

N° d'ordre: 3241

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Ingrid JACQUEMIN

Équipe d'accueil : Symbiose - IRISA

École Doctorale : MATISSE

Composante universitaire : IFSIC

Titre de la thèse :

***Découverte de motifs relationnels en bioinformatique :
application à la prédiction de ponts disulfures***

soutenue le 07 décembre 2005 devant la commission d'examen

M. :	Olivier	RIDOUX	Président
MM. :	Céline	ROUVEIROL	Rapporteurs
	Christophe	GEOURJON	
MM. :	Pascale	SÉBILLOT	Examineurs
	Rumen	ANDONOV	
	Jacques	NICOLAS	

La Théorie, c'est quand rien ne fonctionne mais on sait pourquoi.
La Pratique, c'est quand tout fonctionne mais on ne sait pas pourquoi.
Ici on associe la Théorie à la Pratique :
rien ne fonctionne et personne ne sait pourquoi!

A. Einstein

Remerciements

Après cet agréable séjour à l'Irisa, il m'est difficile de mentionner certaines personnes plutôt que d'autres dans mes remerciements. Merci à tous pour cet accueil chaleureux et ces conditions de travail remarquables !

Je tiens sincèrement à remercier Jacques Nicolas qui a dirigé mes travaux de thèse. Malgré ses nombreuses responsabilités, il a su me faire avancer dans ma recherche. Un grand merci pour sa patience, ses bonnes idées et cette rigueur dans le travail, digne d'un grand chercheur !

Mes seconds remerciements s'adressent à Rumen Andonov sans qui cette thèse n'aurait pas existé. Il m'a orienté et initié au domaine de la recherche alors que j'avais pris une orientation différente. Merci d'avoir cru en moi.

Je remercie Olivier Ridoux qui m'a fait l'honneur de présider mon jury. Je remercie également Céline Rouveirol et Christophe Geourjon, rapporteurs de mon manuscrit, pour leurs précieuses et pertinentes remarques. Un grand merci à Pascale Sébillot qui m'a fait le plaisir de participer à mon jury.

Merci à Dominique Tessier pour m'avoir permis d'utiliser sa base de données et pour les enrichissantes discussions qui m'ont permis de faire évoluer mes idées. Merci également aux membres de l'ACI GenoTo3D qui travaillent sur le problème plus général de la prédiction de la structure tertiaire des protéines. Les discussions durant les réunions ont contribué à l'avancement de mes travaux.

Je voudrais également remercier les membres des équipes Symbiose, Dream et TexMex. Ils savent communiquer leur bonne humeur et créer une formidable et inoubliable ambiance. Merci à Vincent Claveau pour m'avoir toujours aidé, pour la qualité de ses remarques et pour son bon vin...

Un merci tout particulier à mes collègues de bureau, à savoir Yoann Mescam et Elodie Retout, pour m'avoir supporté pendant tout ce temps. Ma "maman" de bureau a toujours su me remotiver dans les moments difficiles. Son soutien a contribué à la réussite de cette thèse.

Certaines personnes ont joué le rôle de confidentes comme Catherine Belleannée et Anne Siegel. Leurs conseils m'ont toujours aidé à avancer. Mille merci pour votre écoute.

Un grand merci aux lecteurs des nombreuses versions de ma thèse, entre autres à Anne-Sophie Valin, Emmanuelle Morin et Patrick Durand. Ils ont toujours réussi à trouver de très jolies phrases pour me dire que tout était à refaire;-).

Je ne peux finir mes remerciements sans citer le très célèbre boute-en-train de l'équipe Symbiose, à savoir François Coste! François, j'adore tes blagues et ta bonne humeur, ne change rien!

Mes remerciements se tournent également vers mes amis bridgeurs et tout particulièrement vers mon remarquable partenaire Daniel Thouroude qui, entre 2 parties de bridge, s'assurait de l'avancement de mes travaux.

Un grand merci aussi à mes parents qui ont toujours cru en moi et qui ont toujours été présents. Sans leur aide et leur grand soutien cette thèse n'aurait surement pas abouti.

J'aurais également aimé citer mon NiNiNe, ma sister, Goulven Kerbellec, Guillaume Collet, Cynthia Alland, Laure Berti, etc., et je voudrais avoir une pensée toute particulière pour Marie Lahaye...

Merci à tous...
Ingrite.

Table des matières

Table des matières	2
Table des figures	5
Table des tableaux	6
1 La prédiction de ponts disulfures : un problème d'apprentissage non régulier sur des séquences de protéines	15
1.1 Les protéines	15
1.2 Prédiction des propriétés fonctionnelles et structurales	20
1.2.1 Stockage des données	20
1.2.2 Identification et classification des protéines	21
1.2.3 Algorithmes de prédiction de structure	22
1.3 De la découverte de motifs à la prédiction d'appariements	22
1.3.1 Prédiction de motifs sur des protéines	22
1.3.1.1 Définition des motifs	22
1.3.1.2 Classification des motifs	23
1.3.1.3 Méthodes de découverte de motifs dans ces protéines	23
1.3.2 Prédiction de motifs appariés	26
1.4 Recherche de relations contextuelles sur des séquences de protéines	27
1.4.1 Prédiction de structures secondaires régulières	28
1.4.2 Prédiction de la structure tertiaire	32
1.5 Prédiction des ponts disulfures dans les protéines	33
1.5.1 Prédiction des cystéines oxydées (cystines)	34
1.5.2 Prédiction de l'appariement des cystéines oxydées	36
2 Apprentissage de relations sur des séquences	41
2.1 Notions de base sur l'apprentissage automatique	41
2.2 L'inférence grammaticale	44
2.2.1 Mots et langages	44
2.2.2 Représentation par système de règles	44
2.2.3 Représentation par automate	46
2.2.4 Apprentissage d'automates finis	48
2.2.5 Apprentissage de langages algébriques	54

2.2.6	Applications de l'inférence grammaticale sur des séquences biologiques	56
2.3	La Programmation Logique Inductive (PLI)	60
2.3.1	Principes et approche de la PLI	60
2.3.1.1	Logique des prédicats	60
2.3.1.2	Déduction en programmation logique	62
2.3.1.3	Induction en programmation logique	63
2.3.2	Algorithmes de programmation logique inductive	66
2.3.3	Applications sur des séquences biologiques	69
3	Apprentissage de langages de contrôles sur des séquences de protéines	71
3.1	Problématique	71
3.2	Expérimentations pour la prédiction des ponts disulfures	72
3.2.1	Cadre expérimental	72
3.2.2	Mise en place de l'expérimentation	72
3.2.3	Résultats	78
3.3	Analyse des limitations des algorithmes d'inférence régulière pour l'inférence de langages de contrôle	86
3.3.1	Influence de la taille de l'alphabet et des instances sur l'inférence	86
3.3.2	Influence de la localisation des motifs sur l'inférence	88
4	Apprentissage de programmes logiques sur des séquences de protéines	93
4.1	Inférence	94
4.1.1	Constitution de l'échantillon pour la prédiction des cystéines oxydées	95
4.2	Exploration de l'espace des hypothèses	98
4.3	Résultats des expérimentations pour la prédiction de l'état d'oxydation des cystéines	103
4.4	Perspectives pour la prédiction de l'appariement des cystéines oxydées	108
4.4.1	Constitution de l'échantillon pour la prédiction de l'appariement des cystéines oxydées	108
4.4.2	Connaissance préalable	109
5	Bilan et discussion	113
5.1	Synthèse	113
5.2	Perspectives	115
	Index	123
	Bibliographie	141

Table des figures

1	Topologie de la protéine 1aho de PDB.	10
1.1	Structure d'un acide aminé, où R est la chaîne latérale spécifique à chaque acide aminé	16
1.2	Structures des protéines	18
1.3	Types d'interactions dans une chaîne protéique	19
1.4	Structure 3D et structures primaire et secondaire associées de la protéine limt (Mamba Intestinal Toxin 1)	37
2.1	Représentation de l'espace des exemples et de l'espace des hypothèses	42
2.2	Ordre sur les hypothèses	43
2.3	Hierarchie de Chomsky et modélisation de structures biologiques [Searls, 1997]	46
2.4	Un automate ambigu	47
2.5	$MCA(\mathcal{E}_+)$ avec $\mathcal{E}_+ = \{a, ab, bab\}$	49
2.6	$PTA(\mathcal{E}_+)$ avec $\mathcal{E}_+ = \{a, ab, bab\}$	49
2.7	Exemple de treillis	50
2.8	Pseudo-noeuds dans une séquence d'ARN	58
3.1	Quelques exemples des différents types de structures des ponts dans les séquences	73
3.2	Diagramme de Taylor	74
3.3	Types de d'appariement des acides aminés	75
3.4	Réécriture d'une instance selon un parcours canonique (en profondeur et de gauche à droite) de l'arbre de dérivation correspondant à l'analyse de cette instance.	76
3.5	Réductions effectuées sur les automates	78
3.6	Deux chemins non disjoints	80
3.7	Nombre de transition de l'automate obtenu par RPNI après application des différentes méthodes de réduction	85
3.8	Taille des automates en fonction des paramètres : taille alphabet, nombre de négatifs, taille des séquences.	87
3.9	Taille des automates par RPNI en fonction de la position du mot dans la séquence	89

3.10	Taille des automates par Blue-Fringe en fonction de la position du mot dans la séquence	89
3.11	Automate reconnaissant les mots de longueur impaire	90
4.1	Arbre général extrait du diagramme de Taylor.	97
4.2	Exemple = juxtaposition de 2 fenêtres de chaque côté d'une cystéine dans une protéine.	97
4.3	Sous-fenêtre pour la détection de modèles de taille 4 dans le contexte d'une cystéine.	99
4.4	Valeur des attributs "position" en fonction de la position de début des sous-fenêtres (contexte de taille 14, sous-fenêtre de taille 4)	100
4.5	Algorithme InHerIt	101
4.6	Taux cumulé de couverture des règles produites par InHerIt sur le problème de prédiction de l'état des cystéines sur la base de données d'apprentissage de D.Tessier (DT) et celle de validation de C.Geourjon (CG)	104
4.7	Configurations de ponts	111

Liste des tableaux

1.1	Les vingt acides aminés apparaissant dans les protéines	17
1.2	Classification des principaux types de motifs	23
1.3	Méthodes de découverte de motifs dans les protéines. Ce tableau contient les références des articles, la forme des motifs générés selon le tableau 1.2, l'échantillon fourni (+/- =échantillon positif/négatif), l'extension possible aux séquences d'ADN, et pour finir le nom des algorithmes.	24
1.4	Algorithmes de prédiction de structure secondaire (H=Hélice, E=Feuillet et C=Boucle), PPV=plus proches voisins, HMM=Hidden Markov Model, NN=Neural Network, SVM=Support Vector Machine	29
1.5	Comparaison des taux de prédiction, Q_3 =taux de prédiction total, Q_H =taux de prédiction des hélices, Q_E =taux de prédiction des feuillets et Q_C =taux de prédiction des boucles	31
1.6	Algorithmes de prédiction de cystéines oxydées.	34
1.7	Tableau récapitulatif des résultats de [Fariselli et Casadio, 2001] et [Vullo et Frasconi, 2004] par rapport à l'aléatoire	37
4.1	Modèles obtenus par PROGOL et leur couverture	105

Introduction

L'exploitation fonctionnelle des informations provenant des grands programmes de séquençage des génomes pose le problème important de la prédiction de la structure tridimensionnelle des protéines. Or, déterminer la structure tridimensionnelle des protéines expérimentalement est une tâche très lourde et coûteuse, qui peut s'avérer parfois impossible à réaliser. Les protéines sont les acteurs essentiels de la cellule : catalyseurs, moteurs, lisant et interprétant l'information génétique, coordonnant la réponse aux signaux externes et aux agressions éventuelles. Avec le séquençage de plus de 1 000 génomes au cours de la dernière décennie, nous connaissons la séquence des génomes de nombreux organismes, bactéries et eucaryotes [Bernot, 2001]. Mais, pour la plupart, la structure tridimensionnelle n'est pas connue. Elles sont pourtant essentielles pour identifier et comprendre la fonction des protéines. En effet, l'expérience a largement démontré que l'agencement tridimensionnel précis des atomes est un facteur essentiel au fonctionnement des protéines comme enzymes ou comme partenaires d'associations spécifiques. L'arrivée massive de données provenant des programmes de séquençage à grande échelle impose donc de passer d'une approche biochimique à une approche bioinformatique, et nécessite en particulier de développer des méthodes de prédiction sur des séquences.

La prédiction de structure d'une protéine (avec la haute précision nécessaire pour comprendre sa fonction) est difficile pour deux raisons. Premièrement, une protéine possède de nombreux degrés de liberté et une "quasi infinité" de conformations possibles. La seconde difficulté provient de la faible stabilité des protéines. Pour dénaturer, ou "déplier", une protéine, il suffit de lui fournir une très faible quantité d'énergie. La difficulté majeure des bioinformaticiens réside dans le fait de devoir passer d'une simple séquence de symboles, la séquence primaire en acides aminés de la protéine, à une structure tridimensionnelle de celle-ci faisant intervenir les interactions physico-chimiques des acides aminés.

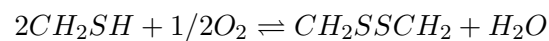
Le problème général de prédiction de structure des protéines à partir de la séquence d'acides aminés est difficile. La résolution de plusieurs sous-problèmes plus spécifiques exploitant les particularités de protéines aide à réduire cette complexité. Notre thèse s'inscrit dans ce cadre, nous nous focalisons sur l'objectif ambitieux de la prédiction des ponts disulfures dans les protéines. Cette liaison covalente stabilise et contraint fortement la conformation spatiale de la protéine et la connaissance des



FIG. 1 – Topologie de la protéine 1aho de PDB.

positions où elle intervient peut réduire considérablement la complexité du problème de la prédiction de la structure 3D des protéines car elle stabilise leur conformation spatiale.

La cystéine est l'un des vingt acides aminés qui constituent les protéines et possède une caractéristique unique. Les protéines dont la séquence possède des cystéines sont sujettes à des modifications covalentes post-traductionnelles et les cystéines peuvent apparaître dans une forme oxydée ou réduite. Deux cystéines oxydées s'apparient pour former un lien covalent, appelé *pont disulfure*, dont la formation peut être décrite par la réaction suivante :



De telles réactions exigent un environnement oxydant pour avoir lieu. Les conditions environnementales requises dépendent de la localisation intra-cellulaire de la protéine. Par exemple, les protéines cytoplasmiques et nucléaires n'ont habituellement pas de pont disulfure. Les ponts disulfures sont souvent vitaux pour le repliement et la stabilité des protéines. L'incidence des ponts disulfures sur la structure des protéines a été reconnue à plusieurs niveaux. Les simulations, les expériences et les études théoriques montrent leur importance dans la stabilisation de l'état natif des protéines.

Le rôle de stabilisation des ponts disulfures se traduit par une réduction du nombre d'états conformationnels possibles pour la protéine. En fonction de leur nombre et de

leur localisation, ces liens covalents peuvent aussi contribuer aux activités catalytiques des biomolécules [Klink *et al.*, 2000]. Le plus important étant qu'ils peuvent connecter des portions très distantes de la séquence, représentant ainsi un ensemble de contraintes structurelles fortes dans la forme des interactions longues distances (cf figure 1).

La connaissance de la localisation correcte des ponts disulfures a une importance considérable pour plusieurs raisons :

- les familles de protéines riches en ponts disulfures représentent un grand nombre de protéines ayant une pertinence biologique (facteurs de croissance, hormones, toxines, etc.). La topologie correcte des ponts disulfures est indispensable pour déterminer leur structure finale et leur fonction [Mas *et al.*, 1998];
- la connaissance de la topologie (S-S) peut aider et accélérer les travaux expérimentaux dans les laboratoires durant la première étape de la détermination de la structure et guide les expérimentations postérieures. Par exemple, la détermination de structure par spectroscopie RMN (résonance magnétique nucléaire) est un processus itératif dans lequel les structures antérieures sont utilisées pour corriger et compléter l'attribution NOE (Nuclear Overhauser Effect). L'information fournie par les liaisons S-S permet d'améliorer la qualité des structures initiales [Boisbouvier *et al.*, 2000];
- les repliements de petites protéines riches en S-S représentent une classe de protéines ayant en général un contenu en structure secondaire irrégulière [Harrison et Sternberg, 1996]. Dans ce cas, les ponts disulfures sont la seule caractéristique structurale de la protéine.
- il a été observé [Chuang *et al.*, 2003] que la connaissance des ponts disulfures peut être utilisée pour détecter des homologues (structurels) éloignées pour une chaîne donnée. Par conséquent, la connaissance de la topologie S-S étend le champ d'applications de la prédiction du repliement ;
- les contraintes topologiques fournies par les connectivités S-S améliorent la performance des algorithmes de reconstruction 3D de novo. L'information a été observée pour améliorer la qualité des structures prédites et pour réduire l'espace de recherche des conformations à explorer [Fain et Levitt, 2001, Huang *et al.*, 1999, Lund *et al.*, 1996];
- l'appariement correct cystéine-cystéine est important pour la détermination et l'exploitation des repliements de protéines pour des applications médicales, pharmaceutiques et agronomiques [Daly *et al.*, 1999, Vita *et al.*, 1998, Oren *et al.*, 1998, Fletcher *et al.*, 1997].

Il est donc clair que la connaissance des ponts disulfures peut aider considérablement toutes les méthodes pour la prédiction de structure 3D des protéines (threading ou reconnaissance de repliement, algorithmes *de novo*).

Deux questions de difficulté croissante se posent :

- Est-il possible de prédire qu'une cystéine donnée est impliquée dans un pont disulfure uniquement avec la connaissance de son contexte ?

- Est-il possible de prédire quelles paires de cystéines sont liées par un pont disulfure dans une protéine donnée?

De nombreux travaux ont été effectués sur le premier problème mais peu sur l'appariement des cystéines. L'objectif de cette thèse est de traiter ces deux problèmes de manière unifiée.

Pour cela, nous utilisons l'apprentissage automatique qui occupe aujourd'hui une place importante au sein de l'Intelligence Artificielle. Un grand intérêt est notamment accordé à l'apprentissage par induction qui consiste à inférer des concepts à partir d'exemples. Pour tenter de résoudre notre problème de prédiction des ponts disulfures nous avons tout d'abord utilisé l'inférence grammaticale en nous appuyant sur l'article de Y. Takada [Takada, 1994b]. Nous avons voulu étudier l'applicabilité pratique de ses travaux pour étendre le domaine d'utilisation des algorithmes d'inférence régulière.

Nous avons ensuite utilisé la programmation logique inductive qui explicite toutes les relations existant sur un ensemble de données, ce qui est un avantage majeur pour la prédiction de contextes disulfures en relation. Une autre propriété attendue de cette technique est son interprétabilité. En effet, plusieurs approches concurrentes performantes sont disponibles mais le prédicteur construit est une "boîte noire", ce qui présente plusieurs inconvénients. D'une part, les résultats obtenus sont difficilement interprétables pour les biologistes. Il est dès lors impossible de comprendre le processus ayant amené tel élément à être acquis ou non. D'autre part, ces approches n'offrent pas de définition opérationnelle des informations structurales qu'elles acquièrent. Celles-ci sont pourtant intéressantes dans les cas où les éléments recherchés ne sont définis que de manière théorique ou par l'intermédiaire d'exemples.

L'originalité de nos travaux réside en plusieurs points concernant aussi bien les approches adoptées que le cadre applicatif de prédiction de ponts disulfures dans les protéines. L'utilisation de techniques d'apprentissage symbolique est rare. Les approches statistiques sont en effet largement dominantes. Nous montrons cependant que l'utilisation de ces techniques d'apprentissage symbolique, et plus spécifiquement de la programmation logique inductive, est parfaitement adaptée à cette tâche. Cette thèse repose sur 4 chapitres, décrits ci-après. Nous terminons en mettant en avant les principales contributions de nos travaux et en proposant quelques pistes de recherche ouvertes à l'issue de ces derniers.

Chapitre 1 : ce chapitre est dédié à la problématique et aux enjeux couverts par la prédiction de structure des protéines. Nous nous attachons, à travers un panorama des travaux existants dans ce domaine, à faire ressortir les grandes familles de techniques utilisées. Nous nous intéressons ensuite plus spécifiquement à la prédiction des ponts disulfures en scindant ce problème en deux parties, chacune pouvant être résolue indépendamment, à savoir la prédiction de l'état d'oxydation des cystéines et la prédiction de l'appariement des cystéines oxydées.

Chapitre 2 : l'objectif de ce chapitre est de présenter en détail les choix et les techniques utilisées pour apprendre dans le cadre applicatif auquel nous nous confrontons. Après une courte introduction au domaine de l'apprentissage automatique, nous nous focalisons sur les deux principales méthodes en apprentissage de relations sur des données : l'inférence grammaticale et la programmation logique inductive. Concernant le premier type de méthode, nous rappelons quelques notions de la théorie des langages, puis nous positionnons nos travaux par rapport à l'inférence de langages réguliers et algébriques. Concernant la deuxième approche, nous rappelons quelques notations et définitions de la logique des prédicats et nous détaillons le positionnement de nos travaux.

Chapitre 3 : ce chapitre décrit les travaux réalisés à partir de l'article théorique de Y. Takada présenté dans le chapitre précédent. Dans un premier temps, nous détaillons les choix effectués sur la grammaire utilisée et sur les données d'apprentissage nécessaires à la prédiction des ponts disulfures dans les protéines. Dans un second temps, nous analysons les résultats des expériences que nous avons menées puis nous terminons par différentes expérimentations montrant les limites des algorithmes utilisés.

Chapitre 4 : ce chapitre décrit les travaux réalisés au moyen de la programmation logique inductive. Nous présentons tout d'abord le cadre opérationnel dans lequel doit se dérouler le processus d'induction. Nous y détaillons les étapes de constitution des exemples nécessaires à l'apprentissage et les contraintes qui sont imposées sur la forme attendue des motifs à travers le langage d'hypothèses. Nous étudions ensuite les problèmes de complexité. Nous présentons notamment la technique utilisée pour parcourir efficacement l'espace de recherche et les moyens d'éviter l'exploration de portions de cet espace ne pouvant conduire à de bonnes solutions. Nous terminons enfin par l'évaluation de notre approche sur la prédiction des ponts disulfures dans les protéines en présentant les résultats obtenus, dans un premier temps sur la prédiction des cystéines oxydées, puis, dans un second temps sur l'appariement des cystines.

Chapitre 1

La prédiction de ponts disulfures : un problème d'apprentissage non régulier sur des séquences de protéines

Nous présentons dans un premier temps la problématique et les enjeux couverts par la prédiction de structure des protéines. Nous nous attachons, à travers un panorama des travaux existants dans ce domaine, à faire ressortir les grandes familles de techniques utilisées. Nous nous intéressons ensuite plus spécifiquement à la prédiction des ponts disulfures en scindant ce problème en deux parties pouvant être résolues indépendamment à savoir, la prédiction de l'état d'oxydation des cystéines et la prédiction de l'appariement des cystéines oxydées.

L'accélération des programmes de séquençage génomique à grande échelle alimente aujourd'hui les banques de données de séquences protéiques, dont la taille double tous les 15 à 18 mois. Or, même si des progrès ont été accomplis, la résolution de la structure 3D d'une protéine reste une tâche relativement lourde et celle-ci n'est disponible que pour une petite fraction de protéines. Ce fossé entre séquences et structures de protéines ne fait que s'accroître, ce qui donne un intérêt évident à des méthodes de prédiction qui permettraient de s'affranchir de l'étape expérimentale, au moins pour certaines applications comme nous allons le voir dans ce chapitre.

1.1 Les protéines

Les protéines sont des polymères composées d'acides aminés qui transportent la plupart des fonctions moléculaires dans l'organisme vivant. Tous les acides aminés partagent la même structure chimique comme illustré figure 1.4(a). Il existe un atome carbone cen-

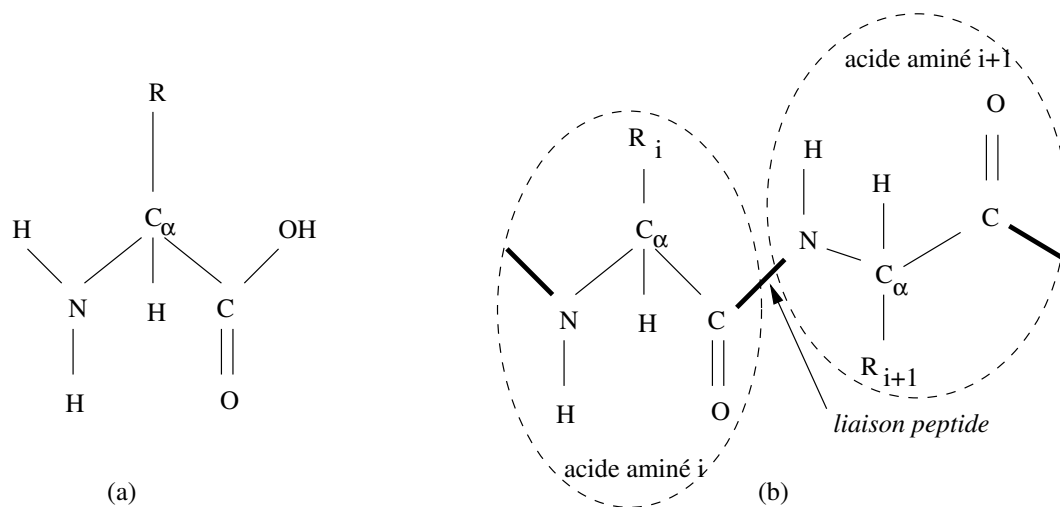


FIG. 1.1 – Structure d’un acide aminé, où R est la chaîne latérale spécifique à chaque acide aminé

tral ($C\alpha$) attaché à un atome hydrogène, un groupe amine (NH_2), un groupe carboxyle ($COOH$) et une chaîne latérale ou résidu (R) qui discrimine un acide aminé d’un autre. Les vingt résidus possibles apparaissant dans une protéine sont listés table 1.1, avec leur code à 3 lettres standard et celui à 1 lettre. La table regroupe les acides aminés selon la nature chimique de leur chaîne latérale. Les acides aminés appartenant à la classe des chaînes latérales allant de la Glycine à la Phenylalanine sont hydrophobes. La classe des résidus chargés va de l’acide Aspartique à l’Arginine, et les derniers, du Tryptophane à l’Histidine, sont ceux dont la chaîne latérale est polaire. Durant le processus de synthèse, les protéines sont assemblées séquentiellement grâce à la formation de liens peptidiques (cf figure 1.1(b)), où le groupe carboxyle d’un acide aminé est relié à un groupe amine d’un autre acide aminé. D’un point de vue biochimique, une protéine est ainsi représentée comme une chaîne polypeptidique formée d’une épine dorsale (la répétition séquentielle de l’unité de base $NH - C\alpha H - C = O$) et d’une chaîne latérale (la séquence de résidus attachée à l’épine dorsale). Conventionnellement, la structure des protéines est représentée hiérarchiquement avec quatre niveaux de description, illustrés figure 1.2 : structure primaire, secondaire, tertiaire et quaternaire.

La structure primaire : 1D

La structure primaire d’une protéine est la séquence d’acides aminés de la chaîne polypeptidique. Formellement, elle peut être modélisée comme une séquence sur un alphabet fini Σ_{aa} où $|\Sigma_{aa}| = 20$ (les codes 1-lettre de la table 1.1). Ces séquences se retrouvent dans des entrepôts de données, tels que Swissprot [Boeckmann *et al.*, 2003a].

La structure secondaire : 2D

Glycine	Gly (G)	Lysine	Lys (K)
Alanine	Ala (A)	Arginine	Arg (R)
Valine	Val (V)	Tryptophane	Try (W)
Leucine	Leu (L)	Sérine	Ser (S)
Isoleucine	Ile (I)	Thréonine	Thr (T)
Méthionine	Met (M)	Cystéine	CySH (C)
Proline	Pro (P)	Thyrosine	Tyr (Y)
Phénylalanine	Phe (F)	Asparagine	Asn (N)
acide Aspartique	Asp (D)	Glutamine	Gln (Q)
acide Glutamique	Glu (E)	Histidine	His (H)

TAB. 1.1 – Les vingt acides aminés apparaissant dans les protéines

Cette structure correspond à l'arrangement spatial des chaînes polypeptidiques représentant les motifs réguliers que l'on retrouve dans toutes les séquences. On reconnaît principalement deux grands types de structures secondaires régulières :

- l'hélice α (H) : dans cette structure, illustrée figure 1.2(b), la chaîne d'acides aminés s'enroule en une spirale (3 à 7 acides aminés par tour de spire) dont les différentes spires sont stabilisées par des liaisons hydrogènes formées entre le groupe $C=O$ du résidu à la position i et NH du résidu à la position $i+4$;
- le feuillet β (E) : à l'intérieur de ce feuillet, illustré figure 1.2(b), se forme des liaisons hydrogènes entre certains segments de la chaîne, disposés parallèlement ou anti-parallèlement les uns par rapport aux autres.

Les hélices α et les feuillets β sont souvent connectés par des boucles ou coudes (C) de taille et de structure variables qui n'ont pas de forme régulière comme les deux derniers éléments. Chaque acide aminé dans la séquence appartient à l'un des trois types structurels, et la structure secondaire peut être représentée par une séquence sur un alphabet $\Sigma_{ss} = \{H, E, C\}$ de même longueur que la structure primaire. Certains utilisent un alphabet de 4 lettres en rajoutant la notion de *virage* (T) pour les structures irrégulières, ainsi $\Sigma_{ss} = \{H, E, C, T\}$. Les cystéines se retrouvent dans tous ces types de structures et peuvent former des ponts en étant dans des structures différentes.

La structure tertiaire : 3D

A température physiologique, une chaîne d'acides aminés se replie sur elle-même et adopte une conformation native très spécifique dans l'espace que l'on appelle structure tridimensionnelle ou tertiaire. Elle résulte avant tout du fait qu'en milieu aqueux les groupes hydrophiles sont dirigés vers l'extérieur, alors que les groupes hydrophobes se trouvent majoritairement à l'intérieur de la molécule protéique. Cette conformation est thermodynamiquement stable grâce à un ensemble de liaisons non covalentes comme les liaisons hydrogènes ou les ponts salins, ainsi qu'aux fortes liaisons covalentes des ponts disulfures. Ces interactions sont de plusieurs types (cf figure 1.3) caractérisant ces liaisons faibles ou covalentes :

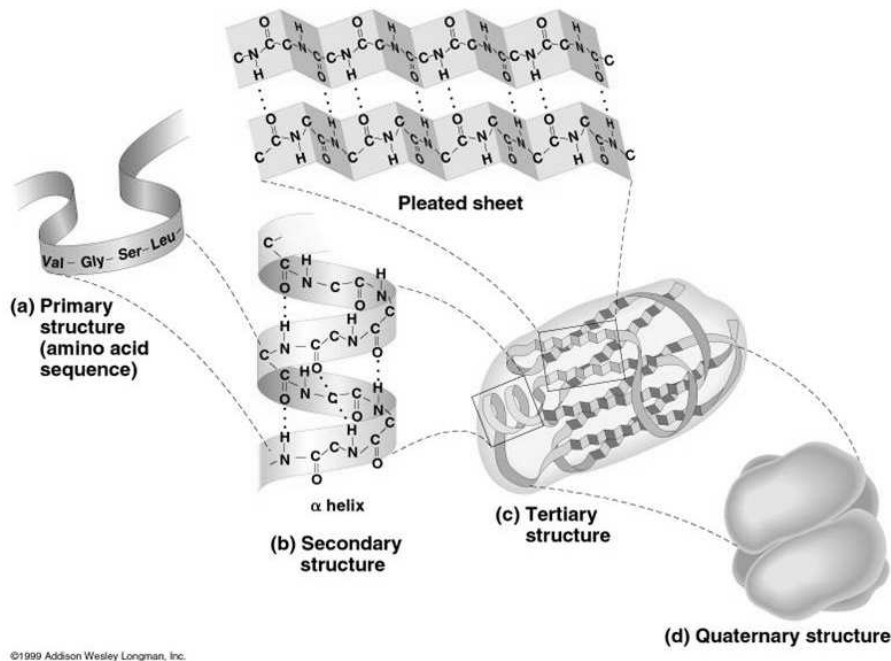


FIG. 1.2 – Structures des protéines

– **liaisons faibles :**

- l'effet hydrophobe : dans le milieu aqueux de la cellule, les acides aminés hydrophobes ont davantage d'affinité entre eux qu'avec les molécules d'eau entourant la protéine. La chaîne a donc tendance à se replier de façon à les regrouper entre eux au centre de la molécule, en minimisant les contacts avec l'eau. Inversement, les acides aminés hydrophiles ont tendance à se disposer à la périphérie de façon à être en contact avec l'eau ;
- les liaisons ioniques : les acides aminés chargés positivement forment des liaisons avec ceux qui s'ionisent négativement, c'est le cas des ponts salins ;
- les liaisons hydrogènes : ces liaisons permettent d'avoir des architectures moléculaires très bien définies (hélices, feuillettes). Elles sont de faible intensité, ce qui donne à ces architectures une certaine souplesse ;

- **liaisons covalentes :** un acide aminé particulier, la cystéine, contient un atome de soufre qui lui permet de former une liaison covalente avec une autre cystéine, appelée les ponts disulfures. Cette liaison peut relier deux cystéines éloignées l'une de l'autre sur la chaîne. Nous nous intéressons à ces liaisons covalentes car la prédiction correcte de ce type de lien peut jouer un rôle très important dans la réduction de la complexité pour le problème de la prédiction de la structure 3D des protéines, en construisant une forme globale de la

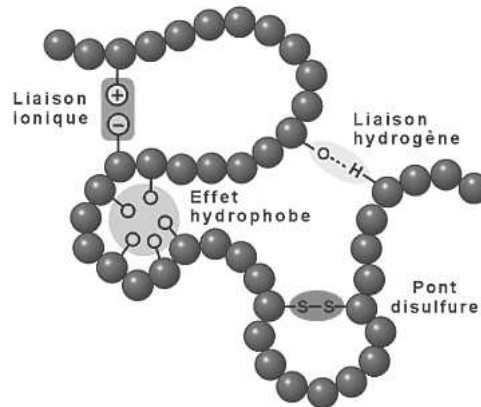


FIG. 1.3 – Types d’interactions dans une chaîne protéique

protéine prédite [Chuang *et al.*, 2003, Creighton, 1992].

La conformation native d’une protéine dépend à la fois de sa séquence et du milieu dans lequel elle est solubilisée. On peut dénaturer cette structure de façon réversible en la chauffant à des températures variables selon la thermosensibilité de chaque protéine [Scopes, 1994, Lee, 1991]. On peut aussi dénaturer la protéine en utilisant des agents oxydants et de l’urée comme l’a fait Anfinsen [Anfinsen, 1973]. La composition en acides aminés affecte la stabilité thermique des protéines car les protéines qui contiennent une grande proportion de résidus hydrophobes sont plus stables que les protéines hydrophiles.

La structure quaternaire : 4D

Cette structure concerne l’assemblage macromoléculaire de plusieurs monomères. Les ponts disulfures peuvent se créer entre deux cystéines de deux chaînes différentes, on les appelle les ponts inter-moléculaires. Dans cette thèse nous nous limitons à la prédiction des ponts disulfures intra-moléculaires, *i.e.* des ponts formés entre deux cystéines d’une même chaîne d’acides aminés.

Détermination de la structure des protéines :

Plusieurs techniques expérimentales permettent d’obtenir la structure 3D des protéines telles que la diffraction des rayons X sur des cristaux de protéines ou la résonance magnétique nucléaire (RMN) [Branden et Tooze, 1999]. Ces techniques sont très coûteuses et lourdes et ne peuvent s’appliquer à toutes les protéines (inutilisables pour les protéines non solubles, comme les protéines membranaires). A l’inverse, les séquences de protéines deviennent relativement faciles à obtenir à partir de la connaissance des séquences de génomes. Obtenir des connaissances structurelles sur les protéines à partir de la seule connaissance de la séquence de protéines analysées est en conséquence devenu un enjeu de taille pour la bioinformatique.

1.2 Prédictions des propriétés fonctionnelles et structurales

Un des enjeux majeurs de la bioinformatique est de prédire *in silico* la structure des protéines afin de mieux comprendre leur rôle biologique, leur mode de fonctionnement et leurs interactions. La bioinformatique peut aider à la détermination de la structure des protéines de plusieurs manières différentes :

- en archivant et en organisant les données de séquences et structures connues en bases de données ;
- en identifiant et classifiant les fonctions et les structures des protéines permettant d'établir des comparaisons ;
- en proposant des algorithmes de prédiction de structures et de motifs.

1.2.1 Stockage des données

La détermination expérimentale des structures est un travail très délicat et le volume d'information disponible sur les séquences et structures de protéines ne cesse de croître de manière exponentielle depuis que des bases de données nationales ont été mises en place.

Banques de séquences

Ces données sont répertoriées dans différentes banques de séquences protéiques disponibles sur le réseau internet. Uniprot¹ [Schneider *et al.*, 2005] est le catalogue de référence de l'information sur les protéines. C'est un entrepôt de données sur les séquences et fonctions des protéines rassemblant les informations contenues dans les banques de données TrEMBL [Boeckmann *et al.*, 2003b], SWISS-PROT [Junker *et al.*, 1999] et PIR [Wu *et al.*, 2003]. Uniprot est constitué de trois couches :

- UniProtKB qui est la base de données centrale des séquences protéiques fusionnant les données issues de Swiss-Prot, TrEMBL et PIR-PSD. Cette base de connaissance contient les annotations et les informations fonctionnelles sur les séquences ;
- UniParc qui est une archive fournissant la collection complète des séquences protéiques publiées à ce jour, avec un système de mise à jour de versions. Elle contient plus de 4 millions de séquences différentes (release septembre 2004) ;
- UniRef qui est une base de données de clusters de séquences permettant d'effectuer des recherches rapides de similarité de séquences ayant 100, 90 ou 50% d'identité. Elle possède plus de 4 millions de références croisées vers plus de 60 banques de données différentes.

Banques de structures

La principale banque de données contenant les informations structurales des protéines est la PDB (Protein Data Bank [Berman *et al.*, 2000]). Elle contient les coor-

¹<http://www.uniprot.org>

données 3D des protéines dont la structure a été déterminée, ce qui représente 32268 structures aujourd'hui (Release 16 août 2005). Ces structures ont été déterminées expérimentalement par cristallographie et RMN. Parmi celles-ci, on relève près de 2 000 protéines ayant moins de 25% d'acides aminés identiques entre deux séquences. De 50 à 100 structures sont déposées dans cette banque chaque semaine.

1.2.2 Identification et classification des protéines

Afin d'identifier et de classer une nouvelle protéine, une recherche de similarité est effectuée avec les séquences déjà publiées dans les banques de données. Ceci permet d'associer une fonction potentielle à une protéine à partir d'une similitude de séquences. Pour cela, il existe de nombreux outils comme FASTA [Pearson et Lipman, 1988] et BLAST [Altschul *et al.*, 1990], ayant l'avantage d'être très rapides car ils utilisent des heuristiques permettant d'élaguer l'espace de recherche, mais aussi PSI-BLAST [Altschul *et al.*, 1997], SSEARCH et MPsrch qui fournissent des résultats plus complets au prix d'une plus grande complexité.

Les outils de recherche de similarité ne mettent pas toujours en évidence des ressemblances lorsque les séquences sont trop éloignées de celles déjà répertoriées. Cependant, un même groupe d'acides aminés dans la séquence peut être représentatif d'une *signature* ou d'un *motif*. Ces signatures protéiques pertinentes sont limitées à quelques résidus mais possèdent néanmoins une probabilité d'occurrence au hasard très faible, d'où leur spécificité. L'existence de tels motifs conservés peut s'expliquer par le fait que les régions importantes de la protéine sont les mieux conservées (au niveau structural et/ou séquentiel). Ces régions peuvent souvent être associées aux sites actifs de la protéine.

Aujourd'hui, InterPro [Mulder *et al.*, 2005] est la base de données de référence des familles, des domaines et des sites fonctionnels de protéines. Elle rassemble les signatures des protéines de différentes bases de données telles que PROSITE [Bairoch *et al.*, 1997], PRINTS [Attwood *et al.*, 2003], SMART [Letunic *et al.*, 2004], Pfam [Bateman *et al.*, 2004], ProDom [Bru *et al.*, 2005], etc. Cette intégration de données permet de travailler sur un format et une nomenclature commune.

On peut également observer des motifs de structure. CATH [Pearl *et al.*, 2005] est une base de classification sur la ressemblance structurale des protéines contenant 4 niveaux :

- *classe* qui est déterminée selon la composition de la structure secondaire ;
- *architecture* qui décrit l'orientation des structures secondaires, indépendamment des connectivités ;
- *topologie* qui représente les connections topologiques et le nombre de structures secondaires
- *homologie* qui rassemble les protéines ayant une forte similarité de structures et de fonctions.

1.2.3 Algorithmes de prédiction de structure

Afin d'accélérer la détermination de la structure des protéines, il importe de concevoir des méthodes automatiques capables d'inférer des repliements à partir des données de séquences et éventuellement de structures connues d'autres protéines. Les algorithmes de prédiction de structure de protéines sont très nombreux et traitent différents problèmes. Certains recherchent des motifs caractéristiques dans des séquences permettant d'identifier des familles de protéines pour associer une structure connue à une protéine inconnue. Plusieurs tentent de prédire la structure secondaire en identifiant des structures en forme d'hélice, de feuillet, ou de boucle. D'autres enfin, tentent de prédire la structure tertiaire afin de connaître le repliement de la protéine. Dans ce qui suit, nous présentons plusieurs algorithmes traitant chacun de ces problèmes.

Cette thèse contribue à l'objectif ambitieux de la prédiction de structure des protéines en se focalisant sur la prédiction d'interactions particulières à l'intérieur de macro-molécules, les ponts disulfures. Elle s'inscrit dans le cadre de l'apprentissage automatique et pose deux problèmes intéressants. En effet, la prédiction d'un pont disulfure peut être vue comme la recherche de motifs, caractéristiques de l'état d'oxydation des cystéines. Ces motifs vont ensuite s'apparier pour former un contexte favorable à l'établissement d'une liaison covalente. Nous allons préciser dans la prochaine section ces deux aspects : découverte de motifs et prédiction de motifs appariés.

1.3 De la découverte de motifs à la prédiction d'appariements

1.3.1 Prédiction de motifs sur des protéines

L'objectif de la découverte de motifs est de localiser des régions susceptibles de correspondre à une activité, une structure ou une fonction biologique précise, conservées pour un ensemble de protéines apparentées (famille) : la suppression de la zone dans la séquence doit avoir un impact sur son activité. La notion de motif prend suivant les auteurs un sens assez variable et nous commençons par préciser celle que nous utilisons.

1.3.1.1 Définition des motifs

Les méthodes de découverte de motifs recherchent des régularités sous la forme de sous-classes des expressions régulières.

Définition 1.1 (motif) *Un motif, ou pattern, est un mot appartenant à $(\Sigma \cup \Pi \cup X \cup \{*\})^*$, où Σ est l'alphabet des acides aminés sur lequel sont construits les mots, Π est l'ensemble des parties de Σ de taille comprise entre 2 et $|\Sigma| - 1$, X est l'ensemble des jokers, de la forme $x(i, j)$ associé à la reconnaissance de $\bigcup_{i \leq n \leq j} \Sigma^n$, et $*$ est un symbole associé à la reconnaissance de Σ^* .*

Concepts	Exemples de motifs	Classes (représentées par la 1 ^e lettre du concept)
	Notations tirées de la classification de Brazma [Brazma <i>et al.</i> , 1998]	
Facteur (sous-mot)	T-C-T-T-G-A	(F)
Composant (mot sur l'ensemble des parties de l'alphabet)	D-R-C-C-H-D-x-C	(Ca)
	G-G-G-T-F-[ILV]-[ST]-[ILV]	(Cb)
	V-x-P-[RQ]-G-D-H-x-L-[LM]	(Cc)
Distance entre composants	V-H-P-x(2)-R-Q-x(4)-G-x(2)-L-C	(Da)
	G-C-x(1,3)-C-P-x(8,10)-C-C	(Db)
	D-T-A-G-Q-E-*-L-V-G-N-K	(Dc)

TAB. 1.2 – Classification des principaux types de motifs

1.3.1.2 Classification des motifs

Brazma *et al.* proposent une classification des motifs, représentant des zones conservées dans des familles de protéines [Brazma *et al.*, 1998]. On peut y distinguer 3 niveaux illustrés table 1.2. Tout d'abord, les motifs simples, ou sous-mots, ne contenant que des lettres de l'alphabet initial. Ensuite, des motifs pouvant contenir des caractères quelconques introduit par le symbole joker x ou contenant un composant autorisant un choix d'acides aminés mis entre crochet à une position fixe. Ces mots sont construits sur l'ensemble des parties de l'alphabet. Et enfin, des motifs faisant intervenir une notion de distance, appelés *gaps*, représentés généralement sous la forme $x(i)$ pour un nombre fixe d'acides aminés successifs quelconques, $x(i,j)$ pour un nombre borné d'acides aminés successifs quelconques variant de i à j , et $*$ pour un nombre quelconque non borné d'acides aminés successifs.

