Apprentissage probabiliste de similarités d'édition

Laurent Boyer

To cite this version:

Laurent Boyer. Apprentissage probabiliste de similarités d'édition. Autre [cs.OH]. Université Jean Monnet - Saint-Etienne, 2011. Français. NNT : 2011STET4027 . tel-00718835

HAL Id: tel-00718835
https://theses.hal.science/tel-00718835

Submitted on 18 Jul 2012

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

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

École Doctorale ED488 “Sciences, Ingénierie, Santé”

Apprentissage probabiliste de similarités d’édition

Thèse préparée
pour obtenir le grade de :

Docteur de l’université Jean Monnet de Saint-Étienne

Mention : Informatique

par Laurent Boyer

Laboratoire Hubert Curien, UMR CNRS 5516 Faculté des Sciences et Techniques

Soutenance le 24 Mars 2011, devant le jury composé de :

M. Laurent **Miclet**	IRISA, Lannion	*Rapporteur*
Mme. Christine **Solnon**	LIRIS, Lyon	*Rapporteur*
M. Ludovic **Denoyer**	LIP6, Paris	*Examinateur*
M. François **Jacquenet**	LaHC, Saint-Étienne	*Examinateur*
M. Marc **Sebban**	LaHC, Saint-Étienne	*Directeur de thèse*
M. Amaury **Habrard**	LIF, Marseille	*Co-Directeur de thèse*

Remerciements

Voici enfin la dernière étape de l'écriture de ce mémoire, après quelques quatre années de dur labeur je vais profiter de ces quelques lignes pour remercier l'ensemble des personnes qui ont contribué de près ou de loin à ces travaux.

Je tiens à remercier Laurent Miclet, Professeur à l'université de Rennes I, d'avoir accepté d'être rapporteur de cette thèse. Laurent Miclet est l'un des auteurs du livre français majeur sur l'apprentissage automatique et je le remercie de s'être intéressé à mes travaux.

Je souhaite remercier Christine Solnon, Maître de conférence à l'université de Lyon I, d'avoir bien voulu rapporter mon travail de thèse. Je tiens à la remercier pour l'ensemble des remarques et questions suite à la lecture de mon mémoire.

Je désire également remercier François Jacquenet, Professeur à l'université de Saint Étienne, d'avoir présidé mon jury et je terminerai par Ludovic Denoyer, Maître de conférence à l'université de Paris VI, d'avoir accepté de compléter ce jury.

Je désire également remercier Marc Sebban, mon directeur de thèse. Sans lui, ces travaux ne seraient pas ce qu'ils sont aujourd'hui. Il a su me diriger, m'encadrer, m'écouter quand j'en avais besoin tout en me laissant libre de mes choix. Malgré ses nombreuses fonctions à l'université de Saint Étienne, il a toujours été disponible. Travailler à ses côtés a toujours été un plaisir et m'a fait progresser tant dans le domaine scientifique que dans ma communication écrite ou orale. Je le remercie donc encore une fois de m'avoir proposé ce sujet de thèse et de m'avoir fait confiance pour la réalisation de celle-ci.

La seconde personne sans qui cette thèse n'aurait pas pu aboutir est sans aucun doute mon codirecteur de thèse, Amaury Habrard. Même s'il n'était pas présent physiquement tous les jours du fait de son rattachement à l'université d'Aix-Marseille I, il a toujours été disponible pour répondre à mes (trop) nombreuses questions. Je le remercie lui aussi pour la patience dont il a du faire preuve tout au long de cette thèse.

Je les remercie tous les deux de m'avoir supporté, et ce depuis le stage de master, et de m'avoir offert un cadre de travail agréable.

Je voudrais aussi remercier Émilie et Christophe qui ont commencé cette aventure avec moi et qui, malgré tous nos doutes, interrogations, arriveront comme moi à la fin. Merci à vous deux de m'avoir supporté pendant toute cette période.

J'aimerais enfin remercier tous mes collègues de travail. Un clin d'œil à Thierry et Frédéric pour l'ouverture à l'univers unix : les scripts, les fichiers de configuration... Ensuite je remercie les personnes du laboratoire Hubert-Curien avec qui j'ai pu avoir des

conversations par rapport à mes travaux ou pas : Adriana, Aurélien, Baptiste, Catherine, Cécile, Chahrazed, Christine, Christophe, Colin, David, Élisa, Fabrice A, Fabrice M, Florian, François, Franck, Hazaël, Jean, Jean-Christophe, Jean-Philippe, Julien, Léo, Marc B, Mathias, Mattias, Pierre, Philippe, Sabri, Stéphanie, Tung et merci à tous ceux que j'oublie.

Une petite pensée aussi à kk, p2t, tlmvpsp, conquiz avec qui les moments de détente avaient un côté divertissant et instructif.

Je terminerai par mes amis et ma famille qui ont eux aussi du me supporter. Merci pour leurs messages d'encouragements et leur soutien.

Sommaire

Introduction générale **1**

1 Notations-Définitions **5**
1.1 Introduction . . . 5
1.2 Apprentissage supervisé . . . 5
1.3 Les métriques . . . 8
1.3.1 Propriétés d'une métrique . . . 8
1.3.2 Métriques numériques . . . 9
1.3.3 Métriques symboliques . . . 10
1.3.4 Similarités symboliques . . . 13
1.4 Les machines à états . . . 15
1.4.1 Transducteurs . . . 15
1.4.2 Pair-HMM . . . 18
1.5 Conclusion . . . 20

2 Apprentissage de similarités d'édition **23**
2.1 Introduction . . . 23
2.2 Méthodes non probabilistes . . . 24
2.2.1 Méthodes basées sur la descente de gradient . . . 24
2.2.2 Méthodes basées sur les algorithmes génétiques . . . 25
2.2.3 Autres approches d'inférence non probabilistes . . . 27
2.3 Méthodes probabilistes . . . 27
2.3.1 L'algorithme EM . . . 27
2.3.2 Machine d'édition à un état . . . 29
2.3.3 Machine d'édition à plusieurs états . . . 32
2.3.4 Machine d'édition à plusieurs états avec ajout de connaissances . . 34
2.4 Conclusion . . . 35

3 Un modèle *memoryless* pour les arbres **37**
3.1 Introduction . . . 37
3.2 Définitions génériques des arbres . . . 38
3.3 DE selon Selkow . . . 39
3.3.1 Opérations et coûts d'édition . . . 39
3.3.2 Algorithme classique de calcul de la DE selon Selkow . . . 40
3.3.3 Apprentissage d'une DE stochastique jointe . . . 42
3.3.4 Apprentissage d'une DE stochastique conditionnelle . . . 50
3.4 Résultats expérimentaux . . . 56

3.4.1 Données artificielles 56
3.4.2 Reconnaissance de caractères manuscrits 59
3.5 DE selon Zhang & Shasha 64
3.5.1 Opérations d'édition 64
3.5.2 Algorithme classique du calcul de la DE de Zhang & Shasha 65
3.5.3 Apprentissage d'une DE stochastique 66
3.6 Plateforme SEDiL 70
3.7 Conclusion 73

4 Un modèle d'édition *non-memoryless* contraint **75**
4.1 Introduction 75
4.2 CSM 76
4.2.1 Cadre général 76
4.2.2 Calcul de $p(Y|X)$ 78
4.2.3 Apprentissage des paramètres 79
4.3 Relation entre CSM et pair-HMM 83
4.4 Application 89
4.4.1 Contexte scientifique 89
4.4.2 Données 90
4.4.3 MATCH^{TM} 90
4.4.4 Protocole expérimental et initialisation du CSM 91
4.4.5 Résultats 93
4.5 Conclusion et discussion 97

Conclusion générale **98**

Annexes **101**
A Expérimentations sur le TFBS ZF5_B 104
B Expérimentations sur le TFBS PAX4_03 108
C Expérimentations sur le TFBS PPARG_02 112

Introduction générale

Dans notre société moderne, une très grande quantité d'informations est quotidiennement générée, et souvent sauvegardée. Ces données peuvent provenir, par exemple, d'une chaîne commerciale stockant l'ensemble des achats effectués par ses clients, d'un moteur de recherche sauvegardant les requêtes et les pages visitées par des internautes, ou encore d'un journal créant un ensemble de bulletins d'informations sous la forme de dépêches AFP. L'ensemble de ces données doit souvent être traité *a posteriori*[1], majoritairement à l'aide d'outils informatiques, afin, par exemple, de proposer les bons produits en tête de gondole, de fournir des résultats de recherche sur le WEB plus pertinents, ou encore d'extraire des news sur un sujet particulier. Les outils permettant de résoudre ces tâches peuvent provenir de la fouille de données[2], ou *Data Mining*, dont le but est d'extraire un savoir, une connaissance, d'un ensemble de données de manière automatique ou semi-automatique. Ces outils peuvent également faire appel à des techniques d'**apprentissage automatique**, ou *Machine Learning*, pour lesquelles plusieurs cadres existent selon le type de données manipulées. Si celles-ci sont classées en différentes catégories (ou classes), nous les nommerons alors *données étiquetées*. Par exemple, la classe pourra être le genre d'une personne (homme ou femme) et les données seront une collection d'attributs (*e.g.* taille, poids, âge). Dans ce premier cas de figure, nous parlons d'*apprentissage supervisé*, l'objectif étant à partir d'un échantillon de données étiquetées de construire une fonction de classification permettant d'assigner une étiquette à tout nouvel exemple non connu. Si les étiquettes correspondent à des valeurs réelles (continues), nous parlons alors de régression. Dans le cas où l'échantillon d'apprentissage est composé d'exemples non-étiquetés, l'objectif n'est plus de classer mais plutôt d'extraire des groupes d'exemples partageant des ressemblances. Nous parlons alors d'*apprentissage non supervisé*. Tel sera le cas, par exemple, d'un marcheur découvrant au cours d'une promenade différents champignons, pour lesquels il ne pourra que produire une description physique, *e.g.* couleurs, formes, dimensions, sans être capable de déterminer leur catégorie. Le but de l'algorithme d'apprentissage est alors de découvrir automatiquement les catégories.

Pour effectuer leur tâche, la majorité des algorithmes d'apprentissage automatique (supervisés ou non supervisés) font appel à des calculs de **métriques** ou de **similarités**. Une métrique est une fonction qui mesure une distance entre deux objets. Dans le cas d'es-

[1] Nous nommerons ce procédé méthode *batch*. Notons qu'il est aussi possible de traiter les données suivant leur ordre d'arrivée, ce qui caractérise les méthodes *online*.

[2] La délégation générale à la langue française et aux langues de France (http://www.culture.gouv.fr/culture/dglf/) préconise l'utilisation du terme 'exploration de données'.

pace vectoriel, la distance la plus utilisée est certainement la distance euclidienne, définie de la manière suivante : $d_E(x,x') = \sqrt{\sum_i |x_i - x'_i|^2}, \forall x, x' \in \mathbb{R}^p$. Une autre solution est d'utiliser la distance de Mahalonobis : $d_M(x,x') = \sqrt{(x-x')^T M^{-1}(x-x')}, \forall x, x' \in \mathbb{R}^p$ qui permet d'associer, par l'intermédiaire de la matrice de variance/covariance M, un poids différent à chaque attribut, contrairement à la distance euclidienne qui associe un même poids à chaque *feature* du vecteur[3]. L'utilisation de poids permet donc de définir une distance mieux adaptée au problème considéré. Intuitivement, fixer *a priori* des bons poids pour améliorer les performances des algorithmes de classification ou de clustering est une opération difficile. L'utilisation de techniques d'apprentissage automatique s'avère être une solution adaptée à cette problématique, et a donné naissance ces dernières années à un domaine de recherche intitulé *Metric Learning*.

Une métrique doit respecter un ensemble de propriétés pour être considérée comme une vraie fonction de distance : semi-définie positivité, symétrie et inégalité triangulaire. Notons que certaines fonctions, qui semblent être naturellement pertinentes, ne remplissent pas ces contraintes, malgré le fait qu'elles soient efficaces pour résoudre des problèmes réels. C'est le cas par exemple du produit scalaire normalisé (cosinus) très utilisé dans les problématiques de recherche d'informations. De telles fonctions sont alors appelées fonctions de similarité.

Dans le cas de données structurées, c'est-à-dire des données représentées sous forme de chaînes (*e.g.* les mots de la langue française), d'arbres (*e.g.* des fichiers XML) ou de graphes (*e.g.* le réseau GSM), nous ne pouvons pas directement utiliser les distances ou similarités citées précédemment, les données n'appartenant pas à un espace vectoriel. Une solution est de faire appel à des métriques spécifiques, dédiées à la manipulation de telles données structurées, comme la **distance d'édition**. La distance d'édition entre deux objets correspond au nombre minimal de transformations élémentaires, également appelées opérations d'édition (insertion, délétion, substitution), nécessaires pour passer du premier objet au second. Une extension intuitive de la distance d'édition consiste à affecter à chaque opération un poids (ou coût). Comme dans le cas de la distance de Mahalonobis, ces poids sont dépendants du problème traité et s'ils sont correctement définis, ils permettent alors d'améliorer la pertinence de la distance de sorte qu'elle reflète au mieux les ressemblances et dissemblances entre objets comparés. Dans la littérature, un certain nombre d'approches a été proposé, même si elles sont bien moins nombreuses que dans le cas de données non structurées, pour apprendre ces coûts. Certaines techniques sont basées sur des méthodes d'optimisation numériques et nécessitent des exemples positifs et négatifs pour l'apprentissage. D'autres approches, stochastiques, permettent l'utilisation d'exemples positifs seulement. À la lecture de la littérature, le constat est que la majorité des travaux se concentrent sur la distance d'édition entre chaînes, qui constitue un problème algorithmiquement relativement simple. Cependant, face au développement des applications faisant intervenir des structures arborescentes (le WEB en étant un des meilleurs exemples), l'apprentissage de distances entre arbres est devenu un enjeu considérable. Dans le cadre de ce travail de thèse, nous avons décidé d'aborder cette problématique difficile. La recherche effectuée nous permet aujourd'hui de présenter une première contribution qui prend la forme d'une méthode d'apprentissage stochastique de coûts d'édition à partir de données arborescentes. Nous montrons que ce modèle peut être adapté à deux types de distances d'édition d'arbres différentes. Un autre constat

[3]Notons que si la matrice M est égale à la matrice identité alors $d_M = d_E$.

tiré de la littérature est qu'il existe peu de modèles permettant l'ajout de contraintes *a priori* et leur prise en compte durant le processus d'apprentissage. Nous apportons dans le cadre de cette thèse une contribution originale à cette problématique, sous la forme d'un modèle probabiliste d'édition contraint. Nous montrons que les machines à états ainsi apprises ont un pouvoir d'expression supérieur aux pair-Hidden Markov Models. Nous exploitons notre modèle sur une application de biologie moléculaire, dans le but d'extraire des sites de facteurs de transcription de gènes.

Contexte de la thèse

Cette thèse s'est déroulée au sein de l'équipe de Machine Learning du Laboratoire Hubert Curien de Saint-Étienne de l'Université de Lyon, UMR CNRS 5516. Les travaux présentés dans ce manuscrit ont été réalisés dans le cadre de deux projets ANR : Marmota (ANR MMSA) et Bingo2 (ANR MDCO). *Marmota (MAchine learning pRobabilistic MOdels Tree lAnguages*[4]*)*, est un projet qui se situe à l'intersection de trois domaines de recherche : les langages formels d'arbres, l'apprentissage automatique, et les modèles probabilistes. *Bingo2 (Knowledge Discovery For and By Inductive Queries in post-genomic applications*[5]*)*, vise à développer de nouvelles approches en fouille de données sur des applications post-génomiques.

Plan du manuscrit

Ce mémoire de thèse est organisé en quatre chapitres. Les deux premiers introduisent les notations, les objets et les modèles de la littérature qui concernent l'apprentissage de paramètres de la distance d'édition. Les deux chapitres suivants présentent les deux contributions de cette thèse. Le premier modélise un processus probabiliste d'édition entre arbres sous la forme d'un transducteur stochastique. Le second intègre des connaissances du domaine dans un processus d'édition sous contraintes entre chaînes.

Nous détaillons ci-dessous le contenu des quatre chapitres du mémoire.

Chapitre 1. Ce chapitre regroupe l'ensemble des notations et définitions utilisées dans la suite du manuscrit. Nous nous positionnons dans le cadre de l'apprentissage supervisé à partir de données structurées, l'objectif étant d'apprendre les paramètres d'une mesure de similarité sous la forme d'une machine à états stochastique.

Chapitre 2. Dans ce chapitre, nous dressons un état des lieux des différentes méthodes de la littérature ayant pour objectif d'apprendre les paramètres de la distance d'édition. Après une rapide présentation des techniques non probabilistes, nous introduisons l'algorithme *Expectation-Maximisation* EM, à l'origine de plusieurs méthodes d'apprentissage de similarités d'édition stochastiques.

Chapitre 3. Ce chapitre présente une méthode probabiliste pour apprendre des modèles joints et conditionnels de distances d'édition stochastiques entre arbres. Nous montrons sur plusieurs séries d'expérimentations qu'il est alors possible d'apprendre les coûts des opérations d'édition permettant d'améliorer les distances d'édition classiques entre

[4]http://marmota.gforge.inria.fr/.
[5]https://bingo2.greyc.fr/.

arbres. Nous présentons également SEDiL, une plateforme logicielle, disponible pour la communauté *Machine Learning*, qui intègre plusieurs algorithmes d'inférence (dont ceux présentés dans ce chapitre) et qui permet ainsi d'effectuer des expérimentations avec les distances d'édition entre chaînes et arbres.

Chapitre 4. Dans ce dernier chapitre de contribution, nous présentons une méthode de type EM pour apprendre un modèle conditionnel contraint de distance d'édition stochastique entre chaînes. Nous utilisons une machine à états permettant de modéliser différents contextes d'édition. Pour ajouter de la connaissance du domaine, les transitions entre états sont contraintes, s'exprimant sous la forme d'un ensemble de fonctions booléennes. Nous étudions le pouvoir d'expression de ce modèle avant de montrer son intérêt sur une tâche de biologie moléculaire.

1 Notations-Définitions

Résumé

Dans cette thèse, nous nous positionnons dans le contexte de l'apprentissage supervisé à partir de données structurées (chaînes ou arbres). L'objectif est d'apprendre des fonctions de similarités d'édition probabilistes basées sur des modèles stochastiques. Ces derniers sont induits par maximisation de vraisemblance d'un échantillon d'apprentissage de paires d'exemples jugés similaires. Nous introduisons dans ce chapitre l'ensemble des notations et définitions utiles à la manipulation de ces concepts.

1.1 Introduction

Dans cette thèse, nous nous intéressons à l'apprentissage supervisé de modèles d'édition entre données structurées. Ces modèles sont généralement probabilistes et induits par maximisation de vraisemblance d'échantillons de paires d'exemples (chaînes ou arbres).

Nous commençons par une brève introduction de l'apprentissage supervisé. Ensuite, nous introduisons les notions de métrique et de similarité. Nous terminons ce chapitre par la présentation de différentes machines à états utilisées dans ce mémoire de thèse pour modéliser des processus d'édition.

1.2 Apprentissage supervisé

Le but d'un processus d'apprentissage supervisé[1] est de construire automatiquement un modèle, à partir d'un ensemble d'observations, permettant de faire des prédictions. Nous pouvons ainsi apprendre des modèles de classification, pour prédire par exemple l'étiquette d'un chiffre manuscrit par un système de reconnaissance de l'écriture, ou des modèles de régression, pour prédire, par exemple, la température par un système de prévision météorologique.

Définition 1.1 (Échantillon d'apprentissage) *Soit un ensemble de m exemples d'apprentissage $\{x_1, \dots, x_m\}$. Chaque valeur x_i est supposée tirée de manière indépendante et identiquement distribuée suivant une distribution $D_{\mathcal{X}}$ sur l'espace d'entrée $\mathcal{X}$.*

---

[1] Notons qu'il existe d'autres types d'apprentissage (et d'algorithmes associés) que nous n'aborderons pas ici. Nous renvoyons le lecteur intéressé au livre *Apprentissage artificiel, concepts et algorithmes* d'A. Cornuéjols et L. Miclet [CM10].

À chaque exemple x_i, un oracle ou superviseur, noté f, associe une valeur $z_i \in \mathcal{Z}$ appelée étiquette. Si $\mathcal{Z}$ forme un ensemble discret, nous appelons z_i la classe de l'exemple x_i. Il s'agit alors de traiter un problème de classification. Si $\mathcal{Z}$ prend des valeurs continues, il s'agit alors d'un problème de régression.

L'ensemble $LS = \{(x_1, z_1), \dots, (x_m, z_m)\}$ constitue l'échantillon d'apprentissage.

Les valeurs $x_1, \dots, x_m$ peuvent prendre la forme de vecteurs $x_i =< x_{il}, \dots, x_{ip} >$ pour lesquels chaque composant est appelé *feature*.

Dans le cas de données structurées, ce qui est le cas dans cette thèse, les exemples $x_1, \dots, x_m$ peuvent correspondre à des chaînes de symboles ou des arbres.

Définition 1.2 (Alphabet et chaîne) *Un alphabet Σ est un ensemble fini non vide de caractères. Une chaîne x (ou mot) est une suite finie $x = a_1a_2 \dots a_n$ de symboles de Σ. Le mot vide est symbolisé par λ.*

L'ensemble de toutes les chaînes, y compris la chaîne vide, construites à partir de Σ est noté $\Sigma^\star$. La taille d'un mot se note $|x|$.

Définition 1.3 (Arbre) *Soit $\mathcal{V}$ un ensemble de nœuds. Nous définissons un arbre de manière récursive comme suit : un nœud est un arbre, et étant donnés T arbres $a_1, \dots, a_T$ et un nœud $r \in \mathcal{V}$, $x = r(a_1, \dots, a_T)$ est un arbre. De plus, nous pouvons associer une fonction d'étiquetage $l : \mathcal{V} \rightarrow \Sigma$ qui permet d'associer un symbole à chaque nœud de l'arbre. L'arbre vide est noté λ.*

L'ensemble de tous les arbres, y compris l'arbre vide, construits à partir de Σ est noté $\mathcal{T}(\Sigma)$.

Nous pouvons maintenant définir formellement le problème d'apprentissage supervisé.

Définition 1.4 (Apprentissage supervisé) *Un algorithme L d'apprentissage supervisé apprend automatiquement un classifieur, ou hypothèse, $h \in \mathcal{H}$, de la fonction cible f à partir d'un échantillon d'apprentissage $LS = \{(x_1, z_1), \dots, (x_m, z_m)\}$. Notons que $\mathcal{H}$ représente la classe d'hypothèses dans laquelle L peut effectuer la recherche de h.*

La phase d'apprentissage revient donc à trouver l'hypothèse h qui estime au mieux la cible f. Pour comparer h et f, nous définissons une mesure de perte l pour chaque élément x_i qui évalue le coût d'avoir pris comme décision $h(x_i)$ alors que la valeur désirée est $f(x_i)$. Par exemple, l peut correspondre au nombre d'erreurs de classification, ou encore à l'erreur quadratique dans le cas de problèmes de régression.

Définition 1.5 (Risque réel) *L'espérance de coût, ou risque réel ϵ_h, entre la fonction cible f et l'hypothèse h est définie par :*

$$\epsilon_h = \mathbf{P}_{x_i \sim D_\mathcal{X}} l(h(x_i), f(x_i))$$

Pour évaluer la qualité de l'hypothèse h, nous devons donc calculer le risque réel. Cependant, la mesure de ce risque nécessite de connaître la distribution $D_\mathcal{X}$ des exemples d'entrée, qui est malheureusement inconnue.

Apprendre un bon modèle, c'est-à-dire minimisant le risque réel, nécessite donc d'être guidé par un principe inductif. Nous présentons les principaux dans ce qui suit.

Le principe ERM. Une solution consiste à choisir l'hypothèse minimisant le risque empirique (*Empirical Risk Minimisation* ou principe ERM).

Définition 1.6 (Risque empirique) *Le risque empirique $\hat{\epsilon}_h$ d'une hypothèse $h \in \mathcal{H}$ est la perte moyenne observée sur l'échantillon d'apprentissage LS.*

$$\hat{\epsilon}_h = \frac{1}{|LS|} \sum_{i=1}^{|LS|} l(h(x_i), f(x_i))$$

L'idée intuitive de ce principe inductif est que l'hypothèse qui s'accorde le mieux aux données d'apprentissage est probablement une bonne hypothèse en généralisation. Ce principe suppose évidemment que les données d'apprentissage soient représentatives de la distribution $D_{\mathcal{X}}$ et de la fonction cible associée f.

Le principe ERM sera valide s'il est possible de montrer que l'hypothèse sélectionnée est proche de l'hypothèse optimale (inconnue).

Pour analyser la relation entre le risque réel et le risque empirique, le cadre le plus utilisé est certainement le paradigme PAC (Probablement Approximativement Correct) introduit par Valiant [Val84]. Dans ce modèle, l'objectif est de déterminer sous quelles conditions, la probabilité de choisir une mauvaise hypothèse (*i.e.* dont le risque réel est supérieur au risque réel optimal de plus de ρ) est bornée par δ, *i.e.* :

$$\forall \rho \geq 0, \delta \leq 1, Pr(|\epsilon_{\hat{h}} - \epsilon_{h^\star}| \geq \rho) < \delta, \tag{1.1}$$

où $h^\star$ est l'hypothèse optimale au sens du risque réel :

$$h^\star = \underset{h \in \mathcal{H}}{\text{argmin}}\ \epsilon_h,$$

et $\hat{h}$ est l'hypothèse optimale au sens du risque empirique :

$$\hat{h} = \underset{h \in \mathcal{H}}{\text{argmin}}\ \hat{\epsilon}_h.$$

Les travaux de Vapnik [Vap98] ont donné les conditions de garantie selon lesquelles le risque empirique pouvait converger vers le risque réel lorsque l'échantillon d'apprentissage tend vers l'infini. Plus formellement, le principe ERM est valide si la borne suivante est garantie sur la taille de l'échantillon d'apprentissage :

$$|LS| \geq \frac{2}{\rho^2} ln \frac{2n_{\mathcal{H}}}{\delta}$$

où $n_{\mathcal{H}}$ est soit la taille de l'espace d'hypothèses $\mathcal{H}$, quand celui-ci est de taille finie, soit la dimension de Vapnik-Chervonenkis[2], dans le cas infini.

Notons que ce paradigme nécessite de disposer d'exemples positifs et négatifs. Dans la situation où nous disposons d'exemples positifs seulement, ce qui est le cas dans le cadre de cette thèse, nous préférerons le principe de maximisation de vraisemblance.

[2]Pour rappel, la dimension de Vapnik-Chervonenkis est la taille du plus grand ensemble d'exemples pouvant être pulvérisés par la famille d'hypothèses $\mathcal{H}$.

Le maximum de vraisemblance. Dans ce cas, le choix de l'hypothèse optimale se porte sur celle qui maximise la vraisemblance des données de l'échantillon d'apprentissage. Dans ce cadre, nous supposons qu'il est possible de définir une distribution de probabilité sur l'espace des fonctions hypothèses. L'échantillon d'apprentissage est alors considéré comme une information modifiant la distribution de probabilité sur $\mathcal{H}$. Nous pouvons alors choisir l'hypothèse la plus probable *a posteriori*, c'est le principe du maximum de vraisemblance.

Sous l'hypothèse où les données sont i.i.d., le calcul de la vraisemblance est le produit des probabilités de chaque exemple sachant l'hypothèse choisie. Plus formellement :

$$L_h(LS) = \prod_{i=1}^{|LS|} Pr(x_i|h).$$

Le choix de l'hypothèse se porte donc sur celle maximisant $L_h(LS)$:

$$\hat{h} = \underset{h \in \mathcal{H}}{\text{argmax}}\ L_h(LS).$$

Le principe MDL. Notons enfin qu'il est possible de choisir l'hypothèse qui comprime au mieux les informations contenues dans l'échantillon d'apprentissage. C'est l'approche utilisée dans le cas du principe MDL (*Minimum Description Length*). L'idée est d'éliminer les redondances présentes dans les données afin d'en extraire des régularités permettant la description des exemples.

1.3 Les métriques

Dans cette section, nous présentons la notion de métrique. Après une présentation des propriétés à respecter, nous effectuons une revue rapide des métriques dans les contextes numériques puis symboliques. Nous terminons cette section par l'introduction des notions de similarités.

1.3.1 Propriétés d'une métrique

L'utilisation d'un grand nombre d'algorithmes d'apprentissage supervisé ou non supervisé nécessite de faire appel à la notion de métrique ou distance[3] entre éléments.

Pour être une distance, une fonction d doit respecter les quatre conditions suivantes, pour tout couple de données $(X, Y) \in \mathcal{X}$:

1) Condition de séparation.

$$\text{Si } d(X,Y) = 0, \text{ alors } X = Y \quad (1.2)$$

2) Condition de non-négativité.

$$d(X,Y) \geq 0 \quad (1.3)$$

[3] Nous ne ferons aucune distinction dans ce manuscrit entre les notions de distance et de métrique même s'il est possible de les différencier selon le fait que les données appartiennent à un espace métrique ou pseudo-métrique [Kel55].

Notons que ces deux premières conditions peuvent être réduites à la seule contrainte de semi-définie positivité.

3) Condition de symétrie.

$$d(X,Y) = d(Y,X) \tag{1.4}$$

4) Condition d'inégalité triangulaire.

$$\forall Z \in \mathcal{X},\ d(X,Y) + d(Y,Z) \geq d(X,Z) \tag{1.5}$$

Cette dernière propriété assure que la distance la plus courte entre deux éléments est toujours le chemin le plus direct. Elle est notamment à la base de nombreux algorithmes permettant d'accélérer la recherche des plus proches voisins.

1.3.2 Métriques numériques

Les distances suivantes sont décrites pour des espaces vectoriels de dimension p. Dans la suite du paragraphe, X et Y décrivent des vecteurs de dimension p, x_i (respectivement y_i) dénote le $i^{\text{ème}}$ élément du vecteur X (respectivement Y).

Distance de Minkowski. Les métriques les plus usitées sont certainement les distances issues de la L_k, appelée distance de Minkowski.

$$L_k = \sqrt[k]{\sum_{i=1}^{p} |x_i - y_i|^k}. \tag{1.6}$$

Suivant les différentes valeurs de k, nous distinguons généralement les distances suivantes :

- Si $k = 1$, cette distance est plus connue sous le nom de la distance Manhattan.

$$L_1 = d_1(X,Y) = \sum_{i=1}^{p} |x_i - y_i|. \tag{1.7}$$

- Dans le cas, $k = 2$, nous parlons de la distance euclidienne.

$$L_2 = d_2(X,Y) = \sqrt{\sum_{i=1}^{p} |x_i - y_i|^2}. \tag{1.8}$$

- Si $k = \infty$, nous obtenons la distance de Tchebychev, appelée aussi distance infinie :

$$L_\infty = d_\infty(X,Y) = \max_{i=1..p} |x_i - y_i|. \tag{1.9}$$

La Figure 1.1 représente l'ensemble des points à distance 1 du point de coordonnées $(0,0)$ pour les trois cas de la métrique de Minkowski présentés précédemment.

Au delà de cette relation géométrique entre les L_k, notons qu'il est possible de montrer qu'il existe deux constantes α et β telles que, pour chaque entier positif a, b :

$$\alpha L_a \leq L_b \leq \beta L_a.$$

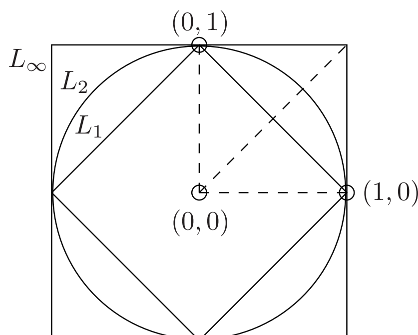


Fig. 1.1 – Ensemble des points à distance 1 du point de coordonnées $(0, 0)$ dans un plan en utilisant les distances L_1 (Équ. 1.7), L_2 (Équ. 1.8) et L_∞ (Équ. 1.9).

Ceci signifie que les distances de Minkowski peuvent être considérées comme "équivalentes" à une constante près.

Distance de Mahalanobis. Un autre type de distances utilisées entre données numériques sont les distances de Mahalanobis $d_M(X, Y)$. Elle se définit par :

$$d_M(X, Y) = \sqrt{(X - Y)^T M^{-1} (X - Y)}, \qquad (1.10)$$

où M est la matrice de covariance.

Notons que si M est la matrice identité I, la distance de Mahalanobis se réduit à la distance euclidienne. Cette distance est souvent utilisée en *clustering* pour tenir compte des corrélations entre les *features*.

Les métriques présentées précédemment s'appliquent à des données numériques. Dans le cadre de ce manuscrit, nous voulons calculer et apprendre des distances entre données symboliques. Nous avons donc deux solutions :

- transformer les données symboliques sous forme de données vectorielles (par exemple en utilisant des n-grammes [Sha48]) et ensuite y appliquer une distance de Minkowski, par exemple,
- ou utiliser des métriques construites spécialement pour les données symboliques.

1.3.3 Métriques symboliques

Dans cette section, nous présentons un ensemble de métriques qui s'appliquent à des données symboliques. La majorité de ces métriques ont été développées pour mesurer des distances entre chaînes.

La distance de Hamming

La *distance de Hamming* [Ham50] d_h s'applique entre deux chaînes X et Y de même longueur et correspond au nombre de caractères différents entre X et Y.

$$d_h(X, Y) = |\{i : x_i \neq y_i\}|, \; 0 \leq i \leq |X|. \qquad (1.11)$$

Exemple. *Soient les mots* $X = MAR\underline{TH}A$ *et* $Y = MAR\underline{HT}A$*, la distance de Hamming* d_h *est égale à* $d_h(X, Y) = 2$. ◦

Distance de Jaccard

La *distance de Jaccard* d_J [Jac12] est une mesure utilisée pour mesurer la diversité de deux ensembles de symboles. Elle correspond au rapport du nombre de caractères communs entre les deux ensembles sur le nombre de caractères différents des deux ensembles.

$$d_J(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}. \tag{1.12}$$

Exemple. *Soient les chaînes* $MARTHA$ *et* $MARHTAS$ *desquelles les ensembles* $X = \{M, A, R, T, H\}$ *et* $Y = \{M, A, R, H, T, S\}$ *sont déduits. La distance de Jaccard est égale à* $d_J(X, Y) = 1 - \frac{5}{6} = .1667$. ◦

Distance de Jaro

La *distance de Jaro* d_{jaro} [Jar95] est une mesure de similarité entre deux chaînes de caractères. Cette distance est utilisée pour la détection de doublons. Elle est adaptée au traitement de chaînes courtes. Le calcul de cette distance nécessite le calcul de deux accumulateurs. Le premier, m, est le nombre de caractères communs entre les deux chaînes, le second, t, correspond au nombre de transpositions parmi les caractères communs entre les deux chaînes.

$$d_{jaro}(X, Y) = \frac{1}{3}\left(\frac{m}{|X|} + \frac{m}{|Y|} + \frac{m - t/2}{m}\right). \tag{1.13}$$

Exemple. *Soient les mots* $X = MARTHA$ *et* $Y = MARHTAS$*, la similarité de Jaro est de* 0.896*. En effet, le nombre de caractères communs* m *entre* X *et* Y *est de 6 :* M, A, R, T, H, A*. Étant donnés les caractères communs, nous observons deux transpositions :* $t = 2$*. En effet, en lisant le premier mot, nous observons que deux caractères sont dans une position différente de celle du second mot. D'où* $d_{jaro} = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{7} + \frac{6-2/2}{6}\right) = .896$*.* ◦

Distance d'édition

Une des métriques les plus connues sur les chaînes de caractères est la *distance d'édition* [WF74]. La distance d'édition utilise un ensemble de trois opérations d'édition :

- la substitution d'un caractère par un autre noté (x_t, y_v) qui transforme le mot $X = ux_tw$ en $Y = uy_vw$ avec $u, w \in \Sigma^\star$ et $x_t, y_v \in \Sigma$,
- la délétion d'un caractère de la première chaîne noté (x_t, λ) qui transforme le mot $X = ux_tw$ en $Y = uw$ avec $u, w \in \Sigma^\star$ et $x_t \in \Sigma$,

– l'insertion d'un caractère dans la seconde chaîne (λ, y_v) qui transforme le mot $X = uw$ en $Y = uy_vw$ avec $u, w \in \Sigma^\star$ et $y_v \in \Sigma$.

Définition 1.7 (Distance d'édition ou de Levenshtein [Lev66]) *La distance d'édition d_l entre deux mots X et Y est le nombre minimal d'opérations d'édition qui permettent de réécrire le mot X en le mot Y.*

Cette distance peut être généralisée en associant à chaque opération d'édition un coût. La distance devient donc la somme minimale des coûts d'opération pour transformer X en Y.

Le pseudo-code de l'algorithme du calcul de la distance d'édition est présenté ci-dessous. Par des techniques de programmation dynamique, nous pouvons calculer la d_l en temps quadratique $O(|X| \times |Y|)$ et linéaire $O(max(|X|, |Y|))$ en espace.

$$\begin{array}{ll} d_l(\lambda, \lambda) & = 0 \\ d_l(x_1 \cdots x_t, \lambda) & = d_l(x_1 \cdots x_{t-1}, \lambda) + c(x_t, \lambda) \\ d_l(\lambda, y_1 \cdots y_v) & = d_l(\lambda, y_1 \cdots y_{v-1}) + c(\lambda, y_v) \\ d_l(x_1 \cdots x_t, y_1 \cdots y_v) & = \min \left\{ \begin{array}{l} d_l(x_1 \cdots x_{t-1}, y_1 \cdots y_{v-1}) + c(x_t, y_v) \\ d_l(x_1 \cdots x_t, y_1 \cdots y_{v-1}) + c(\lambda, y_v) \\ d_l(x_1 \cdots x_{t-1}, y_1 \cdots y_v) + c(x_t, \lambda). \end{array} \right. \end{array} \tag{1.14}$$

Comme nous l'avons dit, un coût peut être attribué à chaque opération d'édition. Notons que la distance de Hamming, présentée précédemment, peut être vue comme un cas particulier de la distance d'édition pour laquelle le coût des opérations d'insertion et de délétion est infini et le coût des opérations de substitution est égal à 1.

Exemple. *Soient les mots $X = MARTHA$ et $Y = MARHTAS$, la distance de Levenshtein entre X et Y est égale à $d_l(X, Y) = 3$. La matrice de la Table 1.2 montre le calcul de la distance par programmation dynamique. La suite des opérations correspond en la substitution de M, A et R par eux-mêmes, la substitution de H par T puis de T par H, une nouvelle substitution de A par A et enfin de l'insertion du caractère S. Remarquons qu'il existe d'autres solutions de coûts identiques. En effet, en milieu de réécriture, nous pourrions utiliser la séquence suivante : la délétion du H puis la substitution du T par lui-même et suivie de l'insertion du H.* ○

Dans le cadre de la bio-informatique, d'autres distances basées sur la distance d'édition peuvent être utilisés pour mesurer des similarités entre chaînes de nucléotides. Par exemple, nous pouvons citer :

– La *distance de Needleman-Wunch* [NW70]. Elle est utilisée pour calculer un alignement global entre deux séquences de protéines ou de nucléotides. Les opérations d'insertion et de délétion ont un même coût, appelé pénalité de trou, et une fonction de coût est définie entre les différents éléments alignés.

– La *distance de Smith-Waterman* [SW81]. Elle permet de procéder à des alignements locaux, c'est-à-dire de déterminer des régions similaires entre morceaux de séquences d'ADN.

	λ	M	A	R	H	T	A	S
λ	**0**	1	2	3	4	5	6	7
M	1	**0**	1	2	3	4	5	6
A	2	1	**0**	1	2	3	4	5
R	3	2	1	**0**	1	2	3	4
T	4	3	2	1	**1**	1	2	3
H	5	4	3	2	1	**2**	3	4
A	6	5	4	3	3	2	**2**	**3**

Tab. 1.2 – Matrice du calcul de la distance de Levenshtein entre les mots MARTHA et MARHTAS. En gras, la suite des opérations d'édition de coût minimal.

L'ensemble des distances présentées dans cette section permet de calculer des distances entre chaînes. Notons que la distance d'édition peut être étendue à d'autres types de données structurées comme les données arborescentes ou les graphes. Comme nous l'avons vu, la distance d'édition utilise une fonction de coût pour les opérations. Ceci en fait une distance adaptable à différents types de contextes. Malheureusement, définir ces coûts est une tâche souvent difficile dès lors que nous ne disposons pas de connaissance suffisante du domaine. C'est pourquoi, dans ce manuscrit, nous proposons des méthodes pour les apprendre automatiquement. Comme nous le verrons dans la suite, nous nous plaçons dans un cadre stochastique pour trouver les paramètres optimisés pour un problème donné. Cependant, dans certains cas, nous verrons que la phase d'apprentissage peut conduire à une non-vérification des contraintes de métrique présentées en Section 1.3.1. Dans ce cas, nous ne parlerons plus de réelle distance, mais plutôt de similarité.

1.3.4 Similarités symboliques

Les fonctions présentées dans le paragraphe précédent ont la particularité d'être des vraies métriques. Néanmoins, de nombreuses fonctions de similarité sont utilisées de manière efficace, malgré le fait qu'elles ne respectent pas toutes les propriétés d'une distance (citons par exemple la matrice BLOSUM utilisée en bioinformatique pour calculer des distances entre chaînes de nucléotides [SVA06]).

Un premier exemple de similarité peut être pris dans le contexte de l'analogie.

Le raisonnement par analogie permet de comprendre et d'interpréter de nouvelles situations à partir de situations analogues. Le raisonnement analogique consiste à utiliser les ressemblances et les dissemblances entre deux domaines, appelés la source et la cible, pour transférer les résultats connus de la source vers la cible. Une analogie pourra prendre la forme suivante : *A est à B ce que C est à D* [Lep03].

Il est alors possible d'utiliser ce cadre pour mesurer la similarité entre éléments. En effet, si nous savons que A et B sont similaires alors C et D le seront aussi. Dans le cadre d'éléments non-similaires, le raisonnement reste inchangé, nous parlons alors de dissimilarité analogique.

Dans [BMD07, MBD08], les auteurs montrent que l'analogie peut être utilisée en apprentissage automatique. Par exemple, comme présenté dans [MBDM07], il est possible

de modéliser la relation entre le futur et l'infinitif d'un verbe. Si nous avons le mot *recherchera* dont nous voulons trouver la classe (*futur* ou *infinitif*) et que dans l'échantillon d'apprentissage, nous avons les mots *rechercher*, *commencer* et *commencera* avec pour classe respective infinitif, infinitif et futur. Nous pouvons observer que : *commencer est à commencera ce que rechercher est à recherchera.* Nous pouvons alors remplacer chaque mot de l'analogie par sa classe associée, ce qui donne : *infinitif est à futur ce que infinitif est à A*. Par raisonnement par analogie, nous obtenons que *A=futur*. Nous en concluons que la classe de *recherchera* est *futur*.

Un certain nombre d'autres similarités sont issues de la distance d'édition. Afin d'exploiter les algorithmes de SVM sur des données séquentielles, plusieurs noyaux d'édition ont été proposés dans la littérature.

Le noyau le plus courant exploitant la distance d'édition est le suivant [CHM04, LJ04] :

$$K(X, Y) = e^{-t.d_l(X,Y)}, \forall X, Y \in \Sigma^\star,$$

avec t une valeur réelle positive. Cependant, comme mentionné dans [CHM04], il existe certaines valeurs de t pour lesquelles ce noyau n'est pas défini positif, ce qui ne garantit pas la convergence d'algorithmes d'apprentissage tels que les SVMs. En pratique, t doit donc être correctement paramétré afin d'espérer obtenir un noyau valide.

Dans [NB06], Neuhaus et Bunke proposent l'utilisation du noyau suivant :

$$K(X, Y) = k_{X_0}(X, Y) = 1/2(d_l(X, X_0)^2 + d_l(X_0, Y)^2 - d_l(X, Y)^2).$$

où X_0 est appelé la chaîne *zéro*. Ce noyau mesure donc la similarité des chaînes X et Y étant donnée une chaîne de référence X_0. Cette approche peut être généralisée à un ensemble de chaînes références. Notons que ce problème de la recherche du meilleur ensemble de chaînes *zéro* est néanmoins un problème NP-complet.

