Inférence grammaticale sur des alphabets ordonnés : application à la découverte de motifs dans des familles de protéines

Aurélien Leroux

N° d'ordre: 3158

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Aurélien Leroux

Équipe d'accueil : Symbiose
École doctorale : MATISSE
Composante universitaire : IFSIC/IRISA

Titre de la thèse :

Inférence grammaticale sur des alphabets ordonnés : application à la découverte de motifs dans des familles de protéines

soutenue le 24/06/2005 devant la commission d'examen

M. :	Guy	Lorette	Président
MM. :	Colin	de la Higuera	Rapporteurs
	Laurent	Trilling	
MM. :	Marie-France	Sagot	Examinateurs
	François	Coste	
	Jacques	Nicolas	

À mes parents Brigitte et Philippe et à ma femme Éléna.

Remerciements

Merci, tout d'abord, à mon directeur de thèse Jacques Nicolas. Malgré un emploi du temps très serré, il a su se montrer disponible à mon égard et a pu, grâce à ses réponses précises à mes multiples interrogations, m'aider à trouver mon propre chemin dans la recherche.

Merci aussi à Colin de la Higuera et Laurent Trilling d'avoir accepté d'apporter leurs réflexions et jugements sur mon travail en dépit du peu de temps que je leur ai laissé.

Je remercie Marie-France Sagot, François Coste et Guy Lorette pour avoir acceptés d'être membres du jury. Les travaux de Marie-France Sagot m'ont beaucoup inspirés dans mes recherches et sa participation au jury me fait honneur. Au cours de ma première année de thèse, François Coste m'a beaucoup aidé à m'immerger dans le domaine de l'inférence grammaticale. Guy Lorette me fait la faveur de présider le jury.

Merci également à Daniel Fredouille pour les nombreuses et enrichissantes discussions qui m'ont permis de faire évoluer mes idées et de les conforter.

Merci aussi à Israël-César Lerman et Tallur Basavanneppa. Leur compétence et leur expérience dans le domaine de la statistique m'ont permis d'enrichir mes connaissances dans ce domaine.

Je voudrais remercier l'équipe Symbiose dans son ensemble pour sa gaieté, sa chaleur et son dynamisme : Anne-sophie, Cynthia, Elodie, Emmanuelle, Esther, Ingrid, Mathieu, Patrick, Stephane, Yoann, Yves et tous les autres pour les discussions intéressantes, les matchs de ping pong acharnés et les soirées inoubliables.

Enfin, je remercie tout particulièrement mon frère Mathieu pour la correction orthographique et grammaticale de ce texte et surtout ma femme Éléna, à qui je dois ce document, de m'avoir soutenu et aidé tout au long de la rédaction.

Aurélien Leroux.

Table des matières

Introduction

Dans les organismes vivants, un ensemble très important de molécules sont en activité permanente. Parmi celles-ci, on trouve les protéines qui sont des enchaînements de quelques centaines de plus petites molécules appelées acides aminés. Bien que depuis de nombreuses années ces molécules aient été largement étudiées, la difficulté et le coût des expérimentations à réaliser pour progresser dans ce domaine font que les connaissances sur le sujet sont encore très insuffisantes.

Les protéines sont apparues petit à petit au cours de l'évolution et sont en perpétuelle mutation. L'apparition d'une nouvelle protéine se fait généralement par copie (inexacte) d'une autre. Les protéines possèdent des fonctions très diverses et sont classées par familles partageant une même propriété (par exemple même structure ou même fonction). Elles présentent donc certaines ressemblances au niveau des acides aminés.

Prenons un ensemble de protéines de la même famille. Parce qu'elles partagent certaines propriétés, leur séquence d'acides aminés présentent des ressemblances qui les distinguent des protéines appartenant à une autre famille. La ressemblance des protéines se traduit à travers celle des acides aminés qui n'est pas simple à mesurer. En effet, le rôle des acides aminés dépend de plusieurs facteurs environnementaux comme leur contexte séquentiel ou spatial. Cette ressemblance a été mesurée de nombreuses façons plus ou moins complexes depuis plusieurs décennies [Sagot 1996, Jiménez Montaño et Ramos Fernández 2004]. Elle est souvent calculée dans un cadre précis, fixé à l'avance.

Objectifs de nos travaux et principaux résultats obtenus. Dans la thèse, nous abordons principalement deux problèmes.

- Le premier concerne la mesure de la ressemblance des acides aminés sans supposer *a priori* que l'on se situe dans un cadre particulier. Le cadre sera déterminé *a posteriori* par l'analyse et la comparaison de l'ensemble des protéines de la famille que nous étudions.
- Le second problème traite de l'élaboration d'un modèle pour une famille de protéines en se fondant sur la ressemblance des acides aminés. Il existe plusieurs façons de résoudre ce deuxième problème, par exemple, la *décou-*

verte de motifs ou l'*inférence grammaticale*. Les méthodes de découverte de motifs donnent de bons résultats mais ont un pouvoir d'expression faible. Pour cette raison, nous nous plaçons dans le cadre de l'inférence grammaticale. Ce domaine de recherche qui consiste à apprendre, à partir d'un ensemble de données, une grammaire inconnue qui les explique [Higuera (de la) 2002], est apparu [Gold 1967] peu après la formalisation de la notion de langage [Chomsky 1956]. Les méthodes d'inférence grammaticale existantes donnent de bons résultats sur les langages formels. Leur application à des données non mathématiques, par exemple des séquences protéiques, ne produit pas les résultats espérés.

Dans cette thèse, nous proposons une nouvelle méthode heuristique d'inférence grammaticale fondée sur la ressemblance des acides aminés et adaptée au traitement de familles de protéines. Le modèle produit par cette méthode appartient à la classe des langages réguliers et consiste en un automate non déterministe. Nous avons expérimenté notre méthode sur des données artificielles et sur des données biologiques. Les résultats montrent que l'heuristique proposée est tout à fait compétitive par rapport aux méthodes existantes (par exemple Teiresias [Rigoutsos et Floratos 1998a;b] et Pratt [Jonassen *et al.* 1995, Jonassen 1997]). En effet, sur des données artificielles modélisant une certaine réalité biologique, les résultats qu'elle produit sont meilleurs que ceux des autres méthodes et sur des données biologiques réelles, ils sont équivalents.

La thèse est divisée en six chapitres.

Chapitre 1 Ce chapitre établit un rapide état de l'art de la découverte de motifs dans les séquences. Nous commençons par présenter tout l'intérêt de la découverte de motifs dans les protéines. Nous définissons ensuite le cadre formel de l'apprentissage. Enfin, nous présentons le problème de la découverte de motifs et nous proposons une vue synthétique des différentes approches abordées pour le traiter.

Chapitre 2 Dans ce chapitre, nous donnons des notions sur l'inférence grammaticale de langages réguliers représentés par des automates d'états finis. Ces langages, bien qu'étant les moins expressifs de la hiérarchie de Chomsky, permettent d'approximer la plupart des langages intéressants. Ils sont plus généraux que les motifs classiques puisqu'ils permettent d'introduire la disjonction de manière non contrainte et les répétitions. Dans la première partie du chapitre, nous définissons les langages réguliers et leurs différentes représentations en mettant l'accent sur les automates d'états finis. Dans la deuxième partie, nous exposons le problème de l'inférence grammaticale des langages réguliers et introduisons deux modèles d'apprentissage parmi les plus importants : l'identification à la limite à partir de données à la demande et à partir de données fixées. Nous présentons les méthodes d'inférence as-

sociés.

Chapitre 3 Dans ce chapitre, nous retravaillons la représentation utilisée pour décrire les langages réguliers et nous introduisons un nouveau type d'automate dont le pouvoir d'expression est identique à celui des automates d'états finis. Ce type d'automate, que nous l'appelons automate de transitions finies (FTA), est mieux adapté à la modélisation et l'étude des protéines. Dans la première section du chapitre, nous donnons les motivations qui nous ont amené à proposer un tel modèle. Puis, nous définissons les automates de transitions. Dans la troisième section, nous montrons que les automates de transitions et les automates d'états représentent la même classe de langages et nous établissons les méthodes de transformation entre les deux modèles. Nous approfondissons ensuite les liens entre les automates de transitions et les langages réguliers. La compacité de la représentation est établie de façon précise en comparant la taille des automates de transitions et des automates d'états. Enfin, dans la dernière section, nous définissons la notion d'automate de transitions finies minimal.

Chapitre 4 Dans ce chapitre, nous abordons le problème de la prise en compte dans l'inférence de l'information sur les propriétés physico-chimiques entre les acides aminés. Dans la première section, nous décrivons brièvement les acides aminés et leur rôle dans la structure des protéines. Nous passons ensuite en revue les propriétés physico-chimiques des acides aminés et montrons leur importance dans l'étude des protéines. La troisième section, correspond à un état de l'art succinct sur la prise en compte de ces propriétés dans l'établissement d'une ressemblance entre acides aminés. Dans la quatrième section, nous donnons notre approche de la ressemblance entre acides aminés, en proposant une méthode de transformation automatique d'une couverture en une relation d'ordre partielle sous forme de treillis, exploitable par une méthode d'inférence grammaticale.

Chapitre 5 Ce chapitre décrit notre méthode d'inférence grammaticale à partir d'instances positives pour l'apprentissage de motifs dans les séquences protéiques. Le but de cette méthode est de calculer, à partir d'un ensemble de séquences protéiques appartenant à une même famille, un automate non déterministe représentant le langage décrivant au mieux les séquences de l'ensemble de départ. Cette méthode d'inférence grammaticale est fondée sur les alignements de séquences et s'inspire fortement des méthodes de découverte de motifs. Elle se divise en trois étapes. La première calcule des statistiques sur l'échantillon de départ donnant ainsi une connaissance globale des séquences. La deuxième construit des alignements à partir des données statistiques et de l'ordre sur les groupes d'acides aminés défini sous forme de treillis au chapitre précédent. La troisième étape construit un

automate dont les transitions correspondent aux différentes positions des alignements sélectionnés et qui constitue le modèle décrivant la famille de protéines donnée.

Chapitre 6 Dans ce chapitre nous présentons les particularités de l'implémentation de la méthode SDTM (*Similarity Driven Transition Merging*)[1] qui a été réalisée en Prolog et les expérimentations que nous avons menées. Nous proposons une représentation des automates de transitions finies non déterministes, particulièrement adaptée à la programmation logique et à la fusion de transitions. Avec cette structure, la fusion de deux transitions correspond à leur unification. La fin du chapitre présente les résultats de notre méthode et la comparaison avec Teiresias et Pratt sur des données artificielles puis sur des données réelles.

[1] Fusion de transition dirigée par la similarité en français

Chapitre 1

Motifs dans les protéines

Ce chapitre établit un rapide état de l'art de la découverte de motifs dans les séquences. Nous commençons par présenter tout l'intérêt de la découverte de motifs dans les protéines. Nous définissons ensuite le cadre formel de l'apprentissage. Enfin, nous présentons le problème de la découverte de motifs et nous proposons une vue synthétique des différentes approches abordées pour le traiter. Pour écrire cette section, nous nous sommes basés principalement sur les articles de [Brazma *et al.* 1998], [Hudak et Mcclure 1999] *et* [Brejová *et al.* 2000] *que le lecteur pourra consulter pour plus de détails.*

1.1 Conservation dans les protéines

Les protéines sont des molécules que l'on peut décrire comme un assemblement d'acides aminés (ou résidus). Certains de ces résidus jouent un rôle important dans la fonction de la protéine (résidus de surface) ou dans la formation de la structure du cœur de la protéine (résidus enfouis) et sont conservés au cours de l'évolution [Zvelebil et Sternberg 1988], [Casari *et al.* 1995], [Branden et Tooze 1999], [Armon *et al.* 2001].

Dans le cas des résidus enfouis, la conservation est due au fait que la structure compacte des protéines ne tolère que les remplacements conservatifs d'acides aminés, tandis que les remplacements radicaux, comme des échanges entre des résidus de tailles différentes, déstabilisent souvent la structure de la protéine et impliquent le dysfonctionnement de la protéine.

Les protéines interagissent presque toujours avec d'autres molécules pour exercer leurs fonctions biologiques. Ces interactions incluent, par exemple, la liaison de ligands dans les sites récepteurs, les interactions protéine-ADN ou les interactions protéine-protéine. Les facteurs clés dans toutes ces interactions sont la forme et les propriétés chimiques de la surface de la protéine. Ainsi, les sites actifs des

enzymes sont souvent caractérisés par un sillon particulièrement large et profond, tandis qu'à l'inverse les interactions entre les molécules d'un dimère impliquent plutôt des surfaces planes.

Ces observations laissent supposer que les résidus fonctionnellement importants, qui sont impliqués dans la reconnaissance moléculaire entre les protéines (ou entre des protéines et de l'ADN) ou dans l'activité enzymatique, doivent être conservés durant l'évolution [Meyer *et al.* 1994], [Pazos *et al.* 1997], [Lockless et Ranganathan 1999], [Kisters-Woike *et al.* 2000], [Gallet *et al.* 2000]. Ainsi, les résidus des sites de liaison protéine-protéine sont mieux conservés que sur le reste de la surface de la protéine [Hu *et al.* 2000], [Goh *et al.* 2000], [Valdar et Thornton 2001], [Goh et Cohen 2002], [Valencia et Pazos 2002], [Lichtarge et Sowa 2002], [Pupko *et al.* 2002], [Glaser *et al.* 2003], [Ma *et al.* 2003] et, souvent, co-évoluent [del Sol Mesa *et al.* 2003].

1.2 Problématique de la thèse

La recherche de similitudes dans des ensembles de séquences de protéines est devenue très importante pour l'annotation de génomes et de protéomes. Il existe déjà un grand nombre de méthodes de découverte de motifs. Elles recherchent la présence de motifs simples dans les familles de protéines. L'extraction de motifs caractéristiques, pour des familles de protéines, peut donner quelques indices sur les sites actifs, la localisation dans la cellule et même la fonction de ces protéines.

Dans ce chapitre, nous passons en revue des méthodes qui modélisent les motifs de manière explicite. Nous ne nous intéressons pas à celles qui produisent des modèles probabilistes de motifs comme les matrices *profile* [Gribskov *et al.* 1990] ou les modèles de Markov cachés [Krogh *et al.* 1994, Hughey et Krogh 1996].

Du point de vue de la théorie des langages formels, les classes de langages qui peuvent être apprises avec les méthodes actuelles de découverte de motifs sont des sous-ensembles des langages réguliers, sans l'étoile de Kleene (pas de boucle) et avec des disjonctions ponctuelles limitées à des mots de longueur 1 (les caractères ambigus sont autorisés mais pas les alternatives entre des sous-séquences de longueur supérieure à 2). La première restriction ne paraît pas essentielle dans le contexte des protéines, sauf pour la modélisation de régions particulières comme les régions avec des répétitions. La deuxième, cependant, limite inutilement le type des motifs qui peuvent être inférés. Notre but est d'explorer des méthodes qui considèrent des classes plus larges de langages réguliers. Les séquences protéiques sont vues comme des chaînes d'un langage régulier inconnu à apprendre. Dans ce contexte, la découverte de motifs devient un problème d'inférence grammaticale. Dans cette thèse, nous présentons une nouvelle méthode qui permet de décrire de façon plus précise les similitudes sous forme de langages réguliers.

1.3 Apprentissage

1.3.1 Cadre général de l'apprentissage

Certaines parties des séquences protéiques, conservées au cours de l'évolution, se retrouvent dans des protéines que l'on va vouloir rapprocher à cause de leur propriétés communes. Cette conservation est due, comme nous l'avons vu dans la section précédente, au fait que certaines régions d'une protéine ont un rôle particulier. Les méthodes d'apprentissage permettent dans certains cas d'identifier ces régions *a priori* inconnues et peuvent ainsi être utilisées pour progresser dans la connaissance des fonctions de protéines pour lesquelles on dispose seulement de la séquence. Les problèmes d'apprentissage peuvent être divisés en deux classes principales. Pour introduire ces problèmes nous donnons quelques notations.

Notations On considère une famille de protéines, représentée comme un ensemble F_+ de mots sur l'alphabet des acides aminés. On note F_- l'ensemble des mots n'appartenant pas à la famille. On appelle *fonction de chaîne* une fonction f qui affecte à chaque mot composé d'acides aminés une valeur booléenne vrai ou faux. La fonction f est une *fonction caractéristique* de la famille F_+ si :

$$f(s) = \begin{cases} \text{vrai} & \text{si } s \in F_+ \\ \text{faux} & \text{si } s \in F_- \end{cases}$$

En apprentissage, on cherche à trouver automatiquement la fonction de chaîne qui est caractéristique d'une famille F_+ donnée de protéines. Malheureusement, on ne dispose pas de l'ensemble complet des séquences de F_+, mais seulement d'un échantillon $\mathcal{S}_+ \subseteq F_+$. Parfois, on a aussi à notre disposition un ensemble de séquences n'appartenant pas à la famille. On parle alors d'échantillon négatif et on le note $\mathcal{S}_- \subseteq F_-$. La fonction de chaîne $f(s)$ caractéristique d'une famille donnée peut, en théorie, ne pas être calculable.

Problèmes d'apprentissage Les notations que nous avons introduites au paragraphe précédent nous permettent de formuler les deux classes de problèmes d'apprentissage :

La famille des problèmes de discrimination (c'est-à-dire : on a deux ensembles de mots et on cherche les différences entre ces mots. En d'autres termes, on cherche les régularités qui sont présentes dans l'un des groupes mais pas dans l'autre). Supposons que nous disposons d'échantillons positifs et négatifs $\mathcal{S}_+ \subseteq F_+$ et $\mathcal{S}_- \subseteq F_-$. Le but est de trouver une fonction de chaîne qui approche la fonction caractéristique pour la famille F_+. Pour ce

faire, nous voulons à la fois exprimer de façon plus compacte les propriétés des séquences connues et prédire l'appartenance de séquences encore inconnues à F_+.

La famille des problèmes de conservation (c'est-à-dire : on cherche les régularités des séquences). Supposons que nous disposons uniquement d'un échantillon positif $\mathcal{S}_+ \subseteq F_+$ de la famille. Le but est de trouver les caractéristiques des séquences de $\mathcal{S}_+$. Les fonctions de chaîne les plus intéressantes dans ce cas sont celles qui ont une faible probabilité de rendre vrai pour des séquences aléatoires et une grande probabilité de rendre vrai pour les séquences de F_+. Un problème de conservation est similaire à un problème de classification dans lequel on a les échantillons $\mathcal{S}_+$ et $\mathcal{S}_-$ où l'échantillon $\mathcal{S}_-$ est un ensemble de séquences choisies au hasard dans F_-.

1.3.2 Le problème de la découverte de motifs

En découverte de motifs, les fonctions de chaîne s'expriment en termes de motifs π présents ou non dans les séquences. Les fonctions de chaîne $f_\pi(s)$ considérées sont telles que :

$$f_\pi(s) = \begin{cases} \text{vrai} & \text{si le motif } \pi \text{ est présent dans la séquence } s \\ \text{faux} & \text{sinon.} \end{cases}$$

Le problème posé par la découverte de motifs est, étant donné un échantillon $\mathcal{S}_+$ et, parfois, un échantillon $\mathcal{S}_-$, de découvrir automatiquement les motif π présents dans les séquences de l'ensemble $\mathcal{S}_+$ tels que les fonctions de chaîne f_π approchent au mieux la fonction caractéristique f de la famille F_+.

1.4 Méthodes de découverte de motifs

Un grand nombre de méthodes de découverte de motifs ont été développées pour l'étude des séquences biologiques. Dans cette section, nous proposons une vue générale de ceux dédiés à l'étude des molécules auxquelles nous nous intéressons dans cette thèse, c'est-à-dire les protéines. Nous ne parlons pas ici des méthodes spécialisés dans le traitement des séquences d'ADN malgré la richesse des travaux récents dans ce domaine. Le faible nombre d'études récentes dans le domaine des protéines peut s'expliquer par le fait que les données actuelles sont des données transcriptome et laisse penser que les méthodes dans ce domaine sont arrivés à maturité. Mais ces méthodes restent peu employées en pratique et les biologistes préfèrent utiliser Prosite [Bairoch 1992]. Il existe aussi des méthodes de localisation de motifs parmi lesquelles on trouve MODEL [Hernandez *et al.* 2004] et PIMA [Smith et Smith 1992]. Ces méthodes sont intéressantes

car, contrairement à celles de découverte de motifs, elles identifient la position de régions conservées dans les séquences. Nous commençons par rappeler la classification des motifs proposée dans [Brazma *et al.* 1998]. Ensuite, nous présentons les différentes approches utilisées pour la découverte de motifs. Nous verrons que les méthodes de découverte de motifs sont dirigées soit par le modèle, soit par les données. Nous finissons la section par une liste de méthodes combinant les deux approches.

1.4.1 Classes de motifs de Brazma

Dans l'article [Brazma *et al.* 1998], A. Brazma et ses collaborateurs ont proposé une classification des motifs que nous reprenons dans le tableau 1.1). Un motif est une formule (dont le pouvoir d'expression est plus faible que celui des langages réguliers) qui représente une région conservée d'une famille de protéines. Dans ce tableau on utilise les notations suivantes :

(1) Σ est l'ensemble des lettres à partir desquelles sont construits les mots,

(2) Π est l'ensemble des parties de Σ de taille comprise entre 2 et $|\Sigma| - 1$,

(3) $x = \Sigma$ est appelé joker (de longueur 1),

(4) X est l'ensemble des jokers de taille variable de la forme $x(p, q) = \bigcup_{p \leq r \leq q} \Sigma^r$,

(5) $*$ est le joker de taille arbitraire,

Classe	**Définition**	**Exemple de motif**
A	$\pi \in \Sigma^*$	t-c-t-t-g-a
B	$\pi \in (\Sigma \cup \{x\})^*$	D-R-C-C-x(2)-H-D-x-C
C	$\pi \in (\Sigma \cup \Pi)^*$	G-G-G-T-F-D-[ILV]-[ST]-[ILV]
D	$\pi \in (\Sigma \cup \Pi \cup \{x\})^*$	V-x-P-x(2)-[RQ]-x(4)-G-x(2)-L-[LM]
E	$\pi \in (\Sigma \cup X)^*$	G-C-x(1,3)-C-P-x(8,10)-C-C
F	$\pi \in (\Sigma \cup \Pi \cup X)^*$	C-x(2,4)-C-x(3)-[ILVFYC]-x(8)-H-x(3,5)-H
G	$\pi \in (\Sigma \cup \{*\})^*$	D-T-A-G-Q-E-*-L-V-G-N-K
H	$\pi \in (\Sigma \cup \Pi \cup \{*\})^*$	D-T-A-G-[NQ]-*-L-V-G-N-[KEH]
I	$\pi \in (\Sigma \cup \Pi \cup X \cup \{*\})^*$	D-T-A-x(2,5)-G-[NQ]-*-L-V-G-N-[KEH]

TAB. 1.1 – Définition des classes de motifs

Nous utilisons ce tableau pour classifier les méthodes de découverte de motifs existantes qui sont montrées dans le tableau 1.2. Dans les sections suivantes nous détaillons certaines de ces méthodes. La deuxième colonne du tableau 1.2 donne les caractéristiques des méthodes. Elle est divisée en quatre parties. *Motif* indique la classe des motifs générés. *Pre.* correspond au pré-requis de la méthode et prend la valeur *N* si seules les séquences à étudier sont nécessaires, *Alignées* si les séquences fournies doivent être alignées et *Taille* si la longueur des motifs doit

Référence	**Méthode** Motif	Pre.	Échant.	Domaine	**Nom**
[Waterman *et al.* 1984]	Ab	N	+	protéine, ADN	
[Martinez 1988]	Gb	N	+	protéine, ADN	GENALIGN
[Landraud *et al.* 1989]	Gb	N	+	protéine, ADN	
[Smith et Smith 1990]	Fa	N	+	protéine	
[Smith *et al.* 1990]	Ba	N	+	protéine	MOTIF
[Waterman et Jones 1990]	Fa	N	+	protéine, ADN	PRALIGN
[Vingron et Argos 1991]	Ga, Gb	N	+	protéine	FILTER
[Ogiwara *et al.* 1992]	Ga	N	+/−	protéine	
[Roytberg 1992]	Ab	N	+	protéine, ADN	MuSCo
[Smith et Smith 1992]	Fa	N	+	protéine	PIMA
[Arikawa *et al.* 1993]	Gd	N	+/−	protéine	
[Neuwald et Green 1994]	Da	N	+	protéine	ASSET
[Saqi et Sternberg 1994]	Ca	N	+	protéine	
[Wang *et al.* 1994]	Gb	N	+	protéine	DISCOVER, CLASSIFY
[Jonassen *et al.* 1995]	Fa	N	+	protéine	Pratt
[Sagot *et al.* 1995c]	Ca	N	+	protéine	
[Sagot *et al.* 1995b]	Ab	N	+	protéine	
[Shoudai *et al.* 1995]	Fc, Fd	N	+/−	protéine	BONSAI
[Suyama *et al.* 1995]	Ea	N	+	protéine	GAPE
[Wu et Brutlag 1995]	Ca	Alignées	+	protéine	SEQCLASSx
[Sagot et Viari 1996]	Da	N	+	protéine, ADN	
[Brazma *et al.* 1996]	Fc	N	+	protéine	MDLPratt
[Grundy *et al.* 1997]	Da	N	+	protéine	Meta-MEME
[Jonassen 1997]	Fa	N	+	protéine	Pratt2
[Nevill-Manning *et al.* 1997]	Fa	Alignées	+	protéine	EMOTIF
[Rigoutsos et Floratos 1998a]	Ba	N	+	protéine, ADN	Teiresias
[Guralnik et Karypis 2001]	Aa	Taille	+	protéine	

TAB. 1.2 – Méthodes de découverte de motifs sous forme de chaîne dans les protéines

être connue. *Échant.* correspond à l'échantillon fournit. L'échantillon positif est indiqué par un + et l'échantillon négatif par un −. *Domaine* précise le domaine d'application des méthodes et prend la valeur *protéine* si elle est applicable aux protéines et *ADN* si elle est applicable à l'ADN.

1.4.2 Méthodes dirigées par le modèle

Les méthodes dirigées par le modèle sont les méthodes les plus basiques. Les méthodes directes, les plus simples, procèdent par énumération explicite de l'ensemble des motifs de la classe A [Queen et Wegman 1982], [Waterman *et al.* 1984], [Staden 1989], [Wolferstetter *et al.* 1996]. Ces méthodes sont très coûteuses et permettent de générer uniquement des motifs dont le pouvoir d'expression est faible. 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 qui comportent un mot proche du motif. Ces approches ont été étendues à des motifs plus complexes. Celle décrite dans [Smith *et al.* 1990] génère des motifs de la classe B. L'extension de cette méthode proposée par [Suyama *et al.* 1995] permet de découvrir des motifs de la classe E.

Le problème des approches par énumération est l'efficacité. La taille de l'espace de motifs peut néanmoins être réduit par la restriction de la classe des motifs qui est en général à la charge de l'utilisateur, via le réglage d'un nombre plus où moins important de paramètres. Pour des classes plus générales de motifs, cette méthode devient de toute façon inutilisable. Dans le but de réduire la complexité, des techniques d'élagage de l'espace de solutions ont été proposées, soit par une méthode dont la précision peut être bornée soit par l'utilisation d'heuristiques.

De nombreuses méthodes procédant par énumération avec élagage existent. Dans la suite, nous décrivons plusieurs approches associées aux problèmes de conservation. Une des façons d'élaguer l'espace de recherche est de le représenter sous forme d'arbre et d'élaguer les sous-arbres dont la racine correspond à un motif avec un score faible. Le parcours de cet arbre peut être effectué en largeur d'abord ou en profondeur d'abord. La première approche permet un élagage plus efficace en pratique [Sagot *et al.* 1995c]. Cependant, ce type de recherche ne peut être effectuée que pour des motifs très courts. [Karp *et al.* 1972] ont implémentés de façon très efficace cette idée. [Sagot *et al.* 1995c] ont étendu cette idée pour trouver des motifs plus expressifs (classe C). L'efficacité de cette méthode dépend fortement de la taille des groupes présents dans les motifs. Ainsi, si certains groupes sont très grands, l'arbre de recherche devient trop gros. Ceci a contraint [Sagot *et al.* 1995c] à interdire les jokers.

[Neuwald et Green 1994] décrivent une méthode simple, en profondeur d'abord, pour obtenir des motifs de la classe D autorisant la présence de groupes de deux lettres. L'idée est que seuls les motifs de score suffisamment élevé sont étendus

par des jokers. Le mécanisme d'élagage est basé sur la mesure de la significativité statistique des motifs.

[Sagot et Viari 1996] présentent une approche alternative produisant des motifs de classe D, en profondeur d'abord, autorisant les jokers et les groupes d'acides aminés. Suivant les groupes que l'on va permettre ou pas, la complexité sera très différente.

[Jonassen *et al.* 1995] proposent une recherche en profondeur d'abord en autorisant la présence d'un motif dans un sous-ensemble des séquences d'apprentissage comme dans l'approche de [Neuwald et Green 1994]. Les motifs produits sont de classe F avec des restrictions définies par l'utilisateur. La méthode procède en deux phases. Durant la première, les meilleurs motifs contenant des jokers, mais aucun groupe de Π, sont calculés. Lors de la deuxième phase, les motifs de la première sont soumis à une recherche soit exhaustive, soit heuristique permettant l'addition des groupes d'acides aminés. L'approche garantit de trouver les meilleurs motifs (suivant un score), dans la sous-classe de F, qui sont présents dans au moins un nombre fixé d'exemples positifs.

[Jonassen 1997] a proposé une méthode heuristique très proche de [Jonassen *et al.* 1995] pour améliorer l'efficacité, principalement dans le cas de séquences très similaires.

Les méthodes que nous avons décrit précédemment appartiennent à la classe des problèmes de conservation. Il existe aussi des méthodes par énumération avec élagage de la classe des problèmes de classification comme par exemple celui de [Ogiwara *et al.* 1992].

Malgré les améliorations successives de l'efficacité, les approches dirigés par le modèle sont toujours de complexité exponentielle en la longueur des motifs.

1.4.3 Méthodes dirigées par les données

Il existe aussi plusieurs méthodes dirigées non pas par le modèle mais par les données que l'on peut classer en deux catégories. Ces méthodes heuristiques sont basées sur la comparaison, soit des meilleures paires de séquences, soit de toutes les paires. Dans cette section, nous présentons des méthodes de la classe des problèmes de conservations. Il existe aussi des approches, peu nombreuses, qui disposent à la fois d'exemples positifs et négatifs comme par exemple ceux de [Kudo *et al.* 1992], [Tateishi et Miyano 1995] et [Tateishi *et al.* 1995].

Heuristiques basées sur la comparaison des meilleures paires Une partie des méthodes de découverte de motifs dirigées par les données suivent la procédure suivante. La méthode prend en entrée un ensemble de séquences $S = \{s_1, \ldots, s_n\}$

et les traduit en un ensemble de motifs $M = \{m_1, \ldots, m_n\}$ où chaque m_i couvre une seule séquence (c'est-à-dire que pour tout i compris entre 1 et n, $m_i = \{s_i\}$) est considérée comme un motif. Ensuite, il calcule itérativement la meilleure paire d'ensembles de motifs de M, élimine les ensembles de motifs de cette paire de M et ajoute à M un ensemble, construit par généralisation des éléments de cette paire. Le résultat est un ensemble de motifs.

Différentes heuristiques ont été proposées pour sélectionner la meilleure paire de motifs à réunir à chaque étape. [Shinohara 1983] et [Nix 1983] ont par exemple proposé une heuristique qui produit des motifs de la classe G. À chaque étape, la meilleure paire sélectionnée est celle qui contient les séquences ou motifs les plus courts. La plus longue sous-séquence commune est extraite et est ajoutée à l'ensemble M. Une autre heuristique construit un arbre phylogénétique binaire des séquences à analyser suivant leurs distances respectives [Smith et Smith 1990]. Ensuite, l'ordre des réunions correspond à celui indiqué par l'arbre en allant des feuilles vers la racine.

Heuristiques basées sur la comparaison de toutes les paires Une autre partie des méthodes de découverte de motifs dirigées par les données utilisent une heuristique basée sur la comparaison de toutes les paires. La méthode commence par calculer les alignements entre toutes les paires de séquences. À chaque pas, à partir des meilleurs alignements entre n séquences obtenus à l'étape précédente, elle calcule les alignements entre $n + 1$ séquences. Il existe plusieurs heuristiques qui permettent de comparer toutes les paires de séquences. Parmi celles-ci on trouve celles de [Schuler *et al.* 1991] ou [Brodsky *et al.* 1992] où les alignements entre n séquences sont représentés par des blocs de taille n, et celle de [Vingron et Argos 1991] dans laquelle les alignements sont représentés par des matrices de points.

1.4.4 Approches combinant les deux méthodes

Beaucoup de travaux combinent les méthodes décrites dans les deux sections précédentes. La plupart du temps, les méthodes dirigées par les séquences sont utilisées pour raffiner les motifs trouvés par les méthodes dirigées par le modèle. Cette combinaison peut être réalisée de différentes façons. l'une d'elles consiste à utiliser une méthode dirigée par le modèle pour identifier des motifs candidats. Ensuite, les séquences sont alignées en respectant l'alignement des lettres du motif candidat. Enfin, les motifs candidats sont étendus tant que le score du nouveau motif est plus élevé. Cette méthode est utilisée par [Smith *et al.* 1990], [Jonassen *et al.* 1995], [Jonassen 1997]. Des variations de cette méthode ont aussi été proposées [Landraud *et al.* 1989], [Martinez 1988]. D'autre méthodes pour raffiner les motifs existent, parmi lesquelles [Ogiwara *et al.* 1992], [Neuwald et

Green 1994], [Saqi et Sternberg 1994], [Henikoff et Henikoff 1991].

Il est aussi possible de limiter, *a priori*, l'espace de recherche d'une méthode dirigée par le modèle, à l'aide d'une méthode dirigée par les données [Saqi et Sternberg 1994], [Jonassen 1997].

1.5 Discussion

Dans ce chapitre, nous avons présenté les méthodes de découverte de motifs. Ces méthodes sont les plus usitées pour identifier les régions conservées dans les protéines et donnent de bons résultats. Par contre, l'expressivité des motifs générés ne permet pas de décrire précisément les similitudes entre les protéines. La limitation principale des motifs que nous avons présentés dans ce chapitre est qu'ils n'autorisent que les disjonctions ponctuelles (au niveau de chacune des positions) et ne permettent de représenter qu'une petite sous-partie des langages réguliers. Certaines tentatives ont été réalisées pour aller au-delà de la classe J des motifs. Certaines approches génèrent des unions de motifs ou des arbres de décision sur les motifs [Arikawa *et al.* 1992], [Arikawa *et al.* 1993], [Arimura *et al.* 1994], [Shoudai *et al.* 1995]. D'autres génèrent des motifs reliés [Brazma *et al.* 1997] qui permettent de considérer certaines relations de dépendance entre les acides aminés par l'introduction de positions corrélées. Ces extensions, bien qu'intéressantes, ne permettent pas d'obtenir le pouvoir d'expression des langages réguliers.

D'autres méthodes d'apprentissage moins répandues existent parmi lesquelles on trouve l'inférence grammaticale régulière que nous présentons dans le chapitre suivant. Les méthodes d'inférence grammaticale, souvent coûteuses, présentent l'avantage de décrire les conservations des protéines à l'aide de représentations dont le pouvoir d'expression est plus grand que celui des motifs. Cependant, ces méthodes, appliquées aux protéines produisent souvent des modèles trop généraux pour être exploitables.

Dans cette thèse, nous proposons d'utiliser les techniques de la découverte de motifs pour générer, par inférence grammaticale, des modèles dont le pouvoir d'expression est celui des langages réguliers. En général les méthodes d'inférence grammaticale cherchent des ressemblances globales entre les séquences. Les méthodes de découverte de motifs cherchent plutôt des ressemblances linéaires entre séquences. La comparaison d'arbres est plus précise que la comparaison de séquences car elle met en jeu un plus grand nombre de séquences. Cependant, les méthodes de comparaison d'arbres sont beaucoup plus complexes que celles utilisées pour comparer des séquences et donnent souvent des résultats trop approximatifs pour être utilisés. Nous proposons dans cette thèse une méthode d'inférence grammaticale qui identifie les régions conservées des protéines par comparaison

de séquences. Cette méthode génère, dans un premier temps, des motifs qui sont ensuite synthétisés pour former un automate donnant une représentation d'un langage régulier.

Chapitre 2

Inférence grammaticale de langages réguliers

Bien qu'étant les moins expressifs de la hiérarchie de Chomsky, les langages réguliers permettent d'approximer la plupart des langages intéressants. Ils sont aussi plus généraux que les motifs classiques puisqu'ils permettent d'introduire la disjonction de manière non contrainte et les répétitions. Dans ce chapitre, nous donnons des notions sur l'inférence grammaticale de tels langages représentés par des automates d'états finis. Dans la première section nous définissons les langages réguliers et leurs différentes représentations en mettant l'accent sur les automates d'états finis. Dans la deuxième partie, nous exposons le problème de l'inférence grammaticale des langages réguliers et introduisons deux modèles d'apprentissage parmi les plus importants : l'identification à la limite à partir de données à la demande et à partir de données fixées. Nous présentons les algorithmes d'inférence associés.

2.1 Théorie des langages

Dans cette section, nous introduisons les notions de théorie des langages nécessaires à la compréhension de cette thèse. Nous nous sommes essentiellement basés sur les ouvrages de Yu [Yu 1997] et de Hopcroft et Ullman [Hopcroft et Ullman 1980] que le lecteur pourra consulter pour plus de détails.

2.1.1 Mots et langages

Un *mot* (ou *séquence*) est une suite finie de symboles. La longueur d'un mot w, notée $|w|$, est le nombre de symboles le composant. Le mot vide, noté ε, est

le mot de longueur nulle.

La *concaténation* de deux mots est le mot formé du premier mot suivi du deuxième sans espace entre les deux mots. On note $u.v$ (ou simplement uv) la concaténation des mots u et v. ε est l'élément neutre de la concaténation.

Soient x, y, z trois mots et w le mot tel que $w = xyz$. Alors, x, y et z sont appelés respectivement *préfixe*, *facteur* et *suffixe* de w. Si le facteur y est non vide, alors y peut s'écrire $w[i, j]$ dans le contexte de w, où y commence à la position i de w et finit à la position j de w.

Un *alphabet* est un ensemble fini de symboles. On note Σ^* l'ensemble des mots définis sur l'alphabet Σ. Un *langage* défini sur Σ est un ensemble de mots de Σ^*. On note $\overline{L}$ le complémentaire d'un langage L.

Soient L_1 et L_2 deux langages. La *concaténation* de L_1 et L_2, notée L_1L_2, est l'ensemble :

$$\{uv \mid u \in L_1, v \in L_2\}$$

On définit pour tout entier $n \geq 0$ et tout langage L, la $n^{ième}$ puissance de L, notée L^n, par :

(1) $L^0 = \{\varepsilon\}$,

(2) $L^n = L^{n-1}L$, pour $n > 0$.

L'*étoile* d'un langage L, notée L^*, est l'ensemble :

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

De façon similaire, on définit :

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

2.1.2 Grammaires (régulières)

Une grammaire est un moyen de décrire un langage sans donner une liste explicite des mots le composant. Une grammaire est un ensemble de règles définissant les structures des mots. Plus formellement, une *grammaire* est un quadruplet $G = (\Sigma, V, S, P)$ où :

(1) Σ est l'alphabet des *terminaux*,

(2) V est l'alphabet des *non terminaux*,

(3) Σ et V sont disjoints, c'est-à-dire : $\Sigma \cap V = \emptyset$

(4) $S \in V$ est le *symbole d'entrée* (ou *axiome*),

(5) P est un ensemble fini de *productions*.

Chaque règle de production de P est de la forme $\alpha \rightarrow \beta$ avec $\alpha \in \{\Sigma \cup V\}^+$ et $\beta \in \{\Sigma \cup V\}^*$.