Nous avons vu précédemment qu'Interpro intègre les données de la banque de données PROSITE. Ses motifs sont complexes car il peuvent contenir des jokers (x), des composants à une position donnée [ADE], $\{ADE\}$, des gaps ($x(3)$, $x(2,4)$) et des répétitions d'acides aminés ($A(3)$). Les motifs de PROSITE sont générés à la main. Il existe un certain nombre de méthodes permettant de générer ce type de motifs de manière automatique.

1.3.1.3 Méthodes de découverte de motifs dans ces protéines

Ces méthodes recherchent des motifs possédant un certain nombre d'occurrences dans un ensemble de séquences. Le nombre minimal de ces occurrences est appelé *sup-*

Références	Méthodes			Noms
	Motifs	Échant.	Applicable ADN	
[Waterman <i>et al.</i> , 1984]	F	+	oui	genalign
[Martinez, 1988]	Dc	+	oui	
[Landraud <i>et al.</i> , 1989]	Dc	+	oui	
[Smith et Smith, 1990]	CbDaDb	+	non	motifSmith
[Smith <i>et al.</i> , 1990]	DaCa	+	non	
[Waterman et Jones, 1990]	CbDaDb	+	oui	pralign
[Vingron et Argos, 1991]	Dc	+	non	filter
[Ogiwara <i>et al.</i> , 1992]	Dc	+/-	non	musco
[Roytberg, 1992]	F	+	oui	
[Smith et Smith, 1992]	CbDaDb	+	non	pima
[Arikawa <i>et al.</i> , 1993]	Dc	+/-	non	asset
[Neuwald et Green, 1994]	CcDa	+	non	
[Saqi et Sternberg, 1994]	Cb	+	non	discover, classify
[Wang <i>et al.</i> , 1994]	Dc	+	non	
[Jonassen <i>et al.</i> , 1995]	CbDaDb	+	non	pratt
[Sagot <i>et al.</i> , 1995b]	Cb	+	non	bonsai
[Sagot <i>et al.</i> , 1995a]	F	+	non	
[Shoudai <i>et al.</i> , 1995]	CbDaDb	+/-	non	gape
[Suyama <i>et al.</i> , 1995]	Db	+	non	seqclassx
[Wu et Brutlag, 1995]	Cb	+	non	mdlpratt
[Sagot et Viari, 1996]	CcDa	+	oui	
[Brazma <i>et al.</i> , 1996]	CbDaDb	+	non	metameme
[Grundy <i>et al.</i> , 1997]	CcDa	+	non	prattTwo
[Jonassen, 1997]	CbDaDb	+	non	emotif
[Nevill-Manning <i>et al.</i> , 1997]	CbDaDb	+	non	teiresias
[Rigoutsos et Floratos, 1998a]	CaDa	+	non	
[Guralnik et Karypis, 2001]	F	+	non	

TAB. 1.3 – Méthodes de découverte de motifs dans les protéines. Ce tableau contient les références des articles, la forme des motifs générés selon le tableau 1.2, l'échantillon fournit (+/- =échantillon positif/négatif), l'extension possible aux séquences d'ADN, et pour finir le nom des algorithmes.

port. La plupart des méthodes recherchent des motifs dits *maximaux* (ou les plus spécifiques) ce qui signifie qu'ils sont maximaux en terme de longueur et de composition par rapport à leur support. Ces méthodes peuvent être dirigées par le modèle, dirigées par les données, ou hybrides. Vu le nombre de méthodes existantes, nous ne pouvons les détailler toutes, le lecteur intéressé pourra se reporter aux différents articles cités et aux revues suivantes [Brazma *et al.*, 1998], [Brejova *et al.*, 2000] et [Notredame, 2002]. Nous proposons le tableau 1.3 répertoriant les principales méthodes.

Méthodes dirigées par le modèle

Ces méthodes consistent en une énumération explicite de l'ensemble des motifs dont le niveau d'expressivité est faible [Queen et Wegman, 1982], [Waterman *et al.*, 1984], [Staden, 1989], [Wolfertetter *et al.*, 1996]. Les algorithmes efficaces effectuant ce type de recherche travaillent généralement sur des motifs sans gap car la complexité de cette approche est exponentielle sur la longueur du motif. L'espace de recherche est limité aux motifs d'une longueur fixée et les méthodes calculent le nombre de séquences de l'ensemble d'apprentissage comportant un mot proche du motif. Néanmoins, ces méthodes ont été étendues pour des motifs plus complexes. [Smith et Smith, 1990] proposent un algorithme qui recherche des motifs pouvant contenir des gaps fixes dont la taille est spécifiée par l'utilisateur. La méthode de Suyama permet de découvrir des motifs contenant des gaps de taille variable [Suyama *et al.*, 1995]. Cette méthode ne peut être utilisée pour rechercher des motifs complexes, *i.e.* combinant plusieurs classes de motifs.

Une recherche en largeur sur la taille du motif peut être effectuée pour réduire la complexité. Le principe de cette méthode est de développer un arbre de recherche par spécialisation des motifs. Cette approche consiste à étendre itérativement un motif ne contenant initialement qu'un seul symbole tout en respectant la contrainte de support. Chaque extension correspond à un noeud de l'arbre, si cette extension ne vérifie pas les contraintes de support alors le sous-arbre de ce noeud est élagué. Cette approche est exponentielle dans le pire des cas mais elle permet de considérer des motifs complexes. Le parcours de l'arbre peut être effectué en largeur ou en profondeur d'abord. Le parcours en largeur permet un élagage plus efficace en pratique mais sur des motifs très courts. [Karp *et al.*, 1972] proposent un algorithme très efficace utilisant ce type de parcours. Une extension de cette méthode a été proposée par [Sagot *et al.*, 1995b] pour trouver des motifs contenant des composants sans joker (cf motifs (Cb) de la table 1.2). L'efficacité de cette méthode est linéaire en la taille de la séquence.

En effectuant un parcours en profondeur, Neuwald *et al.* recherchent des motifs pouvant contenir des composants de deux lettres ainsi que des jokers [Neuwald et Green, 1994]. L'idée est que seuls les motifs de score élevé sont étendus par des jokers. Le mécanisme d'élagage est basé sur la mesure de significativité statistique des motifs. [Sagot et Viari, 1996] recherchent des motifs avec jokers contenant des composants. La complexité varie en fonction de la taille des composants.

Méthodes dirigées par les données

Ce type de méthode repose sur l'utilisation de l'ensemble des séquences pour guider la construction des motifs. La plupart de ces méthodes n'utilisent que des exemples positifs à l'exception de quelques approches comme [Kudo *et al.*, 1992], [Tateishi et Miyano, 1995] et [Tateishi *et al.*, 1995]. Les méthodes dirigées par les données sont basées sur la comparaison des séquences. Elles utilisent pour la plupart des alignements de séquences représentés de différentes façons.

Les alignements de [Schuler *et al.*, 1991] ou [Brodsky *et al.*, 1992] entre n séquences sont représentés par des blocs de taille n , ceux de [Vingron et Argos, 1991] sont représentés par des matrices booléennes.

Les deux algorithmes les plus connus utilisant l'ensemble des séquences pour guider la recherche sont Teiresias [Rigoutsos et Floratos, 1998a, Rigoutsos et Floratos, 1998b] et Splash [Califano, 2000]. Le but est de trouver des motifs maximaux en terme de longueur et de composition. Ces algorithmes permettent de considérer des motifs plus complexes que les méthodes dirigées par le modèle. Elle autorisent notamment la présence de jokers dans les motifs, de composants et de gaps.

Méthodes hybrides

De nombreux travaux combinent les deux méthodes précédemment citées. Les données sont alors utilisées pour raffiner les motifs trouvés par les méthodes dirigées par le modèle. Des motifs candidats peuvent être identifiés en utilisant une méthode dirigée par le modèle. Puis, les séquences sont alignées afin d'étendre les motifs candidats tant que le score du nouveau motif est plus élevé. [Smith *et al.*, 1990], [Jonassen *et al.*, 1995] et [Jonassen, 1997] utilisent cette méthode dont il existe diverses variantes telles que [Landraud *et al.*, 1989] et [Martinez, 1988].

Les méthodes que nous venons de présenter donnent de bons résultats pour l'identification de régions conservées dans les protéines même si les motifs ont un faible pouvoir d'expressivité et ne permettent pas de décrire précisément les similitudes entre les protéines. La limitation principale des motifs que nous avons présentés est qu'ils n'autorisent que les disjonctions ponctuelles limitées à des mots de longueur 1, ce qui limite considérablement le type de motif pouvant être inféré.

Certaines tentatives ont été réalisées pour générer des motifs reliés [Brazma *et al.*, 1997] afin de considérer des relations de dépendance entre les acides aminés par l'introduction de positions corrélées.

1.3.2 Prédiction de motifs appariés

Comme nous venons de le voir, les approches de découverte de motifs se focalisent sur la recherche de motifs conservés dans un ensemble de séquences. Elles ne prennent

pas en compte les interactions qui se produisent à l'intérieur ou entre plusieurs macro-molécules et qui déterminent la forme finale de la protéine.

Si ces mécanismes d'interaction sont utiles à la survie de l'organisme, ils vont avoir tendance à être préservés par le processus de l'évolution. Pour que ces interactions soient conservées, il faut, soit que les molécules concernées restent totalement inchangées dans les sites impliqués dans les interactions, soit que les changements dans ces molécules préservent les propriétés physico-chimiques nécessaires à l'interaction.

Dans le premier cas, les méthodes de découverte de motifs s'appliquent, à condition que ces motifs apparaissent dans le même ordre dans toutes les séquences. Dans le second cas, les conservations ne peuvent plus être représentées par de simples motifs.

Les règles d'appariement simples existant dans les séquences génomiques et plus particulièrement dans les ARN ont permis la conception d'algorithmes de prédiction de structure relativement fiables [Eskin et Pevzner, 2002]. Cela s'avère beaucoup plus difficile lorsqu'il s'agit de séquences protéiques. Quelques approches ont été réalisées pour déterminer des corrélations au sein du même motif structurel [Afonnikov *et al.*, 1997, Crooks et Brenner, 2004, Korber *et al.*, 1993, Larson *et al.*, 2000, Gobel *et al.*, 1994]. Elles se basent soit sur un alignement existant soit sur un motif déjà connu et localisé et utilisent des mesures basées sur les fréquences observées et attendues de paires de symboles ou sur l'information mutuelle.

Y. Mescam *et al.* ont développé un algorithme permettant de localiser des couples de sites conservant globalement leurs propriétés d'interactions par des mécanismes de mutation compensatoires [Gras *et al.*, 2003]. Pour cela ils utilisent l'algorithme MODEL [Hernandez *et al.*, 2004] qui cherche à localiser des zones suffisamment conservées pour préserver les propriétés physico-chimiques. Ils se focalisent sur la recherche de motifs dyadiques liés [Eskin et Pevzner, 2002], *i.e.* des motifs composés de deux sites distants en interaction.

La recherche de motifs appariés n'est qu'un cas particulier du problème plus général de la mise en relation de régions particulières de la séquence impliquées dans une interaction. Ceci suppose à la fois de détecter des contextes favorisant une certaine conformation de la molécule étudiée et de prédire les types de relations intervenant entre différents contextes.

1.4 Recherche de relations contextuelles sur des séquences de protéines

Lors du repliement de la chaîne peptidique, la structure est stabilisée par des interactions à longue distance entre les acides aminés. Si l'on dispose de suffisamment de ces contacts à longue distance, il est dans certains cas possible de construire un modèle plus

ou moins détaillé de la structure 3D des protéines. Étant donnée une protéine dont on connaît seulement la séquence primaire en acides aminés, différents types de prédictions sont aujourd'hui réalisables :

- prédiction de la position d'éléments de structure secondaire (hélice α , brin β) ;
- prédiction du repliement, appelée *threading*, par comparaison avec des repliements connus.

1.4.1 Prédiction de structures secondaires régulières

Des efforts importants de recherche ont été consacrés aux méthodes de prédictions des éléments de structure secondaire des protéines et nous nous y attarderons plus longuement. La plupart des méthodes actuelles utilisent un modèle de structure secondaire à trois états : le feuillet β , l'hélice α et la boucle ou structure irrégulière. Il s'agit de prédire dans lequel de ces trois états se trouve chaque acide aminé de la séquence. De nombreux systèmes de prédiction ont été développés.

Les grandes catégories de ces méthodes sont : les méthodes statistiques, les méthodes basées sur la similarité de séquences, les méthodes d'apprentissage et les méthodes empiriques.

Le tableau 1.4 récapitule les méthodes de prédiction existantes. Nous présentons ci-dessous quelques unes des méthodes les plus utilisées.

Les premiers résultats significatifs datent de 1974 [Chou et Fasman, 1978]. Les auteurs se basent sur les propriétés physico-chimiques des acides aminés définissant la stabilité de la protéine, telles que l'hydrophobicité. Pour cela ils utilisent des statistiques issues de l'étude systématique de la distribution des acides aminés dans les structures 3D de protéines disponibles dans la PDB. Le principe sous-jacent de la méthode, est de détecter s'il existe des biais de composition dans les hélices, les feuillets ou les boucles pour ensuite les utiliser à des fins prédictives. Chou et Fasman utilisent une table d'occurrences contenant la fréquence de chaque acide aminé dans un état structural donné qui est établie à partir des connaissances des structures tertiaires d'un échantillon de protéines modèles. Le principe est de parcourir la séquence en utilisant une fenêtre glissante de quatre acides aminés. Un score, ainsi qu'une probabilité de courbure, sont calculés sur chaque acide aminé en tenant compte des trois acides aminés suivants. Cette méthode présente une efficacité modérée de l'ordre de 50 à 60% mais ces taux sont basés, à l'époque, sur la connaissance de peu de structure.

La méthode GOR *et al.*, très semblable à la précédente dans son fonctionnement, considère les huit résidus en amont et en aval d'un résidu central d'une fenêtre de 17 acides aminés [Garnier *et al.*, 1978]. Des matrices de probabilités sont construites : matrice avec le résidu central dans une hélice alpha, dans un feuillet bêta, ou dans une boucle. Ils appliquent ensuite successivement ces matrices sur la fenêtre de taille 17 afin de déterminer la probabilité que le résidu central fasse partie de l'une ou l'autre des structures secondaires. Quatre résidus consécutifs doivent faire partie d'une hélice

Nom+Références	Méthode	structures prédites	taux de prédictions correctes(%)
[Chou et Fasman, 1974]	statistique	H	50
[Fasman, 1989]	statistique	H,E,C	60
[Levin <i>et al.</i> , 1986]	similarité	H,E,C	62
[Garnier <i>et al.</i> , 1978]	théorie de l'information	H,E,C	62,3
[Qian et Sejnowski, 1988]	NN	H,E,C	64,3
GOR IV [Garnier <i>et al.</i> , 1996]	théorie de l'information	H,E,C	64,4
NNPREDICT [Kneller <i>et al.</i> , 1990]	NN	H,E,C	65
[Zhang <i>et al.</i> , 1992]	NN	H,E,C	66,4
SIMPA [Levin, 1997]	PPV+	H,E,C	67,7
	alignement multiple		72,8
[Frishman et Argos, 1996]	NN+	H,E,C	68
	info alignement multiple		
[Levin <i>et al.</i> , 1993]	similarité	H,E,C	68,5
COMBINE [Francesco <i>et al.</i> , 1995]	statistique+	H,E,C	69
	alignement multiple		
SOPM [Geourjon et Deleage, 1994]	similarité+	H,E,C	69
	auto-optimisation		
SOPMA [Geourjon et Deleage, 1995]	auto-optimisation+	H,E,C	72,5
	info alignement multiple		
[Rost et Sander, 1994]	NN	H,E,C	72
	+ similarité		
[Martin <i>et al.</i> , 2005]	HMM+	H,E,C	72
	statistique		
SSPAL [Salamov et Solovyev, 1997]	PPV+	H,E,C	73,5
	alignement multiple		
[Won <i>et al.</i> , 2005]	multi HMM+	H,E,C	75
	alignement multiple		
JUFO [Meiler <i>et al.</i> , 2002]	NN	H,E,C	75
JPred [Cuff et Barton, 2000]	NN+	H,E,C	76,4
	alignement multiple		
PROF [Ouali et King, 2000]	NN+	H,E,C	76,7
	statistique		
[Ward <i>et al.</i> , 2003]	multi SVM+	H,E,C	77
	alignement multiple		
[Donnelly <i>et al.</i> , 1994]	table substitution+Fourier	H	79
	+info alignement multiple		
[Nguyen et Rajapakse, 2005]	2 multi-classes SVM+	H,E,C	79,5
	alignement multiple		

TAB. 1.4 – Algorithmes de prédiction de structure secondaire (H=Hélice, E=Feuillet et C=Boucle), PPV=plus proches voisins, HMM=Hidden Markov Model, NN=Neural Network, SVM=Support Vector Machine

alpha pour que cette prédiction soit validée et seulement deux résidus consécutifs sont nécessaires dans le cas d'un feuillet bêta. GOR utilise quelques principes de la théorie de l'information et les règles de Bayes pour effectuer ses prédictions avec un taux de succès de 62% en moyenne.

Les méthodes de deuxième génération sont celles appliquant une architecture connexionniste. Les interactions locales sont prises en compte. Les premiers travaux originaux viennent de Qian *et al.* qui utilisent les réseaux de neurones et obtiennent un taux de succès de 64% en moyenne [Qian et Sejnowski, 1988].

Par la suite, les méthodes exploitent la connaissance de l'arbre d'évolution des espèces. L'observation de base est que la structure secondaire à l'intérieur d'une famille de protéines apparentées est plus conservée que la structure primaire. Les méthodes se sont donc améliorées dans les années 1993 avec l'utilisation de l'alignement multiple des séquences, introduit par Rost et Sander [Rost et Sander, 1993]. Ils ont utilisé les réseaux de neurones couplés à un alignement multiple et ont atteint un taux de succès de 72% en moyenne [Rost et Sander, 1994].

La méthode SOPM de [Geourjon et Deleage, 1994] utilise un principe d'auto-optimisation des paramètres prédictifs sur une sous-base d'apprentissage. Pour chaque prédiction, la sous-base est régénérée afin qu'elle soit la mieux adaptée à la protéine en cours d'étude. Ils ont utilisé l'approche des plus proches voisins sur une base de données non redondante de structure secondaire de 609 chaînes protéiques. Ils ont obtenu un taux de succès de 68,8% en moyenne. La prise en compte dans la méthode SOPMA [Geourjon et Deleage, 1995], de l'alignement de séquences appartenant à la même famille de protéine a permis d'améliorer le taux de succès à 72,5% en moyenne. La méthode PHD [Rost, 1996] utilise des profils de matrices. Le système est composé d'une cascade de réseaux de neurones. Un nouveau réseau est utilisé pour raffiner les résultats obtenus par le réseau précédent. Ils obtiennent un taux de succès de 70% en moyenne testés sur une base ne contenant que 126 protéines. Cette méthode a été combinée avec la précédente sur les 609 protéines, ce qui permet d'atteindre des taux de reconnaissance élevés mais seulement sur une partie de la protéine. Le taux de succès pour les acides aminés co-prédits atteint les 82% pour 74% des acides aminés.

D'autres prédicteurs de structures secondaires utilisent l'approche des plus proches voisins tels que NNSP [Salamov et Solovyev, 1997]. NNSP utilise des segments de protéines de structure 3D connue et établit un score en fonction du degré de similarité de la structure secondaire du résidu test avec ces segments.

Les modèles de Markov cachés (HMM) peuvent aussi être utilisés pour prédire la structure secondaire, permettant ainsi d'incorporer des contraintes syntaxiques sur la forme des séquences de sortie. Cependant, jusqu'à présent, aucune méthode basée sur les HMM n'a été capable de prédictions aussi bonnes que les réseaux de neurones [Martin *et al.*, 2005].

Les machines à noyaux ont été utilisées dans [Hua et Sun, 2001] ainsi que les extensions multi-classes en combinant des classifieurs binaires [Ward et al., 2003, Ceroni et al., 2003a] ou des SVM (Support Vector Machine) multi-classes [Nguyen et Rajapakse, 2003, Nguyen et Rajapakse, 2005]. Cette approche permet de considérer les formations structurales du reste de la séquence et ne pas se focaliser uniquement sur un acide aminé ou un groupe d'acides aminés voisins. Ces méthodes obtiennent un taux de succès de 79% en moyenne.

Les performances de prédiction de la structure secondaire des protéines progressent de plus en plus lentement et ont probablement atteint un palier qu'il semble difficile à dépasser. La prédiction de ces structures est intéressante pour notre problème car elle fournit une information supplémentaire sur la localisation des cystéines. Diverses questions peuvent se poser, par exemple : est ce que toutes les cystéines dans les hélices sont impliquées dans des ponts disulfures ? Si cette question se transforme en affirmation, la prédiction des ponts disulfures commence par une prédiction de structure secondaire au niveau des cystéines et nous venons de montrer que les résultats obtenus pour ces prédictions sont très satisfaisants. Comme nous le verrons par la suite la difficulté majeure pour la prédiction des ponts disulfures est la prédiction de l'appariement des cystéines, *i.e.* être capable de prédire les paires de cystéines qui forment un pont disulfure. Si la question : est ce que toutes les cystéines oxydées dans une hélice forme un pont disulfure avec une cystéine dans une hélice ? se transforme en affirmation, ceci réduirait considérablement le problème de prédiction des ponts disulfures en limitant le nombre de conformation possible des ponts. La prédiction des feuillets β dans les protéines est un problème assez similaire à la prédiction des ponts disulfures. En effet, prédire les feuillets β c'est prédire des contextes en relation dans la séquence, ce qui est similaire à la prédiction de contextes disulfures en relation. Les taux des méthodes de prédiction annoncés précédemment se réfèrent à une moyenne des prédictions des hélices, des feuillets et des coudes. Si l'on regarde plus en détail la prédiction des feuillets β , les prédictions sont nettement inférieures aux prédictions totales.

Méthodes	Q_3	Q_H	Q_E	Q_C
[Rost et Sander, 1994]	70	72	66	72
[Riis et Krogh, 1996]	71,3	68,9	57	79,2
[Hua et Sun, 2001]	71,2	73	58	75
[Nguyen et Rajapakse, 2003]	77,7	74,3	55,2	88,7
[Nguyen et Rajapakse, 2005]	79,5	76,7	60,2	87,4

TAB. 1.5 – Comparaison des taux de prédiction, Q_3 =taux de prédiction total, Q_H =taux de prédiction des hélices, Q_E =taux de prédiction des feuillets et Q_C =taux de prédiction des boucles

Toutes les méthodes ne donnent pas le détail des différentes prédictions, le tableau 1.5 illustre quelques méthodes détaillant leur taux de prédiction. Nous pouvons

constater que les taux de prédiction des feuillets β (colonne Q_E) sont nettement inférieurs aux autres taux de prédiction. Cette prédiction est un problème difficile car elle prend en compte les interactions longues distance qu'il est difficile de modéliser.

La prédiction de la structure secondaire des protéines peut servir de point de départ à la prédiction de la structure 3D des protéines.

1.4.2 Prédiction de la structure tertiaire

On estime que le nombre de repliements 3D possibles est limité (entre 1000 et 10000 [Chothia, 1992, Zhang et DeLisi, 1998, Alexandrov et Go, 1994]), comparé au nombre de séquences protéiques (plusieurs millions). Aujourd'hui, la base de données CATH référence quelques 6100 familles de repliement pour un ensemble de plus de 32000 structures dans la PDB. Les méthodes existantes pour prédire la structure 3D des protéines sont :

- les méthodes *ab initio* ;
- les méthodes *de novo* ;
- les méthodes par homologie comparative ;
- et les méthodes de reconnaissance de repliements.

Les méthodes *ab initio* : elles tentent de prédire la structure des protéines à partir de la seule séquence d'acides aminés. Il s'agit d'une modélisation physique avec un calcul de minimisation d'énergie que l'on ne peut effectuer que sur des fragments courts et sous certaines hypothèses. Ces méthodes se basent sur les interactions physico-chimiques des acides aminés à partir desquelles sont calculées des fonctions de score [Brooks *et al.*, 1990, Weiner *et al.*, 1984]. Le coût de ces calculs est important car les interactions atomiques d'une protéine sont complexes et nombreuses (l'espace des conformations est potentiellement l'ensemble des partitions possibles sur l'ensemble des positions dans la séquence).

Les méthodes *de novo* : elles tentent de prédire la structure des protéines en extrayant au préalable des fragments de structures 3D dans les banques de données. Les deux principales méthodes sont SAMT-02 [Karplus *et al.*, 1999] et ROSETTA [Bonneau *et al.*, 2001]. Ces méthodes sont très prometteuses et ont déjà donné des résultats satisfaisants lors des dernières compétitions CASP² et CAFASP³ dont le but est de faire l'état de l'art des recherches en prédiction de structures de protéines.

Les méthodes par homologie : elles sont basées sur la comparaison de séquences. Ces méthodes considèrent que deux protéines ayant plus de 30% d'identité sont homologues et partagent une structure 3D similaire [Mika et Rost, 2003]. MODÉLER est un des algorithmes les plus utilisés pour la modélisation par homologie de séquences protéiques [Marti-Renom *et al.*, 2000]. Il compare une séquence inconnue avec d'autres dont la structure est connue en utilisant un alignement. Si

²<http://predictioncenter.org/casp6/Casp6.html>

³<http://www.cs.bgu.ac.il/~dfischer/CAFASP4/>

le taux de similarité est suffisant, la structure de la nouvelle protéine peut être inférée.

Les méthodes de reconnaissance de repliement (threading) : Deux séquences ayant moins de 30% d'identité peuvent être structurellement très proches [Chothia et Lesk, 1986]. On peut alors utiliser le threading, qui consiste à estimer la compatibilité entre la séquence de la protéine et un ensemble représentatif des repliements connus (comparaison avec une base de données de structures non homologues).

À la différence des méthodes *de novo* qui utilisent des fragments significatifs de différentes protéines, ces méthodes comparent une séquence entière avec une structure connue. Des squelettes de structures 3D de protéines connues, appelés *coeurs*, sont représentés par des graphes de contact dont les noeuds sont des blocs représentant des parties structurales conservées. Les arcs de ces graphes représentent soit les acides aminés en contact quand la protéine se replie, soit les interactions entre deux blocs. Le problème de reconnaissance de repliements (*Protein Threading Problem*), consiste alors à aligner les coeurs sur les séquences recherchées. Une fonction de score est ainsi calculée pour évaluer la compatibilité entre les acides aminés de la séquence et leurs positions dans le coeur. Cette recherche de l'alignement optimal est un problème NP-complet [Lathrop, 1994].

L'approche la plus efficace pour tenter de résoudre ce problème est l'utilisation des méthodes exactes dont la complexité est exponentielle dans le pire des cas. L'algorithme *Branch&Bound*, proposé par Lathrop et Smith, permet de résoudre ce problème sur des instances biologiques de petite taille (inférieure à 100 acides aminés) [Lathrop et Smith, 1996]. Afin de pouvoir travailler sur des instances de grande taille, la programmation linéaire en nombre entiers a été utilisée [Andonov *et al.*, 2003, Xu *et al.*, 2003, Yanev et Andonov, 2003, Yanev et Andonov, 2004]. Les méthodes utilisant la relaxation lagrangienne sont encore plus performantes [Veber *et al.*, 2005, Balev, 2004].

La prédiction de la structure 3D des protéines reste un problème très complexe. C'est pourquoi nous nous focalisons sur la prédiction des ponts disulfures afin de découper le problème de repliement en sous-parties.

1.5 Prédiction des ponts disulfures dans les protéines

Ainsi que nous l'avons décrit dans les sections précédentes, les protéines peuvent être stabilisées par des ponts disulfures formés uniquement entre des paires de cystéines lors du repliement [Matsumura et Matthews, 1989]. Errami *et al.* décrivent une stratégie afin d'améliorer une analyse statistique objective et exhaustive des structures des protéines [Errami *et al.*, 2003]. Ils ont montré que les acides aminés impliqués dans des ponts disulfures sont particulièrement bien conservés car la conformation de la protéine est liée à la conservation des propriétés physico-chimiques globales plutôt qu'à de faibles interactions. La prédiction de ces ponts disulfures peut être divisée en deux étapes. Premièrement, prédire l'état oxydé ou réduit des cystéines dans une séquence donnée. Une

Références	Méthodes	taux de prédictions correctes(%)
[Fariselli <i>et al.</i> , 1999]	réseaux de neurones + alignement multiple	81
[Muskal <i>et al.</i> , 1990]	réseaux de neurones	81,4
[Fiser et Simon, 2000]	classifieur + alignement multiple	82
[Mucchielli-Giorgi <i>et al.</i> , 2002]	fonctions logiques	84
[Frasconi <i>et al.</i> , 2002]	SVM	85
[Ceroni <i>et al.</i> , 2003b]	SVM	85
[Martelli <i>et al.</i> , 2002]	réseaux de neurones +HMM	88
[Song <i>et al.</i> , 2004]	discriminateur linéaire	89
[Chen <i>et al.</i> , 2004]	SVM	90

TAB. 1.6 – Algorithmes de prédiction de cystéines oxydées.

cystéine oxydée est impliquée dans un pont disulfure. Deuxièmement, prédire l'appariement de deux cystéines oxydées.

1.5.1 Prédiction des cystéines oxydées (cystines)

Les études effectuées sur la prédiction des cystéines oxydées reposent généralement sur l'analyse des propriétés physico-chimiques des contextes englobant chaque cystéine. [Muskal *et al.*, 1990] et [Fiser *et al.*, 1992] ont montré que ces contextes sont différents pour les cystéines et les cystines.

Le tableau 1.6 récapitule les différents travaux effectués sur la prédiction de l'état oxydé des cystéines. Nous présentons ces méthodes ci-dessous.

La plupart des méthodes travaillent sur des séquences protéiques issues de la PDB et ayant moins de 25% d'identité. C. Geourjon a créé une base de données de 1426 séquences protéiques ayant des ponts disulfures intra et inter moléculaires. Cette base de données a été créée pour les membres du groupe de travail de l'AS CNRS "Apprentissage et séquences" devenue ACI GenoTo3D. Elle est disponible sur le web⁴ et permet aux informaticiens de travailler sur une base de données commune afin de pouvoir comparer leurs taux de prédiction.

Muskal *et al.* [Muskal *et al.*, 1990] utilisent les réseaux de neurones et obtiennent un taux de succès, obtenu par une validation croisée 20-fold, de 81% de moyenne pour la prédiction de cystéines oxydées et 80% de moyenne pour la prédiction des cystéines libres, sur un ensemble de séquences protéiques ayant moins de 25% d'identité issues de la PDB.

⁴<http://www.loria.fr/~teguemeur/GdT/>

Le programme CYPRED⁵ développé par Fariselli *et al.* utilise un réseau de neurones alimenté par une fenêtre contenant $2k+1$ résidus centrée autour de la cystéine cible [Fariselli *et al.*, 1999]. Chaque élément de la fenêtre est un vecteur de 20 composants (un pour chaque acide aminé) obtenu à partir de profils d'alignements multiples. Cette méthode a un taux de succès de 79% en moyenne ($k = 8$) mesurée par une validation croisée 20-fold et utilisant un ensemble de 640 protéines non homologues issues de la PDB. Le taux de succès peut atteindre les 81% en utilisant 6 réseaux de neurones.

Le programme CYSREDOX⁶ développé par Fiser & Simon se base sur l'observation que les cystéines libres et les cystines sont rarement dans les mêmes protéines [Fiser et Simon, 2000]. Pour cela, leur algorithme effectue le test suivant : si une grande proportion des cystéines sont prédites dans une même classe alors les cystéines restantes sont classées dans cette même classe. Leur méthode atteint un taux de succès de 82% en moyenne, mesuré par un test *leave-one-out* appliqué au niveau des protéines sur un ensemble de 81 protéines alignées.

Plus tard, Mucchelli-Giorgi *et al.* proposent un prédicteur exploitant à la fois le contexte local, *i.e.* une fenêtre centrée sur les cystéines cibles, et le contexte global [Mucchielli-Giorgi *et al.*, 2002]. Ils montrent qu'en l'absence d'information évolutive, la prédiction de l'état covalent basée sur des descripteurs globaux a un meilleur taux de succès (77% en moyenne) que celle basée sur des descripteurs locaux (67% en moyenne). Leur meilleur prédicteur basé sur un classifieur multiple atteint les taux de succès de 84% en moyenne mesuré par une validation croisée 5-fold sur un ensemble de 559 protéines issues de PDB.

Frasconi *et al.* utilisent un prédicteur SVM prenant en compte le fait que toutes les cystéines dans une protéine sont généralement dans le même état [Frasconi *et al.*, 2002, Ceroni *et al.*, 2003b]. Ils emploient pour cela deux classifieurs binaires, l'un agissant sur le contexte global de la protéine, et l'autre agissant sur le contexte local des cystéines. Ils travaillent sur un ensemble de 716 protéines non homologues en effectuant une validation croisée 5-fold. Leur taux de succès atteint les 85%.

En 2002, le meilleur taux de succès atteint est de 88% selon Martelli *et al.* [Martelli *et al.*, 2002]. Ils ont utilisé un système qui combine un réseau de neurones identique à CYPRED et un modèle de Markov. Ce taux a été mesuré par une validation croisée 20-fold sur un ensemble de 969 protéines non homologues issues de la PDB.

En 2004, Song *et al.* ont travaillé sur un discriminateur linéaire plus simple basé sur la composition dipeptidique des protéines [Song *et al.*, 2004]. La prédiction a été

⁵<http://gpcr.biocomp.unibo.it/predictors/cyspred/>

⁶<http://pipe.rockefeller.edu/cysredox/cysredox.html>

effectuée sur 1856 protéines non homologues et atteint un taux de succès de 89% en moyenne pour la prédiction de l'état oxydé des cystéines.

Le meilleur taux de succès de 90% en moyenne a été atteint par Chen *et al.* qui proposent méthode SVM basée sur des vecteurs de caractéristiques multiples couplés avec des séquences d'états des cystéines [Chen *et al.*, 2004]. Ceci permet de combiner les séquences locales et les compositions globales des acides aminés. La prédiction a été effectuée sur la même base de données que celle dans [Martelli *et al.*, 2002].

Si les résultats de ces méthodes sont satisfaisants car ils atteignent les 90% de prédiction correcte pour l'oxydation des cystéines, elles présentent toutefois l'inconvénient de produire des résultats difficilement interprétables. Ces méthodes "boîtes noires" ne permettent pas de construire un modèle général de contexte de cystéines.

Nous présentons dans cette thèse une nouvelle approche pour prédire de manière explicite l'état des cystéines dans les protéines. Cette première étape de la prédiction de l'état des cystéines va servir de point de départ pour tenter de prédire la formation du pont disulfure dans sa globalité, *i.e.* être capable de prédire l'appariement des cystéines oxydées. Ce deuxième point est un problème très complexe encore mal résolu que nous décrivons dans la sous-section suivante.

1.5.2 Prédiction de l'appariement des cystéines oxydées

Le but ici est d'être capable de distinguer quelles sont les deux cystéines qui vont s'apparier. Par la suite nous appellerons *contexte disulfure* le contexte d'une cystéine oxydée, *i.e.* le contexte centré sur la cystéine et contenant ses acides aminés voisins dans la séquence. Lors de la formation d'un pont disulfure, deux contextes disulfures vont s'apparier et se retrouver l'un en face de l'autre. Actuellement, la localisation des ponts disulfures est prédite à partir de la connaissance de l'état oxydé des cystéines. Ce problème est complexe car il explose combinatoirement en fonction du nombre de ponts dans la protéine. En effet, étant donné $2B$ cystéines, le nombre de connexions possibles est $(2B - 1)!! = \prod_{i=1}^B (2i - 1)$. C'est pourquoi, les prédictions se distinguent selon le nombre de ponts disulfures dans les protéines. Les bases de données utilisées sont les mêmes que pour la prédiction des cystéines oxydées, *i.e.* un ensemble de séquences non homologues issues de la PDB. La figure 3.1 montre un exemple de structure de protéine ayant 5 ponts disulfures.

Les approches prédictives existantes utilisent l'optimisation globale stochastique [Fariselli et Casadio, 2001], [Boisbouvier *et al.*, 2000], l'optimisation combinatoire [Klepeis et Floudas, 2003] et les techniques d'apprentissage automatique [Fariselli *et al.*, 2002, Vullo et Frasconi, 2004]. La méthode utilisée dans [Fariselli et Casadio, 2001] représente l'ensemble des ponts disulfures dans une séquence comme un graphe pondéré complet non dirigé. Les sommets représentent les cystéines oxydées et les arcs sont pondérés par le contact potentiel entre les deux

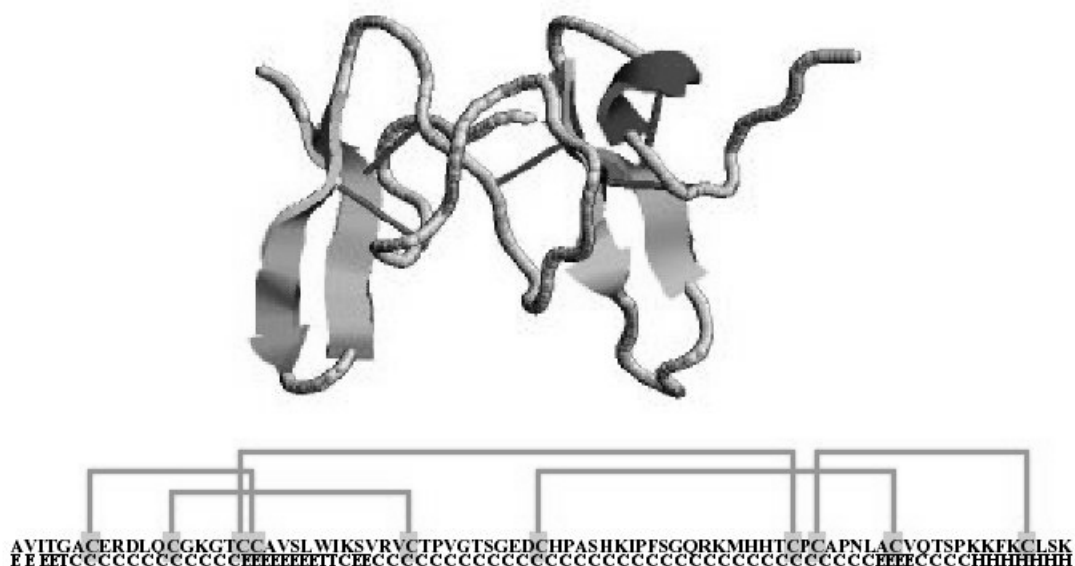


FIG. 1.4 – Structure 3D et structures primaire et secondaire associées de la protéine limt (Mamba Intestinal Toxin 1)

Références	Méthodes	Nb Ponts	Prédictions correctes(%)	Facteur Amélioration\ Aléatoire
[Fariselli et Casadio, 2001]	Réseaux de neurones	2	56	1,7
		3	36	1,7
		4	37	2,6
		5	21	1,9
[Vullo et Frasconi, 2004]	Réseaux de neurones	2	71	2,1
		3	47	2,3
		4	29	2
		5	33	2,9

TAB. 1.7 – Tableau récapitulatif des résultats de [Fariselli et Casadio, 2001] et [Vullo et Frasconi, 2004] par rapport à l'aléatoire

cystéines. Un recuit simulé est utilisé pour trouver un ensemble de poids, pour les arcs du graphe, potentiels. Les auteurs travaillent sur des ensembles de protéines non homologues contenant de 2 à 5 ponts disulfures.

A. Fariselli *et al.* utilisent des réseaux de neurones pour apprendre les contacts potentiels et ainsi pondérer un graphe comme précédemment [Fariselli et Casadio, 2001, Fariselli *et al.*, 2002]. Cette méthode donne des résultats corrects pour des protéines ne possédant pas plus de deux ponts disulfures. Le taux de succès est de 56% en moyenne pour 2 ponts disulfures par protéine, c'est à dire seulement 1,7 fois mieux que l'aléatoire (cf tableau 1.7), 21% pour 3 ponts, 17% pour 4 ponts et 2% pour 5 ponts par protéines.

Une autre approche est utilisée dans [Klepeis et Floudas, 2003]. Rechercher la localisation des ponts disulfures et une partie d'un protocole plus général qui vise à prédire les feuillets β dans les protéines. Cette approche suppose que la formation de feuillet β est liée à des interactions hydrophobes plutôt qu'hydrogène. Les contacts entre résidus sont prédits en résolvant une série de problèmes de programmation linéaire en nombres entiers dans lesquels les contacts hydrophobiques doivent être maximaux. Les contraintes du modèle définissent des feuillets admissibles et des configurations de ponts disulfures. L'intérêt de cette méthode c'est qu'elle peut prédire des contacts cystéine-cystéine indépendamment de la connaissance de l'état d'oxydation des cystéines. Les auteurs n'ont pas testé leur prédiction sur un échantillon comparable aux précédents résultats.

Vullo *et al.* ont travaillé sur le même ensemble de séquences que celui dans [Fariselli et Casadio, 2001] afin de pouvoir comparer au mieux leurs résultats [Vullo et Frasconi, 2004, Vullo et Frasconi, 2003]. Un contexte disulfure est représenté en terme de graphe non orienté où les noeuds représentent les cystéines oxydées et les arcs correspondent aux ponts disulfures. Une cystéine ne peut être reliée qu'à une seule autre cystéine. Le but est de trouver le meilleur candidat possible étant donnée une fonction de score. Ils ont développé une architecture *ad-hoc* RNN (réseaux de neurones récurrents) afin de pondérer les graphes non orientés étiquetés représentant des modèles connectés. Vu la complexité du problème, ils se limitent à des protéines ne contenant pas plus de cinq ponts disulfures. Les taux de succès sont de 71% pour deux ponts par protéine, ce qui ne représente que 2,1 fois mieux que l'aléatoire (cf tableau 1.7), 33% pour 3 ponts, et 10% pour 4 et 5 ponts par protéine. Les résultats obtenus sont meilleurs que ceux de Fariselli *et al.* mais doivent encore être approfondis.

Les résultats obtenus jusqu'à présent ne sont pas satisfaisants pour la prédiction des ponts disulfures dans les protéines. Ceci est principalement dû au fait que les interactions longues distances sont difficilement modélisables. Cette thèse propose l'exploration de deux nouvelles pistes pour progresser dans la résolution de ce problème difficile de prédiction de ponts disulfures dans les protéines, en utilisant dans un premier temps l'inférence grammaticale, puis dans un deuxième temps la programmation logique in-

ductive. La présentation de ces deux méthodes fait l'objet du chapitre suivant.

Chapitre 2

Apprentissage de relations sur des séquences

L'objectif de ce chapitre est de présenter en détail les choix et les techniques utilisées pour apprendre dans le cadre applicatif auquel nous nous confrontons. Après une courte introduction au domaine de l'apprentissage automatique, nous nous focalisons sur deux méthodes principales en apprentissage de relations sur des données : l'inférence grammaticale et la programmation logique inductive. Concernant le premier type de méthode, nous rappelons quelques notions de la théorie des langages, puis nous positionnons nos travaux par rapport à l'inférence de langages réguliers et algébriques. Concernant la deuxième approche, nous rappelons quelques notations et définitions de la logique des prédicats et nous détaillons le positionnement de nos travaux. Pour plus de détails sur ces méthodes, le lecteur intéressé pourra se référer aux ouvrages [Cornuéjols et Miclet, 2002], [Mitchell, 1997], [Muggleton et Raedt, 1994], [Lloyd, 1987] et [Lavrac et Dzeroski, 1994].