Une autre possibilité pour représenter des similarités d'édition est de considérer que le processus d'édition est un processus aléatoire, où les opérations, et donc les scripts d'édition, relèvent d'une distribution de probabilité à déterminer [RY98]. Dans ce cas, nous ne calculons plus la distance d'édition entre deux chaînes X, Y mais plutôt la probabilité que X soit changée en Y par un script d'édition, quel qu'il soit.

Soit $e = e_1 \dots e_n$, où $e_i \in (\Sigma \cup \{\lambda\})^2$ un script d'édition possible transformant X en Y par une séquence d'opérations[4]. La probabilité de ce script, notée $\pi_\delta(e)$, est le produit des probabilités des opérations d'édition successives, tel que

$$\pi_\delta(e) = \prod_{i=1}^{n} \delta(e_i) \times \delta(\#),$$

où $\delta(\#)$, noté également $\delta(\lambda, \lambda)$, est la probabilité de fin de script et $\delta(e_i)$ la probabilité de l'opération e_i.

[4]Dans le cas des chaînes $X = MARTHA$ et $Y = MARHTAS$, nous pourrons avoir $e = (M, M)(A, A)(R, R)(T, H)(H, T)(A, A)(\lambda, S)(\lambda, \lambda)$.

Pour modéliser la distance entre deux chaînes, nous pouvons calculer la probabilité de chacun des chemins qui permettent de transformer X en Y. Cette probabilité, notée $p_\delta(X,Y)$, permet de définir une similarité d'édition stochastique (SE).

Définition 1.8 (SE stochastique) *Soit $E(X,Y)$, l'ensemble des scripts d'édition permettant de transformer X en Y. La SE stochastique entre deux chaînes est définie par :*

$$d_s(X,Y) = -\log p_\delta(X,Y) = -\log \sum_{e \in E(X,Y)} \pi_\delta(e).$$

Notons que nous pouvons utiliser une autre stratégie pour obtenir une SE stochastique en gardant seulement le script le plus probable et ainsi calculer la similarité $d_v(X,Y)$ dite de Viterbi.

$$d_v(X,Y) = -\log \max_{e \in E(X,Y)} \pi_\delta(e).$$

Déterminer une distribution de probabilité sur les scripts d'édition nécessite de définir une ou plusieurs distributions sur les opérations d'édition elles-mêmes. L'objectif de ce manuscrit est de présenter différentes méthodes pour apprendre ces distributions à partir d'exemples d'apprentissage. Pour modéliser ces distributions, nous allons faire appel à des machines à états probabilistes, que nous introduisons dans la section suivante.

Notons que par abus de langage, et pour simplifier nos propos dans la suite du document, nous utiliserons indifféremment les termes de métrique, de distance ou de similarité, pour exprimer une similarité d'édition stochastique.

1.4 Les machines à états

Dans cette section, nous présentons un ensemble de machines probabilistes utilisables dans le cadre de l'apprentissage des paramètres d'une similarité d'édition. Tout d'abord, nous présentons les transducteurs. Nous utiliserons ces machines dans le Chapitre 3 pour l'apprentissage des paramètres d'une distance d'édition stochastique entre arbres. Nous introduirons ensuite les pair-HMMs qui sont à la base des travaux proposés dans le Chapitre 4 pour apprendre les distances d'édition contraintes entre chaînes.

1.4.1 Transducteurs

Un processus d'édition peut se voir comme une transduction entre une chaîne d'entrée et une chaîne de sortie.

Avant de présenter la notion de transducteur, nous introduisons la notion d'automate fini permettant de définir des modèles de langage.

Définition 1.9 (Automate) *Un automate à états fini est défini par un 5-uplet $\mathcal{A} = \langle \mathcal{Q}, \Sigma, T, I, F \rangle$, où :*
- *$\mathcal{Q}$ est un ensemble fini d'états,*
- *Σ est un ensemble fini de symboles appelé l'alphabet de l'automate,*
- *$T : (\mathcal{Q} \times \Sigma \rightarrow \mathcal{Q})$ est une fonction de transition,*
- *$I \subseteq \mathcal{Q}$ est l'ensemble des états initiaux,*
- *$F \subseteq \mathcal{Q}$ est l'ensemble des états finaux.*

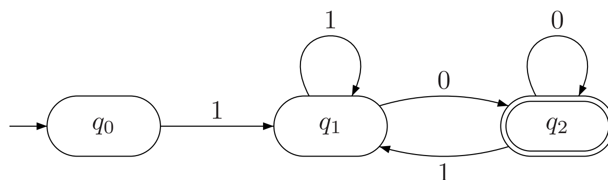


Fig. 1.3 – Représentation graphique de l'automate $\mathcal{A} = \langle \mathcal{Q}, \Sigma, T, I, F \rangle$ avec $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $T = \{(q_0, 1, q_1), (q_1, 1, q_1), (q_1, 0, q_2), (q_2, 1, q_1), (q_2, 0, q_2)\}$, $I = \{q_0\}$ et $F = \{q_2\}$.

Définissons maintenant la sémantique opérationnelle d'un automate.
Soient $\mathcal{A} = \langle \mathcal{Q}, \Sigma, T, I, F \rangle$ un automate fini et $w \in \Sigma^\star$ un mot à analyser.

- Une configuration de $\mathcal{A}$ est caractérisée par un couple (S, u) tel que $S \in \mathcal{Q}$ est l'état courant et $u \in \Sigma^\star$ correspond à un suffixe du mot à analyser.
- La configuration (S', u') est dérivable en une étape de la configuration (S, u) (notée $(S, u) \rightarrow (S', u')$) si $u = a.u'$ avec $a \in \Sigma$, $u' \in \Sigma^\star$ et $(S, a, S') \in T$.
- La configuration (S', u') est dérivable en plusieurs étapes de la configuration (S, u) (notée $(S, u) \xrightarrow{\star} (S', u')$) si (S', u') peut être obtenu de (S, u) par une succession de dérivations en une étape et u' est un suffixe de u.
- Un mot w est accepté par l'automate $\mathcal{A}$ s'il existe une dérivation $(S_0, w) \xrightarrow{\star} (S_i, \lambda)$ où $S_0 \in I$ est un état initial et $S_i \in F$ est un état final.

La Figure 1.3 représente un automate $\mathcal{A}$ permettant de vérifier si un entier, écrit en base 2, est pair. En effet, lors de l'analyse du mot, si nous atteignons un état final à partir de l'état initial, alors le mot est accepté. Si nous arrivons dans un état non-final, le mot est alors rejeté.

Pour pouvoir utiliser de tels modèles dans le cadre de la distance d'édition, nous utilisons une extension des automates, appelée transducteur, qui permet de transformer une chaîne d'entrée construite sur un alphabet Σ en une chaîne de sortie construite sur un alphabet Γ.

Définition 1.10 (Transduction) *Une transduction de $\Sigma^\star$ en $\Gamma^\star$ est une relation inclue dans $\Sigma^\star \times \Gamma^\star$.*

Les transducteurs rationnels sont un premier type de machine pour reconnaître des transductions.

Définition 1.11 (Transducteur rationnel) *Un transducteur rationnel est un 5-uplet $\{\mathcal{Q}, \Sigma, \Gamma, q_0, T\}$:*

- *$\mathcal{Q}$ est un ensemble fini d'états,*
- *Σ, Γ sont respectivement l'alphabet d'entrée et l'alphabet de sortie,*
- *$q_0 \in \mathcal{Q}$ est l'unique état initial,*
- *$T \subset (\mathcal{Q} \times \Sigma^\star \times \Gamma^\star \times \mathcal{Q})$ est un ensemble fini de transitions.*

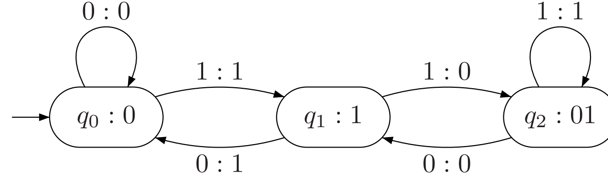


Fig. 1.4 – Représentation graphique d'un transducteur $\mathcal{T} = \langle \mathcal{Q}, \Sigma, \Gamma, q_0, T, \sigma \rangle$, représentant la multiplication d'un entier par 3 en base 2 avec $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $\Gamma = \{0,1\}$, $T = \{(q_0,0,0,q_0),(q_0,1,1,q_1),(q_1,0,1,q_0),(q_1,1,0,q_2),(q_2,0,0,q_1),(q_2,1,1,q_2)\}$ et $\sigma = \{(q_0,0),(q_1,1),(q_2,01)\}$. La transition $(q_0,1,1,q_1)$ signifie qu'à partir de l'état q_0, la lecture d'un 1 en entrée génère un 1 en sortie et nous atteignons l'état q_1. Par exemple, le nombre 5 dont la représentation binaire est 101 en entrée génère la chaîne 1111 de sortie qui est la représentation du nombre 15.

Ces machines s'utilisent comme des automates et reconnaissent des transductions. Notons que l'ensemble des états sont terminaux et donc une transduction peut s'arrêter dans tous les états.

Nous pouvons définir un transducteur permettant la lecture déterministe, *i.e.* séquentielle, des symboles d'entrée.

Définition 1.12 (Transducteur sous-séquentiel) *Un transducteur sous-séquentiel est un transducteur rationnel tel que $T \subset (\mathcal{Q} \times \Sigma \times \Gamma^\star \times \mathcal{Q})$ et $\forall (q,a,u,q'),(q,a,v,q'') \in T \Rightarrow u = v \wedge q' = q''$ et tel qu'il existe une fonction totale $\sigma : \mathcal{Q} \to \Gamma^\star$.*

La Figure 1.4 représente graphiquement un transducteur. Celui-ci permet la multiplication d'un entier par 3 en base 2.

De tels transducteurs permettent la traduction d'un mot de la gauche vers la droite. De plus, sachant que tous les états sont terminaux, la lecture déterministe assure que si une traduction existe alors elle est unique. Notons que si le symbole vide λ est inclus dans les alphabets Σ et Γ, nous pouvons simuler un processus d'édition, au sens de la distance d'édition.

Dans le cadre de nos travaux, nous souhaitons générer des mots suivant une distribution de probabilité. Si dans le cadre non probabiliste, les automates et les transducteurs sont utilisés pour l'analyse et la reconnaissance de mots, ils permettent en plus d'être utilisés comme générateur de mots dans un cadre stochastique.

Définition 1.13 (Automate probabiliste) *Un automate probabiliste ou stochastique à états fini est défini par un 5-uplet $\mathcal{A} = \langle \mathcal{Q}, \Sigma, T_P, I_P, F_P \rangle$, où :*
- *$\mathcal{Q}$ est un ensemble fini d'états,*
- *Σ est l'alphabet de l'automate,*
- *$T_P : \mathcal{Q} \times \Sigma \times \mathcal{Q} \to \mathbb{Q} \cap [0,1]$ est la fonction de probabilité associée aux transitions,*
- *$I_P : \mathcal{Q} \to \mathbb{Q} \cap [0,1]$ est la fonction de probabilité initiale,*

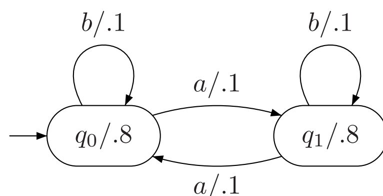


Fig. 1.5 – Représentation graphique d'un automate stochastique. Par soucis de lisibilité, l'unique probabilité initiale (= 1.0) est marquée par une flèche. La transition de l'état 0 à l'état 1 est possible par la lettre a avec une probabilité de .1.

– $F_P : \mathcal{Q} \rightarrow \mathbb{Q} \cap [0,1]$ *est la fonction de probabilité finale.*
De plus T_P*,* I_p *et* F_p *satisfont les contraintes suivantes :*

$$\sum_{q \in \mathcal{Q}} I_p(q) = 1 \tag{1.15}$$

et $\forall q \in \mathcal{Q}$ *:*

$$F_p(q) + \sum_{q' \in \mathcal{Q}, a \in \Sigma} T_p(q, a, q') = 1. \tag{1.16}$$

Contrairement aux automates non-probabilistes, nous n'avons pas d'états dits acceptants ou rejetants. Ils sont remplacés par une probabilité de terminaison. Dans ce nouveau cadre, nous pouvons dire que les automates sont génératifs : à chaque mot généré, nous pouvons associer une probabilité. Si l'automate vérifie les Équations 1.15 et 1.16, alors nous définissons une distribution de probabilité sur l'ensemble des mots de l'alphabet. Suivant le même modèle, nous pouvons définir des transducteurs stochastiques. Étant donné l'automate stochastique de la Figure 1.5, la probabilité du mot *baab* est de :

$$\begin{aligned} Pr(baab) &= I_p(0) * T_p(0,b,0) * T_p(0,a,1) * T_p(1,a,0) * T_p(0,b,0) * F_p(0) \\ &= 1.0 * 0.1 * 0.1 * 0.1 * 0.1 * 0.8 \\ &= .00008. \end{aligned} \tag{1.17}$$

1.4.2 Pair-HMM

Les Pair-HMMs [DEKM98] sont des machines à états basées sur les modèles de Markov [Rab90]. Les modèles de Markov sont des modèles statistiques dans lesquels la génération du symbole suivant dépend du symbole précédent.

Un modèle de Markov caché (HMM) généralise le modèle de Markov observable. Il produit une séquence de deux suites de variables aléatoires : l'une cachée et l'autre observable.

- la suite cachée correspond à la suite des états $q_1, q_2, \ldots, q_n$, et
- la suite observable correspond à la séquence des observations $O_1, O_2, \ldots, O_n$ où les $O_i \in \Sigma$ sont les lettres de la séquence.

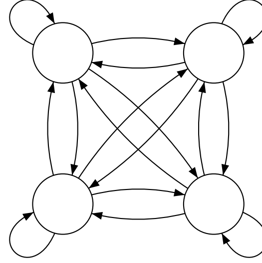


Fig. 1.6 – Représentation graphique d'un modèle de Markov caché pour la génération d'ADN. Dans cette machine, l'ensemble des états peuvent être initiaux. Tous les symboles de l'alphabet *ACGT* peuvent être émis dans chaque état. Les probabilités de transition sont omises par soucis de lisibilité.

Définition 1.14 (Modèle de Markov caché) *Un modèle ou automate de Markov à états cachés est un 5-uplet $\mathcal{H} = \langle \mathcal{Q}, \Sigma, \Pi, A, B \rangle$, avec :*
- *$q_i \in \mathcal{Q}$ l'état i,*
- *$x \in \Sigma$ un symbole de l'alphabet,*
- *$\pi_{q_i} \in \Pi$ la probabilité que q_i soit l'état initial,*
- *$a_{q_i q_j} \in A$ la probabilité de la transition q_i vers q_j,*
- *$b_{q_i}(x) \in B$ la probabilité d'émettre le symbole x étant dans l'état q_i.*

De plus, l'automate $\mathcal{H}$ doit respecter les contraintes :

- *la somme des probabilités initiales est égale à 1 :*

$$\sum_{q_i \in \mathcal{Q}} \pi_{q_i} = 1,$$

- *la somme des probabilités des transitions partant d'un état q_i est égale à 1 :*

$$\forall q_i \in \mathcal{Q}, \sum_{q_j \in \mathcal{Q}} a_{q_i q_j} = 1,$$

- *la somme des probabilités des émissions partant d'un état q_i est égale à 1 :*

$$\forall q_i \in \mathcal{Q}, \sum_{x \in \Sigma} b_{q_i}(x) = 1.$$

La Figure 1.6 représente une chaîne de Markov pour la génération d'ADN dans laquelle nous pouvons observer la génération de toutes les lettres par état et une probabilité est associée à chaque transition. De plus, une probabilité est associée à chaque état pour indiquer les probabilités initiales. La propriété clé d'un modèle de Markov est que la probabilité d'un nouveau symbole x_i dépend seulement de la probabilité du symbole précédent x_{i-1}.

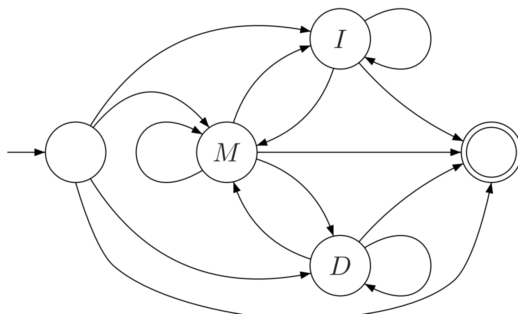


Fig. 1.7 – Représentation graphique d'un pair-HMM utilisé pour l'alignement de séquences. L'état M est un état de substitution, l'état D de délétion et l'état I d'insertion. L'état de gauche est l'état initial et l'état de droite, l'état final.

Dans [DDE05], Dupont *et al.* montrent que les HMMs et les automates probabilistes sont équivalents. Ils proposent également un algorithme permettant de déduire un HMM à partir d'un automate probabiliste.

Une extension des HMMs, les pair-HMMs, est de proposer la possibilité de gérer simultanément deux mots : la chaîne d'entrée et la chaîne de sortie. Chaque état du pair-HMM peut soit générer une nouvelle lettre sur chacun des mots, soit seulement sur le mot d'entrée, ou seulement sur celui de sortie.

Définition 1.15 (Pair-HMM) *Un pair-HMM produit une séquence de deux suites de variables aléatoires : l'une cachée et l'autre observable.*
- *la suite cachée correspond à la suite des états* $q_1, q_2, \ldots, q_n$, *et*
- *la suite observable correspond à la séquence des observations* $O_1, O_2, \ldots, O_n$ *où les* $O_i \in \Sigma \times \Sigma$ *sont des paires de lettres.*

La Figure 1.7 est le modèle proposé par [DEKM98] pour l'alignement de séquences ADN. Dans ce modèle, l'état M permet l'émission d'une paire de symboles, l'état D permet l'émission d'un symbole sur la chaîne d'entrée alors que l'état I permet l'émission d'un symbole sur la chaîne de sortie.

1.5 Conclusion

Dans ce chapitre, nous avons introduit les différents concepts nécessaires à la compréhension de la suite de ce manuscrit, qui se positionne dans le cadre de l'apprentissage à partir de données structurées (chaînes, arbres). L'objectif que nous cherchons à atteindre est d'apprendre les paramètres d'une distance d'édition. Celle-ci, dans sa version standard, correspond au nombre minimal de transformations élémentaires (insertion, délétion, substitution) nécessaires pour transformer une chaîne en une autre. Dans le contexte de ce manuscrit, nous représentons cette distance d'édition sous une forme stochastique dans laquelle chaque opération d'édition est représentée par une probabilité. L'objectif

est donc d'apprendre une distribution sur ces opérations. Pour mener à bien cet objectif, nous proposons d'utiliser des machines à états.

2 Apprentissage de similarités d'édition

Résumé

Dans ce chapitre, nous dressons un état des lieux des différentes méthodes de la littérature ayant pour objectif d'apprendre les paramètres de la distance d'édition. Après une rapide présentation des techniques non probabilistes, nous introduisons l'algorithme *Expectation-Maximisation* EM, à l'origine de plusieurs algorithmes d'apprentissage de similarités d'édition stochastiques. Comme nous le verrons plus tard dans les Chapitres 3 et 4, EM se retrouvera également au cœur des contributions de cette thèse.

2.1 Introduction

L'apprentissage de fonctions de distance et de similarité a donné lieu à de nombreux travaux ces dernières années. En général, l'objectif est de tenir compte de connaissances du domaine (principalement sous la forme de contraintes de type *must-link* entre deux données – les données sont de la même classe – ou *cannot-link* – les données sont de classes différentes) pour améliorer les algorithmes de clustering ou de classification.

La très grande majorité des approches proposées dans la littérature se situent dans le contexte où les données sont représentées dans des espaces vectoriels. L'objectif consiste alors souvent à apprendre des matrices semi-définies positives pouvant être utilisées par exemple dans la distance de Mahalanobis (voir [XNZ08] et [Yan07] pour un état de l'art). Ces matrices apprises modélisent de manière implicite une modification de l'espace des exemples par une transformation linéaire permettant de rapprocher des paires d'exemples jugées similaires, tout en éloignant d'autres jugées non similaires. Lorsque les données sont représentées sous forme structurée (chaînes, arbres ou graphes), comme c'est le cas dans le contexte de cette thèse, il n'est pas possible d'adapter les techniques précédentes sans une perte d'informations. En effet, n'ayant pas de représentation vectorielle naturelle des exemples d'apprentissage, il n'existe pas de projection de l'espace d'entrée par une transformation linéaire. Pour contourner ce problème – autrement qu'en choisissant une représentation vectorielle des données arbitraire, entraînant généralement une perte d'information – une solution consiste à apprendre des fonctions de similarités reposant sur la distance d'édition. Autrement dit, ceci revient à inférer les coûts des opérations d'édition selon un paradigme d'apprentissage donné. Dans ce contexte, l'objectif de l'apprentissage peut être vu comme la recherche d'une projection des données dans l'espace des scripts

d'édition. L'ensemble des scripts possibles, pouvant être extraits d'un échantillon, fournit alors des informations permettant d'optimiser les coûts d'édition.

Dans ce chapitre, nous présentons les principales méthodes (probabilistes et non probabilistes) permettant d'apprendre des similarités d'édition entre données structurées.

2.2 Méthodes non probabilistes

Les méthodes d'inférence non probabilistes ayant été exploitées dans la littérature pour l'apprentissage des paramètres d'édition peuvent être séparées en deux catégories. Soit l'optimisation des paramètres est liée à une fonction objectif connue, dérivable et calculable en temps raisonnable. Dans ce cas, il est possible de calculer une solution analytique au problème en utilisant par exemple des techniques reposant sur une descente de gradient. Soit la solution est inconnue ou difficile à calculer. Elle requiert alors l'utilisation d'heuristiques afin d'être approximée. Comme nous allons le voir, les algorithmes génétiques sont une solution à ce problème et ont été exploités pour apprendre des distances d'édition.

2.2.1 Méthodes basées sur la descente de gradient

La descente de gradient (DG) est un algorithme d'optimisation de premier ordre, qui consiste à trouver le minimum d'une fonction cible par étapes successives et proportionnelles à l'opposé du gradient de la fonction au point courant.

La DG est basée sur l'observation suivante : si une fonction $F(x)$ à valeurs dans $\mathbb{R}$ est définie et dérivable en un point a, alors $F(x)$ décroît le plus rapidement en une direction opposée à celle du gradient de F en a : $-\nabla F(a)$. Ainsi, si nous définissons

$$b = a - \gamma \nabla F(a)$$

avec $\gamma > 0$ suffisamment petit, alors nous avons $F(b) \leq F(a)$.

Dans ce contexte, Hourai *et al.*, dans [HAA04], proposent une méthode d'optimisation pour l'algorithme de Smith-Waterman dans le cas de données biologiques. À partir d'exemples positifs (gènes orthologues) et négatifs (gènes non-orthologues), ils utilisent un algorithme de descente de gradient pour optimiser l'ensemble des coûts de substitution, ainsi que la valeur correspondant à la création d'une suite de délétions ou d'insertions. La fonction objectif à minimiser est le taux d'erreur ϵ, défini comme le rapport de la somme des faux-positifs et des faux-négatifs sur la somme des positifs et des négatifs :

$$\epsilon = \frac{\#FP + \#FN}{\#P + \#N}.$$

Les auteurs appliquent leur méthode sur la base de données biologiques COG[1] et montrent que la classification finale est meilleure en ayant appris la matrice de substitutions qu'en utilisant les matrices habituellement considérées dans la communauté bioinformatique, comme les matrices de type BLOSUM [HH92] ou PAM [DSO79].

[1]Cluster of Orthologous Group database [TGNK00] : http://www.ncbi.nlm.nih.gov/COG/.

Dans [SVA06], les auteurs présentent un noyau d'alignements de chaînes utilisé pour la détection d'homologie entre séquences biologiques. Ce noyau est calculé à partir d'une matrice de substitution entre acides aminés. Dans ce contexte, une similarité s entre deux chaînes est calculée à partir de l'ensemble des scripts d'édition E applicables entre les deux chaînes. Les opérations d'insertion et de délétion sont traitées par une notion de "gap" (ou trou). Chaque gap étant modélisé par un coût d'ouverture g_o et un coût d'extension g_e (*i.e.* le nombre de caractères insérés ou supprimés). Le calcul de la similarité est alors défini comme suit :

$$s(X, Y, E) = \sum_{a,b} \eta_{a,b}(X, Y, E) \cdot S(a, b) - \eta_o(X, Y, E) \cdot g_o - \eta_e(X, Y, E) \cdot g_e, \qquad (2.1)$$

avec $\eta_{a,b}(X, Y, E)$ le nombre de fois qu'un symbole a est aligné avec un symbole b, $S(a, b)$ est le coût de substitution (à apprendre) entre les lettres a et b, g_o et g_e sont les paramètres, respectivement, d'ouverture et d'extension de "gaps" et leur nombre d'occurrences associé $\eta_o(X, Y, E)$ et $\eta_e(X, Y, E)$.

Les auteurs utilisent une DG pour apprendre les paramètres S, g_o et g_e de l'Équation 2.1. La fonction objectif optimisée est le Z-score :

$$Z = \frac{s - \mu}{\sigma},$$

avec s la similarité entre une chaîne et un candidat homologue de l'échantillon (cette paire représentant donc un exemple positif), et μ et σ la moyenne et la variance des similarités pour l'ensemble des séquences requêtes *versus* toutes les séquences non-homologues de l'échantillon (correspondant donc aux exemples négatifs). Le but de la fonction objectif est donc de trouver les paramètres qui permettront de rapprocher les chaînes homologues tout en éloignant les chaînes non homologues. Leurs expérimentations ont aussi porté sur la base COG et montrent des améliorations significatives.

Malgré son intérêt pour la recherche de solution analytique, la descente de gradient présente certains inconvénients :
- Le nombre d'itérations nécessaires pour la convergence de l'algorithme est parfois élevé.
- La recherche du pas γ optimal est délicate.
- Une fonction objectif dérivable est nécessaire.

2.2.2 Méthodes basées sur les algorithmes génétiques

Les algorithmes génétiques (AG) sont des méthodes de recherche heuristiques s'inspirant du processus d'évolution naturelle. Ces heuristiques sont utilisées pour trouver des solutions approchées à des problèmes d'optimisation difficiles.

Pour pouvoir utiliser un algorithme génétique, il est nécessaire de définir :
- une représentation génétique, appelée chromosome, d'une solution au problème étudié,

– une fonction d'évaluation (fonction *fitness*) pour mesurer la pertinence d'une solution,
– des opérateurs de sélection, de croisement et de mutation pour générer de nouvelles solutions à partir de solution existantes.

Un algorithme générique possible se présente de la manière suivante :

1. Génération aléatoire d'une population P de n solutions potentielles.
2. Répéter tant que la condition d'arrêt n'est pas satisfaite :
   (a) **Évaluation** des individus de P.
   (b) Génération d'une nouvelle population P'.
      i. **Sélection**[2] de deux individus x et y de P.
      ii. **Croisement** de x et y pour obtenir un nouvel exemple z.
      iii. **Mutation** de z.
      iv. Ajout de z à P'.
   (c) P' est la nouvelle population P.

Dans [PGH98], les auteurs utilisent un AG pour apprendre les paramètres de la distance d'édition. Dans cette approche, le problème d'optimisation vise à minimiser la distance entre les exemples de même classe et à maximiser la distance entre ceux de classes différentes. Une solution y est représentée sous la forme d'une concaténation de l'ensemble des poids de chacune des opérations d'édition. L'opération de mutation s'applique indépendamment pour chaque opération d'édition et consiste simplement en un changement de poids, alors que l'opération de croisement permet d'obtenir une nouvelle matrice de poids en utilisant deux solutions précédentes. Les expérimentations ont été menées sur des données artificielles avec des résultats significatifs et sur une application de reconnaissance de chiffres manuscrits, mais cette fois avec des gains de performance plutôt limités. Les auteurs concluent que l'optimisation d'une seule matrice de coûts pour l'ensemble des 10 chiffres ne permettrait pas un gain caractéristique[3].

Dans [GAdM05], l'évaluation de chaque chromosome est calculée expérimentalement en apprenant et testant un classifieur utilisant les paramètres courants. La fonction de *fitness* mesurée est le rapport entre les faux positifs et les vrais positifs. Les auteurs utilisent leur algorithme pour la classification de morceaux de musique représentés sous forme arborescente (la profondeur de l'arbre montre la longueur de la note et le symbole de chaque nœud représente la hauteur de la note). Les auteurs ont pris part à une compétition[4] et ont obtenu les meilleurs résultats. Notons que les autres participants ont utilisé une représentation séquentielle puis ont appliqué des méthodes à bases de n-grammes ou de distance d'édition.

De manière générale, les limites de l'utilisation des algorithmes génétiques sont les suivantes :

[2]Différentes stratégies de sélection peuvent s'appliquer, allant du choix des deux meilleures solutions au sens de la fonction de *fitness*, jusqu'à un tirage aléatoire selon une certaine distribution.
[3]Ceci a été remis en cause par Oncina et Sebban [OS06] avec une approche stochastique (voir Section 2.3.2).
[4]MIREX : http://www.music-ir.org/mirex/wiki/MIREX_HOME.

- Le choix de la fonction d'évaluation peut entraîner la convergence de l'algorithme dans des optima locaux si elle est trop simple ou à une explosion algorithmique si elle est trop complexe.
- Le critère de terminaison de l'algorithme peut être difficile à évaluer, ne permettant donc pas de savoir si la solution est optimale.
- Le choix des paramètres (taille de la population, taux de croisements et taux de mutations) influe fortement sur la qualité des résultats obtenus.

2.2.3 Autres approches d'inférence non probabilistes

Pour terminer cette partie sur les méthodes non probabilistes, notons également que d'autres techniques pour l'optimisation des paramètres de la distance d'édition ont été utilisées. Sans entrer dans les détails ici, nous pouvons citer :

- Une approche basée sur les SVMs [Joa03] dans le cadre de la distance de Smith-Waterman entre chaînes.
- Une méthode d'optimisation par essaims particulaires [Meh09] pour la distance d'édition entre arbres.
- Une technique utilisant des cartes auto-adaptatives [NB05] dans le cadre de la distance d'édition entre graphes.

Une remarque générale doit être faite ici. Pour l'ensemble des méthodes présentées précédemment, il est nécessaire de disposer d'exemples positifs et négatifs. Néanmoins, pour de nombreux problèmes réels, l'accès à des exemples de classes différentes peut être compliqué. C'est le cas par exemple dans des applications en langage naturel (*e.g.* en correction grammaticale ou orthographique), dans certains domaines de la biologie moléculaire (*e.g.* apprendre des phénomènes de mutations). Pour résoudre ce problème, il est donc nécessaire d'avoir recours à des techniques utilisant des exemples positifs seulement. Dans la section suivante, nous présentons différentes approches adaptées à cette situation et faisant appel à l'algorithme *Expectation-Maximisation*.

2.3 Méthodes probabilistes

2.3.1 L'algorithme EM

Nous introduisons ici l'algorithme *Expectation-Maximisation* EM [Har58, DLR77] qui est à l'origine des travaux référencés dans cette section, et qui sera également fortement utilisé dans les contributions de cette thèse aux Chapitres 3 et 4. Le lecteur souhaitant disposer de plus amples informations pourra lire les tutoriaux suivants [Col97, Del02, Bor04].

L'algorithme EM est un algorithme permettant d'optimiser les paramètres d'un modèle probabiliste en maximisant la vraisemblance d'un échantillon d'apprentissage. L'algorithme EM comporte deux étapes :

- une étape E d'évaluation de l'espérance mathématique de la log vraisemblance en tenant compte des paramètres courants,

– une étape de maximisation M qui estime le maximum de vraisemblance des paramètres en maximisant les espérances trouvées durant l'étape E.

L'étape M permet donc d'obtenir de nouveaux paramètres. Le même processus est alors itéré jusqu'à convergence.

Plus formellement, soient $\mathbf{X}$ un ensemble de données observables, $\mathbf{Z}$ un ensemble de données cachées ou incomplètes et Θ un vecteur de paramètres à estimer. La vraisemblance de l'échantillon $\mathbf{X}$ selon les paramètres Θ du modèle est donnée par :

$$L(\mathbf{X}|\Theta) = \prod_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\Theta).$$

Nous considérons généralement plutôt la log vraisemblance $L(\Theta) = \log L(\mathbf{X}, \Theta)$, nous permettant ensuite de bénéficier des propriétés de l'inégalité de Jensen (propriétés sur les fonctions convexes et concaves). Lors de l'étape E, l'algorithme revient à calculer l'espérance de la fonction log vraisemblance, selon $\mathbf{Z}$ et étant donnés $\mathbf{X}$ et les paramètres courants Θ^t.

$$Q(\Theta|\Theta^t) = E_{\mathbf{Z}|\mathbf{X},\Theta^t}[\log L(\mathbf{X}, \mathbf{Z}|\Theta)].$$

L'étape M revient alors à rechercher les nouveaux paramètres maximisant Q :

$$\Theta^{t+1} = \underset{\Theta}{\operatorname{argmax}}\ Q(\Theta|\Theta^t).$$

Une propriété cruciale de l'algorithme EM vient de la garantie de l'augmentation de la vraisemblance $L(\Theta)$ à chaque itération. En effet, en définissant l'application $\Theta^t \rightarrow \Theta^{t+1}$ avec $\Theta^{t+1} = \underset{\Theta'}{\operatorname{argmax}}\ Q(\Theta', \Theta^t)$ il est possible de prouver que $L(\Theta^{t+1}) \geq L(\Theta^t)$ (voir [Bor04] pour plus de détails).

Exemple. *Soient deux pièces C_1 et C_2. Nous procédons à un ensemble de n tirages simultanés de ces deux pièces et nous notons les résultats dans un tableau T (0 pour pile, 1 pour face). Supposons que nous n'ayons pas noté le résultat du 4$^{\text{ème}}$ lancer pour la pièce C_2. Cette information correspond donc à une donnée incomplète ou cachée que nous souhaitons estimer.*

	n	1	2	3	4	5	6	7	8
T	C_1	0	0	0	0	0	1	1	1
	C_2	0	0	0	?	1	0	1	1

L'objectif est d'estimer les probabilités de la distribution jointe des paramètres $\Theta = \{p(00), p(01), p(10), p(11)\}$ où $p(ij)$ est la probabilité d'obtenir les résultats i pour C_1 et j pour C_2, $i, j \in \{0, 1\}$.

Appliquons l'algorithme EM pour résoudre ce problème. L'étape E estime le nombre de fois qu'un événement $c_1c_2 \in \{00, 01, 10, 11\}$ est effectif, i.e.

$$E[c_1c_2|T, \Theta] = \#\{C_1 = c_1 \cap C_2 = c_2\}\ \forall\ c_1, c_2 \in \{0, 1\}.$$

L'étape M permet d'obtenir une distribution jointe de probabilité, en effectuant la normalisation suivante :

$$\Theta_{c_1c_2} = \frac{E[c_1c_2|T, \Theta]}{\sum_{c_1c_2} E[c_1c_2|T, \Theta]}.$$

Soit $\Theta_0 = \{^2/_8, ^2/_8, ^2/_8, ^2/_8\}$, *l'ensemble des paramètres initiaux uniformément distribués, dans ce cas.*

Étant donné Θ_0, T_4 *peut prendre la valeur* (00) *avec une probabilité de* .5 *et* (01) *avec une probabilité de* .5*. En appliquant les étapes E et M décrites précédemment, nous obtenons* $\Theta_1 = \{^{3.5}/_8, ^{1.5}/_8, ^1/_8, ^2/_8\}$ *à la première itération.*

En itérant le processus, nous obtenons pour les étapes suivantes :

	$P(00)$	$P(01)$	$p(10)$	$p(11)$	$L(\Theta)$
Θ_2	$^{3.7}/_8$	$^{1.3}/_8$	$^1/_8$	$^2/_8$	-8.98243
Θ_3	$^{3.74}/_8$	$^{1.26}/_8$	$^1/_8$	$^2/_8$	-8.98142
Θ_4	$^{3.748}/_8$	$^{1.252}/_8$	$^1/_8$	$^2/_8$	-8.98138
Θ_5	$^{3.7496}/_8$	$^{1.2504}/_8$	$^1/_8$	$^2/_8$	-8.98138
Θ_6	$^{3.74992}/_8$	$^{1.25008}/_8$	$^1/_8$	$^2/_8$	-8.98138
Θ_7	$^{3.749984}/_8$	$^{1.250016}/_8$	$^1/_8$	$^2/_8$	-8.98138
Θ_8	$^{3.7499967}/_8$	$^{1.2500033}/_8$	$^1/_8$	$^2/_8$	-8.98138
Θ_9	$^{3.74999935}/_8$	$^{1.25000065}/_8$	$^1/_8$	$^2/_8$	-8.98138

Étant donné Y*, nous pouvons voir que l'algorithme converge au bout de quelques itérations (tout en assurant que la vraisemblance continue d'augmenter). Nous obtenons finalement la distribution jointe suivante :*

$$\Theta = \{0.46875, 0.15625, 0.125, 0.25\}.$$

◦

L'objectif des sections suivantes est de présenter une adaptation de l'algorithme EM au contexte de l'apprentissage des paramètres d'édition. Les données observables $\mathbf{X}$ sont alors constituées de paires positives d'apprentissage, les paramètres Θ sont les probabilités des opérations d'édition et les données incomplètes $\mathbf{Z}$ sont les scripts d'édition entre deux exemples de l'échantillon. Adapter l'algorithme EM au contexte des distances d'édition peut revenir alors à apprendre les paramètres d'une machine probabiliste à états. Les modèles proposés dans la littérature se différencient de par le fait qu'ils utilisent des machines soit à un seul état (ou *memoryless*) soit à plusieurs états (tenant compte alors du contexte dans lequel s'opèrent les opérations d'édition).

2.3.2 Machine d'édition à un état

Les premiers modèles que nous présentons ici prennent la forme de transducteurs stochastiques (voir Section 1.4.1 pour une définition formelle) à un seul état modélisant une similarité d'édition comme un processus stochastique basé sur la définition d'une distribution statistique sur l'ensemble des scripts d'édition. Ces modèles peuvent donc être utilisés pour calculer la probabilité d'édition d'un mot en un autre en analysant la paire de chaînes. Dans la suite du manuscrit, nous nommerons ces transducteurs à un état des modèles *memoryless*, du fait que la probabilité d'une opération d'édition ne dépend pas des opérations précédentes. L'apprentissage de ces modèles est donc plus simple tout en permettant néanmoins d'appréhender divers types d'applications.

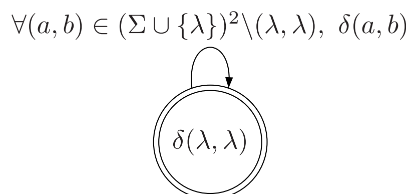


Fig. 2.1 – Transducteur utilisé dans [RY98].

Une première contribution dans le cas de l'apprentissage de similarité entre chaînes, a été proposée dans [RY98]. Les auteurs y présentent un modèle génératif sous la forme d'un transducteur à un état dont les transitions représentent les opérations d'édition (voir Fig. 2.1). Ils utilisent l'algorithme EM pour apprendre les paramètres du transducteur en maximisant la vraisemblance d'un échantillon d'apprentissage composé d'un ensemble de paires (entrée, sortie) de mots jugés similaires.

Pour calculer les espérances de chaque opération d'édition (étape E), ils utilisent deux fonctions auxiliaires, appelées *forward* et *backward*. La fonction *forward* permet de calculer dynamiquement la probabilité d'un couple de préfixes de mots, la fonction *backward* quant à elle calcule la probabilité d'un couple de suffixes. Le calcul de ces deux fonctions est basé sur l'algorithme de programmation dynamique de la distance d'édition standard.

Pour l'étape M, ils définissent une distribution jointe des opérations d'édition satisfaisant la contrainte suivante :

$$\sum_{(l,l')\in(\Sigma\cup\{\lambda\})^2\backslash\{\lambda,\lambda\}} \delta(l,l') + \delta(\#) = 1 \text{ avec } \delta(l,l') \geq 0 \text{ et } \delta(\#) > 0. \tag{2.2}$$

où $\delta(\#) > 0$ représente la probabilité de fin de script. En d'autres termes, la contrainte signifie que nous devons au moins disposer d'une paire d'apprentissage.

Les auteurs ont illustré leur approche en l'appliquant au problème de l'apprentissage de la prononciation des mots de la langue anglaise. La comparaison avec une distance de Levenshtein, non apprise, montre l'intérêt et l'efficacité de leur approche.

Une extension de ces travaux a été proposée dans [OS06] pour contourner les inconvénients de biais statistique lié au modèle génératif de Ristad & Yianilos [RY98]. En effet, ce modèle requiert que les exemples doivent suivre la distribution marginale des chaînes d'entrée pour converger vers la distribution cible. Les auteurs proposent d'utiliser les conditions de normalisation suivantes pour définir un modèle discriminatif.

$$\forall l \in \Sigma, \sum_{l'\in\Sigma\cup\{\lambda\}} \delta(l'|l) + \sum_{l'\in\Sigma} \delta(l'|\lambda) = 1 \tag{2.3}$$

$$\sum_{l'\in\Sigma} \delta(l'|\lambda) + \delta(\#) = 1. \tag{2.4}$$

Ces équations suggèrent que pour éviter le biais des modèles génératifs qui ne déduisent qu'*a posteriori* les probabilités conditionnelles nécessaires à une utilisation dans une

tâche de classification, il est souhaitable de maximiser directement à chaque itération de EM ces probabilités conditionnelles. Les auteurs montrent l'intérêt de leur approche dans une tâche de reconnaissance de caractères manuscrits (codés sous forme de chaînes en utilisant les codes de Freeman [Fre61]), en comparaison avec la distance apprise dans [RY98] et la distance de Levenshtein classique.

Une comparaison des modèles génératifs et discriminatifs peut être trouvée dans [BT04]. De manière générale, les modèles génératifs ont une variance plus faible mais un biais important alors que les modèles discriminatifs ont un biais moindre mais une plus grande variance. Ces derniers doivent être donc privilégiés si nous disposons de beaucoup de données d'apprentissage. Ce sont donc les contraintes liées à l'application traitée qui imposeront le choix de tel ou tel modèle. Nous trouverons dans le chapitre suivant une expérimentation portant sur la comparaison de ces deux approches.

Malgré des contraintes algorithmiques fortes, notons que nous trouvons dans la littérature des modèles s'appliquant à la distance d'édition entre graphes [GXTL10]. Traditionnellement, les opérations applicables aux graphes sont l'insertion d'un nœud, la délétion d'un nœud, la substitution de deux nœuds, l'insertion d'un arc, la délétion d'un arc et la substitution de deux arcs. Ces opérations permettent de modéliser les différentes distorsions possibles entre deux graphes. Par soucis de simplification, nous pouvons admettre que le coût de chaque opération dépend seulement du label associé au nœud ou à l'arête. Le calcul de la distance d'édition entre graphes est alors très similaire au calcul de la distance d'édition entre chaînes, *i.e.* il correspond à la valeur du script d'édition (composé de six types d'opération) de coût minimal. Le calcul de la distance étant NP-difficile en le nombre de nœuds du graphe, l'utilisation de cette distance est limitée à des graphes de petite taille et l'emploi d'heuristiques est nécessaire pour appréhender des graphes plus importants.

Malgré cette contrainte algorithmique, dans [NB07], les auteurs proposent une version stochastique de la DE entre graphes sur le même modèle que celui utilisé dans [RY98] dans le cadre des chaînes. Pour approximer la densité de chacune des opérations d'édition, les auteurs suggèrent d'insérer un mélange de densités gaussiennes. Ils proposent d'en utiliser une par type d'opération : délétion de nœuds, insertion de nœud, substitution de nœud et trois autres pour les opérations sur les arêtes. Les auteurs ont expérimenté leur approche sur une tâche de *clustering* de caractères dessinés uniquement avec des lignes droites (lettres `A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z`). Ils montrent expérimentalement que l'apprentissage permet d'obtenir un *clustering* de meilleure qualité.

