# Optimisation et parallélisation des programmes informatiques

L'objectif principal de cet annexe est d'expliquer la méthode utilisée et les techniques que nous avons utilisé pour l'optimisation et la parallélisation des programmes que nous avons développé. En effet, l'implémentation des différents modules techniques, associés aux contributions scientifiques apportées par notre travail, a été confrontée aux problématiques de performance et de lenteur d'exécution.

## C.1 Optimisation et tuning de performance

### C.1.1 Méthodologie et approche

Le souci d'optimisation doit être considéré dès l'écriture du code. Il est très difficile d'optimiser un code mal architecturé et fragile. Car, parfois, on a besoin de réécrire une partie du code pour faciliter l'exploitation des différentes granularités de parallélismes offertes par la machine cible.

La démarche la plus naturelle, quand on a des soucis de performance, est de vérifier si le compilateur offre des options qui lui permettent de générer des binaires plus performants<sup>1</sup>. C'est une approche saine, car il est peu pertinent (voire inutile) de réaliser des opérations manuelles quand on peut les faire exécuter par des processus automatiques. Nous préconisons de ne faire des modifications de code que quand c'est vraiment nécessaire. Également, de favoriser la délégation des binaires optimisés à des processus automatiques et/ou semi-automatiques. De se contenter seulement de les activer et de les contrôler.

#### C.1.1.1 Profil de performance

Il est utile de dresser un profil de performance d'un code afin d'identifier les parties les plus gourmandes en temps de calcul. En effet le profil de performance permet de répondre aux questions suivantes :

---

1. la majorité des compilateurs de production (Gnu/Gcc, LLVM, Microsoft VS, Intel ICC, etc.) définissent des options génériques qui activent des ensembles prédéfinis et d'optimisations : O1, O2, O3, etc.

- Que faut-il optimiser ?
- Quelle partie du code faut-il accélérer ?

Ainsi, l’optimisation dépendra alors de la nature du code identifié<sup>2</sup> et des types de parallélismes qui pourront être exploités pour l’accélérer. En effet, la nature du code source détermine la manière de l’accélérer dans le sens où ce n’est pas possible de paralléliser un code qui ne fait qu’un calcul simple. Il n’est pas possible aussi d’optimiser l’utilisation des mémoires cache pour un code qui n’utilise quasiment pas de mémoire. Il n’est pas possible de vectoriser un code qui n’utilise pas des tableaux, etc.

#### C.1.1.1.1 Outils d’analyse de performance

Nous avons utilisé trois outils d’analyse de performance : valgrind, gnu/gprof et oprofile. La littérature offre d’autres outils<sup>3</sup> proposés par les professionnels du calcul haute performance. Mais nous nous contentons d’exposer les trois outils qui restent simple d’utilisation pour notre cas.

*Valgrind* [125] est un outil disponible sur les plateformes de type Unix (Gnu/Linux, BSD) [58]. C’est une plateforme d’instrumentation<sup>4</sup> qui agit comme une machine virtuelle. Le binaire ne s’exécute donc pas directement sur le processeur, mais il est décodé par le noyau de Valgrind. Ce dernier se charge de l’exécution du binaire et de la collecte de diverses données de debug et de performance. L’exécution du binaire pourra, par conséquent, être considérablement ralentie.

En ce qui concerne l’instrumentation mise-en-œuvre par l’outil *gnu/gprof* [66], elle se fait à la compilation et provoque un ralentissement acceptable du binaire instrumenté. L’avantage de *gnu/gprof* est sans doute son support natif<sup>5</sup> dans l’écosystème gnu (et donc gnu/linux).

Quant à *OPProfile* [40], il ne nécessite aucune instrumentation. En effet, il repose sur les compteurs de performances des microprocesseurs pour dresser un profil temps-réel des performances d’un programme en cours d’exécution.

#### C.1.1.1.2 Loi d’Amdahl

La loi d’Amdahl [4] est une formule mathématique qui permet d’apprécier l’accélération globale d’une application suite à une optimisation d’une partie de cette application :

$$\text{accélération globale} = \frac{1}{(1 - P) + \frac{P}{AI}} \quad (\text{C.1})$$

Où :

- $P$  représente la proportion de temps de la partie à accélérer.
- $AI$  représente l’accélération locale de la partie à accélérer.

Cette formule permet de guider le choix des parties des codes à optimiser suite à l’établissement d’un profil de performance. Par exemple, si un programme est construit en deux parties A et B

---

2. Nature du code identifié : c’est le code qui est gourmand en temps CPU et qui a été identifié par le profiling de performance. Sa nature (code simple, utilise des tableaux?, utilise des objets?, appel des fonctions?)

3. Intel VTune Profiler, Nvidia Visual Profiler, AMD CodeAnalyst, Google GPerfTool, Allinea MAP, etc.

4. L’instrumentation consiste à injecter du code supplémentaire dans un programme afin de permettre de l’analyser.

5. Le support natif est une expression utilisée pour qualifier les outils qui font partie d’un écosystème global. Ici il s’agit de l’univers gnu, et l’ensemble des ces outils (gprof et gcc par exemple).

qui consomment respectivement 20 et 80% du temps d'exécution. Et que nous disposons de deux possibilités d'optimisation :

- 1) accélérer la partie A pour la rendre 4 fois plus rapide ou
- 2) accélérer la partie B pour la rendre 1,5 fois plus rapide.

Les applications numériques associées aux deux scénarios sont :

$$\text{accélération globale1} = 1.17 = 1/((1 - 0.2) + (0.2/4))$$

$$\text{accélération globale2} = 1.36 = 1/((1 - 0.8) + (0.8/1.5))$$

Il est évident alors que le rendement (en accélération globale) de l'optimisation de la partie B (plus coûteuse en temps de calcul) est supérieur à celui de l'optimisation de la partie A. Il est souvent inutile de consacrer un énorme effort d'optimisation à des parties de code peu gourmandes en temps de calcul.

Dans le cadre de la présente thèse, nous utilisons la formule C.1 pour mesurer le rendement d'une tentative d'optimisation. La raison principale pour laquelle nous avons introduit cette loi est le fait de vouloir appuyer une démarche d'optimisation désormais bien connue : « il ne faut pas optimiser un code qui ne consomme pas beaucoup de temps CPU », effectivement cette loi le confirme.

## C.2 Granularités et types de parallélismes

Le profil de performance et la loi d'Amdahl (formule C.1) permettent d'identifier les parties de code qui nécessitent un effort d'optimisation en cas de besoin. Les possibilités d'accélération dépendent alors des granularités de parallélismes offertes par le code à optimiser.

### C.2.1 Parallélisme d'instructions

Améliorer le parallélisme d'instructions revient à augmenter le potentiel d'exploitation des différents pipelines du microprocesseur. Concrètement, ceci revient à réduire, dans le binaire généré, le nombre de branchements et de rassembler au mieux les instructions indépendantes.

Les compilateurs modernes sont en mesure de réduire le nombre d'instructions de branchements dans le binaire généré [60]. Ceci est obtenu grâce à des transformations automatiques ou semi-automatiques :

- Transformations automatiques : sont activées par l'utilisateur grâce à des options du compilateur.
- Transformations semi-automatiques : sont activées par l'utilisateur grâce à des directives de compilation (ou *pragma*). Ces directives sont insérées par l'utilisateur dans le code source pour indiquer les transformations qu'il souhaite appliquer sur le code.

Les transformations réalisées par les compilateurs sont toujours conservatrices. i.e. Le binaire transformé ne fait aucun choix qui peut entraîner la non conformité du binaire généré au code source. Ceci peut, parfois, empêcher l'aboutissement de certaines transformations légales<sup>6</sup> car le compilateur

---

6. les transformation légales sont des transformations qui ne violent pas les dépendances exprimées dans le code d'origine. Par exemple nous pouvons changer l'ordre de certaines instructions dans le code source sans qu'il y aura un changement de résultats. Cela est possible si les instructions ne soient pas dépendantes entre elles en opérant par exemple sur les mêmes variables.

n'arrive pas à faire des analyses précises<sup>7</sup> du code source. Dans ce genre de situation, l'utilisateur peut réécrire son code afin de faciliter les analyses conduites par le compilateur.

### C.2.2 Le parallélisme de mémoire et défaut de caches

Les défauts de mémoires caches sont l'une des sources les plus coûteuses de ralentissement. C'est également, la plus difficile à tuner, car ils dépendent fortement des caractéristiques des machines cibles : niveaux de caches absents, tailles des mémoires caches différentes entre machines, partage entre microprocesseurs, etc.

Plusieurs solutions existent pour optimiser l'utilisation des mémoires caches. Elles ne sont pour la plupart pas conduites d'une façon automatique. Nous préférons ne pas les aborder car cela sort du champ de cette étude.

Cependant, la règle la plus élémentaire concernant la maximisation de l'utilisation des mémoires caches est d'éviter la duplication des objets en mémoire quand c'est possible. Le standard C++ de 2011 (C++11) offre le moyen de transférer la responsabilité<sup>8</sup> d'une zone mémoire d'un objet composé à un autre. Les constructions par transfert, permettent d'éviter des opérations de copie inutiles tout en maintenant le même niveau de lisibilité du code source.

### C.2.3 Parallélisme de données

Le parallélisme de données [32] est exploité par les unités de calculs SIMD. Il est souvent associé aux calculs algébriques. L'exploitation de ce parallélisme varie en fonction des compilateurs. Il peut être exploité d'une façon automatique et/ou semi-automatique. Nous proposons d'explorer les outils semi-automatiques à travers l'exemple suivant : `void somme(int a[], int b[], int c[])`

```
{
    #pragma vector always
    for(int i=0; i<1024;++i)
        c[i] = a[i] + b[i];
}
```

Ce programme calcule la somme de deux tableaux (ou vecteurs) a et b, et stocke le résultat dans le vecteur c. Dans ce code nous avons rajouté la ligne `#pragma vector always` pour indiquer au compilateur<sup>9</sup> de vectoriser le code binaire généré. C'est-à-dire, faire en sorte que le code généré soit exécuté sur l'une des unités SIMD du microprocesseur cible.

La littérature offre plusieurs outils qui fonctionnent d'une façon similaire. Ci-haut, le pragma est au format compréhensible par le compilateur d'Intel. Nous pouvons trouver d'autres syntaxes (notamment celle définie par OpenMP [33]) qui produisent les mêmes effets.

La vectorisation automatique repose entièrement sur le compilateur. Autrement dit, la détection des

---

7. les transformations de code ont besoin de faire une analyse statique pour construire des graphes de dépendances entre instructions. Plus les dépendances sont précises, plus large seront les possibilités de transformations. Mais parfois, il est impossible de faire une analyse précise (donc empêcher de faire des transformations de code). Par exemple dans le cas du code suivant : I1 : `x[y[i]] = ...` ; I2 : `a = x[0]` alors l'instruction I2 dépend de I1, le tout dépend des valeurs de y.

8. C'est l'objet qui est responsable de libérer la zone mémoire dont il a réservé pour l'exécution.

9. Dans notre cas, le compilateur C++ d'Intel.

codes susceptibles d'être « vectorisable », et la mise en place de la vectorisation sont complètement déléguées au compilateur. La réussite d'une vectorisation dépend alors de la capacité d'analyse et la qualité du compilateur en question, et du code source<sup>10</sup>.

#### C.2.4 Parallélisme de threads

Les threads sont de processus allégés qui ont la particularité de partager une partie de l'état de l'application. Sur des architectures multi-cœurs, ils peuvent être utilisés pour paralléliser et accélérer un traitement intensif. On parle alors du threading et d'application multi-threadées.

Nombreux sont les outils qui facilitent la mise en place du threading. Nous listerons ici les deux approches majeures :

**Langage orienté parallélisme** : ce sont des nouveaux langages qui supportent nativement le parallélisme et les routines parallèles. Cela s'exprime par la définition de mots clés (du langage) qui expriment le parallélisme et les contraintes liées à l'exécution parallèle.

Le support du parallélisme natif dans un langage impératif comme le C++ est ajouté grâce à des extensions de langage. L'exemple le plus abouti d'une telle approche est l'extension Cilk<sup>11</sup>, implémentée par MIT [19] et adoptée par Intel.

**Une parallélisation dirigée** : dans ce cas, la parallélisation est contrôlée par des directives de compilation.

```
#pragma omp parallel for
for(int i=0; i<N;++i)
    calcul_intensif(i);
```

Quelque soit la méthode adoptée pour exploiter le parallélisme de threads, il convient d'observer l'empreinte de l'utilisation des mémoires caches. Les threads s'exécutent en parallèle sur des cœurs différents, mais partagent le même cache L3. Ils peuvent donc saturer cette mémoire provoquant des défauts de cache à répétition. Ceci pourra même dégrader les performances de l'application multi-threadée pour au final avoir une décélération.

OpenMP [33] est aujourd'hui, sans doute, la technologie de threading la plus démocratisée. Elle est supportée par la majorité des compilateurs C/C++ de production : Intel icc, gnu/gcc, llvm/clang, Microsoft msvc. C'est la raison principale pour laquelle nous l'avons choisie.

#### C.2.5 Parallélisme de tâches

Ce type de parallélisme est exploité sur des machines ayant des processeurs hétérogènes (généralement CPUs et GPUs), ou sur des systèmes faiblement couplés comme les clusters et les grilles de calculs. Les frameworks les plus connus qui permettent de résoudre les difficultés intrinsèques à ces deux types de systèmes sont : OpenCL<sup>12</sup> pour les systèmes hétérogènes et MPI<sup>13</sup> pour les systèmes faiblement couplés.

---

10. Comme pour les transformations automatiques de code, le compilateur ne réalise que des transformations conservatrices. Le code peut être écrit d'une façon qui rend le caractère vectoriel du code très difficile à déceler.

11. Cilk : <https://www.cilkplus.org/>

12. OpenCL (Open Computing Language) : <https://www.khronos.org/opencv1/>

13. MPI (Message Passing Interfaces) : <http://mpi-forum.org/>



## C.3 Conclusion

Cet annexe présente intentionnellement et d'une manière limitée les techniques d'optimisation et de parallélisation des programmes informatiques, que j'ai exploité au cours de cette thèse.

Les problématiques de performance auxquelles j'ai été confrontée à l'élaboration de mon travail m'ont conduit à la découverte du très vaste domaine qu'est le Calcul à Haute performance ou HPC <sup>14</sup> [161]. Ce chapitre présente un aperçu de l'apprentissage qui a été le mien dans cette découverte. En effet, le domaine du HPC est très vaste et son application dans les domaines bioinformatiques pourrait à elle seule, à mon avis, constituer un travail de recherche à part entière.

Ces techniques seront d'une grande utilité pour assimiler la méthodologie d'optimisation que j'ai utilisé, dans le cadre de cette thèse et qui sera détaillée dans la section "Optimisation du code" des chapitres 2 et 3. Cette section essentielle fait partie de la démarche méthodologique du développement du nouvel alphabet structural, ce qui représente la contribution principale de cette thèse.

Cependant les notions abordées dans la dernière section de cet annexe "Granularités et types de parallélismes", seront utilisées pour la parallélisation du parser qui constitue également une partie principale de notre contribution (chapitre "Un nouveau parser de fichiers PDB" à la page 11).

---

14. HPC (High performance Computing)

# Annexe **D**

## Tableaux des résultats du jeu de tests de performances pour les modules développées pour l'élaboration de l'alphabet structural SAFLex avant et après optimisation

Ci-joint les tableaux de résultats du jeu de tests de performances avant optimisation pour le module1 (voir table D.1), le module 2 (voir table D.2), module 3 (voir table D.3) et le 4ème module (voir table D.4).

n° test	Nb fichiers PDB	Nb chaînes	Nb desc	Taille fragment	Temps d'exécution (secondes)	Temps après optimisation	Temps après parallélisation
1	1	1	4	4	0,103	0,014	0,009
2	1	1	4	4	0,311	0,023	0,017
3	1	1	4	4	0,196	0,019	0,014
4	1	1	4	4	1,479	0,046	0,040
5	1	2	4	4	4,320	0,105	0,096
6	1	4	4	4	0,350	0,032	0,028
7	1	3	4	4	0,087	0,016	0,010
8	1	12	4	4	3,433	0,178	0,166
9	2	2	4	4	1,684	0,057	0,049
10	4	5	4	4	6,826	0,169	0,163
11	8	10	4	4	12,416	0,328	0,310
12	16	28	4	4	18,537	0,585	0,569
13	32	52	4	4	29,711	0,979	0,961
14	64	106	4	4	71,541	2,206	2,183
15	100	168	4	4	110,201	3,349	3,200
16	500	1213	4	4	497,017	25,648	10,623
17	8565	21842	4	4	38 934,54	506,226	151,404

**Table D.1** – Tests de performance du parser SAFlexPDB avant et après optimisation et parallélisation

n° Test	Nb fichiers PDB	Nb chaînes	Nb LS	Nb desc	Temps d'exécution (secondes)	Temps après optimisation
1	1	1	2	4	0,024	0,010
2	2	2	2	4	0,084	0,028
3	4	5	2	4	0,293	0,066
4	8	10	2	4	0,566	0,126
5	16	28	2	4	1,008	0,228
6	32	52	2	4	1,686	0,370
7	64	106	2	4	3,591	0,777
8	100	168	2	4	5,352	1,157
9	1	1	3	4	0,032	0,012
10	2	2	3	4	0,120	0,035
11	4	5	3	4	0,423	0,102
12	8	10	3	4	0,795	0,177
13	16	28	3	4	1,448	0,321
14	32	52	3	4	2,418	0,524
15	64	106	3	4	5,124	1,091
16	100	168	3	4	7,615	1,662
17	1	1	4	4	0,042	0,015
18	2	2	4	4	0,154	0,042
19	4	5	4	4	0,536	0,126
20	8	10	4	4	1,026	0,226
21	16	28	4	4	1,873	0,411
22	32	52	4	4	3,120	0,674
23	64	106	4	4	6,587	1,417
24	100	168	4	4	9,841	2,135

**Table D.2** – Tests de performances du module 2 avant et après optimisation

n° Test	Nb fichiers PDB	Nb chaînes	Nb LS	Nb desc	Temps d'exécution (secondes)	Temps après optimisation
1	1	1	2	4	0,008	0,006
2	2	2	2	4	0,027	0,019
3	4	5	2	4	0,092	0,035
4	8	10	2	4	0,094	0,036
5	16	28	2	4	0,326	0,101
6	32	52	2	4	0,494	0,171
7	64	106	2	4	1,084	0,349
8	100	168	2	4	1,596	0,520
9	1	1	3	4	0,012	0,010
10	2	2	3	4	0,048	0,017
11	4	5	3	4	0,163	0,041
12	8	10	3	4	0,301	0,086
13	16	28	3	4	0,503	0,145
14	32	52	3	4	0,811	0,239
15	64	106	3	4	1,764	0,503
16	100	168	3	4	2,618	0,734
17	1	1	4	4	0,014	0,010
18	2	2	4	4	0,066	0,021
19	4	5	4	4	0,254	0,059
20	8	10	4	4	0,470	0,109
21	16	28	4	4	0,736	0,192
22	32	52	4	4	1,190	0,314
23	64	106	4	4	2,649	0,658
24	100	168	4	4	3,955	0,979

**Table D.3** – Tests de performances du module 3 avant et après optimisation

n° Test	Nb fichiers PDB	Nb chaînes	Nb LS	Nb desc	Nb fragments	Nb itération maximale	Temps exécution (secondes)	Temps après optimisation
1	1	1	2	4	102	100	24,698	5,119
2	2	2	2	4	448	100	25,404	8,772
3	4	5	2	4	1 583	100	40,156	11,252
4	8	10	2	4	3 050	100	46,814	14,016
5	16	28	2	4	5 598	100	83,562	19,563
6	32	52	2	4	9 308	100	113,703	26,862
7	64	106	2	4	19 702	100	270,534	47,406
8	100	168	2	4	29 437	100	515,414	490,811
9	1	1	3	4	102	200	25,308	7,978
10	2	2	3	4	448	200	41,561	8,975
11	4	5	3	4	1 583	200	70,133	15,627
12	8	10	3	4	3 050	200	148,753	22,368
13	16	28	3	4	5 598	200	200,954	32,312
14	32	52	3	4	9 308	200	310,431	39,123
15	64	106	3	4	19 702	200	1 940,012	55,218
16	100	168	3	4	29 437	200	2 487,942	274,955
17	1	1	4	4	102	300	29,119	9,682
18	2	2	4	4	448	300	35,169	10,139
19	4	5	4	4	1 583	300	91,774	17,949
20	8	10	4	4	3 050	300	203,716	37,826
21	16	28	4	4	5 598	300	608,311	61,995
22	32	52	4	4	9 308	300	2 394,860	165,513
23	64	106	4	4	19 702	300	5 589,977	193,568
24	100	168	4	4	29 437	300	7 095,456	692,578