Le *langage* $L(G)$ *engendré* par une grammaire $G = (\Sigma, V, S, P)$ est constitué de l'ensemble des mots construits sur Σ et qui peuvent être générés à partir du symbole d'entrée S par application successive des règles de production de P jusqu'à ce qu'aucun non terminal ne soit présent. On dira aussi que $L(G)$ est le *langage reconnu* par la grammaire G.

Exemple 2.1 Soit la grammaire $G = (\Sigma, V, S, P)$ avec $\Sigma = \{a, c\}$, $V = \{S, C\}$, et $P = \{S \rightarrow aC, C \rightarrow cC|caa\}$. Le langage $L(G)$ reconnu par la grammaire G est l'ensemble des mots commençant par un a, suivi de un ou plusieurs c puis de deux a. Par exemple, les mots *acaa*, *acccaa* et *accccaa* appartiennent au langage. □

Dans [Chomsky 1956] N. Chomsky à proposé une classification hiérarchique des grammaires en quatre catégories. Ces différents types se distinguent par le pouvoir d'expression des règles de production que leurs grammaires contiennent.

Dans cette thèse, seuls les langages réguliers nous intéressent. Cependant, les grammaires ne sont pas le seul moyen de représenter les langages réguliers. Nous présentons dans les sections suivantes deux autres représentations classiques équivalentes : les expressions régulières et les automates d'états finis. Dans le chapitre 3, nous présentons un nouveau modèle (appelé automates de transitions finies) très proche des automates d'états finis et permettant de mettre l'accent sur la notion d'acides aminés et de position.

2.1.3 Expressions régulières

Les langages réguliers peuvent être facilement représentés par des expressions très simples appelées expressions régulières, que nous utiliserons pour décrire les langages dans les exemples.

Soit Σ un alphabet. Les expressions régulières sur Σ peuvent être décrites récursivement comme suit :

(1) $\emptyset$ est une expression régulière représentant l'ensemble vide,

(2) ε est une expression régulière représentant l'ensemble $\{\varepsilon\}$,

(3) pour tout a de Σ, a est une expression régulière représentant l'ensemble $\{a\}$,

(4) Si e_1, e_2 et e sont des expressions régulières reconnaissant respectivement les langages E_1, E_2 et E, alors $e_1 + e_2$, $e_1 e_2$ et e^* sont des expressions régulières représentant respectivement les langages $E_1 \cup E_2$, $E_1 E_2$ et E^*.

Pour simplifier l'écriture, l'expression ee^* peut se noter e^+.

Exemple 2.2 L'expression régulière ac^+aa représente le langage des mots commençants par un a suivi de un ou plusieurs c suivi de deux a. □

2.1.4 Automates d'états finis

La représentation par automates est celle que nous allons utiliser dans notre processus d'inférence. Elle offre une excellente perspective globale sans négliger les caractères locaux des langages. Nous présentons ici les automates d'états finis et certaines de leurs propriétés. Cette section est divisée en deux sous-sections. Dans la première, nous rappelons la définition d'un automate d'états finis. Nous décrivons ensuite les formes déterministe, non déterministe, complète et réduite de ce type d'automates. Nous illustrons toutes ces définitions par des exemples concrets.

Dans la deuxième sous-section nous présentons les automates d'états finis minimaux complet et réduit qui sont utilisés en apprentissage automatique d'automates. Ils sont basés sur une relation d'équivalence particulière que nous introduisons. Nous présentons aussi deux théorèmes reliés aux automates minimaux.

Définitions autour des automates d'états finis

Définition 2.1 (FSA) Un *automate d'états finis* est un quintuplet $(Q, \Sigma, I, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet fini, δ est une fonction de transition de $Q \times \Sigma$ vers 2^Q (étendue à $Q \times \Sigma^* \mapsto 2^Q$), I est l'ensemble des états initiaux et $F \subseteq Q$ est l'ensemble des états d'acceptation ou finals. □

Les automates d'états finis sont représentés par un graphe dirigé appelé *diagramme de transitions*. Les sommets du graphe correspondent aux états de l'automate. S'il existe une transition d'un état q vers un état p par a, alors il existe un arc étiqueté a du sommet q vers le sommet p dans le diagramme de transitions. Un chemin dans le graphe est une séquence de transitions

$$(q_1, l_1, q_2)(q_2, l_2, q_3) \dots (q_n, l_n, q_{n+1})$$

telle que pour tout $i \in [1, n+1]$ q_i est un sommet du graphe et pour tout $i \in [1, n]$ $l_i \in \Sigma$. Un mot w est *reconnu* (ou *accepté*) par un automate $(Q, \Sigma, I, F, \delta)$ s'il existe un chemin correspondant au mot w allant d'un état initial vers un état final. Le *langage reconnu* (ou *accepté*) par un automate $\mathcal{A} = (Q, \Sigma, I, F, \delta)$, noté $L(\mathcal{A})$, est l'ensemble $\{w \in \Sigma^* \mid \exists q \in I \ [\delta(q, w) \cap F \neq \emptyset]\}$ des mots acceptés par l'automate $\mathcal{A}$.

Pour chaque état q de $\mathcal{A}$, on définit deux automates particuliers qui correspondent à l'automate $\mathcal{A}$ dans lequel on a modifié l'ensemble des états initiaux ou finals. Le premier est appelé *automate d'accessibilité de l'état* q et noté $\mathcal{A}^{\rightsquigarrow q} = (Q, \Sigma, I, \{q\}, \delta)$. Il accepte le langage régulier reconnu par l'état q de $\mathcal{A}$ (c'est-à-dire : $L(\mathcal{A}^{\rightsquigarrow q}) = \{w \in \Sigma^* \mid \exists q' \in I \ [q \in \delta(q', w)]\}$). Le deuxième est appelé *automate de coaccessibilité de l'état* q et noté $\mathcal{A}^{q\rightsquigarrow} = (Q, \Sigma, \{q\}, F, \delta)$. Il accepte le langage régulier reconnu à partir de l'état q de $\mathcal{A}$ (c'est-à-dire : $L(\mathcal{A}^{q\rightsquigarrow}) = \{w \in \Sigma^* \mid \delta(q, w) \cap F \neq \emptyset\}$).

Définition 2.2 (DFA) Un *automate d'états finis déterministe* (DFA) est un quintuplet $(Q, \Sigma, q_0, F, \delta)$ où Q est un ensemble fini d'états, Σ est un alphabet fini, δ est une fonction de transition de $Q \times \Sigma$ vers Q (étendue à $Q \times \Sigma^* \mapsto Q$), q_0 est l'état initial et $F \subseteq Q$ est l'ensemble des états d'acceptation ou finals. □

Un automate d'états finis qui n'est pas déterministe est appelé *automate d'états finis non déterministe* et noté NFA.

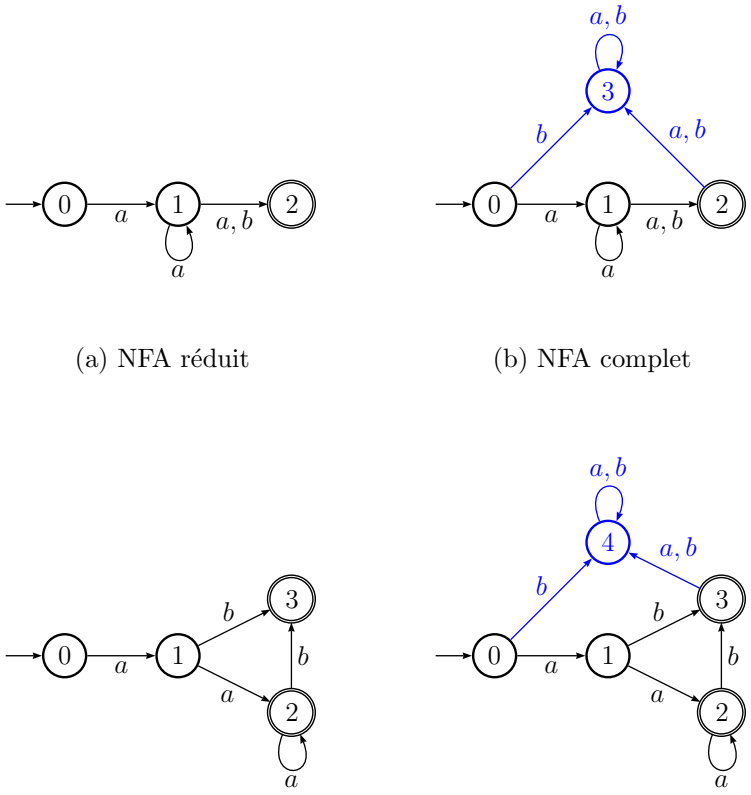
Définition 2.3 (DFA complet) Un DFA dont la fonction de transition est totale, c'est-à-dire définie pour chaque paire de $Q \times \Sigma$, est *complet*. □

Définition 2.4 (DFA réduit) Un DFA tel que pour chaque état q il existe un chemin de l'état initial vers q et un chemin de q vers un état d'acceptation est appelé DFA *réduit*. □

Exemple 2.3 La figure 2.1 page 22 montre quatre FSA sur l'alphabet $\Sigma = \{a, b\}$ représentant le langage L défini comme l'ensemble des mots commençant par un ou plusieurs a suivi d'un a ou d'un b ($a^+(a+b)$). Les états initiaux sont indiqués par une petite flèche entrante et les états finals sont indiqués par des doubles cercles.

(a) L'automate de la figure 2.1(a) est un NFA réduit reconnaissant les mots du langage L. Il est non déterministe parce que deux transitions étiquetées par le même symbole a sortent de l'état 1. Il est réduit parce que pour tous les états q il existe un chemin de l'état initial vers q et de q vers un état final.

(b) L'automate de la figure 2.1(b) est un NFA reconnaissant les mots du langage L. Il est non déterministe pour la même raison que l'automate précédent. Il est complet parce qu'à partir de tous les états on peut lire n'importe quelle lettre de Σ.



(a) NFA réduit

(b) NFA complet

(c) DFA réduit

(d) DFA complet

FIG. 2.1 – Exemples de FSA reconnaissant le langage $a^+(a+b)$.

(c) Il est facile de vérifier que l'automate de la figure 2.1(c) est un DFA réduit reconnaissant les mots du langage L. En effet, pour tous ses états q, il existe un chemin de l'état initial vers q et de q vers un état final.

(d) L'automate de la figure 2.1(d) est un DFA complet reconnaissant L (à partir de tous les états on peut lire n'importe quelle lettre de Σ).

□

Automate d'états finis minimal

Après un rappel de la notion de relation d'équivalence, nous définissons dans cette section les automates d'états finis minimaux complet et réduit, et nous présentons deux théorèmes importants associés à ces notions.

Définition 2.5 (quotient gauche, relation d'équivalence, index, classe d'équivalence, raffinement)

(1) Étant donné un langage L sur Σ, un ensemble $E \subseteq \Sigma^*$ et un mot $w \in \Sigma^*$, on appelle *quotient gauche* restreint à E du langage L par w (aussi appelé résiduel ou dérivé) tout langage $w_E^{-1}L = \{v \in E, wv \in L\}$.
Dans le cas où $E = \Sigma^*$, on parlera de quotient gauche du langage L par w.

(2) Nous définissons une relation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ par

$$u \equiv_L v \iff u_{\Sigma^*}^{-1}L = v_{\Sigma^*}^{-1}L$$

pour chaque paire $u, v \in \Sigma^*$. La relation $\equiv_L$ est une relation d'équivalence.

(3) On appelle *index* d'une relation d'équivalence son nombre de classes d'équivalence.

(4) On note $[w]_\equiv$ ou plus simplement $[w]$ la *classe d'équivalence* qui contient w, c'est-à-dire :

$$[w]_\equiv = \{v \in \Sigma^* \mid v \equiv w\}$$

Pour simplifier l'écriture, on notera $[w]_L$ (et non pas $[w]_{\equiv_L}$) la classe d'équivalence qui contient w pour la relation $\equiv_L$.

(5) Étant données deux relations R_1 et R_2 sur le langage L, on dit que R_1 *raffine* R_2 si et seulement si $xR_1y \implies xR_2y$ pour tous x et y de L.

□

La relation d´équivalence $\equiv_L$ sert à définir le FSA minimal. Le théorème suivant définit, pour tout langage régulier L, le nombre minimum d'états d'un DFA reconnaissant L.

Théorème 2.1 Soit L un langage régulier. Le nombre minimum d'états d'un DFA complet acceptant L est égal à l'index de $\equiv_L$. □

Il existe un lien entre les langages réguliers et la relation d'équivalence $\equiv_L$ introduite dans la Définition 2.5 (voir item (2)).

Théorème 2.2 **([Myhill 1957, Nerode 1958])** Un langage $L \subseteq \Sigma^*$ est régulier si et seulement si $\equiv_L$ a un index fini. □

Nous utilisons ce résultat pour formuler un théorème équivalent pour les automates de transitions finies que nous introduisons dans le chapitre 3.

Nous définissons maintenant le DFA minimal complet dont les états sont identifiés par le théorème précédent.

Définition 2.6 (DFA minimal complet en nombre d'états) Le *DFA minimal complet en nombre d'états* reconnaissant L est $(Q, \Sigma, q_0, F, \delta)$, où :

(1) Q est l'ensemble des classes d'équivalence de $\equiv_L$,

(2) δ est défini par $\delta([w], l) = [wl]$, pour tout $[w] \in Q$ et $l \in \Sigma$,

(3) $q_0 = [\varepsilon]$ et

(4) $F = \{[w] \mid w \in L\}$.

□

L'*automate d'états finis déterministe minimal réduit* est l'automate d'états finis déterministe minimal complet dans lequel on a supprimé les états correspondant à une classe d'équivalence avec un quotient gauche vide. Plus formellement :

Définition 2.7 (DFA minimal réduit en nombre d'états) Le *DFA minimal réduit en nombre d'états* reconnaissant L est $(Q, \Sigma, q_0, F, \delta)$, où :

(1) Q est l'ensemble des classes d'équivalence de $\equiv_L$ avec un quotient gauche non vide, c'est-à-dire : $Q = \left\{[w] \mid w \in \Sigma^*, w_{\Sigma^*}^{-1} L \neq \emptyset\right\}$,

(2) δ est défini par $\delta([w], l) = [wl]$, pour tout $[w] \in Q$ et $l \in \Sigma$,

(3) $q_0 = [\varepsilon]$ et

(4) $F = \{[w] \mid w \in L\}$.

□

2.2 Inférence grammaticale

Le concept d'inférence grammaticale est apparu dans les années $50-60$, c'est-à-dire à l'époque où N. Chomsky a formalisé la notion de langage [Chomsky 1956]. L'inférence grammaticale consiste à apprendre, à partir de données séquentielles ou structurées (par exemple des chaînes ou des arbres), une grammaire inconnue qui explique ces données [Higuera (de la) 2002]. Ce domaine de recherche est transversal à de nombreux autres domaines comme par exemple l'apprentissage automatique, la théorie des langages formels ou la bioinformatique. Malgré la richesse des recherches autant théoriques que pratiques, l'inférence grammaticale n'a pas eu beaucoup de succès dans les applications réelles [Higuera (de la) 2002, Miclet 1986]. Cependant, cette situation pourrait s'améliorer rapidement car on note des applications récentes pour lesquelles l'inférence grammaticale est utilisée avec succès. De plus, actuellement, les meilleures méthodes d'apprentissages sur les séquences biologiques sont celles du type apprentissage de Modèles cachés de Markov (HMM). Ces méthodes permettent d'apprendre les probabilités

d'un modèle dont la structure est connue *a priori*. Malheureusement, en général, la structure du modèle est inconnue. L'inférence grammaticale peut aider à la découverte de cette structure et intervenir en amont de méthodes de type HMM.

Dans la thèse, nous proposons une méthode d'inférence grammaticale *régulière* (c'est-à-dire qu'on s'intéresse au cas où la grammaire recherchée appartient à la classe des grammaires régulières) spécialisée dans le traitement des protéines. L'inférence régulière fera l'objet de la section 2.3.

Dans cette section, nous introduisons le modèle d'*identification à la limite* proposé par [Gold 1967] et nous en présentons deux variations. La première correspond au cas où les *données* sont *à la demande*. Le processus d'apprentissage est alors infini, sans contrainte de complexité en temps ou en espace. Pour la deuxième, les *données* sont *fixées* et l'apprentissage est un processus fini comportant certaines contraintes d'efficacité. Il existe bien sûr d'autres paradigmes (par exemple, les modèles MAT [Angluin 1988] ou PAC [Valiant 1984] qui sont parmi les plus connus) que nous ne présentons pas ici. Le lecteur pourra se référer aux articles [Sakakibara 1997, Higuera (de la) 1997] pour une vue globale du domaine de l'inférence grammaticale.

2.2.1 Identification à la limite à partir de données à la demande

Le modèle d'apprentissage le plus simple est développé dans [Gold 1967]. Dans sa version initiale, le processus d'identification à la limite est infini. Nous commençons par donner quelques définitions. Notons $\mathbb{B} = \{\mathsf{vrai}, \mathsf{faux}\}$ l'ensemble des booléens. Un *exemple étiqueté* d'un langage L est un couple (w, b) de $\Sigma^* \times \mathbb{B}$, tel que b vaut vrai si w est un mot de L, et faux sinon. Une *présentation* d'un langage L est une suite infinie d'exemples étiquetés. Une *présentation* est *positive* si toutes ses étiquettes sont à vrai et ses éléments sont appelés *exemples* ou *exemples positifs*. Une *présentation* est *négative* si toutes ses étiquettes sont à faux et ses éléments sont appelés *contre-exemples* ou *exemples négatifs*. Une *présentation* sur Σ est *complète* si elle contient tous les mots de Σ^*. Une *présentation positive* d'un langage L sur Σ est *complète* si elle contient tous les mots de L.

Dans le modèle d'*identification à la limite à partir de données à la demande*, un algorithme d'apprentissage dispose d'une présentation d'un langage L et produit la représentation de ce langage. À chaque pas, l'algorithme traite un exemple de la présentation et propose une représentation pour L. On dira qu'un algorithme d'apprentissage *identifie à la limite à partir de données à la demande* une classe de langages si, pour tout langage L de cette classe et pour toute présentation complète de L, à partir d'un certain pas toutes les représentations retournées par l'algorithme sont des représentations de L. Une classe de langages est *identifiable à la limite* s'il existe un algorithme d'apprentissage qui l'identifie à la limite.

E. M. Gold a montré deux résultats importants [Gold 1967] :

(1) toute classe de langages récursifs est identifiable à la limite à partir de présentations complètes,

(2) aucune *classe superfinie* de langages (c'est-à-dire , une classe qui contient tous les langages de cardinalité finie et au moins un langage de cardinalité infinie) n'est identifiable à la limite à partir de présentations positives.

Ces deux résultats montrent l'importance des exemples négatifs dans ce cadre théorique. Néanmoins, En pratique, ce changement du type de présentation pour le modèle de l'identification à la limite n'est pas exploitable telle quelle. En effet, cette version du paradigme a pour inconvénient de ne poser aucune contrainte sur :

(1) l'espace mémoire nécessaire à l'inférence,

(2) le nombre d'exemples nécessaires au succès du processus d'inférence et

(3) la manière dont doit être constituée la présentation.

De plus, on ne sait jamais, à un instant donné, si la représentation a été identifiée. De ce fait, l'identification à la limite à partir de données à la demande est insuffisante d'un point de vue pratique.

2.2.2 Identification à la limite à partir de données fixées

En 1978 E. M. Gold propose la deuxième version de son modèle incluant une contrainte d'efficacité pour le rendre plus facilement exploitable et propose l'*identification à la limite à partir de données fixées* [Gold 1978]. Contrairement au cas où les données sont à la demande, le nombre d'exemples fourni à l'algorithme est fini. Un *échantillon* d'un langage L, noté $\mathcal{S}$, est un ensemble fini d'exemples étiquetés. Un *échantillon positif* d'un langage L, noté $\mathcal{S}_+$, est un ensemble fini de mots appartenant au langage L. Un *échantillon négatif* d'un langage L, noté $\mathcal{S}_-$, est un ensemble fini de mots n'appartenant pas au langage L.

L'identification à la limite à partir de données fixées ne tient pas seulement compte de la classe de langages, mais également de la classe de représentation. Ainsi, un type de représentation d'une classe de langages (par exemple les automates finis déterministes) est *identifiable à la limite à partir de données fixées* si l'on peut associer à tout langage L de la classe un échantillon particulier, appelé *échantillon caractéristique* $\mathcal{CS}$, tel qu'à partir de tout échantillon contenant $\mathcal{CS}$, la méthode retourne une représentation de L du type choisi. De plus, un type de représentation d'une classe de langages est *polynomialement identifiable à la limite à partir de données fixées* si :

(1) il est identifiable à la limite à partir de données à la demande,

(2) l'algorithme propose, à chaque fois qu'un exemple lui est fourni, une représentation en un temps polynomial en fonction de la taille de l'échantillon déjà traité et

(3) la taille de l'échantillon caractéristique est polynomiale en la taille de la plus petite représentation du langage dans le type choisi.

E. M. Gold montre [Gold 1978] que les automates déterministes sont polynomialement identifiables à la limite à partir de données fixées.

En 1997, [Higuera (de la) 1997] propose une autre formulation du modèle d'*identification à la limite polynomiale à partir de données fixées* de [Gold 1978]. Une classe de représentation $\mathcal{R}$ est *polynomialement identifiable à la limite à partir de données fixées* selon C. De La Higuera si et seulement s'il existe deux polynômes p et q et un algorithme A tels que :

(1) étant donné un échantillon $(\mathcal{S}_+, \mathcal{S}_-)$, de taille m, A renvoie une représentation R dans $\mathcal{R}$ compatible avec $(\mathcal{S}_+, \mathcal{S}_-)$ en un temps $p(m)$.

(2) pour toute représentation R de taille n, il existe un échantillon caractéristique $(\mathcal{CS}_+, \mathcal{CS}_-)$ de taille au plus $q(n)$ pour lequel, sur les données $(\mathcal{S}_+, \mathcal{S}_-)$, avec $\mathcal{S}_+ \supseteq \mathcal{CS}_+$ et $\mathcal{S}_- \supseteq \mathcal{CS}_-$, A retourne une représentation R' équivalente à R.

Dans le même article, C. De La Higuera reformule le résultat de E. M. Gold [Gold 1978] stipulant que la classe des automates finis déterministes est polynomialement identifiable à la limite à partir de données fixées. Il montre également que la classe des automates non déterministes (et plusieurs autres classes importantes comme les grammaires algébriques) ne sont pas polynomialement identifiables à la limite à partir de données fixées.

2.3 Inférence régulière

L'inférence régulière est un cas particulier de l'inférence grammaticale dans lequel le langage recherché est régulier. Nous présentons ici l'algorithme d'inférence régulière dit par fusion d'états et nous discutons de son utilisation dans le cas où le langage que nous recherchons décrit un ensemble de protéines.

2.3.1 Un algorithme d'identification à la limite à partir de données fixées

Dans cette section nous présentons l'algorithme d'apprentissage à partir de données fixées par fusion d'états. Dans un premier temps, nous donnons quelques définitions intermédiaires puis nous décrivons l'algorithme et ses heuristiques.

Algorithme parcourant un espace de recherche d'automates à l'aide d'un opérateur de fusion d'états

Nous introduisons quelques notations sur les partitions et nous présentons deux notions fondamentales pour l'algorithme par fusion d'états qui sont les opérations de fusion et les automates sur lesquels les fusions sont effectuées.

Une partition π sur un ensemble de E est un ensemble de sous-ensembles non vides de E, appelés *blocs*, deux à deux disjoints et dont l'union est égale à E. Le bloc B d'une partition π sur E contenant l'élément $e \in E$ est noté $\mathsf{B}_\pi(e)$.

Définition 2.8 (FSA dérivé) Soient $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ un FSA et π une partition sur Q. Alors, le *FSA* $\mathcal{A}/\pi = (Q', \Sigma, I', F', \delta')$, *dérivé* de $\mathcal{A}$ par la partition π, est défini par :

(1) $Q' = \{\mathsf{B}_\pi(q) \mid q \in Q\} = \pi$,

(2) $I' = \{\mathsf{B}_\pi(q) \mid q \in I\}$,

(3) $F' = \{\mathsf{B}_\pi(q) \mid q \in F\}$,

(4) $\delta' : Q' \times \Sigma \mapsto 2^{Q'} : \forall B \in Q' \; \forall l \in \Sigma \; \left[\delta'(B, l) = \bigcup_{q \in B, q' \in \delta(q,l)} \{\mathsf{B}_\pi(q')\}\right]$.

□

Définition 2.9 (opération de fusion d'états) Soient q_1 et q_2 deux états d'un FSA $\mathcal{A}$ donné. Alors, le résultat de l'application de l'*opération de fusion des états* q_1 et q_2 sur $\mathcal{A}$ est le FSA dérivé $\mathcal{A}/\pi$ où $\pi = \{\{q\} \mid q \in (Q \setminus \{q_1, q_2\})\} \cup \{\{q_1, q_2\}\}$. □

Dans le chapitre 5, nous introduirons l'opération de fusion de transitions basée sur la fusion d'états.

Nous nous intéressons dans cette section à l'inférence d'automates déterministes. Or, l'opération précédente ne garantit pas de conserver le déterminisme. Un opérateur de *fusion pour déterminisation* a donc été introduit. Cette opération fusionne récursivement tous les états q_1 et q_2 dont l'intersection des langages de leur automate d'accessibilité est non vide (c'est-à-dire $L(\mathcal{A}^{\rightsquigarrow q_1}) \cap L(\mathcal{A}^{\rightsquigarrow q_2}) \neq \emptyset$). Chaque fusion peut modifier l'automate d'accessibilité de certains états. Ceci oblige à calculer dynamiquement les paires d'états à fusionner au cours de la récursion. L'*opération de fusion déterministe* correspond alors à l'enchaînement de l'opération de fusion d'états et de la procédure de fusion pour déterminisation.

La deuxième notion importante est celle de l'automate initial sur lequel les fusions sont réalisées. Une méthode d'apprentissage dispose en entrée d'un échantillon. Les algorithmes par fusion d'états commencent par transformer l'échantillon en un automate qui reconnaît comme langage l'ensemble des mots de

l'échantillon positif. Si l'on infère des automates non déterministes, l'automate sur lequel on effectue les fusions est l'automate d'états finis maximal canonique (MCA [1]). Intuitivement, le MCA peut être vu comme un ensemble d'arbres ayant tous exactement une feuille et correspondant chacun à un exemple (ou mot) de l'échantillon. Formellement :

Définition 2.10 (automate d'états finis maximal canonique [Dupont *et al.* 1994, Fredouille 2003]) Soit $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ l'échantillon d'apprentissage positif composé des exemples positifs $s_1, s_2, \ldots, s_n$ où pour tout $i \in [1, n]$, $s_i = l_{i1} l_{i2} \ldots l_{in}$. L'*automate d'états finis maximal canonique* relatif à $\mathcal{S}$, noté MCA, est l'automate fini $(Q, \Sigma, I, F, \delta)$ tel que :

(1) Σ est l'alphabet sur lequel $\mathcal{S}$ est défini

(2) $Q = \{q_{ij} \mid i \in [1, |\mathcal{S}|], j \in [0, |s_i|]\}$

(3) $I = \{q_{i0} \mid i \in [1, |\mathcal{S}|]\}$

(4) $F = \left\{q_{i|s_i|} \mid i \in [1, |\mathcal{S}|]\right\}$

(5) $\forall i \in [1, |\mathcal{S}|], \forall j \in [1, |s_i| - 1]. \left[\delta\left(q_{ij}, l_{ij}\right) = \{q_{ij+1}\}\right]$

□

Dans le cas de l'inférence d'automates déterministes, le MCA est remplacé par sa version déterministe : l'*arbre accepteur de préfixes*. L'*arbre accepteur de préfixes* (PTA[2]) sur un échantillon $\mathcal{S}$ est l'automate obtenu en fusionnant tous les états du MCA ayant un langage préfixe égal.

L'exemple suivant illustre les définitions précédentes.

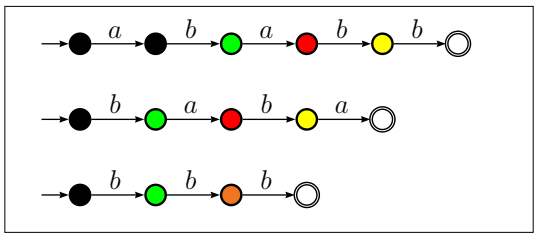
Exemple 2.4 La figure 2.2 de la page 30 illustre les différents types de fusion d'états. L'échantillon d'apprentissage considéré est $\mathcal{S}_+ = \{ababb, baba, bbb\}$. Les deux premières sous-figures représentent le MCA et le PTA sur l'échantillon $\mathcal{S}_+$. La figure 2.2(f) correspond à l'automate obtenu après la fusion des deux états noirs du PTA. La figure 2.2(g) correspond à l'automate obtenu à partir du PTA par l'application de l'opération de fusion déterministe sur les deux états noirs de la figure 2.2(c). □

L'algorithme par fusion d'états et ses heuristiques

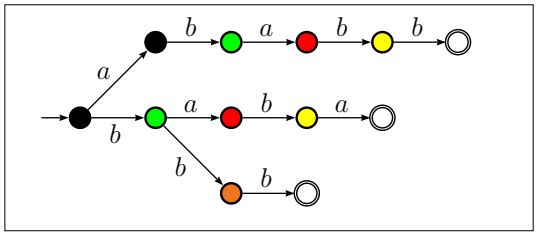
Dans cette sous-section, nous présentons l'algorithme d'apprentissage par fusion d'états et deux heuristiques associées. Cet algorithme a été décrit pour la

[1] Provient de la terminologie anglaise "Maximal Canonical Automaton

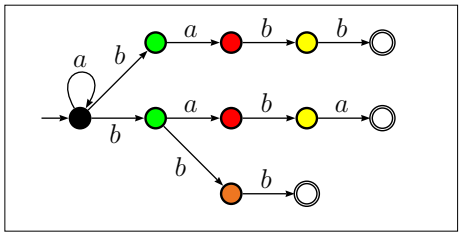
[2] Provient de la terminologie anglaise "Prefix Tree Acceptor"



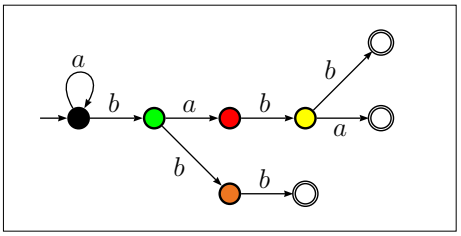
(a) MCA



(c) PTA



(f) Automate après une fusion d'états.



(g) Automate après une fusion déterministe.

FIG. 2.2 – Fusions d'états.

première fois dans [Trakhenbrot et Barzdin 1973]. La fusion d'états est une opération de généralisation de l'automate. Si l'on fusionne les états sans aucune contrainte, on arrive dans tous les cas à l'*automate universel*, le plus petit automate dont le langage accepté est Σ^*. Pour stopper la généralisation, l'algorithme et les heuristiques présentées dans cette section utilisent un échantillon contenant des exemples positifs et des exemples négatifs. Ce sont ces derniers qui limitent la généralisation en interdisant certaines fusions. La méthode d'apprentissage proposée dans [Trakhenbrot et Barzdin 1973] utilise l'opération de fusion déterministe.

L'algorithme de B. Trakhenbrot and Y. Barzdin nécessite un échantillon d'apprentissage complet. Cette hypothèse est trop forte en pratique. Ainsi, [Oncina et García 1992, Lang 1992] ont proposé simultanément l'heuristique RPNI. Dans cette méthode, à chaque état on associe le mot le plus court permettant d'atteindre cet état à partir de l'état initial. On ordonne statiquement les états suivant l'ordre standard sur ces mots. Par *ordre standard*, on désigne l'ordre qui prend d'abord en compte la longueur des mots puis, en cas d'égalité, l'ordre lexicographique. La fusion des états s'effectue dans l'ordre ainsi défini.

La seconde heuristique (EDSM) a été proposée par [Lang 1998]. Elle est plus complexe que la précédente mais a gagné la compétition d'inférence grammaticale abbadingo [Lang *et al.* 1998]. La paire d'états à fusionner à chaque étape est déterminée dynamiquement. La paire d'états que l'on va fusionner est celle dont la fusion déterministe entraîne la fusion du plus grand nombre de paires d'états tous deux finals (positifs ou négatifs). Cela revient à maximiser l'évidence d'une fusion par maximisation de la différence entre le nombre d'états accepteurs de l'automate avant et après fusion.

2.3.2 Discussion sur l'utilisation de l'inférence grammaticale par fusion d'états sur des protéines

L'algorithme le plus utilisé en inférence grammaticale d'automates est celui par fusion d'états. On peut s'étonner que cet algorithme n'ait pas été largement appliqué au domaine de la bioinformatique. Plusieurs raisons concourent à ce fait.

Régions conservées des protéines. La première raison est liée à la nature biologique des séquences. Une protéine est une séquence orientée et finie dont les extrêmités sont appelées respectivement région N- et C-terminale. Certaines régions des protéines sont conservées à cause de leur rôle particulier de maintient de la structure 3D de la protéine ou de liaison avec d'autre molécules. De ce fait, ces régions conservées se trouvent rarement au niveau des régions N- ou C-terminales.

Les meilleures heuristiques actuelles pour l'algorithme de fusion d'états sont RPNI et EDSM. La première fusionne prioritairement les états qui sont proches de l'état initial. Si les mots de l'échantillon d'apprentissage sont des séquences protéiques, cela revient à fusionner en premier les parties N-terminales, c'est-à-dire des régions non conservées. La localisation des régions conservées dans les protéines enlève tout son sens à cette heuristique. En effet, si l'on commence par réaliser des fusions au niveau des régions non conservées du début des portéines, cela conduit à la formation d'une boucle qui risque de contenir aussi le début de la première région conservée des protéines étudiées. Le modèle obtenu contient donc au mieux uniquement une sous-partie des régions conservées des protéines.

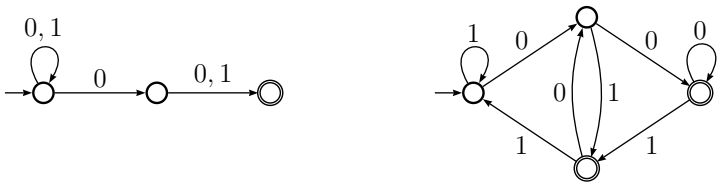
De la même façon, la deuxième heuristique (EDSM) tend à maximiser les fusions entre les états terminaux. Dans le cas où les données sont des séquences protéiques, l'heuristique maximise alors les fusions entre les acides aminés C-terminaux. La maximisation repose sur la comparaison des régions C-terminales des protéines qui n'ont aucune raison d'être similaires d'une séquence à l'autre. Ceci est dû à la localisation des régions conservées dans les protéines. L'utilisation de cette heuristique sur ce type de données n'a donc aucun sens.

Ces observations tendent à montrer qu'il serait plus judicieux de commencer par fusionner les régions de la protéine correspondant aux régions conservées. Actuellement, en dehors des travaux de l'équipe Symbiose [Idmont 2003, Fredouille 2003, Coste *et al.* 2004] de l'Inria, aucune heuristique ne permet de réaliser les fusions dans un tel ordre.

Forme des motifs dans les protéines. La deuxième raison est liée à la forme des motifs que l'on trouve dans les protéines. En effet, une famille de protéines peut souvent être décrite comme une union de suites de motifs (comme ceux décrits dans le chapitre 1) espacés d'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^*.

La grande majorité des heuristiques développées pour la fusion d'états est adaptées aux automates déterministes. La figure 2.3 montre les automates minimaux déterministes et non déterministes reconnaissant le langage très simple $(1+0)^*0(1+0)$. Ce langage est très facilement représentable par un automate non déterministe (figure 2.3(a)), mais beaucoup moins par un automate déterministe (figure 2.3(b)).

De ce fait, l'état actuel de la recherche sur la fusion d'états rend impossible le traitement des protéines car d'une part, il est très difficile de modéliser le langage décrivant une famille de protéines par un automate déterministe et, d'autre part, les heuristiques adaptées aux automates non déterministes sont presque inexistantes.



(a) NFA minimal (b) DFA minimal

FIG. 2.3 – NFA et DFA minimaux pour le langage $(0+1)^*0(0+1)$ sur l'alphabet $\Sigma = \{0, 1\}$.

Utilisation d'exemples négatifs. La troisième raison est la difficulté à obtenir des exemples négatifs intéressants. Effectivement, bien que la recherche sur les protéines soit extrêmement active, les connaissances dans ce domaine restent encore très insuffisantes. La notion de famille est complexe car certaines protéines peuvent se révéler multi-fonctionnelles. Il nous est donc difficile d'exhiber, pour une famille de protéines, des séquences très proches de celles des protéines de la famille et ne lui appartenant pas de façon certaine. La plupart des algorithmes d'inférence grammaticale utilisent à la fois des exemples positifs et des exemples négatifs pour réaliser l'apprentissage. En effet, il a souvent été montré que l'utilisation d'information additionnelle concernant le concept cible ou la classe auquel il appartient améliore grandement la qualité des méthodes d'inférences. Dans le cas où l'information additionnelle est sous forme d'exemples négatifs, l'existance de cette information nécessite la connaissance *a priori* de ce que le langage ne contient pas (dans le papier [Knuutila 1996], on peut trouver une discussion approfondie de ce point).

Ces remarques donnent toute leur importance aux travaux qui concernent le problème de l'inférence à partir d'exemples positifs seulement. Dans les papiers de T. Knuutila [Knuutila 1996], de Y. Sakakibara [Sakakibara 1997] et de C. de la Higuera [Higuera (de la) 2002], le lecteur peut trouver une étude bibliographique sur l'apprentissage à partir de données positives. Néanmoins, il est important de rappeler que E. M. Gold [Gold 1967] a montré qu'aucune *classe superfinie* de langages n'est identifiable à la limite à partir de présentations positives et que de ce fait la classe des langages réguliers ne l'est pas non plus. Ceci a mené les chercheurs à s'intéresser à des sous-classes particulières des langages réguliers apprenables de cette façon et à proposer les algorithmes correspondants. Par exemple, D. Angluin [Angluin 1982] a défini la classe des langages réversibles et a proposé un algorithme pour cette classe de langages. P. García et E. Vidal [García et Vidal 1990] ont donné un algorithme pour la classe des langages

k-testables. T. Koshiba *et al.* [Koshiba *et al.* 1997] ont proposé un algorithme pour une sous-classe des langages linéaires équilibrés. F. Denis *et al.* [Denis *et al.* 2002] ont défini des sous-classes des langages réguliers et ont proposé des algorithmes produisant un automate non déterministe et permettant d'identifier ces classes. H. Fernau [Fernau 2000] a généralisé ces résultats. H. Rulot [Rulot et Vidal 1988, Rulot *et al.* 1989] ont développé un algorithme heuristique inférant des automates acycliques.

Dans la thèse nous proposons une méthode fortement inspirée de l'algorithme de fusion d'états et prenant en entrée un échantillon composé uniquement d'exemples positifs. Nous utilisons des machines proches des automates d'états finis (que nous appelons automates de transitions finies) dans leur forme non déterministe pour représenter les langages réguliers. La généralisation est maîtrisée par l'utilisation de méthodes statistiques.

Chapitre 3

Le modèle FTA : une représentation adaptée de langages réguliers

Le but de ce travail est d'étendre et d'adapter à l'étude des régularités observées dans les protéines l'algorithme de fusion d'états et les heuristiques associées. Nous avons été conduit pour cela à retravailler la représentation utilisée pour décrire les langages réguliers. Dans ce chapitre, nous introduisons un nouveau type d'automate dont le pouvoir d'expression est identique à celui des automates d'états finis. Nous l'appelons automate de transitions finies (FTA). Dans la première section, nous donnons les motivations qui nous ont amenés à proposer un tel modèle. Puis, nous définissons les automates de transitions. Dans la troisième section, nous montrons que les automates de transitions et les automates d'états représentent la même classe de langages et nous établissons les méthodes de transformation entre les deux modèles. Nous approfondissons ensuite les liens entre les automates de transitions et les langages réguliers. La compacité de la représentation est établie de façon précise en comparant la taille des automates de transitions et des automates d'états. Enfin, dans la dernière section, nous définissons la notion d'automate de transitions finies minimal.