Comme nous l'avons déjà mentionné, les modèles *memoryless* associent à chaque opération d'édition une unique probabilité. Cependant, pour certains problèmes, il semble intéressant de pouvoir définir la probabilité d'une opération en fonction d'un contexte donné. Par exemple, supposons que nous voulons apprendre un modèle de correction pour les mots du langage $a^\star b^\star$: l'ensemble des mots qui sont composés d'une suite de a, suivie d'une suite de b. Pour ce faire, le modèle devra être capable de modéliser le fait que dans la première partie du mot (la suite de a), les opérations (a, a) et (b, a) devront avoir une forte probabilité tandis que les opérations (b, b) et (a, b) seront impossibles. Dans la seconde partie du mot, les probabilités devront s'inverser. Ce petit exemple montre que

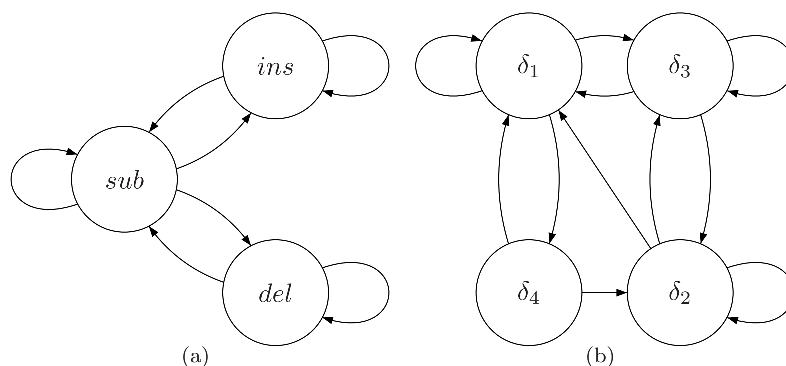


Fig. 2.2 – (a) Un pair-HMM *memoryless* pour l'alignement local. (b) Un pair-HMM *non-memoryless* pour la distance d'édition, à chaque état une matrice de probabilités d'édition δ_i est assignée.

l'utilisation d'un modèle à un état est insuffisant pour traiter certains types de problèmes, ce qui justifie l'intérêt de considérer plusieurs états pour représenter différents contextes.

2.3.3 Machine d'édition à plusieurs états

Dans cette section, nous présentons des modèles permettant de définir des probabilités d'opérations d'édition conditionnellement à un contexte à l'aide de machines à plusieurs états.

Une première approche possible est de s'inspirer des pair-HMMs introduits en bio-informatique par [DEKM98] pour l'alignement de chaînes d'ADN et présentée dans le Chapitre 1. Il est possible de modifier l'algorithme d'apprentissage des HMMs (reposant également sur EM) pour l'adapter à des paires de chaînes.

Le modèle génératif ainsi appris, présenté dans la Figure 2.2(a), est utilisé pour calculer des alignements locaux entre chaînes d'ADN. Dans ce contexte, la machine est constituée de trois états correspondant à trois types d'opérations différentes : un pour les substitutions *sub*, un pour les délétions *del* et un pour les insertions *ins*. Quoique disposant de 3 états, ce modèle original reste néanmoins *memoryless*, mais peut être étendu à des modèles *non-memoryless*, si l'ensemble des opérations d'édition (substitution, insertion et délétion) est alors possible dans chaque état de la machine (voir Fig. 2.2(b)). Soit $\mathcal{Q}$ l'ensemble d'états associé à un pair-HMM, pour définir une distribution de probabilité sur les paires de chaînes, ce pair-HMM doit satisfaire les propriétés suivantes[5] :

$$\forall q \in \mathcal{Q}, \sum_{(l,l')\in(\Sigma\cup\{\lambda\})^2\setminus(\lambda,\lambda)} \delta_q(l,l') + \delta_q(\#) = 1, \tag{2.5}$$

et

$$\forall q \in \mathcal{Q}, \sum_{q'\in\mathcal{Q}} a_{qq'} = 1, \tag{2.6}$$

[5]Ces propriétés doivent être complétées par les notions d'accessibilités et de co-accessibilité des états [DDE05].

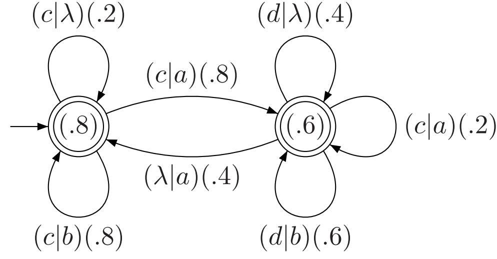


Fig. 2.3 – Transducteur stochastique déduit d'un automate probabiliste appris à partir de chaînes représentant les scripts d'édition entre les paires de séquences d'apprentissage.

avec $a_{qq'}$ la probabilité de transition entre l'état q et q'. Notons que dans l'Équation 2.5, $\delta_q(\#)$ représente la probabilité que l'état soit final et contrairement à l'Équation 2.2, $\delta_q(\#)$ peut être égal à 0 dans certains états.

Un des inconvénients de ce type de modèle est que la structure doit être fixée *a priori*. L'apprentissage revient alors à optimiser les paramètres conditionnellement à cette structure.

Dans [BJS06], les auteurs utilisent un algorithme d'inférence d'automates probabilistes par fusion d'états pour apprendre à la fois la structure (les états) et les paramètres (les probabilités d'édition) d'un modèle discriminatif. Inférer la structure permet donc d'apprendre automatiquement les contextes d'édition. Chaque transition est étiquetée par une opération d'édition et une probabilité finale est assignée à chaque état (voir Fig. 2.3). Dans le but d'apprendre la structure, le processus suivant est répété jusqu'à convergence :

- Construction des alignements optimaux (chemin de Viterbi) entre les différentes paires d'exemples de l'échantillon d'apprentissage. Étant donnés ces alignements, un algorithme classique d'inférence grammaticale (*e.g.* ALERGIA [CO94] ou MDI [TDdlH00]) est utilisé pour apprendre la structure du nouveau modèle.

- Normalisation des paramètres sous les contraintes suivantes :

$$\forall q \in \mathcal{Q},\ \delta_q(\#) + \sum_{l' \in \Sigma, q' \in \mathcal{Q}} t(q, \delta_q(l'|\lambda), q') = 1 \tag{2.7}$$

et

$$\forall q \in \mathcal{Q}, \forall l \in \Sigma,\ \sum_{l' \in \Sigma, q' \in \mathcal{Q}} t(q, \delta_q(l'|\lambda), q') + \sum_{l' \in \Sigma \cup \{\lambda\}, q' \in \mathcal{Q}} t(q, \delta_q(l'|l), q') = 1, \tag{2.8}$$

où $t(q, \delta_q(l'|l), q')$ est la probabilité de transition entre l'état q et q' par l'opération $(l'|l)$. Ces équations sont une transposition des contraintes des Équations 2.3 et 2.4, valables pour les transducteurs à un état, dans ce nouveau contexte à plusieurs états.

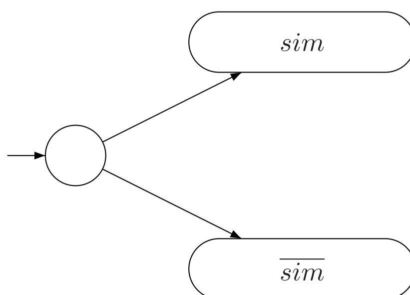


Fig. 2.4 – CRF pour l'apprentissage de la distance d'édition.

Qu'il s'agisse de pair-HMMs ou de transducteurs à plusieurs états, il est donc possible avec les modèles précédents de prendre en compte le contexte. De plus, apprendre automatiquement cette structure peut permettre d'extraire de la connaissance *a posteriori* en interprétant ces contextes. Dans la section suivante, nous montrons qu'il est également possible d'utiliser de la connaissance *a priori* pour apprendre un modèle d'édition.

2.3.4 Machine d'édition à plusieurs états avec ajout de connaissances

Dans [MBP05], les auteurs proposent d'utiliser les CRFs (*Conditionnal Random Fields* [LMP01]) dans le cadre de l'apprentissage de la distance d'édition entre chaînes. Le principal intérêt de ce modèle discriminatif est l'incorporation de connaissance du domaine durant l'apprentissage. Cette adaptation des CRFs se présente sous la forme d'une machine à états finis permettant le calcul d'une probabilité conditionnelle d'une paire de chaînes. Elle est composée de deux ensembles disjoints d'états : un pour les chaînes similaires (sim) et un pour les chaînes dissimilaires ($\overline{sim}$). Il existe un unique état initial qui mène dans l'un ou l'autre des ensembles. De plus, notons qu'il n'existe aucune transition entre les deux ensembles (voir Fig. 2.4). L'intégration de connaissance est possible par l'utilisation de *features* (*e.g.* "un des caractères courant est numérique", "les caractères forment un acronyme", "ne pas tenir compte des mots entre parenthèses", *etc.*). Les transitions du modèle sont étiquetées soit par des *features* soit par des opérations d'édition.

Dans ce modèle, les paramètres sont estimés par l'algorithme EM. Il est à noter que les auteurs ont besoin ici d'exemples positifs et négatifs. En effet, le modèle est construit autour des deux ensembles d'états. Nous avons donc les exemples positifs pour le groupe d'états concernant les chaînes similaires et les exemples négatifs (*i.e.* dissimilaires) qui sont donc positifs pour le groupe des états concernant ce type de chaînes. Les expérimentations ont été menées dans le cadre de la désambiguïsation de noms (adresse de restaurants, citations d'article) et ont montré de meilleurs résultats par rapport à la distance de Levenshtein et la distance apprise suivant [RY98].

2.4 Conclusion

Dans ce chapitre, nous avons présenté l'état de l'art des modèles permettant l'apprentissage de paramètres de similarités d'édition. Ces travaux ont principalement porté sur des similarités entre chaînes et possèdent des caractéristiques différentes. Nous avons, en effet, pu voir que les modèles à un état [RY98, OS06], dits *memoryless*, sont simples, efficaces mais ne permettent pas une prise en compte du contexte pour définir les valeurs des coûts des opérations d'édition. Ceci peut, par contre, être réalisé par des machines à états [DEKM98, BJS06], qui, quoi que plus complexes à apprendre, permettent ainsi d'appréhender des situations où les coûts d'une même opération peuvent évoluer durant l'édition. Enfin, nous avons vu que l'adaptation des CRF à l'apprentissage de distance d'édition [MBP05], permet de tenir compte de *features* des données d'apprentissage et donc d'une certaine manière de connaissance *a priori*. Cependant, ce modèle nécessite de disposer d'exemples positifs et négatifs et ne permet pas d'interprétation *a posteriori* du modèle.

Un premier constat peut être fait à l'issue de ce chapitre. Très peu de travaux ont porté sur les arbres et les graphes, principalement pour des raisons algorithmiques. Si l'apprentissage de similarités d'édition entre graphes contraint alors de travailler sur de petites structures, et en limitant l'ensemble des opérations possibles, nous pensons que l'apprentissage sur des données arborescentes peut être résolu de manière efficace. Compte tenu également du fait que les structures arborescentes sont de plus en plus exploitées dans de nombreuses applications (notamment liées au Web), une première contribution de cette thèse a été de proposer des algorithmes d'apprentissage de similarité d'édition entre arbres. C'est le sujet du chapitre suivant.

D'autre part, afin de contourner les limites des modèles actuels sur les séquences, nous avons travaillé ensuite sur un nouveau type de machine d'édition, appelées CSMs (*Constrained State Machines*). Ce modèle que nous présenterons en tant que deuxième contribution dans le Chapitre 4 a les caractéristiques suivantes :

- Utilisation d'un ensemble fini d'états pour la modélisation de contextes.
- Définition d'une matrice de probabilité d'opérations d'édition par état dans le but de pouvoir exploiter le modèle *a posteriori*.
- Ajout de connaissance *a priori* par l'ajout de contraintes sur les transitions du modèle proposé.

3 Un modèle memoryless pour les arbres

Résumé

Dans ce chapitre, nous présentons une méthode probabiliste pour apprendre des modèles joints et conditionnels de similarités d'édition stochastiques entre arbres. Nous montrons sur plusieurs séries d'expérimentations qu'il est alors possible d'apprendre les coûts des opérations d'édition permettant d'améliorer les distances d'édition classiques entre arbres. Nous présentons également SEDiL, plateforme logicielle, qui est disponible pour la communauté Machine Learning pour effectuer des expérimentations avec les distances d'édition entre chaînes, et arbres.

Les articles suivants ont été tirés du travail présenté dans ce chapitre :
- Laurent Boyer, Amaury Habrard, et Marc Sebban. Learning metrics between tree structured data: Application to image recognition. Dans *European Conference on Machine Learning (ECML)*, volume LNCS 4701, pages 54–66. Springer, 2007.
- Marc Bernard, Laurent Boyer, Amaury Habrard, et Marc Sebban. Learning probabilistic models of tree edit distance. *Pattern Recognition*, 41(8) : 2611–2629, 2008.
- Laurent Boyer, Yann Esposito, Amaury Habrard, José Oncina, et Marc Sebban. SEDiL: Software for edit distance learning. Dans *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, volume LNCS 5212, pages 672–677. Springer, 2008.

3.1 Introduction

Ces dernières années ont vu apparaître un intérêt grandissant pour les données structurées sous forme d'arbres dans les domaines de l'apprentissage automatique et de la reconnaissance de formes. L'utilisation de structures d'arbre dans les modèles permet, en effet, d'appréhender des tâches complexes telles que l'extraction de données sur le Web, la prédiction de structure biologique (ARN), la reconnaissance de la musique, ou encore la transformation de données semi-structurées (*e.g.*, les documents XML). Nous pouvons noter que dans ces applications, l'appel à des similarités entre arbres est souvent nécessaire. Dans ce contexte, la distance d'édition (DE) entre arbres a été au cœur de nombreux tra-

vaux de recherches ces dernières années. Comme nous l'avons évoqué précédemment, dans son utilisation classique, la DE requiert l'utilisation de coûts d'édition souvent arbitraires et fixés *a priori*. Dans ce chapitre, nous nous concentrons sur l'apprentissage automatique d'une distance d'édition stochastique entre arbres, ou dit autrement, sur l'apprentissage des probabilités de chacune des opérations d'édition. Nous nous intéressons à deux types d'approches probabilistes. La première construit un modèle génératif de la DE à partir d'une distribution jointe des opérations d'édition, tandis que la seconde génère une distribution conditionnelle pour définir un modèle discriminatif. L'apprentissage de ces distributions se fait à l'aide de l'adaptation de l'algorithme *Expectation-Maximisation* décrit en Section 2.3.1.

La suite de ce chapitre est organisée comme suit : nous présenterons en Section 3.2 quelques définitions génériques sur les arbres. Dans la Section 3.3, nous décrirons notre première contribution visant à apprendre les paramètres d'édition dans le cadre de la distance de Selkow [Sel77], qui est une des deux distances d'édition standard sur les arbres. Nous montrerons en Section 3.4 l'intérêt de notre approche à travers deux expérimentations : la première sur des données artificielles pour comparer les approches jointe et conditionnelle et ensuite sur une tâche de reconnaissance de formes pour montrer l'apport de l'apprentissage d'une telle similarité dans un processus de reconnaissance d'écriture manuscrite. Dans la Section 3.5, nous présenterons notre seconde contribution sur les arbres, portant sur l'apprentissage des paramètres de la distance d'édition définie selon la distance de Zhang & Shasha [ZS89]. La différence avec la distance précédente vient, comme nous le verrons, du type des opérations d'édition autorisées. Enfin nous présenterons, en Section 3.6 une plateforme d'expérimentation, baptisée SEDiL, qui est proposée en libre accès à la communauté en apprentissage.

3.2 Définitions génériques des arbres

Dans la suite de ce chapitre, nous considérons des arbres ordonnés, d'arité arbitraire.

Définition 3.1 (Arbre) *Soient $\mathcal{V}$ un ensemble de nœuds. Nous définissons un arbre de manière récursive comme suit : un nœud est un arbre, étant donnés T arbres $a_1, \ldots, a_T$ et un nœud $r \in \mathcal{V}$, $x = r(a_1, \ldots, a_T)$ est un arbre. $\rho(x) = r$ est la racine de $r(a_1, \ldots, a_T)$ et $a_1, \ldots, a_T$ sont des sous-arbres.*

Définition 3.2 (Arbre étiqueté) *Soit Σ un ensemble de symboles, et soit $\lambda \notin \Sigma$ le symbole vide. Soit $l : \mathcal{V} \to \Sigma$ une fonction d'étiquetage. $r(a_1, \ldots, a_T)$ est un arbre étiqueté si l'ensemble de ses nœuds sont étiquetés suivant l.*

Par soucis de simplification, nous définissons un arbre étiqueté avec la notation $l(a_1, \ldots, a_T)$ avec l l'étiquette de v. Notons $\mathcal{T}(\Sigma)$ l'ensemble de tous les arbres étiquetés construits à partir de l'alphabet Σ.

Les algorithmes de calcul de distance d'édition nécessitent de définir la notion de forêt.

Définition 3.3 (Forêt) *Une forêt $F = \{a_1, \ldots, a_T\}$ est une séquence d'arbres. F est une forêt ordonnée s'il existe un ordre gauche-droite entre les arbres et si chaque arbre est ordonné.*

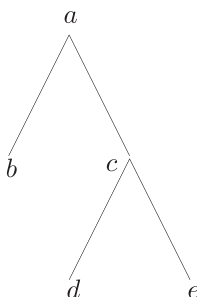


Fig. 3.1 – Représentation graphique de l'arbre $a(b, c(d, e))$.

Définition 3.4 *Soit F une forêt, et $\rho(a)$ la racine d'un arbre $a \in F$. $F - a$ est la forêt obtenue en supprimant l'arbre a de F. $F - \rho(a)$ est la forêt obtenue à partir de F en supprimant le nœud $\rho(a)$. Les enfants de $\rho(a)$ deviennent une séquence d'arbres de la forêt résultante $F - \rho(a)$. $f(a)$ est une forêt composée des enfants du nœud $\rho(a)$.*

Exemple. *Soit l'arbre $x = a(b, c(d, e))$ représenté graphiquement dans la Figure 3.1, $\rho(x) = a$ est la racine de l'arbre. $f(x) = \{b, c(d, e)\}$ est la forêt constituée des fils de l'arbre enraciné en $\rho(x)$. Soit $x_2 = c(d, e)$ l'arbre le plus à droite de la forêt $f(x)$, $f(x) - x_2 = \{b\}$ est la forêt obtenue en supprimant l'arbre x_2. $f(x) - \rho(x_2) = \{b, d, e\}$ est la forêt obtenue par suppression du nœud $\rho(x_2)$.* ◦

Les techniques d'apprentissage que nous allons proposer nécessitent l'utilisation de paires d'arbres. Dans la suite, nous utilisons les notations $l(a_1, \ldots, a_T)$ et $l'(b_1, \ldots, b_V)$ lorsque la définition récursive des arbres est nécessaire. Sinon, nous utilisons la notation x et y pour désigner les arbres de ces paires.

3.3 DE selon Selkow

Dans cette section, nous faisons appel à la DE définie par Selkow [Sel77] qui autorise la délétion et l'insertion de sous-arbres entiers. L'utilisation d'une telle distance peut être intéressante dans le cas des documents HTML. Par exemple, considérons un ensemble d'items d'une liste (Fig. 3.2(a)), il semble non pertinent de supprimer le nœud <UL> sans supprimer les nœuds <LI>.

Dans ce cas précis, nous préférerons donc supprimer le sous-arbre entier <UL>(<LI>,<LI>) (Fig. 3.2(c)) plutôt que de faire remonter les enfants de <UL> comme fils de <BODY> (Fig. 3.2(b)). Un effet direct de ce choix d'opération d'édition réside dans la simplification du calcul de la DE. Nous verrons en Section 3.5 avec la DE de Zhang & Shasha que cette stratégie permet en effet de réduire fortement la complexité algorithmique.

3.3.1 Opérations et coûts d'édition

Dans cette section, les opérations d'édition utilisées pour transformer un arbre d'entrée $l(a_1, \ldots, a_T)$ en un arbre dit de sortie $l'(b_1, \ldots, b_V)$ sont la substitution d'un label

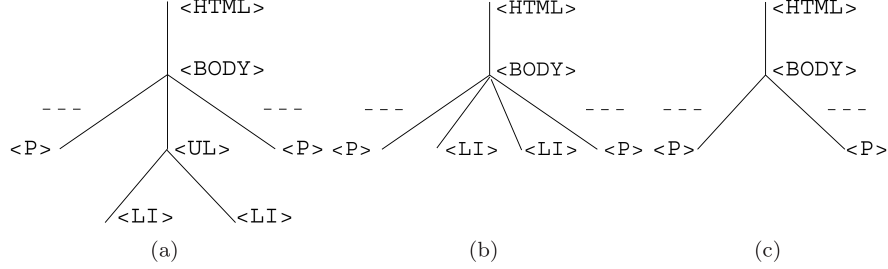


Fig. 3.2 – Stratégies de délétion de nœuds.

$l \in \Sigma$ par un autre $l' \in \Sigma$ (notée (l, l')), la délétion d'un sous-arbre a_i (notée (a_i, λ)) et l'insertion d'un sous-arbre b_j (notée (λ, b_j)) (Fig. 3.3). Comme nous allons le voir, ces deux dernières opérations peuvent récursivement se dériver afin d'avoir *in fine* à appliquer des opérations d'insertion et de délétion de symboles.

Soit une fonction de coût δ définie sur l'ensemble de ces opérations d'édition individuelles. Étant donné que la délétion et l'insertion d'un sous-arbre sont, respectivement, définies récursivement par la suppression et l'ajout d'un ensemble de nœuds, le coût δ_t de délétion ou d'insertion d'un arbre peut être directement défini à partir de la fonction de coût individuel δ. Formellement, δ est une fonction définie sur $(\Sigma \cup \{\lambda\})^2 \backslash \{(\lambda, \lambda)\}$ vers $[0, 1]$.

Le coût de délétion d'un arbre peut être récursivement calculé comme suit :

$$\delta_t(l(a_1, \ldots, a_T), \lambda) = \delta(l, \lambda) + \sum_{i=1}^{T} \delta_t(a_i, \lambda). \tag{3.1}$$

Comme déjà mentionné précédemment, la matrice de coûts δ est habituellement fixée a priori. Par exemple, si nous considérons le Tableau 3.4 et l'arbre $b(c, d)$ alors $\delta_t(b(c, d), \lambda) = \delta(b, \lambda) + \delta_t(c, \lambda) + \delta_t(d, \lambda) = \delta(b, \lambda) + \delta(c, \lambda) + \delta(d, \lambda) = 1.5$.

Sur le même principe, l'insertion d'un arbre nécessite l'ajout successif des différents nœuds le constituant :

$$\delta_t(\lambda, l'(b_1, \ldots, b_V)) = \delta(\lambda, l') + \sum_{j=1}^{V} \delta_t(\lambda, b_j). \tag{3.2}$$

Par exemple, si nous considérons la matrice présentée dans le Tableau 3.4 et l'arbre $b(c, d)$ alors $\delta_t(\lambda, b(c, d)) = \delta(\lambda, b) + \delta_t(\lambda, c) + \delta_t(\lambda, d) = \delta(\lambda, b) + \delta(\lambda, c) + \delta(\lambda, d) = 1.5$.

Avec les expressions 3.1 et 3.2, nous sommes désormais en mesure de calculer une DE entre arbres.

3.3.2 Algorithme classique de calcul de la DE selon Selkow

Soient F_1 et F_2 deux forêts et x et y les arbres les plus à droite de F_1 et F_2 respectivement. Soit δ une fonction de coût sur les paires de symboles, représentant les opérations

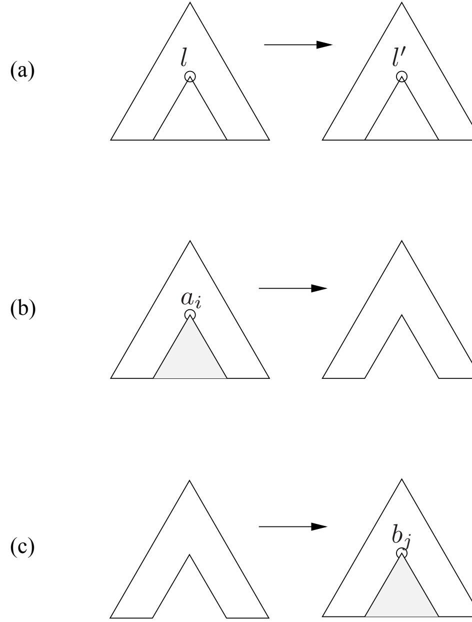


Fig. 3.3 – Opérations autorisées dans l'algorithme de DE de Selkow (a) Substitution de l par l' ; (b) Délétion de a_i ; (c) Insertion de b_j.

δ	λ	a	b	c	d
λ	–	0.5	0.5	0.5	0.5
a	0.5	0	1	1	1
b	0.5	1	0	1	1
c	0.5	1	1	0	1
d	0.5	1	1	1	0

Tab. 3.4 – Un exemple de matrice δ.

d'édition élémentaires entre symboles. La DE $d(F_1, F_2)$ dans le cas général des forêts est donnée par :

$$\begin{aligned}
d(\lambda,\lambda) &= 0 \\
d(F_1,\lambda) &= d(F_1 - x, \lambda) + \delta_t(x,\lambda) \\
d(\lambda,F_2) &= d(\lambda, F_2 - y) + \delta_t(\lambda,y) \\
d(F_1,F_2) &= \min \begin{cases} d(F_1 - x, F_2) + \delta_t(x,\lambda) \\ d(F_1, F_2 - y) + \delta_t(\lambda, y) \\ d(F_1 - x, F_2 - y) + d(f(x), f(y)) + \delta(l(\rho(x)), l(\rho(y))). \end{cases}
\end{aligned}$$

Dans le pseudo-code ci-dessus, nous pouvons noter que les trois opérations d'édition sont prises en compte et que la moins coûteuse est retenue comme dans le cas de la distance d'édition entre chaînes. La principale différence provient du double niveau de récursion opéré dans le cas des arbres, un en largeur et un en profondeur. La Figure 3.5 montre un exemple de script d'édition entre les arbres $a(b, a(b, c))$ et $a(c, a(c))$.

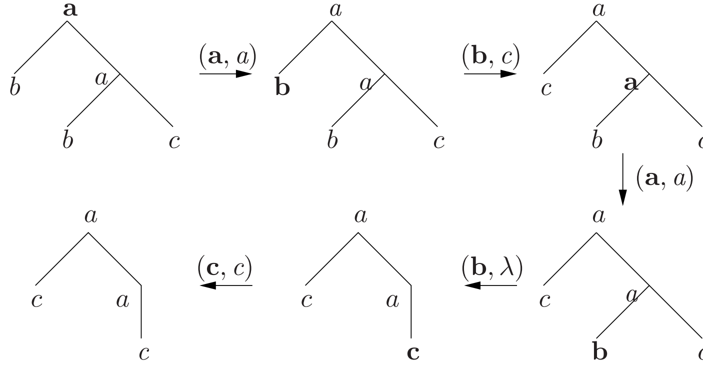


Fig. 3.5 – Exemple de script d'édition entre les arbres $a(b, a(b, c))$ et $a(c, a(c))$.

Notons que la distance peut être efficacement calculée en utilisant la programmation dynamique. Dans ce calcul, nous admettons que la matrice δ nous est donnée en entrée. Nous allons montrer que nous pouvons automatiquement apprendre cette fonction de coûts à partir d'un échantillon d'apprentissage de paires d'arbres. Notre approche stochastique est basée sur une adaptation de l'algorithme EM (présenté en Section 2.3.1), qui permet l'estimation de paramètres cachés (ici la fonction δ) d'un modèle probabiliste à partir d'un échantillon d'apprentissage. Dans ce nouveau contexte, la DE que nous allons définir devient une fonction probabiliste. L'objectif est donc d'apprendre un ensemble de probabilités d'édition et non plus de coûts. Dans la suite, $\delta(l, l')$ représentera donc la probabilité de changer un symbole l en un symbole l'. Dans ce contexte stochastique, nous pourrons ainsi accepter que la probabilité d'une opération d'édition (l, l) ne soit donc pas toujours égale à 1 (ce qui peut être d'ailleurs souhaité dans le cas de situations bruitées). Comme nous l'avons déjà mentionné, par abus de langage, nous continuerons dans la suite à utiliser le terme de distance d'édition DE, même s'il s'agira en fait d'une similarité d'édition, certaines propriétés d'une métrique n'étant plus respectées.

Dans la suite de cette section, nous proposons deux solutions pour apprendre une DE stochastique entre arbres. La première solution propose l'apprentissage d'un modèle génératif tandis que la seconde apprend directement un modèle discriminatif. La différence entre les deux approches se situe dans l'étape de *Maximisation* de l'algorithme EM.

3.3.3 Apprentissage d'une DE stochastique jointe

Fonctions *forward* et *backward*

Pour apprendre la matrice δ, notre adaptation de l'algorithme EM utilise deux fonctions auxiliaires appelées *forward* ($\boldsymbol{\alpha}$) et *backward* ($\boldsymbol{\beta}$), qui sont respectivement définies dans les Algorithmes 3.6 et 3.7 et qui renvoient la probabilité jointe p_δ de transformer un arbre $l(a_1, \ldots, a_i)$ en un autre $l'(b_1, \ldots, b_j)$ avec les paramètres d'édition courants δ. Notons que les caractères en gras sont utilisés pour les appels récursifs tandis que les caractères de fonte normale décrivent des valeurs stockées temporairement.

Ces deux fonctions prennent en argument une paire d'arbres et calculent la probabilité de tous les scripts d'édition applicables entre les deux arbres. D'un point de vue algo-

Algorithme 3.6 : Fonction *forward* $\boldsymbol{\alpha}(l(a_1,\ldots,a_i),l'(b_1,\ldots,b_j))$

Données : Deux arbres $l(a_1,\ldots,a_i)$ et $l'(b_1,\ldots,b_j)$ et δ une matrice de probabilités d'édition.
Résultat : $p_\delta(l(a_1,\ldots,a_i),l'(b_1,\ldots,b_j))$.
Soit $\alpha[0..i,0..j]$ une matrice de taille $(i+1)\times(j+1)$;
$\alpha[0,0]\leftarrow\delta(l,l')$;
pour t *de* 0 *à* i **faire**
  pour v *de* 0 *à* j **faire**
    si $(t>0)$ *ou* $(v>0)$ **alors** $\alpha[t,v]\leftarrow 0$;
    si $(t>0)$ **alors** $\alpha[t,v]\leftarrow\alpha[t,v]+\boldsymbol{\alpha}(a_t,\lambda)\times\alpha[t-1,v]$;
    si $(v>0)$ **alors** $\alpha[t,v]\leftarrow\alpha[t,v]+\boldsymbol{\alpha}(\lambda,b_v)\times\alpha[t,v-1]$;
    si $(t>0)$ *et* $(v>0)$ **alors** $\alpha[t,v]\leftarrow\alpha[t,v]+\boldsymbol{\alpha}(a_t,b_v)\times\alpha[t-1,v-1]$;
retourner $\alpha[i,j]$;

Algorithme 3.7 : Fonction *backward* $\boldsymbol{\beta}(l(a_i,\ldots,a_T),l'(b_j,\ldots,b_V))$

Données : Deux arbres $l(a_i,\ldots,a_T)$ et $l'(b_j,\ldots,b_V)$, $1\leq i\leq T$ et $1\leq j\leq V$ et δ une matrice de probabilités d'édition.
Résultat : $p_\delta(l(a_i,\ldots,a_T),l'(b_j,\ldots,b_V))$.
Soit $\beta[0..T,0..V]$ une matrice de taille $(T+1)\times(V+1)$;
$\beta[T,V]\leftarrow 1$;
pour t *de* T *à* $i-1$ **faire**
  pour v *de* V *à* $j-1$ **faire**
    si $(t<T)$ *ou* $(v<V)$ **alors** $\beta[t,v]\leftarrow 0$;
    si $(t<T)$ **alors** $\beta[t,v]\leftarrow\beta[t,v]+\boldsymbol{\beta}(a_{t+1},\lambda)\times\beta[t+1,v]$;
    si $(v<V)$ **alors** $\beta[t,v]\leftarrow\beta[t,v]+\boldsymbol{\beta}(\lambda,b_{v+1})\times\beta[t,v+1]$;
    si $(t<T)$ *et* $(v<V)$ **alors** $\beta[t,v]\leftarrow\beta[t,v]+\boldsymbol{\beta}(a_{t+1},b_{v+1})\times\beta[t+1,v+1]$;
si $(i=1)$ *et* $(j=1)$ **alors retourner** $\beta[0,0]\times\delta(l,l')$;
sinon retourner $\beta[i-1,j-1]$;

rithmique, cela requiert de calculer récursivement les probabilités d'édition entre paires de sous-arbres dont les racines sont situées à la même profondeur. Bien que les processus soient différents, les fonctions sont symétriques et donnent la même probabilité finale :

$$\begin{aligned}&p_\delta(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))\\&\quad=\boldsymbol{\alpha}(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V))\\&\quad=\boldsymbol{\beta}(l(a_1,\ldots,a_T),l'(b_1,\ldots,b_V)).\end{aligned}$$

La fonction *forward* visite la racine puis les fils de la gauche vers la droite alors que la fonction *backward* commence par les fils de la droite vers la gauche et ensuite la racine. La Figure 3.8 illustre ces deux algorithmes de complexité quadratique en le nombre de nœuds des arbres en utilisant la programmation dynamique. Notons que contrairement à la distance standard, nous remplaçons la fonction minimum par la somme des probabilités sur l'ensemble des chemins permettant de transformer un arbre en un autre. En effet, l'objectif étant d'apprendre *in fine* une distribution de probabilité sur les scripts d'édition, la notion de coût minimal devient obsolète. Par contre, si l'objectif est de conserver un

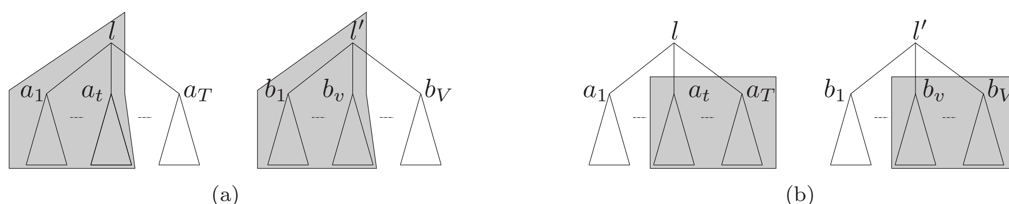


Fig. 3.8 – (a) Évaluation de $\boldsymbol{\alpha}(l(a_1, \ldots, a_t), l'(b_1, \ldots, b_v))$ avec l'algorithme *forward* ; (b) Évaluation de $\boldsymbol{\beta}(l(a_t, \ldots, a_T), l'(b_v, \ldots, b_V))$ avec l'algorithme *backward*.

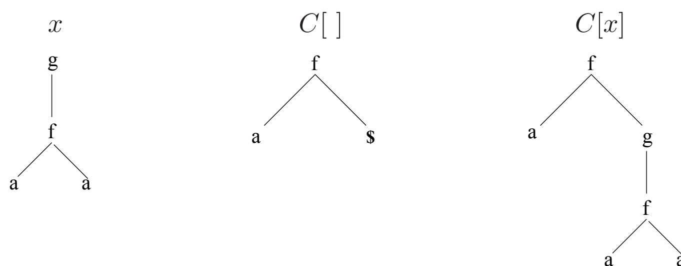


Fig. 3.9 – Un arbre x, un contexte $C[\ ]$ de profondeur $depth_c(C[\ ]) = 1$ et l'arbre résultat $C[x]$.

seul chemin, nous pouvons nous restreindre au chemin le plus probable, appelé script de Viterbi (voir Section 2.3.1). Pour l'obtenir, il suffit de remplacer les sommes par un maximum.

Étape Expectation

Durant la phase *Expectation*, le but est d'estimer l'espérance des événements cachés, *i.e.* les opérations d'édition utilisées pour transformer un arbre en un autre. Ces espérances sont stockées dans une matrice auxiliaire γ de dimension $(|\Sigma|+1) \times (|\Sigma|+1)$. Ce processus prend une paire d'apprentissage (x, y) en entrée. Ensuite, pour chaque sous-arbre (a_t, b_v), où a_t est un sous-arbre de x et b_v un sous-arbre de y, et où a_t et b_v sont à la même profondeur dans x et dans y, l'algorithme accumule l'espérance des trois opérations d'édition que sont la délétion de a_t, l'insertion de b_v et la substitution de $l(a_t)$ par $l(b_v)$.

Comme a_t et b_v sont des sous-arbres, le calcul de ces espérances nécessite de prendre en compte non seulement leurs frères mais aussi les restes des deux arbres x et y qui ne sont pas directement concernés par a_t et b_v. Nous appelons ces parties des contextes.

Définition 3.5 (Contexte) *Un contexte $C[\ ]$ est un arbre non-vide où exactement une feuille (un nœud sans sous-arbre) est étiquetée par le symbole \$ tel que $\$ \notin \Sigma \cup \{\lambda\}$. Si $C[\ ]$ est un contexte et x un arbre, $C[x]$ dénote l'arbre obtenu en substituant le nœud étiqueté \$ par x (voir Fig. 3.9). La profondeur d'un contexte $C[\ ]$, noté $depth_C(C[\ ])$, correspond à la profondeur du nœud étiqueté \$.*

Cette définition permet de comprendre ce qui est fait dans l'Algorithme 3.12. Concentrons nous sur le cas de la substitution, qui est l'opération la plus générale, présentée

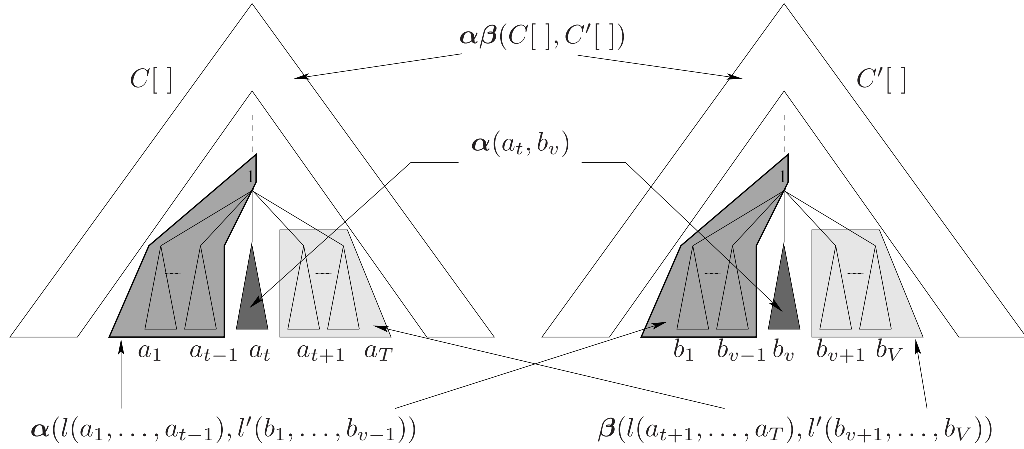


Fig. 3.10 – Explication graphique de la substitution (a_t, b_v).

Algorithme 3.11 : $\boldsymbol{\alpha\beta}(C[\ ], C'[\ ])$

Données : Deux contextes $C[\ ]$ et $C'[\ ]$.
Résultat : Probabilité de la paire $(C[\ ], C'[\ ])$.
si $C[\ ] = \$$ *et* $C'[\ ] = \$$ **alors retourner** *1* ;
sinon
  Soit l,l' deux étiquettes, $C_l[\ ]$ et $C_{l'}[\ ]$ deux contextes et $(T-1)+(V-1)$ arbres $a_1, \ldots, a_{t-1}, a_{t+1}, \ldots, a_T, b_1, \ldots, b_{v-1}, b_{v+1}, \ldots, b_V$ tel que
  $C[\ ] = l(a_1, \ldots, a_{t-1}, C_l[\ ], a_{t+1}, \ldots, a_T)$ et
  $C'[\ ] = l'(b_1, \ldots, b_{v-1}, C_{l'}[\ ], b_{v+1}, \ldots, b_V)$;
  $A \leftarrow \boldsymbol{\alpha}(l(a_1, \ldots, a_{t-1}), l'(b_1, \ldots, b_{v-1})) \times \boldsymbol{\beta}(l(a_{t+1}, \ldots, a_T), l'(b_{v+1}, \ldots, b_V))$;
  retourner $A \times \boldsymbol{\alpha\beta}(C_l[\ ], C_{l'}[\ ])$;

dans la Figure 3.10 :

- La fonction $\boldsymbol{\alpha\beta}$, décrite dans l'Algorithme 3.11, calcule la probabilité jointe de deux contextes $C[\ ]$ de a_t et $C'[\ ]$ de b_v. Nous appelons cette fonction $\boldsymbol{\alpha\beta}$ car elle utilise la fonction *forward* et la fonction *backward* pour calculer respectivement la partie droite et la partie gauche des contextes.
- La fonction *forward* est utilisée pour calculer la probabilité des parties à gauche, *i.e.* $\boldsymbol{\alpha}(l(a_1, \ldots, a_{t-1}), l'(b_1, \ldots, b_{v-1}))$, tandis que la fonction *backward* calcule celle des parties à droite, *i.e.* $\boldsymbol{\beta}(l(a_{t+1}, \ldots, a_T), l'(b_{v+1}, \ldots, b_V))$, de l'opération de substitution entre a_t et b_v.
- La fonction *forward* prend en compte l'opération d'édition elle-même ainsi que la transformation des deux sous-arbres liés à l'opération, ce qui implique une simple récursion verticale.

Étape Maximisation et algorithme EM

L'étape *Maximisation* est cruciale dans l'algorithme EM. Elle représente l'étape de normalisation des espérances des valeurs $\gamma(l, l')$ obtenues pendant l'étape *Expectation*,

Algorithme 3.12 : *Expectation*(x,y)

Données : Deux arbres x et y.

Soit $Paires_{\{x,y\}} = \{(a_t, b_v)\}$ avec a_t sous-arbre de x, b_v sous-arbre de y tel qu'il existe deux contextes $C[\,]$ et $C'[\,]$ telle que $depth_C(C[\,]) = depth_C(C'[\,])$ et $(T-1)+(V-1)$ arbres $a_1, \ldots, a_{t-1}, a_{t+1}, \ldots, a_T$ et $b_1, \ldots, b_{v-1}, b_{v+1}, \ldots, b_V$ tel que $x = C[l(a_1, \ldots, a_{t-1}, a_t, a_{t+1}, \ldots, a_T)]$ et
$y = C'[l'(b_1, \ldots, b_{v-1}, b_v, b_{v+1}, \ldots, b_V)]$;

pour chaque *paire* $(a_t, b_v) \in Paires_{\{x,y\}}$ **faire**

- **si** $(a_t \neq \lambda)$ **alors** `/* délétion */`
  - $\gamma(l(\rho(a_t)), \lambda) \leftarrow \gamma(l(\rho(a_t)), \lambda) + \frac{\boldsymbol{\alpha\beta}(C[\,],C'[\,])\boldsymbol{\alpha}(l(a_1,\ldots,a_{t-1}),l'(b_1,\ldots,b_v))\boldsymbol{\alpha}(a_t,\lambda)\boldsymbol{\beta}(l(a_{t+1},\ldots,a_T),l'(b_{v+1},\ldots,b_V))}{\boldsymbol{\alpha}(x,y)}$;
- **si** $(b_v \neq \lambda)$ **alors** `/* insertion */`
  - $\gamma(\lambda, l(\rho(b_v))) \leftarrow \gamma(\lambda, l(\rho(b_v))) + \frac{\boldsymbol{\alpha\beta}(C[\,],C'[\,])\boldsymbol{\alpha}(l(a_1,\ldots,a_t),l'(b_1,\ldots,b_{v-1}))\boldsymbol{\alpha}(\lambda,b_v)\boldsymbol{\beta}(l(a_{t+1},\ldots,a_T),l'(b_{v+1},\ldots,b_V))}{\boldsymbol{\alpha}(x,y)}$;
- **si** $(a_t \neq \lambda)$ *et* $(b_v \neq \lambda)$ **alors** `/* substitution */`
  - $\gamma(l(\rho(a_t)), l(\rho(b_v))) \leftarrow \gamma(l(\rho(a_t)), l(\rho(b_v))) + \frac{\boldsymbol{\alpha\beta}(C[\,],C'[\,])\boldsymbol{\alpha}(l(a_1,\ldots,a_{t-1}),l'(b_1,\ldots,b_{v-1}))\boldsymbol{\alpha}(a_t,b_v)\boldsymbol{\beta}(l(a_{t+1},\ldots,a_T),l'(b_{v+1},\ldots,b_V))}{\boldsymbol{\alpha}(x,y)}$;

Algorithme 3.13 : Normalisation_jointe(γ)

Données : Une matrice γ d'espérance des opérations d'édition.
Résultat : Une matrice δ qui définit une distribution jointe des opérations d'édition.