**Table D.4** – Tests de performances du module 4 avant et après optimisation

Nombre de lettres structurales	Temps exécution avant optimisation	Temps exécution après optimisation
2	5M 27S	2M 6S
3	25M 31S	7M 10S
4	1H 26M 43S	35M 44S
5	2H 48M 27S	49M 39S
6	6H 1M 8S	1H 26M 38S
7	9H 11M 41S	1H 58M 10S
8	13H 20M 29S	2H 49M 14S
9	21H 13M 57S	3H 39M 8S
10	1d 0H 42M 41S	6H 30M 49S
11	1d 9H 52M 19S	8H 13M 11S
12	2d 2H 22M 28S	13H 42M 55S
13	2d 8H 49M 40S	20H 17M 4S
14	3d 7H 59M 0S	23H 39M 27S
15	4d 12H 27M 20S	1d 6H 49M 42S
16	5d 21H 14M 0S	1d 17H 12M 33S
17	7d 15H 3M 48S	2d 5H 8M 39S
18	9d 20H 54M 20S	2d 18H 54M 54S
19	13d 21H 21M 8S	3d 10H 43M 30S
20	20d 9H 47M 11S	4d 5H 31M 19S
21	29d 16H 19M 10S	4d 22H 44M 9S
22	42d 2H 13M 29S	6d 0H 36M 15S
23	55d 6H 44M 32S	6d 14H 37M 20S
24	69d 8H 42M 27S	7d 0H 3M 19S
25	72d 0H 30M 35S	9d 16H 17M 11S
26	83d 1H 58M 35S	10d 11H 14M 1S
27	90d 20H 42M 26S	12d 12H 57M 27S

**Table D.5** – Temps d’exécution du module de génération de l’alphabet structural après regroupement des modules 2,3 et 4 dans un seul module avant et après optimisation

## Tableaux du parser SAFlex-PDB

### E.1 Les statistiques sur le jeu de données PDBSelect

Nb PDB	Nb chaines	Nb PDB des Prot Contenant RES Manq	Nb PDB Contenant des Atom C $\alpha$ Manq	NB PDB Contenant des Trous
8565	21842	6189	6424	7705

**Table E.1** – Statistiques sur le jeu de données PDBSelect

### E.2 Tableaux tests comparatifs entre BioPython, BioJava et Parser SAFlex-PDB

Test	nb PDB	SAFlex.PDB avant optim	SAFLex.PDB apres optim	SAFLex.PDB parallel	Biopython	BioJava
1	100	110.20	3.35	3.20	10.16	20.12
2	500	497.07	25.65	10.62	171.79	66.43
3	1500	1544.24	78.06	26.79	563.26	170.47
4	2500	2502.64	143.73	43.62	1002.15	278.18
5	5000	4771.54	300.73	96.03	2187.17	836.38
6	8565	8734.84	506.23	151.40	3521.34	1091.57

**Table E.2** – Tableau de tests comparatifs entre les *parsers* : BioPython, BioJava et SAFlex-PDB





# Annexe **F**

Tableau récapitulatif des librairies et AS

Equipe	Année	Nom de la librairie	Nombre protéines utilisées	Nombre fragments utilisés	Méthode d'apprentissage	Longueur du fragment	Taille librairie (fragment)	Descripteurs utilisés	Référence
[177]	1989	Building Blocks	4	426	k-moyennes	6	103	C $\alpha$ RMSD	[177]
[152]	1990	Recurrent local structural motifs	75	12 978	classification hiérarchique	4-7	4	C $\alpha$ RMSD	[152]
[139]	1992	Substructures	14	2 347	Fonction de coût	5	113	Distance linéaire et angle $\alpha$	[139]
[186]	1993	Structural Building Blocks	74	13 114	AutoANN et k-moyennes	7	6	Distance entre C $\alpha$ angles dièdres et de valence	[186]
[159]	1996	Local structural motifs	136	24 239	carte topologique de Kohonen	9	100	Angles dièdres	[159]
[64]	1997	Structural Building blocks	116	23 335	AutoANN et k-moyennes	7	6	Distance entre C $\alpha$ angles dièdres et de valence	[64]
[24]	1998	I-sites	471	/	k-moyennes	3-19	82 clusters groupés en 13 classes de motifs	Profil de séquence et RMSD	[24]
[31]	1999	Short structural Building Blocks	100	19 137	HMM	4	12	Distances entre C $\alpha$	[31]
[123]	2000	Oligons	75 chaînes	10 936	Classification itérative	4-7	28, 202, 932, 2 561	C $\alpha$ RMSD	[123]

Equipe	Année	Nom de la librairie	Nombre protéines utilisées	Nombre fragments utilisés	Méthode d'apprentissage	Longueur du fragment	Taille librairie	Descripteurs utilisés	Référence
[48]	2000	Protein Blocks	342	87 996	Classification non supervisée (the self-organised learning of the Kohonen network (SOM))	5	16	Angles dièdres	[48]
[25]	2000	HMMSTR (nouveau I-Sites)	691	/	k-moyennes et HMM (utilisé pour détecter les chevauchements et les corrélations entre les fragments de I-Sites	3-19	250	Profiles de séquence et RMSD	[25]
[100]	2002	Fragment Libraries	200 domaines	/	k-moyennes et recuit simulé	4-7	4 librairies 8 949, 7 123, 5 910 et 5 029 pour 4, 5, 6, 7 résidus	C $\alpha$ cRMS	[100]
[88]	2003	centroids	790	156 643	hypercosine clustering method	7	28	Hypercosine C $\alpha$	[88]
[28]	2004	HMM-SA	1 429	332 493	HMM	4	27	Distances entre C $\alpha$	[28]

Equipe	Année	Nom de la librairie	Nombre protéines utilisées	Nombre fragments utilisés	Méthode d'apprentissage	Longueur du fragment	Taille librairie	Descripteurs utilisés	Référence
[45, 63]	2005	Protein Blocks	1 407 chaînes	/	(the self-organised learning of the Kohonen network (SOM)) et recuit simulé	5	16	Angles dièdres	[63], [45]
[10]	2006	LPS	675 et 1 401	139 503 et 251 497	Modèle de la protéine hybride	11	120	Blocs protéiques et C $\alpha$ RMSD	[10]
[155]	2006	Structural representatives	1 999	/	« leader clustering algorithm » et k-moyennes	7	27	Matrice de distance entre C $\alpha$	[155]
[172]	2007	SADB (structural alphabet data base)	1 348	225 523	méthode du plus proche voisin	5	23	Angles $\kappa$ et $\alpha$ et C $\alpha$	[172]
[57]	2007	/	1 400 chaînes	179 463	k-moyennes	7	28	C $\alpha$ RMSD	[57]
[183]	2008	Protein Folding Shape Code (PFSC)	200 paires	/	méthode PFSC	5	27	2 Angles de torsion et distance C $\alpha$	[183]
[102]	2008	centroids	/	20 953 584	carte topologique de Kohonen et k-moyennes	5	18	Angles dièdres	[102]
[56]	2008	Building Blocks	1 219 chaînes	/	méthode du plus proche voisin	4-7	43, 50, 51, 36	Angles dièdres, profils de séquences	[56]

Equipe	Année	Nom de la librairie	Nombre protéines utilisées	Nombre fragments utilisés	Méthode d'apprentissage	Longueur du fragment	Taille librairie	Descripteurs utilisés	Référence
[5]	2008	BriX	1 261	260 000	classification hiérarchique	4-14	Plus de 1 000 fragments par longueur	RMSD	[5]
[180]	2008	Conformational Letter (CLe)	27 181	/	mélanges gaussiens	4	17	angles dièdres	[180]
[106]	2009	Fragment Library	2 930	/	SVM	4	20	C $\alpha$ cRMS	[106]
[132]	2010	M32K25	1 830	325 923	Classification basée sur la densité et la méthode OPTICS	4	25	angle de torsion et distances aRMSD	[132]
[23]	2010	FragBag	2 928	/	Méthode Bag Of Words (BOW)	11	400	distance C $\alpha$	[23]
[94]	2011	Dynamic Fragment Library	4 824	/	HHFrag	6-21	/	C $\alpha$ RMSD	[94]
[173]	2013	USA (Units of Structural Alphabet)	1 603 chaînes	5 525	réseau de neurones	5	23	Angles $\kappa$ et $\alpha$ et C $\alpha$	[173]
[51]	2015	FLib librairies	/	/	Extraction aléatoire et extraction exhaustive	6-20	à peu près 33	angle de torsion et Distance RMSD	[51]

**Table F.1 – Récapitulatif des différentes librairies et des alphabets structuraux**

Ce tableau est tiré de [127] et mis à jour.



# Annexe **G**

## Résultats de SAFlex

### **G.1 Résultats de SAFlex.V1**

Ci-joint les tableaux de résultats SAFlex.V1.



	A1	A2	A3	A4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	B1	B2	B3	B4	B5
A1	70.49	21.11		*	3.01	1.63								2.89													
A2			65.06			13.09			6.03	*	2.66	*		4.47	3.08					3.15	*	1.08					
A3	51.82	27.06		3.46	9.63	5.07								1.91	*						*						
A4		7.15		74.90	2.95	4.11			*		*			6.66	*				1.37		*	*					
C1	11.38	11.71		1.75	20.75	15.15	*		13.95		2.53	1.15		5.15	7.48				*	5.55	1.53	1.06					
C2		0.65	14.95	1.65		13.54	*		10.71	3.59	6.97	4.93		7.23	7.95				1.27	15.21	3.46	5.80			1.02		
C3	0.63	3.77	3.76	1.07	16.65	8.82	1.21		10.98	7.17	4.63	20.50			4.55				2.77	2.60	7.39	1.88			1.19	*	
C4						*	13.49	12.93	*	7.92			12.88				7.33	4.16	1.30		*		1.48	11.44	15.15	8.20	2.02
C5							19.03	15.41		6.26	*		4.80			4.67	2.56	5.50	1.63			7.11	15.16	5.51	5.04	7.12	
C6		1.73	3.62	*	11.78	9.84	2.03		8.72	4.06	4.26	25.73		2.02	5.64		*	0.65	2.88		5.31	5.18		*	2.11	2.62	
C7							11.43	18.20	*	6.47			5.17			1.19	6.34	4.51	*			0.68	4.20	20.04	11.84	4.58	4.03
C8							12.64	9.24		2.08			8.44			8.39	1.74	8.07	2.22			21.47	7.91		1.84	15.98	
C9	10.30	13.70	7.08	4.34	30.51	9.46	*		4.96	1.69	3.95	5.52		*	1.46				*		3.32	1.77					
C10			*			*			12.96	3.89	53.78	1.22		*	1.10				3.35		6.71	16.17					
C11	1.22	1.91	13.79	1.57	17.61	11.42	*		10.83	6.14	4.46	7.84		1.73	8.61				1.44	3.64	3.26	3.51				*	
C12						*	9.10	7.67		2.75			4.32			19.20	10.05	9.46	6.88			7.49	8.40	4.16	5.49	4.72	
C13			1.51		5.41	2.58			18.47	3.54	27.24	2.49		*	8.09				14.06	*	5.73	9.30					
C14	1.12		5.28	*	5.81	11.66	8.87	9.75	3.13	3.88		2.88	6.44		1.92	*	14.91		*	4.22	2.32	*		3.38	6.80	4.75	
C15			*	*	2.03	2.26			16.02	3.25	13.96	3.67		3.24	3.51		1.18		28.34	2.21	11.09	8.34					
C16			*	*		9.44	2.28		11.47	10.97	3.04	8.09		14.55	6.78		1.26			17.91	4.31	3.32		*	3.95	1.03	*
C17						*	8.54	7.13	*	5.46	*		5.13		*	23.22	9.05	7.45	5.53			3.43	8.10	7.12	3.75	4.60	
C18	*	4.04	9.78	*	12.52	13.55	4.12		10.69	4.89	4.06	12.30		4.02	6.66		2.75			2.29	6.28					*	
B1							4.46	2.11		5.56			1.77			2.58	1.29	1.12	*			15.19	8.76	32.99	11.87	11.44	
B2							8.94	8.77		9.69	*		10.41			2.02	2.64	5.01	*			6.96	10.76	19.03	9.85	5.16	
B3							6.85	6.52		3.20			6.54			2.85	2.55	4.01	1.16			22.70	12.87	5.29	5.83	19.64	
B4							7.54	15.57		*			5.88			7.59	2.93	8.58	2.55			20.44	16.46		1.93	9.69	
B5							5.66	5.38		2.00			1.66			3.58	2.92	3.63	*			25.44	12.26		8.53	27.98	

Table G.1 – La matrice de transition de SAFlex.

États	$\mu$	$\Sigma$
<b>A1</b>	$\begin{bmatrix} 5.43 & 5.09 & 5.42 & 2.94 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0.01 & 0.00 & -0.00 \\ 0.01 & 0.02 & 0.00 & 0.01 \\ 0.00 & 0.00 & 0.01 & -0.00 \\ -0.00 & 0.01 & -0.00 & 0.02 \end{bmatrix}$
<b>A2</b>	$\begin{bmatrix} 5.41 & 5.23 & 5.61 & 2.86 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0.01 & -0.00 & -0.00 \\ 0.01 & 0.04 & 0.00 & 0.03 \\ -0.00 & 0.00 & 0.01 & -0.00 \\ -0.00 & 0.03 & -0.00 & 0.04 \end{bmatrix}$
<b>A3</b>	$\begin{bmatrix} 5.62 & 5.25 & 5.42 & 2.87 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0.01 & 0.00 & -0.00 \\ 0.01 & 0.04 & 0.01 & 0.03 \\ 0.00 & 0.01 & 0.01 & -0.01 \\ -0.00 & 0.03 & -0.01 & 0.04 \end{bmatrix}$
<b>A4</b>	$\begin{bmatrix} 5.39 & 5.09 & 5.38 & 2.92 \end{bmatrix}$	$\begin{bmatrix} 0.03 & 0.01 & -0.01 & -0.01 \\ 0.01 & 0.06 & 0.01 & 0.03 \\ -0.01 & 0.01 & 0.03 & -0.01 \\ -0.01 & 0.03 & -0.01 & 0.06 \end{bmatrix}$
<b>C1</b>	$\begin{bmatrix} 5.40 & 5.58 & 5.42 & 3.39 \end{bmatrix}$	$\begin{bmatrix} 0.02 & 0.02 & -0.00 & -0.00 \\ 0.02 & 0.06 & 0.01 & 0.03 \\ -0.00 & 0.01 & 0.02 & -0.01 \\ -0.00 & 0.03 & -0.01 & 0.03 \end{bmatrix}$
<b>C2</b>	$\begin{bmatrix} 5.59 & 5.49 & 5.78 & 2.58 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0.06 & -0.02 & 0.02 \\ 0.06 & 0.19 & 0.03 & 0.11 \\ -0.02 & 0.03 & 0.04 & -0.01 \\ 0.02 & 0.11 & -0.01 & 0.16 \end{bmatrix}$
<b>C3</b>	$\begin{bmatrix} 6.57 & 8.96 & 5.58 & -2.19 \end{bmatrix}$	$\begin{bmatrix} 0.14 & 0.06 & 0.02 & -0.08 \\ 0.06 & 0.10 & 0.06 & 0.07 \\ 0.02 & 0.06 & 0.06 & -0.00 \\ -0.08 & 0.07 & -0.00 & 0.44 \end{bmatrix}$
<b>C4</b>	$\begin{bmatrix} 6.72 & 9.12 & 6.41 & -3.31 \end{bmatrix}$	$\begin{bmatrix} 0.10 & 0.08 & 0.00 & -0.01 \\ 0.08 & 0.13 & 0.04 & 0.04 \\ 0.00 & 0.04 & 0.05 & 0.03 \\ -0.01 & 0.04 & 0.03 & 0.04 \end{bmatrix}$
<b>C5</b>	$\begin{bmatrix} 5.66 & 8.07 & 6.70 & 2.96 \end{bmatrix}$	$\begin{bmatrix} 0.08 & 0.09 & 0.01 & -0.01 \\ 0.09 & 0.32 & 0.07 & -0.09 \\ 0.01 & 0.07 & 0.09 & -0.11 \\ -0.01 & -0.09 & -0.11 & 0.16 \end{bmatrix}$

<b>C6</b>	$\begin{bmatrix} 6.21 & 9.21 & 5.77 & 0.27 \end{bmatrix}$	$\begin{bmatrix} 0.15 & 0.11 & 0.04 & -0.02 \\ 0.11 & 0.12 & 0.09 & -0.01 \\ 0.04 & 0.09 & 0.09 & 0.03 \\ -0.02 & -0.01 & 0.03 & 0.57 \end{bmatrix}$
<b>C7</b>	$\begin{bmatrix} 5.66 & 8.95 & 6.54 & 2.09 \end{bmatrix}$	$\begin{bmatrix} 0.06 & 0.05 & -0.00 & 0.01 \\ 0.05 & 0.10 & 0.03 & -0.13 \\ -0.00 & 0.03 & 0.08 & -0.02 \\ 0.01 & -0.13 & -0.02 & 0.40 \end{bmatrix}$
<b>C8</b>	$\begin{bmatrix} 5.70 & 7.26 & 7.02 & 0.88 \end{bmatrix}$	$\begin{bmatrix} 0.08 & 0.13 & 0.01 & -0.01 \\ 0.13 & 0.40 & 0.10 & -0.06 \\ 0.01 & 0.10 & 0.05 & -0.10 \\ -0.01 & -0.06 & -0.10 & 1.07 \end{bmatrix}$
<b>C9</b>	$\begin{bmatrix} 6.71 & 8.27 & 5.47 & -3.56 \end{bmatrix}$	$\begin{bmatrix} 0.09 & 0.03 & -0.00 & -0.03 \\ 0.03 & 0.09 & 0.04 & 0.03 \\ -0.00 & 0.04 & 0.04 & 0.00 \\ -0.03 & 0.03 & 0.00 & 0.04 \end{bmatrix}$
<b>C10</b>	$\begin{bmatrix} 5.55 & 7.74 & 5.60 & -3.31 \end{bmatrix}$	$\begin{bmatrix} 0.06 & 0.04 & -0.01 & -0.00 \\ 0.04 & 0.13 & -0.01 & 0.11 \\ -0.01 & -0.01 & 0.05 & -0.03 \\ -0.00 & 0.11 & -0.03 & 0.14 \end{bmatrix}$
<b>C11</b>	$\begin{bmatrix} 5.60 & 6.71 & 5.58 & 3.69 \end{bmatrix}$	$\begin{bmatrix} 0.08 & 0.12 & 0.03 & -0.00 \\ 0.12 & 0.36 & 0.10 & 0.01 \\ 0.03 & 0.10 & 0.08 & -0.01 \\ -0.00 & 0.01 & -0.01 & 0.01 \end{bmatrix}$
<b>C12</b>	$\begin{bmatrix} 6.89 & 8.94 & 6.76 & -0.48 \end{bmatrix}$	$\begin{bmatrix} 0.09 & 0.10 & -0.02 & -0.17 \\ 0.10 & 0.84 & 0.25 & 0.06 \\ -0.02 & 0.25 & 0.13 & 0.17 \\ -0.17 & 0.06 & 0.17 & 3.74 \end{bmatrix}$
<b>C13</b>	$\begin{bmatrix} 6.47 & 5.92 & 5.56 & 0.53 \end{bmatrix}$	$\begin{bmatrix} 0.10 & 0.18 & 0.00 & 0.03 \\ 0.18 & 0.45 & 0.06 & 0.20 \\ 0.00 & 0.06 & 0.04 & 0.04 \\ 0.03 & 0.20 & 0.04 & 1.30 \end{bmatrix}$
<b>C14</b>	$\begin{bmatrix} 6.87 & 8.28 & 6.03 & -3.44 \end{bmatrix}$	$\begin{bmatrix} 0.07 & 0.05 & -0.03 & -0.01 \\ 0.05 & 0.28 & 0.15 & 0.03 \\ -0.03 & 0.15 & 0.16 & 0.06 \\ -0.01 & 0.03 & 0.06 & 0.07 \end{bmatrix}$