3.1 Motivations

Les familles de protéines peuvent être vues comme des ensembles de mots sur l'alphabet à vingt lettres des acides aminés. Il est donc possible de les représenter sous forme d'automates d'états finis. Dans ce modèle, les étiquettes des transitions correspondent aux acides aminés et les états initiaux et finaux indiquent

(a) NFA minimal (b) DFA minimal

FIG. 3.1 – NFA et DFA minimaux pour le langage $(A + G)^*AG(A + G)^*$ sur l'alphabet $\Sigma = \{A, G\}$.

respectivement le début et la fin de la reconnaissance des protéines. Les états définissent des paires de langages (suffixes et préfixes) et peuvent être associés aux liaisons entre les acides aminés. Certaines méthodes d'inférence grammaticale partent d'un automate spécifique dont le langage reconnu correspond à l'échantillon donné. Elles utilisent l'opération de fusion d'états (voir la Définition 2.9 page 28) pour généraliser successivement l'automate initial. Cette opération, qui fusionne des paires de langages préfixes et suffixes, est difficile à interpréter du point de vue des ensembles de protéines correspondants. Notre proposition est plutôt de comparer et de fusionner les éléments constitutifs des protéines : les acides aminés. Pour cela, nous introduisons l'opération de fusion de transitions, qui sera développée dans le chapitre 5. Cette opération, appliquée au modèle d'automate d'états finis, nous permet effectivement de fusionner les acides aminés entre eux. Par contre, on perd alors l'information de position initiale, finale ou intermédiaire lors des fusions. Pour régler ce problème, nous proposons un nouveau modèle d'automate dans lequel les transitions portent toute l'information. Ce modèle, appelé automate de transitions finies, est parfaitement adapté à la fusion de transitions. De plus, il possède l'avantage d'être légèrement plus compact que celui des automates d'états finis tout en gardant le même pouvoir d'expression.

Nous utilisons les automates de transitions finies non déterministes dans le processus d'inférence pour construire itérativement le modèle décrivant la famille de protéines étudiée. Nous avons choisi les automates non déterministes parce que, dans le cas de l'inférence, ils sont mieux adaptés à la modélisation des séquences biologiques que les automates déterministes. Donnons un exemple typique pour les séquences biologiques qui illustre l'intérêt de l'utilisation de ces automates non déterministes. La figure 3.1 montre les automates d'états finis minimaux déterministes et non déterminites représentant l'ensemble des séquences sur l'alphabet $\Sigma = \{A, G\}$ contenant le motif AG. L'automate non déterministe est à la fois

plus expressif et plus intuitif que l'automate déterministe. D'une part, ces deux qualités sont particulièrement importantes pour choisir les transitions à fusionner à chaque étape du processus d'inférence. D'autre part, le non déterminisme permet d'éviter de confondre le début d'un motif avec la boucle sur l'alphabet complet qui le précède.

À la fin du processus d'inférence, nous obtenons un automate de transitions finies non déterministe. Cet automate peut être transformé ensuite dans une autre forme, par exemple déterministe pour une analyse plus efficace de nouvelles séquences par le modèle.

Le nouveau modèle des automates de transitions finies que nous proposons dans ce chapitre n'est pas meilleur que les automates d'états finis mais il est plus direct pour réaliser des fusions de transitions. De plus, les automates non déterministes ne sont pas supérieurs en puissance d'expression aux automates déterministes, mais nous pensons qu'ils permettent un meilleur apprentissage du modèle pour les séquences biologiques.

3.2 Définitions autour des automates de transitions finies

Dans cette section, nous définissons les automates de transitions finies (FTA), très proches des FSA. Ils présentent malgré tout des différences originales. En effet, dans les FSA, l'information du début et de la fin des mots se trouve sur les états. Dans le cas des FTA, aucune information n'est portée par les états. Dans ce modèle, elles se trouvent sur les transitions. Un FTA est ainsi un ensemble de transitions dont certaines sont initiales ou finales. L'exemple 3.1 page 40 présente plusieurs d'automates de transitions finies qui illustrent chacune des définitions que nous présentons dans la section. Nous donnons maintenant la définition formelle d'un FTA.

Définition 3.1 (FTA) Un *automate de transitions finies* (FTA) est un triplet (Q, Σ, T), où Q est un ensemble fini d'états, Σ est un alphabet fini et T, défini sur $Q \times \Sigma \times 2^{\{i,f\}} \times Q$, est un ensemble fini de transitions. Chaque *transition* du FTA est un quadruplet $t = (q, l, m, q')$, où $q \in Q$ (*resp.* $q' \in Q$) est l'origine (*resp.* la destination) de t, $l \in \Sigma$ est le label de la transition et $m \subseteq \{i, f\}$ est la marque de la transition indiquant si t est initiale et/ou finale. □

Remarquons que, le modèle des FTA pour les machines de Mealy est l'analogue (si l'on oublie la différence aux niveau de la marque initiale entre les machines de Mealy et les FTA) du modèle des automates d'états finis (FSA) pour les machines de Moore [Moore 1956] qui a été présenté dans le chapitre 2.

Nous introduisons maintenant les notions de chemin, de mot accepté, d'ensemble de transitions étendu et de langage accepté. Elles permettent de faire le lien entre les mots, les langages et les automates de transitions finies.

(1) Un *chemin* dans un FTA est une séquence de transitions

$$(q_1, l_1, m_1, q_2)\,(q_2, l_2, m_2, q_3)\ldots(q_n, l_n, m_n, q_{n+1})$$

(2) Un mot w est *reconnu* (ou *accepté*) par un automate de transitions finies (Q, Σ, T) s'il existe un chemin correspondant au mot w commençant par une transition initiale et terminant par une transition finale.

(3) L'ensemble T^* des transitions étendues est l'ensemble des transitions de la forme

$$(q_1, w, m, q_{n+1}) \in Q \times \Sigma^+ \times \{i, f\} \times Q$$

telles qu'il existe un chemin

$$(q_1, l_1, m_1, q_2)\,(q_2, l_2, m_2, q_3)\ldots(q_n, l_n, m_n, q_{n+1})$$

où $w = l_1 l_2 \ldots l_n$, $i \in m$ si et seulement si $i \in m_1$, et $f \in m$ si et seulement si $f \in m_n$.
Intuitivement, chaque transition étendue correspond à un mot représenté par une suite de transitions consécutives du FTA. L'état d'origine de la transition étendue est celui de la première transition et son état destination celui de la dernière. La marque de la transition étendue contient i si la première transition est initiale et f si la dernière transition est finale.

(4) Le *langage reconnu* (ou *accepté*) par un automate de transitions finies $\mathcal{A} = (Q, \Sigma, T)$, noté $L(\mathcal{A})$, est l'ensemble

$$\{w \in \Sigma^* \mid \exists q, q' \in Q\ [(q, w, \{i, f\}, q') \in T^*]\}$$

des mots acceptés par l'automate.

Nous définissons à présent deux automates particuliers qui nous servent à isoler une partie d'un automate relativement à un état q et donc une partie du langage reconnu par cet automate. Nous les utilisons dans la section qui définit le DTA minimal d'un langage.

Pour chaque état q d'un FTA $\mathcal{A} = (Q, \Sigma, T)$, on définit deux automates particuliers qui correspondent à l'automate $\mathcal{A}$ dans lequel on a modifié les marques des transitions. Le premier est appelé *automate d'accessibilité de l'état* q et noté $\mathcal{A}^{q\rightsquigarrow} = (Q, \Sigma, T^{q\rightsquigarrow})$ avec :

$$T^{q\rightsquigarrow} = \left\{(q_1, l, m', q_2) \mid (q_1, l, m, q_2) \in T, m' = \left[\begin{cases} m \setminus \{i\} & \text{si } q_1 \neq q \\ m \cup \{i\} & \text{sinon} \end{cases}\right]\right\}$$

Il accepte le langage régulier reconnu à partir des transitions d'origine q de $\mathcal{A}$ (c'est-à-dire : $L\left(\mathcal{A}^{q\rightsquigarrow}\right) = \{w \in \Sigma^* \mid \exists q' \in Q, \exists m \subseteq \{i, f\} \ [(q, w, m, q') \in T^*, f \in m]\}$). Le deuxième est appelé *automate de coaccessibilité de l'état* q et noté $\mathcal{A}^{\rightsquigarrow q} = (Q, \Sigma, T^{\rightsquigarrow q})$ avec :

$$T^{\rightsquigarrow q} = \left\{(q_1, l, m', q_2) \mid (q_1, l, m, q_2) \in T, m' = \left[\begin{cases} m \setminus \{f\} & \text{si } q_2 \neq q \\ m \cup \{f\} & \text{sinon} \end{cases}\right]\right\}$$

Il accepte le langage régulier reconnu par les transitions de destination q de $\mathcal{A}$ (c'est-à-dire : $L\left(\mathcal{A}^{\rightsquigarrow q}\right) = \{w \in \Sigma^* \mid \exists q' \in Q, \exists m \subseteq \{i, f\} \ [(q', w, m, q) \in T^*, i \in m]\}$).

Dans la suite du chapitre, nous supposons qu'aucun langage régulier ne contient le mot vide. Ceci est nécessaire car un FTA ne peut pas reconnaître le mot vide. En effet, un mot est reconnu par l'automate $\mathcal{A}$ s'il existe un chemin correspondant dans $\mathcal{A}$. Par définition, un mot reconnu contient au moins une lettre. Le mot vide ne peut donc pas être reconnu. Cette restriction n'est pas forte. Ainsi, un langage régulier L reconnaissant le mot vide n'est rien d'autre que l'union de deux langages réguliers : $L \setminus \{\varepsilon\}$[1] et $\{\varepsilon\}$.

Nous définissons l'automate de transitions finies déterministe. Le déterminisme impose que, pour tout mot w de Σ^*, il existe au plus une acceptation de celui-ci. Ainsi, de la même façon que pour les automates d'états finis déterministes, il n'existe pas deux transitions de même état origine et portant la même étiquette. Il faut de plus ajouter la condition que toutes les transitions initiales d'un automate de transitions finies déterministe portent des étiquettes différentes. Formellement :

Définition 3.2 (DTA) Un *automate de transitions finies déterministe* (DTA) est un FTA dont l'ensemble des transitions T est tel que, pour chaque paire de transitions distinctes $t_1 = (q_1, l, m_1, {q_1}')$ et $t_2 = (q_2, l, m_2, {q_2}')$ de T, $i \notin (m_1 \cap m_2)$ et $q_1 \neq q_2$. □

Un automate de transitions finies qui n'est pas déterministe est appelé *automate de transitions finies non déterministe* et noté NTA.

Nous définissons maintenant un FTA complet. De façon informelle, un automate complet est tel que :

(1) pour tout mot de Σ^+, il existe un chemin correspondant à ce mot dans l'automate et

(2) toute transition du FTA a au moins un successeur pour chaque lettre de l'alphabet.

[1] Notons que $L \setminus \{\varepsilon\}$ est effectivement un langage régulier car $\{\varepsilon\}$ est régulier, $L \setminus \{\varepsilon\} = \overline{\overline{L} \cup \{\varepsilon\}}$ et les langages réguliers sont clos par union et par complémentation.

Formellement :

Définition 3.3 (FTA complet) Un FTA *complet* est un FTA dont l'ensemble T des transitions est tel que :

(1) pour chaque paire $\{q, l\} \in Q \times \Sigma$, il existe $q' \in Q$ et $m \subseteq \{i, f\}$ tels que (q, l, m, q') appartient à T et

(2) pour chaque lettre l de l'alphabet Σ, il existe $q, q' \in Q$ et $m \in \{\{i, f\}, \{i\}\}$ tels que (q, l, m, q') appartient à T.

□

La prochaine définition que nous proposons est celle du DTA réduit. Dans un DTA réduit, toutes les transitions et tous les états peuvent être utilisés pour reconnaître un mot du langage accepté par cet automate. La définition formelle est la suivante :

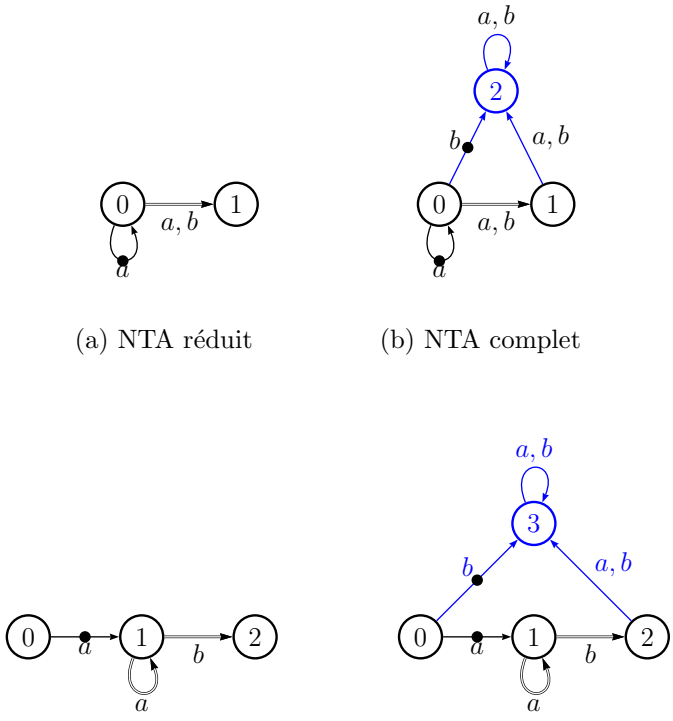
Définition 3.4 (DTA réduit) Un DTA *réduit* est un DTA tel que pour chaque transition t il existe un chemin passant par t, commençant par une transition initiale et finissant par une transition finale. □

Pour illustrer l'ensemble des définitions précédentes, l'exemple suivant présente quatre FTA (un NTA réduit, un NTA complet, un DTA réduit et un DTA complet) acceptant le même langage.

Exemple 3.1 Comme les automates d'états finis, les automates de transitions finies sont représentés graphiquement par un graphe de transitions. Les transitions initiales sont indiquées par un petit rond central noir et les transitions finales par des doubles flèches. Cet exemple illustre l'ensemble des définitions précédentes. Les quatre FTA (voir la figure 3.2 page 41) sur l'alphabet $\Sigma = \{a, b\}$ représentent le langage L reconnu par les automates proposés dans l'exemple 2.3 page 21, c'est-à-dire le langage $(a^+(a + b))$.

(a) L'automate représenté figure 3.2(a) est un NTA réduit. Il est non déterministe parce que deux transitions étiquetées par le même symbole a sortent de l'état 0. Il est réduit parce que pour toutes ses transitions t, il existe un chemin passant par t, commençant par une transition initiale et finissant par une transition finale. Par exemple, les transitions $(0, a, \{i\}, 0)$, $(0, a, \{f\}, 1)$ et $(0, b, \{f\}, 1)$ appartiennent aux chemins $(0, a, \{i\}, 0)\,(0, a, \{f\}, 1)$, $(0, a, \{i\}, 0)\,(0, a, \{f\}, 1)$ et $(0, a, \{i\}, 0)\,(0, b, \{f\}, 1)$ respectivement.

(b) L'automate représenté figure 3.2(b) est un NTA complet. Il est non déterministe pour la même raison que l'automate précédent. Il est complet parce qu'à partir des états 0, 1 et 2, on peut lire les lettres a et b et il existe dans l'automate deux transitions initiales étiquetées a et b.



(a) NTA réduit (b) NTA complet

(c) DTA réduit (d) DTA complet

FIG. 3.2 – Exemples de FTA reconnaissant le langage $a^+(a+b)$.

(c) L'automate dépeint figure 3.2(c) est un DTA réduit. Il est réduit pour la même raison que l'automate représenté figure 3.2(a).

(d) L'automate de la figure 3.2(d) est un DTA complet pour la même raison que l'automate représenté figure 3.2(b).

□

3.3 D'un modèle à l'autre

Dans cette section, nous présentons, sous la forme de deux théorèmes, la construction d'un DTA à partir d'un DFA, et celle d'un FSA à partir d'un DTA. De cette façon, nous montrons l'équivalence des deux représentations (FSA et FTA) du point de vue classe de langages.

3.3.1 Transformation DFA vers DTA

La transformation d'un DFA vers un DTA est la plus simple. L'ensemble des états et l'alphabet sont exactement les mêmes pour les deux automates. L'ensemble des transitions du DTA est obtenu à partir de la fonction de transition du DFA. Une transition du DTA est initiale si elle est issue de l'état initial du DFA. Une transition est finale si l'état destination est un état final du DFA. Dans le théorème suivant, nous donnons la construction formelle du DTA et vérifions que l'automate obtenu reconnaît exactement le même langage que le DFA.

Théorème 3.1 Soient L un langage régulier ne contenant pas le mot vide et $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ un DFA reconnaissant L. Alors, il existe un DTA reconnaissant L, $\mathcal{A}' = (Q, \Sigma, T)$ tel que :

$$T = \{(q, l, m, q') \mid \delta(q, l) = q', [i \in m \iff q = q_0], [f \in m \iff q' \in F]\}$$

□

Preuve Montrons d'abord que $\mathcal{A}'$ est déterministe. Par hypothèse, on sait que l'automate $\mathcal{A}$ est déterministe. On a donc :

$$\forall q \in Q, \forall l \in \Sigma \ [|\delta(q, l)| \leq 1]$$

Ceci entraîne :

$$\forall q \in Q, \forall l \in \Sigma \ |\{(q, l, m, q') \mid q' \in Q\}| \leq 1 \tag{3.1}$$

De plus, on sait que

$$\forall q, q' \in Q, \forall l \in \Sigma, \forall (q, l, m, q') \in T^* \ [i \in m \iff q = q_0]$$

Donc,

$$\forall l \in \Sigma \ \{(q, l, m, q') \mid q' \in Q, i \in m, q \neq q_0\} = \emptyset$$

On en déduit que :

$$\forall l \in \Sigma \ \{(q, l, m, q') \mid q, q' \in Q, i \in m\} = \{(q_0, l, m, q) \mid q \in Q, i \in m\} \tag{3.2}$$

D'après (3.1) et (3.2), en posant $q = q_0$, on a directement :

$$\forall l \in \Sigma \ |\{(q, l, m, q') \mid q, q' \in Q, i \in m\}| \leq 1 \tag{3.3}$$

(3.3) et (3.1) impliquent que l'automate $\mathcal{A}'$ est déterministe.

Prouvons maintenant que $\mathcal{A}'$ et $\mathcal{A}$ reconnaissent le même langage L. Soit $w \in \Sigma^+$.

($\Rightarrow$)

$$\begin{array}{ll} & \text{le mot } w \text{ est reconnu par } \mathcal{A} \\ \Longrightarrow & \exists q' \in F \left[\delta\left(q_0, w\right) = q'\right] \\ \Longrightarrow & \exists q' \in Q \left[\left(q_0, w, \{i, f\}, q'\right) \in T^*\right] \\ \Longrightarrow & \text{le mot } w \text{ est reconnu par } \mathcal{A}'. \end{array}$$

($\Leftarrow$)

$$\begin{array}{ll} & \text{le mot } w \text{ est reconnu par } \mathcal{A}' \\ \Longrightarrow & \exists q, q' \in Q \left[\left(q, w, \{i, f\}, q'\right) \in T^*\right] \\ \Longrightarrow & \exists q' \in F \left[\delta\left(q_0, w\right) = q'\right] \\ \Longrightarrow & \text{le mot } w \text{ est reconnu par } \mathcal{A}. \end{array}$$

Ainsi, nous avons construit, pour le langage L, à partir d'un DFA reconnaissant L, un DTA reconnaissant le même langage L. **C.Q.F.D.**

3.3.2 Transformation DTA vers FSA

Dans cette section, nous exposons la transformation, plus complexe, d'un DTA (Q, Σ, T) vers un FSA $(Q', \Sigma, I, \delta, F)$. L'ensemble Q' des états contient trois fois plus d'états que Q. En effet, à chaque état de Q, correspondent dans Q' un état initial, un état final et un état ni initial, ni final. L'ensemble I des états initiaux contient tous les états initiaux de Q'. L'ensemble F des états finals contient tous les états finals de Q'. La fonction de transition δ est définie à partir de l'ensemble T en tenant compte de la marque portée par la transition. Le FSA ainsi obtenu est non déterministe. Le non déterminisme de l'automate obtenu est localisé uniquement sur l'état initial et il n'existe pas deux transitions portant la même étiquette et issues de deux états initiaux distincts. Il est donc très simple et peu coûteux de le transformer en un DFA. Le théorème suivant explicite la construction d'un FSA à partir d'un DTA et prouve que le langage accepté par les deux automates est le même.

Théorème 3.2 Soient L un langage régulier ne contenant pas le mot vide et $\mathcal{A} = (Q, \Sigma, T)$ un DTA reconnaissant L. Alors, il existe un FSA reconnaissant L, $\mathcal{A}' = (Q', \Sigma, I, \delta, F)$ tel que :

(1) $Q' = \bigcup_{q \in Q} \{(q, \emptyset), (q, \{i\}), (q, \{f\})\}$,

(2) $I = \{(q, \{i\}) \mid q \in Q\}$,

(3) $F = \{(q, \{f\}) \mid q \in Q\}$,

(4) pour tout état q et q' de Q, pour tout symbole l de Σ et pour toute marque m de $\{i, f\}$, on a :

$$(q', m \setminus \{i\}) \in \delta\left((q, m'), l\right) \text{ ssi } (q, l, m, q') \in T$$

avec $m' \in \{\{i\}, \emptyset, \{f\}\}$ si $i \in m$ et $m' \in \{\emptyset, \{f\}\}$ sinon.

□

Preuve Soit $w = l_1 l_2 \dots l_n \in \Sigma^+$.

(⇒) Supposons que le mot w est reconnu par $\mathcal{A}$. Nous allons montrer que w est aussi reconnu par $\mathcal{A}'$. Il existe un chemin

$$(q_0, l_1, m_1, q_1)(q_1, l_2, m_2, q_2) \dots (q_{n-1}, l_n, m_n, q_n)$$

dans $\mathcal{A}$ tel que $i \in m_1$ et $f \in m_n$. Nous montrons que, pour tout entier j compris entre 2 et n, la relation suivante est vraie :

$$(q_j, m') \in \{e \in \delta^*\left((q_1, m), l_2 l_3 \dots l_j\right) \mid m \in \{\emptyset, \{f\}\}\}$$

avec $m' = \{f\}$ si $f \in m_j$ et $m' = \emptyset$ sinon.

Nous effectuons cette preuve par induction sur la position des lettres du mot w.

Base d'induction Par définition, on a :

$$(q_2, m') \in \{e \in \delta\left((q_1, m), l_2\right) \mid m \in \{\emptyset, \{f\}\}\}$$

avec $m' = \{f\}$ si $f \in m_2$ et $m' = \emptyset$ sinon.

Ceci implique directement la propriété pour $j = 2$.

Étape d'induction Soit j un entier compris entre 2 et n. Étant donné que (q_{j-1}, l_j, m_j, q_j) appartient à T, on a :

$$(q_j, m') \in \{e \in \delta\left((q_{j-1}, m), l_j\right) \mid m \in \{\emptyset, \{f\}\}\}$$

avec $m' = \{f\}$ si $f \in m_j$ et $m' = \emptyset$ sinon.

Ceci entraîne directement le résultat, en utilisant l'hypothèse d'induction.

On sait de plus que :

$$\begin{cases} (q_1, \{f\}) \in \delta\left((q_0, i), l_1\right) & \text{si } f \in m_1 \\ (q_1, \emptyset) \in \delta\left((q_0, i), l_1\right) & \text{sinon} \end{cases}$$

En remarquant que $f \in m_n$, on obtient que :

$$(q_n, \{f\}) \in \delta^*\left((q_0, i), l_1 l_2 \dots l_n\right)$$

Le mot w est donc reconnu par $\mathcal{A}'$.

($\Leftarrow$) Nous supposons maintenant que le mot w est reconnu par $\mathcal{A}'$ et nous montrons qu'il l'est aussi par $\mathcal{A}$.

Nous savons qu'il existe $n+1$ états $q_0, q_1, \ldots q_n$ dans Q et $n+1$ marques $m_0, m_1, \ldots m_n$ dans $\{\emptyset, \{i\}, \{f\}\}$ tels que :

(1) $\forall i \in [1,n]\ \ (q_i, m_i) = \delta\left((q_{i-1}, m_{i-1}), l_i\right)$,

(2) $(q_n, m_n) \in F$,

(3) $(q_0, m_0) \in I$.

Nous en déduisons ainsi qu'il existe $m_i{}' \subseteq \{i, f\}$ telle que :

(1) $\forall i \in [1,n]\ \ (q_{i-1}, l_i, m_i{}', q_i) \in T$,

(2) $f \in m_n{}'$ car $(q_n, m_n) \in F$,

(3) $i \in m_1{}'$ car $(q_0, m_0) \in I$.

Il est alors évident que $(q_0, l_1 l_2 \ldots l_n, \{i, f\}, q_n) \in T^*$. Ainsi, le mot w est reconnu par $\mathcal{A}$.

Ainsi, nous avons construit, pour le langage L, à partir d'un DTA reconnaissant L, un FSA reconnaissant L. **C.Q.F.D.**

Exemple 3.2 Considérons le DTA $\mathcal{A} = (Q, \Sigma, T)$ avec $Q = \{0\}$, $\Sigma = \{a\}$ et $T = \{(0, a, \{i, f\}, 0)\}$, représenté sur la figure 3.3(a) page 46. Il reconnaît le langage régulier a^+. Nous expliquons, ci-dessous, la transformation de ce DTA en le FSA $\mathcal{A}'$ correspondant. Ce FSA est un quintuplet $(Q', \Sigma, I, \delta, F)$ où :

(1) l'ensemble Q' contient les trois états $(0, \emptyset)$, $(0, \{i\})$ et $(0, \{f\})$ construits à partir de l'état 0 du DTA $\mathcal{A}$,

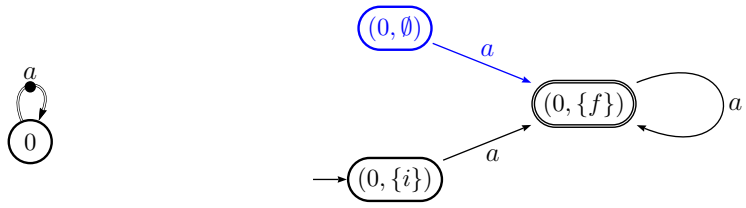
(2) l'alphabet Σ de $\mathcal{A}'$ est le même que celui de $\mathcal{A}$,

(3) l'ensemble des états initiaux I contient un seul état $(0, \{i\})$ car le DTA $\mathcal{A}$ a une transition initiale d'origine 0,

(4) l'ensemble de transitions de $\mathcal{A}'$ contient les trois transitions $((0, \emptyset), l, (0, \{f\}))$, $((0, \{i\}), l, (0, \{f\}))$ et $((0, \{f\}), l, (0, \{f\}))$ car la transition de $\mathcal{A}$ est initiale. Ces transitions définissent la fonction de transitions δ.

(5) l'ensemble des états finals F contient un seul état $(0, \{f\})$ car le DTA $\mathcal{A}$ a une transition finale de destination 0,

Le FSA résultat est montré sur la figure 3.3(b) page 46. □



(a) DTA (b) FSA

FIG. 3.3 – Exemple de transformation un DTA vers un FSA pour le langage a^+.

3.4 Une relation d'équivalence sur les mots adaptée aux DTA

Notre but est d'exhiber une relation d'équivalence sur les mots liée aux états d'un DTA. Nous définissons maintenant la relation d´équivalence $\equiv_L^+$ analogue à $\equiv_L$ qui nous sert à définir le DTA minimal. La difficulté réside dans le fait que la minimalité est définie par rapport aux états et non par rapport aux transitions. Nous rappelons que pour définir le DFA minimal d'un langage régulier, nous avons introduit la notion de quotient gauche. L'ensemble des états du DFA minimal $\mathcal{A}$ de L correspond à l'ensemble des quotients gauches de L par chacun des mots de Σ^*. Pour chaque état q de $\mathcal{A}$, le langage reconnu par l'automate $\mathcal{A}^{q\rightsquigarrow}$ est le quotient gauche de L associé à q.

Nous savons qu'un FTA ne peut pas reconnaître le mot vide. De ce fait, aucun FTA $\mathcal{A}$ ne possède d'état q tel que $\mathcal{A}^{q\rightsquigarrow}$ reconnaisse le mot vide. Pour calculer le DTA minimal d'un langage régulier ne reconnaissant pas le mot vide, nous utilisons le quotient gauche de L restreint à Σ^+ par tous les mots de Σ^*.

Nous introduisons maintenant la relation d´équivalence $\equiv_L^+$ sur $\Sigma^* \times \Sigma^*$ tenant compte de ces considérations.

Définition 3.5 (relation d'équivalence $\equiv_L^+$) La relation $\equiv_L^+ \subseteq \Sigma^* \times \Sigma^*$ est définie comme suit :

$$\boxed{u \equiv_L^+ v \iff u_{\Sigma^+}^{-1} L = v_{\Sigma^+}^{-1} L}$$

Pour simplifier l'écriture, on notera $[w]_L^+$ (et non pas $[w]_{\equiv_L^+}$) la classe d'équivalence qui contient w. □

Pour illustrer la définition précédente, nous considérons le langage régulier L représenté par l'expression régulière a^+ et nous comparons les relations $\equiv_L^+$ et $\equiv_L$ (voir page 23). La relation $\equiv_L$ définit sur L les deux classes d'équivalence

$[\varepsilon]_L = a^+$ et $[a]_L = a^*$. Par contre, la relation $\equiv_L^+$ ne définit qu'une seule classe sur L. En effet, $a_{\Sigma^+}^{-1}L = \varepsilon_{\Sigma^+}^{-1}L$ car $a_{\Sigma^*}^{-1}L$ est l'union de $\varepsilon_{\Sigma^*}^{-1}L$ et de $\{\varepsilon\}$.

Nous allons maintenant démontrer un théorème qui met en avant, pour tout langage régulier L, l'existence d'un DTA complet dont le nombre d'états est égal soit à l'index de $\equiv_L^+$, soit à l'index de $\equiv_L^+$ moins un, suivant le langage reconnu par l'automate.

Nous commençons par comparer l'index de $\equiv_L$ et de $\equiv_L^+$.

Proposition 3.1 La relation $\equiv_L$ raffine la relation $\equiv_L^+$. □

Preuve Soient u et v deux mots de Σ^* tels que $u \equiv_L v$. On a alors $u_{\Sigma^*}^{-1}L = v_{\Sigma^*}^{-1}L$. Ceci entraîne que $u_{\Sigma^+}^{-1}L = v_{\Sigma^+}^{-1}L$, c'est-à-dire $u \equiv_L^+ v$. Ceci prouve la propriété.
C.Q.F.D.

Proposition 3.2 Si $\equiv_L$ a un index fini, alors $\equiv_L^+$ a un index fini. □

Preuve Soient $\equiv_1$ et $\equiv_2$ deux relations d'équivalence. Si $\equiv_1$ raffine $\equiv_2$, alors toute classe d'équivalence de $\equiv_1$ est incluse dans une classe d'équivalence de $\equiv_2$ ([Harel *et al.* 2000])). Donc, l'index de $\equiv_1$ est supérieur ou égal à l'index de $\equiv_2$. Ceci entraîne que $\equiv_L^+$ a un index inférieur ou égal à $\equiv_L$. Ceci démontre la proposition. **C.Q.F.D.**

Dans la suite, nous utilisons $<_{abc}$ qui dénote la relation d'ordre alphabétique sur les mots. Nous introduisons maintenant une propriété sur les langages qui nous sert à définir la classe des langages L tels que ε est le seul élément de la classe $[\varepsilon]_L^+$ et les états de l'automate de transitions finies minimal de L correspondent aux classes de $\equiv_L^+$ sur $\Sigma^+ \times \Sigma^+$. Un langage L est dit *k-initial* si pour tout mot $w \in \Sigma^k$, il existe un mot $u \in \Sigma^+$ tel que $[uw]_L^+ = [w]_L^+$. Nous montrons maintenant l'inclusion suivante :

Propriété 3.1 Tout langage k-initial est $k+1$-initial. □

Preuve Soient $k \in \mathbb{N}$ et L un langage k-initial. Alors :

$$
\begin{array}{ll}
 & \forall w \in \Sigma^k, \exists u \in \Sigma^+ \ [uw]_L^+ = [w]_L^+ \\
\Longrightarrow & \forall l_1, \ldots, l_k \in \Sigma, \exists u \in \Sigma^+ \ [ul_1 \ldots l_k]_L^+ = [l_1 \ldots l_k]_L^+ \\
\Longrightarrow & \forall l_1, \ldots, l_k \in \Sigma, \exists u \in \Sigma^+, \forall l_{k+1} \in \Sigma \ [ul_1 \ldots l_k l_{k+1}]_L^+ = [l_1 \ldots l_k l_{k+1}]_L^+ \\
\Longrightarrow & \forall l_1, \ldots, l_k, l_{k+1} \in \Sigma, \exists u \in \Sigma^+ \ [ul_1 \ldots l_k l_{k+1}]_L^+ = [l_1 \ldots l_k l_{k+1}]_L^+ \\
\Longrightarrow & L \text{ est } k+1\text{-initial.}
\end{array}
$$

(a) DTA complet réduit sur $\Sigma^* \times \Sigma^*$ (b) DTA complet réduit sur $\Sigma^+ \times \Sigma^+$

FIG. 3.4 – Exemples de DTA.

C.Q.F.D.

Pour un langage 0-initial L, le nombre de classes d'équivalence de $\equiv_L^+$ est le même sur $\Sigma^+ \times \Sigma^+$ et sur $\Sigma^* \times \Sigma^*$ car $[\varepsilon]_L^+$ contient au moins deux éléments.

Pour tout langage L, on peut construire un automate dont les états correspondent aux classes d'équivalence de $\equiv_L^+$. Si L est 1-initial, pour chaque transition sortant de l'état correspondant à la classe $[\varepsilon]_L^+$, il existe une transition de même étiquette et de même destination sortant d'un autre état, par définition d'un langage 1-initial. Donc, dans le cas où le langage est 1-initial et pas 0-initial, ε est le seul élément de $[\varepsilon]_L^+$ et l'état correspondant à la classe $[\varepsilon]_L^+$ est alors inutile.

Nous notons $\mathcal{L}$ la classe des langages dont le DTA minimal complet, que nous définirons par la suite, a pour ensemble d'états l'ensemble des classes de $\equiv_L^+$ privé de $[\varepsilon]_L^+$. Formellement, $\mathcal{L}$ est l'ensemble des langages réguliers L 1-initial et pas 0-initial ne contenant pas le mot vide.

Dans le théorème suivant nous montrons, pour tout langage régulier L, l'existence d'un DTA complet qui accepte L dont le nombre d'états est en général égal à l'index de $\equiv_L^+$, et égal à un moins cet index pour la classe $\mathcal{L}$. Pour le démontrer, nous construisons un DTA (Q, Σ, T) en nous fondant sur les classes d'équivalence de $\equiv_L^+$ où :

(1) L'ensemble Q des états est l'ensemble des classes d'équivalence de la relation $\equiv_L^+$ sur $\Sigma^* \times \Sigma^*$. Dans le cas particulier où toutes les transitions initiales issues de $[\varepsilon]_L^+$ peuvent être remplacées par des transitions initiales issues de plusieurs états de l'automate, alors, l'ensemble des états peut être réduit à l'ensemble des classes d'équivalence de la relation $\equiv_L^+$ sur $\Sigma^+ \times \Sigma^+$.

La figure 3.4 montre un exemple de langage et de DTA tels que les deux transitions initiales issues de l'état 0 peuvent être remplacées par la transition t_a, étiquetée par a et bouclant sur l'état 1, et la transition t_b, d'origine 2 et de destination 1, et étiquetée par b. Dans cet exemple, on peut supprimer l'état 0, ajouter la marque initiale aux transitions t_a et t_b et reconnaître toujours le même langage.

(2) L'ensemble T des transitions est tel que, pour chaque état q de Q et pour chaque lettre a de l'alphabet, il existe une transition étiquetée a qui sort de q. Pour chaque transition, on indique si elle est finale et/ou initiale. On veut que (Q, Σ, T) soit un DTA. Chaque transition de T étiquetée l et dont la destination est $[l]_L^+$ peut porter la marque initiale. Si l'on marque effectivement toutes ces transitions, et en prenant l'ensemble des classes d'équivalence de $\equiv_L^+$ sur $\Sigma^* \times \Sigma^*$ comme ensemble d'états, on obtient pour le langage $a+(a+b)(aa+ba)^*aa$, l'automate complet saturé de la figure 3.5(a) page 50. Pour chaque langage, l'automate défini de cette façon est unique et peut donc être utilisé comme forme canonique. Cependant, dans la plupart des cas, cet automate est non déterministe. Pour qu'un FTA soit déterministe, il ne doit donc pas exister deux transitions initiales étiquetées par une même lettre dans T. Il n'existe en général pas de solution unique, pour un langage et un automate donné, dans l'affectation du caractère initial des transitions.

Par exemple, dans la figure 3.5(a) page 50, les deux transitions t_a et $t_a{}'$ étiquetées par a allant respectivement de l'état 0 vers l'état 1 et de l'état 3 vers l'état 1 peuvent être toutes deux initiales, mais n'ont pas besoin d'être toutes les deux marquées comme initiales.

Dans ce cas, nous choisissons arbitrairement de n'en considérer qu'une : celle dont l'état d'origine est la classe d'équivalence contenant le plus petit mot (selon l'ordre alphabétique).

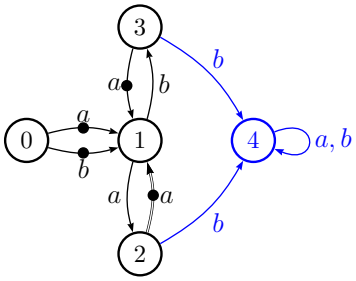
Ainsi, dans l'automate de la figure 3.5(b) page 50, la transition t_a est choisie car elle est issue de l'état correspondant à la classe d'équivalence $[\varepsilon]_L^+$ alors que $t_a{}'$ est issue de l'état correspondant à la classe d'équivalence $[a]_L^+$ et car $\varepsilon <_{abc} a$. Il aurait aussi été possible de choisir $t_a{}'$ (voir l'automate de la figure 3.5(c)) sans changer ni le langage reconnu, ni le déterminisme, ni la complétude.

Théorème 3.3 Soit L un langage régulier ne contenant pas le mot vide. Alors, si k est l'index de $\equiv_L^+$, il existe un DTA complet qui accepte L, avec $k-1$ états si $L \in \mathcal{L}$ et k états sinon. □

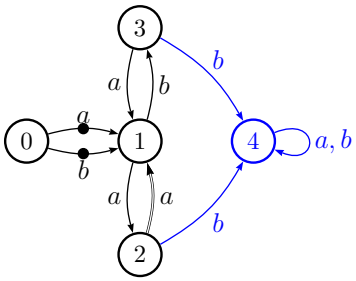
Preuve Nous construisons un DTA $\mathcal{A} = (Q, \Sigma, T)$ où

(1) les éléments de Q sont des classes d'équivalence de $\equiv_L^+$, c'est-à-dire :

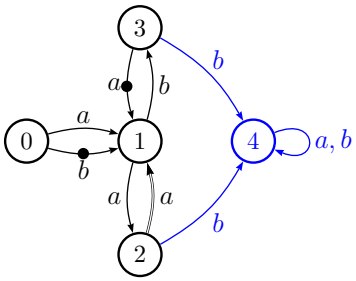
$$Q = \begin{cases} \left\{ [u]_L^+ \mid u \in \Sigma^+ \right\} & \text{si } \forall l \in \Sigma, \exists w \in \Sigma^+ \ \left[[l]_L^+ = [wl]_L^+ \right] \\ \left\{ [u]_L^+ \mid u \in \Sigma^* \right\} & \text{sinon} \end{cases}$$



(a) DTA complet saturé



(b) DTA complet



(c) DTA complet

FIG. 3.5 – Exemples de DTA du langage $(a+b)(aa+ba)^*aa+a$.