$TA \leftarrow 0$;
pour chaque $(a, b) \in (\Sigma \cup \{\lambda\})^2$ **faire**
- $TA \leftarrow TA + \gamma(a, b)$;

pour chaque $(a, b) \in (\Sigma \cup \{\lambda\})^2$ **faire**
- $\delta(a, b) \leftarrow \gamma(a, b)/TA$;

---

l'objectif étant de maximiser la vraisemblance de l'échantillon d'apprentissage.

En plus de la maximisation de la vraisemblance, la normalisation va aussi permettre à la matrice δ de définir une distribution statistique. Dans l'Algorithme 3.13, en normalisant simplement chaque espérance par le nombre total d'opérations d'édition effectuées durant l'itération, nous assurons l'apprentissage d'une distribution jointe.

En combinant les Algorithmes 3.6, 3.7, 3.11, 3.12 et 3.13, nous sommes en mesure de présenter le modèle d'apprentissage de notre DE stochastique jointe (voir Alg. 3.14). Notons que le processus est itératif et est donc répété jusqu'à convergence. Celle-ci pourra être atteinte soit lorsqu'un certain nombre d'itérations est atteint, soit lorsque les différentes probabilités ne changent pas de manière significative (des tests statistiques peuvent être alors utilisés) entre deux itérations.

Exemple. *Dans le but d'aider le lecteur aux subtilités techniques des algorithmes présentés dans les pages précédentes, nous allons montrer un exemple d'exécution. Considérons*

Algorithme 3.14 : Expectation-Maximisation

Données : LS : un ensemble d'apprentissage de paire d'arbres.
répéter
  pour chaque $(l, l') \in (\Sigma \cup \{\lambda\})^2$ **faire**
    $\gamma(l, l') \leftarrow 0$;
  $\gamma(\lambda, \lambda) \leftarrow |LS|$;
  pour chaque $(x, y) \in LS$ **faire**
    Expectation(x, y) ;
  Normalisation_jointe(γ) ;
jusqu'à *convergence* ;

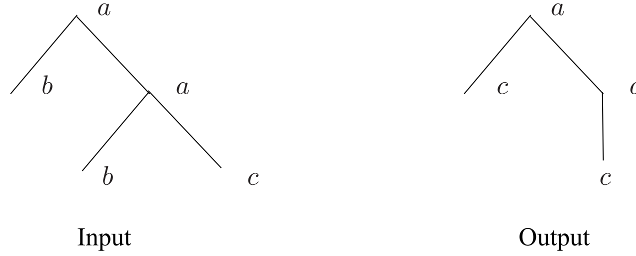


Fig. 3.15 – Exemple de paire d'apprentissage.

l'alphabet $\Sigma = \{a, b, c\}$ *et un ensemble d'apprentissage composé de l'unique paire d'arbres* $(a(b, a(b, c)), a(c, a(c)))$ *décrite graphiquement dans la Figure 3.15.*

La matrice δ *est initialisée de manière uniforme avec une valeur de* $1/16$ *(i.e.* $\frac{1}{(|\Sigma|+1)^2}$*) pour chaque opération d'édition.*

Le Tableau 3.16 présente l'ensemble des 26 paires d'arbres et leurs contextes associés pour la paire d'arbres étudiée pour l'exécution de l'algorithme Expectation 3.12. Notons que pour modéliser des opérations d'insertion ou de délétion, c'est-à-dire lorsque a_t *ou* b_v *correspondent à* λ*, nous ajoutons dans le contexte le \$ à l'endroit où se place l'opération. Par exemple dans la paire 4, pour l'arbre* $a(b, a(b, c))$*, nous représentons une insertion entre* b *et* $a(b, c)$ *par le contexte* $C[] = a(b, \$, a(b, c))$ *;* $C[\lambda]$ *correspondant alors à* $a(b, a(b, c))$*.*

Voici un exemple d'exécution de l'algorithme $\boldsymbol{\alpha\beta}$ *(Alg. 3.11) pour les contextes de la paire 16.*

$\boldsymbol{\alpha\beta}(a(b, \$), a(\$, a(c)))$
  $C[\ ] = a(b, \$) = a(b, C_l[\ ])$
  $C'[\ ] = a(\$, a(c)) = a(C'_l[\ ], a(c))$
  $A < -\boldsymbol{\alpha}(a(b), a) \times \boldsymbol{\beta}(a, a(a(c)))$ $(1/16^2 \times 1/16^2)$
  retour $A \times \boldsymbol{\alpha\beta}(\$, \$)$ $(1/16^4)$.

	$C[\,]$	a_t	b_v	$C'[\,]$
1	$\$$	$a(b,a(b,c))$	$a(c,a(c))$	$\$$
2	$a(\$,b,a(b,c))$	λ	c	$a(\$,a(c))$
3	$a(b,\$,a(b,c))$	λ	c	$a(\$,a(c))$
4	$a(b,a(b,c),\$)$	λ	c	$a(\$,a(c))$
5	$a(\$,b,a(b,c))$	λ	$a(c)$	$a(c,\$)$
6	$a(b,\$,a(b,c))$	λ	$a(c)$	$a(c,\$)$
7	$a(b,a(b,c),\$)$	λ	$a(c)$	$a(c,\$)$
8	$a(\$,a(b,c))$	b	λ	$a(\$,c,a(c))$
9	$a(\$,a(b,c))$	b	λ	$a(c,\$,a(c))$
10	$a(\$,a(b,c))$	b	λ	$a(c,a(c),\$)$
11	$a(b,\$)$	$a(b,c)$	λ	$a(\$,c,a(c))$
12	$a(b,\$)$	$a(b,c)$	λ	$a(c,\$,a(c))$
13	$a(b,\$)$	$a(b,c)$	λ	$a(c,a(c),\$)$
14	$a(\$,a(b,c))$	b	c	$a(\$,a(c))$
15	$a(\$,a(b,c))$	b	$a(c)$	$a(c,\$)$
16	$a(b,\$)$	$a(b,c)$	c	$a(\$,a(c))$
17	$a(b,\$)$	$a(b,c)$	$a(c)$	$a(c,\$)$
18	$a(b,a(\$,b,c))$	λ	c	$a(c,a(\$))$
19	$a(b,a(b,\$,c))$	λ	c	$a(c,a(\$))$
20	$a(b,a(b,c,\$))$	λ	c	$a(c,a(\$))$
21	$a(b,a(\$,c))$	b	λ	$a(c,a(\$,c))$
22	$a(b,a(\$,c))$	b	λ	$a(c,a(\$,c))$
23	$a(b,a(b,\$))$	c	λ	$a(c,a(c,\$))$
24	$a(b,a(b,\$))$	c	λ	$a(c,a(c,\$))$
25	$a(b,a(\$,c))$	b	c	$a(c,a(\$))$
26	$a(b,a(b,\$))$	c	c	$a(c,a(\$))$

Tab. 3.16 – Ensemble des paires d'arbres et leurs contextes associés à la paire d'arbres $(a(b,a(b,c)),a(c,a(c)))$ pour l'exécution de l'algorithme Expectation 3.12.

γ	λ	a	b	c	$Total$
λ	–	0.0053	0.0	0.2045	0.2098
a	0.0053	1.9931	0.0	0.0016	2
b	0.6585	0.0016	0.0	1.339	2
c	0.5460	0.0	0.0	0.4540	1

Tab. 3.17 – Matrice γ obtenue à la fin de la première itération. La colonne $Total$ décrit la distribution des symboles de l'arbre d'entrée. *i.e.* le nombre de fois que chaque lettre de Σ est utilisée dans l'arbre d'entrée. La valeur 0.2098 correspond à l'espérance du nombre d'insertion, étant donné la distribution courante δ. Nous avons omis la valeur $\gamma(\#)$ qui est ici égale à 1.0 après la première itération.

Développons maintenant l'exécution de $\boldsymbol{\alpha}(a(b), a))$.

$\boldsymbol{\alpha}(a(b), a)$
$\quad\#\ \ i = 1,\ j = 0$
$\quad t = 0,\ v = 0$
$\quad\quad \alpha[0,0] = 1/16$
$\quad t = 1,\ v = 0$
$\quad\quad \#\ \ t > 0 \mid\mid v > 0$
$\quad\quad \alpha[1,0] = 0$
$\quad\quad \#\ \ t > 0$
$\quad\quad \alpha[1,0] += \boldsymbol{\alpha}(b,\lambda) \times \alpha[0,0]\ \ (1/16 \times 1/16)$
$\quad retour\ \alpha[1,0]\ \ (1/16^2)$.

Enfin, l'exécution de $\boldsymbol{\beta}(a, a(a(c)))$ *s'effectue comme suit :*

$\boldsymbol{\beta}(a, a(a(c)))$ *(les arbres à l'origine sont : $a(b,\$)$ et $a(\$, a(c))$)*
$\quad\#\ \ T = 2,\ V = 2,\ i = 3,\ j = 2$
$\quad t = 2,\ v = 2$
$\quad\quad \beta[2,2] = 1$
$\quad t = 2,\ v = 1$
$\quad\quad \#\ \ t < T \mid\mid v < V$
$\quad\quad \beta[2,1] = 0$
$\quad\quad \#\ \ v < V$
$\quad\quad \beta[2,1] += \boldsymbol{\beta}(\lambda, a(c)) \times \beta[2,2]\ \ (1/16^2 \times 1)$
$\quad retour\ \beta[2,1]\ \ (1/16^2)$.

La matrice γ obtenue après la première itération est présentée dans le Tableau 3.17 tandis que la matrice δ correspondante est présentée dans le Tableau 3.18.

Après 10 itérations, l'algorithme converge vers un optimum décrit par la matrice δ (Tableau 3.19). Notons que notre algorithme a correctement appris une solution maximisant la vraisemblance de la paire (ou tout au moins une solution possible), consistant en la délétion du symbole c, la substitution de b par c, le a restant inchangé.

Avec une initialisation différente, qui donnerait une part plus importante aux substitutions d'un symbole par lui-même, nous obtiendrions un modèle pour lequel le a se changerait en a, le c en c, et le symbole b se transformerait soit en c, soit il serait supprimé. Ceci confirme que EM ne peut atteindre que des optima locaux. ∘

δ	λ	a	b	c
λ	–	0.00086	0.0	0.03294
a	0.00086	0.32096	0.0	0.00025
b	0.10604	0.00025	0.0	0.21577
c	0.0879	0.0	0.0	0.0731

Tab. 3.18 – Matrice δ obtenue à la fin de la première itération. Notons que $\delta(\#) = .16107$.

δ	λ	a	b	c
λ	–	0.0	0.0	0.0
a	0.0	0.3333	0.0	0.0
b	0.0	0.0	0.0	0.3333
c	0.1667	0.0	0.0	0.0

Tab. 3.19 – Matrice δ obtenue après 10 itérations. Notons que $\delta(\#) = .1667$, ce qui signifie que le symbole terminal est pris en compte comme les autres symboles et qu'il apparaît avec une probabilité de $\frac{1}{6}$. Notons que les valeurs de δ somment à 1.

Discussion

Notons que le cœur de l'étape de *Maximisation* est la normalisation qui nous permet d'apprendre une distribution jointe sur les scripts d'édition et de définir par la même occasion une probabilité jointe $p_\delta(x, y)$ [RY98]. Cependant, pour l'utilisation d'un tel modèle dans une tâche de classification, nous avons besoin d'avoir à disposition une probabilité conditionnelle $p_\delta(y|x)$. Une première solution classique consiste à calculer $p_\delta(y|x)$ à partir de la valeur jointe telle que : $p_\delta(y|x) = p_\delta(x, y)/p_\delta(x)$. Cependant, ceci implique une dépendance à la distribution des exemples d'entrées générant ainsi un biais.

Une autre solution consiste à apprendre directement une distribution conditionnelle $p_\delta(y|x)$, *i.e.* un modèle discriminatif. L'avantage de cette approche est de supprimer le biais lié aux modèles génératifs.

3.3.4 Apprentissage d'une DE stochastique conditionnelle

Dans cette section, nous visons donc à apprendre un modèle discriminatif. Nous considérons désormais les opérations d'édition de manière conditionnelle. Elles seront donc notées $(l'|l)$ où l' est une étiquette de l'arbre de sortie y et l une étiquette de l'arbre d'entrée x. Ainsi, un script d'édition est représenté par une séquence d'opérations conditionnelles. Exceptée cette modification de notation, les fonctions *forward*, *backward* et *Expectation* déjà présentées dans le cadre joint restent inchangées dans le contexte conditionnel. La différence majeure se situe dans l'étape de *Maximisation* qui doit prendre en compte de nouvelles contraintes statistiques permettant ainsi d'obtenir directement une probabilité conditionnelle $p_\delta(y|x)$ à chaque étape de l'algorithme EM.

Contraintes conditionnelles

Pour apprendre un modèle conditionnel, nous utilisons les contraintes d'optimisation déjà vues en Section 2.3.2. En effet, il est possible de modéliser la distribution d'un

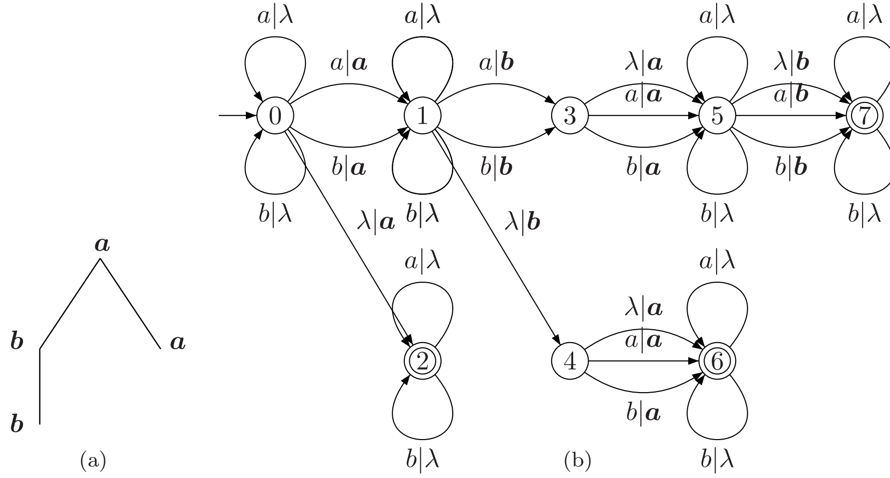


Fig. 3.20 – Distribution conditionnelle d'un arbre de sortie étant donné un arbre d'entrée. La transition $\lambda|a$ entre l'état 0 et 2 correspond à la délétion de la racine de l'arbre $\boldsymbol{a}(\boldsymbol{b}(\boldsymbol{b}), \boldsymbol{a})$. Cette délétion implique la délétion de l'ensemble des nœuds de l'arbre. Par soucis de simplification, nous n'avons pas dessiné les transitions correspondantes à la délétion des trois autres nœuds. Donc, seulement des insertions peuvent apparaître dans l'état 2. La même remarque est valable pour la transition entre l'état 1 et 4 correspondant à la délétion des deux nœuds b.

script d'édition conditionnellement à l'arbre d'entrée x par un automate à états fini non déterministe. Prenons un petit exemple pour expliquer ce principe. Soit l'arbre $a(b(b), a)$ défini dans la Figure 3.20(a). Pour calculer la distance d'édition probabiliste, nous avons considéré que nous allions prendre en compte l'ensemble des scripts possibles. Nous pouvons donc modéliser cet ensemble par un transducteur (Fig. 3.20(b)).

Les cycles sur les états correspondent aux insertions possibles avant la "consommation" des nœuds de l'arbre d'entrée. Les états finaux sont marqués par un double cercle et correspondent à la fin de la lecture de l'arbre d'entrée. Notons que l'automate de la Figure 3.20(b) a trois états finaux. Ceci est du au fait que nous utilisons la distance de Selkow, pour laquelle la délétion d'un nœud signifie la délétion de l'ensemble de tous ses fils. Ceci explique le chemin entre l'état 0 et l'état 2 qui illustre la délétion de la racine de l'arbre (impliquant la délétion de tout l'arbre) et donc les seules opérations possibles sont des insertions. Le chemin commençant à l'état 1 et finissant à l'état 6 suit le même principe : la délétion du symbole $\boldsymbol{b}$ (premier fils de la racine $\boldsymbol{a}$) qui implique la délétion de son unique fils $\boldsymbol{b}$, puis le traitement de la feuille $\boldsymbol{a}$ avant la fin de la lecture de l'arbre. Le reste de l'automate (chemin de 0 à 7) est facilement interprétable vu qu'il suit la lecture de l'arbre d'entrée (les délétions étant déjà traitées par les chemins mentionnés précédemment).

La représentation des scripts d'édition possibles sous la forme d'un transducteur nous permet de définir facilement de nouvelles contraintes satisfaisant une distribution de probabilité conditionnelle. Il est en effet connu que pour modéliser une distribution statistique, un automate probabiliste doit satisfaire les deux conditions suivantes :

1. Premièrement, la somme des probabilités sortantes d'un état doit être égale à un.

Plus formellement,

$$\forall l \in \Sigma, \sum_{l' \in \Sigma \cup \{\lambda\}} \delta(l'|l) + \sum_{l' \in \Sigma} \delta(l'|\lambda) = 1. \tag{3.3}$$

2. Deuxièmement, les probabilités sortantes des états finaux doivent aussi représenter une distribution.

$$\sum_{l' \in \Sigma} \delta(l'|\lambda) + \delta(\#) = 1. \tag{3.4}$$

En remplissant ces contraintes, nous pouvons prouver (voir [BBHS08], Appendice A pour plus de détails) que nous apprenons bien une distribution sur l'ensemble des scripts d'édition définis conditionnellement à un arbre donné. Malgré le fait que cette preuve ne soit pas nécessaire pour définir l'algorithme d'optimisation que nous allons présenter, elle assure que nous définissons réellement une distribution sur l'ensemble des transformations possibles d'un arbre d'entrée en un autre et que nous avons un modèle stochastique consistant.

Normalisation optimale

La normalisation optimale qui satisfait les nouvelles contraintes est la solution à un problème d'optimisation présenté dans [DLR77]. Dans cette section, nous adaptons aux arbres le principe de la preuve présentée dans [OS06] dans le cas des chaînes de caractères.

Soient $\mathcal{O}$ et $\mathcal{U}$ deux espaces, respectivement, des données observables et non-observables pour lesquels nous supposons qu'il existe un vecteur de paramètres θ qui lie les deux distributions. L'objectif de notre algorithme consiste à trouver le θ optimal qui maximise la fonction de vraisemblance $L(O, \theta) = \ln(p(O|\theta))$, pour un ensemble $O \subset \mathcal{O}$ de données observables. Comme nous l'avons déjà évoqué dans le Chapitre 2, il est montré dans [DLR77] qu'il est possible de construire, étant donné θ_t estimé à partir de θ, un meilleur θ_{t+1} en maximisant la fonction suivante :

$$Q(\theta_n, \theta_{n+1}) = \mathbf{E}[\ln(p(O, \mathcal{U}|O_{n+1}))|O, \theta_n] \tag{3.5}$$

où $\mathbf{E}$ est une espérance conditionnelle sur la distribution $\mathcal{U}$.

Dans notre cas, le vecteur θ représente l'ensemble des opérations d'édition de la matrice δ. À partir de l'ensemble des paires d'arbres $LS \subset \mathcal{T}(\Sigma)^2$, nous pouvons construire $LS_{in} = \{x : (x, y) \in LS\}$ et $LS_{out} = \{y : (x, y) \in LS\}$. Nous pouvons maintenant écrire la fonction de vraisemblance à maximiser comme suit :

$$L(LS_{out}, \theta, LS_{in}) = \ln(p(LS_{out}|\theta, LS_{in})) = \ln \Pi_{(x,y) \in LS} p(y|\theta, x).$$

La transformation d'un arbre d'entrée x en un arbre de sortie y peut être exprimée sous la forme d'un script d'édition $e = e_1 \dots e_n$ de n opérations d'édition. Nous disons que x est l'arbre d'entrée correspondant à e (noté $x = in(e)$) si et seulement si x est composé successivement des symboles d'entrée $l_1 \dots l_n$ (l_k peut être le symbole vide) du script e. Symétriquement, y est l'arbre de sortie correspondant à e (noté $y = out(e)$) si et seulement si y est composé successivement des symboles de sortie du script e.

Comme nous pouvons le voir dans la Figure 3.21, étant donné un script d'édition e, plusieurs x ou y peuvent correspondre. Cependant, sachant que chaque nœud peut être

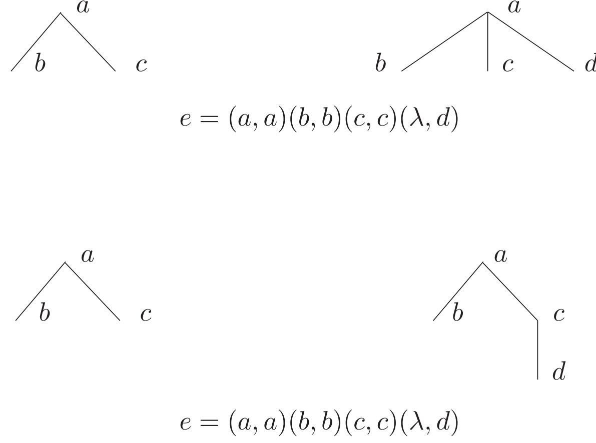


Fig. 3.21 – Exemple d'un script d'édition e qui correspond à deux paires d'arbres. Étant donné e, $in(e) = \{a, b, c, \lambda\}$ et $out(e) = \{a, b, c, d\}$.

caractérisé par sa localisation dans l'arbre, nous pouvons considérer que $in(e)$ et $out(e)$ sont uniques.

Étant donné un arbre d'entrée $in(e)$, la probabilité conditionnelle d'un script d'édition $e = e_1 \dots e_n$, où $e_i = (l'_i|l_i) \in (\Sigma \cup \{\lambda\})^2 \backslash \{(\lambda, \lambda)\}$, est alors égale à :

$$\pi_\delta(e|in(e)) = \Pi_{i=1}^n \delta(l'_i|l_i)\delta(\#) = \Pi_{i=1}^n \delta(e_i)\delta(\#).$$

Pour chaque paire d'apprentissage (x, y), nous pouvons associer un ensemble $E(y|x)$ de l'ensemble des scripts d'édition entre x et y tel que

$$E(y|x) = \{e \in ((\Sigma \cup \{\lambda\})^2 \backslash \{(\lambda, \lambda)\})^\star : x = in(e), y = out(e)\}.$$

Nous pouvons alors déduire que

$$p_\delta(y|x) = \sum_{e \in E(y|x)} \pi_\delta(e|x).$$

De plus, définissons par $E(LS)$ l'ensemble des scripts d'édition sur toutes les paires d'apprentissage, tel que :

$$E(LS) = \bigcup_{(x,y) \in LS} E(y|x).$$

Étant données les notations précédentes, la fonction Q de l'Équation 3.5 peut être

écrite comme suit :

$$\begin{array}{rcl} Q(\theta_n, \theta_{n+1}) & = & \mathbf{E}[\ln(p(LS_{out}, e|\theta_{t+1}, LS_{in}))|LS_{out}, \theta_t, LS_{in}] \\ & = & \sum_{e \in ((\Sigma \cup \{\lambda\})^2)^\star} \pi_\delta(e|LS_{out}, \theta_t, LS_{in}) \\ & & \times \ln p(LS_{out}, e|\theta_{t+1}, LS_{in}) \\ & & \text{avec } \pi_\delta(e|y, \theta_t, x) = 0 \text{ si } x \neq in(e) \text{ ou } y \neq out(e) \\ & = & \sum_{e \in E(LS)} \pi_\delta(e|out(e), \theta_t, in(e)) \\ & & \times \ln \pi_\delta(out(e), e|\theta_{t+1}, in(e)) \\ & = & \sum_{e \in E(LS)} \pi_\delta(e|out(e), \theta_t, in(e)) \\ & & \times \ln \pi_\delta(e|\theta_{t+1}, in(e)) \\ & = & \sum_{e \in E(LS)} \pi_\delta(e|out(e), \theta_t, in(e)) \\ & & \times (\sum_{i=0}^{|e|} \ln \delta(out(e_i)|\theta_{t+1}, in(e_i)) \\ & & + \ln \delta(\lambda|\theta_{t+1}, \lambda)) \\ & = & \sum_{e_j \in (\Sigma \cup \{\lambda\})^2} \\ & & \times \sum_{e e_j e' \in E(LS)} \pi_\delta(e e_j e'|out(e e_j e'), \theta_t, in(e e_j e')) \\ & & \times \ln \delta(e|\theta_{t+1}) \\ & & + \sum_{e \in E(LS)} \pi_\delta(e|out(e), \theta_t, in(e)) \\ & & \times \ln \delta((\lambda|\lambda)|\theta_{t+1}) \\ & = & \sum_{e_j \in (\Sigma \cup \{\lambda\})^2} \gamma(e_j) \ln \delta(e_j|\theta_{t+1}) \\ & & + |LS| \ln \delta((\lambda|\lambda)|\theta_{t+1}). \end{array}$$

Nous avons à choisir durant l'apprentissage θ_{t+1} qui minimise la fonction $Q(\theta_t, \theta_{t+1})$ tout en respectant les conditions 3.3 et 3.4.

En utilisant les multiplicateurs de Lagrange, définissons la fonction suivante

$$\begin{array}{rl} LaMu = & \sum_{e \in (\Sigma \cup \{\lambda\})^2} \gamma(e) \ln \delta(e|\theta_{t+1}) + |LS| \ln \delta((\lambda|\lambda)|\theta_{t+1}) \\ & - \sum_{l \in \Sigma} \mu_l (\sum_{l' \in \Sigma} \delta((l'|l)|\theta_{t+1}) \\ & + \sum_{l' \in \Sigma} \delta((l'|\lambda)|\theta_{t+1}) + \delta((\lambda|l)|\theta_{t+1}) - 1) \\ & -\mu(\sum_{l' \in \Sigma} \delta((l'|\lambda)|\theta_{t+1}) + \delta((\lambda|\lambda)|\theta_{t+1}) - 1). \end{array}$$

En recherchant les valeurs qui annulent les dérivés partielles de $LaMu$, nous obtenons aisément :

$$\delta((l'|l)|\theta_{t+1}) = \frac{\gamma((l'|l))}{\mu_l}, \quad \delta((l'|\lambda)|\theta_{t+1}) = \frac{\gamma((l'|\lambda))}{\sum_l \mu_l + \mu},$$
$$\delta((\lambda|l)|\theta_{t+1}) = \frac{\gamma((\lambda|l))}{\mu_l}, \quad \delta((\lambda|\lambda)|\theta_{t+1}) = \frac{|LS|}{\mu}.$$

En substituant ces valeurs dans les Équations 3.3 et 3.4, nous obtenons :

$$\frac{\sum_{l'} \gamma((l'|\lambda))}{\sum_l \mu_l + \mu} + \frac{\sum_{l'} \gamma((l'|l))}{\mu_l} + \frac{\gamma((\lambda|l))}{\mu_l} = 1, \forall l \in \Sigma,$$
$$\frac{\sum_{l'} \gamma((l'|\lambda))}{\sum_l \mu_l + \mu} + \frac{|LS|}{\mu} = 1$$

où $\delta((\lambda, \lambda)|\theta_{t+1})$ est équivalent à $\delta(\#)$.

Nous avons un système à $|\Sigma| + 1$ équations et $|\Sigma| + 1$ inconnues. Une solution possible du système est donnée par :

Algorithme 3.22 : Normalisation_conditionnelle

Données : Une matrice γ d'espérance des opérations d'édition.
Résultat : Une matrice δ qui définit une distribution conditionnelle des opérations d'édition.

$N \leftarrow \sum_{e\in(\Sigma\cup\{\lambda\})^2} \gamma(e)$;
$N(\lambda) \leftarrow \sum_{l'\in\Sigma} \gamma(l'|\lambda)$;
pour chaque $l \in \Sigma$ **faire**
  $N(l) \leftarrow \sum_{l'\in\Sigma\cup\{\lambda\}} \gamma(l'|l)$;
$\delta(\gamma|\gamma) \leftarrow \frac{N-N(\lambda)}{N}$;
pour chaque $(l, l') \in \Sigma^2$ **faire**
  $\delta(l'|l) \leftarrow \frac{\gamma(l'|l)}{N(l)} \frac{N-N(\lambda)}{N}$;
pour chaque $l \in \Sigma$ **faire**
  $\delta(\lambda|l) \leftarrow \frac{\gamma(\lambda|l)}{N(l)} \frac{N-N(\lambda)}{N}$;
pour chaque $(l' \in \Sigma$ **faire**
  $\delta(l'|\lambda) \leftarrow \frac{\gamma(l'|\lambda)}{N}$;

$$\mu = |LS|\frac{N}{N - N(\lambda)}, \quad \mu_l = N(l)\frac{N}{N - N(\lambda)}$$

avec

$$N = \sum_{e\in\Sigma^2} \gamma(e) + |LS| = \sum_{e\in(\Sigma\cup\{\lambda\})^2} \gamma(e),$$

$$N(\lambda) = \sum_{l'\in\Sigma} \gamma((l'|\lambda)), \quad N(l) = \sum_{l'\in\Sigma\cup\{\lambda\}} \gamma((l'|l)).$$

Ces paramètres sont ceux utilisés dans l'Algorithme 3.22 de normalisation permettant donc d'apprendre une distribution conditionnelle.

Exemple. *Comme nous l'avons fait pour la partie jointe, nous exécutons notre algorithme sur l'exemple de la Figure 3.15. Nous utilisons la même matrice d'initialisation* δ*. Les Tableaux 3.23 et 3.24 montrent le résultat de l'algorithme après respectivement 1 et 10 itérations. Comme prévu, nous pouvons voir que le concept cible a été correctement appris. Les seules opérations possibles sont bien la délétion du symbole b, la substitution d'un b par un c, alors que le symbole a reste inchangé. La différence avec le modèle joint est que le modèle appris est indépendant de la distribution de l'arbre d'entrée. En d'autres termes, quelle que soit la distribution d'entrée utilisée (ici, par exemple le nombre de fois où le symbole b apparaît dans l'arbre), nous apprenons le même modèle. Nous reparlerons de ce comportement dans la section expérimentale suivante.* ◦

δ	λ	a	b	c
λ	–	0.0011	0.0	0.0415
a	0.0028	0.9535	0.0	0.0008
b	0.3198	0.0008	0.0	0.6365
c	0.5255	0.0	0.0	0.4317

Tab. 3.23 – Matrice δ obtenue après 1 itération de EM. Notons que $\delta(\#) = 0.9573$.

δ	λ	a	b	c
λ	–	0.0	0.0	0.0
a	0.0	1.0	0.0	0.0
b	0.0	0.0	0.0	1.0
c	1.0	0.0	0.0	0.0

Tab. 3.24 – Matrice δ obtenue après 10 itérations. Notons que $\delta(\#) = 1.0$.

3.4 Résultats expérimentaux

Dans cette section, nous menons deux séries d'expérimentations. La première est une comparaison des modèles joint et conditionnel sur des données artificielles afin de mettre en évidence les propriétés des deux approches. La seconde est une tâche de reconnaissance de formes réelle visant à montrer l'apport de l'apprentissage d'une distance d'arbres à la reconnaissance de chiffres manuscrits.

3.4.1 Données artificielles

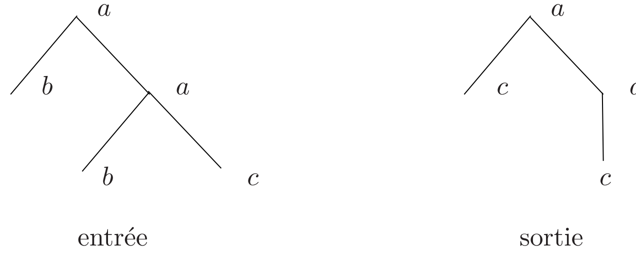
Nous menons l'expérience suivante pour montrer la pertinence de nos deux modèles (joint et conditionnel) dans l'estimation des paramètres d'un modèle cible. Si nous sommes capables de les apprendre, *i.e.* de converger vers la distribution cible, cela signifiera qu'un modèle stochastique appris aura quoiqu'il arrive de meilleurs résultats qu'une DE utilisant des coûts fixés *a priori*. Dans le meilleur des cas, en effet, ces coûts devraient être fixés aux probabilités apprises par le modèle.

Le protocole expérimental utilisé est le suivant : premièrement, nous créons une distribution cible définie par un matrice théorique $\delta^\star$ (Tab. 3.25(a)). Ensuite, nous générons un ensemble d'arbres suivant une distribution d'entrée $\mathcal{P}$. Ceci est fait en donnant une probabilité à chacun des symboles d'entrée (ici $\Sigma = \{a, b, c\}$), et en fixant deux paramètres limitant la taille et la profondeur de l'arbre généré. Pour construire un ensemble d'apprentissage LS de paires d'arbres, nous assignons à chaque arbre d'entrée un arbre de sortie. Ce dernier est généré en utilisant l'arbre d'entrée et les opérations d'édition décrites dans $\delta^\star$. Notons que la question de la génération d'un arbre de sortie est difficile et nécessiterait de plus amples recherches théoriques. D'un point de vue pratique, pour conduire nos expériences, nous analysons simplement l'arbre d'entrée avec un parcours en profondeur, et pour chaque nœud nous appliquons une opération suivant la distribution cible $\delta^\star$. La Figure 3.25 montre un exemple de génération d'arbre.

Notons que dans le cas d'expériences avec des données réelles, les paires d'appren-

δ^*	λ	a	b	c
λ	–	0.0	0.0	0.0
a	0.0	0.3333	0.0	0.0
b	0.0	0.0	0.0	0.3333
c	0.1667	0.0	0.0	0.0

(a) matrice δ^* avec $\delta(\#) = .1667$



(b) arbres d'entrée et de sortie

Fig. 3.25 – Pour construire un arbre de sortie à partir d'un arbre d'entrée et une matrice jointe $\delta^\star$, nous utilisons un parcours en profondeur. Pour chaque nœud rencontré, nous appliquons aléatoirement une opération d'édition. Dans cette exemple, le symbole b est toujours changé en c, tandis que le symbole c est toujours supprimé.

tissage peuvent être obtenues de manière plus naturelle. Par exemple, si nous voulons apprendre un modèle de débruitage, alors les paires seront les couples (donnée bruitée, donnée non bruitée). Dans le cadre de la reconnaissance de formes, chaque paire peut être construite à partir d'un élément et de son plus proche voisin. Dans ce cas, le but est d'apprendre les différentes distortions possibles entre éléments de la même classe (par exemple, apprendre différentes possibilités d'écrire un chiffre, différentes interprétations d'un morceau de musique, etc.). Néanmoins, notons ici que la constitution de paires d'exemples optimales est un problème ouvert.

Dans cette section, le but est donc d'apprendre $\delta^\star$ à partir de LS (constitué d'un nombre croissant de paires) en utilisant soit notre modèle génératif, soit celui discriminatif. Pour vérifier l'effet de la distribution d'entrée $\mathcal{P}$ sur les modèles appris, nous utilisons différentes densités pour générer les arbres d'entrée. Le critère de performance utilisé dans cette étude comparative est la distance normalisée $dis(\delta, \delta^\star)$ entre la distribution apprise et la distribution cible. Dans le cas du modèle joint :

$$dis(\delta, \delta^\star) = \frac{\sum_{l \in \Sigma \cup \{\lambda\}} \sum_{l' \in \Sigma \cup \{\lambda\}} |\delta(l, l') - \delta^\star(l, l')|}{2}$$

$dis(\delta, \delta^\star)$ ayant une valeur comprise dans l'intervalle $[0, 1]$.

Dans le cas conditionnel, nous définissons $dis(\delta, \delta^\star)$ comme :

$$dis(\delta, \delta^\star) = \frac{(A + B|\Sigma|)}{2|\Sigma|}$$

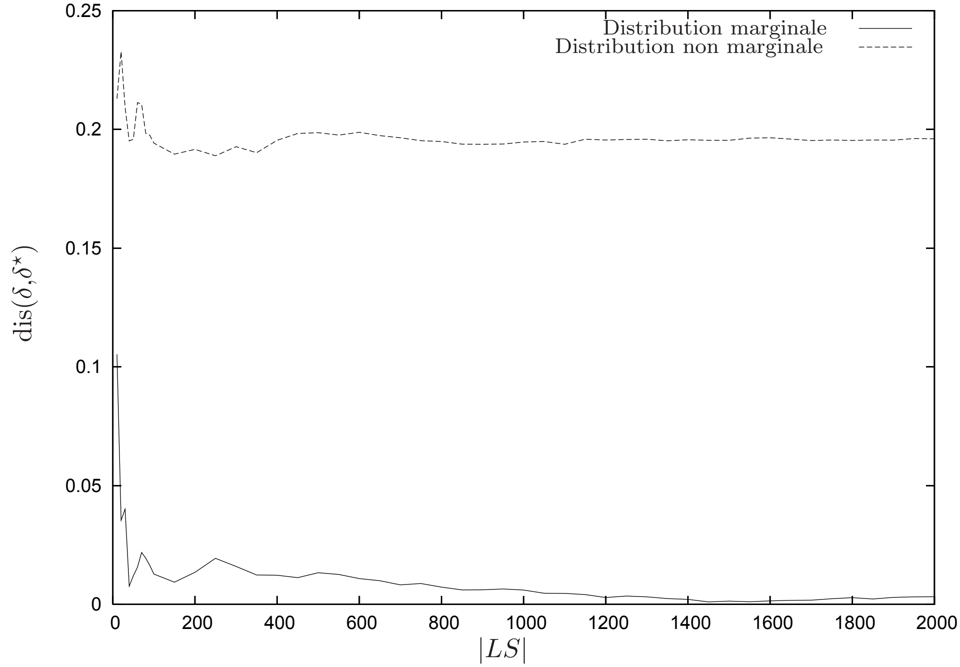


Fig. 3.26 – Résultats dans le cas joint.

avec

$$A = \sum_{l \in \Sigma} \sum_{l' \in \Sigma \cup \{\lambda\}} |\delta(l'|l) - \delta^\star(l'|l)|$$

et

$$B = \sum_{l' \in \Sigma \cup \{\lambda\}} |\delta(l'|\lambda) - \delta^\star(l'|\lambda)|$$

Expériences avec le modèle génératif

Nous construisons deux ensembles d'arbres d'entrée, pour mettre en avant le problème de biais des modèles génératifs. Le premier ensemble est directement généré suivant la distribution marginale de $\delta^\star$ qui est définie par : $\forall l \in \Sigma \cup \{\lambda\}, \delta^\star(l) = \sum_{l' \in \Sigma \cup \{\lambda\}} \delta^\star(l, l')$. Le second ensemble est généré suivant une distribution $\mathcal{P}$ aléatoire mais différente de la distribution marginale. Le graphe de la Figure 3.26 montre, comme attendu, que la seule possibilité pour apprendre la distribution cible est d'utiliser des données générées suivant la distribution marginale. L'utilisation d'une autre distribution mène au biais déjà évoqué précédemment, cela se caractérisant sur le graphe par une grande valeur de $dis(\delta, \delta^\star)$.

Est-ce que cette remarque remet en question le fait d'apprendre des modèles génératifs ? Probablement pas, mais elle permet de souligner un aspect important concernant les hypothèses posées en apprentissage statistique. Pendant longtemps, et aujourd'hui encore dans bon nombre de travaux dans la communauté d'apprentissage automatique,

il a été supposé que l'échantillon d'apprentissage est généré selon la même distribution que celle du concept cible[1]. Pourtant, de nombreuses applications du monde réel peuvent remettre en question cette hypothèse. C'est pourquoi apprendre directement un modèle conditionnel peut être une approche alternative pour contourner ce problème. Cette problématique est au cœur de nombreux travaux récents, notamment dans les domaines du *transfer learning* [PY10], de l'apprentissage à partir de données non-iid [AHRU09], ou encore du *domain adaptation* [BDBC+10].

Expériences avec le modèle conditionnel

Dans la seconde série d'expérimentations, nous utilisons le même protocole expérimental pour apprendre cette fois une distribution conditionnelle. Dans ce cas, nous avons testé trois différentes distributions d'entrée (dont la marginale). Le graphe de la Figure 3.27 confirme que quelle que soit la distribution utilisée, notre modèle discriminatif est capable d'apprendre la distribution cible. Cependant, nous devons prendre en compte la remarque suivante. L'apprentissage d'un tel modèle peut nécessiter un grand nombre de paires d'apprentissage pour converger. C'est un comportement normal et facilement explicable. Comme nous l'avons déjà évoqué, les modèles discriminatifs sont connus pour avoir une variance plus importante que celle des modèles génératifs. Vu qu'ils modélisent une distribution conditionnelle, chaque probabilité est estimée par une partie de l'échantillon d'apprentissage, conduisant à une plus grande variance. Mais, comme il est mentionné dans [BT04], asymptotiquement, un classifieur discriminatif sera préféré.

3.4.2 Reconnaissance de caractères manuscrits

Nous avons montré dans la section précédente que notre modèle discriminatif est capable d'apprendre une cible artificielle sans aucune connaissance sur la distribution d'entrée. Dans cette section, nous étudions le comportement de notre approche sur une application réelle et montrons que la DE apprise est meilleure qu'une DE classique. Pour la réalisation de cette tâche, nous utilisons une partie de la base de données NIST Special Database 3 décrivant un ensemble de caractères manuscrits (lettres et chiffres).

Afin d'appliquer nos modèles d'arbres, nous allons transformer des fichiers bitmaps de caractères par une structure arborescente. Pour les raisons mentionnées dans la section précédente, nous allons mener les expériences en utilisant seulement le modèle discriminatif.

Construction des arbres à partir des images bitmap

Dans cette expérimentation, nous traitons des images bitmap 128×128 représentant des chiffres. Nous construisons un ensemble d'apprentissage de 8000 chiffres et un échantillon test de 2000 exemples. Chaque image est codée sous la forme d'un arbre à partir d'un ensemble de symboles extraits selon l'algorithme présenté dans [GBMO95]. Étant donné un chiffre, nous construisons tout d'abord une racine étiquetée avec un symbole fictif -1. L'algorithme lit ensuite l'image, de la gauche vers la droite, de haut en bas jusqu'à trouver le premier pixel. Ensuite, il suit le contour de la forme jusqu'à revenir

[1] Notons tout de même que l'apprentissage bayésien MAP utilise les informations liées aux deux distributions avant de prendre une décision. Pour plus de détails, nous renvoyons le lecteur au livre *Apprentissage artificiel, concepts et algorithmes* d'A. Cornuéjols et L. Miclet [CM10].

