MASTER EN SCIENCES INFORMATIQUES

Réseaux bayésiens
application à la reconnaissance de textes manuscrits

Bonbled, Antoine; Munezero, Sandrine

Award date:
2006

Réseaux bayésiens

Application à la reconnaissance de textes manuscrits

Antoine Bonbled, Sandrine Munezero

Mémoire présenté en vue de l'obtention du grade de
Maître en Informatique
Année académique 2005 - 2006

Résumé

Ce travail a pour premier objectif de définir le concept de réseau bayésien, les mécanismes de propagation d'information au sein de ceux-ci et les méthodes d'apprentissage automatique de ces réseaux.
Son second objectif est d'appliquer ces différentes notions à la problématique de la reconnaissance de textes structurés écrits à la main, en particulier la reconnaissance de formules mathématiques manuscrites. Cette problématique sera traitée en trois temps : la reconnaissance du texte, la correction de ses erreurs et l'analyse du comportement de l'auteur du texte.
Le travail envisage ensuite l'extension des mécanismes de propagation dans les réseaux bayésiens afin de mieux les adapter à la problématique de la reconnaissance. Il permet d'affirmer que les réseaux bayésiens peuvent être adaptés à des situations particulières.
Enfin, leur confrontation avec un modèle concurrent - le modèle des grammaires stochastiques - permet de mettre en évidence certaines limites des réseaux Bayésiens.

Mots-clefs : réseau bayésien ; écrit à la main ; reconnaissance ; formule mathématique ; grammaire non contextuelle ; maximum propagation ; grammaire stochastique

Abstract

In a first stage, this document aims at defining Bayesian Networks, propagation mechanisms and learning methods. In a second stage, the goal is to apply the Bayesian Networks theory to the recognition of handwritten mathematical formulas. This application is perfomed in three steps : formulas recognition, user mistakes correction and user behaviour analyzis.
Further, an extension of the propagation mechanisms is considered and applied to the recognition. This extention brings the proof that the Bayesian networks theory can be adapted to particular field. Finally, the concurrent model of stochastic grammars is considered in order to find some limits to the Bayesian model.

Keywords : Bayesian network ; handwritten ; recognition ; mathematical formula ; context-free grammar ; maximum propagation ; stochastic grammar

Avant-propos

Que Monsieur J. Fichefet, qui a accepté d'être le directeur de ce mémoire et que Monsieur F.V. Jensen, qui a soutenu notre travail de recherche, trouvent ici le témoignage de notre vive reconnaissance.

Nous tenons aussi à remercier l'Université d'Aalborg pour son accueil et la société Hugin Expert pour la licence de logiciel qu'elle nous a accordée.

Nous remercions enfin toutes les personnes qui, par leur expérience et leur connaissance du problème, ont apporté leur contribution à cette étude.

Table des matières

Glossaire

Algorithme EM	Algorithme d'apprentissage de paramètres d'un réseau bayésien à partir d'un ensemble de données dont le mécanisme de base consiste à estimer les données manquantes pour ensuite calculer les paramètres	38
Arbre de jonction	Arbre dont les noeuds font partie d'un ensemble de cliques C et dont les arcs vérifient la propriété d'intersection courante	35
Classifieur	Entité qui est capable de lire une phrase écrite à la main et de donner pour chaque lettre de la phrase un ensemble de poids non négatifs pour les différents symboles que cette lettre pourrait représenter	58
Clique	Ensemble maximal de noeuds tous connectés entre eux	35
Correction a posteriori	Mécanisme de correction d'erreur dans lequel la correction est effectuée après le choix de la phrase admissible la plus probable	77
Correction a priori	Mécanisme de correction d'erreur dans lequel la correction est effectuée au même moment que le choix de la phrase admissible la plus probable	77
Formule admissible la plus probable	Formule avec la plus grande probabilité et respectant le langage mathématique utilisé	58
Graphe complet	Ensemble de noeuds tous connectés entre eux	35

Interpreteur	Entité qui utilise les résultats du classifieur afin de déterminer la phrase admissible la plus probable et corrige cette phrase si elle contient des erreurs tout prenant en compte le comportement de l'utilisateur	58
Mauvais utilisateur	Utilisateur insérant des fautes dans les formules mathématiques qu'il écrit	88
Maximum-propagation	Algorithme de propagation utilisant l'opérateur maximum pour marginaliser les variables et permettant de trouver la configuration la plus probable des variables du réseau	35
Propagation	Transmission d'information dans un réseau bayésien d'un noeud à un autre dans le but de mettre à jour leurs probabilités après l'arrivée d'un évènement	35
Propriété d'intersection courante	Si C_1 et C_2 sont des cliques alors pour chaque chemin entre C_1 et C_2 chaque clique du chemin contient $C_1 \cap C_2$	35
Règle de précédence	Règle d'une grammaire non contextuelle qui peut être exprimée par une grammaire régulière	77
Règle non contextuelle	Règle d'une grammaire non contextuelle qui ne peut pas être exprimée par une grammaire régulière	77
Réseau bayésien	Graphe orienté acyclique exprimant des relations causales entre variables aléatoires où le poids d'un arc est donné par la probabilité du noeud enfant dans le contexte du noeud parent.	18
Somme-propagation	Algorithme de propagation utilisant l'opérateur somme pour marginaliser les variables	35

Utilisateur fatigué	Utilisateur écrivant de manière peu précise et difficile à reconnaître	88

Introduction

L'intelligence artificielle se trouve aujourd'hui au coeur des préoccupations majeures de l'informatique. Elle trouve ses applications dans des champs de connaissances très variés, dans la mesure où elle offre la possibilité d'automatiser des tâches telles que l'analyse et la déduction d'informations ou d'actions. A la source de ces différentes tâches, se trouve généralement un ensemble d'expériences répétitives qui permet d'obtenir des informations prédictives.

L'une des particularités des domaines dans lesquels l'intelligence artificielle est mise à l'épreuve est l'incertitude quant aux conclusions obtenues. En effet, les données étudiées par une application dite intelligente ne représentent qu'un échantillon réduit de la réalité. Par ailleurs, dans certaines situations, les données observées expriment également une certaine incertitude.

C'est à ce titre que les modèles qui permettent d'intégrer de façon aisée le concept de probabilité sont très rapidement associés aux recherches effectuées en intelligence artificielle. Les réseaux bayésiens n'ont pas fait exception à la règle. Dès les années quatre vingts, ils ont attiré l'attention des informaticiens. Leur relative simplicité en termes de représentation de l'incertitude et la sophistication des calculs mis au point par Thomas Bayes au dix-huitième siècle ont constitué un atout non négligeable.

En guise d'illustration, citons notamment l'emploi très répandu des réseaux bayésiens dans le domaine médical, comme outils de diagnostic ou encore dans le champ de la reconnaissance, un domaine phare de l'intelligence artificielle.

Les réseaux bayésiens sont en outre de puissants outils probabilistes qui permettent d'intégrer le concept de causalité et, plus encore, de déduire l'existence de relations causales entre évènements à partir d'ensembles de données. De cette façon, ils ouvrent la porte à de nouvelles avancées dans le domaine de l'analyse statistique d'expérimentations.
Judea Pearl, auteur des premiers algorithmes modernes en matière de réseau bayésiens, affirme dans son dernier ouvrage ([23]) : *"Causality is not mystical or metaphysical. It can be understood in terms of simple processes, and it can be expressed in a friendly mathematical language, ready for computer analysis."*

Aujourd'hui, les nombreux travaux développés autour de ces modèles témoignent de l'engouement des chercheurs pour les modèles de Bayes. Toutefois, l'enthousiasme suscité par leur potentiel tend à laisser croire que la méthode est à même de modéliser indifféremment n'importe quelle combinaison d'évènements aléatoires.

Cette dérive empêche de déterminer clairement les avantages apportés par les réseaux bayésiens et, surtout, les conditions dans lesquelles l'utilisation de ceux-ci s'avère pertinente.

C'est dans cette perspective que s'inscrit la présente étude. Dans un premier temps (**I**), elle poursuit l'objectif d'appréhender la théorie générale à la source du développement des réseaux bayésiens.
Elle prendra en considération à la fois la définition stricte de ces modèles, les différentes méthodes de construction automatique de ceux-ci à partir de données et les algorithmes qui amènent à s'en servir. Néanmoins, au vu de l'étendue de cette théorie et de la diversité de techniques existantes pour une même fonctionnalité, nous limiterons volontairement le travail aux méthodes les plus représentatives.

Une conclusion quant à l'intérêt réel d'un outil requiert également son évaluation dans un contexte pratique, à condition qu'il soit assez représentatif du champ d'application habituel de l'outil.

Notre démarche nous amènera donc à étudier dans un second temps (**II**) l'application des réseaux de Bayes à la reconnaissance de textes manuscrits. La reconnaissance manuscrite est aujourd'hui primordiale. Dans le cadre de l'informatisation progressive de notre société, elle en constitue une préoccupation constante puisqu'elle passe nécessairement par la numérisation de tous les documents papier utilisés jusqu'ici.

Cependant, étant donné l'ampleur du problème et le nombre de travaux déjà effectués dans ce cadre, notre démarche se limitera à la problématique de la reconnaissance de textes structurés mathématiquement. L'enjeu n'est pas ici seulement de déterminer quels caractères ont été écrits mais bien aussi de vérifier si la grammaire mathématique utilisée est respectée par le texte reconnu. Autrement dit, notre analyse portera sur la problématique de l'interprétation de formules mathématiques.

Cette analyse comportera trois étapes. D'abord, les efforts seront concentrés sur la détermination de la formule mathématique respectueuse de la grammaire qui est la plus proche de celle à reconnaître (*section* **5.2**).
Nous nous attarderons ensuite à la prise en compte des erreurs de grammaire que l'auteur du texte mathématique aurait pu effectuer et de leur correction (*section* **5.3**). Enfin, des types d'utilisateurs seront pris en considération du point de vue de leur apport à l'activité d'interprétation (*section* **5.4**).

Au terme de cette analyse, nous évaluerons dans un troisième temps (**III**) les résultats obtenus et essaierons de dépasser les impasses dans lesquelles notre raisonnement aurait éventuellement abouti, en explorant d'avantage la théorie des réseaux bayésiens ou en portant notre regard sur des modèles concurrents.

Notre projet est d'abord de s'interroger sur les fonctionnalités de base offertes par les réseaux bayésiens et plus spécifiquement sur leur capacité d'adaptation aux exigences particulières de situations diverses (*chapitre* **6**).

Dans le cas de l'application à la reconnaissance de textes structurés, nous tenterons de voir s'il est possible d'étendre les modèles bayésiens pour qu'ils puissent donner non pas une réponse et sa probabilité mais bien un classement de différentes réponses selon leur probabilité. Nous interrogerons la possibilité de combiner plusieurs techniques existantes en vue d'améliorer la qualité des réponses obtenues.

Nous confronterons enfin le modèle de Bayes à un modèle concurrent - celui des grammaires stochastiques - afin d'apprécier la capacité de ce dernier à prendre en charge les faiblesses du premier (*chapitre* **7**).

Nous serons alors en position de déterminer dans quelle mesure l'engouement collectif évoqué plus haut pour les modèles de Bayes se justifie, en particulier dans le cadre de la reconnaissance de formules mathématiques manuscrites.

Première partie

Théorie générale des réseaux bayésiens

Chapitre 1

Etat de l'art *A. Bonbled, S. Munezero*

1.1 Réseaux bayésiens

Dans sa première ébauche du concept de réseau qui allait porter son nom, Thomas Bayes, au dix-huitième siècle, proposait un modèle mathématique pour calculer des probabilités au travers de plusieurs variables, liées de manière causale. Ce modèle permettait de rencontrer les difficultés associées au calcul de variables difficilement décelables par expérimentation.

Il a fallu attendre ensuite les années quatre vingt pour que ce modèle connaisse de nouveaux développements. Le regain d'intérêt soulevé par les réseaux bayésiens s'explique notamment par les avancées dans un modèle voisin, celui des réseaux neuronaux. Ce dernier a ouvert la porte à de nombreuses applications dans le domaine de l'intelligence artificielle puisqu'il autorisait l'apprentissage efficace d'un champ de connaissances.

Cependant, son principal défaut résidait dans son incapacité à modéliser l'imprévu, n'ayant pas à sa disposition de données suffisamment sûres. Les réseaux bayésiens, quant à eux, comblent cette lacune. Ils offrent en effet la possibilité d'intégrer l'incertitude et se révèle être de cette manière des outils prédictifs par excellence.

On doit à l'américain Judea Pearl ([22]) la définition des algorithmes nécessaires à l'utilisation des réseaux bayésiens, soit les algorithmes calculant la propagation de l'information entre les différentes variables composant un réseau de ce type. Ses travaux ont suscité d'emblée l'intérêt et ont engendré de nouvelles perspectives de développement du modèle bayésien.

Citons en particulier les recherches effectuées au Danemark ([15]) pour augmenter l'efficacité des algorithmes. Celles-ci sont à l'origine, dans les années nonante, de Hugin Expert, l'outil actuel de référence en matière d'utilisation des réseaux bayésiens.

Ces dernières années, les réseaux bayésiens continuent à faire l'objet de développements considérables.

Mentionnons en premier lieu les travaux consacrés à la propagation de l'information dans un réseau bayésien ou, en d'autres, mots, le calcul de probabilités a posteriori, une fois qu'une nouvelle information est introduite dans le modèle. Un nombre non négligeable de recherches travaillent à l'optimisation des algorithmes de propagation qui se sont révélés en pratique relativement inefficaces dans le cas de réseaux de taille importante.

A cet égard, dans [13], les auteurs exposent les méthodes exactes ou approchées, mises au point dernièrement pour les algorithmes de propagation dont le but est de trouver la configuration la plus probable des variables d'un réseau. Trouver la meilleure explication d'un ensemble de variables dans un réseau bayésien est une fonctionnalité extrêmement utile dans le cas de la reconnaissance, comme nous le montrerons.

Dans [25], les chercheurs essayent de transformer un réseau bayésien en un autre modèle, appelé *weighted model counting*, dont la complexité peut se révéler parfois meilleure que celle de la propagation dans les réseaux bayésiens. Ce modèle considère un ensemble de clauses propositionnelles auxquelles sont associées des poids.

Un autre aspect des réseaux bayésiens concerne leur incapacité à prendre en charge les cycles qui pourraient apparaître dans leur structure. Ceci peut constituer un obstacle qui empêche la modélisation de certains champs de la réalité. Plusieurs recherches se sont penchées sur le traitement de cette difficulté.
Dans [28], on suggère qu'un cycle représente une famille de distributions probabilistes plutôt qu'une seule distribution. Considérant cette interprétation des cycles, les auteurs proposent des extensions des réseaux bayésiens afin qu'ils puissent prendre en compte les cycles. Etudiant la sémantique et la complexité de ces réseaux étendus, ils arrivent à la conclusion qu'il faut définir un nouveau formalisme puisque les cycles changent tous les calculs définis pour les réseaux bayésiens classiques.

D'autres formes particulières de réseaux bayésiens ont été présentées, dans le cadre de certaines applications. Une forme étendue bien connue d'un réseau bayésien classique est le réseau bayésien dynamique. Son principe est d'intégrer la contrainte du temps, en découpant le réseau en tranches temporelles. Les relations entre variables d'une même tranche expriment des relations de simultanéité alors que les relations inter tranches expriment un déplacement dans le temps. Les premiers travaux sur ces structures sont abordés dans [12] et trouvent des applications utiles le domaine génétique.

De nouvelles études tentent de particulariser le concept de réseau bayésien à des situations particulières. Un exemple parmi d'autres est celui du réseau qui permet la modélisation des concepts de fiabilité des informations et de l'analyse de systèmes : chaque noeud du réseau représente un composant d'un système et, ses états, les échecs possibles du composant, correspondant à des points consécutifs dans le temps ([4]).

Parallèlement au développement des algorithmes de propagation d'information dans les réseaux bayésiens, les chercheurs se sont aussi concentrés sur le concept d'apprentissage de réseaux, c'est-à-dire leur construction automatique à partir d'un ensemble de données observées.

Le premier type d'apprentissage est l'apprentissage de paramètres. Il consiste à partir d'un ensemble de données et d'une structure afin de calculer les probabilités a priori du réseau.

Les algorithmes d'apprentissage de paramètres sont multiples. Le plus connu est sans doute l'algorithme *Expectation Maximisation* défini dans [8] et réadapté dans [18] pour profiter au mieux de la structure des réseaux bayésiens. Il est étudié dans le cadre de ce mémoire.

Néanmoins, ils ne couvrent pas toutes les situations, en particulier dans le cas où des données de l'ensemble étudié sont manquantes. Les travaux les plus récents visent à éviter le risque latent de trouver un maximum local lors de la recherche du paramètre le plus adapté à l'ensemble de données étudiés. Ainsi, dans [21], les auteurs présentent plusieurs algorithmes évolutifs qui permettent de ne pas se cloisonner aux maxima locaux, réévaluant plusieurs fois un résultat lorsqu'il a été trouvé, et fonctionnant par "mutation adaptative".

D'autres recherches prennent en charge un problème dans l'apprentissage qui apparaît lorsque le réseau traité contient des milliers de variables. L'apprentissage demeure dans ce cas extrêmement complexe. Des nouveaux algorithmes ont été mis au point. Leur principe est d'effectuer des apprentissages locaux, en sélectionnant des régions dites d'intérêt. L'algorithme *Max Min Bayesian Network*, cité dans [27] a, par exemple, pour avantage de pouvoir être arrêté à tout moment et de ne travailler que dans une région d'intérêt, définie autour d'une variable cible.

Le deuxième type d'apprentissage est l'apprentissage de structure, qui permet de trouver la structure d'un réseau bayésien à partir d'un ensemble de données. Ce type d'apprentissage n'est pas dénué d'intérêt. En effet, il a été prouvé que s'il n'y a pas de variables cachées (non trouvées à partir de l'ensemble de données), les algorithmes d'apprentissage de structure sont capables de trouver, de manière automatique, des relations de causalité entre variables, par une simple analyse de données ([23]).

Les recherches sur les techniques d'apprentissage sont loin d'avoir abouti. Des objectifs restent à atteindre.
D'une part, il s'agit de trouver les algorithmes les plus efficaces puisque les actuels ne le sont pas toujours. Dans le cas d'un apprentissage de structure qui génère un ensemble de structures possibles et essaie ensuite de choisir la meilleure, ceci est imputable à leur complexité. Au contraire, dans le cas d'un apprentissage qui effectue des tests de corrélation sur les données, le manque d'efficacité provient des tests de corrélation eux-mêmes.
D'autre part, il faut parvenir à détecter les variables cachées par différents tests sur les données, afin de réellement déceler leurs relations de causalité ([16]).

Dans le cas de très grands réseaux, il a été montré qu'un réseau de type classifieur (permettant, sur base de variables représentant des attributs, de trouver la classe d'une variable cible), devient très inefficace si toutes les variables représentant des attributs sont les causes directes de la variable cible. On essaie généralement, dans ce cas, de regrouper certaines variables représentant des attributs et de les faire influencer la variable cible indirectement (figure 1.1). Les probabilités a priori sont alors moins complexes à trouver.

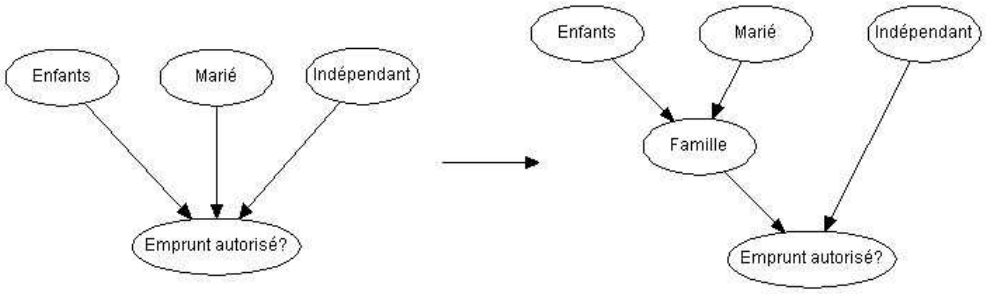


FIG. 1.1 – Transformation d'un classifieur

Les auteurs de [10], évaluent des approches pour effectuer ces regroupements de variables et construire des structures de classifieur plus intelligentes, à partir d'ensembles de données. Il s'agit d'une forme d'apprentissage de structure, adaptée à une situation particulière.

1.2 Reconnaissance de caractères

Le domaine de la reconnaissance de textes et de formules mathématiques est lui aussi l'objet de recherches variées. Cependant, ces travaux se concentrent généralement sur des problématiques particulières autour de l'activité de reconnaissance mais portent rarement sur le processus global de la reconnaissance, à savoir la découverte de la phrase, sa correction, la prise en compte de l'utilisateur, etc.

Une première catégorie de travaux essaye de trouver le meilleur moyen de reconnaître les symboles, en divisant correctement une phrase en suite de blocs. Dans [6] et [3], l'importance de cette division est démontrée via des exemples non linéaires comme cet exemple utilisant des limites, où le but est de séparer le mot "*lim*" de la variable de cette limite et du symbole infini en dessous (figure 1.2). Les auteurs présentent des techniques efficaces pour effectuer ce calcul de blocs.

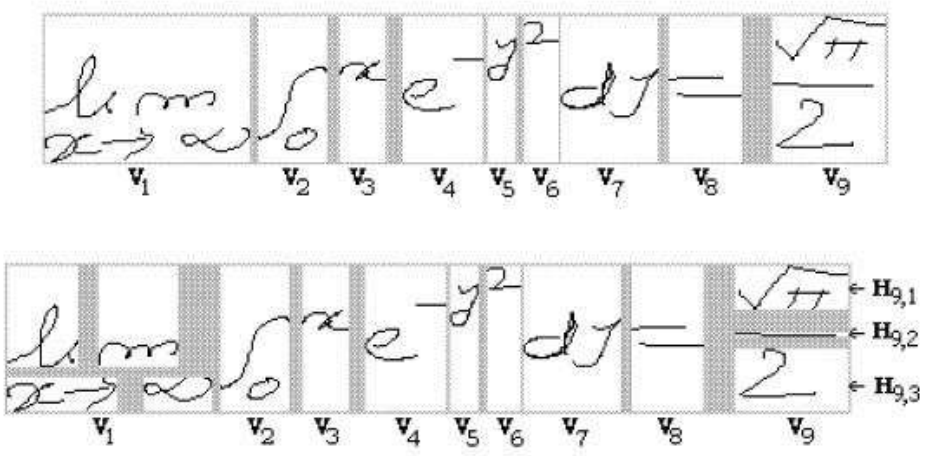


FIG. 1.2 – Formule non linéaire divisée en blocs

Le but principal de cette catégorie de travaux est de distinguer les expressions mathématiques dans des textes manuscrits.
Malheureusement, en ce qui nous concerne, ils ne permettent pas de reconnaître l'ensemble de la phrase et sa structure, parce qu'ils utilisent uniquement le voisinage direct de chaque symbole (voisins de gauche et de droite) pour choisir le bon symbole, sans tenir compte de l'ensemble de la phrase.

Dans [20], les auteurs utilisent une évaluation plus globale des caractères, via la propagation de l'information mais uniquement dans le cas où deux ou plusieurs caractères se chevauchent et peuvent être une partie du même objet mathématique. Ils ne s'occupent pas seulement du voisinage gauche et droite mais vont plus loin dans l'exploration de la phrase. Toutefois, ils ne vont pas jusqu'à évaluer l'entièreté de la formule.

Les travaux de la deuxième catégorie proposent des modèles et des méthodes reconnaissant une chaîne de caractères dans une grammaire non contextuelle particulière. Leur pertinence réside dans la non contextualité du langage mathématique dont ils font usage. Leur tort est de supposer que les symboles de la chaîne qui sont reconnus, sont certains.
En général, la notion d'incertitude de la reconnaissance n'est pas prise en compte dans cette catégorie.

Dans cette même catégorie, d'autres études visent à prendre en compte les symboles non linéaires, l'évolution des règles de grammaire, les erreurs que l'utilisateur peut éventuellement commettre ou encore l'apprentissage à partir de ses habitudes.

Dans [11], le but est de reconnaître des adresses postales à partir de chaînes de caractères fournies en entrée. Le modèle proposé utilise une grammaire non contextuelle bien définie, un modèle graphique de réseaux de transitions utilisé pour pouvoir trouver une adresse valide et un convertisseur de la grammaire en graphe.

Ils définissent aussi, comme beaucoup d'autres travaux, un moyen automatique ou semi automatique pour trouver la grammaire non contextuelle à partir d'une collection de données. Par exemple, dans [24], les auteurs utilisent une table de dérivation qui enregistre tous les arbres de dérivation possibles pour une chaîne de terminaux et des algorithmes génériques pour partitionner et fusionner les non terminaux de cette table sur base d'exemples de chaînes de caractères.

La dernière catégorie de recherches traite de la détection d'erreurs et de la correction dans les phrases manuscrites analysées par un parseur, objet que les travaux des catégories précédentes délaissent souvent.

Les auteurs de [5] divisent les erreurs trouvées en plusieurs catégories : les erreurs lexicales (un symbole est écrit de manière peu précise), les erreurs syntaxiques (parenthèses de droite manquantes), les erreurs sémantiques (mauvais opérande pour un opérateur) et les erreurs de logique (la somme de un et de un fait trois). Leur idée est que, dans le but de corriger ce genre d'erreurs, il est nécessaire d'améliorer le parseur utilisé pour éviter que ce parseur n'échoue à cause d'une de ces erreurs.

Le parseur ainsi amélioré est alors capable d'accepter des erreurs et de les corriger d'une certaine manière. Par exemple, si un opérateur est sans opérande, le parseur ajoute un epsilon après cet opérateur. Si une parenthèse de gauche n'est pas contrebalancée (i.e. fermée par une parenthèse de droite), la parenthèse sera considérée comme un caractère C. Avec cette technique, il n'y a plus d'erreurs dans la formule et l'utilisateur peut éventuellement changer ce qu'il souhaite. Dans [26], les auteurs préfèrent utiliser des méthodes complètement interactives où l'utilisateur peut intervenir et ordonner au programme ses souhaits.

Le champ de la reconnaissance de textes structurés est large : les travaux exposés précédemment éveillent beaucoup de questions. La question principale à laquelle l'application développée dans ce travail essaie de répondre concerne l'utilisation de l'incertitude et l'obtention de la phrase la plus probable qui tiendrait compte à la fois des règles et des probabilités. Elle devrait en outre prendre en compte le comportement spécifique de l'utilisateur et la correction de ses erreurs.

Chapitre 2

Concepts des réseaux bayésiens

A. Bonbled, S. Munezero

2.1 Définition

Les réseaux bayésiens désignent une structure de données probabilistes particulière, qui allie une définition formelle stricte à une représentation lisible et permet de modéliser l'incertitude. Nés des règles fondamentales de la probabilité, dont celle de Bayes, ils sont aujourd'hui utilisés de manière performante dans le domaine du datamining et de l'intelligence artificielle.

Une définition plus formelle est donnée dans [15], elle utilise les concepts de la théorie des graphes. Un réseau bayésien est, dans cette perspective, un graphe orienté sans cycle dont les arcs décrivent des relations causales entre les noeuds qui le composent. Rappellons qu'un graphe orienté est acyclique s'il ne contient pas de chemin $A_1 \rightarrow A_2 \rightarrow ... \rightarrow A_n$ avec $A_1 = A_n$ (où A_i est un noeud).
Il pourrait arriver que le domaine à modéliser contienne des cycles (figure 2.2). Par conséquent, il ne pourra pas être modélisé par un réseau bayésien qui, par définition, ne peut en contenir.

Les noeuds du réseau bayésien représentent la plupart du temps des variables aléatoires discrètes. Une variable aléatoire peut être définie comme un ensemble exhaustif de possibilités mutuellement exclusives que sont ses états.

La structure du réseau symbolise aussi des relations de causalité, le noeud a la source d'un arc est une cause dont l'effet est le noeud ciblé par cet arc. Ces relations de causalité sont aussi considérées comme relations de dépendance probabiliste entre les variables aléatoires le composant, apportant ainsi une information supplémentaire au modèle.

En outre, les relations causales ont un aspect quantitatif, soit le poids donné aux arcs du réseau. Supposons par exemple qu'un réseau bayésien est composé de trois noeuds et que le noeud A est parent du noeud B (figure 2.1).
Le poids du lien entre A et B est alors la probabilité conditionnelle de A par B, soit $Pr[B|A]$, qui donne "la probabilité de B, dans le contexte de A". Ce concept de probabilité conditionnelle est plus rigoureusement défini dans la section suivante.

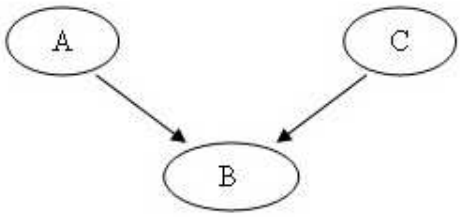


FIG. 2.1 – Réseau bayésien où les variables A et C causent conjointement l'état de la variable B

Si on considère à présent que C est aussi un parent de B (figure 2.1), les deux probabilités conditionnelles $Pr[B|A]$ et $Pr[B|C]$ prises séparément n'apprennent rien sur l'influence jointe de A et C sur B. A et C pourraient interagir de plusieurs manières pour déterminer B. Il est par conséquent nécessaire de spécifier $Pr[B|A, C]$.

Formellement, à chaque variable A avec pour parents les variables $B_1, ..., B_n$, une table de potentiels est attachée, $Pr[A|B_1, ..., B_n]$. Si A n'a pas de parents, la table se réduit à la probabilité non conditionnelle $Pr[A]$. Dans le cas du réseau bayésien de la figure 2.1, les probabilités à priori $Pr[A]$ et $Pr[C]$ doivent être spécifiées.

Les probabilités qu'il est nécessaire de spécifier lors de la construction du réseau sont appellées "probabilités a priori", en opposition aux probabilités mises à jour après l'arrivée d'une nouvelle information, qui sont appellées "probabilités a posteriori".

Bien que l'interprétation des arcs est une interprétation causele, la définition mathématique des réseaux bayésiens n'implique pas le principe de causalité. Elle implique cependant la dépendance probabiliste qui existe entre un noeud parent et son noeud enfant.

La figure 2.2 donne un exemple de graphe orienté contenant un cycle, ce qui n'est pas autorisé dans un réseau bayésien.

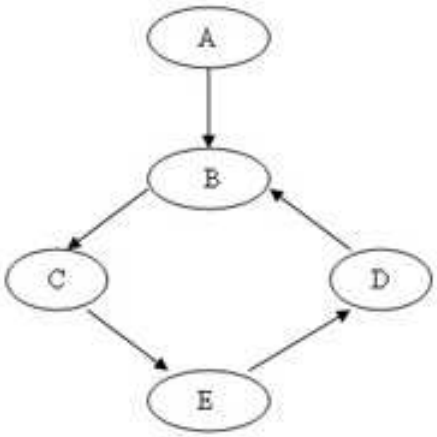


FIG. 2.2 – Graphe orienté contenant un cycle

2.2 Illustration

Pour mieux comprendre le concept de réseau bayésien, prenons une illustration simple.

Supposons que le domaine étudié est celui des étudiants Danois, en particulier ceux qui choisissent de faire leurs études à l'Université d'Aalborg, réputée pour son concept novateur d'organisation des cours par projets.

La première chose qu'il peut être intéressant à représenter, c'est la probabilité qu'un étudiant Danois choisisse l'Université d'Aalborg. Sur un échantillon, on désire alors analyser pour ces étudiants là, mais aussi pour les autres, l'importance du vélo comme moyen de se rendre sur le campus universitaire et la participation effective dans les projets organisés tout au long de l'année (un étudiant a le choix de participer a des projets ou non).
On peut considérer qu'être étudiant à Aalborg peut causer le fait de se rendre à l'université à vélo et influencer la participation à des projets universitaires.

Se rendre chaque jour aux cours à vélo et participer à des projets peut engendrer une fatigue chez l'étudiant qui peut, par exemple, être mesurée en quatre paliers (fatigue inexistante, peu importante, importante, surmenage).
La participation à des projets universitaires, apporte à l'étudiant une expérience pratique non négligeable.

On déduit de cette explication les variables aléatoires suivantes : *Aalborg* (l'étudiant est-il inscrit à l'Université d'Aalborg ou non ?), *Projet* (participe-t-il à des projets universitaires ?), *Vélo* (se rend-il à vélo sur le campus ?), *Fatigue* (quel est le degré de fatigue de l'étudiant ?) et *Expérience* (a-t-il acquis de l'expérience pratique ?).
En analysant les influences décrites dans l'explication précédente et en utilisant les variables découvertes, il est à présent possible de représenter le domaine par un réseau bayésien (figure 2.3).

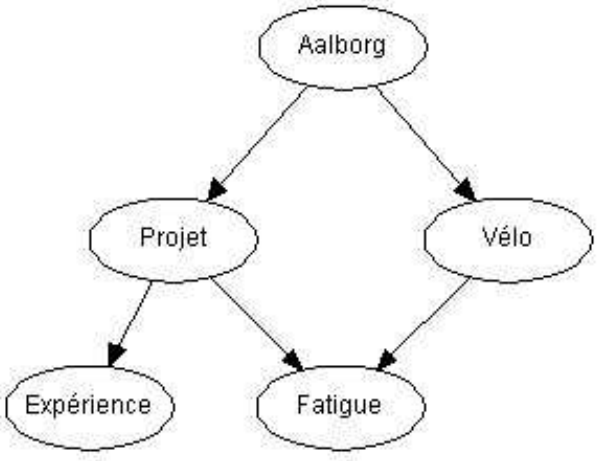


FIG. 2.3 – Réseau bayésien pris en exemple

Les probabilités à définir pour pouvoir utiliser ce réseau sont les probabilités *Pr[Aalborg]*, *Pr[Projet |Aalborg]*, *Pr[Vélo|Aalborg]*, *Pr[Fatigué |Projet, Vélo]* et *Pr[Expérience | Projet]*. Ces probabilités pourraient être celles définies en 2.4.

Aalborg	**Pr[Aalborg]**
oui	0.6
non	0.4

Pr[Projet | Aalborg]	*Aalborg*	
Projet	**oui**	**non**
oui	0.9	0.6
non	0.1	0.4

Pr[Vélo | Aalborg]	*Aalborg*	
Vélo	**oui**	**non**
oui	0.4	0.35
non	0.6	0.65

Pr[Expérience | Projet]	*Projet*	
Expérience	**oui**	**non**
oui	0.6	0.2
non	0.4	0.8

Pr[Fatigue | Vélo, Projet]	*Vélo* = **oui**		*Vélo* = **non**	
	Projet		*Projet*	
Fatigue	**oui**	**non**	**oui**	**non**
aucune	0.0	0.05	0.05	0.85
peu	0.1	0.7	0.2	0.1
beaucoup	0.75	0.2	0.65	0.05
trop	0.15	0.05	0.1	0.0

FIG. 2.4 – Tables de probabilités du réseau 3.10

A partir d'un réseau bayésien tel que celui-ci, un certain nombre de concepts et de propriétés peut être observé. Il s'agit en particulier des notions de probabilité conditionnelle, d'indépendance, d'évidence et de d-séparation, définies dans les quatre sections suivantes. En outre, elles donnent aux réseaux bayésiens un rôle plus simple que le simple rôle de représentation graphique, ce qui est étudié plus loin.

2.3 Probabilité conditionnelle

Un réseau bayésien est un modèle constitué de concepts prenant leur définition dans un univers incertain. Ceci est exprimé grâce aux probabilités conditionnelles. En général, quand la probabilité d'un évènement A est donnée par $Pr[A]$, celle-ci est conditionnée par d'autres facteurs connus.

Une probabilité conditionnelle peut être expliquée de la manière suivante : "*Connaissant l'évènement B, la probabilité de l'évènement A est X*", ce qui est noté $Pr[A|B] = X$.

Cela ne signifie pas que si B est vrai, la probabilité de A vaut X. Cela signifie plutôt que si B est vrai et que tout autre facteur connu est non pertinent, alors la probabilité de A vaut X ("tout autre facteur" peut être séparé de A dans le contexte B).

La règle fondamentale de calcul des probabilités conditionnelles est :

$$Pr[A|B].Pr[B] = Pr[A, B] \tag{2.1}$$

où $Pr[A, B]$ est la probabilité jointe de $A \wedge B$.

Si l'on considère que toute probabilité est conditionnée par un contexte C, la formule doit devenir :

$$Pr[A|B, C].Pr[B|C] = Pr[A, B|C] \tag{2.2}$$

A partir de (2.1), on peut obtenir $Pr[A|B].Pr[B] = Pr[B|A].Pr[A]$, et par conséquent, la règle de Bayes :

$$Pr[B|A] = \frac{Pr[A|B].Pr[B]}{Pr[A]} \tag{2.3}$$

A nouveau, le contexte doit pouvoir être considéré et la règle de Bayes devient dans ce cas :

$$Pr[B|A, C] = \frac{Pr[A|B, C].Pr[B|C]}{Pr[A|C]} \tag{2.4}$$

La formule (2.2) peut être considérée comme un axiome fondamental de la probabilité.

Elle peut être justifiée en comptant simplement les fréquences, ce que montre l'exemple suivant.

Supposons que l'ensemble C est l'ensemble des étudiants internationaux à l'Université d'Aalborg (n), que l'ensemble B est l'ensemble de ces étudiants au département d'informatique et que A (m) est l'ensemble des étudiants venant de Belgique (i d'entre eux étudient l'informatique).

La figure 2.5 exprime les relations entre ces ensembles.

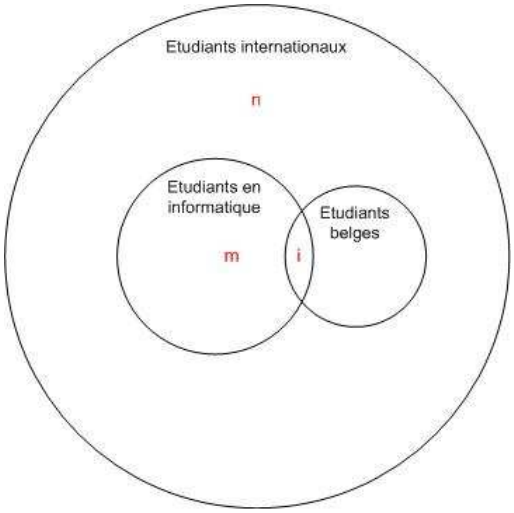


FIG. 2.5 – Classement non exhaustif des étudiants internationaux

Si f est la fréquence des évènements, les propriétés suivantes peuvent être obtenues : $f(A|B,C) = \frac{i}{m}$ et $f(B|C) = \frac{m}{n}$.

Par conséquent, $f(A,B|C) = \frac{i}{n} = \frac{i}{m}.\frac{m}{n} = f(A|B,C).f(B|C)$, qui est la règle fondamentale appliqué à l'exemple.

$Pr[A|B]$ est aussi appellée la *vraisemblance* de A si B, et est alors écrite $V[A|B]$. La raison de cette dénomination peut être expliquée comme suit. Supposons que $B_1, ..., B_n$ sont des scénarios possibles qui ont un effet sur l'évènement A, et que A est certain ; alors, $Pr[A|B_i]$ est une mesure de la vraisemblance de B_i comme cause de A. En particulier, si tous les B_i ont la même probabilité a priori, la règle de Bayes permet de calculer la formule de leur vraisemblance :

$$\begin{aligned} Pr[B_i|A] &= \frac{Pr[A|B_i].Pr[B_i]}{Pr[A]} \\ &= k.Pr[A|B_i] \end{aligned}$$

où k est une constance indépendante de i.

De la probabilité jointe $Pr[A,B]$,la probabilité $Pr[A]$ peut être calculée, en utilisant la sommation ($\sum_B Pr[A,B]$). Cette opération est appellée *marginalisation* et la variable B est dite *marginalisée* de $Pr[A,B]$.

2.4 Indépendance conditionnelle

Le concept d'indépendance conditionnelle est un des concepts fondamentaux des réseaux bayésiens, en particulier lors de la transmission d'information au travers de ceux-ci. Deux variables aléatoires A et C sont indépendantes dans le contexte de B si :

$$Pr[A_i|B_j] = Pr[A_i|B_j, C_k], \forall i, j, k \tag{2.5}$$

Ceci signifie que, si l'état de B est connu, aucune information sur C n'influencera à posteriori la probabilité de A. Ceci est écrit $Pr[A|B] = Pr[A|B, C]$, même si les tables des deux probabilités n'ont pas la même dimension. Si la condition B est inexistante, cela revient simplement à dire que A et C sont indépendants.

Le concept d'indépendance conditionnelle apparaît dans le cas de connexions sérielles. Prenons la figure 2.6, représentant le fait qu'un étudiant international effectuant son stage à Aalborg, s'il est interessé par les traditions Danoises (*TraditionsDanoises*), ira au moins à un souper de Noël (*SouperNoël*) et, par conséquent, boira du Schnaps (*Schnaps*).

Ce réseau de petite taille illustre une connexion sérielle et la propriété d'indépendance conditionnelle intrinsèque à cet type de connexion : *Pr[Schnaps | SouperNoel, TraditionsDanoises] = Pr[Schnaps | SouperNoël]*. En d'autres mots, si l'on sait qu'un étudiant international à Aalborg est allé à un souper de Noël, alors le fait qu'il soit interessé par les traditions Danoises n'a plus aucune influence sur son ingestion de Schnaps.

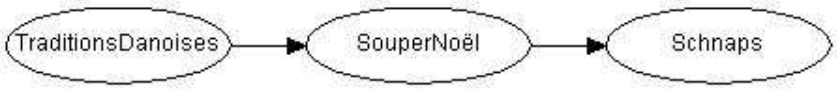


FIG. 2.6 – Réseau bayésien illustrant une connexion sérielle

Le concept d'indépendance conditionnelle apparaît aussi dans le cas de connexions divergentes, comme dans l'exemple de la figure 2.7. Ce dernier représente le fait maintenant bien connu, qu'être étudiant à Aalborg implique à une certaine probabilité l'utilisation du vélo comme moyen de transport vers le campus et, à une certaine probabilité, la participation au système de projets.

Si l'on sait qu'un étudiant fait ses études à Aalborg, alors se rendre sur le campus à vélo n'aura aucune influence sur la participation à des projets. Par contre, si l'on ignore si l'étudiant est oui ou non à Aalborg, le fait qu'il se rende sur le campus à vélo augmente la probabilité qu'il soit bien à Aalborg, ce qui influence la participation à des projets : *Pr[Projet | Aalborg, Vélo] = Pr[Projet | Aalborg]*.