(2) l'ensemble des transitions est défini par :

$$\begin{aligned} T = \{([u], l, m, [ul]) \ \mid \\ & l \in \Sigma,\ \forall w \in [u]\ u \leq_{abc} w, \\ & (i \in m)\ \text{ssi}\ ([ul] = [l],\ (\forall v \in \Sigma^+\ [\,[vl] = [l]\ \Longrightarrow\ (u \leq_{abc} v)\ ])), \\ & (f \in m)\ \text{ssi}\ (ul \in L)\} \end{aligned}$$

Dans la suite nous établissons que, pour tout mot $w \in \Sigma^+$, il existe $u \in \Sigma^*$ tel que $([u], w, m, [w])$ appartient à T^* avec $(i \in m)$ et $(w \in L \iff f \in m)$. Nous effectuons cette preuve par induction sur la longueur de w.

Base d'induction Soit $l \in \Sigma$ un mot de longueur 1. Considérons les deux cas suivant :

(1) si $[\varepsilon] \in Q$, il existe une transition $([\varepsilon], l, m, [l])$ avec $i \in m$,

(2) sinon, par définition de l'ensemble des états, $\exists w \in \Sigma^+$ tel que la transition $([w], l, m, [l])$ existe. On prend le plus petit tel w. Il vérifie également $i \in m$.

Il nous reste à montrer que $(l \in L \iff f \in m)$, ce qui est bien le cas par définition de T.

Étape d'induction Soit $w = w'l \in \Sigma^+$ un mot de longueur n. On sait par hypothèse d'induction qu'il existe $u \in \Sigma^*$ et $m \in \{\{i, f\}, \{i\}\}$ tels que $([u], w', m, [w'])$ appartient à T^*. On sait aussi qu'il existe $([w'], l, m, [w]) \in T$ telle que $(f \in m \iff w \in L)$. Alors par définition de l'ensemble des transitions étendues, $([u], w, m, [w]) \in T^*$ avec $(i \in m)$ et $(f \in m \iff w \in L)$.

Nous avons ainsi prouvé que pour tout mot $w \in \Sigma^+$, il existe $u \in \Sigma^*$ tel que $([u], w, m, [w])$ appartient à T^* avec $(i \in m)$ et $(f \in m \iff w \in L)$. Donc, $w \in L$ si et seulement s'il existe $u \in \Sigma^*$ tel que $([u], w, \{i, f\}, [w])$ appartient à T^*. Ainsi, $L(\mathcal{A}) = L$.

De plus, en posant k l'index de $\equiv_L^+$ sur $\Sigma^* \times \Sigma^*$, alors l'index de $\equiv_L^+$ sur $\Sigma^+ \times \Sigma^+$ est $k-1$ si $L \in \mathcal{L}$ car pour tout mot w de Σ^+, $[\varepsilon]_L^+ \neq [w]_L^+$. Ceci entraîne que $k-1$ est le nombre d'états de $\mathcal{A}$. Dans le cas contraire, Q est égal à l'ensemble des classes d'équivalence de $\equiv_L^+$ sur $\Sigma^* \times \Sigma^*$, c'est-à-dire k. **C.Q.F.D.**

Théorème 3.4 Un langage L ne contenant pas le mot vide est régulier si et seulement si $\equiv_L^+$ a un index fini. □

Preuve

(⇒) Soit L un langage régulier. D'après le Théorème 2.2 page 23, la relation d'équivalence $\equiv_L$ a un index fini et d'après la Proposition 3.2 page 47, l'index de $\equiv_L^+$ est fini.

($\Leftarrow$) Le Théorème 3.3 page 49 prouve, pour tout langage régulier L ne contenant pas le mot vide, l'existence d'un DTA reconnaissant L.

C.Q.F.D.

3.5 Automate de transitions finies minimal

Pour définir le DTA minimal, nous avons besoin de connaître son nombre d'états. Pour cela, nous allons prouver un lemme que nous utiliserons pour montrer que, étant donnés un langage régulier L et k l'index de $\equiv_L^+$, le DTA minimal complet possède $k-1$ états si $L \in \mathcal{L}$ et k états sinon.

Soient L un langage régulier reconnu par un DTA (Q, Σ, T) et u et v deux mots de Σ^*. Le lemme suivant prouve que s'il existe dans le DTA deux chemins correspondant respectivement à u et à v, commençant par une transition initiale et terminant par des transitions ayant même destination, alors ces mots sont équivalents au sens de $\equiv_L^+$.

Lemme 3.1 Soient $\mathcal{A} = (Q, \Sigma, T)$ un DTA et $L = L(\mathcal{A})$. Alors :

$$\forall u, v \in \Sigma^*, \forall q \in Q \ \ [u, v \in L(\mathcal{A}^{\curvearrowright q}) \implies u \equiv_L^+ v]$$

où $\mathcal{A}^{\curvearrowright q}$ est défini à la page 39. □

Preuve Soient $u, v \in L(\mathcal{A}^{\curvearrowright q})$. Alors, clairement, $u_{\Sigma^*}^{-1} L = v_{\Sigma^*}^{-1} L = L(\mathcal{A}^{q\curvearrowright})$. Ainsi, $u \equiv_L v$ par définition (voir page 23). D'après la Propriété 3.1 page 47, la relation $\equiv_L$ raffine la relation $\equiv_L^+$. On obtient bien alors $u \equiv_L^+ v$. **C.Q.F.D.**

Le théorème suivant définit la taille en nombre d'états du DTA minimal complet reconnaissant un langage régulier donné L.

Théorème 3.5 Soient L un langage régulier ne contenant pas le mot vide et k l'index de $\equiv_L^+$. Alors, le nombre minimum d'états d'un DTA complet acceptant L est $k-1$ si $L \in \mathcal{L}$ et k sinon. □

Preuve Soit k l'index de $\equiv_L^+$. D'après le Théorème 3.3 page 49, il existe un DTA complet qui accepte L avec $k-1$ états si $L \in \mathcal{L}$ et k états sinon. Notons n le nombre d'états (n est égal à k ou $k-1$ suivant les cas). Nous prouvons maintenant que n est minimum. Supposons que L est accepté par un DTA complet $\mathcal{A} = (Q, \Sigma, T)$ de n' états où $n' < n$. Alors, il existe $q \in Q$ tel que $L(\mathcal{A}^{\curvearrowright q})$ contient des mots appartenant à deux classes d'équivalence distinctes de $\equiv_L^+$, c'est-à-dire qu'il existe $u, v \in L(\mathcal{A}^{\curvearrowright q})$ tels que $u \not\equiv_L^+ v$. Ceci contredit le Lemme 3.1. **C.Q.F.D.**

Nous pouvons maintenant définir formellement l'automate minimal. Cet automate est celui défini dans la preuve du Théorème 3.3 de la page 49.

Définition 3.6 (DTA minimal complet en nombre d'états) Le *DTA minimal complet en nombre d'états* reconnaissant un langage régulier L est (Q, Σ, T), où :

(1) Q est un sous-ensemble des classes d'équivalence de $\equiv_L^+$, c'est-à-dire :

$$Q = \begin{cases} \left\{[u]_L^+ \mid u \in \Sigma^+\right\} & \text{si } \forall l \in \Sigma, \exists w \in \Sigma^+ \ \left[[l]_L^+ = [wl]_L^+\right] \\ \left\{[u]_L^+ \mid u \in \Sigma^*\right\} & \text{sinon} \end{cases}$$

(2) l'ensemble des transitions est défini par :

$$\begin{aligned} T = \{([u], l, m, [ul]) \ \mid \\ & l \in \Sigma, \ \forall w \in [u] \ u \leq_{abc} w, \\ & (i \in m) \ \text{ssi} \ ([ul] = [l], \ (\forall v \in \Sigma^+ \ [\,[vl] = [l] \implies (u \leq_{abc} v)\ ])), \\ & (f \in m) \ \text{ssi} \ (ul \in L)\} \end{aligned}$$

□

L'automate de la figure 3.6(b) page 54 est le DTA minimal complet du langage $a + (a + b)(aa + ba)^*aa$. La relation $\equiv_L^+$ définit cinq classes sur ce langage :

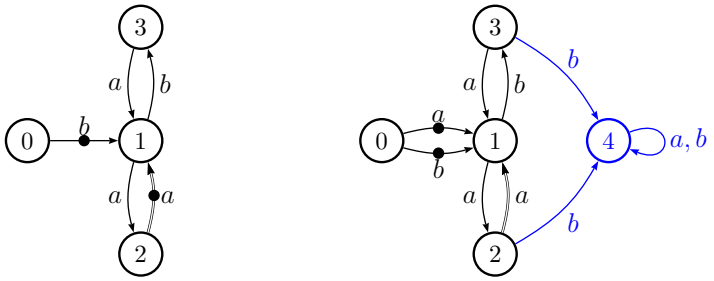
(1) $\varepsilon_{\Sigma^*}^{-1} L = L$ (correspondant à l'état 0),
(2) $a_{\Sigma^*}^{-1} L = (aa + ba)^*aa$ (correspondant à l'état 1),
(3) $(aa)_{\Sigma^*}^{-1} L = a + a(aa + ba)^*aa$ (correspondant à l'état 2),
(4) $(ab)_{\Sigma^*}^{-1} L = a(aa + ba)^*aa$ (correspondant à l'état 3),
(5) $(aab)_{\Sigma^*}^{-1} L = \emptyset$ (correspondant à l'état 4),

Nous définissons maintenant l'*automate de transitions finies déterministe minimal réduit*. Intuitivement, il correspond à l'automate de transitions finies déterministe minimal complet dans lequel on a supprimé les transitions inutiles et les états correspondant aux classes d'équivalence de $\equiv_L^+$ dont le quotient gauche est vide. Formellement :

Définition 3.7 (DTA minimal réduit en nombre d'états) Le *DTA minimal réduit en nombre d'états* reconnaissant un langage régulier L est (Q, Σ, T), où :

(1) Q est un sous-ensemble des classes d'équivalence de $\equiv_L^+$ avec un quotient gauche non vide, c'est-à-dire :

$$Q = \begin{cases} \left\{[u]_L^+ \mid u \in \Sigma^+, u_{\Sigma^*}^{-1} L \neq \emptyset\right\} & \text{si } \forall l \in \Sigma, \exists w \in \Sigma^+ \ \left[[l]_L^+ = [wl]_L^+\right], \\ \left\{[u]_L^+ \mid u \in \Sigma^*, u_{\Sigma^*}^{-1} L \neq \emptyset\right\} & \text{sinon} \end{cases}$$



(a) DTA minimal réduit

(b) DTA minimal complet

FIG. 3.6 – DTA minimaux reconnaissant le langage $(a+b)(aa+ba)^*aa+a$.

(2) l'ensemble des transitions est défini par :

$$\begin{aligned} T = \{([u], l, m, [ul]) \ \mid \\ & l \in \Sigma,\ \forall w \in [u]\ u \leq_{abc} w, \\ & ([u] = \{\varepsilon\} \implies \forall w \in \Sigma^+ [([w], l, m, [l]) \in T \implies w = \varepsilon]), \\ & (i \in m) \text{ ssi } ([ul] = [l],\ (\forall v \in \Sigma^+ \ [\ [vl] = [l] \implies (u \leq_{abc} v)\ ])), \\ & (f \in m) \text{ ssi } (ul \in L)\} \end{aligned}$$

□

L'automate de la figure 3.6(a) est le DTA minimal réduit du langage $a+(a+b)(aa+ba)^*aa$.

3.6 Comparaison de la taille des DFA et DTA minimaux complets

Dans cette section, nous comparons la taille en nombre d'états et de transitions, pour un langage régulier donné, des DTA et DFA minimaux complets reconnaissant ce langage.

Propriété 3.2 Soit L un langage régulier. Le DTA minimal complet reconnaissant L ne possède pas plus d'états ni de transitions que le DFA minimal complet reconnaissant le langage L. □

Preuve En ce qui concerne le nombre d'états, la propriété est une conséquence directe de la Proposition 3.1 page 47 ($\equiv_L^+$ a un index inférieur ou égal à $\equiv_L$) et des Théorèmes 2.1 page 23 et 3.5 page 52 (sur la taille en nombre d'états des DFA et DTA complets).

En ce qui concerne le nombre de transitions, notons k, la taille de l'alphabet du langage régulier L. Étant donné que les automates sont complets, chaque état possède exactement k transitions sortantes. On a donc, à la fois pour le DFA et pour le DTA, exactement k fois plus de transitions que d'états.

C.Q.F.D.

Propriété 3.3 Pour tout langage régulier L, le nombre d'états du DFA minimal complet reconnaissant le langage L est au plus $2s+1$ si s est le nombre d'états du DTA minimal complet reconnaissant le même langage. □

Preuve Pour prouver cette propriété, on peut remarquer qu'aucune classe d'équivalence de $\equiv_L^+$ ne contient trois classes d'équivalence de $\equiv_L$. On peut prouver cette observation par l'absurde. Supposons qu'il existe trois mots u_1, u_2 et u_3 de Σ^* tels que $[u_1]_L$, $[u_2]_L$ et $[u_3]_L$ sont trois classes d'équivalence distinctes de $\equiv_L$ et tels qu'ils appartiennent à la même classe d'équivalence de $\equiv_L^+$, c'est-à-dire : (⋆)

$$[u_1]_L^+ = [u_2]_L^+ = [u_3]_L^+$$

Ceci implique que :

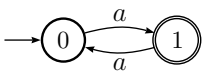
$$u_1 {}^{-1}_{\Sigma^+} L = u_2 {}^{-1}_{\Sigma^+} L = u_3 {}^{-1}_{\Sigma^+} L$$

Ceci implique que :

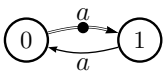
$$\left\{u_1 {}^{-1}_{\Sigma^*} L, u_2 {}^{-1}_{\Sigma^*} L, u_3 {}^{-1}_{\Sigma^*} L\right\} \subseteq \left\{u_1 {}^{-1}_{\Sigma^+} L, u_1 {}^{-1}_{\Sigma^+} L \cup \{\varepsilon\}\right\} \ ,$$

en contradiction avec l'hypothèse que $[u_1]_L$, $[u_2]_L$ et $[u_3]_L$ sont distinctes.

D'après l'observation (⋆), on déduit qu'au maximum deux classes de $\equiv_L$ sont incluses dans une classe d'équivalence de $\equiv_L^+$. Comme on sait que toute classe d'équivalence de $\equiv_L$ est incluse dans une classe d'équivalence de $\equiv_L^+$ (conséquence de la Proposition 3.1 page 47), on déduit que l'index de j de $\equiv_L$ est au maximum deux fois plus élevé que l'index k de $\equiv_L^+$ si $L \notin \mathcal{L}$ et au maximum deux fois moins une plus élevé que k sinon (car dans ce cas $\left|[\varepsilon]_L^+\right| = 1$). Le nombre d'états du DFA minimal complet reconnaissant L est égal à l (Théorème 2.1). D'après le Théorème 3.5 page 52, on sait que le DTA minimal complet reconnaissant L possède $k-1$ états si $L \in \mathcal{L}$ et k états sinon avec k l'index de $\equiv_L^+$. Alors, si s est le nombre d'états du DTA minimal complet reconnaissant L, le DFA minimal complet reconnaissant L a donc au plus :



(a) DFA minimal complet réduit



(b) DTA minimal complet réduit

FIG. 3.7 – Les DFA et DTA minimaux du langage $a(aa)^*$ ont la même taille.

(1) $2s + 1$ états si $L \in \mathcal{L}$,

(2) $2s$ sinon

C.Q.F.D.

Les deux propriétés précédentes comparent de façon très précise le nombre d'états des DFA et DTA minimaux complets pour un langage donné. On peut obtenir les mêmes résultats sur le nombre de transitions en remarquant que, dans un DFA ou DTA complet sur un alphabet de taille k, chaque état possède exactement k transitions sortantes.

De plus, les automates des figures 3.7 et 3.8 montrent que les bornes définies dans les propriétés précédentes sur la taille des DFA et DTA minimaux complets d'un même langage sont atteintes. L'exemple suivant détaille la figure 3.8.

Exemple 3.3 la figure 3.8 page 57 présente les DFA et DTA minimaux complets du langage engendré par la grammaire G suivante : $G = (\Sigma, V, S, P)$ avec $\Sigma = \{a, b\}$, $V = \{S, S_0, S_1\}$ et

$$\begin{aligned} P = \{\ & S \to aS_0 \mid bS_1 \mid b, \\ & S_0 \to aS_0 \mid bS_1, \\ & S_1 \to aS_1 \mid bS_0 \mid a \mid b\ \} \end{aligned}$$

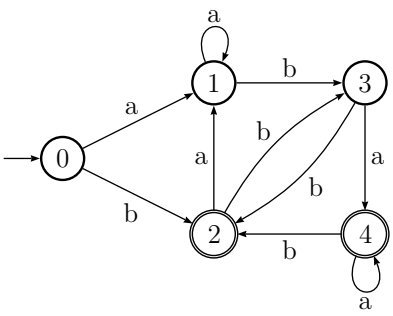
Le langage L engendré par la grammaire G est l'union de l'ensemble contenant uniquement le mot b et de l'ensemble des mots commençant par un a ou un b, suivi d'un mot contenant un nombre impair de b, puis d'un a ou d'un b.

Soit B_p le langage reconnaissant les mots de Σ^* avec un nombre pair de b et B_i celui reconnaissant les mots avec un nombre impair de b. Le langage L est $b+(a+b)B_i(a+b)$. La relation d'équivalence $\equiv_L$ définit cinq classes d'équivalence :

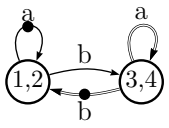
(1) $\varepsilon^{-1}_{\Sigma^*}L = L$ (correspondant à l'état 0),

(2) $a^{-1}_{\Sigma^*}L$ (correspondant à l'état 1) est $B_i(a + b)$,

(3) $b^{-1}_{\Sigma^*}L = a^{-1}_{\Sigma^*}L \cup \{\varepsilon\} = B_i(a + b) + \varepsilon$ (correspondant à l'état 2),



(a) DFA minimal complet réduit



(b) DTA minimal complet réduit

FIG. 3.8 – DFA et DTA minimaux.

(4) $(ab)^{-1}_{\Sigma^*}L$ (correspondant à l'état 3) est $B_p(a+b)$,

(5) $(aba)^{-1}_{\Sigma^*}L = (ab)^{-1}_{\Sigma^*}L \cup \{\varepsilon\} = B_p(a+b)+\varepsilon$ (correspondant à l'état 5).

La relation d'équivalence $\equiv^+_L$ définit les trois classes d'équivalence suivantes :

(1) $\varepsilon^{-1}_{\Sigma^+}L = L$ (ne correspondant à aucun état),

(2) $a^{-1}_{\Sigma^+}L$ (correspondant à l'état $(1,2)$) est $B_i(a+b)$,

(3) $(ab)^{-1}_{\Sigma^+}L$ (correspondant à l'état $(3,4)$) est $B_p(a+b)$.

On peut remarquer que ε est le seul mot de la classe $[\varepsilon]$ et que $\varepsilon^{-1}_{\Sigma^+}L \subset a^{-1}_{\Sigma^+}L \cup (ab)^{-1}_{\Sigma^+}L$. Cela implique que les états du DTA sont définis par les classes d'équivalence de la relation $\equiv^+_L$ restreinte à $\Sigma^+ \times \Sigma^+$. Ainsi, l'index de $\equiv^+_L$ restreinte à $\Sigma^+ \times \Sigma^+$ est de 2.

Notons de plus qu'en général il est beaucoup plus facile d'identifier un langage à partir de son DTA minimal complet qu'à partir de son DFA minimal complet. La figure 3.8 illustre cette observation. □

3.7 Discussion

Dans ce chapitre, nous avons introduit une nouvelle représentation des langages réguliers appelée automates de transitions finies (FTA) qui a la particularité de regrouper toute l'information sur les transitions. Ces machines sont très proches de celles de la sous-classe des machines de Mealy [Mealy 1955] pour lesquelles les sorties sont réduites à "acceptation" et "rejet". La seule différence est

que l'information initiale est portée par les transitions dans les FTA et non par les états.

Remarquons que le modèle des FTA pour les machines de Mealy est l'analogue du modèle des automates d'états finis (FSA) pour les machines de Moore [Moore 1956] qui a été présenté dans le chapitre 2.

Dans ce chapitre, nous avons aussi montré que le modèle des FTA possède la propriété d'être au moins aussi compact que celui des FSA. Dans les cas les plus favorables, il peut être jusqu'à deux fois plus compact que celui des FSA. Dans le chapitre 5, nous proposons un algorithme d'inférence régulière qui utilise ce type d'automates.

Chapitre 4

Alphabets ordonnés sur des séquences de protéines

Dans ce chapitre, nous abordons le problème de la prise en compte dans l'inférence de l'information sur les propriétés physico-chimiques entre les acides aminés. Dans la première section, nous décrivons brièvement les acides aminés et leur rôle dans la structure des protéines. Nous passons ensuite en revue les propriétés physico-chimiques des acides aminés et montrons leur importance dans l'étude des protéines. La troisième section, correspond à un état de l'art succinct sur la prise en compte de ces propriétés dans l'établissement d'une ressemblance entre acides aminés. Dans la quatrième section, nous donnons notre approche de la ressemblance entre acides aminés, en proposant une méthode de transformation automatique d'une couverture en une relation d'ordre partielle sous forme de treillis, exploitable par un algorithme d'inférence grammaticale.

4.1 Les acides aminés et les protéines

Les acides aminés sont formés d'un groupement amine NH_2, d'un groupement carboxyle COOH, d'un atome d'hydrogène et d'une chaîne latérale représentant un radical organique, le tout lié à un atome de carbone appelé carbone α (voir figure 4.1 page 60). Il existe 20 acides aminés principaux dans les protéines. Ils sont caractérisés par leur chaîne latérale. Ils s'assemblent pour former des protéines qui adoptent une certaine conformation spatiale étroitement liée à leur fonction.

La structure des protéines. La structure spatiale d'une protéine est souvent très compacte et spécifique à la protéine en question. Si la même chaîne

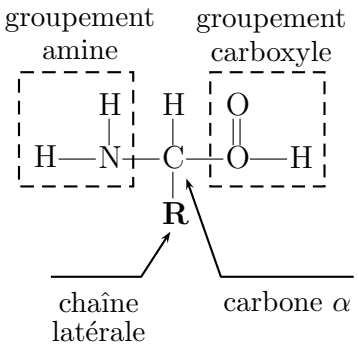


FIG. 4.1 – Un acide aminé

d'acides aminés est synthétisée plusieurs fois, dans un même contexte adéquat de repli, elle adoptera toujours la même conformation à quelques différences minimes près [Kabsch et Sander 1983, Argos 1987]. Les lois régissant le repli sont mal connues. La structure d'une protéine peut être considérée à plusieurs niveaux. La structure primaire correspond simplement à l'enchaînement des acides aminés. La structure secondaire concerne la structure spatiale locale. Les hélices α et les feuillets β sont les deux structures locales principales. La structure tertiaire correspond à la forme globale. Enfin, certaines protéines sont formées d'un agrégat de plusieurs chaînes d'acides aminés. Ce niveau de description est appelé structure quaternaire. Deux séquences différentes peuvent adopter une même structure. Plusieurs hypothèses peuvent expliquer ce fait :

(1) tous les acides aminés de la séquence ne sont pas déterminants pour la structure,

(2) ce n'est pas l'acide aminé situé à une position donnée qui compte mais une de ses propriétés physico-chimiques,

(3) ce ne sont pas les acides aminés isolés qui influent sur le repli de la protéine mais les relations entre les acides aminés.

Les protéines contiennent des régions que l'on appelle *sites actifs* et qui interagissent avec d'autres molécules. Ce qui importe, c'est la forme du site (dépendant de la séquence). Pour obtenir cette structure locale particulière, les acides aminés présents dans ces zones doivent répondre à des critères très précis pour conserver la structure locale, à la différence de ceux présents dans le reste de la séquence. La taille de ces sites varie suivant leurs fonctions. Elle peut être très petite (cas d'un site structural), plus étendue (cas des sites de fixation à une séquence nucléique) ou égale à la taille de la séquence (principalement pour les protéines de structure). Plus la taille est grande et moins les acides aminés sont contraints. Les sites actifs peuvent se trouver à la surface de la protéine [Creighton 1993],

souvent dans des régions peu structurées appelées boucles [Rooman *et al.* 1992] ou à l'intérieur des protéines. C'est au centre de la protéine et dans les structures secondaires que l'entassement des atomes doit exiger les propriétés chimiques ou stériques les plus strictes.

Nous n'aborderons pas ici le problème complexe des liens entre structure et fonction.

4.2 Propriétés physico-chimiques des acides aminés

Les biophysiciens ont regroupé les connaissances considérées comme pertinentes pour la structure des protéines sous forme de propriétés physiques et chimiques des acides aminés. Les propriétés physiques (par exemple : branchu, volumineux, flexible ou court) se rapportent à la structure de la chaîne latérale des acides aminés. Les propriétés chimiques (par exemple : chargé, hydrophobe, aromatique ou réactif) sont quant à elles déterminées par la constitution atomique et par leurs liaisons covalentes. Ces propriétés déterminent le rôle de chaque acide aminé au sein de la protéine.

[Richardson et Richardson 1989] proposent une étude approfondie des acides aminés et fournissent un catalogue de leurs propriétés ainsi qu'une preuve expérimentale du rôle qu'ils jouent dans la structure de la protéine. Ils expliquent également l'importance de la chaîne latérale. Les propriétés physico-chimiques peuvent être vues comme des échelles permettant d'ordonner les acides aminés. Parmi les plus connues, on trouve l'hydrophobicité dont Cornette et ses collaborateurs ont fait une synthèse dans [Cornette *et al.* 1987]. [Kidera *et al.* 1985] ont analysé 188 propriétés physico-chimiques collectées dans des travaux précédent et les ont classées dans neuf groupes généraux.

Le tableau 4.1 page 62 donne une idée des propriétés physico-chimiques des acides aminés que l'on trouve dans les protéines. Elles sont importantes pour mesurer la ressemblance entre les acides aminés.

Ces propriétés sont prises en compte dans les analyses de séquences via les mesures de ressemblance qu'elles introduisent sur les acides aminés. Cette ressemblance est exploitée aussi bien dans les algorithmes de comparison de séquences, de découverte de motifs, ou en ce qui nous concerne, pour la fusion de transitions.

4.3 Ressemblance entre acides aminés

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

Nom	Code	Principales propriétés
Alanine	Ala (A)	très petit, hydrophobe
Arginine	Arg (R)	positif, polaire
Asparagine	Asn (N)	petit, polaire
Acide Aspartique	Asp (D)	petit, négatif
Cystéine	Cys (C)	petit, polaire, hydrophobe
Glutamine	Gln (Q)	polaire
Acide Glutamique	Glu (E)	polaire, négatif
Glycine	Gly (G)	très petit, hydrophobe ou hydrophile
Histidine	His (H)	aromatique, hydrophobe, positif, polaire
Isoleucine	Ile (I)	aliphatique, hydrophobe
Leucine	Leu (L)	aliphatique, hydrophobe
Lysine	Lys (K)	hydrophobe, polaire, positif
Méthionine	Met (M)	hydrophobe
Phénylalanine	Phe (F)	hydrophobe, aromatique
Proline	Pro (P)	petit
Sérine	Ser (S)	très petit, polaire
Thréonine	Thr (T)	petit, hydrophobe, polaire
Tryptophane	Trp (W)	hydrophobe, aromatique, polaire
Tyrosine	Tyr (Y)	hydrophobe, aromatique, polaire
Valine	Val (V)	petit, aliphatique, hydrophobe

TAB. 4.1 – Les acides aminés

qu'il 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 peut donc parler d'une ressemblance entre les acides aminés. Dans cette section nous voyons comment il est possible de mesurer cette ressemblance.

4.3.1 Matrices de substitution

La représentation de la ressemblance la plus utilisée pour prendre en compte la connaissance sur les acides aminés dans l'analyse des protéines est une matrice dite de *substitution* qui va associer à chaque paire d'acides aminés une valeur numérique qui quantifie leur similarité. Deux méthodes philosophiquement différentes existent. La première consiste à comparer directement les acides aminés entre eux suivant certaines de leurs caractéristiques. On peut par exemple calculer le nombre maximum de nucléotides que les codons de deux acides aminés ont en commun. Les matrices obtenues sont alors appelées génétiques [Feng *et al.* 1985] [Fitch 1966] [Fitch 1967]. En général [Grantham 1974] [Miyata *et al.* 1979] [Rao 1987], les caractéristiques utilisées sont de nature physico-chimique.

La deuxième façon de procéder est indirecte puisque la fréquence des acides aminés est prise en compte dans un ensemble fixé de protéines. L'idée est d'aligner les séquences et d'observer la fréquence de substitution des acides aminés. De nombreuses matrices ont été proposées en utilisant cette méthode [Dayhoff *et al.* 1978] [Henikoff et Henikoff 1992] [Overington *et al.* 1992] [Luthy *et al.* 1991].

Les matrices de substitution posent plusieurs problèmes. Ils concernent la qualité de l'alignement initial, la sélection du jeu de séquences initial et le contexte des acides aminés. Ce dernier problème a été abordé par [Overington *et al.* 1992] [Overington *et al.* 1990] [Luthy *et al.* 1991] [Gonnet *et al.* 1994].

En pratique, malgré ces problèmes, les matrices donnent de bons résultats. Elles mesurent assez bien la ressemblance en moyenne, mais ne sont pas capables de distinguer des ressemblances plus subtiles. En effet, elles sont parfaitement adaptées lorsqu'aucune information contextuelle sur la ressemblance entre les acides aminés n'est fournie sur les séquences. Les matrices donnent des relations entre les paires d'acides aminés. Lorsqu'il s'agit de comparer plus de deux acides aminés, ce type de matrices est difficile à utiliser. En effet, Il est difficile de capter la ressemblance d'acides aminés partageant une même propriété en se fondant uniquement sur des relations binaires entre les acides aminés. Considérer la somme des paires de similarités n'est pas satisfaisant. Par exemple, l'utilisation des matrices devient limitante lorsqu'elles sont appliquées aux régions locales de la séquence protéique où, pour des raisons structurelles, la liberté de mutation d'un résidu particulier peut être grandement restreinte. Par exemple, la matrice de Dayhoff contient une information déformée quant à la résistance des cystéines à la mutation. Ceci vient sans doute de la nécessité de conserver les ponts disulfures.

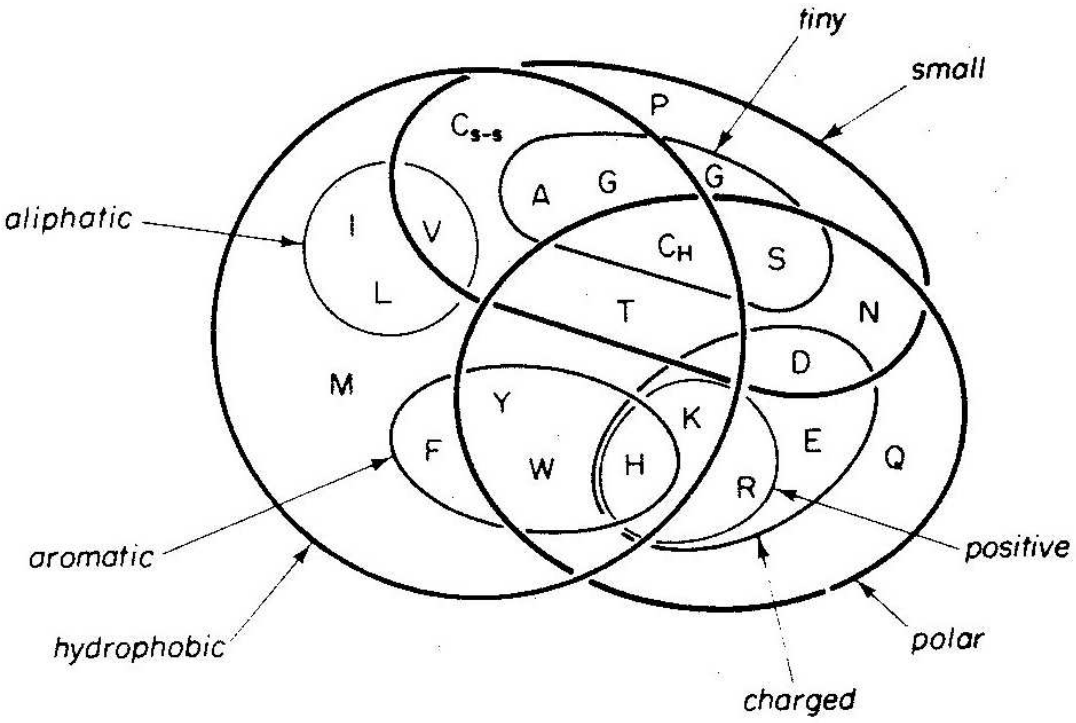


FIG. 4.2 – Classification de conservation des acides aminés de W. R. Taylor

De ce fait, l'utilisation de cette matrice de probabilités sur des cystéines n'étant pas impliquées dans un pont disulfure conduit inéluctablement à l'erreur. Dans la section suivante, nous présentons une méthode qui permet d'avoir une approche plus fine pour mesurer la potentialité de substitution des acides aminés.

4.3.2 Couvertures

Dans la section précédente nous avons vu qu'il existait beaucoup de matrices. Chacune capte la ressemblance entre les acides aminés suivant un point de vue, mais aucune ne peut réunir toute la richesse des liens existant entre les acides aminés. D'autres structures comme les diagrammes de Venn (par exemple celui de la figure 4.2 proposé par W. R. Taylor [Taylor 1986]) permettent de mieux modéliser la complexité des liens entre les acides aminés. Ce diagramme est fondé d'une part sur des données de mutation (matrice de Dayhoff) et d'autre part sur des propriétés physico-chimiques. [Sagot *et al.* 1995a] [Sagot *et al.* 1995c] ont montré que ce type de structures est le mieux adapté pour modéliser les ressemblances. Notons de plus que le diagramme de Venn donne une description qualitative de la ressemblance entre les acides aminés sans en interdire une ma-

nipulation quantitative [Taylor 1986] en utilisant le formalisme de la théorie des ensembles.

Le tableau 4.3.2 page 66 propose une vue synthétique des travaux ayant utilisés sous diverses formes la notion de couverture. Dans le tableau, pour chaque couverture, nous indiquons le modèle le plus simple permettant de la représenter ((P) si ce modèle est une partition, (A) si c'est une structure hiérarchique sous forme d'arbre et (DV) si c'est un diagramme de Venn).

L'importance des propriétés des acides aminés a d'abord été mise en évidence par Dayhoff et ses collaborateurs dans [Dayhoff *et al.* 1972]. Ils ont remarqué que la loi de substitution entre acides aminés n'était pas uniforme mais qu'au contraire, certains remplacements étaient fortement préférés à d'autres. Dayhoff a proposé cinq *groupes d'échange* non chevauchants (hydrophobe, aromatique, positif, petit et disulfide) tels que les remplacements d'acides aminés s'opèrent généralement entre ceux d'un même groupe. Il a suggéré que la structure d'une protéine est une fonction de ces cinq propriétés chimiques et physiques. De la même façon, d'autres chercheurs ont proposé des couvertures représentables par une partition [Dayhoff *et al.* 1978] [Miyata *et al.* 1979] [Karlin et Ghandour 1985] [Mocz 1995] [Cannata *et al.* 2002] [Figureau *et al.* 2003] [Majewski et Ott 2003] [Andersen et Brunak 2004]. Ce type de couverture à l'avantage d'être très simple et permet de coder les séquences protéiques par un système de réécriture [Overington *et al.* 1992] [Ioerger 1996] [Gambin *et al.* 2002]. Son pouvoir d'expression est aussi très faible et, de ce fait, il est insuffisant pour modéliser la complexité des substitutions entre les acides aminés, qui sont dépendantes du contexte des acides aminés [Ioerger 1997]. Des structures plus complexes ont aussi été présentées. [Smith et Smith 1990] [Murphy *et al.* 2000] [Prlic *et al.* 2000] [Fan et Wang 2003] proposent une structure hiérarchisée, sous forme d'arbre, plus riche que la partition, mais toujours insuffisante pour prendre en compte différents points de vue.

[Taylor 1986] a généré une liste étendue de propriétés en prenant des sous-ensembles et sur-ensembles des groupes d'échange de Dayhoff, ainsi que des intersections et unions arbitraires de groupes. Il a proposé 70 classes qui paraissent pertinentes pour la structure des protéines. Cette classification plus souple que les précédentes peut être représentée sous forme de diagramme de Venn (voir figure 4.2). D'autres couvertures avec la même souplesse ont été avancées [Jiménez-Montaño et Zamora-Cortina 1981] [Naor *et al.* 1996] [Klingler 1996] [Wu et Brutlag 1996] [Sagot 1996] [Ioerger 1997] [Galitsky *et al.* 1998] [Yu 2001] [Li *et al.* 2003]. [Jiménez Montaño et Ramos Fernández 2004] proposent une revue des partitions existantes sur les acides aminés.