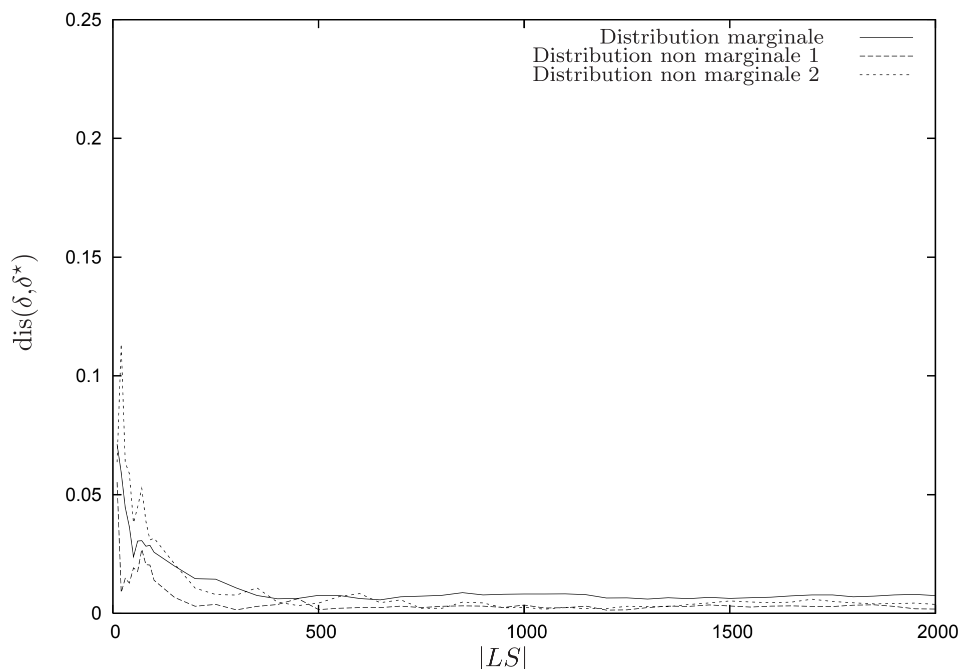


Fig. 3.27 – Résultats dans le cas conditionnel.

au point de départ. Durant ce parcours, l'algorithme construit un arbre en utilisant la primitive directionnelle permettant d'atteindre le prochain pixel (appelée code de Freeman) et génère un nouveau fils depuis la racine. Si un même code est trouvé à l'itération suivante, ce symbole devient un fils du nœud courant. La Figure 3.28 décrit un exemple sur un chiffre 3. La représentation structurée de ce dernier est révélée en bas à droite de la figure.

La construction d'une représentation arborescente à partir d'une image est un problème difficile et mériterait des investigations plus poussées. Elle est d'ailleurs au cœur de nombreux travaux au laboratoire Hubert Curien. Néanmoins, celle proposée ici possède les propriétés intéressantes suivantes :

- Premièrement, elle permet de garder l'information structurelle de l'image.
- Deuxièmement, elle est robuste à un changement d'échelle. En effet, celui-ci ne requiert que la délétion ou l'insertion de sous-arbres. Cette remarque montre l'intérêt de l'usage de la distance de Selkow qui autorise de telles opérations. Par exemple, la Figure 3.29 montre deux chiffres 3 de tailles différentes. Nous pouvons remarquer que les deux représentations ont la même série de fils au premier niveau, et que la transformation du premier en second ne nécessite que la délétion de quelques sous-arbres.

Comme nous l'avons précédemment mentionné, notre méthode requiert l'utilisation d'un ensemble de paires d'exemples pour l'apprentissage de la DE probabiliste. En suivant le principe illustré dans [RY98] dans le cas des chaînes, une solution consiste à

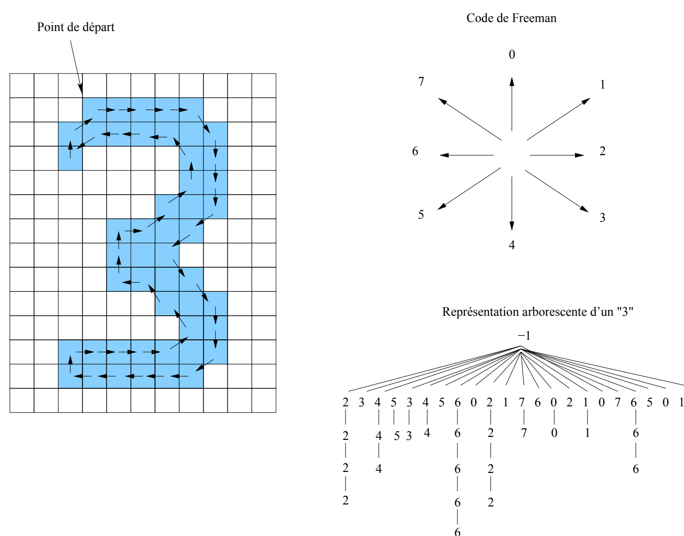


Fig. 3.28 – Exemple d'une représentation arborescente.

construire des paires d'exemples 'similaires' qui décrivent les distorsions possibles entre deux instances de la même classe. De telles paires peuvent être données par un expert du domaine. Pour nos expérimentations, nous avons décidé de construire automatiquement ces paires où l'arbre d'entrée est un élément de l'ensemble d'apprentissage LS et la sortie est son plus proche voisin dans LS trouvé à l'aide de la DE classique entre arbres.

Comparaisons expérimentales

Le but ici est de montrer l'intérêt d'apprendre une DE stochastique pour réaliser une tâche de reconnaissance de chiffres avec un algorithme de type plus-proches voisins [CH67]. Pour ce faire, nous comparons le comportement de notre DE apprise avec des distances non-apprises. Quatre comparaisons sont réalisées :

- La première concerne la comparaison avec la distance de Selkow classique en utilisant des poids standard fixés à 1 pour toutes les opérations d'édition.
- Suivant les résultats donnés dans [MO98], une meilleure stratégie consisterait à donner des poids aux opérations d'édition suivant l'angle relatif entre les codes de Freeman utilisés pour décrire les chiffres. Dans cette expérience, nous utiliserons la matrice donnée dans le Tableau 3.30.
- Nous comparons aussi nos résultats à une autre distance d'édition d'arbres. Même si notre approche est basée sur la distance de Selkow, nous avons décidé de nous comparer avec la distance entre arbres définie par Zhang & Shasha[2] [ZS89] qui admet la délétion et l'insertion de nœuds. Une valeur de 1 est donnée à chaque

[2]Nous reviendrons sur l'apprentissage de cette distance en Section 3.5.

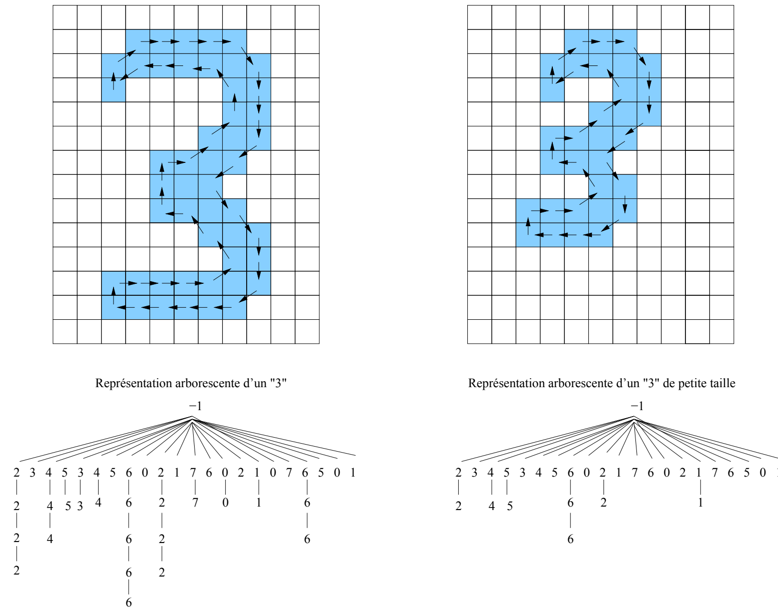


Fig. 3.29 – Intérêt de la représentation arborescente. Pour transformer l'arbre de gauche en celui de droite, la DE requiert simplement la délétion de quelques sous-arbres.

opération.

– Pour finir, nous comparons notre approche à la distance de Zhang & Shasha en utilisant les poids décrits dans le Tableau 3.30.

Résultats et discussion

Les résultats présentés dans la Figure 3.31 sont obtenus à partir de différentes tailles de l'échantillon d'apprentissage (de 50 à 8000 paires d'arbres). La précision est calculée sur l'échantillon test TS de 2000 exemples. Chaque arbre contenu dans TS est étiqueté en utilisant l'algorithme du 1 plus-proche voisin. La métrique utilisée est la distance d'édition (apprise ou standard) calculée selon les différents algorithmes que nous avons mentionnés précédemment. La Figure 3.31 nous permet de tirer les conclusions suivantes.

L'apprentissage d'une DE probabiliste permet d'obtenir de meilleurs résultats que dans le cas de l'utilisation d'une DE non apprise. Quelle que soit la taille de l'échantillon d'apprentissage, les résultats obtenus avec la DE probabiliste sont meilleurs que tous les autres.

Deuxièmement, comme déjà observé dans [MO98], l'utilisation de la matrice de coûts de la Table 3.30 donne de meilleurs résultats que la configuration naïve qui consiste à utiliser le même poids pour l'ensemble des opérations d'édition. Cette remarque est confirmée pour les deux approches Selkow et Zhang & Shasha.

W_s	0	1	2	3	4	5	6	7
0	0	1	2	3	4	3	2	1
1	1	0	1	2	3	4	3	2
2	2	1	0	1	2	3	4	3
3	3	2	1	0	1	2	3	4
4	4	3	2	1	0	1	2	3
5	3	4	3	2	1	0	1	2
6	2	3	4	3	2	1	0	1
7	1	2	3	4	3	2	1	0

Tab. 3.30 – Coûts de substitution W_s entre deux directions étant donnée leur position relative. Les coûts des insertions et des délétions sont fixés à 1.

W'_s	0	1	2	3	4	5	6	7
0	0	1.2	2.3	2.9	2.7	2.5	1.8	2.5
1	1	0	2.1	2.8	2.7	2.5	2.2	2.6
2	2.3	2	0	2.5	2.1	2.3	2.3	2.6
3	3.7	2.8	1.7	0	0.2	0.9	2.8	3.9
4	2.6	2.5	2.2	2.5	0	1.2	2	2.9
5	2.7	2.5	2.5	2.7	0.5	0	1.9	2.9
6	2.2	2.3	2.2	2.7	2.2	2.1	0	2.6
7	0.5	0	3.4	5.7	4.7	1.2	0.5	0

Tab. 3.32 – Coûts de substitution W'_s appris à partir d'un échantillon de 8000 exemples par notre algorithme discriminatif.

Observons la matrice apprise à partir de l'ensemble complet de l'échantillon d'apprentissage (Tab. 3.32). Pour être comparées avec les coûts d'édition de la Table 3.30, notons que les valeurs présentées dans cette table ont été déduites à partir des probabilités apprises par notre algorithme en appliquant l'opération -log sur les valeurs apprises. Nous pouvons remarquer que les matrices sont assez similaires, *i.e.* le poids d'une opération d'édition entre deux codes consécutifs est faible tandis que celui entre deux codes de directions opposées est plus important. Malgré cette forte ressemblance, nous pouvons noter que notre algorithme a de bien meilleurs résultats. Cela signifie qu'avec autant de degrés de liberté, une légère modification des poids d'édition peut avoir une forte influence sur le résultat final de la DE. Cela confirme encore une fois la difficulté de définir manuellement les poids des opérations d'édition et donc l'intérêt de disposer de techniques d'apprentissage automatique.

Pour finir, nous pouvons remarquer que la précision de l'algorithme de Zhang & Shasha est meilleure que celle de l'algorithme de Selkow lorsque nous utilisons les coûts de la Table 3.30. Cela nous a donc amenés à penser qu'il serait intéressant d'adapter notre stratégie à l'apprentissage de la DE de Zhang & Shasha. C'est l'objectif de la section suivante.

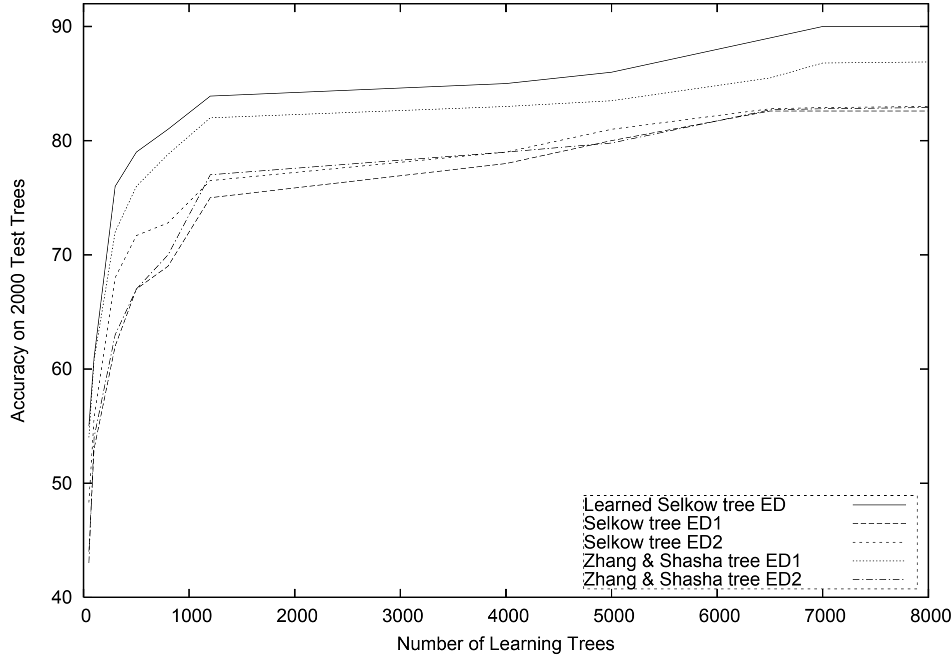


Fig. 3.31 – Résultats sur une application de reconnaissance de caractères manuscrits. ED1 utilise les poids standard (1 pour chaque opération), tandis que ED2 utilise les poids définis dans la Table 3.30.

3.5 DE selon Zhang & Shasha

Dans cette partie, après une présentation de l'algorithme de base de Zhang & Shasha [Tai79, ZS89], nous proposons un algorithme d'apprentissage de cette distance qui autorise des insertions et délétions de nœuds individuels dans l'arbre. Notons qu'une étude de ce type de distance peut être trouvée dans [Kle98, Bil05, DT05].

3.5.1 Opérations d'édition

Les opérations d'édition autorisées sont : la substitution d'un nœud de label $l \in \Sigma$ par un nœud de label $l' \in \Sigma$; la délétion d'un nœud de label l et l'insertion d'un nœud de label l' (Fig. 3.33). L'opération de délétion consiste à supprimer un nœud r et ses fils deviennent alors ceux du père de r. Sur un schéma identique, l'opération d'insertion consiste à ajouter un nœud r' tel que une sous-séquence des fils du père de r' devient l'ensemble des fils de r'. Dans les opérations de délétion et d'insertion, il est important de noter que l'ordre établi entre les fils avant l'opération d'édition reste inchangé après la transformation.

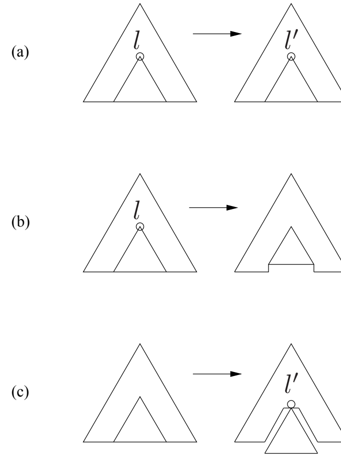


Fig. 3.33 – (a) Substitution de l par l' ; (b) Délétion du nœud de label l ; (c) Insertion du nœud de label l'.

3.5.2 Algorithme classique du calcul de la DE de Zhang & Shasha

Soient F_1 et F_2 deux forêts et a et b les arbres les plus à droite de F_1 et F_2 respectivement. Soit δ une fonction de coût sur les paires de labels, représentant les opérations d'édition.

La DE $d(F_1, F_2)$ dans le cas général des forêts est donnée par :

$$\begin{aligned}
d(\lambda, \lambda) &= 0 \\
d(F_1, \lambda) &= d(F_1 - \rho(a), \lambda) + \delta(l(\rho(a)), \lambda) \\
d(\lambda, F_2) &= d(\lambda, F_2 - \rho(b)) + \delta(\lambda, l(\rho(b))) \\
d(F_1, F_2) &= \min \begin{cases} d(F_1 - \rho(a), F_2) + \delta(l(\rho(a)), \lambda) \\ d(F_1, F_2 - \rho(b)) + \delta(\lambda, l(\rho(b))) \\ d(F_1 - a, F_2 - b) + d(f(a), f(b)) \\ \qquad\qquad +\delta(l(\rho(a)), l(\rho(b))). \end{cases}
\end{aligned}$$

La Figure 3.34 montre un script d'édition possible entre les arbres $a(b, c(a, b))$ et $b(a(b), a, b))$ pour la distance de Zhang & Shasha.

Ce pseudo-code suggère à nouveau une approche de programmation dynamique pour calculer la DE entre arbres. Nous pouvons noter que $d(F_1, F_2)$ dépend d'un nombre constant de sous-problèmes de taille moindre. Zhang & Shasha définissent ces sous-problèmes avec la notion de *keyroots* d'un arbre donné a :

$$keyroots(a) = \{\rho(a)\} \cup \{r \in \mathcal{V}(a) \text{ tel que } r \text{ a un frère gauche}\}.$$

À partir de cet ensemble de *keyroots* (voir Fig. 3.35(a)), on peut déduire un ensemble de sous-forêts de a, appelées *special subforests* (Fig. 3.35(b)), défini par les forêts $f(r)$ avec $r \in keyroots(a)$. Zhang & Shasha définissent la notion d'ensemble de sous-problèmes pertinents qui autorisent le calcul dynamique de la DE. Ces sous-problèmes pertinents

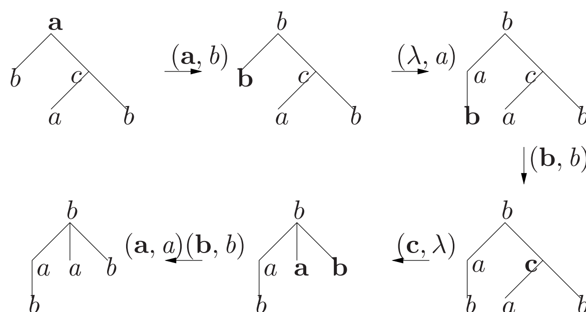


Fig. 3.34 – Exemple de script d'édition entre les arbres $a(b, c(a, b))$ et $b(a(b), a, b))$.

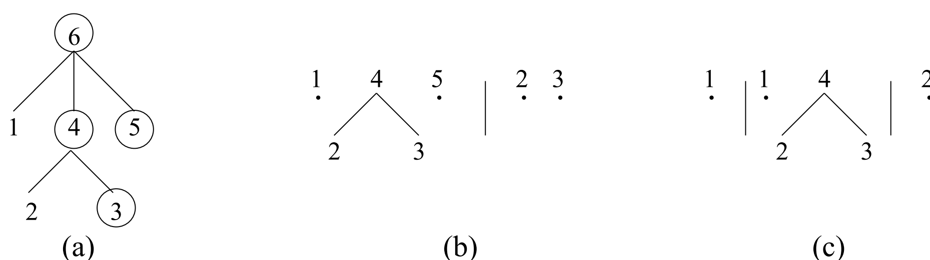


Fig. 3.35 – (a) les *keyroots* sont représentés par les nœuds avec un cercle ; (b) les *special subforests* qui correspondent aux forêts dont la racine est dans l'ensemble des *keyroots* ; (b)+(c) les sous-problèmes pertinents (*special subforests* et forêts préfixes des *special subforests*).

correspondent aux forêts issues des préfixes des *special subforests* (Fig. 3.35(b+c)). Ensuite, pour le calcul de la DE $d(F_1, F_2)$, il peut être montré qu'il est suffisant de calculer $d(S_1, S_2)$ pour tous les couples (S_1, S_2) de sous-problèmes pertinents.

3.5.3 Apprentissage d'une DE stochastique

Fonctions *forward* et *backward*

Nous proposons ici un algorithme d'apprentissage d'une DE stochastique basé sur l'algorithme de Zhang & Shasha. Encore une fois, notre approche repose sur une adaptation de l'algorithme EM, pour lequel nous avons conçu deux nouvelles fonctions auxiliaires *forward* ($\boldsymbol{\alpha}$) et *backward* ($\boldsymbol{\beta}$) respectivement décrites dans les Algorithmes 3.36 et 3.37. Notons que les caractères en gras sont utilisés pour les appels récursifs tandis que les caractères de fonte normale décrivent des valeurs stockées temporairement.

Nous pouvons également noter que les deux fonctions $\boldsymbol{\alpha}$ et $\boldsymbol{\beta}$ retournent la quantité $\sum_{e \in E(x,y)} p_\delta(e)$, c'est-à-dire la somme des probabilités de chacun des chemins permettant la transformation de l'arbre x en y. Au delà du fait qu'elles autorisent le calcul de la DE, elles permettent également d'estimer les espérances des opérations (voir Fig. 3.38 et la section suivante) pendant l'étape d'*Expectation* de l'algorithme EM.

Algorithme 3.36 : $\boldsymbol{\alpha}(\{a_1,..,a_T\},\{b_1,..,b_V\})$

Données : Deux forêts $\{a_1,..,a_T\},\{b_1,..,b_V\}$.
Soit α une matrice de dimension $(T+1)\times(V+1)$;
$\alpha[\{\},\{\}] \leftarrow 1$;
pour t *de 0 à* T **faire**
  pour v *de 0 à* V **faire**
    si $(t>0)$ *ou* $(v>0)$ **alors**
      $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] \leftarrow 0$;
    si $t>0$ **alors**
      $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] \leftarrow$
      $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] + \delta(l(\rho(a_t)),\lambda)\times\alpha[\{a_1,..,f(a_t)\},\{b_1,..,b_v\}]$;
    si $v>0$ **alors**
      $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] \leftarrow$
      $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] + \delta(\lambda,l(\rho(b_v)))\times\alpha[\{a_1,..,a_t\},\{b_1,..,f(b_v)\}]$;
    si $(t>0)$ *et* $(v>0)$ **alors**
      $\alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] \leftarrow \alpha[\{a_1,..,a_t\},\{b_1,..,b_v\}] + \boldsymbol{\alpha}(f(a_t),f(b_v))\times$
      $\delta(l(\rho(a_t)),l(\rho(b_v)))\times\alpha[\{a_1,..,a_{t-1}\},\{b_1,..,b_{v-1}\}]$;

retourner $\alpha[\{a_1,..,a_T\},\{b_1,..,b_V\}]$

Expectation

Pendant l'étape d'*Expectation*, rappelons que le but est d'estimer les espérances des événements cachés, c'est-à-dire ici les opérations d'édition utilisées pour transformer un arbre d'entrée en un arbre de sortie. Ces espérances sont toujours stockées dans une matrice auxiliaire γ de taille $(|\Sigma|+1)\times(|\Sigma|+1)$. Ce processus prend une paire d'apprentissage (x,y) en entrée. Ensuite, pour chaque paire de sous-arbres (a_t,b_v) où a_t est un sous-arbre de x et b_v un sous-arbre de y, il accumule les espérances des trois opérations d'édition possibles. Le pseudo-code de l'étape d'*Expectation* est décrit dans l'Algorithme 3.39 qui utilise les définitions suivantes.

Définition 3.6 (Parcours postfixé) *Un parcours postfixé d'un arbre $x=r(a_1,..,a_T)$ est obtenu par la visite récursive des sous-arbres a_t, $t=1..T$ et finalement de la racine r. La numérotation postfixée donne un numéro à chaque nœud de x suivant le parcours postfixé.*

Définition 3.7 (ϕ_α et ϕ_β) *Soit $x=r(a_1,\ldots,a_T)$ un arbre ordonné. nous définissons $\phi_\alpha:\mathcal{V}(x)\rightarrow\mathcal{T}(\Sigma)$ comme étant la fonction qui prend en entrée un nœud r et retourne la forêt ordonnée composée des sous-arbres dont les racines ont leurs numéros strictement inférieurs à celui de r suivant le parcours postfixé de l'arbre. Et soit $\phi_\beta:\mathcal{V}(x)\rightarrow\mathcal{T}(\Sigma)$ une fonction qui prend un nœud r et retourne l'arbre ordonné dont les nœuds ont un numéro strictement inférieur à celui de r suivant le parcours postfixé inverse de l'arbre.*

La Figure 3.38 montre un exemple de parcours postfixé (en chiffre arabe) et postfixé inverse (en chiffre romain). Considérons le nœud numéroté $4|III$ sur l'arbre de gauche, ϕ_α retourne la forêt composée des 3 sous-arbres qui ont pour racine les nœuds d'étiquettes $1|VI$, $2|V$ et $3|IV$, ϕ_β retourne le sous-arbre dont la racine est numérotée $6|I$ avec comme

Algorithme 3.37 : $\boldsymbol{\beta}(\{a_1, .., a_T\}, \{b_1, .., b_V\})$

Données : Deux arbres $\{a_1, .., a_T\}, \{b_1, .., b_V\}$.
Soit β une matrice de dimension $(T+1) \times (V+1)$;
$\beta[\{\}, \{\}] \leftarrow 1$;
pour t *de* T *à 0* **faire**
  pour v *de* V *à 0* **faire**
    si $(t < T)$ *ou* $(v < V)$ **alors**
      $\beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] \leftarrow 0$;
    si $t < T$ **alors**
      $\beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] \leftarrow \beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] + \delta(l(\rho(a_t)), \lambda) \times \boldsymbol{\beta}(\{a_t, .., f(\rho(a_T))\}, \{b_v, .., b_V\})$;
    si $v < V$ **alors**
      $\beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] \leftarrow \beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] + \delta(\lambda, l(\rho(b_v))) \times \boldsymbol{\beta}(\{a_t, .., a_T\}, \{b_v, .., f(\rho(b_V))\})$;
    si $(t < T)$ *et* $(v < V)$ **alors**
      $\beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] \leftarrow \beta[\{a_t, .., a_T\}, \{b_v, .., b_V\}] + \delta(l(\rho(a_t)), l(\rho(b_v))) \times \boldsymbol{\beta}(f(a_T), f(b_V)) \times \boldsymbol{\beta}(\{a_t, .., a_{T-1}\}, \{b_v, .., b_{V-1}\})$;

retourner $\beta[\{a_1, .., a_T\}, \{b_1, .., b_V\}]$;

fils le nœud numéroté $5|II$.

Rappelons que l'algorithme *Expectation* calcule l'espérance du nombre de fois où chaque opération est utilisée pour changer l'arbre x en y. Pour chaque opération d'édition (dont la probabilité est donnée par δ), nous considérons non seulement la probabilité des chemins qui mènent à cette opération (donnée par $\boldsymbol{\alpha}$) mais aussi celle des chemins qui finissent la transformation (donnée par $\boldsymbol{\beta}$). Tandis que la délétion et l'insertion sont facilement interprétables, l'opération de substitution nécessite quelques explications. La Figure 3.38 décrit la substitution du nœud $4|III$ en $4|IV$. Cette opération nécessite le calcul de la fonction $\boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) - f(a_t), \phi_\alpha(\rho(b_v)) - f(b_v))$, *i.e.* la probabilité de la paire de forêts $[\{1|VI\}, \{2|VI(1|VII)\}]$. Cette paire de forêts est constituée des sous-arbres dont les nœuds ont un numéro inférieur à 4 suivant une numérotation postfixée (donnée par ϕ_α), moins les sous-arbres qui sont les enfants de $4|III$ et $4|IV$ (donné par la fonction f). Nous estimons la valeur $\boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))$ de la paire $[\{6|I(5|II)\}, \{7|I(6|II(5|III))\}]$, des nœuds inférieurs à III pour la forêt d'entrée et IV pour celle de sortie, étant donnée une numérotation postfixée inverse. Le calcul nécessite aussi la valeur de $\boldsymbol{\alpha}(f(a_t), f(b_v))$ sur la paire $[\{2|V, 3|IV\}, \{3|V\}]$ correspondant aux enfants des nœuds traités par l'opération de substitution.

Dans cette partie, nous ne reviendrons pas sur l'étape de Maximisation qui est similaire au cas de l'apprentissage de la DE de Selkow. Dans le cas de la DE de Zhang & Shasha, notons néanmoins que la complexité algorithmique est fonction du nombre de nœuds des arbres, de leur profondeur et de leur nombre de feuilles. Elle est donc supérieure à celle dans le cas de Selkow.

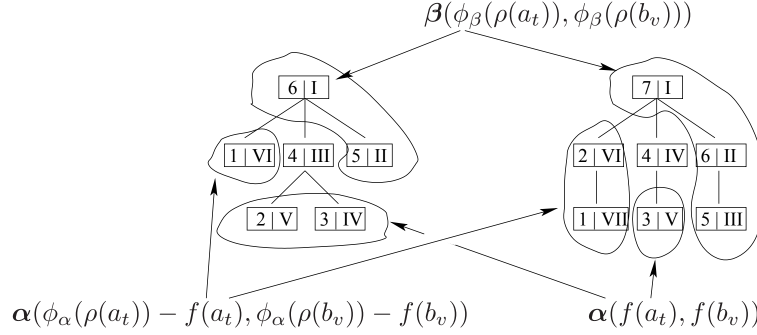
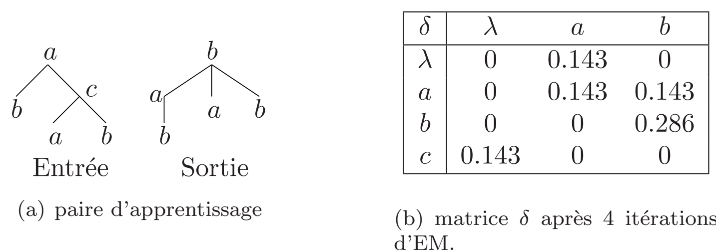


Fig. 3.38 – Illustration de l'espérance d'une opération de substitution.

Algorithme 3.39 : *Expectation*(x, y)

Données : Deux arbres x et y.
Soit λ l'arbre vide ;
pour chaque a_t *t.q.* $\rho(a_t) \in \mathcal{V}(x) \cup \lambda, b_v$ *t.q.* $\rho(b_v) \in \mathcal{V}(y) \cup \lambda$ **faire**
- **si** $a_t \neq \lambda$ **alors**
  - $\gamma(l(\rho(a_t)), \lambda) \leftarrow \gamma(l(\rho(a_t)), \lambda) + \frac{1}{\boldsymbol{\alpha}(x,y)} \times \boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)), \phi_\alpha(\rho(b_v)) \cup \{b_v\}) \times \delta(l(\rho(a_t)), \lambda) \times \boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))$;
- **si** $b_v \neq \lambda$ **alors**
  - $\gamma(\lambda, l(\rho(b_v))) \leftarrow \gamma(\lambda, l(\rho(b_v))) + \frac{1}{\boldsymbol{\alpha}(x,y)} \times \boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) \cup \{a_t\}, \phi_\alpha(\rho(b_v))) \times \delta(\lambda, l(\rho(b_v))) \times \boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))$;
- **si** $(a_t \neq \lambda)$ *et* $(b_v \neq \lambda)$ **alors**
  - $\gamma(l(\rho(a_t)), l(\rho(b_v))) \leftarrow \gamma(l(\rho(a_t)), l(\rho(b_v))) + \frac{1}{\boldsymbol{\alpha}(x,y)} \times \boldsymbol{\alpha}(\phi_\alpha(\rho(a_t)) - f(\rho(a_t)), \phi_\alpha(\rho(b_v)) - f(\rho(b_v))) \times \boldsymbol{\alpha}(f(\rho(a_t)), f(\rho(b_v))) \times \delta(l(\rho(a_t)), l(\rho(b_v))) \times \boldsymbol{\beta}(\phi_\beta(\rho(a_t)), \phi_\beta(\rho(b_v)))$;

Exemple. *Pour aider le lecteur à comprendre notre algorithme, nous présentons ici une exécution de celui-ci sur un exemple jouet. L'alphabet d'entrée est* $\Sigma_1 = \{a, b, c\}$*, l'alphabet de sortie* $\Sigma_2 = \{a, b\}$ *et l'ensemble d'apprentissage est composé de la seule paire* $[a(b, c(a, b)), b(a(b), a, b)]$ *(voir Fig. 3.40(a)). L'algorithme converge après 4 itérations de l'algorithme. La matrice* δ *apprise (initialisée avec des valeurs aléatoires) est décrite dans le Tableau 3.40(b). Nous pouvons noter que notre algorithme a convergé vers une cible potentielle. En effet, une solution optimale parait consister en : (i) changer la racine* a *en* b*, (ii) insérer le symbole* a *qui devient le père de* b *et garder inchangé le symbole* b*, (iii) supprimer le symbole* c*, et (iv) garder inchangés les symboles* a *et* b*.* ○



(a) paire d'apprentissage

δ	λ	a	b
λ	0	0.143	0
a	0	0.143	0.143
b	0	0	0.286
c	0.143	0	0

(b) matrice δ après 4 itérations d'EM.

Fig. 3.40 – Exemple d'apprentissage à partir d'une paire.

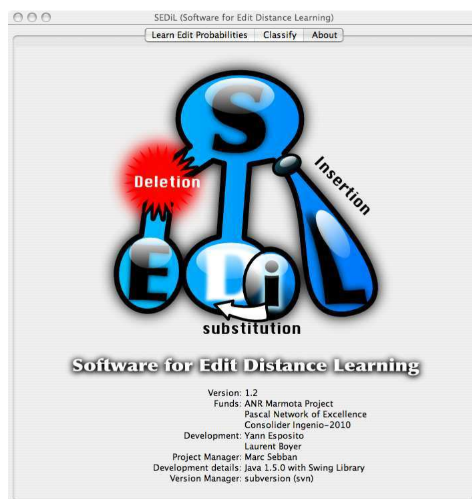


Fig. 3.41 – La plateforme SEDiL

3.6 Plateforme SEDiL

Comme nous l'avons vu dans les différentes sections de ce chapitre, le calcul de la distance d'édition entre arbres est assez technique nécessitant l'appel à de la programmation dynamique avec un double niveau de récursion. Ce qui est vrai sur la distance classique entre arbres, l'est encore plus lorsque nous souhaitons apprendre une distance stochastique. Afin de diffuser notre travail au niveau de la communauté en machine learning et permettre des expérimentations poussées, nous avons développé une plateforme logicielle en libre accès sur le Web. Elle se nomme SEDiL, pour Software for Edit Distance Learning. Cette plateforme regroupe les versions non-stochastiques des distances d'édition entre chaînes [Lev66] et arbres [Sel77, ZS89], les méthodes stochastiques sur les chaînes présentées en Section 2.3.2 et également tous les modèles stochastiques sur les arbres que nous venons de présenter dans ce chapitre.

D'un point de vue technique, SEDiL est écrit en Java et est disponible en ligne accessible à l'adresse suivante : http://labh-curien.univ-st-etienne.fr/SEDiL. Nous pouvons utiliser la plateforme en local par l'intermédiaire de la technologie Java Web Start ou à distance en utilisant une applet. À ce jour, la page a déjà été visitée plus de

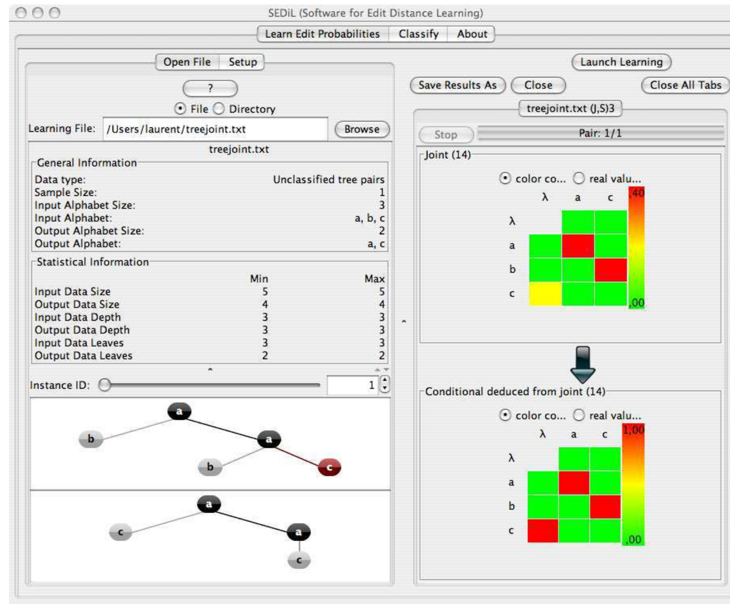


Fig. 3.42 – Capture d'écran de SEDiL : phase d'apprentissage

1000 fois.

Le développement de cette plateforme a démarré durant le post-doctorat de Yann Esposito dans le cadre de l'ANR Marmota[3] : MAchine learning pRobabistic MOdels Tree lAnguages. SEDiL a ensuite été complété durant cette thèse avec l'intégration de tous les nouveaux modèles d'arbres proposés.

SEDiL est partagé en deux parties distinctes : une partie dédiée à l'apprentissage et une autre résolvant des problèmes de classification.

Apprentissage (Fig. 3.42) SEDiL autorise l'utilisation de chaînes ou d'arbres. Nous pouvons directement lui donner des paires d'exemples et lancer l'algorithme d'apprentissage de distance avec les paramètres (type de distance, nombre d'itérations, etc.) fixés dans le sous-onglet *setup*. Si nous lui fournissons seulement des singletons, diverses méthodes sont proposées pour construire les paires : génération aléatoire, utilisation d'une DE classique (poids tous à 1), où utilisation d'une DE stochastique.

Le résultat de l'apprentissage peut être présenté de manière graphique avec un code couleur pour indiquer les opérations d'édition les plus utilisées ou directement avec les valeurs calculées à la fin de l'exécution de l'algorithme.

Classification (Fig. 3.43) Cette partie de SEDiL permet de tester les différentes distances (apprises ou non) dans une tâche de classification de chaînes ou d'arbres, à l'aide d'un classifieur par k-plus-proches voisins. L'utilisateur peut charger des fichiers

[3]ANR-05-MMSA-0016

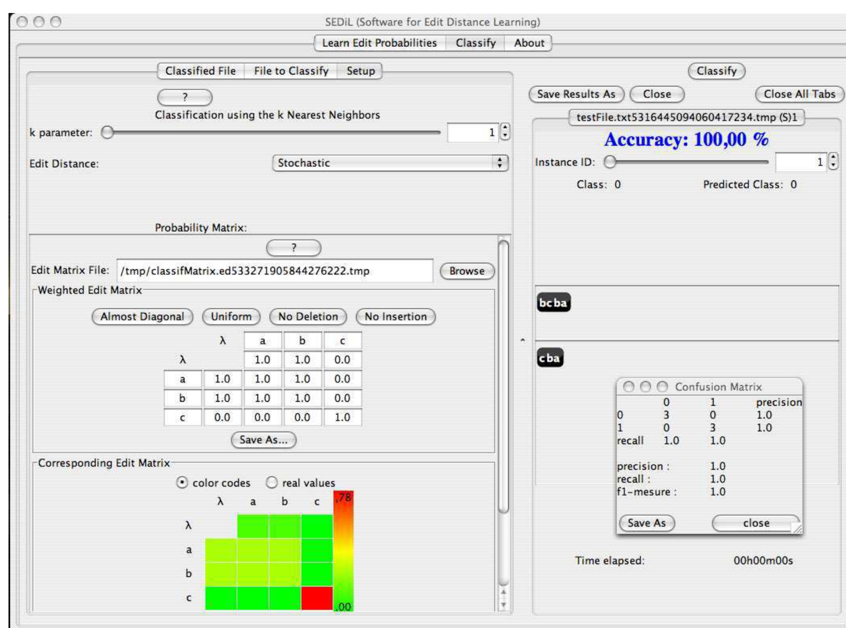


Fig. 3.43 – Capture d'écran de SEDiL : phase de classification

d'apprentissage et de test, ainsi qu'éventuellement une matrice de coûts d'opérations d'édition.

Atouts À notre connaissance, SEDiL est la seule plateforme regroupant plusieurs modèles d'apprentissage de distance d'édition.

Les différents modèles implémentés sont :
- pour les chaînes de caractères :
  - l'apprentissage d'une similarité jointe entre chaînes [RY98],
  - l'apprentissage d'une similarité conditionnelle entre chaînes [OS06],
- pour les arbres :
  - l'apprentissage d'une similarité jointe entre arbres suivant la définition de Selkow [BBHS08],
  - l'apprentissage d'une similarité conditionnelle entre arbres suivant la définition de Selkow [BBHS08],
  - l'apprentissage d'une similarité jointe entre arbres suivant la définition de Zhang & Shasha [BHS07],
  - l'apprentissage d'une similarité conditionnelle entre arbres suivant la définition de Zhang & Shasha [BHS07].

De plus SEDiL :
- donne des statistiques sur les données d'apprentissage,
- permet de comparer les résultats avec les calculs classiques des DE,
- autorise des contraintes de régularisation sur les délétions et insertions durant l'apprentissage.

Notons que SEDiL est utilisé dans le cadre universitaire mais aussi par certaines entreprises. En effet, d'un point de vue académique, il sert de support de cours à des enseignements en apprentissage automatique de la faculté des sciences de Saint-Étienne. De plus, dans le contexte de l'ANR Bingo2, SEDiL est utilisé pour la recherche de similarités entre séquences d'ADN. Enfin, le laboratoire DLSI de l'université d'Alicante l'utilise dans le cadre d'expérimentations concernant des données musicales [HQRS08]. Mais il faut aussi noter que SEDiL a aussi été utilisé dans des tâches de reconnaissance de caractères manuscrits par l'entreprise A2IA[4].

3.7 Conclusion

Dans ce chapitre, nous avons présenté notre premier ensemble de contributions : une extension aux arbres des modèles d'apprentissage sur les paramètres de similarité d'édition qui avaient jusque là été limités au traitement de chaînes [RY98, OS06]. Les modèles proposés sont basés sur l'algorithme itératif *Expectation-Maximisation* qui maximise la vraisemblance d'un échantillon d'apprentissage. Les expérimentations sur des données artificielles nous ont permis de mettre en avant les problèmes liés aux distributions d'entrée des exemples d'apprentissage. Si le nombre d'exemples est important, nous préférons apprendre un modèle discriminatif (distribution conditionnelle) plutôt qu'un modèle génératif (distribution jointe).

Problème de divergence. Lors de diverses expérimentations sur des données artificielles et réelles, nous avons remarqué des problèmes de divergence sur les opérations d'insertion et de délétion. Un système de régularisation a alors été mis au point dans la plateforme SEDiL. Il permet de contrôler la proportion de délétion et d'insertion que nous voulons autoriser pendant le processus d'apprentissage.

Pour rappel, pendant le processus d'apprentissage, la fonction objectif est la suivante :

$$Q(\theta_n, \theta_{n+1}) = \mathbf{E}[\ln(p(O, \mathcal{U}|O_{n+1}))|O, \theta_n],$$

avec O les données observables et $\mathcal{U}$ les données non-observables. L'idée est d'ajouter un terme permettant de minimiser l'impact des délétions et des insertions. Le terme régularisateur est le suivant :

$$T(\theta_n, \theta_{n+1}) = \alpha \sum_{l \in \Sigma} p((\lambda|l)|\theta_n) \ln \delta(\lambda|l) + \beta \sum_{l' \in \Sigma} p((l'|\lambda)|\theta_n) \ln \delta(l'|\lambda),$$

avec $\alpha, \beta \in [0, 1]$. Notons que si un des paramètres est égal à zéro alors il n'y a pas de régularisation, et si il est égal à un alors les opérations de délétion ou d'insertion sont supprimées.

La nouvelle fonction objectif à optimiser est donc :

$$Q^\star(\theta_n, \theta_{n+1}) = Q(\theta_n, \theta_{n+1}) - T(\theta_n, \theta_{n+1}).$$

[4] http://www.a2ia.com.

```
<ul>
  <li> 3 Un modèle memoryless pour les arbres </li>
    <ul>
      <li> 3.3 DE selon Selkow </li>
        <ul>
          <li> 3.3.1 Opérations et coûts d'édition </li>
          <li> 3.3.2 Algorithme classique de calcul de la DE selon Selkow </li>
        </ul>
      <li> 3.4 Résultats expérimentaux </li>
    </ul>
  <li> 4 Un modèle non-memoryless contraint </li>
</ul>
```

Fig. 3.44 – Extrait de table des matières sous un format HTML.