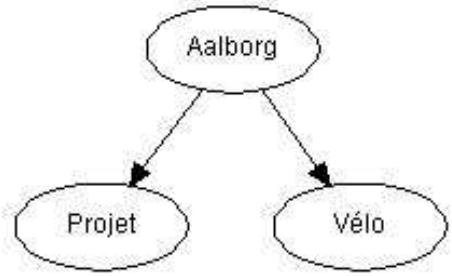


FIG. 2.7 – Réseau bayésien illustrant une connexion divergente

Si la définition (2.5) est vérifiée, la règle de Bayes (2.4) implique :

$$Pr[C|B,A] = \frac{Pr[A|C,B].Pr[C|B]}{Pr[A|B]} \tag{2.6}$$

$$= \frac{Pr[A|B].Pr[C|B]}{Pr[A|B]} \tag{2.7}$$

$$= Pr[C|B] \tag{2.8}$$

Dans [22] et [29], les auteurs montrent que, lorsqu'est supposé un domaine caractérisé par un ensemble de variables aléatoires formant un réseau bayésien, ce dernier n'exprime pas seulement les probabilités conditionnelles explicites mais représente aussi des assertions d'indépendance conditionnelle, implicitement.

Si $X_1, X_2, ...X_n$ est une énumération de tous les noeuds d'un réseau bayésien telle que chaque noeud apparaît après ses parents et si $pa(X_i)$ est l'ensemble des parents d'un noeud x_i, le réseau bayésien exprime l'assertion d'indépendance conditionnelle suivante : chaque variable X_i est conditionnellement indépendante des noeuds $X_1, X_2, ...X_{i-1}$, dans le contexte de ses noeuds parents.

Cette propriété d'une part et les probabilités conditionnelles du réseau d'autre part, définissent une probabilité jointe sur toutes les variables aléatoires de ce réseau. A partir de la règle de chaînage (règle obtenue ci-dessous), il est possible de calculer cette probabilité jointe :

$$\begin{aligned} Pr[X_1, X_2, ...X_n] &= Pr[X_n|X_1, X_2, ...X_{n-1}].Pr[X_1, X_2, ...X_{n-1}] \\ &= Pr[X_n|pa(X_n)].Pr[X_{n-1}|X_1, X_2, ...X_{n-2}].Pr[X_1, X_2, ...X_{n-2}] \\ &= ... \\ &= \prod_{i=1}^{n} Pr[X_i|pa(X_i)] \end{aligned}$$

, où les différentes assertions sont obtenues par les propriétés d'indépendance conditionnelle.

La règle de chaînage permet donc de calculer la probabilité jointe de toutes les variables d'un réseau Bayésien, en multipliant les probabilités de chaque noeud conditionnées par les parents de celui-ci.
La probabilité conditionnelle $Pr[X_i|pa(X_i)]$ doit être donnée dans la spécification du réseau bayésien, pour pouvoir y appliquer n'importe quel raisonnement.

2.5 Evidence

Une *évidence* est une information qui concerne une situation actuelle et pertinente pour le réseau qu'elle modifie. Les réseaux bayésiens sont utilisés pour calculer de nouvelles probabilités quand une information particulière est reçue. Le type d'information reçue est généralement du type "$A = a$", où A est une variable aléatoire et a est un état de A.

Supposons maintenant que A possède n états ($Pr[A] = (x_1, ..., x_n)$) et qu'une certaine information e est obtenue, selon laquelle A peut seulement être dans l'état i ou j. Cette affirmation indique par la même occasion que tous les états à l'exception de i et j sont impossibles et, par conséquent, la nouvelle distribution de probabilités est $Pr[A, e] = (0, ..., 0, x_i, 0, ..., 0, ..., xj, 0, ..., 0)$.

Les évidences peuvent être fortes ou faibles et sont définies de la manière suivante :

- L'*évidence forte* est l'évidence qu'un noeud a une probabilité de 100% d'être dans un état et de 0% d'être dans un autre état.

- L'*évidence faible* est toute évidence qui n'est pas forte. En d'autres mots, c'est l'évidence qu'un noeud a une probabilité de moins de 100% d'être dans un état et/ou une probabilité de plus de 0% d'être dans les autres états. Les évidences faibles sont souvent utilisées pour exprimer une information incertaine comme dans le cas de sources non fiables ou contradictoires. On les appelle aussi *vraisemblances*.

Si A est une variable aléatoire avec n états, une évidence sur A est une table de dimension n de zéros et de un. Pour faire la distinction entre l'information e selon laquelle "A est soit dans l'état i soit dans l'état j" et l'évidence correspondante, la notation soulignée $\underline{e}$ est utilisée dans la formule suivante.

Supposons que $Pr[U]$ est une table de probabilité jointe et que e est une évidence. La table de probabilité jointe $Pr[U, e]$ est la table résultant de $Pr[U]$ où toutes les entrées pour lesquelles A n'est ni dans l'état i ni dans l'état j, reçoivent la valeur 0. Les autres entrées restent inchangées. Ceci revient au même que multiplier $Pr[U]$ par $\underline{e}$:

$$Pr[U, e] = Pr[U].\underline{e}$$

Il est à noter que $Pr[e] = \sum_U Pr[U, e] = \sum_U (Pr[U].\underline{e})$

Concernant les évidences faibles, il est possible de les traiter dans les réseaux bayésiens mais $Pr[e]$ ne peut pas être calculé en suivant le raisonnement précédent, au vu de sa sémantique floue. Dans le cas d'évidences faibles, on procède généralement à une mise à jour des probabilités de la variable par l'arrivée de l'évidence, en utilisant la règle fondamentale et la marginalisation.

Par exemple, supposons que la variable A, qui a deux états (a_1 et a_2), a reçu l'évidence selon laquelle "la probabilité d'être dans l'état a_1 est double de celle d'être dans l'état a_2", ce qui s'écrit $V[A] = (0.66, 0.33)$. La probabilité à priori de A est $Pr[A] = (0.2, 0.8)$.

Pour prendre en compte l'évidence, on calcule $Pr[A|e] = \dfrac{Pr[A,e]}{Pr[e]}$

, avec $Pr[e] = \sum_A Pr[A,e]$ et $Pr[A,e] = Pr[A].e$.

Dans l'exemple, $Pr[A,e] = (0.2, 0.8).(0.66, 0.33) = (0.132, 0.264)$ et $Pr[e] = \sum_A (0.132, 0.264) = 0.396$.

Le résultat est donc $Pr[A|e] = \dfrac{(0.132, 0.264)}{(0.396)} = (0.33, 0.67)$

Lorsqu'on parle de la probabilité d'un noeud dans un réseau bayésien, on parle aussi de croyance en ce noeud. L'évidence est alors considérée comme un changement, une mise à jour de la croyance en un noeud.
Quand une évidence est ajoutée sur un réseau bayésien, il est possible d'analyser les probabilités afin de prédire l'effet que cette évidence aura sur le réseau, en observant les enfants des noeuds modifiés. Il est aussi possible de déterminer la cause de cette évidence en regardant les parents des noeuds modifiés. Ceci se fait par le biais du mécanisme de *propagation*, développé dans le chapitre suivant.

2.6 D-séparation

La définition du concept de d-séparation fait appel aux notions de connexions divergentes, sérielles et convergentes. Un exemple de connexion sérielle et un exemple de connexion divergente ont été donnés dans la section consacrée à l'indépendance conditionnelle.
La figure 2.8 donne un aperçu d'une connexion convergente : pour un étudiant, utiliser le vélo pour se rendre sur le campus et participer à des projets engendrent conjointement une certaine fatigue.

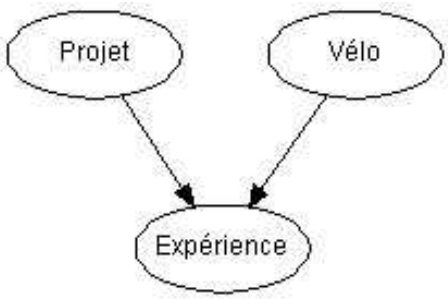


FIG. 2.8 – Réseau bayésien illustrant une connexion convergente

Deux variables aléatoires distinctes A et B sont dites *d-séparées* dans un réseau si, pour tous les chemins reliant A à B, il existe une variable intermédiaire V (distincte de A et B) telle que l'une des deux conditions suivantes est respectée :

- La connexion entre A et B est sérielle ou divergente et V est instancié (a reçu une évidence).

  ou

- Le connexion entre A et B est convergente et, ni V, ni aucun des descendants de V n'a reçu d'évidence.

Si A et B ne sont pas d-séparés, ils sont d-connectés.

On peut remarquer un corrolaire entre les concepts d'indépendance conditionnelle et de d-séparation. Dans le cas d'un réseau à trois noeuds, par exemple, le fait que deux noeuds soient d-séparés par un autre revient à dire qu'ils sont indépendants dans le contexte de cet autre noeud.

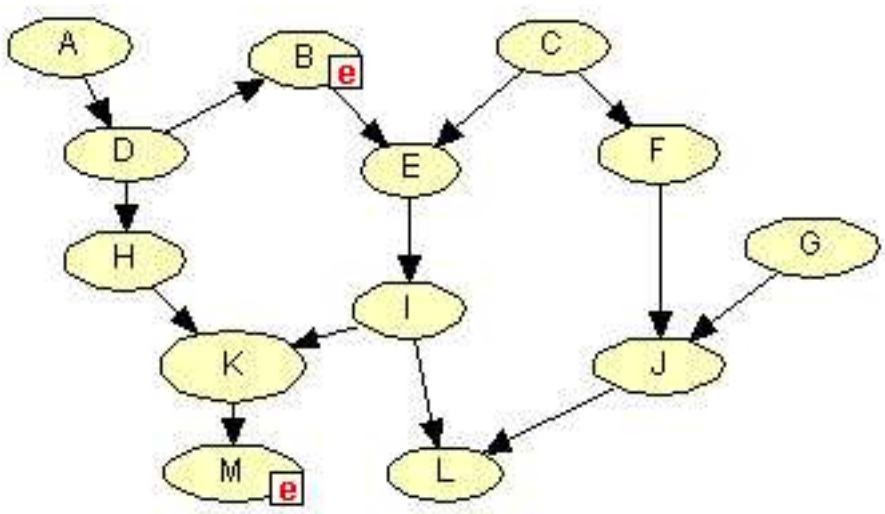


FIG. 2.9 – Réseau bayésien avec B et M instanciés. A est d-séparé de G seulement

Dans [15], un exemple de d-séparation sur un réseau de plus grande taille est donné (figure 2.9). L'évidence donnée sur B et M représente une instanciation. Si une évidence est ajoutée sur A, l'information peut être transmise à D. La variable B, par contre, est bloquée et par conséquent l'information ne peut pas passer de B à E. Cependant, elle peut être transmise de H à K. Puisque le noeud enfant M de K a reçu une évidence, cette évidence peut être transmise de H à I et plus loin, à E, C, F, J, et L. Par conséquent, le chemin $A-D-H-K-I-E-C-F-J-L$ est un chemin d-connecté.

Chapitre 3

Propagation dans un réseau bayésien

A. Bonbled, S. Munezero

3.1 Mise à jour de probabilités

Lorsqu'une évidence est ajoutée sur un noeud du réseau, les probabilités a priori des noeuds voisins doivent changer. Avant d'étudier les algorithmes de propagation qui permettent de déterminer l'ensemble des changements de croyance sur tous les noeuds du réseau, il est nécessaire de montrer comment la mise à jour se fait sur un réseau composé de quelques noeuds, .

Dans un premier temps, un exemple simple peut être donné sur un réseau avec deux noeuds : l'un, indiquant la religion (protestants, catholiques ou musulmans) et l'autre, le nombre d'enfants par famille de cette religion. Le but du réseau est de montrer en quoi la religion influence le nombre d'enfants des familles. Cet exemple est donné dans [16]. La figure 3.1 donne un aperçu de ce réseau et le tableau 3.2 montre ses probabilités a priori.

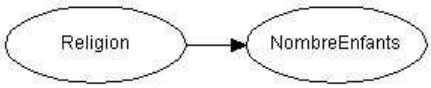


FIG. 3.1 – Réseau bayésien indiquant l'influence religion - nombre d'enfants dans la famille

	Religion		
$NombreEnfants$	**p**	**c**	**m**
0	0.15	0.05	0.05
1	0.2	0.1	0.1
2	0.4	0.2	0.1
3	0.2	0.4	0.4
4 et plus	0.05	0.25	0.35

FIG. 3.2 – $Pr[NombreEnfants|Religion]$

En outre, dans l'échantillon analysé, $Pr[Religion = p] = 0.9$, $Pr[Religion = c] = 0.04$ et $Pr[Religion = m] = 0.06$.

Supposons maintenant que l'évidence $Pr[NombreEnfants = 3]$ est donnée. La question est alors de connaître la modification de la probabilité du noeud religion, suite à cette nouvelle information. En d'autres mots, il faut calculer $Pr[Religion|NombreEnfants = 3]$

La première chose à faire est de calculer la probabilité jointe sur les deux noeuds du réseau, $Pr[Religion, NombreEnfants]$ qui est calculée en divisant $Pr[Religion|NombreEnfants]$ par $Pr[Religion]$, en utilisant la règle fondamentale. Le tableau 3.3 est ainsi obtenu.

	Religion		
NombreEnfants	**p**	**c**	**m**
0	0.135	0.002	0.003
1	0.18	0.004	0.006
2	0.36	0.008	0.006
3	0.18	0.016	0.024
4 et plus	0.045	0.01	0.021

FIG. 3.3 – $Pr[Religion, NombreEnfants]$

A présent, on peut obtenir $Pr[Religion|NombreEnfants = 3]$ en réutilisant la règle fondamentale : on divise $Pr[Religion, NombreEnfants]$ par $Pr[NombreEnfants = 3]$.

Pour obtenir $Pr[NombreEnfants = 3]$, on passe par l'étape de marginalisation, en calculant $\sum_{Religion} Pr[Religion, NombreEnfants = 3]$. Le résultat de la division est la table 3.4

	Religion		
NombreEnfants	**p**	**c**	**m**
3	0.82	0.07	0.11

FIG. 3.4 – $Pr[Religion|NombreEnfants = 3]$

C'est ce principe qui est à la base de toute mise à jour de croyance dans un réseau bayésien : diviser la probabilité jointe sur toutes les variables du réseau par la marginalisation de certaines variables de cette même probabilité jointe.

Si l'on part du réseau bayésien de la figure 3.5, que la probabilité jointe sur toutes les variables, $Pr[A, B, C, D, E, F, G, H]$, a été calculée et que les évidences a, f, g et h ont été mises sur les noeuds A, F, G et H, alors $Pr[B|a, f, g, h]$ peut par exemple être obtenue par cette technique :

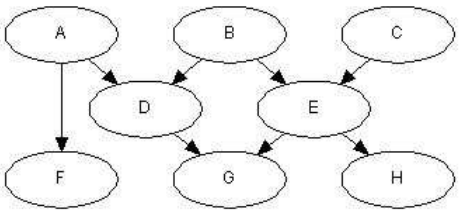


FIG. 3.5 – Réseau bayésien à huit noeuds

$$Pr[B, a, f, g, h] = \sum_{C,D,E} Pr[A, B, C, D, E, F, G, H]$$

$$Pr[B|a, f, g, h] = \sum_{C,D,E} \frac{Pr[B, a, f, g, h]}{Pr[a, f, g, h]}$$

où $Pr[a, f, g, h] = \sum_B Pr[B, a, f, g, h]$

Si la probabilité jointe sur un groupe de variables est exigée par cette technique, le plus logique est de calculer la probabilité jointe sur toutes les variables du réseau et de marginaliser les variables, comme cela a été fait dans l'exemple précédent pour calculer $Pr[B, a, f, g, h]$.
Pour obtenir la probabilité jointe sur toutes les variables d'un réseau, on emploie le plus souvent la règle de chaînage, expliquée dans le chapitre précédent, qui consiste à multiplier les probabilités de tous les noeuds, conditionnées aux parents de ces noeuds. Dans l'exemple, la règle de chaînage donne ceci :

$$Pr[A, B, C, D, E, F, G, H]$$

$$=$$

$$Pr[H|E].Pr[G|D, E].Pr[F|A].Pr[D|A, B].Pr[E|B, C].Pr[A].Pr[B].Pr[C]$$

Cependant, ce calcul peut être très fastidieux et inefficace dans le cas de grands réseaux. Par conséquent, on utilise généralement la technique d'élimination de Bucket, qui consiste à marginaliser les variables inutiles avant de faire les calculs impliqués par la règle de chaînage.
Dans le cas de l'exemple précédent :

$$\sum_{C,D,E} Pr[A, B, C, D, E, F, G, H]$$

$$=$$

$$\sum_{C,D,E} Pr[H|E].Pr[G|D,E].Pr[F|A].Pr[D|A,B].Pr[E|B,C].Pr[A].Pr[B].Pr[C]$$

$$=$$

$$Pr[G|D,E].Pr[H|E].Pr[F|A]Pr[A].Pr[B].\sum_C \sum_D \sum_E Pr[D|A,B].Pr[E|B,C].Pr[C]$$

$$=$$

$$Pr[G|D,E].Pr[H|E].Pr[F|A]Pr[A].Pr[B].\sum_D Pr[D|A,B].\sum_C Pr[C].\sum_E Pr[E|B,C]$$

$$=$$

$$Pr[G|D,E].Pr[H|E].Pr[F|A]Pr[A].Pr[B].1.1.1$$

$$=$$

$$Pr[G|D,E].Pr[H|E].Pr[F|A]Pr[A].Pr[B]$$

Cette technique réduit considérablement les calculs et la taille des tables de probabilités à multiplier. Etant donné le grand nombre de variables gardées et la forme du réseau, l'exemple ci-dessus n'est pas particulièrement probant mais, en général, il a été démontré qu'employer cette technique était beaucoup plus intéressant qu'utiliser la règle de chaînage en entier avant de marginaliser.

Avec ces différentes méthodes de calcul, il est possible de mettre à jour les probabilités de certains noeuds après l'arrivée d'une ou plusieurs évidences. Il reste alors à définir les algorithmes permettant de mettre à jour de manière performante toutes les probabilités d'un réseau suite à l'arrivée d'évidence.
Il existe deux principaux algorithmes de propagation, chacun propose une information résultante différente : l'agorithme de *somme-propagation* et l'agorithme de *maximum-propagation*. Les deux sections suivantes ont pour but de décrire ces deux algorithmes.

3.2 Somme-propagation

L'algorithme de somme-propagation a pour but de mettre à jour toutes les probabilités a priori suite à l'évidence ajoutée, selon la technique de mise à jour expliquée précedemment, qui utilise entre autres l'opérateur de somme et lui donne son nom. Il existe en réalité plusieurs algorithmes de somme-propagation dont les principaux sont ceux de Pearl ([17]) et de Jensen et Olesen ([14]).

L'agorithme de Pearl va mettre à jour le réseau bayésien en effectuant un calcul local pour chaque noeud du réseau. Il utilise, contrairement à l'algorithme de Jensen, la forme originelle du graphe formé par le réseau bayésien mais ne fonctionne cependant que lorsque ce graphe a une structure d'arbre. Pour rappel, un arbre est un graphe connexe (pour toute paire de sommets (a, b) il existe une chaîne entre les sommets a et b) et acyclique.
L'algorithme de Jensen, quant à lui, utilise une structure intermédiaire et lui applique une forme simplifiée de l'algorithme de Pearl.

3.2.1 Algorithme de Pearl

Dans l'algorithme de Pearl, chaque noeud communique à ses noeuds voisins l'information qu'il a collectionnée, jusqu'à ce que chaque noeud puisse mettre à jour sa probabilité marginale sur base de toute l'information reçue par le réseau. Cette propagation est réalisée grâce au passage de messages entre noeuds voisins par le biais des arcs qui les relient.

Le but est que chaque noeud enregistre toute l'information. Il est naturel de considérer qu'à chaque fois, deux messages au moins seront envoyés. En effet, pour deux noeuds X et Y, un message doit passer de X à Y. Y connaîtra alors l'information sur X et X devra en tenir compte.

Puisque le réseau a une structure d'arbre, il peut être divisé autour du noeud X avec, d'une part, les parents de X et d'autre part, les enfants X. Les règles que l'agorithme va ensuite appliquer sont :

- X a besoin de l'information de ses voisins pour calculer $Pr[X|\epsilon]$ où ϵ est l'information externe changeant X.
- X a besoin de l'information de tous ses enfants pour calculer $\lambda(x)$ qui est la représentation de l'information venant des noeuds enfants de X. $\lambda(x)$ est une vraisemblance.
- X a besoin de l'information de tous ses parents pour calculer $\pi(x)$ qui est la représentation de l'information venant des noeuds parents de X. $\pi(x)$ est une vraisemblance.
- Le noeud Y qui veut envoyer un λ-message $\lambda_Y(X)$ à son parent X a besoin des λ-messages venant de ses propres enfants et des π-messages venant de ses parents, excepté X.
- Le noeud U qui veut envoyer un π-message $\pi_Y(U)$ à son enfant X a besoin des π-messages de ses propres parents et des λ-messages venant de ses enfants, excepté X.

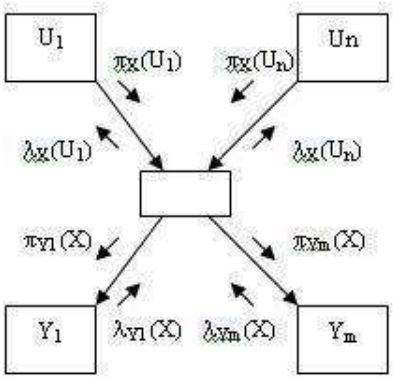


FIG. 3.6 – Messages liés à X dans une propagation de type Pearl

Il en résulte que chaque noeud X peut être dans cinq différents états :

- Etat 1 : attente de messages. Si n_X est le nombre de voisins de X, tant qu'il n'a pas reçu au moins $n_X - 1$ messages, il ne peut rien faire.
- Etat 2 : calcul des messages collectionnés. X a reçu $n_X - 1$ messages, il peut donc calculer le message de son seul voisin Y qui ne lui a rien envoyé. X est toujours dans une étape de

collecte.
- Etat 3 : attente de réponses. X attend le message de son dernier voisin.
- Etat 4 : calcul des messages par distribution de probabilité. X a reçu le dernier message. Il peut maintenant calculer $\lambda(x)$, $\pi(x)$ et $Pr[x|S]$. Il peut aussi distribuer les $n_X - 1$ messages qu'il n'a pas encore lui-même envoyé.
- Etat 5, final : l'algorithme est fini pour le noeud X.

En fait, l'ordre des noeuds mis à jour n'a pas d'importance.

Formellement, supposons un réseau bayésien dont le graphe est $G = (U, E)$, l'algorithme suivant permet de calculer $Pr[X|\epsilon]$ pour chaque noeud X de ce réseau :

Algorithm 1 Algorithme de somme-propagation selon [1]

$\forall$ noeud X, initialiser X dans l'état 1 ;
$U_{collection} \leftarrow U$;
while $U_{collection} \neq \emptyset$ **do**
  $\exists X \in U_{collection}$ tel que X peut aller dans l'état 2 ;
  $U_{collection} \leftarrow U_{collection} \backslash \{X\}$;
  X entre dans l'état 2, envoie à Y son message et entre ensuite dans l'état 3 ;
end while
Le dernier Y peut entrer dans l'état 4. $U_{distrib} = \{Y\}$;
while $U_{distrib} \neq \emptyset$ **do**
  $\exists Y \in U_{distrib}$ et $U_{distrib} = U_{distrib} \backslash \{Y\}$;
  Y entre dans l'état 4, envoie à tous ses voisins et entre dans l'état 5 ;
  $U_{distrib} = U_{distrib} \cup \theta_Y$ où θ_Y représente le voisinage de Y ;
end while

L'idée de cet algorithme est que, pour calculer $Pr[X|\epsilon]$, l'arbre est divisé en trois parties : les noeuds pour lesquels le chemin vers X passe par un parent de X, les noeuds pour lesquels le chemin vers X passe par un enfant de X et enfin, X lui-même.

Par conséquent, ϵ peut être divisé en trois sous-ensembles d'informations : ϵ_X^+ (l'information liée aux noeuds du premier groupe) ; ϵ_X^- (l'information liée aux noeuds du second groupe) et ϵ_X^0 (l'information liée au noeud du dernier groupe).
Si X n'est ni une racine ni une feuille, X a au moins un parent et un enfant. Dans ce cas, une séparation peut être effectuée.

En appliquant la règle de Bayes, cette séparation donne, $\forall x \in D(X)$, où $D(X)$ est l'ensemble des états possibles pour X :

$$\begin{aligned} Pr[X = x|\epsilon] &= Pr[X = x|\epsilon_X^-, \epsilon_X^+] \\ &= Pr[\epsilon_X^-|X = x, \epsilon_X^+].Pr[X = x|\epsilon_X^+] \end{aligned}$$

Puisqu'il y a d-séparation entre un noeud venant de ϵ_X^+ et un noeud venant de ϵ_X^- si X, cette assertion s'écrit comme suit :

$$\begin{aligned} Pr[X = x|\epsilon] &= Pr[\epsilon_X^-|X = x].Pr[X = x|\epsilon_X^+] \\ &= \lambda(x).\pi(x) \end{aligned}$$

Le dernier point à éclaircir concerne la définition formelle des λ-messages et des π-messages. Dans [1], en utilisant les résultats précédents, la définition est la suivante :

- Si X, Y, U, V, Z sont des noeuds du réseau bayésien ;
- Si $D(W)$ est le domaine d'un noeud (ou d'un ensemble de noeuds) W, c'est à dire l'ensemble des états possibles pour un noeud W (ou l'ensemble des états joints possibles pour tous les noeuds d'un ensemble W) ;
- Si $y \in D(Y)$, $u \in D(U)$, $v \in D(V)$, $z \in D(Z)$;
- Si Ξ_W représente l'ensemble des enfants d'un noeud W et Π_X, l'ensemble des parents d'un noeud W ;
- Si ϵ est l'information externe modifiant le réseau ;

Alors, $\forall x \in D(X)$

- $Pr[X = x|\epsilon] = \lambda(x).\pi(x)$
- $\lambda(x) = \prod_{Y \in \Xi_X} \lambda_Y(x)$
- $\pi(x) = \sum_{u \in D(\Pi_X)} Pr[X = x|\Pi_X = u].\prod_{U_i \in \Pi_X} \pi_X(U_i = u_i)$
- $\lambda_Y(x) = \sum_{y \in D(Y)} \lambda(y).\sum_{z \in D(\Pi_Y \setminus \{X\})} Pr[Y = y|X = x, Z = z].\prod_{Z_l \in \Pi_Y \setminus \{X\}} \pi_Y(Z_l = z_l)$
- $\forall u \in D(U),\ \pi_X(u) = \pi(u).\prod_{V \in \Xi_U \setminus \{X\}} \lambda_V(U = u)$

, avec $Y \in \Xi_X$ et $U \in \Pi_X$

Dans le cas des racines et des feuilles, les formules sont plus simples :

- Si X est une racine, alors $\epsilon_X^+ = \emptyset$ et $\pi(x) = Pr[X = x|\epsilon_X^+] = Pr[X = x]$. X n'envoie pas de λ-messages et, pour envoyer un π-message a un de ses enfants, il a besoin des λ-messages venant de ses autres enfants.
- Si X est une feuille, alors $\epsilon_X^- = \emptyset$ et $\lambda(x) = Pr[\epsilon_X^-|X = x]$ est constante. X n'envoie pas de π-messages et, pour envoyer un λ-message à l'un de ses parents, il a besoin des π-messages venant de ses autres parents.

3.2.2 Algorithme de Jensen et Olesen

L'algorithme précédent ne fonctionne que sur des réseaux bayésiens ayant une structure d'arbre. Pour éviter cette contrainte, mais aussi dans un soucis de simplicité, on recourt à la méthode de somme-propagation qui exploite une structure intermédiaire appellée *arbre de jonction*. L'algorithme de Jensen et Olesen, l'un des plus utilisés aujourd'hui, fait appel à cette méthode.

Arbre de jonction

Pour arriver à une définition formelle d'un arbre de jonction, il est nécessaire de définir les notions de *graphe complet* et de *clique*.

- *Graphe complet* : Un graphe complet est un ensemble de noeuds tous connectés entre eux.

- *Clique* : Une clique est un ensemble maximal de noeuds tous connectés entre eux.

- *Arbre de jonction* : Supposons que $G = (V, e)$ est un graphe non orienté et que C est l'ensemble des cliques de G. Un *arbre de jonction* est une arbre dont les noeuds sont éléments de C et dont les arcs vérifient la propriété d'intersection courante : si C_1 et C_2 sont des cliques, alors, pour tout chemin entre C_1 et C_2, chaque clique du chemin contient $C_1 \cap C_2$.

Envisager l'exploitation des arbres de jonction exige la définition d'une méthode permettantde générer un arbre de jonction à partir d'un réseau bayésien. Cette génération est faite en trois étapes : construire le *graphe moral*, construire le *graphe triangulé* et construire l'*arbre de jonction* à partir des cliques du graphe triangulé.

La première étape, qui consiste à créer la version morale du graphe d'origine, va d'une part transformer tous les arcs orientés en arêtes non orientées et d'autre part, créer de nouvelles arêtes entre les parents d'un même noeud. Dans [1], la transformation est effectuée comme suit.

Si $G = (V, E)$ est un graphe orienté, le graphe moral $G^m = (V, E^m)$ de G est construit en appliquant :

$$\begin{gathered} \textit{Il y a une arête entre } u \textit{ et } v \in E^m \\ \Longleftrightarrow \\ \big((u \leftarrow v) \in E\big) \bigvee \big((v \leftarrow u) \in E\big) \bigvee \big(\exists w \in V : u, v \subset \Pi_w\big) \end{gathered}$$

On remarque que la version morale du graphe crée un graphe complet pour chaque noeud et ses parents (figure 3.7). L'exemple en images est présenté dans [2].

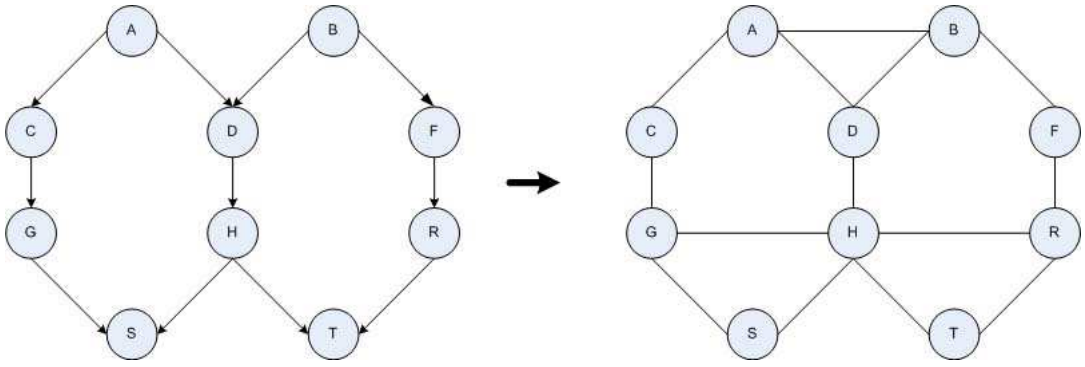


FIG. 3.7 – Moralisation : à gauche, un réseau bayésien, à droite la version moralisée de ce graphe

A partir du graphe moral, il faut à présent fusionner toutes les variables d'une même clique en une unité de plus haut niveau. Cependant, l'opération est complexe, dans le mesure où un noeud peut apparaître dans plusieurs cliques différentes. Par conséquent, une relation particulière entre des cliques partageant un ou plusieurs noeuds doit être créée. Cette relation, déterminée par les noeuds communs à une paire de cliques, forme une sorte de séparateurs entre les deux cliques de la paire.
Le processus qui permet de construire un graphe de jonction à partir d'un graphe moral peut être défini de la manière suivante.

Si $G = (V, E)$ est une graphe non orienté et C, l'ensemble de ses cliques, le graphe de jonction (C, E_C) est construit en appliquant :

$$(C_1, C_2) \in E_C \Longleftrightarrow C_1 \cap C_2 \neq \emptyset$$

Et $S_{1,2} = C_1 \cap C_2$ est le separateur des cliques C_1 et C_2.

Les graphes de jonction posent un problème. Ils n'ont pas toujours une structure d'arbre, or ce deuxième algorithme de somme-propagation ne peut travailler que sur des arbres de jonction. Par conséquent, avant d'appliquer le procédé précédent qui constitue la troisième étape de la construction d'un arbre de jonction, il faut passer par un graphe moral intermédiaire : le graphe moral triangulé.

Un graphe non orienté $G = (V, E)$ est dit triangulé si et seulement si tout cycle de longueur supérieure à quatre possède une corde, c'est à dire contient un couple de noeuds qui ne sont pas connectés dans ce cycle mais qui sont connectés dans E. Dans [2], le processus de triangulation est défini comme suit :

Si $G = (V, E)$ est un graphe non orienté où $V = \{X_1, ..., X_n\}$ et si σ est une permutation de $\{1, ...n\}$, on considère qu'un noeud X_i est éliminé de G quand des arcs entre tous ses voisins sont ajoutés à E (un graphe complet est ainsi construit) et quand, ensuite, X_i et ses arcs adjacents sont supprimés de G.
La triangulation de G est effectuée :

- En éliminant successivement les noeuds $X_{\sigma(1)}, ..., X_{\sigma(n)}$;
- En construisant E_T, l'ensemble des arcs successivement ajoutés (lors de l'étape précédente) ;
- En construisant le graphe triangulé $G_T = (V, E \cup E_T)$;

La figure 3.8 montre la triangulation du graphe moralisé de la figure 3.7. Quant à la figure 3.9, elle transforme ce graphe triangulé, en arbre de jonction.

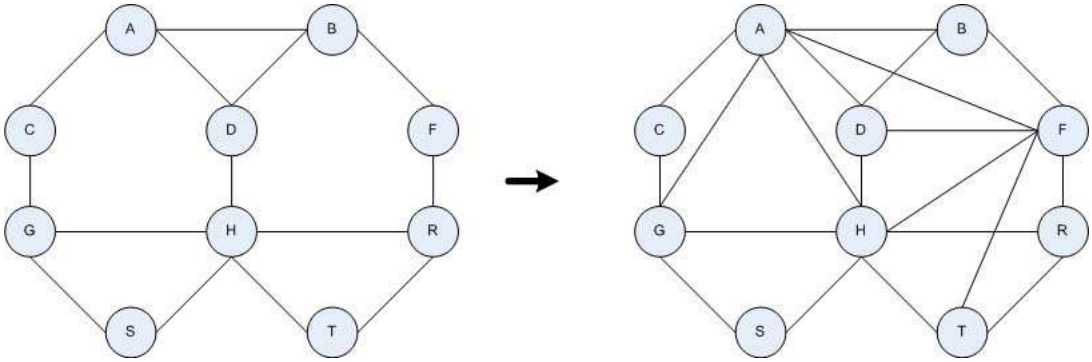


FIG. 3.8 – Triangulation : à gauche, un graphe moral, à droite la version triangulée de ce graphe

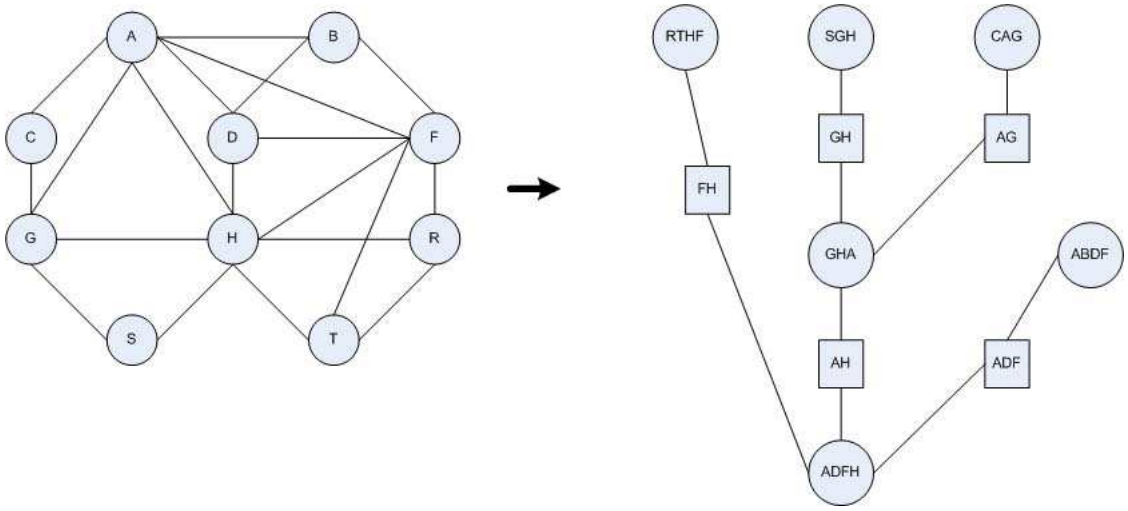


FIG. 3.9 – Création d'un arbre de jonction à partir d'un graphe triangulé

A partir des procédés expliqués précédemment, il est à présent possible de définir la méthode de construction d'un arbre de jonction. L'arbre de jonction d'un graphe est obtenu en construisant d'abord le graphe moral du graphe d'origine, en appliquant ensuite le processus de triangulation sur le graphe moral et en générant finalement l'arbre de jonction sur base du graphe moral triangulé.

Somme-propagation sur l'arbre de jonction

Une fois que l'arbre de jonction d'un réseau bayésien est construit, le deuxième algorithme de somme-propagation peut être appliqué.

Si $G(V, E)$ est un arbre de jonction, un pontentiel $\psi_S(V)$ est une fonction de V qui dépend uniquement des noeuds de S.
La propagation dans un arbre de jonction utilise la notion de potentiel et l'assertion suivante :

$$Pr[V] = \frac{\prod_{C \in \mathcal{C}} \psi_C(V)}{\prod_{S \in \mathcal{S}} \psi_S(V)}$$

, où $\mathcal{C}$ est l'ensemble des cliques de G et $\mathcal{S}$ est l'ensemble des séparateurs de G.

L'objectif de cet algorithme est qu'au terme de tous les calculs, chaque potentiel de clique est mis à jour avec la probabilité jointe des variables de la clique. La marginalisation des différents potentiels de cliques pour une variable (les cliques contenant cette variable) donne le même résultat. En d'autres mots, la probabilité marginale d'une variable peut être calculée en marginalisant les potentiels des cliques contenant cette variable.

Par conséquent, l'algorithme de propagation fonctionne de la manière suivante :

Initialisation :
$\forall C_i \in \mathcal{C}$, s'ils sont énumérés dans l'ordre de la propriété d'intersection courante, alors, $\forall S \in \mathcal{S}$,

$$\psi_S^0 = 1$$

$$\psi_{C_i}^0 = \prod_{X \in C_i, X \notin C_j, j<i} Pr[X|\Pi_X]$$

Collection :
Si C_i est une clique pour laquelle toutes les cliques adjacentes C_k à l'exception d'une C_j ont calculé leur $\psi_{C_k}^1$, alors les potentiels du séparateur $S_{i,j}$ et de la clique S_j sont mis à jour comme suit,

$$\psi_{S_{i,j}}^1 = \sum_{C_i \setminus S_{i,j}} \psi_{C_i}^1$$

$$\psi_{C_j}^1 = \psi_{C_j}^0 . \frac{\psi_{S_{i,j}}^1}{\psi_{S_{i,j}}^0}$$

Cette étape est répétée tant qu'une telle clique existe encore.

Distribution :

Le dernier noeud de l'étape précédente (la racine) envoie l'information changée à tous ses voisins (qui feront ensuite la même chose), en utilisant les formules suivantes,

$$\psi^2_{S_{i,j}} = \sum_{C_i \setminus S_{i,j}} \psi^2_{C_i}$$

$$\psi^2_{C_j} = \psi^1_{C_j} \cdot \frac{\psi^2_{S_{i,j}}}{\psi^2_{S_{i,j}}}$$

Dans [19], cet algorithme de propagation est légèrement modifié afin de le rendre plus efficace. La version modifiée de l'algorithme utilise uniquement les potentiels nécessaires au message produit, plutôt que d'utiliser les potentiels de toutes les cliques de l'arbre de jonction. Pour déterminer quels potentiels sont pertinents, cette version de l'algorithme utilise au mieux les propriétés de d-séparation du réseau bayésien de départ.

Exemple

Un exemple complet sur un réseau de quelques noeuds, permet comprendre la méthode plus aisément.

Reprenons le domaine analysé dans le chapitre exemple, soit celui des étudiants à l'Université d'Aalborg (fiugre 3.10), exprimant le fait que pour un étudiant Danois, le fait qu'il étudie ou non à l'Université d'Aalborg, implique à une certaine probabilité de participer à des projets et à une certaine probabilité, d'utiliser le vélo comme moyen de transport pour se rendre sur le campus. Ces deux conséquences peuvent amener l'étudiant à un degré de fatigue plus ou moins important. Enfin, participer à des projets permet à l'étudiant d'obtenir une expérience pratique dans son domaine d'étude.

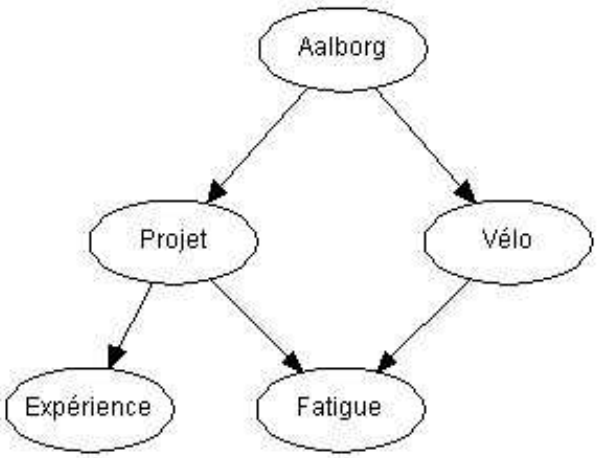


FIG. 3.10 – Réseau bayésien pris en exemple

En 3.11, les tables de probabilités de tous les noeuds du réseau sont données.

Aalborg	**Pr[Aalborg]**
oui	0.6
non	0.4

Pr[Projet | Aalborg]	*Aalborg*	
Projet	**oui**	**non**
oui	0.9	0.6
non	0.1	0.4

Pr[Vélo | Aalborg]	*Aalborg*	
Vélo	**oui**	**non**
oui	0.4	0.35
non	0.6	0.65

Pr[Expérience | Projet]	*Projet*	
Expérience	**oui**	**non**
oui	0.6	0.2
non	0.4	0.8

Pr[Fatigue | Vélo, Projet]	*Vélo* = **oui**		*Vélo* = **non**	
	Projet		*Projet*	
Fatigue	**oui**	**non**	**oui**	**non**
oui	0.9	0.15	0.7	0.95
non	0.1	0.85	0.3	0.05

FIG. 3.11 – Tables de probabilités du réseau 3.10

Après transformation, l'arbre de jonction de la figure 3.12 est obtenu.