<b>C15</b>	$\begin{bmatrix} 6.03 & 6.85 & 5.64 & -0.63 \end{bmatrix}$	$\begin{bmatrix} 0.44 & 0.50 & -0.02 & 0.43 \\ 0.50 & 1.87 & 0.23 & 0.79 \\ -0.02 & 0.23 & 0.22 & -0.11 \\ 0.43 & 0.79 & -0.11 & 6.57 \end{bmatrix}$
<b>C16</b>	$\begin{bmatrix} 5.78 & 5.68 & 6.07 & 1.46 \end{bmatrix}$	$\begin{bmatrix} 0.08 & 0.09 & -0.01 & 0.02 \\ 0.09 & 0.26 & 0.06 & 0.06 \\ -0.01 & 0.06 & 0.06 & -0.04 \\ 0.02 & 0.06 & -0.04 & 0.36 \end{bmatrix}$
<b>C17</b>	$\begin{bmatrix} 5.66 & 8.91 & 6.66 & -1.46 \end{bmatrix}$	$\begin{bmatrix} 0.12 & 0.10 & 0.02 & -0.04 \\ 0.10 & 0.28 & -0.02 & 0.29 \\ 0.02 & -0.02 & 0.14 & -0.08 \\ -0.04 & 0.29 & -0.08 & 1.15 \end{bmatrix}$
<b>C18</b>	$\begin{bmatrix} 5.69 & 8.09 & 5.67 & 3.09 \end{bmatrix}$	$\begin{bmatrix} 0.06 & 0.03 & 0.00 & 0.01 \\ 0.03 & 0.14 & 0.04 & -0.13 \\ 0.00 & 0.04 & 0.07 & 0.00 \\ 0.01 & -0.13 & 0.00 & 0.22 \end{bmatrix}$
<b>B1</b>	$\begin{bmatrix} 6.87 & 10.06 & 6.51 & -1.41 \end{bmatrix}$	$\begin{bmatrix} 0.06 & 0.06 & 0.04 & -0.03 \\ 0.06 & 0.10 & 0.08 & 0.02 \\ 0.04 & 0.08 & 0.08 & 0.00 \\ -0.03 & 0.02 & 0.00 & 0.26 \end{bmatrix}$
<b>B2</b>	$\begin{bmatrix} 6.71 & 9.64 & 6.50 & -2.60 \end{bmatrix}$	$\begin{bmatrix} 0.10 & 0.08 & 0.02 & -0.03 \\ 0.08 & 0.11 & 0.06 & 0.04 \\ 0.02 & 0.06 & 0.06 & 0.03 \\ -0.03 & 0.04 & 0.03 & 0.12 \end{bmatrix}$
<b>B3</b>	$\begin{bmatrix} 6.39 & 9.93 & 6.75 & -1.07 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0.05 & 0.02 & -0.01 \\ 0.05 & 0.06 & 0.04 & 0.04 \\ 0.02 & 0.04 & 0.05 & 0.03 \\ -0.01 & 0.04 & 0.03 & 0.25 \end{bmatrix}$
<b>B4</b>	$\begin{bmatrix} 6.48 & 10.17 & 7.09 & 0.66 \end{bmatrix}$	$\begin{bmatrix} 0.08 & 0.08 & 0.01 & 0.05 \\ 0.08 & 0.08 & 0.02 & 0.01 \\ 0.01 & 0.02 & 0.02 & 0.01 \\ 0.05 & 0.01 & 0.01 & 0.31 \end{bmatrix}$
<b>B5</b>	$\begin{bmatrix} 6.80 & 10.35 & 6.85 & -0.25 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0.06 & 0.03 & -0.00 \\ 0.06 & 0.07 & 0.05 & 0.01 \\ 0.03 & 0.05 & 0.05 & 0.01 \\ -0.00 & 0.01 & 0.01 & 0.20 \end{bmatrix}$

**Table G.2** – Les paramètres des lettres structurales de l’alphabet SAFlex.V1

## G.2 Résultats de SAFlex. $\beta$

Ci-joint les tableaux de résultats SAFlex. $\beta$  :

<b>LS</b>	A1	A2	A3	B1	B2	B3	B4	B5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
<b>d1</b>	5.45	5.50	5.42	6.81	6.62	6.32	6.88	6.54	5.63	5.65	6.78	6.63	5.67	6.52	5.72	6.10	6.45	6.94	5.58	6.74	5.61	5.84	5.78	5.73	6.74	5.59	7.59
<b>d2</b>	5.12	5.25	5.63	10.23	9.34	9.81	9.88	10.23	8.20	5.47	8.91	8.68	9.07	9.16	7.34	9.16	5.90	8.11	7.84	8.08	6.93	6.48	8.18	7.56	9.44	5.81	10.31
<b>d3</b>	5.45	5.52	5.42	6.69	6.44	6.71	6.47	7.03	6.65	5.94	6.40	5.53	6.60	5.62	7.02	5.79	5.57	6.22	5.62	5.48	5.59	6.00	5.68	6.75	7.03	5.54	7.50
<b>d4</b>	2.93	2.85	3.41	-0.71	-2.98	-1.27	-2.21	0.33	3.01	1.94	-3.38	-3.00	1.78	-1.30	1.04	0.63	0.75	-2.16	-3.22	-3.70	3.74	2.97	3.08	-1.64	1.75	-3.18	-0.30

**Table G.3** – Tableau récapitulatif des moyennes et des occurrences des lettres structurales

	A1	A2	A3	B1	B2	B3	B4	B5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
<b>A1</b>	0.82	0.12	0.02	.	.	.	.	.	.	0.01	.	.	.	.	.	.	.	.	0.03	.	.	.	.	.	.	.	.
<b>A2</b>	0.15	0.54	0.04	.	.	.	.	.	0.06	0.10	.	.	0.02	.	.	.	.	.	0.04	.	0.03	0.01	0.02	0.01	.	.	.
<b>A3</b>	0.12	0.17	0.25	.	.	.	.	.	0.16	0.12	.	.	0.01	.	0.01	0.01	.	.	0.05	.	0.07	0.02	0.01	0.02	.	.	.
<b>B1</b>	.	.	.	0.33	0.08	0.11	0.15	0.17	.	.	0.02	0.01	.	0.05	.	0.01	0.01	0.02	.	0.01	.	.	.	.	0.02	.	.
<b>B2</b>	.	.	.	0.04	0.13	0.17	0.02	0.13	.	.	0.07	0.12	0.01	0.05	.	0.06	0.07	0.01	.	0.08	.	0.01	.	.	0.02	.	.
<b>B3</b>	.	.	.	0.23	0.08	0.06	0.10	0.13	.	.	0.05	0.05	.	0.07	.	0.02	0.05	0.03	.	0.09	.	0.01	.	.	0.02	.	.
<b>B4</b>	.	.	.	0.15	0.09	0.23	0.07	0.19	.	.	0.03	0.04	.	0.07	.	0.05	0.01	0.01	.	0.04	.	.	.	.	0.02	.	.
<b>B5</b>	.	.	.	0.18	0.10	.	0.29	0.04	.	.	0.12	0.06	.	0.03	.	0.01	0.03	0.06	.	0.04	.	.	.	.	0.04	.	.
<b>C1</b>	.	.	.	0.08	0.21	0.06	0.11	0.07	.	.	0.08	0.09	.	0.13	.	0.03	0.03	0.03	.	0.03	.	.	.	.	0.03	.	.
<b>C2</b>	.	0.05	0.01	.	.	0.03	.	0.01	0.11	0.23	.	.	0.03	.	0.09	0.08	.	.	0.13	.	0.05	0.08	0.06	0.04	.	.	.
<b>C3</b>	.	.	.	.	0.16	0.11	0.03	0.08	.	.	0.11	0.17	.	0.06	.	0.03	0.14	.	.	0.07	.	0.01	.	.	0.02	.	.
<b>C4</b>	0.03	0.17	0.30	.	.	0.02	.	.	0.07	0.02	.	.	0.08	.	0.15	0.06	.	.	0.01	.	0.02	0.02	0.01	0.03	.	.	.
<b>C5</b>	.	.	.	0.08	0.21	0.12	0.09	0.07	.	.	0.11	0.07	.	0.08	.	0.03	0.05	0.03	.	0.03	.	.	0.01	.	0.01	.	0.01
<b>C6</b>	0.02	0.09	0.18	.	.	0.06	.	0.01	0.09	0.03	.	.	0.04	0.02	0.22	0.04	.	.	0.02	.	0.04	0.07	0.02	0.07	.	0.01	.
<b>C7</b>	.	.	.	0.24	0.07	.	0.17	0.04	.	.	0.12	0.11	.	0.06	.	0.01	0.02	0.07	.	0.05	.	.	.	.	0.04	.	.
<b>C8</b>	.	0.12	0.15	.	0.02	0.04	.	0.01	0.05	0.02	.	.	0.03	0.02	0.22	0.06	0.02	.	0.03	.	0.04	0.06	0.04	0.06	.	0.01	.
<b>C9</b>	.	0.01	0.06	.	.	0.04	.	.	0.18	0.01	.	.	0.31	.	0.02	0.06	.	.	0.01	.	0.10	0.02	0.07	0.01	.	0.10	.
<b>C10</b>	0.01	0.06	0.05	0.07	0.07	0.06	0.02	0.04	0.02	0.03	0.05	0.03	0.02	0.04	0.02	0.02	0.15	0.04	.	0.02	0.03	0.03	0.02	0.02	0.02	0.05	.
<b>C11</b>	.	.	.	.	.	0.04	.	.	0.15	0.01	.	.	0.49	.	0.02	0.05	.	.	.	.	0.02	0.02	0.15	0.01	.	0.04	.
<b>C12</b>	0.20	0.26	0.29	.	.	0.01	.	.	0.04	0.04	.	.	0.03	.	0.03	0.02	.	.	0.01	.	0.01	0.04	0.01	0.02	.	.	.
<b>C13</b>	0.02	0.23	0.20	.	.	0.01	.	.	0.07	0.05	.	.	0.04	0.01	0.06	0.04	.	.	0.03	.	0.10	0.06	0.03	0.05	.	.	.
<b>C14</b>	0.02	0.06	0.04	.	0.05	0.06	.	0.07	0.07	0.03	0.01	0.01	0.09	0.08	0.08	0.07	0.03	0.01	0.02	.	0.04	0.04	0.08	0.04	0.01	0.03	.
<b>C15</b>	0.05	0.18	0.15	.	0.01	0.02	.	.	0.03	0.07	0.01	0.03	0.04	0.01	0.11	0.04	0.04	.	0.05	.	0.04	0.03	0.05	0.04	.	0.01	.
<b>C16</b>	.	.	.	0.02	0.05	0.03	0.07	.	0.02	.	0.04	0.04	0.01	0.03	0.02	0.02	0.09	0.30	.	0.03	.	0.05	0.03	.	0.11	0.03	0.02
<b>C17</b>	.	.	.	0.04	0.01	0.01	0.05	0.01	.	.	0.13	0.07	.	.	.	0.01	0.05	0.56	.	0.02	.	0.01	.	.	.	.	0.02
<b>C18</b>	.	.	.	.	.	0.03	.	.	0.46	0.01	.	.	0.22	.	0.02	0.03	.	.	.	.	0.04	0.02	0.07	0.02	.	0.08	.
<b>C19</b>	.	0.01	0.03	0.02	0.01	0.03	0.03	0.05	.	.	0.01	.	0.06	.	0.02	0.01	0.03	.	.	.	.	0.01	.	0.08	.	0.01	0.60

Table G.4 – Matrice de transitions entre les lettres structurales

	Prob de SORTIE			Prob de PROVENANCE			Caractéristique
	Helice	Brin	Boucle	Helice	Brin	Boucle	
<b>A1</b>	0.96	0.00	0.04	0.96	0.00	0.04	Cœur hélice
<b>A2</b>	0.73	0.00	0.27	0.78	0.00	0.22	Bord hélice
<b>A3</b>	0.54	0.00	0.46	0.41	0.00	0.59	Bord hélice relié à des boucles
<b>B1</b>	0.00	0.85	0.15	0.00	0.77	0.23	Cœur Brind
<b>B2</b>	0.00	0.50	0.50	0.00	0.49	0.51	Bord Brind relié à des boucles
<b>B3</b>	0.00	0.61	0.39	0.00	0.60	0.39	Bord Brind
<b>B4</b>	0.00	0.72	0.28	0.00	0.69	0.31	Cœur Brind
<b>B5</b>	0.00	0.61	0.39	0.00	0.75	0.25	Bord Brind
<b>C1</b>	0.00	0.54	0.46	0.40	0.00	0.60	Boucle fin hélice et entrée brin
<b>C2</b>	0.06	0.04	0.91	0.59	0.00	0.41	Boucle courte fin hélice et entrée boucle
<b>C3</b>	0.00	0.38	0.62	0.00	0.46	0.54	Boucle fin brin
<b>C4</b>	0.50	0.02	0.48	0.00	0.44	0.56	Boucle entrée hélice et fin brin
<b>C5</b>	0.00	0.57	0.43	0.07	0.01	0.92	Boucle entrée brin
<b>C6</b>	0.29	0.07	0.64	0.00	0.47	0.53	Boucle fin brin
<b>C7</b>	0.00	0.52	0.48	0.04	0.00	0.95	Boucle entrée brin
<b>C8</b>	0.27	0.07	0.66	0.01	0.30	0.69	Boucle
<b>C9</b>	0.07	0.04	0.89	0.00	0.38	0.62	Boucle
<b>C10</b>	0.12	0.26	0.62	0.00	0.31	0.69	Boucle
<b>C11</b>	0.00	0.04	0.96	0.59	0.00	0.41	Boucle fin hélice et entrée boucle
<b>C12</b>	0.75	0.01	0.25	0.00	0.64	0.36	Boucle entrée hélice et fin brin
<b>C13</b>	0.45	0.01	0.54	0.38	0.00	0.61	Boucle entrée hélice
<b>C14</b>	0.11	0.17	0.72	0.14	0.07	0.78	Boucle
<b>C15</b>	0.38	0.03	0.59	0.16	0.03	0.81	Boucle
<b>C16</b>	0.00	0.17	0.83	0.17	0.01	0.81	Boucle
<b>C17</b>	0.00	0.12	0.88	0.00	0.54	0.46	Boucle fin brin
<b>C18</b>	0.00	0.03	0.97	0.01	0.01	0.99	Boucle courte
<b>C19</b>	0.04	0.13	0.83	0.02	0.16	0.82	Boucle

**Table G.5** – Tableau des probabilités de sortie et de provenance des lettres structurales



	A1	A2	A3	B1	B2	B3	B4	B5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
A1	0.82	0.11	0.04	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.02	.	.	.	.	.	.	.
A2	0.17	0.54	0.08	.	.	.	.	.	.	0.01	.	0.04	.	0.02	.	0.02	.	0.01	.	0.04	0.03	0.01	0.02	.	.	.	.
A3	0.08	0.08	0.25	.	.	.	.	.	.	.	.	0.16	.	0.09	.	0.07	0.02	0.02	.	0.10	0.07	0.01	0.04	.	.	.	.
B1	.	.	.	0.33	0.03	0.18	0.11	0.12	0.05	.	.	0.04	.	0.10	.	.	.	0.02	.	.	.	.	.	.	0.01	.	.
B2	.	.	.	0.11	0.13	0.08	0.08	0.09	0.17	.	0.10	.	0.13	.	0.03	0.01	.	0.03	.	.	.	0.01	.	0.01	.	.	.
B3	.	.	.	0.14	0.18	0.06	0.22	.	0.05	0.02	0.07	0.01	0.07	0.04	.	0.02	0.02	0.03	0.02	.	.	0.02	0.01	0.01	.	.	.
B4	.	.	.	0.21	0.03	0.11	0.07	0.28	0.10	.	0.02	.	0.06	.	0.10	.	.	0.01	.	.	.	.	.	0.02	0.01	.	.
B5	.	.	.	0.24	0.15	0.14	0.19	0.04	0.06	.	0.05	.	0.05	0.01	0.02	.	.	0.02	.	.	.	0.02	.	.	.	.	.
C1	0.01	0.18	0.21	.	.	.	.	.	.	0.10	.	0.05	.	0.06	.	0.03	0.10	0.01	0.07	0.02	0.03	0.03	0.01	0.01	.	0.07	.
C2	0.05	0.35	0.20	.	.	.	.	.	.	0.23	.	0.02	.	0.03	.	0.01	.	0.02	.	0.02	0.02	0.01	0.03	.	.	.	.
C3	.	.	.	0.05	0.12	0.08	0.04	0.18	0.11	.	0.11	.	0.11	.	0.10	.	.	0.04	.	.	.	.	.	0.01	0.05	.	.
C4	.	.	.	0.03	0.19	0.07	0.06	0.08	0.12	.	0.17	.	0.07	.	0.10	.	.	0.02	.	.	.	0.01	0.02	0.02	0.03	.	.
C5	.	0.06	0.01	.	0.01	.	.	.	.	0.03	.	0.08	.	0.04	.	0.03	0.23	0.01	0.32	0.02	0.03	0.05	0.02	.	.	0.05	0.01
C6	.	.	.	0.10	0.09	0.12	0.12	0.05	0.18	.	0.06	.	0.08	0.02	0.06	0.02	.	0.03	.	.	0.01	0.04	0.01	0.01	.	.	.
C7	0.01	0.01	0.02	.	.	.	.	.	.	0.11	.	0.16	.	0.24	.	0.20	0.02	0.02	0.02	0.02	0.04	0.04	0.06	0.01	.	0.01	.
C8	.	.	0.01	0.02	0.12	0.04	0.10	0.01	0.05	0.12	0.04	0.08	0.04	0.04	0.01	0.06	0.05	0.02	0.04	0.02	0.03	0.05	0.03	0.01	.	0.01	.
C9	.	.	.	0.04	0.16	0.10	0.02	0.07	0.05	.	0.18	.	0.07	.	0.02	0.02	.	0.14	.	.	.	0.02	0.03	0.05	0.02	.	.
C10	.	.	.	0.07	0.03	0.07	0.03	0.12	0.06	.	.	.	0.04	.	0.08	.	.	0.04	.	.	.	0.01	.	0.17	0.29	.	.
C11	0.24	0.22	0.13	.	.	.	.	.	.	0.22	.	0.01	.	0.03	.	0.03	0.01	.	.	0.01	0.03	0.01	0.04	.	.	.	.
C12	.	.	.	0.02	0.21	0.23	0.10	0.09	0.06	.	0.12	.	0.05	.	0.08	.	.	0.02	.	.	.	.	.	0.02	0.01	.	.
C13	.	0.17	0.21	.	.	.	.	.	.	0.09	.	0.04	.	0.06	.	0.06	0.13	0.04	0.02	0.01	0.10	0.03	0.04	.	.	0.01	.
C14	.	0.08	0.06	.	0.03	0.03	.	0.01	.	0.17	0.01	0.04	0.01	0.13	.	0.10	0.02	0.04	0.03	0.05	0.07	0.04	0.03	0.04	0.01	0.01	.
C15	.	0.14	0.02	.	0.01	.	.	0.01	.	0.14	.	0.01	0.02	0.03	.	0.06	0.11	0.03	0.19	0.02	0.04	0.08	0.05	0.02	.	0.03	.
C16	.	0.08	0.09	.	0.01	.	.	.	.	0.13	0.01	0.08	.	0.16	.	0.12	0.02	0.04	0.02	0.02	0.07	0.05	0.05	.	.	0.01	0.02
C17	.	.	.	0.12	0.09	0.10	0.07	0.16	0.11	.	0.05	.	0.02	.	0.10	.	.	0.04	.	.	.	0.01	.	0.12	.	.	.
C18	.	0.01	.	.	.	0.01	.	.	.	0.02	0.01	0.02	.	0.03	.	0.03	0.37	0.16	0.12	.	.	0.06	0.03	0.06	.	0.08	0.01
C19	.	.	0.02	0.02	0.05	0.05	0.03	0.01	.	.	.	0.01	0.05	.	0.01	.	0.03	.	.	.	.	0.02	0.05	0.05	.	0.60	.

Table G.6 – Matrice de provenance des lettres structurales

Annexe **H**

Article soumis à PLOS One

# SAFlex: A structural alphabet extension to integrate protein structural flexibility and missing data information

Ikram Allam<sup>1,2,3,4</sup>, Delphine Flatters<sup>1,4</sup>, Géraldine Caumes<sup>1,4</sup>, Leslie Regad<sup>1,4</sup>, Vincent Delos<sup>2,3,4</sup>, Grégory Nuel<sup>2,3,4,\*</sup>, Anne-Claude Camproux<sup>1,4,\*</sup>

**1** Molécules thérapeutiques in silico (MTi), INSERM UMR-S973, University Paris Diderot, Paris 7, France

**2** Probability Statistique and Biology (PSB), LPMA laboratory, CNRS INSMI UMR 7599, University Pierre et Marie Curie, Paris 6, France

**3** Mathématiques Appliquées, MAP5 laboratory, CNRS UMR 8145, University Paris Descartes, Paris 5, France

**4** Sorbonne Paris Cité, Paris, France

‡These authors also contributed equally to this work.

\* Corresponding authors email: Anne-Claude.Camproux@univ-paris-diderot.fr , Gregory.Nuel@math.cnrs.fr

## Abstract

In this paper, we describe a structural alphabet: SAFlex (Structural Alphabet Flexibility), improved from an existing alphabet (HMM-SA), to better explore increasing protein three dimensional structure information. A SA aims to reduce three dimensional conformations of proteins as well as their analysis and comparison complexity by simplifying any conformation in a series of structural letters. Our methodology presents several novelties. Firstly, it can account for the encoding uncertainty by providing a wide range of encoding options: the maximum a posteriori, the marginal posterior distribution, and the effective number of letters at each given position. Secondly, our new algorithm deals with the missing data in the protein structure files (concerning more than 75% of the proteins from the Protein Data Bank) in a rigorous probabilistic framework. Thirdly, SAFlex is able to encode and to build a consensus encoding from different replicates of a single protein such as several homomer chains. This allows localizing structural differences between different chains and detecting structural variability, which is essential for protein flexibility identification. These improvements are illustrated on different proteins, such as the crystal structure of an eukaryotic small heat shock protein. They are promising to explore increasing protein redundancy data and obtain useful quantification of their flexibility.

## Abbreviation

SA: structural alphabet, HMM: hidden Markov model, SL: structural letters, PDB: Protein Data Bank, 1D: one-dimensional, 3D: three-dimensional, MAP: Maximum a Posteriori, POST: marginal posterior distribution, ENT: marginal posterior entropy, NEFF: effective number of structural letters.

## Introduction

Over the past two decades, the notion of a structural alphabet (SA) has attracted much attention. SA encodes protein fragments into structural letters (SL). SA encoding plays a key role in compressing the three-dimensional (3D) protein conformations into a one-dimensional (1D) SL representation, thereby allowing for a simplified protein structure analysis [1–5]. This approach also dramatically simplifies the comparison of 3D conformations by using well-known sequence comparison algorithms (ex: local score from Smith and Waterman, [6]) on SL sequences.