2.1 Notions de base sur l'apprentissage automatique

Nos travaux se placent dans le cadre de l'apprentissage artificiel. Nous reprenons la définition de T. Mitchell [Mitchell, 1997] de l'apprentissage automatique :

Définition 2.1 (apprentissage) *Un programme informatique apprend une tâche \mathcal{T} à partir de l'expérience \mathcal{E} et de la mesure de performance \mathcal{P} , si sa capacité à exécuter la tâche \mathcal{T} , mesurée par \mathcal{P} , augmente avec l'expérience \mathcal{E} .*

Pour spécifier un problème d'apprentissage il faut définir correctement la tâche \mathcal{T} à apprendre, la mesure de performance \mathcal{P} à optimiser, et l'ensemble des données \mathcal{E} d'apprentissage sur lequel le programme se base pour apprendre.

Dans notre cas, la tâche \mathcal{T} représente la classification des cystéines à partir de la séquence primaire des protéines, l'expérience \mathcal{E} est une base de données de protéines

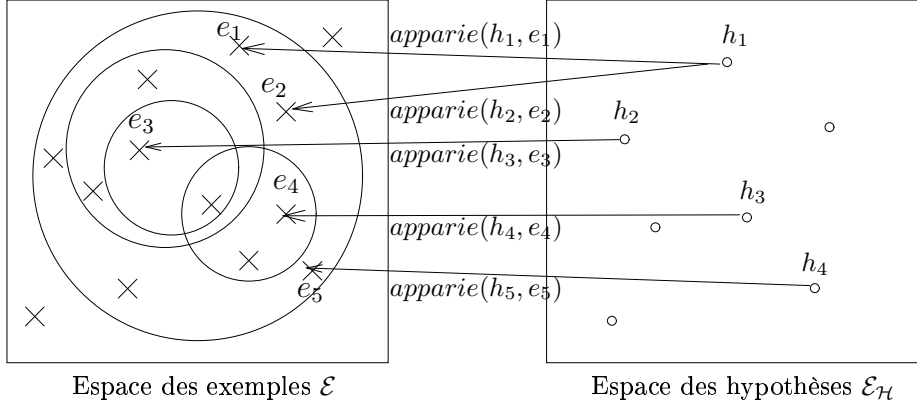


FIG. 2.1 – Représentation de l'espace des exemples et de l'espace des hypothèses

dont les ponts disulfures sont déjà connus, et la mesure de performance \mathcal{P} est le pourcentage de cystéines correctement classées.

De manière générale, le but de l'apprentissage artificiel est d'expliquer un ensemble d'observations à l'aide d'un ensemble fini minimal de modèles (hypothèses). Ce processus d'induction doit donc manipuler deux classes d'objets distincts : l'espace des exemples \mathcal{E} qui représente les observations et l'espace des hypothèses $\mathcal{E}_{\mathcal{H}}$ représentant les modèles. Les observations peuvent parfois se découper en plusieurs classes ; en particulier, dans nombre de problèmes, on considère le cas d'un ensemble \mathcal{E} composé d'exemples positifs \mathcal{E}_+ et d'exemples négatifs \mathcal{E}_- du concept que l'on cherche à acquérir. Deux langages doivent ainsi être définis afin de manipuler ces classes, il s'agit du langage des exemples noté $\mathcal{L}_{\mathcal{E}}$ et du langage des hypothèses noté $\mathcal{L}_{\mathcal{H}}$.

Une relation d'appariement, notée *apparie* sur la figure 2.1, permet de faire le lien entre les deux espaces \mathcal{E} et $\mathcal{E}_{\mathcal{H}}$, *i.e.* de pouvoir faire correspondre une hypothèse h décrite dans le langage $\mathcal{L}_{\mathcal{H}}$ à des éléments de \mathcal{E} décrits avec $\mathcal{L}_{\mathcal{E}}$.

On peut spécifier une relation d'ordre partiel dite de généralité sur l'espace des hypothèses vérifiant $\{e|apparie(h_1, e)\} \supseteq \{e|apparie(h_2, e)\}$. La problématique générale en apprentissage automatique consiste à définir syntaxiquement la relation de généralité entre deux éléments, sans se référer aux ensembles d'instances correspondant. Cette relation, que l'on note $\succeq: \mathcal{E}_{\mathcal{H}} \rightarrow \mathcal{E}_{\mathcal{H}}$ ($h_1 \succeq h_2$ (resp. $h_1 \preceq h_2$) se lit h_1 est plus générale (resp. spécifique) que h_2), doit bien sur être compatible avec la relation d'appariement pour être utile et on a la relation fondamentale :

$$h_1 \succeq h_2 \Leftrightarrow \{e|apparie(h_1, e)\} \supseteq \{e|apparie(h_2, e)\}$$

Le schéma 2.1 peut ainsi être enrichi par une relation de couverture notée *couvre* et par la relation \succeq , comme illustré dans la figure 2.2. L'exploration de l'espace d'hypothèses $\mathcal{E}_{\mathcal{H}}$ pourra se faire sans énumérer toutes les hypothèses de manière exhaustive mais en suivant une stratégie tirant profit de l'ordre de généralité, par

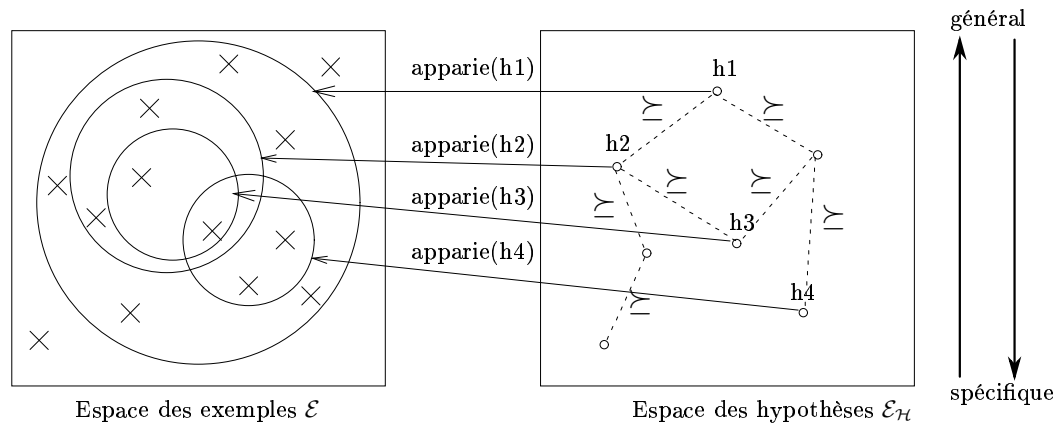


FIG. 2.2 – Ordre sur les hypothèses

exemple, une stratégie du “plus général” au “plus spécifique”.

On peut définir un lien entre le cadre d’inférence et le cadre plus général concernant les algorithmes d’induction à partir d’exemples positifs et négatifs. T. Mitchell [Mitchell, 1977] a défini l’espace des versions représentant l’ensemble des hypothèses consistantes avec les exemples d’apprentissage. Il a défini une relation d’ordre pour les hypothèses : de la plus spécifique à la plus générale. L’apprentissage se traduit alors comme un problème de parcours d’un espace de recherche.

Le treillis des solutions utilisé en inférence grammaticale peut être vu comme un cas particulier de l’espace des versions [Mitchell, 1977]. Les hypothèses sont des automates déterministes et la relation d’ordre est l’inclusion des langages pour l’inférence grammaticale. Pour Mitchell, la borne maximale représente l’hypothèse la plus générale, ce qui est équivalent à l’automate universel en inférence grammaticale, et la borne minimale représentant l’hypothèse la plus spécifique est équivalente à l’ensemble frontière pour les treillis d’automates. On peut considérer simplement que le $\text{PTA}(\mathcal{E}_+)$ est l’unique généralisation la plus spécifique. Toutes ces notions vont être détaillées dans ce qui suit.

Nous nous intéressons maintenant au cas où les exemples sont des mots et les hypothèses des représentations de langage. La relation d’appariement se confond alors avec la notion d’appartenance à un langage. Après une présentation des principaux résultats du domaine, nous décrirons quelques travaux ayant été appliqués aux séquences biologiques.

2.2 L'inférence grammaticale

Dans cette section nous présentons le cadre sur lequel l'inférence grammaticale repose, à savoir la théorie des langages. Nous présentons les notions de grammaires et d'automates pour représenter les langages. Nous consacrons la sous-section 2.2.4 à l'apprentissage de langages réguliers, accompagnée d'une description des algorithmes que nous avons utilisés pour nos expérimentations. Les sous-sections suivantes seront consacrées à l'apprentissage de langages algébriques et à diverses applications biologiques.

2.2.1 Mots et langages

Définition 2.2 (alphabet et mots) *Un alphabet Σ est un ensemble fini de symboles. Nous nous intéressons aux chaînes u de longueur finie sur Σ ($u \in \Sigma^*$) appelées mots ou séquences. Le mot vide est représenté par le symbole ε et $|u|$ désigne la longueur du mot u .*

Dans notre cadre applicatif, nous utilisons en particulier l'alphabet des acides aminés, *i.e.* l'ensemble des 20 résidus qu'utilisent les protéines, et les mots représentent des protéines.

Définition 2.3 (langage) *Un langage \mathcal{L} est un sous-ensemble de Σ^* .*

Nous cherchons ainsi à caractériser le langage des contextes possibles de cystéines oxydées dans les protéines. Nous y reviendrons dans le chapitre 3.

La représentation d'un langage pourrait se faire par l'énumération du sous-ensemble de mots de Σ^* qui le compose mais d'une part, ceci ne permettrait de représenter que des langages finis, et d'autre part ne permettraient pas de représenter la structure interne des mots. C'est pourquoi, plusieurs systèmes finis d'analyse ou de génération de langages ont été développés, dont la principale que nous décrivons maintenant est la représentation par grammaires formelles. Ce qui suit est un rappel de quelques définitions utiles pour la compréhension de cette thèse.

2.2.2 Représentation par système de règles

Une grammaire est un système de règles permettant de définir un langage formel. C'est cette représentation que nous allons utiliser par la suite et plus particulièrement les grammaires algébriques.

Définition 2.4 (grammaire) *Une grammaire est un quintuplet $G=(\Sigma, N, P, S)$ telle que Σ est un alphabet fini de terminaux, N est un alphabet fini de non-terminaux, P est un ensemble fini de règles de production (ou réécriture) de la forme $A \rightarrow w$ où $A \in (N \cup \Sigma)^+$ et $w \in (N \cup \Sigma)^*$, S est un non terminal (axiome) initial.*

Le langage L engendré par une grammaire G est l'ensemble $L(G) = \{w \in \Sigma^* / S \Rightarrow^* w\}$, où le symbole de dérivation \Rightarrow^* indique un nombre quelconque d'application des règles

de production. Par exemple, la grammaire $G = (\{a, b\}, \{S\}, P, S)$ dont les règles de production de P sont de la forme :

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow bS \\ S &\rightarrow \varepsilon \end{aligned}$$

reconnaît le langage $L(G) = \{a, b\}^*$; elle est appelée grammaire *universelle*. Une telle grammaire reconnaît tous les mots appartenant à Σ^* .

N. Chomsky [Chomsky, 1965] propose une classification des langages à partir du niveau de représentation nécessaire en fonction de la forme des règles de production de la grammaire. Elle est formée des quatre niveaux suivants, du plus restrictif au plus large :

- les langages réguliers ;
- les langages algébriques ;
- les langages contextuels ;
- les langages à structure de phrase.

Les quatre types ci-dessus sont strictement inclus les uns dans les autres. Dans le cadre de notre travail, nous nous intéressons plus particulièrement aux grammaires régulières et algébriques que l'on peut définir comme suit :

Définition 2.5 (grammaire régulière) *Une grammaire est dite régulière si les éléments de P sont de la forme $A \rightarrow aB$ ou $A \rightarrow a$ avec $A, B \in N$ et $a \in \Sigma$. Un langage engendré par une telle grammaire est appelé langage régulier.*

Définition 2.6 (grammaire algébrique + linéaire) *Une grammaire est dite algébrique (ou context-free) si toutes les transitions $A \rightarrow \alpha$ sont telles que $A \in N$, $\alpha \in (N \cup \Sigma)^*$. Une telle grammaire engendre un langage dit algébrique. Une grammaire algébrique est linéaire si de plus $\alpha \in (\Sigma^* N \Sigma^* \cup \Sigma)$.*

La figure 2.3 illustre l'expressivité des différentes classes de grammaires. Décrire les ponts disulfures dans les séquences protéiques nécessite de modéliser des dépendances de nature variée entre cystéines : les dépendances locales correspondent à une succession de ponts sans chevauchement dans la séquence, les dépendances parenthésées autorisent une imbrication des ponts et les dépendances croisées autorisent le chevauchement des ponts. Comme nous venons de le voir, pour représenter les ponts disulfures dans les protéines nous avons besoin d'un haut niveau d'expressivité dans la hiérarchie de Chomsky, car ceux-ci dans la réalité peuvent se chevaucher ou se croiser. Le problème est donc très complexe car il est nécessaire d'avoir recours aux langages au moins contextuels. Il existe en effet des algorithmes permettant d'analyser un langage hors-contexte de manière polynomiale, alors qu'en ce qui concerne les langages contextuels, les algorithmes sont exponentiels. C'est pourquoi, lors de nos expérimentations, nous ne traiterons pas dans toute sa généralité le cas des ponts qui se croisent afin de ne pas avoir à utiliser un niveau d'expressivité trop grand dans la hiérarchie de Chomsky.

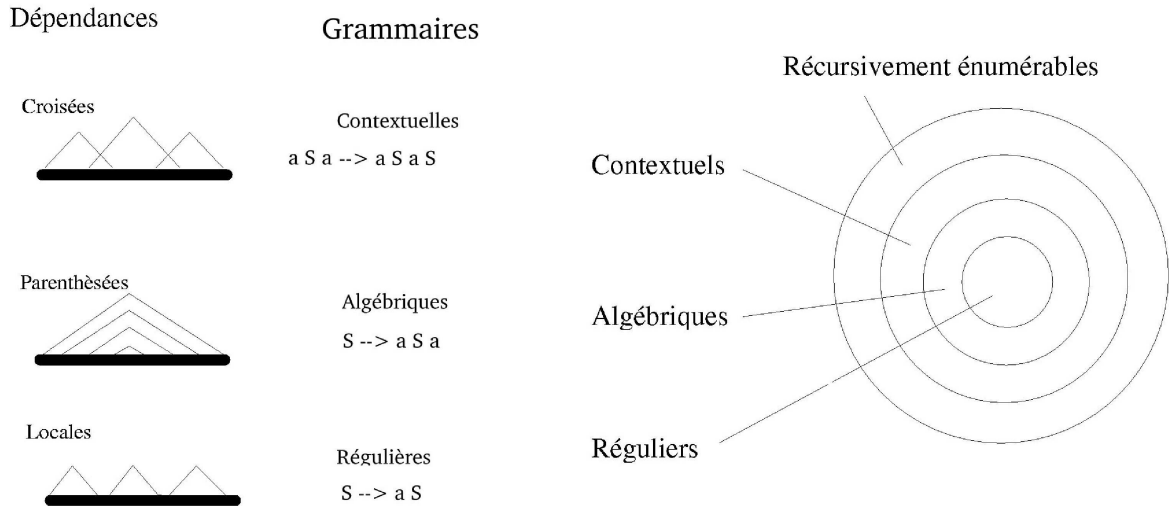


FIG. 2.3 – Hiérarchie de Chomsky et modélisation de structures biologiques [Searls, 1997]

Nous reviendrons sur ce point au chapitre 3.

On peut aussi représenter les langages à l'aide de machines. L'identification d'une machine, vue comme une boîte noire, à partir d'exemples de fonctionnement est une situation d'apprentissage couramment traitée en Intelligence Artificielle. Si les entrées-sorties sont sous une forme symbolique, on peut chercher à décrire le fonctionnement de la boîte noire par une machine à états finie. Une machine à états finie est un système dynamique discret qui traduit une séquence de symboles d'entrée en une séquence de symboles de sortie. La traduction est assurée par un parcours sur un ensemble d'états selon une fonction de transition qui définit les changements d'états, en fonction du symbole d'entrée. Les machines à états finies reconnaissant ou acceptant une séquence sont les automates à états finis. Ces automates sont équivalents aux grammaires régulières et permettent de plus une représentation graphique. Dans cette thèse, nous utilisons ce type de représentation car elle offre une excellente perspective globale sans négliger les caractères locaux des langages.

2.2.3 Représentation par automate

Les automates à états finis permettent de représenter les langages réguliers. Cette représentation est souvent utilisée en inférence grammaticale.

Définition 2.7 (FSA : automate d'états finis) *Un automate d'états finis est un quintuplet $(Q, \Sigma, \delta, I, F)$ où Q est un ensemble fini d'états, Σ un alphabet fini et δ une fonction de transition, application de $Q \times \Sigma \rightarrow 2^Q$. I est l'ensemble des états initiaux et F est un sous-ensemble de Q identifiant les états finals ou d'acceptation.*

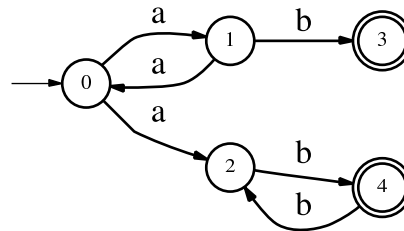


FIG. 2.4 – Un automate ambigu

Un langage L est reconnu par un automate $(Q, \Sigma, \delta, I, F)$ si et seulement si $\forall w \in L, \exists q_0 \in I, \delta(q_0, w) \cap F \neq \emptyset$. L'automate universel est celui qui reconnaît Σ^* .

Un exemple d'automate est illustré figure 2.4. Cet automate reconnaît le langage des mots commençant par un nombre impair de a suivi d'un nombre impair de b , c'est à dire le langage $(aa)^*a(bb)^*b$. Il se représente par le quintuplet $(\{0, 1, 2, 3, 4\}, \{a, b\}, \delta, 0, \{3, 4\})$. La fonction de transition correspond aux arcs reliant les états (par exemple $\delta(0, a) = \{1, 2\}$). Les états sont représentés par des cercles ($Q = \{0, 1, 2, 3, 4\}$), l'état initial est représentée par un cercle possédant une flèche entrante reliée à aucun autre cercle ($q_0 = 0$) et les états finals sont représentés par des états doublement cerclés ($F = \{3, 4\}$). On dit qu'une séquence est *acceptée* par un automate A s'il est possible, à partir de l'état initial, d'associer un état à chacune des lettres de la séquence et si la dernière lettre correspond à un état final (par exemple ab est reconnue par l'automate de la figure 2.4).

On peut facilement passer d'un automate à une grammaire. Par exemple la grammaire G représentant l'automate de la figure 2.4 est $G = (\{a, b\}, \{S_0, S_1, S_2, S_3, S_4\}, P, S_0)$ où P est l'ensemble des règles suivantes :

$$\begin{array}{ll}
 S_0 \rightarrow aS_1 & S_2 \rightarrow bS_4 \\
 S_0 \rightarrow aS_2 & S_3 \rightarrow \varepsilon \\
 S_1 \rightarrow aS_0 & S_4 \rightarrow \varepsilon \\
 S_1 \rightarrow bS_3 & S_4 \rightarrow bS_2
 \end{array}$$

Théorème 2.1 (Kleene) *Un langage est régulier ssi c'est le langage d'un automate fini.*

Définition 2.8 (DFA : automate d'états finis déterministe) *A est un automate d'états finis déterministe si, $\forall q \in Q$ et $\forall a \in \Sigma$, $\delta(q, a)$ possède au plus un élément. Tout automate non déterministe peut être représenté par un DFA équivalent.*

Divers travaux d'inférence grammaticale ont été effectués dans la classe des langages réguliers qui est la classe la plus restrictive. Or, comme nous l'avons vu précédemment, les grammaires nécessaires pour modéliser les ponts disulfures sont des grammaires au moins algébriques, puisqu'il s'agit de repérer des structures d'appariement dans la séquence. Néanmoins, nous détaillons en section 2.2.5 comment il est théoriquement possible de se restreindre aux langages réguliers pour l'apprentissage de ces langages. Dans le paragraphe suivant nous présentons le principe des algorithmes d'inférence régulière basés sur la fusion d'états. Nous détaillons ensuite ceux que nous avons utilisés, RPNI et Blue-Fringe.

2.2.4 Apprentissage d'automates finis

De nombreux travaux ont été effectués sur l'apprentissage d'automates finis conduisant à de multiples variantes d'un algorithme principal. Le principe de ces algorithmes, dont le pseudo-code est illustré ci-après, est la progression (souvent gloutonne) dans un espace des automates possibles, par fusion d'états. Les variantes se distinguent par le choix des états à fusionner.

Définition 2.9 (partition et raffinement de partition) Une partition π d'un ensemble \mathcal{E} est un ensemble de blocs, deux à deux disjoints, non vides, dont l'union est égale à \mathcal{E} . On note $B_\pi(e)$ le bloc B d'une partition π sur \mathcal{E} contenant l'élément $e \in \mathcal{E}$.

L'ensemble des partitions d'un ensemble fini peut être ordonné par la relation suivante : Soit π_1 et π_2 deux partitions d'un ensemble fini. On dit que π_1 est plus fine que π_2 , noté $\pi_1 \preceq \pi_2$, si tout bloc de π_1 est inclus dans un bloc de π_2 .

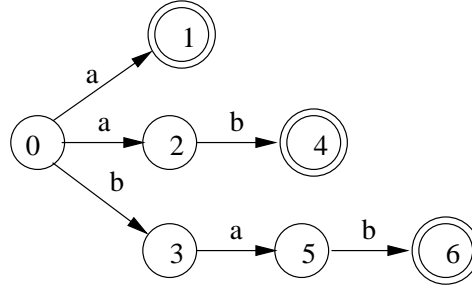
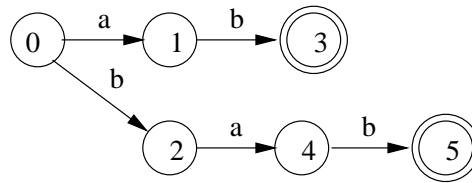
Définition 2.10 (FSA dérivé) Un automate dérivé du FSA $A = (Q, \Sigma, I, F, \delta)$ par la partition π est le FSA $A/\pi = (Q', \Sigma, I', F', \delta')$ tel que :

- $Q' = \{B_\pi(q) \mid q \in Q\} = \pi$
- $I' = \{B_\pi(q) \mid q \in I\}$
- $F' = \{B_\pi(q) \mid q \in F\}$
- $\delta' : Q' \times \Sigma \rightarrow 2^{Q'} : \forall B \in Q', \forall l \in \Sigma, \delta'(B, l) = \bigcup_{q \in B, q' \in \delta(q, l)} B_\pi(q')$.

Définition 2.11 (fusion d'états) La fusion des états q_1 et q_2 du FSA A conduit au FSA dérivé A/π où $\pi = \{\{q\} \mid q \in (Q \setminus \{q_1, q_2\})\} \cup \{\{q_1, q_2\}\}$.

Une fusion d'états effectuée sur un automate déterministe ne garantit pas de conserver le déterminisme. Pour cela, il est possible d'utiliser une fonction pour déterminisation qui rend tout automate déterministe. La fusion d'états agit sur la structure des automates. Elle permet une généralisation du langage accepté par l'automate : si un automate A_1 dérive d'un automate A_2 alors le langage accepté par A_2 est inclus dans A_1 . Sa faible complexité calculatoire permet de l'utiliser en pratique.

Cette relation est un ordre partiel sur l'ensemble des partitions. On peut naturellement l'étendre aux automates quotients : $A/\pi_1 \preceq A/\pi_2$ si $\pi_1 \preceq \pi_2$.

FIG. 2.5 – $MCA(\mathcal{E}_+)$ avec $\mathcal{E}_+ = \{a, ab, bab\}$ FIG. 2.6 – $PTA(\mathcal{E}_+)$ avec $\mathcal{E}_+ = \{a, ab, bab\}$

Proposition 1 (treillis des automates (lattice)) *L'ensemble des automates dérivés d'un automate A , partiellement ordonné par la relation \preceq , forme un treillis fini noté $\text{Lat}(A)$. L'automate A correspond au plus petit minorant et l'automate universel (UA) correspond au plus grand majorant.*

Plusieurs automates pouvant représenter le même langage, on appelle *automate canonique* d'un langage L , noté $A(L)$, le plus petit automate (en nombre d'états) déterministe représentant L . L'identification exacte d'un langage à partir d'un échantillon d'apprentissage ne peut être assurée que si l'échantillon est suffisamment "représentatif" du langage à apprendre. Pour l'inférence de grammaires régulières, l'échantillon positif \mathcal{E}_+ est généralement supposé structurellement complet relativement au langage cible :

Définition 2.12 (structurellement complet) *Un échantillon \mathcal{E}_+ est structurellement complet relativement à un langage $L(G)$ si chaque règle de production de la grammaire G est utilisée pour la génération d'au moins un mot de \mathcal{E}_+ .*

Pour un échantillon \mathcal{E} , on peut construire l'*automate canonique maximal* $MCA(\mathcal{E}_+)$ pour lequel \mathcal{E} est structurellement complet. Cet automate en forme d'arbre est composé d'une branche par mot de \mathcal{E} . On peut définir l'automate canonique maximal comme suit :

$MCA(\mathcal{E}_+)$ est l'automate, souvent non-déterministe, ayant le plus grand nombre d'états pour lequel \mathcal{E}_+ est structurellement complet et $L(MCA(\mathcal{E}_+)) = \mathcal{E}_+$. Un exemple de MCA est illustré figure 2.5.

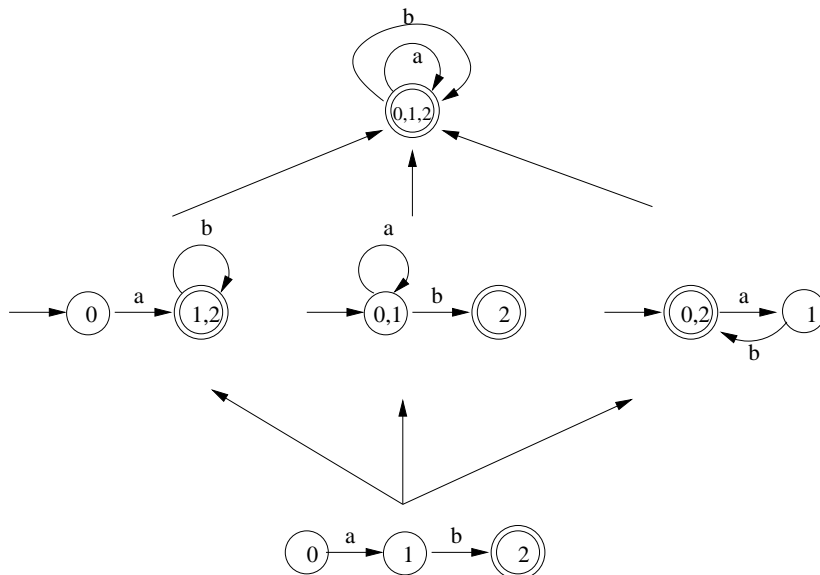


FIG. 2.7 – Exemple de treillis

Dans le cas de l'inférence d'automates déterministes, le MCA est remplacé par sa version déterministe à savoir l'arbre accepteur de préfixes. Il se définit de la manière suivante :

Définition 2.13 (*PTA*(\mathcal{E}_+), **Prefix Tree Acceptor**) L'arbre accepteur de préfixes de \mathcal{E}_+ est l'automate quotient $MCA(\mathcal{E}_+)/\Pi_{\mathcal{E}_+}$ où la partition $\Pi_{\mathcal{E}_+}$ est définie comme suit :

$$B(q, \Pi_{\mathcal{E}_+}) = B(q', \Pi_{\mathcal{E}_+}) \text{ si et seulement si } Pr(q) = Pr(q').$$

Le *PTA*(\mathcal{E}_+) est déterministe car il est obtenu à partir du *MCA*(\mathcal{E}_+) en fusionnant les états ayant mêmes préfixes. Un exemple de *PTA* est illustré figure 2.6, c'est l'automate résultant de la fusion des états 1 et 2 du *MCA*(\mathcal{E}_+) de la figure 2.5. Étant donné un échantillon structurellement complet pour des automates, l'espace de recherche est un treillis représentant l'ensemble des automates pouvant être dérivé du MCA. Un exemple de treillis est donné figure 2.7.

Le pseudo code des algorithmes de fusion d'états est le suivant :

Algorithme glouton par fusion d'états :

Entrées : Échantillon d'exemples positifs \mathcal{E}_+ et négatifs \mathcal{E}_-

début

$A = PTA(\mathcal{E}_+)$

tant que *choix_de_2_états*($A, q1, q2$) **faire**

$A' = \text{fusion}(A, q1, q2)$

$A'' = \text{déterminisation}(A')$

si *compatible*(A', \mathcal{E}_-) **alors** $A = A''$

retourner A

On dispose en entrée d'un échantillon d'exemples positifs et négatifs pour lequel on construit le PTA. On fusionne ensuite deux états et on applique une fonction de déterminisation sur l'automate. On vérifie que l'automate résultant est compatible avec l'échantillon, *i.e.* qu'il reconnaît tous les exemples positifs et aucun négatif. On réitère ce procédé jusqu'à ce qu'il n'y ait plus de fusion possible.

Nous allons maintenant présenter les algorithmes d'inférence grammaticale dont nous nous sommes servis pour nos expérimentations, à savoir RPNI et Blue-Fringe. Ce dernier est une variante de l'algorithme EDMSM que nous présenterons également. Il utilise tous la fusion d'états et diffèrent au niveau du choix des états à fusionner.

Algorithme RPNI

L'algorithme RPNI (Regular Positive and Negative Inference) [Oncina et Garcia, 1992, Lang, 1992] effectue une recherche en profondeur dans $Lat(PTA(\mathcal{E}_+))$ et trouve une solution la plus générale possible au problème du plus petit automate fini déterministe compatible avec \mathcal{E}_+ et \mathcal{E}_- . L'ordre utilisé pour la numérotation des états est l'ordre standard sur les plus petits mots associés, c'est-à-dire un ordre suivant la longueur des mots et l'ordre alphabétique pour les mots de même longueur. La partition initiale de l'ensemble du $PTA(\mathcal{E}_+)$ est constituée d'un bloc par état. RPNI procède en au plus $N-1$ étapes où N est le nombre d'états du $PTA(\mathcal{E}_+)$. À l'étape i , les deux premiers blocs, selon l'ordre standard, de l'étape $i-1$ sont fusionnés. Si l'automate quotient de A résultant est non déterministe on applique une fonction de déterminisation qui fusionne tous les états créant le non déterminisme. RPNI teste ensuite la compatibilité avec les exemples négatifs et s'arrête dès que celle-ci n'est plus vérifiée.

Lorsque l'on ne dispose pas d'un échantillon caractéristique (*i.e.* un échantillon S_c tel qu'à partir de tout échantillon contenant S_c on soit capable de trouver une représentation du langage cible), ce qui arrive fréquemment si on travaille avec peu de données, l'automate inféré par RPNI peut être de mauvaise qualité. L'efficacité de l'algorithme glouton RPNI peut être grandement améliorée en introduisant une heuristique sur le tri des états potentiellement fusionnables. Nous introduisons ci-dessous les deux principales, EDMSM et Blue-Fringe. Oliveira *et al.* ont également proposé une procédure de *backtracking* qui permet de remettre en cause des fusions déjà effectuées

[Oliveira et Silva, 1998].

Heuristique EDSM

L’heuristique EDSM, initiée par Rodney Price, a été utilisée dans les deux programmes vainqueurs de la compétition d’inférence grammaticale Abbadingo en 1998 [Lang et al., 1998]. EDSM permet d’atteindre l’automate solution à partir d’échantillons de moins bonne qualité que RPNI (taille plus petite, éloignement de l’échantillon caractéristique). L’idée de cette heuristique consiste à ordonner les fusions à effectuer suivant leur score, attestée par la mise en correspondance d’états finals par la fusion. Le tri des états à fusionner se fait dynamiquement à chaque itération, *i.e.* les scores sont évalués de la façon suivante : à chaque étape les deux états qui, par fusion déterministe, engendrent la fusion d’un maximum d’états finals vont être choisis.

Le but est d’éviter les “mauvaises fusions” en sélectionnant les paires d’états dans le PTA ayant le meilleur score. Malheureusement, la difficulté de détecter les mauvaises fusions augmente quand la densité des données d’apprentissage diminue. En effet, le nombre d’états finals décroît alors dans le PTA. La complexité de l’algorithme est en $O(n^4)$, où n est le nombre d’états du PTA, ce qui limite l’applicabilité de cette heuristique.

Algorithme Blue-Fringe

Pour limiter le nombre de fusions à évaluer, H. Juillé *et al.* ont proposé une approche appelée *Blue-Fringe* utilisant l’heuristique EDSM sur un sous-ensembles d’états [Juillé et Pollack, 1998]. L’algorithme *Blue-Fringe* est un algorithme du type recouvrement qui consiste à maintenir un ensemble d’états déjà testés pour la fusion et non fusionnables entre eux (états rouges) et un ensemble d’états candidats à la fusion (états bleus), ces états étant directement accessibles à partir des états rouges. Tant qu’il reste des états bleus, l’algorithme considère tous les couples d’états bleu/rouge. Si au moins un état bleu est incompatible avec tous les états rouges, le plus petit, dans l’ordre lexicographique, est coloré en rouge. Sinon, les couples d’états conduisant à une fusion compatible sont triés selon le score retenu (selon le score EDSM). La fusion du couple de plus fort score est effectuée. L’algorithme s’arrête lorsqu’il n’y a plus d’états bleus. La méthode Blue-fringe permet d’accélérer le calcul des scores EDSM en restreignant le nombre des fusions. C’est une implémentation par Lang de cet algorithme, appelée *red-blue* [Lang et al., 1998], que nous avons utilisé lors de nos expérimentations.

L’approche par fusions d’états a donné de très bons résultats sur des jeux de données artificielles [Lang et al., 1998]. Les automates obtenus contiennent 506 états pour un échantillon de 60000 séquences et 65 états pour un échantillon contenant 1521 séquences. Ces résultats sont accessibles sur le web¹. Néanmoins, les premières expériences menées sur des séquences protéiques n’ont pas donné de résultats satisfaisants tant au niveau

¹<http://abbadingo.cs.unm.edu/data-sets.html>

de la taille des automates obtenus qu'au niveau du temps d'exécution des automates [Henry, 2002]. Ceci peut s'expliquer de différentes façons.

Régions conservées. Les séquences protéiques sont de longues chaînes orientées commençant (resp. finissant) par une région appelée N-terminale (resp. C-terminale). Certaines régions des protéines jouent un rôle important dans le maintien de la structure 3D ou de liaisons avec d'autres molécules. Ces régions conservées se situent rarement au début ou à la fin des protéines. Or, comme nous venons de le voir, RPNI et Blue-Fringe fusionnent prioritairement les états proches de l'état initial. Si l'échantillon fourni à RPNI représente des séquences protéiques, cela revient à fusionner en priorité les parties N-terminales, *i.e.* les régions non conservées. La fusion de ces régions conduit à la création d'une boucle pouvant contenir le début d'une zone conservée. Le modèle obtenu contient donc au mieux uniquement une sous-partie des régions conservées des protéines. Le problème est le même pour l'heuristique EDSM qui tente de fusionner le plus d'états finals. Ce qui revient ici à fusionner les parties C-terminales des protéines.

Ces observations montrent qu'il serait judicieux d'effectuer prioritairement les fusions des régions conservées dans les protéines. Il existe des travaux en cours de développement sur ce sujet [Idmont, 2003, Coste *et al.*, 2004].

Forme des motifs. Une famille de protéines est souvent identifiée par une succession de motifs caractéristiques espacés par un nombre variable d'acides aminés. On peut simplifier l'expression décrivant une famille de protéines en remplaçant les intervalles de longueur variable séparant les motifs par l'ensemble des mots de Σ^* . Ceci revient à apprendre le langage $(\Sigma^*w\Sigma^*)^*$ où w représente le motif caractéristique de la famille de protéines. Ce langage est facilement représentable par un automate non déterministe. La plupart des algorithmes de fusions d'états sont adaptés aux automates déterministes. Or, il apparaît difficile de modéliser le langage représentant une famille de protéines par un automate déterministe et les heuristiques adaptées aux automates non déterministes commencent juste à apparaître [Coste et Kerbellec, 2005].

Exemples négatifs. La dernière raison que nous citerons est la difficulté d'obtenir des exemples négatifs intéressants. La plupart des algorithmes d'inférence grammaticale utilisent des contre-exemples afin d'améliorer la qualité des méthodes d'inférence [Gold, 1978], ce qui nécessite la connaissance *a priori* de ce que le langage ne contient pas. La notion de famille de protéines est complexe car certaines protéines peuvent se révéler multi-fonctionnelles. Il est donc difficile de sélectionner, pour une famille des séquences très proches, des protéines de cette famille qui ne lui appartiennent pas de façon certaine.

Dans cette thèse, nous visons une expressivité plus grande que celle offerte par les langages réguliers, c'est pourquoi nous utilisons les langages algébriques. Le problème de l'inférence grammaticale a surtout été étudié pour les grammaires régulières. En ce qui concerne les grammaires algébriques, l'apprentissage est plus difficile. Pourtant, le passage aux grammaires algébriques est important du point de vue des applications car il permet de considérer des structures auto-imbriquées, ce dont nous avons besoin

pour représenter nos ponts disulfures. Le paragraphe suivant présente quelques travaux réalisés sur les grammaires algébriques et plus particulièrement ceux de Y. Takada représentant la base de nos expérimentations.

2.2.5 Apprentissage de langages algébriques

Les premières méthodes d'inférence de grammaires algébriques sont assez rudimentaires. En général, seul un échantillon positif est fourni à l'algorithme. Ces méthodes n'aboutissent à un résultat que si l'échantillon est particulièrement bien choisi. Diverses tentatives d'inférence de grammaires algébriques peuvent être trouvées dans [Feldman et Reder, 1969], [Crespi-Reghizzi, 1972], [Fu et Booth, 1975], [Gonzalez et Thomason, 1978] mais d'une manière générale, ces méthodes ne permettent pas de concevoir un algorithme efficace, essentiellement dû à l'absence d'un cadre théorique défini.

Par la suite, des résultats positifs ont été obtenus pour l'inférence de (sous-classes) de grammaires algébriques en réduisant le problème à un problème d'inférence régulière [Takada, 1988, Takada, 1995] ou en utilisant des méthodes d'inférence régulière [Sakakibara, 1990]. Avant de rentrer dans le détail de ces méthodes, définissons tout d'abord, la notion d'arbre de dérivation.

Définition 2.14 (arbre de dérivation) *Un arbre de dérivation pour une grammaire $G=(\Sigma, N, P, S)$ est un arbre ordonné tel que :*

- la racine de l'arbre est étiquetée par S ;
- chaque noeud est étiqueté par un symbole de $\Sigma \cup N$;
- s'il existe un noeud A ayant comme descendants X_1, \dots, X_n alors $A \rightarrow X_1 \dots X_n$ est une production de G .

Y. Sakakibara effectue un apprentissage à partir de données structurées qui consiste à fournir au système la structure des arbres de dérivation des mots présentés (sous forme de parenthésage ou d'arbres non étiquetés par exemple) [Sakakibara, 1990]. Il propose un algorithme qui relie l'apprentissage de grammaires algébriques et d'automates d'arbres : l'idée est de produire un automate d'arbres correspondant aux données structurées présentées, puis de construire la grammaire algébrique dont les arbres de dérivation sont reconnus par cet automate à partir de la fonction de transition. La grammaire produite par l'algorithme de Sakakibara engendre les mots présentés de la façon décrite par la structure qui leur est associée. L'inconvénient de cet algorithme est qu'il nécessite des conditions d'application permettant de connaître la structure des instances.

Dans le même ordre d'idée, Y. Takada établit l'existence d'une hiérarchie des familles de langages, dans lesquelles le problème d'apprentissage pour chaque famille est réduit au problème d'apprentissage pour les langages réguliers [Takada, 1995]. Cette hiérarchie est définie de manière inductive en contrôlant un certain type de grammaires, les *gram-*

maires linéaires équilibrées. Les grammaires linéaires équilibrées ont été proposées par Amar et Putzolu [Amar et Putzolu, 1964]. Elles se définissent de la manière suivante :

Définition 2.15 (grammaire linéaire équilibrée) Une grammaire linéaire équilibrée est un quadruplet $G=(N, \Sigma, P, S)$ où N est un ensemble fini non vide de non-terminaux, P est un ensemble fini non vide de productions. Chaque production dans P est de la forme $A \rightarrow uBv$ ou $A \rightarrow w$, où $A, B \in N$, $u, v, w \in \Sigma^*$, et $|u|=|v|$. S est l'axiome, N, Σ et P sont disjoints.

Cette famille contient la famille des langages réguliers et est contenue dans la famille des langages linéaires. Radhakrishnan et Nagaraja ont présenté un algorithme d'apprentissage pour les grammaires linéaires équilibrées et son application aux langages de description d'image [Radhakrishnan et Nagaraja, 1988]. Le fait que le problème d'apprentissage des langages linéaires équilibrés soit réduit au problème d'apprentissage des langages réguliers, rend les algorithmes d'inférence régulière applicables à l'apprentissage de langages linéaires équilibrés [Takada, 1988], [Takada, 1994b], [Takada, 1994a].

La notion de langage de contrôle utilise des règles de production étiquetées que nous définissons tout d'abord :

Définition 2.16 (grammaire linéaire étiquetée) Une grammaire linéaire étiquetée est un 5-uple $G = (N, \Sigma, P, S, \Pi)$ où $G = (N, \Sigma, P, S)$ est une grammaire linéaire et Π un ensemble de symboles d'étiquettes pour les productions. Une production étiquetée s'écrit de la manière suivante : $\pi : A \rightarrow \alpha$.

On note $x \xRightarrow{\pi} y$ si y dérive de x en utilisant la production π . Le langage de contrôle associé à cette grammaire est l'ensemble des enchaînements canoniques des règles de production utilisées pour analyser les instances.

Définition 2.17 (ensemble de contrôle) Un ensemble de contrôle sur G est un sous-ensemble C de Π^* . Le langage généré par G avec l'ensemble de contrôle C se définit comme suit :

$$L(G, C) = \{w \in \Sigma^* \mid S \xRightarrow[G]{\sigma} w, \sigma \in C\}.$$

Le langage de contrôle, appelé aussi langage de Szilard, de la grammaire G est défini par : $Sz(G) = \{\sigma \in C^* \mid S \xRightarrow{\sigma} w, w \in \Sigma^*\}$.

Si \mathcal{L} est une classe de langages sur un alphabet fixé Σ , alors une classe de représentations pour \mathcal{L} est un langage \mathcal{R} (sur un alphabet Γ) tel que :

- \mathcal{R} est récursif;
- à chaque $r \in \mathcal{R}$ correspond un langage $\Delta(r)$ dans \mathcal{L} ;
- il existe un algorithme qui pour un élément e de Σ^* et une représentation $r \in \mathcal{R}$, renvoie vrai si $e \in \Delta(r)$ et faux sinon.

Soit Γ la classe de toutes les grammaires linéaires équilibrées et \mathcal{R}_r la famille des langages réguliers. Takada définit L_i , $i \geq 0$ de la manière suivante :

- $L_0 = \mathcal{R}_r$

$$- L_i = \{L(U, C) \mid U \in \Gamma \text{ et } C \in L_{i-1}\}, \forall i \geq 1.$$

Il montre de façon générale que $\forall i \geq 1, L_{i-1} \subseteq L_i$ et que $\forall i \geq 1, L_i$ est contenu dans une famille de langages sensibles au contexte. D'où la hiérarchie :

$$(\mathcal{R}_r = L_0) \subset L_1 \subset L_2 \subset \dots \subset CS \quad (2.1)$$

où CS représente la classe des langages dépendants du contexte (Context-Sensitive).

Takada démontre ensuite que pour chaque langage $\mathcal{L} \in L_i$, il existe un ensemble de contrôle canonique unique et régulier. Ce qui signifie qu'identifier un langage $\mathcal{L} \in L_i$, revient pour un algorithme d'apprentissage à identifier son ensemble de contrôle canonique régulier pour \mathcal{L} .