Pour illustrer la richesse de l'information ensembliste, nous reprenons l'illustration de la thèse de M.-F. Sagot [Sagot 1996] (figure 4.3). Ici, il ne s'agit pas de comparer des acides aminés mais des animaux. Nous considérons trois animaux

Référence	**Propriétés**	**Groupes de substitution**
[Dayhoff *et al.* 1972]	(P, A, DV)	(P) C, FWY, HKR, ILMV, ADEGNPQST (A) C, ST, NQ, DE, AGP, FWY, HKR, ILMV, ADEGNPQST (DV) C, ST, NQ, DE, HW, CS, FWY, HKR, AGP, ILMV, FILM, CFHILMSW, ADEGNPQST
[Dayhoff *et al.* 1978]	(P)	C, FWY, HKR, DENQ, ILMV, AGPST
[Miyata *et al.* 1979]	(P)	C, FWY, HKR, DENQ, ILMV, AGPST
[Jiménez-Montaño et Zamora-Cortina 1981]	(DV)	AG, DE, KR, NQ, ST, FWY, ILMV, CFILMVWY, ADEGHKNPQRST
[Karlin et Ghandour 1985]	(P)	DE, AGILV, NQ, FWY, HKR, ST, P, CM DE, HKR, AFILMPVW, CGNQSTY DE, HKR, ACFGILMNPQSTVWY ACGPSTWY, DEHKNQR, FILMV
[Taylor 1986]	(DV)	Environ 70 groupes obtenus par union, intersection et complémentation de HKR, ILV, ACGS, HFWY, DEHKR, ACDGNPSTV, CDEHKNQRSTWY, ACFGHIKLMTVWY
[Smith et Smith 1990]	(A)	P, AG, DE, NQ, ST, FWY, HKR, ILV, CFILMVWY, DEHKNQRST
[Mocz 1995]	(P)	AEHMQY, FIKLVW, CDGNPRST
[Naor *et al.* 1996]	(DV)	DN, GP, DGNP, EKQR, FILV, DEKNQR, ACFILMVWY, DEGHKNPQRST
[Klingler 1996]	(DV)	H, K, N, AP, CF, DE, GS, KR, AGS, ILV, NQR, QTY, HMTWY, CFILMVW
[Wu et Brutlag 1996]	(DV)	FY, IV, DE, ST, DN, HY, KR, AS, EQ, ILV, FWY, EKQ, AST, KQR, FLY, ILMV, EKQR, FILMV, FILMY, FILMVY
[Sagot 1996]	(DV)	ACGST, ACDGNSTV, FHWY, ILMV, HKR, DE, QN, P ILMV, ACGST, HKR, KR, FHWY, FWY, DE, QN, P, C, W
[Ioerger 1997]	(DV)	ILMV, FILMVW, AILMPV, AFILMPVW, FWY, GN, ACGS, ST, DE, NQ, DENQ, HKR, RK, DEHKR, NQST, DENQST, DEHKNQR, DEHKNQRST
[Galitsky *et al.* 1998]	(DV)	C, P, W, FY, ST, DE, NQ, AGS, HKR, AFILMV
[Wang et Wang 1999]	(P)	proposent une méthode de recherche du nombre minimal de classes qu'une partition sur les acides aminés peut posséder pour obtenir des bonnes propriétés de repli pour les protéines,
[Murphy *et al.* 2000]	(A)	AG, ST, FY, KR, LM, IV, EQ, DN, FYW, CHP, HKR, ILMV, DENQ, AGST, ILMVC, AGPST, DEHKNQR, AGILMPSTVC, AFGILMPSTVCWY
[Prlic *et al.* 2000]	(A)	3 arbres de 18, 19 et 19 nœuds
[Yu 2001]	(DV)	258 groupes valides et 31 groupes conservés
[Cannata *et al.* 2002]	(P)	130 partitions dont la meilleure est ACGPST, DEHKNQR, FILMVWY
[Figureau *et al.* 2003]	(P)	A, G, T, PS, DN, EK, LM, IV, HQR, CFWY A, C, P, S, DE, DKNQ, H, GT, ILMV, FWY A, C, P, ER, DN, KQ, GH, ST, LMW, FIVY
[Fan et Wang 2003]	(A)	LM, IV, EQ, ST, HN, KR, FWY, AST, DEQ, CFWY, ILMV, APST, DEHKNQR, CFILMVWY, ADEGHKNPQRST
[Majewski et Ott 2003]	(P)	DE, HKR, CGNQSTY, AFILMPVW
[Li *et al.* 2003]	(P, A, DV)	(P) AGSTP, DEHKNQR, CFILMVWY (A) 18 groupes qui ne s'entrecroisent pas (DV) 26 groupes chevauchants
[Andersen et Brunak 2004]	(P)	18 partitions contenant entre 2 et 19 groupes qu'ils comparent avec celles de [Wang et Wang 1999]

TAB. 4.2 – Groupes de substitution entre acides aminés disponibles dans des recherches déjà menées

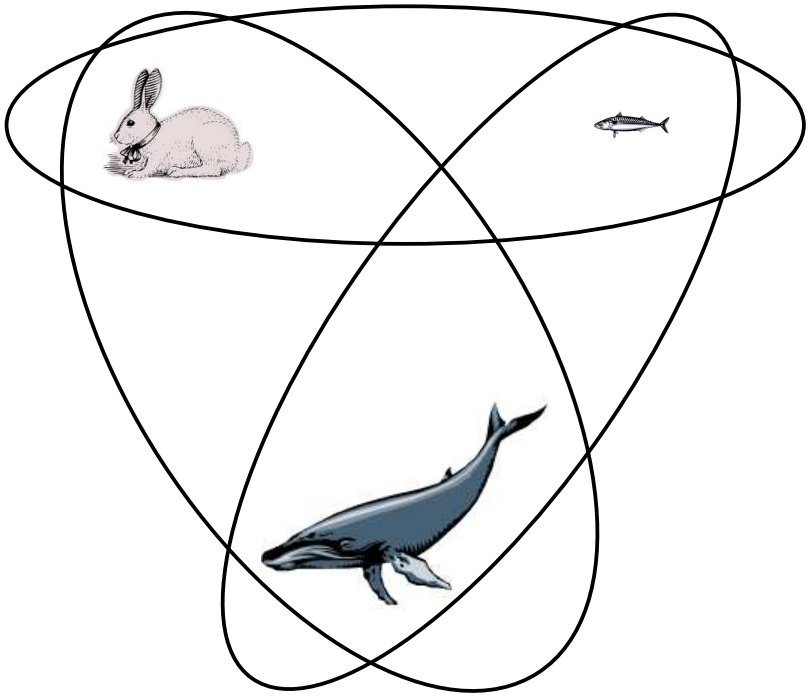


FIG. 4.3 – points de vue différents

(un lapin, une baleine et une sardine) sous trois angles différents. Un zoologiste va vouloir rapprocher les deux mammifères que sont le lapin et la baleine. Le patron d'une entreprise alimentaire va plutôt regrouper la sardine et la baleine car ces deux animaux vivent dans le milieu marin. Finalement, un enfant souhaitant adopter un animal de compagnie associera le lapin et la sardine à cause de leur taille. Ces trois points de vues correspondent à trois classifications qui ne peuvent pas être captées par une seule matrice mais qu'un diagramme de Venn modélise parfaitement. Cet exemple illustre que la ressemblance entre des objets dépend du sous-ensemble de leurs propriétés qui est pertinent pour leur fonction. Les critères de ressemblance varient par exemple suivant la famille de protéines que l'on étudie, mais aussi suivant que l'acide aminé se situe dans une hélice α ou dans un brin β.

Nous avons voulu, pour la méthode que nous décrivons dans cette thèse, nous fonder sur un critère de ressemblance qui n'est pas fixé *a priori* mais qui permet de comparer les acides aminés suivant différents points de vue.

4.4 Notre approche de la ressemblance des acides aminés

Pour mesurer la ressemblance entre les acides aminés, nous proposons d'utiliser un treillis[Grätzer 1971, Birkhoff 1973]. Cette structure possède des propriétés mathématiques intéressantes pour la généralisation et contribuera donc à l'efficacité de l'inférence grammaticale. Une classification de ce type permet de définir un ordre partiel sur les acides aminés qui renferme toute l'information des classes intéressantes inférables à partir d'une couverture comme celle de W. R. Taylor. Dans cette section, nous commençons par rappeler les notions élémentaires d'ordre partiel, de borne inférieure et supérieure et de treillis. Nous détaillons ensuite une méthode de construction automatique d'un treillis, à partir d'une couverture des acides aminés, définissant ainsi un ordre sur les groupes d'acides aminés.

4.4.1 Ordres partiels et treillis

Définition 4.1 (ordre partiel, borne supérieure, borne inférieure)

(1) Une relation $\sqsubseteq$ est un *ordre partiel* si elle est réflexive, antisymétrique et transitive.

(2) La *borne supérieure* de deux éléments a et b d'un ordre partiel, notée $a \sqcup b$, est le plus petit élément tel que $a \sqsubseteq a \sqcup b$ et $b \sqsubseteq a \sqcup b$.

(3) De façon duale, la *borne inférieure* $a \sqcap b$ est le plus grand élément tel que $a \sqcap b \sqsubseteq a$ et $a \sqcap b \sqsubseteq b$.

□

Définition 4.2 (treillis) Un *treillis* A est un ordre partiel tel que :

(1) il existe un plus petit élément, $\bot$ et un plus grand élément $\top$,

(2) $a \sqcup b$ et $a \sqcap b$ existent pour toute paire d'éléments $a, b \in A$.

□

Exemple 4.1 La figure 4.4 page 69 représente un treillis qui classifie les acides aminés de $H \cup S$ où H et S représentent respectivement l'ensemble des acides aminés hydrophobes et petits. Les acides aminés appartenant à l'ensemble $H \cup S$ peuvent, d'après le diagramme de Taylor, être classés en 6 groupes : $H \cup S$, H, S, $H \setminus S$, $S \setminus H$ et $H \cap S$. Ces groupes correspondent aux noeuds du treillis. $\emptyset$ est le plus petit élément et $H \cup S$ est le plus grand. L'ordre partiel est celui d'inclusion.

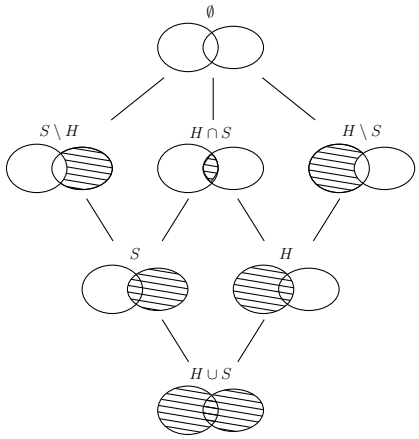


FIG. 4.4 – Treillis de classification des acides aminés hydrophobes ou petits

Par exemple, $H \sqsubseteq (H \cap S) \sqsubseteq \emptyset$. $(H \cup S)$ est la borne supérieure des éléments $(S \setminus H)$ et $(H \setminus S)$ et S est celle de $(S \setminus H)$ et de S. $(H \cap S)$ est la borne inférieure de S et H. □

Propriété 4.1 [Birkhoff 1973] Dans un treillis, la borne supérieure de n'importe quelle paire de sommets est unique. □

4.4.2 Construction d'un treillis sur l'ensemble des acides aminés

Nous décrivons une méthode de transformation d'une couverture donnée C en une relation d'ordre partiel sur les groupes d'acides aminés sous forme de treillis. Nous proposons de construire cet ordre de façon systématique.

Construction du treillis en quatre étapes

(1) *Ensemble initial :* on construit un ensemble N contenant les acides aminés, l'ensemble vide et les groupes de l'ensemble initial I défini par la couverture C.

(2) *Stabilité pour les intersections :* on calcule itérativement les intersections entre les paires d'ensembles de $N \times N$ et on remplace N par l'ensemble des intersections calculées. Le calcul se termine quand l'ensemble des intersections est égal à N (point fixe).

(3) *Graphe des inclusions :* on construit un graphe G sur N en reliant deux de ses ensembles par un arc si le premier est inclus dans le second.

(4) *Treillis :* la transitivité est enlevée du graphe G.

Treillis obtenu pour le diagramme de Taylor Dans les expérimentations, la couverture que nous considérons est issue du diagramme de Taylor. Elle définit un ensemble initial I qui contient l'ensemble des propriétés élémentaires du diagramme de Taylor (voir la figure 4.2) et ses complémentaires, ainsi que les 70 classes identifiées par W. R. Taylor comme étant pertinentes pour la structure des protéines [Taylor 1986].

Nous obtenons un treillis contenant 672 sommets et 2397 arcs dont le plus petit élément est l'ensemble vide et le plus grand élément est l'ensemble des acides aminés. Une petite partie du treillis est représenté figure 4.5 page 71. Nous associons à chaque sommet n du treillis son *niveau*, qui est la longueur du chemin maximal menant à n à partir du sommet correspondant à l'ensemble vide. Par exemple, les sommets $\{i\}$ et $\{v\}$ sont au niveau 1 du treillis de la figure et le sommet $\{i, l, v\}$ est au troisième niveau.

Élagage du treillis Ce treillis ainsi constitué peut être élagué aux niveaux les plus hauts, où les classes sont trop larges pour être utilisées avec confiance (Elles nécessitent un très grand échantillon pour être considérées sans aucun doute comme une propriété conservée). Pour les expériences que nous avons menées, nous avons utilisé le treillis restreint qui est construit comme suit : (1) nous créons, au niveau 7, un sommet contenant tous les acides aminés et nous le connectons avec tous les sommets de niveau inférieur à 7 qui sont connectés à un sommets de niveau 7 et (2) nous enlevons du treillis basé sur le diagramme de Taylor tous les sommets dont le niveau est supérieur ou égal à 7 sauf celui contenant tous les acides aminés. Le treillis élagué contient 421 nœuds et 1278 arcs et est montré en annexe A.

Ce treillis est utilisé dans notre opération de fusion de transitions à travers la relation de borne supérieure (voir Définition 4.1 page 68). Il s'agit maintenant de pouvoir utiliser cette structure de la manière la plus efficace possible dans le contexte de généralisation des acides aminés. Dans la section suivante nous présentons différent types de codages pour les treillis. Il est aussi important de noter que notre algorithme n'est pas dépendant du treillis lui même et peut être utilisé sur d'autres treillis ordonnançant les acides aminés.

4.5 Codage des treillis

Un grand nombre d'implémentations ont été proposées pour coder les relations d'ordre sous forme de treillis. Les opérations courantes effectuées sur les treillis sont, entre autres, le calcul de la borne inférieure ou supérieure et le test consistant

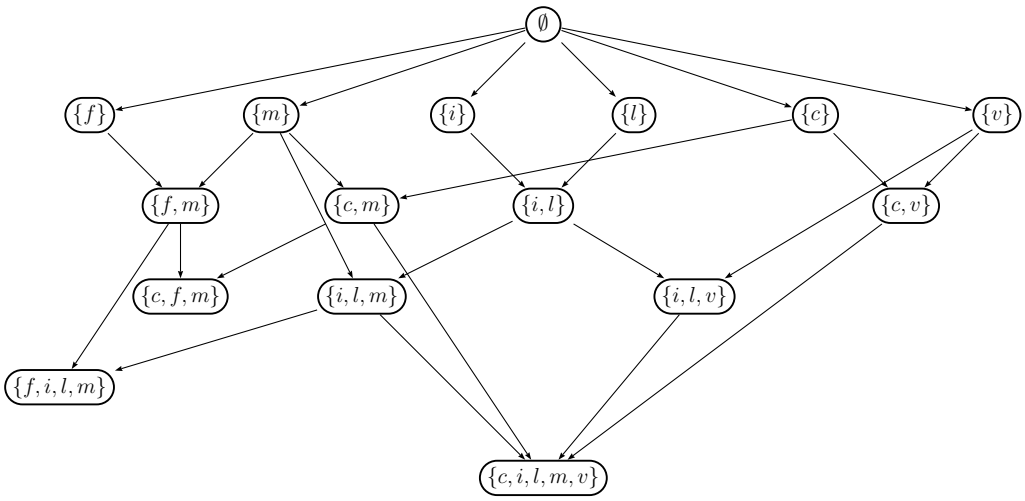


FIG. 4.5 – Partie du treillis basé sur le diagramme de Taylor.

à déterminer si un élément est ancêtre d'un autre. La plupart des implémentations utilisées pour coder les treillis ont été développées pour l'écriture de compilateurs, dans le but d'améliorer le test de l'inclusion de types. Ce test sert à déterminer si un type est un sous-type d'un autre. Cette opération correspond à déterminer si un élément est ancêtre d'un autre.

Les trois paragraphes suivants présentent quelques méthodes en temps constant développées dans ce cadre. Elles proposent toutes un codage des éléments du treillis par vecteurs de bits. La disjonction de deux éléments ancêtres l'un de l'autre correspond au descendant et la conjonction à l'ancêtre.

Matrice binaire Le codage le plus simple des treillis est celui par matrice binaire. Chaque case de la matrice correspond à un couple (x, y) de nœuds du treillis et contient 1 si y est un ancêtre de x et 0 sinon.

Le codage par ce type de matrice permet de tester en temps constant si un élément est ancêtre d'un autre. Son principal inconvénient est qu'il nécessite un espace quadratique. Les autres algorithmes que nous présentons peuvent être vus comme des formes compressées du codage par matrice binaire.

Algorithme de Cohen N. H. Cohen a proposé dans [Cohen 1991] le premier algorithme utilisant un espace mémoire restreint. Cet algorithme est utilisable uniquement pour les pré-ordres tels que chaque élément a au maximum un parent. Chaque type est identifié par un nombre. À chaque élément du treillis on associe le chemin complet de la racine vers cet élément par une séquence d'éléments.

Codages hiérarchiques Comme nous l'avons vu au premier paragraphe, une façon simple, mais inefficace, de construire les vecteurs de bits est d'associer à chaque élément sa ligne correspondante dans la matrice binaire. Les vecteurs de bits résultants sont fortement épars. Ceci est dû au fait qu'en général le nombre d'ancêtres d'un élément est beaucoup plus petit que le nombre total d'éléments. Des techniques plus efficaces ont été proposées, en particulier la méthode de modulation de H. Ait-Kaci et collaborateurs [Ait-Kaci *et al.* 1989] et la technique du codage par gène de Y. Caseau [Caseau 1993] qui essaye de minimiser la taille des vecteurs de bits utilisés. [Vitek *et al.* 1997] ont proposé trois codages différents pour les treillis. Le premier, le codage compacté est une extension, pour les sous-types multiples, de l'algorithme de N. H. Cohen [Cohen 1991]. Le deuxième est le codage compacté par bit. Il réduit davantage l'espace nécessaire du premier codage au prix d'un test de l'inclusion de types moins efficace. Le dernier codage, appelé le codage compact a été conçu pour les hiérarchies très grandes.

Tous ces codages permettent de tester si un élément est ancêtre d'un autre. Par contre, seulement le codage par matrice binaire et ceux proposés par H. Ait-Kaci permettent de calculer la borne inférieure ou supérieure. Le codage que nous utilisons pour notre treillis est l'un de ceux de H. Ait-Kaci [Ait-Kaci *et al.* 1989], qui est plus efficace que ceux par matrice binaire. Ce codage très simple associe à chaque élément du treillis, si c'est possible, la disjonction des codes de ses fils. Ceci permet d'obtenir un code très compact.

4.6 Conclusion

Comme nous l'avons vu dans ce chapitre, beaucoup de travaux ont été réalisés pour déterminer des groupes d'acides aminés. De plus, une partie de ces travaux utilisent une structure de treillis pour ordonnancer les groupes. Souvent, ces treillis sont de taille très restreinte et proposent une vision des acides aminés dans un cadre particulier donné. Nous proposons un treillis de taille beaucoup plus importante utilisable dans n'importe quel contexte et basé sur le diagramme de Taylor. Ce treillis contient un grand nombre d'informations sur les propriétés physico-chimiques des acides aminés. À la différence des autres treillis qui ne contiennent qu'un petit sous-ensemble des propriétés, l'avantage de celui-ci est de ne pas restreindre *a priori* les propriétés physico-chimiques utiles à l'étude des protéines. La difficulté est que le choix de la (ou des) propriétés qu'un acide aminé doit posséder suivant qu'il se trouve dans telle ou telle région conservée d'une protéine est plus difficile et demande à être identifié pendant l'apprentissage.

Chapitre 5

Inférence grammaticale par fusion de transitions sur des alphabets ordonnés

Ce chapitre décrit notre méthode d'inférence grammaticale à partir d'instances positives pour l'apprentissage de motifs dans les séquences de protéines. Le but de cette méthode est de calculer, à partir d'un ensemble de séquences de protéines appartenant à une même famille, un automate non déterministe représentant le langage décrivant au mieux les séquences de l'ensemble de départ. L'automate généré doit reconnaître toutes les séquences de l'échantillon d'apprentissage et être suffisamment général pour prédire de nouvelles séquences potentielles pour la famille. Au début du chapitre, nous introduisons les principales étapes de la méthode, puis nous les expliquons dans le détail.

5.1 Schéma général d'inférence

Il existe plusieurs méthodes qui permettent de découvrir des similarités entre des séquences parmi lesquelles on trouve la découverte de motifs et l'inférence grammaticale. La première méthode donne de bons résultats mais les motifs générés ne décrivent qu'une région des séquences. La deuxième méthode produit une description globale des séquences mais ne donne pas des résultats suffisamment précis sur les protéines. La méthode d'inférence grammaticale pour la découverte de motifs dans les protéines que nous proposons dans ce chapitre allie ces deux méthodes pour produire un résultat tirant parti de leurs avantages respectifs.

Les principales étapes sont résumées dans la figure 5.1 (voir page 75). La méthode prend en entrée un échantillon de séquences biologiques. La méthode commence par un calcul de statistiques sur l'échantillon donné, apportant ainsi une

connaissance globale sur les séquences de l'échantillon. Elles sont utilisées dans la deuxième étape qui consiste à construire, pour chaque paire de séquences de l'échantillon, le meilleur ensemble d'alignements locaux deux à deux compatibles sans trou. Cette étape permet d'identifier les régions communes à une proportion importante des séquences de l'échantillon. La méthode calcule alors l'automate de transitions finies en accord avec les similarités calculées à l'étape précédente. On compacte enfin l'automate par fusion au niveau des transitions peu représentatives. Ce dernier constitue le modèle décrivant la famille de protéines donnée.

5.2 Définitions et notations préliminaires

Avant de détailler chaque pas de notre méthode d'inférence grammaticale, nous introduisons des définitions et notations préliminaires. Nous commençons par définir les alignements de séquences. Le but est ensuite de définir des ensembles compatibles d'alignements locaux sans trou, pour lesquels nous proposons un score. Ce type d'alignement permettra en effet de décider des fusions à considérer dans l'automate de reconnaissance des séquences.

5.2.1 Alignement de séquences

Dans cette section, nous définissons le vocabulaire utilisé en alignement de séquences. D'une manière informelle, un alignement consiste à mettre en évidence les similarités entre plusieurs séquences. Le but est d'associer, tant que possible, chaque lettre d'une des séquences avec la lettre la plus ressemblante de l'autre, tout en conservant l'ordre des lettres.

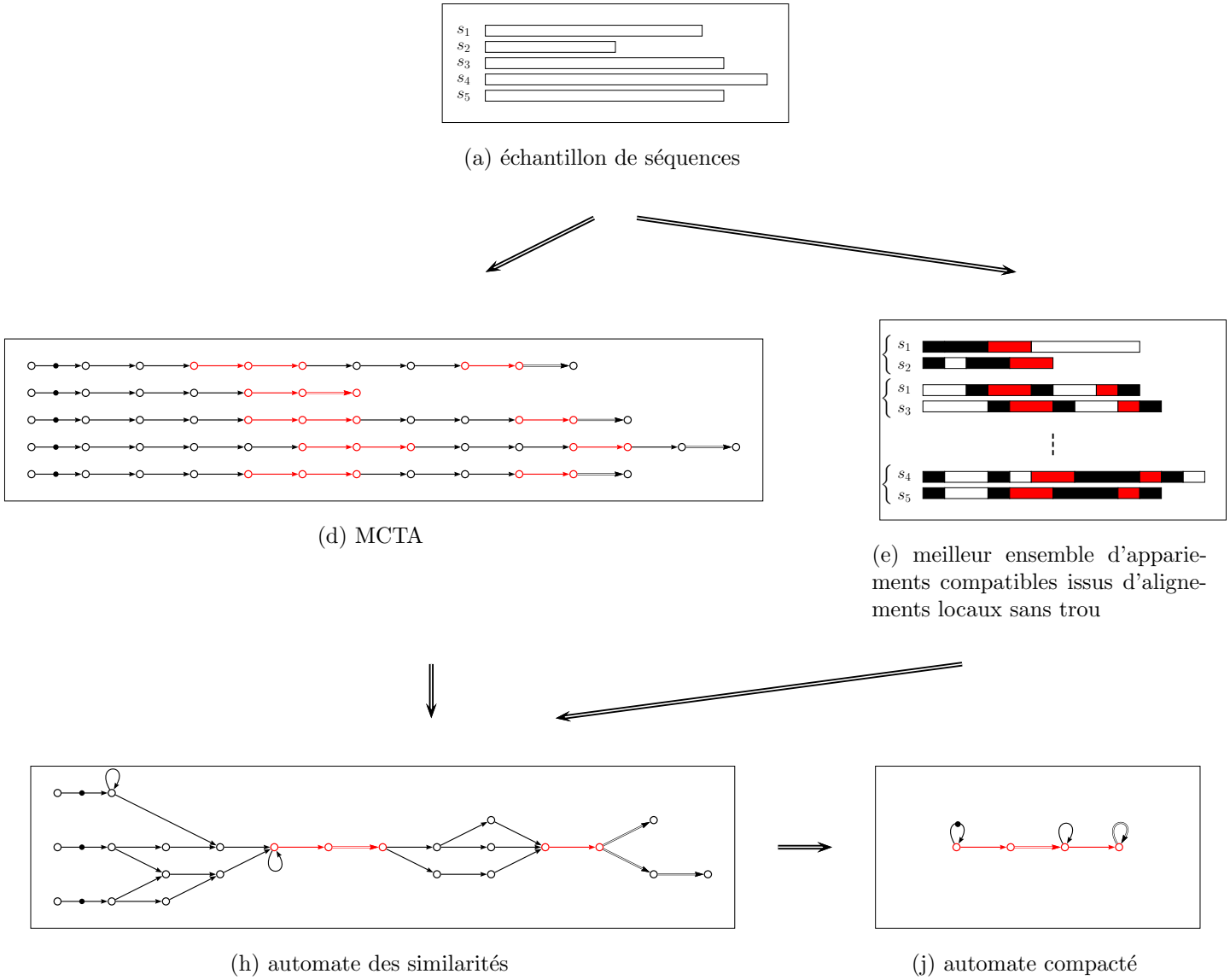
Dans un alignement, certaines lettres d'une des séquences sont alignées avec une lettre d'une autre séquence. On dit que ces deux lettres sont *appariées*. On note $A \leftrightarrow B$ l'appariement des lettres A et B. Cette relation est une relation d'équivalence. Dans le cas contraire, la lettre est alignée avec un '$-$' appelé *trou*.

Les séquences peuvent être alignées soit sur toute leur longueur, on parle alors d'*alignement global*, soit sur une portion de la séquence, on parle alors d'*alignement local*. Pour plus de détails, le lecteur intéressé par l'état de l'art pourra se référer à la thèse de J.-P. Comet [Comet 1998].

Nous définissons formellement ces notions. Notons Σ l'alphabet sur lequel sont définies les séquences et $\Sigma \cup \{-\}$ celui sur lequel sont définis les alignements.

Définition 5.1 (alignement global entre n séquences) Un *alignement global entre n séquences* $E = \{S_1, S_2, \ldots, S_n\}$, avec S_i défini sur Σ pour tout $i \in [1, n]$, est un ensemble $\overline{E} = \left\{\overline{S}_1, \overline{S}_2, \ldots, \overline{S}_n\right\}$ tel que :

(1) pour tout $i \in [1, n]$, $\overline{S}_i$ est défini sur $\Sigma \cup \{-\}$,



(a) échantillon de séquences

(d) MCTA

(e) meilleur ensemble d'appariements compatibles issus d'alignements locaux sans trou

(h) automate des similarités

(j) automate compacté

FIG. 5.1 – Méthode d'inférence pour la recherche de motifs.

(2) tous les mots de $\overline{E}$ ont la même longueur,

(3) toute séquence S_i de E s'obtient par suppression des trous de la séquence $\overline{S}_i$ dans $\overline{E}$,

(4) pour toutes les positions i de l'alignement, il existe au moins une séquence de $\overline{E}$ dont le $i^{ème}$ caractère n'est pas un trou.

□

Soit S une séquence. On note $S[i]$ la $i^{ème}$ lettre de S et $S[i,j]$ la sous-chaîne de la séquence S entre les positions i et j. On note aussi $A[i]$ la $i^{ème}$ position d'un alignement A.

Définition 5.2 (alignement local entre n séquences) Un *alignement local entre n séquences* $\{S_1, S_2, \ldots, S_n\}$ est un alignement global entre des sous-chaînes des séquences $\{S_1[l_1, r_1], S_2[l_2, r_2], \ldots, S_n[l_n, r_n]\}$ où $\forall i \in [1,n], 1 \leq l_i \leq r_i \leq |S_i|$. □

Nous introduisons maintenant la notion de *sous-alignement*. Un sous-alignement est une partie d'un alignement. Les correspondances entre les acides aminés sont les mêmes dans le sous-alignement et dans la partie correspondante de l'alignement. Formellement,

Définition 5.3 (sous-alignement) Un *sous-alignement* d'un alignement local ou global $\overline{E} = \left\{\overline{S}_1, \overline{S}_2, \ldots, \overline{S}_n\right\}$ entre n séquences est un alignement

$$\left\{\overline{S}_1[l,r], \overline{S}_2[l,r], \ldots, \overline{S}_n[l,r]\right\}$$

où $\forall i \in [1,n], 1 \leq l \leq r \leq \left|\overline{S}_i\right|$. □

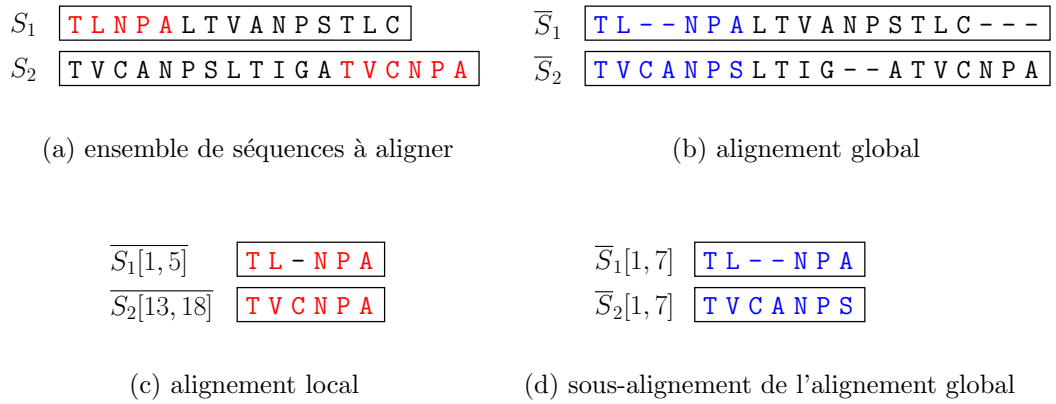
Un sous-alignement d'un alignement A, débutant à la position l de A et terminant à la position r est noté $A[l,r]$.

Exemple 5.1 La figure 5.2 page 77 donne des exemples d'alignements.

(a) La figure 5.2(b) donne un exemple d'alignement global entre les deux séquences d'acides aminés de la figure 5.2(a).

(b) Un exemple d'alignement local entre les deux séquences d'acides aminés de la figure 5.2(a) est donné à la figure 5.2(c).

(c) La figure 5.2(d) donne un exemple de sous-alignement de l'alignement global de la figure 5.2(b).



(a) ensemble de séquences à aligner

(b) alignement global

(c) alignement local

(d) sous-alignement de l'alignement global

FIG. 5.2 – Exemples d'alignements.

□

Nous pouvons observer qu'un sous-alignement d'un alignement local ou global entre n séquences est un alignement local entre ces séquences.

En pratique, nous nous intéresserons aux alignements locaux sans trou.

5.2.2 Contexte d'une paire de positions dans une paire de séquences

Nous introduisons maintenant la notion de contexte d'une paire de positions dans une paire de séquences. Le contexte droit de taille t d'une paire de positions i et j respectivement dans les séquences S_1 et S_2 est l'ensemble constitué des mots que l'on peut construire sur $\Sigma_1 \dots \Sigma_t$ où Σ_k est constitué des $k^{èmes}$ lettres des mots de $S_1[i+1, i+t]$ et $S_1[j+1, j+t]$ pour tout $k\ in[1, t]$. Le contexte gauche est défini de la même façon. Notons que si la séquence S_1 (respectivement S_2) a une taille inférieure à $i+t$ (respectivement $j+t$), on complétera les contextes par le mot $\$^{i+t-|S_1|}$ (respectivement $\$^{j+t-|S_2|}$) où $\$$ est un caractère spécial.

Définition 5.4 (contexte d'une paire de positions dans une paire de séquences) Soient S_1 et S_2 deux séquences, i une position de S_1 et j une position de S_2.

$$\begin{aligned}\mathsf{contexte}_{\mathsf{droit}}\left(S_1\left[i\right], S_2\left[j\right], t\right) = &\{l_1 l_2 \dots l_t \mid \forall k \in [1,t], \\ &l_k \in \{S_1\left[i+k\right], S_2\left[j+k\right]\}\} \\ \mathsf{contexte}_{\mathsf{gauche}}\left(S_1\left[i\right], S_2\left[j\right], t\right) = &\{l_1 l_2 \dots l_t \mid \forall k \in [1,t], \\ &l_k \in \{S_1\left[i+k-t-1\right], S_2\left[j+k-t-1\right]\}\}\end{aligned}$$

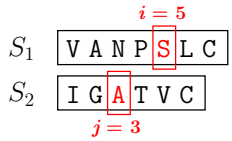


FIG. 5.3 – Exemple de contexte d'une paire de positions

□

Remarquons qu'en pratique, nous utilisons de petits contextes (de taille 2 typiquement).

Exemple 5.2 Cet exemple illustre le calcul des contextes gauche et droit de taille 2 des positions i de la séquence S_1 et j de la séquence S_2 (voir figure 5.3). Les contextes gauches sont NP, NG, IP et IG, et les contextes droits sont LC, LV, TC et TV. □

5.2.3 Une proposition de calcul du score d'un alignement local

Les méthodes classiques d'alignement local génèrent les alignements locaux maximaux qui ont un score positif. Dans cette thèse nous proposons de ne pas générer les alignements les plus grands possibles, mais des alignements ne contenant pas de position correspondant à une substitution entre acides aminés trop différents. Dans les protéines, l'information mutuelle est traditionnellement calculée pour des alignements multiples sur des lettres à des positions données. Dans ce chapitre nous proposons un score d'alignements qui prend en compte deux éléments. Le premier est le treillis que nous avons construit à partir du diagramme de Taylor (voir section 4.4 page 68). La relation entre le score et le treillis est importante à cause de notre utilisation des alignements. En effet, à chaque position d'un alignement que nous générons et sélectionnons correspondra, dans l'automate résultat, une transition dont l'étiquette sera la borne supérieure entre les acides aminés présents, à cette position, dans l'une et l'autre des séquences. Le deuxième élément que nous voulons prendre en compte dans le score est la dépendance entre une lettre et son contexte.

Dans cette section, nous introduisons les nouveaux indices nécessaires à l'élaboration du nouveau score que nous proposons pour comparer deux alignements locaux de séquences. Nous exposons dans un premier temps le calcul du score pour une paire de positions dans une paire de séquences donnée et nous finissons la section en proposant une formule de calcul de score total d'un alignement local sans trou.

Score contextuel d'une paire de positions dans une paire de séquences protéiques Nous disposons d'un alignement entre deux séquences et nous voulons en calculer le score. Pour chacune de ses positions, nous calculons un score fondé sur deux critères. Le premier est la ressemblance physico-chimique des acides aminés rencontrés à cette position dans l'une et l'autre des séquences alignées (voir chapitre 4). Le deuxième critère est la prise en compte du contexte local de la position considérée. Pour traiter ce point, nous comparons les fréquences des mots de n lettres englobant les acides aminés situés à cette position dans les séquences.

Le score fondé sur ces deux critères attribue à un alignement un score d'autant plus élevé qu'il a une cohérence globale. Dans cette section nous détaillons la méthode de calcul du score de deux positions. Nous commençons par rappeler la définition de l'information mutuelle puis nous introduisons une nouvelle notion que nous appelons l'information contextuelle et qui mesure la dépendance entre une position et son contexte dans un alignement. Nous donnons ensuite l'expression du score.

Soit w un mot de Σ^*. On note $\mathsf{freq}\,(w)$ la *fréquence* de w. Le critère d'information mutuelle généralement utilisé pour mesurer la dépendance entre deux éléments x et y [Shannon et Weaver 1949] est définie comme suit.

Définition 5.5 (information mutuelle) L'*information mutuelle* entre deux éléments x et y est :

$$\mathsf{information\_mutuelle}\,(x,y) = \mathsf{freq}\,(x,y) \cdot \log\left(\frac{\mathsf{freq}\,(x,y)}{\mathsf{freq}\,(x)\cdot\mathsf{freq}\,(y)}\right)$$

□

Notons que si les éléments x et y sont indépendants, leur information mutuelle est nulle. Pour les séquences biologiques, l'information mutuelle est généralement calculée pour une paire de lettres. Par exemple, [Gorodkin *et al.* 1997] a introduit une notion d'information mutuelle entre deux positions d'un alignement de séquences de nucléotides prenant en compte la complémentarité des bases. Dans cette thèse, nous proposons de considérer les contextes gauche et droit d'une position, ce qui nous amène à introduire une nouvelle notion, proche de l'information mutuelle, mais non symétrique et que appelons *information contextuelle*.

Définition 5.6 (information contextuelle) L'*information contextuelle* entre deux mots w_1 et w_2 de Σ^* est :

$$\mathsf{information\_contextuelle}\,(w_1,w_2) = \mathsf{freq}\,(w_1w_2) \cdot \log\left(\frac{\mathsf{freq}\,(w_1w_2)}{\mathsf{freq}\,(w_1)\cdot\mathsf{freq}\,(w_2)}\right)$$

□

Notons que si c'est le contexte droit (c_{droit}) d'une lettre l qui nous intéresse, nous calculerons $\mathsf{information\_contextuelle}\,(l, c_{droit})$ et si c'est le contexte gauche (c_{gauche}), nous calculerons $\mathsf{information\_contextuelle}\,(c_{gauche}, l)$. Il est aussi important de remarquer que la dissymétrie de l'information contextuelle est due au fait que l'opération de concaténation n'est pas symétrique.

Nous pouvons maintenant donner l'expression du score d'une position dans un alignement de deux séquences. Le score d'une position dans un alignement est celui de la paire de positions dans la paire de séquences comparée. L'idée est de détecter le caractère exceptionnel de la dépendance entre une paire de positions dans une paire de séquences et ses contextes (gauche ou droit). La relation de borne supérieure entre les deux acides aminés a_1 et a_2 présents à une position i de l'alignement de deux séquences (voir définition 4.1 page 68), appliquée au treillis défini au chapitre 4, définit le plus petit groupe d'acides aminés contenant à la fois a_1 et a_2. Nous appelons ce groupe l'ensemble des acides aminés possiblement présents à la position i de l'alignement. Le *score gauche* (*resp. droit*) d'une paire de positions donnée est la somme normalisée, pour tous les acides aminés a possiblement présents à cette position et pour tous les contextes c gauche (*resp.* droit) de la paire de positions (voir Définition 5.4) de l'information contextuelle de a et de c. Le *score d'une paire de positions* donnée est le maximum entre les scores gauche et droit de cette paire. Le score est pondéré par la distance entre les acides aminés présents à ces positions dans l'une et l'autre des séquences. La distance entre les acides aminés est mesurée à l'aide de la relation de borne supérieure (voir définition 4.1 page 68) appliquée au treillis défini au chapitre 4. Formellement,

Définition 5.7 (score d'une paire de positions dans une paire de séquences) Soient S_1 et S_2 deux séquences, i (respectivement j) une position dans la séquence S_1 (respectivement S_2) et t un entier définissant la taille des contextes. Le score des positions i et j est :

$$\mathsf{position\_score}\,(S_1[i], S_2[j]) = \frac{1}{1+\log\left(|S_1[i] \sqcup S_2[j]|\right)} \cdot \max\Bigg(\frac{1}{\left|\mathsf{contexte_{gauche}}\,(S_1[i], S_2[j], t)\right|} \sum_{\substack{c \in \mathsf{contexte_{gauche}}(S_1[i],S_2[j],t),\\ l \in S_1[i] \sqcup S_2[j]}} \mathsf{information\_contextuelle}\,(c, l),$$

$$\frac{1}{\left|\mathsf{contexte_{droit}}\,(S_1[i], S_2[j], t)\right|} \sum_{\substack{c \in \mathsf{contexte_{droit}}(S_1[i],S_2[j],t),\\ l \in S_1[i] \sqcup S_2[j]}} \mathsf{information\_contextuelle}\,(l, c)\Bigg)$$

□

Le score normalisé d'un alignement est une fonction dépendant de la somme des scores de toutes ses positions (voir le premier paragraphe de la section 5.2.3)

et de sa taille. Formellement, le score normalisé d'un alignement de taille n est :

$$\mathsf{score}\,(A) = S \text{ si } S > \texttt{MinS} \text{ et } 0 \text{ sinon} \tag{5.1}$$

$$\text{où } S = \frac{1}{a \cdot n + b} \cdot \left(\sum_{\substack{i \in [1,n] \text{ tel que} \\ \mathsf{pos}(A,i,S_1) \neq 0, \\ \mathsf{pos}(A,i,S_2) \neq 0}} \mathsf{position\_score}\left(\mathsf{pos}\,(A,i,S_1), \mathsf{pos}\,(A,i,S_2)\right) \right)$$

et `MinS` est un seuil fixé. a et b sont deux constantes à déterminer en fonction de l'homogénéité des séquences de l'échantillon. En pratique, nous prenons `MinS` $= 4$, $a = 0.7$ et $b = 0.7$ qui sont des valeurs qui donnent de bon résultats dans nos expérimentations.

5.2.4 Compatibilité entre appariements de positions

Les données avec lesquelles nous travaillons sont très particulières. En effet, les protéines sont des enchaînements d'acides aminés qui possèdent des régions que l'on appelle sites actifs et dans lesquelles il n'y a pas de notion naturelle de répétition. Le modèle que nous générons, c'est à dire un automate de transitions finies, permet lui, d'introduire des répétitions de n'importe quel sous-chaîne d'une protéine. Ceci présente l'avantage de permettre la création de boucles d'attente entre les sites actifs des protéines mais a l'inconvénient potentiel d'autoriser la création de boucles à l'intérieur même des sites actifs. Dans le but d'interdire les boucles au sein des régions conservées des protéines, la méthode que nous proposons dans ce chapitre commence par calculer des alignements entre séquences suivant le score défini précédemment. Ceci nous permet d'identifier les sites actifs des protéines et ainsi, les zones qui ne doivent pas participer à une boucle. Dans cette section, nous commençons par définir formellement les propriétés du langage cible que nous recherchons pour respecter cette contrainte. Ceci nous amène à définir une notion de compatibilité entre appariements de positions qui transpose, au niveau d'un ensemble d'appariements, les restrictions définies au niveau des langages.