FIG. 3.12 – Arbre de jonction du réseau bayésien en 3.10
(A = Aalborg ; P = Projet ; V = Vélo ; F = Fatigue ; E = Expérience)

Initialisation Les potentiels à définir pour cet arbre de jonction sont calculés lors de l'étape d'initialisation comme suit :

$$\psi_{PAV} = Pr[Projet|Aalborg].Pr[Vélo|Aalborg].Pr[Aalborg]$$

$$\psi_{PVF} = Pr[Fatigue|Projet, Vélo]$$

$$\psi_{EP} = Pr[Expérience|Projet]$$

Les tables obtenues en appliquant ces calculs sont modifiées afin de prendre en compte l'évidence qui ici est définie par le triplet *(Fatigue = oui, Vélo = oui, Expérience = oui)*. Toutes les parties du tableau qui ne respectent pas ce triplet, sont mises à zéro.

Concernant les séparateurs, ils sont tous initialisés à un. On obtient les tableaux 3.13, 3.14 et 3.15.

	Aalborg = **oui**		*Aalborg* = **non**	
	Projet		*Projet*	
Vélo	**oui**	**non**	**oui**	**non**
oui	0.216	0.024	0.084	0.056
non	0.0	0.0	0.0	0.0

FIG. 3.13 – $\psi_{PAV} = Pr[Projet|Aalborg].Pr[Vélo|Aalborg].Pr[Aalborg]$

ψ_{PV}	*Projet*	
Vélo	**oui**	**non**
oui	1.0	1.0
non	1.0	1.0

Projet	ψ_P
oui	1.0
non	1.0

FIG. 3.14 – Potentiels des séparateurs PV et P

ψ_{PVF}	*Projet* = **oui**		*Projet* = **non**	
	Vélo		*Vélo*	
Fatigue	**oui**	**non**	**oui**	**non**
oui	0.9	0.15	0.7	0.95
non	0.0	0.0	0.0	0.0

ψ_{EP}	*Projet*	
Expérience	**oui**	**non**
oui	0.6	0.2
non	0.0	0.0

FIG. 3.15 – $\psi_{PVF} = Pr[Fatigue|Projet, Vélo]$ et $\psi_{EP} = Pr[Expérience|Projet]$

Collecte L'objectif est à présent de faire circuler l'information d'une clique C_1 vers une clique C_2, par le biais de leur séparateur, S_0, en modifiant leurs potentiels ψ. Pour effectuer cette opération, rappellons qu'il faut :

- Obtenir un nouveau potentiel pour S_0 en marginalisant les variables de C_1 qui ne sont pas dans S_0 : $\sum_{C1/S0} \psi_{C1}$
- Calculer le nouveau potentiel pour C_2 : $\psi'_{C_2} = \psi_{C_2}.\frac{\psi S_0{}'}{\psi S_0}$

Il est dès lors possible de calculer, par exemple, la collecte d'information effectuée par la clique PAV à partir des deux autres cliques.

Pour effectuer cette collecte, dans un premier temps, on calcule la nouvelle valeur du séparateur P à partir de la clique EP. On obtient le tableau 3.16.

Projet	ψ'_P
oui	0.6
non	0.2

FIG. 3.16 – $\psi'_P = \sum_{EP/\psi_P} \psi_{EP}$

Cette nouvelle valeur du séparateur permet de calculer les nouveaux potentiels de la clique PVF, en 3.17.

	Projet = **oui**		*Projet* = **non**	
	Vélo		*Vélo*	
Fatigue	**oui**	**non**	**oui**	**non**
oui	0.54	0.09	0.14	0.19
non	0.0	0.0	0.0	0.0

FIG. 3.17 – $\psi'_{PVF} = \psi_{PVF}.\frac{\psi P'}{\psi P}$

La valeur du potentiel du séparateur PV peut alors être calculée et le résultat obtenu est le tableau 3.18.

	Projet	
Vélo	**oui**	**non**
oui	0.54	0.14
non	0.09	0.19

FIG. 3.18 – $\psi'_{PV} = \sum_{PVF/\psi_{PV}} \psi_{PVF}$

Enfin, les nouveaux potentiels de la clique PAV sont obtenus, en 3.19.

	Aalborg = **oui**		*Aalborg* = **non**	
	Projet		*Projet*	
Vélo	**oui**	**non**	**oui**	**non**
oui	0.117	0.003	0.045	0.008
non	0.0	0.0	0.0	0.0

FIG. 3.19 – $\psi'_{PAV} = \psi_{PAV} . \frac{\psi PV'}{\psi PV}$

A présent, supposons que la valeur que nous cherchons est $Pr[Aalborg = oui]$. Dans ce cas, il n'est pas nécessaire d'effectuer une étape de distribution. Il suffit d'utiliser le dernier tableau obtenu et considérer la probabilité recherchée rapport à la probabilité totale de la clique (normalisation). Il faut calculer $\sum_{P,V} \psi_{PAV} / \sum_{P,A,V} \psi_{PAV}$, ce qui vaut 0.6936.

3.3 Maximum-propagation

L'idée principale de la maximum-propagation est de trouver la configuration la plus probable des variables du réseau bayésien.

Le premier algorithme de maximum propagation a été développé par Dawid ([7]). Son principe est simple, il consiste d'abord à utiliser l'algorithme de Jensen et Olesen pour propager l'information mais en utilisant non pas l'opérateur de somme mais celui de maximum. Une fois que cette propagation utilisant l'opérateur maximum a été effectuée, Dawid a défini un second algorithme dont le but est de trouver la configuration la plus probable du réseau, sur base des résultats de la maximum-propagation.

On considère un réseau composé des variables A, B, et C, avec un arc allant de A à B et un arc de B à C et on désire obtenir la configuration (A, B, C) avec la probabilité maximum.
La première étape consiste à trouver la probabilité α de la configuration la plus probable, ce que fait l'algorithme de Jensen et Olesen où les opérateurs somme sont remplacé par les opérateurs maximum. α est alors le plus grand nombre dans la table de probabilité jointe Pr[A, B, C].

$$
\begin{aligned}
\alpha &= \max_{A,B,C} Pr[A,B,C] \\
&= \max_{A,B,C} Pr[A].Pr[B|A].Pr[C|B] \\
&= \max_{A} \left(\max_{B} \left(\max_{C} (Pr[A].Pr[B|A].Pr[C|B]) \right) \right) \\
&= \max_{A} \left(\max_{B} \left(Pr[A].Pr[B|A].\max_{C} (Pr[C|B]) \right) \right) \\
&= \max_{A} \left(Pr[A].\max_{B} \left(Pr[B|A].\max_{C} (Pr[C|B]) \right) \right)
\end{aligned}
$$

Dans la technique précédente, la règle de chaînage et la loi de distributivité de l'opérateur "maximum" ont été utilisées pour obtenir le résultat.

Dès lors, pour obtenir la configuration maximum qui se cache derrière cette probabilité, il suffit de prendre la probabilité de la configuration la plus probable (α) et, avant chaque marginalisation, trouver l'état qui a la probabilité maximum.

En d'autres mots, pour retrouver la meilleure configuration, dont la valeur est α, on utilise le raisonnement suivant : puisque le maximum en A a été atteint pour $A = a_{max}$, regardons la colonne a_{max} dans le table de la probabilité $Pr[B|A].max_C Pr[C|B]$ et déterminons l'état de B qui est maximum. La résultat est l'état b_{max}.
Sachant cela, utilisons la table $Pr[C|B]$ et regardons, dans la colonne b_{max}, l'état de C qui est le maximum. C'est l'état c_{max}. Par conséquent, la meilleure configuration est $a_{max}b_{max}c_{max}$.

L'algorithme de Dawid, quant à lui, fonctionne sur ce principe mais travaille directement sur l'arbre de jonction.

Chapitre 4

Apprentissage des paramètres d'un réseau bayésien *A. Bonbled, S. Munezero*

4.1 Objectif

Cette section a pour but de décrire l'algorithme le plus populaire d'apprentissage de paramètres, soit l'apprentissage des probabilités (conditionelles ou non) d'un évènement.
Il s'agit de l'algorithme *Expectation Maximisation* (EM), qui gère en outre le problème de valeurs manquantes dans les ensembles de données et qui est utilisé tout au long de ce projet. Cet algorithme a été proposé dans [8] et est appliqué aux réseaux bayésiens dans [18]. Il est présenté de manière simplifiée dans [16] et dans [1].

4.2 Ensemble de données complet

Quand toutes les variables sont observées et qu'aucune valeur ne manque, une méthode plus simple est utilisée, il s'agit de la méthode d'*estimation de la vraisemblance maximum* (maximum likelihood estimation). Le principe est d'estimer des paramètres en comptant, dans l'ensemble de données, le nombre de cas pour lesquels l'évènement analysé est observé. L'idée est que, de cette façon, la fréquence de l'évènement peut être obtenue en divisant ce nombre de cas positifs (respectant l'évènement) par le nombre total de cas.

Formellement, si i représente le numéro d'une variable ; k, le numéro de son état et j, le numéro de la configuration de ses parents, $N_{i,j,k}$ est le nombre de cas dans l'ensemble de données pour lesquels la variable X_i est dans l'état x_k et ses parents sont dans l'état x_j. $\hat{\theta}_{i,j,k}$ est la fréquence de cet évènement.

L'estimation de vraisemblance maximum est alors :

$$\boxed{\hat{\theta}_{i,j,k} = \frac{N_{i,j,k}}{\sum_{k=1}^{r_i} N_{i,j,k}}}$$

En réalité, il faudrait respectivement diviser le numérateur et le dénominateur par le nombre total de cas dans l'ensemble de données, mais les deux diviseurs s'annulent.

4.3 Ensemble de données incomplet

Dans de grands ensembles de données, les données peuvent parfois être manquantes pour certains cas, soit à cause du support physique qui rencontre des problèmes, soit parce que certaines variables n'ont pas été observées pour diverses raisons. C'est ici que l'algorithme EM est le plus utile. Selon [1], il peut être défini comme suit.

L'algorithme spécifie en premier lieu des probabilités initiales choisies au hasard pour les paramètres ayant des valeurs manquantes. En second lieu, l'algorithme répète deux étapes différentes jusqu'à convergence des paramètres recherchés. Il s'agit des étapes d'*expectation* et de *maximisation*.

Supposons que $\mathcal{X} = \{\mathcal{X}^{(l)}\}_{l=1,...N}$ est l'ensemble de données et que $\theta^{(t)} = \{\theta^{(t)}_{i,j,k}\}$ sont les paramètres du réseau bayésien lors de l'itération t de l'algorithme.

Le but de l'étape d'**expectation** est d'estimer les nombres manquants $N_{i,j,k}$ à une étape, en utilisant la moyenne de ces nombres donnés par les paramètres obtenus à l'étape précédente et les données observées. Sur base de ces nombres, l'étape de maximisation est alors appliquée. En général, l'étape d'expectation calcule tous les comptages de configurations nécessaires à l'étape suivante (même si la donnée est observée).

$$E_{\theta^{(t)}}[N(X_i = k, pa(X_i) = j|\mathcal{X})] = \sum_{l=1}^{N} Pr[X_i = k, pa(X_i) = j|\mathcal{X}_{(l)}, \theta^{(t)}]$$

Maintenant que les nombres $N_{i,j,k}$ ont été calculés, l'étape de **maximisation** peut obtenir les paramètres $\theta^{(t+1)}_{i,j,k}$ pour l'itération suivante de l'algorithme, en utilisant l'estimation de vraisemblance maximum expliquée précédemment : $\theta^{(t+1)}_{i,j,k} = \dfrac{E_{\theta^{(t)}}[N(X_i = k, pa(X_i) = j|\mathcal{X})]}{\sum_k E_{\theta^{(t)}}[N(X_i = k, pa(X_i) = j|\mathcal{X})]}$

Ce principe peut être appliqué pour la recherche de chaque paramètre du réseau, en changeant i, j et k.
L'algorithme EM peut être synthéthisé par le graphique suivante, où ϵ est le niveau de convergence souhaité :

Algorithm 2 Algorithme EM selon [1]

Initialiser $\theta^{(0)}$
t $\leftarrow 0$;
while $(\theta^{(t)} - \theta^{(t-1)}) \geq \epsilon$ **do**
  $t \leftarrow t + 1$
  $E_{\theta^{(t)}}[N(X_i = k, pa(X_i) = j|\mathcal{X})] \leftarrow \sum_{l=1}^{N} Pr[X_i = k, pa(X_i) = j|\mathcal{X}_{(l)}, \theta^{(t)}]$
  $\theta^{(t)}_{i,j,k} \leftarrow \dfrac{E_{\theta^{(t)}}[N(X_i = k, pa(X_i) = j|\mathcal{X})]}{\sum_k E_{\theta^{(t)}}[N(X_i = k, pa(X_i) = j|\mathcal{X})]}$
end while

4.4 Expérience

Il peut être intéressant d'utiliser des probabilités à priori lors de l'apprentissage de paramètres, afin de donner plus de poids au passé (aux anciens paramètres) et pour éviter qu'une information ancienne mais toujours utile soit perdue. Cette dernière situation survient lorsque, par exemple, il n'y a pas assez de cas dans l'ensemble de données sur base duquel on désire effectuer l'apprentissage. On risque de perdre une information solide au prix d'une information fragile, apprise sur des données trop peu nombreuses. Pour éviter cela, on utilise des tables de comptages d'"expériences" dont le but est de mesurer le poids de l'information apprise.

En donnant un comptage d'expérience de zéro aux paramètres avant d'effectuer un apprentissage, la technique d'apprentissage utilisée ne considérera pas de probabilités à priori et les nouveaux paramètres ne tiendront compte que de l'information présente dans l'ensemble de données à apprendre. Donner un comptage de zéro revient donc au même que de ne pas considérer d'expérience.

Par contre, si un comptage d'expérience de x est donné aux paramètres, les anciens paramètres auront un certain poids (x représente le "nombre de cas" sur base desquels l'information à priori s'appuie).

En pratique, les comptages d'expérience peuvent être utilisés pour considérer des cas "virtuels" dans des bases de données, ce qui est pris en charge par le calcul suivant.

- En premier lieu, les probabilités à priori sont multipliées par les comptages d'expérience liés (Exp), donnant comme résultat un comptage virtuel ($N^{(virtual)}$) :

$$N_{i,j,k}^{(virtual)} = Pr[X_i = x_k | pa(X_i) = x_j].Exp[X_i = x_k | pa(X_i) = x_j]$$

- En second lieu, la formule d'estimation de vraisemblance maximum est modifiée pour considérer ces cas virtuels :

$$\theta_{i,j,k} = \frac{N_{i,j,k} + N_{i,j,k}^{(virtual)}}{\sum_k (N_{i,j,k} + N_{i,j,k}^{(virtual)})}$$

Deuxième partie

Application à la reconnaissance de formules mathématiques

Chapitre 5

Application : interprétation de textes manuscrits *A. Bonbled, S. Munezero*

Les réseaux bayésiens sont beaucoup utilisés dans le domaine de l'intelligence artificelle. Ils permettent de modéliser l'incertitude - ce que peu d'autres modèles font - et peuvent être appris automatiquement par des algorithmes, tant au niveau de la structure qu'au niveau des paramètres. L'utilisation de ces mêmes réseaux et leur adaptation en fonction de l'arrivée de nouveaux évènements est en outre devenue très efficace.
Cependant, l'engouement de certains pour ces structures ne devrait pas faire croire que ces modèles peuvent résoudre efficacement n'importe quel problème.

Afin de déméler le vrai du faux et de voir, en pratique, comment les réseaux bayésiens peuvent apporter leur aide à des situations concrètes mais complexes, une application est proposée, dans le domaine de la reconnaissance de textes structurés à la main. Dans un premier temps, les spécifications de l'application, concrétisée par la mise au point d'un interpréteur de texte structuré, vont être précisées (hypothèses et objectifs). Dans un second temps, l'interpréteur lui-même sera développé au travers de trois phases : reconnaissance de texte structuré, correction des erreurs et analyse du comportement de l'utilisateur. Enfin, un logiciel implantant la solution proposée au cours du développement de cette application, sera présenté.

5.1 Spécifications

5.1.1 Hypothèses

La plus importante des hypothèses concerne le programme appellé classifieur, qui est supposé exister et dont les résultats seront utilisés par l'application ici définie. Le classifieur est un entité qui est capable de lire une phrase écrite à la main (ou une formule mathématique) et qui donne des informations à propos des lettres qui la composent. C'est sur base de ces informations que l'interpréteur sera apte à accomplir l'activité qui lui est propre.

L'hypothèse est faite que ce classifieur est capable de donner, pour chaque lettre de la formule mathématique lue, un ensemble de poids non négatifs pour les différents caractères possibles et autorisés. Ces poids sont appellés *vraisemblances* parce qu'ils donnent d'une certaine façon, pour chaque lettre, la probabilité que cette lettre soit un caractère bien précis. Certains conversions peuvent être nécessaires pour obtenir des probabilités à partir de ces poids.

De nombreux logiciels sont actuellement capables de reconnaître un caractère et de donner sa vraisemblance. Le problème majeur qui n'est pas toujours résolu est de déterminer comment ces probabilités peuvent être utilisées pour trouver la phrase admissible la plus probable.

Il est possible que le classifieur ne soit pas capable de définir une vraisemblance pour une ou plusieurs lettres. Cela peut arriver, par exemple, quand le texte est endommagé ou que certaines parties sont illisibles. Les valeurs manquantes doivent donc pouvoir aussi être gérées par l'interpréteur. Dans ce cas, une probabilité égale est donnée pour chaque caractère possible.

Le classifieur n'est pas forcé de donner une réelle distribution probabiliste pour chaque caractère, avec une somme de probabilités égale à un. Il peut s'agir de proportions ou de pourcentages. La principale hypothèse sur le classifieur est que les poids qu'il donne sont cohérents.

Une seconde hypothèse est faite sur l'utilisateur. L'utilisateur peut écrire des erreurs. Une "erreur" consiste en l'écriture d'une suite de caractères qui n'est pas autorisée par la grammaire mathématique choisie. L'activité d'interprétation doit considérer le fait que l'utilisateur peut commettre des erreurs et doit éventuellement pouvoir les corriger.
Par contre, lorsqu'il est nécessaire de déterminer si l'utilisateur commet constamment des erreurs ou lorsqu'il est demandé de définir une probabilité a priori sur la qualité de l'utilisateur, l'utilisateur sera considéré comme connaissant le domaine mathématique. L'utilisateur peut donc parfois oublier une parenthèse ou un opérateur, mais il est conscient d'une partie des règles, et n'écrira pas uniquement des erreurs.

5.1.2 Objectifs

Le terme "interpréteur" réfère à la fois à la méthode et au logiciel qui sont définis dans ce chapitre.
L'activité fondamentale de cet interpréteur est de travailler avec les vraisemblances fournies par un classifieur et de calculer sur cette base quelle phrase ou formule est à la fois celle qui respecte le plus ces vraisemblances et à la fois celle qui respecte les contraintes du langage mathématique utilisé. Cela signifie que l'interpréteur essaye toujours de respecter les règles de la grammaire et ne donne une formule fausse que s'il n'a vraiment aucun autre choix, c'est à dire si le classifieur est totalement sûr des caractères écrits par l'utilisateur et que conjointement, ces caractères sont faux.

La conséquence est que si l'utilisateur écrit un caractère faux mais qu'il y a une probabilité que ce caractère soit en réalité un autre ressemblant qui, lui, est correct, l'interpréteur choisira le caractère correct, et cela même si cette probabilité est très petite. Ceci implique que l'interpréteur risque de renvoyer une formule très différente de l'originale, puisqu'il préfère corriger et choisir des caractères que l'utilisateur n'a pas écrit. Cette conséquence doit être prise en compte.

L'interpréteur doit donc accomplir une autre activité et réagir à ce type de situation. Il doit découvrir les fautes de l'utilisateur et proposer des corrections. En outre, l'interpréteur peut aussi apprendre des erreurs de l'utilisateur dans certaines situations, afin de faire évoluer la grammaire qui est peut être dépassée. Plus généralement, apprendre les habitudes de l'utilisateur est une fonctionnalité qu'il est intéressant de donner à l'interpréteur.

Ce dernier a en outre à distinguer deux types réellement différents d'erreurs : les erreurs de précédence, qui concernent la succession de caractères et les erreurs liées au fait qu'une grammaire mathématique est non contextuelle. Cette notion est précisément définie et de nombreux exemples d'erreurs sont donnés dans la suite de ce chapitre.

Finalement, lorsque l'interpréteur hésite sur la pertinence d'une erreur et de sa correction, il est imaginable de rendre son activité interactive et par conséquent, d'interroger l'utilisateur. Dans le même contexte, on envisage que l'interpréteur analyse le comportement de l'utilisateur, en plus d'en tenir compte. Il pourrait par exemple prendre en considération la qualité de l'utilisateur selon la fréquence des erreurs qu'il commet.

5.2 Reconnaissance de formules mathématiques écrites à la main

5.2.1 Objectif : Trouver la formule

Etant donné que le but de l'application est de parvenir à reconnaître des formules mathématiques manuscrites, l'objectif de cette première section est de trouver la formule la plus probable et la plus admissible.
Cela signifie d'une part, que la formule trouvée doit être la plus proche de ce qu'a écrit l'utilisateur et d'autre part, qu'elle est syntaxiquement correcte, eu égard à la grammaire mathématique utilisée.

Afin de décomposer le problème, dans un premier temps, l'hypothèse considère que l'utilisateur ne commet pas d'erreurs et que les résultats du classifieur sont totalement fiables.

En outre, on envisage de commencer avec une grammaire simple, non mathématique, en ignorant temporairement les problèmes non contextuels. Les problèmes ajoutés par la prise en compte d'une grammaire mathématique complète sont ensuite envisagés.

Définitions

Certains concepts doivent être auparavant précisés. Il s'agit notamment du concept de "*grammaire régulière*", de celui de "*grammaire hors contexte*", de "*formules mathématiques*" ou encore de "*la phrase la plus probable et la plus admissible*".

Selon la hiérarchie de Chomsky, les grammaires hors contexte et régulières peuvent être qualifiées de grammaires formelles. Dans [9], l'auteur indique que les grammaires formelles ont émergé des tentatives linguistiques qui cherchaient à décrire structurellement les langages naturels.

Ces grammaires, connues en tant que *grammaires structurelles de phrase*, sont établies en termes de règles qui font référence à des structures de haut niveau (telles qu'une phrase) pour classer hiérarchiquement les sous-structures (telles que les noms de phrase, les verbes dans une phrase) et finalement, les symboles terminaux (mots).

Par exemple, la phrase 'La femme frappe la balle' peut être générée avec une grammaire contenant ces règles :

$S \rightarrow NP\ VP$

$NP \rightarrow Det\ N$

$VP \rightarrow V\ N$

$Det \rightarrow la$

$N \rightarrow femme$

$N \rightarrow balle$

$V \rightarrow frappe$

En informatique, une grammaire régulière de droite est une grammaire formelle définie par le quadruplet (N, Σ, P, S). N est l'ensemble des caractères non terminaux ; Σ est l'alpha utilisé (l'ensemble des caractères terminauax) ; P est l'ensemble des règles de production et S est le symbole de départ. Toutes les règles de production dans P ressemblent à l'une des règles suivantes :

$A \to a$ - où A est un non terminal $\in N$ et a est un terminal $\in \Sigma$
$A \to aB$ - où A et B sont $\in N$ et $a \in \Sigma$
$A \to \epsilon$ - où $A \in N$

Dans une grammaire régulière de gauche, toutes les règles obéissent aux formes :

$A \to a$ - où A est un non terminal $\in N$ et a est un terminal $\in \Sigma$
$A \to Ba$ - où A et B sont $\in N$ et a est $\in \Sigma$
$A \to \epsilon$ - où A est $\in N$

Une grammaire régulière est soit une grammaire régulière de droite, soit une grammaire régulière de gauche.

Comme exemple d'une grammaire régulière de droite, prenons G avec $N = \{S, A\}$, $\Sigma = \{a, b, c\}$, P et avec, comme ensemble des règles de production, les règles suivantes :

$S \to aS$
$S \to bA$
$A \to \epsilon$
$A \to cA$

, où S est le symbole de départ.

Cette grammaire décrit le même langage que l'expression régulière $a_n b c_n$. N est l'ensemble des symboles non-terminaux, Σ est l'alphabet ou l'ensemble des symboles terminaux. Toute grammaire régulière est une grammaire hors contexte.

Toute grammaire hors contexte peut être réécrite dans une combinaison de règles de grammaires régulières de droite et de gauche. Ainsi, de telles grammaires peuvent exprimer tous les langages hors contexte. Des grammaires régulières qui utilisent soit les règles régulières de droite soit les règles régulières de gauche mais non les deux, peuvent uniquement exprimer un ensemble plus petit de langages, appelés langages réguliers.

Pour illustrer un exemple de langage hors contexte qui ne peut être exprimé par une grammaire régulière, prenons celui qui définit l'ensemble des chaînes de caractères de la forme $a_i b_i$.

Ce langage est généré par la grammaire G avec $N = \{S, A\}$, $\Sigma = \{a, b\}$, P avec les règles :

$S \rightarrow aA$
$A \rightarrow Sb$
$S \rightarrow \epsilon$

Cette grammaire a des règles régulières de droite et des règles régulières de gauche et n'est plus une grammaire régulière.

En résumé, une grammaire hors contexte est une grammaire formelle dans laquelle chaque règle de production est de la forme :

$V \rightarrow w$

, où V est un symbole non-terminal et w une chaîne de caractères constituée de terminaux et/ou de non-terminaux.

Le terme "*non contextuel*" vient du fait que le non-terminal V peut toujours être remplacé par w, sans prendre en compte le contexte dans lequel il apparaît. Un langage formel est hors contexte s'il y a une grammaire hors contexte qui le génère. BNF (Backus-Naur Form) est la notation la plus commune utilisée pour exprimer les grammaires hors contexte.
Tous les langages formels ne sont pas hors contexte, un contre-exemple est $a_n b_n c_n : n \geq 0$.

Une définition plus formelle d'une grammaire hors contexte G est : $G = (Vt, Vn, P, S)$ où

- V_t est un ensemble fini de terminaux
- V_n est un ensemble de non-terminaux
- P est un ensemble fini de règles de production
- S est un élément de V_n, non-terminal de départ distinct des autres.
- Les éléments de P ont la forme $V_t \rightarrow (V_t \cup V_n)^*$

Une autre notion à définir est celle de *phrase la plus probable et la plus admissible*.
Pourquoi probable ? Parce le classifieur donne un ensemble de probabilités pour chaque lettre d'une phrase et que la phrase la plus probable est la phrase construite avec les caractères qui mis ensemble, forment la configuration la plus probable (les probabilités qui maximisent la probabilité de toute la phrase).
Pourquoi admissible ? Une phrase est dite admissible quand elle respecte les règles de la grammaire qui a été formellement définie, qu'elle soit régulière ou non contextuelle.

Concernant les formules mathématiques, notons que ces dernières sont souvent exprimées à l'aide d'une grammaire non-contextuelle parce que certaines règles imposent qu'un caractère doit être contrebalancé par un second, quelle que soit sa position dans la formule. C'est le cas des parenthèses, des intégrales, des accolades, etc.

Ce type de contrainte est inexprimable par une grammaire régulière. Les fonctions sont un autre exemple de problèmes non-contextuels car le nombre de parenthèses doit être correct autant dans l'expression de droite (définition de la fonction) que dans l'expression de gauche (définition des arguments).

Cette interprétation amène à extraite deux types de règles d'une grammaire non-contextuelle : les *règles de précédence* et les *règles non-contextuelles*.
Les règles de précédence sont celles que nous pouvons exprimer grâce aux grammaires régulières. C'est tout simplement l'idée qu'un caractère doit obligatoirement être suivi par un caractère faisant partie d'une série bien spécifique et ne peut pas être suivi par certains autres caractères.
Les règles non-contextuelles sont celles citées précédemment (une parenthèse de gauche équilibrée par une de droite,...).

Modélisation du domaine

Pour traiter la problématique de l'interprétation des formules mathématiques manuscrites, la méthode suivie a consisté à commencer par une grammaire très simple, une grammaire régulière.

Concrètement, nous avons supposé avoir un ensemble de symboles terminaux et non-terminaux avec un alphabet tel que $\{a, b, c, \&\}$ où & est utilisé pour exprimer un caractère signifiant la fin d'une phrase.

Formellement,

$S \rightarrow aS$
$S \rightarrow bB$
$B \rightarrow cB$
$B \rightarrow \epsilon$

Cette grammaire simple permet de générer des phrases du type "$a_n b c_n$". Plusieurs modèles de réseaux bayésiens se référant à cette grammaire ont été envisagés, mais un seul a été finalement retenu. Il constitue l'objet de la section suivante.

Les probabilités conditionnelles sont définies grâce aux règles de grammaire et les évidences fournies par le classifieur sont introduites dans le réseau bayésien en tant que *vraisemblances*.
Les probabilités obtenues à partir de ces vraisemblances du classifieur (évidences faibles) sont contrebalancées par les possibilités offertes par la grammaire (probabilités à priori).
Ce choix de modélisation implique qu'une seule et même forme de réseau bayésien peut être utilisée pour chaque phrase écrite par l'utilisateur, étant donné la grammaire définie au préalable.

En premier lieu, des tests ont été effectués pour la grammaire régulière définie ci-dessus. C'est ce qu'aborde la section 5.2.2.
En second lieu, puisque les tests étaient concluants, le modèle a été adapté pour permettre la reconnaissance de véritables formules mathématiques, ce qu'aborde la section 5.2.3.

Les règles de précédence ont été aisément intégrées dans un réseau bayésien. Pour pouvoir intégrer les règles non contextuelles, le réseau a du être combiné avec un mécanisme extérieur.

Le réseau finalement obtenu accepte tous les caractères possibles définis par l'alphabet mathématique que nous avons choisi, à savoir { "(", ")", "{", "}", "a", "b", "+", "−", "∗", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ",", "&" }.

5.2.2 Grammaire régulière

Construction du réseau

Après avoir défini la grammaire régulière utilisée et ses règles de production (section précédente), un réseau bayésien qui s'y réfère a pu être construit. Concrètement, le réseau est composé d'une série de noeuds et chaque noeud représente un caractère dans la phrase. Le réseau de la figure 5.1 modélise une phrase de cinq caractères.

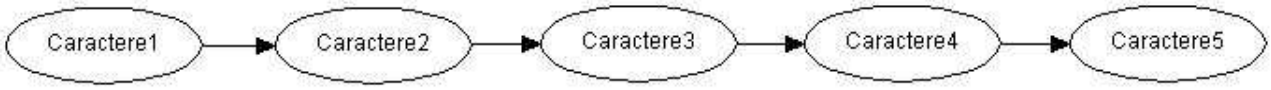


FIG. 5.1 – Réseau bayésien pour une grammaire régulière

Les probabilités conditionnelles du réseau sont ensuite extraites des règles de grammaire. Pour le premier caractère dans la phrase, une table typique de probabilité se trouve dans 5.2.

$Caractere_1$	$Pr[Caractere_1]$
a	0.5
b	0.5
c	0.0
&	0.0

FIG. 5.2 – $Pr[Caractere1]$

Le premier caractère est un caractère particulier car ses probabilités sont tirées du symbole de départ dans les règles de grammaire. Dans l'exemple, il y a les règles $S \rightarrow aS$ et $S \rightarrow bS$, ce qui veut dire que la phrase peut débuter soit par un "a" soit par un "b". C'est pour cette raison qu'il y a une probabilité d'occurrence égale autant pour "a" que pour "b" pour le premier caractère du réseau.
Pour tous les autres caractères, les tables de probabilités sont identiques et ont la forme reprise dans la figure 5.3.

Le raisonnement utilisé pour cette deuxième table est assez aisé. Dans les règles de la grammaire utilisée, des règles de précédence sont visibles et ce sont elles qui peuvent être converties en probabilités.

$Caractere_i$	$Caractere_{i-1}$ **a**	**b**	**c**	**&**
a	0.5	0.0	0.0	0.0
b	0.5	0.0	0.0	0.0
c	0.0	1.0	0.5	0.0
&	0.0	0.0	0.5	1.0

FIG. 5.3 – $Pr[Caractere_i|Caractere_{i-1}]$

Les règles de grammaire sont :

$S \rightarrow aS$
$S \rightarrow bB$
$B \rightarrow cB$
$B \rightarrow \epsilon$

Par conséquent, les règles de précédence extraites montrent que quand le caractère est un "a", il y a une probabilité d'avoir soit un "a" soit un "b" juste après. Quand le caractère est un "b", il ne peut être suivi que d'un "c". Quand le caractère est un "c", il y a une probabilité équivalente d'avoir soit un "c" ou un "&" juste après.

Ce sont les probabilités conditionnelles qui sont utilisées pour tout le reste de la section. Le réseau a été construit avec succès avec le logiciel *Hugin Expert A/S* et a été testé grâce à ce même logiciel. L'annexe 2 de la présente étude donne quelques informations au sujet de ce logiciel.

Propagation d'information

Le réseau construit, il est intéressant d'observer son évolution lorsque les vraisemblances du classifieur sont ajoutées aux noeuds. Sur chaque noeud de caractère, les vraisemblances ont été introduites comme évidences faibles.

Les deux différents algorithmes de propagation ont été testés (somme et maximum et des résultats différents ont été obtenus. C'est cette dernière différence qu'il faut expliquer, afin de déterminer l'algorithme le plus adapté. Le raisonnement est établi sur un réseau contenant quatre noeuds (A, B, C, D), dont le but est de calculer les probabilités des deux premiers noeuds, A et B, après avoir mis des vraisemblances en A.

Si on considère les deux premiers noeuds du réseau construit auparavant et que la vraisemblance (0.6 0.4 0 0) est ajoutée, le résultat devrait être cohérent. Comme "0.6" correspond à l'état "a" et "0.4" à l'état "b", les phrases possibles et admissibles devraient être "aa", "ab" et finalement "bc". Par conséquent, la phrase la plus probable et la plus admissible eu égard à la grammaire devrait être une de ces trois phrases.

Pour l'algorithme somme-propagation, le raisonnement est basé sur les probabilités a priori. La table des probabilités à priori du premier noeud ($Pr[A]$) est celle en 5.2.

Quant à la table des probabilités a priori du second noeud ($Pr[B]$), il faut la calculer à partir des probabilités conditionelles de la table 5.3. Le calcul effectué est :

$$\sum_A Pr[B|A].Pr[A]$$

$$=$$

$$\sum_A \left(\begin{array}{|c||cccc|} \hline \text{Pr[B|A]} & \multicolumn{4}{c|}{A} \\ B & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{\&} \\ \hline\hline \mathbf{a} & 0.5 & 0.0 & 0.0 & 0.0 \\ \mathbf{b} & 0.5 & 0.0 & 0.0 & 0.0 \\ \mathbf{c} & 0.0 & 1.0 & 0.5 & 0.0 \\ \mathbf{\&} & 0.0 & 0.0 & 0.5 & 1.0 \\ \hline \end{array} \times \begin{array}{|c||c|} \hline A & \\ \hline\hline \mathbf{a} & 0.5 \\ \mathbf{b} & 0.5 \\ \mathbf{c} & 0.0 \\ \mathbf{\&} & 0.0 \\ \hline \end{array} \right)$$

$$=$$

$$\sum_A \left(\begin{array}{|c||cccc|} \hline \text{Pr[B|A].Pr[A]} & \multicolumn{4}{c|}{A} \\ B & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{\&} \\ \hline\hline \mathbf{a} & 0.25 & 0.0 & 0.0 & 0.0 \\ \mathbf{b} & 0.25 & 0.0 & 0.0 & 0.0 \\ \mathbf{c} & 0.0 & 0.5 & 0.0 & 0.0 \\ \mathbf{\&} & 0.0 & 0.0 & 0.0 & 0.0 \\ \hline \end{array} \right)$$

$$=$$

B	Pr[B]
a	0.25
b	0.25
c	0.5
&	0.0

Les vraisemblances ($L[A]$) sont alors introduites dans le premier noeud. Par conséquent, les probabilités à priori du noeud A sont modifiées et ce sont les probabilités a posteriori qui sont obtenues, de la manière suivante :

$$Pr[A|L[A]]_{sum}$$

$$=$$

$$\frac{Pr[A].L[A]}{\sum_A Pr[A].L[A]}$$

$$=$$

$$\left(\left(\begin{array}{|c||c|}\hline A & \text{Pr[A]} \\ \hline \mathbf{a} & 0.5 \\ \mathbf{b} & 0.5 \\ \mathbf{c} & 0.0 \\ \& & 0.0 \\ \hline\end{array} \times \begin{array}{|c||c|}\hline A & \text{L[A]} \\ \hline \mathbf{a} & 0.6 \\ \mathbf{b} & 0.4 \\ \mathbf{c} & 0.0 \\ \& & 0.0 \\ \hline\end{array}\right) \div \sum_A \left(\begin{array}{|c||c|}\hline A & \text{Pr[A]} \\ \hline \mathbf{a} & 0.5 \\ \mathbf{b} & 0.5 \\ \mathbf{c} & 0.0 \\ \& & 0.0 \\ \hline\end{array} \times \begin{array}{|c||c|}\hline A & \text{L[A]} \\ \hline \mathbf{a} & 0.6 \\ \mathbf{b} & 0.4 \\ \mathbf{c} & 0.0 \\ \& & 0.0 \\ \hline\end{array}\right)\right)$$

$$=$$

$$\left(\begin{array}{|c||c|}\hline A & \text{Pr[A] . L[A]} \\ \hline \mathbf{a} & 0.3 \\ \mathbf{b} & 0.2 \\ \mathbf{c} & 0.0 \\ \& & 0.0 \\ \hline\end{array}\right) \div 0.5$$

$$=$$

A	Pr[A|L[A])
a	0.6
b	0.4
c	0.0
&	0.0

Enfin, il reste à mettre à jour les probabilités des parents ainsi que des enfants du premier noeud (A). Le premier noeud n'ayant pas de parent, seul son enfant doit être mis à jour (B) et ses probabilités a posteriori, calculées :

$$\sum_A Pr[B|A].Pr[A]$$

$$=$$

$$\sum_A \left(\begin{array}{|c||cccc|}\hline \text{Pr[B\|A]} & A & & & \\ B & \mathbf{a} & \mathbf{b} & \mathbf{c} & \& \\ \hline \mathbf{a} & 0.5 & 0.0 & 0.0 & 0.0 \\ \mathbf{b} & 0.5 & 0.0 & 0.0 & 0.0 \\ \mathbf{c} & 0.0 & 1.0 & 0.5 & 0.0 \\ \& & 0.0 & 0.0 & 0.5 & 1.0 \\ \hline\end{array} \times \begin{array}{|c||c|}\hline A & \text{Pr[A]} \\ \hline \mathbf{a} & 0.6 \\ \mathbf{b} & 0.4 \\ \mathbf{c} & 0.0 \\ \& & 0.0 \\ \hline\end{array}\right)$$

$$=$$

$$\sum_A \left(\begin{array}{|c||cccc|}\hline \text{Pr[B\|A].Pr[A]} & A & & & \\ B & \mathbf{a} & \mathbf{b} & \mathbf{c} & \& \\ \hline \mathbf{a} & 0.3 & 0.0 & 0.0 & 0.0 \\ \mathbf{b} & 0.3 & 0.0 & 0.0 & 0.0 \\ \mathbf{c} & 0.0 & 0.4 & 0.0 & 0.0 \\ \& & 0.0 & 0.0 & 0.0 & 0.0 \\ \hline\end{array}\right)$$

$$=$$

B	Pr[B]
a	0.3
b	0.3
c	0.4
&	0.0

Dans chacun des deux tableaux de probabilités à posteriori (pour A et pour B), l'état à la probabilité maximum est choisi. Le résultat "ac" est obtenu.

Pour l'algorithme de maximum propagation, le raisonnement se fonde également sur les probabilités a priori. L'unique différence avec l'algorithme de somme propagation est que pour calculer les nouvelles probabilités a priori, il utilise l'opérateur maximum et non celui de somme.

Les probabilités a posteriori du premier noeud sont d'abord calculées. Elles réclament le calcul la meilleure configuration de la table du second noeud ($Pr[B|A]$), qui est :

$\max_B \Bigg($

Pr[B|A]	A			
B	**a**	**b**	**c**	**&**
a	0.5	0.0	0.0	0.0
b	0.5	0.0	0.0	0.0
c	0.0	1.0	0.5	0.0
&	0.0	0.0	0.5	1.0

$\Bigg) =$

A	$\max_B Pr[B\|A]$
a	0.5
b	1.0
c	0.5
&	1.0

Les probabilités a posteriori pour le premier noeud sont alors être calculées avec l'algorithme de maximum propagation :

$$Pr[A|L[A]]_{max}$$

$$=$$

$$\frac{(Pr[A|L[A]]_{sum}).\max_B(Pr[B|A])}{\max_A((Pr[A|L[A]]_{sum}).\max_B(Pr[B|A]))}$$

$$=$$

$\Bigg($

A	$Pr[A\|L[A]]_{sum}$
a	0.6
b	0.4
c	0.0
&	0.0

$\times$

A	$\max_B(Pr[B\|A])$
a	0.5
b	1.0
c	0.5
&	1.0

$\Bigg)$

$\div$

$\max_A \Bigg($

A	$Pr[A\|L[A]]_{sum}$
a	0.6
b	0.4
c	0.0
&	0.0

$\times$

A	$\max_B(Pr[B\|A])$
a	0.5
b	1.0
c	0.5
&	1.0

$\Bigg)\Bigg)$

=

A	$(Pr[A\|L[A]]_{sum}).\max_B(Pr[B\|A])$
a	0.3
b	0.4
c	0.0
&	0.0

$\Big) \div 0.4$

=

A	$Pr[A\|L[A]_{max}$
a	0.75
b	1.0
c	0.0
&	0.0

Quant aux probabilités a posteriori pour le second noeud selon l'algorithme max-propagation, elles sont obtenues par le calcul :

$$Pr[B|L[A]]_{max}$$

=

$$\frac{\max_A(Pr[B|A]).\max_C(Pr[C|B]).Pr[B]}{\max_B(\max_A Pr[B|A].\max_C(Pr[C|B]).Pr[B])}$$

=

$\Big($ $\max_A \Big($

$Pr[B\|A].\max_C(Pr[C\|B])$	A			
B	**a**	**b**	**c**	**&**
a	0.25	0.0	0.0	0.0
b	0.5	0.0	0.0	0.0
c	0.0	0.5	0.25	0.0
&	0.0	0.0	0.5	1.0

$\times$

B	Pr[B]
a	0.3
b	0.3
c	0.4
&	0.0

$\Big)$

÷

$\max_B \Big(\Big($

$Pr[B\|A].\max_C(Pr[C\|B])$	A			
B	**a**	**b**	**c**	**&**
a	0.25	0.0	0.0	0.0
b	0.5	0.0	0.0	0.0
c	0.0	0.5	0.25	0.0
&	0.0	0.0	0.5	1.0

$\times$

B	Pr[B]
a	0.3
b	0.3
c	0.4
&	0.0

$\Big)\Big)\Big)$

=

$$\left(\left(\begin{array}{|c||l|}\hline B & \max_A(Pr[B|A]).\max_C(Pr[C|B]) \\ \hline\hline \mathbf{a} & 0.25 \\ \mathbf{b} & 0.5 \\ \mathbf{c} & 0.5 \\ \& & 1.0 \\ \hline\end{array} \times \begin{array}{|c||l|}\hline B & Pr[B] \\ \hline\hline \mathbf{a} & 0.3 \\ \mathbf{b} & 0.3 \\ \mathbf{c} & 0.4 \\ \& & 0.0 \\ \hline\end{array}\right)\right)$$

$$\div$$

$$\max_B\left(\left(\begin{array}{|c||l|}\hline B & \max_A(Pr[B|A]).\max_C(Pr[C|B]) \\ \hline\hline \mathbf{a} & 0.25 \\ \mathbf{b} & 0.5 \\ \mathbf{c} & 0.5 \\ \& & 1.0 \\ \hline\end{array} \times \begin{array}{|c||l|}\hline B & Pr[B] \\ \hline\hline \mathbf{a} & 0.3 \\ \mathbf{b} & 0.3 \\ \mathbf{c} & 0.4 \\ \& & 0.0 \\ \hline\end{array}\right)\right)$$

$$=$$

$$\left(\begin{array}{|c||l|}\hline B & \max_A(Pr[B|A]).\max_C(Pr[C|B]).Pr[B] \\ \hline\hline \mathbf{a} & 0.075 \\ \mathbf{b} & 0.15 \\ \mathbf{c} & 0.2 \\ \& & 0.0 \\ \hline\end{array}\right) \div 0.2$$

$$=$$

B	$Pr[B\|L[A]]_{max}$
a	0.375
b	0.75
c	1.0
&	0.0

A partir de ces derniers résultats, la configuration la plus probable selon l'algorithme de maximum propagation est trouvé, il s'agit de "*bc*".