Définition 2.18 (ensemble de contrôle canonique) *Pour tout $i \geq 1$ et tout alphabet Σ , soit \mathcal{L} un langage sur Σ dans la famille L_i . Un langage C est dit ensemble de contrôle canonique pour \mathcal{L} ssi :*

- $\mathcal{L} = L(U_1, L(U_2, \dots, L(U_i, C) \dots))$ pour des grammaires linéaires équilibrées universelles U_1, U_2, \dots, U_i
 - pour tout $\alpha \in C$, $A(U_1, A(U_2, \dots, A(U_i, \alpha) \dots)) \neq \infty$,
- où $A(U_i, \alpha) = \begin{cases} w & \text{si } S \xrightarrow{\alpha} U_i w \text{ et } w \in \Sigma^* \\ \infty & \text{sinon} \end{cases}$

Dans un contexte pratique, l'utilisation des travaux de Takada n'a jamais été tentée. La classe des grammaires linéaires équilibrées n'est en effet pas très naturelle et le gain en expressivité est acquis au prix d'un codage extrêmement lourd des langages. Par contre, le déplacement de l'apprentissage sur le langage de contrôle nous semble une idée intéressante à conserver, qui permet de bien distinguer la partie modélisation de la partie apprentissage en maîtrisant la complexité de l'apprentissage.

2.2.6 Applications de l'inférence grammaticale sur des séquences biologiques

Les principaux travaux effectués sur les problèmes d'analyses de séquences biologiques utilisent des grammaires stochastiques [Durbin *et al.*, 1998, Krogh *et al.*, 1994, Sakakibara *et al.*, 1994, Eddy, 1998, Searls et Murphy, 1995, Baldi *et al.*, 1994]. Une *grammaire stochastique* est obtenue en spécifiant une probabilité sur chaque production de la grammaire. Elle donne une probabilité à chaque chaîne dérivée et définit une distribution de probabilité sur l'ensembles des séquences. Les *automates stochastiques* sont l'équivalence probabiliste des automates finis plus connus sous le nom de *modèles de Markov cachés* (HMM). Les *grammaires algébriques stochastiques* (SCFG) sont une super-classe et se situent un palier au dessus des HMM dans la hiérarchie de Chomsky.

Un HMM est un modèle stochastique apparenté aux automates probabilistes. Un tel automate est une structure composée d'états et de transitions entre états possédant une probabilité (*probabilité de transition*). A chaque transition est associée un symbole

d'un alphabet fini, généré à chaque fois que la transition est empruntée. Contrairement aux automates probabilistes, un HMM génère un symbole au niveau des états et non des transitions ; à chaque état est associé une *probabilité d'émission* d'un symbole.

Définition 2.19 (modèles de Markov cachés : HMM) *Un modèle de Markov caché est un quintuplet $A = (Q, \Sigma, \Delta, \pi, O)$, où Q est un ensemble fini d'états, Σ est un alphabet de symboles de sortie, Δ est une distribution de probabilité de transition, O une distribution de probabilité d'émission, et π une distribution d'état initial.*

Pour un HMM A , trois problèmes basiques sont à résoudre : étant donnée une séquence $w = a_1 \cdots a_m$

- calculer $Pr(w|A)$, la probabilité de la séquence w générée par A
- trouver le chemin d'états le plus probable $s = q_{i_1} \cdots q_{i_l}$ qui maximise $Pr(s|w, A)$
- évaluer les paramètres dans A qui maximise $Pr(w|A)$.

Ces problèmes peuvent être résolus efficacement en utilisant des techniques de programmation dynamique [Rabiner, 1990].

Les limitations des HMM sont qu'ils ne peuvent modéliser que des structures primaires de séquences biologiques et ne peuvent pas explicitement modéliser des structures d'ordre supérieur comme les structures secondaires et tertiaires des séquences biologiques car ils ne prennent pas en compte les interactions longues distances. Ainsi pour la prédiction des ponts disulfures dans les protéines la modélisation devient difficile car une cystéine interagit avec une autre cystéine mais ce principe d'interaction ne peut pas être mis en évidence au moyen de HMM. Pour cela, il est nécessaire d'avoir recours à des systèmes grammaticaux plus puissants dans la hiérarchie de Chomsky des grammaires formelles.

Les grammaires algébriques ont été présentées en section 2.2.2 et permettent de définir les grammaires algébriques stochastiques.

Définition 2.20 (grammaires algébriques stochastiques : SCFG) *Une grammaire algébrique stochastique est une grammaire algébrique dont les règles de production sont indexées par une probabilité.*

La probabilité associée à chaque production de la forme $A \rightarrow \alpha$ est notée $Pr(A \rightarrow \alpha)$, et il existe une distribution de probabilité sur l'ensemble des productions ayant le même non terminal dans la partie gauche des règles. Les trois problèmes basiques à résoudre pour une SCFG sont les mêmes que pour les HMM et peuvent être résolus efficacement. Les deux premiers problèmes, calculer la probabilité $Pr(w|G)$ d'une séquence donnée w et trouver l'arbre de dérivation le plus probable de w par G peuvent être résolus en utilisant les méthodes de programmation dynamique analogues aux méthodes Cocke-Younger-Kasami [Aho et Ullman, 1972]

Sakakibara *et al.* propose une méthode efficace pour l'apprentissage et la construction de grammaires algébriques stochastiques pour modéliser une famille de séquences d'ARN [Sakakibara *et al.*, 1994]. Dans l'ARN, les nucléotides A, C, G, et U interagissent

Pseudo-noeuds

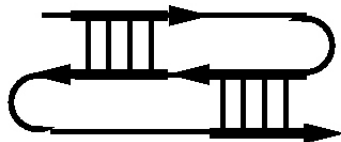


FIG. 2.8 – Pseudo-noeuds dans une séquence d'ARN

de manière spécifique pour former des motifs de structure secondaire caractéristiques. En général, le repliement d'une chaîne ARN en molécule fonctionnelle est gouverné par la formation de paires Watson-Crick intra-moléculaires A-U et G-C. De telles paires de base constituent ce que l'on appelle les palindromes biologiques dans le génome. Y. Sakakibara *et al.* sont partis du fait que les paires de base dans l'ARN peuvent être modélisées par une grammaire algébrique dont les règles de production sont de la forme $X \rightarrow AYU$, $X \rightarrow UYA$, $X \rightarrow GYC$, $X \rightarrow CYG$. Les productions de la forme $X \rightarrow aYb$ décrivent des paires de base, où X et Y sont des non terminaux et a un symbole terminal représentant un nucléotide. Les productions de la forme $X \rightarrow aY$ et $X \rightarrow a$ décrivent des bases non appariées. Cette idée nous semble intéressante pour identifier les ponts disulfures dans les protéines. En effet, considérer une règle de production de la forme $X \rightarrow cYc$ où c représente une cystéine permet de mettre en évidence l'appariement de deux cystéines distantes dans une séquence. De plus, les productions de la forme $X \rightarrow xY$ et $X \rightarrow x$ décrivent les différents acides aminés des protéines. Afin de reconnaître toutes les protéines nous utiliserons ces règles de production dans une grammaire universelle.

Des travaux ont été effectués sur les pseudo-noeuds dans des séquences ARN [Rivas et Eddy, 2000, Cai *et al.*, 2003, Matsui *et al.*, 2005]. Ces structures, illustrées figure 2.8, font intervenir des interactions longues distances entre paires de base, appartenant des motifs croisés dans la séquence. Les travaux effectués dans ce domaine sont intéressants car ils s'apparentent à notre problème de prédiction de ponts disulfures dans les protéines. En effet l'appariement croisé des motifs dans les pseudo-noeuds peut être apparenté à l'appariement croisé des contextes englobants les cystéines oxydées dans les protéines. Afin de modéliser ces structures, il est nécessaire d'utiliser le pouvoir d'expression des grammaires contextuelles plus complexes que les grammaires algébriques. Afin de pallier à cette complexité, Brown et Wilson ont essayé de modéliser un croisement en utilisant une intersection de deux grammaires algébriques mais cette approche n'a pas été implémentée [Brown et Wilson, 1996]. Uemura *et al.* ont tenté une approche en utilisant les TAG (Tree Adjoining Grammars) qui semble trop compliquée pour être implémentée [Uemura *et al.*, 1999]. Les résultats ne sont pas encore satisfaisants mais les idées utilisant des grammaires algébriques pour détour-

ner la complexité des grammaires contextuelles semblent intéressantes [Cai *et al.*, 2003].

Récemment, des travaux ont été effectués sur des séquences protéiques par F. Coste *et al.* [Coste et Kerbellec, 2005, Coste *et al.*, 2004] qui proposent une nouvelle approche permettant d'apprendre des automates non déterministes pour la caractérisation et la modélisation de séquences protéiques. Cette approche est basée sur la fusion de fragments significativement similaires pour la caractérisation de la famille. Dans leur article, ils présentent trois heuristiques d'ordonnement des paires de fragments, dont une utilisant la présence de contre-exemples, et un processus de généralisation, basé sur l'identification des propriétés physico-chimiques des acides aminés. Comme nous l'avons vu précédemment, les heuristiques existantes fusionnent prioritairement les parties N et C-terminale des protéines, ce qui pose problème pour des séquences protéiques. Effectuer ces fusions de fragments similaires revient à fusionner prioritairement les régions conservées des protéines. Les premières expérimentations menées sur la caractérisation de protéines de la famille MIP montrent la pertinence des automates appris, attestée par un bon pouvoir de prédiction. Cette idée est intéressante car elle permet de travailler sur des automates non déterministes. Cette approche permet d'apprendre le langage $(\Sigma^*c\Sigma^*c\Sigma^*)^*$ où c représente une cystéine et Σ l'alphabet des acides aminés. Ce langage caractérise un pont disulfure dans une protéine et est facilement interprétable par un automate non déterministe.

Dans sa thèse, A. Leroux propose une nouvelle méthode heuristique d'inférence grammaticale (SDTM) fondée sur les notions fondamentales de découverte de motifs ainsi que sur la ressemblance des acides aminés [Leroux, 2005]. Pour cela, il a introduit un treillis de ressemblance fondé sur le diagramme de Taylor et qui constitue un ordre partiel sur les groupes d'acides aminés. Le modèle produit par cette méthode appartient à la classe des langages réguliers et consiste en un automate non déterministe dont les étiquettes des transitions correspondent aux acides aminés. Le principe est la fusion successive de paires de transitions correspondant aux acides aminés appariés. Dans notre cadre applicatif, il serait intéressant d'utiliser ce treillis de ressemblance car la structure des protéines n'est pas gouvernée par l'identité des acides aminés de la séquence mais plutôt par le volume qu'ils occupent et par les liens qu'ils peuvent établir avec les autres acides aminés. Ainsi, certains acides aminés peuvent être remplacés sans modifier gravement la structure de la protéine et sa fonction. On pourrait ainsi mettre en évidence des ressemblances entre les acides aminés autour des cystéines oxydés.

Nous avons également utilisé une seconde méthode d'apprentissage de structures à savoir la programmation logique inductive, qui ne traite pas particulièrement de séquences mais qui explicite toutes les relations existant sur un ensemble de données, ce qui est un avantage majeur pour la prédiction de contextes disulfures en relation. Cette méthode fait l'objet de la section suivante.

2.3 La Programmation Logique Inductive (PLI)

La programmation logique inductive est à l'intersection de l'apprentissage automatique et de la programmation logique. Le langage de description utilisé, pour les exemples comme pour les concepts inférés, est un sous ensemble de la logique du premier ordre. La PLI se fonde sur des principes de la programmation logique (subsomption, résolution) pour induire des concepts à partir d'exemples et d'une théorie logique du domaine étudié. La PLI a deux caractéristiques fortes : tout d'abord le langage de description des hypothèses a de fortes propriétés mathématiques. Ensuite, la combinatoire de l'apprentissage doit être spécifiée par l'utilisateur : il s'agit d'explorer un espace initialement immense en le restreignant par des choix syntaxiques et sémantiques. Il est donc fondamental en PLI de bien décrire les biais d'apprentissage permettant de limiter cet espace de recherche.

Nous donnons dans cette section quelques fondements et notions de la programmation logique inductive. Une bonne présentation du domaine se trouve dans [Lavrac et Dzeroski, 1994] et [Claveau, 2003] dont nous reprenons quelques éléments ici. La PLI nous permet d'inférer des règles définissant en intention le concept que l'on cherche à acquérir et qui est décrit en extension (au moins partiellement) à l'aide d'exemples. Les règles, des clauses de Horn, sont ensuite utilisées pour découvrir tous les éléments répondant au concept décrit. Nous terminerons cette section en présentant divers travaux du domaine ayant été appliqués aux séquences biologiques.

2.3.1 Principes et approche de la PLI

Cette section débute par quelques rappels de logique et de programmation logique, qui vont nous permettre de préciser les notations utilisées dans ce document. Les principes de l'induction logique sont décrits ensuite.

En programmation logique inductive, le langage de description des exemples et des hypothèses se fait au moyen de la logique des prédicats, appelée aussi logique du premier ordre. Cette logique est un langage formel utilisé pour représenter des relations entre objets et pour déduire de nouvelles relations à partir de relations connues comme vraies. Elle peut être vue comme un formalisme utilisé pour traduire des phrases et déduire de nouvelles phrases à partir de phrases connues. Ces notions sont détaillées dans le paragraphe suivant.

2.3.1.1 Logique des prédicats

Le langage de logique du premier ordre est l'ensemble de toutes les formules construites à partir des symboles de l'alphabet et admises par la grammaire du langage. Le calcul des prédicats s'appuie sur six classes de symboles :

- les variables ;
- les symboles de constantes ;
- les symboles de fonctions ;

- les symboles de prédicats ;
- les connecteurs \neg (non), \wedge (et), \vee (ou), \rightarrow (implique), \leftrightarrow (équivalent) ;
- les quantificateurs \forall et \exists .

Rappelons quelques définitions usuelles et notations.

Définition 2.21 (Terme) *Un terme est soit :*

- une constante ;
- soit une variable ;
- soit un terme construit $f(t_1, \dots, t_n)$ où f est un symbole de fonction et t_i des termes.

Définition 2.22 (Prédicat) *Un prédicat est une fonction dans $\{\text{vrai}, \text{faux}\}$. Il peut être représenté par un symbole muni d'une arité (par exemple $\text{pattern}/2$).*

Définition 2.23 (Atome, littéral) *Un atome est un prédicat appliqué à des termes. Par exemple $p(t_1, t_2, \dots, t_n)$ est un atome où p est un symbole de prédicat d'arité n et t_i des termes. Un littéral est un atome éventuellement précédé du symbole de négation \neg .*

Définition 2.24 (Formule) *Une formule se définit récursivement comme suit :*

- un atome est une formule
- si f et g sont des formules alors $\neg f$, $\neg g$, $f \wedge g$, $f \vee g$, $f \rightarrow g$ et $f \leftrightarrow g$ sont des formules
- si f est une formule et x une variable alors $(\forall x f)$ et $(\exists x f)$ sont des formules

Nb : Une *sous-formule* est une formule contenue dans une formule.

Le langage des clauses, et plus particulièrement celui des clauses de Horn et des clauses définies (voir ci-dessous), permet de ne manipuler que des formules ayant une certaine forme. Les programmes logiques sont composés d'une conjonction de clauses définies.

Définition 2.25 (Clause) *Une clause est une formule composée d'une disjonction finie de littéraux dans laquelle toutes les variables sont quantifiées universellement.*

Par souci de lisibilité, on simplifie généralement l'écriture de ces clauses en omettant de faire apparaître les quantificateurs universels. Ainsi, $\forall X \forall Y (f(X) \vee g(Y))$ s'écrira simplement $f(X) \vee g(Y)$.

Définition 2.26 (Clause de Horn) *Une clause de Horn est une clause contenant au plus un littéral positif.*

Les exemples suivants représentent des clauses de Horn : $\neg A$, $\neg A \vee \neg B$, $\neg A \vee \neg B \vee C$. Par contre $A \vee B$, $\neg A \vee \neg B \vee C \vee D$ n'en sont pas. Soit C une clause de Horn telle que $C = l_t \vee \neg l_1 \vee \dots \vee \neg l_n$. Cette clause peut s'écrire $l_t \wedge \dots \wedge l_n \rightarrow l_t$, ou encore en inversant le sens de l'implication $l_t \leftarrow l_1 \wedge \dots \wedge l_n$. Finalement en remplaçant la conjonction par une virgule et l'implication par le symbole $:-$ on obtient la syntaxe utilisée en Prolog Edimbourgh : $l_t :- \neg l_1, \dots, l_n$.

Définition 2.27 (Clause définie) Une clause définie est une clause de Horn contenant exactement un littéral positif.

Un programme logique est constitué d'un ensemble de clauses définies de la forme $(H : -A_1, \dots, A_n)$, où H et A_1, \dots, A_n sont des atomes. On appelle H la tête de la clause et A_1, \dots, A_n son corps. Un *fait* est une clause de Horn dont le corps est vide.

Différents types de raisonnement sont possibles à partir d'une représentation logique. On distingue essentiellement la déduction, l'abduction et l'induction. Considérons l'exemple suivant où chacune des propositions est accompagnée de sa représentation Prolog standard :

- (a) Les acides aminés sont des molécules.
molecule(X) :- acide_amine(X).
- (b) La cystéine est un acide aminé.
acide_amine(cysteine).
- (c) La cystéine est une molécule.
molecule(cysteine).

Les différents types d'inférence peuvent être décrits comme suit :

1. La déduction est le mécanisme qui, connaissant (a) et (b) permet de déduire (c) en appliquant des règles conservant la validité des formules comme le *modus ponens*. Ce type d'inférence est utilisé dans les démonstrateurs automatiques.
2. L'abduction est le phénomène qui, connaissant (a) et (c) permet de supposer (b). Étant donnée une observation et connaissant les lois du système il faut retrouver l'hypothèse manquante expliquant l'observation. Ce type d'inférence est utilisé dans le diagnostic automatique.
3. L'induction est le phénomène qui, connaissant les faits (b) et (c) conduit à inférer l'hypothèse (a).

La programmation logique utilise le premier type et la PLI le troisième type de raisonnement. Les exemples sont représentés par des faits Prolog de type (b) et les classes qui leur sont assignées par les faits Prolog de type (c). Les règles inférées sont des clauses de Horn.

2.3.1.2 Déduction en programmation logique

La signification associée aux constantes, symboles de fonctions et symboles de prédicats, se donne au moyen d'une *interprétation*, qui associe un sens à chaque élément d'une formule. Un *modèle* de formule est une interprétation pour laquelle la formule est vraie. La validité d'une formule F par rapport à un ensemble de formules S s'exprime à l'aide de la *conséquence logique* (conséquence sémantique) : F est une conséquence logique de S, et on note $S \models F$, si tout modèle de S est un modèle de F. Le mécanisme de preuve est la *dérivation* (conséquence syntaxique) : F est conséquence syntaxique de

S , noté $S \vdash F$, si et seulement si F peut être dérivé de S en utilisant un ensemble de règles d'inférence déductives. Les systèmes de programmation logique emploient généralement une seule règle d'inférence : la *résolution*. Celle-ci est basée sur les opérations de substitution et d'unification que nous définissons ci-après.

Définition 2.28 (Substitution) Une substitution θ est une application de l'ensemble des variables dans l'ensemble des termes, souvent notée sous forme de liste comme suit : $\theta = [X_1/t_1, \dots, X_n/t_n]$. Les variables X_i sont remplacées par les termes t_i . Par exemple, l'application de la substitution $\theta = [X/2, Y/3]$ à la formule $f(X, Y, Z)$ donne la formule $f(2, 3, Z)$.

Une substitution θ est un *unificateur* de deux littéraux l_1 et l_2 si $l_1\theta = l_2\theta$. Cette substitution est le *plus grand unificateur (pgu)* si de plus, pour tout unificateur σ de l_1 et l_2 , il existe une substitution γ telle que $\sigma = \theta\gamma$. Par exemple, $l_1 = f(X, a, Y)$ et $l_2 = f(b, Z, W)$ sont unifiables en $l_1\sigma = l_2\sigma = f(b, a, Y)$ avec $\sigma = [X/b, Z/a, Y/c, T/c]$ ou avec la combinaison du pgu $\theta = [Z/a, X/b, Y/T]$ et de la substitution $\gamma = [T/c]$.

Définition 2.29 (Résolvante) Soient deux clauses C_1 et C_2 n'ayant pas de variable en commun. Une clause C est une *résolvante* de C_1 et C_2 si et seulement si les deux conditions suivantes sont vérifiées :

- $\exists l_1 \in C_1$ et $\exists l_2 \in C_2$ tels que $\exists \theta$ pgu de l_1 et l_2 , i.e. $l_1\theta = (l_2)\theta$;
- $C = (C_1 \setminus \{l_1\})\theta_1 \cup (C_2 \setminus \{l_2\})\theta_2$ avec $\theta = \theta_1\theta_2$.

Plus concrètement, dans le cas de clauses définies, le littéral l_1 est un des littéraux du corps de C_1 , et la clause C_2 , dont l_1 est la tête, représente une définition de la propriété notée par le prédicat de l_1 (et de l_2). La résolvante est donc la clause C_1 dans laquelle on remplace un littéral par sa définition, i.e. par le corps de C_2 . Ainsi, cette méthode de construction permet d'inférer la résolvante de deux clauses, soit encore, avec les mêmes notations que précédemment $C_1, C_2 \vdash C$.

Pour Robinson [Robinson, 1965] la résolution est une forme d'inférence reposant sur une seule règle et capable de prouver si des phrases logiques sont vraies ou fausses. Cette résolution est celle appliquée dans Prolog à des clauses définies D . Pour choisir la clause aidant à résoudre le but, une fonction de sélection S est utilisée : elle consiste à choisir la première clause dont la tête correspond au but (à une substitution près). Enfin, la clause résultant de l'étape précédente de résolution est utilisée à l'étape suivante. Cette technique de résolution linéaire L porte le nom de SLD-résolution.

2.3.1.3 Induction en programmation logique

L'induction est souvent présentée comme le processus inverse de la déduction. Son principe est effectivement de "remonter" des faits aux lois qui les régissent.

Le but de la programmation logique inductive est de trouver un ensemble de formules H tel que, étant donné un ensemble de connaissances B (théorie du domaine), un ensemble d'observations \mathcal{E} (exemples positifs \mathcal{E}_+ et négatifs \mathcal{E}_-), H couvre tous les exemples positifs et aucun négatif en vérifiant les conditions suivantes (\square est la clause vide) :

- $B \cup \mathcal{E}_- \not\models \square$ (consistance *a priori*);
- $B \not\models \mathcal{E}_+$ (nécessité *a priori*);
- $B \cup H \models \mathcal{E}_+$ (complétude);
- $B \cup H \cup \mathcal{E}_- \not\models \square$ (consistance *a posteriori*).

La condition de complétude exige que chaque exemple puisse être prouvé à partir de la connaissance et des hypothèses induites. La condition de consistance *a posteriori* précise qu'aucun contre-exemple ne doit pouvoir en être dérivé. Ces deux conditions constituent un invariant du problème de la programmation logique inductive.

Les quatre conditions précédentes se traduisent en sémantique définie (clauses définies), sous la forme suivante [Muggleton et Raedt, 1994] :

- $\forall e_- \in \mathcal{E}_-, e_-$ est faux dans $\mathcal{M}(B)$;
- $\exists e_+ \in \mathcal{E}_+, e_+$ est faux dans $\mathcal{M}(B)$;
- $\forall e_- \in \mathcal{E}_-, e_-$ est faux dans $\mathcal{M}(B \wedge H)$;
- $\forall e_+ \in \mathcal{E}_+, e_+$ est vrai dans $\mathcal{M}(B \wedge H)$.

où $\mathcal{M}(\Phi)$ est le plus petit modèle de Herbrand associé au programme défini Φ dans lequel toute formule logique est soit vraie soit fausse et où les exemples négatifs sont maintenant représentés par des faits faux.

L'ensemble des clauses définies qu'il est possible d'exprimer dans le langage des hypothèses constitue un espace de recherche dans lequel il s'agit de trouver les formules qui composeront un programme satisfaisant les conditions d'apprentissage. Comme nous l'avons vu au début de ce chapitre, cet espace est structuré à l'aide d'une relation d'ordre partiel sur les formules. En programmation logique inductive, la mise en correspondance des hypothèses et des exemples peut s'effectuer de manière générale au moyen de la dérivation logique (\vdash). On peut définir la relation de généralité suivante, B étant la théorie du domaine : $G \succeq S \Leftrightarrow B, G \vdash S$. Cependant la mise en oeuvre de cette définition pose quelques problèmes. L'implication logique entre théories ou même entre clauses n'est pas décidable [Schmidt-Schauss, 1988, Nienhuys-Cheng et Wolf, 1996, Marcinkowski et Pacholski, 1992]. Certaines relations de généralité se contentent donc de comparer deux clauses, indépendamment de la théorie du domaine. Un ordre largement utilisé dans les systèmes de PLI est la théta-subsumption définie par Plotkin en 1970 [Plotkin, 1970] :

Définition 2.30 (θ -subsumption) Une clause C_1 θ -subsume une clause C_2 , notée $C_1 \succeq_\theta C_2$ si et seulement si il existe une substitution θ telle que $C_1\theta \subseteq C_2$

La théta-subsumption est plus faible que l'implication mais décidable [Robinson, 1965], même si elle reste NP-difficile [Kapur et Narendran, 1986]. Il existe d'autres notions de généralité. En 1997, Rouveirol et Sebag ont introduit une subsumption stochastique afin de rendre la complexité de la comparaison de deux hypothèses polynomiales [Sebag et Rouveirol, 1997]. Esposito *et al.* [Esposito *et al.*, 1996] et Badea *et al.* [Badea et Stanciu, 1999] ont introduit la subsumption sous identité objet afin de contraindre les substitutions possibles.

En pratique, l'espace de recherche en PLI est souvent très grand, voire infini. C'est pourquoi il est nécessaire d'imposer des contraintes sur l'espace des hypothèses pour en réduire la taille et ainsi rendre possible son exploration. Ceci se fait au travers de *biais*. Les biais peuvent être intégrés dans le système, l'utilisateur peut alors ajuster un ensemble de variables globales. Ils peuvent également être définis de façon déclarative par l'utilisateur, on parle alors de *biais déclaratifs*. L'autre effet majeur de ces biais est de permettre d'améliorer la qualité des hypothèses en ajoutant au système de la connaissance sur le concept recherché. On distingue trois grands types de biais : les biais syntaxiques, les biais sémantiques et les biais de recherche [Muggleton et Raedt, 1994].

Biais syntaxiques : ils agissent directement sur la syntaxe des clauses produites.

Ainsi, il est possible de limiter le nombre de littéraux des clauses induites et le nombre de variables apparaissant dans les clauses. On peut également agir sur l'ordre d'apparition des variables ou sur la manière dont elles sont reliées les unes aux autres à travers les littéraux.

Biais sémantiques : ils introduisent un typage des prédicats et variables dans les programmes induits. Différents *types* et *modes* sont ainsi définis. La déclaration des types permet d'attribuer un type définitif à une variable. Ainsi, si une variable possède un type t , les littéraux ne peuvent l'utiliser que s'ils peuvent fonctionner avec des variables de type t . La déclaration des modes permet de restreindre la taille de l'espace de recherche en agissant sur les variables d'entrée et de sortie. Par exemple, une variable en entrée dans une tête de clause doit se retrouver dans le corps de clause. On peut également restreindre certaines variables en sortie dans le corps d'une clause à servir de résultats intermédiaires pour être en entrée dans un autre prédicat, ou à apparaître en sortie dans la tête de la clause.

Le système que nous avons utilisé, PROGOL [Muggleton, 1995], permet de déclarer différents modes sur les variables des littéraux grâce aux déclaratives *modeh* et *modeb*. Le premier spécifie les prédicats pouvant apparaître en tête de clause et le second, ceux pouvant apparaître dans le corps de clause. Par exemple, la déclaration suivante :

$$\text{modeh}(1, \text{somme}(+\text{entier}, +\text{entier}, -\text{entier}))$$

signifie que la tête des clauses de l'espace de recherche sera constituée du prédicat *somme/3* et possédera 3 arguments de type entier dont les 2 premiers sont en mode entrée et le dernier en mode sortie.

On peut citer aussi un autre type de biais, les biais de recherche :

Biais de recherche : ils sont utilisés pour influencer la tâche d'apprentissage. Ce sont, par exemple, les heuristiques de recherche qui vont diriger le parcours de l'espace des hypothèses et ainsi favoriser certaines hypothèses par rapport à d'autres.

PROGOL nous permet de bénéficier de biais de recherche et de toutes les possibilités de biais déclaratifs tels que les déclarations de types ajoutés aux déclarations de modes

vus précédemment. Ces biais et d'autres permettent de définir très précisément le langage \mathcal{L}_H que nous utilisons pour que les hypothèses produites soient non seulement valides vis à vis des exemples mais aussi pertinentes d'un point de vue biologique. Les choix de biais que nous avons effectués sont décrits au chapitre 4. Nous présentons dans la section suivante quelques unes des approches utilisées en pratique pour l'induction au sein des différents systèmes de PLI existants.

2.3.2 Algorithmes de programmation logique inductive

Il existe différents algorithmes de programmation logique inductive pour lesquels le parcours de l'espace de recherche se fait au moyen d'opérateurs d'apprentissage ou opérateurs de raffinement [Shapiro, 1981] satisfaisant la relation de subsomption imposée. Le pseudo code des algorithmes de PLI est illustré ci-après.

Algorithme de PLI :

début

Initialiser H

tant que *condition d'arrêt de H non satisfaite* **faire**

Supprimer(h,H)

Choisir des règles d'inférence (inductives ou déductives) r_1, \dots, r_k à appliquer à h

Appliquer les r_i à h pour obtenir h_1, \dots, h_n

$H \leftarrow H \cup \{h_1, \dots, h_n\}$

Élaguer(H)

fin tant que

Les deux principales approches utilisées sont la méthode descendante et la méthode ascendante.

Recherche descendante

Pour la recherche descendante, ou *top-down*, on peut distinguer deux étapes. Tout d'abord, trouver une règle par spécialisation à partir des ensembles d'exemples \mathcal{E}_+ et \mathcal{E}_- en explorant l'espace des hypothèses du plus général au plus spécifique et en prenant comme point de départ la définition la plus générale du concept comme la clause vide. Ensuite, trouver un ensemble de règles en réitérant la première étape et en mettant à jour les ensembles d'exemples autant de fois que nécessaire. Cette mise à jour peut être, par exemple, la suppression des exemples déjà couverts par une règle trouvée à la première étape. L'exploration se fait de manière constructive car à chaque hypothèse, un opérateur propose toutes les clauses qui lui sont les plus spécifiques selon la notion de généralité retenue. Des opérateurs, appelés opérateurs de raffinement, transforment une hypothèse en d'autres hypothèses. Des tests de couverture des exemples sont ensuite effectués sur les hypothèses générées pour ne retenir que celles couvrant le maximum d'exemples positifs et aucun exemple négatif. Le choix des spécialisations

suivies peut se faire à l'aide d'heuristiques, souvent calculées à l'aide des taux de couverture des exemples. Ce type d'approche appartient à la famille des algorithmes dits *generate-and-test* (générer-et-tester). Le système HAIKU, proposé par Nédellec et Rouveirol en 1994 [Nédellec et Rouveirol, 1994], reprend les caractéristiques de la majorité des algorithmes basés sur cette approche. Cet outil permet à un développeur d'applications intégrant un outil d'apprentissage de PLI de prototyper rapidement un système d'apprentissage de PLI adapté à une application spécifique, en testant par exemple les performances de diverses instanciations du système sur des échantillons de la base d'apprentissage.

Le premier système de programmation logique inductive de ce type est MIS de Shapiro [Shapiro, 1981]. MIS introduit le principe de la recherche de clauses dans un graphe d'affinement qui structure l'espace de recherche par une relation de généralité entre les clauses. La recherche exhaustive effectuée est cependant très inefficace. D'autres algorithmes ont ensuite vu le jour comme ML-SMART [Bergadano *et al.*, 1988], LINUS [Lavrac et Dzeroski, 1994], FLIPPER [Cohen, 1995] et le célèbre FOIL [Quinlan et Cameron-Jones, 1993] et ses nombreuses variantes [Pazzani et Kibler, 1992], [Mooney et Califf, 1995], [Dzeroski, 1993]. FOIL est un algorithme glouton qui tente, avec chacune de ses nouvelles règles, de couvrir le plus d'exemples positifs non encore couverts et le moins d'exemples négatifs. Chaque clause apprise dans FOIL est construite par spécialisation successive de la clause la plus générale par ajout d'un nouveau littéral guidé par un critère de *gain d'information*. Un inconvénient à cette méthode est que le nombre de littéraux peut croître rapidement et ainsi rendre le système inefficace. De plus, il n'utilise pas de connaissance *a priori*.

Recherche ascendante

Pour la recherche ascendante, ou *bottom-up*, le but est de découvrir une clause maximale spécifique couvrant un sous-ensemble des exemples positifs. À l'inverse de la méthode précédente, les règles utilisées sont des règles d'inférence inductive puisqu'il s'agit d'obtenir un ensemble de formules G à partir de la conjonction des exemples sélectionnés S telles que $G \models S$. Cette approche utilise la notion des moindres généralisés définie comme suit par Torre en 2000 [Torre, 2000] :

Définition 2.31 (moindre généralisé) *Un moindre généralisé G de H_1, \dots, H_n doit vérifier :*

- $\forall H_i : G \succeq H_i$
- $[\exists G' : (\forall H_i : G' \succeq H_i) \wedge (G \succeq G')] \Rightarrow (G \sim G')$

Lorsque les moindres généralisés sont associés à la relation de θ -subsumption on parle de *lgg (généralisation la moins générale)* [Plotkin, 1970]. Si de plus elles prennent en compte la connaissance *a priori*, ou *background knowledge (BK)*, alors on parle de

rlgg (*généralisation la moins générale relative*) [Plotkin, 1971]. La *rlgg* de deux clauses n'est pas forcément finie et peut se définir comme suit :

Définition 2.32 (rlgg, relative-least-general-generalization) *La généralisation la moins générale de deux clauses C_1 et C_2 relative à BK est la borne supérieure de C_1 et C_2 dans l'ordre de la subsomption relative.*

L'algorithme GOLEM utilise cette approche. Il a été développé par Muggleton et Feng en 1990 [Muggleton et Feng, 1992]. Pour l'induction d'une clause, il sélectionne aléatoirement des paires d'exemples positifs et calcule leurs *rlggs*. Parmi ces *rlggs*, GOLEM choisit celle qui couvre le plus grand nombre d'exemples positifs et qui est consistante avec les exemples négatifs. Cette clause est ensuite généralisée. Le processus de généralisation est répété jusqu'à ce que la couverture de la meilleure clause cesse d'augmenter. Afin d'éviter que la *rlgg* soit infinie, GOLEM impose des restrictions dans la connaissance *a priori* et le langage d'hypothèses qui assure que la taille des *rlggs* croît au pire polynomialement avec le nombre d'exemples positifs.

L'algorithme que nous avons utilisé pour nos expérimentations est PROGOL. Il a été développé par Muggleton en 1995 [Roberts, 1997].

Algorithme PROGOL :

Entrées : Échantillon d'exemples positifs \mathcal{E}_+ et négatifs \mathcal{E}_-

début

tant que \mathcal{E}_+ *non vide* **faire**

 Prendre un exemple positif e

 Construire la clause C_1 la plus spécifique impliquant e

 Trouver une clause C_2 plus générale que C_1 (au sens de la θ -subsomption) telle que la mesure de compression soit maximale

 Retirer tous les exemples couverts par la clause C_2 de \mathcal{E}_+

fin tant que

fin

PROGOL combine une approche ascendante avec une approche descendante. Pour la partie ascendante, l'inférence inductive s'appuie sur l'implication inverse proposée par S. Muggleton [Muggleton, 1992]. Le principe de cet algorithme est le suivant : partant d'un exemple positif non couvert, exprimé sous forme de clause définie, il construit la clause \perp la plus spécifique qui le couvre, appelée *bottom clause*, en appliquant la théorie B de toute les manières possibles. Il effectue ensuite sa recherche parmi toutes les clauses qui subsument \perp en utilisant un algorithme de recherche A* intégrant une mesure de compression que PROGOL cherche à maximiser (une combinaison du nombre de positifs et de négatifs couverts et de la taille de la clause). Quand la meilleure clause est trouvée, elle est ajoutée à la connaissance *a priori* et le processus recommence avec un nouvel exemple positif.

Il reste encore de nombreux problèmes à résoudre en PLI. Par exemple la notion de probabilité, commence juste à être prise en compte [Raedt et Kersting, 2004]. Des améliorations peuvent aussi être apportées sur la rapidité d'exécution et la complexité des traitements des systèmes de PLI quand l'espace de recherche est de taille importante.

2.3.3 Applications sur des séquences biologiques

La PLI a été utilisée avec succès dans des nombreuses applications en biologie structurale [King *et al.*, 1996, Hirst *et al.*, 1994, Muggleton *et al.*, 1992, King *et al.*, 1992].

En 1992, Muggleton *et al.* utilisent déjà la PLI pour prédire la structure secondaire des protéines [Muggleton *et al.*, 1992]. Ils utilisent le système GOLEM pour l'apprentissage de règles de prédiction de structure secondaire. Un ensemble de 12 protéines non homologues de structure connue est donné à l'algorithme avec une connaissance *a priori* décrivant les propriétés physico-chimiques des résidus. GOLEM apprend un petit ensemble de règles qui prédit quels résidus appartiennent à une hélice α en fonction des relations et des propriétés physico-chimiques. Les règles ont ensuite été testées sur 4 protéines indépendantes non homologues donnant un taux de prédiction de 81%. L'expérimentation reste très limitée mais les résultats montrent que l'approche est potentiellement intéressante.

Des travaux plus récents ont été publiés sur l'apprentissage de signatures sur le repliement tri-dimensionnel des protéines [Cootes *et al.*, 2001]. Pour 20 repliements issus de la base de données de classifications structurales de protéines SCOP², 59 règles ont été générées automatiquement. Le taux de prédiction correcte est de 74%. Les auteurs ont montré que leurs règles peuvent être exprimées en terme de concepts structurels et/ou fonctionnels, tels que la localisation de sites actifs.

Des améliorations ont été apportées dans [Cootes *et al.*, 2003]. Les auteurs travaillent toujours sur un ensemble de domaines protéiques extraits de la base de données SCOP sur lesquels ils ont effectués des alignements de structure multiples pour définir des coeurs de structure secondaire. Ils utilisent PROGOL-4.4 qui génère 66 règles sur ces coeurs, pour apprendre 45 repliements différents.

Ces derniers travaux sont intéressants car nous utilisons le même algorithme pour nos expérimentations. Les auteurs ont utilisé le paramètre *noise* de PROGOL qui permet de couvrir un certain taux d'instances négatives de l'échantillon d'apprentissage. Les règles générées peuvent ainsi couvrir 20% des instances négatives. Ils ont également utilisé le paramètre *inflate* qui donne un poids plus important aux exemples positifs. Celui-ci a été fixé à 200%, *i.e.* les exemples positifs sont deux fois plus important que les exemples négatifs. Ils ont aussi imposé une contrainte, sur le corps des clauses générées, en particulier sur les éléments des structures secondaires qu'elles possèdent. Ainsi la règle suivante est rejetée :

²<http://scop.berkeley.edu/>

Le repliement A possède une hélice B à la position 'b' et une hélice C à la position 'c'.

Par contre, la règle suivante est conservée :

Le repliement A possède une hélice B à la position 'b' et une hélice C à la position 'c', B et C sont en contact.

L'échantillon est réalisé de la façon suivante : ils considèrent une classe de repliement dans SCOP (*all - α , all - β , α/β , ou $\alpha + \beta$*). Dans cette classe, ils considèrent une catégorie de repliement et les domaines appartenant à la catégorie du repliement considérée sont des exemples positifs, les autres sont des exemples négatifs. Ils utilisent une validation croisée 5-fold. Les taux de prédiction correcte sont très variables (de 0 à 100%) selon la catégorie de repliement choisie mais la plupart des règles obtenues par PROGOL contiennent les informations décrites dans le manual de description des repliements de la base de données SCOP. Un autre avantage majeur est que les règles obtenues peuvent être interprétées et analysées par un biologiste.

Toutes ces travaux montrent de bons résultats en biologie moléculaire grâce à l'avantage de l'utilisation d'une connaissance *a priori* qui permet de prendre en compte de l'information supplémentaire sur les données comme les relations longues distances. De plus, les règles produites peuvent être facilement interprétées et analysées par des experts humains en structure de protéines.

Les deux méthodes que nous venons de présenter nous semblent intéressantes pour prédire les ponts disulfures dans les protéines.

Les deux chapitres suivants présentent les expérimentations effectuées tout d'abord en utilisant l'inférence grammaticale puis en utilisant la programmation logique inductive.

Chapitre 3

Apprentissage de langages de contrôles sur des séquences de protéines

Ce chapitre décrit les travaux réalisés à partir de l'article théorique de Y. Takada présenté dans le chapitre précédent. Dans un premier temps, nous détaillons les choix effectués sur la grammaire utilisée et sur les données d'apprentissage nécessaires à la prédiction des ponts disulfures dans les protéines. Dans un second temps, nous analysons les résultats des expériences que nous avons menées puis nous terminons par différentes expérimentations montrant les limites des algorithmes utilisés.

3.1 Problématique

Il s'agissait pour nous de tenter de résoudre le problème de la prédiction des ponts disulfures dans les protéines par inférence grammaticale. Comme nous l'avons vu dans le chapitre précédent, les grammaires nécessaires pour modéliser les ponts sont des grammaires au moins algébriques, puisqu'il s'agit de repérer des structures d'appariement dans la séquence. Or les algorithmes performants d'inférence disponibles travaillent essentiellement sur une classe plus petite, la classe des langages réguliers. Nous avons voulu étudier l'applicabilité pratique des travaux de Y. Takada [Takada, 1994b] pour étendre le domaine d'utilisation de ces algorithmes.

L'approche que nous avons retenue relâche le critère de caractérisation de la classe apprenable et conserve l'idée d'apprentissage du contrôle des travaux de Takada. Elle consiste donc à séparer le problème de l'inférence de ponts disulfures en deux étapes :

- modéliser par une grammaire universelle l'ensemble des appariements de ponts possibles ;
- apprendre à spécialiser cette grammaire par un langage de contrôle sur les ponts réellement observés.

La première étape permet d'introduire *a priori* l'ensemble des connaissances utiles à la résolution du problème. La seconde étape est celle de l'apprentissage proprement dit.

3.2 Expérimentations pour la prédiction des ponts disulfures

3.2.1 Cadre expérimental

L'expérimentation de l'apprentissage de langages de contrôle sur des données de protéines suppose la résolution d'un certain nombre de problèmes : le choix d'une classe de langages, la mise au point d'une grammaire universelle pertinente dans cette classe, la génération d'instances positives et négatives, et enfin la simplification *a posteriori* de l'automate pour le rendre plus explicite et plus robuste. Nous présentons ces différents points dans cette section.

Pour nos expérimentations, nous avons utilisé un fichier contenant 692 protéines qui nous a été envoyé par P. Fariselli de l'université de Bologne en Italie ¹. Ces séquences protéiques ont moins de 25% d'identité et sont issues de la PDB. La seule connaissance dont nous nous sommes servis est l'indication de la position des ponts disulfures intra-moléculaires.

3.2.2 Mise en place de l'expérimentation

Le but est de prédire les ponts disulfures dans les protéines, il faut donc tout d'abord, trouver une grammaire universelle permettant de modéliser l'ensemble des appariements de ponts possibles. Les choix que nous avons fait sont détaillés dans ce qui suit.

Choix de la classe de grammaires

Nous avons choisi de nous limiter à la classe des grammaires algébriques, qui représente un bon compromis entre la nécessité d'un pouvoir d'expression plus élevé et la nécessité d'explorer un espace de solutions raisonnable.

Du point de vue des séquences de protéines, ceci signifie que les ponts dans les séquences ne doivent pas se croiser, ce qui est une simplification de la réalité (cf figure 3.1). On peut distinguer différentes complexités d'imbrication et de chevauchement, que l'on peut mesurer par exemple par le nombre de croisements minimal de ponts nécessaires pour analyser les structures.