Un critère de choix de langages et d'automates adapté aux protéines. L'hypothèse habituelle pour l'inférence d'automates est celle de complétude structurelle de l'échantillon d'apprentissage par rapport à l'automate cible. Sur les automates d'états finis, la définition de la complétude structurelle est la suivante :

Définition 5.8 (complétude structurelle pour les FSA) Un échantillon $\mathcal{S}$ est structurellement complet par rapport à un FSA $\mathcal{A}$ s'il existe une acceptation de $\mathcal{S}$ par $\mathcal{A}$ telle que :

(1) toute transition de $\mathcal{A}$ est exercée,
(2) tout état initial de $\mathcal{A}$ est l'état initial d'au moins une acceptation et
(3) tout état final de $\mathcal{A}$ est utilisé comme état d'acceptation.

□

Sur les automates de transitions finies, la définition devient :

Définition 5.9 (complétude structurelle pour les FTA) Un échantillon $\mathcal{S}$ est structurellement complet par rapport à un FTA $\mathcal{A}$ s'il existe une acceptation de $\mathcal{S}$ par $\mathcal{A}$ telle que :

(1) toute transition de $\mathcal{A}$ est exercée,
(2) toute transition initiale de $\mathcal{A}$ est la transition initiale d'au moins une acceptation et
(3) toute transition finale de $\mathcal{A}$ est utilisée comme transition d'acceptation.

□

Nous posons ici une hypothèse supplémentaire qu'aucune acceptation d'une chaîne de l'échantillon d'apprentissage ne boucle. Autrement dit, les langages cibles L admissibles pour un échantillon $\mathcal{S}$ sont tels que, si le langage $B = L_1L_2{L_2}^+L_3$ est inclus dans L, alors aucun exemple de $\mathcal{S}$ n'appartient à B.

Notons que l'automate cible dont nous parlons ici n'est pas le résultat de notre méthode puisqu'il est clairement trop spécifique. C'est l'automate que nous appelons automate des similarité, qui contient les sites actifs et que nous généralisons dans un deuxième temps de façon à introduire des boucles d'attente entre les sites actifs.

Compatibilité entre appariements de positions Nous définissons maintenant la notion de compatibilité entre appariements qui traduit, à leur niveau, l'absence de sites actifs dans les boucles. L'idée est de dire que deux appariements sont compatibles s'ils n'associent pas une position d'une séquence avec plusieurs positions d'une autre séquence. En d'autres termes, ils doivent permettre de définir, pour chaque paire de séquences, une bijection entre l'ensemble des positions alignées.

Soit $\mathcal{A}$ un ensemble d'alignements. Cet ensemble d'alignements définit un ensemble d'appariement. L'*ensemble de séquences des appariements définis par* $\mathcal{A}$ est l'union des ensembles de séquences E tels que $\mathcal{A}$ contient un alignement sur E.

Nous définissons maintenant une notion binaire entre les lettres des séquences d'un ensemble d'alignements que nous appelons *relation d'appariement*.

Définition 5.10 (relation d'appariement) Soient $\mathcal{A}$ un ensemble d'alignements défini sur l'ensemble de séquences E, S_1 et S_2 deux séquences de E, i_1 une position dans S_1 et i_2 une position dans S_2. On dit que $S_1[i_1]$ et $S_2[i_2]$ sont en *relation d'appariement* par rapport à $\mathcal{A}$ (noté $S_1[i_1] \doteq_{\mathcal{A}} S_2[i_2]$) si et seulement s'il existe un alignement de S_1 et S_2 (appartenant à $\mathcal{A}$) tel que $S_1[i_1]$ est appariée à $S_2[i_2]$ ou s'il existe une séquence S_3 de E et une position i_3 dans S_3 telle que $S_1[i_1]$ est appariée à $S_3[i_3]$ et $S_3[i_3] \doteq_{\mathcal{A}} S_2[i_2]$. □

Cette relation est une relation d'équivalence.

Nous introduisons maintenant une nouvelle notion qui permet de comparer un ensemble de séquences.

Définition 5.11 (appariements compatibles) Soient $\mathcal{A}$ un ensemble d'alignements deux à deux sur un ensemble E de séquences. $\mathcal{A}$ défini un ensemble d'appariements sur E. Les appariements ainsi définis sont *compatibles* si et seulement si, pour toutes les paires de séquences $\{S_1, S_2\}$ de E de longueur respective n_1 et n_2, on a :

$$\forall i \in [1, n_1], \forall j, k \in [1, n_2], [(S_1[i] \doteq_{\mathcal{A}} S_2[j] \wedge S_1[i] \doteq_{\mathcal{A}} S_2[k]) \implies j = k]$$

□

Remarquons que la définition précédente ne porte pas sur des paires de séquences mais sur l'ensemble des séquences E car la relation d'appariement $\doteq$ est définie par rapport aux appariements engendrés par l'ensemble d'alignements $\mathcal{A}$.

5.3 Description de la méthode d'inférence par fusion de transitions

Dans cette section nous présentons une méthode d'inférence grammaticale régulière par fusion de transitions que nous appelons SDTM (*Similarity Driven Transition Merging*)[1]. Cette méthode était déjà schématiquement présentée sur la figure 5.1 page 75. La méthode SDTM (voir algorithme 1) prend en entrée un échantillon $\mathcal{S}$ de séquences d'une famille de protéines et un sous-échantillon $\mathcal{S}'$ de $\mathcal{S}$ (pouvant être produit aléatoirement et automatiquement). Elle génère en trois étapes l'automate de transitions finies modélisant les séquences de cette famille. Nous introduisons le sous-échantillon $\mathcal{S}'$ dans le but de réduire les opérations combinatoires complexes à un sous-ensemble d'apprentissage tout en conservant

[1] Fusion de transition dirigée par la similarité en français

Algorithme 1 : SDTM : inférence par fusion de transitions

Données : $\mathcal{S}$ - échantillon de séquences, $\mathcal{S}'$ - sous-échantillon de $\mathcal{S}$.
Résultat : $\mathcal{A}$ - automate de transitions finies.
début
  $Stat \leftarrow$ `CalculDesStatistiquesSurEchantillon(`$\mathcal{S}$`)`;
  $Align \leftarrow$ `CalculDesAlignements(`$\mathcal{S}'$, $Stat$`)`;
  $\mathcal{A}' \leftarrow$ `GenerationDuFTA(`$\mathcal{S}'$, $Align$`)`;
  $\mathcal{A} \leftarrow$ `Compactage(`$\mathcal{A}'$`)`;
  retourner $\mathcal{A}$;
fin

l'avantage de prendre en compte des connaissances sur un échantillon plus vaste par le biais de statistiques.

Pour calculer la complexité de cette méthode, nous utilisons les notations suivantes. La taille N de l'échantillon $\mathcal{S}$ est la somme des longueurs des séquences le composant. On note p la taille moyenne d'une séquence. $\mathcal{S}'$ est de taille n ($n < N$). On note m la taille de l'alphabet sur lequel sont construites les séquences. Dans notre étude l'alphabet est l'ensemble des acides aminés et m est donc égal à 20.

Dans le reste de la section, nous détaillons chaque étape de la méthode SDTM, et donnons à chaque fois sa complexité.

5.3.1 Statistiques

La première étape de la méthode SDTM calcule des fréquences associées à l'échantillon de séquences donné $\mathcal{S}$ et est décrite par l'algorithme 2 où `NbFeuilles` est un prédicat récursif qui calcule le nombre de feuilles d'un nœud de la manière suivante : le nombre de feuilles d'une feuille est 1 et celui d'un nœud est la somme des nombres de feuilles de ses fils. Si on considère des contextes de taille k, l'algorithme calcul la fréquence des lettres, des mots de $k+1$ lettres et des mots de k lettres (qui seront utilisés comme contextes). Les données statistiques sont utilisées lors de la comparaison des acides aminés.

Exemple 5.3 Considérons l'ensemble des séquences $\{abbb\$1, bab\$2\}$. À partir de cet ensemble, l'algorithme construit l'arbre des suffixes montré sur la figure 5.4. Ensuite, en un parcours récursif de l'arbre, il associe à chaque nœud le nombre de feuilles du sous-arbre issu de ce nœud. Par exemple, la valeur de l'attribut "nombre de feuilles" pour la racine de l'arbre est 7. □

La complexité au pire de cette étape est $O(N)$ et est indépendante de la taille des contextes.

Algorithme 2 : Statistiques

Données : $\mathcal{S}$ - échantillon de séquences.
Résultat : *Stat* - statistiques sur l'échantillon donné.
début
  `ConstruireArbreSuffixesGeneraliseDesSequences(`$\mathcal{S}$, *ArbSuf*`);`
  `NbFeuilles(`*racine(ArbSuf),ArbSuf*`);`
  retourner *ArbSuf*;
fin

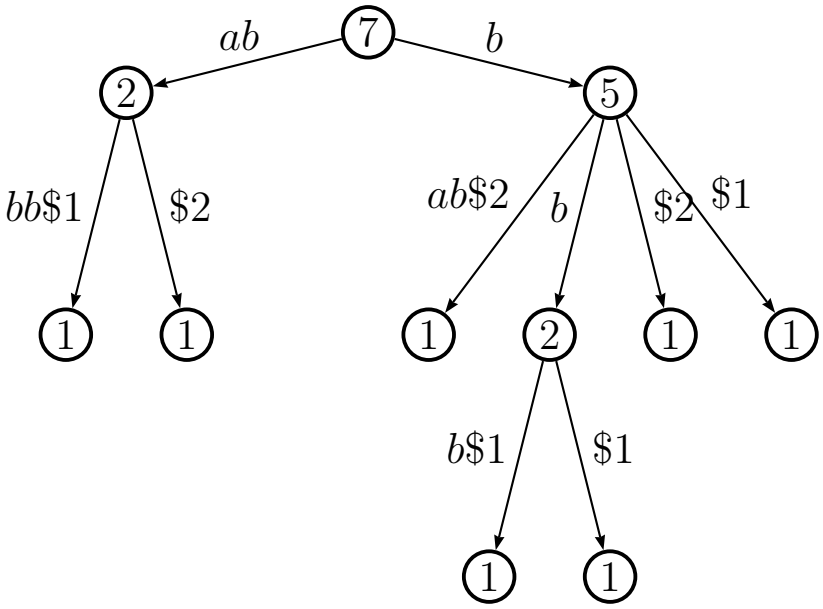


FIG. 5.4 – Arbre des suffixes compacté de l'ensemble des séquences $\{abbb\$1, bab\$2\}$.

5.3.2 Calcul des meilleures paires de transitions à fusionner par alignement local

Dans cette section, nous présentons la deuxième partie de notre méthode qui consiste à calculer le meilleur ensemble d'appariements compatibles sans trou pour les séquences de l'échantillon $\mathcal{S}$. Les différentes étapes de cette partie sont détaillées dans le reste de la section. Brièvement, la première étape construit les alignements locaux deux à deux sans trou à partir de l'ensemble des séquences. La seconde étape construit un graphe dont les sommets correspondent aux positions des séquences de $\mathcal{S}$ et les arcs aux appariements (définis par les alignements) de ces positions. La troisième étape recherche dans le graphe des ensembles de sommets et un arbre de recouvrement maximal pour chaque ensemble afin de proposer une approximation du meilleur ensemble d'appariements compatibles.

Méthode de génération des meilleurs alignements locaux sans trou de deux séquences. De nombreuses méthodes existent ([Thompson *et al.* 1994, Gambin *et al.* 2002]) pour extraire des alignements locaux d'un ensemble de séquences. Cependant, nous avons besoin d'une version restreinte de l'alignement dans notre contexte d'apprentissage. Les alignements que nous générons ne sont que le résultat intermédiaire de notre méthode d'inférence par fusion de transitions. Nous les utilisons pour construire automatiquement un automate qui décrit la famille de protéines que nous étudions. Les alignements doivent donc nous indiquer précisément quelles régions des séquences se ressemblent. Notre méthode génère des alignements locaux deux à deux qui prennent en compte les statistiques que nous avons calculé sur l'échantillon de séquences (voir section 5.3.1). Cette méthode est hybride entre celle des alignements multiples et celle des alignements deux à deux. En effet, elle présente les avantages d'un alignement multiple puisque toutes les séquences sont considérées pour le calcul du score, tout en autorisant un motif à n'être présent que dans un sous-ensemble des séquences de l'échantillon.

Le calcul des alignements locaux deux à deux sans trou est réalisé par programmation dynamique [Needleman et Wunsch 1970]. Nous donnons l'équation de récurrence associée. Nous dirons que deux acides aminés a et b sont proches si la différence de niveau entre leur borne supérieure dans le treillis (voir section 4.4) et le minimum des niveaux de a et de b est inférieure ou égale à un seuil fixé (3 en pratique).

Considérons deux séquences S_1, S_2 et deux positions respectives i, j dans ces séquences. La formule suivante défini $\mathsf{score}\,(i, j)$ qui est le score maximal que l'on peut obtenir en alignant un suffixe du préfixe de taille i de S_1 et un suffixe du

préfixe de taille j de S_2.

$$\mathsf{score}\,(i,j) = \begin{cases} \mathsf{score}\,(i-1,j-1) + \quad \mathsf{position\_score}\,(S_1[i], S_2[j]) \\ \text{si les acides aminés } S_1[i] \text{ et } S_2[j] \text{ sont proches} \\ 0 \text{ sinon} \end{cases} \tag{5.2}$$

où $\mathsf{position\_score}\,(S_1[i], S_2[j])$ est le score de la paire des positions i et j.

La complexité au pire de cette étape est $O(m \cdot p^2)$. En effet, pour une paire de séquences S_1 et S_2, nous parcourons la matrice de taille p^2 qui correspond à l'ensemble des paires d'une position de S_1 et d'une position de S_2. De plus, le calcul du score d'une paire de positions ($\mathsf{position\_score}$) a une complexité au pire de $O(m)$.

À cette étape de la méthode, nous avons calculé un ensemble d'alignements auxquels nous avons associé un score. Chaque alignement définit un ensemble d'appariements entre positions. Pour chaque alignement A de taille r et pour toutes les positions i de A, nous associons à l'appariement défini à cette position, un score qui correspond à la différence entre le score de l'alignement A et la somme des scores des sous-alignements $A[1, i-2]$ et $A[i+2, r]$ de A. Cette mesure nous indique le score de la réalisation de cet appariement et a l'avantage de considérer la position dans son contexte.

Construction du graphe des appariements. Nous décrivons ici la deuxième étape du calcul du meilleur ensemble d'appariements compatibles du sous-échantillon $\mathcal{S}'$. Cette étape construit un graphe des appariements. Les sommets du graphe correspondent aux différentes positions dans les séquences de l'échantillon $\mathcal{S}'$. Un arc correspond à un appariement entre deux positions et est valué par le score de l'appariement qu'il modélise. Remarquons qu'étant donné que nous générons les alignements par programmation dynamique, deux positions appariées participent à un seul alignement. Le graphe obtenu est un graphe k-partis, k étant le nombre de séquences, puisqu'on n'apparie jamais deux positions d'une même séquence.

Exemple 5.4 Cet exemple montre la construction du graphe des appariements sur l'échantillon $\{S_1, S_2, S_3\}$ avec $S_1 = ababc$, $S_2 = abc$, $S_3 = ab$ en considérant les alignements suivants : $\langle (a^1b^2)_1, (a^1b^2)_2 \rangle$, $\langle (a^3b^4c^5)_1, (a^1b^2c^3)_2 \rangle$, $\langle (a^1b^2)_1, (a^1b^2)_3 \rangle$, $\langle (a^3b^4)_1, (a^1b^2)_3 \rangle$ et $\langle (a^1b^2)_2, (a^1b^2)_3 \rangle$ où les chiffres en exposant correspondent à la position dans la séquence et les indices correspondent au numéro de la séquence concernée. □

Utilisation du graphe des appariements pour extraire les alignements compatibles. Le graphe des appariements contient les appariements définis par les alignements calculés précédemment, par programmation dynamique, sur

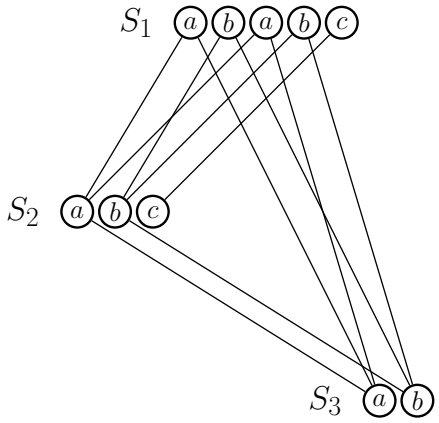


FIG. 5.5 – Une graphe des appariements.

l'échantillon $\mathcal{S}'$. Dans la section 5.2.4 page 81, nous avons défini la notion de compatibilité d'appariements qui nous permet de vérifier qu'aucune position dans une séquence n'est associée à deux positions distinctes d'une même autre séquence. Dans le graphe des appariements (noté G), cette notion de compatibilité peut s'exprimer aisément sous la forme d'arbres construit à partir du graphe G contenant au plus une position de chaque séquence. En effet, les associations de positions définies par un tel arbre (que nous appelons arbre d'appariements compatibles) forment un ensemble d'appariements compatibles.

Recherche des arbres d'appariements compatibles Notre but est de sélectionner le meilleur ensemble d'arbres d'appariements compatibles, dans le graphe des appariements G d'un ensemble de séquences $\mathcal{S}'$, dont le score est maximal. Ce problème d'optimisation est NP-difficile. Nous proposons une méthode gloutonne qui permet d'approximer une solution optimale avec une complexité acceptable.

La méthode (voir l'algorithme 3 et les sous-algorithmes 4 et 5) prend en entrée un échantillon de séquences E et le graphe des appariements associé à l'échantillon E. Il renvoie une approximation du meilleur ensemble d'appariements compatibles de G. Nous détaillons, dans ce paragraphe, les deux étapes de la méthode. La première consiste à trier les arcs de G suivant leur valuation puis à associer à chaque sommet du graphe la liste triée des arcs auxquels il participe. Cette étape a une complexité au pire de $O(m \log(m))$ si m est le nombre d'arcs du graphe. La deuxième étape construit itérativement les ensembles d'appariements compatibles recherchés. Un tel ensemble est un arbre sur un sous-ensemble de positions appartenant à des séquences différentes. L'idée est de calculer à chaque fois un arbre recouvrant maximal par rapport à ces positions.

La méthode commence avec un ensemble d'arbres d'appariements compatibles

Algorithme 3 : Approximation du meilleur ensemble d'appariements compatibles

Données : $E = \{S_1, S_2, \ldots, S_n\}$ - sous-échantillon de séquences,
$G = (S_G, A_G)$ - graphe des appariements pour l'échantillon E.
Résultat : A - approximation du meilleur ensemble d'appariements compatibles de G.

```
1  début
2      ArcTriés ← TriArcsParValuation(A_G);
3      ArcTriésParPosition ← DonneArcsTriéParPosition(ArcTriés);
4      A ← ∅;
5      tant que S_G ≠ ∅ faire
6          % MWST_max est une approximation du meilleur
7          % arbre construit à partir du graphe G (courant) et
8          % contenant au plus une position de chaque séquence.
9          numSeq ← 0;
10         score_max ← 0;
11         MWST_max ← ∅;
12         pour chaque i ∈ [1, n] faire
13             (MWST_c, score_c) ← MWSTInit(pos(S_i) ∩ S_G, (E \ {S_i}) ∩ S_G);
14             si score_c > score_max alors
15                 MWST_max ← MWST_c;
16                 score_max ← score_c;
17                 numSeq ← i;
18         sinon si MWST_max = ∅ alors
19             S_G ← ∅
20         A' ← ExtraitArcs(MWST_max);
21         S ← ExtraitSommets(MWST_max);
22         A ← A ∪ A';
23         S_G ← S_G \ S;
24     retourner A;
25 fin
```

Algorithme 4 : `MWSTInit` : approximation du meilleur arbre contenant la meilleure position d'un ensemble de positions et au plus une position de chacune des séquence d'un ensemble de séquences

Données : pos - ensemble de positions, $E = \{S_1, S_2, \ldots, S_n\}$ - ensemble de séquences.
Résultat : `MWST` - approximation du meilleur arbre construit à partir du graphe G et contenant exactement une position de pos (celle telle que l'arc de plus forte valuation la relie à un sommet de $pos(E)$) et au plus une position de chaque séquence de E, `score` - score de `MWST`.

début
  $\{p, p'\} \leftarrow \mathsf{argmax}_{i \in pos, j \in pos(E)}(\mathsf{score}\ (i, j))$;
  $(\texttt{MWST}, \texttt{score}) \leftarrow \texttt{MWST}(\{p, p'\}, E)$;
  retourner $(\texttt{MWST}, \texttt{score})$;
fin

vide et construit itérativement cet ensemble. À chaque fois qu'un arbre est rajouté à l'ensemble, les nœuds de l'arbre sont enlevés du graphe pour l'itération suivante. La méthode s'arrête quand plus aucun arbre d'appariements compatibles ne peut être généré.

Pour générer un arbre, la méthode choisit aléatoirement, dans le graphe courant et pour chacune des séquences S de $\mathcal{S}'$, un arc de valuation maximale et dont l'un des sommets appartient à S. Il construit ensuite une approximation de l'arbre des appariements compatibles qui contient cet arc. Ce calcul se fait par extensions successives de l'arbre qui contient initialement l'arc de valuation maximale calculé précédemment. Il s'agit d'une adaptation de l'algorithme de Kruskal [], qui garantit que l'on construit un arbre de recouvrement maximal pour l'ensemble des sommets sélectionnés. De façon détaillée, l'extension de l'arbre se fait par la recherche, pour tous les nœuds n de l'arbre, d'un arc de valuation maximale dont l'une des extrémités est n et l'autre correspond à une position p d'une séquence dont aucune des positions n'est dans l'arbre. Pour chacune de ces positions p, nous calculons la somme des valuations des arcs $(p,\ p')$ tels que p' est un nœud de l'arbre. La position p dont la somme des valuations est maximale est ajoutée à l'arbre avec la branche (p, n) de valuation maximale telle que n est un nœud de l'arbre. Lorsque plus aucune position ne peut être trouvée, l'arbre obtenu est renvoyé et constitue une approximation de l'arbre des appariements compatibles recherché.

La complexité au pire de la deuxième étape est $O(m \log(m) + n \cdot m \cdot k)$ si n est la taille du sous-échantillon $\mathcal{S}'$ de séquences (c'est-à-dire le nombre de sommets du graphe G), m est le nombre d'arcs du graphe et k est le nombre de séquences de $\mathcal{S}'$.

Algorithme 5 : `MWST` : approximation du meilleur arbre contenant toutes les positions d'un ensemble de positions et au plus une position de chacune des séquences d'un ensemble de séquences

Données : *pos* - ensemble de positions, $E = \{S_1, S_2, \ldots, S_n\}$ - ensemble de séquences, $G = (S_G, A_G)$ - graphe des appariements pour l'échantillon E.

Résultat : `MWST` - approximation du meilleur arbre construit à partir du graphe G et contenant toutes les positions de *pos* et au plus une position de chaque séquence de E, `score` - score de `MWST`.

```
1  début
2      MWST ← ∅;
3      score ← 0;
4      tant que E ≠ ∅ faire
5          arc_max ← ∅;
6          p_max ← ∅;
7          score_max ← 0;
8          pour chaque i ∈ pos faire
9              j_max = argmax_{j∈pos(E)}(score (i, j));
10             arc_c ← {(i, j_max)};
11             p_c ← {j_max};
12             score_c ← Σ_{l∈pos} score (l, j_max);
13             si score_c > score_max alors
14                 arc_max ← arc_c;
15                 p_max ← p_c;
16                 score_max ← score_c;
17         si p_max = ∅ alors
18             Break ;
19         MWST ← MWST ∪ arc_max;
20         score ← score + score_max;
21         pos ← pos ∪ p_max;
22         S ← DonneLaSequenceAssociée(p_max);
23         E ← E \ {S};
24     retourner (MWST, score);
25 fin
```

En effet, le coût du tri des arcs du graphe est $m \log(m)$ et au pire, nous générons $n/2$ arbres (avec deux sommets chacun). À chaque itération de la méthode, un arbre est généré pour chacune des k. séquences. De plus, la génération de l'arbre au un coût égal, au maximum, au parcours des listes (de taille m) d'arcs triés des sommets de l'arbre.

Conclusion Dans cette section, nous avons proposé une méthode gloutonne qui calcule une approximation du meilleur ensemble d'arbres d'appariements compatibles. Ceci nous donne par construction un ensemble d'appariements compatibles qui est une approximation du meilleur ensemble d'appariements compatibles. La qualité de l'approximation est la résultante de deux facteurs :

(1) un échantillonnage des arbres de recouvrement possibles par une boucle sur l'ensemble des séquences (voir la ligne 12 de l'algorithme 3),

(2) une évaluation du score des arbres qui prend en compte les cycles d'appariements (voir la ligne 12 de l'algorithme 5)

5.3.3 Inférence de l'automate par fusion de transitions

La dernière étape de la méthode consiste à construire (voir algorithme 6), à partir des alignements générés à l'étape précédente, un automate qui représente le langage de la famille des séquences protéiques que nous étudions. Dans cette section, nous présentons les différentes étapes de la génération de cet automate qui correspond au résultat de la méthode.

Algorithme 6 : Génération du FTA

Données : $\mathcal{S}'$ - sous-échantillon de séquences, *Align* - alignements.
Résultat : $\mathcal{A}$- automate de transitions finies.

```
1 début
2 |  MCTA ← GenerationDuMcta(Align);
3 |  A_sim ← CalculAutomateDesSimilarites(MCTA);
4 |  A ← CalculAutomateCompacté(A_sim);
5 |  retourner A;
6 fin
```

Génération de l'automate de transitions finies maximal canonique (MCTA). La première étape de la méthode consiste à construire le plus grand automate de transitions finies canonique non déterministe (MCTA) reconnaissant exactement le langage composé des séquences appartenant à l'échantillon. Intuitivement, le MCTA peut être vu comme un ensemble d'arbres ayant tous exactement une

feuille et correspondant chacun à un exemple (ou mot) de l'échantillon. Formellement :

Définition 5.12 (automate de transitions finies maximal canonique) Soit $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ l'échantillon d'apprentissage positif composé des exemples positifs $s_1, s_2, \ldots, s_n$ où pour tout $i \in [1, n]$, $s_i = l_{i1} l_{i2} \ldots l_{in}$. L'*automate de transitions finies maximal canonique* relatif à $\mathcal{S}$, noté MCTA, est l'automate de transitions finies (Q, Σ, T) tel que :

(1) Σ est l'alphabet sur lequel $\mathcal{S}$ est défini,

(2) $Q = \{q_{ij} \mid i \in [1, |\mathcal{S}|], j \in [0, |s_i|]\}$,

(3) $\forall i \in [1, |\mathcal{S}|], \forall j \in [0, |s_i| - 1]$.
$\left[\left(q_{ij}, l_{ij}, m, q_{ij+1}\right) \in T, (i \in m \iff j = 0), (f \in m \iff j = |s_i| - 1)\right]$.

□

Exemple 5.5 L'automate représenté figure 5.6 est l'automate de transitions finies maximal canonique relatif à l'échantillon $\mathcal{S} = \{aeg, ad, acdg\}$. □

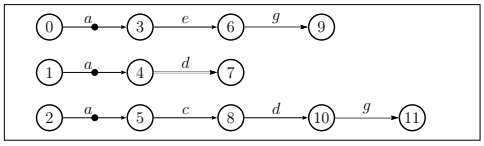


FIG. 5.6 – Automate de transitions finies maximal canonique MCTA($\mathcal{S}$), $\mathcal{S} = \{aeg, ad, acdg\}$.

Algorithme 7 : Fusion des appariements des meilleurs alignements compatibles

Données : BAC - meilleur ensemble d'alignements compatibles ; $\mathcal{A}$ - FTA.
début
  pour $i \leftarrow 1$ ***à*** $|BAC|$ **faire**
    pour $j \leftarrow 1$ ***à*** $|BAC[i]|$ **faire**
      $\{t_1, t_2\} \leftarrow$ `ObtenirTransitions(`$BAC[i, j]$`,`$\mathcal{A}$`)`;
      `FusionDesTransitions(`$\mathcal{A}$`,`t_1`,`t_2`)`;
fin

Fusion de transitions. Dans cette section, nous définissons les opérations de fusion d'états et de transitions sur les FTA et nous les comparons. La deuxième est l'opération centrale de la dernière étape de notre méthode SDTM. Contrairement à la fusion d'états, elle nous permet de prendre en compte explicitement l'information biologique car les éléments que l'on fusionne (les transitions) sont ceux qui portent l'information.

Définition 5.13 (FTA dérivé) Soient $\mathcal{A} = (Q, \Sigma, T)$ un FTA et π une partition sur Q. Alors, le *FTA dérivé* de $\mathcal{A}$ par la partition π, est $\mathcal{A}/\pi = (Q', \Sigma, T')$ où :

(1) $Q' = \{\mathsf{B}_\pi(q) \mid q \in Q\} = \pi$,

(2) T' est l'ensemble des transitions de la forme (B_1, l, m', B_2) telles que B_1 et B_2 appartiennent à Q', l est une lettre de Σ, il existe une marque m'' telle que (q_1, l, m'', q_2) appartient à T, et m' est l'union des marques m des transitions (q_1, l, m, q_2) de T telles que $q_1 \in B_1$, $q_2 \in B_2$, $m \in \{i, f\}$.

□

Définition 5.14 (opération de fusion d'états) Soient q_1 et q_2 deux états d'un FTA $\mathcal{A}$ donné. Alors, le résultat de l'application de l'*opération de fusion des états* q_1 et q_2 sur $\mathcal{A}$ est le FTA dérivé $\mathcal{A}/\pi$ où $\pi = \{\{q\} \mid q \in (Q \setminus \{q_1, q_2\})\} \cup \{\{q_1, q_2\}\}$. □

Définition 5.15 (opération de fusion de transitions) Soient $t_1 = (q_1, l_1, m_1, {q_1}')$ et $t_2 = (q_2, l_2, m_2, {q_2}')$ deux transitions d'un FTA $\mathcal{A}$ donné. Notons $q_{1,2}$ l'ensemble $\{q_1, q_2\}$ et ${q_{1,2}}'$ l'ensemble $\{{q_1}', {q_2}'\}$. Soit $\mathcal{A}/\pi = (Q', \Sigma, T')$ l'automate dérivé de $\mathcal{A}$ par la partition $\pi = \{\{q\} \mid q \in (Q \setminus \{q_1, q_2, {q_1}', {q_2}'\})\} \cup P$ où

$$P = \begin{cases} \{q_{1,2}, {q_{1,2}}'\} & \text{si } \forall q \in q_{1,2}, \forall q' \in {q_{1,2}}' \ [q \neq q'] \\ \{q_{1,2}\} = \{{q_{1,2}}'\} = \{\{q_1\}\} & \text{sinon} \end{cases}$$

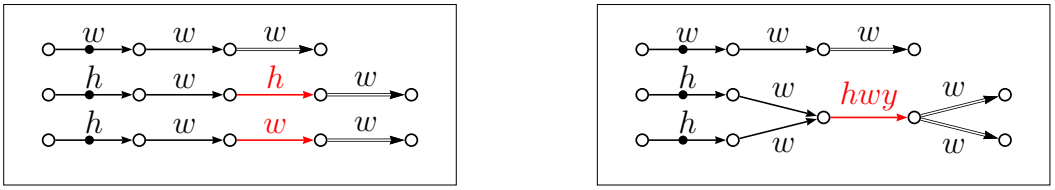
Alors, le résultat de l'application de l'*opération de fusion des transitions* t_1 et t_2 sur $\mathcal{A}$ est le FTA $\mathcal{A}'' = (Q', \Sigma, T'')$ où

$$T'' = T' \setminus \{(q_{1,2}, l_1, m_1, {q_{1,2}}'), (q_{1,2}, l_2, m_2, {q_{1,2}}')\} \cup \{t_{1,2}\}$$

avec $t_{1,2} = (q_{1,2}, (l_1 \sqcup l_2), (m_1 \cup m_2), {q_{1,2}}')$. □

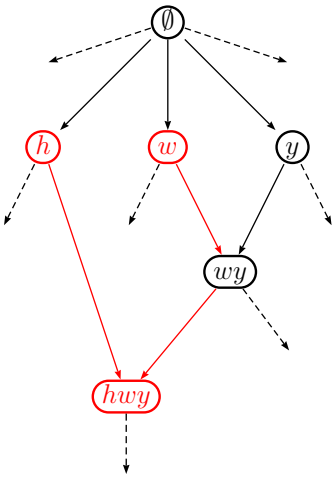
Exemple 5.6 La figure 5.7 de la page 95 montre un exemple de fusion de transitions.

□



(a) avant la fusion

(b) après la fusion



(c) portion du treillis basé sur le diagramme de Taylor

FIG. 5.7 – Exemple de fusion de transitions.

L'opération de fusion de transitions généralise plus le langage que l'opération de fusion d'états pour deux raisons. La première vient du fait que l'opération de fusion de transitions réalise deux fusions d'états. La deuxième est une conséquence de l'utilisation du diagramme de Taylor. En effet, la fusion de deux transitions portant les étiquettes l_1 et l_2 produit une transition dont l'étiquette n'est pas seulement l'union de l_1 et de l_2 mais la borne supérieure de l_1 et de l_2. Cette plus grande généralisation préserve la validité des choix de fusion du fait de l'ajout de l'information biologique dans l'opération.

La complexité de la génération du MCTA dépend de son nombre de transitions, c'est-à-dire du nombre et de la longueur des séquences de $\mathcal{S}'$, et est égale au pire à $O(n)$.

Calcul de l'automate des similarités. Le calcul de l'automate des similarités (voir ligne 3 de l'algorithme 6) consiste à générer, à partir des ressemblances locales des séquences de l'échantillon (voir section 5.3.2), et du MCTA, un automate plus général contenant les principaux motifs des séquences de l'échantillon.

Plus précisément, nous calculons l'automate des similarités par fusions de transitions sur le MCTA par rapport aux sous-alignements compatibles. En effet, pour chaque paire de séquences de l'échantillon $\mathcal{S}'$ donné, nous pouvons calculer un ensemble d'alignements compatibles (voir section 5.3.2). Cela définit un ensemble d'appariements entre les lettres des séquences de l'échantillon. Pour chaque appariement nous fusionnons, dans le MCTA, les transitions correspondant aux lettres appariées et obtenons ainsi l'automate des similarités. La figure 5.1(h) page 75 illustre ce calcul.

La complexité de cette étape est $O(\frac{n^2}{p})$. Elle correspond au parcours des alignements de l'ensemble des alignements compatibles position par position.

Calcul de l'automate compacté. La dernière étape de la méthode consiste à compacter l'automate obtenu à l'étape précédente. Pour toutes les transitions, de l'automate des similarités qui résultent de la fusion d'un faible nombre de transitions (fixé en pratique à $\frac{1}{4}$ de la taille de $\mathcal{S}'$) du MCTA, nous fusionnons ensemble leur état origine et leur état destination. La figure 5.1(j) page 75 illustre cette étape.

La complexité du calcul de l'automate compacté est la même que celle de la génération du MCTA.

5.3.4 Complexité générale de la méthode SDTM

Dans ce chapitre, nous avons présenté la méthode SDTM. Elle est divisée en trois étapes. La première consiste en le calcul de statistiques sur l'échantillon $\mathcal{S}$ de séquences et a une complexité de $O(N)$ (voir la section 5.3.1 page 84). La

deuxième étape, le cœur de la méthode, est la plus complexe. Le point critique de la méthode se trouve théoriquement au niveau du calcul de la compatibilité des appariements. La complexité de la deuxième étape de la méthode correspond à celle de ce point critique ($O(m\log(m) + n \cdot m \cdot k)$) si n est la taille du sous-échantillon $\mathcal{S}'$ de séquences, m est le nombre d'appariements et k est le nombre de séquences de $\mathcal{S}'$. La troisième partie de la méthode est l'étape de génération de l'automate résultat de notre méthode et a une complexité au pire de $O(\frac{n^2}{p})$.

En résumé, la complexité au pire totale de la méthode SDTM est $O(m\log(m)+ n \cdot m \cdot k)$ où n est la taille du sous-échantillon $\mathcal{S}'$ de séquences, m est le nombre d'appariements et k est le nombre de séquences de $\mathcal{S}'$.

Chapitre 6

Implémentation et résultats expérimentaux

Dans ce chapitre nous présentons les particularités de l'implémentation de la méthode SDTM qui a été réalisée en Prolog et les expérimentations que nous avons menées. Au début, nous proposons une représentation des automates de transitions finies non déterministes, particulièrement adaptée à la programmation logique et à la fusion de transition. Avec cette structure, la fusion de deux transitions correspond à leur unification. Nous terminons le chapitre par l'analyse des résultats de notre méthode et la comparaison avec ceux de Teiresias et de Pratt sur des données artificielles puis sur des données réelles.

6.1 Représentation d'un automate en programmation logique

L'utilisation de la programmation logique nous a amené à construire une représentation d'automates originale adaptée à la fusion de transitions et permettant de représenter les automates non déterministes. Nous commençons cette section par la présentation de deux implémentations différentes des automates d'états finis en programmation logique. La première ([van Noord 1997]) permet de représenter de façon simple les automates d'états finis non déterministes comme un ensemble de relation de transitions dont les arguments (états et lettres) sont constantes. La deuxième ([Nicolas 1999]) représente un automate par un terme infini et est adaptée à la fusion d'états. Nous proposons ensuite une nouvelle implémentation où un automate est représenté par la liste de ses transitions initiales et nous en expliquons les avantages.

6.1.1 Représentation d'un automate

Dans le rapport technique [van Noord 1997], G. van Noord propose une représentation pour les automates d'états finis non déterministes qui est simple et adaptée à la programmation logique. Un automate est alors représenté par un ensemble de clauses définies comme suit :

(1) `start(Etat).` – signifie que `Etat` est un état initial,

(2) `final(Etat).` – indique que `Etat` est un état final,

(3) `trans(Etat1, Etiq, Etat2).` – signifie qu'il existe une transition entre `Etat1` et `Etat2` étiquetée par `Etiq`,

(4) `jump(Etat1, Etat2).` – indique que les états `Etat1` et `Etat2` sont liés par une ε-transition (où une ε-transition est une transition étiquetée par le mot vide).

Exemple 6.1 Cet exemple est repris de [van Noord 1997] et illustre la représentation de G. van Noord. Les clauses suivantes décrivent l'automate d'états finis complet minimal déterministe représentant le langage des mots construits sur l'alphabet $\{a, b\}$ et contenant n'importe quel nombre de a mais pas de b (c'est-à-dire a^*).

```
start(0).      trans(0,a,0).      trans(0,b,1).
final(0).      trans(1,a,1).      trans(1,b,1).
```

□

Discussion. L'utilisation de cette représentation pour la fusion d'états nécessite, pour chaque fusion, la mise à jour de *tout* l'automate. Il est possible de proposer des représentations d'automates adaptées à la fusion d'états telles que l'opération d'unification met automatiquement à jour la structure. Deux de ces représentations sont décrites dans les sections suivantes. La première n'autorise que les automates déterministes tandis que la seconde (celle que nous proposons) autorise aussi les automates non déterministes.