En comparant les résultats de l'algorithme somme et de maximum propagation, et en les confrontant aux trois phrases possibles, il est frappant de voir que le résultat de la somme propagation, "*ac*", n'est pas une phrase possible. "*ac*" est un mauvais résultat. Par contre, le résultat "*bc*" de l'algorithme de maximum propagation apparaît dans l'ensemble de phrases admissibles défini auparavant. "*bc*" est un résultat correct.

En réalité, comme cela a été expliqué précédemment, l'algorithme somme propagation s'occupe des mises à jour locales. Prendre le maximum pour chaque caractère, sur base des chiffres fournis par la somme propagation, revient à calculer un maximum local pour chaque caractère, sans tenir compte de la configuration la plus probable.
L'algorithme maximum propagation, quant à lui, prend en compte la configuration la plus probable selon les noeuds qui suivent.

La localité de l'algorithme de somme propagation, et l'utilisation de l'opérateur somme masque le fait que certaines suites de caractères doivent être écartées, à cause de la grammaire.

Les calculs effectués dans l'exemple sont généralisés sur tous les autres noeuds dans les deux algorithmes.

La conclusion que l'algorithme maximum propagation est le plus approprié eu égard aux objectifs de la reconnaissance, s'impose dans la mesure où il considère toujours l'admissibilité donnée par les règles de grammaire.

L'algorithme maximum propagation peut aussi générer plusieurs phrases admissibles et probables dans les situations où elles ont été équivalentes compte tenu des vraisemblances. Ainsi, il est possible d'avoir comme résultat une ou plusieurs phrases les plus probables et admissibles, avec pour certains noeuds, plusieurs états avec une probabilité maximum. L'idée est alors de générer toutes les combinaisons de toutes les possibilités pour avoir toutes les phrases.

Apprentissage de paramètres

Le but de l'opération d'apprentissage de paramètres est d'appréhender les habitudes de l'utilisateur, à partir des phrases qu'il est accoutumé à écrire. Pour ce faire, le moyen classique est de simuler aléatoirement des cas d'introduction de phrases pour qu'elles soient enregistrés dans une base de données. Seul l'algorithme *Expectation Maximisation* a été utilisé pour cet apprentissage.

Puisque le principe est de conserver le même réseau que celui présenté précédemment, avec la même structure de tables de paramètres, l'apprentissage des paramètres est un peu particulier. Il doit permettre d'apprendre des règles de précédence de la forme "*tel caractère peut être suivi de tel autre et/ou tel autre*".

Pour chaque utilisation du réseau, et pour chaque phrase générée, deux fichiers vont être employés. Dans le premier fichier, seul le premier caractère de chaque phrase est enregistré. Dans le deuxième fichier, sont enregistrées toutes les combinaisons de caractères deux à deux selon l'ordre dans lequel ils sont suivis dans la phrase.

Si par exemple, les phrases introduites sont "$aabc\&$" et "$abcc\&$", alors, dans le premier fichier, les deux cas seront "a", "a" et dans le deuxième fichier, les cas seront "aa", "ab", "bc", "$c\&$", "ab", "bc", "cc", "$c\&$".

Il est maintenant possible d'analyser l'évolution de la grammaire selon les habitudes d'un utilisateur. Par exemple, l'utilisateur qui écrit toujours ses phrases avec "a" et "b" peut être considéré. A partir du premier fichier, l'algorithme EM apprend la table $Caractere_1$. La table 5.4 est celle avant l'apprentissage et la table 5.5 est celle modifiée par cet apprentissage.

En ce qui concerne les tables de probabilités des autres noeuds $Caractere_i$, c'est le deuxième fichier qui est utilisé. Dans le second fichier, ont été simulés aléatoirement dix milles cas avec, pour chaque cas, deux caractères qui peuvent seulement être un "a" and "b". L'ancienne table est la table 5.6 et la nouvelle est la table 5.7.

Pour la table du premier noeud (figure 5.5), on remarque que l'utilisateur commence de temps en temps avec un "b" mais que globalement il a tendance à commencer par un "a".

$Caractere_1$	$Pr[Caractere_1]$
a	0.5
b	0.5
c	0.0
&	0.0

FIG. 5.4 – $Pr[Caractere_1]$ avant apprentissage

$Caractere_1$	$Pr[Caractere_1]$
a	0.69
b	0.31
c	0.0
&	0.0

FIG. 5.5 – $Pr[Caractere_1]$ après apprentissage

	$Caractere_{i-1}$			
$Caractere_i$	**a**	**b**	**c**	**&**
a	0.5	0.0	0.0	0.0
b	0.5	0.0	0.0	0.0
c	0.0	1.0	0.5	0.0
&	0.0	0.0	0.5	1.0

FIG. 5.6 – $Pr[Caractere_i|Caractere_{i-1}]$ avant apprentissage

	$Caractere_{i-1}$			
$Caractere_i$	**a**	**b**	**c**	**&**
a	0.5095	0.0495	0.25	0.25
b	0.4905	0.9595	0.25	0.25
c	0.0	0.0	0.25	0.25
&	0.0	0.0	0.25	0.25

FIG. 5.7 – $Pr[Caractere_i|Caractere_{i-1}]$ après après apprentissage

Imaginons à présent le cas où l'utilisateur ne commençe ses phrases qu'avec des "a", la probabilité d'avoir un "a" au premier caractère serait alors de 1 et de ce fait, une nouvelle grammaire où toutes les phrases commencent par "a", aurait été apprise.

Pour la table du deuxième noeud (figure 5.7), l'algorithme *EM* apprend le fait qu'après un "b" peut suivre un "b". Après un "c", les caractères "a", "b", "c" ou encore "&" sont possibles. C'est la même chose pour les caractères qui peuvent suivre un &".

L'information sur "c" et "&" n'est pas intéressante, pour la simple et bonne raison que l'apprentissage ne donne aucune information pertinente sur ces caractères, ce qui s'explique par le fait que dans les phrases de l'utilisateur, il n'y a ni de "c" ni de "&". Dans ce cas là, l'effacement du caractère c pourrait éventuellement être envisagé, puisqu'il n'est jamais utilisé.
Cependant, pour "&", l'idée est de toujours intégrer le règle qui spécifie que un "&" peut toujours être suivi par un autre "&" car il s'agit d'un caractère signalant une fin de phrase.

L'apprentissage autorise donc l'évolution de la grammaire et permet par conséquent, dans certains cas, d'éviter les hésitations de l'interpréteur qui préférera choisir ce qui correspond aux habitudes de l'utilisateur. Quand les vraisemblances du classifieur ne disent rien sur un caractère, l'interpréteur préfère toujours, avec l'aide d'une grammaire apprise, les caractères que l'utilisateur a pris l'habitude d'écrire. L'interpréteur considérera moins les caractères que l'utilisateur utilise moins.
Si, avec la grammaire originelle, l'interpréteur pouvait hésiter de temps en temps entre deux probabilités équivalentes, il a maintenant à disposition plus d'informations.

Valeurs manquantes

Traiter les valeurs manquantes est une option que les réseaux bayésiens offrent grâce à l'algorithme *EM*. Il est possible d'apprendre une nouvelle grammaire même si il y a des valeurs qui manquent.
Dans le cas de l'interpréteur, cela arrive si, par exemple, les vraisemblances ne sont pas correctement enregistrées dans le classifieur. Dans cette situation, il pourrait y avoir des probabilités pour certaines lettres et pas pour d'autres, avec une somme des probabilités pour les états d'un caractère lu dans la phrase, qui n'est pas de 1.

En pratique, dans le deuxième fichier utilisé par l'activité d'apprentissage, il y a dix mille cas dont 20% contenant des valeurs manquantes. Les valeurs manquantes n'empêchent pas l'algorithme *EM* d'apprendre cette table, qui indique que "a" est plus souvent suivi par un "b", que "b" est toujours suivi par "b", et que les autres caractères sont souvent suivis par "a" ou "b". La table 5.8 nous montre les résultats.

Le seul problème avec les valeurs manquantes est que, s'il y a trop de valeurs manquantes dans l'ensemble des données à partir duquel l'utilisateur veut effectuer l'apprentissage ou s'il n'y a pas assez de cas dans cet ensemble, il y aura une perte d'informations sur la grammaire.

	$Caractere_{i-1}$			
$Caractere_i$	**a**	**b**	**c**	**&**
a	0.4784	0.0	0.2626	0.2626
b	0.5216	1.0	0.7374	0.7374
c	0.0	0.0	0.0	0.0
&	0.0	0.0	0.0	0.0

FIG. 5.8 – $Pr[Caractere_i|Caractere_{i-1}]$ après apprentissage avec valeurs manquantes

La grammaire actuelle, complètement certaine, risque d'être remplacée au profit d'une autre, non pertinente car apprise sur trop peu de cas (par exemple une table avec une probabilité 0.25 pour tous les états).

Une solution simple existe pour éviter cela : utiliser le concept de l'expérience comme expliqué dans le chapitre 4. Il s'agit de donner du poids au passé et aux anciennes tables de probabilités. En donnant de l'*expérience* à une ou plusieurs tables de probabilités antérieures, si la nouvelle grammaire ne procure pas beaucoup d'informations, l'ancienne grammaire sera gardée. Il serait donc plutôt intéressant de procurer du poids aux probabilités antérieures en fonction de la proportion des valeurs manquantes ou du nombre de cas dans la base de données.

5.2.3 Formules mathématiques

Construction du réseau

Jusqu'ici, seule une grammaire régulière simple, non mathématique, a été envisagée. Les formules mathématiques complexes, doivent être traduites à l'aide de grammaires non-contextuelles. Dans les grammaires non-contextuelles, rappellons que deux concepts ont été définis : les règles de précédence (tel caractère suit tel autre) et les règles non contextuelles (parenthèses contrebalancées, accolades, etc.)

Dans cette partie, seules les règles de précédence ont été prises en charge, ce qui permet d'utiliser l'approche de grammaire régulière de la section précédente, dans le problème des formules mathématiques.

En ce qui concerne les problèmes hors contexte, la difficulté trouve son origine dans l'incapacité de la forme simple du réseau bayésien de les intégrer. Une parenthèse doit être contrebalancée par une autre, mais rien n'est dit sur la position où cette autre doit être placée. Or, les tables de paramètres des noeuds d'un réseau concernent un noeud et ses parents. Ne connaissant pas la distance d'une parenthèse de gauche par rapport à une parenthèse de droite, il n'est pas possible de définir une relation parent-enfant entre ces deux caractères.

Par conséquent, nous ne considérons plus les règles non contextuelles comme règles de grammaires devant être traduites dans des tables. L'idée est de laisser l'interpréteur accepter les phrases sans considérer le non contextuel et, ensuite, par un mécanisme semi-externe, de corriger les erreurs (en ajoutant les parenthèses manquantes, par exemple). Une partie entière est consacrée à ce problème (section 5.3).

Pour en revenir aux règles de précédence, il s'agit donc de pouvoir extraire une grammaire régulière qui permet l'écriture d'une formule mathématique. Prenons la grammaire non-contextuelle suivante :

$A \rightarrow (A)$
$A \rightarrow B$
$B \rightarrow aB|bB|\epsilon$

Elle peut être transformée en :

$A \rightarrow (A$
$A \rightarrow B$
$B \rightarrow aB|bB|\epsilon$
$B \rightarrow)$
$B \rightarrow \epsilon$

Dans la grammaire régulière obtenue, une phrase comme "$((a$" peut être acceptée ce qui n'est pas une formule mathématiquement correcte. L'information sur le fait que les parenthèses doivent être contrebalancées a disparu. Il ne reste plus que les informations sur les règles de précédence.

A partir du moment où une grammaire régulière a été obtenue, la construction du réseau est la même que dans le cas de la grammaire régulière simple envisagée au début du chapitre, avec juste des états différents. Le réseau typique pour une formule mathématique se trouve en 5.9. Le réseau représente une phrase telle que "$b + 12\&$".

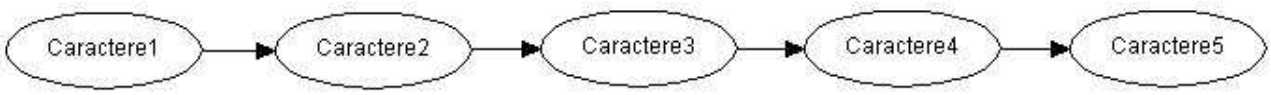


FIG. 5.9 – Réseau bayésien d'une grammaire mathématique simple

Par contre, les règles de grammaire sont différentes. La table 5.10 montre la table du premier caractère. La table 5.11 montre la table pour les autres caractères.

$Caractere_1$	$Pr[Caractere_1]$
(	0.17
)	0.0
{	0.17
}	0.0
a	0.17
b	0.17
+	0.0
1	0.17
0	0.17
&	0.0

FIG. 5.10 – $Pr[Caractere_1]$

	$Caractere_{i-1}$									
$Caractere_i$	(	)	**{**	**}**	**a**	**b**	+	**0**	**1**	**&**
(	0.2	0.0	0.17	0.0	0.0	0.0	0.17	0.0	0.0	0.0
)	0.0	0.25	0.0	0.0	0.25	0.25	0.0	0.17	0.17	0.0
{	0.0	0.0	0.17	0.0	0.0	0.0	0.17	0.0	0.0	0.0
}	0.0	0.25	0.0	0.33	0.25	0.25	0.0	0.17	0.17	0.0
a	0.2	0.0	0.17	0.0	0.0	0.0	0.17	0.0	0.0	0.0
b	0.2	0.0	0.17	0.0	0.0	0.0	0.17	0.0	0.0	0.0
+	0.0	0.25	0.0	0.33	0.25	0.25	0.0	0.17	0.17	0.0
0	0.2	0.0	0.17	0.0	0.0	0.0	0.17	0.17	0.17	0.0
1	0.2	0.0	0.17	0.0	0.0	0.0	0.17	0.17	0.17	0.0
&	0.0	0.25	0.0	0.33	0.25	0.25	0.0	0.17	0.17	1.0

FIG. 5.11 – $Pr[Caractere_i|Caractere_{i-1}]$

Propagation d'information

En ce qui concerne la propagation de l'information pour les formules mathématiques, le principe est le même que celui de la propagation pour la simple grammaire régulière puisque les problèmes non-contextuels n'ont pas été pris en compte dans le réseau lui-même. Les vraisemblances du classifieur sont ajoutées au réseau et l'information est propagée avec l'algorithme maximum propagation.

Une phrase admise est par exemple "$(a + a\&$", correcte au point de vue des règles de précédence mais n'intègrant pas l'aspect non-contextuel.

Apprentissage de paramètres

L'apprentissage de paramètres est identique, avec le système des deux fichiers contenant l'information sur les règles de précédence. L'utilisateur peut changer la grammaire en introduisant de nouvelles phrases dans la base de données.

Valeurs manquantes

Les valeurs manquantes sont traitées de la même manière que dans la grammaire régulière simple. Il faut comme avant prêter attention au fait que les valeurs manquantes peuvent biaiser les résultats.

5.3 Correction des erreurs

5.3.1 Objectif : Détecter et corriger les erreurs

L'interpréteur est à présent capable de reconnaître une formule mathématique, en choisissant, sur base des vraisemblances fournies par le classifieur, la formule à la fois la plus probable, à la fois admissible eu égard à la grammaire mathématique définie.

En d'autres mots, si le classifieur rencontre des difficultés dans la reconnaissance d'une lettre et que les vraisemblances sont réparties de manière uniforme sur plusieurs possibilités de caractères, l'interpréteur, pour choisir le caractère, se base sur les caractères avant et après le caractère en question afin de choisir la meilleure possibilité parmi celles offertes. Il s'exécute en utilisant uniquement les règles de précédence définies par la grammaire.

Prenons par exemple la phrase "$(a + b$", où les vraisemblances du classifieur ne donnent pas un meilleur candidat clair pour le troisième caractère (il hésite entre "+" et ")" mais où, le classifieur est presque complètement certain pour le "b".
Comme cela a été montré dans la section précédente, la grande probabilité de la lettre "b" augmente la probabilité du "+" parce que l'interpréteur élimine les états impossibles pour le troisième caractère (il supprimera la possibilité que le caractère soit un ")", parce qu'un ")" ne peut pas précéder un "b"). La troisième lettre doit être un "+" et non un ")".
Ensuite, le choix de "+" augmente la possibilité que le second caractère soit un "a" et ainsi de suite.

Grâce à l'utilisation du mécanisme de propagation dans les réseaux bayésiens, l'interpréteur ne regarde pas uniquement le maximum local de probabilité pour chaque caractère. Il ne se contente plus de regarder le voisin de droite et le voisin de gauche. Au contraire, l'interpréteur regarde plus loin et considère l'ensemble de la phrase.
Ainsi, l'interpréteur, en usant des règles de précédence, choisit toujours une phrase admissible si possible.

Cependant, un problème est lié à cette façon de procéder. L'interpréteur n'est pas vraiment capable de traiter les fautes dans la phrase. Il essaie toujours de prendre une phrase correcte quand les vraisemblances du classifieur le permettent.

Lorsque l'utilisateur écrit d'une main sûre une faute, et que le classifieur le reconnait sans hésitation, deux situations apparaissent.

Soit l'interpréteur reconnaît de manière **presque** sûre la mauvaise lettre et peut alors choisir un autre caractère possible à la place. Il choisira le caractère correct qui lui semble le plus probable.
Soit il est **complètement** certain du caractère et ne peut donc pas choisir un autre caractère (tous les autres caractères ont une probabilité nulle). Il met alors une évidence sur l'état relatif au mauvais caractère.

Dans la première situation, l'interpréteur exclut dès lors l'état impossible pour choisir le premier état possible et, cela, même si la probabilité est infime. Par conséquent, l'interpréteur choisit un caractère arbitrairement.
En effet, si cette probabilité infime existe, c'est parce que le classifieur trouve que cette lettre ressemble un peu au caractère réellement introduit. Cela ne respecte donc pas le choix et le souhait initial de l'utilisateur. Le choix est effectué par l'interpréteur en se basant sur l'aspect graphique d'une lettre fournie par les informations du classifieur.

Par exemple si l'utilisateur a écrit le phrase incorrecte "(+" ' et si la probabilité pour le second caractère est de 0.95 pour "+" et 0.05 pour "a",même si c'est évident, selon les probabilités, que l'utilisateur a choisi d'écrire un "+", l'interpréteur choisira un "‘a" qui, lui, peut être mis après une parenthèse. Or, si le "a" a été cité par le classifieur, c'est uniquement parce qu'il trouve que le "+" ressemble à un "a".

L'auteur de [5] essaie de corriger les erreurs logiquement. Par exemple, en mettant un ϵ quand un opérateur est sans opérande. Ici, si rien n'est changé, la correction se fait en choisissant le caractère qui ressemble graphiquement le plus à l'original, ce qui ne constitue pas une correction de type "logique".

Dans la seconde situation, où le classifieur est complètement certain du caractère lu (probabilité de 1) alors que ce caractère est impossible, le problème est encore plus important. Parce que si l'utilisateur a écrit "$a++b$" et que le classifieur est complètement sûr des quatre caractères, le réseau entre dans un état incohérent avec les règles de précédence (car un "+" ne peut jamais suivra un autre). Dans un cadre logiciel, mettre une évidence sur un état impossible génère une erreur de "propagation incohérente", expliquée par une division par 0 dans la règle fondamentale.

Nous allons à présent essayer de résoudre ces deux problèmes. Il faut observer qu'une tentative de résolution de ces obstacles a quelques conséquences. Par exemple, si dans la première section, la taille de la formule entrée ne changeait pas après l'activité de l'interpréteur, maintenant, parce que nous allons essayer de corriger les fautes, la taille peut changer (certaines lettres peuvent être ajoutées ou supprimées pour corriger une faute).

Puisque depuis le début, deux types de règles sont considérées, il faut aussi diviser les erreurs en deux groupes.
Le premier groupe comprend les *erreurs non contextuelles*. Elles portent sur la partie non-contextuelle d'une grammaire mathématique (comme définie dans la précédente section). Il s'agit par exemple des caractères qui doivent être contrebalancés avec d'autres (parenthèses de droite et de gauche, accolades, symbole intégral et variables d'intégration,...) et plus généralement toutes les contraintes appliquées au voisinage non direct d'un caractère.
C'est également le cas des nombres. Dans un nombre, il ne peut juste y avoir qu'une seule virgule et un nombre comme "05" ne peut être accepté (un zéro ne peut être devant un autre chiffre s'il apparaît comme premier chiffre d'un nombre). Ces problèmes peuvent se résoudre algorithmiquement avec l'aide des réseaux bayésiens.

Le deuxième groupe rassemble les *erreurs de précédence*. Elles concernent les règles de précédence, soit la partie régulière de la grammaire mathématique. Citons par exemple les opérateurs sans opérandes, une mauvaise séquence de caractères,... Concernant ce type d'erreurs, le problème est différent et nous aurons à le décomposer.

5.3.2 Erreurs non contextuelles

Idée générale

Notre première préoccupation concernant les erreurs hors contexte est de traiter les *couples* mathématiques. Nous entendons par là les caractères qui doivent être contrebalancés, comme par exemple c'est le cas des parenthèses de droite et de gauche, des crochets de droite et de gauche, des accolades de gauche et de droite, du symbole d'intégrale qui doit être suivi de près ou de loin de la variable d'intégration ("dx").

L'idée est ici d'utiliser le réseau bayésien existant pour obtenir de l'information à propos du nombre de caractères manquants et, ensuite, d'obtenir de l'information à propos des positions où il manquerait des caractères que l'interpréteur pourrait ajouter.
Avec l'information provenant du réseau, il est alors possible d'utiliser un mécanisme externe, un algorithme, dans le but de générer toutes les possibilités de phrases correctes (avec ou sans dialogue avec l'utilisateur, les deux cas sont possibles).

Ces activités sont appliquées dans le cas où un ou plusieurs membres de droite manquent. A la fin de cette section, le cas opposé (un ou plusieurs membres de gauche) sera analysé. Comme le traitement est identique pour tout type de couples (parenthèses, intégrales,...), c'est toujours le cas des parenthèses qui est pris en exemple.

Nous aborderons encore les nombres et les fonctions dont les définitions respectives sont elles aussi non contextuelles.
Enfin, un réseau bayésien est proposé, qui généralise l'ensemble des solutions proposées pour ces différents obstacles.

Trouver les caractères manquants

La première activité est de vérifier s'il y a vraiment un problème de couples dans la phrase analysée. Cela peut se faire en ajoutant des noeuds supplémentaires dans la structure de réseau bayésien présentée dans le cadre de la reconnaissance simple de formules. Ces noeuds ont pour but de compter le nombre de membres de droite de couples manquants (dans l'exemple, les parenthèses de droite manquantes).

Ces noeuds supplémentaires possèdent autant d'états que le réseau est capable de traiter des parenthèses manquantes. Si le réseau est configuré pour accepter un maximum de n parenthèses de droite manquantes, ces noeuds auront $n+1$ états : zéro parenthèses manquantes, une parenthèse manquante, deux parenthèses manquantes,...,n parenthèses manquantes.

Le nombre de ces noeuds (appelés *ParentheseAbsente*) est le même que le nombre de noeuds *Caractere*. Il y a une flèche orientée du noeud de parenthèse manquante d'un caractère vers le noeud *ParenthAbsente* du caractère suivant. Quand il y a une évidence sur un état "(" d'un noeud *Caractere*, le nombre de parenthèses manquantes augmente et de ce fait, le noeud *ParenthAbsente* correspondant change d'état (s'il ne manquait aucune parenthèse, il passe dans l'état "Une parenthèse manquante"). Quand c'est un ")", le nombre diminue.

Par conséquent, quand la formule la plus probable est trouvée, une évidence est mise sur chaque caractère, suivant la configuration la plus probable. Ensuite, l'algorithme somme propagation est utilisé et l'état le plus probable du dernier noeud *ParenthAbsente* donne le nombre réel de parenthèses manquantes.
En dessous, la figure 5.12 montre le nouveau réseau bayésien avec les noeuds *ParenthAbsente*.

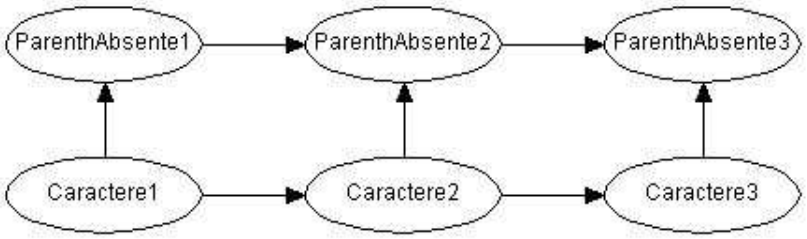


FIG. 5.12 – Réseau bayésien avec des noeuds supplémentaires comptant les paranthèses manquantes

On aboutit à de nouvelles tables de probabilités. La table (figure 5.13 montre le cas où aucune parenthèse ne manque et la table 5.14) montre le cas où une parenthèse est signalée comme manquante. Ces tables prouvent la simplicité de leur calcul. Elles expriment une logique booléenne.

	$Caractere_i$							
$ParenthAbsente_i$	(	)	{	}	**a**	**b**	+	&
0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.13 – $Pr[ParenthAbsente_i|Caractere_i, ParenthAbsente_{i-1} = 0]$

Correction a priori

Quand le réseau bayésien est construit avec les nouveaux noeuds *ParenthAbsente*, une première idée de correction peut être imaginée, que nous appellons correction *a priori*. Pour ce faire, il est cependant nécessaire de définir une nouvelle relation parent-enfant du noeud $ParenthAbsente_{i-1}$ vers le noeud $Caractere_i$ (voir 5.15) et de changer les paramètres du réseau (donc, les règles de précédence).

	$Caractere_i$							
$ParenthAbsente_i$	(	)	{	}	**a**	**b**	+	&
0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.14 – $Pr[ParenthAbsente_i|Caractere_i, ParenthAbsente_{i-1} = 1]$

Si, pour le noeud $Caractere_i$, $ParenthAbsente_i$ affirme qu'il n'y a pas de parenthèses manquantes, alors les règles sont les mêmes que précédemment mais si par contre, le noeud contient l'information qu'il y a des parenthèses manquantes, alors les règles changent pour le dernier caractère (pour le caractère &, fin de la formule). Le changement est simple : si on attend une ou plusieurs parenthèses de droite et qu'un caractère & se manifeste, alors la probabilité que le caractère suivant soit une parenthèse de droite est de 1.

En pratique, cela force le réseau bayésien à ajouter des parenthèses après le & à cause du mécanisme de propagation de l'information. Un algorithme est alors utilisé dans le but de vérifier si le réseau bayésien considère qu'il faut ajouter des parenthèses à la fin de la formule. S'il en faut, l'algorithme les rajoute, jusqu'à épuisement (jusqu'à ce que le dernier noeud n'exige plus de rajouter une parenthèse, et permet de remettre le &). La réponse finale est la phrase la plus probable avec des parenthèses supplémentaires en fin de formule.

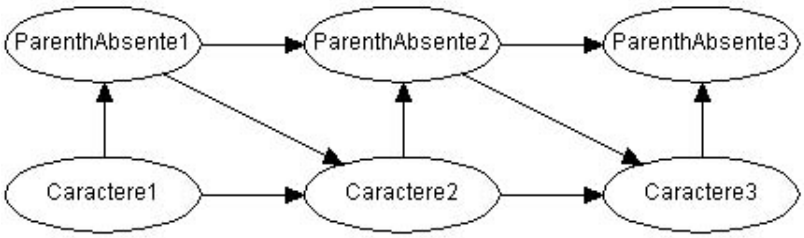


FIG. 5.15 – Réseau bayésien avec un noeud supplémentaire comptant les parenthèses de droite manquantes et rajoutant des parenthèses à la fin

Cette correction est très simple à implémenter mais la phrase correcte générée n'est pas satisfaisante. D'une part parce qu'une seule réponse est proposée à l'utilisateur et d'autre part parce que les parenthèses sont uniquement ajoutées à la fin.
La phrase est correcte mais il y a, dans la plupart des cas, plusieurs autres possibilités d'ajouts de parenthèses.

Par exemple, dans une phrase telle que "$(a + (b * a\&$", la solution "$(a + (b * a\&))$" ne peut pas être l'unique possible. De plus, pourquoi utiliser un réseau bayésien pour la correction si la tâche peut être basiquement faite par de simples opérations (lire la phrase, compter le nombre de parenthèses et ajouter le nombre de parenthèses manquantes de droite) ?

Dans la même catégorie de corrections a priori, on peut aussi considérer un réseau bayésien forçant les phrases à être corrigées au fur et à mesure. Ici, l'idée est de changer encore plus de paramètres en mettant la contrainte que, s'il y a des parenthèses manquantes en i et si le caractère $i-1$ est un caractère après lequel on peut ajouter une parenthèse de droite ("a", ")",...), alors le réseau bayésien est forcé d'ajouter une parenthèse de droite et rien d'autre. La tableau (5.16 montre comment le faire en pratique).

	$Caractere_{i-1}$							
$Caractere_i$	(	)	{	}	**a**	**b**	+	&
(	0.33	0.0	0.25	0.0	0.0	0.0	0.25	0.0
)	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0
{	0.0	0.0	0.25	0.0	0.0	0.0	0.25	0.0
}	0.0	0.0	0.0	0.33	0.0	0.0	0.0	0.0
a	0.33	0.0	0.25	0.0	0.0	0.0	0.25	0.0
b	0.33	0.0	0.25	0.0	0.0	0.0	0.25	0.0
+	0.0	0.0	0.0	0.33	0.0	0.0	0.0	0.0
&	0.0	0.0	0.0	0.33	0.0	0.0	0.0	0.0

FIG. 5.16 – $Pr[Caractere_i|Caractere_{i-1}, ParenthAbsente_{i-1} = 1]$

L'avantage de cette correction est qu'elle permet de garder la même taille de phrase dans certains cas : quand une parenthèse de droite est manquante, alors la priorité est de la mettre même si cela doit remplacer un caractère existant dans la formule. Mais le désavantage de cette solution est considérable car elle change la phrase originelle de l'utilisateur, supprime certains caractères qu'il a écrit et finalement, à cause de la propagation, risque de modifier profondément la phrase de départ.

Par exemple, si l'utilisateur a écrit "$(a + b$", parce qu'il y a des parenthèses manquantes et que les paramètres du réseau bayésien amènent à ajouter une parenthèse avant le signe "+" (position possible pour ajouter des parenthèses), grâce au mécanisme de propagation, la phrase deviendra "(a)&&". La formule se voit entièrement transformée.

Les deux propositions de correction *a priori* changent les règles de grammaire pour amener le réseau bayésien à ajouter lui-même des parenthèses. Cependant la première correction n'est pas satisfaisante et la seconde pourrait empêcher de trouver la phrase la plus probable, nous éloignant de l'objectif initial.

Par ailleurs, la seconde méthode a un sérieux inconvénient : l'algorithme maximum propagation ne peut plus être utilisé, car il cherche la configuration globale la plus probable sur tous les noeuds. Par conséquent, la phrase la plus probable sera la plus probable considérant aussi la configuration maximum pour les noeuds *ParenthAbsente*. Prendre le maximum des tables de *ParenthAbsente* modifie la phrase et, par exemple, diminue la probabilité des parenthèses de gauche. Ceci est du au fait que les noeuds *ParenthAbsente* deviennent les parents des noeuds de type *Caractere*. Il n'est plus possible d'obtenir la phrase la plus probable.

Il vaudrait donc mieux demander le plus d'informations possible au réseau bayésien et essayer de définir un mécanisme externe utilisant cette information pour pratiquer des corrections intelligentes. En effet, amener le réseau bayésien à corriger lui-même les formules grâce à ses paramètres ne s'avère pas constituer la meilleure piste.

Trouver les positions

Puisque les deux corrections précédentes (*a priori*) ne conviennent pas, c'est que l'information demandée au réseau bayésien sur le nombre de parenthèses manquantes, n'est pas assez riche. En reprenant la forme de réseau permettant de compter les parenthèses sans forcer l'ajout (figure 5.12), une autre information pourrait être exigée. Elle permettrait de détecter les positions où il est syntaxiquement correct de rajouter des parenthèses manquantes.

Concrètement, l'idée est d'ajouter de nouveaux noeuds, appelés *AjoutParenth*. Chacun de ces noeuds $AjoutParenth_i$ (où i est la position du caractère dans la phrase) est ajouté pour chaque différent noeud $ParenthAbsente_i$ et a deux parents : le noeud $ParenthAbsente_i$ et le noeud $Caractere_i$.

L'information donnée par un noeud $AjoutParenth_i$ peut être interprétée de la manière suivante : "*Si nous supposons qu'il y a des parenthèses de droite manquantes à la position i de la formule, si nous connaissons la valeur du caractère i et que ce caractère peut être suivi d'une parenthèse de droite, alors c'est qu'il est possible d'en rajouter après*".

L'information serait d'autant plus riche si le réseau pouvait mettre à jour la probabilité des parenthèses de droite manquantes quand une parenthèse est effectivement ajoutée grâce au noeud $AjoutParenth_i$. Par conséquent, si une évidence est mise sur un noeud $AjoutParenth_i$, alors l'état du noeud $ParenthAbsente_{i+1}$ doit changer (le nombre de parenthèses de droite manquantes doit diminuer). La figure 5.17 montre à quoi ressemble le réseau étendu.

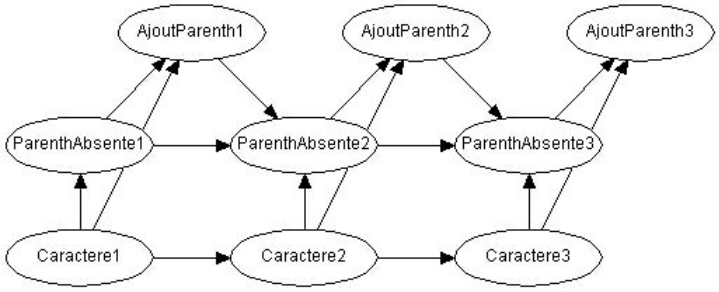


FIG. 5.17 – Réseau bayésien avec des noeuds supplémentaires indiquant où il est possible d'ajouter des parenthèses de droite manquantes

Une fois encore, les tables sont faciles à calculer et sont les mêmes quelle que soit la position du caractère. Concernant les noeuds *AjoutParenth*, les tables doivent intégrer le fait que quand il n'y a pas de parenthèses de droite manquantes, l'interpréteur ne peut pas en ajouter (table 5.18).

	$Caractere_i$							
$AjoutParenth_i$	(	)	{	}	**a**	**b**	+	&
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.18 – $Pr[AjoutParenth_i|Caractere_i, ParenthAbsente_i = 0]$

Quand il y a une parenthèse manquante, si le caractère est ")" ou "+", l'interpréteur peut ajouter une ou zéro parenthèses (exprimant le fait que l'interpréteur peut décider de ne pas rajouter de parenthèse à cet endroit, pour trouver une autre position possible). Le tableau 5.19 le montre.
Si le caractère est un caractère après lequel il n'est pas possible d'ajouter de parenthèses, alors l'interpréteur ne peut rien ajouter.
Si le nombre de parenthèses manquantes est plus grand que un, alors quand le caractère est un caractère après lequel on peut rajouter une parenthèse, l'interpréteur peut rajouter plusieurs parenthèses de droite manquantes en une fois mais peut aussi les répartir. Tous les choix sont laissés possibles, les noeuds rajoutés n'étant là que pour donner une information, dans le cas où une évidence est mise après un choix.

	$Caractere_i$							
$AjoutParenth_i$	(	)	{	}	**a**	**b**	+	&
0	1.0	0.5	1.0	1.0	1.0	1.0	0.5	0.5
1	0.0	0.5	0.0	0.0	0.0	0.0	0.5	0.5
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.19 – $Pr[AjoutParenth_i|Caractere_i, ParenthAbsente_i = 1]$

Les tables des noeuds $ParenthAbsente$ doivent elles être modifiés puisque chacun de ces noeuds a un nouveau parent, un noeud $AjoutParenth$

Si aucune parenthèse de droite n'est ajoutée après le caractère $i-1$, alors les tables du noeud $ParenthAbsente_i$ sont exactement les mêmes qu'avant.

Quand certaines parenthèses de droite sont ajoutées au caractère précédent (par exemple, le noeud $AjoutParenth_{i-1}$ montre que k parenthèses ont été ajoutées, avec $k \neq 0$), les tables sont différentes.
Supposons que le noeud $ParenthAbsente_{i-1}$ dise qu'il y a j parenthèses de droite manquantes. Alors, le noeud $ParenthAbsente_i$, ne doit pas dire qu'il y a j, $j-1$ ou $j+1$ parenthèses manquantes comme avant (caractérisant respectivement le cas où le caractère i est n'est pas une parenthèse, le cas où il est une parenthèse de droite et le cas où il est une parenthèse de gauche). Il doit à présent dire qu'il y a $j-k$, $j-1-k$ ou $j+1-k$.

Evidemment, le résultat de ces nombres est compris entre un minimum de zéro (le nombre de parenthèses manquantes ne peut pas être négatif) et un maximum ayant pour valeur le nombre maximum de parenthèses manquantes acceptées.

La table 5.20 représente le cas où une parenthèse a été ajoutée au précédent caractère et où il y avait une parenthèse manquante. La comparaison avec la table 5.14 marque la différence.

	$Caractere_i$							
$ParenthAbsente_i$	(	)	{	}	**a**	**b**	+	&
0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.20 – $Pr[ParenthAbsente_i|Caractere_i, ParenthAbsente_{i-1} = 1, AjoutParenth_{i-1} = 1]$

Un autre cas, où une parenthèse a été ajoutée au précédent caractère alors qu'il y avait **deux** parenthèses manquantes peut être pris en considération. Dans la table 5.21, le caractère que l'interpréteur est en train de lire est un ")", le nombre de parenthèses de droite manquantes passe donc de 2 à 0.
Cette diminution est d'une part expliquée par l'ajout de parenthèse de droite au caractère précédent et, d'autre part, par le fait que le caractère en cours de lecture est une parenthèse de droite, qui ferme une parenthèses de gauche précédente.

	$Caractere_i$							
$ParenthAbsente_i$	(	)	{	}	**a**	**b**	+	&
0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.21 – $Pr[ParenthAbsente_i|Caractere_i, ParenthAbsente_{i-1} = 2, AjoutParenth_{i-1} = 1]$

Correction a posteriori

A partir du réseau bayésien, l'interpréteur peut dès à présent connaître le nombre de parenthèses de droite qui manquent réellement et à quelles positions il peut les ajouter. De plus, si l'interpréteur informe le réseau qu'il ajoute effectivement une parenthèse, le réseau a la capacité de mettre à jour le nombre de parenthèses manquantes après l'ajout.

Plutôt que d'essayer de corriger les parenthèses manquantes par le biais du mécanisme de propagation des réseaux bayésiens comme dans la correction *a priori*, l'idée de la correction *a posteriori* consiste à obtenir le plus d'informations du réseau bayésien (ce qui a été fait dans les paragraphes précédents) et appliquer un algorithme externe qui, faisant usage de cette information, rajoute les parenthèses intelligemment.

Cet algorithme a pour but de générer de manière exhaustive les phrases corrigées possibles d'une phrase originelle incorrecte (où il y a des parenthèses de droite manquantes) ou de les générer en fonction d'interventions de l'utilisateur.
L'algorithme 3 est la version récursive d'un algorithme génèrant toutes les possibilités de correction. Il appelle une sous fonction définie par l'algorithme 4.

Algorithm 3 Algorithme principal d'ajout de parenthèse pour la formule $Formule$

1: Mettre les évidences correspondantes à $Formule$ sur les noeuds du réseau bayésien ;
2: $ParenthAbsente \leftarrow$ nombre de paranthèses manquantes (données par le réseau) ;
3: $ListeNoeudsPossibles \leftarrow$ la liste des positions où il est possible d'ajouter des parenthèses (donnée par le réseau) ;
4: $ListeFormulesCorriges \leftarrow \emptyset$;
5: Appeller la fonction récursive $AjoutRecursif(Formule, ParenthAbsente, ListeNoeudsPossibles, ListeFormulesCorriges)$, définie par l'algorithme 4 ;
6: Renvoyer $CorrectSentenceList$;

Cet algorithme présente un sérieux désavantage, dans la mesure où le nombre de phrases corrigées possibles est exponentiel (selon le nombre maximum de caractères manquants que l'interpréteur accepte, le nombre de caractères effectivement manquants et le nombre de positions où on peut ajouter des caractères).

Cependant, il est possible d'améliorer cet algorithme en suscitant l'avis de l'utilisateur (version interactive).
Le principe est de lui proposer une première position pour l'ajout d'une parenthèse manquante (lui proposant ainsi toutes les positions, une par une). Quand l'utilisateur est d'accord avec une position proposée, l'ajout est validé et l'interpréteur lui fournit une position pour la parenthèse manquante suivante.
Dans le pire des cas (si l'utilisateur choisit toujours la dernière position proposée), l'algorithme est le même que le précédent (version complète). Mais il s'avère que, dans la plupart des cas, le nombre de phrases corrigées générées est en moyenne réduit considérablement.