En fait, il est possible de prendre en compte des structures plus complexes de ponts au prix d'une grammaire plus compliquée, du fait du nombre fini de structures rencontrées en pratique. Ainsi, un chevauchement peut être décrit par une règle spécifique

¹<http://lipid.biocomp.unibo.it/piero/>

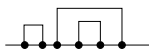
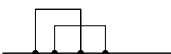
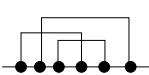
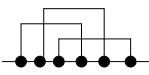
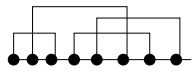
Nombre de croisements	Types de pont	Nombre de séquences cumulé de la Bdd de Fariselli
0		387
1		561
2		609
3		663
4		672

FIG. 3.1 – Quelques exemples des différents types de structures des ponts dans les séquences

reliant 4 cystéines comme : $S \rightarrow c_1 S c_2 S c_3 S c_4 S$. Le cas d'un chevauchement impliquant 3 ponts (figure 3.1) n'est pas résolu pour autant, mais il est clair qu'un nombre limité de règles doit permettre de résoudre la plupart des cas.

Nous donnons le tableau des principaux types de structure rencontrés dans la base (cf figure 3.1 avec des valeurs cumulées du nombre de séquences couvertes). Trouver le meilleur jeu de règles couvrant l'ensemble des exemples de la base est un problème d'apprentissage en soi. En fait, un nombre limité de règles suffit à couvrir la plupart des instances.

Dans notre expérimentation, nous avons juste pris en compte le type le plus simple de règle, qui couvre environ 50% de la base initiale. Il s'agit d'un compromis pour de premières expériences. Ajouter une règle permet de prendre en compte plus d'exemples mais introduit un modèle avec plus de paramètres à apprendre.

Mise au point d'une grammaire universelle

Une grammaire universelle simple reconnaissant n'importe quelle structure de protéine utilise à la fois des règles régulières de type $S \rightarrow aa S$, où aa est un acide aminé, et une ou des règles spécifiques de détection d'un pont cystéine de la forme : $S \rightarrow c S c S$.

Notons qu'une telle grammaire est non déterministe et même ambiguë à cause des deux règles $S \rightarrow c S$ et $S \rightarrow c S c S$. Cela n'est pas gênant dans le contexte d'apprentissage d'un langage de contrôle sous forme d'un automate, déterministe cette fois, qui sélectionnera à chaque fois la règle à appliquer. Il apparaît cependant qu'un tel schéma

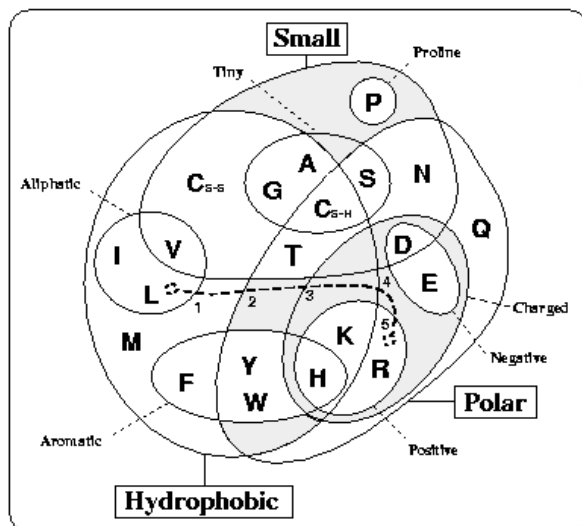


FIG. 3.2 – Diagramme de Taylor

rend extrêmement difficile l'apprentissage, car on tient très peu compte des propriétés physico-chimiques des protéines ainsi que de la structure de proximité des contextes en liaison dans la structure, dont l'étude sur la localisation des ponts a pourtant montré l'importance [Errami *et al.*, 2003, Muskal *et al.*, 1990, Fiser *et al.*, 1992]. Ce type de connaissance *a priori* sur le domaine peut être pris en compte dans le modèle de grammaire universelle, et c'est là certainement un avantage décisif de l'approche.

Ainsi, l'introduction de propriétés physico-chimiques se traduit par l'insertion de nouvelles règles décrivant les propriétés.

Exemple :

Les règles $S \rightarrow d S$ et $S \rightarrow e S$, qui impliquent les acides aminés chargés d et e , vont être remplacées par les règles $S \rightarrow \text{Chargé } S$, $\text{Chargé} \rightarrow d$, $\text{Chargé} \rightarrow e$.

Notons cependant qu'il y a deux limitations à cette introduction. Tout d'abord, l'ambiguïté devient gênante à ce niveau car un acide aminé peut appartenir à plusieurs groupes physico-chimiques comme illustré sur le diagramme de Taylor figure 3.2. En effet, s'il est possible de recoder facilement l'alphabet initial par cette technique, il n'est pas possible d'utiliser des classes empiétantes, comme c'est le cas ici, sauf si l'on dispose de connaissances spécifiques sur les instances d'apprentissage qui permettent de choisir la règle à appliquer lors de l'analyse de cette instance. L'autre limitation vient du nombre de règles à considérer dans la grammaire, qui croît rapidement avec le codage de cette connaissance. Ainsi, bien que l'on puisse travailler sur un alphabet plus petit via le recodage des acides aminés, c'est le nombre de règles de production qui compte dans la complexité de l'apprentissage puisqu'on apprend le langage de contrôle.

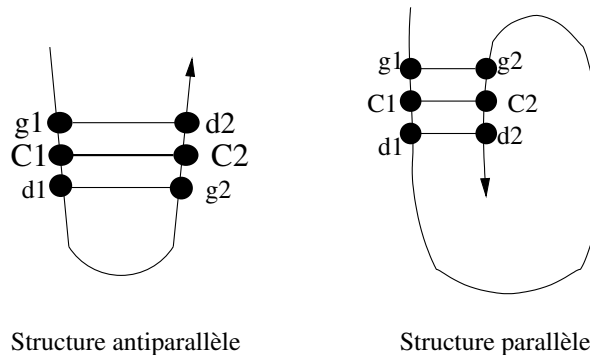


FIG. 3.3 – Types de d'appariement des acides aminés

Nous avons également essayé de considérer les contextes entre les cystéines oxydées. Quand un pont disulfure se crée, les acides aminés autour des cystéines se retrouvent les uns en face des autres. Nous avons effectué un travail sur le calcul des distances entre les acides aminés de la protéine après repliement, à l'aide du site de structures 3D PDB [Berman *et al.*, 2000]. Nous avons récupéré les coordonnées de chaque acide aminé autour des cystéines, coordonnées du carbone C_{α} , puis nous avons calculé les distances de ces acides aminés entre eux afin d'identifier les rapprochements des acides aminés. Nous avons considéré deux types d'appariement selon que les contextes s'apparient dans le même sens de lecture ou dans le sens inverse (structure parallèle ou anti-parallèle, comme illustrés figure 3.3). Nous donnons la partie de la grammaire universelle permettant de décrire ces deux types de structure :

Structure antiparallèle :

1. *Pont* \rightarrow *Acide Pont Acide*
2. *Pont* \rightarrow *c Entrepont c*
3. *Entrepoint* \rightarrow *Acide Entrepoint Acide*
4. *Entrepoint* \rightarrow *Acide Entrepoint*
5. *Entrepoint* \rightarrow ϵ

Structure parallèle :

1. *Pont* \rightarrow *Acide Pont^{Acide}*
2. *Pont* \rightarrow *S Droit*
3. *Droit^{Acide}* \rightarrow *Droit^{Acide}*

En ce qui concerne des connaissances liées à la structure même à inférer, on peut les insérer au niveau de la production des instances d'apprentissage.

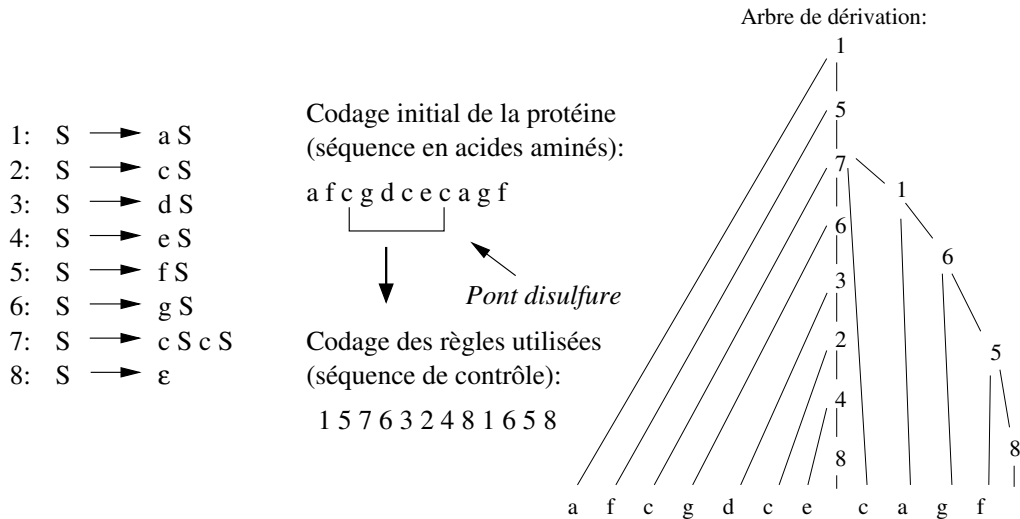


FIG. 3.4 – Réécriture d’une instance selon un parcours canonique (en profondeur et de gauche à droite) de l’arbre de dérivation correspondant à l’analyse de cette instance.

Génération des instances

Les instances sont générées à partir des arbres de dérivation de la grammaire universelle. La manière la plus directe de transformer un arbre en séquence est d’effectuer un parcours canonique de celui-ci et de recueillir les étiquettes des règles de production utilisées. Un exemple de réécriture des instances est illustré figure 3.4.

Si l’on veut prendre en compte la notion de contexte englobant les cystéines oxydées dans la protéine, on peut réécrire cependant de façon différente cet arbre. Ainsi, il semble intéressant de générer à la place de l’enchaînement canonique des règles utilisées pour analyser la séquence protéique, un enchaînement reflétant les proximités spatiales observées dans la structure.

Nous proposons d’exprimer cette proximité “à façon” dans les règles d’analyse de la grammaire, en introduisant dans celle-ci un calcul d’attributs dont le résultat est la séquence de contrôle. Une règle de type $S \rightarrow A_1 \cdots A_n$ dans la grammaire universelle sera ainsi munie d’un calcul d’attribut $S(f_s(X_1, \dots, X_n)) \rightarrow A_1(X_1) \cdots A_n(X_n)$ où X_1, \dots, X_n est le codage correspondant à l’analyse de A_1, \dots, A_n et f_s est une fonction produisant le codage de S. On a utilisé la fonction f définie comme suit :

$f(ax, yb) = abf(x, y)$, $f(x, \epsilon) = x$ et $f(\epsilon, y) = y$, qui définit un appariement de type palindromique des contextes.

Exemple : Considérons la règle générale suivante de reconnaissance de contextes de cystéines :

$$S(X \text{ f}(G1, D2) Y \text{ f}(D1, G2) Z) \rightarrow S(X) \text{ contexte}(G1) \text{ "C"} \text{ contexte}(D1) S(Y) \text{ contexte}(G2) \text{ "C"} \text{ contexte}(D2) S(Z)$$

Soit la grammaire suivante :

$$\begin{array}{ll}
 1 : S \rightarrow aS & 6 : S \rightarrow nS \\
 2 : S \rightarrow dS & 7 : S \rightarrow sS \\
 3 : S \rightarrow fS & 8 : S \rightarrow cScS \\
 4 : S \rightarrow iS & 9 : S \rightarrow \epsilon \\
 5 : S \rightarrow mS &
 \end{array}$$

Soit l'exemple suivant : 'SSADNCFISNDNAMCSMIFF' où les deux C forment un pont disulfure. Considérons des contextes de taille 3 autour des cystéines alors :

- G1 = ADN ;
- D1 = FIS ;
- G2 = NAM ;
- D2 = SMI.

La manipulation de l'arbre de dérivation avec un contexte de taille 3 palindromique revient à réécrire la séquence comme suit : '77 142567 8 3541769 339'. Le calcul d'attribut se fait de la façon suivante :

$S(SS \text{ f(ADN,SMI) ND f(FIS,NAM) FF}) \rightarrow S(SS) \text{ contexte(ADN) C contexte(FIS)}$
 $S(ND) \text{ contexte(NAM) C contexte(SMI) S(FF)}$, avec :

- $f(\text{ADN,SMI}) = \text{AIDMNS}$ (fonction palindromique) ;
- $f(\text{FIS,NAM}) = \text{FMIASN}$.

Les instances négatives représentent des séquences ne devant pas être reconnues par l'automate. Ceci pose en général un problème en biologie où il est difficile de spécifier la négation d'un concept. Dans notre cas, nous faisons l'hypothèse réaliste que les cystéines impliquées dans un pont donné ne peuvent être appariées différemment au sein de la même séquence. Autrement dit, les cystéines sont en compétition pour s'apparier entre elles et adoptent toujours la même conformation lors du repliement de la protéine. Toute autre conformation est *a priori* impossible. Ces séquences doivent de plus rester assez proches des séquences positives afin d'aider à la convergence de l'algorithme.

Nous proposons donc de construire ces exemples négatifs à l'aide de 3 fonctions appliquées sur les instances positives :

1. échange algébrique de 2 ponts ;
2. suppression d'un pont ;
3. sélection des conformations algébriques.

Afin d'illustrer les fonctions précédentes, considérons la conformation des ponts disulfures suivante dans une séquence :

$C_1C_2C_2C_1$, conformation à deux ponts disulfures, qui signifie que la 1^{re} (resp. 2^e) cystéine crée un pont disulfure avec la 4^{me} (resp. 3^e) cystéine de la protéine.

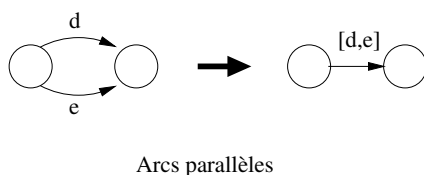


FIG. 3.5 – Réductions effectuées sur les automates

La première fonction va créer les conformations suivantes pour les instances négatives :

- $C_1C_1C_2C_2$;
- $C_1C_2C_1C_2$.

La deuxième fonction va créer les conformations suivantes pour les instances négatives :

- C_1C_1CC ;
- C_1CCC_1 ;
- CC_2CC_2 ;
- CCC_2C_2 .

La dernière fonction va élaguer la conformation suivante : $C_1C_2C_1C_2$.

Grâce aux fonctions citées ci-dessus, nous avons obtenu 1859 négatifs. Les exemples négatifs générés respectent ainsi la condition de non-chevauchement des ponts.

3.2.3 Résultats

Résultats bruts

En règle générale, et de façon surprenante, nous obtenons de meilleurs résultats en utilisant l'algorithme RPNI plutôt que l'algorithme Blue-fringe.

Après l'exécution de l'algorithme RPNI, nous obtenons un automate fini contenant 227 états et 4723 arcs. La taille de l'automate est trop importante pour continuer l'expérimentation de manière classique par une validation sur un ensemble test. Pourtant, cet automate correspond à une compression de 97% du PTA initial, qui montre un premier niveau d'apprentissage.

Avec l'utilisation de Blue-Fringe les résultats se dégradent significativement avec un automate 3 à 4 fois plus volumineux que celui obtenu par RPNI.

Afin de réduire la taille de l'automate nous avons effectué une série d'expérimentations que nous allons maintenant détailler, en agissant directement sur les transitions de l'automate ou sur les instances de l'échantillon. Le but est de trouver un compromis entre la réduction de taille de l'automate et le taux de couverture des instances.

Agir sur les transitions de l'automate pour diminuer la taille de l'automate appris

La première méthode que nous avons envisagée consiste à effectuer une opération de réduction du nombre de transitions, comme illustrée figure 3.5. Les transitions parallèles sont regroupées en une seule transition. Cette opération se formalise comme suit :

Définition 3.1 (parallèle) *Un ensemble de transitions parallèles est un ensemble de transitions ayant même état source et même état destination. Une fusion d'un ensemble de transitions parallèles est une transition dont l'étiquette est l'union des étiquettes des transitions parallèles.*

Soit $A = (Q, \Sigma, \delta, I, F)$ un automate fini, $\text{parallele}(A) = A'$ avec $A' = (Q, \Sigma', \delta', I, F)$ où $\Sigma' = \{B \mid B \subseteq \Sigma\}$, et on a :

$$\forall q_1, q_2 \in Q, \delta'(q_1, B) = q_2 \text{ où } B = \{\alpha \in \Sigma \mid \delta(q_1, \alpha) = q_2\}.$$

Cette opération réduit la taille de l'automate obtenu par RPNI de 32%.

Nous avons également considéré les propriétés physico-chimiques des acides aminés. Pour cela nous avons utilisé le diagramme de Taylor illustré figure 3.2 [Taylor, 1986]. Chaque acide aminé appartient à un ou plusieurs groupes physico-chimiques. Ainsi une transition $\delta(q_1, i) = q_2$ est réécrite $\delta(q_1, [i, l, v]) = q_2$. Nous avons défini l'opérateur de groupe suivant :

Définition 3.2 (groupe) *On appelle transition groupée une transition dont on remplace l'étiquette par l'union des étiquettes de même groupe selon Taylor.*

Soit $A = (Q, \Sigma, \delta, I, F)$ un automate fini, $\text{groupe}(A) = A'$ avec $A' = (Q, \Sigma', \delta', I, F)$, et on a :

$$\forall q_1, q_2 \in Q, \delta'(q_1, B) = q_2 \text{ si } \exists x \in \Sigma, B = \text{groupe}(x) \text{ et } \exists q_1, q_2 \in Q, \delta(q_1, x) = q_2, \text{ avec } \text{groupe}(x) \text{ l'ensemble des acides aminés du groupe de Taylor auquel appartient } x.$$

On applique ensuite une fonction pour déterminisation sur les automates. Pour compresser un automate, il faut soit être dans le cas précédent et utiliser l'opération parallèle, soit que la déterminisation provoque des fusions supplémentaires. Cet opérateur est une généralisation de l'opérateur parallèle.

Les réductions obtenues sont minimales avec cet opérateur dans nos expérimentations

Ces deux opérations sont automatiquement appliquées par la suite sur chaque automate obtenu.

Une autre idée que nous avons mise en oeuvre consiste à rechercher des sous graphes de l'automate dont les transitions sont faiblement empruntées pour la reconnaissance des instances positives. Pour cela, on attribue un score à chaque transition. Ce score correspond au nombre de passages dans cette transition, nécessaire pour reconnaître l'ensemble des séquences positives. L'idée est ensuite de supprimer un ensemble de transitions de faible score, c'est à dire les transitions qui ne sont empruntées que peu de fois. Nous proposons pour cela l'algorithme suivant :

```

Algorithmel(A automate, x entier, y entier) :
début
  EC = ∅ \\ensemble des domaines
  Pour tout état e de A faire
    Pour tout c=cheminSimple(e,A) faire
      si longueur(c)=x et score(c)≤y alors
        EC = EC ∪ {c}
      fin si
    fin pour
  fin pour tout
renvoyer EC

```

Cet algorithme recherche des sous graphes de l'automate, de longueur X, dont la somme des scores n'excède pas une certaine valeur Y. La fonction *cheminSimple* recherche des chemins simples dans un automate. La définition de ces chemins est la suivante :

Définition 3.3 (chemin simple) *Un chemin d'un automate A est une suite finie d'états de A, notée $c = [q_1q_2q_3 \cdots q_k]$, telle que $\forall q_i, q_{i+1} \in Q, \exists x_i \in \Sigma, \delta(q_i, x_i) = q_{i+1}$, q_1 est l'extrémité initiale ou origine de c et q_k son extrémité finale.*

La longueur $l(c)$ du chemin est égale au nombre minimal de transitions qu'il comporte, (une même transition peut éventuellement être répétée).

Si aucune transition ne figure plus d'une fois dans le chemin, ce dernier est dit simple et $l(c) = k - 1$.

Nous avons considéré ici des chemins de longueur 3 et de score inférieur ou égal à 3 (c'est à dire que si le chemin contient 4 états et chacune des transitions a un score égal à 1). Au total nous avons extrait 1361 chemins de ce type. Ces chemins ne sont pas tous obligatoirement disjoints (cf figure 3.2.3). Ainsi, un chemin de longueur supérieure à trois sera traité comme une succession de chemins chevauchants et supprimé dans son ensemble.



FIG. 3.6 – Deux chemins non disjoints

Nous avons donc retiré ces sous-graphes de notre automate initial. Le nouvel automate obtenu est réduit de 59.8% en nombre de transitions et de 28% en nombre d'états. Par contre, il ne reconnaît plus que les deux tiers de nos instances positives.

Agir sur les instances pour diminuer la taille de l'automate appris

Nous avons ensuite agi directement sur les instances, repérant et supprimant de l'échantillon d'apprentissage des séquences bruitées, *i.e.* des séquences exceptionnelles rendant l'automate cible beaucoup plus volumineux. Nous avons voulu explorer, dans un premier temps, l'ordre de grandeur des réductions envisageables. Pour cela, nous avons testé des méthodes de sélection d'instances :

- La première méthode repère les transitions peu utilisées dans l'automate courant. Elle consiste à mettre à jour un compteur sur l'automate qui calcule le nombre de passages dans chacune des transitions pour reconnaître l'ensemble des séquences positives. Les séquences qui empruntent le plus de transitions uniques (*i.e.* des transitions empruntées par cette seule séquence) sont supprimées. Les instances négatives que l'on a construites à partir des ces instances positives sont également retirées. RPNI sur ce nouvel échantillon, produit un automate réduit de 25%. Le pourcentage de séquences positives/négatives couvertes par cet automate par rapport à l'échantillon d'origine est de 50%. Nous avons réitéré le processus sur ce nouvel automate mais la réduction de taille obtenue est cette fois insignifiante (3%).
- La deuxième méthode de sélection d'instances que nous avons imaginée consiste à retirer itérativement les séquences conduisant à la plus grande réduction. Nous proposons l'algorithme suivant :

Algorithme2 :

début

$tailleAutomate = tailleInitiale \setminus \setminus$ taille de l'automate généré avec l'échantillon complet

Pour tout $S \in \mathcal{E}_+$ **faire**

$negatifs = rechercheNegatifs(S, \mathcal{E}_-)$

$A = AlgoApprentissage(\mathcal{E}_+ \setminus \{S\}, \mathcal{E}_- \setminus \{negatifs\})$

$A' = parallele(A)$

si $tailleAutomate > |A'|$ **alors**

$positifARetirer = \{S\}$

$negatifARetirer = \{negatifs\}$

$tailleAutomate = |A'|$

fin si

fin pour tout

$A'' = AlgoApprentissage(\mathcal{E}_+ \setminus \{positifARetirer\}, \mathcal{E}_- \setminus \{negatifARetirer\})$

$A''' = parallele(A'')$

renvoyer $A''', \mathcal{E}_+, \mathcal{E}_-$

Pour chaque séquence positive, nous calculons le nouvel automate sans cette séquence et les instances négatives construites à partir de la séquence positive. La recherche de ces instances négatives se fait grâce à la fonction *rechercheNegatifs*. Nous retirons la séquence fournissant un automate avec le moins de transitions. Ces séquences sont considérées bruitées car elles empêchent certaines fusions d'états lors de l'exécution de RPNI. Les automates obtenus ont un nombre de transitions variant de 3513 à 4896 sans application des opérateurs de réduction. Après l'application de l'opérateur parallèle illustré figure 3.5, l'automate ayant le plus petit nombre de transitions contient 168 états et 2417 arcs. Ce processus a été réitéré sur l'échantillon de départ moins cette séquence positive et ses négatifs associés. Ce processus aboutit à un seuil de réduction maximal de 51%.

Nous avons aussi raffiné la méthode en ne retirant pas les exemples négatifs de notre échantillon. Sur chaque automate obtenu nous avons analysé toutes les séquences positives de notre échantillon comme précédemment et nous retirons les transitions qui ne servent pas à la reconnaissance de ces séquences. Au final, ce sont les transitions empruntées par les exemples positifs qui sont utiles c'est pourquoi il est préférable d'avoir le plus d'exemples négatifs possibles. Ainsi certaines instances négatives peuvent entraîner des fusions d'états supplémentaires et donc réduire notre automate en nombre d'états et de transitions. L'algorithme 2 devient :

Algorithme3 :

début

tailleAutomate = *tailleInitiale* \\ taille de l'automate généré avec l'échantillon complet

Pour tout $S \in \mathcal{E}_+$ **faire**

$A = \text{AlgoApprentissage}(\mathcal{E}_+ \setminus \{S\}, \mathcal{E}_-)$

$A' = \text{parallele}(A)$

si *tailleAutomate* > $|A'|$ **alors**

positifARetirer = S

tailleAutomate = $|A'|$

fin si

fin pour tout

$A'' = \text{AlgoApprentissage}(\mathcal{E}_+ \setminus \{\text{positifARetirer}\}, \mathcal{E}_-)$

$A''' = \text{parallele}(A'')$

$\mathcal{E}_+ = \mathcal{E}_+ \setminus \{\text{positifARetirer}\}$

renvoyer $A''', \mathcal{E}_+, \mathcal{E}_-$

Nous avons réitéré ce processus 4 fois, du fait d'une convergence moins rapide. Nous avons ainsi abouti à un seuil de réduction du nombre de transitions de l'automate de 53%.

Nous avons également proposé un algorithme qui retire les séquences sans lesquelles RPNI génère un automate de taille bornée par l'utilisateur. L'algorithme est le suivant :

Algorithme4(x entier) :

début

Pour tout $S \in \mathcal{E}_+$ **faire**

$negatifs = rechercheNegatifs(S, \mathcal{E}_-)$

$A = AlgoApprentissage(\mathcal{E}_+ \setminus \{S\}, \mathcal{E}_- \setminus \{negatifs\})$

$A' = parallele(A)$

si $x > |A'|$ **alors**

$positifARetirer = positifARetirer \cup \{S\}$

$negatifARetirer = negatifARetirer \cup \{negatifs\}$

si

fin pour tout

$A'' = AlgoApprentissage(\mathcal{E}_+ \setminus \{positifARetirer\}, \mathcal{E}_- \setminus \{negatifARetirer\})$

$A''' = parallele(A'')$

renvoyer $A''', \mathcal{E}_+, \mathcal{E}_-$

Dans notre expérimentation avec RPNI et en fixant arbitrairement la borne à 4320, on retire ainsi 10 séquences. Nous relançons ensuite RPNI sur l'échantillon de départ moins ces 10 séquences et leurs négatifs associés. Nous obtenons ainsi une réduction de la taille de l'automate de 37%.

Un autre choix que nous avons mis en oeuvre est de repérer les instances similaires et d'effectuer le processus précédent une seule fois pour toutes ces séquences. L'algorithme est le suivant :

 Algorithme 5 :

début
 $tailleAutomate = tailleInitiale \setminus \setminus$ taille de l'automate généré avec l'échantillon complet
Pour tout $S \in \mathcal{E}_+$ **faire**
 $identique = similaire(S, \mathcal{E}_+)$
 $A = AlgoApprentissage(\mathcal{E}_+ \setminus \{S, identique\}, \mathcal{E}_-)$
 $A' = parallele(A)$
si $tailleAutomate > |A'|$ **alors**
 $positifARetirer = S$
 $tailleAutomate = |A'|$
si
 $\mathcal{E}_+ = \mathcal{E}_+ \setminus \{identique\}$
fin pour tout
 $A'' = AlgoApprentissage(\mathcal{E}_+ \setminus \{positifARetirer\}, \mathcal{E}_-)$
 $A''' = parallele(A'')$
 $\mathcal{E}_+ = \mathcal{E}_+ \setminus \{positifARetirer\}$
renvoyer $A''', \mathcal{E}_+, \mathcal{E}_-$

Notons que la taille de l'automate cible peut varier considérablement en fonction de l'échantillon et qu'il est difficile de prédire l'ensemble optimal d'instances à supprimer. Dans le cas des données de cystéines, nous disposons de 2 séquences S_1 et S_2 de 426 caractères tous identiques à l'exception d'un unique caractère. Nous avons exécuté RPNI sur notre échantillon sans S_1 . Nous avons obtenu l'automate A_1 , puis nous avons exécuté RPNI sur notre échantillon sans S_2 pour obtenir l'automate A_2 . En comparant ces deux automates nous avons pu remarquer que A_1 contenait 100 transitions de moins que l'automate A_2 ! Nous n'avons aucun moyen de savoir quelle séquence choisir par rapport à l'autre.

Discussion des résultats

Les diverses tentatives que nous avons réalisées pour réduire la taille de l'automate ont été de deux niveaux. Nous avons dans un premier temps agi directement sur les séquences puis dans un deuxième temps directement sur les transitions de l'automate. Le tableau 3.7 récapitule les réductions du nombre de transition des automates obtenus par RPNI en fonction des méthodes utilisées. En ce qui concerne le premier niveau, la première méthode qui supprime les arcs parallèles ne modifie pas le taux de couverture des instances puisque c'est une manière différente d'écrire l'automate. Cette opération de suppression a été appliquée sur tous les automates obtenus par la suite.

En ce qui concerne le regroupement des acides aminés selon leur propriétés physico-chimiques, les réductions obtenues sont minimales. Il s'agit en fait d'une généralisation de l'opérateur précédent. Si on ne réduit pas l'automate, cela peut provenir de plusieurs causes :

méthode	nombre transitions
initial	4723
parallèle	3212
groupe	4698
Algo1	1899
compteur	3400
Algo2	2314
Algo3	2219
Algo4	2975

FIG. 3.7 – Nombre de transition de l’automate obtenu par RPNI après application des différentes méthodes de réduction

- il n’y a pas assez d’exemples pour que les chemins supplémentaires créés existent dans ces exemples ;
- la généralisation est trop forte, ce qui conduit à des reconnaissances d’instances négatives ;
- la structure très régulière a déjà été prise en compte par le simple changement d’alphabet.

La méthode de recherche des sous-graphes de score faible a permis de réduire la taille de l’automate en nombre de transitions mais aussi en nombre d’états. Par contre l’automate obtenu ne reconnaît plus que les deux tiers de nos instances. Cette réduction au niveau de l’automate est donc proportionnelle au nombre de séquences positives rejetées. Les tentatives de réduction au niveau des transitions de l’automate n’ont pas aboutis aux résultats escomptés, c’est pourquoi nous avons souhaité agir directement sur les séquences de notre échantillon.

La recherche de séquences considérées bruitées nous semblait être une piste intéressante. En effet, certaines séquences peuvent empêcher des fusions d’états de notre automate et ainsi augmenter sa taille. Le premier test effectué en retirant les séquences qui empruntent le plus de transitions uniques mène à une diminution de la taille de notre automate mais de manière proportionnelle au taux de couverture de nos instances positives. Par contre la deuxième méthode qui consiste à retirer les séquences fournissant les plus gros automates mène à une réduction de la taille de notre automate intéressante tant au niveau des transitions qu’au niveau des états. Plus encore lorsque les séquences sont retirées une à une. En effet, la suppression d’une unique séquence offre une diminution de la taille de notre automate de 28%. Ce qui est le meilleur score obtenu.

Nous avons raffiné cette méthode en ne supprimant pas les instances négatives associées aux séquences positives retirées. En effet, nous retirions certaines séquences qui favorisaient des fusions d’états car RPNI génère parfois des “fausses” transitions en fonction des instances négatives. Ces transitions ne sont pas empruntées par les instances

positives mais favorisent des fusions supplémentaires. Au final, le nombre de transitions que nous comptons correspond aux transitions empruntées par nos instances positives. C'est pourquoi, il est préférable d'avoir le plus d'exemples négatifs possibles.

La dernière méthode présentée ici prouve qu'il est difficile de faire des choix sur les instances à ne pas considérer. Le résultat de cette expérience est tout de même spectaculaire. Deux séquences entièrement identiques sauf à une unique position donne des automates de taille complètement différente. En effet, l'un des deux automates obtenus a plus de cent transitions de plus. Ce qui signifie qu'un unique acide aminé, parmi des milliers, réussit à faire diminuer la taille de l'automate de façon considérable.

D'autres expériences ont été réalisées mais n'ont pas été décrites ici car les résultats obtenus semblent encore loin d'approcher un automate de taille minimale par rapport aux contraintes et instances données. Ce sont les algorithmes eux-mêmes qui semblent devoir être remis en cause dans ce contexte. Nous avons voulu commencer à analyser les raisons de cet échec relatif d'algorithmes obtenant pourtant des résultats intéressants sur des automates aléatoires. Nous présentons par la suite deux classes de problèmes génériques correspondant à deux des difficultés présentes dans notre application, et montrons en effet expérimentalement l'inadaptation des critères de fusion habituels.

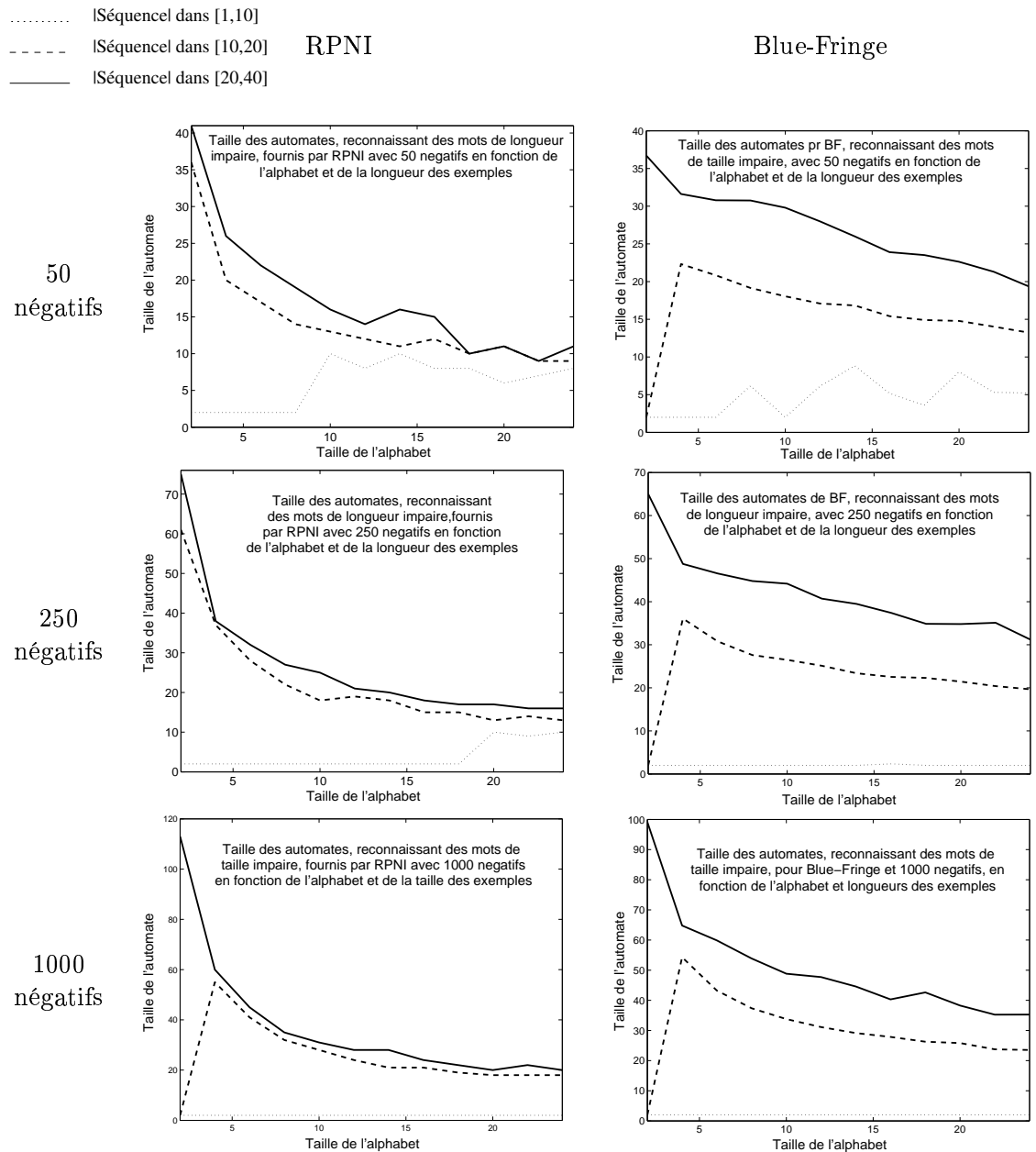
3.3 Analyse des limitations des algorithmes d'inférence régulière pour l'inférence de langages de contrôle

Nous avons effectué différentes expérimentations mettant en jeu tout d'abord la taille de l'alphabet, la longueur des séquences ainsi que le nombre d'instances négatives. Ensuite, nous avons effectué une étude sur le problème de recherche d'un motif dans une séquence qui est un problème classique en bioinformatique. Cela revient à apprendre le langage $\Sigma^n w \Sigma^m$ en faisant varier les paramètres n et m ainsi que la taille du motif w .

Nous avons effectué des comparaisons sur la taille des automates fournis par ces deux algorithmes, RPNI et Blue-Fringe (nous n'avons pas ré-implémenté ces algorithmes mais utilisé ceux existant, *i.e.* [Oncina et Garcia, 1992] pour RPNI et [Lang *et al.*, 1998] pour Blue-Fringe). Nous présentons dans cette section deux séries de tests visant à souligner le comportement des algorithmes actuels d'inférence régulière par rapport au type de données rencontrées dans notre application. Tous les tableaux ou graphiques qui suivent représentent une moyenne sur 100 expériences.

3.3.1 Influence de la taille de l'alphabet et des instances sur l'inférence

Nous étudions la dégradation de l'identification en fonction de la taille des séquences d'apprentissage disponibles et en fonction de la taille de l'alphabet. Le cadre d'expérimentation généralement utilisé pour l'inférence d'automates aléatoires utilise un générateur produisant des tailles de séquences variées et en particulier des mots courts, et se



restreint à un alphabet binaire. Si les séquences sont des protéines, ces hypothèses sont loin d'être remplies : la disponibilité de séquences courtes n'est pas évidente (>300 en moyenne) et l'alphabet de base comporte 20 lettres.

Plus généralement, dans le cadre d'apprentissage de langages de contrôle, l'alphabet est celui des numéros de règles et peut donc être assez grand. La taille des séquences est généralement comparable à celle des séquences d'origine.

Le problème est d'estimer dans quelle mesure l'absence des séquences courtes et l'augmentation de la taille du vocabulaire vont retarder ou empêcher la convergence des algorithmes d'inférence.

La figure 3.8 montre l'influence de la taille de l'alphabet, de la taille des séquences et de la taille de l'échantillon caractéristique sur la taille des automates fournis par RPNI et Blue-Fringe. Le but de ces automates est de reconnaître les mots de longueur impaire, ce qui correspond à l'automate à 2 états représenté figure 3.11. C'est un automate élémentaire dont le nombre d'états est indépendant de l'alphabet choisi.

Les exemples utilisés pour nos expérimentations ont été générés aléatoirement. Nous avons créé un algorithme qui renvoie un nombre X de séquences de longueur Y où X est un entier et Y est un intervalle d'entiers spécifiés par l'utilisateur.

Les deux premiers graphiques figure 3.8 représentent la taille des automates fournis par RPNI (à gauche) et Blue-Fringe (à droite) en utilisant un échantillon positif de 100 séquences et négatif de 50 séquences. Ce tableau met en évidence l'importance de la taille des séquences. Plus l'alphabet est petit et les séquences longues, plus il est difficile pour les deux algorithmes d'obtenir un automate de petite taille.

Sur une taille de séquences fixée, plus la taille de l'alphabet augmente, plus l'automate grossit. Nous pouvons remarquer que plus la taille des séquences est importante et plus l'alphabet est grand, les automates obtenus avec RPNI sont moins volumineux que ceux obtenus avec Blue-Fringe.

Les graphiques suivants ont été obtenus avec un échantillon comprenant respectivement 250 et 1000 séquences négatives. Plus il y a de séquences négatives de grande taille, sur un alphabet à deux éléments, plus il est difficile d'obtenir l'automate à deux états. On peut remarquer qu'en règle générale les résultats sont meilleurs avec RPNI (sauf pour un alphabet à deux éléments). Clairement, même pour un automate cible très simple à 2 états, ces algorithmes ne sont pas efficaces pour un alphabet de grande taille et de longues séquences. Il faudrait donc que les algorithmes soient guidés également par le modèle et pas simplement par les données.

3.3.2 Influence de la localisation des motifs sur l'inférence

Notre but est de reconnaître un langage du type $\Sigma^*w\Sigma^*$ où w représente le mot à apprendre. Nous devons donc reconnaître des mots de la forme $\Sigma^m w \Sigma^n$, où m et n sont deux entiers positifs ou nuls.

pref	0						5					
suff	0	5	10	15	20	25	0	5	10	15	20	25
2	1.49	6.52	8.98	25.52	61.31	66.17	2.52	7.48	10.05	31.03	62.25	67.19
4	2.94	7.66	12.21	62.38	70.81	82.78	3.81	8.51	12.94	59.78	76.23	82.31
6	3.99	8.64	15.33	71.59	83.96	91.98	4.89	9.68	15.11	73.83	82.17	89.21
8	4.63	9.95	17.25	71.93	87.15	96.4	5.63	10.79	19.12	75.34	87.71	91.71
10	5.59	10.94	19.12	74.04	90.41	98.39	6.28	12.01	19.17	73.04	91.69	100

pref	10						15					
suff	0	5	10	15	20	25	0	5	10	15	20	25
2	2.63	7.56	10.34	29.94	62.81	67.72	2.55	7.4	10.03	32.23	61.09	67.1
4	3.95	8.41	13.66	57.53	76.35	81.94	3.99	8.45	14.17	65.19	76.6	82.35
6	4.78	9.46	15.95	73.23	86.07	90.84	4.96	9.82	17.02	68.82	86.98	91.53
8	5.54	10.85	18.44	70.24	86.55	94.27	5.44	10.55	18.15	74.15	88.52	94.32
10	6.22	11.98	20.43	69.73	94.48	99.95	6.21	11.83	20.96	71.55	95.35	99.59

pref	20						25					
suff	0	5	10	15	20	25	0	5	10	15	20	25
2	2.48	7.5	10.04	31.22	61.43	67.22	2.52	7.49	10.1	32.2	60.48	67.26
4	3.89	8.44	13.99	63.5	73.24	82.33	3.62	8.57	13.36	59.55	75.5	81.06
6	4.78	9.66	15.66	71.62	83.28	91.81	4.75	9.74	14.81	70.93	84.21	90.03
8	5.52	10.86	20.77	74.7	88.08	96.64	5.58	10.67	19.13	67.7	87.22	94.26
10	6.17	11.92	20.76	73.26	92.2	99.33	6.38	11.89	19.57	73.03	91.46	99.12

FIG. 3.9 – Taille des automates par RPNI en fonction de la position du mot dans la séquence

pref	0						5					
suff	0	5	10	15	20	25	0	5	10	15	20	25
2	2.42	7.41	10.85	50.39	55.24	57.96	2.5	3.96	3.96	24.2	29.02	26.13
4	3.6	8.2	13.05	49.08	52.16	56.76	3.66	5.94	6.02	26.01	33.35	30.05
6	4.86	8.87	12.84	35.39	48.25	46.51	4.74	7.73	8.92	33.9	41.9	39.5
8	5.85	9.84	13.86	30.52	36.78	40	5.5	8.9	10.47	25.18	31.9	35.64
10	6.98	10.9	13.72	34.73	40.7	41.32	6.24	10.1	11.83	28.96	29.3	37.1

pref	10						15					
suff	0	5	10	15	20	25	0	5	10	15	20	25
2	2.53	3.96	3.96	4.96	4.95	5.16	2.43	3.96	3.96	4.51	3.96	3.96
4	3.69	5.94	5.94	7.48	6.99	8.64	3.58	5.94	5.94	5.94	5.94	5.94
6	4.63	7.73	8.96	18.4	25.72	22.53	4.69	7.7	9	19.6	23.64	26.67
8	5.5	9	11.75	22.88	28.83	32.1	5.65	9.27	10.28	23.71	24.5	30.27
10	6.27	10.2	11.48	24.65	27.6	29.19	6.14	10.28	11.3	23.6	25.33	28.12

pref	20						25					
suff	0	5	10	15	20	25	0	5	10	15	20	25
2	2.53	3.96	3.96	3.96	3.96	3.96	2.6	3.96	3.96	3.96	3.96	3.96
4	3.71	5.94	5.94	5.94	5.94	5.94	3.53	5.94	5.94	5.94	5.94	5.94
6	4.67	7.7	9	23.32	21.53	22.87	4.6	7.73	8.97	20.32	21.22	20.92
8	5.48	8.9	10.9	24.76	26.19	27.9	5.31	9	10.41	21.98	24.63	27.93
10	6.35	10	11.79	22.17	25.48	31.29	6	10	11.86	21.97	30.86	26.33

FIG. 3.10 – Taille des automates par Blue-Fringe en fonction de la position du mot dans la séquence

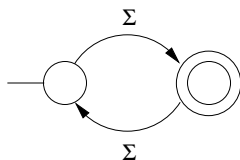


FIG. 3.11 – Automate reconnaissant les mots de longueur impaire

Les tableaux figures 3.9 et 3.10 montrent la taille des automates en fonction des paramètres m (0 à 25 caractères, 1^{re} ligne), n (0 à 25 caractères, 2^e ligne) et de la longueur du mot à apprendre (2 à 10 caractères, 1^{re} colonne). L'alphabet utilisé contient 2 caractères.