Many studies have developed SAs, based on mixture models [7], classification methods such as AutoANN [8], SOM [9] and K-Nearest Neighbor [10]: Structural Building Blocks [11], Protein Blocks [4], SABD [12] and USA [13], M32K25 [14] ; and hidden Markov model (HMM): HMM-SA [2,3,15]. The choice between these methods and models

plays a major part in the construction of an accurate SA. They have been applied in the past to protein structure analysis, including multiple structure alignment, structure mining [16,17], protein fold classification [18], dynamic molecular analysis [19–21], structure fast comparison [17] and generation of 3D peptide conformations [22,23]. The SA approach also appears to be promising to characterize structural variability [24,25], to explore the local backbone deformation involved in protein-protein interactions [26,27] and to predict local protein flexibility [28,29].

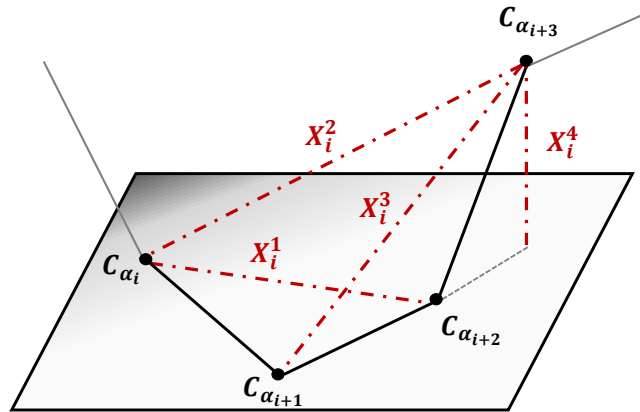
However, current SAs have not been trained to take into account the wealth of available protein structure data: their uncertainty (ex: missing data) and redundancy (ex: multiple homomers chains). The growth and speed of macromolecular structure determination techniques (protein crystallography and NMR spectroscopy) results in a considerable increase to 3D structures in the Protein Data Bank (PDB, [30,31]), which currently has more than 130,000 3D protein structures. More than half of PDB structures share at least 95% sequence identity. Even if this redundancy is considered valuable in investigating families of homologous sequences [32,33], the dominant approach for data mining the PDB considers redundancy as non-informative [34], resulting in an artificial reduction in the variability of the structural space. Yet, protein redundancy analysis is crucial for protein flexibility insight. Proteins are highly flexible macromolecules and their 3D folding and dynamic properties are essential in many biological processes [32,34]. Integrating PDB redundancy has potential to improve understanding of protein intrinsic flexibility [35]. Many PDB files contain multiple chains: either an individual protein chain that is not part of a complex (monomer) or several replicates or identical sub-units forming a protein complex within one unique PDB (homomer) or different 3D chains corresponding to one unique PDB (heteromers) or different PDB files corresponding to a same protein in different conditions (multi-conformations). For instance, in 2015, the non-redundant snapshot of protein crystal structures contained 7,972 monomers and 9,206 homomers and 2,677 heteromers [36]. The authors concluded, 87% of crystal structures involve only a single type of polypeptide chain, and a slight majority (54%) of these self-assemble into homomers. Thus, it is a very important point to be able to model this multiple chain data.

Another source of uncertainty in PDB data analysis is that most PDB structures have some missing parts which strongly impact the determination of their accurate protein folding and lead to important difficulty for protein structure and function interpretation [37]. This kind of issue is very serious, from missing side chains, entire loop regions, to whole domain. For instance, it was demonstrated in 2007 by [38] that ~10% of 16,370 PDB X-ray structure files contain regions of more than 30 missing or ambiguous amino acids and ~40% have missing or ambiguous regions between 10 and 30 amino acids. These missing parts can result from some resolution difficulty or from intrinsic flexibility of proteins [37,38]. The absence of the coordinates of the alpha-carbon, which makes it possible to connect to the peptide skeleton, poses a serious problem because it effectively prevents knowledge of the secondary structure and of the 3D folding. These missing parts often relate to the ends of the protein or loops that are the most flexible regions of proteins and involved in protein interactions and function. Thus the detection and modeling of these missing data could have a very appealing impact for protein structure analysis. However, they have not been explicitly modeled by different SA approaches.

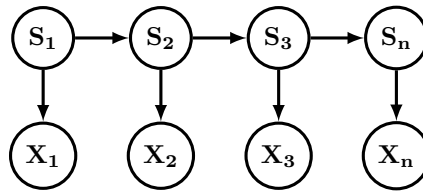
In this paper, we introduce a new HMM-based SA, called SAFlex, able to take into account the uncertainty and redundancy of the structural protein data. SAFlex is based on a previously published SA, named HMM-SA [3,15,39], modeling using HMM, which provides a very precise description of protein structures, particularly loop regions [40] known to play important roles in protein function. One major contribution of HMM is that this model implicitly takes the SL sequential connections into account. For example, this markovian modeling allows for efficient extraction of functional motifs [5,40]. The paper is organized as follows. The “Materials and Methods” section provides descriptions of the SA improvements based on our HMM modeling. It presents a collection of technical advances including a wide range of encoding options (the maximum a posteriori, the marginal posterior distribution, and the effective number of letters at each given position), the robustness of missing data in the PDB files, and the ability to build a consensus encoding from different replicates of a single protein with multiple chains. The “Results” section illustrates the application of our new approach on different PDB structures of interest. Finally, the “Conclusion” section summarizes the manuscript and discusses potential development and application of SAFlex for addressing structural biology challenges.

## Materials and Methods

In this section, we describe the model used to encode protein structures (PDB files) into SL sequences. To that aim, we introduce an HMM in which the spatial conformation of the protein is the observation and the underlying



**Fig 1.** The four descriptors  $X_i = (X_i^1, X_i^2, X_i^3, X_i^4) \in \mathbb{R}^4$  for the  $i^{\text{th}}$  fragment (alpha carbons  $C_{\alpha_i}$  to  $C_{\alpha_{i+3}}$ ).



**Fig 2.** The HMM-SA model.  $X_i \in \mathbb{R}^4$  are the fragment descriptors, and  $S_i \in \{1, 2, \dots, m\}$  are the structural letters.

structural sequence is the hidden part.

60

## Structural Fragments

61

The PDB file input, obtained from the worldwide Protein Data Bank (wwPDB) (<http://www.wwpdb.org/>) [41, 42], contains the atomic coordinates describing the 3D structure of the protein. Since the original observation (3D structures from PDB files) is highly complex, we start by reducing this complexity to a sequence of numeric descriptors as in HMM-SA original publication [2, 3]. Starting from the 3D positions of the alpha carbons (denoted  $C_{\alpha}$ ), we call *fragment* a succession of four consecutive  $C_{\alpha}$  and we use the four descriptors of Fig. 1. Formally, for the  $i^{\text{th}}$  fragment we have:  $X_i^1 = D(C_{\alpha_i}, C_{\alpha_{i+2}})$ ,  $X_i^2 = D(C_{\alpha_i}, C_{\alpha_{i+3}})$ ,  $X_i^3 = D(C_{\alpha_{i+1}}, C_{\alpha_{i+3}})$ , and  $X_i^4 = \eta D(C_{\alpha_{i+1}}, C_{\alpha_{i+2}})$  where  $D$  is the Euclidian distance,  $H$  is the orthogonal projection of  $C_{\alpha_{i+3}}$  on the plane  $(C_{\alpha_i}, C_{\alpha_{i+1}}, C_{\alpha_{i+2}})$ , and where  $\eta = +1$  (resp.  $-1$ ) if the cross-product of vector  $C_{\alpha_i} \rightarrow C_{\alpha_{i+2}}$  and vector  $C_{\alpha_i} \rightarrow C_{\alpha_{i+1}}$  has the same (resp. opposite) direction than vector  $H \rightarrow C_{\alpha_{i+3}}$ . Since a structural fragment is formed by four consecutive alpha carbon, a sequence of  $n + 3$  alpha carbons will have only a total of  $n$  fragments. The Fragment  $i$  hence corresponds to the four alpha carbons:  $C_{\alpha_i}, C_{\alpha_{i+1}}, C_{\alpha_{i+2}}, C_{\alpha_{i+3}}$ .

72

## A Hidden Markov Model

73

Our idea is to consider the sequence  $X_{1:n} \in \mathbb{R}^{n \times 4}$  of  $n$  structural fragments as the observed states of a HMM where the hidden states are the SL  $S_{1:n} \in \{1, \dots, m\}^n$ . A markov dependency is assumed between the  $m$  SL and a (conditional) Gaussian model is used for the fragment descriptor distribution. The resulting model represented in Fig. 2 has the following probability distribution:

77

$$\mathbb{P}(X_{1:n}, S_{1:n}) = \mathbb{P}(S_1) \underbrace{\prod_{i=2}^n \mathbb{P}(S_i | S_{i-1})}_{\text{Markov part}} \times \underbrace{\prod_{i=1}^n \mathbb{P}(X_i | S_i)}_{\text{Emission part}} \quad (1)$$

Assuming that the Markov chain has an uniform starting distribution, a homogeneous transition matrix  $\pi \in \mathbb{R}^{m \times m}$ , and that we denote  $e_i(S_i) = \mathbb{P}(X_i|S_i)$  we get the following simplified equation:

$$\mathbb{P}(X_{1:n}, S_{1:n}) = \frac{1}{m} \prod_{i=2}^n \pi(S_{i-1}, S_i) \prod_{i=1}^n e_i(S_i) \quad (2)$$

For the emission distribution, we simply assume that fragment descriptors are Gaussian distributed with a specific mean vector  $\mu_s \in \mathbb{R}^4$  and covariance matrix  $\Sigma_s \in \mathbb{R}^{4 \times 4}$  for each structural letter  $s \in \{1, \dots, m\}$  which hence gives:

$$\log e_i(s) = \text{cst.} - \frac{1}{2} \log \det \Sigma_s - \frac{1}{2} (X_i - \mu_s)^T \Sigma_s^{-1} (X_i - \mu_s). \quad (3)$$

One should note that the choice of a uniform starting distribution (rather than a estimated one as in HMM-SA original publication [2, 4]) has little impact on the encoding (Markov has indeed short range dependency), but is quite useful in terms of parsimony since the resulting model has less parameter to estimate.

## Protein encoding

### Maximum a Posteriori

The task of encoding a 3D structure (sequence of n fragments) into a structural sequence can be achieved by computing the Maximum a Posteriori (MAP) defined by:

$$\text{MAP}_{1:n} = \arg \max_{S_{1:n}} \mathbb{P}(S_{1:n}|X_{1:n}) = \arg \max_{S_{1:n}} \mathbb{P}(X_{1:n}, S_{1:n}) \quad (4)$$

In order to compute this quantity, we introduce the max-forward and max-backward quantities:

$$F_1^{\max}(s) = e_1(s)/m \quad \text{and} \quad F_i^{\max}(s) = \max_{S_{1:i-1}} \mathbb{P}(X_{1:i}, S_{1:i-1}, S_i = s) \quad \text{for } i = 2 \dots n \quad (5)$$

$$B_n^{\max}(s) = 1 \quad \text{and} \quad B_{i-1}^{\max}(s) = \max_{S_{i:n}} \mathbb{P}(X_{1:i}, S_{i:n}|S_i = s) \quad \text{for } i = n \dots 2 \quad (6)$$

which can be computed recursively for  $i \in \{2, \dots, n\}$  through:

$$F_i^{\max}(s) = \max_r F_{i-1}(r) \pi(r, s) e_i(s) \quad \text{and} \quad B_{i-1}^{\max}(r) = \max_s \pi(r, s) e_i(s) B_i(s) \quad (7)$$

and we finally have:

$$\text{MAP}_i = \arg \max_s F_i^{\max}(s) B_i^{\max}(s) \quad \text{for all } i = 1 \dots n \quad (8)$$

### Marginal Posterior Distribution

Alternatively, we can focus on the marginal posterior distribution (POST) rather than the MAP:

$$\text{POST}_i(s) = \mathbb{P}(S_i = s|X_{1:n}) = \frac{\mathbb{P}(S_i = s, X_{1:n})}{\mathbb{P}(X_{1:n})} \quad (9)$$

These quantities can be easily computed with an adaptation of the previous MAP computation by simply replacing all ‘max’ occurrences by sums in Equations (5), (6), and (7). Once the sum-forward and sum-backward quantities computed, it is possible to obtain immediately the marginal posterior distribution:

$$\text{POST}_i(s) \propto F_i^{\text{sum}}(s) B_i^{\text{sum}}(s) = \frac{F_i^{\text{sum}}(s) B_i^{\text{sum}}(s)}{\sum_r F_i^{\text{sum}}(r) B_i^{\text{sum}}(r)} \quad (10)$$

Note that since this quantity is a *marginal* posterior distribution,  $\arg \max_s \text{POST}_i(s)$  is not necessary equal to  $\text{MAP}_i$  (even if this is often the case when the posterior distribution is ‘sharp’ enough).

This POST also can be used to quantify the level of uncertainty of the structural letter encoding by computing the marginal posterior entropy (ENT) and effective number of SL (NEFF).

$$\text{ENT}_i = - \sum_s \text{POST}_i(s) \log \text{POST}_i(s) \quad \text{and} \quad \text{NEFF}_i = \exp(\text{ENT}_i) \quad (11)$$

$\text{ENT}_i$  is a measure of disorder for SL  $i$ . If the posterior distribution is a Dirac,  $\text{ENT}_i = 0$ , the minimal entropy, if the encoding uncertainty is maximum with  $\text{POST}_i(s) = 1/m$ ,  $\text{ENT}_i = \log m$  is maximal. The effective number of SL  $\text{NEFF}_i \in [1, m]$  provides a simpler interpretation of the entropy as the effective number of SL acceptable for Fragment  $i$ .

## Missing Data

When dealing with 3D proteins structures, it is quite common that some alpha carbon positions are missing, thus resulting in several missing data in fragment descriptors. It is therefore necessary to deal efficiently with these missing data, which is quite straightforward under the Gaussian assumption. Let denote by  $J \subset \{1, 2, 3, 4\}$  the subset of non-missing descriptors for fragment  $i$ .  $J = \emptyset$  if all descriptors are missing, and  $J = \{1, 2, 3, 4\}$  if none are missing. Then the emission probability of fragment  $j$  for the structural letter  $s$  is obtained by considering the restriction  $X_i[J] \sim \mathcal{N}(\mu_s[J], \Sigma_s[J, J])$ . By convention,  $e_i(s) = 1$  for all  $s$  if  $J = \emptyset$  meaning that the totally missing fragment  $i$  is totally uninformative.

## Multiple Chains

In the PDB, some proteins are represented several times. It could be simple replicates, or 3D structures in different conditions (ex: associated or not with various partners, with or without mutated positions). It is obviously possible to perform one encoding for each of these replicates, but there is also another possibility: build a consensus encoding from the whole replicate set. This can be done easily by considering the model of Fig. 3 which results in the following likelihood:

$$\mathbb{P}(X_{1:n}^{1:k}, S_{1:n}) = \mathbb{P}(S_1) \underbrace{\prod_{i=2}^n \mathbb{P}(S_i | S_{i-1})}_{\text{consensus encoding}} \times \prod_{j=1}^k \underbrace{\prod_{i=1}^n \mathbb{P}(X_i^j | S_i)}_{\text{emission for Chain } j} \quad (12)$$

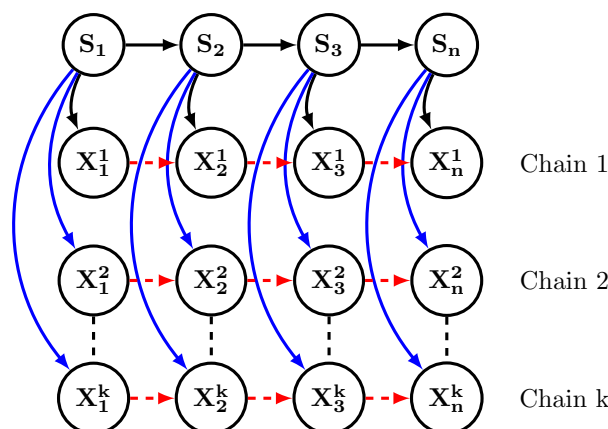
The previous encoding algorithm can easily be adapted to this new context by slightly changing the expression of  $e_i(s)$ :

$$\log e_i(s) = \text{cst.} - \sum_{j=1}^k \left( \frac{1}{2} \log \det \Sigma_s + (X_i^j - \mu_s)^T \Sigma_s^{-1} (X_i^j - \mu_s) \right). \quad (13)$$

Note that missing data can also be easily taken into account in this context. In the particular case when large portion of the data are missing in multiple chain, the model can easily cope with this situation as long as all the chains are properly aligned using a common reference index. In such a situation, a particular position would typically be informative only for a small portion of the chains, which is not a problem for the model.

## SAFlex structural alphabet implementation

A SAFlex preliminary web server including dynamic pages (html/javascript) is made freely available to the scientific community. The backend was developed in the C++ language as a high performance encoding program. All computations (evidence, forward, backward and posterior marginal) are performed in logarithmic scale and thus allow for low probabilities. The SAFlex server is able to complete encoding and indication of data uncertainties in most cases in less than one second. The SAFlex web server has been successfully tested on latest version of Chrome, Firefox and Safari. SAFlex is freely available at the following URL <http://saflex.rpbs.univ-paris-diderot.fr/SA-Encoder.php>.



**Fig 3.** The SAFlex model with k chains.

## Results

### The new structural alphabet: SAFlex

Here we propose to extend SA-based approach to take into account protein redundancy and missing data. To this aim, we have developed a SA, referred to as SAFlex, using our previously developed HMM-SA built using HMM and resulting in  $m = 27$  SL with low geometric variability [3, 15, 39]. In order to improve the interpretability of the 27 SL, a novel nomenclature is applied in SAFlex. This structural letter assignment now depends on the secondary structure type, as identified using the STRIDE software [43] in [3]. Hence, we use only three letters ('A', 'B', 'C') which refer to the class of SL (*Helices* or *Strands* or *Coils* respectively) as indicated in Table 1. Each letter assignment is followed by a number indicating the frequency thereof in the data set after unsupervised learning. Thus, SL are ranked in descending order according to their frequencies, for example: the letter 'A1' is more frequent than the letter 'A2'. The correspondence with the previous HMM-SA nomenclature is provided in Table 1. SA nomenclature update and corresponding frequency, effective number of SL acceptable in input or output are also indicated.

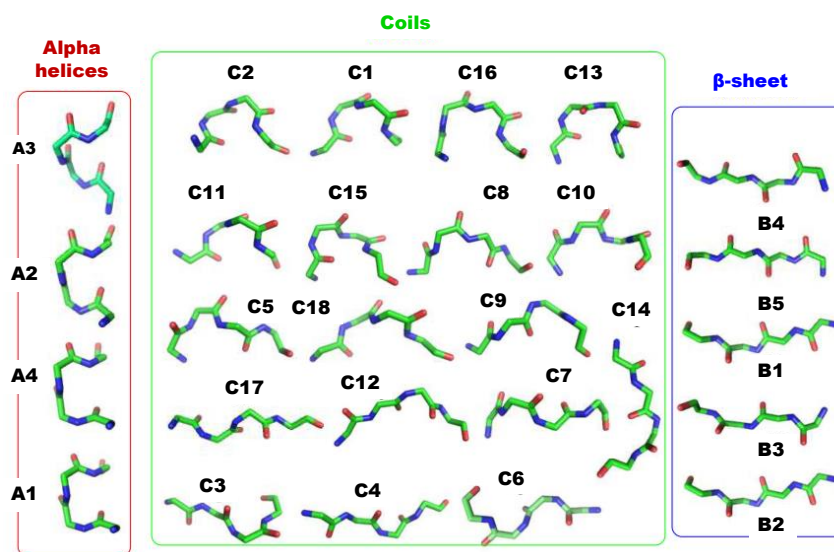
For completeness, the SAFlex 27 representative fragments are illustrated on Fig. 4 the previously published emission parameters and transition matrix of the SA are provided as supplementary material, (S1 and S2).

**Table 1. SAFlex structural description.** Nomenclature correspondence between HMM-SA and SAFlex SL. Frequency corresponds to associated frequency observed in the HMM-SA training data set, NEFF\_output (resp. NEFF\_input) corresponds to the effective number of SL acceptable in output (resp. input).