Algorithm 4 Fonction $AjoutRecursif(Formule, ParenthAbsente, ListeNoeudsPossibles, ListeFormulesCorriges)$, appellée par l'algorithme 3

$ListSize \leftarrow$ nombre d'éléments dans $PossibleNodesList$;
if $ListSize = 1$ **then**
  Obtenir la position du noeud dans $ListeNoeudsPossibles$;
  while $ParenthAbsente > 0$ **do**
    Ajouter une partenthèse de droite à la formule $Formule$;
    $ParenthAbsente \leftarrow ParenthAbsente - 1$;
  end while
  Ajouter $Formule$ dans $ListeFormulesCorriges$;
else
  $PremierNoeud \leftarrow$ le premier noeud de $ListeNoeudsPossibles$;
  $ListeEtats \leftarrow$ la liste des états de $PremierNoeud$ pour lesquels la probabilité est plus grande que zéro (donnés par le réseau) ;
  for all $X \in ListeEtats$ **do**
    Mettre une évidence sur l'état X du noeud $PremierNoeud$ dans le réseau ;
    $ParenthAAjouter \leftarrow$ le nombre de parenthèses représenté par X ;
    $ParenthAbsenteTemp \leftarrow ParenthAbsente - ParenthAAjouter$;
    $FormuleTemp \leftarrow Formule$ où $ParenthAAjouter$ parenthèses ont été rajoutées à la position représentée par $PremierNoeud$;
    $ListeNoeudsPossiblesTemp \leftarrow ListeNoeudsPossibles$ où $PremierNoeud$ a été supprimé ;
    Call $AjoutRecursif(FormuleTemp, ParenthAbsenteTemp, ListeNoeudsPossiblesTemp, ListeFormulesCorriges)$;
  end for
end if

Les deux algorithmes, dans leur forme définie jusqu'ici, rencontrent malgré tout deux obstacles :

- Ils génèrent certaines phrases inutiles. C'est par exemple le cas où la phrase manuscrite est "$((a) + (b * a) + (2\&$" et où la phrase $((a) + (b * a)) + (2)\&$ est générée deux fois (la deuxième fois est inutile).

  Cela s'explique par le fait qu'après une parenthèse de droite, une autre parenthèse peut être mise et qu'après un "a" aussi. Donc, puisque dans la phrase une parenthèse de droite est précédée d'un "a", l'interpréteur propose de mettre une parenthèse après le "a" et aussi après le ")". Mais, malgré tout, la phrase générée est la même dans les deux cas.

  Cela peut se régler en imposant qu'après une parenthèse de droite, aucune autre parenthèse de droite ne peut être ajoutée. En effet, une parenthèse de droite est toujours précédée d'un caractère après lequel on peut rajouter une parenthèse de droite. Aucun cas ne serait ainsi ignoré.

- Ils génèrent des phrases incorrectes par rapport aux règles non contextuelles. Cela vient du fait que le noeud *ParenthAbsente* décide s'il y a des parenthèses manquantes juste sur la base du caractère en cours et du nombre de parenthèses manquantes précédemment.

  Cela veut dire que si nous avons la phrase "$((a) + (b * a) + (2\&$", nous pouvons ajouter une parenthèse juste après le "b" car localement, il y a un manque de deux parenthèses. Mais, globalement, il ne manque pas deux parenthèses à cet endroit, puisqu'il y a une parenthèse de droite fermante, trois caractères plus loin.

  Il est possible d'ajouter un état supplémentaire au noeud *AjoutParenth* qui peut être appelé "cas incohérent", pour résoudre ce deuxième problème. Quand le caractère courant est une parenthèse de gauche et que le noeud de parenthèses manquantes nous dit qu'il n'y a pas de parenthèses manquantes, c'est incohérent, puisqu'à cet endroit, le réseau devrait normalement considérer qu'il en manque une. Dans l'exemple, quand l'algorithme ajoute une parenthèse après le "b", il ajoute une parenthèse inutile et l'état "incohérent" est donc utilisé à hauteur de la dernière parenthèse de gauche : le réseau affirme qu'il n'y a pas de parenthèse manquante alors qu'une parenthèse vient d'être ouverte.
  L'algorithme peut ainsi éviter de considérer les cas pour lesquels un noeud se trouve dans l'état "incohérent".

Afin d'améliorer la version interactive et d'obtenir une meilleure complexité algorithmique, il pourrait être intéressant d'utiliser la notion de caractères non-terminaux. L'idée est que, quand une première parenthèse est ajoutée, un caractère non-terminal est créé, dans lequel il n'est plus possible d'ajouter d'autres parenthèses.
En faisant cela, des propositions inutiles de la version interactive originelle de l'algorithme pourraient être filtrées puisque cette version de l'algorithme propose toutes les positions possibles pour la seconde parenthèse.

Par exemple, si nous avons la phrase "$((a + b$" et qu'en suscitant l'avis de l'utilisateur, la première parenthèse manquante est ajoutée après le b, alors, après la première itération, la phrase est "$((a + b)$". En faisant cela, l'interpréteur considère que "$a + b$" est un non terminal A. La phrase est par conséquent de la forme "$((A)$".

Par conséquent, la seule possibilité pour la seconde parenthèse est de l'ajouter à la fin de la phrase. Si on avait appliqué l'ancienne version interactive, l'algorithme aurait proposé une seconde solution "$((a)+b)$" où la seconde parenthèse est ajoutée avant le b. C'est inutile parce que l'utilisateur a déjà ajouté la première parenthèse en fin de phrase, il considère donc que la seconde parenthèse doit être ajoutée après la première, à la fin de la phrase. La manière de raisonner de l'utilisateur est structurée, il pense qu'il y a une sorte de hiérarchie entre les parenthèses. Utiliser la notion des caractères non-terminaux s'inscrit dans cette logique de raisonnement.

Parenthèses de gauche manquantes

La méthode est différente dans le cas de où l'interpréteur doit traiter les parenthèses de gauche manquantes et plus généralement, tout membre de gauche d'un couple.
Deux éléments doivent être modifiés dans le raisonnement : d'abord, la manière de compter les parenthèses, ensuite l'algorithme d'ajout.

Concernant la manière de compter les parenthèses, il faut ajouter des noeuds supplémentaires, exactement comme ceux utilisés pour compter le nombre de parenthèses de droite manquantes mais à la différence près que l'orientation des arcs doivent doit être inversée. C'est le premier noeud de parenthèses de gauche manquantes (celui le plus à gauche) qui doit dire à l'interpréteur le nombre des parenthèses de gauche effectivement manquantes. La figure 5.22, montre ce changement.

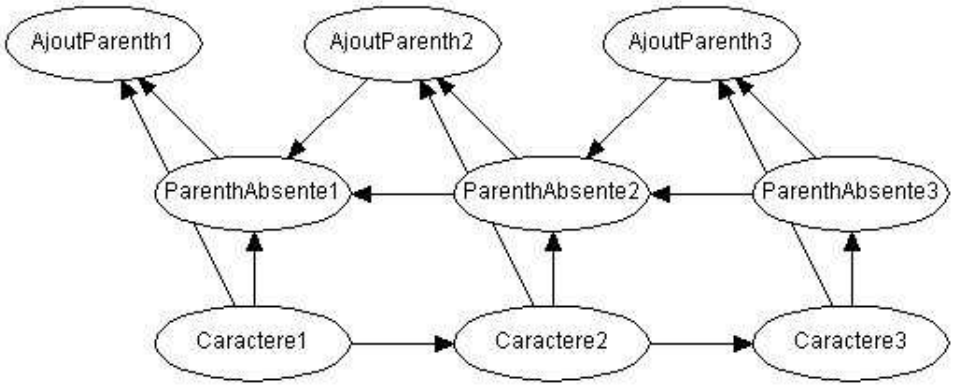


FIG. 5.22 – Réseau bayésien avec des noeuds supplémentaires résolvant le cas des parenthèses de gauche manquantes

Quant à l'algorithme, son mode de fonctionnement doit être transformé. Les positions données par le noeud *AjoutParenth* doivent être interprétées par l'algorithme de la façon suivante : si une position est donnée, l'interpréteur peut ajouter une parenthèse de gauche **après** cette position (et non plus avant cette position, comme dans le cas des parenthèses de droite).

Par exemple, si la formule erronée est "$b + a) + (b))$&", la première position donnée est la position deux. Zéro, une ou deux parenthèses peuvent être rajoutées après le +.

L'algorithme doit encore prendre en compte le fait qu'on peut toujours ajouter des parenthèses de gauche au début d'une phrase, avant le premier caractère.
Pour régler cela, il suffit d'ajouter un caractère & avant la phrase et de considérer qu'après le signe, des parenthèses de gauche manquantes peuvent toujours être ajoutées.

Nombres

Quand une formule contient des nombres, deux fautes sont susceptibles de survenir. La première concerne la virgule : il ne peut y avoir plus d'une virgule dans un nombre. La seconde concerne la validité d'un chiffre.

Si un nombre commence avec un zéro, soit le nombre est fini (et c'est juste un zéro), soit il y a une virgule. Le but est d'interdire des nombres tels que "05" qui, dans la plupart des langages mathématiques, ne sont pas valides .

Le principe est similaire à celui exposé dans la *correction a posteriori* utilisée pour les parenthèses. Deux séries de noeuds sont ajoutées au réseau.

La première série est composée de noeuds appelés $Nombre_i$ ayant deux parents, $Caractere_i$, $Nombre_{i-1}$ et cinq états : "*Pas un nombre*" (quand le caractère n'est pas un nombre), "*Premier chiffre 0*" (quand le caractère est le premier chiffre d'un nombre et qu'il vaut zéro), "*0 Virgule*" (quand le caractère est une partie d'un nombre et qu'il n'y a pas encore eu de virgule), "*1 Virgule*" (quand le caractère est une partie d'un nombre et qu'il y a déjà eu une virgule) et "*Virgule manquante*" (quand une virgule a été supprimée et qu'elle peut être rajoutée loin dans le nombre).

L'objectif de ces noeuds est de prendre une décision quant au statut exact d'une partie d'un nombre, lorsque le réseau connaît la valeur de cette partie du nombre et qu'il a des informations sur tout ce qui précède cette partie dans le nombre. "*Si l'état de la partie précédente du chiffre est "0 Virgule" et que la partie courante est une virgule, alors le noeud Nombre suivant doit être dans l'état "1 Virgule"*" est un exemple d'information apportée par ces nouveaux noeuds.

Les noeuds de la seconde série sont appelés $ChangeNombre_i$ et ont quatre états : "*Rien changer*", informant qu'il n'y a rien à changer ; "*Supprimer zéro*", décidant qu'un zéro inutile doit être supprimé ; "*Supprimer virgule*" pour supprimer des virgules inutiles et enfin, "*Garder virgule*" pour garder une virgule. Un noeud $ChangeNombre_i$ a deux parents, $Nombre_{i-1}$ et $Caractere_i$. Connaissant le statut du caractère précédent (une virgule a été lue) et le caractère courant (c'est une virgule), le noeud détermine la bonne décision à prendre (il faut supprimer la virgule inutile).

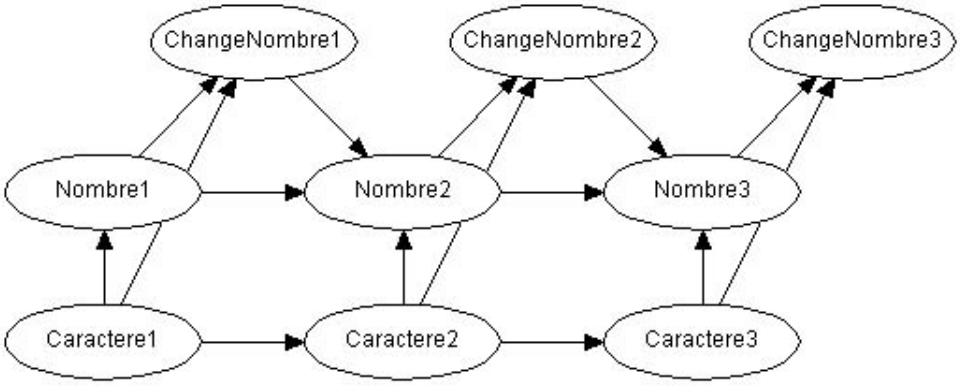


FIG. 5.23 – Réseau bayésien avec des noeuds supplémentaires permettant de gérer les contraintes liées aux chiffres

Si toutes les possibilités de nombres corrigés doivent être générées, comme dans l'algorithme *a posteriori* pour les parenthèses, il suffit de décider de supprimer les virgules inutiles quand une virgule a déjà été lue. L'interpréteur doit aussi considérer que la première virgule rencontrée peut être supprimée et par conséquent, qu'une autre peut être ajoutée après.

Cela signifie que selon la décision prise par l'interpréteur (qui essaye toutes les solutions), l'état du noeud $Nombre_i$ doit être mis à jour. "*Si j'ai décidé de supprimer la première virgule que j'ai rencontré, maintenant, il y a une virgule manquante*" est le type d'information qui doit mettre à jour le réseau. Dans cet exemple, l'état "*Virgule manquante*" a son utilité.

Afin d'effectuer cette mise à jour, le noeud $Nombre_i$ doit avoir un troisième parent, qui est $ChangeNombre_i$. Grâce à cela, certaines décisions peuvent être prises : "*S'il y a une virgule manquante et que j'en lis une, celle-là doit être gardée*". L'état "*Garder virgule*" est, dans ce cas, justifié.

Avec cette méthode, des chiffres sans virgules sont aussi générés. Le réseau ressemble à celui exposé à la figure 5.23.

En ce qui concerne les problèmes de zéros inutiles, le principe est plus simple. Si le réseau sait que la première partie du nombre est un zéro (zéro est le premier chiffre qu'il lit) et que ce zéro se trouve avant une série de chiffres qui ne contient pas de zéros (le noeud $Nombre_i$ est dans l'état "*Premier chiffre 0*"), alors ce zéro doit être supprimé (état "*Supprimer 0*" du noeud $ChangeNombre_i$).
Il n'y a presque pas de conflit entre cette décision et les décisions sur les virgules puisque les problèmes de zéros inutiles apparaissent avant les problèmes de virgules. Le seul inconvénient est que, si l'interpréteur supprime la première virgule d'un nombre, alors, de nouveaux zéros inutiles peuvent apparaître. Pour règler cette situation, deux solutions sont possibles : soit le réseau est utilisé une seconde fois pour corriger de nouveaux problèmes apparus, soit les deux problèmes de chiffres (celui des virgules, celui des zéros) sont séparés deux séries de noeuds différentes.

Fonctions

Prendre en compte des textes mathématiques contenant des fonctions et des équations demande une nouvelle adaptation du réseau. Dans une fonction ou dans une équation, il y a différentes parties, les expressions de droite et de gauche, séparées par le signe "=". Dans chacune des deux parties, le nombre de parenthèses doit être localement contrebalancé. L'interpréteur ne doit pas considérer le nombre global de parenthèses de gauche et de droite manquantes sur les deux expressions mais doit considérer respectivement le nombre de ces parenthèses dans chaque partie de la phrase manuscrite.

En ajoutant un nouvel état dans le noeud comptant les parenthèses manquantes, la situation devient gérable. Quand un noeud *ParenthAbsente* a pour parent un noeud *Caractere* qui représente le symbole "=", le noeud *ParenthAbsente* doit entrer dans l'état *Nouvelle expression de droite*.
Ensuite, quand un noeud $ParenthAbsente_i$ a pour parent un noeud $ParenthAbsente_{i-1}$ qui est effectivement dans cet état, l'interpréteur doit considérer qu'il n'y a plus de parenthèses manquantes. Il fait comme s'il analysait une nouvelle formule et recommence à compter les nouvelles parenthèses manquantes. Le mécanisme d'ajout de parenthèses est alors lancé séparément pour chaque expression de droite.

Généralisation

Les sections précédentes ont permis de maîtriser une série d'erreurs commises par l'utilisateur. Les solutions proposées doivent cependant être intégrées dans un même réseau bayésien. Puisque la solution consiste à utiliser des nouveaux noeuds, apportant différentes sortes d'informations, intégrer les solutions proposées revient à ajouter tous les nouveaux noeuds dans un réseau. Il n'y a pas de risques de conflits entres des noeuds résolvant différents types de problèmes, parce qu'ils analysent sous des angles différents la formule manuscrite.

Intégrer tous ces noeuds permet de contourner une contrainte qui n'avait jusqu'à présent pas été prise en compte, il s'agit des règles à appliquer entre différents symboles couplés, par exemple les règles à appliquer aux interactions entre les parenthèses et les accolades. C'est pour cette situation qu'une solution est ici proposée. Elle permet d'entrevoir la solution générale dans le cas d'influence entre les différentes sortes de couples.

Dans le langage mathématique choisi, les accolades sont utilisées comme des parenthèses de haut niveau. Elles ont la même utilisation mais une parenthèse de droite normale ne peut jamais suivre une accolade de droite : la formule "$\{(a+b)*c\}+c$" est valide mais la formule "$(\{a\}*b)$" n'est pas correcte. Par conséquent, dans les règles de précédence, le réseau doit spécifier le fait qu'après un "}", il n'y a jamais un ")". La table 5.24 contient cette contrainte.

L'algorithme qui ajoute les parenthèses et les accolades pourrait être le même que celui qui ajoute les parenthèses, puisque les accolades et les parenthèses doivent être contrebalancées de la même façon. La différence est qu'une accolade ne peut jamais se trouver entre une parenthèse de droite et de gauche, car les accolades représentent un niveau plus élevé. La phrase "$\{(a+b\}*c)$" n'est donc pas valide.

	$Caractere_i$							
$AjoutParenth_i$	(	)	{	}	**a**	**b**	+	&
(	0.33	0.0	**0.25**	0.0	0.0	1.0	0.25	0.0
)	0.0	0.25	0.0	**0.0**	0.25	0.25	0.0	0.0
{	**0.0**	0.0	0.25	0.0	0.0	0.0	0.25	0.0
}	0.0	**0.25**	0.0	0.33	0.25	0.25	0.0	0.0
a	0.33	0.0	0.25	0.0	0.0	0.0	0.25	0.0
b	0.33	0.0	0.25	0.0	0.0	0.0	0.25	0.0
+	0.0	0.25	0.0	0.33	0.25	0.25	0.0	0.0
&	0.0	0.25	0.0	0.33	0.25	0.25	0.0	1.0

FIG. 5.24 – $Pr[Caractere_i|Caractere_{i-1}]$

Quand l'algorithme ajoute des accolades manquantes, il doit vérifier les règles de précédence (il ne peut pas ajouter une accolade juste avant une parenthèse) mais il doit aussi vérifier qu'il n'ajoute pas une accolade de droite dans un non-terminal englobé par des parenthèses. Au lieu de changer d'algorithme, il est plus aisé de demander au réseau bayésien d'intégrer cette contrainte. Pour ce faire, le noeud $AjoutAccolade_i$ a un nouveau parent, soit $ParenthAbsente_i$.

Avec l'arrivée de ce nouveau parent, il faut définir la probabilité conditionnelle $Pr[AjoutAccolade_i | AccoladeAbsente_i, Caractere_i, ParenthAbsente_i]$ qui permet une décision comme "*Si je dois ajouter une accolade manquante et que le caractère courant le permet, alors, je dois vérifier d'abord qu'il n'y a pas de parenthèses manquantes, avant de décider si je peux ajouter des accolades*".

Si le caractère en cours de lecture est à un endroit où il y a des parenthèses manquantes, cela signifie que le caractère lu se trouve dans un non-terminal englobé par deux parenthèses. Cela indique que l'interpréteur se voit interdire l'ajout d'accolades. Par contre, si le noeud $ParenthAbsente_i$ affirme qu'il n'y a pas de parenthèses manquantes, l'interpréteur est autorisé à ajouter les accolades manquantes.

Le réseau intégrant les parenthèses et les accolades est caractérisé par la figure 5.25 et la table 5.26 est celle du noeud $AjoutAccolade_i$ quand il y a des parenthèses manquantes et une accolade manquante. Dans ce cas, l'interpréteur ne peut pas ajouter des accolades, à cause des parenthèses manquantes.

Cependant, ce réseau a aussi des noeuds appelés $ParenthAbsente'_i$ qui sont les parents des noeuds $AjoutAccolade_i$. Ils sont ajoutés parce que le comportement est le même quand il y a une, deux ou plus de parenthèses manquantes. Le noeud $ParenthAbsente'_i$ est là pour dire que s'il y a plus de zéro parenthèses manquantes, il est dans l'état *Plusieurs parenthèses*. Son rôle est de simplifier la complexité des tables en enlevant le lien de parenté imaginé entre les noeuds $ParenthAbsente_i$ et $AjoutAccolade_i$. Cela se voit dans figure 5.27.

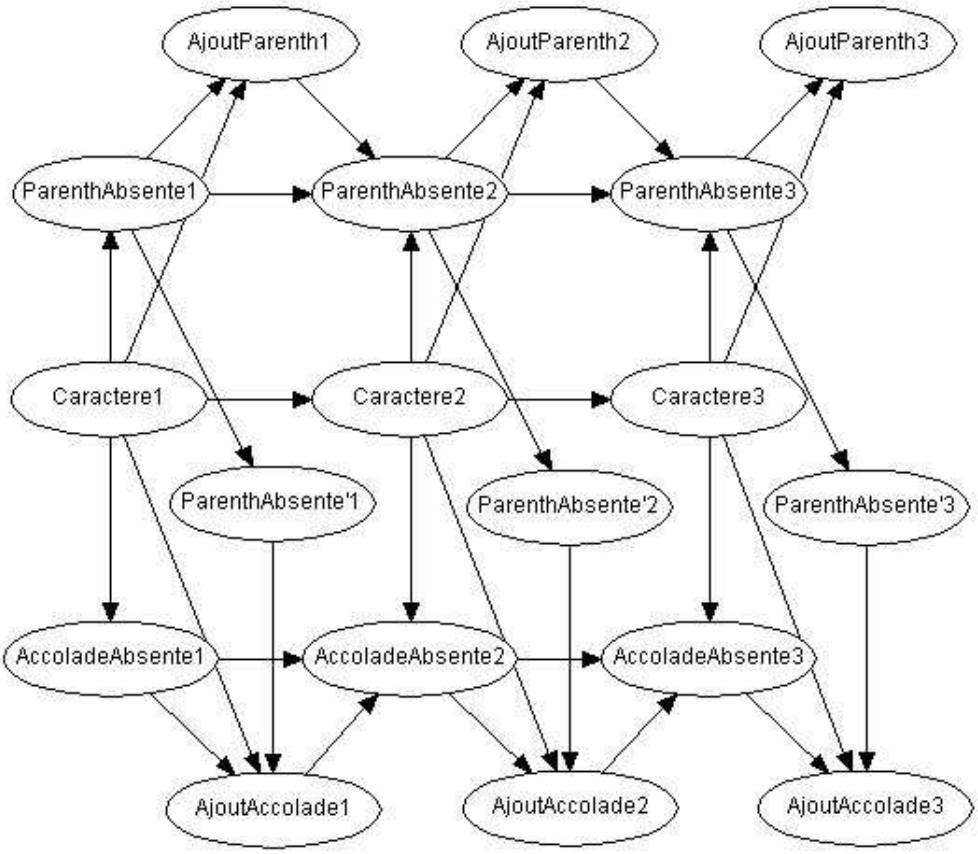


FIG. 5.25 – Réseau bayésien intégrant les parenthèses manquantes et les accolades manquantes

	$Caractere_i$							
$ParenthAbsente_i$	(	)	{	}	a	b	+	&
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

FIG. 5.26 – $Pr[AjoutAccolade_i|Caractere_i, ParenthAbsente'_i = Plusieursparenthèses, AccoladeAbsente_i = 1]$

	$Caractere_i$		
$ParenthAbsente_i$	0	1	2
0 ParenthAbsente	1.0	0.0	0.0
Several ParenthAbsente	0.0	1.0	1.0

FIG. 5.27 – $Pr[ParenthAbsente'_i|ParenthAbsente_i]$

5.3.3 Erreurs de précédence

Par "erreurs de précédence", on entend les erreurs qui ne sont pas typiques d'une grammaire hors contexte. C'est le cas lorsqu'un utilisateur place un certain caractère après un autre, de manière incorrecte. Par exemple, l'utilisateur écrit un opérateur sans opérande ("(+)", "()", "$sin()$",...), écrit une phrase où il a oublié les opérateurs ("(aa)"), ou encore écrit des opérateurs doubles ("++", "5, , 7"). On relève de nombreuses fautes de ce type.

Le problème fondamental est que, si le classifieur est complètement certain de lire cette faute (probabilité égale à *1*), le réseau génère une erreur (propagation incohérente), parce qu'il essaie de mettre une évidence sur un état impossible, considérant les règles du langage mathématique choisi, causant ainsi une division par 0.

De plus, si le classifieur n'est pas complètement sûr du caractère, il mettra les vraisemblances sur le noeud mais s'il existe une probabilité infime que le classifieur lise autre chose de correct, le classifieur choisit cette infime probabilité, plutôt que d'accepter une erreur.
Par exemple, si l'utilisateur a écrit "(+)" et que le classifier considère qu'il y a une probabilité de 0.99 que le second caractère soit "+" et une probabilité de 0.01 pour que le second caractère soit un "a", l'interpréteur choisita le "a" et renvoie "(a)". Ceci s'avère correct mais modifie complètement la formulation exprimée à l'origine par l'utilisateur.

Ce que l'interpréteur doit d'abord faire, c'est détecter et accepter ces erreurs quand il est sûr que ce sont des erreurs de l'utilisateur et non pas une mauvaise interprétation du classifieur. Une fois que la phrase erronée est acceptée, il peut essayer de la corriger en respectant l'utilisateur. Ces corrections peuvent être simples mais restent difficiles à lister. Pour chaque erreur particulière, une solution particulière devra être proposée et être effectuée de manière algorithmique.
Plusieurs ouvrages et articles traitent de ce genre d'erreurs et des corrections à y apporter, comme par exemple dans [5]. Il s'agit par exemple d'ajouter un signe ϵ lorsque l'opérande d'un opérateur a été oublié (l'opérande de la racine carrée).

Il subsiste aussi une difficulté qui concerne directement l'interpréteur : comment l'interpréteur peut accepter des erreurs ?
Il lui est interdit de le faire systématiquement, sinon il ne serait plus en mesure de reconnaître une phrase dans une grammaire.
Il faut donc d'abord être sûr que l'utilisateur a effectivement fait une erreur, avant d'accepter son erreur. Parce que s'il n'en a pas commise, les règles doivent être appliquées comme d'habitude. Dans le but de distinguer les deux cas, il faut envisager différents types d'utilisateur, et par conséquent prendre en compte le comportement de l'utilisateur dans le processus d'interprétation. La section suivante poursuit cet objectif.

5.4 Comportement de l'utilisateur

5.4.1 Objectif : Considérer la source des erreurs

La section précédente n'a pas complètement atteint son but. En effet, comme expliqué précédemment, pour détecter les erreurs dites de précédence, la suggestion est d'analyser le comportement de l'utilisateur et de le classifier en catégories. Avec cette nouvelle information, si l'interpréteur conclut que l'utilisateur est un utilisateur habitué à écrire des formules mathématiques erronnées, les règles de grammaire sont adaptées afin d'accepter les erreurs. L'évolution de la grammaire permet à la fois de détecter ces erreurs et d'envisager de les corriger par après. Elle permet aussi, à plus long terme, d'étendre la grammaire mathématique d'origine. Dans un premier temps, ce cas sera pris en compte.

Toutefois, il est aussi nécessaire de considérer que certaines erreurs sont imputables au classifieur qui n'arrive pas à reconnaître efficacement ce que l'utilisateur a écrit. C'est par exemple le cas lorsque l'utilisateur est fatigué et applique peu d'effort à l'écriture de la formule.

5.4.2 Utilisateur en erreur

Idée

Le principe de base consiste à définir deux types d'utilisateurs : ceux qui font des erreurs de précédence (*mauvais* utilisateurs) et ceux qui n'en font pas (*bons* utilisateurs). Ensuite, en analysant les formules mathématiques écrites par l'utilisateur, il est possible de le faire évoluer entre ces deux bornes. Si l'utilisateur écrit un grand nombre d'erreurs, son type empirera (i.e. évoluera vers le type "mauvais". Dans le cas contraire, son type s'améliorera.

En pratique, il est possible de placer précisément le type de l'utilisateur entre ces deux bornes en définissant un nouveau noeud, *TypeUtilisateur*, parent de tous les noeuds *Charactère*. Ce nouveau noeud a deux états : "*Mauvais*" et "*Bon*". La nouvelle structure du réseau est montrée en 5.28.

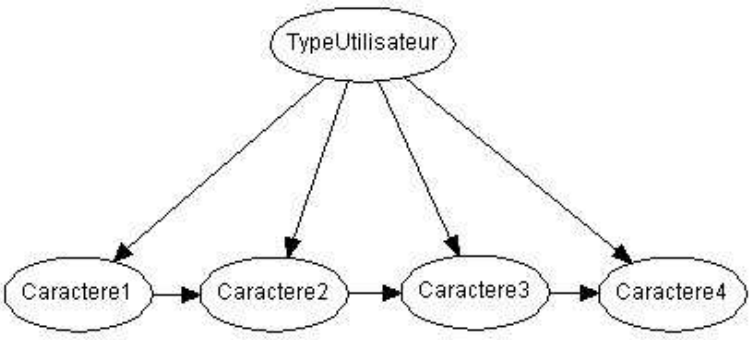


FIG. 5.28 – Réseau bayésien intégrant le type d'utilisateur

Les probabilités conditionnelles $Pr[Caractere_i|TypeUtilisateur]$ sont les mêmes quelle que soit la valeur de i. Si le type d'utilisateur est bon, alors les règles de précédence habituelles seront appliquées (les probabilités à priori des noeuds *Charactère* sont inchangées).

Par contre, si le type d'utilisateur est mauvais, aucune règle n'est appliquée (la probabilité est égale pour chaque état). Le tableau 5.29 le montre.

	$Caractere_i$							
$AjoutParenth_i$	(	)	{	}	**a**	**b**	+	&
(	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
)	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
{	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
}	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
a	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
b	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
+	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
&	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125

FIG. 5.29 – $Pr[Caractere_i|Caractere_{i-1}, TypeUtilisateur = Mauvais]$

Avec ces paramètres et en ajoutant des vraisemblances sur les noeuds *Caractere*, il est à présent possible de déterminer si l'utilisateur a effectivement écrit des erreurs ou non. Si l'utilisateur a écrit des erreurs de précédence et par conséquent n'a pas respecté les paramètres habituels du réseau, la grammaire qu'il utilise est plus proche d'une grammaire où aucune règle n'est d'application (vu qu'il ne respecte pas les règles). Son type d'utilisateur a évolué vers le type d'utilisateur "mauvais".
En fait, la probabilité $Pr[TypeUtilisateur|CharacterNode_i = x]$ donne le nouveau type d'utilisateur une fois qu'une nouvelle formule a été soumise au réseau.

Le seul problème lié à cette méthode est qu'une nouvelle fois, à cause du mécanisme de maximum propagation utilisé, il n'est pas possible d'intégrer le noeud sur le réseau existant. S'il est ajouté tel quel, quand la configuration la plus probable du réseau est calculée, l'algorithme considérera la configuration la plus probable, avec le type d'utilisateur le plus probable (au lieu de propager l'influence des noeuds *Caractere* sur les noeuds *TypeUtilisateur*, on prend le type d'utilisateur qui a la probabilité maximale).
Une nouvelle fois, c'est la somme propagation qui devrait être employée pour trouver le type d'utilisateur, alors que c'est la maximum propagation qui doit être appliquée pour déterminer ce que l'utilisateur a écrit.

Par conséquent, dans l'état actuel des outils cités dans la théorie générale des réseaux bayésiens, il est nécessaire d'utiliser deux réseaux bayésiens différents. Le premier est le réseau qui a été défini dans les deux sections précédentes. Le second est le réseau 5.28, qui nous permet d'obtenir de l'information sur le type d'utilisateur. Une fois que la phrase la plus probable est calculée, les vraisemblances originales données par le classifieur sont ajoutées sur les noeuds du second réseau et le type d'utilisateur est ensuite calculé.

Une autre possibilité consiste à définir un algorithme de propagation combinant les deux mécanismes (sommes et maximum) de manière différée, où la maximum-propagation est effectuée après avoir marginalisé les noeuds *TypeUtilisateur*. Une définition de cet algorithme de propagation hybride est développée dans le chapitre 6. Nous utilisons toutefois la solution des deux réseaux dans le cadre de l'application.

Maintenant que le type d'utilisateur peut être calculé, il s'avère intéressant d'utiliser cette nouvelle information pour donner plus ou moins d'importance aux règles de la grammaire originale afin d'accepter les erreurs.
Si le type d'utilisateur devient mauvais après que ce dernier ait écrit plusieurs erreurs, le but est à présent d'accepter ces erreurs et d'éviter le problème de propagation inconsistante expliquée dans la section précédente (si l'utilisateur écrit une erreur et que le classifieur est complètement sûr de cela, à cause des contraintes données par la grammaire la propagation donnera lieu à une impossibilité de calcul). Il est possible d'éviter ce problème, puisque lorsque l'utilisateur est mauvais, aucune règle n'est d'application. Pour accepter les erreurs, il faut que la grammaire soit adaptée, à chaque changement de type d'utilisateur.
Par conséquent, un interpréteur intégrant ce mécanisme est capable de détecter les erreurs de précédence en surveillant l'évolution du type d'utilisateur.

En pratique, pour calculer une nouvelle grammaire adaptée au type d'utilisateur (et selon ce type d'utilisateur, accepter un certain degré d'erreurs), il faut calculer la moyenne entre les probabilités conditionnelles dans le cas où l'utilisateur est bon et les probabilités conditionnelles dans le cas où l'utilisateur est mauvais.

Ainsi, si le ratio de type d'utilisateur est de 80% (la probabilité que l'utilisateur soit bon vaut 0.8) et que la probabilité qu'un caractère a soit suivi d'un "b" est 0.2 dans le cas d'un bon utilisateur (paramètres originaux du réseau) et de 0.1 dans l'autre cas (règles de grammaire avec une probabilité égale pour tous les états), alors la nouvelle probabilité dans le cas d'un utilisateur bon vaut (0.2*0.8)+(0.1*0.2).

Formellement,

$$\begin{gathered} Pr[Caractere_i|Caractere_{i-1} = k, TypeUtilisateur = Bon]_{j+1} \\ = \\ (Pr[Caractere_i|Caractere_{i-1} = k, TypeUtilisateur = Bon]_j.Pr[TypeUtilisateur = Bon]) \\ + \\ (Pr[Caractere_i|Caractere_{i-1} = k, TypeUtilisateur = Mauvais]_j.Pr[TypeUtilisateur = \\ Mauvais]) \end{gathered}$$

Il est nécessaire d'effectuer ce calcul de manière externe au réseau puisque deux réseaux séparés sont employés.
S'il n'y avait pas le problème de maximum propagation, qui provoque la séparation en deux réseaux, il serait possible d'utiliser un seul réseau et de simplement changer les probabilités à priori du noeud $TypeUtilisateur$. Par la probabilité conditionnelle $Pr[CharacterNode_i|TypeUtilisateur]$, le réseau bayésien aurait lui-même mis à jour les probabilités $Pr[CharacterNode]$, en utilisant le calcul expliqué ci-dessus.

En fait, parce qu'un second réseau est utilisé pour calculer la nouvelle valeur du noeud *TypeUtilisateur* et par conséquent le nouveau ratio de type d'utilisateur, nous sommes contraints de reconstruire un réseau complet pour intégrer ce nouveau ratio et de calculer la mise à jour des paramètres selon le nouveau ratio, de manière externe.

Une dernière chose à préciser concerne la probabilité à priori pour le type d'utilisateur quand le réseau est utilisé pour la première fois. Dans ce cas, en fait, les règles de grammaire sont les règles originales, qui posent le problème de propagation inconsistante lorsque des erreurs de précédence sont faites (quand les erreurs sont certaines à 100% et que le classifieur est sûr de lui). En d'autres mots, les règles originales ne permettent pas d'accepter des erreurs grâce à un ratio de type d'utilisateur puisqu'aucun n'a encore été défini.

Afin d'éviter cela, une probabilité à priori est définie par l'utilisateur pour son type, lors de la première utilisation du réseau (lors de sa construction). Un ratio par défaut peut cependant être donné : 90% de chances d'être un bon utilisateur et 10% d'être un mauvais utilisateur. Ceci est justifié par la deuxième hypothèse formulée au début du chapitre : l'utilisateur écrit parfois des erreurs mais globalement, connaît le domaine mathématique.

Un court exemple peut déjà donner un aperçu de l'évolution d'un type d'utilisateur en fonction des fautes qu'il aurait commis. Ci-dessous, ce type d'exemple est réalisé avec l'aide d'une grammaire régulière très simple, définie par :
$A \leftarrow aB$
$A \leftarrow -C$
$B \leftarrow -C$
$B \leftarrow \&$
$C \leftarrow aB$

Cela signifie que les paramètres de ce réseau devraient ressembler à ceux indiqués dans les tableaux 5.30 et 5.31 :

$CharacterNode_1$	$Pr[Caractre_1]$
a	0.5
-	0.5
&	0.0

FIG. 5.30 – $Pr[Caractre_1]$

	$Caractre_{i-1}$		
$Caractre_i$	**a**	-	&
a	0.0	1.0	0.0
-	0.5	0.0	0.0
&	0.5	0.0	1.0

FIG. 5.31 – $Pr[Caractre_i|Caractre_{i-1}]$

Le ratio de type d'utilisateur est celui par défaut (une probabilité de 0.9 d'être un bon utilisateur). Le choix de ce ratio implique, comme nous l'avons expliqué précédemment, une mise à jour du tableau des règles originales dans le cas où l'utilisateur est bon. Après calcul, les paramètres sont ceux des tableaux 5.32 et 5.33.

$Caractre_1$	$Pr[Caractre_1]$
a	0.483
-	0.483
&	0.033

FIG. 5.32 – $Pr[Caractre_1]$

	$Caractre_{i-1}$		
$CharacterNode_i$	**a**	-	&
a	0.033	0.933	0.033
-	0.483	0.033	0.033
&	0.483	0.033	0.933

FIG. 5.33 – $Pr[Caractre_i|Caractre_{i-1}]$

A présent, l'utilisateur écrit sa première phrase, qui devrait être : "$a-a-a$&". Les vraisemblances des caractères de cette formule mathématique sont celles du tableau 5.34.

V[Formule écrite à la main]	x		
	a	-	&
$V[CharacterNode_1 = x]$	1.0	0.0	0.0
$V[CharacterNode_2 = x]$	0.1	0.7	0.2
$V[CharacterNode_3 = x]$	0.95	0.05	0.0
$V[CharacterNode_4 = x]$	0.05	0.85	0.1
$V[CharacterNode_5 = x]$	1.0	0.0	0.0
$V[CharacterNode_6 = x]$	0.0	0.0	1.0

FIG. 5.34 – Vraisemblances fournies par le classifieur

Grâce au réseau bayésien principal, l'interpréteur arrive à la conclusion que la phrase admissible la plus probable est "$a-a-a$&". En utilisant le second réseau bayésien (celui avec le noeud supplémentaire *User Type*), l'interpréteur calcule le nouveau ratio, qui est : $Pr[TypeUtilisateur = Bon] = 0.95; Pr[TypeUtilisateur = Mauvais] = 0.05$
L'utilisateur est devenu meilleur parce que sa formule mathématique est correcte et que les hésitations de lecture du classifieur sont relatives.

Puisque le ratio de type d'utilisateur a changé, l'interpréteur adapte à nouveau les règles de grammaire et les tables qui sont les tables 5.35 et 5.36 :

$Caractre_1$	$Pr[Caractre_1]$
a	0.475
-	0.475
&	0.05

FIG. 5.35 – $Pr[Caractre_1]$

	$Caractre_{i-1}$		
$CharacterNode_i$	**a**	-	**&**
a	0.05	0.9	0.05
-	0.475	0.05	0.05
&	0.475	0.05	0.9

FIG. 5.36 – $Pr[Caractre_i|Caractre_{i-1}]$

L'utilisateur écrit une deuxième formule qui est "$--a\&$", avec les vraisemblances définies en 5.37 :

V[Formule écrite à la main]	x		
	a	-	**&**
$V[Caractre_1 = x]$	0.0	1.0	0.0
$V[Caractre_2 = x]$	0.0	1.0	0.0
$V[Caractre_3 = x]$	0.7	0.3	0.0
$V[Caractre_4 = x]$	0.0	0.0	1.0

FIG. 5.37 – Vraisemblances du classifieur

La réponse de l'interpréteur est "$--a\&$", ce qui signifie qu'il n'a pas corrigé la formule (les deux signes "$-$" qui se suivent), même si le ratio considère que l'utilisateur est assez bon. Cependant, en analysant les vraisemblances, on remarque qu'il y a une évidence forte sur le caractère faux, ce qui ne laisse aucun choix à l'interpréteur. Cet exemple montre que si l'utilisateur et bon et qu'il y a une évidence forte sur une erreur, cette erreur est acceptée.

A présent, supposons que les vraisemblances n'aient pas été celles du tableau précédent mais ressemblent plutôt à celles présentes dans le tableau 5.38.

Ces vraisemblances semblent indiquer que la réponse de l'interpréteur va être identique comme c'est le cas avec l'évidence dure. Mais, à cause du ratio d'utilisateur qui le considère bon, la formule est corrigée et l'erreur n'est pas acceptée. L'utilisateur n'est pas assez mauvais que pour qu'on accepte son erreur. La réponse de l'interpréteur est : "$a - a\&$".

V[Formule écrite à la main]	x		
	a	-	&
$V[CharacterNode_1 = x]$	0.3	0.7	0.0
$V[CharacterNode_2 = x]$	0.2	0.8	0.0
$V[CharacterNode_3 = x]$	0.7	0.3	0.0
$V[CharacterNode_4 = x]$	0.0	0.0	1.0

FIG. 5.38 – Vraisemblances du classifieur

Revenons aux vraisemblances avec évidence forte, définies en 5.37. Grâce au second réseau bayésien, avec ces vraisemblances, l'interpréteur calcule un nouveau ratio : $Pr[TypeUtilisateur = Bon] = 0.72; Pr[TypeUtilisateur = Mauvais] = 0.28$. L'interpréteur a accepté la phrase mais il sait que l'utilisateur a écrit une erreur, le ratio a donc évolué vers le bas.

Les tables de paramètres sont recalculées afin d'intégrer ce ratio et sont celles définies en 5.39 et 5.40.