L'échantillon d'apprentissage contient 100 séquences positives obtenus de manière aléatoire pour chaque cas. Le mot w (obtenu aléatoirement) sera le même dans les 100 exemples. Chaque exemple contiendra donc un préfixe aléatoire de taille fixée, suivi du mot w et d'un suffixe aléatoire de taille fixée. Les exemples négatifs sont créés de la même manière avec le même mot que dans les séquences positives avec une ou 2 substitutions dans ce mot. Pour chaque exemple positif, le nombre de séquences positives associées est de $\sum_{i=1}^{taille\ mot\ w} i$.

Lorsque la partie caractéristique du langage cible est incluse dans une expression plus large, l'identification se dégrade là aussi rapidement en fonction de la taille des contextes préfixes et suffixes considérés. Nous pouvons remarquer que la taille du plus petit automate appris suit la taille du mot. La deuxième est que la taille des préfixes (m) ne joue pas un rôle très important sur la taille des résultats. Par contre, la taille des automates croît en fonction de celle du suffixe. Dans cette expérience c'est l'algorithme Blue-Fringe qui donne de meilleurs résultats (peut être parce que nous travaillons sur un alphabet à 2 lettres).

Dans le cas de nos protéines, les préfixes et les suffixes sont de taille très variable. C'est pourquoi nous obtenons des automates aussi volumineux.

Les algorithmes apparaissent peu robustes vis à vis de ces critères, ce qui a des conséquences importantes dans le contexte de notre application.

En ce qui concerne le thème principal de cette thèse, à savoir la prédiction des ponts disulfures, la difficulté majeure est que l'on travaille sur un alphabet à 20 lettres avec des séquences de grande taille. En effet, la plus longue de nos séquences possède 1300 acides aminés et aucune de nos séquences n'a une taille inférieure à 100 acides aminés. Nous avons montré que les algorithmes d'inférence régulière ne sont pas du tout adaptés tout d'abord pour des alphabets de grande taille mais surtout pour de longues séquences.

Une idée envisageable est de travailler sur des morceaux de protéines, *i.e.* ne

considérer uniquement les acides aminés autour des cystéines dans la protéine. Une instance positive serait par exemple deux fenêtres d'acides aminés juxtaposées. La première contiendrait les acides aminés voisins, dans la séquence protéique, de la première cystéine oxydée d'un pont disulfure et la deuxième contiendrait les acides aminés voisins de la deuxième cystéine oxydée du même pont disulfure. Même si l'on considérait un faible nombre d'acides aminés voisins, par exemple 5 acides aminés de chaque côté de la cystéine, cela reviendrait à travailler sur des séquences d'au moins 22 acides aminés. Si l'on regarde les résultats de la figure 3.8, les séquences de longueur supérieure ou égale à 22 correspondent à la courbe pleine. Dans le meilleur des cas présentés ici, *i.e.* avec un échantillon contenant dix fois plus d'instances négatives que positives, sur un alphabet de 20 lettres, l'automate élémentaire fourni par RPNI ou Blue-Fringe n'est pas l'automate cible. En effet, cet automate contient plus de 30 états alors qu'il ne devrait en contenir que deux (automate cible).

Cette étude visait à confronter un cadre théorique d'apprentissage et des algorithmes généraux d'inférence grammaticale à une application pratique de prédictions d'appariement spécifiques au sein d'une séquence protéique. Un point qui nous semble très positif dans cette étude est de relancer l'intérêt pour la conception de nouveaux algorithmes ou de nouvelles heuristiques d'inférence régulière. Ceci permettra d'aborder ensuite l'inférence de classes de langages plus complexes, via l'apprentissage de leur langage de contrôle. Nous avons souligné un avantage décisif de cette méthode, l'introduction facile de connaissances *a priori* dans le modèle et montré les différentes étapes à résoudre pour son apprentissage. Une autre méthode nous semble intéressante pour résoudre le problème des ponts disulfures dans les protéines, la programmation logique inductive. En effet, elle offre la possibilité d'insérer des connaissances sur les données lors de l'apprentissage et permet donc de prendre en compte des relations *a priori* entre les données. Les expériences et résultats obtenus au moyen de cette méthode font l'objet du chapitre suivant.

Chapitre 4

Apprentissage de programmes logiques sur des séquences de protéines

Ce chapitre décrit nos travaux réalisés en apprentissage par programmation logique inductive. Nous présentons tout d'abord le cadre opérationnel dans lequel doit se dérouler le processus d'induction. Nous y détaillons les étapes de constitution des exemples nécessaires à l'apprentissage et les contraintes qui sont imposées sur la forme attendue des motifs à travers le langage d'hypothèses. Nous étudions ensuite les problèmes de complexité. Nous présentons notamment la technique utilisée pour parcourir efficacement l'espace de recherche et les moyens d'éviter l'exploration de portions de cet espace ne pouvant conduire à de bonnes solutions. Nous terminons enfin par l'évaluation de notre approche sur la prédiction des ponts disulfures dans les protéines en présentant les résultats obtenus dans un premier temps, sur la prédiction des cystéines oxydées, puis dans un second temps sur l'appariement des cystines.

L'apprentissage effectué ici repose sur la programmation logique inductive pour inférer des motifs caractéristiques autour des cystéines oxydées et expliquant l'appariement de celles-ci. Il est cependant nécessaire d'adapter à notre tâche cette technique d'apprentissage. Il faut en particulier qu'elle génère des règles qui soient biologiquement pertinentes, c'est à dire reflétant un phénomène biologique identifiable par un expert. Pour cela, de nombreuses contraintes sont fixées sur le format des motifs attendus. Par ailleurs, comme beaucoup de techniques d'apprentissage, la PLI a un coût calculatoire élevé qu'il convient de maîtriser. Il faut donc ajuster la modélisation pour qu'elle tire parti au mieux des informations disponibles sur les exemples et que l'inférence soit réalisée en un temps raisonnable.

4.1 Inférence

Les structures des protéines sont le résultat d'interactions complexes entre sous-structures. Pour cette raison, les algorithmes de la PLI, qui apprennent des relations entre des données et pas simplement des fonctions de prédiction, semblent particulièrement pertinents pour ce genre de données. Pour effectuer nos expérimentations nous avons utilisé l'algorithme PROGOL [Roberts, 1997] que nous avons présenté au chapitre 2. Comme pour la plupart des méthodes d'apprentissage, la difficulté majeure pour l'utilisation des techniques de la PLI est le réglage de nombreux paramètres. Résoudre un problème nécessite un compromis entre la taille de l'espace d'hypothèses à explorer et l'expressivité qu'apporte cet espace pour discriminer les situations. Plus le langage d'expression des hypothèses est riche et plus sera aisée la séparation des exemples et contre-exemples mais plus il sera difficile d'explorer efficacement l'ensemble des modèles possibles. On peut distinguer trois types de tâches :

- La première consiste à concevoir une représentation adéquate des instances d'apprentissage en même temps que la connaissance *a priori*. Les instances d'apprentissage sont codées sous forme de faits logiques (à l'aide du prédicat *example* dans PROGOL). La connaissance *a priori* est représentée par un ensemble de clauses définies. Elle est essentielle pour la convergence de l'apprentissage car elle contient la définition de relations potentielles intéressantes, ainsi que des contraintes d'intégrité. La connaissance *a priori* assure la capacité d'inférer des relations à la fois comme classifieurs efficaces de nouvelles instances mais aussi pour expliquer de manière interprétable l'appartenance de ces instances à la classe. Ceci diffère clairement des méthodes telles que les réseaux de neurones où il est difficile d'interpréter les classifieurs.
- La seconde tâche est la spécification de l'espace d'hypothèses. Ceci est réalisé dans PROGOL par les déclarations des types tête et corps des règles (prédicats *modeh* et *modeb*). Les règles générales corps => tête que PROGOL construit sont contraintes d'utiliser uniquement les prédicats, les variables et les constantes indiqués par les types. Tout ceci décrit alors un espace fini, généralement énorme, bâti sur l'espace des constantes possibles appelé l'univers de Herbrand. L'utilisateur peut de plus définir une restriction explicite de cet espace (prédicat *prune*) qui indique les règles qui ne sont pas valides dans l'espace précédent.
- La dernière tâche concerne divers paramètres influençant la stratégie et la complexité de la recherche. Par exemple, on peut fixer le nombre maximal de règles à explorer durant la recherche, spécifier la taille maximale du corps des règles générales que PROGOL construit, autoriser que les règles prédisent un faible pourcentage d'exemples négatifs (paramètre *noise*), ou forcer PROGOL à effectuer son apprentissage à partir d'exemples positifs uniquement (paramètre *posonly*).

Dans cette section nous décrivons le processus d'apprentissage par PLI et plus particulièrement les adaptations de l'algorithme d'inférence utilisé, PROGOL, nécessaires à la production de clauses pertinentes. Le problème de la prédiction des ponts disulfures est ici décomposé en deux sous-problèmes. Tout d'abord l'apprentissage de règles caractéristiques de l'état d'oxydation des cystéines. Puis, l'apprentissage de règles caractéristiques de l'appariement des cystéines. Nous allons dans un premier temps détailler la prédiction de l'état oxydé des cystéines. Les règles obtenues serviront de point de départ à la prédiction de la mise en relation des deux contextes oxydés.

4.1.1 Constitution de l'échantillon pour la prédiction des cystéines oxydées

Notre première tâche consiste à construire les ensembles d'exemples positifs \mathcal{E}_+ et négatifs \mathcal{E}_- nécessaires à PROGOL pour inférer des clauses généralisées expliquant ce qui distingue en terme de contexte physico-chimique les cystéines oxydées des cystéines libres.

La plupart des auteurs extraient les ponts disulfures à partir de protéines provenant de SwissProt¹ dont on connaît les ponts intra-chaînes. Depuis quelques années, la détermination de ponts disulfures est plus simple [Gorman *et al.*, 2002], et de nombreuses annotations devraient devenir progressivement disponibles. Cependant, on peut clairement distinguer les ponts vérifiés expérimentalement et prédits (par similarité, potentiel, probable, ...). En fait, moins de 12% des ponts disulfures répertoriés sont déterminés expérimentalement ! En 2004, de nombreux efforts ont mené à des bases de données de modèles disulfures. Dans l'article [Vlijmen *et al.*, 2004], une méthode basée sur SwissProt et les bases de données d'alignement multiple Pfam² conduit à 94499 modèles disulfures, mais les auteurs n'ont pas donné l'accès de cette base de données.

La base de données d'apprentissage que nous utilisons nous a été fournie par D. Tessier, qui a soigneusement sélectionné un ensemble de chaînes d'une base de données pdb_select_25 (moins de 25% d'identité entre les protéines). Ces chaînes contiennent au moins un pont disulfure répertorié et sont extraites à partir de cellules eukaryotes. Dans son article [Tessier *et al.*, 2004], D. Tessier insiste sur le fait que les signaux peptidiques et les localisations sous-cellulaires sont les descripteurs primaires qui doivent être considérés, s'ils sont accessibles, pour la prédiction de l'état oxydé des cystéines. Néanmoins, nous nous sommes restreint à la séquence protéique uniquement. En effet, nous ne nous intéressons pas uniquement à la prédiction de l'état des cystéines, mais nous essayons également de comprendre l'appariement des cystéines entre elles en fonction de la séquence. Nous avons sélectionné dans sa base de données les protéines extra-cellulaires avec des ponts disulfures intra-moléculaires. Nous avons travaillé sur un ensemble d'apprentissage contenant 722 instances positives (cystines)

¹<http://www.expasy.org/sprot/>

²<http://www.sanger.ac.uk/Software/Pfam/>

et 47 instances négatives (cystéines libres). Cet échantillon contient des protéines de taille variée allant de 30 acides aminés à plus de mille. Nous n'avons que 47 séquences négatives car nous travaillons sur des protéines provenant du même milieu. Celui ci favorise la création des ponts disulfures. Considérer des cystéines non oxydées issues d'un milieu différent peut fausser l'apprentissage car nous ne savons pas si ces cystéines s'oxydent dans notre milieu.

Pour la constitution de notre échantillon d'apprentissage, nous utilisons l'hypothèse que la présence d'un pont entre deux cystéines est corrélée avec l'environnement local des résidus voisins dans la conformation spatiale de la protéine. La plupart des auteurs suggèrent qu'il devrait être suffisant d'extraire de la séquence primaire une fenêtre de contexte de 11 résidus centrée autour des cystéines. Nous avons utilisé une longueur de fenêtre un peu plus grande de 14 résidus de chaque côté de la cystéine (une fenêtre pour chaque cystéine constituée des 7 acides aminés de chaque côté de la cystéine). Nous avons déterminé le choix de la taille de cette fenêtre à partir d'une étude systématique sur la position des acides aminés dans la structure 3D des protéines de l'échantillon. Nous avons récupéré pour cela les coordonnées des atomes $C\alpha$ des acides aminés de nos protéines, sur le site de structures 3D PDB [Berman *et al.*, 2000]. Nous avons tout d'abord calculé la distance des acides aminés entre eux. Nous avons ensuite centré une boule sur les cystéines oxydées. Le rayon de cette boule a été déterminé afin de capter les trois acides aminés voisins de chaque côté de la cystéine. Ce rayon correspond à environ 10Å. Nous avons ensuite centré cette boule sur les acides aminés voisins de la cystéine. Puis nous avons regardé si cette boule contenait des acides aminés proches de l'autre cystéine impliquée dans le pont disulfure. En général, seuls les sept acides aminés voisins captent dans leur boule des acides aminés proches des deux cystéines oxydées. C'est pourquoi nous avons choisi de considérer une fenêtre de taille 7.

Un exemple positif de notre échantillon, illustré figure 4.2, est ainsi représenté par un prédicat *example/1* : *example(Context)*, où *Context* représente un mot de taille 14. Pour les exemples positifs, ces fenêtres contiennent des acides aminés entourant des cystines tandis que pour les exemples négatifs ce sont des acides aminés autour des cystéines libres dans la protéine, représentés par la clause “: *-example(Contexte)*”.

Connaissance préalable

Nous introduisons de la connaissance sur nos données d'apprentissage dans la connaissance *a priori* (Background Knowledge). Ce fichier est fourni en annexe 5.2. L'ensemble des propriétés physico-chimiques des acides aminés est codé en utilisant un arbre reflétant les dépendances d'inclusion parmi les propriétés. Plus précisément, nous avons utilisé la représentation 'astucieuse', souvent utilisée en apprentissage, qui est de coder les acides aminés et leurs propriétés à l'intérieur d'une même représentation.

D'après le diagramme de Taylor (cf [Taylor, 1986]) présenté au chapitre 3, il existe trois grands groupes physico-chimiques qui englobent l'ensemble des acides aminés :

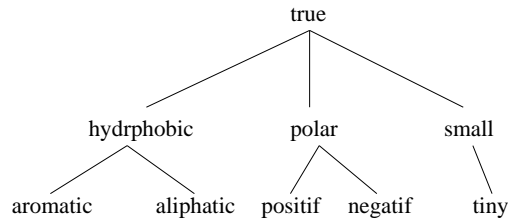


FIG. 4.1 – Arbre général extrait du diagramme de Taylor.

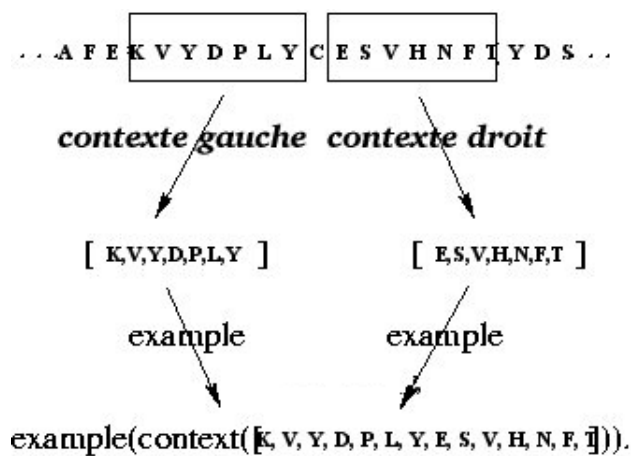


FIG. 4.2 – Exemple = juxtaposition de 2 fenêtres de chaque coté d’une cystéine dans une protéine.

hydrophobe, polaire, et petit.

Nous proposons de représenter chaque acide aminé par un arbre, illustré figure 4.1 traduisant l’ensemble de ses propriétés de manière structurée : le père d’une propriété est une propriété plus générale et on introduit la négation de manière explicite pour chaque propriété. Les propriétés, qui correspondent à des ensembles d’acides aminés, sont représentées par des arbres partiellement instanciés, *i.e.* comportant des variables.

Exemple :

D’après le diagramme de Taylor, l’acide aminé *G* est codé comme $true(hydrophobic(not_ali_aro),not_polar,small(tiny))$ et la propriété *small* est codée comme $true(X,Y,small(Z))$, où *X*, *Y* et *Z* sont des variables. Nous avons introduit cette connaissance dans notre connaissance *a priori*, sous forme de clauses définies.

Certaines contraintes sont imposées sur la forme attendue des règles à travers le langage d’hypothèses. Nous présentons les choix que nous avons effectués dans la section suivante.

4.2 Exploration de l'espace des hypothèses

L'espace de recherche $\mathcal{E}_{\mathcal{H}}$ est généralement très grand voire infini. Une recherche exhaustive des hypothèses possibles, et leur test de couverture, à travers tout l'espace est donc exclue. Des biais sont utilisés à la fois pour réduire cet espace de recherche mais également pour assurer la sélection d'hypothèses pertinentes (cf section 2.3.1.3). Comme nous l'avons vu précédemment, parmi les différents types de biais existants, l'un des plus naturels et des plus utilisés est le langage d'hypothèses qui permet de définir des contraintes syntaxiques sur les clauses recherchées. Nous présentons ci-dessous ceux que nous imposons à notre tâche d'apprentissage.

Notre approche s'appuie sur la recherche de propriétés de sous-séquences dans les fenêtres précédemment présentées. Bien que notre but soit de révéler le maximum de relations potentielles entre des complexes de résidus, nous mixons les aspects logiques et statistiques : la propriété finale est une conjonction de modèles sur les sous-séquences, mais la recherche des modèles eux-mêmes correspond à une étude de la composition dans les sous-fenêtres de taille fixée. Commençons par quelques définitions précisant le cadre formel de l'espace d'hypothèses correspondant.

Définition 4.1 (Sous-séquence) *Soit S une séquence sur un alphabet Σ , alors $W_1 \dots W_k \in \Sigma^k$ est une sous-séquence de S ssi*

$$\exists u_1 \dots u_{k+1} \in \Sigma^*, S = u_1 W_1 \dots u_k W_k u_{k+1}$$

Par exemple, KVV.P.E est une sous-séquence de la protéine AFEKVVYDPLYESVH.

Définition 4.2 (k-formule) *Soit S une séquence sur un alphabet Σ , alors une k-formule est une conjonction de sous-séquences de propriétés $P_1 \& \dots \& P_l$ telle que :*

il existe une bijection σ de $\{1, \dots, k\}$ dans $\{1, \dots, 1m(1), \dots, l1, \dots, lm(l)\}$ telle que $P_i = p_{i1} \dots p_{im(i)}$ pour i variant de 1 à l , où p_{ij} correspond à des propriétés élémentaires (sur des acides aminés).

Définition 4.3 (interprétation d'une k-formule) *Une k-formule est vraie sur une séquence S ssi il existe un mot $v_1 \dots v_k$ de S tel que $\forall i \in [1, l], \forall j \in [1, m(i)] p_{ij}(v_{\sigma(ij)})$*

L'espace d'hypothèses est composé de conjonctions de k-formules. Dans nos expériences, nous avons fixé k à 4, ce qui revient à dire que les propriétés sont vérifiées sur des sous-fenêtres de taille 4 à l'intérieur du contexte de la cystéine de taille 14 (cf figure 4.3). Nous avons choisi une taille de 4 pour limiter la combinatoire mais cette valeur peut être modifiée.

Par exemple, dans la protéine de la figure 4.3, KVVYD est un sous-mot de taille 4. Les propriétés physico-chimiques de chacun de ces acides aminés sont :



FIG. 4.3 – Sous-fenêtre pour la détection de modèles de taille 4 dans le contexte d'une cystéine.

- $K \in \{\text{positif, chargé, polaire, hydrophobe}\}$;
- $V \in \{\text{aliphatique, hydrophobe, petit}\}$;
- $Y \in \{\text{aromatique, hydrophobe, polaire}\}$;
- $D \in \{\text{négatif, chargé, polaire, petit}\}$.

Un 4-modèle du sous-mot KVYD est par exemple : 'hydrophobe aliphatique polaire petit'.

Les sous-séquences de propriétés sont des prédicats sous forme de mots $p_1 \dots p_m$ de propriétés physico-chimiques vraies ssi la sous-séquence correspondante $v_1 \dots v_m$ vérifie toutes les propriétés physico-chimiques données. Remarquons qu'il est utile d'ajouter la classe entière X des acides aminés à ce diagramme. Ceci permet d'introduire des positions contenant un caractère quelconque dans la k -formule de sous-mots. Cette propriété universelle sera directement ajoutée dans l'ensemble des arrangements possibles de sous-séquences (voir plus loin la description de types de sous-séquences dans la connaissance *à priori*).

Exemple :

Considérons l'exemple positif suivant : $example(context[A, A, R, D, H, E, W, A, G, N, G, E, D, Q])$. D'après notre représentation, les acides aminés de ce contexte sont décrits par les arbres :

- $A = true(hydrophobic(not_ali_aro), not_polar, small(tiny))$
- $R = true(not_hydro, polar(charged(positive)), not_small)$
- $D = true(not_hydro, polar(charged(negative)), small(not_tiny))$
- $H = true(hydrophobic(aromatic), polar(charged(positive)), not_small)$
- $E = true(not_hydro, polar(charged(negative)), not_small)$
- $W = true(hydrophobic(aromatic), polar(not_charged), not_small)$
- $G = true(hydrophobic(not_ali_aro), not_polar, small(tiny))$
- $N = true(not_hydro, not_polar, small(not_tiny))$

Pour simplifier, nous utilisons les notations suivantes :

- $a = aliphatic = true(hydrophobic(aliphatic), X, Y)$
- $r = aromatic = true(hydrophobic(aromatic), X, Y)$
- $h = hydrophobic = true(hydrophobic(Z), X, Y)$
- $c = charged = true(X, polar(charged(Z)), Y)$
- $+ = positive = true(X, polar(charged(positive)), Y)$
- $- = negative = true(X, polar(charged(negative)), Y)$

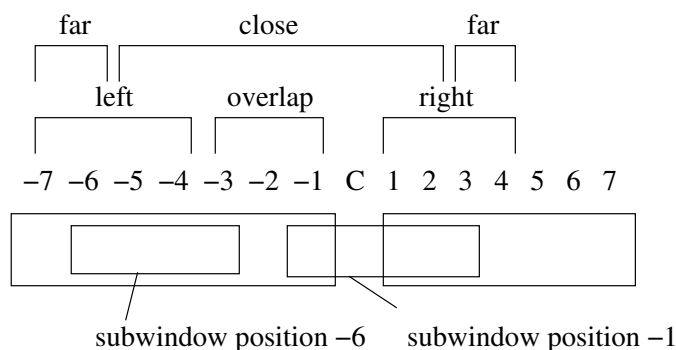


FIG. 4.4 – Valeur des attributs “position” en fonction de la position de début des sous-fenêtres (contexte de taille 14, sous-fenêtre de taille 4)

$$s = \text{small} = \text{true}(X, Y, \text{small}(Z))$$

$$t = \text{tiny} = \text{true}(X, Y, \text{small}(\text{tiny}))$$

Avec ces notations, $[\text{cccc} \wedge \text{ssss}]$ est une conjonction (représentée par le symbole \wedge) valide de 4-formules, vraie sur les mots *RDHE* et *AGNG* de l'exemple (aux positions 3 et 8 du contexte par rapport à la cystéine). *cccc* signifie que tous les acides aminés de *RDHE* sont *chargés* et *ssss* signifie que tous les acides aminés de *AGNG* sont petits.

Une autre conjonction valide de 4-modèles est $[(++\&--)\wedge(\text{hs}\&\text{tt})]$, vraie sur les mots *RDHE* et *AGNG* de l'exemple (aux positions -5 et +1 du contexte par rapport à la cystéine). $(++\&--)$ signifie que deux acides aminés du mot *RDHE* sont chargés positivement et les deux autres chargés négativement, sans préciser la position exacte de ces acides aminés.

On conserve par contre la position de la sous-fenêtre dans le contexte observé autour de la cystéine. Après essais préliminaires, garder la position exacte dans chaque sous-fenêtre du contexte mène à une sur-spécialisation des règles de production. C'est pourquoi nous avons choisi de ne retenir qu'un nombre limité de positions pour chaque sous-fenêtre, en terme d'attribut de positions. Les valeurs de positions dans nos expériences appartiennent à l'ensemble $\{\text{overlap}, \text{close}, \text{far}, \text{left}, \text{right}\}$ (illustré figure 4.4). *overlap* signifie que le modèle chevauche la cystéine. Les indices couverts par *overlap* correspondent aux indices du début du modèle. Pour des 4-modèles, *overlap* couvre les -3, -2 ou -1 car un 4-modèle commençant à la position -3 se termine à la position 1 et chevauche donc la cystéine. *close* représente un 4-modèle proche de la cystéine, *far* représente un 4-modèle situé loin de la cystéine (aux extrémités du contexte), *left*, resp. *right*, représente un 4-modèle situé dans le contexte gauche, resp. droit, de la cystéine. Ces notions sont codées dans la connaissance *a priori*.

Dans la suite nous appellerons *k-formule ancree* une k-formule dont on a précisé de cette façon la position dans la fenêtre.

```

InHerIt(instances, connaissances)
début
taille_clause = taille_min
bruit = bruit_max
tant que taille_clause ≤ taille_max faire
    hyp = generalise(instances, connaissances, bruit)
    connaissance = specialise(connaissances, hyp)
    taille_clause = taille_clause + 1
    bruit = bruit - B
fin tant que
fin

```

FIG. 4.5 – Algorithme InHerIt

L'espace d'hypothèses correspondant est énorme et nous devons limiter le nombre de conjonctions : étant donné que chaque acide aminé vérifie environ 4 propriétés élémentaires, avec des sous-fenêtres de taille 4 et 5 types de positions de sous-fenêtres différents, il existe 256 instanciations possibles de propriétés pour une sous-fenêtre donnée, 19 arrangements possibles de sous-séquences et $5 \cdot 256 \cdot 19 = 24320$ 4-formules ancrées possibles. Même si nous nous limitons à un maximum de 3 conjonctions, la taille de l'espace est donc de l'ordre de : $24320^3 \simeq 1.5 \times 10^{13}$.

Nous avons utilisé un algorithme général, que nous avons appelé InHerIt (Induce from Herbrand space Iteratively). Le pseudo code est illustré figure 4.5.

Nous proposons une méthode générale pour fortement diminuer la combinatoire dans ces espaces d'hypothèses logiques. Notre algorithme explore cet espace avec 2 niveaux de recherche. Le premier niveau tend à découvrir un sous-ensemble pertinent de l'univers de Herbrand, en analysant un sous-ensemble restreint des hypothèses. Le 2^e niveau recherche les formules dans la totalité de l'espace cible, en utilisant maintenant l'espace restreint de Herbrand. Au 2^e niveau, la connaissance *a priori* est enrichie avec la liste des modèles extraits des règles induites durant le premier niveau de recherche. Ainsi, le programme d'induction peut construire des règles plus discriminantes en prenant en compte les règles qu'il a trouvé au 1^e niveau.

Nous avons choisi d'utiliser l'algorithme 4.5 en trois phases. Le premier espace d'hypothèses est restreint à la fois au niveau k-formules (pas d'ancrage des formules) et au niveau de la taille des clauses produites (au plus 1 littéral dans le corps des clauses). Le deuxième espace contient des k-formules simples ou ancrées et la taille des clauses est d'au plus 2 littéraux. Le troisième espace contient des k-formules simples ou ancrées et la taille des clauses est d'au plus 3 littéraux. Le détail de la connaissance *a priori* que nous avons fournie dans le cadre de la prédiction de ponts cystéines est donné en l'annexe.

PROGOL offre la possibilité d'introduire du bruit, *i.e.* autoriser que les clauses générées couvrent un certain nombre d'exemples négatifs. Cet ajustement du bruit est très important pour notre recherche, car s'il n'y a pas d'ajustement de ce taux de bruit alors nous ne récolterions que des termes qui n'ont pas besoin d'être spécialisés. Autrement dit, on recueille à un niveau donné des hypothèses partielles (clauses de taille réduite), qui sont globalement pertinentes sur les instances positives mais insuffisamment discriminantes sur les instances négatives, puis on raffine les hypothèses à l'étape suivante en augmentant la discrimination et donc en diminuant le nombre d'instances négatives acceptable par diminution du niveau de bruit.

L'implémentation de la notion de sous-séquence est délicate et nous consacrons un paragraphe à expliquer la manière dont nous avons procédé.

Pratiquement, les sous-séquences de propriétés sont énumérées en extension en utilisant un prédicat *pattern* qui agit sur les sous-séquences de taille k . Ceci ne permet pas de traiter de grandes valeurs de k mais reste utile dans le cadre qui nous intéresse, où la valeur $k = 4$ s'est avérée suffisante. Pour $k=4$, nous avons défini plusieurs types de modèles possibles : *quatuor*, *quatuor1*, *quatuor2*, *quatuor3*, *quatuor4*, *triplet*, *triplet1*, et *pair*. Ces modèles permettent de couvrir toutes les k -formules possibles. Nous expliquons ces modèles ci-après.

Quatuor est juste un sous-mot de taille 4 et *quatuor1*, *quatuor2*, *quatuor3*, *quatuor4* introduisent exactement un caractère universel dans ce sous-mot ; *triplet* définit 2 sous-séquences de taille 3 et 1 et *triplet1* définit une sous-séquence de taille 3 et un caractère universel ; et enfin *pair* définit 2 sous-séquences de taille 2 et 2.

Exemple :

Ces notations sont utilisées sur les 4-modèles. Si s , p et h définissent respectivement les propriétés *small*, *polar* et *hydrophobic* :

- $quatuor(h,h,s,p)$ représente le modèle $hhsp$ et signifie que toutes les propriétés doivent apparaître dans cet ordre exact dans le contexte autour d'une cystéine
- $quatuor1(x,h,s,p)$ (resp. $quatuor2(h,x,s,p)$, $quatuor3(h,h,x,p)$ et $quatuor4(h,h,s,x)$) représente le modèle $xhsp$ et signifie que toutes les propriétés doivent apparaître dans cet ordre dans le contexte, incluant n'importe quelle valeur à la 1^e position (resp. à la 2^e, 3^e et 4^e position)
- $triplet(hhhs)$ représente le modèle $hhh&x$ et exige dans la sous-fenêtre de taille 4 du contexte à la fois une sous-séquence hydrophobe de taille 3 et un acide aminé *small*. Les arrangements possibles sont : $shhh$, $hshh$, $hhsh$ et $hhhs$.
- $triplet1(h,h,h)$ représente un modèle $hhh&x$ et exige dans une sous-fenêtre de taille 4 une sous-séquence hydrophobe de taille 3. Les arrangements possibles sont : $xhhh$, $hxhh$, $hhxh$ et $hhhx$.

- $pair(h, h, p, p)$ représente le modèle $hh&pp$ et signifie que dans la sous-fenêtre de taille 4, on trouve à la fois une sous-séquence hydrophobe de taille 2 et une sous-séquence polaire de taille 2. Les arrangements possibles sont $hhpp$, $hphp$, $hpqh$, pqh , $phqh$ et $pphh$

Durant la deuxième phase, la connaissance *à priori* est enrichie avec la liste des modèles extraite des règles induites. La définition de formules ancrées diffère de celle du simple modèle sur deux points : tout d'abord les modèles sont instanciés en utilisant l'espace de Herbrand de la phase 1. Ensuite, l'attribut de position de la sous-fenêtre type est calculée durant l'extraction de la sous-fenêtre.

PROGOL construit la clause la plus spécifique à partir d'un exemple en utilisant la règle d'inférence de la programmation logique, la *résolution*, pour produire toutes les conséquences découlant de l'exemple et de la théorie du domaine. Pour la recherche des hypothèses, il utilise un score. Cette fonction permet de décider quelle hypothèse est la meilleure pour la tâche d'apprentissage visée et s'appuie souvent sur la couverture des exemples par l'hypothèse. La fonction de score que nous utilisons est $score(h) = P - N - L$, où P est le nombre d'exemples positifs couverts par l'hypothèse h , N le nombre d'exemples négatifs couverts et L est la taille de la clause. Nous n'avons pas utilisé d'autres fonctions de score.

La section suivante présente les résultats que nous avons obtenus sur les deux sous problèmes de prédiction : prédiction de l'état d'oxydation, et prédiction de l'appariement des cystéines.

4.3 Résultats des expérimentations pour la prédiction de l'état d'oxydation des cystéines

Nous avons utilisé la version de PROGOL 4.4 [Muggleton, 1998]. Pour le 2e niveau de recherche, trois étapes ont été effectuées.

Phase1 : Rappelons tout d'abord les différentes contraintes imposées : nous avons autorisé PROGOL à trouver des règles pouvant couvrir des négatifs (bruit autorisé). Chaque règle peut couvrir de 0 à 8 négatifs. On force PROGOL à trouver des règles dont la longueur de leur corps est 1, c'est à dire contenant uniquement un 4-modèle (pas d'ancrage des fenêtres avec les attributs "position"). Nous avons obtenu 32 règles. Un exemple de ces règles est illustré ci-après :

$example(A) : -pattern1(A, lprop(quatuor(polar, h(other), h(aromatic), h(other))))$.
 $example(A) : -pattern1(A, lprop(quatuor(small, small, hydrophobic, charged)))$.
 $example(A) : -pattern1(A, lprop(quatuor3(c(negative), h(aliphatic), charged)))$.
 ...

Ici, $h(other)$ représente un acide aminé hydrophobe qui ne soit ni aliphatique, ni aromatique.

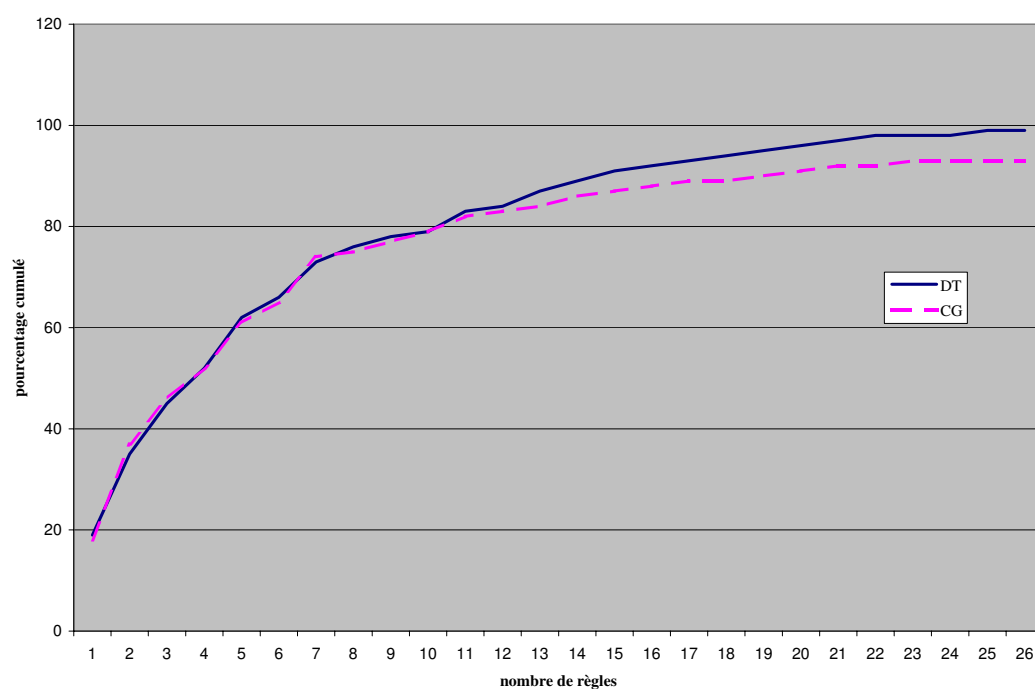


FIG. 4.6 – Taux cumulé de couverture des règles produites par InHerIt sur le problème de prédiction de l'état des cystéines sur la base de données d'apprentissage de D.Tessier (DT) et celle de validation de C.Geourjon (CG)

Pattern P3	%DT	DT	CG	%CG
h3hh3h1-close	19	137	168	18
sshc hh3ht	35	173	237	37
sshc ppc	45	162	215	46
hh2c1h-close	52	90	123	52
c2sss pchh	62	148	200	61
h3sc pph1	66	100	148	65
ph3h3h2 ppsh3	73	171	248	74
h2pc2p-close	76	43	51	75
hh2t-close	78	62	104	77
h3sc-left	79	139	182	79
h3h1h ssh3s	83	118	174	82
c1pc2p-close	84	43	58	83
shh3 sc1sp	87	139	149	84
h1tts	89	87	139	86
h1ssh3 ssh1h1	91	65	91	87
phhs phhs-far ptps	92	113	150	88
sc2ss tsc2s	93	101	132	89
cth2p sph	94	75	93	89
sh1pp-far	95	52	95	90
h2ph1h h3ps	96	70	79	91
h3hhh3 shh3-right spps	97	153	189	92
h1hts-left	98	86	121	92
tpps h2pp-close	98	53	68	93
h1sth	98	80	132	93
pph1s tpps-far spps	99	18	36	93
h1stc1-right	99	26	42	93

h=hydrophobic h1=h(aliphatic)
 h2=h(aromatic) h3=h(other)
 c=charged c1=c(positive)
 c2=c(negative) c3=c(other)
 p=polar s=small
 a=aliphatic t=tiny
 +=positive

%DT = taux d'exemples couverts avec la BD de D.Tessier
 DT = nombre d'exemples couverts dans la BD de D.Tessier
 CG = nombre d'exemples couverts dans la BD de C.Geourjon
 %CG = taux d'exemples couverts avec la BD de C.Geourjon

TAB. 4.1 – Modèles obtenus par PROGOL et leur couverture

Phase2 : Pour la 2^e phase, nous avons ajouté ces 32 règles à notre connaissance *a priori* et relancé PROGOL en restreignant le nombre de négatifs couverts (entre 0 et 6). Nous avons également introduit la notion de k-formules ancrées (c'est à dire en utilisant les attributs *position*). Le corps des règles est de longueur maximale 2. Nous avons obtenu 27 règles que nous avons aussi ajoutées à notre connaissance *a priori* pour la troisième étape. Un exemple de ces règles est donné ci-après :

example(A) : -pattern2(A, lprop(quatuor(polar, hydrophobic, hydrophobic, small), right)), pattern1(A, lprop(quatuor4(small, small, h(other))))).

example(A) : -pattern2(A, lprop(pair(hydrophobic, h(aromatic)), c(positive), hydrophobic), close).

example(A) : -pattern2(A, lprop(quatuor(small, small, hydrophobic, charged), close)), pattern1(A, lprop(pair(tiny, charged, polar, hydrophobic))).

...

Phase3 : Pour cette dernière phase, le bruit n'est plus autorisé, ce qui signifie que les règles trouvées par PROGOL ne doivent couvrir aucun exemple négatif et peuvent être composées des règles précédemment obtenues lors de la 1^e et 2^e étape, et/ou de nouvelles. Nous avons obtenu 26 règles décrites table 4.1. Cette table contient 5 colonnes. La première représente les règles construites par PROGOL, une conjonction de 4-modèles typés et de 4-modèles. La deuxième colonne représente le taux de couverture cumulé par les règles sur nos exemples positifs de la base de données de D. Tessier. La troisième colonne représente le nombre d'exemples positifs couverts par chaque règle dans notre base de données. Les quatrième et cinquième colonnes sont identiques à la 2^e et première colonne mais sur la base de données de C. Geourjon. Les courbes comparant les taux cumulés de couverture des règles sont représentées figure 4.6. La base de données de C. Geourjon est utilisée pour la validation de nos expériences. Elle contient 950 exemples positifs tandis que la notre n'en contient que 722. Ces 2 bases de données n'ont aucun exemple en commun. Les règles du tableau sont écrites de manière simplifiées. Par exemple, les trois premières règles correspondent aux règles PROGOL suivantes :

example(A) : -pattern2(A, lprop(quatuor(h(other), hydrophobic, h(other), h(aliphatic)), close)).

example(A) : -pattern3(A, lprop(quatuor(small, small, hydrophobic, charged))), pattern1(A, lprop(triplet4(hydrophobic, h(other), hydrophobic, tiny))).

example(A) : -pattern3(A, lprop(quatuor(small, small, hydrophobic, charged))), pattern1(A, lprop(quatuor3(polar, polar, charged))).

La première règle obtenue par PROGOL contient le type *close*, ce qui signifie que ce modèle est proche de la cystéine. Ce modèle traduit essentiellement le fait que la cystéine est enfouie dans un contexte aqueux car son contexte proche ne contient que des acides aminés hydrophobes. Il couvre quasiment le même pourcentage d'exemples dans

les deux bases de données soit près d'un cinquième de la base, ce qui est considérable. La plupart des règles décrivent des contextes hydrophobes et contiennent également de petits acides aminés. Cette propriété semble favorable aux ponts disulfures car la protéine est plus flexible avec de petits atomes qu'avec de plus gros. Nous pouvons remarquer qu'il suffit de 19 règles pour couvrir 90% des bases de données, ce qui est un excellent résultat en ce qui concerne la simplicité de la modélisation. De plus, les règles montrent que les cystéines oxydées ont davantage de contacts avec des acides aminés polaires ou chargés.

Un des avantages de la programmation logique inductive est de fournir des règles directement interprétables. Ce n'est pas le cas pour la plupart des autres méthodes d'apprentissage telles que les SVM, les réseaux de neurones, les HMM, etc. Comparons, par exemple, nos résultats avec les SVM.

Les machines à noyau, et en particulier les machines à vecteur de support (SVM), sont motivées par le principe de minimisation du risque structurel de Vapnik. Ce principe est basé sur la minimisation conjointe des deux causes d'erreur : le risque empirique et l'intervalle de confiance. Dans le cas le plus simple, un algorithme d'apprentissage SVM commence à partir d'une représentation vecteurs-exemples des points de données et cherche un hyperplan séparateur ayant une distance maximale de l'ensemble des données, on dit aussi que cet hyperplan maximise la marge. Plus généralement, quand les exemples ne sont pas des vecteurs linéairement séparables, l'algorithme les représente dans un espace cible de grande dimension où ils sont quasiment tous linéairement séparables. Ceci se fait au moyen d'une fonction à noyau qui calcule le point produit par les images de deux exemples dans l'espace cible. La fonction de décision associée à un SVM est basée sur le signe de la distance à partir de l'hyperplan séparateur :

$$f(x) = \sum_{i=1}^N y_i \alpha_i K(x, x_i) \quad (4.1)$$

où x est le vecteur d'entrée, $\{x_1, \dots, x_n\}$ est l'ensemble des vecteurs support, $K(.,.)$ est la fonction noyau, et y_i est la classe du i^e vecteur support (+1 ou -1 pour les instances positives et négatives respectivement).