	27 structural letters																										
	helices				loops																		strands				
SAFlex	A1	A2	A3	A4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	B1	B2	B3	B4	B5
HMM-SA	A	V	W	a	B	Z	P	K	Q	G	S	I	H	D	E	J	U	Y	F	C	R	O	M	L	N	X	T
Frequency %	12.6	5.6	5.3	2.6	4.7	4.5	4.4	4.1	4.1	3.4	3.2	2.9	2.7	2.0	2.0	2.0	2.0	2.0	1.9	1.8	1.7	1.5	5.3	5.1	4.9	4.7	3.0
NEFF_output	2.4	3.6	3.7	2.8	9.6	11.7	11.8	10.4	10.7	12.6	10.4	9.3	9.6	4.3	12.4	11.7	8.5	15.5	8.9	11.6	11.8	13.0	7.9	10.8	9.6	9.2	7.6
NEFF_input	2.3	4.5	3.4	3.0	9.0	12.4	13.3	10.5	12.2	17.5	8.3	7.1	9.7	9.6	11.2	8.8	12.5	10.8	13.5	6.2	12.4	9.8	8.0	10.8	7.0	10.4	7.7

As explained in the “Materials and Methods” section, our model provides three different encoding information for each chain: 1) the MAP offers the most probable SL sequence fully taking account the complex dependence structure between the SL (including transition probabilities). This is typically the primary encoding used by experimentalists for structure analysis and comparison. 2) In order to account for encoding uncertainty, the POST provides the weighted distribution of the possible SL at each given position. This information is particularly useful for the structural regions where the encoding is difficult or variable and could be more representative of the 3D structure than the MAP. 3) For better interpretation, the NEFF of SL at each given position is also derived from the entropy of POST. This NEFF is typically close to 1 when the encoding is highly certain and can reach 27 for totally uncertain





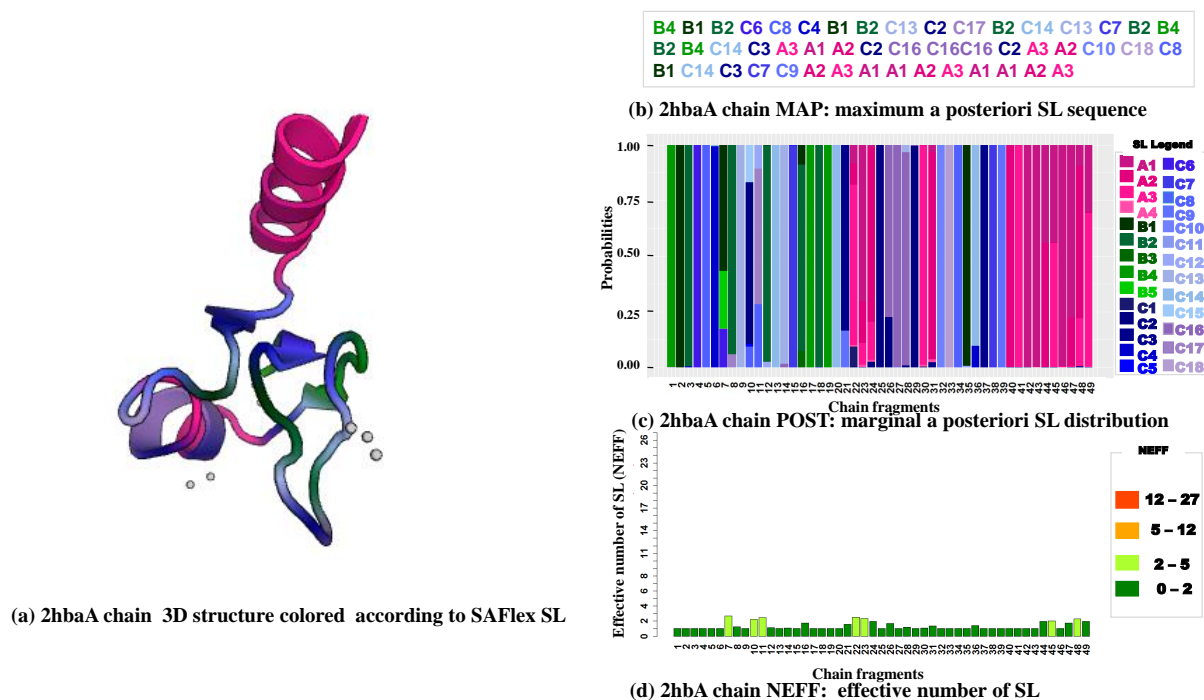
**Fig 4. Structural letter prototypes of SAFlex** The 27 representative 3D structural fragments associated with 27 structural letters of SAFlex alphabet. The LS are classified into three groups relatively to their secondary structure correspondence: alpha helices, coils and beta-sheet, as described in [3]. The alpha helices correspond to 4 SL, the coils to 16 SL and beta-sheet to 5 SL.

positions. This NEFF hence provides a convenient measurement of the encoding certainty.

We can see on Fig. 5 the 3D Chain A of the 2hba PDB and the three corresponding outputs of SAFlex . This structure corresponds to the N-terminal domain of the ribosomal protein L9, (NTL9), a small alpha-beta protein of 52-residue mixed protein. NTL9 has been widely used as a model system for experimental and computational studies of protein folding and for investigations of the unfolded state [44]. On the left panel of Fig. 5, the 3D structure of 52 residues is represented, which gives 49 overlapping structural fragments colored according to the 27 SL encoding. On the right panel are represented from top to bottom: MAP, POST, and NEFF. The MAP clearly corresponds to a alpha-beta mixed protein with few coil links. In the POST, we can see that most encoding positions are highly certain, even if few positions display some uncertainty. We observe on the protein three regions of relative uncertainty in fragment positions [7-11], [21-24], and the end region [44-49]. For example, Fragment 10 posterior distribution highlights three possible coil letters: C2 (prob=0.726), C15 (prob=0.167), C8 (prob=0.093). Finally, the NEFF graph provides at a quick glance a good idea of the encoding uncertainty at each position, for instance, NEFF=2.244 for Fragment 10.

## Missing data

Many structures in the PDB have missing parts. For example, a representative data set, PDBselect [45], includes 8,565 PDB files with 75% of chains faced with a problem of missing data. It may be the missing coordinates of the complete residues or of the alpha-carbon atoms. As explained in the “Materials and Methods” section, our new model rigorously takes into account these missing information through probabilistic computations depending on the missing pattern of descriptors at the corresponding positions. Indeed when all four descriptors are unavailable, the position is totally uninformative (local likelihood of 1.0 for all SL) but the context of the position is still accounted for thanks to the Markov dependence of the model through the transition matrix. When only few descriptors are missing, the local likelihood can still be computed using the marginal Gaussian emission probabilities and this partial information can be combined with the local context to provide reliable estimations. However, when repeated missing patterns appear in a given region of the 3D structure, the resulting encoding remains highly uncertain which is



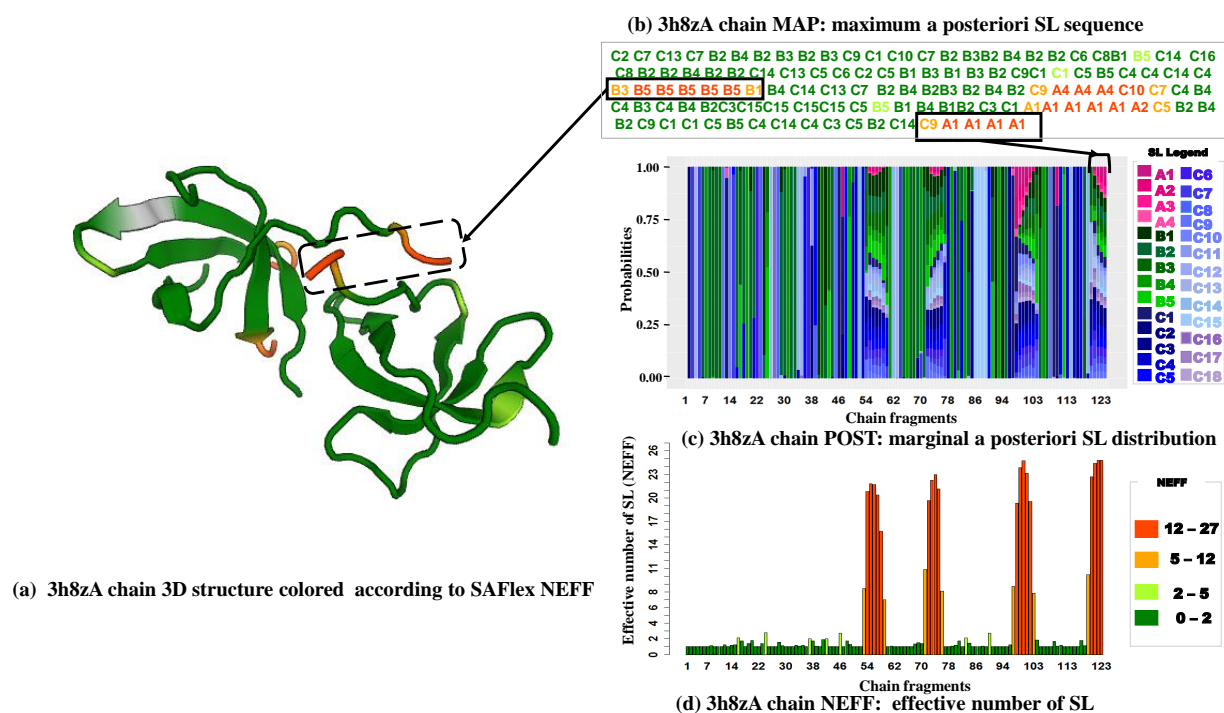
**Fig 5. SAFlex encoding of 2hba pdb structure, corresponding to the N-terminal domain of the ribosomal protein L9 (NTL9).** On the left side, the 2hba 3D structure itself is represented, colored according to the 27 SAFlex SL. On the right side are represented from top to bottom, the 2hba corresponding SAFlex MAP, POST encoding colored according to the 27 SAFlex SL and NEFF values.

typically pointed out by high NEFF values.

As an illustration we consider the protein 3h8z corresponding to the homo sapiens fragile X mental retardation syndrome-related protein 2 associated with FXR2 gene. The FMRP, FXR1 and FXR2 proteins comprise a small family of highly conserved proteins that appear to be important in translational regulation, particularly in neuronal cells [46]. We can see on Fig. 6, the crystal structure of the tudor domains from FXR2 (left panel) from the PDB 3h8z and the SAFlex outputs from top to bottom: MAP, POST, and NEFF (right panel). This 3D structure correspond to 123 overlapping structural fragments colored according to their NEFF values. We observe in Fig. 6 (a), (b) that a big part of the protein correspond to beta-strand SL linked by short coil SL regions, associated with weak uncertainties Fig. 6 (c), (d). This is in agreement with the fact Tud1 domain forms a canonical tudor barre comprising five highly twisted antiparallel beta-strands. However we clearly observe the presence of four regions of high uncertainty (NEFF close to 27). These four regions correspond to the 16 residues with missing 3D coordinates information in the PDB file with the first region (53-59) predicted as disorder using psipred website [47,48]. Despite this high uncertainty, one should note that the MAP suggests encodings for these four regions. However these positions being associated to a NEFF close to 27, one's should be aware of the high uncertainty of this encoding. This further illustrates the interest of the multiple outputs of SAFlex.

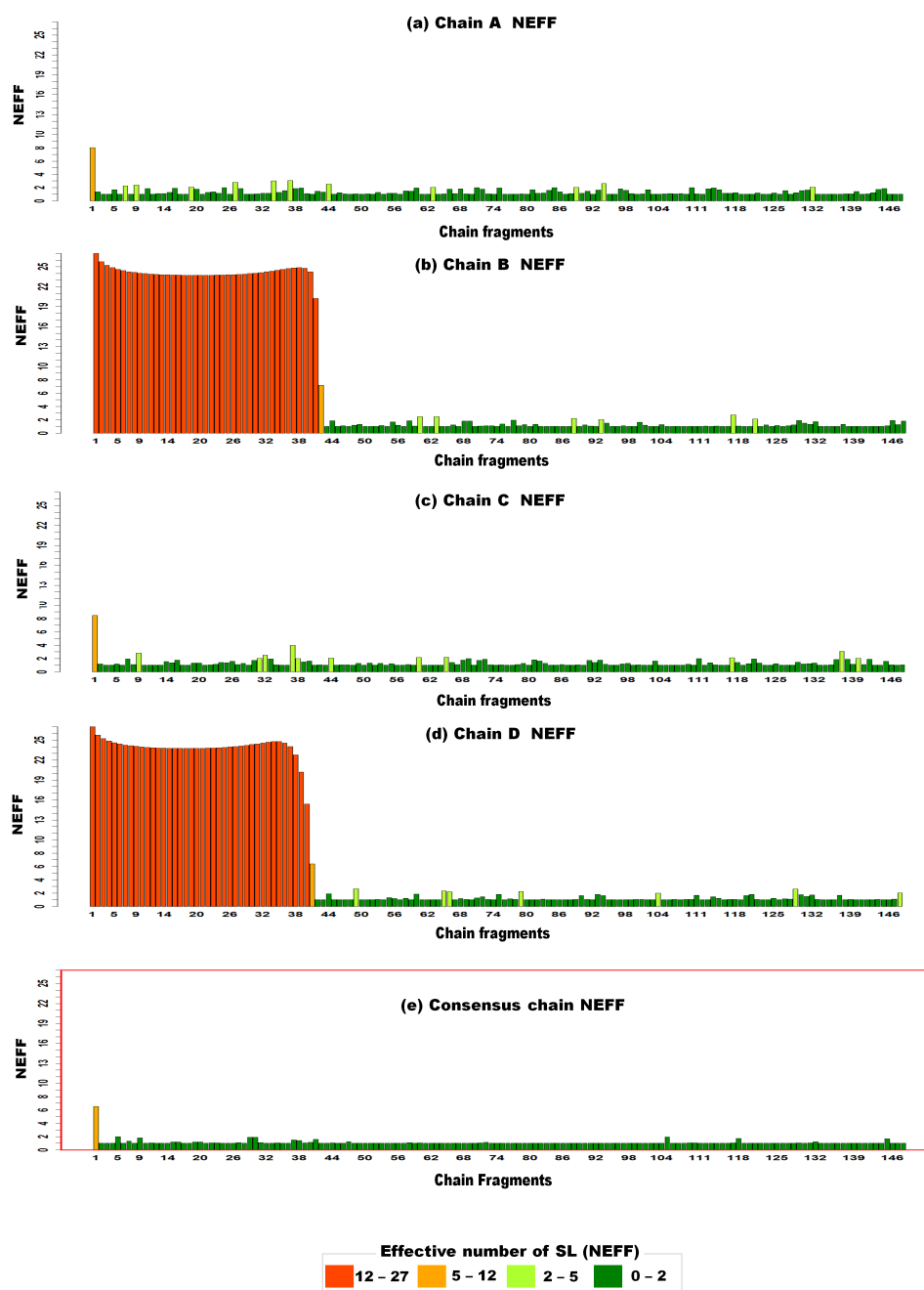
## Multi-chains

In recent decades, many protein complex structures have been determined containing multiple chains: homomers or heteromers or different PDB files corresponding to a same protein in different conditions. If heteromers are naturally encoded as different structural sequences, homomers can be assumed as replicates of the same underlying structure. SAFlex proposes to encode homomers either as independent structures (one structural sequence per chain) or as a single consensus one, where a single hidden structural sequence is shared by all homomeric chains. The resulting



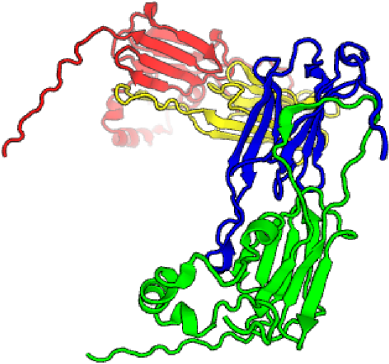
**Fig 6.** Missing conformation detection of the 3h8z pdb entry, corresponding to the homo sapiens fragile X mental retardation syndrome-related protein 2 associated with FXR2 gene. On the left panel the 3h8z3D structure is presented colored according to the NEFF value legend. On the right panel, are represented from top to bottom, the 3h8Z corresponding SAFlex MAP colored according to the the NEFF value legend, the POST encoding and the NEFF values.

consensus encoding hence represents the variability of the homomer across the chains, this variability being either due to measurement uncertainty or to intrinsic flexibility. One illustration is presented on the small Heat Shock Proteins (sHSPs), which are important in stress tolerance and play an essential role in preventing aggregation of target proteins [49]. They participate in protecting, maintaining and regulating specific protein functions. The PDB entry 1gme includes HSP16.9, a member of the sHSPs, that assembles into a dodecameric double disk. In the PDB file, a tetramer is available that allows reconstruction of the dodecamer by symmetry operations [50]. The monomers length is 151 which gives 148 overlapping structural fragments. The four monomers have a global common structure, called the alpha crystallin domain signature [49] but differences in some regions: the 42 N-terminal residues are missing in the two monomers B and D, whereas the N-terminal arm in A and C monomers is fully resolved and composed of helices connected by random coil. We can see on Fig. 7 the values of NEFF for the four chains (upper panel) and for the consensus encoding (lower panel). The two missing regions of chains B and D are clearly highlighted (NEFF close to 27 on positions [1, 42]). The overall uncertainty of encoding along the four chains is quite large with an average value of  $NEFF \simeq 4.4$  on all the regions and of  $NEFF \simeq 1.3$  when excluding missing regions. On the lower panel, we can see that the consensus encoding on all regions has a much lower uncertainty of  $NEFF \simeq 1.1$ . This illustrates the interest of the consensus approach which not only provides an encoding for the complete protein despite the missing patterns of chains B and D, but also takes advantage of the replicates to refine the structural encoding. In Fig. 8 we see (a) the 3D structure of the four chains of 1gme (left panel), (b) the multiple alignment of the MAP for the four chains and the consensus encoding (upper-right panel) as well as (c) the POST encoding for the consensus encoding (lower-right panel). The missing regions in chains B and D clearly appear in the MAP with long runs of the A4 SL; this is quite natural in the absence of any additional information since this SL has the highest probability of self-transition in the SA (see transition matrix in the supplementary material). However, chains A and C provide informative MAP for the corresponding positions and the result is clearly consistent with the consensus' MAP. For most of the remaining positions, we observe a strong agreement between all encodings thus reflecting the low variability of the structures. But interestingly for some regions in fragment positions [29-32], [85-92], [110-116] and [136-142], there is a lot of variability across the MAP of the four chains, indicated by different SL for the four chains. This suggest some of these positions could correspond to intrinsic flexible chain positions or to resolution uncertainties. In this context, the consensus encoding tries to find the most adequate common structural letter to reflect this variability and selects the C15 SL (former F in HMM-SA) which is known to correspond to the fuzzy coil state of the alphabet [39], associated with very high posterior probabilities (close to one), Fig. 8 (c). This result is clearly consistent with the assumption of a common underlying structure among the different chains but also shows its limits in case of conformation plasticity. In this case, the independent chain encodings can be carefully explored to detect variable positions and SL changes potentially due to intrinsic flexibility, to partner binding or to sequence mutation effects [25].

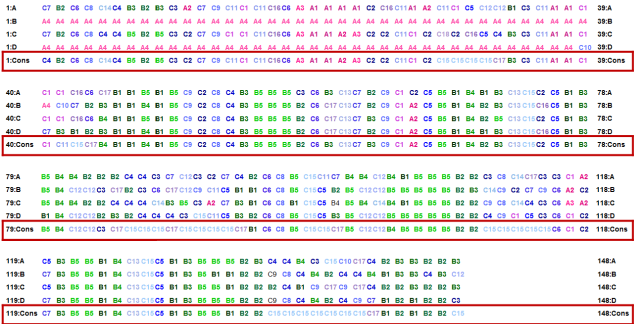


**Fig 7. NEFF of the four monomers and consensus chain of the 1gme pdb, corresponding to the small Heat Shock Proteins (sHSPs).** The x-axis of the charts correspond to the 149 fragment numbers and the y-axis to the NEFF values, from 1 to 27. Each bar is colored according to the NEFF value legend. The charts (a), (b), (c), (d) correspond to the NEFF values for the four monomer chains A, B, C and D (upper panel) and the chart (e), in a red box to the consensus encoding of the four monomers (lower panel).

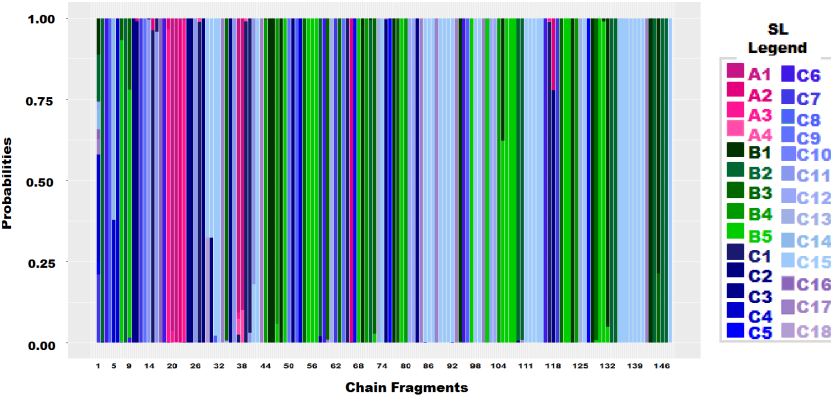
(a) 1gme 3D structure colored according to its four chains (A, B, C, D)



(b) Multiple alignment of 1gme four chains and its consensus chain, encoded and colored according to SAFlex



(c) 1gme consensus chain POST distribution



**Fig 8. SAFlex independent and consensus encoding of the 1gme four chains.** On the left panel,(a) the 3D structure of 1gme is displayed by PV, colored according to its four chains: A in red, B in yellow, C in green and D in blue. Each monomer corresponds to148 overlapping structural fragments. On the right panel, (b) the multiple alignment of the MAP of the four encoded chains and the consensus encoding in a red box. Each letter is colored according to SAFlex SL. (c) The POST encoding for the consensus chain (lower-right panel). The x-axis corresponds to the fragment numbers and the y-axis represents the posterior probabilities.