6.1.2 Représentation d'automates adaptée à la fusion d'états

Dans le papier [Nicolas 1999], l'auteur propose de représenter un automate déterministe par un terme infini qui permet de parcourir l'ensemble des acceptations possibles. Pour simplifier la présentation, nous donnons une description de cette structure pour les automates complets. Elle est très facilement extensible aux automates incomplets. Chaque état d'un automate d'états finis complet est

représenté par un terme `state`(`Classe`, `EnsOrdonneDeSuccesseurs`) où `Classe` représente la classe de l'état (c'est-à-dire 1 pour les états d'acceptation et 0 pour les états de rejet) et `EnsOrdonneDeSuccesseurs` représente la liste des états accessibles par chacune des lettres de l'alphabet à partir de cet état. Remarquons que la liste des états successeurs d'un état a la taille de l'alphabet et est donné suivant l'ordre alphabétique des étiquettes des transitions correspondantes.

Exemple 6.2 Cet exemple illustre la représentation par terme infini sur l'automate d'états finis complet minimal déterministe décrivant le langage a^* sur l'alphabet $\{a, b\}$ de l'exemple 6.1 page 100. Dans ce cas, les variables des états vont prendre les valeurs suivantes :

$$\texttt{X} = \texttt{state}(\texttt{1}, [\texttt{X}, \texttt{Y}]) \qquad \texttt{Y} = \texttt{state}(\texttt{0}, [\texttt{Y}, \texttt{Y}])$$

Cela signifie que :

(1) l'état `X` est un état d'acceptation à partir duquel on peut rejoindre l'état `X` par une transition étiquetée par a et l'état `Y` par une transition étiquetée par b,

(2) l'état `Y` est un état de rejet tel que les transitions sortantes étiquetées par a et par b mènent à `Y`.

Le terme infini est ainsi de la forme :

$$\texttt{state}(\texttt{1}, [\texttt{state}(\texttt{1}, \ldots), \texttt{state}(\texttt{0}, [\texttt{state}(\texttt{0}, [\ldots, \ldots]), \texttt{state}(\texttt{0}, [\ldots, \ldots])])])$$

□

Discussion. L'avantage de cette représentation est que la fusion de deux états se fait par simple unification des deux termes associés aux états, sans nécessiter la création d'une nouvelle structure. Cette représentation rend ainsi la mise à jour de l'automate automatique. Une telle représentation n'est cependant adaptée ni aux automates non déterministes (car pour chaque état, la liste des états successeurs permet d'associer exactement un état à une lettre de l'alphabet), ni à l'utilisation des prédécesseurs d'un état (car on a un accès direct uniquement aux successeurs des états). Dans la section suivante, nous présentons une généralisation de cette représentation.

6.1.3 Notre représentation d'un automate

Nous proposons une représentation originale pour les automates de transitions finies (FTA) non déterministes (NFA) dans laquelle un automate est décrit par

la liste de ses transitions initiales. À chaque transition et à chaque état du FTA, nous associons une variable attribuée. L'attribut des transitions est `etiquette` (qui correspond à l'étiquette de la transition) et ceux des états sont `pred` et `succ` (qui correspondent respectivement aux ensembles de transitions qui précèdent et qui succèdent un état donné).

Les prédécesseurs (*resp.* les successeurs) d'un état q sont représentés par la liste des transitions dont l'état de destination (*resp.* d'origine) est q. L'étiquette d'une transition est représentée sous la forme de la liste des lettres portées par la transition. Plus précisément, chaque transition est représentée par un terme $\texttt{trans}(\texttt{T}, \texttt{Marque}, \texttt{X}, \texttt{Y})$ où `T` est la variable associée à la transition, `X` et `Y` sont les variables associées aux états origine et destination de la transition et `Marque` est la liste des marques de la transition.

Exemple 6.3 Considérons l'alphabet de deux lettres $\Sigma = \{a, b\}$ et le langage L défini comme l'ensemble des mots commençant par un (ou plusieurs a) suivi d'un a ou d'un b : $(a^+(a+b))$. Alors, l'automate de transitions finies non déterministe minimal de L est représenté par la liste $[\texttt{trans}(\texttt{T}_1, [i], \texttt{X}, \texttt{X})]$ où :

1. l'attribut `etiquette` de $\texttt{T}_1$ est $[a]$,
2. l'attribut `pred` de `X` est $[\texttt{trans}(\texttt{T}_1, [i], \texttt{X}, \texttt{X})]$,
3. l'attribut `succ` de `X` est $[\texttt{trans}(\texttt{T}_1, [i], \texttt{X}, \texttt{X}), \texttt{trans}(\texttt{T}_2, [f], \texttt{X}, \texttt{Y})]$, où :
   (a) l'attribut `etiquette` de $\texttt{T}_2$ est $[a, b]$,
   (b) l'attribut `pred` de `Y` est $[\texttt{trans}(\texttt{T}_2, [f], \texttt{X}, \texttt{Y})]$,
   (c) l'attribut `succ` de `Y` est $[\,]$.

□

L'utilisation des variables attribuées nécessite la redéfinition du processus d'unification. À chaque attribut, nous associons un protocole d'unification. L'unification de deux variables provoque celle de leurs attributs conformément au protocole associé. Par exemple, le résultat de l'unification de deux transitions dont les attributs `etiquette` sont e_1 et e_2 est une transition dont l'attribut étiquette est la borne supérieure de e_1 et de e_2, calculée à partir du treillis que nous avons défini (voir section 4.4 page 68).

Discussion. L'implémentation décrite dans cette section permet de représenter les automates de transitions finies déterministes et non déterministes et est particulièrement adaptée à la fusion de transitions. Notons qu'elle est tout aussi adaptée à la fusion d'états et qu'elle est très facilement adaptable à la représentation des automates d'états finis.

6.2 Expérimentations

Dans cette section, nous décrivons les expérimentations que nous avons effectuées sur notre méthode SDTM. Elles sont séparées en deux parties. Nous présentons d'abord celles réalisées sur des données artificielles, puis celles sur des données réelles. Nous avons comparé les résultats de notre méthode à ceux de Teiresias [Rigoutsos et Floratos 1998a;b] et ceux de Pratt [Jonassen *et al.* 1995, Jonassen 1997].

6.2.1 Préliminaires

Nous commençons par rappeler les indices utilisés en général pour comparer les résultats de plusieurs méthodes de prédiction/classification.

On dispose d'un ensemble de protéines (par exemple SwissProt version 45) dont on sait si elles appartiennent ou pas à une famille donnée (par exemple la famille des toxines de serpent). Les méthodes que nous testons (Teiresias, Pratt et notre méthode SDTM) produisent un modèle que nous utilisons pour tester l'appartenance d'une protéine à la famille. Si les protéines appartiennent à la famille, elles vont être soit correctement classées comme positives (fraction des vrais positifs, notée `VP`), soit incorrectement classées comme négatives (fraction des faux négatifs, notée `FN`). Si elles n'appartiennent pas à la famille, elles seront soit correctement classées comme négatives (fraction des vrai négatifs, notée `VN`), soit incorrectement classées comme positives (fraction des faux positifs, notée `FP`). Ces notions sont résumées dans le tableau 6.1.

	présent	**absent**
prédit positif	`VP`	`FP`
prédit négatif	`FN`	`VN`

TAB. 6.1 – Matrice de confusion

Nous rappelons les définitions de deux mesures statistiques (la *sensibilité* et la *corrélation*) couramment utilisées pour le test.

Définition 6.1 (Sensibilité) La *sensibilité* (ou rappel) est le taux de données testées réellement positives sur lesquelles le résultat du test est positif (c'est le taux de vrais positifs exprimé en terme de pourcentage). Elle est définie par la formule suivante :

$$\mathtt{S_n} = \frac{\mathtt{VP}}{\mathtt{VP} + \mathtt{FN}}$$

□

Définition 6.2 (Coefficient de corrélation) Le *coefficient de corrélation* (ou coefficient de Matthews) mesure la corrélation statistique entre la prédiction et la réalité. Il est défini par :

$$\mathtt{CC} = \frac{(\mathtt{VP} \cdot \mathtt{VN}) - (\mathtt{FN} \cdot \mathtt{FP})}{\sqrt{(\mathtt{VP} + \mathtt{FN})(\mathtt{VN} + \mathtt{FP})(\mathtt{VP} + \mathtt{FP})(\mathtt{VN} + \mathtt{FN})}}$$

□

Celui-ci mesure un compromis entre sensibilité et spécificité (taux de rappel et taux de pollution par des faux positifs).

6.2.2 Présentation des jeux de test

Données artificielles. Pour les données artificielles, on souhaite reconnaître les éléments d'un langage défini comme l'ensemble des mots de taille au plus 20 qui contiennent un motif π_1 ou un motif π_2), suivit d'un motif π_3. Pour tester ce type de motifs, on considère l'ensemble des motifs (sans joker) de taille cinq dont chaque position correspond à un noeud de notre treillis (voir page 68) contenant un ou deux acides aminés. Ensuite on choisit aléatoirement trois motifs de cet ensemble (π_1, π_2 et π_3) et on considère le langage L constitué des séquences de taille comprise entre 10 et 20 qui contiennent, dans l'ordre, le motif π_1 (ou π_2) et le motif π_3. On génère aléatoirement trois ensembles de 500 séquences : un échantillon d'apprentissage et deux ensembles de test (l'un constitué de séquences appartenant à L (appelé *ensemble de test positif*) et l'autre contenant des mots de L dans lesquels on a rajouté trois erreurs (appelé *ensemble de test négatif*)).

Le modèle d'erreurs que nous avons choisi pour la génération des échantillons et pour les tests de l'acceptation des séquences au langage décrit par le modèle inféré est celui dans lequel la seule erreur autorisée est la substitution d'une lettre par une autre.

Pour réaliser les tests sur les données artificielles, on construit dix jeux de données indépendants qui respectent le protocole de génération de données que nous venons de décrire.

Données biologiques. La famille des protéines sur laquelle nous avons effectué les tests est celle des toxines de serpent. Nous l'avons choisie en particulier à cause de sa difficulté. En effet, cette famille contient des motifs assez courts à base de cystéine et qui sont très courants dans les protéines. Cette famille correspond au motif Prosite [Bairoch 1992] "PS00272" :

$$\mathtt{G} - \mathtt{C} - \mathtt{x}(1,3) - \mathtt{C} - \mathtt{P} - \mathtt{x}(8,10) - \mathtt{C} - \mathtt{C} - \mathtt{x}(2) - [\mathtt{PDEN}]$$

et comprend 313 séquences.

Pour exécuter les tests, nous procédons de manière classique par validation croisée : nous avons divisé l'ensemble des séquences de toxines de serpent en 10 parties de même taille. Nous avons ensuite retiré pour chacun des $i \in [1, 10]$ la $i^{ème}$ partie de cet ensemble. Le reste de l'ensemble nous sert d'échantillon d'apprentissage et la partie que nous avons retiré nous sert d'ensemble de test positif. Dans tous les cas, l'ensemble des autres séquences de la version 45 de SwissProt (celles ne correspondant pas à des toxines de serpent) nous sert d'ensemble de test négatif.

6.2.3 Variation des paramètres du programme

Le programme SDTM a un ensemble de paramètres qui doivent être réglés précisément. Les réglages optimaux dépendent, dans certains cas, de l'échantillon considéré. Dans cette section, nous présentons une étude expérimentale relative aux paramètres du programme dont le but est de montrer leur influence sur les résultats. Nous présentons la variation des paramètres importants du programme et discutons les résultats obtenus dans le cas des données artificielles et biologiques. Lors de l'étude des variations d'un paramètre particulier, les autres paramètres sont fixés à leur valeur par défaut. Les valeurs par défaut sont trouvées expérimentalement et permettent au programme de produire un résultat optimal, c'est-à-dire celui qui donne le meilleur compromis entre la corrélation et la sensibilité.

Seuil indiquant le score minimal d'un alignement sélectionné. Ce paramètre correspond au seuil noté `MinS` et introduit à la fin de la section 5.2.3 page 78. Il permet de séparer les alignements utilisés pour réaliser les fusions de transitions dans l'automate de ceux qui seront rejetés. Plus ce seuil est élevé, moins on fusionne de paires de transitions.

La figure 6.1 illustre les résultats de la variation de `MinS`. Elle est composée de quatre sous-figures qui montrent la sensibilité et la corrélation du modèle inféré par l'algorithme dans le cas de données artificielles puis dans celui de données biologiques. L'axe vertical indique la sensibilité (ou la corrélation). L'axe horizontal détermine le nombre maximal d'erreurs de substitution autorisées lors de la reconnaissance d'une séquence par le modèle. Les courbes donnent les résultats pour une valeur donnée (indiquée en légende) du paramètre.

Les expériences sur la variation de `MinS` mettent en évidence que la valeur optimale du paramètre étudié est différente suivant le type de données. Ceci peut s'expliquer par la nature différente des séquences étudiées. En effet, dans le cas des données artificielles, les régions des séquences ne participant pas à un motif sont aléatoire. Au contraire, dans le cas des données biologiques, ces régions, qui

n'appartiennent pas *a priori* à un motif, peuvent se ressembler à cause du fait que les protéines sont créées par copie (inexacte) d'une protéine existante. Dans nos expériences, la valeur optimale du paramètre est 12 pour les données artificielles et 3 pour les données biologiques car elles donnent le meilleur rapport entre la corrélation et la sensibilité. Cette valeur doit donc être déterminée expérimentalement pour chaque échantillon étudié. Pour cela, une partie de l'échantillon doit être réservée pour tester le modèle et ne doit pas être utilisée pour le générer. L'algorithme devra être modifié de façon à rechercher automatiquement la valeur optimale de ce paramètre par rapport à la reconnaissance des séquences de l'échantillon réservé au test par le modèle construit à partir de l'échantillon réservé à l'apprentissage.

Il est néanmoins possible de prendre 4 comme valeur par défaut. Cela permet d'obtenir des résultats satisfaisants dans tous les cas.

Variation des paramètres de normalisation du score. À la fin de la section 5.2.3 page 78 nous avons introduit deux paramètres a et b utilisés pour normaliser le score.

Les courbes des figures 6.2 et 6.3 montrent que si l'on prend pour a et b la valeur $0,7$ par défaut, les résultats sont en général bons. Cette valeur donne une corrélation et sensibilité quasi optimale dans tous les cas (sauf la corrélation pour le paramètre b dans le cas des données artificielles).

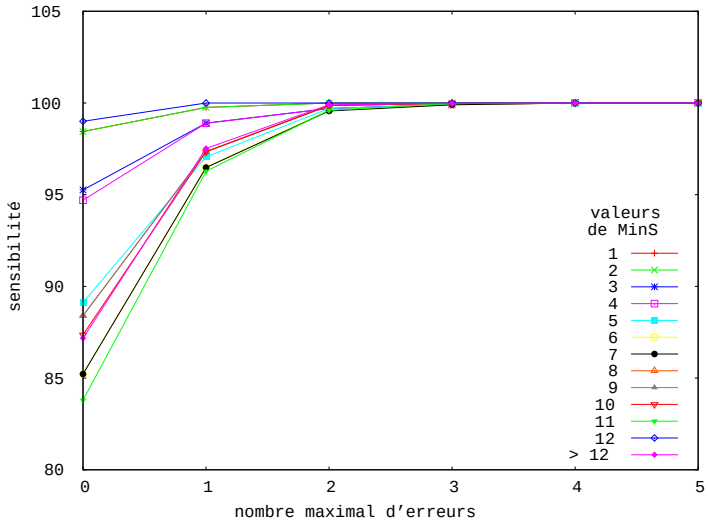
Seuil de compactage. Ce paramètre est un seuil qui permet d'identifier les transitions de l'automate des similarités (voir sa construction à la fin de la section 5.3.3) qui résultent de la fusion d'un faible nombre de transitions du MCTA. Plus le seuil est élevé, moins on généralise l'automate.

Les courbes de la figure 6.4 montrent que les valeurs 2 et 3 donnent les meilleurs résultats, autant pour les données artificielles que pour les données biologiques. Nous fixons par défaut ce seuil à 3 qui est légèrement meilleur que 2.

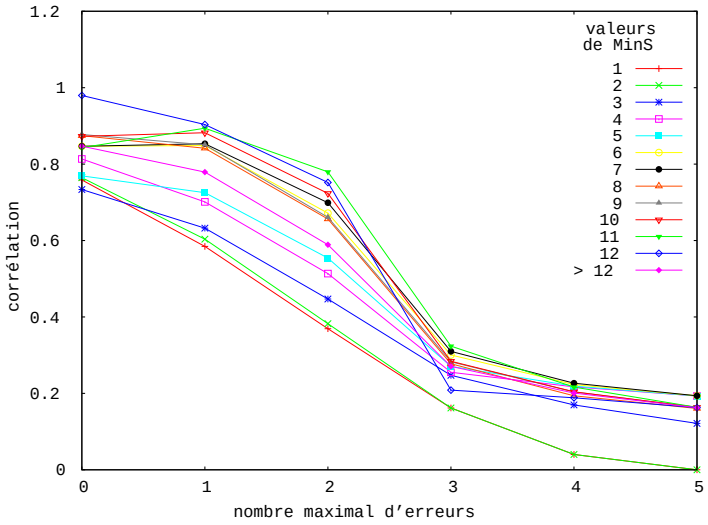
6.2.4 Comparaison avec d'autres méthodes

Nous comparons notre méthode SDTM, en utilisant la valeur par défaut des paramètres, aux méthodes de découverte de motifs Teiresias et Pratt. Le choix de ces méthodes à été guidé par les deux raisons suivantes : (1) à notre connaissance il n'existe pas de méthode d'inférence grammaticale suffisamment bonne qui génère, uniquement à partir de séquences d'une famille de protéines, un modèle de cette famille et (2) les méthodes de découverte de motifs que nous avons choisis sont les plus performantes sur les protéines.

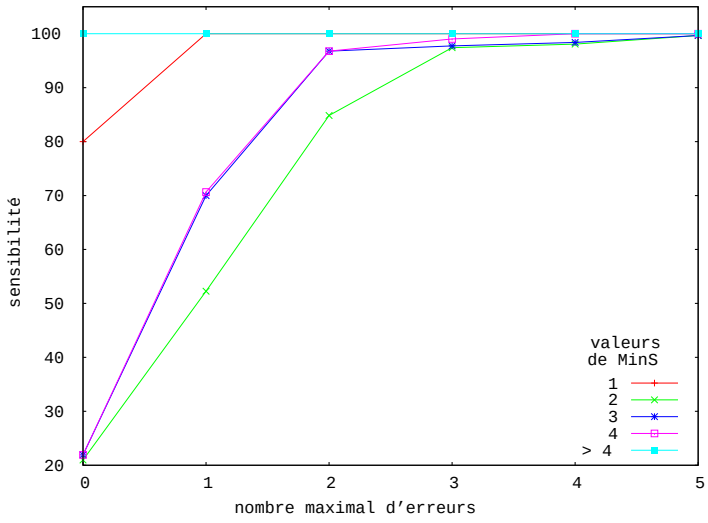
Nous avons exécuté ces trois méthodes sur les données artificielles et biologiques et nous avons obtenu les résultats résumés sur les figures 6.5, 6.6, 6.7 et



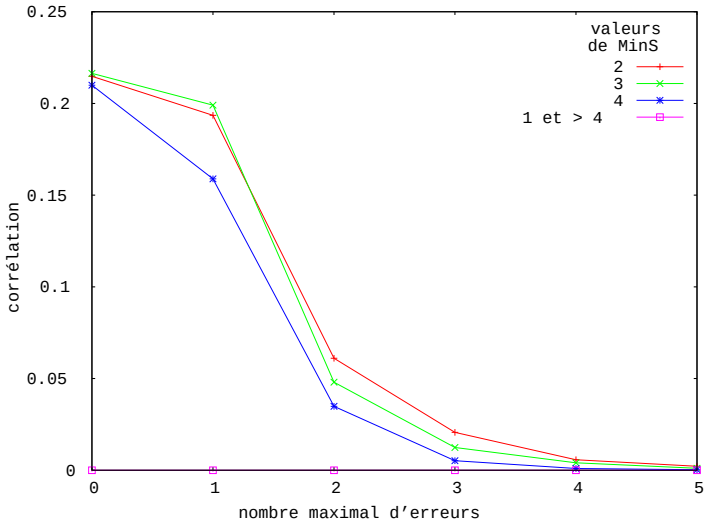
(a) Sensibilité pour données artificielles



(b) Corrélation pour données artificielles

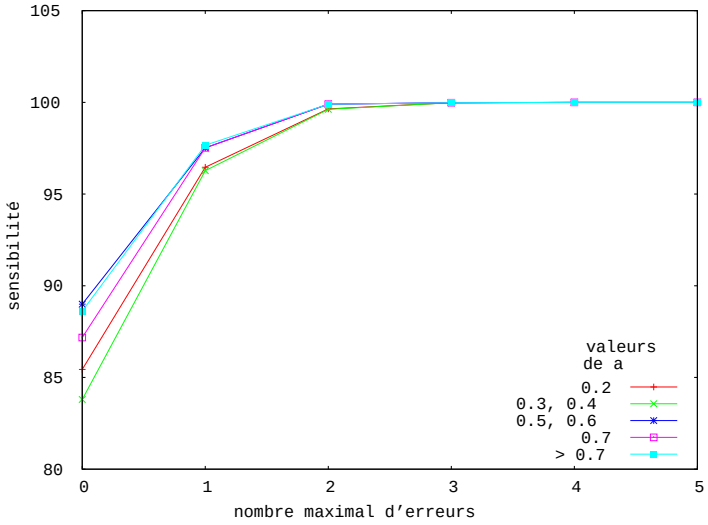


(c) Sensibilité pour données biologiques

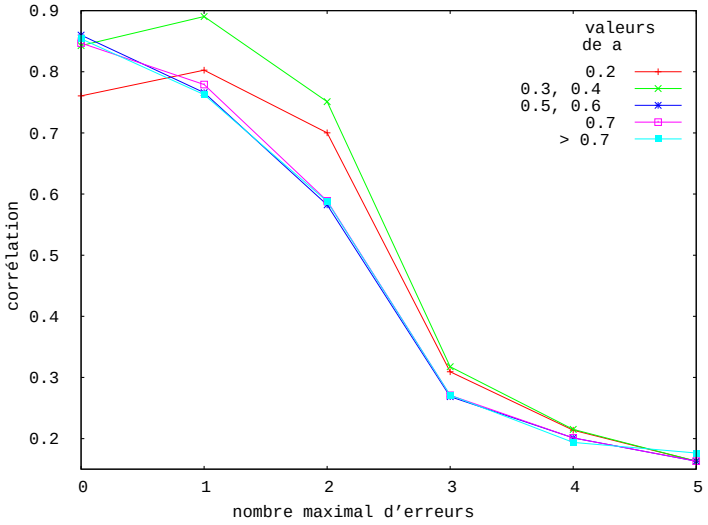


(d) Corrélation pour données biologiques

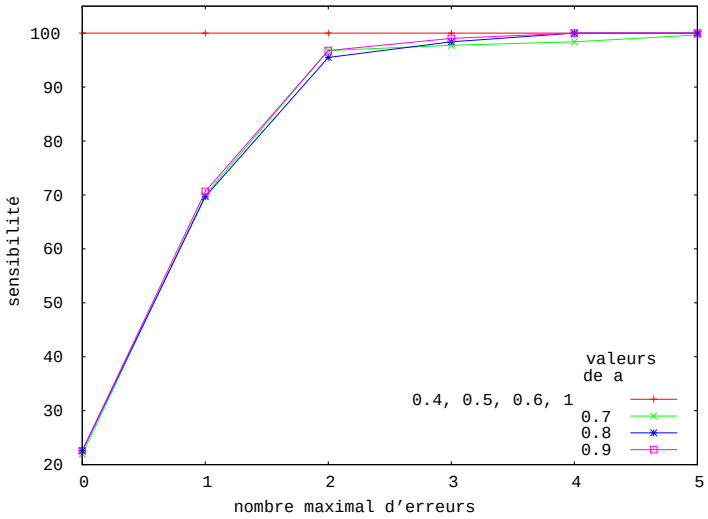
FIG. 6.1 – Variation du paramètre `MinS` (seuil de score minimal pour accepter un alignement).



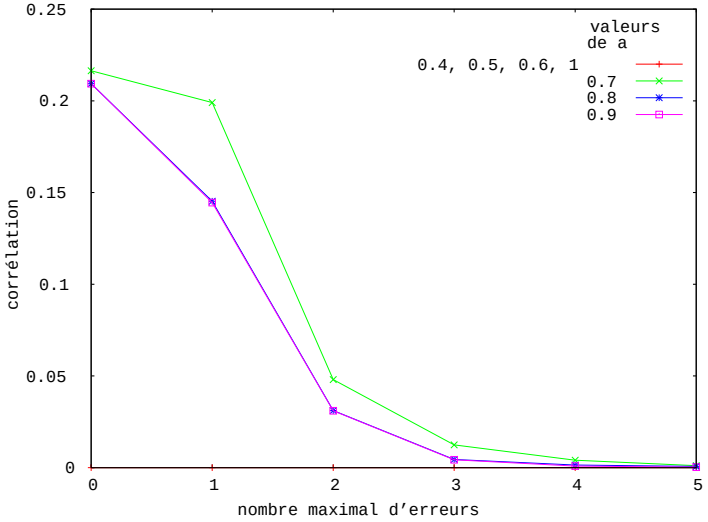
(a) Sensibilité pour données artificielles



(b) Corrélation pour données artificielles

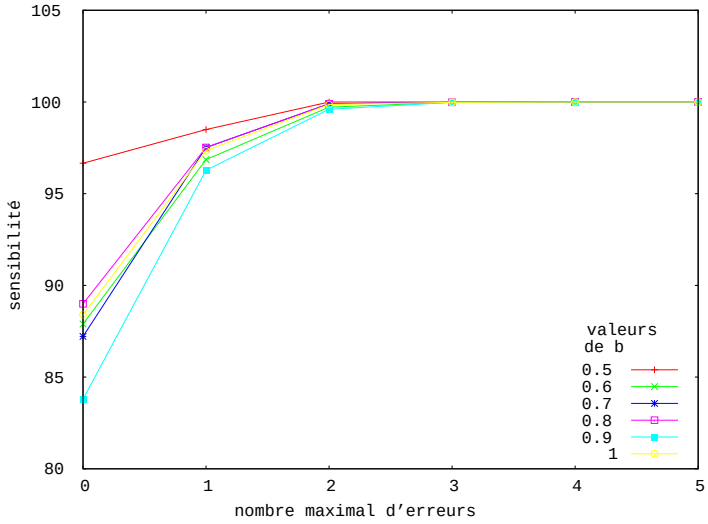


(c) Sensibilité pour données biologiques

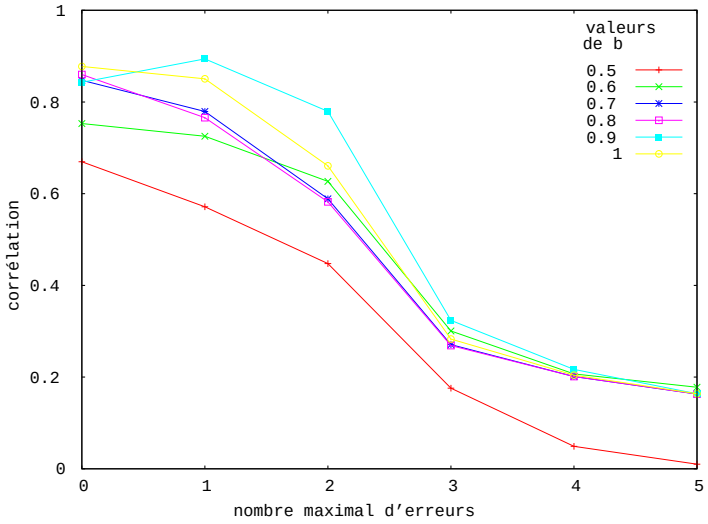


(d) Corrélation pour données biologiques

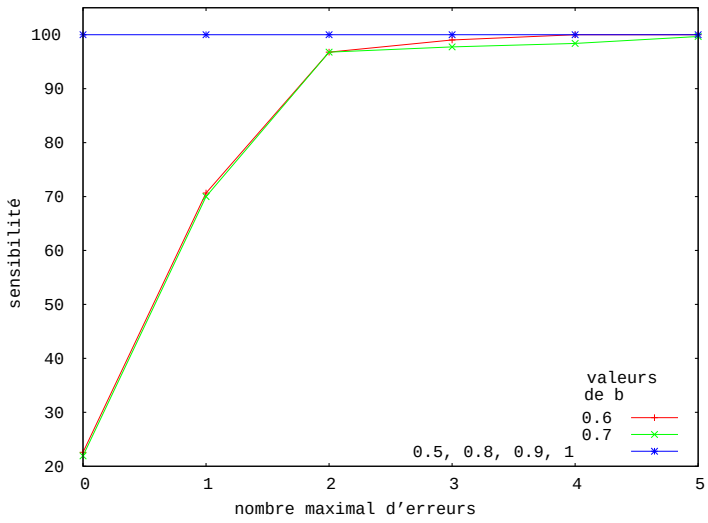
FIG. 6.2 – Variation du paramètre a de normalisation du score.



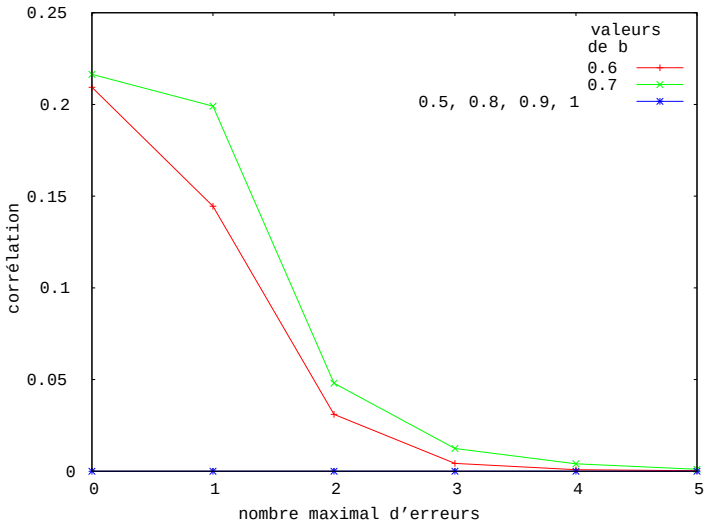
(a) Sensibilité pour données artificielles



(b) Corrélation pour données artificielles

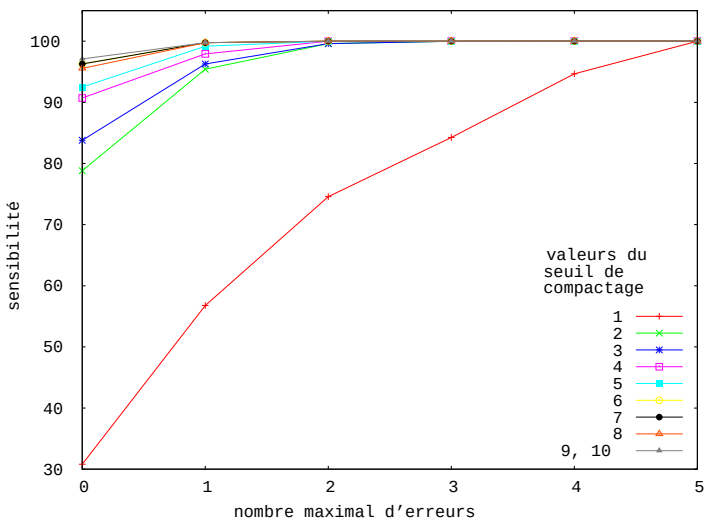


(c) Sensibilité pour données biologiques

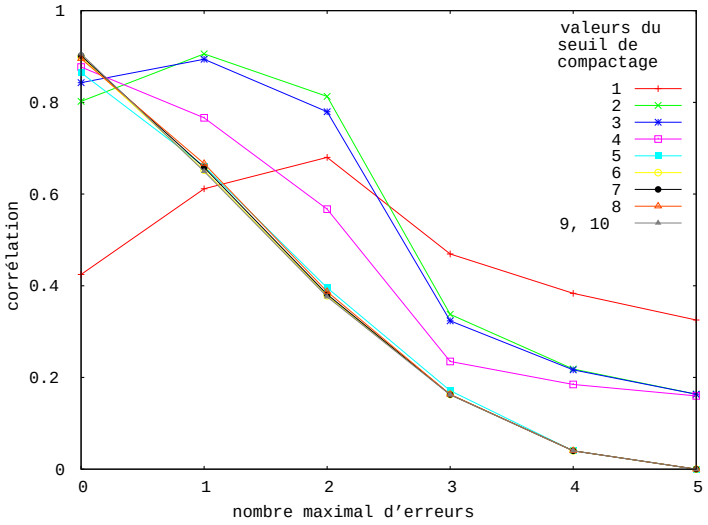


(d) Corrélation pour données biologiques

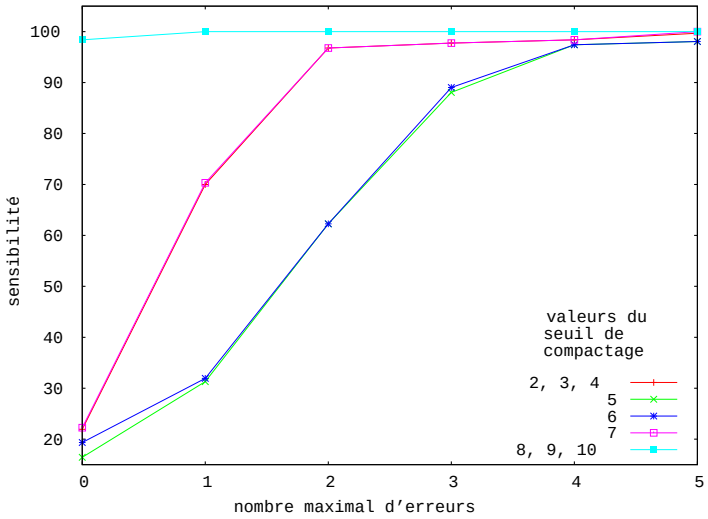
FIG. 6.3 – Variation du paramètre b de normalisation du score.



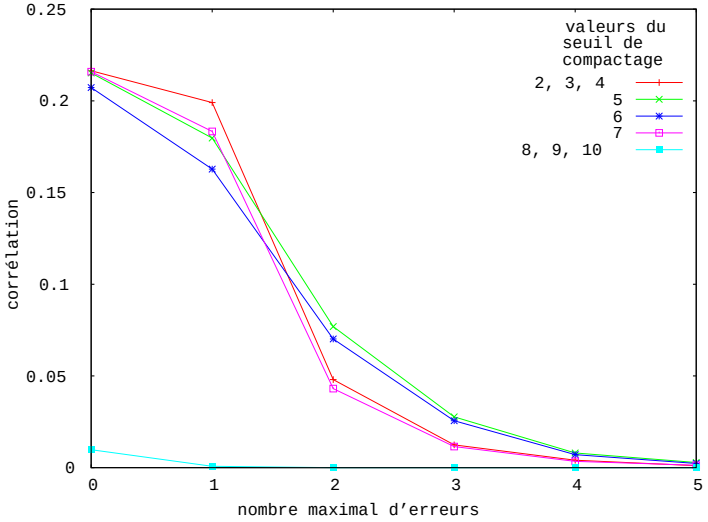
(a) Sensibilité pour données artificielles



(b) Corrélation pour données artificielles



(c) Sensibilité pour données biologiques



(d) Corrélation pour données biologiques

FIG. 6.4 – Variation du seuil de compactage

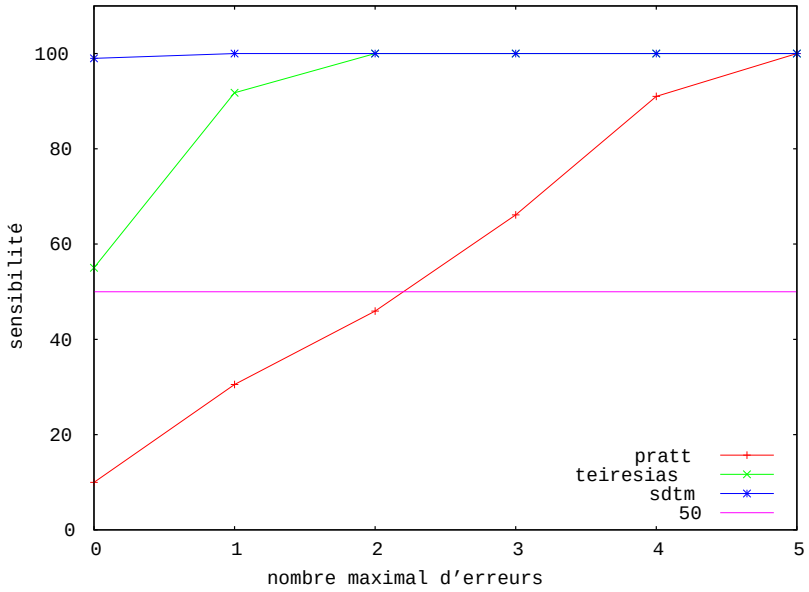


FIG. 6.5 – Sensibilité pour données artificielles

6.8. Les points des courbes correspondent aux valeurs de sensibilité (ou de corrélation) des méthodes pour un nombre d'erreur maximal fixé (c'est-à-dire que les séquences sont considérées acceptées si elles ont un nombre d'erreurs inférieur ou égal au nombre d'erreur maximal). Pour Teiresias et Pratt, chaque point des courbes correspond au meilleur résultat parmi ceux obtenus pour le premier et le deuxième motif produits par les méthodes. Pour les trois méthodes nous avons pris la moyenne des résultats obtenus pour tous les tests effectués sur les données artificielles (respectivement biologiques).

Les résultats des tests ont montré que, sur les données artificielles, notre méthode donne de bien meilleurs résultats que Teiresias et Pratt, tant du point de vue de la sensibilité que de la corrélation. Nous expliquons ceci par le fait qu'un langage décrit beaucoup plus finement un ensemble de mots qu'un simple motif. Ceci montre l'importance de la recherche de nouvelles représentations pour les méthodes de découverte de motifs. Pour les données biologiques, nous avons rajouté la courbe correspondant au motif Prosite des toxines de serpent (généré manuellement). Les expérimentations placent naturellement Prosite devant les trois méthodes comparés. Les courbes montrent que, sur les données réelles, Teiresias donne de bon résultats mais le modèle produit a en général une faible corrélation. Pratt donne aussi de bon résultats mais avec une sensibilité qui reste faible. Par contre, notre méthode propose un compromis entre une bonne sensibilité et une bonne corrélation.