Les résultats de ces méthodes sont exprimés sous forme de probabilité. Nos règles sont, quand à elles, explicites et permettent de traduire le pourquoi de l'oxydation des cystéines au moyen de modèles caractéristiques des contextes oxydés. Par exemple, la règle : *example(A) : -pattern3(A, lprop(quatuor(small, small, hydrophobic, charged)))* peut être interprétée avec un minimum de connaissance sur le modèle. Cette règle met en évidence le fait qu'une cystéine qui contient, dans son contexte, un modèle constitué de 2 *petits* atomes suivis d'un acide aminé *hydrophobe* puis d'un acide aminé *chargé*, est oxydée. Le modèle ne se contente pas renvoyer une réponse booléenne mais il indique aussi pourquoi telle cystéine est oxydée.

En ce qui concerne la prédiction de l'appariement des cystéines oxydées, l'idée est d'introduire les règles obtenues pour la prédiction de l'oxydation des cystéines dans notre connaissance *a priori*. Ce problème est très difficile car il fait intervenir des relations longues distances dans les séquences protéiques. Le point fort des travaux effectués dans ce chapitre est que nous avons généré de la connaissance supplémentaire pour résoudre le problème de prédiction de l'appariement des cystéines oxydées. Ces règles serviront de point de départ pour de futurs travaux sur la prédiction de l'appariement.

4.4 Perspectives pour la prédiction de l'appariement des cystéines oxydées

Nous présentons dans cette section diverses perspectives pour la prédiction de l'appariement des cystéines oxydées. Nous proposons un échantillon d'apprentissage et divers relations supplémentaires sur les propriétés physico-chimiques des acides aminés. Les premiers résultats obtenus sont prometteurs et sont présentés en fin de section.

4.4.1 Constitution de l'échantillon pour la prédiction de l'appariement des cystéines oxydées

Un exemple positif de notre échantillon est constitué des deux fenêtres de contextes disulfures. Il se présente sous la forme :

$$\text{example}(\text{context1}([I, T, P, V, N, A, T, A, I, R, H, P, C, H]), \\ \text{context2}([S, N, V, L, C, R, L, N, K, Y, R, V, G, H])).$$

qui représente le fait que la cystéine oxydée observée dans le contexte1 crée un pont disulfure avec la cystéine oxydée observée dans le contexte2.

Les exemples négatifs contiennent aussi deux fenêtres de contexte mais formant de faux appariements.

Ainsi, une protéine ayant la conformation suivante de ponts disulfures : $C1_1C2_1C2_2C1_2$, *i.e.* la 1^{re} (resp. 2^e) cystéine crée un pont disulfure avec la 4^e (resp. 3^e) cystéine de la protéine, va donner naissance aux instances négatives $C1_1C2_1$, $C1_1C2_2$, $C2_1C1_2$. Les instances négatives sont créées entre cystéines appartenant à la même protéine.

Nous avons ainsi créé 260 instances positives et 1629 instances négatives.

Nous avons également créé un échantillon précisant une structure secondaire pour les cystéines analysées afin d'obtenir des motifs caractéristiques dépendant de la structure secondaire des cystéines. L'ajout de cette connaissance est réaliste. En effet, les programmes de prédiction de structures secondaires donnent de bons résultats. De plus, même dans le cas de résultat incorrect, ce qui importe est la cohérence avec la classification obtenue pour une nouvelle instance. En effet, nous allons ensuite utiliser le même

programme pour étiqueter les nouvelles instances et nous conservons ainsi la même logique d'étiquetage. En particulier, les instances d'apprentissage seront correctement reconnues. Pour cela, nous avons utilisé le serveur SOPMA³ [Geourjon et Deleage, 1995], qui est un serveur de prédiction de structure secondaire des protéines utilisant des méthodes consensus de prédiction à partir d'un alignement multiple. Il prédit chaque acide aminé dans un des états suivants : {H,E,C,T}, où H représente les hélices, E les feuillettes, C les boucles et T les tours.

Pour chacune de nos protéines nous avons récupéré et conservé uniquement la structure secondaire de nos cystines. Cette structure a été insérée dans nos exemples de la façon suivante :

```
example(context1([I,T,P,V,N,A,T,A,I,R,H,P,C,H]),coil,
         context2([S,N,V,L,C,R,L,N,K,Y,R,V,G,H]),helix).
```

Cet exemple signifie que la cystine appartenant au premier contexte se situe dans une boucle et que la cystine appartenant au deuxième contexte se situe dans une hélice.

4.4.2 Connaissance préalable

Nous avons utilisé les mêmes connaissances que pour la prédiction de l'état oxydé des cystéines sur les propriétés physico-chimiques des acides aminés. Ce que nous cherchons à obtenir est un ensemble de règles faisant apparaître des k-formules dans les deux contextes *contexte1* et *contexte2*. Pour cela nous avons introduit les règles obtenues lors de la prédiction de l'état des cystéines oxydées (règles de l'étape 1 à 3). Ainsi, PROGOL peut directement utiliser ces clauses au lieu de les recalculer.

Nous avons de plus introduit des types de relation entre les k-formules qui traduisent la nature des liaisons potentielles établissant dans l'environnement des ponts disulfures. Elles sont donc cruciales pour la prédiction des appariements. Ainsi, nous avons défini les relations suivantes : *identique2*, *identique3*, *opposé2*, *opposé4* :

- *identique2* = 2 propriétés identiques dans le contexte ;
- *identique3* = 3 propriétés identiques dans le contexte ;
- *opposé2* = 1 propriété et sa négation sont observées dans un même contexte ;
- *opposé4* = 2 propriétés et sa négation sont observées dans un même contexte.

En ce qui concerne l'espace, nous avons séparé des termes décrivant le contexte en 2 parties, les termes instanciés et les termes variables. Les termes variables permettent de spécifier non pas la présence d'une propriété physico-chimique donnée mais une nature de relation entre deux propriétés à deux positions dans les contextes. Ceci apporte une grande souplesse à l'expression des propriétés.

Par exemple, on pourra décrire une liaison électrostatique de type charge positive/charge négative par une propriété générale *opposé2* définie précédemment.

³http://npsa-pbil.ibcp.fr/cgi-bin/secpred_sopma.pl

Nous avons également défini la symétrie des contextes, *i.e.* l'ordre des contextes des deux cystéines n'a pas d'importance dans la caractérisation.

Les règles jusqu'ici obtenues lors de nos expérimentations ne permettent pas encore de distinguer de manière générale l'appariement des cystéines. Nous donnons un exemple des règles obtenues et leur couverture :

```
[C:-799,24,94,0 example(A,B):-bridge(A,B,C,quatuor,D,quatuor,
fixed(1-h(aliphatic),2-small,3-tiny,6-hydrophobic),same2(hydrophobic)),
typepos(C,left),typepos(D,left).]
```

```
[C:-799,24,94,0 example(A,B):-bridge(A,B,C,quatuor,D,quatuor,
fixed(1-h(aliphatic),2-small,6-hydrophobic,7-tiny),same2(hydrophobic)),
typepos(C,left),typepos(D,left).]
```

```
[C:-799,24,94,0 example(A,B):-bridge(A,B,C,quatuor,D,quatuor,
fixed(1-h(aliphatic),3-tiny,4-small,7-hydrophobic),same2(hydrophobic)),
typepos(C,left),typepos(D,left).]
```

```
[C:-1005,48,230,0 example(A,B,C,D):-bridge(A,B,C,D,E,quatuor,F,quatuor,
fixed(1-h(aliphatic),2-small,3-tiny,5-polar),same2(hydrophobic)),
secondarystruct(G,helix),bridge(A,G,C,D,E,quatuor,H,quatuor2,
fixed(1-h(aliphatic),2-small,4-hydrophobic,6-small),null).]
```

```
[C:-1005,48,230,0 example(A,B,C,D):-bridge(A,B,C,D,E,quatuor,F,quatuor,
fixed(1-h(aliphatic),2-small,3-tiny,5-polar),same2(hydrophobic)),
secondarystruct(G,helix),bridge(A,G,C,D,E,quatuor,H,quatuor2,
fixed(1-h(aliphatic),2-small,5-small,7-hydrophobic),null).]
```

```
[C:-1005,48,230,0 example(A,B,C,D):-bridge(A,B,C,D,E,quatuor,F,quatuor,
fixed(1-h(aliphatic),2-small,3-tiny,5-polar),same2(hydrophobic)),
secondarystruct(G,helix),bridge(A,G,C,D,E,quatuor,H,quatuor2,
fixed(1-h(aliphatic),4-hydrophobic,5-small,7-small),null).]
```

Le premier chiffre associé à ces règles est le score attribué par PROGOL. Ces scores sont ici négatifs ce qui montre une faiblesse au niveau des règles générées. Le 2^e nombre est le nombre d'exemples positifs couverts et le troisième est le nombre d'exemples négatifs couverts. Nous remarquons que ces règles couvrent davantage d'exemples négatifs que positifs. Néanmoins, nous disposons d'un échantillon contenant 260 exemples positifs et 1629 exemples négatifs, ce qui représente environ 6,26 fois plus d'exemples négatifs que positifs. Si nous comparons ces chiffres aux taux de couvertures de nos règles, pour l'échantillon ne contenant pas la structure secondaire des cystéines, les règles couvrent environ 3,91 fois plus d'exemples négatifs que positifs, ce qui est un meilleur rapport qu'initialement. Il en est de même

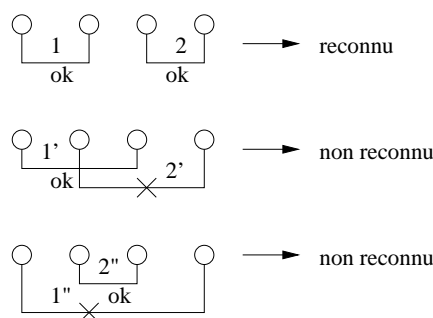


FIG. 4.7 – Configurations de ponts

pour l'échantillon contenant la structure secondaire des cystéines car ce taux est de 4,79.

Aujourd'hui, nous cherchons d'autres types de relations possibles afin de raffiner notre recherche et d'améliorer les scores obtenus par PROGOL. Une des dernières règles obtenues en introduisant des relations supplémentaires est la suivante :

```
[C:-361,15,32,0 example(A,B) :- bridge(A,B,C,quatuor,D,quatuor,
fixed(1-h(other),2-h(other),4-not(small),7-h(aromatic)),
oppositesame4(not(charged),hydrophobic)),
typepos(C,close),typepos(D,right).]
```

On peut remarquer que cette règle couvre 15 instances positives pour 32 instances négatives, le rapport entre les instances positives et négatives est donc maintenant de 2. Ce qui est très encourageant.

Remarque

Nous pouvons distinguer des faiblesses de trois ordres :

- l'espace des relations que nous avons imaginé reste limité ;
- la taille de la clause la plus spécifique et la taille de l'espace sont trop grandes. Ce qui nous oblige à explorer l'espace partiellement ;
- prise en compte des ponts un à un.

En ce qui concerne le dernier point, nous pensons qu'une cystéine peut créer des ponts disulfures avec plusieurs autres cystéines. Le fait qu'elle s'apparie au final avec une seule cystéine est probablement dû à la conformation globale de la protéine. La figure 4.7 illustre ce phénomène. Le premier schéma illustre la configuration correcte de deux ponts disulfures dans une protéine. Le deuxième schéma montre que la première cystéine peut s'apparier à la troisième cystéine mais l'appariement de la 2^e et de la quatrième cystéine ne peut pas se faire pour des raisons physico-chimiques ou autres.

Le dernier schéma illustre le fait que les deuxièmes et troisièmes cystéines peuvent s'apparier mais les deux autres ne le peuvent pas. Il se pourrait que la configuration finale des ponts disulfures soit celle qui favorise la création du maximum de ponts. Ce qui signifie que nos exemples négatifs peuvent être des exemples positifs. Ce qui fait que l'apprentissage est faussé. Pour cela il faudrait distinguer les exemples en fonction de la protéine à laquelle ils appartiennent.

En conclusion, nous avons obtenu un bon taux de prédiction des cystéines oxydées en générant des règles interprétables pour les biologistes. Seulement 19 règles sont nécessaires pour couvrir plus de 90% de nos bases de données d'apprentissage et de validation. En ce qui concerne la prédiction de l'appariement des cystéines oxydées, l'idée d'introduire ces règles dans notre connaissance *a priori* semble être une bonne piste. Les travaux effectués dans cette thèse serviront de point de départ pour de futurs travaux sur la prédiction de l'appariement car nous avons généré de la connaissance supplémentaire pour résoudre le problème de prédiction de l'appariement des cystéines oxydées. Nous avons proposé diverses perspectives de résolution de la prédiction de l'appariement des cystéines oxydées et présenté les premiers résultats obtenus. Ces résultats sont prometteurs car le rapport entre le nombre d'instances positives et négatives couvertes par les règles obtenues jusqu'à présent progresse dans le bon sens. Ce problème est très complexe car il fait intervenir des relations longues distances dans les séquences protéiques qu'il est difficile de prendre en compte dans les modèles.

Chapitre 5

Bilan et discussion

5.1 Synthèse

Le but premier des travaux présentés dans cette thèse était de formaliser un problème bioinformatique de prédiction de structures en terme d'apprentissage symbolique et d'étudier l'applicabilité des méthodes actuelles d'apprentissage de relations. Le problème auquel nous nous sommes intéressés est la prédiction des ponts disulfures dans les protéines. Ce problème est complexe car il fait intervenir des interactions longue distance dans les séquences et qui peuvent se croiser de manière arbitraire. Pour résoudre ce problème nous nous sommes placé dans le cadre formel de l'apprentissage symbolique supervisé et utilisé successivement deux méthodes non explorées jusqu'à présent dans la résolution de ce problème, l'inférence grammaticale et la programmation logique inductive.

Pour résoudre notre problème grâce à l'inférence grammaticale nous nous sommes inspirés de l'article de Takada [Takada, 1995] qui permet de passer d'un problème algébrique à un problème régulier. Pour cela, nous avons modélisé l'ensemble des appariements des ponts possibles au moyen d'une grammaire algébrique universelle. Il a fallu trouver le meilleur jeu de règles de production couvrant l'ensemble des exemples positifs de notre échantillon. Divers choix ont été réalisés allant d'une grammaire algébrique universelle simple, *i.e.* reconnaissant chaque acide aminé et un type de pont unique, à une grammaire algébrique universelle plus complexe prenant en compte la structure des protéines ou les propriétés physico-chimiques des acides aminés. Au final, les résultats obtenus par les algorithmes d'inférence régulière pour la prédiction de nos ponts disulfures dans les protéines n'ont pas été satisfaisants, alors qu'ils donnent des résultats intéressants sur des automates aléatoires. Afin de comprendre les difficultés rencontrées au cours de notre apprentissage nous avons effectué une série d'expérimentations qui a permis de mettre en évidence les failles de ces algorithmes.

Nous avons tout d'abord montré que la taille des séquences de l'échantillon d'apprentissage est un des plus importants facteurs influençant les résultats obtenus par nos

algorithmes. Ceci est un inconvénient majeur car nous travaillons sur des séquences de plus de 100 caractères, pouvant même aller jusqu'à plus de 1000 caractères.

Le deuxième problème que nous avons mis en évidence est l'influence de la taille de l'alphabet sur la qualité des résultats. En effet nous travaillons sur un alphabet ayant plus de vingt symboles, or les algorithmes d'inférence régulière sont performants sur un alphabet de deux symboles. Travailler ici sur un tel alphabet est impossible car nous travaillons sur l'enchaînement des règles de production de la grammaire, ce qui impliquerait de n'avoir que deux règles de production dans notre grammaire.

Grâce à ces expériences, nous avons montré que les algorithmes d'inférence régulière actuels ne sont pas adaptés à notre problème de prédiction des ponts disulfures. Nous proposons néanmoins dans la partie perspective quelques pistes de recherche possibles qui nous font penser que les méthodes d'inférence grammaticale méritent d'être étudiées davantage.

Nous nous sommes ensuite tournés vers la programmation logique inductive. Cette approche permet de travailler sur des instances de grande taille et ne se limite pas à un alphabet particulier. De plus, l'introduction facile de connaissance *a priori* dans le modèle permet de prendre en compte des relations entre les données. Le but est, dans un premier temps, d'inférer des règles caractéristiques de l'état oxydé des cystéines. Ces règles sont ensuite utilisées pour en acquérir de nouvelles sur l'appariement de ces cystéines.

Cette approche permet d'obtenir de bons résultats sur la prédiction de l'état oxydé des cystéines et les règles inférées fournissent une définition opérationnelle et interprétable par des biologistes. Cette interprétabilité, ainsi que l'efficacité de la phase d'inférence, sont assurées en pratique par les adaptations de l'algorithme de PLI afin de l'adapter à nos besoins spécifiques que nous avons effectuées. De plus, la souplesse d'utilisation de la PLI permet, à travers l'utilisation d'exemples adaptés, de s'intéresser à l'acquisition de tout type d'information structurale. Il ressort de cette étude que les règles caractéristiques de l'état oxydé des cystéines contiennent principalement des acides aminés hydrophobes et de petite taille. Ces règles sont ensuite utilisées pour caractériser l'appariement des cystéines entre elles. Nous avons proposé un algorithme d'induction heuristique qui peut être utilisé dans différents problèmes. L'idée d'effectuer plusieurs phases d'apprentissage en tenant compte des résultats obtenus aux phases précédentes permet de diminuer considérablement la combinatoire dans les espaces d'hypothèses logiques car à chaque étape, notre algorithme construit des règles de plus en plus discriminantes.

Cette deuxième prédiction est toujours en cours d'expérimentation car les règles obtenues jusqu'à présent ne permettent pas de distinguer des motifs caractéristiques d'appariement. Néanmoins, l'introduction de la connaissance de la structure secondaire de nos protéines semble améliorer les scores des règles obtenues. Ces résultats sont encourageants et montrent l'importance de l'introduction de connaissances supplémentaires.

Un des inconvénients majeurs est que les règles que nous obtenons couvrent trop

d'instances négatives. Ceci est principalement dû au fait que nous n'effectuons pas de prédiction globale.

5.2 Perspectives

De nombreuses perspectives sont ouvertes sur différents aspects de notre travail, aussi bien au niveau de l'inférence grammaticale que de la programmation logique inductive.

En ce qui concerne l'inférence grammaticale, la conception de nouveaux algorithmes et de nouvelles heuristiques d'inférence régulière est importante pour pouvoir ensuite aborder l'inférence de classes de langages plus complexes, via l'apprentissage de leur langage de contrôle. Il serait intéressant de pouvoir interdire certaines fusions d'états ou tout du moins de pouvoir les retarder. Dans le cas de notre application bioinformatique, supposons que l'on appelle *graphe de propriété X* un graphe dont les arcs sont étiquetés par des acides aminés dont les propriétés physico-chimiques appartiennent au même groupe X. On pourrait fusionner prioritairement les sous-graphes de même propriété. Une autre idée serait de fusionner prioritairement des sous-graphes dont les acides aminés représentés sur les arcs appartiennent à la même structure secondaire. Bien sûr, ceci implique d'avoir accès à de la connaissance supplémentaire sur les données. Les travaux [Leroux, 2005] et [Coste et Kerbellec, 2005] effectués en parallèle de ceux-ci abordent ce problème. Il serait donc intéressant de les utiliser pour notre prédiction de ponts disulfures.

En ce qui concerne la programmation logique inductive, une des difficultés est le réglage de la fonction de score utilisée par PROGOL, lorsque l'échantillon est très déséquilibré. La fonction de score que nous avons utilisée est $P - N - L$ où P représente le nombre d'exemples positifs, N le nombre d'exemples négatifs et L la taille du corps des clauses. Il serait intéressant de trouver une fonction de score plus adaptée car nous travaillons sur un échantillon contenant sept fois plus d'exemples négatifs que positifs. Ceci pourrait être pris en considération par notre fonction de score.

À l'issue des expériences présentées dans le dernier chapitre de ce mémoire, de nombreuses perspectives restent également ouvertes. Tout d'abord, il serait intéressant de disposer d'une autre base de données contenant des protéines issues du même milieu afin de pouvoir tester nos règles sur des exemples positifs et négatifs. En effet, la création des ponts disulfures dépend également du milieu dans lequel se trouvent les cystéines. Bénéficier d'un ensemble de protéines issues du même milieu permet de considérer des exemples négatifs relatif au milieu.

En ce qui concerne l'appariement des cystéines oxydées, il serait intéressant de ne pas considérer les ponts un à un mais d'expérimenter globalement en supposant que toutes les cystéines oxydées sont impliquées, si un seul des appariements ne peut être mis en évidence, alors on rejète globalement l'instance.

Une idée originale serait de coupler les deux méthodes d'apprentissage utilisées dans cette thèse. En effet, nous pourrions effectuer la prédiction de l'état oxydé des cystéines en utilisant l'inférence grammaticale, et la prédiction de l'appariement des cystéines oxydées en utilisant la programmation logique qui permettrait de prendre en compte des relations entre les données.

L'essentiel de notre méthode a porté sur la recherche des ponts disulfures. Il serait maintenant intéressant d'appliquer notre démarche à la prédiction d'autres structures biologiques, afin d'en tirer des enseignements généraux sur les atouts et limites de ces méthodes, et de mettre en place une méthodologie générale.

Annexe

Connaissance *a priori*

```
% tête des règles
:- modeh(1,example(+context))?

%Phase 1
% déclaration du format des règles que l'on souhaite obtenir à la phase 1.
% corps des règles de taille 1. On souhaite un modèle appelé pattern1
% contenant une liste de propriétés physico-chimiques dans un contexte.
%:- modeb(600,pattern1(+context,#list_property)), set(c, 0), set(noise, 5)?

%Phase 2
% déclaration du format des règles que l'on souhaite obtenir à la phase 2.
% corps des règles de taille 2. On souhaite une conjonction de modèles
% appelés pattern1 et pattern2 contenant une liste de propriétés
% physico-chimiques dans un contexte. Les pattern1 peuvent être ceux obtenus
% lors de la phase 1. La liste de propriétés physico-chimiques dans les pattern2
% est typée par des attributs positions(voir p90).
%:- modeb(300,pattern2(+context,#typed_list_property)),
    modeb(600,pattern1(+context,#list_property)),
    set(c, 1), set(noise, 3), unset(memoing), consult(groundpattern1)?

%Phase 3
% déclaration du format des règles que l'on souhaite obtenir à la phase 3.
% corps des règles de taille 3. On souhaite une conjonction de modèles
% appelés pattern1, pattern2 et pattern3 contenant une liste de
% propriétés physico-chimiques dans un contexte. Les pattern1 et pattern2
% peuvent être ceux obtenus lors des deux phases précédentes.
:- modeb(400,pattern3(+context,#list_property)),
    modeb(300,pattern2(+context,#typed_list_property)),
    modeb(600,pattern1(+context,#list_property)),
    set(c, 2), set(noise, 1), unset(memoing),
    consult(groundpattern1),consult(groundpattern2)?
```

```

:- set(i, 4)?
:- set(h, 1000)?
:- set(r, 240000)?
:- set(nodes, 250000)?
:- set(cover, off)?

context(context(_)).
list_property(lprop(_)).
typed_list_property(lprop(_,_)).

%Phase 1
% définition du format de pattern1
pattern1(context(Context), lprop(TupleProperties)):-
    subword4(Context, SubContext),
    property_letters(SubContext,TupleProperties).

%Phase 2
% définition du format de pattern2
% ground_pattern2 must be defined in file groundpattern2.pl
% ground_pattern2 is generated after the 2nd phase of induction of the algorithm
% ground_pattern2(P) for all P
% such that pattern2(A,P) in best induced rules during Phase 2
pattern2(context(Context), lprop(Pattern,Type)):-
    ground_pattern1(Pattern),
    typed_subword4(Context,[-7,-6,-5,-4,-3,-2,-1,1,2,3,4], SubContext,Type),
    property_letters(SubContext, Pattern).

%Phase 2
typed_subword4([A,B,C,D|_], [N|LN], [A,B,C,D],T):- type(N,T).
typed_subword4([_|L], [_|LN], Subword,T):- typed_subword4(L, LN, Subword, T).

%Phase 3
% définition du format de pattern3
% ground_pattern1 must be defined in file groundpattern1.pl
% ground_pattern1 is generated after the 1st phase of induction of the algorithm
% ground_pattern1(P) for all P
% such that pattern1(A,P) in best induced rules during Phase 1
pattern3(context(Context), lprop(Pattern)):-
    ground_pattern2(Pattern,_),
    subword4(Context, SubContext),
    property_letters(SubContext, Pattern).

%Phase 3
subword4([A,B,C,D|_], [A,B,C,D]).

```

```

subword4([_|L],Subword):- subword4(L,Subword).

% prédicat 'prune' élagant l'espace de recherche
% Rules for Phase3 are with general form pattern3(A,X), pattern2(A,Y), pattern1(A,Z)
prune(_,pattern1(_,_)):-!.
prune(_,(_,pattern1(_,_),pattern1(_,_)):-!.
prune(_,(_,pattern1(_,_),pattern2(_,_)):-!.
prune(_,(pattern2(_,_),pattern2(_,_),pattern2(_,_)):-!.
prune(_,(pattern2(_,_),pattern1(_,_)):-!.    %deja generé phase2
prune(_,(_,_,pattern3(_,_)):-!.
prune(_,(_,pattern3(_,_)):-!.

%déclaration des attributs positions (p90)
type(-7, far).
type(-7, left).
type(-6, far).
type(-6, left).
type(-5, close).
type(-5, left).
type(-4, close).
type(-4, left).
type(-3, close).
type(-3, overlap).
type(-2, close).
type(-2, overlap).
type(-1, close).
type(-1, overlap).
type(1, close).
type(1, right).
type(2, close).
type(2, right).
type(3, far).
type(3, right).
type(4, far).
type(4, right).

property_letters(SubContext,AnalysedTuple):-
    properties_letters(SubContext,Properties),!,
    pattern(AnalysedTuple,Properties).

properties_letters([Letter|Letters],[LetterProperties|RProperties]):-
    properties_letter(Letter,LetterProperties),!,
    properties_letters(Letters,RProperties).

```

```
properties_letters([], []).
```

```
%properties are coded in a tree, with constants or variables
%[hydrophobic(other/aliphatic/aromatic),
% polar(other/charged(positive/negative)),
% small(other/tiny)]
```

```
% déclaration des propriétés physico-chimiques selon le diagramme de Taylor (p66)
```

```
property(hydrophobic, [hydrophobic(_) | _]).
property(polar,      [_ , polar(_) , _]).
property(small,     [_ , _ , small(_)]).
property(charged,   [_ , polar(charged(_)) , _]).
property(tiny,      [_ , _ , small(tiny)]).
property(h(P),      [hydrophobic(P) | _]).
property(c(P),      [_ , polar(charged(P)) , _]).
property(not(hydrophobic), [not_hydro | _]).
property(not(polar),    [_ , not_polar , _]).
property(not(charged),  [_ , not_polar , _]).
property(not(charged),  [_ , polar(not_charged) , _]).
property(not(small),    [_ , _ , not_small]).
```

```
% déclaration des propriétés physico-chimiques de chaque acide aminé
```

```
properties_letter(a, [hydrophobic(not_ali_aro), not_polar, small(tiny)]).
properties_letter(c, [hydrophobic(not_ali_aro), not_polar, small(not_tiny)]).
properties_letter(d, [not_hydro, polar(charged(negative)), small(not_tiny)]).
properties_letter(e, [not_hydro, polar(charged(negative)), not_small]).
properties_letter(f, [hydrophobic(aromatic), not_polar, not_small]).
properties_letter(g, [hydrophobic(not_ali_aro), not_polar, small(tiny)]).
properties_letter(h, [hydrophobic(aromatic), polar(charged(positive)), not_small]).
properties_letter(i, [hydrophobic(aliphatic), not_polar, not_small]).
properties_letter(k, [hydrophobic(not_ali_aro), polar(charged(positive)), not_small]).
properties_letter(l, [hydrophobic(aliphatic), not_polar, not_small]).
properties_letter(m, [hydrophobic(not_ali_aro), not_polar, not_small]).
properties_letter(n, [not_hydro, polar(not_charged), small(not_tiny)]).
properties_letter(p, [not_hydro, not_polar, small(not_tiny)]).
properties_letter(q, [not_hydro, polar(not_charged), not_small]).
properties_letter(r, [not_hydro, polar(charged(positive)), not_small]).
properties_letter(s, [not_hydro, polar(not_charged), small(tiny)]).
properties_letter(t, [hydrophobic(not_ali_aro), polar(not_charged), small(not_tiny)]).
properties_letter(v, [hydrophobic(aliphatic), not_polar, small(not_tiny)]).
properties_letter(w, [hydrophobic(aromatic), polar(not_charged), not_small]).
properties_letter(y, [hydrophobic(aromatic), polar(not_charged), not_small]).
```

```

% définition des différents types de modèles que l'on souhaite obtenir
%One subsequence of size 4 : 1 case
pattern(quatuor(A,B,C,D), [LA, LB, LC, LD]) :-
    property(A, LA), property(B, LB), property(C, LC), property(D, LD).
%One subsequence of size 3 : 4 cases
pattern(quatuor1(B,C,D), [_LA, LB, LC, LD]) :-
    property(B, LB), property(C, LC), property(D, LD).
pattern(quatuor2(A,C,D), [LA, _LB, LC, LD]) :-
    property(A, LA), property(C, LC), property(D, LD).
pattern(quatuor3(A,B,D), [LA, LB, _LC, LD]) :-
    property(A, LA), property(B, LB), property(D, LD).
pattern(quatuor4(A,B,C), [LA, LB, LC, _LD]) :-
    property(A, LA), property(B, LB), property(C, LC).
%One subsequence of size 3 and one of size 1: 4 cases
pattern(triplet4(A1,A2,A3,B), [L1,L2,L3,L4]) :-
    (property(A1,L1),
     (
      (property(A2,L2),
       ( (property(A3,L3), property(B,L4));
         (property(A3,L4), property(B,L3))
        )
      )
     );
    (property(B,L2), property(A2,L3), property(A3,L4))
   )
);
(property(B,L1), property(A1,L2), property(A2,L3), property(A3,L4)).
%One subsequence of size 3 : 4 cases
pattern(triplet3(A1,A2,A3), [L1,L2,L3,L4]) :-
    (property(A1,L1),
     (
      (property(A2,L2),
       ( property(A3,L3); property(A3,L4)
        )
      );
     (property(A2,L3), property(A3,L4))
    )
);
(property(A1,L2), property(A2,L3), property(A3,L4)).

%Two subsequences of size 2 : 6 cases
pattern(pair(A1,A2,B1,B2), [L1,L2,L3,L4]) :-
    (property(A1,L1),
     (
      (property(A2,L2), property(B1,L3), property(B2,L4));
      (

```

```

        property(B1,L2),
        (
            (property(A2,L3),property(B2,L4));
            (property(B2,L3),property(A2,L4))
        )
    )
);
(
    property(B1,L1),
    (
        (
            property(A1,L2),
            (
                (property(A2,L3),property(B2,L4));
                (property(B2,L3),property(A2,L4))
            )
        )
    );
    (property(B2,L2),property(A1,L3),property(A2,L4))
)
).

```

% Positive Examples

```

example(context([i,t,p,v,n,a,t, a,i,r,h,p,c,h])).
example(context([t,c,a,i,r,h,p, h,g,n,l,m,n,q])).
example(context([p,n,n,l,d,k,l, g,p,n,v,t,d,f])).
...

```

% Negative examples

```

:-example(context([s,d,k,v,g,q,a, c,r,p,v,a,f,d])).
:-example(context([s,h,m,e,e,d,p, e,c,k,s,i,v,k])).
:-example(context([m,e,e,d,p,c,e, k,s,i,v,k,f,q])).
...

```

Index

A	
acides aminés	16
alphabet	44
apprentissage	41
arbre de dérivation	54
atome	61
automate	46
dérivé	48
déterministe	47
B	
banques	20
de séquences	20
de structures	20
biais	65
de recherche	65
sémantiques	65
syntaxiques	65
Blue-Fringe	52
Brazma	22
C	
classification	23
clause	61
définie	62
de Horn	61
connaissance	63
a priori	53
conséquence logique	62
D	
découverte de motifs	23
dérivation	62
E	
échantillon	95
EDSM	52
F	
formule	61
fusion	48
G	
golem	68
grammaire	44
algébrique	45
stochastique	57
linéaire	45
équilibrée	55
étiquetée	55
régulière	45
H	
HMM	56
homologie	21
I	
induction	63
inférence	94
InHerIt	101
interprétation	62, 98
K	
k-formule	98
ancrée	100
kleene	47
L	
langage	44
de contrôle	55
canonique	56
lattice	49
liaisons	18
covalentes	18
faibles	18

littéral 61
 logique des prédicats 60

M

méthodes ab initio 32
 méthodes de novo 32
 MCA 49
 modèle 62
 moindre généralisé 67
 motif 22
 mots 44

P

partition 48
 pont disulfure 10
 prédicat 61
 prédiction
 appariement de cystines 36
 cystines 34
 motifs 22
 motifs appariés 26
 ponts disulfures 33
 structure secondaire 28
 structure tertiaire 32
 PROGOL 65, 68
 programmation logique inductive 60
 protéines 15
 PTA 50

R

résolution 63
 résolvante 63
 raffinement 48
 rlgg 68
 RPNI 51

S

sous-séquence 98
 structure 16
 primaire 16
 quaternaire 19
 secondaire 17
 tertiaire 17
 substitution 63

T

Taylor 74
 terme 61
 theta-subsumption 64
 threading 28, 33
 topologie 21
 treillis 49

U

unificateur 63

Bibliographie

- [Afonnikov *et al.*, 1997] AFONNIKOV, D., KONDRAKHIN, I. et TITOV, I. (1997). Detection of correlating DNA-binding positions in CREB and AP-1 family transcription factors]. *Mol Biol (Mosk)*, 31(4):741–8. 27
- [Aho et Ullman, 1972] AHO, A. V. et ULLMAN, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*. Prentice Hall Professional Technical Reference. 57
- [Alexandrov et Go, 1994] ALEXANDROV, N. et GO, N. (1994). Biological meaning, statistical significance, and classification of local spatial similarities in nonhomologous proteins. *Protein Science*, 3(6):866–75. 32
- [Altschul *et al.*, 1990] ALTSCHUL, S., GISH, W., MILLER, W., MYERS, E. et LIPMAN, D. (1990). Basic local alignment search tool. *J Mol Biol*, 215(3):403–10. 21
- [Altschul *et al.*, 1997] ALTSCHUL, S., MADDEN, T., SCHAFFER, A., ZHANG, J., ZHANG, Z., MILLER, W. et LIPMAN, D. (1997). Gapped BLAST and PSI-BLAST : a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–402. 21
- [Amar et Putzolu, 1964] AMAR, V. et PUTZOLU, G. (1964). On a family of linear grammars. *Information and Control*, 7:283–291. 55
- [Andonov *et al.*, 2003] ANDONOV, R., BALEV, S. et YANEV, N. (2003). Protein threading : From mathematical models to parallel implementations. Rapport technique 1552, IRISA. 33
- [Anfinsen, 1973] ANFINSEN, C. B. (1973). Principles that govern the folding of protein chains. *Science*, 181(96):223–30. 19
- [Arikawa *et al.*, 1993] ARIKAWA, S., MIYANO, S., SHINOHARA, A., KUHARA, S., MUKOUCHI, Y. et SHINOHARA, T. (1993). A machine discovery from amino acid sequences by decision trees over regular patterns. *New Generation Computing*, 11:361–375. 24
- [Attwood *et al.*, 2003] ATTWOOD, T. K., BRADLEY, P., FLOWER, D. R., GAULTON, A., MAUDLING, N., MITCHELL, A. L., MOULTON, G., NORDLE, A., PAINE, K., TAYLOR, P., UDDIN, A. et ZYGOURI, C. (2003). PRINTS and its automatic supplement, prePRINTS. *Nucleic Acids Res*, 31(1):400–2. 21
- [Badea et Stanciu, 1999] BADEA, L. et STANCIU, M. (1999). Refinement operators can be (weakly) perfect. In DŽEROSKI, S. et FLACH, P., éditeurs : *ILP99*, volume 1634 de *LNAI*, pages 21–32. SV. 64

- [Bairoch *et al.*, 1997] BAIROCH, A., BUCHER, P. et HOFMANN, K. (1997). The PROSITE database, its status in 1997. *Nucleic Acids Res*, 25(1):217–21. 21
- [Baldi *et al.*, 1994] BALDI, P., CHAUVIN, Y., HUNKAPILLER, T. et MCCLURE, M. (1994). Hidden Markov models of biological primary sequence information. *Proc Natl Acad Sci U S A*, 91(3):1059–63. 56
- [Balev, 2004] BALEV, S. (2004). Solving the protein threading problem by lagrangian relaxation. *4th Workshop on Algorithms in Bioinformatics, WABI*. 33
- [Bateman *et al.*, 2004] BATEMAN, A., COIN, L., DURBIN, R., FINN, R., HOLLICH, V., GRIFFITHS-JONES, S., KHANNA, A., MARSHALL, M., MOXON, S., SONNHAMMER, E., STUDHOLME, D., YEATS, C. et EDDY, S. (2004). The pfam protein families database. *Nucleic Acids Res.*, 32:138–41. 21
- [Bergadano *et al.*, 1988] BERGADANO, F., GIORDANA, A. et SAITTA, L. (1988). Automated concept acquisition in noisy environments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(4):555–578. 67
- [Berman *et al.*, 2000] BERMAN, H., WESTBROOK, J., FENG, Z., GILLILAND, G., BHAT, T., WEISSIG, H., SHINDYALOV, I. et BOURNE, P. (2000). The protein data bank. *Nucleic Acids Research*, 28:235–242. 20, 75, 96
- [Bernot, 2001] BERNOT, A. (2001). *Analyse des génomes, transcriptomes et protéomes*. Paris : Dunod. 222p. 9
- [Boeckmann *et al.*, 2003a] BOECKMANN, B., A. BAIROCH AND, R. A., BLATTER, M., ESTREICHER, A., GASTEIGER, E., MARTIN, M., MICHOD, K., O'DONOVAN, C., PHAN, I., PILBOUT, S. et SCHNEIDER, M. (2003a). The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Res.*, 31:365–370. 16
- [Boeckmann *et al.*, 2003b] BOECKMANN, B., BAIROCH, A., APWEILER, R., BLATTER, M., ESTREICHER, A., GASTEIGER, E., MARTIN, M., MICHOD, K., O'DONOVAN, C., PHAN, I., PILBOUT, S. et SCHNEIDER, M. (2003b). The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Res*, 31(1):365–70. 20
- [Boisbouvier *et al.*, 2000] BOISBOUVIER, J., BLACKLEDGE, M., SOLLIER, A. et MARION, D. (2000). Simultaneous determination of disulphide bridge topology and three-dimensional structure using ambiguous intersulphur distance restraints : possibilities and limitations. *J Biomol NMR*, 16(3):197–208. 11, 36
- [Bonneau *et al.*, 2001] BONNEAU, R., STRAUSS, C. et BAKER, D. (2001). Improving the performance of Rosetta using multiple sequence alignment information and global measures of hydrophobic core formation. *Proteins*, 43(1):1–11. 32
- [Branden et Tooze, 1999] BRANDEN, C. et TOOZE, J. (1999). *Introduction to protein structure*. Garland publishing Inc. 19
- [Brazma *et al.*, 1998] BRAZMA, A., JONASSEN, I., EIDHAMMER, I. et GILBERT, D. (1998). Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–304. 23, 25

- [Brazma *et al.*, 1997] BRAZMA, A., JONASSEN, I., EIDHAMMER, I. et UKKONEN, E. (1997). Relation patterns and their automatic discovery in biosequences. Rapport technique 135, Department of Informatics, University of Bergen, Bergen, Norway. [26](#)
- [Brazma *et al.*, 1996] BRAZMA, A., JONASSEN, I., UKKONEN, E. et VILO, J. (1996). Discovering patterns and subfamilies in biosequences. *In Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, ISMB 1996.* [24](#)
- [Brejova *et al.*, 2000] BREJOVA, B., DIMARCO, C., VINAR, T., HIDALGO, S., HOLGUIN, G. et PATTEN, C. (2000). Finding patterns in biological sequences. unpublished project report for cs798g, university of waterloo, fall 2000. [25](#)
- [Brodsky *et al.*, 1992] BRODSKY, L. I., VASSILYEV, A. V., KALAYDZIDIS, Y. L., OSIPOV, Y. S., TATUZOV, R. L. et FERANCHUK, S. I. (1992). Genebee : the program package for biopolymer structure analysis. *In GINDIKIN, S., éditeur : Mathematical methods of analysis of biopolymer sequences*, volume 8 de *DIMACS series in discrete mathematics and theoretical computer science*. American Mathematical Society. [26](#)
- [Brooks *et al.*, 1990] BROOKS, C., KARPLUS, M. et PETTITT, B. (1990.). A theoretical perspective of dynamics, structure and thermodynamics. *Proteins.* [32](#)
- [Brown et Wilson, 1996] BROWN, M. et WILSON, C. (1996). RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. *Pac Symp Biocomput*, pages 109–25. [58](#)
- [Bru *et al.*, 2005] BRU, C., COURCELLE, E., CARRERE, S., BEAUSSE, Y., DALMAR, S. et KAHN, D. (2005). The ProDom database of protein domain families : more emphasis on 3D. *Nucleic Acids Res*, 33(Database issue):D212–5. [21](#)
- [Cai *et al.*, 2003] CAI, L., MALMBERG, R. L. et WU, Y. (2003). Stochastic modeling of RNA pseudoknotted structures : a grammatical approach. *Bioinformatics*, 19:66–73. [58](#), [59](#)
- [Califano, 2000] CALIFANO, A. (2000). Splash : structural pattern localization analysis by sequential histograms. *Bioinformatics*, 16(4):341–57. [26](#)
- [Ceroni *et al.*, 2003a] CERONI, A., FRASCONI, P., PASSERINI, A. et VULLO, A. (2003a). A combination of support vector machines and bidirectional recurrent neural networks for protein secondary structure prediction. *In AIIA 2003 Advances in Artificial Intelligence*, pages 142–153. [31](#)
- [Ceroni *et al.*, 2003b] CERONI, A., FRASCONI, P., PASSERINI, A. et VULLO, A. (2003b). Predicting the disulfide bonding state of cysteins with combinations of kernel machines. *Journal of VLSI Signal Processing*, 35(3):287–295. [34](#), [35](#)
- [Chen *et al.*, 2004] CHEN, Y., LIN, Y., LIN, C. et HWANG, J. (2004). Prediction of the bonding states of cysteines using the support vector machines based on multiple feature vectors and cysteine state sequences. *Proteins : Structure, Function, and Bioinformatics*, 55(4):1036–1042. [34](#), [36](#)
- [Chomsky, 1965] CHOMSKY, N. (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA. [45](#)