Limites. Les modèles présentés dans ce chapitre ne permettent pas de prendre en compte le contexte de l'opération d'édition. En effet, il pourrait être utile de donner des poids différents à la même opération d'édition suivant que l'on se trouve à telle ou telle profondeur de l'arbre. Pour expliquer ce phénomène, prenons par exemple un extrait de table des matières représenté sous un format HTML dans la Figure 3.44. Dans le contexte d'un arbre HTML, il semble que l'importance de la délétion d'un nœud `<li>` dépende de la profondeur à laquelle celui-ci se trouve. En effet, si nous prenons l'exemple de la Figure 3.44, les délétions des nœuds `3` et `4` vont détruire la structure du document alors que les délétions des nœuds `3.3.x` vont avoir comme conséquence une diminution de la granularité de la table des matières.

De plus, supposons que pour un domaine donné, nous disposions d'un ensemble de connaissances *a priori*. Il semble que ces connaissances pourraient être utilisées pendant le processus d'apprentissage ou directement intégrées dans le modèle appris. Prenons par exemple, le cas de l'expérience cognitive suivante [BHMS08] : nous présentons durant quelques instants (sur un écran par exemple) une suite de mots à un individu. Celui-ci doit ensuite répéter l'ensemble de ces mots (en les saisissant sur un clavier). Un ensemble d'hypothèses cognitives on été validées montrant que le candidat sera en mesure de répéter plus facilement les premiers mots entendus, ce qui est appelé l'effet de *primauté* et les mots de la fin de la liste, l'effet de *récence*. Si nous voulons utiliser ces informations dans un processus informatique pour modéliser ces mêmes effets, nous devons alors être capable d'apprendre un modèle à trois états (un pour l'effet de primauté, un pour les mots du milieu de liste, et un pour l'effet de récence). Le constat est qu'aucun des modèles proposés dans la littérature ne permet d'intégrer de telles connaissances.

Le chapitre suivant propose une solution à cette limite.

4 Un modèle d'édition non-memoryless contraint

Résumé

Dans ce chapitre, nous présentons une méthode de type EM pour apprendre un modèle conditionnel contraint de distance d'édition stochastique entre chaînes. Nous utilisons une machine à états permettant de modéliser différents contextes d'édition. Pour ajouter de la connaissance du domaine, les transitions entre états sont contraintes. Ces contraintes s'expriment sous la forme d'un ensemble de fonctions booléennes. Nous montrons que ce type de modèle contraint est une généralisation stricte des pair-HMMs, puis nous évaluons notre modèle sur une problématique en biologie moléculaire dont le but est de détecter automatiquement des sites de transcription de gènes.

Ce chapitre est basé sur les contributions suivantes :

– Laurent Boyer, Amaury Habrard, Fabrice Muhlenbach, et Marc Sebban. Learning string edit similarities using constrained finite state machines. Dans *Conférence Francophone sur l'Apprentissage Automatique (CAp)*, pages 37–52. Cépaduès, 2008.

– Laurent Boyer, Olivier Gandrillon, Amaury Habrard, Mathilde Pellerin, et Marc Sebban. Learning constrained edit state machines. Dans *International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 734–741. IEEE Computer Society, 2009.

4.1 Introduction

Dans le chapitre précédent, nous avons présenté un modèle permettant d'apprendre les paramètres d'une distance d'édition entre arbres dans le contexte des distances de Selkow et de Zhang & Shasha. En conclusion, nous avons évoqué deux perspectives pour étendre nos modèles : d'une part, permettre l'apprentissage de coûts différents selon la position (ou plus généralement le contexte) de l'opération, et d'autre part, autoriser l'intégration de connaissances du domaine. Dans ce chapitre, nous abordons ces deux points en nous limitant pour des raisons algorithmiques au cas de la distance d'édition entre chaînes.

Pour pouvoir appliquer des coûts différents à une même opération d'édition (c'est à dire mettant en jeu les mêmes symboles) dans un contexte différent, nous allons utiliser un modèle à états finis. Des machines telles que les pair-HMMs, ou encore les automates à états multiples, sont des solutions d'un point de vue structurel, mais l'intégration de connaissances du domaine y est difficile et généralement limitée. Nous proposons, dans ce chapitre, un nouveau modèle à états, inspiré du principe des pair-HMMs mais autorisant la prise en compte de contraintes, notamment non régulières (*i.e.* non représentables de manière équivalente par un automate fini, nous définirons ces contraintes plus tard dans le document). Nous proposons d'introduire ces contraintes au niveau des transitions entre chacun des états. Nous appelons ces modèles contraints des machines à états contraintes ou *Constrained State Machines* (CSM). Leur apprentissage est effectué grâce à une adaptation de l'algorithme EM prenant en compte, cette fois-ci, les contraintes manipulées. Comme nous le verrons, notre modèle à états contraint permet de définir des classes de distributions plus générales que celles définies par les pair-HMMs.

Le reste du chapitre est organisé comme suit : la Section 2 présente notre machine et l'algorithme d'apprentissage des différents paramètres. Dans la Section 3, nous comparons formellement notre modèle aux pair-HMMs. Enfin, en Section 4, nous évaluons l'intérêt de notre modèle sur une tâche de biologie moléculaire qui est au cœur du projet ANR BINGO2.

4.2 CSM

4.2.1 Cadre général

Notre machine à états contrainte (CSM) s'inspire du principe des pair-HMMs (nous montrerons dans la section suivante qu'elle est, en fait, une stricte généralisation de ces modèles). En effet, comme pour les pair-HMMs, un CSM est composé d'un ensemble fini d'états reliés par des transitions, et pour chaque état, une distribution sur les opérations d'édition est définie. La principale différence entre un CSM et un pair-HMM provient de l'utilisation de contraintes sur la chaîne d'entrée X guidant le passage d'un état à un autre. Chaque contrainte peut avoir une expression globale (par exemple "il y a un nombre pair de lettres dans la chaîne") ou locale autour du caractère étudié pendant le processus d'édition (par exemple "est-ce que le symbole courant est suivi de deux lettres 'a' ?").

Dans ce nouveau cadre, nous définissons une contrainte sous la forme d'un ensemble de fonctions booléennes. Cependant, dans le but de n'avoir qu'une seule contrainte active entre deux états et réduire ainsi la complexité algorithmique de notre processus d'apprentissage, nous imposons de n'avoir exactement qu'une seule fonction booléenne vraie.

Contrairement à la *fouille de données* où une contrainte aura pour conséquence de permettre d'élaguer l'espace de recherche et ainsi de limiter le nombre de motifs répondant à une requête, nos contraintes nous permettent de guider l'édition de la chaîne d'entrée à travers l'ensemble des états. Prenons par exemple la contrainte assignée à l'état 0 de la Figure 4.1 qui analyse la lettre précédemment éditée. Supposons que nous venons

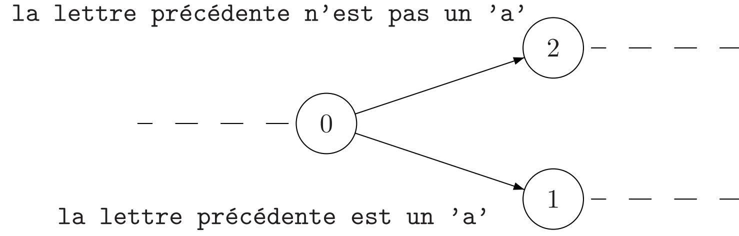


Fig. 4.1 – Exemple de contrainte. Si la lettre de la chaîne d'entrée, traitée dans l'état 0, est un 'a', alors la prochaine étape du calcul se fera dans l'état 1, sinon elle se réalisera dans l'état 2.

d'éditer la lettre 'a' dans l'état 0, nous allons alors continuer l'étude de la transformation de la chaîne d'entrée dans l'état 1. Dans le cas contraire, la transduction continuera dans l'état 2.

Définissons maintenant formellement ce que nous appelons *contrainte*.

Définition 4.1 (Contrainte) *Une contrainte c est un ensemble fini de fonctions booléennes $c_k : \Sigma^\star \times \mathbb{N} \rightarrow \{true, false\}$, $1 \leq k \leq m$. Chacune de ces fonctions est définie sur une chaîne d'entrée X et une position t dans X. De plus, $c_k(X,t)$ est la valeur booléenne associée à la $k^{ième}$ fonction de c pour la chaîne X à la position t ($0 \leq t \leq |X|$). Nous imposons qu'une contrainte doit respecter la propriété suivante : pour tout $X \in \Sigma^\star$ et pour tout $t \in \mathbb{N}$, il existe exactement une fonction c_k de c telle que $c_k(X,t)$ soit vraie.*

Dans les pair-HMMs, les probabilités sortantes pour un état donné sont apprises sans connaissance préalable. Quelle que soit la paire de chaînes traitée dans l'état, chaque transition sortante n'a *a priori* aucune raison de ne pas être utilisée. Dans notre modèle, nous incorporons de la connaissance du domaine pendant le processus d'apprentissage sous la forme de contraintes, afin de pouvoir contrôler l'utilisation des transitions de ce modèle. Plus précisément, les transitions sortantes sont utilisées si et seulement si la fonction booléenne $c_k(X,t)$ d'une contrainte c donnée est satisfaite à l'instant t pour la chaîne X. Notons que nous assignons à chaque état une contrainte, ce qui signifie que les différentes transitions correspondent aux différentes modalités de la contrainte. Considérant que c_{q_i} dénote la contrainte assignée à l'état q_i, nous pouvons désormais définir formellement notre CSM.

Définition 4.2 (CSM) *Une machine à états contrainte est un 6-uplet $\langle \Sigma, \mathcal{Q}, C, T, \delta, \pi \rangle$ avec Σ un alphabet fini, $\mathcal{Q}$ un ensemble fini d'états, C un ensemble de contraintes et :*

- *$T : \mathcal{Q} \times C \times \mathcal{Q} \rightarrow [0,1]$ est une fonction de transition. $T(q_j|q_i, c_{q_i,k}(X,t))$ dénote la probabilité d'aller dans l'état q_j sachant que nous sommes dans l'état q_i et que la contrainte $c_{q_i,k}(X,t)$ est vérifiée. Étant donné un état q_i et une contrainte $c_{q_i,k}$,*

les transitions sortantes doivent vérifier :

$$\sum_{q_j \in \mathcal{Q}} T(q_j|q_i, c_{q_i,k}) = 1. \quad (4.1)$$

- *δ est une famille de $|\mathcal{Q}|$ matrices. Pour chaque état q_i, δ_{q_i} doit respecter l'équation suivante (déjà définie dans les chapitres précédents) :*

$$\forall a \in \Sigma, \sum_{b \in \Sigma \cup \{\lambda\}} \delta_{q_i}(b|a) + \sum_{b \in \Sigma} \delta_{q_i}(b|\lambda) = 1. \quad (4.2)$$

- *$\pi : \mathcal{Q} \rightarrow [0,1]$ est la fonction de probabilité initiale, celle-ci doit satisfaire la contrainte suivante :*

$$\sum_{q \in \mathcal{Q}} \pi(q) = 1. \quad (4.3)$$

Les Équations 4.1, 4.2 et 4.3, déjà utilisées dans d'autres modèles, par exemple les pair-HMMs, permettent de définir une distribution conditionnelle sur les scripts d'édition.

Définissons maintenant la sémantique opérationnelle d'un CSM.
Soient $\mathcal{C} = \langle \Sigma, \mathcal{Q}, C, T, \delta, \pi \rangle$ un CSM et $(X, Y) \in (\Sigma^\star)^2$ un couple de mots à analyser.

- Une configuration de $\mathcal{C}$ est caractérisée par un 4-uplet (S, u, v, p) tel que $S \in \mathcal{Q}$ est l'état courant, $u \in \Sigma^\star$ correspond à un suffixe de X, $v \in \Sigma^\star$ correspond à un suffixe de Y et p est la probabilité de la paire (s, t) avec $X = s.u$ et $Y = t.v$.
- La configuration (S', u', v', p') est dérivable en une étape de la configuration (S, u, v, p) (notée $(S, u, v, p) \rightarrow (S', u', v', p')$) s'il existe $(a, b) \in (\Sigma \cup \{\lambda\})^2 \backslash \{\lambda, \lambda\}$ tels que $u = a.u'$ et $v = b.v'$ avec $\delta_S(b|a) \neq 0$, $u', v' \in \Sigma^\star$, et qu'il existe une modalité k pour laquelle la contrainte C_S est satisfaite et tel que $T(S'|S, c_{S,k}) \neq 0$ et $p' = p \times \delta_S(b|a) \times T(S'|S, c_{S,k})$.
- La configuration $(S_f, \lambda, \lambda, p)$ est dérivable en plusieurs étapes de la configuration (S, X, Y, π_S) (notée $(S, X, Y, \pi_S) \xrightarrow{\star} (S_f, \lambda, \lambda, p)$) si $(S_f, \lambda, \lambda, p)$ peut être obtenu de (S, X, Y, π_S) par une succession de dérivations en une étape et tel que $p \neq 0$. Nous appellons cette succession de dérivations un chemin auquel nous associons la probabilité p.
- Soient $\mathcal{C}$ un CSM, (X, Y) une paire de mots et M l'ensemble fini des chemins de la paire (X, Y), la probabilité $p(Y|X)$ est la somme des probabilités associées à chacun des chemins de M :

$$p(Y|X) = \sum_{((S,X,Y,\pi_S) \xrightarrow{\star} (S',\lambda,\lambda,p_i)) \in M} p_i.$$

Nous présentons dans le paragraphe suivant un algorithme en programmation dynamique pour calculer la probabilité $p(Y|X)$.

4.2.2 Calcul de $p(Y|X)$

Le calcul de la probabilité $p(Y|X)$, c'est-à-dire la probabilité de transformer par des opérations d'édition la chaîne X en Y, peut être à nouveau réalisé avec une fonction

forward. Soient $X = x_1 \dots x_T$ et $Y = y_1 \dots y_V$ (x_0 et y_0 dénotent la chaîne vide), la fonction *forward* (α) est définie récursivement pour calculer la probabilité d'un préfixe d'une chaîne de sortie étant donné le préfixe d'une chaîne d'entrée. Ce calcul peut être fait à partir de chacun des états $q \in \mathcal{Q}$. Pour simplifier les notations, $c_{q,k}(X,t)$ sera noté $c_q(x_t)$ dans la suite sans mentionner la fonction booléenne c_q satisfaite à la t^{ieme} position de X.

$$\begin{array}{l}\alpha_{q_i}(y_0|x_0) = \pi(q_i),\\ \alpha_{q_i}(y_v|x_t) =\\ (\sum_{q_j\in\mathcal{Q}} \alpha_{q_j}(y_{v-1}|x_{t-1}) \cdot \delta_{q_j}(y_v|x_t) \cdot T(q_i|q_j, c_{q_j}(x_t)))_{v\geq 1, t\geq 1}\\ +(\sum_{q_j\in\mathcal{Q}} \alpha_{q_j}(y_v|x_{t-1}) \cdot \delta_{q_j}(\lambda|x_t) \cdot T(q_i|q_j, c_{q_j}(x_t)))_{t\geq 1}\\ +(\sum_{q_j\in\mathcal{Q}} \alpha_{q_j}(y_{v-1}|x_t) \cdot \delta_{q_j}(y_v|\lambda) \cdot T(q_i|q_j, c_{q_j}(x_t)))_{v\geq 1}.\end{array}$$

Dans ce calcul, les différences par rapport aux modèles *memoryless*, vus aux chapitres précédents, viennent de la somme effectuée sur tous les états et de la fonction de transition. Par rapport aux pair-HMMs, nous devons prendre en compte les contraintes dans la fonction de transition.

Selon un principe similaire, nous pouvons aussi utiliser la fonction *backward* (β) pour calculer les probabilités à partir des suffixes de X et de Y.

$$\begin{array}{l}\beta_{q_i}(y_{V+1}|x_{T+1}) = 1,\\ \beta_{q_i}(y_v|x_t) =\\ (\delta_{q_i}(y_v|x_t) \cdot \sum_{q_j\in\mathcal{Q}} T(q_j|q_i, c_{q_i}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1}))_{v\leq V, t\leq T}\\ +(\delta_{q_i}(\lambda|x_t) \cdot \sum_{q_j\in\mathcal{Q}} T(q_j|q_i, c_{q_i}(x_t)) \cdot \beta_{q_j}(y_v|x_{t+1}))_{t\leq T}\\ +(\delta_{q_i}(y_v|\lambda) \cdot \sum_{q_j\in\mathcal{Q}} T(q_j|q_i, c_{q_i}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_t))_{v\leq V}.\end{array}$$

Dans le calcul des fonctions *forward* et *backward*, nous devons prendre en compte l'ensemble des états possibles. Cela implique un calcul en programmation dynamique dont la complexité est en $O(|Y| \times |X| \times |\mathcal{Q}| \times f(X))$ où $f(X)$ est une borne supérieure de la complexité du calcul des fonctions booléennes. Cette complexité est supposée polynomiale en la taille de X et est généralement linéaire dans notre cas.

Nous pouvons maintenant calculer $p(Y|X)$ en utilisant les fonctions précédentes :

$$p(Y|X) = \sum_{q\in\mathcal{Q}} \alpha_q(y_V|x_T) = \sum_{q\in\mathcal{Q}} \pi(q)\beta_q(y_1|x_1).$$

4.2.3 Apprentissage des paramètres

Pour apprendre les paramètres de notre modèle, nous devons apprendre une matrice δ_{q_i} pour chaque état q_i du CSM. En d'autres termes, cela revient à dire que nous apprenons une similarité d'édition stochastique par contexte. De plus, nous devons estimer les probabilités des transitions et les probabilités initiales. Pour cette tâche, nous utilisons à nouveau une adaptation de l'algorithme EM. L'originalité de celle-ci provient de la normalisation qui doit prendre en compte les contraintes.

Étape *Expectation*

Pour rappel, l'étape d'*Expectation* correspond à l'estimation du nombre de fois que chaque opération d'édition est utilisée dans l'échantillon d'apprentissage LS.

L'espérance du nombre de fois qu'un état initial est utilisé est stockée dans $\gamma_{\pi_{q_i}}$.

$$\gamma_{\pi_{q_i}} = \sum_{(X,Y)\in LS} \frac{\beta_{q_i}(y_1|x_1).\pi(q_i)}{p(Y|X)}.$$

L'espérance de chaque opération d'édition $(b|a)$ pour un état q_i donné est conservée dans la variable $\gamma_{\delta_{q_i}(b|a)}$.

$$\gamma_{\delta_{q_i}(y_v|x_t)} = \sum_{(X,Y)\in LS} \sum_{t=1}^{|X|}\sum_{v=1}^{|Y|} \frac{\alpha_{q_i}(y_{v-1}|x_{t-1}) \cdot \delta_{q_i}(y_v|x_t) \cdot \sum_{q_j} T(q_j|q_i, c_{q_i}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1})}{p(Y|X)}.$$

$$\gamma_{\delta_{q_i}(\lambda|x_t)} = \sum_{(X,Y)\in LS} \sum_{t=1}^{|X|}\sum_{v=1}^{|Y|} \frac{\alpha_{q_i}(y_v|x_{t-1}) \cdot \delta_{q_i}(\lambda|x_t) \cdot \sum_{q_j} T(q_j|q_i, c_{q_i}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1})}{p(Y|X)}.$$

$$\gamma_{\delta_{q_i}(y_v|\lambda)} = \sum_{(X,Y)\in LS} \sum_{t=1}^{|X|}\sum_{v=1}^{|Y|} \frac{\alpha_{q_i}(y_{v-1}|x_t) \cdot \delta_{q_i}(y_v|\lambda) \cdot \sum_{q_j} T(q_j|q_i, c_{q_i}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1})}{p(Y|X)}.$$

Enfin, les occurrences des transitions entre états sont stockées dans les variables $\gamma_{T(q_j|q_i,c_{q_i,k})}$.

$$\begin{aligned}\gamma_{T(q_j|q_i,c_{q_i,k})} = & \\ \sum_{(X,Y)\in LS} \sum_{t=1}^{|X|}\sum_{v=1}^{|Y|} \Bigg(& \frac{\alpha_{q_i}(y_{v-1}|x_{t-1}) \cdot \delta_{q_i}(y_v|x_t) \cdot \sum_{q_j} T(q_j|q_i, c_{q_i,k}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1})}{p(Y|X)} \\ & + \frac{\alpha_{q_i}(y_{v-1}|x_t) \cdot \delta_{q_i}(y_v|\lambda) \cdot \sum_{q_j} T(q_j|q_i, c_{q_i,k}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1})}{p(Y|X)} \\ & + \frac{\alpha_{q_i}(y_v|x_{t-1}) \cdot \delta_{q_i}(\lambda|x_t) \cdot \sum_{q_j} T({q_j}'|q_i, c_{q_i,k}(x_t)) \cdot \beta_{q_j}(y_{v+1}|x_{t+1})}{p(Y|X)} \Bigg).\end{aligned}$$

Notons, que pour chaque transition entre les états q_i et q_j, il y a autant de variables que de fonctions booléennes dans la contrainte assignée à l'état q_i.

Étape Maximisation

L'étape de maximisation permet de normaliser les différentes espérances pour définir une distribution de probabilité. La preuve d'optimalité de la normalisation n'est pas

donnée ici, mais une adaptation directe de celle donnée dans [DEKM98] pour les HMMs peut s'appliquer de manière naturelle. Pour définir une distribution conditionnelle, notre normalisation doit satisfaire les contraintes suivantes :

- Pour chaque état q_i, la probabilité initiale est évaluée par :

$$\pi(q_i) = \frac{\gamma_{\pi_{q_i}}}{\sum_{q_j} \gamma_{\pi_{q_j}}}.$$

- À partir de chaque q_i, pour chaque modalité de la contrainte $c_{q_i,k}$ et pour chaque état d'arrivée q_j, nous avons :

$$T(q_j|q_i, c_{q_i,k}) = \frac{\gamma_{T(q_j|q_i,c_{q_i,k})}}{\sum_{q_j} \gamma_{T(q_j|q_i,c_{q_i,k})}}.$$

- Pour chaque matrice d'édition δ_{q_i}, la normalisation déjà présentée en Section 2.3.2 pour les *memoryless* s'applique également ici :

  Pour chaque $l \in \Sigma$ et chaque $l' \in \Sigma \cup \{\lambda\}$:

$$\delta_{q_i}(l'|l) = \frac{\gamma_{\delta_{q_i}(l'|l)}}{N(l)} \times \frac{N - N(\lambda)}{N}.$$

  Et pour chaque $l' \in \Sigma$, nous avons :

$$\delta_{q_i}(l'|\lambda) = \frac{\gamma_{\delta_{q_i}(l'|\lambda)}}{N(\lambda)}.$$

  Avec

$$N = \sum_{l \in \Sigma \cup \{\lambda\}} \sum_{l' \in \Sigma \cup \{\lambda\}} \gamma_{\delta_{q_i}(l'|l)},$$

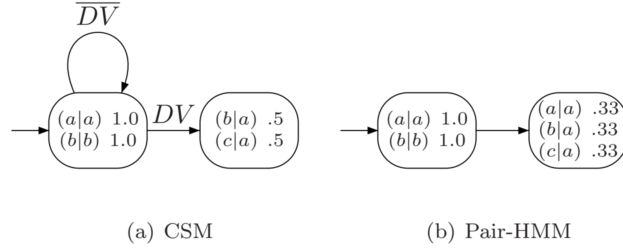
$$N(\lambda) = \sum_{l' \in \Sigma} \gamma_{\delta_{q_i}(l'|\lambda)}$$

  et

$$N(l) = \sum_{l' \in \Sigma \cup \{\lambda\}} \gamma_{\delta_{q_i}(l'|l)}.$$

Exemple. *Dans le but d'illustrer le processus d'apprentissage de notre CSM, nous allons présenter dans ce paragraphe un exemple. Dans plusieurs langues européennes, la probabilité d'avoir deux voyelles identiques consécutives dans un mot est très faible. Par exemple, en français, excepté les mots dérivés de zoo, la probabilité est quasiment nulle. Considérons l'ensemble de paires de mots artificiels* $LS = \{(aa, ab), (ba, ba), (aa, ac)\}$*, où la première chaîne de la paire est un mot erroné et la seconde sa correction. Supposons (ce n'est qu'un exemple illustratif) que cet ensemble soit représentatif d'un langage cible qui satisfait la contrainte selon laquelle la présence de mots avec des doubles voyelles identiques consécutives est impossible.*

Pour apprendre un modèle de correction, nous pouvons aisément supposer ici qu'une machine probabiliste à deux états est suffisante (ceci est justifié par le fait que les mots de



(a) CSM (b) Pair-HMM

Fig. 4.2 – Illustration d'un CSM et comparaison avec un pair-HMM. Par souci de lisibilité, seulement les probabilités non-nulles sont indiquées.

l'échantillon ne contiennent que deux symboles). Notre but est d'apprendre un CSM avec l'algorithme décrit précédemment. Considérons la connaissance du domaine qui énonce qu'il est impossible d'avoir deux voyelles identiques consécutives dans le même mot dans le langage décrit par LS. Assignons ensuite au premier état la contrainte c_{q_1} *avec les fonctions booléennes suivantes :*

DV : *le symbole courant est le même que le précédent,*
$\overline{DV}$: *le symbole courant est différent du précédent.*

Notons que lorsque $t = 1$*, nous considérons que* $\overline{DV}$ *est à vrai et que* DV *est à faux.*

La Figure 4.2(a) montre le CSM obtenu après convergence de notre adaptation de l'algorithme EM. Pour analyser l'effet de notre contrainte, nous apprenons également un pair-HMM qui est représenté graphiquement dans la figure 4.2(b). Notons que l'ensemble des paramètres des deux modèles a été initialisé aléatoirement. L'analyse de ces modèles nous permet de faire les remarques suivantes :

1. *Dans le pair-HMM, l'analyse des matrices d'édition montre que le premier état représente les opérations effectuées sur le premier caractère de la chaîne, tandis que le second correspond aux opérations d'édition du second caractère.*
2. *Sans la contrainte, le pair-HMM converge vers un optimum local. Le second état indique que la seconde lettre (qui est toujours un a) de chaque mot d'entrée peut être changée indistinctement en un a, b ou c. Ce qui est probablement faux, étant donnée la connaissance a priori.*
3. *D'un autre côté, l'optimum local atteint par le CSM apprend correctement la cible étant donnée la contrainte. Les états ne décrivent plus ici, contrairement au pair-HMM, une position d'une lettre dans la chaîne. Le second état modélise la correction du mot lorsque la contrainte* DV *est satisfaite, alors que le premier ne fait pas de correction puisque* $\overline{DV}$ *est vérifiée.*

Nous avons également évalué la vraisemblance de LS pour estimer la qualité des modèles appris. Comme attendu, notre CSM modélise mieux l'échantillon d'apprentissage puisqu'il lui attribue une vraisemblance de 0.25 alors qu'elle n'est que de 0.037 pour le pair-HMM. ○

4.3 Relation entre CSM et pair-HMM

Dans cette section, nous analysons l'expressivité des CSMs par rapport aux pair-HMMs. De manière assez claire, la différence d'expressivité provient des contraintes utilisées dans le CSM. Pour estimer cette différence, nous définissons tout d'abord un type de contraintes dites régulières.

Définition 4.3 (Contrainte régulière) *Une contrainte régulière est une contrainte telle que le résultat de chacune de ses fonctions booléennes peut être défini comme un problème d'appartenance à un langage régulier. En d'autres termes, chacune de ces fonctions peut être représentée par un automate à états fini. Dans celui-ci, les états terminaux correspondent au fait que que la contrainte soit satisfaite.*

Dans le théorème suivant, nous montrons que les CSMs généralisent les pair-HMMs lorsque les CSMs utilisent des contraintes non régulières.

Théorème 4.4 *Un CSM avec des matrices conditionnelles (respectivement jointes) est une stricte généralisation d'un pair-HMM avec des matrices conditionnelles (respectivement jointes).*

Preuve *Premièrement, nous montrons que tout pair-HMM peut être converti en un CSM équivalent. Soit* $A = \langle \Sigma, \mathcal{Q}, T, \delta, \pi \rangle$ *un pair-HMM, nous construisons un CSM* $C = \langle \Sigma, \mathcal{Q}, \{c\}, T_c, \delta, \pi \rangle$ *avec la même structure. L'unique contrainte* $c = \{c_1\}$ *a exactement une seule fonction booléenne* $c_1 : \Sigma^\star \times \mathbb{N} \rightarrow$ *true et est assignée à chaque état. Pour chaque paire d'états* q_i, q_j, *nous définissons* $T_c(q_j|q_i, c_{q_i,1}) = T(q_j|q_i)$. *Par construction,* A *et* C *ont la même structure et les mêmes paramètres. Nous pouvons donc facilement vérifier qu'ils définissent la même distribution.*

Deuxièmement, considérons un CSM utilisant des contraintes non régulières (i.e. non représentables par un langage régulier). Par exemple $c_k(X, t)$ *est vrai si le préfixe de taille* t *de* X *appartient à un langage non rationnel tel que les langages hors-contextes :* $\{a^n b^n | n \geq 0\}$. *Par définition, cette contrainte ne peut être représentée par une machine à états finie. Il n'est donc pas possible de construire un pair-HMM équivalent à ce CSM.* □

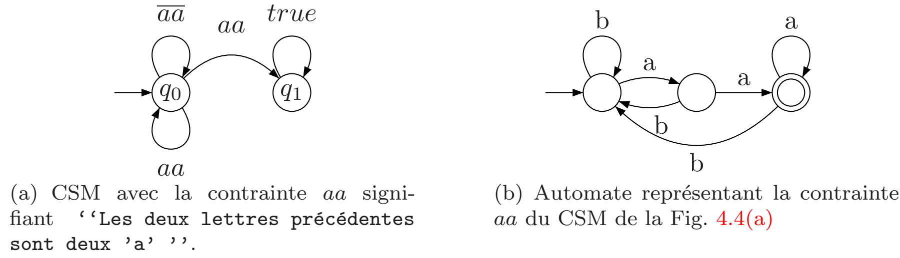
Nous venons de voir que l'utilisation de contraintes non régulières permet un gain d'expressivité du CSM.

Nous nous intéressons maintenant au cas des CSMs possédant uniquement des contraintes régulières. Une question naturelle qui se pose est de savoir si une telle contrainte peut être modélisée de manière équivalente dans un pair-HMM grâce à l'ajout d'états supplémentaires. Nous allons montrer que cette transformation n'est pas toujours possible avec la Proposition 4.5 et nous affirmons que lorsque c'est possible, nous obtenons un pair-HMM avec une structure plus complexe en terme d'états et de transitions. Cette explosion structurelle conduit à un nombre plus important de paramètres à estimer pendant le processus d'apprentissage.

Nous présentons dans l'Algorithme 4.3, le principe général d'une méthode de conversion d'un CSM en un pair-HMM. Expliquons les différentes étapes de cet algorithme à partir d'un exemple de CSM, présenté en Figure 4.4(a), dont les paramètres (choisis arbitrairement) sont présentés dans le Tableau 4.6. Ce CSM modélise la contrainte `‘‘Les deux lettres précédentes sont deux ’a’ ’’`.

Algorithme 4.3 : Conversion d'un CSM en un pair-HMM

Entrée : Un CSM avec contraintes régulières
Sortie : Un pair-HMM
1. Convertir chaque contrainte en un automate et mettre à jour chaque automate pour autoriser les insertions.
2. Déduire la structure du pair-HMM.
3. Déduire les paramètres du pair-HMM à partir du CSM.



(a) CSM avec la contrainte aa signifiant ``Les deux lettres précédentes sont deux 'a' ''.

(b) Automate représentant la contrainte aa du CSM de la Fig. 4.4(a)

Fig. 4.4 – (a) Un CSM avec une contrainte régulière (la boucle sur l'état q_1 est la contrainte "true" qui force à rester dans l'état) ; (b) L'automate correspondant à la contrainte aa.

Première étape. Elle consiste à représenter la contrainte régulière sous la forme d'un automate (voir Fig. 4.4(b), l'état final est représenté par un double cercle). Nous pouvons noter que seulement les chaînes satisfaisant la contrainte précédente terminent leurs parcours dans l'état final. Rappelons que les mots qui nous intéressent ici sont les préfixes de la chaîne d'entre X jusqu'à une position t de X traitée dans l'état q_0 pendant le processus d'édition.

Sachant que la chaîne d'entrée X est impliquée dans le processus d'édition, nous devons prendre en compte la possibilité d'avoir des insertions sur la chaîne Y sans que des lettres de X soient traitées par le processus d'édition. Nous représentons ces insertions par des cycles étiquetés par λ sur chaque état de l'automate (voir Fig. 4.5(a)). Cela ne modifie pas le fait que les mots terminant avec deux 'a' consécutifs terminent leur parcours dans l'état final.

Deuxième étape. Elle consiste à transformer l'automate en un pair-HMM en utilisant le principe décrit dans [DDE05] qui convertit un automate probabiliste en un HMM. Nous donnons rapidement le principe de ce processus. Chaque état q de l'automate déduit à l'étape 1 est divisé en autant d'états que le nombre d'étiquettes différentes présentes sur les transitions entrantes de l'état q. Ces états vont donc concerner seulement un symbole d'entrée.

Par exemple, l'état 1 de la la Figure 4.5(a) est divisé en deux états 1λ et $1b$ dans la Figure 4.5(b) car ses transitions entrantes sont étiquetées par $\{\lambda, b, b, b\}$.

Troisième étape. Comme nous l'avons dit précédemment, le Tableau 4.6 représente les paramètres (choisis arbitrairement) de notre CSM. Le Tableau 4.7 décrit les probabilités associées aux différents états et transitions du pair-HMM de la Figure 4.4(b).

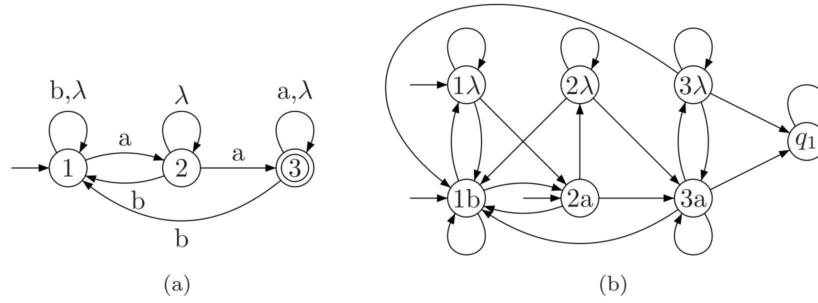


Fig. 4.5 – (a) L'automate modélisant la contrainte régulière aa et autorisant des insertions sur Y ; (b) Le pair-HMM déduit des Fig. 4.4(a) et 4.5(a).

(a)

δ	λ	a	b
λ		.05	.05
a	.05	.3	.1
b	.05	.1	.3

(b)

δ	λ	a	b
λ		.05	.05
a	.05	.35	.05
b	.05	.05	.35

(c)

état initial	valeur
q_0	1.0
q_1	0.0

(d)

état départ	état arrivée	valeur contrainte	valeur
q_0	q_1	aa	$^2/_3$
q_0	q_0	aa	$^1/_3$
q_0	q_0	$\overline{aa}$	1
q_1	q_1	$true$	1

Tab. 4.6 – Représentation des probabilités du CSM de la Figure 4.4(a) : (a) état q_0, (b) état q_1, (c) initiales, (d) transitions.

L'ensemble des probabilités des transitions du pair-HMM déduit est une combinaison des transitions entre les états du CSM et la distribution marginale de chacune des lettres dans les états. Par exemple, la probabilité de transition entre l'état 3λ et $1b$ du pair-HMM est le produit de la probabilité de la transition entre l'état q_0 et q_0 par la contrainte 'aa' (en effet, nous atteignons l'état 3λ lorsque nous avons lu au moins 2 a consécutifs) et la distribution marginale du nombre de b rencontrés dans l'état q_0 : *i.e.* $T(1b|3\lambda) = {}^1/_3 * .45$.

Sur le même principe, nous déduisons toutes les probabilités des transitions du pair-HMM, résumées dans le Tableau 4.7(f). Les probabilités des opérations d'édition dans chacun des états se déduisent simplement en normalisant les valeurs présentes dans l'état q_0 du CSM pour prendre en compte le fait qu'une seule lettre de la chaîne d'entrée peut être traitée par une opération d'édition dans cet état (voir Tab. 4.7(a)(b)(c)). Notons que l'état q_1 du CSM n'est pas contraint, nous retrouvons donc les mêmes probabilités dans l'état q_1 du pair-HMM (voir Tab. 4.7(d)). Les probabilités initiales (voir Tab. 4.7(e)) sont une combinaison de la probabilité initiale présente dans le CSM et de la distribution marginale des symboles dans l'état q_0.

(a)

δ	λ	a	b
λ		.5	.5

(b)

δ	λ	a	b
b	$1/9$	$2/3$	$2/9$

(c)

δ	λ	a	b
a	$1/9$	$2/3$	$2/9$

(d)

δ	λ	a	b
λ		.05	.05
a	.05	.35	.05
b	.05	.05	.35

(e)

état initial	valeur
1λ	.1
$1b$	.45
$2a$	.45

(f)

état départ	état arrivée	valeur
1λ	1λ	.1*1
1λ	$1b$	.45*1
1λ	$2a$	.45*1
$1b$	$1b$	.45*1
$1b$	1λ	.1*1
$1b$	$2a$	.45*1
3λ	q_1	$2/3$

état départ	état arrivée	valeur
2λ	2λ	.1*1
2λ	$1b$	.45*1
2λ	$3a$	.45*1
$2a$	$1b$	.45*1
$2a$	2λ	.1*1
$2a$	$3a$	.45*1
$3a$	q_1	$2/3$

état départ	état arrivée	valeur
3λ	3λ	.1*1/3
3λ	$1b$	.45*1/3
3λ	$3a$	.45*1/3
$3a$	$3a$	.45*1/3
$3a$	$1b$	.45*1/3
$3a$	3λ	.45*1/3
q_1	q_1	1

Tab. 4.7 – Représentation des probabilités du pair-HMM de la Figure 4.5(b) : (a) états 1λ-2λ-3λ, (b) état $1b$, (c) états $2a$-$3a$, (d) état q_1, (e) initiales et (f) transitions.

Nous pouvons noter que l'automate final de la Figure 4.5(b) est un pair-HMM, équivalent à notre CSM. Cependant, nous remarquons qu'il a 7 états, 21 transitions et 3 probabilités initiales non nulles, ce qui nous fait un total de 47 paramètres (y compris les opérations d'édition de chaque état). En comparaison, le CSM de la Figure 4.4(a) a un total de 21 paramètres (opérations d'édition comprises). Les paramètres du pair-HMM devront être estimés pendant le processus d'apprentissage, menant à une augmentation de la complexité et du nombre d'exemples nécessaires. Notons néanmoins ici que quelques paramètres sont liés et qu'il serait alors intéressant de définir une borne du nombre de paramètres utiles. Cependant, le lien entre paramètres n'est pas exploité *a priori* par l'algorithme EM d'apprentissage des pair-HMMs. Une solution serait donc de proposer une adaptation de l'algorithme pour prendre en compte les paramètres liés.

Dans le but de montrer l'influence de l'explosion des paramètres dans la convergence du processus d'apprentissage, nous menons l'expérience suivante. Nous considérons toujours le CSM de la Figure 4.4(a) et le pair-HMM correspondant de la Figure 4.5(b). Nous générons, à partir d'une distribution cible, un ensemble d'échantillons d'apprentissage constitués d'un nombre croissant de paires (de 100 à 8000). Pour chaque échantillon, nous apprenons les paramètres des modèles des Figures 4.4(a) et 4.5(b). Ensuite, nous évaluons la qualité du modèle sur un échantillon test TS de 500 paires de chaînes en comparant la mesure de perplexité $\mathcal{P}$ (*i.e.* une mesure basée sur l'entropie) entre les

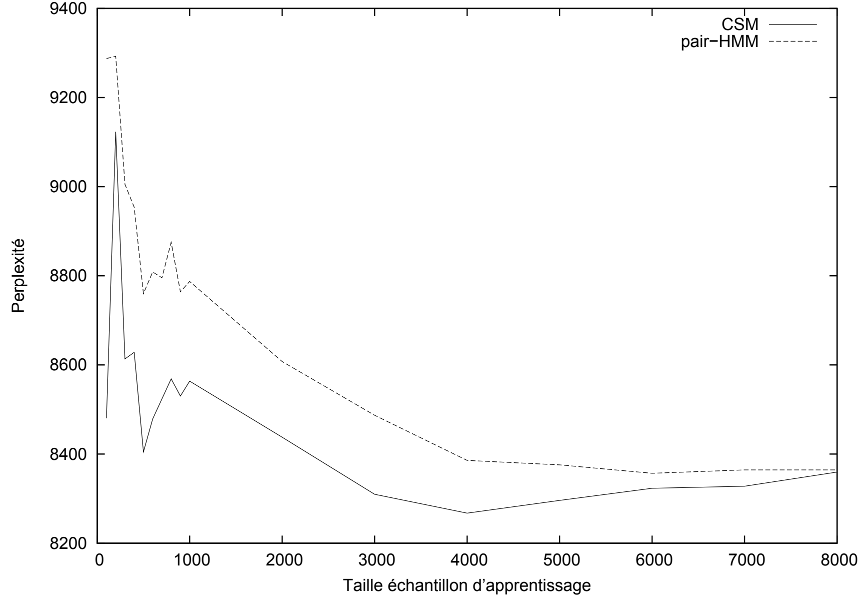


Fig. 4.8 – Comportement du CSM et de son pair-HMM équivalent.

deux modèles, telle que

$$\mathcal{P} = 2^{-\sum_{(X,Y)\in TS} P(Y|X) \log_2 p(Y|X)}.$$

Les résultats sont présentés dans la Figure 4.8. Les deux modèles convergent vers le même point, ce qui signifie qu'ils représentent tous les deux la même distribution statistique, mais le pair-HMM a besoin de plus d'exemples pour produire un modèle équivalent en termes de qualité. Ce petit exemple montre clairement l'intérêt des contraintes pour simplifier les modèles probabilistes à états finis, et donc l'intérêt de nos CSMs.

Montrons désormais avec la proposition suivante qu'un CSM ne peut pas toujours être transformé en un pair-HMM équivalent même avec la présence de contraintes régulières.

Proposition 4.5 *Les CSMs définissent une classe de distribution plus générale que les pair-HMMs.*

Preuve *Par le Théorème 4.4, nous savons que tout pair-HMM est un cas particulier de CSM. Considérons le CSM de la Figure 4.9 avec une simple contrainte régulière sur la lettre précédente. Ce modèle définit deux distributions conditionnelles telles que* $p(aa|a) = p(bb|b) = 1$. *Nous allons montrer que nous ne pouvons pas construire un pair-HMM équivalent avec des matrices conditionnelles.*

Chaque paire de chaînes admet un ensemble de cinq scripts d'édition. Nous avons respectivement pour $(aa|a)$ *et* $(bb|b)$ *:*

$$\begin{aligned} S(aa|a) &= \{[(a|a)(a|\lambda)], [(\lambda|a)(a|\lambda)(a|\lambda)], [(a|\lambda)(\lambda|a)(a|\lambda)], \\ &\quad [(a|\lambda)(a|\lambda)(\lambda|a)], [(a|\lambda)(a|a)]\} \end{aligned}$$

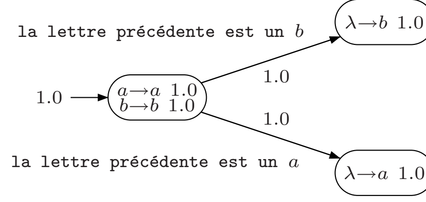


Fig. 4.9 – Exemple d'un CSM définissant deux distributions conditionnelles non représentables par un pair-HMM. Dans chaque état, nous indiquons seulement les opérations d'édition avec une probabilité non-nulle.

et

$$\begin{aligned} S(bb|b) &= \{[(b|b)(b|\lambda)],[(\lambda|b)(b|\lambda)(b|\lambda)],[(b|\lambda)(\lambda|b)(b|\lambda)],\\ &\quad [(b|\lambda)(b|\lambda)(\lambda|b)],[(b|\lambda)(b|b)]\}. \end{aligned}$$

Notons que, selon le CSM, seul le premier script d'édition de chacun des deux ensembles $S(aa|a)$ et $S(bb|b)$ est possible, de probabilité 1. Tous les autres scripts sont de probabilité nulle.