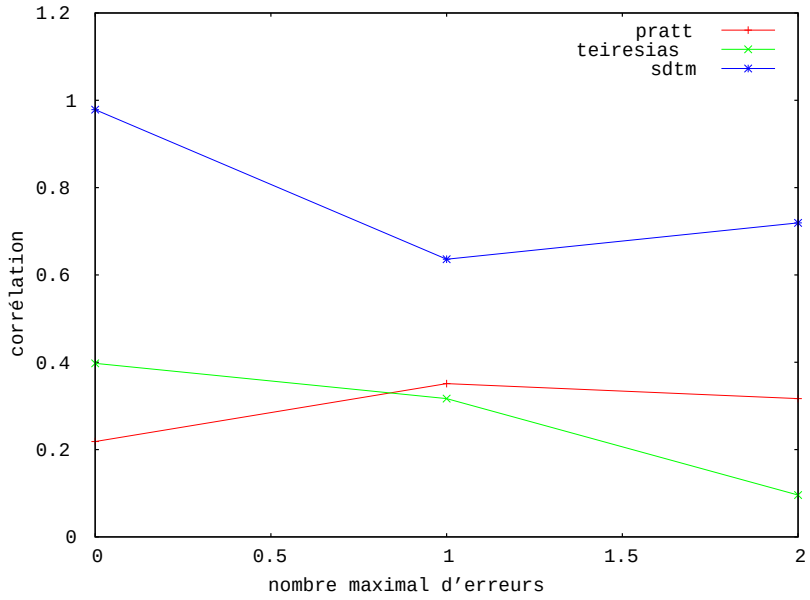


FIG. 6.6 – Corrélation pour données artificielles

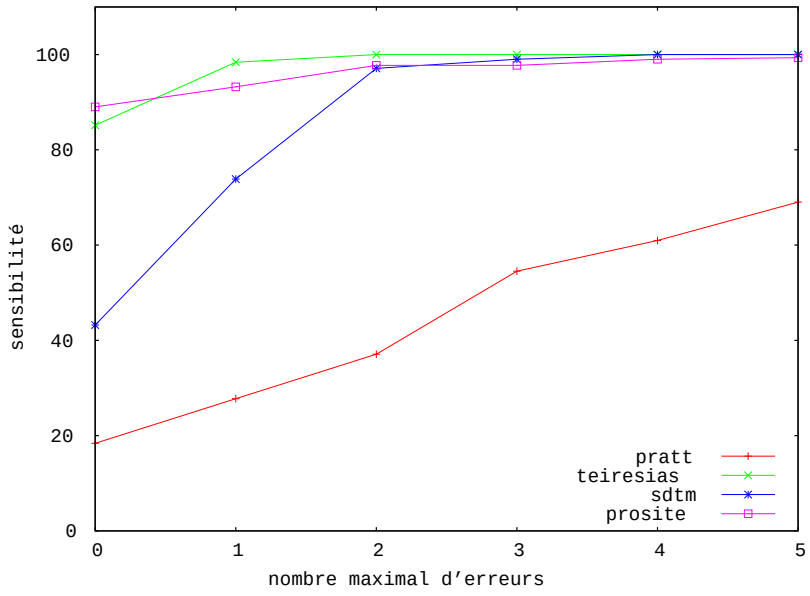


FIG. 6.7 – Sensibilité pour données biologiques

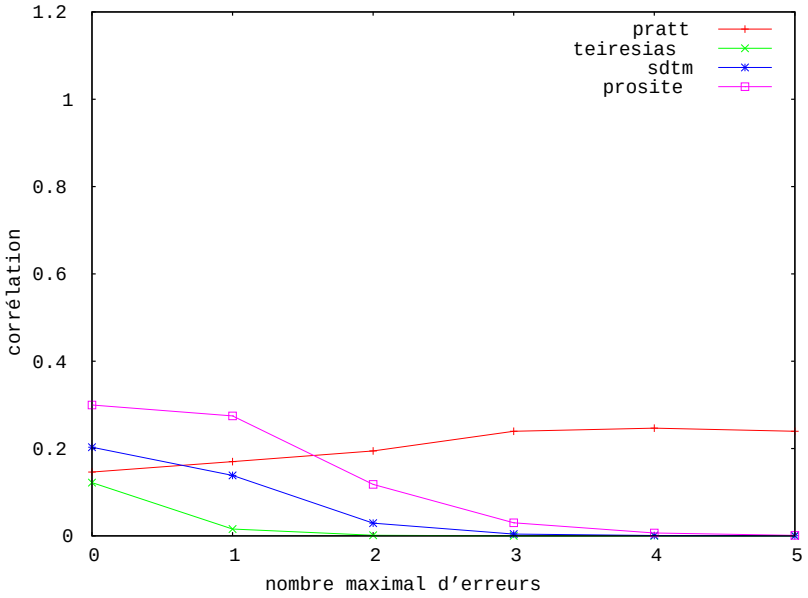


FIG. 6.8 – Corrélation pour données biologiques

Discussion. Il ressort des expérimentations que, sur la famille de protéines que nous avons testée et qui est particulièrement difficile, aucune des méthodes ne donne un résultat satisfaisant sur les données biologiques, même Prosite (qui est le meilleur motif) n'a jamais une corrélation très élevée. À noter qu'en général Prosite donne de bien meilleures corrélations. Ceci peut s'expliquer par la complexité des raisons de l'appartenance d'une protéine à une famille. En effet, d'un côté, beaucoup de motifs ne sont présents que dans une sous-famille des séquences d'une famille de protéines. D'un autre côté, un motif peut être présent dans des protéines n'appartenant pas à la même famille. La raison de l'appartenance d'une protéine à une famille est en général la présence conjointe d'un ensemble de motifs (c'est-à-dire par exemple, la présence d'un motif A ou d'un motif B, suivi des motifs C, D ou E).

Les méthodes qui existent ont tendance à n'identifier qu'une sous-partie de l'ensemble des motifs définissant l'appartenance des protéines à la famille. Ceci a deux conséquences. D'un côté, la sous-partie identifiée de l'ensemble des motifs va être trop générale pour ne reconnaître que les protéines de la famille et d'un autre côté, elle sera trop spécifique pour les reconnaître toutes.

On peut diviser le problème de la génération d'un modèle pour une famille de protéines en deux sous-problèmes. Le premier est l'identification des motifs élémentaires présents dans les protéines. Le deuxième est la combinaison de ces motifs. Les trois méthodes traitent (plus ou moins bien) le premier problème. En

revanche, seule la méthode SDTM traite le second problème.

Nous pensons que les méthodes Teiresias et Pratt ne donnent pas de résultats pleinement satisfaisants parce qu'elles ne traitent pas le problème de la combinaison des motifs. Notre méthode SDTM ne produit pas non plus de résultats pleinement satisfaisants parce qu'elle découvre les motifs élémentaires de manière encore limitée (seuls ceux présents dans un grand nombre de séquences sont identifiés). Nous pensons qu'il faut continuer comme nous l'avons initié à combiner les méthodes de découverte de motifs avec les méthodes d'inférence grammaticale en complexifiant les calculs de similarité sur les transitions. Ceci permettrait de générer des modèles de forte expressivité (ce que ne permet pas la découverte de motifs) et d'une grande précision (ce que ne permet pas l'inférence grammaticale pour les séquences biologiques). Une des solutions consiste à remplacer la première partie (génération des alignements sans trou) de notre méthode SDTM par une (ou plusieurs) méthode(s) de découverte de motifs qui identifie(nt) précisément la position des motifs dans les alignements et d'utiliser l'inférence grammaticale uniquement pour sélectionner et organiser les motifs découverts afin de générer un automate décrivant le langage.

Conclusion et perspectives

Nos travaux, motivés par la recherche de signatures pour les familles de protéines, nous ont menés à développer une technique pour mesurer la ressemblance des protéines entre elles. La mesure de cette ressemblance passe par celle des composants des protéines, c'est-à-dire les acides aminés. Dans la thèse, nous avons introduit une nouvelle façon de représenter et de mesurer la ressemblance entre les acides aminés. Pour cela, nous avons proposé un treillis fondé sur le diagramme de Taylor [Taylor 1986] et qui constitue un ordre partiel sur les groupes d'acides aminés. À la différence des autres treillis, celui que nous proposons contient un grand nombre d'informations sur les propriétés physico-chimiques des acides aminés. Son utilisation permet de ne pas restreindre *a priori* les propriétés physico-chimiques utiles à l'étude des protéines. Ceci est essentiel car les propriétés importantes d'un acide aminé dépendent de la région dans laquelle il se situe au sein de la protéine et demandent à être identifiées pendant l'apprentissage.

Nous avons proposé une nouvelle méthode d'inférence grammaticale (SDTM) fondée sur les notions fondamentales des méthodes de découverte de motifs ainsi que sur la ressemblance entre les acides aminés, calculée à travers le treillis que nous avons introduit. Cette méthode fonctionne sur le même principe que celles par fusion d'états à la différence qu'ici ce sont les transitions qui sont fusionnées. Elle produit un automate de transitions finies non déterministe qui peut ensuite être transformé dans une autre forme d'automate. Notre méthode présente l'originalité de calculer le score des alignements sur un ensemble d'apprentissage complet et de réaliser l'inférence sur un échantillon réduit. Cette technique permet de réduire le nombre de séquences comparées, et donc de diminuer le temps d'exécution d'un méthode coûteuse tout en considérant l'échantillon entier au travers de données statistiques. Pour faciliter l'utilisation de notre méthode, nous avons introduit un nouveau type d'automates très proche des machines de Mealy que nous avons appelé automates de transitions finies (FTA). Ces machines sont légèrement plus compactes que les automates d'états finis (FSA) et ont le même pouvoir d'expression.

Nous avons mené une étude de cas pour notre méthode, d'une part sur des données artificielles et d'autre part sur des données biologiques. Une première comparaison de notre méthode avec d'autres sur les données artificielles et sur les

données réelles a montré la faisabilité des idées développées dans cette thèse pour l'étude des protéines. Sur les données artificielles que nous avons générées pour simuler des séquences de protéines, notre méthode donne de bon résultats, meilleurs que les autres méthodes. Sur les données réelles, beaucoup moins homogènes que les séquences artificielles, les résultats de notre méthode sont équivalents à ceux des méthodes auquelles nous nous sommes comparés. Ces résultats demandent à être confirmés par des expérimentations de plus grande échelle. Néanmoins, notre approche (du type inférence grammaticale) est différente de celle des deux autres méthodes (du type découverte de motifs). Cette différence nous permet d'envisager l'utilisation conjointe de notre méthode et d'une méthode de découverte de motifs pour générer un modèle suffisamment expressif et précis et possèdant à la fois une forte sensibilité et une forte corrélation.

Nous présentons maintenant brièvement quelques perspectives de nos recherches.

Calcul de score dynamique. Le score de notre méthode est fondé sur des données statistiques issues de l'échantillon de séquences donné. Une étude intéressante concernerait l'influence d'un calcul dynamique de ces statistiques sur la qualité des resultats. L'idée serait alors de recalculer les statistiques à chaque étape de fusion des transitions, en ne comptant qu'une seule fois un mot présent dans deux régions différentes qui auraient été alignées lors d'une itération précédente de la méthode. Ainsi, cela permettrait de masquer les régions les plus similaires et de mettre en évidence des régions moins similaires mais néanmoins intéressantes. Nous pensons qu'un score dynamique nous permettrait de faire apparaître des régions faiblement conservées cachées par les régions les plus conservées.

Limiter la sur-généralisation. Beaucoup de méthodes d'inférence utilisent des exemples négatifs pour limiter la sur-généralisation du modèle ([Fredouille 2003, Lemay 2002]). Certaines autres études limitent la généralisation du modèle par le fait que la classe de langages est une sous-classe du modèle utilisé pour représenter les données et cherchent donc le plus petit modèle appartenant à la classe recherchée ([Angluin 1982]). Deux voies de recherche intéressantes peuvent alors être suivies : l'utilisation de méthodes alternatives pour limiter la sur-généralisation et l'utilisation conjointe de différentes méthodes pour limiter la sur-généralisation.

Utiliser des méthodes de localisation de motifs. Dans le premier chapitre, nous avons présenté des méthodes de découverte de motifs. Certaines autres méthodes permettent de découvrir et de localiser des motifs dans les séquences et donnent souvent en résultat un alignement de séquences (par exemple MODEL [Hernandez *et al.* 2004] ou TOPMODEL (Mescam *et al 2005, soumis à BMC*)). Nous ne les avons pas exploitées mais nous pensons qu'ils peuvent être

utilisés avantageusement pour améliorer notre méthode. L'avenir est dans l'étude de méthodes mixtes combinant des aspects de découverte locale et des méthodes plus globales du type inférence grammaticale. Ceci permettrait d'améliorer considérablement la complexité de notre méthode et poserait des problèmes intéressants de sélection et d'ordonnancement de motifs.

Annexe A

Treillis Élagué

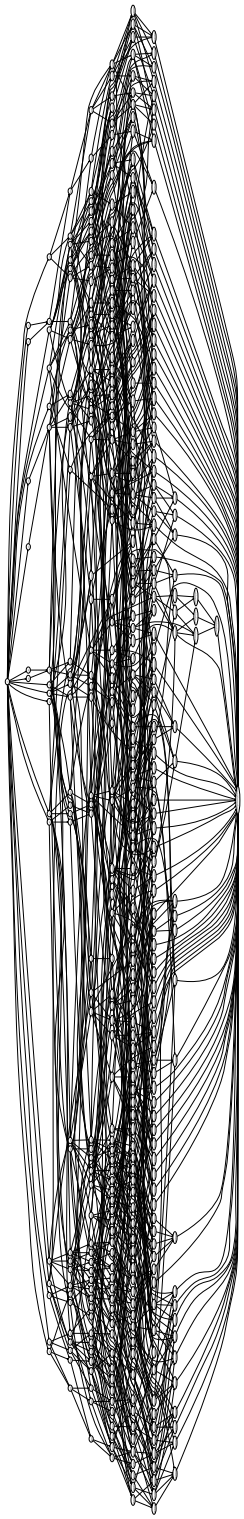


FIG. A.1 – Treillis élagué

Table des figures

Bibliographie

Ait-Kaci, H., Boyer, R. S., Lincoln, P. et Nasr, R. (1989). Efficient implementation of lattice operations. *Programming Languages and Systems*, 11(1):115–146. 72

Andersen, C. A. F. et Brunak, S. (2004). Representation of protein-sequence information by amino acid subalphabets. *AI Mag.*, 25(1):97–104. 65, 66

Angluin, D. (1982). Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765. 33, 116

Angluin, D. (1988). Identifying languages from stochastic examples. Rapport technique YALEU/DCS/RR-614, Yale University. 25

Argos, P. (1987). Analysis of sequence-similar pentapeptides in unrelated protein tertiary structures : strategies for protein folding and a guide for site-directed mutagenesis. *Journal of Molecular Biology*, 197:331–348. 60

Arikawa, S., Kuhara, S., Miyano, S., Shinohara, S. et Shinohara, T. (1992). A learning algorithm for elementary formal systems and its experiments on identification of transmembrane domains. *In Proceedings of the twenty-fifth Hawaii International Conference on System Science*, pages 675–684. 14

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. 10, 14

Arimura, H., Fujino, R., Shinohara, T. et Arikawa, S. (1994). Protein motif discovery from positive examples by minimal multiple generalization over regular patterns. *In Proceedings of Genome Informatics Workshop*, pages 39–48, Tokyo. Universal Academy Press. 14

Armon, A., Graur, D. et Ben-Tal, N. (2001). Consurf : An algorithmic tool for the identification of functional regions in proteins by surface mapping of phylogenetic information. *Journal of Molecular Biology*, 307:447–463. 5

Bairoch, A. (1992). Prosite : a dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 20:2013–2018. 8, 104

Birkhoff, G. (1973). *Lattice Theory*. American Mathematical Society Colloquium Publications, Rhode Island. 68, 69

Branden, C. et Tooze, J. (1999). *Introduction to Protein Structure*. Garland Publishing, Inc. New York, second edition édition. 5

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):279–305. 5, 9

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. 14

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*. 10

Brejová, B., DiMarco, C., Vinař, T., Hidalgo, S. R., Holguin, G. et Patten, C. (2000). Finding patterns in biological sequences. Rapport technique, University of Waterloo, Canada. 5

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. 13

Cannata, N., Toppo, S., Romualdi, C. et Valle, G. (2002). Simplifying amino acid alphabets by means of a branch and bound algorithm and substitution matrices. *Bioinformatics*, 18(8):1102–1108. 65, 66

Casari, G., Sander, C. et Valencia, A. (1995). A method to predict functional residues in proteins. *Nature Structural Biology*, 2:171–178. 5

Caseau, Y. (1993). Efficient handling of multiple inheritance hierarchies. *In Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 271–287. ACM Press. 72

Chomsky, N. (1956). Three models for the description of language. *IRE Trans. on Information Theory*, 2(3). 2, 19, 24

Cohen, N. H. (1991). Type-extension type test can be performed in constant time. *ACM Transactions on Programming Languages and Systems*, 13(4):626–629. 71, 72

Comet, J.-P. (1998). *Programmation Dynamique et Alignements de Séquences Biologiques*. Thèse de doctorat, Université de technologie de Compiègne. 74

Cornette, J. L., Cease, K. B., Margalit, H., Spouge, J. L., Berzofsky, J. A. et Delisi, C. (1987). Hydrophobic scales and computational techniques for detecting amphipathic structures in proteins. *Journal of Molecular Biology*, 195:659–685. 61

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*. 32

Creighton, T. E. (1993). *Proteins*. Freeman. 60

Dayhoff, M. O., Eck, R. et Park, C. M. (1972). A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, pages 89–99. published by National biomedical research foundation Washington DC. 65, 66

Dayhoff, M. O., Schwartz, R. M. et Orcutt, B. (1978). A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5 suppl.3:345–352. published by National biomedical research foundation Washington DC. 63, 65, 66

del Sol Mesa, A., Pazos, F. et Valencia, A. (2003). Automatic methods for predicting functionally important residues. *Journal of Molecular Biology*, 326:1289 1302. 6

Denis, F., Lemay, A. et Terlutte, A. (2002). Some classes of regular languages identifiable in the limit from positive data. *In* Peter Adriaans and Henning Fernau and Menno van Zaanen, éditeur : *Grammatical Inference: Algorithms and Applications. Proceedings of the Sixth International Colloquium, ICGI 2002*, volume 2484 de *Lecture Notes in Artificial Intelligence*, pages 63–76, Amsterdam, The Netherlands. Springer-Verlag Heidelberg. 34

Dupont, P., Miclet, L. et Vidal, E. (1994). What is the search space of the regular inference ? *In* R. C. Carrasco and José Oncina, éditeur : *Grammatical Inference: Algorithms and Applications. Proceedings of the Second International Colloquium, ICGI 1994*, volume 862 de *Lecture Notes in Artificial Intelligence*, pages 25–37, Alicante, Spain. Springer-Verlag Heidelberg. 29

Fan, K. et Wang, W. (2003). What is the minimum number of letters required to fold a protein ? *Journal of Molecular Biology*, 328:921–926. 65, 66

Feng, D.-F., Johnson, M. S. et Doolittle, R. F. (1985). Aligning amino acids sequences : comparison of commonly used methods. *Journal of Molecular Evolution*, 21:112–125. 63

Fernau, H. (2000). Identification of function distinguishable languages. *In* Arimura, H., Jain, S. et Sharma, A., éditeurs : *Proceedings of Algorithmic Learning Theory : 11th International Conference, ALT 2000*, volume 1968, pages 116–130, Sydney, Australia. Springer-Verlag Heidelberg. 34

Figureau, A., Soto, M. A. et Tohá, J. (2003). A pentapeptide-based method for protein secondary structure prediction. *Protein Engineering*, 16(2):103–107. 65, 66

Fitch, W. M. (1966). An improved method of testing for evolutionary homology. *Journal of Molecular Biology*, 16:9–16. 63

Fitch, W. M. (1967). Construction of phylogenetic trees. *Science*, 155(760):279–284. 63

Fredouille, D. (2003). *Inférence d'automates finis non déterministes par gestion de l'ambiguïté, en vue d'applications en bioinformatique*. Thèse de doctorat, Université de Rennes 1, France. 29, 32, 116

Galitsky, B., Gelfand, I. et Kister, A. (1998). Predicting amino acid sequences of the antibody human v_H chains from its first several residues. *In Proceedings of the National Academy of Science of the USA*, volume 95, pages 5193– 5198, USA. 65, 66

Gallet, X., Charloteaux, B., Thomas, A. et Brasseur, R. (2000). A fast method to predict protein interaction sites from sequences. *Journal of Molecular Biology*, 302:917–926. 6

Gambin, A., Lasota, S., Szklarczyk, R., Tiuryn, J. et Tyszkiewicz, J. (2002). Contextual alignment of biological sequences. *In Proceedings of the 2nd European Conference on Computational Biology ECCB*, volume 18 de *Oxford University Press*, pages S116–S127. 65, 86

García, P. et Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattren Analysis and Machine Intelligence*, 12(9):920–925. 33

Glaser, F., Pupko, T., Paz, I., Bell, R., Bechor-Shental, D., Martz, E. et Ben-Tal, N. (2003). Consurf : identification of functional regions in proteins by surface-mapping of phylogenetic information. *jBioinformatics*, 19:163–164. 6

Goh, C., Bogan, A., Joachimiak, M., Walther, D. et Cohen, F. (2000). Co-evolution of proteins with their interaction partners. *Journal of Molecular Biology*, 299:283–293. 6

Goh, C. et Cohen, F. (2002). Co-evolutionary analysis reveals insights into protein protein interactions. *Journal of Molecular Biology*, 324:177–192. 6

Gold, E. M. (1967). Language identification in the limit. *Information and control*, 10(5):447 – 474. 2, 25, 26, 33

Gold, E. M. (1978). Complexity of automaton identification from given data. *Information and control*, 37:302 – 320. 26, 27

Gonnet, G. H., Cohen, M. A. et Benner, S. A. (1994). Amino acid substitution during divergent evolution : the 400 dipeptide substitution matrix. *Biochem. Biophys. Res. Commun.*, 199:489–496. 63

Gorodkin, J., Heyer, L., Brunak, S. et Stormo, G. (1997). Displaying the information contents of structural rna alignments : the structure logos. *Computer Applications in the Biosciences*, 13:583–586. 79

Grantham, R. (1974). Amino acid difference formula to help explain protein evolution. *Science*, 185:862–864. 63

Gribskov, M., Luthy, R. et Eisenberg, D. (1990). Profile analysis. *Methods in Enzymology*, 183:146–159. 6

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. 10

Grätzer, G. (1971). *Lattice Theory : First Concepts and Distributive Lattices*. W. H. Freeman, San Francisco, CA. 68

Guralnik, V. et Karypis, G. (2001). A scalable algorithm for clustering protein sequences. *In BIOKDD'01*, pages 73–80. 10

Harel, D., Tiuryn, J. et Kozen, D. (2000). *Dynamic Logic*. MIT Press. 47

Henikoff, S. et Henikoff, J. G. (1991). Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19(23):6565–6572. 14

Henikoff, S. et Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *In Proceedings of the National Academy of Science of the USA*, volume 89, pages 10915–10919, USA. 63

Hernandez, D., Gras, R. et Appel, R. (2004). Model : an efficient strategy for ungapped local multiple alignment. *Computational Biology and Chemistry*, 28:119–128. 8, 116

Higuera (de la), C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138. 25, 27

Higuera (de la), C. (2002). A bibliographical study of grammatical inference. nothing to say. 2, 24, 33

Hopcroft, J. et Ullman, J. (1980). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, N. Reading, MA. 17

Hu, Z., Ma, B., Wolfson, H. et Nussinov, R. (2000). Conservation of polar residues as hot spots at protein interfaces. *Proteins: Structure, Function, and Genetics*, 39:331–342. 6

Hudak, J. et Mcclure, M. (1999). A comparative analysis of computational motif-detection methods. *In Proceedings of the Pacific Symposium of Biocomputing PSB'99*, pages 138–139. 5

Hughey, R. et Krogh, A. (1996). Hidden markov models for sequence analysis : extention and analysis of the basic method. *CABIOS*, 12(2):95–107. 6

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. 32

Ioerger, T. R. (1996). *Change of representation in machine learning, and application to protein structure prediction*. Thèse de doctorat, University of Illinois, Urbana-Champaign, USA. 65

Ioerger, T. R. (1997). The context-dependence of amino acid properties. *In* Press, A., éditeur : *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology, ISMB 1997*, pages 157–166. 65, 66

Jiménez Montaño, M. A. et Ramos Fernández, A. (2004). Application of reduced alphabets to the analysis and design of proteins. IV National Meeting of Biotechnology , IPN Biotechnology Network. Santa Cruz, Tlaxcala, México. Nov. 10-12, 2004 (in Spanish). 1, 65

Jiménez-Montaño, M. A. et Zamora-Cortina, L. (1981). Evolutionary model for the generation of amino acid sequences and its applicatyion to the study of mammal alpha-hemoglobin chains. Abstract published in Proceedings of the Seventh International Biophysics Congress, Maxico City. 65, 66

Jonassen, I. (1997). Efficient discovery of conserved patterns using a pattern graph. *Computer Applications in the Biosciences*, 13:509–522. 2, 10, 12, 13, 14, 103

Jonassen, I., Collins, J. et Higgins, D. (1995). Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4(8):1587–1595. 2, 10, 12, 13, 103

Kabsch, W. et Sander, C. (1983). Dictionary of protein secondary structure : pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22:2577–2637. 60

Karlin, S. et Ghandour, G. (1985). Multiple alphabet amino-acid sequence comparisons of the immunoglobulin κ-chain constant domain. *In Proceedings of the National Academy of Science of the USA*, volume 82, pages 8597–8601, USA. 65, 66

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. 11

Kidera, A., Konishi, Y., Oka, M., Ooi, T. et Scheraga, H. A. (1985). Statistical analysis of the physical properties of the 20 naturally occurring amino acids. *Journal of Protein Chemistry*, 4(1):23–55. 61

Kisters-Woike, B., Vangierdegom, C. et Müller-Hill, B. (2000). On the conservation of protein sequences in evolution. *Trends Biochem. Sci.*, 25:419–421. 6

Klingler, T. M. (1996). *Structural inference from correlations in biological sequences*. Thèse de doctorat, Program in Medical Informatics, Stanford University. 65, 66

Knuutila, T. (1996). Inductive inference from positive data : from heuristic to characterizing methods. *In* Laurent Miclet and Colin Higuera (de la), éditeur : *Grammatical Inference: Learning Syntax from Sentences. Proceedings of the Third International Colloquium, ICGI 1996*, volume 1147 de *Lecture Notes in Artificial Intelligence*, pages 22–47, Montpellier, France. Springer-Verlag Heidelberg. 33

Koshiba, T., Mäkinen, E. et Takada, Y. (1997). Learning deterministic even linear languages from positive examples. *Theoretical Computer Science*. 34

Krogh, A., Brown, M., Mian, S., Sjölander, K. et Haussler, D. (1994). Hidden markov models in computational biology. applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531. 6

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. 12

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. 10, 13

Lang, K. J. (1992). Random DFA's can be approximately learned from sparse uniform examples. *In Proceedings of the 5th ACM workshop on Computation Learning Theorie*, pages 45–52. 31

Lang, K. J. (1998). Evidence driven state merging with search. Rapport technique TR98–139, NECI. 31

Lang, K. J., Pearlmutter, B. A. et Price, R. A. (1998). Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *In* V. Honavar and G. Slutzki, éditeur : *Grammatical Inference: Algorithms and Applications. Proceedings of the Fourth International Colloquium, ICGI 1998*, volume 1433 de *Lecture Notes in Artificial Intelligence*, pages 1–12, Ames, Iowa, USA. Springer-Verlag Heidelberg. 31

Lemay, A. (2002). *De l'apport des langages résiduels en inférence grammaticale de langages réguliers*. Thèse de doctorat, Université de Lille I, France. 116

Li, T., Fan, K., Wang, J. et Wang, W. (2003). Reduction of protein sequence complexity by residue grouping. *Protein Engineering*, 5:323–330. 65, 66

Lichtarge, O. et Sowa, M. (2002). Evolutionary predictions of binding surfaces and interactions. *Curr. Opin. Struct. Biol.*, 12:21–27. 6

Lockless, S. W. et Ranganathan, R. (1999). Evolutionarily conserved pathways of energetic connectivity in protein families. *Science*, 286:295–299. 6

Luthy, R., McLachlan, A. D. et Eisenberg, D. (1991). Secondary structure-based profiles : use of structure-conserving scoring tables in searching protein sequence databases for structural similarities. *Proteins: Structure, Function, and Genetics*, 10:229–239. 63

Ma, B., Elkayam, T., Wolfson, H. et Nussinov, R. (2003). Protein protein interactions : structurally conserved residues distinguish between binding sites and exposed protein surfaces. *In Proceedings of the National Academy of Science of the USA*, volume 100, page 5772 5777, USA. 6

Majewski, J. et Ott, J. (2003). Amino acid substitutions in the human genome : evolutionary implications of single nucleotide polymorphisms. *GENE*, 305(2): 167–173. 65, 66

Martinez, H. M. (1988). A flexible multiple sequence alignment program. *Nucleic Acids Research*, 16(5):1683–1691. 10, 13

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *BSTJ*, 34:1045–1079. 57

Meyer, T., Tollin, G. et Cusaovich, M. (1994). Protein interaction sites obtained vis homology. the site of complexation of electron transfer portions of cytochrome c revealed by mapping amino acid substitution onto three-dimensional protein surfaces. *Biochimie*, 76:480–488. 6

Miclet, L. (1986). *Structural methods in Pattern Recognition*. Chapman and Hall, New York. 24

Miyata, T., Miyazawa, S. et Yasunaga, T. (1979). Two types of amino asid substitution in protein evolution. *Journal of Molecular Evolution*, 12:219–236. 63, 65, 66

Mocz, G. (1995). Fuzzy cluster analysis of simple physicochemical properties of amino acids for recognizing secondary structure in proteins. *Protein Science*, 4:1178–1187. 65, 66

Moore, E. F. (1956). Gedanken-experiments on sequential machines. *In* Shannon, C. E. et McCarthy, J., éditeurs : *Automata Studies*, pages 129–153. Princeton University Press. 37, 58

Murphy, L. R., Wallqvist, A. et Levy, R. M. (2000). Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Engineering*, 13(3):149–152. 65, 66

Myhill, J. (1957). Finite automata and the representation of events. Rapport technique 57-624, WADC. 23

Naor, D., Fischer, D., Jernigan, R. L., Wolfson, H. J. et Nussinov, R. (1996). Amino acid pair interchanges at spatially conserved locations. *Journal of Molecular Biology*, 256(5):924–938. 65, 66

Needleman, S. B. et Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453. 86

Nerode, A. (1958). Linear automaton transformation. *In Proceedings of the American Mathematical Society*, volume 9, pages 541–544. 23

Neuwald, A. F. et Green, P. (1994). Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712. 10, 11, 12, 13

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. 10

Nicolas, J. (1999). Grammatical inference as unification. Rapport technique 3632, Inria Rennes, France. 99, 100

Nix, R. (1983). *Editing by Exemple*. Thèse de doctorat, Yale University, Xerox palo Alto Research Center, California, USA. 13

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. 10, 12, 13

Oncina, J. et García, P. (1992). Inferring regular languages in polynomial updated time. *In* Pérez de la Blanca, S. et Vidal, éditeurs : *Processings of Pattern Recognition and Image Analysis*, World Scientific, pages 49–61. 31

Overington, J., Donnelly, D., Johnson, M. S., Sali, A. et Blundell, T. L. (1992). Environment-specific amino acid substitution tables : Tertiary templates and prediction of protein folds. *Protein Science*, 1(2):216–226. 63, 65

Overington, J., Johnson, M. S., Sali, A. et Blundell, T. L. (1990). Tertiary structural constraints on protein evolutionary diversity : templates, key residues and structure prediction. *Proc. R. Soc. Lond. B*, 241:132–145. 63

Pazos, F., Helmer-Citterich, M., Ausiello, G. et Valencia, A. (1997). Correlated mutations contain information about protein-protein interaction. *Journal of Molecular Biology*, 271:511–523. 6

Prlic, A., Domingues, F. S. et Sippl, M. J. (2000). Structure-derived substitution matrices for alignment of distantly related sequences. *Protein Engineering*, 13(8):545–550. 65, 66

Pupko, T., Bell, R., Mayrose, I., Glaser, F. et Ben-Tal, N. (2002). Rate4site : an algorithmic tool for the identification of functional regions in proteins by surface mapping of evolutionary determinants within their homologues. *jBioinformatics*, 18:71S–777. 6

Queen, C. et Wegman, M. (1982). L. j. korn. *Nucleic Acids Research*, 10:449–456. 11

Rao, J. K. M. (1987). New scoring matrix for amino acid residue exchanges based on residue characteristic physical parameters. *Int. J. Pept. Protein Res.*, 29:276–281. 63

Richardson, J. S. et Richardson, D. C. (1989). *Prediction of Protein Structure and the Principles of Protein Conformation*, chapitre Principles and patterns of protein conformation, pages 1–98. Plenum Press, New York, g. d. fasman édition. 61

Rigoutsos, I. et Floratos, A. (1998a). Combinatorial pattern discovery in biological sequences : the TEIRESIAS algorithm. *Bioinformatics*, 14(1):55–67. 2, 10, 103

Rigoutsos, I. et Floratos, A. (1998b). Motif discovery without alignment or enumeration (extended abstract). *In Proceedings of the second Annual International Conference on Computational Molecular Biology, RECOMB 1998*, pages 221–227. ACM Press. 2, 103

Rooman, M. J., J.-P., A. K. et Wodak, S. J. (1992). Extracting information on folding from the amino acid sequence : accurate predictions for protein regions with preferred conformation in the absence of tertiary interactions. *Biochimistry*, 31:10226–10238. 61

Roytberg, M. (1992). A search for common patterns in many sequences. *Computer Applications in the Biosciences*, 8(1):57–64. 10

Rulot, H. et Vidal, E. (1988). An efficient algorithm for the inference of circuit-free automata. *In* Ferratè, G., Pavlidis, T., Sanfeliu, A. et Bunke, H., éditeurs : *Advances in Structural and Syntactic Pattern Recognition*, pages 173–184. Springer-Verlag New York, Inc. 34

Rulot, H., Vidal, E. et Prieto, N. (1989). Learning accurate finite-state structural models of words through the ECGI algorithm. *In ICASSP*, pages 643–646. 34

Sagot, M., Escalier, V., Viari, A. et Soldano, H. (1995a). Searching for repeadted words in a text allowing for mismatches and gaps. *In Proceedings of the Second South American Workshop on string processing*, pages 87–100, Viñas del Mar, Chili. 64

Sagot, M., Viari, A. et Soldano, H. (1995b). 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. 10

Sagot, M., Viari, A. et Soldano, H. (1995c). Multiple comparison : a peptide matching approach. *In* Galil, Z. et Ukkonen, E., éditeurs : *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching1995*, volume 937 de *Lecture Notes in Computer Science*, pages 366–385. Springer-Verlag Heidelberg. 10, 11, 64

Sagot, M.-F. (1996). *Ressemblance lexicale et structurale entre macromolécules - Formalisation et approches combinatoires*. Thèse de doctorat, Université de Marne-la-vallée. 1, 65, 66

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 Matching1996*, Lecture Notes in Computer Science, pages 186–208. Springer-Verlag Heidelberg. 10, 12

Sakakibara, Y. (1997). Recent advances of grammatical inference. *Theoretical Computer Science*, 185(1):15–45. 25, 33

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. 10, 14

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. 13

Shannon, C. E. et Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois. republished in paperback 1963. 79

Shinohara, T. (1983). Polynomial time inference of extended regular pattern languages. *In Proceedings of RIMS Symposium on Software Science and Engineering*, Lecture Notes in Computer Science, pages 115–127. Springer-Verlag Heidelberg. 13

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. 10, 14

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, USA. 10, 11, 13

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. 10, 13, 65, 66

Smith, R. F. et Smith, T. F. (1992). Pattern-induced multisequence alignment (pima) algorithm employing secondary structure-dependent gap penalties for comparitive protein modelling. *Protein Engineering*, 5(1):35–41. 8, 10

Staden, R. (1989). Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences*, 5(4):293–298. 11

Suyama, M., Nishioka, T. et Oda, J. (1995). Searching for commun sequence patterns among distantly related proteins. *Protein Engineering*, 8(11):1075–1080. 10, 11

Tateishi, E., Maruyama, O. et Miyano, S. (1995). Extracting best consensus motifs from positive and negaitve examples. Rapport technique RIFIS-TR-CS-115, Research Institute of Fundamental Information Science, Kyushu University, Japan. 12

Tateishi, E. et Miyano, S. (1995). A greedy strategy for finding motifs from positive and negaitve examples. Rapport technique RIFIS-TR-CS-118, Research Institute of Fundamental Information Science, Kyushu University, Japan. 12

TAYLOR, W. (1986). The classification of amino acid conservation. *Journal of theoretical Biology*, 119:205–218. 64, 65, 66, 70, 115

THOMPSON, J., HIGGINS, D. et GIBSON, T. (1994). CLUSTAL W : improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680. 86

TRAKHENBROT, B. et BARZDIN, Y. (1973). *Finite automata : Behavior and synthesis*. Amsterdam, North Holland Pub. Comp. 31

VALDAR, W. S. J. et THORNTON, J. M. (2001). Protein-protein interfaces : analysis of amino acid conservation in homodimers. *Proteins: Structure, Function, and Genetics*, 42:108–124. 6

VALENCIA, A. et PAZOS, F. (2002). Computational methods for the prediction of protein interactions. *jCosb*, 12:368–373. 6

VALIANT, L. G. (1984). A theory of the learnable. *Communication of the ACM*, 27(11):1134 – 1142. 25

VAN NOORD, G. (1997). Fsa utilities : A toolbox to manipulate finite-state automata. *In* DARRELL RAYMOND, Derick Wood, S. Y., éditeur : *Automata Implementation*, volume 1260 de *Lecture Notes in Computer Science*, pages 87–108. Springer-Verlag Heidelberg. 99, 100

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. 10, 13

VITEK, J., HORSPOOL, R. N. et KRALL, A. (1997). Efficient type inclusion tests. *In Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 142–157. ACM Press. 72

WANG, J. et WANG, W. (1999). A computational approach to simplifying the protein-folding alphabet. *Nature*, 6(11):1033–1038. 66

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. 10

WATERMAN, M. S., ARRATIA, R. et GALAS, D. J. (1984). Pattern recognition in several sequences : consensus ans alignment. *Bull. Math. Biol.*, 46:515–527. 10, 11

Waterman, M. S. et Jones, R. (1990). Consensus methods for DNA and protein sequence alignment. *Methods in Enzymology*, 183:221–237. 10

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. 11

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. 10

Wu, T. D. et Brutlag, D. L. (1996). Discovering empirically conserved amino acid substitution groups in databases of protein families. *In Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, ISMB 1996*, pages 230–240. 65, 66

Yu, K. (2001). Theoretical determination of amino acid substitution groups based on qualitative physicochemical properties. Final Projects Submitted for Credit in Computational Molecular Biology, Biochemistry 218/ Medical Information Sciences 231, Project Selected from 2001. 65, 66

Yu, S. (1997). *Regular Languages, in Handbook of Formal Languages*, chapitre 2. Springer-Verlag. 17

Zvelebil, M. J. et Sternberg, M. J. (1988). Analysis and prediction of the location of catalytic residues in enzymes. *Protein Engineering*, 2:127–138. 5

Index

Résumé

Durant cette thèse, nous avons travaillé sur l'adaptation des algorithmes d'inférence grammaticale pour la recherche des propriétés communes à un ensemble de protéines. L'inférence grammaticale positive cherche à générer, à partir d'un ensemble de mots appartenant à un langage cible particulier inconnu, une représentation grammaticale qui est "optimale" par rapport à ce langage, c'est-à-dire qui rassemble et organise les particularités des mots du langage. Nous avons utilisé le diagramme de Taylor, qui classe les acides aminés suivant leurs propriétés physico-chimiques, pour construire, sous forme de treillis, un ordre sur les groupes d'acides aminés. Nous avons aussi développé une méthode d'inférence (SDTM) qui calcule les meilleurs alignements locaux entre les paires de protéines suivant un score fondé à la fois sur cet ordre et sur les propriétés statistiques de l'ensemble de protéines donné. Le résultat est une machine séquentielle proche de celle de Mealy avec des sorties réduites à "accepte" et "rejette". L'algorithme commence par construire le plus grand automate reconnaissant exactement les mots du langage et le généralise par fusions successives des paires de transitions correspondant aux acides aminés appariés dans les alignements sélectionnés. Les expérimentations ont montré l'intérêt de cette combinaison de méthodes importées de la découverte de motifs et de l'inférence grammaticale.

Mots-clés : bioinformatique, théorie des langages, automates, apprentissage automatique, inférence grammaticale, programmation logique, protéines, motifs.

Abstract

This work has addressed the problem of the adaptation of grammatical inference algorithms for the discovery of common properties in a set of proteins. Positive grammatical inference generates a particular grammatical representation which is "optimal" for this language, *i.e.* which gathers and organises the specific properties of the words of the given language, from a set of words belonging to a given target language. We used the Taylor diagram, which classifies amino acids according to their physico-chemical properties, in order to propose a specific order on groups of amino acids in the form of a lattice. During this work, we also developed an inference algorithm (SDTM) which computes best local alignments between pairs of proteins according to a score based on the order defined by the lattice and on the statistical properties of the given set of proteins. The result of the algorithm is a sequential machine close to a Mealy machine in which the outputs are reduced to "accept" and "reject". The algorithm begins by the construction of the biggest automaton recognising exactly the words of the language. Then, it generalizes the automaton by successively merging some pairs of transitions corresponding to paired amino acids in the selected alignments. Experiments have shown the interest of this combination of pattern discovery and grammatical inference methods.

Keywords : bioinformatics, theory of languages, automata, machine learning, grammatical inference, logical programming, proteins, motif.