## Conclusion

Conceiving protein structures has moved beyond static representations to include dynamic aspects of quaternary structures, like conformational changes upon binding and structural fluctuations occurring within fully assembled complexes [37]. SA encodings are known to perform well for fine study of their structural properties. However SA had to be improved to better understand uncertainties such as missing data, intrinsic flexibility observed between different available replicates in the PDB. This has an impact on our knowledge of protein functions and their disordered regions, which contribute to the protein capacity to establish interactions with different partners. In this paper, we presented a new structural alphabet, SAFlex, improved from HMM-SA, with similar fragment descriptors, number of letters (27), Gaussian distribution per letter, and transition matrix. SAFlex has the following three main novelties: 1) allows for three different encoding outputs (MAP, marginal posterior distribution, and entropy-related statistics); 2) new implementation is robust for any missing data pattern in the PDB file; and 3) new model can take into account multi-chains data and include a new consensus encoding for homomers. All these improvements are freely available to the public through a web-server application. Now that a basic version of SAFlex has been implemented, our next step would be to use up-to-date large scale data to train a new alphabet with the following characteristics: more parsimony with uniform starting distribution, model selection using penalized approaches (ex: adaptive ridge [51]) for the (very sparse) transition matrix, more descriptors to ensure a bijection between the 3D fragment conformation and the descriptor space, and model extensions to allow for multi-conformations (heteromers) in addition to the multi-chains feature.

## Acknowledgments

This work was supported by an USPC grant (SA-Flex) and an ANR grant (ANR-10-BINF-0003, BIP-BIP). We are grateful to M. Petitjean for helpful discussions.

## Supplementary data

### S1. SAFlex 27 Gaussian Emission Distributions

State	$\mu$	$\Sigma$
A1	[5.43 5.09 5.42 2.94]	0.01 0.01 0.00 0.00
		0.01 0.02 0.00 0.01
		0.00 0.00 0.01 0.00
		0.00 0.01 0.00 0.02
A2	[5.41 5.23 5.61 2.86]	0.01 0.01 0.00 0.00
		0.01 0.04 0.00 0.03
		0.00 0.00 0.01 0.00
		0.00 0.03 0.00 0.04
A3	[5.62 5.25 5.42 2.87]	0.01 0.01 0.00 0.00
		0.01 0.04 0.01 0.03
		0.00 0.01 0.01 -0.01
		0.00 0.03 -0.01 0.04
A4	[5.39 5.09 5.38 2.92]	0.03 0.01 -0.01 -0.01
		0.01 0.06 0.01 0.03
		-0.01 0.01 0.03 -0.01
		-0.01 0.03 -0.01 0.06
C1	[5.40 5.58 5.42 3.39]	0.02 0.02 0.00 0.00
		0.02 0.06 0.01 0.03
		0.00 0.01 0.02 -0.01
		0.00 0.03 -0.01 0.03

<b>C2</b>	[5.59 5.49 5.78 2.58]	<div>0.05 0.06 −0.02 0.02</div> <div>0.06 0.19 0.03 0.11</div> <div>−0.02 0.03 0.04 −0.01</div> <div>0.02 0.11 −0.01 0.16</div>
<b>C3</b>	[6.57 8.96 5.58 −2.19]	<div>0.14 0.06 0.02 −0.08</div> <div>0.06 0.10 0.06 0.07</div> <div>0.02 0.06 0.06 0.00</div> <div>−0.08 0.07 0.00 0.44</div>
<b>C4</b>	[6.72 9.12 6.41 −3.31]	<div>0.10 0.08 0.00 −0.01</div> <div>0.08 0.13 0.04 0.04</div> <div>0.00 0.04 0.05 0.03</div> <div>−0.01 0.04 0.03 0.04</div>
<b>C5</b>	[5.66 8.07 6.70 2.96]	<div>0.08 0.09 0.01 −0.01</div> <div>0.09 0.32 0.07 −0.09</div> <div>0.01 0.07 0.09 −0.11</div> <div>−0.01 −0.09 −0.11 0.16</div>
<b>C6</b>	[6.21 9.21 5.77 0.27]	<div>0.15 0.11 0.04 −0.02</div> <div>0.11 0.12 0.09 −0.01</div> <div>0.04 0.09 0.09 0.03</div> <div>−0.02 −0.01 0.03 0.57</div>
<b>C7</b>	[5.66 8.95 6.54 2.09]	<div>0.06 0.05 0.00 0.01</div> <div>0.05 0.10 0.03 −0.13</div> <div>0.00 0.03 0.08 −0.02</div> <div>0.01 −0.13 −0.02 0.40</div>
<b>C8</b>	[5.70 7.26 7.02 0.88]	<div>0.08 0.13 0.01 −0.01</div> <div>0.13 0.40 0.10 −0.06</div> <div>0.01 0.10 0.05 −0.10</div> <div>−0.01 −0.06 −0.10 1.07</div>
<b>C9</b>	[6.71 8.27 5.47 −3.56]	<div>0.09 0.03 0.00 −0.03</div> <div>0.03 0.09 0.04 0.03</div> <div>0.00 0.04 0.04 0.00</div> <div>−0.03 0.03 0.00 0.04</div>
<b>C10</b>	[5.55 7.74 5.60 −3.31]	<div>0.06 0.04 −0.01 0.00</div> <div>0.04 0.13 −0.01 0.11</div> <div>−0.01 −0.01 0.05 −0.03</div> <div>0.00 0.11 −0.03 0.14</div>
<b>C11</b>	[5.60 6.71 5.58 3.69]	<div>0.08 0.12 0.03 0.00</div> <div>0.12 0.36 0.10 0.01</div> <div>0.03 0.10 0.08 −0.01</div> <div>0.00 0.01 −0.01 0.01</div>
<b>C12</b>	[6.89 8.94 6.76 −0.48]	<div>0.09 0.10 −0.02 −0.17</div> <div>0.10 0.84 0.25 0.06</div> <div>−0.02 0.25 0.13 0.17</div> <div>−0.17 0.06 0.17 3.74</div>
<b>C13</b>	[6.47 5.92 5.56 0.53]	<div>0.10 0.18 0.00 0.03</div> <div>0.18 0.45 0.06 0.20</div> <div>0.00 0.06 0.04 0.04</div> <div>0.03 0.20 0.04 1.30</div>



<b>C14</b>	[6.87 8.28 6.03 -3.44]	<div> 0.07 0.05 -0.03 -0.01  0.05 0.28 0.15 0.03  -0.03 0.15 0.16 0.06  -0.01 0.03 0.06 0.07 </div>
<b>C15</b>	[6.03 6.85 5.64 -0.63]	<div> 0.44 0.50 -0.02 0.43  0.50 1.87 0.23 0.79  -0.02 0.23 0.22 -0.11  0.43 0.79 -0.11 6.57 </div>
<b>C16</b>	[5.78 5.68 6.07 1.46]	<div> 0.08 0.09 -0.01 0.02  0.09 0.26 0.06 0.06  -0.01 0.06 0.06 -0.04  0.02 0.06 -0.04 0.36 </div>
<b>C17</b>	[5.66 8.91 6.66 -1.46]	<div> 0.12 0.10 0.02 -0.04  0.10 0.28 -0.02 0.29  0.02 -0.02 0.14 -0.08  -0.04 0.29 -0.08 1.15 </div>
<b>C18</b>	[5.69 8.09 5.67 3.09]	<div> 0.06 0.03 0.00 0.01  0.03 0.14 0.04 -0.13  0.00 0.04 0.07 0.00  0.01 -0.13 0.00 0.22 </div>
<b>B1</b>	[6.87 10.06 6.51 -1.41]	<div> 0.06 0.06 0.04 -0.03  0.06 0.10 0.08 0.02  0.04 0.08 0.08 0.00  -0.03 0.02 0.00 0.26 </div>
<b>B2</b>	[6.71 9.64 6.50 -2.60]	<div> 0.10 0.08 0.02 -0.03  0.08 0.11 0.06 0.04  0.02 0.06 0.06 0.03  -0.03 0.04 0.03 0.12 </div>
<b>B3</b>	[6.39 9.93 6.75 -1.07]	<div> 0.05 0.05 0.02 -0.01  0.05 0.06 0.04 0.04  0.02 0.04 0.05 0.03  -0.01 0.04 0.03 0.25 </div>
<b>B4</b>	[6.48 10.17 7.09 0.66]	<div> 0.08 0.08 0.01 0.05  0.08 0.08 0.02 0.01  0.01 0.02 0.02 0.01  0.05 0.01 0.01 0.31 </div>
<b>B5</b>	[6.80 10.35 6.85 -0.25]	<div> 0.05 0.06 0.03 0.00  0.06 0.07 0.05 0.01  0.03 0.05 0.05 0.01  0.00 0.01 0.01 0.20 </div>

**Table 2. The SAFlex gaussian emission distributions.** Rows correspond to the SAFlex SL. The letters are classified by group : alpha helices, coils and beta-sheets. For each LS is given the four descriptors  $X = (X^1, X^2, X^3, X^4) \in \mathbb{R}^4$  and the covariance matrix  $\Sigma$ . Rows and colons of the matrix corresponds to descriptors.

## S2. SAFlex Transition Matrix

**Table 3. The SAFlex transition matrix.** Rows correspond to the output probabilities for each SL (in %). The letters are classified relatively to: helices, coils and strands. The probabilities values are in percentage. All probabilities less than 1% have been replaced by \*. All probabilities equal to zero have been replaced by space.

	A1	A2	A3	A4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	B1	B2	B3	B4	B5
A1	70.49	21.11		*	3.01	1.63								2.89													
A2			65.06			13.09			6.03	*	2.66	*		4.47	3.08					3.15	*	1.08					
A3	51.82	27.06		3.46	9.63	5.07								1.91	*						*						
A4		7.15		74.90	2.95	4.11			*		*			6.66	*				1.37		*	*					
C1	11.38	11.71		1.75	20.75	15.15	*		13.95		2.53	1.15		5.15	7.48				*	5.55	1.53	1.06					
C2		0.65	14.95	1.65		13.54	*		10.71	3.59	6.97	4.93		7.23	7.95				1.27	15.21	3.46	5.80			1.02		
C3	0.63	3.77	3.76	1.07	16.65	8.82	1.21		10.98	7.17	4.63	20.50			4.55				2.77	2.60	7.39	1.88			1.19	*	
C4						*	13.49	12.93	*	7.92			12.88				7.33	4.16	1.30		*		1.48	11.44	15.15	8.20	2.02
C5							19.03	15.41		6.26	*		4.80			4.67	2.56	5.50	1.63				7.11	15.16	5.51	5.04	7.12
C6		1.73	3.62	*	11.78	9.84	2.03		8.72	4.06	4.26	25.73		2.02	5.64		*	0.65	2.88		5.31	5.18		*	2.11	2.62	
C7							11.43	18.20	*	6.47			5.17			1.19	6.34	4.51	*			0.68	4.20	20.04	11.84	4.58	4.03
C8							12.64	9.24		2.08			8.44			8.39	1.74	8.07	2.22				21.47	7.91		1.84	15.98
C9	10.30	13.70	7.08	4.34	30.51	9.46	*		4.96	1.69	3.95	5.52		*	1.46				*		3.32	1.77					
C10			*			*			12.96	3.89	53.78	1.22		*	1.10				3.35		6.71	16.17					
C11	1.22	1.91	13.79	1.57	17.61	11.42	*		10.83	6.14	4.46	7.84		1.73	8.61				1.44	3.64	3.26	3.51			*		
C12						*	9.10	7.67			2.75		4.32			19.20	10.05	9.46	6.88				7.49	8.40	4.16	5.49	4.72
C13			1.51		5.41	2.58			18.47	3.54	27.24	2.49		*	8.09				14.06	*	5.73	9.30					
C14	1.12		5.28	*	5.81	11.66	8.87	9.75	3.13	3.88		2.88	6.44		1.92	*	14.91		*	4.22	2.32	*		3.38	6.80	4.75	
C15			*	*	2.03	2.26			16.02	3.25	13.96	3.67		3.24	3.51		1.18		28.34	2.21	11.09	8.34					
C16			*	*		9.44	2.28		11.47	10.97	3.04	8.09		14.55	6.78		1.26			17.91	4.31	3.32		*	3.95	1.03	*
C17						*	8.54	7.13	*	5.46	*		5.13		*	23.22	9.05	7.45	5.53				3.43	8.10	7.12	3.75	4.60
C18	*	4.04	9.78	*	12.52	13.55	4.12		10.69	4.89	4.06	12.30		4.02	6.66		2.75			2.29	6.28				*		
B1						4.46	2.11		5.56				1.77			2.58	1.29	1.12	*				15.19	8.76	32.99	11.87	11.44
B2						8.94	8.77		9.69	*			10.41			2.02	2.64	5.01	*				6.96	10.76	19.03	9.85	5.16
B3						6.85	6.52		3.20				6.54			2.85	2.55	4.01	1.16				22.70	12.87	5.29	5.83	19.64
B4						7.54	15.57		*				5.88			7.59	2.93	8.58	2.55				20.44	16.46		1.93	9.69
B5						5.66	5.38		2.00				1.66			3.58	2.92	3.63	*				25.44	12.26		8.53	27.98

## References

1. Unger R, Harel D, Wherland S, Sussman JL. A 3D building blocks approach to analyzing and predicting structure of proteins. *Proteins: Structure, Function, and Bioinformatics*. 1989;5(4):355–373.
2. Camproux AC, Tufféry P, Chevrolat J, Boisvieux J, Hazout S. Hidden Markov model approach for identifying the modular framework of the protein backbone. *Protein Engineering*. 1999;12(12):1063–1073.
3. Camproux AC, Gautier R, Tufféry P. A hidden markov model derived structural alphabet for proteins. *Journal of Molecular Biology*. 2004;339(3):591–605.
4. De Brevern AG, Etchebest C, Hazout S. Bayesian probabilistic approach for predicting backbone structures in terms of protein blocks. *Proteins: Structure, Function, and Bioinformatics*. 2000;41(3):271–287.
5. Nuel G, Regad L, Martin J, Camproux AC. Exact distribution of a pattern in a set of random sequences generated by a Markov source: applications to biological data. *Algorithms for Molecular Biology*. 2010;5(1):15.
6. Smith TF, Waterman MS. Identification of common molecular subsequences. *Journal of Molecular Biology*. 1981;147(1):195–197.
7. Wang S, Zheng WM. CLePAPS: fast pair alignment of protein structures based on conformational letters. *Journal of Bioinformatics and Computational Biology*. 2008;6(02):347–366.
8. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin Of Mathematical Biophysics*. 1943;5(4):115–133.
9. Kohonen T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*. 1982;43(1):59–69.
10. Fix E, Hodges Jr JL. Discriminatory analysis-nonparametric discrimination: consistency properties. California Univ Berkeley; 1951.
11. Zhang X, Fetrow JS, Rennie WA, Waltz DL, Berg G. Automatic derivation of substructures yields novel structural building blocks in globular proteins. In: *ISMB*. vol. 1; 1993. p. 438–446.
12. Tung CH, Huang JW, Yang JM. Kappa-alpha plot derived structural alphabet and BLOSUM-like substitution matrix for rapid search of protein structure database. *Genome Biology*. 2007;8(3):R31.
13. Tung CH, Nacher JC. A Complex Network Approach for the Analysis of Protein Units Similarity Using Structural Alphabet. *International Journal of Bioscience, Biochemistry and Bioinformatics*. 2013;3(5):433–437.
14. Pandini A, Fornili A, Kleinjung J. Structural alphabets derived from attractors in conformational space. *BMC Bioinformatics*. 2010;11(1):97.
15. Camproux A, Tufféry P. Hidden Markov model-derived structural alphabet for proteins: the learning of protein local shapes captures sequence specificity. *Biochimica et Biophysica Acta (BBA)-General Subjects*. 2005;1724(3):394–403.
16. Gautier R, Camproux AC, Tufféry P. SCit: web tools for protein side chain conformation analysis. *Nucleic Acids Research*. 2004;32(suppl 2):W508–W511.
17. Guyon F, Camproux AC, Hochez J, Tufféry P. SA-Search: a web tool for protein structure mining based on a Structural Alphabet. *Nucleic Acids Research*. 2004;32(suppl 2):W545–W548.
18. Deschavanne P, Tufféry P. Enhanced protein fold recognition using a structural alphabet. *Proteins: Structure, Function, and Bioinformatics*. 2009;76(1):129–137.
19. Pandini A, Fornili A. Using local states to drive the sampling of global conformations in proteins. *Journal of Chemical Theory and Computation*. 2016;12(3):1368–1379.

20. Pandini A, Fornili A, Fraternali F, Kleinjung J. GSATools: analysis of allosteric communication and functional local motions using a structural alphabet. *Bioinformatics*. 2013;29(16):2053–2055.
21. Mahajan S, de Brevern AG, Offmann B, Srinivasan N. Correlation between local structural dynamics of proteins inferred from NMR ensembles and evolutionary dynamics of homologues of known structure. *Journal of Biomolecular Structure and Dynamics*. 2014;32(5):751–758.
22. Lamiable A, Thevenet P, Tufféry P. A critical assessment of hidden markov model sub-optimal sampling strategies applied to the generation of peptide 3D models. *Journal of Computational Chemistry*. 2016;37(21):2006–2016.
23. Lamiable A, Thévenet P, Rey J, Vavrusa M, Derreumaux P, Tufféry P. PEP-FOLD3: faster de novo structure prediction for linear peptides in solution and in complex. *Nucleic Acids Research*. 2016;44(W1):W449–W454.
24. Craveur P, Joseph AP, Esque J, Narwani TJ, Noël F, Shinada N, et al. Protein flexibility in the light of structural alphabets. *frontiers in Molecular Biosciences*. 2015;2:20.
25. Regad L, Chéron JB, Triki D, Senac C, Flatters D, Camproux AC. Exploring the potential of a structural alphabet-based tool for mining multiple target conformations and target flexibility insight. *PLOS ONE*. 2017;12(8):1–29.
26. Martin J, Regad L, Lecornet H, Camproux AC. Structural deformation upon protein-protein interaction: a structural alphabet approach. *BMC Structural Biology*. 2008;8(1):12.
27. Baussand J, Camproux AC. Deciphering the shape and deformation of secondary structures through local conformation analysis. *BMC Structural Biology*. 2011;11(1):9.
28. de Brevern AG, Bornot A, Craveur P, Etchebest C, Gelly JC. PredyFlexy: flexibility and local structure prediction from sequence. *Nucleic Acids Research*. 2012;40(W1):W317–W322.
29. Dong Q, Wang K, Liu B, Liu X. Characterization and prediction of protein flexibility based on structural alphabets. *BioMed Research International*. 2016;2016(ID 4628025):7.
30. Bernstein FC, Koetzle TF, Williams GJB, Meyer EF, Brice MD, Rodgers JR, et al. The protein data bank: A computer-based archival file for macromolecular structures. *Archives of Biochemistry and Biophysics*. 1978;185(2):584–591.
31. Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, et al. The Protein Data Bank. *Nucleic Acids Research*. 2000;28(1):235–242.
32. Goh CS, Milburn D, Gerstein M. Conformational changes associated with protein–protein interactions. *Current Opinion in Structural Biology*. 2004;14(1):104–109.
33. Grünberg R, Leckner J, Nilges M. Complementarity of structure ensembles in protein-protein binding. *Structure*. 2004;12(12):2125–2136.
34. Lensink MF, Mendez R. Recognition-induced conformational changes in protein-protein docking. *Current Pharmaceutical Biotechnology*. 2008;9(2):77–86.
35. Marsh JA, Teichmann SA. Protein flexibility facilitates quaternary structure assembly and evolution. *PLOS Biology*. 2014;12(5):1–11.
36. Marsh JA, Teichmann SA. Structure, dynamics, assembly, and evolution of protein complexes. *Annual Review of Biochemistry*. 2015;84:551–575.
37. Sormanni P, Piovesan D, Heller GT, Bonomi M, Kukic P, Camilloni C, et al. Simultaneous quantification of protein order and disorder. *Nature Chemical Biology*. 2017;13(4):339–342.