Supposons qu'il existe un pair-HMM $A = \langle \Sigma, \mathcal{Q}, T, \delta, \pi \rangle$ qui combine l'ensemble de ces scripts pour obtenir les distributions désirées. Ceci implique que :

$$\sum_{q \in \mathcal{Q}} \pi(q)\beta_q(aa|a) = \sum_{q \in \mathcal{Q}} \pi(q)\beta_q(bb|b) = 1 \tag{4.4}$$

Maintenant, supposons qu'il existe un état $q_i \in \mathcal{Q}$ tel que $\pi(q_i) > 0$ (notons qu'il doit forcément en exister un) et $\beta_{q_i}(aa|a) < 1$, alors :

$$\begin{aligned} p(aa|a) &= \textstyle\sum_{q \in \mathcal{Q}} \pi(q)\beta_q(aa|a) \\ &= \pi(q_i)\beta_{q_i}(aa|a) + \textstyle\sum_{q \in \mathcal{Q}\setminus\{q_i\}} \pi(q)\beta_q(aa|a) \\ &\leq \pi(q_i)\beta_{q_i}(aa|a) + \textstyle\sum_{q \in \mathcal{Q}\setminus\{q_i\}} \pi(q) \quad (0 \leq \beta_q(aa|a \leq 1) \\ &< \textstyle\sum_{q \in \mathcal{Q}} \pi(q) = 1. \quad (\beta_{q_i}(aa|a) < 1) \end{aligned}$$

Cela implique que $p(aa|a) < 1$ ce qui est en contradiction avec la distribution cible. Donc pour chaque état, avec $\pi(q_i) > 0$, nous avons $\beta_{q_i}(aa|a) = 1$. De plus, à partir de l'Équation 4.4, nous avons :

$$\sum_{q \in \mathcal{Q}} \pi(q)(\beta_q(aa|a) - \beta_q(bb|b)) = 0.$$

Donc, puisque $0 \leq \beta_q(bb|b) \leq 1$, nous avons que pour tout $\pi(q_i) > 0$, $\beta_{q_i}(aa|a) = \beta_{q_i}(bb|b) = 1$.

Ceci implique que les opérations d'édition utilisées dans q_i, correspondant à la première opération d'un script d'édition, doivent sommer à 1 : $\delta_{q_i}(a|\lambda)+\delta_{q_i}(a|a)+\delta_{q_i}(\lambda|a) = 1$ pour $(aa|a)$ et $\delta_{q_i}(b|\lambda) + \delta_{q_i}(b|b) + \delta_{q_i}(\lambda|b) = 1$ pour $(bb|b)$. La condition de distribution

conditionnelle : $\forall a \in \Sigma, \sum_{b \in \Sigma \cup \{\lambda\}} \delta_{q_i}(b|a) + \sum_{b \in \Sigma} \delta_{q_i}(b|\lambda) = 1$ *implique qu'aucune insertion n'est possible dans* q_i*.*

Par conséquent les scripts de probabilité non nulle possibles sont ceux commençant par $(\lambda|a)$ *ou* $(a|a)$ *pour* $(aa|a)$ *et* $(\lambda|b)$ *ou* $(b|b)$ *pour* $(b|b)$*. Dans tous les cas la prochaine opération à faire est une insertion :* $(a|\lambda)$ *pour* $(aa|a)$ *et* $(b|\lambda)$ *pour* $(bb|b)$*.*

Ces deux opérations d'insertion doivent avoir une probabilité de 1 *pour maintenir* β_{q_i} *égal à* 1*. Elles ne peuvent pas être dans le même état, sinon, comme mentionné précédemment, la condition de distribution conditionnelle serait transgressée. Donc, elles doivent être dans deux états différents, ce qui implique d'ajouter deux transitions à partir de* q_i*. Chacune de ces deux transitions doit être de probabilité* 1 *pour assurer que* $p(aa|a) = p(bb|b) = 1$*, ce qui est impossible étant donnée la condition sur les transitions d'un pair-HMM (*$\forall q_i \in Q, \sum_{q_j \in Q} T(q_j|q_i) = 1$*).* □

4.4 Application : Recherche de TFBS dans des régions promotrices

4.4.1 Contexte scientifique

Dans cette section, nous proposons de montrer la pertinence de notre modèle sur une application en biologie moléculaire. Nous allons considérer une tâche dont le but final est de trouver des sites de facteurs de transcription (TFBS) dans des séquences promotrices de gènes orthologues. Les travaux présentés dans cette partie ont été effectués en collaboration avec Olivier Gandrillon et Mathilde Pellerin, chercheurs au centre CGMC[1], dans le cadre du projet ANR BINGO2[2].

Un gène est une séquence d'acides désoxyribonucléiques (ADN) destinée à être transcrite en acide ribonucléique (ARN). La plupart du temps, un gène commence par une séquence de nucléotides appelée promoteur, dont le rôle est de permettre l'initiation mais surtout la régulation (tous les gènes ne sont pas exprimés dans toutes les cellules) de la transcription de l'ADN en ARN. Ce rôle est assuré par certaines zones du promoteur appelées TFBS (voir Fig 4.10). Deux gènes sont orthologues lorsqu'ils partagent la même fonction et ont un ancêtre commun.

Notre objectif est de montrer que l'utilisation de notre CSM peut être utile pour modéliser des zones de TFBS tout en permettant : (i) l'intégration de connaissance du domaine, (ii) de prendre en compte de l'information contextuelle, et (iii) de proposer un modèle stochastique *a posteriori* interprétable par des experts du domaine. Il a été démontré [DD07] que les TFBS sont sensibles à l'évolution des espèces, c'est-à-dire que les sites de transcription ont évolué moins rapidement que le reste de la séquence promotrice. Cette différence d'évolution peut être observée en comparant des séquences orthologues de gènes d'espèces suffisamment éloignées. En termes de DE, cela implique que les zones de TFBS seront plus proches entre elles que le reste de la séquence (*i.e.* les opérations de substitution $(b|b)$, pour tout $b \in \{A, C, G, T\}$, auront une probabilité plus forte que les autres opérations d'édition dans les zones de TFBS).

[1] Centre de Génétique Moléculaire et Cellulaire, Université de Lyon.
[2] https://bingo2.greyc.fr/.

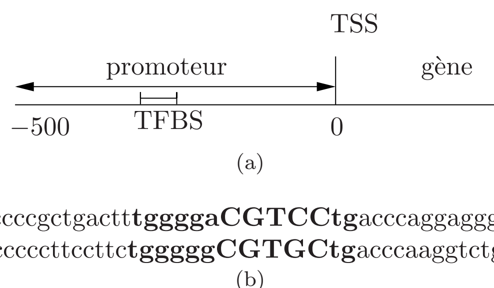


Fig. 4.10 – (a) Représentation graphique d'un promoteur ; (b) Extrait d'une paire de séquences promotrices (la partie en gras de la séquence représente un TFBS, les lettres en majuscule représentent le *core* du TFBS).

4.4.2 Données

Dans les expérimentations suivantes, nous considérons comme région promotrice (ou promoteur) les 500 paires de bases (bp) localisées 5 bases avant le départ du site de transcription (TSS) [KD08] (voir Fig. 4.10(a)). Nous nous restreignons à une liste de gènes orthologues de l'humain et de la souris extraite par l'outil de recherche `BioMart`[3] de la base de données `Ensembl`. Les séquences promotrices de ces gènes sont obtenues ensuite par des requêtes à la base de données `SQUAT` [LSB+08] en utilisant des informations additionnelles à travers `DBTSS`[4]. Tout ce processus a conduit à une liste de 13520 paires de 500 bp de séquences promotrices orthologues.

4.4.3 MATCH™

Une méthode de recherche de TFBS dans des séquences d'ADN a été proposée par Kel *et al.* et baptisée MATCH™ [KGR+03]. MATCH™ est un outil de recherche de facteurs de transcription sur des chaînes d'ADN basé sur des matrices poids-position. Les matrices poids-position sont disponibles dans la base de données TRANSFAC® [WCF+01]. La Table 4.11 représente une matrice poids-position pour un TFBS donné. Pour chaque position du TFBS, elle donne la fréquence de chacun des nucléotides. L'ensemble de ces fréquences est issu d'expérimentations *in vitro* et *in silico*. Il existe une matrice poids-position par TFBS.

De plus, MATCH™ exploite le fait qu'un TFBS est composé d'une partie plus importante qui est appelée le *core*. Ce *core* a une taille de **cinq nucléotides** et correspond à la partie du site de transcription qui est la plus stable d'un site à l'autre.

Pour une sous-séquence d'ADN de taille donnée, MATCH™ calcule des scores pour le *core* du TFBS et pour l'ensemble de la matrice, et les compare à des seuils pour savoir si la sous-séquence étudiée est un TFBS ou non. Il est important de noter que la présence d'un zéro dans la matrice poids-position, (par exemple, position 7, nucléotide A) ne veut pas dire que MATCH™ ne retrouvera jamais cette base à cette position.

[3] `http://www.ensembl.org/biomart/martviez/.`

[4] `http://dbtss.hgc.jp`, mise à jour hg18 et mm8 pour les séquences humaines et souris respectivement.

position	A	C	G	T
01	3	6	3	5
02	6	1	9	1
03	3	2	8	4
04	5	2	10	0
05	3	4	8	2
06	5	1	10	1
07	**0**	**14**	**0**	**3**
08	**2**	**0**	**12**	**3**
09	**1**	**11**	**1**	**4**
10	**2**	**2**	**11**	**2**
11	**0**	**14**	**1**	**2**
12	6	0	1	10
13	5	2	3	7

Tab. 4.11 – Exemple de matrice poids-position pour le TFBS nommé ZF5_B. Les positions du *core* sont en caractères gras.

4.4.4 Protocole expérimental et initialisation du CSM

Avant de décrire les différentes expériences menées pour la reconnaissance de sites de facteurs de transcription, présentons tout d'abord la structure du modèle utilisé.

Les états. Ayant l'ensemble des connaissances précédentes, nous avons décidé de construire un CSM à huit états. Ces états se décomposent en deux sous-ensembles : (i) un état (DI) pour la partie non TFBS de la séquence d'ADN, (ii) les sept autres états pour la partie codant le TFBS. Ce deuxième groupe est composé d'un état pour le *pré-core* (*pre*), un pour le *post-core* (*post*) et les cinq autres (c_i tel que $1 \leq i \leq 5$) pour les différentes positions du *core*. La Figure 4.12 représente graphiquement le CSM utilisé pour les expérimentations.

Les contraintes. Aux états *pré-core* et *post-core*, nous assignons une contrainte sur le temps d'attente. Plus précisément, nous utilisons une contrainte qui a 15 modalités c_k ($k = 0..14$), avec c_k qui signifie : `‘‘est-ce que je suis dans l'état courant depuis (k+1) fois ?’’` (excepté $k = 14$ signifiant : `‘‘est-ce que je suis dans l'état courant depuis au moins 15 fois ?’’`. Cette contrainte nous permet de modéliser la longueur du *pré-core* et du *post-core*. En effet, après quelques analyses des matrices poids-position, nous avons observé que les tailles des zones *pré-core* et *post-core* ne dépassaient jamais 15 nucléotides. Entre les différents états c_i représentant les différentes positions du *core*, nous avons une contrainte `‘‘true’’` avec une seule modalité qui est toujours vérifiée quelle que soit la chaîne d'entrée et la position dans cette chaîne. La contrainte assignée à l'état DI, la partie ne modélisant pas un TFBS, est la contrainte `‘‘true’’`. En effet, n'ayant pas observé de régularité dans la position d'un TFBS par rapport à un autre, nous avons décidé que l'utilisation de contrainte forte sur cet état n'était pas utile. La dernière remarque que nous pouvons faire sur la construction de cette machine est le fait que lorsqu'un TFBS est trouvé, il passe tout d'abord par l'état *pré-core*, ensuite les différents états c_i par ordre croissant et enfin l'état *post-core* avant

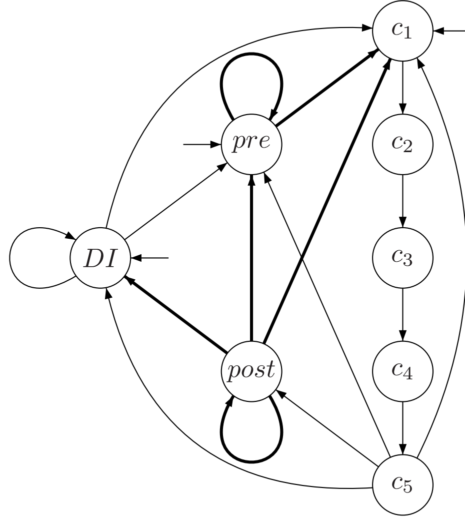


Fig. 4.12 – Représentation graphique du CSM utilisé dans le cadre des expérimentations sur les données biologiques. L'état DI est utilisé pour modéliser la partie non-codante de la séquence d'ADN. Les états *pre* et *post* représentent respectivement les états *pré-core* et *post-core* du TFBS. Enfin, les états c_i identifient les différentes positions du *core*. Les transitions en gras permettent de prendre en compte le temps d'attente, tandis que les autres ne portent pas de contrainte.

de retourner dans l'état DI. De plus, le nombre de TFBS sur une séquence promotrice étant faible (de 3 à 5 pour 500 bp), la transition entre l'état DI et *pre* aura une faible probabilité. Nous avons donc décidé de n'autoriser qu'une sous-partie de l'ensemble des opérations d'édition dans l'état DI : les insertions et les délétions. Dans le cas où des opérations de substitution seront préférables, nous considérerons que nous sommes en train de reconnaître une paire de TFBS.

Initialisation du modèle. L'objectif de l'expérimentation étant de modéliser des TFBS sur des paires de séquences promotrices, nous effectuons l'initialisation suivante. Le nombre de TFBS étant peu élevé sur un promoteur, si nous utilisons une initialisation aléatoire, le processus EM, qui maximise la vraisemblance d'un échantillon, ne tendra probablement pas à optimiser les paramètres pour capturer l'information des TFBS du fait des problèmes de maxima locaux, mais plutôt à trouver d'autres motifs dans ces séquences. Nous avons donc décidé de guider la première itération de l'algorithme EM en utilisant des données étiquetées par MATCH$^{\text{TM}}$. L'étiquetage permet de connaître les positions probables des TFBS sur les promoteurs. Nous obtenons donc un ensemble d'initialisation qui permet de guider le premier tour de l'algorithme EM. Au cours de ce processus, l'étiquetage des données permet d'accumuler les opérations d'édition s'effectuant entre les TFBS des promoteurs dans les états correspondants. No-

tons que MATCH^{TM} utilise en entrée une unique séquence. Sachant que nous travaillons avec des paires, un pré-traitement des données est nécessaire. Pour ce faire, nous proposons de conserver les paires de TFBS pour lesquelles l'écart entre la position de départ d'un TFBS situé sur la chaîne d'entrée et la position de départ d'un TFBS sur la chaîne de sortie est inférieur à une distance maximale fixée *a priori*. Nous justifions ce procédé par le fait que la DE opère naturellement des opérations de substitution sur des caractères situés à une distance raisonnablement faible.

Pour apprendre le modèle, nous utilisons un ensemble de paires de promoteurs pour lesquelles MATCH^{TM} a trouvé au moins un TFBS sur chaque élément de la paire. Au contraire de l'étape d'initialisation, les promoteurs **ne sont plus marqués**, afin de permettre une généralisation. Le but étant de retrouver les TFBS des promoteurs, et en découvrir potentiellement d'autres, nous ne gardons que le chemin de Viterbi (le chemin avec la probabilité maximale). En effet, l'utilisation du script de Viterbi permet l'extraction de la séquence des opérations d'édition. Cette séquence peut alors être annotée par l'état dans lequel l'opération courante est faite et donc nous pouvons marquer sur la paire de séquences promotrices les TFBS.

4.4.5 Résultats

Dans un premier temps, nous avons décidé de mener des expérimentations sur l'apprentissage de CSM spécialisés pour la reconnaissance d'un TFBS donné. Nous avons sélectionné les TFBS suivants : ZF5_B, PAX4_03 et PPARG_02 qui ont conduit à des ensembles d'apprentissage de tailles respectives : 157, 148, 48 paires avec une moyenne de 1.08 TFBS par paire de promoteurs. L'un des premiers objectifs est de comparer les modèles appris et les matrices poids-positions.

Adéquation des opérations d'édition apprises avec les matrices poids-positions

L'objectif de cette première expérimentation est de vérifier que notre algorithme d'apprentissage basé sur EM permet d'attribuer des probabilités d'édition corrélées avec les fréquences observées dans les matrices poids-positions issues de TRANSFAC®. Le but est de vérifier que notre algorithme ne diverge pas.

Pour illustrer cette corrélation, la Figure 4.13 donne les probabilités des opérations d'édition obtenues après cinq itérations de l'algorithme EM sur les données du TFBS ZF5_B pour les différentes positions du *core*. Le lecteur intéressé peut regarder l'ensemble du modèle, et ceux pour les autres TFBS traités, en annexe de ce document.

Pour la position 1 du core, le Tableau 4.11 indiquait que 80% des bases rencontrées sont des C (14/17) et 20% des T (3/17). Rappelons qu'un zéro dans les données de TRANSFAC® ne signifie pas que MATCH^{TM} ne pourra pas retrouver de site de facteur de transcription commençant par une autre lettre que C ou T. Le modèle appris (Fig. 4.13) montre que 71% des bases concernées par les opérations d'édition sont bien des C. De plus, il est intéressant de noter qu'après optimisation, notre modèle a fait disparaître les T et autorisé 29% d'opérations sur des G, qui semblent donc être plus utiles pour maximiser la vraisemblance de l'échantillon d'apprentissage.

c_1			
	$C \rightarrow C$	42.40%	
	$C \rightarrow G$	37.73%	
	$G \rightarrow C$	19.86%	
c_2			
	$G \rightarrow G$	24.51%	
	$T \rightarrow T$	23.72%	
	$G \rightarrow A$	20.88%	
	$C \rightarrow G$	14.33%	
	$G \rightarrow T$	9.03%	
	$T \rightarrow G$	7.52%	
c_3			
	$C \rightarrow C$	19.43%	
	$G \rightarrow G$	18.76%	
	$C \rightarrow G$	11.08%	
c_4			
	$C \rightarrow C$	16.77%	
	$G \rightarrow G$	17.92%	
	$G \rightarrow C$	8.74%	
c_5			
	$C \rightarrow C$	62.01%	
	$G \rightarrow C$	26.20%	
	$A \rightarrow C$	11.77%	

Fig. 4.13 – Modèle appris après cinq itérations de l'algorithme d'apprentissage pour le TFBS ZF5_B. Nous ne mentionnons ici que les opérations sur les différentes positions du *core* (seules les opérations significatives sont indiquées).

Ainsi, par notre processus d'optimisation, nous pouvons voir d'une part que la tendance observée par MATCH^{TM} est bien conservée (*i.e.* la grande majorité des opérations portent bien sur un C) et d'autre part que certaines statistiques descriptives observées via MATCH^{TM} sont remises en cause par notre modèle (*i.e.* suppression de certaines lettres et apparition de nouvelles). La question est désormais de savoir si cela permet de non seulement retrouver des TFBS connus (et reconnus par MATCH^{TM}) et de découvrir de nouveaux TFBS par généralisation. C'est le but des deux paragraphes suivants traitant de deux situations différentes : (i) celle où les promoteurs sont représentés sous forme de paires, *i.e.* ils sont promoteurs de deux gènes orthologues, (ii) celle où ils sont représentés sous la forme de singleton.

Détections de TFBS dans une paire de promoteurs

Si les promoteurs sont par paire, la stratégie de recherche est la suivante. Étant donné un CSM appris selon la procédure présentée précédemment, nous cherchons le script de Viterbi entre les deux promoteurs. Pendant la construction du script, nous annotons les opérations d'édition par l'état dans lequel elles apparaissent pour marquer les zones de TFBS. Si une annotation passe par l'état *pre* de notre CSM, un TFBS est donc découvert. Afin de ne retenir du CSM que les TFBS les plus pertinents, nous avons adopté une approche de sélection, visant à ne conserver, à chaque itération de EM, que les TFBS ayant une grande probabilité. Plus précisement, nous avons choisi comme seuil de sélection $\mathcal{M}$, la moyenne géométrique des probabilités des opérations d'édition :

$$\mathcal{M} = \sqrt[|E|]{\prod_{e \in E} \delta(e)},$$

avec E l'ensemble des opérations d'édition entrant dans la construction de la paire de TFBS. Ce seuil est évalué à chaque itération de l'algorithme EM à l'aide d'un échantillon

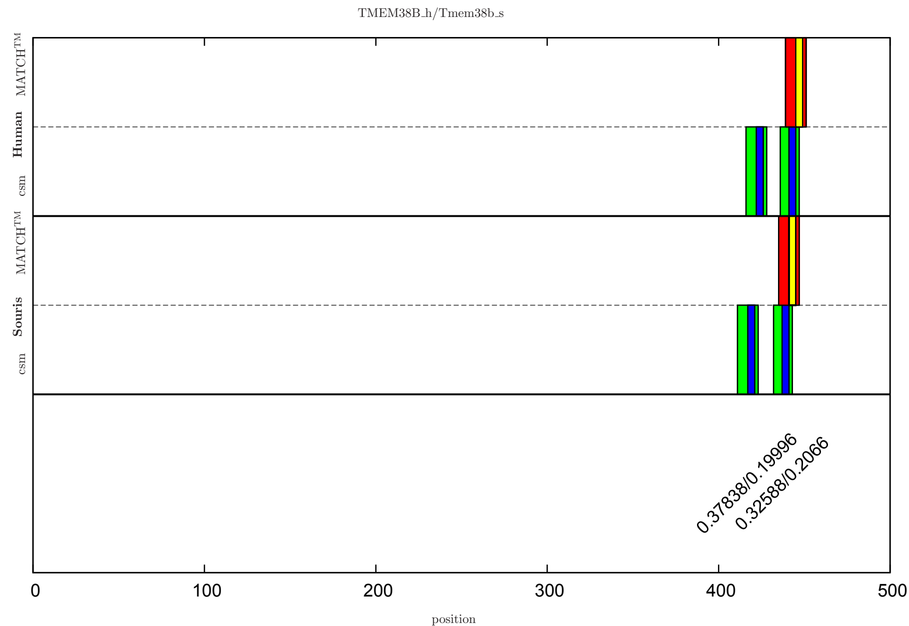


Fig. 4.14 – Résultat sur une paire de promoteurs de gènes orthologues.

d'évaluation contenant des paires de TFBS, construit à partir des données de MATCH$^{\mathrm{TM}}$. Une telle stratégie a ainsi permis de rapidement supprimer les candidats TFBS de faible probabilité.

La Figure 4.14 montre un résultat d'expérimentation pour la paire de promoteurs du gène $TMEM38B$, évidemment non présente dans l'échantillon d'apprentissage. Le graphique est partagé en trois parties : la partie haute est le promoteur humain, celle du milieu le promoteur souris et la dernière retourne des informations de similarités (les moyennes géométriques). Chaque promoteur est découpé en deux parties : en haut les TFBS trouvés par MATCH$^{\mathrm{TM}}$ (jaune/rouge), en bas les TFBS trouvés par le CSM (vert/bleu). Les parties bleu et rouge représentent les *cores* des TFBS.

Pour le gène $TMEM38B$, le CSM retrouve deux paires de TFBS :

humain		**souris**		**moyenne**
position	**motif**	**position**	**motif**	**géométrique**
416	cggcccCGCCCcc	411	ggacccCGCCCcc	0.378/0.199
436	cagggCGCACgc	432	cagggCGCACgc	0.325/0.206

Sur le même promoteur, MATCH$^{\mathrm{TM}}$ retrouve le TFBS suivant :

humain		**souris**	
position	**motif**	**position**	**motif**
439	ggcgcaCGCGCgg	435	ggcgcaCGCGCag

Nous pouvons constater d'une part que le TFBS issu de MATCH^{TM}, donc reconnu comme pertinent par la communauté, est également retrouvé par notre CSM (à un décalage de 3 bases près) ; d'autre part, notre algorithme d'apprentissage, par généralisation, nous a permis de détecter un nouveau TFBS (commençant en position respectives 416 et 411) que MATCH^{TM} n'avait pas identifié. Il est important de noter ici que l'évaluation de la pertinence d'un tel résultat est difficile. En effet, à chaque nouvelle proposition de TFBS, les collègues biologistes doivent effectuer une série d'expérimentations en laboratoire pour valider l'hypothèse. Ce travail n'a malheureusement pas encore pu être réalisé par les chercheurs du laboratoire CGMC car les premiers résultats issus du CSM n'ont été disponibles que récemment.

Notons néanmoins que la moyenne géométrique des probabilités des opérations d'édition de ce nouveau TFBS trouvé par le CSM (et pas par MATCH^{TM}) est de 0.378. Nous pouvons observer que cette valeur est plus importante que pour le premier TFBS qui était reconnu par MATCH^{TM} (0.325), ce qui nous laisse à penser que cette nouvelle connaissance est probablement pertinente.

Détection de TFBS sur un unique promoteur

Lorsque nous ne disposons pas de paires de promoteurs, l'exploitation directe de notre CSM, qui modélise en pratique une transduction d'une chaîne en une autre, n'est plus possible. Afin de détecter des TFBS à partir d'un promoteur unique, nous avons adopté une technique de marche aléatoire. Plus précisemment, pour chaque promoteur unique, nous effectuons 4000 marches aléatoires dans le CSM (pour chaque lettre de la chaîne d'entrée, nous choississons aléatoirement, suivant les paramètres du modèle, l'opération d'édition que nous allons appliquer). Notons que, comme lors de la recherche de TFBS dans des paires, nous utilisons la valeur de la moyenne géométrique pour savoir si nous conservons ou non la partie du script d'édition passant par les états marquant un TFBS. Chacune de ces marches génère alors un script d'édition que nous utilisons pour estimer pour chaque position du promoteur l'appartenance ou non à une position du *core* d'un TFBS.

À titre d'illustration, la Figure 4.15 présente les résultats obtenus pour le promoteur Zfp667, issu de l'échantillon de test. Le graphique met en relation une position et le nombre d'occurences pour lesquelles une marche aléatoire a trouvé un *core* à cette position. Les résultats de MATCH^{TM} sont représentés par des barres d'histogramme, la zone jaune correspondant au *core* d'un TFBS et les parties vertes au reste du TFBS. Cette expérience permet de proposer une région probable de TFBS pour un promoteur donné. Nous pouvons constater à nouveau que le CSM est capable de retrouver les TFBS de MATCH^{TM} (ici 4) et d'en détecter 6 nouveaux, qui devront être validés par les biologistes dans un second temps.

Comparaison des *cores* de TFBS

Nous terminerons ce chapitre d'expérimentations en comparant sur le TFBS ZF5_B (le lecteur intéressé trouvera en annexes les résultats des autres TFBS) les cores retrouvés par le CSM et ceux issus de MATCH^{TM}. Le Tableau 4.16 montre que parmis les cinq cores les plus présents dans les TFBS retrouvés, 3 sont en commun entre les deux méthodes, les deux autres n'étant séparés que d'une ou deux opérations d'édition.

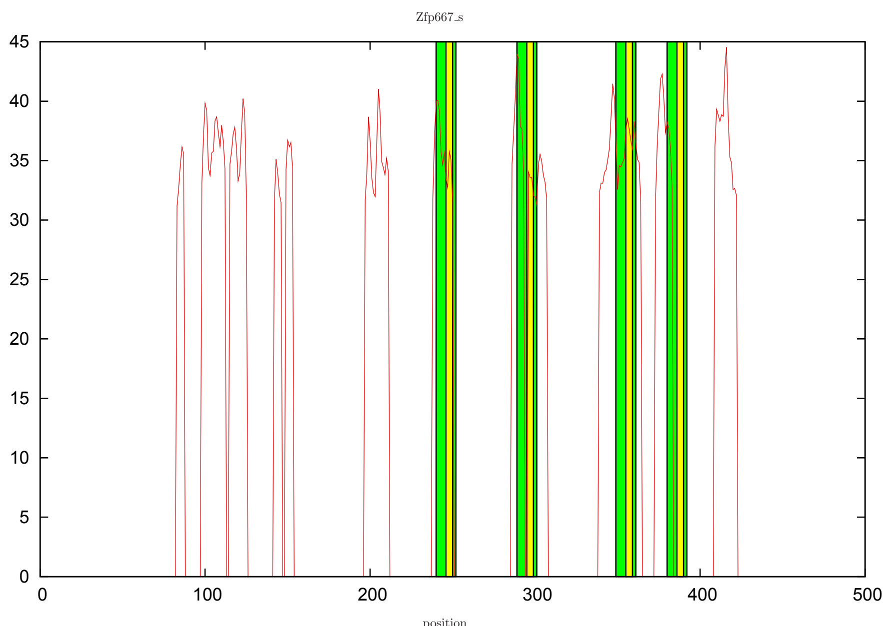


Fig. 4.15 – Résultat de l'application du CSM sur un promoteur unique.

MATCH^{TM}		CSM	
motif	**proportion**	**motif**	**proportion**
CGCGC	49.56%	CGGCC	22.22%
CGCCC	10.79%	CGCGC	16.66%
CGGGC	8.61%	CGCCC	11.11%
CGCTC	8.45%	CGGGC	11.11%
CGCAC	7.17%	CGTGC	11.11%

Tab. 4.16 – Présentation des *cores* représentant le TFBS ZF5_B trouvés par MATCH^{TM} et le CSM.

4.5 Conclusion et discussion

Dans ce chapitre, nous avons présenté un modèle d'édition *non-memoryless* contraint entre chaînes. Nous avons abordé le problème sur des chaînes pour des raisons algorithmiques évidentes mais une extension aux données arborescentes semble manifeste. Nous parlons de modèle *non-memoryless* car l'utilisation de plusieurs états permet de modéliser différents contextes d'édition. L'utilisation de contraintes, au niveau des transitions du modèle, sous forme d'ensemble de fonctions booléennes permet d'ajouter de la connaissance *a priori* dans la machine. Nous avons présenté une méthode d'apprentissage, basée sur EM, qui maximise la vraisemblance d'un échantillon d'apprentissage composé de paires jugées similaires. Dans le cadre de ce chapitre, nous avons expérimenté nos modèles sur une tâche de biologie moléculaire.

Les modèles que nous avons introduits dans ce chapitre sont une généralisation des pair-HMMs [DEKM98]. Nous avons montré que dans le cas d'utilisation de contraintes régulières, il était possible de construire un pair-HMM équivalent mais le nombre de paramètres en est alors augmenté. Dans les CSMs, nous pouvons utiliser des contraintes non-régulières qui permettent donc un pouvoir d'expression plus important. Nous avons aussi montré que les CSMs définissent une classe de distribution plus générale que les pair-HMMs.

Une contrainte se présente comme un ensemble de fonctions booléennes définies par l'utilisateur. Suivant les connaissances liées au problème considéré, la définition de ces contraintes peut être une tâche compliquée. C'est pourquoi, il serait utile de pouvoir extraire les contraintes de l'échantillon d'apprentissage par des outils de recherche d'information ou de fouille de données. Remarquons aussi que les propriétés liées à la définition d'une contrainte sont fortes. Il semblerait intéressant de regarder si nous ne pourrions pas modifier la définition du CSM et l'algorithme d'apprentissage de telle sorte à pouvoir être plus libres sur la définition de nos contraintes. En effet, dans sa description courante, une seule fonction de la contrainte peut être vérifiée pour une position donnée de la chaîne à l'instant t. Nos premières investigations laissent à penser qu'il serait possible de se passer de cette limite, au prix d'une complexification de l'algorithme d'apprentissage.

Enfin, les expérimentations de ce chapitre nous ont mené à une collaboration avec des chercheurs du CGMC qui sont spécialistes de la biologie moléculaire. Nous avons travaillé sur des données issues de gènes orthologues entre humains et souris, l'objectif étant de retrouver des sites connus de facteurs de transcription et d'en découvrir de nouveaux. Deux gènes orthologues ayant de grandes ressemblances au niveau des TFBS, nos travaux sur la distance d'édition et plus particulièrement l'utilisation des CSM dans ce contexte a donc été possible. Mais le domaine de la biologie moléculaire étant complexe, et nécessitant de nombreuses interactions entre informaticiens et biologistes, les expérimentations n'ont pu progresser que par petites étapes successives. C'est pourquoi, aujourd'hui, il nous est difficile de tirer des conclusions définitives. Notons néanmoins que les résultats intermédiaires sont encourageants et l'étude de ce problème mérite donc, à notre avis, d'être prolongée.

Conclusion générale

Dans le cadre de cette thèse, nous nous sommes intéressés à l'apprentissage de similarités d'édition probabilistes entre données structurées sous la forme d'arbres et de chaînes. Nous avons fait le choix délibéré de proposer une conclusion à l'issue de chacun des chapitres du mémoire. Cette démarche nous a permis d'aborder des points de réflexion parfois techniques associés aux approches présentées dans chaque chapitre. Dans cette conclusion, nous nous proposons plutôt de faire un bilan général des travaux effectués durant ces trois années de thèse.

Le point commun entre toutes les méthodes que nous avons proposées réside dans l'apprentissage de coûts d'opérations d'édition dans un cadre probabiliste. Ce processus d'apprentissage est effectué par le biais d'adaptations de l'algorithme *Expectation-Maximisation* utilisé pour maximiser la vraisemblance d'un échantillon de paires d'exemples jugés similaires. Lors de nos travaux, nous avons essayé de fournir plusieurs méthodes d'inférence de similarités d'édition adaptées à différentes problématiques. Sans avoir la prétention d'être exhaustif, les contributions présentées dans cette thèse ont, à notre avis, permis d'augmenter les domaines d'applications des similarités basées sur la distance d'édition en les adaptant soit à des données spécifiques comme les arbres, soit à des applications nécessitant la prise en compte d'informations *a priori*. S'il reste encore de nombreuses problématiques à résoudre, nous pensons que nos travaux ont également eu l'intérêt d'ouvrir plusieurs perspectives de recherche.

Concernant le cas de données arborescentes, nous avons étudié deux algorithmes, chacun d'eux étant adapté à une définition de distance d'édition particulière entre deux arbres. La première - la distance de Selkow - considère notamment des opérations d'insertion et de délétion de sous-arbres entiers, alors que la seconde - la distance de Zhang et Shasha - autorise des opérations d'insertion et de suppression au niveau des noeuds. Suivant le type de problème traité, chacune de ces distances a son intérêt propre. Notons de plus que si la distance de Zhang et Shasha est plus générale, elle est également nettement plus coûteuse en terme de complexité ce qui peut représenter un inconvénient majeur pour certaines applications. Pour chacune de ces deux distances, nous avons proposé un algorithme d'inférence basé sur un modèle à un seul état dit *memoryless*. Ce modèle a été décliné selon deux modalités : l'une basée sur un modèle probabiliste joint et l'autre selon un modèle conditionnel. Nous avons vu que chacun de ces deux modèles avait ses propres défauts : le biais causé par la distribution des chaînes d'entrée pour le premier et une plus grande variance pour le second. Si toutes les méthodes proposées possèdent leur propre champ d'application, nous avons montré qu'elles permettaient toujours d'améliorer les

approches standard sans apprentissage. Nous avons, de plus, développé une plateforme logicielle SEDiL, disponible pour la communauté sur Internet, qui intègre l'ensemble de ces méthodes.

Nous avons également travaillé sur un modèle de machines à états contraintes permettant d'intégrer de la connaissance du domaine. Ce type de modèle, que nous avons appelé CSM pour *Constrained State Machines*, permet d'intégrer de la connaissance *a priori* sous la forme de contraintes booléennes associées aux transitions. De plus, son caractère *non-memoryless*, nous permet de considérer naturellement plusieurs contextes d'édition. Nous avons notamment prouvé que les CSMs étaient en fait une généralisation des pair-HMMs. Notre modèle a été en outre exploité sur une tâche de biologie moléculaire visant à trouver des sites de facteurs de transcription sur des promoteurs de gènes. S'il reste encore beaucoup de travaux à mener, les premiers résultats obtenus nous permettent de penser que nos modèles peuvent offrir des perspectives prometteuses pour ce type de tâches, qui sont d'ailleurs connues comme étant difficiles. Si cette partie a été considérée sous l'angle de distance entre chaînes, elle possède toutefois des liens naturels avec le chapitre précédent sur les distances entre arbres. L'adaptation de nos modèles à états contraints aux arbres pourrait notamment fournir des perspectives intéressantes sur des problèmes de traitement de données XML ou de biologie moléculaire impliquant des données plus complexes.

Quelles sont les perspectives qui se dégagent à l'issue de nos travaux ? Elles sont bien évidemment nombreuses et nous en avons déjà évoqué quelques unes dans les conclusions des chapitres précédents. Toutefois, une problématique générale à toutes les approches présentées dans ce manuscrit concerne la constitution d'un échantillon d'apprentissage de paires d'exemples similaires. La définition de ces paires est bien évidemment primordiale pour obtenir de bons modèles de similarité d'édition. Si pour certaines applications leur construction semble évidente, comme par exemple un mot erroné et sa correction ou encore deux promoteurs de gènes orthologues, dans d'autres cas l'appariement de deux exemples est sujet à discussion. Ce problème est entre autre illustré dans l'application de reconnaissance de chiffres manuscrits que nous avons évoquée dans le Chapitre 3. En effet, nous pouvons nous demander quelle est la définition d'une bonne paire d'apprentissage. Est-ce deux exemples de la même classe pris aléatoirement ? Ou plutôt un exemple avec son plus proche voisin selon la distance de Levenshtein ? Nous n'avons à l'heure actuelle pas de véritable solution à proposer pour ce problème. Une piste pourrait consister à essayer de trouver une notion de **paires pertinentes** permettant de discriminer une partie de l'échantillon, par exemple à l'aide de techniques de clustering.

Une autre manière de contourner ce problème serait de définir des méthodes d'apprentissage nécessitant un échantillon composé de singletons uniquement. L'objectif ne serait alors plus de maximiser la vraisemblance de l'échantillon de paires d'apprentissage mais, par exemple, de maximiser le taux de bonne classification. Pour cette perspective, les approches récentes de Balcan *et al.* sur l'apprentissage à partir de bonnes fonctions de similarités généralisant la notion de noyau des SVM [BBS08b, BBS08a] pourraient fournir des pistes intéressantes.

Pour terminer, nous avons principalement considéré des similarités d'édition et non des distances tout au long du mémoire. Autrement dit, les propriétés de séparation, de symétrie, de non-négativité, et d'inégalité triangulaire, définies en Section 1.3.1, ne sont pas forcément respectées dans nos modèles. La conséquence de cet aspect est qu'une majorité des algorithmes utilisant les propriétés de la notion de distance, pour optimiser les calculs et gagner en complexité algorithmique, ne sont plus utilisables avec nos similarités. Une perspective de nos travaux est donc d'adapter les matrices de probabilités des opérations d'édition avec la notion de similarité d'édition pour construire une distance. Nous pourrions, par exemple, nous inspirer de l'approche introduite récemment dans les travaux de Bellet et al. [BBMS10], où les auteurs proposent de construire un noyau - fonction symétrique et non négative - de chaînes à partir des distributions d'édition apprises.

Annexes

Dans cette partie, nous présentons un échantillon plus complet des résultats des différentes expérimentations menées dans le cadre de l'ANR Bingo2. Nous rappelons que les tests ont été effectués sur trois TFBS différents : ZF5_B, PAX4_03 et PPARG_02. Tout d'abord, précisons les tailles des échantillons d'apprentissage et de test.

TFBS	$\|LS\|$	$\|TS\|$
ZF5_B	182	32
PAX4_03	148	30
PPARG_02	48	10

Les résultats sont présentés en quatre parties. Premièrement, nous montrons pour chaque TFBS, le modèle appris et les données de TRANSFAC®. Deuxièmement, nous comparons les motifs des cores de TFBS retrouvés. Troisièmement, nous présentons différents résultats de recherche de TFBS sur des paires de promoteurs. Enfin, nous montrons des résultats de détection de TFBS sur des promoteurs uniques.

A Expérimentations sur le TFBS ZF5_B

(a) pré-core

	λ	A	C	T	G
λ	0.0	1.12E-4	0.0530	2.02E-4	0.0
A	1.14E-4	0.0416	0.0274	0.0106	0.0334
C	1.07E-4	0.0333	0.2224	0.0224	0.079
T	0.0	0.0153	9.38E-4	0.0311	0.0328
G	0.0175	0.0323	0.0	0.0626	0.2831

(b) core position 1

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.4240	0.0	0.3773
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.1986	0.0	0.0

(c) core position 2

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.1433
T	0.0	0.0	0.0	0.2372	0.0752
G	0.0	0.2088	0.0	0.0903	0.2451

(d) core position 3

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0514	0.0487	0.0191	0.0
C	0.0	0.0455	0.1943	0.0713	0.1108
T	0.0	0.0144	0.0672	0.0582	0.0362
G	0.0	0.0482	0.0463	0.0	0.1876

(e) core position 4

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0646	0.0469	0.0279	0.0416
C	0.0	0.0	0.1677	0.0397	0.0653
T	0.0	0.0190	0.0488	0.0641	0.04959
G	0.0	0.0560	0.0874	0.0415	0.17924

(f) core position 5

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.1177	0.0	0.0
C	0.0	0.0	0.6201	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.2620	0.0	0.0

(g) post-core

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0815	0.0210	0.0385	0.0389
C	0.0	0.0325	0.2012	0.0617	0.0324
T	0.0	0.0394	0.0568	0.1020	0.0299
G	0.0	0.0269	0.0711	0.0268	0.1387

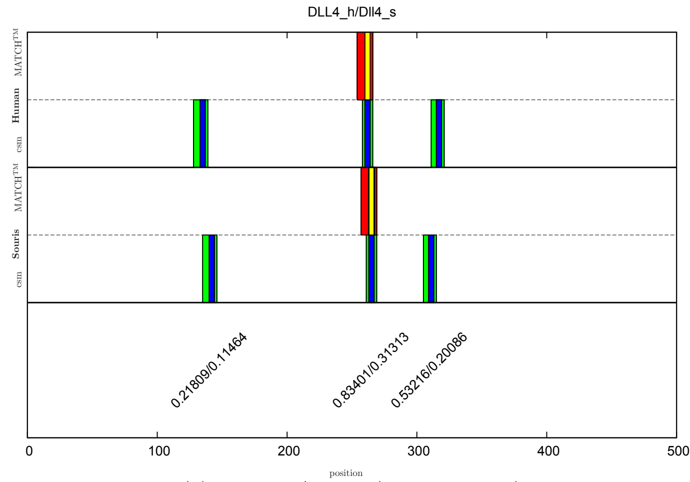
Tab. A.1 – Propabilité des opérations d'édition pour chaque position codant le TFBS ZF5_B dans le CSM.

position	**A**	**C**	**G**	**T**
01	3	6	3	5
02	6	1	9	1
03	3	2	8	4
04	5	2	10	0
05	3	4	8	2
06	5	1	10	1
07	**0**	**14**	**0**	**3**
08	**2**	**0**	**12**	**3**
09	**1**	**11**	**1**	**4**
10	**2**	**2**	**11**	**2**
11	**0**	**14**	**1**	**2**
12	6	0	1	10
13	5	2	3	7

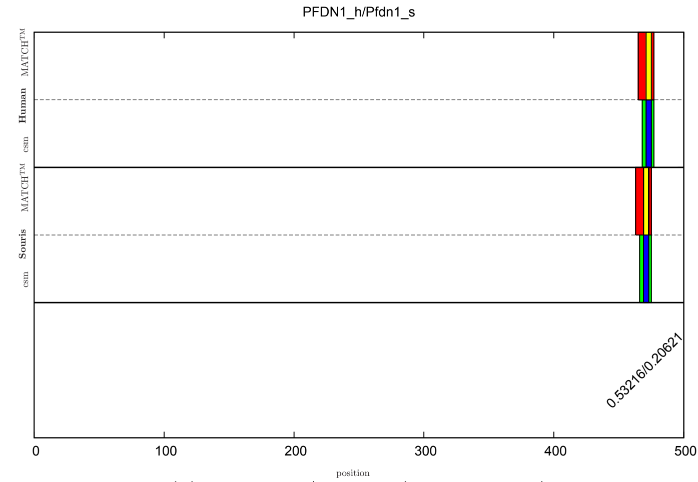
Tab. A.2 – Matrice poids-position pour le TFBS ZF5_B. Les positions du *core* sont en caractères gras.