$CharacterNode_1$	$Pr[Caractre_1]$
a	0.432
-	0.432
&	0.126

FIG. 5.39 – $Pr[Caractre_1]$

	$Caractre_{i-1}$		
$CharacterNode_i$	**a**	-	&
a	0.126	0.738	0.738
-	0.432	0.126	0.126
&	0.432	0.126	0.126

FIG. 5.40 – $Pr[Caractre_i|Caractre_{i-1}]$

L'utilisateur écrit à présent la formule "$-a-a$&" avec les vraisemblances indiquées en 5.41.

La réponse de l'interpréteur est cette fois-ci "$--a-a$&". L'interpréteur a accepté l'erreur parce que le ratio de type d'utilisateur était moins certain de la qualité de l'utilisateur. En outre, les deux premiers caractère étaient assez sûrs.

Le but est cependant atteint : l'interpréteur décide dynamiquement le type d'utilisateur, permet son évolution et prend des décisions adaptées.

Le ratio final est $Pr[TypeUtilisteur = Bon] = 0.57; Pr[TypeUtilisateur = Mauvais] = 0.42$. Si l'utilisateur continue à écrire les mêmes erreurs (un $-$ suivant un autre $-$), cette erreur sera de plus en plus acceptée et finira par ne plus vraiment être considérée comme faute.

V[Formule écrite à la main]	x		
	a	**-**	**&**
$V[Caractre_1 = x]$	0.1	0.9	0.0
$V[Caractre_2 = x]$	0.0	1.0	0.0
$V[Caractre_3 = x]$	0.33	0.33	0.33
$V[Caractre_4 = x]$	0.2	0.8	0.0
$V[Caractre_5 = x]$	0.8	0.1	0.1
$V[Caractre_6 = x]$	0.0	0.0	1.0

FIG. 5.41 – Vraisemblances du classifieur

Considérons à présent que toute phrase écrite par l'utilisateur est enregistrée dans une base de données. Puisque des phrases fausses peuvent être acceptées, cela signifie que l'activité d'apprentissage va tenir compte de ces phrases fausses acceptées. L'apprentissage de l'ensemble de données va changer la grammaire pour qu'elle accepte ces erreurs. On pourrait penser que c'est une nouvelle difficulté à résoudre pourtant, au contraire, cette conséquence de la prise en compte du comportement de l'utilisateur peut être intéressante.

En effet, supposons que le langage mathématique original défini par les probabilités conditionnelles du réseau n'intègrent pas une suite de symbole que l'utilisateur voudrait voir accepter. Par exemple, une succession de signes "$-$" dans la grammaire de l'exemple précédent. Ou encore, imaginons que dans la grammaire originale, mettre un symbole "(" après un symbole ")" est impossible alors que l'utilisateur considère que cela revient à une multiplication. Ecrire "$(a+b)5$" signifierait "$(a+b)*5$".

En combinant l'acceptation progressive des erreurs et l'interpréteur capable d'apprendre et de changer lui-même ses probabilités conditionnelles, l'interpréteur est en fait capable de faire évoluer la grammaire en fonction de ce que l'utilisateur veut. En pratique, à force d'écrire des phrases contenant des ")(" (et par conséquent, des erreurs selon l'interpréteur), le type d'utilisateur devient mauvais et ces phrases sont acceptées.
Après un certain nombre de cas, l'utilisateur peut demander à l'interpréteur d'accomplir un apprentissage, ce qui donne lieu à une grammaire changée, intégrant la nouvelle règle.

Accomplir un apprentissage dans l'exemple complet présenté précedemment, après un certain nombre de phrases interprêtées avec l'erreur des signes $-$, va changer les règles de grammaire pour que cette erreur soit définitivement acceptée.
Notons qu'il est inutile d'apprendre à propos du caractère & parce que c'est un caractère spécial, indiquant tout simplement une fin de phrase. Pour la colonne des tables de paramètres correspondant à ce signe, la règle doit toujours rester identique.

La méthode expliquée permet donc de faire évoluer la grammaire dynamiquement. Elle comporte cependant deux faiblesses.

D'abord, il est impossible de changer les règles non-contextuelles avec ce mécanisme, puisque ces règles là ne sont pas inscrites dans les tables de probabilités conditionnelles entre noeuds de

type caractère.
Ces règles là sont éparpillées dans plusieurs séries de noeuds particuliers (noeuds de parenthèses manquantes,...). Un autre mécanisme doit être trouvé, capable d'apprendre le réseau à partir d'une grammaire non contextuelle et capable d'apprendre une grammaire non contextuelle à partir de cas dans une base de données.

La seconde faiblesse vient du fait qu'il serait intéressant de pouvoir apprendre de nouveaux états pour les noeuds de type caractère. Si un symbole mathématique n'avait pas été défini dans la grammaire originale, l'interpréteur devrait être en mesure d'apprendre l'existence de ce symbole.

Si ces deux défauts sont contournés, l'interpréteur peut alors complètement apprendre une base de données et par conséquent, être dynamiquement configurable.

5.4.3 Utilisateur fatigué

Principe

Afin de décider du type d'utilisateur, l'interpréteur compare les règles de grammaire originales et les règles sans contraintes (probabilité égale pour chaque caractère, quel que soit le caractère précédent). De cette manière, il peut déterminer si l'utilisateur a écrit une faute ou non.

Imaginons à présent une situation où les vraisemblances du classifieur, pour un caractère, sont réparties également entre un nombre important d'états possibles. C'est par exemple le cas quand, pour le second caractère d'une phrase, tous les symboles mathématiques sont reconnus comme candidats et ont une probabilité à peu près égale.

Qu'est-ce que cela signifie ? Cela ne traduit pas nécessairement un manque de connaissance du domaine mathématique de la part de l'utilisateur, qui le voit écrire un grand nombre d'erreurs. En effet, quand l'utilisateur écrit une faute, il en est généralement inconscient. Par conséquent, il écrit cette faute comme s'il écrivait n'importe quel caractère : il l'écrit d'une main sûre. La probabilité n'est pas donc pas fortement dispersée, elle est concentrée en un état).
Le fait que les probabilités sont éparpillées entre un grand nombre d'états signifie que le classifieur a rencontré d'importantes difficultés à lire le caractère parce qu'il a été écrit de manière très peu précise.

Cependant, même si cela ne signifie pas que l'utilisateur est mauvais, l'utilisation du mécanisme expliqué dans la section précédente amène actuellement l'interpréteur à considérer que l'utilisateur est mauvais. Le simple fait que les probabilités sont réparties sur un grand nombre d'états implique dans la plus part des cas que des évidences faibles sont mises sur certains caractères impossibles selon la grammaire originelle. Par conséquent, le type d'utilisateur devient mauvais alors que l'utilisateur n'a pas réellement écrit de faute.

Considérons à présent une autre catégorie d'utilisateurs : l'utilisateur fatigué ou non fatigué. Cette catégorie exige de définir comment l'interpréteur peut exactement déterminer si l'utilisateur est fatigué et, le cas échéant, de prendre une décision.

Détection

Dans un premier temps, il faut déterminer si les vraisemblances sont fortement dispersées entre les états possibles d'un noeud du réseau. Le calcul d'entropie peut alors intervenir. Plus l'entropie est importante pour les vraisemblances d'un caractère, plus les probabilités sont réparties entre les différents états du noeud correspondant.

La définition de l'entropie dans ce cas est la suivante :

Si la variable représentant le caractère de la formule analysée est appellé $Caractere_i$ et peut être dans les états $\{x_1, x_2, ...x_n\}$; si les vraisemblances du classifieur pour cette variable sont $V[Character_i]$, alors

$$Entropie(Caractere_i) = -\sum_{k=1}^{n} V[Caractere_i = x_k].log(V[Caractere_i = x_k])$$

Il donc est possible de déterminer l'état de fatigue de l'utilisateur de la façon suivante :

- Si l'entropie augmente en moyenne d'un caractère au suivant (dans la formule) et que cette augmentation est plus haute qu'un certain niveau (c'est à dire si l'utilisateur semble de moins en moins concentré sur ce qu'il écrit)
- Si l'entropie du dernier caractère de la formule est plus grande qu'un certain niveau (c'est à dire si l'utilisateur, en fin de formule, semble très peu concentré)

$\Longrightarrow$ Alors, l'utilisateur est fatigué.

La première condition est nécessaire afin d'éviter que la décision soit uniquement prise sur un seul caractère. Si elle l'était, la conclusion pourrait être falsifiée par exemple par le fait qu'il y a une tache d'encre sur le papier et que par conséquent, le caractère ne peut pas être lu. La combinaison de ces deux conditions consitue une forme adaptée d'entropie.

Le calcul exact n'a pas encore été défini, puisque les deux "niveaux" cités doivent être estimés à partir d'expériences, en fonction de certains facteurs comme le passé de l'utilisateur (est-il habitué à être peu concentré ?), la taille de la phrase (plus grande est la formule, plus pertinent est le fait de contrôler si l'utilisateur est fatigué), le nombre de formules déjà écrites...

Les explications précédentes ne permettent donc qu'un calcul théorique, tant que les niveaux n'ont pas été calculés.

Utilisation

Il faut à tout prix éviter que le type d'utilisateur (bon ou mauvais) évolue, quand l'utilisateur est fatigué, parce que cela pourrait falsifier les résultats (l'utilisateur ne commet pas de faute, il est juste moins lisible par rapport à d'habitude, pour le classifieur)
En pratique, le plus simple est de désactiver le mécanisme d'évolution du type d'utilisateur quand l'interpréteur est arrivé à la conclusion que l'utilisateur est fatigué.

L'interpréteur doit donc d'abord calculer les conditions d'entropie définies précédemment et s'il conclut que l'utilisateur est fatigué, alors il ne doit pas calculer le nouveau ratio de type d'utilisateur et n'est pas obligé de modifier les règles de grammaire.

En appliquant ce principe, l'interpréteur arrive à résoudre la difficulté que peut représenter un utilisateur fatigué. Supposons un utilisateur qui n'a jamais fait d'erreurs mais qui, pendant quelques jours où il a utilisé l'interpréteur, était peu concentré. Les formules qu'il a écrites seront appréhendées avec la grammaire originale (elle n'aura pas changé vu qu'il n'a jamais fait d'erreur) et, de cette façon, tous les caractères impossibles seront écartés du raisonnement de l'interpréteur. Ceci implique qu'il y aura peut être plusieurs solutions possibles, mais qu'au moins, aucune solution contenant des suites de caractères impossibles ne sera considérée.

5.5 Eléments d'évaluation

Après avoir parcouru la théorie générale des réseaux bayésiens dans la première partie de cette étude, nous avons appliqué celle-ci à la reconnaissance et à l'interprétation de formules mathématiques. Cette application peut être généralisée à l'interprétation de textes structurés.

Afin de déterminer si l'interpréteur obtenu est une solution viable à la reconnaissance de formules mathématiques, un logiciel a été construit et est présenté en annexe (annexe I).

Les différents concepts de la théorie générale ainsi que les méthodes de propagation et d'apprentissage de paramètres ont été intégrées dans le raisonnement qui a abouti à la construction de l'interpréteur.
Notre raisonnement comprend trois axes : la reconnaissance de la formule, la correction de ses erreurs et la prise en compte du comportement de l'utilisateur. Il nous met en mesure d'évaluer les potentialités des modèles bayésiens.

En ce qui concerne la reconnaissance de la formule mathématique, la pertinence des réseaux bayésiens n'est plus à démontrer. En utilisant une modélisation très simple de la phrase et en appliquant l'algorithme de maximum propagation, l'interpréteur trouve de manière efficace la formule admissible la plus probable.
La satisfaction à l'égard de ce résultat nous invite à s'interroger sur l'opportunité de ne pas se contenter de la recherche de la formule la plus probable et de rechercher les formules mathématiques les plus probables, pour éviter d'écarter inutilement des solutions qui auraient pu convenir à l'utilisateur. Le prochain chapitre (chapitre 6) abordera cet aspect à travers une première extension de la théorie des réseaux bayésiens.

La proposition quant à la correction des erreurs présentes dans une formule demeure discutable. Si nous sommes parvenus à mettre au point une méthode de correction pour les problèmes non contextuels (*algorithme a posteriori*), celle-ci n'épuise pas pour autant la puissance de l'outil bayésien. En effet, l'information ajoutée au réseau d'origine est une information booléenne prise en charge par un algorithme externe. D'autres modèles pourraient exprimer la même information.

D'autant plus que pour des formules mathématiques de grande taille, dans lesquelles de nombreux membres de couples - par exemple les parenthèses de droite - sont oubliés, la taille du réseau bayésien devient considérable et rend sa construction complexe. Ceci est principalement imputable aux dimensions des tables de probabilités générées.

En résumé, si les réseaux bayésiens autorisent la correction de formules, la complexité de leur usage nécessite de les confronter à d'autres modèles. Le modèle des grammaires stochastiques présenté dans le chapitre 7 semble plus opérationnel dans cette perspective.

Ceci ne doit pas exclure le rôle spécifique qu'un réseau bayésien peut jouer dans le domaine de la correction. Il peut en effet être utilisé en vue du classement des multiples corrections proposées pour une même formule, selon les préférences de l'utilisateur. Le chapitre 6 envisage ceci comme une deuxième extension de la théorie des réseaux bayésiens.

Enfin, en matière de prise en compte du comportement de l'utilisateur, les réseaux bayésiens ont un apport non négligeable.
La définition de types d'utilisateurs et son intégration dans le réseau d'origine permettent d'apporter une solution à un type de correction qui semblait être impossible, soit la correction des des erreurs de précédence. Elle permet aussi l'évolution de la grammaire mathématique utilisée par l'interpréteur et la prise en considération de nouvelles règles, en étudiant le comportement de l'utilisateur.
L'utilisation du concept d'entropie nous a permis, quant à lui, de prendre en charge l'état de fatigue de l'utilisateur et les erreurs du classifieur.

Cependant, les concepts présentés dans la théorie générale des réseaux bayésiens ont empêché de d'intégrer de manière complète les types d'utilisateurs dans le réseau d'origine. Dans la mesure où nous ne disposions pas d'un outil autorisant la combinaison des algorithmes de somme et de maximum propagation, ils nous ont forcés à scinder le réseau bayésien utilisé.
Une troisième extension de la théorie de Bayes (chapitre 6) présente dans ce registre une solution.

Troisième partie

Extensions et limites des réseaux bayésiens

Chapitre 6

Extensions des algorithmes de propagation *A. Bonbled*

6.1 Objectif

Après avoir respectivement fait le tour de la théorie générale des réseaux bayésiens et l'avoir appliquée à la problématique de la reconnaissance de textes manuscrits, il reste deux tâches à accomplir. La première consiste à montrer les potentialités de l'outil en matière d'adaptation à des situations spécifiques. Elle fait l'objet de ce chapitre.
Toutefois, il faut aussi reconnaître, comme l'augurait l'application à la reconnaissance de texte, les limites de l'outil et son incapacité à modéliser de manière opérationnelle certains domaines. Le chapitre 7 s'engagera dans cette réflexion en explorant un modèle concurrent à celui des réseaux bayésiens.

Dans la perspective d'une mise en exergue des multiples capacités des réseaux bayésiens, envisageons plus précisément la prise en charge par ces réseaux des mécanismes de propagation et les avantages notoires que la méthode lui procure.

Pour cela, on s'interrogera dans un premier temps sur la capacité d'un réseau bayésien d'obtenir plus d'un résultat, en l'occurence un classement de plusieurs résultats par probabilité.
A cet effet, un essai de précision de l'algorithme de maximum propagation sera effectué afin de le voir proposer non seulement la meilleure configuration d'un réseau mais aussi les autres configurations, classées de la meilleure à la moins bonne.
Ensuite, on fera intervenir un réseau bayésien pour classer une liste de résultats provenant d'une source quelconque.
Enfin, on réfléchira à la meilleure manière de combiner les résultats issus de classements différents.
Dans un deuxième temps, la question sera de savoir si les algorithmes de somme et de maximum propagation peuvent être utilisés simultanément pour définir un algorithme hybride. Dans cette optique, on s'interrogera sur leur capacité à configurer de manière optimale une partie des variables du réseau tout en propageant l'information à l'aide de la somme propagation pour les autres noeuds.

6.2 Obtenir un classement de résultats

6.2.1 Affiner les résultats de l'algorithme de maximum propagation

L'algorithme de maximum propagation a pour intention de déterminer la meilleure configuration de tous les noeuds d'un réseau bayésien et de trouver sa probabilité. Par configuration, on entend une combinaison d'états sur tous les noeuds du réseau. L'algorithme de Dawid ([1]) atteint ce résultat en remplaçant, dans l'algorithme de somme propagation, l'opérateur somme par l'opérateur maximum et en retraçant ensuite le raisonnement pour retrouver les états maximisant de chaque noeud.

Dans [4], un exemple complet est donné. Il est reproduit ci-dessous. Il illustre le raisonnement qui a été réalisé afin d'étendre cet algorithme.

Supposons un noeud à trois noeuds (représenté en 6.1) dont les paramètres sont $Pr[A]$ (table 6.2), $Pr[B|A]$ (table 6.3) et $Pr[C|B]$ (table 6.4).

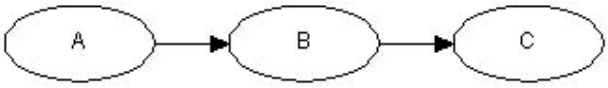


FIG. 6.1 – Réseau bayésien à trois noeuds

A	$Pr[A]$
a_1	0.4
a_2	0.6

FIG. 6.2 – $Pr[A]$

	A	
B	a_1	a_2
b_1	0.6	0.2
b_2	0.4	0.8

FIG. 6.3 – $Pr[B|A]$

Pour obtenir la valeur de la meilleure configuration (α), on effectue le calcul suivant (comme expliqué dans le chapitre 3) :

$$\alpha = max_A Pr[A].max_B Pr[B|A].max_C Pr[C|B]$$

Le résultat de $Pr[B|A].max_C Pr[C|B]$ est donné en 6.5. En maximisant cette table en B, on obtient ($a_1 = 0.48, a_2 = 0.56$). En multipliant cette table par $Pr[A]$, on obtient ($a_1 = 0.192, a_2 = 0.336$) qui, maximisé en A, donne $\alpha = 0.336$.

	B	
C	b_1	b_2
c_1	0.2	0.7
c_2	0.8	0.3

FIG. 6.4 – $Pr[C|B]$

	A	
B	a_1	a_2
b_1	0.48	0.16
b_2	0.28	0.56

FIG. 6.5 – $Pr[B|A].max_C Pr[C|B]$

Pour retrouver la meilleure configuration, dont la valeur est α, on utilise le raisonnement suivant : puisque le maximum en A a été atteint pour $A = a_2$, regardons la colonne a_2 dans le tableau 6.5 et déterminons l'état de B qui est maximum. La résultat est l'état b_2. Sachant cela, utilisons la table 6.4 et regardons, dans la colonne b_2, l'état de C qui est le maximum. C'est l'état c_1. Par conséquent, la meilleure configuration est $a_2 b_2 c_1$.

Changement des opérateurs "maximum"

Afin de calculer les autres meilleures configurations possibles, une première technique consiste à définir des opérateurs maximum de degrés différents. Alors que l'opérateur maximum classique prend le maximum d'un ensemble de nombres, l'opérateur maximum de degré deux (max_2), renvoie parmi ce même ensemble de nombres, la deuxième meilleure valeur. Mathématiquement, si E est un ensemble de nombres,

$$max^2 E = max(E \backslash (maxE))$$

$$max^i E = max(E \backslash (maxE, max^2 E, ..., max^{i-1} E)) \text{ si } |E| \geq i \text{ et } i \geq 2$$

Puisque nous envisageons cette définition sur un ensemble sans doublons et que les tables de probabilité peuvent contenir des doublons, si une table de probabilité contient plusieurs valeurs identiques, on considère qu'elles seront traitées de manière égale (l'opérateur maximum renvoie tous les valeurs identiques maximales).

Une fois ces opérateurs définis, le principe est d'utiliser la formule de l'algorithme de maximum propagation et de remplacer certains opérateurs maximum classiques par des opérateurs maximum de degré inférieur. L'objectif est d'utiliser des combinaisons d'opérateurs maximum de différents degrés dans la formule.

Si, par exemple, c'est la deuxième meilleure configuration qui est recherchée, pour l'obtenir, on remplace un des maxima de degré 1 par un maximum de degré 2.
Toutefois, pour obtenir la deuxième meilleure configuration, il n'est pas possible de connaître sans calcul à quelle position il faut mettre ce maximum de degré 2, .

Dans l'exemple ci-dessus, il n'est pas possible, sans effectuer le calcul, de savoir laquelle des trois formules donnera la deuxième meilleure configuration :

$$\beta_1 = max_A^2 Pr[A].max_B Pr[B|A].max_C Pr[C|B]$$

$$\beta_2 = max_A Pr[A].max_B^2 Pr[B|A].max_C Pr[C|B]$$

$$\beta_3 = max_A Pr[A].max_B Pr[B|A].max_C^2 Pr[C|B]$$

Dès lors, il faut calculer la valeur de ces trois configurations et prendre la configuration qui a la meilleure valeur. En effet, il semble évident que la deuxième meilleure configuration ne sera pas donnée par une des formules suivantes :

$$\gamma = max_A^2 Pr[A].max_B^2 Pr[B|A].max_C Pr[C|B]$$

$$\delta = max_A Pr[A].max_B Pr[B|A].max_C^3 Pr[C|B]$$

Ces deux formules ne peuvent pas donner un meilleur résultat que le maximum des trois formules β. La raison est que pour chacune de ces deux formules, il est possible de trouver une formule dans les formules β qui la "*couvre*", c'est à dire qui donne de toute manière certaine un meilleur résultat. La formule γ donnera un moins bon résultat que β_1. Elle est couverte par cette dernière. Quant à δ, elle est couverte par β_3.
Puisque c'est la formule β donnant la valeur maximum qui est choisie, il est par conséquent certain que ni γ ni γ ne seront prises, considérant la transitivité de l'opérateur maximum.

Formellement,

- Le **rang** d'une configuration est le numéro de cette configuration dans le classement de tous les configurations possibles, de la meilleure à la moins bonne (exemple : la configuration ayant le rang deux est considérée comme la deuxième meilleure configuration).
- Une **combinaison d'opérateurs maximum** est une suite $max^{i_1} max^{i_2} ... max^{i_n}$ d'opérateurs maximum de rang différents. Elle est écrite $combi\_max(i_1, i_2, ..., i_n)$.
- Une combinaison $combi\_max(i_1, i_2, ..., i_n)$ est **couverte** par une combinaison de même taille $combi\_max(j_1, j_2, ..., j_n) \Leftrightarrow \forall 1 \leq k \leq n,\ i_k \leq j_k$.

Remarque : Le signe "$\leq$" est exprimé en terme de rang. Par conséquent, le rang 1 étant meilleur que le rang 2, on a $2 \leq 1$.

Ces définitions vont permettre de développer une procédure qui, pour chaque rang de configuration recherchée (deuxième meilleure configuration, troisième meilleure configuration,...), génère toutes les combinaisons d'opérateurs maximum qui pourraient donner la configuration ayant le rang recherché, en excluant les combinaisons qui ne la donneraient pas. Chaque combinaison générée est alors calculée et c'est la combinaison fournissant le meilleur résultat, qui fournit cette

configuration.

L'enjeu est de générer le moins possible de combinaisons afin de minimiser les calculs. En effet, l'utilisateur ne demande pas forcément un classement de toutes les configurations possibles et imaginables mais peut se contenter des x meilleures solutions. Calculer une configuration est un calcul considérable, en particulier si les tables de probabilité sont de grande taille. Ce qui explique le soucis de minimiser au mieux les calculs.

La procédure fonctionne de manière récursive. Elle utilise d'une part la combinaison d'opérateurs maximum utilisée pour la configuration de rang $i-1$ (*combi_choisie*), et, d'autre part, la liste contenant les combinaisons d'opérateurs maximum qui avaient été évaluées pour obtenir cette configuration de rang $i-1$ mais qui avaient été exclues après comparaison (*liste_combi*). A partir de cela, elle calcule la combinaison d'opérateurs maximum qui permet d'obtenir la configuration de rang i. Cette procédure commence à partir du moment où la meilleure configuration est trouvée (la configuration de rang un) et génère le nombre de configurations demandé ($NombreConfigurationsAGenerer$).

Elle est détaillée dans l'algorithme défini ci-après, dans le cas du réseau à trois noeuds ayant chacun au maximum trois états. Ceci implique que seuls trois opérateurs maximum sont utilisés. La procédure est générique, ce choix n'est fait que pour faciliter la représentation.

Algorithm 5 Procédure permettant de calculer les configuration de rang inférieur à un.

Initialiser *combi_choisie* à (1,1,1)
Initialiser *liste_combi* à la liste vide
Initialiser i à 0
while $i < NombreConfigurationsAGenerer$ **do**
- A partir de *combi_choisie*, générer les combinaisons obtenues en augmentant le(s) élément(s) de cette combinaison ayant le rang le plus bas de 1 (pour chaque élément augmenté, une nouvelle combinaison). Les ajouter à *liste_combi_temp*.
- A partir de *combi_choisie*, générer les combinaisons obtenues en augmentant le(s) élément(s) de cette combinaison ayant le rang le plus haut de 1 (pour chaque élément augmenté, une nouvelle combinaison). Les ajouter à *liste_combi_temp*.
- Pour chaque cas dans *liste_combi_temp*, vérifier qu'il n'est pas couvert par un des cas présents dans *liste_combi*. Si c'est le cas, le supprimer.
- Pour chaque cas dans *liste_combi_temp*, vérifier qu'aucun élément de la combinaison ne dépasse la limite du nombre d'états de la variable qu'il concerne. Si c'est le cas, le supprimer.
- Ajouter les éléments de *liste_combi_temp* à *liste_combi*.
- Pour chaque combinaison d'opérateurs maximum de *liste_combi*, effectuer le calcul utilisant cette combinaison d'opérateurs.
- *combi_choisie* $\leftarrow$ la combinaison donnant le meilleur résultat.
- Supprimer cette combinaison de *liste_combi*.
- $i \leftarrow i+1$

end while

Pour mieux comprendre son principe, montrons ce que l'algorithme présenté génère lors des trois premières étapes :

Initialisation :

combi_choisie = combi(1,1,1) *liste_combi* = []

Etape 1 :

Combinaisons générées en augmentant le rang le plus bas : (1,1,2), (1,2,1), (2,1,1)
Combinaisons générées en augmentant le rang le plus haut : identiques aux précédentes
Combinaisons exclues parce qu'elles sont couvertes par des cas déjà présents dans *liste_combi* : aucune
Combinaisons exclues parce qu'elles dépassent une limite de nombre d'états : aucune

Liste *liste_combi* **actualisée** : [(1,1,2), (1,2,1), (2,1,1)]
On suppose que la combinaison suivante a donné le meilleur résultat (*combi_choisie*) : (1,2,1)
Liste *liste_combi* **actualisée** : [(1,1,2), (2,1,1)]

Etape 2 :

Combinaisons générées en augmentant le rang le plus bas de *combi_choisie* : (1,3,1)
Combinaisons générées en augmentant le rang le plus haut de *combi_choisie* : (2,2,1), (1,2,2)
Combinaisons exclues parce qu'elles sont couvertes par des cas déjà présents dans *liste_combi* : (2,2,1) qui couvre (2,1,1), (1,2,2) qui couvre (1,1,2)
Combinaisons exclues parce qu'elles dépassent une limite de nombre d'états : aucune

Liste *liste_combi* **actualisée** : [(1,1,2), (2,1,1), (1,3,1)]
On suppose que la combinaison suivante a donné le meilleur résultat (*combi_choisie***)** : (2,1,1)
Liste *liste_combi* **actualisée** : [(1,1,2), (1,3,1)]

Etape 3 :

Combinaisons générées en augmentant le rang le plus bas de *combi_choisie* : (3,1,1)
Combinaisons générées en augmentant le rang le plus haut de *combi_choisie* : (2,2,1), (2,1,2)
Combinaisons exclues parce qu'elles sont couvertes par des cas déjà présents dans *liste_combi* : (2,1,2) qui couvre (1,1,2)
Combinaisons exclues parce qu'elles dépassent une limite de nombre d'états : aucune

Liste *liste_combi* **actualisée** : [(1,1,2), (2,1,2), (1,3,1), (3,1,1)]
On suppose que la combinaison suivante a donné le meilleur résultat (*combi_choisie***)** : (3,1,1)
Liste *liste_combi* **actualisée** : [(1,1,2), (2,1,2), (1,3,1)]

Le procédé peut ainsi se poursuivre en épuisant toutes les combinaisons d'opérateurs maximum possibles et par conséquent, toutes les configurations calculables. Notons qu'à l'étape quatre, des combinaisons générées commencent à être exclues parce qu'elles dépassent la limite du nombre d'états (la combinaison (1,4,1) n'est pas possible parce que le second noeud du réseau n'a que trois états, donc quatre valeurs).

Reprenons à présent l'exemple du début de cette section, dans lequel la meilleure configurationd d'un réseau comptant trois noeuds a été calculée. Pour obtenir la deuxième meilleure configuration, les formules à évaluer, basées sur les combinaisons d'opérateurs maximum données par la précédente procédure, sont les suivantes :

$$\beta_1 = max_A^2 Pr[A].max_B Pr[B|A].max_C Pr[C|B] = 0.192$$

$$\beta_2 = max_A Pr[A].max_B^2 Pr[B|A].max_C Pr[C|B] = 0.112$$

$$\beta_3 = max_A Pr[A].max_B Pr[B|A].max_C^2 Pr[C|B] = 0.144$$

Montrons à titre d'exemple comment β_2 est calculé :

Le résultat de $Pr[B|A].max_C Pr[C|B]$ est donné en 6.6. En maximisant cette table en B, avec un opérateur maximum de degré deux, on obtient $(a_1 = 0.28, a_2 = 0.016)$. En multipliant cette table par $Pr[A]$, on obtient $(a_1 = 0.112, a_2 = 0.096)$ qui, maximisé en A, donne $\alpha = 0.112$.

La deuxième meilleure configuration est donc celle donnée par la formule β_1, dont la valeur est 0.192. On utilise le raisonnement suivant : puisque le maximum de rang 2 en A a été atteint pour $A = a_1$, regardons la colonne a_1 dans le tableau 6.6 et déterminons l'état de B qui est maximum. Le résultat est l'état b_2. Sachant cela, utilisons la table 6.4 et regardons, dans la colonne b_2, l'état de C qui est le maximum. C'est l'état c_1. Par conséquent, la meilleure configuration est

	A	
B	a_1	a_2
b_1	0.48	0.16
b_2	0.28	0.56

FIG. 6.6 – $Pr[B|A].max_C Pr[C|B]$

$a_1 b_2 c_1$.

Le principe utilisé par cette procédure a pour principal avantage sa simplicité d'appréhension. Cependant, au niveau de sa complexité algorithmique, il n'est pas des plus efficaces puisqu'il fait appel plusieurs fois à l'algorithme de propapagation de suite. D'autres techniques existent, qui travaillent en bénéficiant de la structure des arbres de jonction.

Autres techniques

Un deuxième type de techniques permet d'obtenir les x meilleures configurations, comme c'est le cas dans [5].

L'auteur, Nilsson justifie sa procédure en prouvant qu'utiliser l'algorithme de Dawid ne permet d'identifier directement que les trois configurations les plus probables et qu'au delà de trois, il n'est plus possible de les trouver directement.

Ceci correspond aux résultats de l'algorithme précédemment décrit. En effet, celui-ci ne travaille plus directement sur l'arbre de jonction et ses potentiels mais utilise des combinaisons d'opérateurs maximum, calculées de manière externe.

L'algorithme de Nilsson (*Max-flow propagation*) se déroule en trois étapes, le partionnement, le calcul des candidats et la sélection.

- **Partitionnement** : L'ensemble de toutes les configurations possibles est partitionné en sous ensembles. Le partionnement doit respecter deux propriétés :
  - D'une part, le partionnement doit être fait de telle manière qu'il est possible de calculer la meilleure configuration sur chacun des sous ensembles générés.
  - D'autre part, le partionnement doit générer le minimum de sous ensembles, puisqu'il faut comparer la meilleure configuration de chaque sous ensemble (l'efficacité exige un nombre minimal de comparaisons).

- **Calcul des candidats** : Dans chaque sous ensemble généré par l'étape précédente, la meilleure configuration est calculée.

- **Sélection** : Une comparaison est faite entre les résultats donnés pour chaque sous ensemble dans l'étape précédente et le meilleur candidat est choisi.

La version complète de cet algorithme travaille directement sur l'arbre de jonction afin d'être le plus efficace possible. Nous ne présentons ici que la version simplifiée de cet algorithme, en expliquant son principe général sur un réseau bayésien.

Supposons que les variables de l'univers U des variables du réseau peuvent être énumérées dans un certain ordre, soit $X_1, X_2, ..., X_n$. Une configuration possible des variables du réseau, est notée $x_1, x_2, ..., x_n$ où chaque x_i représente l'état choisi pour la variable X_i. Notons encore H, l'ensemble des configurations possibles des variables du réseau.
Puisque une configuration est une simple combinaison d'états pour toutes les variables, on peut écrire la meilleure configuration, trouvée par l'algorithme de maximum propagation, comme $x^1 = \{x_1^1, x_2^1, ...x_n^1\}$.

Comme la deuxième meilleure configuration doit différer de la première meilleure configuration pour au moins une variable (avoir un état différent pour au moins une variable), les sous-ensembles suivants peuvent être générés à partir de l'ensemble $\{H/x^1\}$:

$$1 : \{x \in H : x_1 \neq x_1^1\}$$
$$\dots$$
$$i : \{x \in H : x_1 = x_1^1, ..., x_{i-1} = x_{i-1}^1, x_i \neq x_i^1\}$$
$$\dots$$
$$n : \{x \in H : x_1 = x_1^1, ..., x_{n-1} = x_{n-1}^1, x_n \neq x_n^1\}$$

Il faut maintenant calculer les candidats, c'est à dire, pour chaque sous-ensemble, trouver la meilleure configuration. L'idée est d'incorporer au réseau les évidences liées au sous ensemble étudié, et de réappliquer l'algorithme de maximum propagation sur le réseau modifié.
Pour trouver le candidat de l'ensemble i, par exemple, l'évidence $\{X_1 = x_1^1, ..., x_{i-1} = x_{i-1}^1, x_i \neq x_i^1\}$ est incorporée au réseau. L'algorithme de maximum propagation est alors utilisé et la meilleure configuration trouvée sera la configuration candidate pour cet ensemble.

Une fois que tous les candidats ont été trouvé, on compare leur probabilité. La configuration candidate qui a la probabilité la plus importante est la deuxième meilleure configuration.

Pour trouver la configuration suivante, soit la troisième meilleure configuration, on réutilise le même principe que précédemment. La seule différence est que l'ensemble à partionner est maintenant $H\{x^1, x^2\}$.

Pour pouvoir partitionner cet ensemble, il faut premièrement raffiner le partitionnement fait à l'étape précédente, en considérant que si par exemple, la deuxième meilleure configuration x^2 faisait partie du $i$ème sous ensemble, alors ce sous-ensemble doit être lui-même partitionné en excluant x^2. Le partitionnement du $i$ème sous ensemble donne ceci :

$$\{x \in H : x_1 = x_1^2, ..., x_{i-1} = x_{i-1}^2, x_i \notin \{x_i^1, x_i^2\}\}$$
$$\dots$$
$$\{x \in H : x_1 = x_1^2, ..., x_i = x_i^2, x_{i+1} \neq x_{i+1}^2\}$$
$$\dots$$
$$\{x \in H : x_1 = x_1^2, ..., x_{n-1} = x_{n-1}^2, x_n \neq x_n^2\}$$

Ces nouveaux sous-ensembles sont combinés aux sous-ensembles générés à l'étape précédente, sauf le $i$ème. Les candidats sont alors cherchés dans cette combinaison de sous-ensembles et la troisième meilleure configuration peut alors être trouvée. Le même principe est appliqué pour identifier les suivantes.

De cette manière, les x meilleures configurations peuvent être trouvées. Mais à nouveau, cette technique telle qu'elle a été présentée jusqu'ici, implique un très grand nombre de flux de maximum propagation dans l'arbre de jonction. Nilsson a donc complété ce principe et a défini un algorithme plus précis qui, profitant au mieux de la structure d'un arbre de jonction, trouve efficacement les x meilleures configuration d'un réseau bayésien. Cet algorithme effectue des partionnements de l'arbre de jonction lui-même et permet le calcul des potentiels des sous ensembles de cliques générés.

Cas de la reconnaissance

Dans le cadre de l'application à la reconnaissance de formules mathématiques manuscrites, permettre au réseau de donner facilement - en changeant simplement le sens de certains opérateurs utilisés dans le calcul de la maximum propagation - les x meilleures formules mathématiques les plus probables et admissibles, renforce la pertinence de son utilisation.

En effet, il est fort possible qu'à cause de caractères mal lus, le réseau écarte une possibilité dont la probabilité était très peu différente de celle de la meilleure configuration. D'autant plus qu'en introduisant la notion de mauvais utilisateurs, on amène le réseau à accepter certaines formules fausses, ce qui accroît le risque d'erreur.
Par conséquent, permettre à l'utilisateur de voir non pas une seule formule mais une série de formules ayant une très bonne probabilité, lui évite dans certains cas de perdre ce qu'il avait écrit à l'origine.

Il serait même imaginable, lorsque l'utilisateur confirme que la formule qu'il a écrite n'était pas la première affichée, qu'un système d'apprentissage des erreurs et des formules écartées offre la possibilité de revoir certains paramètres du réseau.

6.2.2 Classer les résultats en fonction de paramètres externes

La section précédente a examiné la problématique du classement de résultats dans le cas où le modèle utilisé est un réseau bayésien. En travaillant à partir d'un algorithme existant, l'algorithme de maximum propagation permettait d'obtenir la meilleure configuration d'un réseau et poursuivait l'objectif d'affiner cet algorithme afin de parvenir à un classement de plusieurs résultats.

Bien qu'il s'agisse toujours du domaine du classement de résultats, cette section considère une situation où un outil, quel qu'il soit, renvoie une liste de résultats possibles de manière non ordonnée, non classée. L'outil ne permet cependant pas d'aider un système ou un utilisateur à reconnaître que certains de ces résultats sont plus probables que d'autres. Un réseau bayésien pourrait être utilisé à cet escient. C'est ce que démontre le raisonnement qui suit.

Deux cas sont envisagés. Le premier cas considère un système qui renvoie une liste de résultats distincts dont la différence est justifiée par l'utilisation de règles ou de procédures particulières par le système. C'est le cas où l'on est capable de prédire, sans difficulté, ce qui a généré chaque résultat. Cette situation permet également de déterminer, la liste de toutes les réponses que le système pourrait fournir, sans l'utiliser. Il autorise donc l'étiquetage de chaque résultat produit par la suite d'opérations qui l'ont généré.

Si l'application de reconnaissance de formules manuscrites est prise en exemple, ce premier cas concerne les corrections d'erreurs de précédence. En effet, pour chaque erreur considérée par le système, celui-ci propose une liste de corrections. De plus, tout utilisateur est capable de déterminer pourquoi ces différentes corrections on été proposées. Les types de corrections sont préconfigurés dans le système.

Dans ce cas, un paramètre très simple peut être utilisé, pour permettre de classer les résultats par ordre de préférence. Il s'agit simplement de la fréquence. A chaque utilisation du système, le système enregistre quel résultat a été choisi par l'utilisateur dans la liste des résultats fournis. Ensuite, lors de l'utilisation suivante, lorsque le système doit classer une liste de résultats, il calcule d'abord la fréquence de chaque règle ou procédure utilisée par le système pour produire chaque résultat, sur base des choix précédents des utilisateurs. Cela fait, il classe les résultats selon la fréquence des règles qui les ont produits.

Dans l'exemple de la reconnaissance, supposons que, lorsque l'utilisateur fait l'erreur de ne mettre aucune opérande sous le signe représentant une racine carrée, les deux corrections proposées soient d'ajouter le signe ϵ sous la racine ou de la supprimer. Si l'utilisateur a d'avantage utilisé par le passé la correction "supprimer la racine" que l'ajout du signe ϵ, alors le système classe les résultats en prenant en compte cette préférence.

Utiliser un réseau bayésien n'est pas opportun pour traiter ce type de cas. La solution recquiert uniquement un comptage de fréquence.

Le second cas considère une situation identique à celle du premier cas mais où le paramètre de fréquence ne suffit pas à l'utilisateur, c'est à dire une situation où il est impossible d'utiliser la fréquence. Cette situation est celle où, selon ce qui a été donné en entrée au système, des solutions vont être proposées par un algorithme qui base tout son raisonnement sur ce qui a été donné en entrée et quelques paramètres externes. Une procédure est préconfigurée dans le système, mais elle ne signale pas clairement ce qui peut être produit. D'une entrée à l'autre, le nombre de résultats possibles sera tout à fait divergents.

Une nouvelle fois dans le cas de la reconnaissance, il s'agit de la correction d'erreurs non contextuelles, comme l'ajout de parenthèses de droite manquantes. Le système va générer toutes les formules envisageables en évaluant le rajout des parenthèses manquantes à tous les endroits possibles. Cette génération de formules est faite par un seul et même algorithme, en fonction de la formule lue. Dès lors, il est impossible de donner une fréquence aux règles qui ont produit chaque résultat dans le mesure où une seule et unique règle a été employée.

L'idée consiste à trouver d'autres paramètres que la fréquence et de déterminer à partir de ces paramètres des situations types qui seront pondérées par la fréquence de leur arrivée. Un réseau bayésien est alors utilisé pour joindre ces différents paramètres, en considérant leur poids. A partir de là, un score est attribué à chaque résultat. Le score permet ensuite de les classer. Une fois que l'utilisateur a choisi le résultat qui lui convient, les paramètres du réseau bayésien sont mis à jour.

Dans la figure 6.7, se trouve la forme générale du réseau bayésien utilisé pour joindre les différents paramètres en fonction de leur poids.

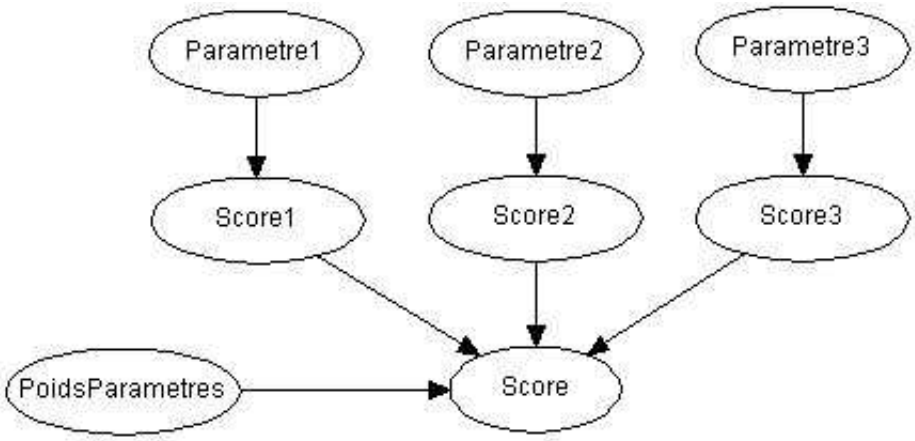


FIG. 6.7 – Réseau bayésien pour l'évaluation de scores

Le noeud $Parametre_i$ sert à intégrer la valeur du paramètre i. Si le paramètre est par exemple la taille de la donnée entrée, et que cette taille peut être respectivement petite, moyenne ou grande, alors le noeud représentant ce paramètre possède trois états correspondant à ces tailles types et la probabilité a priori de ce paramètre est la mesure de la taille de la donnée effectivement entrée (si la donnée est très grande, on aura une probabilité maximale pour l'état "Grande").