- [Chothia, 1992] CHOTHIA, C. (1992). Proteins. One thousand families for the molecular biologist. *Nature*, 357(6379):543–4. 32
- [Chothia et Lesk, 1986] CHOTHIA, C. et LESK, A. (1986). The relation between the divergence of sequence and structure in proteins. *EMBO J*, 5(4):823–6. 33
- [Chou et Fasman, 1974] CHOU, P. et FASMAN, G. (1974). Conformational parameters for amino acids in helical, beta-sheet, and random coil regions calculated from proteins. *Biochemistry*, 13(2):211–22. 29
- [Chou et Fasman, 1978] CHOU, P. et FASMAN, G. (1978). Prediction of the secondary structure of proteins from their amino acid sequence. *Adv Enzymol Relat Areas Mol Biol*, 47:45–148. 28
- [Chuang *et al.*, 2003] CHUANG, C., CHEN, C., YANG, J., LYU, P. et HWANG, J. (2003). Relationship between protein structures and disulfide-bonding patterns. *Proteins*, 53:1–5. 11, 19
- [Claveau, 2003] CLAVEAU, V. (2003). *Acquisition automatique de lexiques sémantiques pour la recherche d'information*. Thèse de doctorat, Université de Rennes 1. 60
- [Cohen, 1995] COHEN, W. (1995). Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*. 67
- [Cootes *et al.*, 2001] COOTES, A., MUGGLETON, S., GREAVES, R. et STERNBERG, M. (2001). Automatic determination of protein fold signatures from structured superposition. *Electronic Transactions in Artificial Intelligence*, 6:245–274. 69
- [Cootes *et al.*, 2003] COOTES, A., MUGGLETON, S. et STERNBERG., M. (2003). The automatic discovery of structural principles describing protein fold space. *Journal of Molecular Biology*. 69
- [Cornuéjols et Miclet, 2002] CORNUÉJOLS, A. et MICLET, L. (2002). *Apprentissage artificiel : concepts et algorithmes*. Louis - Jean. 41
- [Coste et Kerbellec, 2005] COSTE, F. et KERBELLEC, G. (2005). A similar fragments merging approach to learn automata on proteins. In *ECML 2005*, pages 522–529, Portugal, Porto. 53, 59, 115
- [Coste *et al.*, 2004] COSTE, F., KERBELLEC, G., IDMONT, B., FREDOUILLE, D. et DELAMARCHE, C. (2004). Apprentissage d'automates par fusions de paires de fragments significativement similaires et premières expérimentations sur les protéines MIP. In *JOBIM*, pages 45–57. 53, 59
- [Creighton, 1992] CREIGHTON, T. (1992). *Proteins : Structures and molecular properties*. Hardcover, WH. Freeman. 19
- [Crespi-Reghizzi, 1972] CRESPI-REGHIZZI, S. (1972). An efficient model for grammar inference. In *Information Processing 71*, pages 524–529. Elsevier North-Holland. 54
- [Crooks et Brenner, 2004] CROOKS, G. et BRENNER, S. (2004). Protein secondary structure : entropy, correlations and prediction. *Bioinformatics*, 20(10):1603–11. 27
- [Cuff et Barton, 2000] CUFF, J. A. et BARTON, G. J. (2000). Application of multiple sequence alignment profiles to improve protein secondary structure prediction. *Proteins*, 40(3):502–11. 29

- [Daly *et al.*, 1999] DALY, N. L., KOLTAY, A., GUSTAFSON, K. R., BOYD, M. R., CASAS-FINET, J. R. et CRAIK, D. J. (1999). Solution structure by NMR of circulin A : a macrocyclic knotted peptide having anti-HIV activity. *J Mol Biol*, 285(1):333–45. [11](#)
- [Donnelly *et al.*, 1994] DONNELLY, D., OVERINGTON, J. et BLUNDELL, T. (1994). The prediction and orientation of alpha-helices from sequence alignments : the combined use of environment-dependent substitution tables, Fourier transform methods and helix capping rules. *Protein Eng*, 7(5):645–53. [29](#)
- [Durbin *et al.*, 1998] DURBIN, R., EDDY, S., KROGH, A. et MITCHISON, G. (1998). *Biological sequence analysis, Probabilistic models of proteins and nucleic acids*. Cambridge University Press. [56](#)
- [Dzeroski, 1993] DZEROSKI, S. (1993). Handling imperfect data in inductive logic programming. In *Proceedings of the 4th Scandinavian Conference on AI*, pages 111–125. IOS Press. [67](#)
- [Eddy, 1998] EDDY, S. (1998). Profile hidden Markov models. *Bioinformatics*, 14(9):755–63. [56](#)
- [Errami *et al.*, 2003] ERRAMI, M., GEOURJON, C. et DELÉAGE, G. (2003). Conservation of amino acids into multiple alignments involved in pairwise interactions in three-dimensional protein structures. *Journal of Bioinformatics and Computational Biology*, 1(3):505–520. [33](#), [74](#)
- [Eskin et Pevzner, 2002] ESKIN, E. et PEVZNER, P. (2002). Finding composite regulatory patterns in DNA sequences. *Bioinformatics*, 18 Suppl 1:S354–63. [27](#)
- [Esposito *et al.*, 1996] ESPOSITO, F., LATERZA, A., MALERBA, D. et SEMERARO, G. (1996). Refinement of datalog programs. In *ML96WS*, pages 73–94. [64](#)
- [Fain et Levitt, 2001] FAIN, B. et LEVITT, M. (2001). A novel method for sampling alpha-helical protein backbones. *J Mol Biol*, 305(2):191–201. [11](#)
- [Fariselli et Casadio, 2001] FARISELLI, P. et CASADIO, R. (2001). Prediction of disulfide connectivity in proteins. *Bioinformatics*, 17:957–96. [7](#), [36](#), [37](#), [38](#)
- [Fariselli *et al.*, 2002] FARISELLI, P., MARTELLI, P. et CASADIO, R. (2002). A neural network-based method for predicting the disulfide connectivity in proteins. In *Knowledge Based Intelligent Information Engineering Systems and allied Technologies*, volume 1, pages 464–468. [36](#), [38](#)
- [Fariselli *et al.*, 1999] FARISELLI, P., RICCOBELLI, P. et CASADIO, R. (1999). Role of evolutionary information in predicting the disulfide-bonding state of cysteines in proteins. *Proteins*, 36:340–346. [34](#), [35](#)
- [Fasman, 1989] FASMAN, G. (1989). Development of the prediction of protein structure. In : Prediction of protein structure and the principles of protein conformation. Plenum, New York and London. [29](#)
- [Feldman et Reder, 1969] FELDMAN, J. et REDER, S. (1969). Grammatical complexity and inference. Rapport technique, Computer Science Dept., Stanford University. [54](#)

- [Fiser *et al.*, 1992] FISER, A., CSERZO, M., TUDOS, E. et SIMON, I. (1992). Different sequence environments of cysteines and half cysteines in proteins : application to predict disulfide forming residues. *FEBS Lett.* **34**, 74
- [Fiser et Simon, 2000] FISER, A. et SIMON, I. (2000). Predicting the oxidation state of cysteines by multiple sequence alignment. *Bioinformatics*, 16:251–256. **34**, **35**
- [Fletcher *et al.*, 1997] FLETCHER, J., SMITH, R., O'DONOGHUE, S., NILGES, M., CONNOR, M., HOWDEN, M. E., CHRISTIE, M. J. et KING, G. F. (1997). The structure of a novel insecticidal neurotoxin, omega-atracotoxin-HV1, from the venom of an Australian funnel web spider. *Nat Struct Biol*, 4(7):559–66. **11**
- [Francesco *et al.*, 1995] FRANCESCO, V. D., MUNSON, P. et GARNIER, J. (1995). Use of multiple alignments in protein secondary structure prediction. In *HICSS (5)*, pages 285–291. **29**
- [Frasconi *et al.*, 2002] FRASCONI, P., PASSERINI, A. et VULLO, A. (2002). A two-stage svm architecture for predicting the disulfide bonding state of cysteines. In *NNSP'02*. **34**, **35**
- [Frishman et Argos, 1996] FRISHMAN, D. et ARGOS, P. (1996). Incorporation of non-local interactions in protein secondary structure prediction from the amino acid sequence. *Protein Eng*, 9(2):133–42. **29**
- [Fu et Booth, 1975] FU, K. et BOOTH, T. (1975). Grammatical inference : introduction and survey. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5:95–111, 409–423. **54**
- [Garnier *et al.*, 1996] GARNIER, J., GIBRAT, J. et ROBSON, B. (1996). Gor secondary structure prediction method version iv. In *Methods in enzymology*, volume 266, pages 540–553. R.F. Doolittle. **29**
- [Garnier *et al.*, 1978] GARNIER, J., OSGUTHORPE, D. et ROBSON, B. (1978). Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins. *Journal of Molecular Biology*, 120(1):97–120. **28**, **29**
- [Geourjon et Deleage, 1994] GEOURJON, C. et DELEAGE, G. (1994). SOPM : a self-optimized method for protein secondary structure prediction. *Protein Eng*, 7(2):157–64. **29**, **30**
- [Geourjon et Deleage, 1995] GEOURJON, C. et DELEAGE, G. (1995). SOPMA : significant improvements in protein secondary structure prediction by consensus prediction from multiple alignments. *Comput Appl Biosci*, 11(6):681–4. **29**, **30**, **109**
- [Gobel *et al.*, 1994] GOBEL, U., SANDER, C., SCHNEIDER, R. et VALENCIA, A. (1994). Correlated mutations and residue contacts in proteins. *Proteins*, 18(4):309–17. **27**
- [Gold, 1978] GOLD, E. (1978). Complexity of automaton identification from given data. *Information and control*, 37:302–320. **53**
- [Gonzalez et Thomason, 1978] GONZALEZ, R. et THOMASON, M. (1978). Syntactic pattern recognition : an introduction. In *Addison-Wesley*. **54**
- [Gorman *et al.*, 2002] GORMAN, J., WALLIS, T. et PITT, J. (2002). Protein disulfide bond determination by mass spectrometry. *Mass Spectrom. Rev.*, 21:183–216. **95**

- [Gras *et al.*, 2003] GRAS, R., HERNANDEZ, D., HERNANDEZ, P., ZANGGER, N., MESCAM, Y., FREY, J., MARTIN, O., NICOLAS, J. et APPEL, R. (2003). Cooperative metaheuristics for exploring proteomic data. *Artificial Intelligence Review*, 20:95–120. 27
- [Grundy *et al.*, 1997] GRUNDY, W. N., BAILEY, T. L., ELKAN, C. P. et BAKER, M. E. (1997). Hidden markov model analysis of motifs in steroid dehydrogenases and their homologos. *Biochemical and Biophysical Research Communications*, 231:760–766. 24
- [Guralnik et Karypis, 2001] GURALNIK, V. et KARYPIS, G. (2001). A scalable algorithm for clustering protein sequences. *In Book :Biokdd*, pages 73–80. 24
- [Harrison et Sternberg, 1996] HARRISON, P. et STERNBERG, M. (1996). The disulphide beta-cross : from cystine geometry and clustering to classification of small disulphide-rich protein folds. *J Mol Biol*, 264(3):603–23. 11
- [Henry, 2002] HENRY, T. (2002). Heuristiques pour l'apprentissage automatique de modèles biologiques par inférence grammaticale. Rapport de DEA, Université de Rennes 1, France. 53
- [Hernandez *et al.*, 2004] HERNANDEZ, D., GRAS, R. et APPEL, R. (2004). Model : An efficient strategy for ungapped local multiple alignment. *Computational Biology and Chemistry*, 28(2):119–128. 27
- [Hirst *et al.*, 1994] HIRST, J., KING, R. et STERNBERG, M. (1994). Quantitative structure-activity relationships by neural networks and inductive logic programming. II. The inhibition of dihydrofolate reductase by triazines. *J Comput Aided Mol Des*, 8(4):421–32. 69
- [Hua et Sun, 2001] HUA, S. et SUN, Z. (2001). A novel method of protein secondary structure prediction with high segment overlap measure : Svm approach. 31
- [Huang *et al.*, 1999] HUANG, E., SAMUDRALA, R. et PONDER, J. (1999). Ab initio fold prediction of small helical proteins using distance geometry and knowledge-based scoring functions. *J Mol Biol*, 290(1):267–81. 11
- [Idmont, 2003] IDMONT, B. (2003). Mesures de similarité et d'entropie pour l'apprentissage d'automates classifieurs de protéines. Mémoire de D.E.A., Université de Rennes 1, France. 53
- [Jonassen, 1997] JONASSEN, I. (1997). Efficient discovery of conserved patterns using a pattern graph. *CABIOS*, 13:509–522. 24, 26
- [Jonassen *et al.*, 1995] JONASSEN, I., COLLINS, J. et HIGGINS, D. (1995). Finding flexible patterns in unaligned protein sequences. *Protein science*, 4(8):1587–1595. 24, 26
- [Juillé et Pollack, 1998] JUILLÉ, H. et POLLACK, J. (1998). A stochastic search approach to grammar induction. *Grammatical Inference*, pages 126–137. 52
- [Junker *et al.*, 1999] JUNKER, V., APWEILER, R. et BAIROCH, A. (1999). Representation of functional information in the SWISS-PROT data bank. *Bioinformatics*, 15(12):1066–7. 20

- [Kapur et Narendran, 1986] KAPUR, D. et NARENDRAN, P. (1986). Np-completeness of the set unification and matching problems. *In Proc. of the 8th international conference on Automated deduction*, pages 489–495, New York, NY, USA. Springer-Verlag New York, Inc. 64
- [Karp et al., 1972] KARP, R., MILLER, R. et ROSENBERG, A. (1972). Rapid identification of repeated patterns in strings, trees and arrays. *Proceedings of the ACM Symposium on the Theory of Computing*, pages 125–136. 25
- [Karplus et al., 1999] KARPLUS, K., BARRETT, C., CLINE, M., DIEKHANS, M., GRATE, L. et HUGHEY, R. (1999). Predicting protein structure using only sequence information. *Proteins*, Suppl 3:121–5. 32
- [King et al., 1992] KING, R., MUGGLETON, S., LEWIS, R. et STERNBERG, M. (1992). Drug design by machine learning : The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proc. National Academy of Sciences*. 69
- [King et al., 1996] KING, R., MUGGLETON, S., SRINIVASAN, A. et STERNBERG, M. (1996). Structure-activity relationships derived by machine learning : the use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proc Natl Acad Sci U S A*, 93(1):438–42. 69
- [Klepeis et Floudas, 2003] KLEPEIS, J. et FLOUDAS, C. (2003). Prediction of beta-sheet topology and disulfide bridges in polypeptides. *Journal of Computational Chemistry*, 24:191–208. 36, 38
- [Klink et al., 2000] KLINK, T., WOYCECHOWSKY, K., TAYLOR, K. et RAINES, R. (2000). Contribution of disulfide bonds to the conformational stability and catalytic activity of ribonuclease A. *Eur J Biochem*, 267(2):566–72. 11
- [Kneller et al., 1990] KNELLER, D., COHEN, F. et LANGRIDGE, R. (1990). Improvements in protein secondary structure prediction by an enhanced neural network. *J Mol Biol*, 214(1):171–82. 29
- [Korber et al., 1993] KORBER, B., FARBER, R., WOLPERT, D. et LAPEDES, A. (1993). Covariation of mutations in the V3 loop of human immunodeficiency virus type 1 envelope protein : an information theoretic analysis. *Proc Natl Acad Sci U S A*, 90(15):7176–80. 27
- [Krogh et al., 1994] KROGH, A., BROWN, M., MIAN, I., SJOLANDER, K. et HAUSSLER, D. (1994). Hidden Markov models in computational biology. Applications to protein modeling. *J Mol Biol*, 235(5):1501–31. 56
- [Kudo et al., 1992] KUDO, M., KITAMURA-ABE, S., SHIMBO, M. et LIDA, Y. (1992). Analysis of context of 5'-splice site sequences in mammalian mRNA precursor by subclass method. *Computer Applications in the Biosciences*, 8(4):367–376. 26
- [Landraud et al., 1989] LANDRAUD, A., AVRIL, J. et CHRETIENNE, P. (1989). An algorithm for finding a common structure shared by a family strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):890–895. 24, 26

- [Lang, 1992] LANG, K. J. (1992). Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, pages 45–52, New York, N.Y. ACM. 51
- [Lang et al., 1998] LANG, K. J., PEARLMUTTER, B. A. et PRICE, R. A. (1998). Results of the abbingo one dfa learning competition and a new evidence-driven state merging algorithm. In *ICGI '98 : Proceedings of the 4th International Colloquium on Grammatical Inference*, pages 1–12, London, UK. Springer-Verlag. 52, 86
- [Larson et al., 2000] LARSON, S., NARDO, A. D. et DAVIDSON, A. (2000). Analysis of covariation in an SH3 domain sequence alignment : applications in tertiary contact prediction and the design of compensating hydrophobic core substitutions. *Journal of Molecular Biology*, 303(3):433–46. 27
- [Lathrop, 1994] LATHROP, R. (1994). The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Eng*, 7(9):1059–68. 33
- [Lathrop et Smith, 1996] LATHROP, R. et SMITH, T. (1996). Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of molecular biology*, 255(4):641–65. 33
- [Lavrac et Dzeroski, 1994] LAVRAC, N. et DZEROSKI, S. (1994). *Inductive Logic Programming : Techniques and Applications*. Ellis Horwood, New York. 41, 60, 67
- [Lee, 1991] LEE, B. (1991). Isoenthalpic and isoentropic temperatures and the thermodynamics of protein denaturation. *Proceedings of the National Academy of Science*, 88:5154–5158. 19
- [Leroux, 2005] LEROUX, A. (2005). Inférence grammaticale sur des alphabets ordonnés : application à la découverte de motifs dans des familles de protéines. Mémoire de D.E.A., Irista, Rennes. 59, 115
- [Letunic et al., 2004] LETUNIC, I., COPLEY, R., SCHMIDT, S., CICCARELLI, F., DOERKS, T., SCHULTZ, J., PONTING, C. P. et BORK, P. (2004). SMART 4.0 : towards genomic data integration. *Nucleic Acids Res*, 32(Database issue):D142–4. 21
- [Levin, 1997] LEVIN, J. (1997). Exploring the limits of nearest neighbour secondary structure prediction. *Protein Eng*, 10(7):771–6. 29
- [Levin et al., 1993] LEVIN, J., PASCARELLA, S., ARGOS, P. et GARNIER, J. (1993). Quantification of secondary structure prediction improvement using multiple alignments. *Protein Eng*, 6(8):849–54. 29
- [Levin et al., 1986] LEVIN, J., ROBSON, B. et GARNIER, J. (1986). An algorithm for secondary structure determination in proteins based on sequence similarity. *FEBS Lett*, 205(2):303–8. 29
- [Lloyd, 1987] LLOYD, J. (1987). *Foundations of Logic Programming*. Springer-Verlag, second édition. 41
- [Lund et al., 1996] LUND, O., HANSEN, J., BRUNAK, S. et BOHR, J. (1996). Relationship between protein structure and geometrical constraints. *Protein Sci*, 5(11):2217–25. 11

- [Marcinkowski et Pacholski, 1992] MARCINKOWSKI, J. et PACHOLSKI, L. (1992). Undecidability of the horn-clause implication problem. *In* IEEE COMPUTER SOCIETY PRESS, S. S., éditeur : *Proceedings of IEEE International Conference of Foundations of Computer Science (FOCS)*, pages 354–362. 64
- [Martelli *et al.*, 2002] MARTELLI, P., FARISELLI, P., MALAGUTI, L. et R.CASADIO (2002). Prediction of the disulfide-bonding state of cysteines in proteins at 88% accuracy. *Protein Science*, 11:2735–2739. 34, 35, 36
- [Marti-Renom *et al.*, 2000] MARTI-RENOM, M., STUART, A., FISER, A., SANCHEZ, R., MELO, F. et SALI, A. (2000). Comparative protein structure modeling of genes and genomes. *Annu Rev Biophys Biomol Struct*, 29:291–325. 32
- [Martin *et al.*, 2005] MARTIN, J., GIBRAT, J. et RODOLPHE, F. (2005). Hmm for local protein structure. *In Proceedings ASMDA*. 29, 30
- [Martinez, 1988] MARTINEZ, H. M. (1988). A flexible multiple sequence alignment program. *Nucleic Acids Research*, 16(5):1683–1691. 24, 26
- [Mas *et al.*, 1998] MAS, J., ALOY, P., MARTI-RENOM, M., OLIVA, B., BLANCO-APARICIO, C., MOLINA, M., de LLORENS, R., QUEROL, E. et AVILES, F. (1998). Protein similarities beyond disulphide bridge topology. *J Mol Biol*, 284(3):541–8. 11
- [Matsui *et al.*, 2005] MATSUI, H., SATO, K. et SAKAKIBARA, Y. (2005). Pair stochastic tree adjoining grammars for aligning and predicting pseudoknot RNA structures. *Bioinformatics*, 21(11):2611–7. 58
- [Matsumura et Matthews, 1989] MATSUMURA, M. et MATTHEWS, B. (1989). Control of enzyme activity by an engineered disulfide bond. *Science*, 243(4892):792–4. 33
- [Meiler *et al.*, 2002] MEILER, J., MUELLER, M., ZEIDLER, A. et SCHMAESCHKE, F. (2002). Jufo : Secondary structure prediction for proteins. <http://www.jens-meiler.de>. 29
- [Mika et Rost, 2003] MIKA, S. et ROST, B. (2003). UniqueProt : Creating representative protein sequence sets. *Nucleic Acids Res*, 31(13):3789–91. 32
- [Mitchell, 1977] MITCHELL, T. (1977). Version spaces : a candidate elimination approach to rule learning. *In IJCAI*, pages 305–310. 43
- [Mitchell, 1997] MITCHELL, T. (1997). *Machine Learning*. McGraw-Hill. 41
- [Mooney et Califf, 1995] MOONEY, R. et CALIFF, M. (1995). Induction of first-order decision lists : Results on learning the past tense of English verbs. *In* DE RAEDT, L., éditeur : *ILP95*, pages 145–146. DEPTCW. 67
- [Mucchielli-Giorgi *et al.*, 2002] MUCCHIELLI-GIORGI, M., HAZOUT, S. et TUFFÉRY, P. (2002). Predicting the disulfide bonding state of cysteines using protein descriptors. *Proteins*. 34, 35
- [Muggleton, 1992] MUGGLETON, S. (1992). Inverting implication. *In Proc. Second International Workshop on Inductive Logic Programming*, Tokyo, Japon. ILP'92. 68
- [Muggleton, 1995] MUGGLETON, S. (1995). Inverse entailment and prolog. *New Generation Computing Journal*, 13:245–286. 65

- [Muggleton, 1998] MUGGLETON, S. (1998). *Progol*.
<http://www.doc.ic.ac.uk/~shm/progol.html>. 103
- [Muggleton et Feng, 1992] MUGGLETON, S. et FENG, C. (1992). Efficient induction in logic programs. *Inductive Logic Programming*, pages 281–298. 68
- [Muggleton *et al.*, 1992] MUGGLETON, S., KING, R. et STERNBERG, M. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Eng*, 5(7): 647–57. 69
- [Muggleton et Raedt, 1994] MUGGLETON, S. et RAEDT, L. D. (1994). Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19/20:629–679. 41, 64, 65
- [Mulder *et al.*, 2005] MULDER, N., APWEILER, R., ATTWOOD, T., BAIROCH, A., BATEMAN, A., BINNS, D., BRADLEY, P., BORK, P., BUCHER, P., CERUTTI, L., COPLEY, R., COURCELLE, E., DAS, U., DURBIN, R., FLEISCHMANN, W., GOUGH, J., HAFT, D., HARTE, N., HULO, N., KAHN, D., KANAPIN, A., KRESTYANINOVA, M., LONSDALE, D., LOPEZ, R., LETUNIC, I., MADERA, M., MASLEN, J., MCDOWALL, J., MITCHELL, A., NIKOLSKAYA, A., ORCHARD, S., PAGNI, M., PONTING, C., QUEVILLON, E., SELLENGUT, J., SIGRIST, C., SILVENTOINEN, V., STUDHOLME, D., VAUGHAN, R. et WU, C. (2005). InterPro, progress and status in 2005. *Nucleic Acids Res*, 33:201–205. 21
- [Muskal *et al.*, 1990] MUSKAL, S., HOLBROOK, S. et KIM, S. (1990). Prediction of the disulfide-bonding state of cysteine in proteins. *Proteins Eng.*, 3(8):667–72. 34, 74
- [Neuwald et Green, 1994] NEUWALD, A. F. et GREEN, P. (1994). Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712. 24, 25
- [Nevill-Manning *et al.*, 1997] NEVILL-MANNING, C. G., SETHI, K. S., WU, T. D. et BRUTLAG, D. L. (1997). Enumerating and ranking discrete motifs. In *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology, ISMB 1997*, volume 4, pages 202–209. AAAI Press. 24
- [Nguyen et Rajapakse, 2003] NGUYEN, M. et RAJAPAKSE, J. (2003). Multi-class support vector machines for protein secondary structure prediction. *Genome Inform Ser Workshop Genome Inform*, 14:218–27. 31
- [Nguyen et Rajapakse, 2005] NGUYEN, M. et RAJAPAKSE, J. (2005). Two-stage multi-class support vector machines to protein secondary structure prediction. *Pac Symp Biocomput*, pages 346–57. 29, 31
- [Nienhuys-Cheng et Wolf, 1996] NIENHUYS-CHENG, S. et WOLF, R. D. (1996). Foundations of inductive logic programming. *Journal of Artificial Intelligence Research*, 4:341–363. 64
- [Notredame, 2002] NOTREDAME, C. (2002). Recent progress in multiple sequence alignment : a survey. *Pharmacogenomics*, 3(1):131–44. 25
- [Nédellec et Rouveirol, 1994] NÉDELLEC, C. et ROUVEIROL, C. (1994). Specification of the HAIKU system. Rapport technique 928, Université Paris-Sud. 67
- [Ogiwara *et al.*, 1992] OGIWARA, A., UCHIYAMA, I., SETO, Y. et KANEHISA, M. (1992). Construction of a dictionary of sequence motifs that characterize groups of related proteins. *Protein Engineering*, 5(6):479–488. 24

- [Oliveira et Silva, 1998] OLIVEIRA, A. et SILVA, J. (1998). Efficient search techniques for the inference of minimum size finite automata. *In String processing and information retrieval*, pages 81–89. 52
- [Oncina et Garcia, 1992] ONCINA, J. et GARCIA, P. (1992). Inferring regular languages in polynomial update time. *Pattern recognition and image analysis*, pages 49–61. 51, 86
- [Oren et al., 1998] OREN, D., FROY, O., AMIT, E., KLEINBERGER-DORON, N., GUREVITZ, M. et SHAANAN, B. (1998). An excitatory scorpion toxin with a distinctive feature : an additional alpha helix at the C terminus and its implications for interaction with insect sodium channels. *Structure*, 6(9):1095–103. 11
- [Ouali et King, 2000] OUALI, M. et KING, R. (2000). Cascaded multiple classifiers for secondary structure prediction. *Protein Sci*, 9(6):1162–76. 29
- [Pazzani et Kibler, 1992] PAZZANI, M. et KIBLER, D. (1992). The utility of knowledge in inductive learning. *ML*, 9(1):57–94. 67
- [Pearl et al., 2005] PEARL, F., TODD, A., SILLITOE, I., DIBLEY, M., REDFERN, O., LEWIS, T., BENNETT, C., MARSDEN, R., GRANT, A., LEE, D., AKPOR, A., MAIBAUM, M., HARRISON, A., DALLMAN, T., REEVES, G., DIBOUN, I., ADDOU, S., LISE, S., JOHNSTON, C., SILLERO, A., THORNTON, J. et ORENGO, C. (2005). The CATH Domain Structure Database and related resources Gene3D and DHS provide comprehensive domain family information for genome analysis. *Nucleic Acids Res*, 33:247–251. 21
- [Pearson et Lipman, 1988] PEARSON, W. et LIPMAN, D. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the U S A*, 85(8):2444–8. 21
- [Plotkin, 1970] PLOTKIN, G. (1970). A note on inductive generalization. *In Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press. 64, 67
- [Plotkin, 1971] PLOTKIN, G. (1971). A further note on inductive generalization. *In Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press. 68
- [Qian et Sejnowski, 1988] QIAN, N. et SEJNOWSKI, T. (1988). Predicting the secondary structure of globular proteins using neural network models. *J Mol Biol*, 202(4):865–84. 29, 30
- [Queen et Wegman, 1982] QUEEN, C. et WEGMAN, M. (1982). L. j. korn. *Nucleic Acids Research*, 10:449–456. 25
- [Quinlan et Cameron-Jones, 1993] QUINLAN, J. et CAMERON-JONES, R. (1993). FOIL : A midterm report. *In BRAZDIL, P., éditeur : ECML93*, volume 667 de *LNAI*, pages 3–20. SV. 67
- [Rabiner, 1990] RABINER, L. R. (1990). A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296. 57
- [Radhakrishnan et Nagaraja, 1988] RADHAKRISHNAN, V. et NAGARAJA, G. (1988). Inference of even linear grammars and its application to picture description languages. *Pattern recognition*, 21(1):55–62. 55

- [Raedt et Kersting, 2004] RAEDT, L. D. et KERSTING, K. (2004). Probabilistic inductive logic programming. 69
- [Rigoutsos et Floratos, 1998a] RIGOUTSOS, I. et FLORATOS, A. (1998a). Combinatorial pattern discovery in biological sequences : the TEIRESIAS algorithm. *Bioinformatics*, 14(1):55–67. 24, 26
- [Rigoutsos et Floratos, 1998b] RIGOUTSOS, I. et FLORATOS, A. (1998b). Motif discovery without alignment or enumeration. In *Proceedings 2nd Annual ACM International Conference on Computational Molecular Biology RECOMB'98*. 26
- [Riis et Krogh, 1996] RIIS, S. et KROGH, A. (1996). Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *J Comput Biol*, 3(1):163–83. 31
- [Rivas et Eddy, 2000] RIVAS, E. et EDDY, S. R. (2000). The language of RNA : a formal grammar that includes pseudoknots. *Bioinformatics*, 16(4):334–40. 58
- [Roberts, 1997] ROBERTS, S. (1997). *An Introduction to Progol*. University of York. ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/progol-intro.ps.gz. 68, 94
- [Robinson, 1965] ROBINSON, J. (1965). A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41. 63, 64
- [Rost, 1996] ROST, B. (1996). PHD : predicting one-dimensional protein structure by profile-based neural networks. *Methods Enzymol*, 266:525–39. 30
- [Rost et Sander, 1993] ROST, B. et SANDER, C. (1993). Prediction of protein secondary structure at better than 70% accuracy. *J Mol Biol*, 232(2):584–99. 30
- [Rost et Sander, 1994] ROST, B. et SANDER, C. (1994). Combining evolutionary information and neural networks to predict protein secondary structure. *Proteins*, 19(1):55–72. 29, 30, 31
- [Roytberg, 1992] ROYTBURG, M. (1992). A search for common patterns in many sequences. *Computer Applications in the Biosciences*, 8(1):57–64. 24
- [Sagot et al., 1995a] SAGOT, M., VIARI, A. et SOLDANO, H. (1995a). A distance-based block searching algorithm. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, ISMB 1995*, pages 322–331, Menlo Park, California. AAAI Press. 24
- [Sagot et al., 1995b] SAGOT, M., VIARI, A. et SOLDANO, H. (1995b). Multiple comparison : a peptide matching approach. In GALIL, Z. et UKKONEN, E., éditeurs : *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching 1995*, volume 937 de *Lecture Notes in Computer Science*, pages 366–385. Springer-Verlag Heidelberg. 24, 25
- [Sagot et Viari, 1996] SAGOT, M.-F. et VIARI, A. (1996). A double combinatorial approach to discovering patterns in biological sequences. In *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching 1996*, *Lecture Notes in Computer Science*, pages 186–208. Springer-Verlag Heidelberg. 24, 25

- [Sakakibara, 1990] SAKAKIBARA, Y. (1990). Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242. [54](#)
- [Sakakibara *et al.*, 1994] SAKAKIBARA, Y., BROWN, M., HUGHEY, R., MIAN, I., SJOLANDER, K., UNDERWOOD, R. et HAUSSLER, D. (1994). Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Res*, 22(23):5112–20. [56](#), [57](#)
- [Salamov et Solovyev, 1997] SALAMOV, A. et SOLOVYEV, V. (1997). Protein secondary structure prediction using local alignments. *Journal of Molecular Biology*, 268(1):31–6. [29](#), [30](#)
- [Saqi et Sternberg, 1994] SAQI, M. et STERNBERG, M. (1994). Identification of sequence motifs from a set of proteins with related functions with related function. *Protein Engineering*, 7(2):165–171. [24](#)
- [Schmidt-Schauss, 1988] SCHMIDT-SCHAUSS, M. (1988). Implication of clauses is undecidable. *TSC : theoretical computer science*, 59(3):287–296. [64](#)
- [Schneider *et al.*, 2005] SCHNEIDER, M., BAIROCH, A., WU, C. et APWEILER, R. (2005). Plant protein annotation in the UniProt Knowledgebase. *Plant Physiol*, 138(1):59–66. [20](#)
- [Schuler *et al.*, 1991] SCHULER, G. D., ALTSCHUL, S. F. et LIPMAN, D. J. (1991). A workbench for multiple alignment construction and analysis. *Proteins: Structure, Function, and Genetics*, 9:180–190. [26](#)
- [Scopes, 1994] SCOPES, R. (1994). *Protein purification, principles and practice*. Springer. [19](#)
- [Searls, 1997] SEARLS, D. (1997). Linguistic approaches to biological sequences. *Comput Appl Biosci*, 13(4):333–344. [5](#), [46](#)
- [Searls et Murphy, 1995] SEARLS, D. et MURPHY, K. (1995). Automata-theoretic models of mutation and alignment. *Proc Int Conf Intell Syst Mol Biol*, 3:341–349. [56](#)
- [Sebag et Rouveirol, 1997] SEBAG, M. et ROUVEIROL, C. (1997). Tractable induction and classification in first-order logic via stochastic matching. *In IJCAI97*, pages 888–893. MK. [64](#)
- [Shapiro, 1981] SHAPIRO, E. (1981). Inductive inference of theories from facts. *In of computer SCIENCE, D., éditeur : Rapport de recherche 624*. Yale university, New Haven, USA. [66](#), [67](#)
- [Shoudai *et al.*, 1995] SHOUDAI, T., LAPPE, M., MIYANO, S., SHINOHARA, A., OKAZAKI, T., ARIKAVA, S., SHIMOZONO, T. U. S., SHINOHARA, T. et KUHARA, S. (1995). Bonsai garden : parallel knowledge discovery system for amino acid sequences. *In Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, ISMB 1995*, pages 359–366, Menlo Park, California. AAAI Press. [24](#)
- [Smith *et al.*, 1990] SMITH, H. O., ANNAU, T. M. et CHANDRASEGARAN, S. (1990). Finding sequence motifs in groups of functionally related proteins. *In Proceedings of the National Academy of Science of the USA*, volume 87, pages 826–830. [24](#), [26](#)

- [Smith et Smith, 1990] SMITH, R. F. et SMITH, T. F. (1990). Automatic generation of primary sequence patterns from sets of related protein sequences. *In Proceedings of the National Academy of Science of the USA*, volume 87, pages 118–122, USA. 24, 25
- [Smith et Smith, 1992] SMITH, R. F. et SMITH, T. F. (1992). Pattern-induced multi-sequence alignment (pima) algorithm employing secondary structure-dependent gap penalties for comparative protein modelling. *Protein Engineering*, 5(1):35–41. 24
- [Song et al., 2004] SONG, J., WANG, M., LI, W. et XU, W. (2004). Prediction of the disulfide-bonding state of cysteines in proteins based on dipeptide composition. *Biochem. Biophys. Res. Commun.*, 318(1):142–147. 34, 35
- [Staden, 1989] STADEN, R. (1989). Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences*, 5(4):293–298. 25
- [Suyama et al., 1995] SUYAMA, M., NISHIOKA, T. et ODA, J. (1995). Searching for common sequence patterns among distantly related proteins. *Protein Engineering*, 8(11):1075–1080. 24, 25
- [Takada, 1988] TAKADA, Y. (1988). Grammatical inference for even linear languages based on control sets. *Information Processing Letters*, 28(4):193–199. 54, 55
- [Takada, 1994a] TAKADA, Y. (1994a). A hierarchy of language families learnable by regular language learners. *Lecture Notes in Computer Science*, 862:16. 55
- [Takada, 1994b] TAKADA, Y. (1994b). Learning formal languages based on control sets. *Lecture notes in AI*, 961. 12, 55, 71
- [Takada, 1995] TAKADA, Y. (1995). A hierarchy of language families learnable by regular language learning. *Information and Computation*, 123(1):138–145. 54, 113
- [Tateishi et al., 1995] TATEISHI, E., MARUYAMA, O. et MIYANO, S. (1995). Extracting best consensus motifs from positive and negative examples. Rapport technique RIFIS-TR-CS-115, Research Institute of Fundamental Information Science, Kyushu University, Japan. 26
- [Tateishi et Miyano, 1995] TATEISHI, E. et MIYANO, S. (1995). A greedy strategy for finding motifs from positive and negative examples. Rapport technique RIFIS-TR-CS-118, Research Institute of Fundamental Information Science, Kyushu University, Japan. 26
- [Taylor, 1986] TAYLOR, W. (1986). The classification of amino acid conservation. *Journal of theoretical biology*, 119:205–218. 79, 96
- [Tessier et al., 2004] TESSIER, D., BARDIAUX, B., LARRÉ, C. et POPINEAU, Y. (2004). Data mining techniques to study the disulfide-bonding state in proteins : signal peptide is a strong descriptor. *Bioinformatics*, 20(16):2509–2512. 95
- [Torre, 2000] TORRE, F. (2000). Intégration des biais de langage à l'algorithme générer-et-tester - contributions à l'apprentissage disjonctif. Thèse de Doctorat, Laboratoire de Recherche en Informatique Université Paris-Sud France. 67

- [Uemura *et al.*, 1999] UEMURA, Y., HASEGAWA, A., KOBAYASHI, S. et YOKOMORI, T. (1999). Tree adjoining grammars for rna structure prediction. *Theor. Comput. Sci.*, 210(2):277–303. 58
- [Veber *et al.*, 2005] VEBER, P., YANEV, N., ANDONOV, R. et POIRRIEZ, V. (2005). Optimal protein threading by cost-splitting. In *5th Workshop on Algorithms in Bioinformatics, WABI*, pages 365–375. 33
- [Vingron et Argos, 1991] VINGRON, M. et ARGOS, P. (1991). Motif recognition and alignment for many sequences by comparison of dot-matrices. *Journal of Molecular Biology*, 218:33–43. 24, 26
- [Vita *et al.*, 1998] VITA, C., VIZZAVONA, J., DRAKOPOULOU, E., ZINN-JUSTIN, S., GILQUIN, B. et MENEZ, A. (1998). Novel miniproteins engineered by the transfer of active sites to small natural scaffolds. *Biopolymers*, 47(1):93–100. 11
- [Vlijmen *et al.*, 2004] VLIJMEN, H. V., GUPTA, A., NARASIMHAN, L. et SINGH, J. (2004). A novel database of disulfide patterns and its application to the discovery of distantly related homologs. *Molecular Biology*, 335(4):1083–1092. 95
- [Vullo et Frasconi, 2003] VULLO, A. et FRASCONI, P. (2003). A recursive connectionist approach for predicting disulfide connectivity in proteins. In *SAC '03 : Proceedings of the 2003 ACM symposium on Applied computing*, pages 67–71, New York, NY, USA. ACM Press. 38
- [Vullo et Frasconi, 2004] VULLO, A. et FRASCONI, P. (2004). Disulfide connectivity prediction using recursive neural networks and evolutionary information. *Bioinformatics*, 20(5):653–659. 7, 36, 37, 38
- [Wang *et al.*, 1994] WANG, J. T., MARR, T. G., SHASHA, D. E., SHAPIRO, B. A. et CHIRN, G.-W. (1994). Discovering active motifs in sets of related protein sequences and using them for classification. *Nucleic Acids Research*, 22(14):2769–2775. 24
- [Ward *et al.*, 2003] WARD, J., MCGUFFIN, L., BUXTON, B. et JONES, D. (2003). Secondary structure prediction with support vector machines. *Bioinformatics*, 19(13):1650–5. 29, 31
- [Waterman *et al.*, 1984] WATERMAN, M. S., ARRATIA, R. et GALAS, D. J. (1984). Pattern recognition in several sequences : consensus ans alignment. *Bull. Math. Biol.*, 46:515–527. 24, 25
- [Waterman et Jones, 1990] WATERMAN, M. S. et JONES, R. (1990). Consensus methods for DNA and protein sequence alignment. *Methods in Enzymology*, 183:221–237. 24
- [Weiner *et al.*, 1984] WEINER, S., KOLLMAN, P. et AL. (1984). A new force field for molecular mechanical simulation of nucleic acids and proteins. *Journal of American Chemistry Society*, 106:765–784. 32
- [Wolferstetter *et al.*, 1996] WOLFERSTETTER, F., FRENCH, K., HERRMANN, G. et WERNER, T. (1996). Identification of functional elements in unaligned nucleic acid sequences by a novel tuple search algorithm. *Computer Applications in the Biosciences*, 12(1):71–80. 25

- [Won *et al.*, 2005] WON, K., HAMELRYCK, T., PRUGEL-BENNETT, A. et KROGH, A. (2005). Evolving hidden markov models for protein secondary structure prediction. *In Proceedings of IEEE Congress on Evolutionary Computation*,, pages 33–40. 29
- [Wu *et al.*, 2003] WU, C., YEH, L.-S., HUANG, H., ARMINSKI, L., CASTRO-ALVEAR, J., CHEN, Y., HU, Z., KOURTESIS, P., LEDLEY, R., SUZEK, B., VINAYAKA, C., ZHANG, J. et BARKER, W. (2003). The Protein Information Resource. *Nucleic Acids Res*, 31(1):345–7. 20
- [Wu et Brutlag, 1995] WU, T. D. et BRUTLAG, D. L. (1995). Identification of protein motifs using conserved amino acid properties and partitioning techniques. *In Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, ISMB 1995*, volume 3, pages 402–410. 24
- [Xu *et al.*, 2003] XU, J., LI, M., LIN, G., KIM, D. et XU, Y. (2003). Protein threading by linear programming. *Pac Symp Biocomput*, pages 264–75. 33
- [Yanev et Andonov, 2003] YANEV, N. et ANDONOV, R. (2003). Solving the protein threading problem in parallel. *In Workshop on HiCOMB'03, held in conjunction with 17th IPDPS Nice, France*, page 157. 33
- [Yanev et Andonov, 2004] YANEV, N. et ANDONOV, R. (2004). Parallel divide and conquer approach for solving the protein threading problem. *Concurrency and Computation : Practice and Experience*, 16(9):961–974. 33
- [Zhang et DeLisi, 1998] ZHANG, C. et DELISI, C. (1998). Estimating the number of protein folds. *Journal of Molecular Biology*, 284(5):1301–5. 32
- [Zhang *et al.*, 1992] ZHANG, X., MESIROV, J. et WALTZ, D. (1992). Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225(4):1049–63. 29

Résumé

Déterminer la structure 3D des protéines expérimentalement est une tâche très lourde et coûteuse, qui peut s'avérer parfois impossible à réaliser. L'arrivée massive de données provenant des programmes de séquençage à grande échelle impose de passer d'une approche biochimique à une approche bioinformatique, et nécessite en particulier de développer des méthodes de prédiction sur des séquences.

Cette thèse propose l'exploration de deux nouvelles pistes pour progresser dans la résolution de prédiction de ponts disulfures dans les protéines. Cette liaison covalente stabilise et contraint fortement la conformation spatiale de la protéine et la connaissance des positions où elle intervient peut réduire considérablement la complexité du problème de la prédiction de la structure 3D. Pour cela, nous utilisons dans un premier temps, l'inférence grammaticale et plus particulièrement les langages de contrôle introduit par Y. Takada, puis dans un deuxième temps, la programmation logique inductive.

Diverses expériences visent à confronter un cadre théorique d'apprentissage et des algorithmes généraux d'inférence grammaticale régulière à une application pratique de prédiction d'appariements spécifiques au sein d'une séquence protéique. D'autres expérimentations montrent que la programmation logique inductive donne de bons résultats sur la prédiction de l'état oxydé des cystéines en inférant des règles interprétables par les biologistes. Nous proposons un algorithme d'induction heuristique dont l'idée est d'effectuer plusieurs phases d'apprentissage en tenant compte des résultats obtenus aux phases précédentes permettant ainsi de diminuer considérablement la combinatoire dans les espaces d'hypothèses logiques en construisant des règles de plus en plus discriminantes.

Abstract

The main properties of proteins may be determined from their three-dimensional structure. Bridging this "knowledge gap" requires automatic methods capable of inferring folds from sequences. Our contribution to this ambitious goal addresses the prediction of particular interactions inside these macro-molecules. The tertiary folds of native proteins are defined by a large number of weak interactions. Proteins are also stabilized covalently by disulfide bonds formed by a pair of cysteine residues in the folded state. We focus on the prediction of these disulfide bonds formed by a pair of cystein residues in the folded state.

We propose to tackle with the problem of prediction of interactions between cysteins in the framework, in the first step, of grammatical inference, especially control sets introduced by Y. Takada and, in the second step, of inductive logic programming. A lot of experiments show that grammatical inference algorithms have to be improve to be able to work on protein sequences. On the over hand, we obtain explicit rules of prediction of oxidized state of cysteins which can be easily interpreted by biologists. We propose an heuristic induction algorithm which can be use for different problems. We obtain several rules and we can see that with only 17 rules, more than 90% of the two data bases we use are covered, which is a very good result given the simplicity of the characterization.