38. Gall TL, Romero PR, Cortese MS, Uversky VN, Dunker AK. Intrinsic Disorder in the Protein Data Bank. *Journal of Biomolecular Structure and Dynamics*. 2007;24(4):325–341.
39. Regad L, Guyon F, Maupetit J, Tufféry P, Camproux AC. A Hidden Markov Model applied to the protein 3D structure analysis. *Computational Statistics & Data Analysis*. 2008;52(6):3198–3207.
40. Regad L, Martin J, Nuel G, Camproux AC. Mining protein loops using a structural alphabet and statistical exceptionality. *BMC Bioinformatics*. 2010;11(1):75.
41. Berman H, Henrick K, Nakamura H. Announcing the worldwide protein data bank. *Nature Structural & Molecular Biology*. 2003;10(12):980–980.
42. Berman H, Henrick K, Nakamura H, Markley JL. The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data. *Nucleic Acids Research*. 2007;35(suppl\_1):D301–D303.
43. Frishman D, Argos P. Knowledge-based protein secondary structure assignment. *Proteins: Structure, Function, and Bioinformatics*. 1995;23(4):566–579.
44. Cho JH, Meng W, Sato S, Kim EY, Schindelin H, Raleigh DP. Energetically significant networks of coupled interactions within an unfolded protein. *Proceedings of the National Academy of Sciences*. 2014;111(33):12079–12084.
45. Griep S, Hobohm U. PDBselect 1992–2009 and PDBfilter-select. *Nucleic Acids Research*. 2009;38(suppl\_1):D318–D319.
46. Adams-Cioaba MA, Guo Y, Bian C, Amaya MF, Lam R, Wasney GA, et al. Structural studies of the tandem Tudor domains of fragile X mental retardation related proteins FXR1 and FXR2. *PLOS ONE*. 2010;5(11):1–9.
47. Buchan DW, Minneci F, Nugent TC, Bryson K, Jones DT. Scalable web services for the PSIPRED Protein Analysis Workbench. *Nucleic Acids Research*. 2013;41(W1):W349–W357.
48. Jones DT. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology*. 1999;292(2):195–202.
49. Poulain P, Gelly JC, Flatters D. Detection and architecture of small heat shock protein monomers. *PLOS ONE*. 2010;5(4):1–10.
50. van Montfort RL, Basha E, Friedrich KL, Slingsby C, Vierling E. Crystal structure and assembly of a eukaryotic small heat shock protein. *Nature Structural & Molecular Biology*. 2001;8(12):1025–1030.
51. Frommlet F, Nuel G. An Adaptive Ridge Procedure for L0 Regularization. *PLOS ONE*. 2016;11(2):1–23.

# Bibliographie

- [1] ADAMS-CIOABA, M. A., GUO, Y., BIAN, C., AMAYA, M. F., LAM, R., WASNEY, G. A., VEDADI, M., XU, C., AND MIN, J. Structural studies of the tandem tudor domains of fragile x mental retardation related proteins fxr1 and fxr2. *PLOS ONE* 5, 11 (2010), 1–9.
- [2] ALLAM, I., FLATTERS, D., L, R., CAMPROUX, A. C., AND NUEL, G. On the interest of semi-supervised approaches with spacial dependence in structural alphabet encoding. In *Proceedings of 23rd Annual International Conference on Intelligent Systems for Molecular Biology and the 14th European Conference on Computational Biology. July 10-14 , 2015, Spring Joint Computer Conference* (2015).
- [3] ALTSCHUL, S. F., MADDEN, T. L., SCHÄFFER, A. A., ZHANG, J., ZHANG, Z., MILLER, W., AND LIPMAN, D. J. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research* 25, 17 (1997), 3389–3402.
- [4] AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (New York, NY, USA, 1967), AFIPS '67 (Spring), ACM, pp. 483–485.
- [5] BAETEN, L., REUMERS, J., TUR, V., STRICHER, F., LENAERTS, T., SERRANO, L., ROUSSEAU, F., AND SCHYMKOWITZ, J. Reconstruction of Protein Backbones from the BriX Collection of Canonical Protein Fragments. *PLOS Computational Biology* 4, 5 (2008), 1–11.
- [6] BATEMAN, A., COIN, L., DURBIN, R., FINN, R. D., HOLLICH, V., GRIFFITHS-JONES, S., KHANNA, A., MARSHALL, M., MOXON, S., SONNHAMMER, E. L. L., STUDHOLME, D. J., YEATS, C., AND EDDY, S. R. The pfam protein families database. *Nucleic Acids Research* 32, suppl\_1 (2004), D138–D141.
- [7] BAUM, L. E. An inequality and associated maximization thechnique in statistical estimation for probabilistic functions of markov process. *Inequalities* 3 (1972), 1–8.
- [8] BAUM, L. E., PETRIE, T., SOULES, G., AND WEISS, N. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics* 41, 1 (1970), 164–171.

- [9] BAUSSAND, J., AND CAMPROUX, A. C. Deciphering the shape and deformation of secondary structures through local conformation analysis. *BMC Structural Biology* 11, 1 (2011), 9.
- [10] BENROS, C., DE BREVERN, A. G., ETCHEBEST, C., AND HAZOUT, S. Assessing a novel approach for predicting local 3d protein structures from sequence. *Proteins: Structure, Function, and Bioinformatics* 62, 4 (2006), 865–880.
- [11] BERMAN, H., HENRICK, K., NAKAMURA, H., AND MARKLEY, J. L. The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data. *Nucleic Acids Research* 35, suppl\_1 (2007), D301–D303.
- [12] BERMAN, H. M. The Protein Data Bank: a historical perspective. *Acta Crystallographica Section A: Foundations of Crystallography* 64, 1 (2008), 88–95.
- [13] BERMAN, H. M., KLEYWEGT, G. J., NAKAMURA, H., AND MARKLEY, J. L. How Community Has Shaped the Protein Data Bank. *Structure* 21, 9 (2013), 1485–1491.
- [14] BERMAN, J. J. *Perl Programming for Medicine and Biology*. Jones and Bartlett series in biomedical informatics. Jones and Bartlett Publishers, 2007.
- [15] BERNSTEIN, F. C., KOETZLE, T. F., WILLIAMS, G. J. B., MEYER, E. F., BRICE, M. D., RODGERS, J. R., KENNARD, O., SHIMANOUCHI, T., AND TASUMI, M. The protein data bank. *European Journal of Biochemistry* 80, 2 (1977), 319–324.
- [16] BERNSTEIN, F. C., KOETZLE, T. F., WILLIAMS, G. J. B., MEYER, E. F., BRICE, M. D., RODGERS, J. R., KENNARD, O., SHIMANOUCHI, T., AND TASUMI, M. The protein data bank: A computer-based archival file for macromolecular structures. *Archives of Biochemistry and Biophysics* 185, 2 (1978), 584–591.
- [17] BIASINI, M., SCHMIDT, T., BIENERT, S., MARIANI, V., STUDER, G., HAAS, J., JOHNER, N., SCHENK, A. D., PHILIPPSEN, A., AND SCHWEDE, T. OpenStructure: an integrated software framework for computational structural biology. *Acta Crystallographica Section D: Biological Crystallography* 69, 5 (2013), 701–709.
- [18] BILLINGSLEY, P. *Statistical inference for Markov processes*. Statistical Research Monographs. University of Chicago Press, 1961.
- [19] BLUMOFE, R. D., JOERG, C. F., KUSZMAUL, B. C., LEISERSON, C. E., RANDALL, K. H., AND ZHOU, Y. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing* 37, 1 (1996), 55–69.
- [20] BORNOT, A. *Analyse et prédiction de la relation séquence - structure locale et flexibilité au sein des protéines globulaires*. PhD thesis, Université Paris-Diderot - Paris VII, 2009.
- [21] BORNOT, A., ETCHEBEST, C., AND DE BREVERN, A. G. Predicting protein flexibility through the prediction of local structures. *Proteins: Structure, Function, and Bioinformatics* 79, 3 (2011), 839–852.

- [22] BUCHAN, D. W. A., MINNECI, F., NUGENT, T. C. O., BRYSON, K., AND JONES, D. T. Scalable web services for the psipred protein analysis workbench. *Nucleic Acids Research* 41, W1 (2013), W349–W357.
- [23] BUDOWSKI-TAL, I., NOV, Y., AND KOLODNY, R. Fragbag, an accurate representation of protein structure, retrieves structural neighbors from the entire pdb quickly and accurately. *Proceedings of the National Academy of Sciences* 107, 8 (2010), 3481–3486.
- [24] BYSTROFF, C., AND BAKER, D. Prediction of local structure in proteins using a library of sequence-structure motifs. *Journal of Molecular Biology* 281, 3 (1998), 565–577.
- [25] BYSTROFF, C., THORSSON, V., AND BAKER, D. Hmmstr: a hidden markov model for local sequence-structure correlations in proteins. *Journal of Molecular Biology* 301, 1 (2000), 173–190.
- [26] BYSTROFF, C., THORSSON, V., AND BAKER, D. HMMSTR: a hidden markov model for local sequence-structure correlations in proteins. *Journal of Molecular Biology* 301, 1 (2000), 173–190.
- [27] CAMPROUX, A. C., DE BREVERN, A. G., HAZOUT, S., AND TUFFÉRY, P. Exploring the use of a structural alphabet for structural prediction of protein loops. *Theoretical Chemistry Accounts* 106, 1-2 (2001), 28–35.
- [28] CAMPROUX, A. C., GAUTIER, R., AND TUFFÉRY, P. A hidden markov model derived structural alphabet for proteins. *Journal of Molecular Biology* 339, 3 (2004), 591–605.
- [29] CAMPROUX, A. C., AND TUFFÉRY, P. Hidden Markov Model-derived structural alphabet for proteins: The learning of protein local shapes captures sequence specificity. *Biochimica et Biophysica Acta (BBA) - General Subjects* 1724, 3 (2005), 394–403.
- [30] CAMPROUX, A. C., AND TUFFÉRY, P. Hidden markov model-derived structural alphabet for proteins: the learning of protein local shapes captures sequence specificity. *Biochimica et Biophysica Acta (BBA)-General Subjects* 1724, 3 (2005), 394–403.
- [31] CAMPROUX, A. C., TUFFÉRY, P., CHEVROLAT, J. P., BOISVIEUX, J. F., AND HAZOUT, S. Hidden markov model approach for identifying the modular framework of the protein backbone. *Protein Engineering* 12, 12 (1999), 1063–1073.
- [32] CAPPELLO, F., AND SANSONNET, J. P. Introduction au parallélisme et aux architectures parallèles. *Techniques de l'ingénieur. Informatique*, H1088 (1999), H1088–1.
- [33] CHANDRA, R. *Parallel Programming in OpenMP*. High performance computing. Morgan Kaufmann Publishers, 2001.
- [34] CHANG, R., AND HANCOCK, J. On receiver structures for channels having memory. *IEEE Transactions on Information Theory* 12, 4 (1966), 463–468.
- [35] CHANG, W., SHINDYALOV, I. N., PU, C., AND BOURNE, P. E. Design and application of PD-Blib, a C++ macromolecular class library. *Computer applications in the biosciences : CABIOS* 10, 6 (1994), 575–586.



- [36] CHAPMAN, B., AND CHANG, J. Biopython: Python Tools for Computational Biology. *SIGBIO Newsl.* 20, 2 (2000), 15–19.
- [37] CHASMAN, D. *Protein Structure: Determination, Analysis, and Applications for Drug Discovery*. CRC Press, 2003, ch. identifying errors in three-dimensional protein models.
- [38] CHO, J. H., MENG, W., SATO, S., KIM, E. Y., SCHINDELIN, H., AND RALEIGH, D. P. Energetically significant networks of coupled interactions within an unfolded protein. *Proceedings of the National Academy of Sciences* 111, 33 (2014), 12079–12084.
- [39] CLAESSENS, M., CUTSEM, E. V., LASTERS, I., AND WODAK, S. Modelling the polypeptide backbone with ‘spare parts’ from known protein structures. *Protein Engineering* 2, 5 (1989), 335–345.
- [40] COHEN, W. E. Tuning programs with oprofile. *Wide Open Magazine* 1 (2004), 53–62.
- [41] COVER, T., AND HART, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (1967), 21–27.
- [42] CRAVEUR, P., JOSEPH, A. P., ESQUE, J., NARWANI, T. J., NOËL, F., SHINADA, N., GOGUET, M., LEONARD, S., POULAIN, P., BERTRAND, O., ET AL. Protein flexibility in the light of structural alphabets. *frontiers in Molecular Biosciences* 2 (2015), 20.
- [43] DALGAARD, P. *Introductory Statistics with R*. Statistics and Computing. Springer New York, 2008.
- [44] DE BREVERN, A. G. *Nouvelles stratégies d’analyses et de prédiction des structures tridimensionnelles des protéines*. phdthesis, Université Paris-Diderot - Paris VII, 2001.
- [45] DE BREVERN, A. G. New Assessment of a Structural Alphabet. *In Silico Biology* 5, 3 (2005), 283–289.
- [46] DE BREVERN, A. G., BENROS, C., GAUTIER, R., VALADIÉ, H., HAZOUT, S., AND ETCHEBEST, C. Local backbone structure prediction of proteins. *In silico biology* 4, 3 (2004), 381–386.
- [47] DE BREVERN, A. G., BORNOT, A., CRAVEUR, P., ETCHEBEST, C., AND GELLY, J. C. Predyflexy: flexibility and local structure prediction from sequence. *Nucleic Acids Research* 40, W1 (2012), W317–W322.
- [48] DE BREVERN, A. G., ETCHEBEST, C., AND HAZOUT, S. Bayesian probabilistic approach for predicting backbone structures in terms of protein blocks. *Proteins: Structure, Function, and Bioinformatics* 41, 3 (2000), 271–287.
- [49] DE BREVERN, A. G., VALADIÉ, H., HAZOUT, S., AND ETCHEBEST, C. Extension of a local backbone description using a structural alphabet: A new approach to the sequence-structure relationship. *Protein Science* 11, 12 (2002), 2871–2886.

- [50] DE BREVERN, A. G., WONG, H., TOURNAMILLE, C., COLIN, Y., LE VAN KIM, C., AND ETCHEBEST, C. A structural model of a seven-transmembrane helix receptor: the duffy antigen/receptor for chemokine (darc). *Biochimica et Biophysica Acta (BBA)-General Subjects* 1724, 3 (2005), 288–306.
- [51] DE OLIVEIRA, S. H. P., SHI, J., AND DEANE, C. M. Building a Better Fragment Library for De Novo Protein Structure Prediction. *PLOS ONE* 10, 4 (2015).
- [52] DELYON, B., LAVIELLE, M., AND MOULINES, E. Convergence of a stochastic approximation version of the EM algorithm. *Annals of statistics* (1999), 94–128.
- [53] DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)* (1977), 1–38.
- [54] DESCHAVANNE, P., AND TUFFÉRY, P. Enhanced protein fold recognition using a structural alphabet. *Proteins: Structure, Function, and Bioinformatics* 76, 1 (2009), 129–137.
- [55] DONG, Q., WANG, K., LIU, B., AND LIU, X. Characterization and prediction of protein flexibility based on structural alphabets. *BioMed Research International* 2016, ID 4628025 (2016), 7.
- [56] DONG, Q., WANG, X., AND LIN, L. Prediction of protein local structures and folding fragments based on building-block library. *Proteins: Structure, Function, and Bioinformatics* 72, 1 (2008), 353–366.
- [57] DONG, Q. W., WANG, X. L., AND LIN, L. Methods for optimizing the structure alphabet sequences of proteins. *Computers in Biology and Medicine* 37, 11 (2007), 1610–1616.
- [58] DREYFUS, E. *BSD: Les dessous d’Unix*. Cahiers de l’Admin. Eyrolles, 2011.
- [59] DUDEV, M., AND LIM, C. Discovering structural motifs using a structural alphabet: application to magnesium-binding sites. *BMC bioinformatics* 8, 1 (2007), 1.
- [60] DUPOUY, B. *Assembleur et éditeur de liens*. Ed. Techniques Ingénieur, 2002.
- [61] DURBIN, R., EDDY, S. R., KROGH, A., AND MITCHISON, G. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.
- [62] DWYER, P. S. Some applications of matrix derivatives in multivariate analysis. *Journal of the American Statistical Association* 62, 318 (1967), 607–625.
- [63] ETCHEBEST, C., BENROS, C., HAZOUT, S., AND DE BREVERN, A. G. A structural alphabet for local protein structures: Improved prediction methods. *Proteins: Structure, Function, and Bioinformatics* 59, 4 (2005), 810–827.

- [64] FETROW, J. S., PALUMBO, M. J., AND BERG, G. Patterns, structures, and amino acid frequencies in structural building blocks, a protein secondary structure classification scheme. *Proteins: Structure, Function, and Bioinformatics* 27, 2 (1997), 249–271.
- [65] FIX, E., AND HODGES, J. L. Discriminatory analysis-nonparametric discrimination: consistency properties. Tech. rep., California Univ Berkeley, 1951.
- [66] FLATER, D. *Configuration of profiling tools for C/C++ applications under 64-bit Linux*. US Department of Commerce, National Institute of Standards and Technology, 2013.
- [67] FOURRIER, L., BENROS, C., AND DE BREVERN, A. G. Use of a structural alphabet for analysis of short loops connecting repetitive structures. *BMC bioinformatics* 5, 1 (2004), 58.
- [68] FRANÇOIS, D. *Les probabilités et la statistique de A à Z - 500 définitions, formules et tests d'hypothèse*. Dunod, 2007.
- [69] FRISHMAN, D., AND ARGOS, P. Knowledge-based protein secondary structure assignment. *Proteins: Structure, Function, and Bioinformatics* 23, 4 (1995), 566–579.
- [70] GALL, T. L., ROMERO, P. R., CORTESE, M. S., UVERSKY, V. N., AND DUNKER, A. K. Intrinsic Disorder in the Protein Data Bank. *Journal of Biomolecular Structure and Dynamics* 24, 4 (2007), 325–341.
- [71] GAMMA, E., AND BECK, K. *Contributing to Eclipse: Principles, Patterns, and Plug-ins*. Eclipse series. Addison-Wesley, 2004.
- [72] GAUTIER, R., CAMPROUX, A. C., AND TUFFÉRY, P. SCit: web tools for protein side chain conformation analysis. *Nucleic Acids Research* 32, suppl 2 (2004), W508–W511.
- [73] GELLY, J. C., JOSEPH, A. P., SRINIVASAN, N., AND DE BREVERN, A. G. ipba: a tool for protein structure comparison using sequence alignment strategies. *Nucleic Acids Research* 39, suppl 2 (2011), W18–W23.
- [74] GHOUZAM, Y., POSTIC, G., DE BREVERN, A. G., AND GELLY, J. C. Improving protein fold recognition with hybrid profiles combining sequence and structure evolution. *Bioinformatics* (2015), 3782–3789.
- [75] GILLE, C., AND FROMMEL, C. STRAP: editor for STRuctural Alignments of Proteins. *Bioinformatics* 17, 4 (2001), 377–378.
- [76] GOSLING, J., JOY, B., STEELE, G. L., BRACHA, G., AND BUCKLEY, A. *The Java Language Specification, Java SE 8 Edition*. Java Series. Pearson Education, 2014.
- [77] GOTO, N., NAKAO, M., KAWASHIMA, S., KATAYAMA, T., AND KANEHISA, M. BioRuby: OpenSource Bioinformatics Library. *Genome Informatics* 14 (2003), 629–630.
- [78] GRIEP, S., AND HOBOTHM, U. Pdbselect 1992–2009 and pdbfilter-select. *Nucleic Acids Research* 38, suppl\_1 (2009), D318–D319.

- [79] GUYON, F., CAMPROUX, A. C., HOCHÉZ, J., AND TUFFÉRY, P. SA-Search: a web tool for protein structure mining based on a Structural Alphabet. *Nucleic Acids Research* 32, suppl 2 (2004), W545–W548.
- [80] HALL, S. R., AND MCMAHON, B. *International Tables for Crystallography, Definition and Exchange of Crystallographic Data*. Springer Science & Business Media, 2005.
- [81] HAMELRYCK, T., AND MANDERICK, B. PDB file parser and structure class implemented in Python. *Bioinformatics* 19, 17 (2003), 2308–2310.
- [82] HENNESSY, J. L., AND PATTERSON, D. *Architecture des ordinateurs: une approche quantitative*. Vuibert, 2001.
- [83] HILDEBRANDT, A., DEHOF, A. K., RURAINSKI, A., BERTSCH, A., SCHUMANN, M., TOUS-SAINTE, N. C., MOLL, A., STÖCKEL, D., NICKELS, S., MUELLER, S. C., LENHOF, H. P., AND KOHLBACHER, O. Ball - biochemical algorithms library 1.3. *BMC Bioinformatics* 11 (2010), 531.
- [84] HINSEN, K. The molecular modeling toolkit: A new approach to molecular simulations. *Journal of Computational Chemistry* 21, 2 (2000), 79–85.
- [85] HIRSH, L., PIOVESAN, D., GIOLLO, M., FERRARI, C., AND TOSATTO, S. C. E. The Victor C++ library for protein representation and advanced manipulation. *Bioinformatics* 31, 7 (2015), 1138–1140.
- [86] HOBOHM, U., AND SANDER, C. Enlarged representative set of protein structures. *Protein Science* 3, 3 (1994), 522–524.
- [87] HOLLAND, R. C. G., DOWN, T. A., POCOCC, M., PRLIC, A., HUEN, D., JAMES, K., FOISY, S., DRAGER, A., YATES, A., HEUER, M., AND SCHREIBER, M. J. BioJava: an open source framework for bioinformatics. *Bioinformatics* 24, 18 (2008), 2096–2097.
- [88] HUNTER, C. G., AND SUBRAMANIAM, S. Protein fragment clustering and canonical local shapes. *Proteins: Structure, Function, and Bioinformatics* 50, 4 (2003), 580–588.
- [89] JAMBU, M., AND LEBEAUX, M. O. *Classification automatique pour l’analyse des donnees / Clc*, vol. 1. Paris : Dunod, 1978.
- [90] JIANZHU, M., AND SHENG, W. Algorithms, applications, and challenges of protein structure alignment. In *Advances in Protein Chemistry and Structural Biology*, vol. 94. Academic Press, 2014, 2014, p. 488.
- [91] JONES, D. T. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology* 292, 2 (1999), 195–202.
- [92] JONES, T. A., AND THIRUP, S. Using known substructures in protein model building and crystallography. *The EMBO Journal* 5, 4 (1986), 819–822.