Le noeud $Score_i$ transforme la valeur du paramètre i en score (les états de ce noeud sont "Bon" et "Mauvais"). Elle considère d'une part la valeur du paramètre i ($Pr[Parametre_i]$) et d'autre part, la préférence de l'utilisateur pour ce paramètre ($Pr[Score_i|Parametre_i]$). Par préférence pour un paramètre, on entend ce que l'utilisateur considère comme bon ou mauvais. Dans l'exemple de la taille, la question posée par ce noeud est "Est-ce que l'utilisateur considère comme bon ou mauvais le fait que la donnée soit grande".

Avec ces deux noeuds, il est possible de déterminer la façon dont l'utilisateur conçoit un paramètre et la valeur du paramètre pour le cas effectivement entré.
Le score donné est un peu atypique puisqu'il est sous la forme d'une probabilité. En pratique, on considère que la valeur réelle du score est la probabilité $Pr[Score_i = Bon]$.

La probabilité $Pr[Score_i|Parametre_i]$ est apprise à partir des cas précédemment soumis au réseau, qui sont enregistrés dans une base de donnée. Sur base des choix précédents effectués par l'utilisateur à partir des classements de résultats qui lui ont été soumis, un calcul de fréquence permet d'évaluer la manière dont l'utilisateur considère un paramètre. Si l'utilisateur a fréquemment choisi une solution pour laquelle la taille de la donnée entrée est envisagée comme "grande", alors le réseau, au fil des utilisations, considérera que quand la valeur du paramètre "taille" est grande, le score de ce paramètre est "bon". En pratique, on utilise le mécanisme d'apprentissage défini pour les réseaux bayésiens comme par exemple l'algorithme d'expectation maximisation.

Le noeud *Score* est là pour joindre les différents scores de paramètres et donner un score global. Par défaut, il ne fait que fournir un poids égal à chaque score. Si le premier paramètre est à "bon" et les deux autres sont à "mauvais", alors la probabilité que le score global soit bon vaut un tiers et la probabilité qu'il soit mauvais, deux tiers ($Pr[Score|Score_1, ...Score_n]$).

Il est cependant possible de changer cette considération par défaut et donner un poids différent à chaque score et, donc, à chaque paramètre. Le principe utilisé pour ce faire est le principe d'*expert disagreement*, concept classique dans la construction des réseau bayésien.

Il s'agit d'exprimer explicitement le fait que selon l'expert qui a construit le réseau, les paramètres peuvent varier. L'intérêt dans le cas du réseau précédemment présenté, est qu'on peut donner des valeur à la table $Pr[Score|Score_1, ...Score_n]$ variant d'après les préférences de poids ($Pr[Score|Poids, Score_1, ...Score_n]$). L'expert par défaut donnerait un poids égal à chaque score ($Pr[Score|Poids = Defaut, Score_1, ...Score_n]$) alors que l'expert qui préfére le premier paramètre à tous les autres, augmenterait son poids.
La seule difficulté réside dans la nécessité de devoir préciser le nombre d'experts et de tables différentes avant la construction du réseau. Proposer à l'utilisateur de choisir deux ou trois modes avant l'utilisation du réseau est une suggestion adéquate.

La reconnaissance de formules manuscrites sert ici encore d'exemple. Dans le cas des propositions d'ajout de parenthèses de droite manquantes faites par l'interprêteur, il est évident que la fréquence des règles utilisées ne peut pas être calculée. Si la phrase écrite par l'utilisateur est "$(a + (b + (c$", quatre solutions de correction sont possibles :

- (a)+(b)+(c)
- (a)+(b+(c))
- (a+(b)+(c))
- (a+(b+(c)))

On ne peut, sur base de ces quatre propositions, donner une fréquence à chacune des quatre formules car elles n'ont été générées que par simple déplacement de parenthèses de droite aux endroits possibles.
Cependant, il est possible de déterminer des paramètres qui pourraient voir l'une des solutions préférée aux autres.

Les paramètres suivants sont pris en considération : concentration des parenthèses en fin de formule (les parenthèses rajoutées sont elles toutes ajoutées en fin de phrase ?), écart en nombre de caractères entre les parenthèses formant un couple (l'écart est-il maximal ?) et voisinage des parenthèses de droite ajoutées (sont-elles toutes l'une à la suite de l'autre ?). Pour chaque paramètre, on impose ensuite des valeurs discrètes à prendre. Dans le cas de l'exemple, ils ne peuvent tous endosser que deux valeurs : oui ou non, soit les réponses aux questions posées par ces paramètres.

Pour chaque paramètre, une formule est définie. Elle détermine la probabilité, pour chaque paramètre, de correspondre à une réponse affirmative.

Sachant que :

- n est le nombre de parenthèses de droite rajoutées (manquantes).
- $TailleFormule$ est la taille de la formule (nombre de caractère dans la formule, en comptant les parenthèses rajoutées).
- $Position$ est la position d'une parenthèse de droite rajoutée.
- $Distance$ est la distance entre une parenthèse de droite rajoutée et la parenthèse de gauche liée (nombre de caractères entre les deux).
- $Voisinage$ est le nombre d'autres parenthèses de droite rajoutées qui entourent une parenthèse de droite rajoutée.

Les formules suivantes sont utilisées :

- Concentration en fin de formule = $\frac{\sum_i^{i=n} \frac{Position_i}{TailleFormule}}{n}$
- Ecart moyen entre les couples recréés = $\frac{\sum_i^{i=n} \frac{Distance_i}{TailleFormule-2}}{n}$
- Voisinage des parenthèses rajoutées = $\frac{\sum_i^{i=n} \frac{Voisinage_i+1}{n}}{n}$

Remarque : Dans le calcul de la variable $Distance$, il faut pouvoir retrouver le correspondant gauche d'une parenthèse de droite rajoutée. Pour cela, il faut lire la formule de droite à gauche en partant de la parenthèse rajoutée concernée et dès qu'une parenthèse de gauche est lue, sans qu'une autre parenthèse de droite n'ait été lue entre temps, la parenthèse de gauche peut être considérée comme le correspondant de la parenthèse rajoutée.

Une fois les formules définies, elles doivent être appliquées aux quatre formules corrigées :

- (a)+(b)+(c)
  - Fin de phrase = (0.27+0.64+1)/3 = 0.64
  - Ecart = (0.11+0.11+0.11)/3 = 0.11
  - Voisinage = (0.33+0.33+0.33)/3 = 0.33

- (a)+(b+(c))
  - Fin de phrase = (0.27+0.91+1)/3 = 0.73
  - Ecart = (0.11+0.56+0.11)/3 = 0.26
  - Voisinage = (0.33+0.67+0.67)/3 = 0.56
- (a+(b)+(c))
  - Fin de phrase = (0.55+0.91+0.1)/3 = 0.82
  - Ecart = (1+0.11+0.11)/3 = 0.41
  - Voisinage = (0.33+0.67+0.67)/3 = 0.56
- (a+(b+(c)))
  - Fin de phrase = (0.82+0.91+1)/3 = 0.91
  - Ecart = (1+0.56+0.11)/3 = 0.56
  - Voisinage = (1+1+1)/3 = 1

On utilise alors le réseau bayésien de la figure 6.8, en choisissant le poids par défaut (les trois paramètres ont la même importance), avec la table 6.9 pour les probabilités $Pr[ScoreFinPhrase|FinPhrase]$, $Pr[ScoreDistance|Distance]$ et $Pr[ScoreVoisinage|Voisinage]$; c'est à dire des tables qui en général, indiquent que l'utilisateur préfère une réponse "oui" aux questions posées par les paramètres).

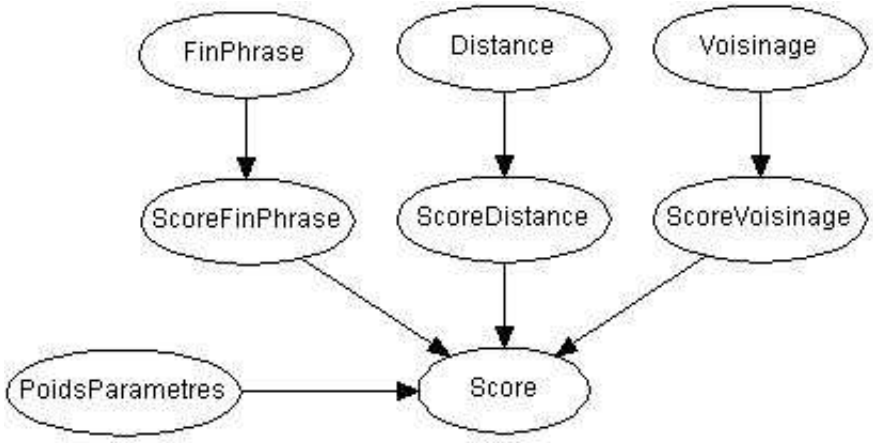


FIG. 6.8 – Réseau bayésien pour l'évaluation des scores donnés aux formules mathématiques corrigées

	$FinPhrase$	
$ScoreFinPhrase$	Oui	Non
Bon	0.9	0.01
$Mauvais$	0.1	0.99

FIG. 6.9 – $Pr[ScoreFinPhrase|FinPhrase]$

En utilisant ce réseau, on obtient les scores suivants :

- (a)+(b)+(c) : 0.33
- (a)+(b+(c)) : 0.47
- (a+(b)+(c)) : 0.54
- (a+(b+(c))) : 0.74

Cet exemple chiffré ne donne pas nécessairement l'impression que passer par le réseau bayésien est avantageux. Il en effet est possible de déterminer quelle est la meilleure phrase en comparant manuellement les scores pour chaque paramètre. Toutefois, si le nombre de possibilités est étendu, si la préférence qu'un utilisateur a pour un paramètre change, si le nombre de paramètres est important ou encore si chaque paramètre a un poids différent, le réseau bayésien permet de simplifier la complexité et d'avoir un raisonnement multicritère efficace.

Le poids des paramètres s'avère très utile. Il est relevant, dans l'exemple précédent, que le score donné par chaque paramètre évolue de manière relativement différente. Alors que les chiffres donnés par le voisinage évoluent rapidement jusqu'à 1, ceux de la distance n'approchent pas aussi vite 1. En outre, le paramètre voisinage et le paramètre fin de phrase sont dépendants. Affecter des poids pour respecter les préférences de l'utilisateur mais aussi pour combattre ce type de biais, est primordial.

6.2.3 Combiner plusieurs classements de résultats

Une dernière question reste intéressante à soulever, dans le cadre de classements de résultats. Même si elle ne concerne plus directement les réseaux bayésiens, elle mérite d'être posée.

Cette question touche à la façon de combiner deux classements. Supposons qu'un classement de réponses soit fourni par système quelconque et qu'ensuite, un autre système vienne donner un classement de sous possibilités pour chaque réponse de base. Que peut-on faire pour obtenir un seul classement global ?

La première solution est de considérer que le premier classement est d'un plus haut niveau que le second. On laisse donc le premier classement tel qu'il est et, pour chaque réponse, on offre la sous possibilité la plus probable, en laissant à l'utilisateur le soin de demander des choix supplémentaires. On considère le premier classement comme prioritaire au second. En d'autres mots, si le premier système donne les réponses classées $(x_1, x_2, ..., x_n)$ et le second donne pour chaque x_i, un classement de la forme $(x_i^1, ...x_i^m)$, le classement global sera $(x_1^1, ..., x_1^m; x_2^1, ..., x_2^m; ...; x_n^1, ..., x_n^m)$.

Une deuxième solution est d'envisager les deux classements sur le même pied et de calculer la probabilité jointe des deux classements. Si le premier classement était classé par la probabilité $Pr[X]$ et le second, pour chaque réponse du premier, par la probabilité $Pr[Y]$, on classe maintenant toutes les réponses de tous les seconds classements, de manière confondue, avec la probabilité $Pr[X, Y]$. Cependant, pour pouvoir calculer cette probabilité jointe comme une multiplication des deux probabilités, il faut que ces deux probabilités soient indépendantes.

En outre, considérer la probabilité jointe ne doit se faire que si cela a du sens.

Dans l'application à la reconnaissance manuscrite, on voudrait pouvoir faire un classement joint sur les différentes phrases admissibles les plus probables (fournies par l'algorithme de maximum propagation affiné) et les différentes corrections possibles de chacune de ces phrases (fournies de manière classées par un réseau bayésien évaluant différents paramètres de préférence).

Toutefois, même si la probabilité d'une correction est indépendante de la probabilité de la phrase qui a été corrigée, cela n'a pas de sens de multiplier les deux types de probabilités pour obtenir la probabilité jointe et faire un classement confondu. Cela n'a pas de sens parce que d'une part, c'est la qualité de l'écriture qui est représentée par probabilités et d'autre part, c'est la préférence de l'utilisateur en matière de correction qui est estimée.
Il serait absurde de faire un classement confondu de toutes les corrections de toutes les phrases probables, parce que les phrases ne doivent être corrigées qu'une fois que l'interprêteur a confirmé qu'il s'agit bien de la phrase que l'utilisateur désirait écrire.

6.3 Combiner les différents algorithmes de propagation

6.3.1 Motivations

Dans de nombreuses situations, pouvoir combiner les algorithmes de somme et de maximum propagation n'est pas dénué d'intérêt.
Combiner ces deux algorithmes revient à vouloir résoudre le problème de l'abduction partielle dans les réseaux bayésiens. Cela consiste à chercher la meilleure configuration d'une partie seulement des variables d'un réseau.

En effet, pour obtenir la meilleure configuration d'un sous ensemble de variables, il faudrait dans un premier temps appliquer l'algorithme de somme propagation sur l'ensemble des variables en marginalisant toutes les variables qui ne se trouvent pas dans le sous ensemble et, dans un second temps, appliquer un algorithme de maximum propagation sur les variables restantes.

Dans le cadre de la reconnaissance de formules mathématiques manuscrites, on a vu l'utilité d'une telle option lors de la prise en compte du comportement de l'utilisateur. L'idée était de construire un réseau contenant les variables représentant les différents caractères de la formule lue et d'y ajouter un noeud "*TypeUtilisateur*", parent de tous les autres noeuds, dont le but est de déterminer la qualité de l'utilisateur, sur base de ce qu'il a écrit.

Toutefois, la difficulté était que, pour trouver la formule la plus probable, il fallait appliquer un algorithme de maximum propagation sur tous les noeuds sauf le noeud "TypeUtilisateur", sans quoi ce n'est plus la formule la plus probable qui serait donnée par le réseau mais la formule la plus probable pour le type d'utilisateur le plus probable.
N'ayant pas l'opportunité de combiner les algorithmes de propagation de type somme et maximum, nous avons été forcés de définir deux réseaux différents, l'un sans le noeud "*TypeUtilisateur*" pour trouver la phrase la plus probable avec la maximum propagation et l'autre avec le noeud, pour trouver le type de l'utilisateur avec la somme propagation.

A présent, nous allons montrer que nous sommes en mesure de combiner ces deux algorithmes de propagation et de trouver l'explication la plus probable d'une partie des variables seulement, ce qui, dans le cas de la reconnaisse, éviterait la complication et le gaspillage de ressources engendrés par l'utilisation de deux réseaux différents.

6.3.2 Utilisation séquentielle des deux algorithmes

A première vue, combiner ces deux algorithmes semble se résumer à les utiliser l'un à la suite de l'autre. Cependant, les choses ne sont pas aussi simple. A cause du comportement non commutatif des opérateurs maximum et somme, l'opérateur somme ne peut pas être appliqué sur un potentiel qui a été obtenu par un opérateur maximum.

Observons pour le comprendre le cas du réseau bayésien de la figure 6.10 où les noeuds Car représentant les caractères d'une phrase de taille trois et $TypeUtilisateur$ le type d'utilisateur.

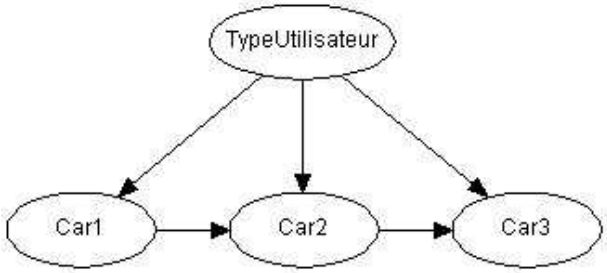


FIG. 6.10 – Exemple de réseau bayésien

Nous savons que la probabilité de l'univers des variables de ce réseau peut être exprimée de la façon suivante :

$$\begin{gathered} Pr[Car1, Car2, Car3, TypeUtilisateur] \\ = \\ Pr[Car3|Car2, TypeUtilisateur].Pr[Car2|Car1, TypeUtilisateur] \\ .Pr[Car1|TypeUtilisateur].Pr[TypeUtilisateur] \end{gathered}$$

Nous désirons obtenir la meilleure configuration des variables Car1, Car2 et Car3, soit obtenir la phrase la plus probable de taille trois. A première vue, il suffirait d'utiliser l'algorithme de somme propagation pour marginaliser la variable TypeUtilisateur :

$$Pr[Car1, Car2, Car3] = \sum_{TypeUtilisateur} Pr[Car1, Car2, Car3, TypeUtilisateur]$$

Ensuite, il ne reste plus qu'à appliquer l'algorithme de maximum propagation sur le résultat, soit en faisant :

$$max_{Car1,Car2,Car3} Pr[Car1, Car2, Car3]$$

Mais comment savoir ce que vaut le potentiel $Pr[Car1, Car2, Car3]$? C'est là que réside l'enjeu fondamental. Il est impossible d'exprimer ce potentiel parce qu'il mélange des variables provenant de plusieurs cliques de l'arbre de jonction de ce réseau (figure 6.11). Pourquoi ne pas essayer alors de travailler sur l'arbre de jonction, en utilisant seulement des potentiels exprimables ?

FIG. 6.11 – Arbre de jonction du réseau bayésien de la figure 6.10

L'arbre de jonction montre que si nous voulons combiner les deux algorithmes à partir de potentiels calculables, il faut d'abord marginaliser $TypeUtilisateur$ de la clique $Car2Car3TypeUtilisateur$ avec l'opérateur somme, puis appliquer l'opérateur maximum sur cette clique.

Ensuite, il faut marginaliser $TypeUtilisateur$ de la clique $Car1Car2TypeUtilisateur$ avec l'opérateur somme et, enfin, appliquer l'opérateur maximum sur le potentiel résultant.

Or, ceci n'est pas possible puisqu'il est interdit d'appliquer un opérateur somme sur un potentiel qui résulte d'un calcul de maximum. La variable $TypeUtilisateur$ a disparu d'une clique mais pas de l'autre et des flux de type maximum sont effectués avant et après. Les opérateurs somme et maximum ne sont pas commutatifs entre eux.

En réalité, appliquer les deux algorithmes l'un à la suite de l'autre ne peut fonctionner que quand le sous-ensemble de variables pour lequel on cherche la meilleure configuration est contenu entièrement dans une clique du réseau ou quand ces variables forment un sous arbre de l'arbre de jonction complet. Dans ce cas, le sous-ensemble de variables peut donc être représenté par un potentiel calculable. Lorsque cette condition n'est pas respectée, il faut donc recourir une autre technique.

6.3.3 Adaptation de l'arbre de jonction

Puisque l'utilisation séquentielle des deux algorithmes ne fonctionne pas de manière générale, à cause de la structure de l'arbre de jonction, des recherches ont été effectuées. Elles cherchent à adapter l'arbre de jonction d'origine d'un réseau bayésien, afin qu'il ne contienne plus que les variables du sous-ensemble pour lequel la meilleure configuration est recherchée.

Dans [6], Xu donne une méthode pour transformer l'arbre original en un seul noeud contenant toutes les variables du sous-ensemble. Le problème de cette approche est que si le sous-ensemble de variables est de grande taille, le calcul du potentiel associé à l'unique noeud du nouvel arbre de jonction devient fastidieux. Plus tard, dans [5], l'auteur a modifié l'algorithme de Xu afin que le nouvel arbre de jonction ne soit pas composé d'un seul noeud contenant toutes les variables du sous-ensemble mais donne plutôt un sous arbre de jonction. Aujourd'hui, la procédure a été détaillée et complétée par certains heuristiques, dans [2].

Par conséquent, l'objectif de la procédure est, à partir de l'arbre de jonction original T et d'un sous-ensemble E de variables, de construire un nouvel arbre de jonction, T_E. Dans [3], elle s'inscrit dans en trois étapes :

1. Identifier le plus petit sous arbre T' de T qui contient les variables de E.

2. Effectuer une somme propagation sur le reste des cliques de T vers T'. De cette manière, T' contient l'information venant du reste des cliques.

3. Tant que T' contient encore des variables $\notin E$, effectuer l'opération de fusion, qui est définie par les actions suivantes :

   (a) Sélectionner deux cliques voisines C_i et C_j dans T'.
   (b) Les remplacer par leur fusion dans un nouveau noeud C_{ij}. Ce nouveau noeud est obtenu à partir de $C^*_{ij} = C_i \cup C_j$, une fois que les variables qui ne sont pas nécessaires pour maintenir la propriété d'intersection courante et qui n'appartiennent pas à E, ont été supprimées de C^*_{ij}.
   (c) Calculer le potentiel du noeud C_{ij} en utilisant la formule :

$$\psi_{C_{ij}} = \sum_{C^*_{ij}/C_{ij}} \frac{\psi_{C_i}.\psi_{C_j}}{\psi S_{ij}}$$

La figure 6.12 montre l'application des trois étapes de la méthode à l'arbre de jonction T (en *(1)*) d'un réseau bayésien à huit variables (A, T, B, E, L, X, D, S). Le sous-ensemble E de variables étudié est (A, T, B). Le plus petit sous arbre T' est montré en *(2)*. Le nouvel arbre issu de l'étape de fusion est l'arbre en *(3)*.

Les améliorations proposées dans [2] pour cette procédure sont les deux suivantes.

- Avant d'effectuer l'étape de fusion, il est proposé de regarder s'il n'y a pas, dans le sous arbre intermédiaire, des variables qui ne sont inclues que dans une seule clique de T'. S'il y en a, elles peuvent être marginalisées directement, ce qui augmente l'efficacité de l'étape de fusion.

- Une autre amélioration consiste à considérer que l'étape de fusion peut être définie comme une opération de suppression de connexions. De cette manière, à la place de définir un processus de fusion, un processus de suppression d'arcs, plus efficace, peut être défini comme suit :

  Si T' est un arbre de jonction et E un sous ensemble de variables, un lien (C_i, C_j) et son séparateur $S_{i,j}$ doivent être supprimés si :

  - $S_{i,j} \not\subseteq E$, ou
  - $S_{i,j} \subseteq E$, mais $S_{i,j} = C_i$ ou $S_{i,j} = C_j$

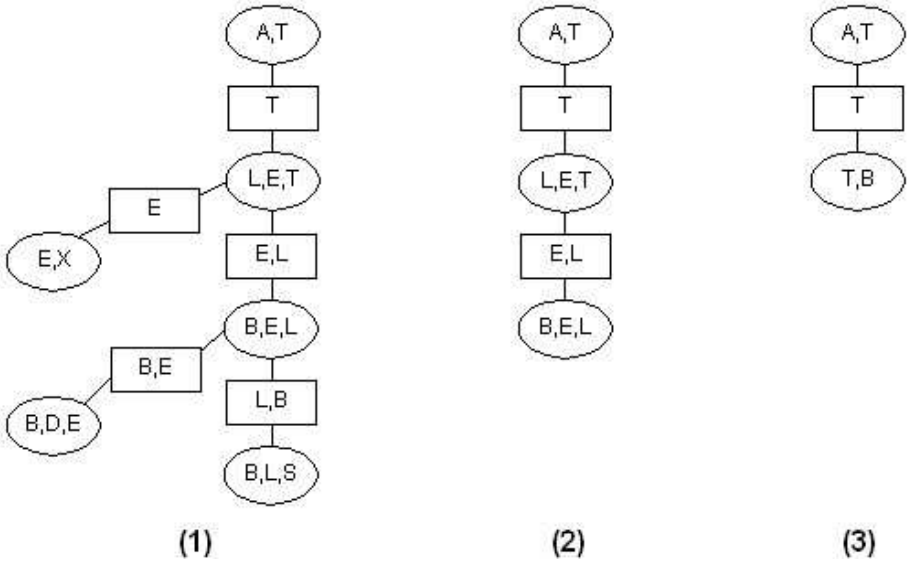


FIG. 6.12 – Application de la technique d'adaptation d'arbre sur un exemple

Une fois que l'arbre de jonction simplifié est obtenu, il suffit d'appliquer l'algorithme de maximum propagation pour obtenir la meilleure configuration du sous-ensemble de variables du réseau bayésien.

Nous n'avons envisagé que cette technique parce qu'elle est aujourd'hui la plus développée et parce que son efficacité a été prouvée. Ceci n'exclut pas l'existence d'autres procédés.
D'une part, dans la lignée de la précédente, une procédure existe qui ne définit pas le sous-arbre adapté comme une modification de l'arbre original mais qui construit directement le sous-arbre de manière plus spécifique, à partir du réseau bayésien. Son efficacité est néanmoins mise en doute, dans [2].
D'autre part, des procédures d'approximation ont été définies. Elles travaillent à partir d'un échantillon de cas observés. Elles sont réputées plus efficaces mais elles manquent parfois de précision.

L'abduction partielle fait actuellement l'objet de nouvelles recherches. Leurs résultats mériteront d'être examinés attentivement.

Bibliographie

[1] A. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2 :25–36, 1992.

[2] L. de Campos, J. Gámez, and S. Moral. Partial abductive inference in bayesian networks by using probability trees. In *Proceedings of the 5th International Conference On Enterprise Information Systems (ICEIS03)*, volume 2, pages 83–91, 2003.

[3] J. A. Gámez. Abductive inference in bayesian networks : a review. Technical report, Computer Sciences Department, Castilla-La Mancha University, Spain, July 2003.

[4] F. V. Jensen. *Bayesian Networks and Decision Graphs.* Statistics for Engineering and Information Science. Springer-Verlag New York, Inc., 2001.

[5] D. Nilsson. An efficient algorithm for finding the m most probable configurations in bayesian networks. *Statistics and Computing*, 8(2) :159–173, 1998.

[6] H. Xu. Computing marginals for arbitrary subsets from marginal representation in markov trees. *Artificial Intelligence*, 74 :177–189, 1995.

Chapitre 7

Grammaires stochastiques *S. Munezero*

7.1 Objectif

Le but de ce chapitre est d'évaluer un modèle considéré comme plus puissant que les réseaux bayésiens pour le cas particulier de la représentation des grammaires non contextuelles. De cette façon, il est montré que si les réseaux bayésiens sont à la fois fort utiles et, comme le montre le chapitre précédent, aisément extensibles, dans certaines situations, d'autres outils doivent être considérés. Le modèle en question est celui des grammaires stochastiques.

En premier lieu les motivations qui ont amené à développer les grammaires stochastiques sont présentées. Ensuite, deux importantes applications des grammaires stochastiques sont étudiées, suivies d'une définition générique de ces grammaires et d'une définition plus formelle.

La question de l'apprentissage des paramètres pour les grammaires stochastiques en tant que grammaires hors-contexte est encore soulevée. Un premier aperçu est donné des différentes méthodes et entraîne vers un exposé du principe des deux algorithmes les plus importants en grammaires stochastiques, l'algorithme Inside-Outside et l'algorithme CYK.

Ce chapître est clôturé par une application concrète, utilisant le cas particulier de l'application à la reconnaissance de phrases structurées et par une comparaison avec les réseaux bayésiens dans le cas de la reconnaissance de formules mathématiques.

7.2 Motivations

Selon les auteurs, dans [2], les motivations principales des travaux qui vont suivre sont les irrégularités (ou exceptions à la grammaire) des langages. En effet, il est important de remarquer que certaines irrégularités peuvent se résoudre en augmentant la grammaire de nouvelles règles de production et en introduisant de nouveaux non-terminaux.

Il est aussi nécessaire de distinguer des règles de productions qui comptent pour une grande majorité de langage de celles qui sont de rares exceptions.

Une manière de résoudre cette distinction serait d'assigner des probabilités aux nombreuses

productions, en ajoutant la propriété que pour tout non-terminal, les probabilités de toutes les productions possibles doivent être sommées à 1.

Pour une phrase donnée x, une grammaire stochastique θ dérivant une phrase assigne une probabilité $Pr[x|\theta]$ que la phrase appartienne au langage spécifié par la grammaire, alors qu'une grammaire conventionnelle ne pourrait que donner une réponse oui/non.
Dès lors, la propriété suivante est vraie : $\sum_{x \in langage} P(x|\theta) = 1$

7.3 Exemples de domaine d'application

7.3.1 Reconnaissance de formes

Les grammaires stochastiques sont beaucoup utilisées pour la reconnaissance des formes. En effet, par définition, la reconnaissance des formes consiste en une automatisation de tâches de perception artificielle réalisées usuellement par le cerveau et le système sensoriel humain (reconnaissance d'un caractère manuscrit, reconnaissance d'un son ou signal, reconnaissance d'un objet dans une image numérique, etc.). Une forme sera définie comme une représentation simplifiée du monde extérieur conçue sous une forme acceptable par l'ordinateur (par exemple un vecteur de réels, ou bien un mot d'un langage donné).

Dans [4], un système de reconnaissance des formes peut comporter une phase d'apprentissage qui consiste à "apprendre" à reconnaître des formes sur la base d'échantillons. Il s'agira en fait de fournir au système un ensemble de formes connues qui permettront de régler l'apprentissage. Lorsque cette phase d'apprentissage est achevée le système est alors prêt à fonctionner pour reconnaître des formes inconnues qui lui sont soumises.

Cependant, un système de reconnaissance des formes peut être aussi un système qui trie (crée des "paquets" homogènes suivant certains critères, création de classes de formes) un ensemble de formes inconnues. Il n'y a alors pas d'apprentissage à proprement parler. Il y a de nombreuses approches possibles pour ce problème.

Le schéma général d'un système de reconnaissance des formes avec une phase d'apprentissage est donné par la figure 7.1.

Le rapport avec les grammaires stochastiques se justifient par le fait qu'après cette phase d'apprentissage, la reconnaissance des formes se fera grâce à un système de probabilités (probabilités que la forme fasse partie d'une classe donnée) qui se résoudront avec l'algorithme Inside/Outside défini ultérieurement.

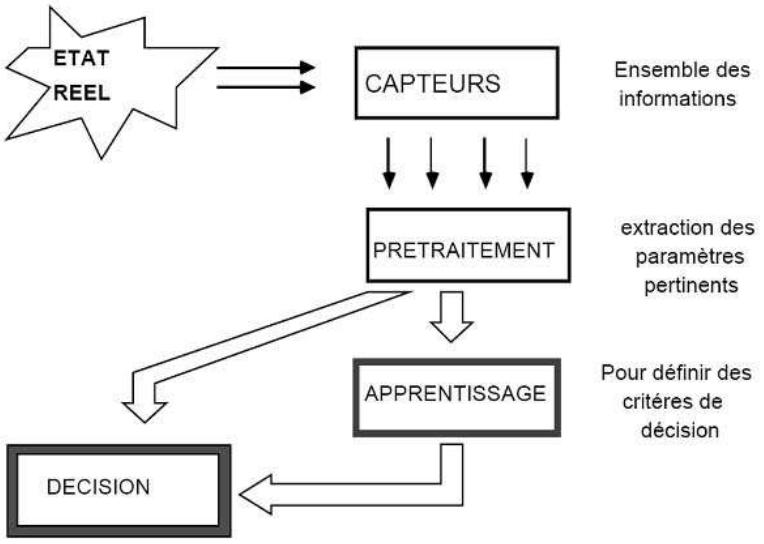


FIG. 7.1 – Schéma général d'un système de reconnaissance des formes

7.3.2 Traitement automatique des langages naturels

D'après [1], il est précisé que le champ d'application du traitement automatique du langage naturel a donné lieu à des recherches stochastiques accrues au cours de la dernière décennie. En effet, les tâches adressées par ces applications ne nécessitent en règle générale qu'une analyse linguistique plus ou moins sommaire.

A noter que le domaine dans lequel la comparaison statistique des grammaires stochastiques est importante est celui de la recherce documentaire où il a été prouvé que la comparaison statistique des mots de l'énoncé et ceux des textes permettent une recherche efficace sans nécessiter la compréhension de la requête en elle-même.

Le traitement automatique du langage naturel utilise une série de pré-traitement et de traitement syntaxique que nous détaillerons sommairement ici afin de ne pas trop nous éloigner de notre problématique de départ.

La première étape dans l'analyse syntaxique du langage naturel est la "Désambiguïsation" (ou tagging) qui consiste à faciliter la tâche des analyseurs en fournissant une séquence désambiguïsée d'étiquettes grammaticales correspondant à la séquence des mots de l'énoncé.

L'étape suivante est la "pré-segmentation syntaxique" qui va intervenir généralement sous forme d'une sortie de l'étiquetage grammatical. Suivant l'analyseur utilisé en sortie, il a pour rôle :

- Soit de fournir une segmentation partielle en constituants non récursifs (les chunks). C'est-à-dire qu'on cherche à identifier l'ensemble des constituants de l'énoncé dans décrire leurs relations de dépendances.
- Soit d'identifier les principales relations lexicales dominants/arguments de l'énoncé.

Modélisation symbolique probabilisée : grammaires stochastiques

Ce modèle se résume à l'ajout de probabilités de déclenchement associées aux règles de grammaire utilisée. Ces probabilités sont estimées sur des corpus de phrases étiquetées. Les grammaires probabilistes fournissent en sortie un ou plusieurs arbres de dérivation classés par probabilités décroissantes.

Ces probabilités globales correspondent au produit des probabilités de transitions liées à chaque dérivation, suivant l'hypothèse d'indépendance conditionnelle inhérente aux modèles stochastiques. Les grammaires stochastiques présentent plusieurs intérêts majeurs qui expliquent leur diffusion significative dans plusieurs domaines d'application :

- D'une manière générale, elles permettent d'améliorer la robustesse d'analyse en autorisant une dégradation douce des performances (dans l'absolu, il peut même ne plus y avoir de situation de rejet).
- Dans le cadre d'analyse faisant suite à un pré-traitement statistique (reconnaissance de la parole ou de l'écriture), elle facilite l'intégration des chaînes de traitement.

Intégration de contraintes symboliques dans un processus stochastique

Il s'agit dans cette section de pouvoir intégrer dès que possible un ensemble de contraintes linguistiques dans l'analyse stochastique.

Cette stratégie a été étudiée avec succès dans le cadre de la reconnaissance de la parole, par l'intermédiaire d'une technique de reclassements d'hypothèses. Ce reclassement peut être envisagé suivant deux approches :

- Travail sur les N-meilleurs hypothèses en parallèle (et non plus filtrage de la meilleure hypothèse acceptée par le parseur).
- Travail sur le treillis de mot élaboré par la reconnaissance de parole. Ce treillis (word graph ou word lattice) représente de manière factorisée les séquences de mots correspondant aux n meilleures hypothèses.

Le reclassement des hypothèses doit nécessairement se baser sur une heuristique d'évaluation qui peut reposer sur des considérations purement linguistiques (choix de l'arbre de dérivation le plus complet, par exemple).

7.4 Introduction aux grammaires stochastiques

7.4.1 Définition

Dans [3], une grammaire stochastique (GS, également appelée grammaire probabiliste, GP) est définie comme une grammaire non contextuelle dans laquelle chaque production est augmentée d'une probabilité.

La probabilité d'une dérivation est alors le produit des probabilités des productions utilisées dans cette dérivation ; ainsi certaines dérivations sont plus cohérentes avec les grammaires stochastiques que d'autres.

Les grammaires stochastiques étendent les grammaires non contextuelles de même que les modèles de Markov cachés étendent les grammaires régulières. Les grammaires stochastiques ont une application directe dans deux principaux domaines : le processus du langage naturel et les molécules d'ADN dans le domaine de la biotechnologie. Les grammaires stochastiques sont une forme spécialisée des grammaires non contextuelles munies de poids.

7.4.2 Techniques

Plusieurs algorithmes ont été défini pour calculer la propagation de l'information dans les grammaires stochastiques et par conséquent, l'ajout d'évidences sur les différentes production. Le plus célèbre d'entre eux est l'algorithme CYK (Cocke-Younger-Kasami) qui permet de calculer la séquence la plus probable pour une grammaire stochastique donnée. Celui-ci est détaillé ultérieurement.

A côté de lui, les algorithmes Inside/Outside sont eux utilisés pour calculer la probabilité totale de toutes les dérivations qui sont consistantes avec une séquence donnée, basée sur une certaine grammaire stochastique.

Ceci est équivalent à la probabilité d'une grammaire stochastique générant une séquence, et est intuitivement une mesure de la manière dont la séquence est consistante avec la grammaire donnée. Les algorithmes Inside/Outside peuvent aussi être utilisées pour calculer les probabilités qu'une production donnée sera utilisée dans une dérivation au hasard d'une séquence.

A ce tire, les algorithmes Inside/Outside sont utilisés comme l'algorithme d'Expectation et Maximisation pour apprendre les probabilités pour une grammaire stochastique basée sur un ensemble de séquences d'entraînement que la grammaire stochastique devrait modéliser. L'algorithme est analogue à celui utilisé par les modèles de Markov cachés.

7.5 Consistance des grammaires stochastiques

7.5.1 Principe

Comme nous pouvons aisément le constater, les grammaires stochastiques constituent un formalisme important de spécification et sont fréquemment utilisées dans des domaines fermés dans la reconnaissance de schémas syntaxiques. En particulier, ces dernières ont été largement utilisées pour caractériser la modélisation probabiliste du langage dans le calcul linguistique et la reconnaissance et compréhension de textes.

Un aspect important de toutes ces applications est l'apprentissage de ces modèles. Le but est d'estimer une description probabiliste adéquate en termes d'un ensemble de séquences d'entraînement.

Pour cela, il existe différentes méthodes pour résoudre l'estimation probabiliste d'une grammaire stochastique. Les méthodes les plus populaires d'estimation probabiliste sont basées sur ce qu'on appelle des transformations de croissance : l'algorithme Inside/Outside (maximiser la vraisemblance d'une séquence) et l'algorithme Viterbi (maximiser la vraisemblance de la meilleure dérivation d'une séquence).

Une question fondamentale qui est liée à ces algorithmes d'estimation probabiliste est de garantir si la grammaire stochastique apprise génère un langage probabiliste, c'est à dire si la grammaire stochastique est *consistante*. De plus, la consistance d'une grammaire stochastique détermine la validité de plusieurs propriétés probabilistes intéressantes. Pour des grammaires stochastiques non ambiguës, il a été prouvé que quand les distributions probabilistes sont estimées par des fréquences relatives dans la séquence, la grammaire stochastique est consistante.

7.5.2 Notations et définitions

Formellement, dans [9], une grammaire stochastique est définie comme une paire (G, π) où G est une grammaire non contextuelle et $\pi = (\pi_1, \pi_2, ..., \pi_r)$ est un vecteur avec les propriétés suivantes pour tout $i, 1 \leq i \leq r$:

- $\pi_i = (p_{i1}, p_{i2}, ..., p_{in_i})$, où $0 \leq p_{ij} \leq 1 (j = 1, 2, ..., n_i)$ et $p_{ij} = Pr[a_{ij}|A_i]$
- $\sum_{j=1}^{n_i} p_{ij} = 1$

Soit G_s une grammaire stochastique, pour chaque $x \in L(G)$, nous notons D_x l'ensemble des dérivations gauche différentes et $d_x^k (1 \leq x \leq |D_x|)$, la $k$ème dérivation pour le string x.
L'expression $N(A_i \rightarrow a_{ij}, d_x^k)$ représente le nombre de fois que la règle $A_i \rightarrow a_{ij}$ a été utilisée dans la dérivation d_x^k et $N(A_i, d_x^k)$ représente le nombre de fois que le non terminal A_i a été dérivé dans d_x^k.

L'équation suivante est satisfaite :

$$N(A_i, d_x^k) = \sum_{j=1}^{n_i} N(A_i \rightarrow a_{ij}, d_x^k)$$

Nous définissons la probabilité de la $k$ème dérivation du string x comme :

$$Pr[x, d_x^k|G_s] = \prod_{i=1,j=1}^{i=r,j=n_i} p_{ij}^{N(A_i \rightarrow a_{ij}, d_x^k)}$$

De la même manière, nous définissons la probabilité de la chaîne de caractères x comme :

$$Pr[x|G_s] = \sum_{k=1}^{|D_x|} Pr[x, d_x^k|G_s]$$

Nous définissons la probabilité de la meilleure dérivation du string x comme :

$$Pr[x|G_s] = \max_{1 \leq k \leq |D_x|} Pr[x, d_x^k|G_s]$$

Le langage généré par la grammaire stochastique G_s est définie par :

$$L(G_s) = x \in L(G) | Pr[x|G_s] > 0$$

Une grammaire stochastique G_s est ainsi dite consistante si :

$$\sum_{x \in L(G_s)} Pr[x|G_s] = 1$$

7.6 Apprentissage de paramètres

7.6.1 Principe

Dans cette section, nous allons développer la question de l'apprentissage de paramètres en grammaires stochastiques telles que définie par les auteurs de [8]. Etant donné un ensemble de données d'apprentissage, constitué de phrases W et d'arbres d'analyse X de ces phrases, on cherche à calculer les paramètres du modèle λ de façon à maximiser la probabilité des arbres conditionnellement aux phrases :

$$\begin{aligned} \lambda &= \max_{\lambda} Pr_{\lambda}[X|W] \\ &= \max_{\lambda} \sum_{x \in X} lnPr_{\lambda}[x|w(w)] \\ &= \max_{\lambda} A(\lambda) \end{aligned}$$

$A(\lambda)$ est la "vraisemblance conditionnelle" de l'ensemble des données. L'apprentissage du modèle consiste en la maximisation de ce critère. On note ici l'une des différences fondamentales de ce modèle avec une SCFG, pour laquelle on cherche en général à maximiser la probabilité de l'ensemble de données $Pr_{\lambda}[X]$, ce qui se résout facilement en affectant à chaque règle une probabilité proportionnelle à sa fréquence dans l'ensemble de données.

7.6.2 Méthode "Improved Iterative Scaling"

La méthode utilisée ici pour le calcul des paramètres s'inspire directement de la méthode IIS (Improved Iterative Scaling) pour la probabilisation de Markov.