MATCH^{TM}		CSM	
motif	**proportion**	**motif**	**proportion**
CGCGC	49.56%	CGGCC	22.22%
CGCCC	10.79%	CGCGC	16.66%
CGGGC	8.61%	CGCCC	11.11%
CGCTC	8.45%	CGGGC	11.11%
CGCAC	7.17%	CGTGC	11.11%

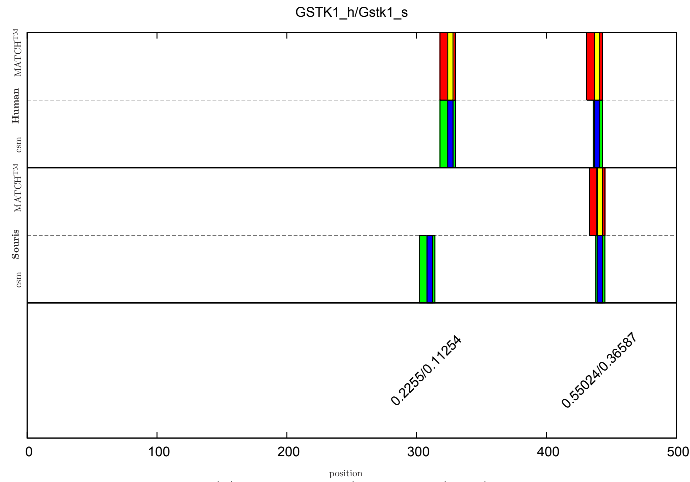
Tab. A.3 – Présentation des principaux *cores* représentant le TFBS ZF5_B trouvés par MATCH^{TM} et le CSM.



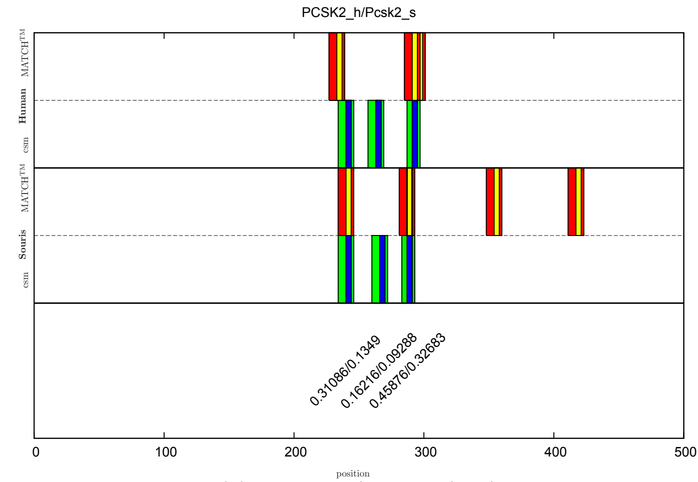
(a) DLL4_h/Dll4_s (apprentissage)



(b) PFDN1_h/Pfdn1_s (apprentissage)

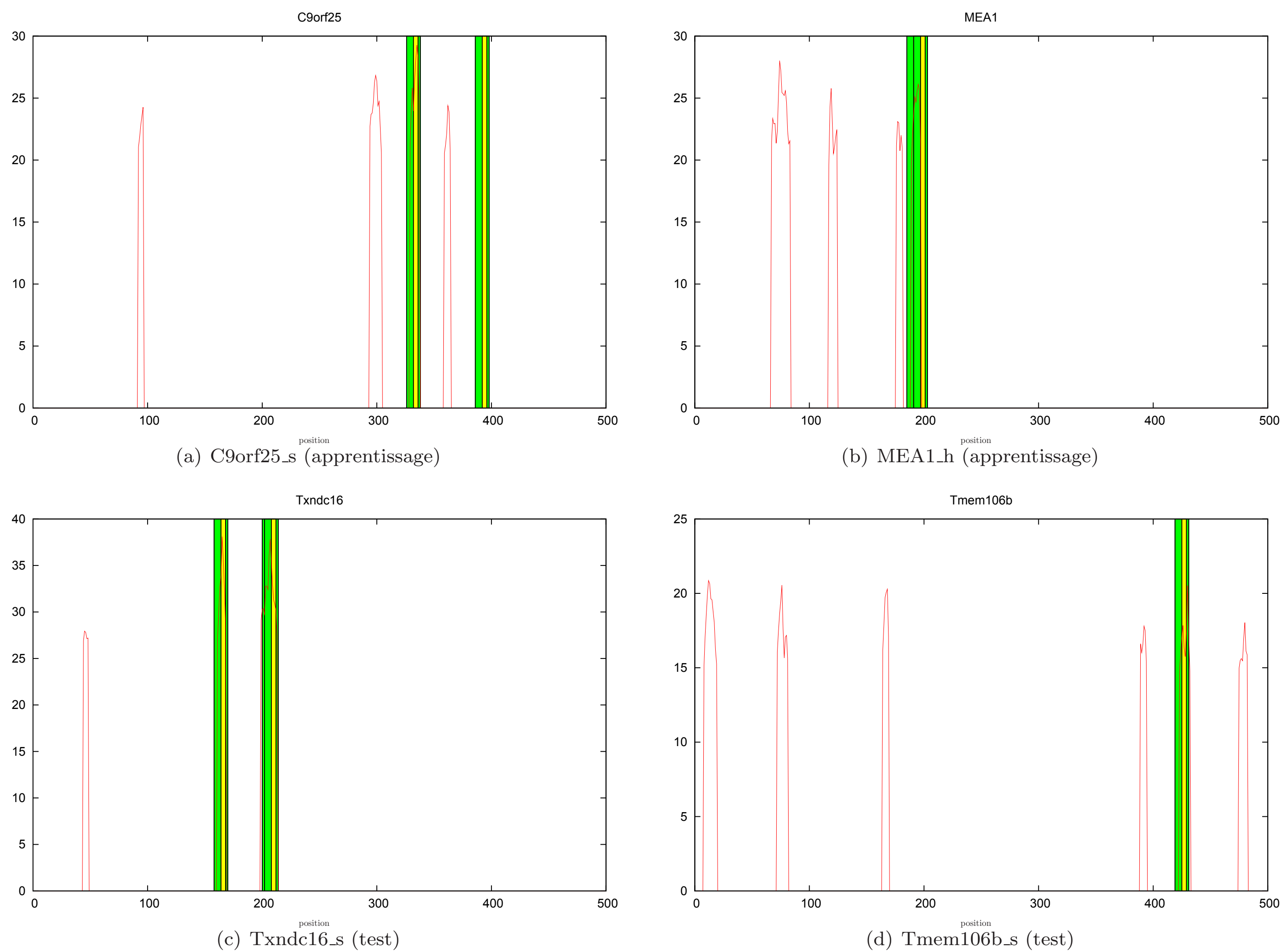


(c) GSTK1_h/Gstk1_s (test)



(d) PCSK2_h/Pcsk2_s (test)

Fig. A.4 – Résultat sur des paires de promoteurs de gènes orthologues.



(a) C9orf25_s (apprentissage)

(b) MEA1_h (apprentissage)

(c) Txndc16_s (test)

(d) Tmem106b_s (test)

Fig. A.5 – Recherche de TFBS sur des promoteurs.

B Expérimentations sur le TFBS PAX4_03

(a) pré-core

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.071	0.033	0.016	0.018
C	0.0	0.042	0.298	0.076	0.032
T	0.0	0.019	0.069	0.125	0.014
G	0.0	0.029	0.055	0.014	0.082

(b) core position 1

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.0	0.0	0.0

(c) core position 2

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	1.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.0	0.0	0.0

(d) core position 3

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.0	0.0	0.0

(e) core position 4

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.0	0.0	0.0

(f) core position 5

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0
G	0.0	0.0	0.0	0.0	0.0

(g) post-core

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.012
A	0.0	0.092	0.055	0.049	0.018
C	0.0	0.024	0.245	0.104	0.055
T	0.012	0.018	0.061	0.147	0.0
G	0.0	0.018	0.030	0.030	0.024

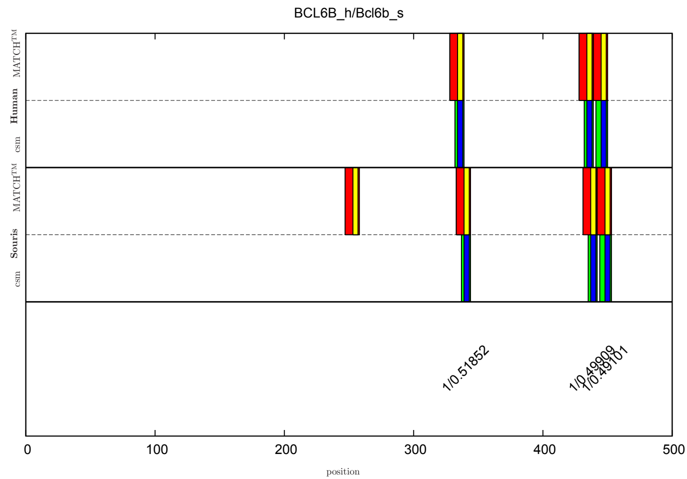
Tab. B.6 – Propabilité des opérations d'édition pour chaque position codant le TFBS PAX4_03 dans le CSM.

position	**A**	**C**	**G**	**T**
01	6	4	4	3
02	8	2	3	4
03	3	5	3	6
04	4	6	3	4
05	4	7	1	5
06	2	9	0	6
07	**1**	**15**	**1**	**0**
08	**16**	**0**	**0**	**1**
09	**1**	**14**	**2**	**0**
10	**3**	**12**	**0**	**2**
11	**1**	**12**	**3**	**1**
12	0	6	6	4

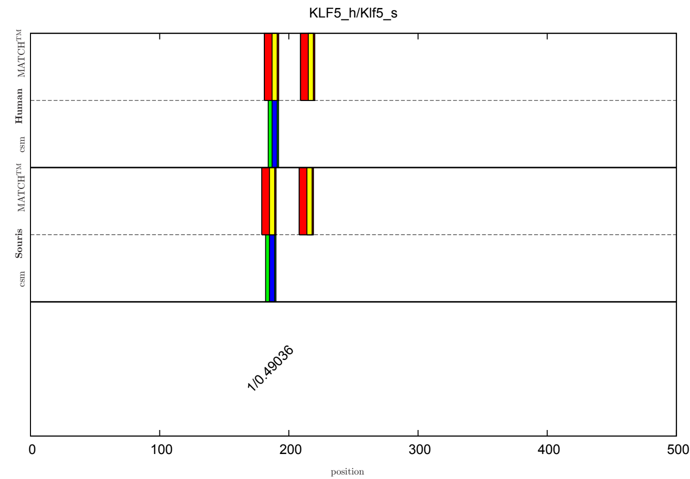
Tab. B.7 – Matrice poids-position pour le TFBS PAX4_03. Les positions du *core* sont en caractères gras.

MATCH^{TM}		CSM	
motif	**proportion**	**motif**	**proportion**
CACCC	100.00%	CACCC	100%

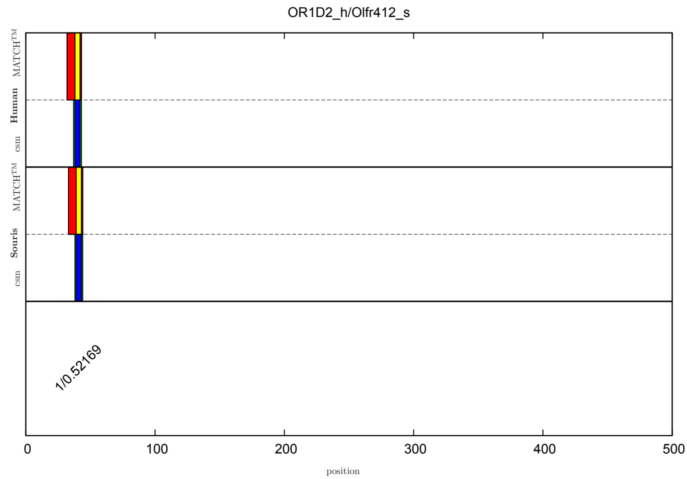
Tab. B.8 – Présentation du *core* représentant le TFBS PAX4_03 trouvé par MATCH^{TM} et le CSM.



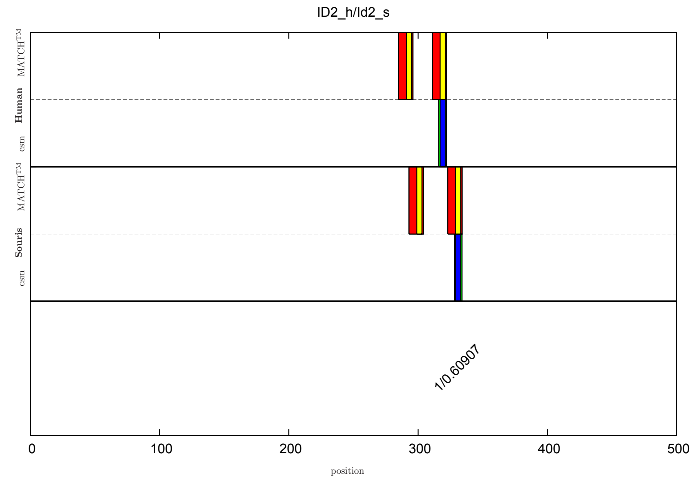
(a) BCL6B_h/Bcl6b_s (apprentissage)



(b) KLF5_h/Klf5_s (apprentissage)

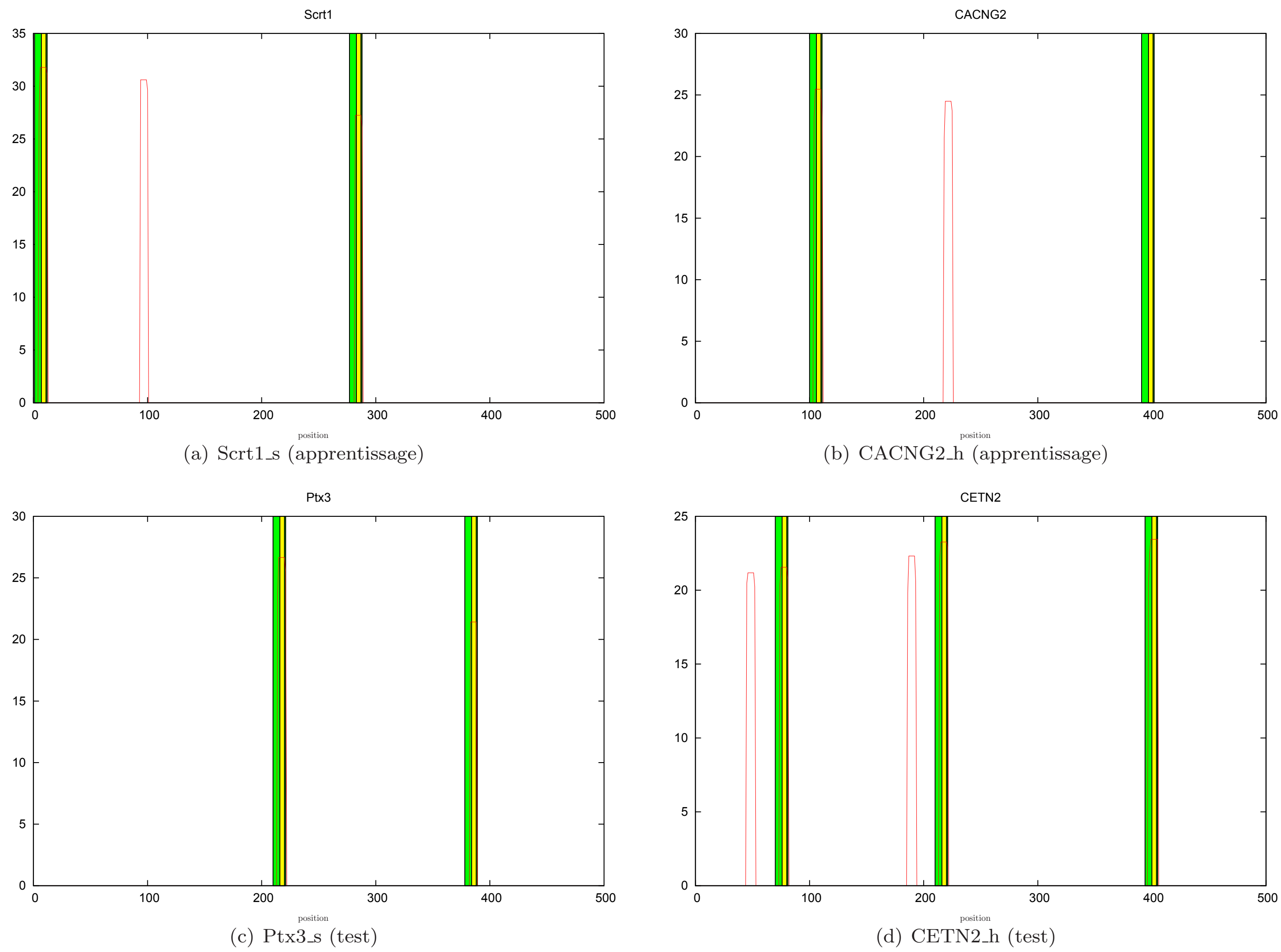


(c) OR1D2_h/Olfr412_s (test)



(d) ID2_h/Id2_s (test)

Fig. B.9 – Résultat sur des paires de promoteurs de gènes orthologues.



(a) Scrt1_s (apprentissage)

(b) CACNG2_h (apprentissage)

(c) Ptx3_s (test)

(d) CETN2_h (test)

Fig. B.10 – Recherche de TFBS sur des promoteurs.

C Expérimentations sur le TFBS PPARG_02

(a) pré-core

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.012	0.0
A	0.008	0.185	0.032	0.049	0.049
C	0.0	0.024	0.098	0.049	0.008
T	0.004	0.049	0.016	0.222	0.020
G	0.0	0.041	0.016	0.008	0.102

(b) core position 1

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0	0.020
C	0.0	0.0	0.041	0.0	0.020
T	0.0	0.0	0.0	0.062	0.0
G	0.0	0.041	0.0	0.0	0.812

(c) core position 2

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.083	0.0	0.0	0.104
C	0.0	0.0	0.020	0.0	0.020
T	0.0	0.0	0.020	0.021	0.020
G	0.0	0.062	0.0	0.0	0.645

(d) core position 3

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	0.0	0.0208	0.0208
C	0.0	0.0	0.0417	0.0208	0.0
T	0.0	0.0833	0.0	0.6875	0.0417
G	0.0	0.0208	0.0	0.0208	0.0417

(e) core position 4

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.020	0.062	0.0	0.062
C	0.0	0.083	0.458	0.041	0.041
T	0.0	0.0	0.020	0.020	0.0
G	0.0	0.041	0.020	0.0	0.125

(f) core position 5

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.0	0.0
A	0.0	0.895	0.0	0.0	0.020
C	0.0	0.0	0.020	0.020	0.0
T	0.0	0.0	0.0	0.041	0.0
G	0.0	0.0	0.0	0.0	0.0

(g) post-core

	λ	A	C	T	G
λ	0.0	0.0	0.0	0.004	0.0
A	0.004	0.124	0.028	0.028	0.025
C	0.0	0.031	0.242	0.035	0.012
T	0.0	0.030	0.025	0.228	0.022
G	0.0	0.019	0.012	0.017	0.105

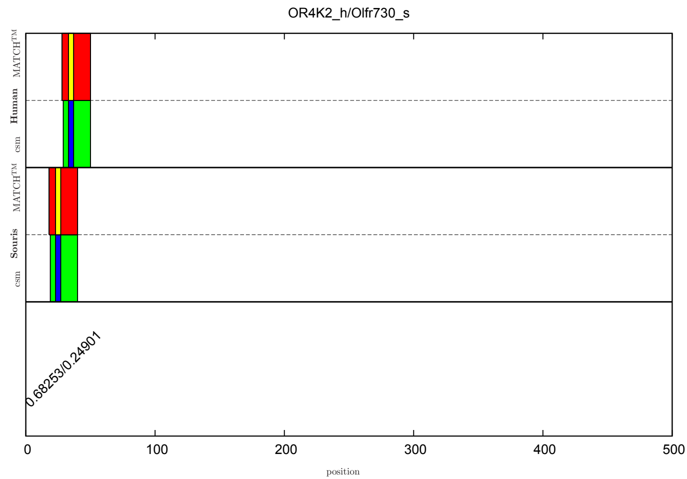
Tab. C.11 – Propabilité des opérations d'édition pour chaque position codant le TFBS PPARG_02 dans le CSM.

position	**A**	**C**	**G**	**T**
01	18	3	0	2
02	21	0	0	6
03	3	7	15	2
04	1	1	0	26
05	21	0	7	0
06	**0**	**0**	**28**	**0**
07	**0**	**1**	**27**	**0**
08	**0**	**0**	**3**	**25**
09	**1**	**23**	**4**	**0**
10	**28**	**0**	**0**	**0**
11	4	17	3	4
12	6	7	9	6
13	3	3	19	3
14	0	0	0	28
15	2	5	20	1
16	24	3	1	0
17	0	28	0	0
18	0	28	0	0
19	0	11	0	17
20	25	1	0	2
21	4	15	6	2
22	7	0	1	19
23	2	0	0	22

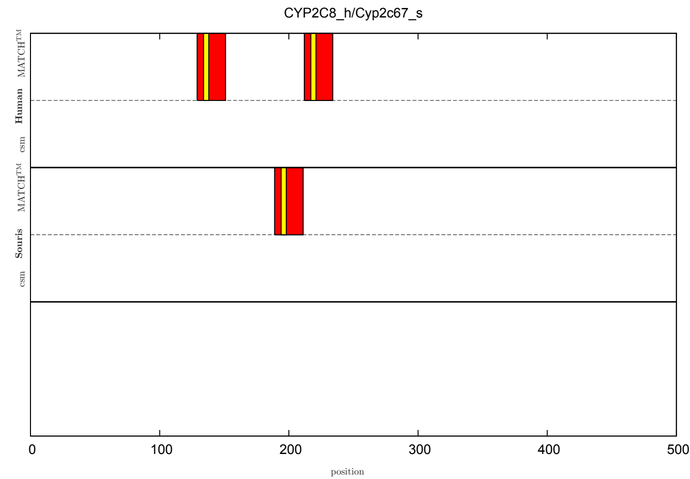
Tab. C.12 – Matrice poids-position pour le TFBS PPARG_02. Les positions du *core* sont en caractères gras.

MATCH^{TM}		CSM	
motif	**proportion**	**motif**	**proportion**
GGTCA	15.05%	GATCA	100%
GGTGA	6.56%		
GGCCA	4.95%		
GGGCA	4.5%		
GCTCA	3.34%		

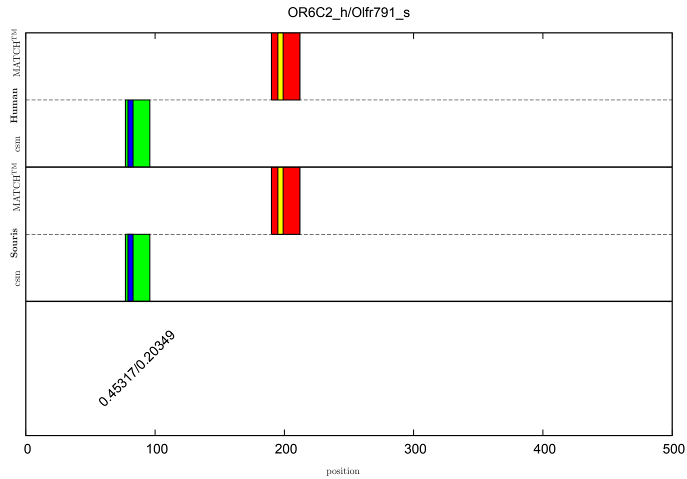
Tab. C.13 – Présentation des principaux *cores* représentant le TFBS PPARG_02 trouvés par MATCH^{TM} et le CSM.



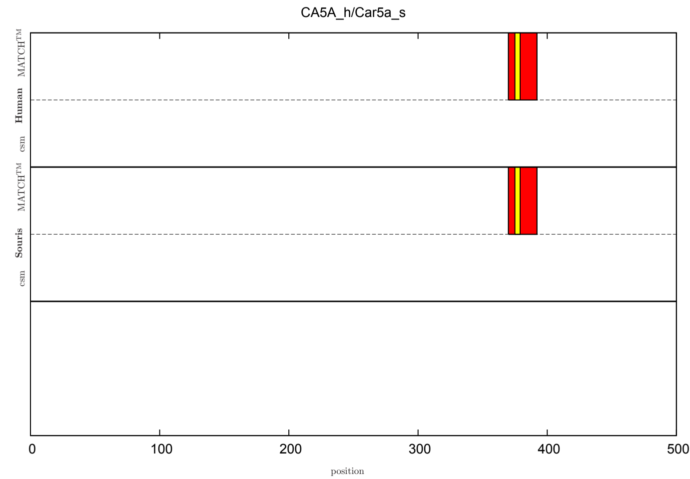
(a) OR4K2_h/Olfr730_s (apprentissage)



(b) CYP2C8_h/Cyp2c67_s (apprentissage)

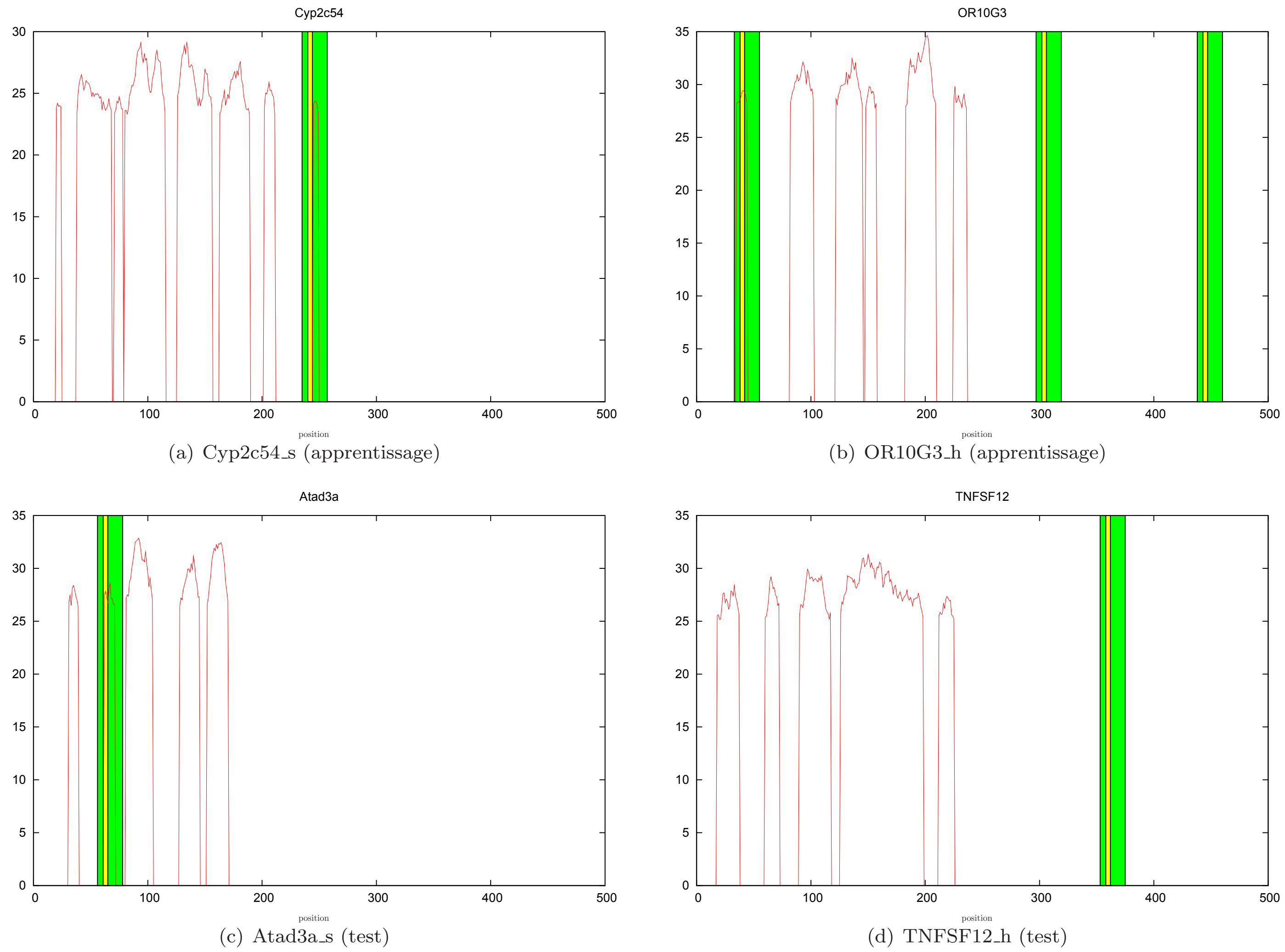


(c) OR6C2_h/Olfr791_s (test)



(d) CA5A_h/Car5a_s (test)

Fig. C.14 – Résultat sur des paires de promoteurs de gènes orthologues.



(a) Cyp2c54_s (apprentissage)

(b) OR10G3_h (apprentissage)

(c) Atad3a_s (test)

(d) TNFSF12_h (test)

Fig. C.15 – Recherche de TFBS sur des promoteurs.

Bibliographie

[AHRU09] Massih-Reza Amini, Amaury Habrard, Liva Ralaivola, et Nicolas Usunier. Proceedings of the ecml-pkdd workshop on learning from non-iid data : Theory, algorithms and practice, 2009. http://www-connex.lip6.fr/~amini/ecml-wk-lniid.html.

[BBHS08] Marc Bernard, Laurent Boyer, Amaury Habrard, et Marc Sebban. Learning probabilistic models of tree edit distance. *Pattern Recognition*, 41(8) : 2611–2629, 2008.

[BBMS10] Aurelien Bellet, Marc Bernard, Thierry Murgue, et Marc Sebban. Learning state machine-based string edit kernels. *Pattern Recognition*, 43 : 2330–2339, 2010.

[BBS08a] Maria-Florina Balcan, Avrim Blum, et Nathan Srebro. Improved guarantees for learning via similarity functions. Dans *Conference on Learning Theory (COLT)*, pages 287–298. Omnipress, 2008.

[BBS08b] Maria-Florina Balcan, Avrim Blum, et Nathan Srebro. A theory of learning with similarity functions. *Machine Learning Journal*, 72 : 89–112, 2008.

[BDBC+10] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, et Jennifer W. Vaughan. A theory of learning from different domains. *Machine Learning Journal*, 79(1-2) : 151–175, 2010.

[BEH+08] Laurent Boyer, Yann Esposito, Amaury Habrard, José Oncina, et Marc Sebban. SEDiL : Software for edit distance learning. Dans *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, volume LNCS 5212, pages 672–677. Springer, 2008.

[BGH+09] Laurent Boyer, Olivier Gandrillon, Amaury Habrard, Mathilde Pellerin, et Marc Sebban. Learning constrained edit state machines. Dans *International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 734–741. IEEE Computer Society, 2009.

[BHMS08] Laurent Boyer, Amaury Habrard, Fabrice Muhlenbach, et Marc Sebban. Learning string edit similarities using constrained finite state machines. Dans *Conférence Francophone sur l'Apprentissage Automatique (CAp)*, pages 37–52. Cépaduès, 2008.

[BHS07] Laurent Boyer, Amaury Habrard, et Marc Sebban. Learning metrics between tree structured data : Application to image recognition. Dans *European Conference on Machine Learning (ECML)*, volume LNCS 4701, pages 54–66. Springer, 2007.

[Bil05] Philip Bille. A survey on tree edit distance and related problem. *Theoretical Computer Science*, 337(1-3) : 217–239, 2005.

[BJS06] Marc Bernard, Jean-Christophe Janodet, et Marc Sebban. A discriminative model of stochastic edit distance in the form of a conditional transducer. *Grammatical Inference : Algorithms and Applications*, 4201 : 240–252, 2006.

[BMD07] Sabri Bayoudh, Laurent Miclet, et Arnaud Delhay. Learning by analogy : A classification rule for binary and nominal data. Dans *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 678–683. Morgan Kaufmann Publishers, 2007.

[Bor04] Sean Borman. The expectation maximization algorithm – a short tutorial. Rapport technique, University of Notre Dame, South Bend, 2004.

[BT04] Guillaume Bouchard et William Triggs. The trade-off between generative and discriminative classifiers. Dans *International Conference on Computational Statistics (COMPSTAT)*, pages 721–728. Springer, 2004.

[CH67] Thomas M. Cover et Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1) : 21–27, 1967.

[CHM04] Corinna Cortes, Patrick Haffner, et Mehryar Mohri. Rational kernels : Theory and algorithms. *Journal Maching Learning Research*, 5 : 1035–1062, 2004.

[CM10] Antoine Cornuéjols et Laurent Miclet. *Apprentissage artificiel : Concepts et algorithmes*. Eyrolles, 2010.

[CO94] Rafael C. Carrasco et José Oncina. Learning stochastic regular grammars by means of a state merging method. Dans *International Colloquium on Grammatical Inference (ICGI)*, pages 139–152. Springer, 1994.

[Col97] Michael Collins. The em algorithm. Rapport technique, MIT Computer Science and Artificial Intelligence Laboratory , Cambridge, 1997.

[DD07] Modan K. Das et Ho-Kwok Dai. A survey of dna motif finding algorithms. *BMC Bioinformatics*, 8(Suppl 7) : S21, 2007.

[DDE05] Pierre Dupont, François Denis, et Yann Esposito. Links between probabilistic automata and hidden markov models : probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9) : 1349–1371, 2005.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, et Graeme Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[Del02] Frank Dellaert. The expectation maximization algorithm. Rapport technique, College of Computing, Georgia Institute of Technology, 2002.

[DLR77] Arthur P. Dempster, Nan M. Laird, et Donald B. Rubin. Maximum likelihood from incomplete data via the *em* algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1) : 1–38, 1977.

[DSO79] Margaret O. Dayhoff, Rebecca M. Schwartz, et Bruce C. Orcutt. A model of evolutionary change in proteins. Dans *Atlas of Protein Sequence and Structure*, volume 5(Suppl. 3), pages 345–351. National Biomedical Reasearch Foundataion, 1979.

[DT05] Serge Dulucq et Hélène Touzet. Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms*, 3(2-4) : 448–471, 2005.

[Fre61] Herbert Freeman. On the encoding of arbitrary geometric configurations. *Institute of Radio Engineers, Transactions on Electronic Computers*, EC-10 : 260–268, 1961.

[GAdM05] Maarten Grachten, Josep Lluis Arcos, et Ramon López de Mántaras. Melody retrieval using the implication/realization model. Dans *International Conference On Music Information Retrieval (ISMIR)*, 2005.

[GBMO95] Eva Gómez-Ballester, Luisa Micó, et José Oncina. Testing the linear approximating and eliminating search algorithm in handwritten character recognition tasks. Dans *Spanish Symposium on Pattern Recognition and Image Analysis (PRIA)*. AERFAI, 1995.

[GXTL10] Xinbo Gao, Bing Xiao, Dacheng Tao, et Xuelong Li. A survey of graph edit distance. *Pattern Analysis & Applications*, 13 : 113–129, 2010.

[HAA04] Yuichiro Hourai, Tatsuya Akutsu, et Yutaka Akiyama. Optimizing substitution matrices by separating score distributions. *BMC Bioinformatics*, 20 : 863–873, 2004.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2) : 147–160, 1950.

[Har58] H. O. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14(2) : 174–194, 1958.

[HH92] Steven Henikoff et Jorja G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22) : 10915–10919, 1992.

[HQRS08] Amaury Habrard, Jose-Manuel Iñesta Quereda, David Rizo, et Marc Sebban. Melody recognition with learned edit distances. Dans *Structural, Syntactic, and Statistical Pattern Recognition*, volume LNCS 5342, pages 86–96. Springer, 2008.

[Jac12] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11 : 37–50, 1912.

[Jar95] Matthew A. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14(5-7) : 491–8, 1995.

[Joa03] Thorsten Joachims. Learning to align sequences : A maximum-margin approach. Rapport technique, Department of Computer Science, Cornell University, Ithaca, NY 14853, 2003.

[KD08] M. Koudritsky et E. Domany. Positional distribution of human transcription factor binding sites. *Nucleic Acids Research*, 36(21) : 6795–6805, 2008.

[Kel55] John L. Kelley. *General topology*. Springer-Verlag, New York, 1955.

[KGR+03] Alexander E. Kel, Ellen Gößling, Ingmar Reuter, Evgeny Cheremushkin, Olga V. Kel-Margoulis, et Edgar Wingender. Match^{TM} : a tool for searching transcription factor binding sites in dna sequences. *Nucleic Acids Research*, 31(13) : 3576–3579, 2003.

[Kle98] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. Dans *European Symposium on Algorithms (ESA)*, pages 91–102. Springer, 1998.

[Lep03] Yves Lepage. De l'analogie rendant compte de la commutation en linguistique. Habilitation à diriger les recherches, Université de Grenoble, 2003.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10 : 707–710, 1966.

[LJ04] Haifeng Li et Tao Jiang. A class of edit kernels for svms to predict translation initiation sites in eukaryotic mrnas. *Journal of Computational Biology*, 12 : 718, 2004.

[LMP01] John Lafferty, Andrew McCallum, et Fernando Pereira. Conditional random fields : Probabilistic models for segmenting and labeling sequence data. Dans *International Conference on Machine Learning (ICML)*, pages 282–289. Morgan Kaufmann, 2001.

[LSB+08] Johan Leyritz, Stephane Schicklin, Sylvain Blachon, Celine Keime, Celine Robardet, Jean-Francois Boulicaut, Jeremy Besson, Ruggero Pensa, et Olivier Gandrillon. Squat : A web tool to mine human, murine and avian sage data. *BMC Bioinformatics*, 9(1) : 378–389, 2008.

[MBD08] Laurent Miclet, Sabri Bayoudh, et Arnaud Delhay. Analogical dissimilarity : Definition, algorithms and two experiments in machine learning. *Journal of Artificial Intelligence Research*, 32 : 793–824, 2008.

[MBDM07] Laurent Miclet, Sabri Bayoudh, Arnaud Delhay, et Harold Mouchère. De l'utilisation de la proportion analogique en apprentissage artificiel. Dans *Journées d'Intelligence Artificielle Fondamentale (AFIA)*, 2007.

[MBP05] Andrew McCallum, Kedar Bellare, et Fernando Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. Dans *Uncertainty in Artificial Intelligence (UAI)*, pages 388–395. AUAI Press, 2005.

[Meh09] Yashar Mehdad. Automatic cost estimation for tree edit distance using particle swarm optimization. Dans *Joint Conference of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 289–292. ACL, 2009.

[MO98] Luisa Micó et José Oncina. Comparison of fast nearest neighbour classifiers for handwritten character recognition. *Pattern Recognition Letters*, 19 : 351–356, 1998.

[NB05] Michel Neuhaus et Horst Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B : Cybernetics*, 35 : 503–514, 2005.

[NB06] Michel Neuhaus et Horst Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 39(10) : 1852–1863, 2006.

[NB07] Michel Neuhaus et Horst Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1) : 239 – 247, 2007.

[NW70] Saul B. Needleman et Christan D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48 : 443–453, 1970.

[OS06] José Oncina et Marc Sebban. Learning stochastic edit distance : application in handwritten character recognition. *Pattern Recognition*, 39(9) : 1555–1812, 2006.

[PGH98] Marc Parizeau, Nadia Ghazzali, et Jean-François Hébert. Optimizing the cost matrix for approximate string matching using genetic algorithms. *Pattern Recognition*, 31 : 431–440, 1998.

[PY10] Sinno Jialin Pan et Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10) : 1345–1359, 2010.

[Rab90] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, 1 : 267–296, 1990.

[RY98] Eric S. Ristad et Peter N. Yianilos. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(20) : 522–532, 1998.

[Sel77] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6) : 184–186, 1977.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27 : 379–423, 623–656, 1948.

[SVA06] Hiroto Saigo, Jean-Philippe Vert, et Tatsuya Akutsu. Optimizing amino acid substitution matrices with a local alignment kernel. *BMC Bioinformatics*, 7 : 246–257, 2006.

[SW81] Temple F. Smith et Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147 : 195–197, 1981.

[Tai79] Kuo C. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26(3) : 422–433, 1979.

[TDdlH00] Franck Thollard, Pierre Dupont, et Colin de la Higuera. Probabilistic dfa inference using kullback-leibler divergence and minimality. Dans *International Conference on Machine Learning (ICML)*, pages 975–982. Morgan Kaufmann, 2000.

[TGNK00] Roman L. Tatusov, Michael Y. Galperin, Darren A. Natale, et Eugene V. Koonin. The cog database : a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, 28(1) : 33–36, 2000.

[Val84] Leslie G. Valiant. A theory of the learnable. Dans *Symposium on Theory of Computing (STOC)*, pages 436–445. ACM, 1984.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[WCF+01] Edgar Wingender, Xin Chen, Ellen Fricke, Robert Geffers, Reinhard Hehl, Ines Liebich, Mathias Krull, Volker Matys, Holger Michael, Richard Ohnhäuser, Manuela Prüß, Frank Schacherer, Susanne Thiele, et Sandra Urbach. The transfac system on gene expression regulation. *Nucleic Acids Research*, 29(1) : 281–283, 2001.

[WF74] Robert A. Wagner et Michael J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21 : 168–173, 1974.

[XNZ08] Shiming Xiang, Feiping Nie, et Changshui Zhang. Learning a mahalanobis distance metric for data clustering and classification. *Pattern Recognition*, 41 : 3600–3612, 2008.

[Yan07] Liu Yang. Distance metric learning : A comprehensive survey. Rapport technique, Michigan State University, 2007.

[ZS89] Kaizhong Zhang et Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6) : 1245–1262, 1989.

Résumé.

De nombreuses applications informatiques nécessitent l'utilisation de distances. Dans le cadre de données structurées, chaînes ou arbres, nous utilisons majoritairement la distance d'édition. Celle-ci correspond au nombre minimal d'opérations d'édition (insertion, délétion et substitution) nécessaire pour transformer la première donnée en la seconde. Suivant l'application traitée, il est possible de paramétrer la distance d'édition en associant à chaque opération d'édition un poids. Dans le cadre de ce manuscrit, nous proposons une technique d'apprentissage automatique supervisé pour apprendre les poids de la distance décrite précédemment. L'algorithme utilisé, appelé Expectation-Maximisation, maximise la vraisemblance des paramètres du modèle à l'aide d'un échantillon d'apprentissage composé de paires d'exemples considérés comme similaires.

La première contribution de ce manuscrit est une extension de précédents travaux sur les chaînes aux arbres sous la forme de transducteur à un unique état. Nous montrons sur une tâche de reconnaissance de caractères manuscrits, l'efficacité de l'apprentissage par rapport à l'utilisation de poids non appris. La seconde est une approche sur les chaînes sous contraintes. Le modèle est représenté par un ensemble fini d'états dans lequel les transitions sont contraintes. Une contrainte est représentée par un ensemble fini de fonctions booléennes définies sur la chaîne d'entrée et une de ses positions. Nous utilisons notre modèle pour aborder une application de recherche de sites de facteur de transcription dans des séquences génomiques.

Abstract.

In computer science, a lot of applications use distances. In the context of structured data, strings or trees, we mainly use the edit distance. The edit distance is defined as the minimum number of edit operation (insertion, deletion and substitution) needed to transform one data into the other one. Given the application, it is possible to tune the edit distance by adding a weight to each edit operation. In this work, we use a supervised machine learning approach to learn the weight of edit operation. The exploited algorithm, called Expectation-Maximisation, is a method for finding maximum likelihood estimates of parameters in a model given a learning sample of pairs of similar examples.

The first contribution is an extension of earlier works on string to trees. The model is represent by a transducer with a single state. We apply successfully our method on a handwritten character recognition task. In a last part, we introduce a new model on strings under constraints. The model is made of a finite set of states where the transitions are constrained. A constraint is a finite set of boolean functions defined over an input string and one of its position. We show the relevance of our approach on a molecular biology task. We consider the problem of detecting Transcription Factor Binding Site in DNA sequences.