- [93] KABSCH, W., AND SANDER, C. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* 22, 12 (1983), 2577–2637.
- [94] KALEV, I., AND HABECK, M. HHfrag: HMM-based fragment detection using HHpred. *Bioinformatics* 27, 22 (2011), 3110–3116.
- [95] KERNIGHAN, B., RITCHIE, D., AND BUFFENOIR, T. *Le langage C*. Manuels informatiques Masson. Masson, 1983.
- [96] KHREICH, W., GRANGER, E., MIRI, A., AND SABOURIN, R. On the memory complexity of the forward-backward algorithm. *Pattern Recognition Letters* 31, 2 (2010), 91–99.
- [97] KINJO, A. R., SUZUKI, H., YAMASHITA, R., IKEGAWA, Y., KUDOU, T., IGARASHI, R., KENGAKU, Y., CHO, H., STANDLEY, D. M., NAKAGAWA, A., AND NAKAMURA, H. Protein Data Bank Japan (PDBj): maintaining a structural data archive and resource description framework format. *Nucleic Acids Research* (2011), D453–D460.
- [98] KIRCHMAIR, J., MARKT, P., DISTINTO, S., SCHUSTER, D., SPITZER, G. M., LIEDL, K. R., LANGER, T., AND WOLBER, G. The Protein Data Bank (PDB), Its Related Services and Software Tools as Key Components for In Silico Guided Drug Discovery. *Journal of Medicinal Chemistry* 51, 22 (2008), 7021–7040.
- [99] KOHLBACHER, O., AND LENHOF, H.-P. BALLrapid software prototyping in computational molecular biology. *Bioinformatics* 16, 9 (2000), 815–824.
- [100] KOLODNY, R., KOEHL, P., GUIBAS, L., AND LEVITT, M. Small Libraries of Protein Fragments Model Native Protein Structures Accurately. *Journal of Molecular Biology* 323, 2 (2002), 297–307.
- [101] KROGH, A., BROWN, M., MIAN, I. S., SJÖLANDER, K., AND HAUSSLER, D. Hidden markov models in computational biology: Applications to protein modeling. *Journal of molecular biology* 235, 5 (1994), 1501–1531.
- [102] KU, S.-Y., AND HU, Y.-J. Protein structure search and local structure characterization. *BMC Bioinformatics* 9 (2008), 349.
- [103] KU, S. Y., AND HU, Y. J. Structural alphabet motif discovery and a structural motif database. *Computers in biology and medicine* 42, 1 (2012), 93–105.
- [104] LAMIABLE, A., THÉVENET, P., REY, J., VAVRUSA, M., DERREUMAUX, P., AND TUFFÉRY, P. Pep-fold3: faster de novo structure prediction for linear peptides in solution and in complex. *Nucleic Acids Research* 44, W1 (2016), W449–W454.
- [105] LAMIABLE, A., THEVENET, P., AND TUFFÉRY, P. A critical assessment of hidden markov model sub-optimal sampling strategies applied to the generation of peptide 3d models. *Journal of Computational Chemistry* 37, 21 (2016), 2006–2016.

- [106] LE, Q., POLLASTRI, G., AND KOEHL, P. Structural alphabets for protein structure classification: A comparison study. *Journal of Molecular Biology* 387, 2 (2009), 431–450.
- [107] LÉONARD, S., JOSEPH, A. P., SRINIVASAN, N., GELLY, J. C., AND DE BREVERN, A. G. mulpba: an efficient multiple protein structure alignment method based on a structural alphabet. *Journal of Biomolecular Structure and Dynamics* 32, 4 (2014), 661–668.
- [108] LEVITT, M. Accurate modeling of protein conformation by automatic segment matching. *Journal of Molecular Biology* 226, 2 (1992), 507–533.
- [109] LIPTAY, J. S. Structural aspects of the system/360 model 85, ii: The cache. *IBM Systems Journal* 7, 1 (1968), 15–21.
- [110] LORIOT, S., CAZALS, F., AND BERNAUER, J. ESBTL: efficient PDB parser and data structure for the structural and geometric analysis of biological macromolecules. *Bioinformatics* 26, 8 (2010), 1127–1128.
- [111] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability* (1967), vol. 1, The Regents of the University of California.
- [112] MADEJ, T., ADDESS, K. J., FONG, J. H., GEER, L. Y., GEER, R. C., LANCZYCKI, C. J., LIU, C., LU, S., MARCHLER BAUER, A., PANCHENKO, A. R., CHEN, J., THIESSEN, P. A., WANG, Y., ZHANG, D., AND BRYANT, S. H. MMDB: 3d structures and macromolecular interactions. *Nucleic Acids Research* 40, Database issue (2012), D461–D464.
- [113] MADEJ, T., LANCZYCKI, C. J., ZHANG, D., THIESSEN, P. A., GEER, R. C., MARCHLER BAUER, A., AND BRYANT, S. H. MMDB and VAST+: tracking structural similarities between macromolecular complexes. *Nucleic Acids Research* 42, D1 (2013), D297–D303.
- [114] MAHAJAN, S., DE BREVERN, A. G., OFFMANN, B., AND SRINIVASAN, N. Correlation between local structural dynamics of proteins inferred from nmr ensembles and evolutionary dynamics of homologues of known structure. *Journal of Biomolecular Structure and Dynamics* 32, 5 (2014), 751–758.
- [115] MARSH, J. A., AND TEICHMANN, S. A. Protein flexibility facilitates quaternary structure assembly and evolution. *PLOS Biology* 12, 5 (2014), 1–11.
- [116] MARSH, J. A., AND TEICHMANN, S. A. Structure, dynamics, assembly, and evolution of protein complexes. *Annual Review of Biochemistry* 84 (2015), 551–575.
- [117] MARTIN, J., DE BREVERN, A. G., AND CAMPROUX, A. C. In silico local structure approach: a case study on outer membrane proteins. *Proteins: Structure, Function, and Bioinformatics* 71, 1 (2008), 92–109.

- [118] MARTIN, J., REGAD, L., ETCHEBEST, C., AND CAMPROUX, A. C. Taking advantage of local structure descriptors to analyze interresidue contacts in protein structures and protein complexes. *Proteins: Structure, Function, and Bioinformatics* 73, 3 (2008), 672–689.
- [119] MARTIN, J., REGAD, L., LECORNET, H., AND CAMPROUX, A. C. Structural deformation upon protein-protein interaction: a structural alphabet approach. *BMC Structural Biology* 8, 1 (2008), 12.
- [120] MAUPETIT, J., DERREUMAUX, P., AND TUFFERY, P. Pep-fold: an online resource for de novo peptide structure prediction. *Nucleic Acids Research* (2009), W498–W503.
- [121] MAUPETIT, J., GAUTIER, R., AND TUFFERY, P. Sabbac: online structural alphabet-based protein backbone reconstruction from alpha-carbon trace. *Nucleic Acids Research* 34, suppl 2 (2006), W147–W151.
- [122] MCCULLOCH, W. S., AND PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin Of Mathematical Biophysics* 5, 4 (1943), 115–133.
- [123] MICHELETTI, C., SENO, F., AND MARITAN, A. Recurrent oligomers in proteins: An optimal scheme reconciling accurate and concise backbone representations in automated folding and design studies. *Proteins: Structure, Function, and Bioinformatics* 40, 4 (2000), 662–674.
- [124] MODAK, B. K. *C++ Application Development with Code::Blocks*. Packt Publishing, 2013.
- [125] NETHERCOTE, N., AND SEWARD, J. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan notices* (2007), vol. 42, ACM, pp. 89–100.
- [126] NIVER, H. M. *Getting to Know Ruby*. Code Power: a Teen Programmer’s Guide Series. Rosen Publishing Group, Incorporated, 2014.
- [127] OFFMANN, B., TYAGI, M., AND DE BREVERN, A. G. Local Protein Structures. *Current Bioinformatics* 2, 3 (2007), 165–202.
- [128] OKUN, V., DELAITRE, A., AND BLACK, P. E. Report on the static analysis tool exposition (sate) iv. *NIST Special Publication 500* (2013), 297.
- [129] PANDINI, A., BONATI, L., FRATERALI, F., AND KLEINJUNG, J. MinSet: a general approach to derive maximally representative database subsets by using fragment dictionaries and its application to the SCOP database. *Bioinformatics* 23, 4 (2007), 515–516.
- [130] PANDINI, A., AND FORNILI, A. Using local states to drive the sampling of global conformations in proteins. *Journal of Chemical Theory and Computation* 12, 3 (2016), 1368–1379.
- [131] PANDINI, A., FORNILI, A., FRATERALI, F., AND KLEINJUNG, J. Gsatools: analysis of allosteric communication and functional local motions using a structural alphabet. *Bioinformatics* 29, 16 (2013), 2053–2055.

- [132] PANDINI, A., FORNILI, A., AND KLEINJUNG, J. Structural alphabets derived from attractors in conformational space. *BMC Bioinformatics* 11, 1 (2010), 97.
- [133] PANG, X., ZHOU, L., ZHANG, M., XIE, F., YU, L., ZHANG, L., XU, L., AND ZHANG, X. A mathematical model for peptide inhibitor design. *Journal of Computational Biology* 17, 8 (2010), 1081–1093.
- [134] PAULING, L., COREY, R. B., AND BRANSON, H. R. The structure of proteins: two hydrogen-bonded helical configurations of the polypeptide chain. *Proceedings of the National Academy of Sciences* 37, 4 (1951), 205–211.
- [135] PITT, W. R., WILLIAMS, M. A., STEVEN, M., SWEENEY, B., BLEASBY, A. J., AND MOSS, D. S. The Bioinformatics Template Library generic components for biocomputing. *Bioinformatics* 17, 8 (2001), 729–737.
- [136] PORTER, C. T., AND MARTIN, A. C. R. BiopLib and BiopTools a C programming library and toolset for manipulating protein structure. *Bioinformatics* 31, 24 (2015), 4017–4019.
- [137] POTTERTON, E., MCNICHOLAS, S., KRISSINEL, E., COWTAN, K., AND NOBLE, M. The CCP4 molecular-graphics project. *Acta Crystallographica Section D: Biological Crystallography* 58, 11 (2002), 1955–1957.
- [138] POULAIN, P., GELLY, J. C., AND FLATTERS, D. Detection and architecture of small heat shock protein monomers. *PLOS ONE* 5, 4 (2010), 1–10.
- [139] PRESTRELSKI, S. J., WILLIAMS, A. L., AND LIEBMAN, M. N. Generation of a substructure library for the description and classification of protein secondary structure. I. Overview of the methods and results. *Proteins: Structure, Function, and Bioinformatics* 14, 4 (1992), 430–439.
- [140] PRLIC, A., YATES, A., BLIVEN, S. E., ROSE, P. W., JACOBSEN, J., TROSHIN, P. V., CHAPMAN, M., GAO, J., KOH, C. H., FOISY, S., HOLLAND, R., RIMSA, G., HEUER, M. L., BRANDSTATTER MULLER, H., BOURNE, P. E., AND WILLIS, S. BioJava: an open source framework for bioinformatics in 2012. *Bioinformatics* 28, 20 (2012), 2693–2695.
- [141] RABINER, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77, 2 (1989), 257–286.
- [142] REGAD, L. *Analyse statistique des boucles protéiques : développement d’une méthode d’extraction systématique de motifs récurrents au sein des régions en boucles*. PhD thesis, Paris 7, 2008.
- [143] REGAD, L., CHÉRON, J. B., TRIKI, D., SENAC, C., FLATTERS, D., AND CAMPROUX, A. C. Exploring the potential of a structural alphabet-based tool for mining multiple target conformations and target flexibility insight. *PLOS ONE* 12, 8 (2017), 1–29.
- [144] REGAD, L., GUYON, F., MAUPETIT, J., TUFFÉRY, P., AND CAMPROUX, A. C. A hidden markov model applied to the protein 3d structure analysis. *Computational Statistics & Data Analysis* 52, 6 (2008), 3198–3207.



- [145] REGAD, L., MARTIN, J., AND CAMPROUX, A. C. Identification of non random motifs in loops using a structural alphabet. In *2006 IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology* (2006), IEEE, pp. 1–9.
- [146] REGAD, L., MARTIN, J., NUEL, G., AND CAMPROUX, A. C. Mining protein loops using a structural alphabet and statistical exceptionality. *BMC Bioinformatics* 11, 1 (2010), 75.
- [147] REGAD, L., SALADIN, A., MAUPETIT, J., GENEIX, C., AND CAMPROUX, A. C. SA-Mot: a web server for the identification of motifs of interest extracted from protein loops. *Nucleic Acids Research* 39 (2011), W203–W209.
- [148] REYNÈS, C., REGAD, L., PÉROT, S., NUEL, G., AND CAMPROUX, A. C. Application of hmm to the study of three-dimensional protein structure. *Hidden Markov Models, Theory and Applications* (2011), 269–290.
- [149] RICE, P., LONGDEN, I., AND BLEASBY, A. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics* 16, 6 (2000), 276–277.
- [150] RITCHIE, D. W. Calculating and scoring high quality multiple flexible protein structure alignments. *Bioinformatics* 32, 17 (2016), 2650–2658.
- [151] ROBSON, R. *Using the STL: The C++ Standard Template Library*. Springer New York, 2012.
- [152] ROOMAN, M. J., RODRIGUEZ, J., AND WODAK, S. J. Automatic definition of recurrent local structure motifs in proteins. *Journal of Molecular Biology* 213, 2 (1990), 327–336.
- [153] ROOMAN, M. J., RODRIGUEZ, J., AND WODAK, S. J. Relations between protein sequence and structure and their significance. *Journal of Molecular Biology* 213, 2 (1990), 337–350.
- [154] SALAH, A., AND LI, K. Par-3d-blast: A parallel tool for searching and aligning protein structures. *Concurrency and Computation: Practice and Experience* 26, 10 (2014), 1705–1714.
- [155] SANDER, O., SOMMER, I., AND LENGAUER, T. Local protein structure prediction using discriminative models. *BMC Bioinformatics* 7, 1 (2006), 14.
- [156] SCHÄLING, B. *The boost C++ libraries*. Boris Schäling, 2011.
- [157] SCHRÖDINGER, LLC. The JyMOL molecular graphics development component, version 1.8. 2015.
- [158] SCHRÖDINGER, LLC. The PyMOL molecular graphics system, version 1.8. 2015.
- [159] SCHUCHHARDT, J., SCHNEIDER, G., REICHEL, J., SCHOMBURG, D., AND WREDE, P. Local structural motifs of protein backbones are classified by self-organizing neural networks. *Protein Engineering* 9, 10 (1996), 833–842.
- [160] SCHWARZ, G., ET AL. Estimating the dimension of a model. *The annals of statistics* 6, 2 (1978), 461–464.

- [161] SEGALL, R. S. *Research and Applications in Global Supercomputing*. Advances in Systems Analysis, Software Engineering, and High Performance Computing:. IGI Global, 2015.
- [162] SHANNON, C. E. A mathematical theory of communication, part i, part ii. *Bell Syst. Tech. J.* 27 (1948), 623–656.
- [163] SHEN, S. *Theory and Mathematical Methods in Bioinformatics*. Biological and Medical Physics, Biomedical Engineering. Springer Berlin Heidelberg, 2008.
- [164] SHEN, Y., PICORD, G., GUYON, F., AND TUFFERY, P. Detecting protein candidate fragments using a structural alphabet profile comparison approach. *PLOS ONE* 8, 11 (2013), 1–12.
- [165] STAJICH, J. E., BLOCK, D., BOULEZ, K., BRENNER, S. E., CHERVITZ, S. A., DAGDIGIAN, C., FUELLEN, G., GILBERT, J. G. R., KORF, I., LAPP, H., LEHVASLAIHO, H., MATSALLA, C., MUNGALL, C. J., OSBORNE, B. I., POCKOCK, M. R., SCHATTNER, P., SENDER, M., STEIN, L. D., STUPKA, E., WILKINSON, M. D., AND BIRNEY, E. The Bioperl Toolkit: Perl Modules for the Life Sciences. *Genome Research* 12, 10 (2002), 1611–1618.
- [166] STRATONOVICH, R. L. Conditional markov processes. *Theory of Probability & Its Applications* 5, 2 (1960), 156–178.
- [167] STROUSTRUP, B. *The C++ Programming Language C Plus Plus Programming Language*. Addison-Wesley, 1986.
- [168] TELLES, M. A. *Python Power! The Comprehensive Guide*. Power! Series. Thomson Course Technology, 2008.
- [169] THÉVENET, P., SHEN, Y., MAUPETIT, J., GUYON, F., DERREUMAUX, P., AND TUFFÉRY, P. Pep-fold: an updated de novo structure prediction server for both linear and disulfide bonded cyclic peptides. *Nucleic Acids Research* 40, W1 (2012), W288–W293.
- [170] THOMAS, A., DESHAYES, S., DECAFFMEYER, M., VAN EYCK, M. H., CHARLOTEAUX, B., AND BRASSEUR, R. Prediction of peptide structure: how far are we? *Proteins: Structure, Function, and Bioinformatics* 65, 4 (2006), 889–897.
- [171] TRIKI, D., CANO, M., FLATTERS, D., VISSEAU, B., DESCAMPS, D., CAMPROUX, A. C., AND REGAD, L. Analysis of the hiv-2 protease adaptation to various ligands: characterization of the backbone asymmetry using a structural alphabet. *en révision dans Scientific reports* (2017).
- [172] TUNG, C. H., HUANG, J. W., AND YANG, J. M. Kappa-alpha plot derived structural alphabet and blosum-like substitution matrix for rapid search of protein structure database. *Genome Biology* 8, 3 (2007), R31.
- [173] TUNG, C. H., AND NACHER, J. C. A complex network approach for the analysis of protein units similarity using structural alphabet. *International Journal of Bioscience, Biochemistry and Bioinformatics* 3, 5 (2013), 433–437.

- [174] TUNG, C. H., AND YANG, J. M. fastscop: a fast web server for recognizing protein structural domains and scop superfamilies. *Nucleic Acids Research* 35, suppl 2 (2007), W438–W443.
- [175] TYAGI, M., BORNOT, A., OFFMANN, B., AND DE BREVERN, A. G. Protein short loop prediction in terms of a structural alphabet. *Computational Biology and Chemistry* 33, 4 (2009), 329–333.
- [176] TYAGI, M., GOWRI, V. S., SRINIVASAN, N., DE BREVERN, A. G., AND OFFMANN, B. A substitution matrix for structural alphabet based on structural alignment of homologous proteins and its applications. *PROTEINS: Structure, Function, and Bioinformatics* 65, 1 (2006), 32–39.
- [177] UNGER, R., HAREL, D., WHERLAND, S., AND SUSSMAN, J. L. A 3d building blocks approach to analyzing and predicting structure of proteins. *Proteins: Structure, Function, and Bioinformatics* 5, 4 (1989), 355–373.
- [178] VAN MONTFORT, R. L., BASHA, E., FRIEDRICH, K. L., SLINGSBY, C., AND VIERLING, E. Crystal structure and assembly of a eukaryotic small heat shock protein. *Nature Structural & Molecular Biology* 8, 12 (2001), 1025–1030.
- [179] WANG, S. Clefaps: fast flexible alignment of protein structures based on conformational letters. *arXiv preprint* (2009).
- [180] WANG, S., AND ZHENG, W. M. Clepaps: fast pair alignment of protein structures based on conformational letters. *Journal of Bioinformatics and Computational Biology* 6, 02 (2008), 347–366.
- [181] WESTBROOK, J., ITO, N., NAKAMURA, H., HENRICK, K., AND BERMAN, H. M. PDBML: the representation of archival macromolecular structure data in XML. *Bioinformatics* 21, 7 (2005), 988–992.
- [182] YADOLAH, D. *Statistique: Dictionnaire encyclopédique*, springer ed. Springer Science & Business Media, 2004.
- [183] YANG, J. Comprehensive description of protein structures using protein folding shape code. *Proteins: Structure, Function, and Bioinformatics* 71, 3 (2008), 1497–1518.
- [184] YANG, J. M., AND TUNG, C. H. Protein structure database search and evolutionary classification. *Nucleic Acids Research* 34, 13 (2006), 3646–3659.
- [185] YANG, Z. R., THOMSON, R., MCNEIL, P., AND ESNOUF, R. M. RONN: the bio-basis function neural network technique applied to the detection of natively disordered regions in proteins. *Bioinformatics* 21, 16 (2005), 3369–3376.
- [186] ZHANG, X., FETROW, J. S., RENNIE, W. A., WALTZ, D. L., AND BERG, G. Automatic derivation of substructures yields novel structural building blocks in globular proteins. In *ISMB* (1993), vol. 1, pp. 438–446.

- [187] ZHENG, W. M. The use of a conformational alphabet for fast alignment of protein structures. In *International Symposium on Bioinformatics Research and Applications* (2008), Springer, pp. 331–342.