Plutôt que de chercher à maximiser directement le critère $A(\lambda)$, ce qui se révèle trop ardu, la méthode améliore itérativement le modèle, à partir d'un modèle initial λ_0.

Une itération consiste à passer d'un modèle λ à un modèle λ', en tentant de maximiser sur λ' le critère suivant :

$$\begin{aligned}\Delta A_\lambda(\lambda') &= A(\lambda') - A(\lambda) \\ &= \sum_{x \in X} ln \frac{Pr_{\lambda'}[x|w(w)]}{Pr_\lambda[x|w(w)]}\end{aligned}$$

Posons $Z_{\lambda,w} = \sum_{x \Rightarrow * w} e^{\lambda.f(x)}$, la constante de normalisation associée à la phrase w dans le modèle λ. On peut alors écrire :

$$Pr_\lambda[x|w(w)] = (Z_{\lambda,w})^{-1} e^{\lambda.f(x)}$$

Et donc :

$$\Delta A_\lambda(\lambda') = \sum_{x \in X} (\lambda' - \lambda).f(x) - \sum_{x \in X} ln \frac{Z_{\lambda',w(x)}}{Z_{\lambda,w(x)}}$$

En majorant le logarithme par $lna = a - 1$, on peut minorer $\Delta A_\lambda(\lambda')$ par :

$$\Delta A_\lambda(\lambda') \geq \sum_{xinX} \Delta_\lambda.f(x) - \sum_{xinX} \frac{Z_{\lambda',w(x)}}{Z_{\lambda,w(x)}} + |X| \tag{7.1}$$

où $\Delta_\lambda = \lambda' - \lambda$, et $|X|$ est le nombre d'arbres de la base d'apprentissage. De plus,

$$\frac{Z_{\lambda',w}}{Z_{\lambda,w}} = \frac{\sum_{y \Rightarrow * w} e^{\lambda'.f(y)}}{Z_{\lambda,w}} \tag{7.2}$$

$$= \sum_{y \Rightarrow * w} \lambda.f(y).\lambda'.f(y) Z_{\lambda,w}.\lambda.f(y) \tag{7.3}$$

$$= \sum_{y \Rightarrow * w} Pr_\lambda[y|w].e^{\Delta_\lambda.f(y)} \tag{7.4}$$

$$\tag{7.5}$$

On note $f^\#(y) = \sum_i f_i(y)$ le nombre de règles utilisées dans un arbre y. On a alors :

$$\Delta_\lambda.f(y) = \sum_i \frac{f_i(y)}{f^\#(y)}.(f^\#(y).\Delta_{\lambda_i})$$

Du fait de la convexité exponentielle :

$$e^{\Delta_\lambda.f(y)} \leq \sum_i \frac{f_i(y)}{f^\#(y)}.e^{f^\#(y).\Delta_{\lambda_i}} \Rightarrow \frac{Z_{\lambda',w}}{Z_{\lambda,w}} \leq \sum_{y \Rightarrow * w} Pr_\lambda[y|w]. \sum_i \frac{f_i(y)}{f^\#(y)}.e^{f^\#(y).\Delta_{\lambda_i}}$$

En injectant ce résultat dans l'inéquation 7.1, il vient :

$$\Delta A_\lambda(\lambda') \geq \sum_{x \in X} \Delta_\lambda . f(x) - \sum_{x \in X} \sum_{y \to w(x)} Pr_\lambda[y|w(x)] \sum_i \frac{f_i(y)}{f^\#(y)} . e^{f^\#(y).\Delta_{\lambda_i}} + |X| = B_\lambda(\lambda') \quad (7.6)$$

On obtient ainsi le critère $B_\lambda(\lambda')$, dont le maximum en λ' est forcément positif, car $B_\lambda(\lambda) = 0$, et qui est toujours inférieur à $A_\lambda(\lambda')$.

De plus, les gradients au point λ de A et B_λ sont égaux, ce qui assure que si le maximum de B_λ se trouve en λ, alors λ est un maximum local A (et assure du même coup que l'algorithme IIS converge vers un maximum local de A).

Pour aller plus vite vers le maximum de A, on va donc chercher à maximiser $B_\lambda(\lambda')$ à chaque itération de l'algorithme. On peut pour cela annuler ses dérivées partielles en Δ_{λ_i}, en résolvant r_i de la grammaire l'équation suivante :

$$0 = -\sum_{x \in X} f_i(x) + \sum x \in X \sum y \to w(x) Pr_\lambda[y|w(x)] . f_i(y) . (e^{\Delta_{\lambda_i}})^{f^\#(y)} \quad (7.7)$$

Il s'agit d'un polynôme $\alpha = e^{\Delta_{\lambda_i}}$, de coéfficients tous positifs sauf celui de degré zéro. Ce polynôme est donc facilement annulé, par exemple par la méthode de Newton.

7.6.3 Algorithme "Inside Outside"

Le premier terme du polynôme $-\sum_{x \in X} f_i(x)$ est trivialement obtenu : il représente la fréquence de la règle r_i dans l'ensemble de données arboré. Il reste le problème du calcul des coefficients de degrés supérieurs : le terme $\sum_{y \Rightarrow * w}$ implique une sommation sur toutes les analyses de la phrase w.

C'est l'une des difficultés majeures de ce modèle, le nombre d'analyses pouvant croître exponentiellement avec la longueur de la phrase. Cette étape de calcul nécessite dans certains modèles de Markov-Gibbs d'employer une méthode approchée par échantillonnage. Ici, une factorisation du calcul peut être effectuée à l'aide d'un algorithme Inside-Outside, comme le montrent les réécritures suivantes.

On réécrit la troisième somme de 7.7 par :

$$\begin{aligned} S_{w,\lambda}(\alpha) &= \sum_{y \to w} Pr_\lambda[y|w] . f_i(y) . \alpha^{f^\#(y)} \\ &= Z_{w,\lambda}^{-1} \sum_{y \to w} e^{\lambda . f(y)} . f_i(y) . \alpha^{f^\#(y)} \end{aligned}$$

Notons $C(y,[j,r_i,k])$ le fait que, dans l'arbre y, la règle r_i domine $w_i...w_k$ ($C(y,[j,r_i,k])=1$) ou non ($C(y,[j,r_i,k])\leq 1$). Avec cette notation, on a :

$$\begin{aligned} S_{w,\lambda}(\alpha) &= Z_{w,\lambda}^{-1} \sum_{1\leq j\leq k\leq |w|} \sum_{y\Rightarrow *w} e^{\lambda.f(y)}.\alpha^{f^{\#}(y)}.C(y,[j,r_i,k]) \\ &= Z_{w,\lambda}^{-1} \sum_{1\leq j\leq k\leq |w|} \sum_{y\Rightarrow *w} V(y).C(y,[j,r_i,k]) \end{aligned}$$

où $V(y) = e^{\lambda.f(y)}.\alpha^{f^{\#}(y)} = \prod r_i \in y(f_i.\alpha)^{f_i(y)}$ est le produit des polynômes $P_i(\alpha) = (\lambda_i.\alpha)$ associés aux règles r_i qui constituent y.

$\sum_{y\Rightarrow *w} V(y).C(y,[j,r_i,k])$ peut être calculé pour tout r_i, j et k à l'aide d'un algorithme Inside-Outside, dans le cas où le grammaire est sans cycle.

Le principe de l'algorithme est le suivant :

- Pour chaque triplet (j,r_i,k), calcul $inside[j,r_i,k]$, qui est la somme des $V(y)$ des arbres y qui ont la première règle r_i et qui dominent les mots $w_j...w_k$
- Pour chaque triplet (j,A,k), calculer $outside[j,A,k]$, qui est la somme des $V(y)$ des arbres y de feuilles $w_1...w_{j-1}Aw_{k+1}...w_n$
- En notant $G(r_i)$ le symbole de la partie gauche de r_i, le résultat obtenu par : $\sum_{y\Rightarrow *w} V(y).C(y,[j,r_i,k]) = inside_r[j,r_i,k] * outside[j,G(r_i),k]$

7.6.4 Algorithme "Improved Iterative Scaling"

Finalement, les différentes méthodes d'apprentissage sont reprises par l'algorithme suivant :

Détails d'optimisation :

- Le critère $A(\lambda)$ est facile à obtenir à la fin d'une passe, du fait que les constantes de normalisation $Z_{\lambda,w(x)}$ sont calculées lors de cette passe. C'est pourquoi le test d'arrêt se base sur le modèle précédent λ plutôt que sur le nouveau modèle λ', quitte à faire une itération inutile : cela évite le recalcul des constantes de normalisation, qui demanderait une nouvelle analyse de tous les exemples de la base à chaque passe.

- L'algorithme Inside-Outside commence par une analyse syntaxique de la phrase examinée, puis calcule les valeurs des polynômes à annuler. Pour accélérer l'apprentissage, les tables des analyses syntaxiques peuvent être sauvegardées dans une première étape, ce qui permet de calculer directement les polynômes lors des itérations.

Algorithm 6 Algorithme Improved Iterative Scaling

Définir un modèle initial λ', par exemple en mettant tous les paramètres à 0
while Le critère $A(\lambda)$ n'a pas convergé **do**
  $\lambda \leftarrow \lambda'$
  Mise à zéro des polynômes $S^r(a)$ associés aux règles r
  for all x dans la base d'apprentissage **do**
    Analyser $w(x)$ par l'algorithme inside-outside
    Calculer $Z_{\lambda,w(x)}$ comme la somme des coéfficients du polynôme $\sum_r inside_r[1,r,n]$
    for all élément [j,r,k] **do**
      $S^r(a) \leftarrow S^r(a) + (Z_{\lambda,w(x)})^{-1} inside_r[j,r,k] * outside[j,G(r),k]$
    end for
  end for
  for all règle r_i de la grammaire **do**
    Résoudre $S^{r_i}(a) = -\sum_{x \in X} f_i(x)$
    Calculer le paramètre du modèle λ' par $\lambda'_i = \lambda_i + log(\alpha)$
  end for
end while

7.7 Déterminer la séquence la plus probable : algorithme Cocke-Younger-Kasami (CYK)

7.7.1 Principe

Dans [7], d'après les auteurs, l'algorithme CYK est un algorithme tabulaire ascendant, qui permet de trouver, à partir d'une grammaire stochastique, la séquence la plus probable de terminaux. Il présente deux avantages :

- Sa complexité pire cas est optimale $O(n_3)$.
- Très simple à comprendre et implémenter.

Il présente cependant (dans sa version de base) deux inconvénients :

- Nécessite la mise de la grammaire non contextuelle sous forme particulière : forme normale de Chomsky.
- Sa complexité reste $O(n_3)$ sur des grammaires de complexité inférieure (e.g. régulières, LR).

Toute grammaire non contextuelle peut se mettre sous forme normale de Chomsky. Les seules formes de règles non autorisées sont :

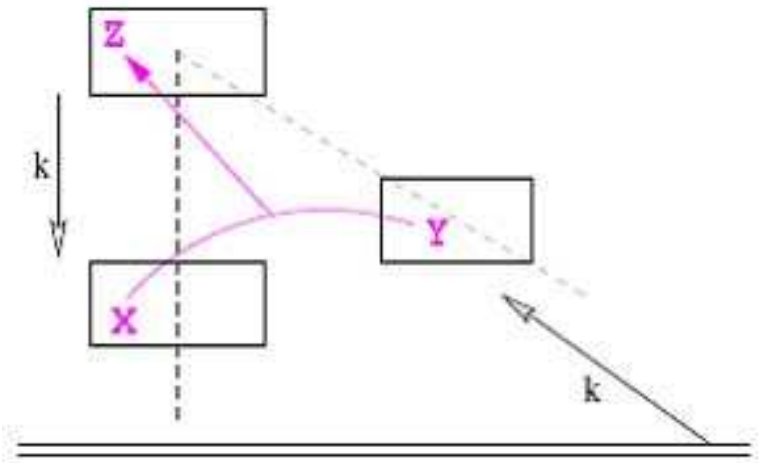
- $X \rightarrow YZ$ où Y et Z sont des non terminaux ($\in C$).
- $X \rightarrow w$ où w est un terminal ($\in L$).

On y autorise parfois aussi les règles du type : $X \rightarrow Y$ où $Y \in C$.

CYK est un analyseur tabulaire ("chart-parser"). Il permet un calcul efficace (en espace et en temps) de toutes les interprétations possibles de toutes les sous-séquences de la séquence à analyser.
L'aspect efficace du calcul est fondé sur la propriété suivante : si la grammaire est sous forme normale de Chomsky (voire sous forme étendue), le calcul des interprétations d'une séquence W de taille l nécessite uniquement l'exploration de toutes les décompositions de W en exactement deux sous-séquences. Le nombre de paires à explorer est donc $l-1$. Idée : mettre les analyses des sous-séquences dans une table.

L'analyse syntaxique d'un phrase de n mots $W = w_1...w_n$ est organisée dans une table triangulaire de cellules $C_{i,l}$ $(1 \leq i \leq n; 1 \leq l \leq n)$, dans laquelle la cellule $C_{i,l}$ contient tous les non-terminaux pouvant dériver la sous-séquence $w_i...w_{i+l-1}$ de longueur l démarrant au i-ième mot de W. Le remplissage de la table s'effectue de bas en haut et de gauche à droite (c'est à dire dans le sens de la phrase).

7.7.2 Algorithme



7.7.3 Complexité

Le calcul des interprétations présentes dans la case $C_{i,j}$ nécessite $i-1$ explorations de paires de cellules $(1 \leq k \leq i-1)$, le nombre total d'explorations est donc :

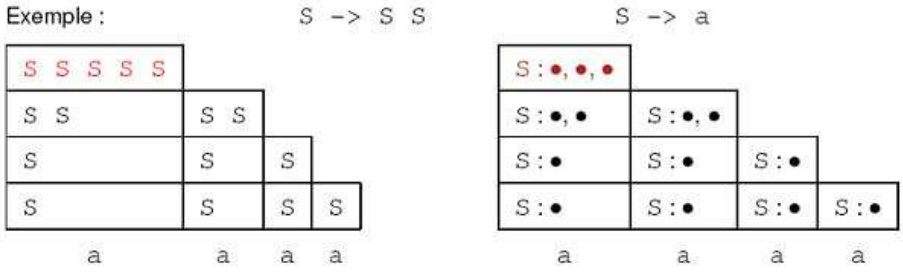
$$\sum_{i=2}^{n} \sum_{j=1}^{n-i+1} (i-1) = \sum_{i=1}^{n} (n-i+1).(i-1) = O(n^3)$$

Dans le pire des cas, chaque cellule contient tous les non-terminaux possibles, soit $|C|$ éléments et donc l'exploration du produit croisé des contenus de deux cases est dans le pire des cas en $|C|^2$. D'où la complexité : $O(n^3)$ et $O(|C|^2)$
On voit ici l'inconvénient de la mise sous forme normale de Chomsky qui augmente C. Une fois la table remplie (en O(n3)), un arbre d'analyse peut s'extraire en $O(n)$.

Algorithm 7 Algorithme CYK

```
for all 2 ≤ i ≤ n do
  for all 1 ≤ j ≤ n − i + 1 do
    for all 1 ≤ k ≤ i − 1 do
      for all X ∈ chart[i − k][j] do
        for all Y ∈ chart[k][i + j − k] do
          for all Z → XY ∈ ℜ do
            Ajouter Z à chart[i][j]
          end for
        end for
      end for
    end for
  end for
end for
```

Cependant, On peut facilement implémenter cet algorithme de manière à avoir une complexité $O(e^n)$. En effet, si l'on duplique les non-terminaux produits (au lieu de factoriser leurs interprétations) on peut faire exploser leur nombre.



7.8 Evaluation de la qualité de l'apprentissage d'un modèle syntaxique stochastique

7.8.1 Critère d'évaluation

D'après les travaux dans ce domaine exposés dans [5], la perplexité est un des critères qui permettent d'évaluer la qualité d'un modèle syntaxique pourvu d'une distribution de probabilités. Elle peut être comprise comme le pouvoir de prédiction de ce modèle.

Ce pouvoir est d'autant plus grand que le nombre de symboles proposés par le modèle pour correspondre au symbole suivant d'une séquence est petit. Si le modèle ne peut proposer aucun symbole (dans notre cas : si la probabilité d'une règle est nulle ou si elle n'existe pas) alors son pouvoir prédictif est nul et sa perplexité est maximale.

Formellement, la perplexité est définie par :

$$PP = 2^{-1/||S||.\sum_{i=1}^{|S|} log_2 Pr[x_i]}$$

où $|S|$ est le nombre de séquences de l'échantillon de test, $||S||$ est la somme des longueurs des séquences de l'échantillon de test, et $Pr[x_i]$ est la probabilité que la $i$ème séquence x_i ait été générée par le modèle.

Ce calcul est conditionné d'une part par la consistance du modèle (ce dont nous sommes assurés), d'autre part par l'attribution d'une probabilité non nulle à toute séquence composée sur S. Dans le cas général, étant données une grammaire stochastique G et une séquence x, il est possible de calculer la probabilité pour que x soit générée par G. Cette probabilité peut valoir zéro, mais seulement si la syntaxe de la séquence n'est pas reconnue par la grammaire.

Cependant, pour pallier le manque de données d'apprentissage, qui conduit à une mauvaise estimation statistique du modèle, on doit lisser ses paramètres. Ce lissage permet en outre d'engendrer toute chaîne avec une probabilité non nulle.

7.8.2 Exemple

Soit G une grammaire stochastique. Soit $G' = (N, S, R', S)$ la grammaire G augmentée par les règles de corrections d'erreurs suivantes :

Insertion de a :

$$A \rightarrow aA, \forall (A \rightarrow bB) \in P$$

Substitution de b par a :

$$A \rightarrow aB, \forall (A \rightarrow bB) \in P$$

Suppression de b :

$$A \rightarrow B, \forall (A \rightarrow bB) \in P$$

La probabilité pour que $x \in \Sigma*$ soit généré par G' est :

$$Pr_{G'}[x] = \sum_{\forall D'_G(x)} (\prod_{\forall r_i \in D'_G(x)} Pr[r_i])$$

où $D'_G(x)$ est une dérivation corrective (quelconque) de G pour engendrer x.

On suppose que l'on connaît les probabilités des opérations correctives :

$$Pr[x_i, x_j], \forall xi, xj \in \{\Sigma \cup \epsilon\}$$

Nous pouvons alors écrire :

$$\forall A \rightarrow x_j B \in R', Pr[A \rightarrow x_j B] = Pr[A \rightarrow x_i B \in R] * Pr[x_i, x_j]$$

Pour que le modèle reste consistant, il est nécessaire que la somme des probabilités des règles dont la partie gauche est constituée d'un même non-terminal soit égale à 1 :

$$\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}, \forall B \in N | \exists A \rightarrow x_i B \in R'} Pr[A \rightarrow x_i B] = 1, \forall A \in N$$

Nous satisfaisons cette condition en ajoutant la contrainte :

$$\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} Pr[x_i, x_j] = 1, \forall x_i \in \{\Sigma \cup \epsilon\} \tag{7.8}$$

Le problème revient alors à calculer les probabilités des opérations d'édition en respectant 7.8.

7.9 Application à la reconnaissance de phrases structurées

7.9.1 Définition de la grammaire

Rappelons qu'une grammaire stochastique est définie comme un tuple $[\Sigma, N, S, P]$, où les ensembles disjoints S et N spécifient les symboles terminaux et non-terminaux, respectivement, avec $S \in N$ comme symbole de départ.

P est un ensemble de productions, qui prend la forme $E \rightarrow \xi(p)$, avec $E \in N$, $\xi \in (\Sigma \cup N)^+$, et $p = Pr[E \rightarrow \xi]$, la probabilité que E sera dérivé dans la chaîne de caractères ξ.

La somme des probabilités p de toutes les dérivations d'un non terminal E doit être 1. Les exemples dans ce chapitre s'inspireront d'un exemple simple de grammaire qui est le suivant :

$$
\begin{aligned}
&S \rightarrow NP\ VP \implies (0.8)\\
&S \rightarrow VP \implies (0.2)\\
&NP \rightarrow NOM \implies (0.4)\\
&NP \rightarrow NOM\ PP \implies (0.4)\\
&NP \rightarrow NOM\ NP \implies (0.2)\\
&VP \rightarrow VERBE \implies (0.3)\\
&VP \rightarrow VERBE\ NP \implies (0.3)\\
&VP \rightarrow VERBE\ PP \implies (0.2)\\
&VP \rightarrow VERBE\ NP\ PP \implies (0.2)\\
&PP \rightarrow PREP\ NP \implies (1.0)\\
&PREP \rightarrow comme \implies (1.0)\\
&VERBE \rightarrow Suzie \implies (0.2)\\
&VERBE \rightarrow vole \implies (0.4)\\
&VERBE \rightarrow comme \implies (0.4)\\
&NOM \rightarrow Suzie \implies (0.05)\\
&NOM \rightarrow vole \implies (0.45)\\
&NOM \rightarrow fourmis \implies (0.5)
\end{aligned}
$$

Dans [6], il apparaît que la définition d'un modèle de grammaires stochastiques hors-contexte interdit les règles de la forme $E \rightarrow \epsilon$, où ϵ représente le caractère vide. Cependant, nous pouvons réécrire toute grammaire stochastique hors contexte pour éliminer de telles règles et continuer à représenter la distribution originale, aussi longtemps que l'on mentionne la probabilité $Pr[S \rightarrow \epsilon]$. Pour plus de clarté, les descriptions algorithmiques qui vont suivre supposent que $Pr[S \rightarrow \epsilon] = 0$.

La probabilité d'appliquer une production particulière $E \rightarrow \xi$ à une chaîne de caractère intermédiaire est conditionnellement indépendante aux productions qui ont générées cette chaîne de caractère, ou aux productions qui seront appliquées aux autres symboles dans la chaîne de caractère, étant donné la présence de E. Ainsi, la probabilité d'une dérivation donnée est simplement le produit des probabilités de chaque production utilisée.

Nous définissons la représentation de l'arbre de dérivation de chacune de ces dérivations comme pour les grammaires non contextuelles. La probabilité d'une chaîne de caractère est la somme prise à partir de toutes les dérivations possibles.

7.9.2 Application des algorithmes standard

Comme le nombre de dérivations possibles grandit de manière exponentielle avec la longueur de la chaîne de caractère, l'énumération directe ne serait pas vraiment mathématiquement intéressant.

A la place, l'approche standard de programmation utilisée pour les grammaires non contextuelles classiques et celles stochastiques exploite les séquences de production commune partagées par les dérivations. La structure standard est une table, ou *chart*, enregistrant les résultats précédents pour chaque sous séquence dans la séquence donnée en entrée. Le chart de l'exemple est donné en 7.2.

Chaque entrée dans le chart correspond à une sous séquence $x_i...x_{i+j-1}$ de l'observation string $x_1...x_L$. Pour chaque symbole E, une entrée contient la probabilité que la sous séquence correspondante soit dérivée de ce symbole, $Pr[x_i...x_{i+j-1}|E]$. L'index i fait référence à la position de la sous séquence à l'intérieur de la chaîne de caractère terminal entier, avec $i = 1$ indiquant la séquence de départ.

La ligne d'en bas dans la table 7.2 contient les résultats pour les sous séquences de longueur 1, et l'entrée d'en haut contient le résultat général, $Pr[x_1...x_L|S]$, qui est la probabilité de la chaîne de caractère observé.

	$i=1$	$i=2$	$i=3$	$i=4$
$j=4$	$(S \rightarrow VP)_{0.00072}$ $(S \rightarrow NP(2)\ VP(2))_{0.000035}$ $(S \rightarrow NP(1)\ VP(3))_{0.000256}$ $(S \rightarrow VERBE\ NP\ PP)_{0.0014}$ $(S \rightarrow VERBE\ NP)_{0.00216}$			
$j=3$		$(VP \rightarrow VERBE\ PP)_{0.016}$ $(NP \rightarrow NOM\ PP)_{0.036}$		
$j=2$	$(NP \rightarrow NOM\ NP)_{0.0018}$		$(VP \rightarrow VERBE\ NP)_{0.024}$ $(PP \rightarrow PREP\ NP)_{0.2}$	
$j=1$	$(NP \rightarrow NOM)_{0.02}$ $(VERBE \rightarrow Suzie)_{0.2}$ $(NOM \rightarrow Suzie)_{0.05}$	$(NP \rightarrow NOM)_{0.18}$ $(VERBE \rightarrow vole)_{0.4}$ $(NOM \rightarrow vole)_{0.45}$	$(PREP \rightarrow comme)_{1.0}$ $(VERBE \rightarrow comme)_{0.4}$	$(NP \rightarrow NOM)_{0.2}$ $(NOM \rightarrow fourmis)_{0.5}$

FIG. 7.2 – Chart pour Suzie vole comme fourmis

Ces probabilités peuvent être calculées de bas en haut, puisque $Pr[x_i|E] = 1$ si E est le symbole observé x_i. Il est possible de définir toutes les autres probabilités récursivement comme la somme, sur toutes les productions $E \rightarrow \xi(p)$, du produit $p.Pr[x_i...x_{i+j-1}|\xi]$.
Remplacer la procédure pour prendre le maximum au lieu de la somme donne l'arbre de dérivation le plus probable pour la chaîne de caractère observée. Les deux algorithmes requièrent une complexité de $O(L^3)$ pour une chaîne de caractère de longueur L, ignorant la dépendance sur la taille de la grammaire.

Pour calculer la probabilité de la séquence "*Suzie vole comme fourmis*", l'algorithme qui génère la table montrée précédemment est utilisé, après avoir éliminé toutes les entrées intermédiaires inutiles. Il y a aussi des entrées séparées pour chaque production, celles-là ne sont pas nécessaires si nous sommes uniquement intéressés par la probabilité de la séquence finale.
Dans l'entrée du haut, il y a deux listes pour la production $S \rightarrow NP\ VP$, avec des longueurs de sous séquence différentes pour les symboles du coté droit. La somme de toutes les probabilités pour les productions S sur le coté gauche dans cette entrée donne la probabilité totale de la séquence qui est de 0.001011.

L'algorithme est capable de calculer toute probabilité de type *inside*, la probabilité d'une chaîne de caractère particulier apparaissant à l'intérieur du sous arbre ayant pour racine un symbole particulier.
En travaillant de haut en bas de manière analogue pour calculer toute probabilité de type *outside*, la probabilité du sous arbre ayant pour racine un symbole particulier apparaissant au milieu d'une chaîne de caractère particulier, est obtenue.

Etant donné ces probabilités, la probabilité de tout symbole non terminal particulier apparaissant dans l'arbre de dérivation comme la racine d'un sous arbre couvrant une certaine sous séquence, peut être calculé.
Par exemple, dans la phrase *Suzie vole comme fourmis*, la probabilité que *comme fourmis* soit une phrase prépositionnelle peut être obtenue, en utilisant la combinaison des probabilités *inside* et *outside*.

L'algorithme Left-to-Right Inside (LRI) spécifie comment il est possible utiliser les probabilités *inside* pour obtenir la probabilité d'une sous séquence initiale donnée, tel que la probabilité d'une phrase (de toute longueur) commençant avec les mots *Suzie vole*. Plus tard, les probabilités d'une telle sous séquence peuvent être utilisées pour calculer la probabilité conditionnelle d'un symbole terminal suivant étant donné une chaîne de caractère préfixe.

7.9.3 Indexer les arbres de dérivation

Certaines problématiques ne sont pas simplement solubles via les manipulations directes des probabilités *inside* et *outside*. Par exemple, étant donné les observations de chaînes de caractères arbitrairement partielles, l'exploitation directe du chart standard n'est pas très claire.
De la même manière, nous ne sommes pas conscients des méthodes pour gérer l'observation de non terminaux uniquement (i.e., que les deux derniers mots forment une phrase prépositionnelle).
Nous cherchons, ainsi, un mécanisme qui admettrait les évidences observationnelles de toute forme comme une part d'une requête à propos d'une grammaire stochastique hors contexte, sans nous demander d'énumérer tous les arbres de dérivation consistants.

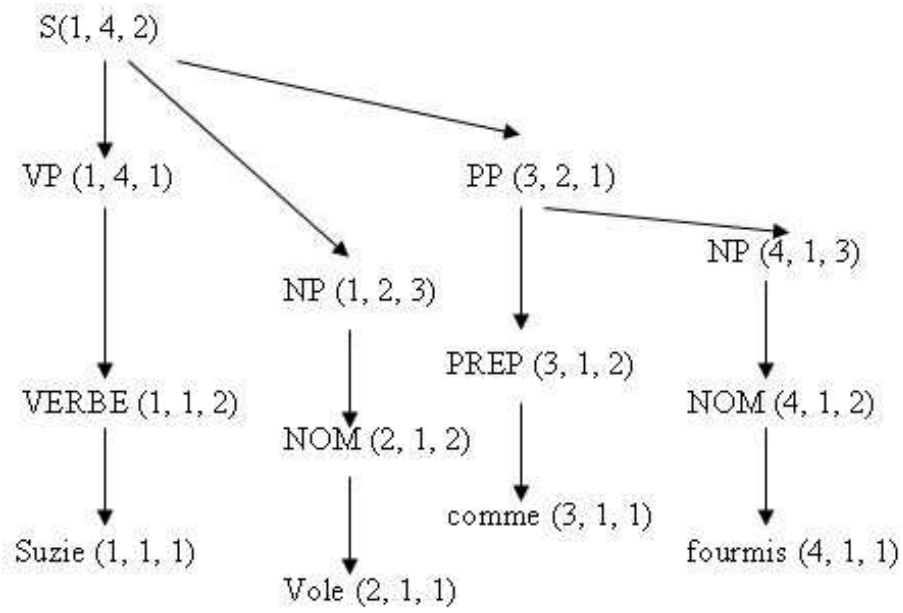


FIG. 7.3 – Arbre de dérivation pour *Suzie vole comme fourmis*, avec (i, j, k) les indices.

En mettant d'abord en place un schéma pour représenter un tel événement comme l'apparition de symboles à des points désignés dans l'arbre de dérivation, il est alors possible d'utiliser les indices i et j pour délimiter les noeuds feuille du sous arbre, comme dans la dérivation des algorithme de chart standard. Par exemple, le noeud PP dans l'arbre de dérivation est la racine du sous arbre dont les noeuds feuilles sont *comme* et *fourmis*, c'est-à-dire $i = 3$ et $j = 2$.

Cependant, il est impossible de représenter un noeud unique avec ces deux indices seulement. Dans la branche de l'arbre de dérivation à travers NP, NOM et vole, tous les trois noeuds ont $i = 2$ et $j = 1$. Pour les différencier, nous introduisons l'index k, défini récursivement. Si un noeud n'a pas d'enfant avec les mêmes indices i et j, alors il a $k = 1$. Sinon, son index k est augmenté de 1 par rapport à l'index k de son enfant. Ainsi, le noeud vole a $k = 1$, le noeud NOM au dessus a $k = 2$, et son parent NP a $k = 3$. Nous avons labellé chaque noeud dans l'arbre de dérivation avec ses indices i, j, k).

L'index k d'un noeud peut être vu comme un niveau d'abstraction. Plus k est grand, plus l'abstraction d'un symbole est importante. Par exemple, le symbole *vole* est la spécialisation du concept NOM, qui, aussi, est une spécialisation du concept NP. Chaque spécialisation possible correspond à une production d'abstraction de la forme $E \rightarrow E'$, qui est, avec uniquement un symbole du coté droit de la règle.

Dans l'arbre de dérivation qui fait intervenir une telle production, les noeuds pour E et E' ont des valeurs identiques pour i et j, mais la valeur k pour E est augmentée de 1 par rapport à E'. Nous notons l'ensemble des *productions d'abstraction* comme $P_A \subseteq P$.

Toutes les autres productions sont des *productions de décomposition*, dans l'ensemble $P_D = P/P_A$. Si le noeud E est dérivé d'une production de décomposition, la somme des valeurs de j pour ses enfants est égale à sa propre valeur de j, puisque la longueur de la sous séquence originale dérivée de E doit être égal aux longueurs totaux des sous séquences de ses enfants.

De plus, puisque que chaque enfant doit être dérivé d'une chaîne de caractère de longueur non nul, aucun enfant n'a le même index j que E, qui doit avoir une valeur de k qui est de 1. Ainsi, les productions d'abstraction connectent les noeuds dont les indices correspondent aux composants i et j, alors que les productions de décomposition connectent les noeuds dont les indices diffèrent.

7.9.4 Formules mathématiques

L'exemple complet qui vient d'être présenté concerne la reconnaissance de phrases structurées, dans la langue française. Il est évident, sur base de cet exemple, que le modèle des grammaires stochastiques peut s'appliquer au cas étudié dans les chapitres précédents, soit celui de la reconnaissance de formules mathématiques, qui ne sont après tout que des phrases structurées de manière complètement formelle.
Les grammaires stochastiques peuvent être utilisées comme modèle concurrent à celui des réseaux bayésiens dans ce cas concret. En comparant les deux modèles, on peut remarquer que :

- Les deux modèles permettent la propagation d'information, en s'appuyant sur les probabilités a priori définies. Ils permettent tous deux la reconnaissance de chaînes de caractères dans une grammaire, devant respecter des règles de grammaire. Ils permettent aussi de trouver la séquence la plus probable d'états ou de non terminaux.

- Les deux modèles permettent la mise d'évidences venant du classifieur (dans le cas des grammaires stochastiques, il suffit de modifier les probabilités a priori des caractères terminaux).

- Les grammaires stochastiques permettent une représentation plus aisée des règles non contextuelles, alors que dans le cadre des réseaux bayésiens, il est très difficile de le représenter, comme cela a été montré dans l'application de correction des erreurs, développée dans le chapitre 5.

- Les deux modèles permettent un apprentissage des paramètres. Cependant, seules les grammaires stochastiques permettent un apprentissage de paramètres liés à des règles non contextuelles. Les réseaux bayésiens pourraient le permettre de manière détournée, comme cela a été envisagé dans l'application aux formules mathématiques mais de manière inefficace.

- Les grammaires stochastiques, dont le domaine se limite à la formulation de règles de grammaires, ne permettent pas des représentations de structures et les contraintes liées. Par conséquent, certaines applications développées dans le cadre de la reconnaissance des formules mathématiques, ne pourraient pas être traduites dans ce modèle. Il s'agit par exemple de déterminer la qualité de l'utilisateur.

Ce chapitre sur les grammaires stochastiques a permis de montrer que certaines difficultés rencontrées lors de l'utilisation des réseaux bayésiens (par exemple, la représentation des concepts non conceptuels) peuvent être surmontés par un modèle concurrent. Cependant, comparer les deux modèles dans le domaine de la reconnaissance manuscrite, amène déjà à penser que cet autre modèle a ses propres lacunes.

Pour développer de manière complète un système permettant la reconnaissance de formules mathématiques, il faudrait en réalité considérer une combinaison de ces deux modèles, utilisant au mieux leurs avantages distincts.

Bibliographie

[1] J.-Y. Antoine and D. Genthial. Méthodes hybrides issues du taln et du tal parlé : état des lieux et perspectives. Atelier Thématique 1999.

[2] J. Case, K. Decker, C. Kambhamettu, L. Liao, and V. Shanker. Stochastic grammars. CISC 889 Bioninformatics, University of Delaware.

[3] Durbin, Eddy, Krogh, and Mitchison. Biological sequence analysis. Cambridge University Press, 1998.

[4] H. Fouchal. Reconnaissance des formes, méthodes stochastiques. Université des Antilles et Guyane, Pointe à Pitre.

[5] L. Miclet and J. Chodorowski. Apprentissage et evaluation de modèles de langages par des techniques de correction d'erreurs. Institut de Recherche en Informatique et de Systèmes Aléatoires-Ecole Normale Supérieure de Sciences Appliquées et de Technologies, Rennes, France.

[6] D. V. Pynadath. Probabilistic grammars for plan recognition. Computer Science and Engineering, University of Michigan, Etats-Unis.

[7] M. Rajman and J.-C. Chappelier. Traitement informatique des données textuelles, analyse syntaxique : Introduction, algorithme c.y.k. Laboratoire d'Intelligence Artificielle, Ecole Polytechnique Fédérale de Lausanne, Suisse.

[8] A. Rozenknop. Une grammaire hors-contexte valuée pour l'analyse syntaxique. IC-IIF-LIA-EPFL, CH-1015 Lausanne, Suisse.

[9] J. A. Sanchez and J.-M. Benedi. Consistancy of stochastic context-free grammars from probabilistic estimation based on growth transfomations. Département d'informatique et de calcul, Université Politechnique de Valence.

Conclusion

L'intérêt des réseaux bayésiens est aujourd'hui incontestable. La théorie de départ a été très considérablement développée depuis sa première ébauche et la plupart des recherches actuelles, viennent enrichir ces modèles et améliorent leur efficacité. Plusieurs logiciels renommés facilitent leur utilisation.

Après avoir présenté la théorie des réseaux bayésiens de manière globale, nous avons essayé de mettre en oeuvre une application qui en fait usage. Celle-ci avait pour but la reconnaissance et l'interprétation de formules mathématiques manuscrites.

Nous sommes parvenus à reconnaître de manière efficace une formule mathématique, sur base des informations incertaines fournies par un classifieur, en utilisant l'algorithme de maximum propagation.

Satisfaits de ces résultats, nous nous sommes alors penchés sur la prise en compte des erreurs mathématiques qu'un utilisateur pourrait écrire. Un problème de taille est apparu, celui des règles non contextuelles de grammaire. La structure même des réseaux bayésiens semble incapable de prendre en charge l'intégration les contraintes non contextuelles.

Par conséquent, en ce qui concerne le versant "correction", la majeure partie de notre travail a été de définir des algorithmes externes qui, aidés des réseaux bayésiens, réussissaient à corriger des erreurs. Ce résultat est cependant discutable, puisqu'il nous a obligé à utiliser des mécanismes externes aux modèles de Bayes.

Néanmoins, le logiciel développé pour tester nos différentes solutions a fait ses preuves et est capable de proposer des corrections, de manière efficace dans le cas des formules ayant un nombre raisonnable d'erreurs.

Par la suite, nos recherches ont pris une autre direction et se sont attardées sur la considérations de types d'utilisateurs différents. Comme pour la reconnaissance des formules, les réseaux bayésiens se sont révélés être des outils d'une simplicité déconcertante, permettant de prendre en compte de manière avantageuse le comportement de l'utilisateur.

Eu égard à l'incertitude qui entourait les conclusions qu'il y avait à tirer de la combinaison de réussites et d'insatisfactions que notre application a démontrées, nous avons alors étudié deux aspects supplémentaires des réseaux bayésiens.

D'une part, l'extensibilité de ces modèles a été analysée, afin déterminer si ceux-ci pouvaient réellement s'adapter à des situations spécifiques, que n'abordait pas toujours la théorie d'origine. Les illustrations auxquelles nous avons recouru concernaient la recherche d'un classement de résultats probables et la combinaison de différents types de propagation d'information.
La réponse s'est avérée positive dans le cas des extensions envisagées pour les algorithmes de propagation.

D'autre part, puisqu'un réseau bayésien ne permettait pas de modéliser le "non contextuel", nous avons cherché s'il n'existait pas de modèle qui pourrait le représenter. Nous nous sommes dès lors vers un modèle plus performant à cet égard, celui des grammaires stochastiques.
Ce modèle probabiliste mais non graphique a été analysé et a permis de démontrer sa meilleure performance dans l'intégration des concepts non contextuels. Nous n'avons toutefois pas reconsidéré la correction d'erreurs de formules mathématiques à la lumière des grammaires stochastiques.

Si dans l'ensemble, les réseaux bayésiens ont un grand intérêt, par leur définition et leur extensibilité, ils doivent néanmoins encore être comparés à d'autres modèles, lorsqu'il s'agit de les utiliser pour une application pratique.

La richesse de la théorie des réseaux bayésiens devrait être complétée d'une analyse exhaustive de leurs réels avantages et inconvénients, confrontés à des modèles probabilistes concurrents. Il s'agit d'un travail de grande envergure qui permettrait de faire le tri dans les fonctionnalités multiples offertes par ces modèles et d'obtenir un support capable de déterminer leurs limites et les conditions auxquelles leur utilisation doit ou ne doit pas être envisagée.

Dans le cas de l'application envisagée dans ce travail, disposer d'une telle méthode nous aurait amené plus rapidement à prévoir quelles facettes du problème n'auraient pas du être prises en charge par un modèle de Bayes. Nous aurions alors gagné en efficacité.

Les réseaux bayésiens présentent un intérêt non contestable pour l'intelligence artificielle et l'informatique en général. Ils permettent de modéliser l'incertitude, souvent oubliée par les modèles mathématiques classiques. Ils sont en mesure d'apprendre un champ de connaissance de manière automatique tout en tenant compte de la fragilité des conclusions qui pourraient être déduites. Il sont en outre aisément extensibles et adaptables.
Cependant, lorsque leur utilisation pratique est envisagée, notre travail démontre qu'il est primordial d'évaluer préalablement et attentivement si la modélisation effectuée avec un réseau bayésien est pertinente et efficiente.

Bibliographie

[1] A. Becker, P. Leray, P. Naïm, O. Pourret, and P.-H. Wuillemin. *Réseaux bayésiens*. Editions Eyrolles, Paris, 2004.

[2] O. Ben Naceur-Mourali and C. Gonzales. Une unification des algorithmes d'inférence de pearl et de jensen. *Revue d'intelligence artificielle*, 18(2) :229–260, 2004.

[3] D. Blostein, J. Cordy, and R. Zanibbi. Recognizing mathematical expressions using tree transformation. *I.E.E. Transformation Pattern Analysis*, 24 :1455–1467, Nov. 2002.

[4] H. Boudali and J. Dugan. A temporal bayesian network framework. In *Presented at the Fourth International Conference on Mathematical Methods in Reliability (MMR-04)*, Santa Fe, 2004.

[5] K.-F. Chand and D.-Y. Yeung. Error detection, error correction and performance evaluation in on-line mathematical expression recognition. Technical report, The Hong Kong University of Science and Technology, Department of Computer Sciences, May 1999.

[6] B. Chaudhuri and U. Garain. Recognition of online handwritten mathematical expressions. *I.E.E. Transactions on systems, man and cybernetics*, 34(6), Dec. 2004.

[7] A. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2 :25–36, 1992.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, B(39) :1–38, 1977.

[9] M. Fairtlough. Introduction to formal grammars, the equivalence of regular grammars and finite automata. Extract of a lecture given at the Department of Computer Science, University of Sheffield, U.K.

[10] N. Friedman and M. Goldszmidt. Building classifiers using bayesian networks. In *AAAI/IAAI, Vol. 2*, pages 1277–1284, 1996.

[11] H. Fujisawa, N. Furukawa, H. Ikeda, M. Koga, and H. Sako. A context-free grammar-based language model for string recognition. *International Journal of Computer Processing of Oriental Languages*, 15(2) :149–163, 2002.

[12] Z. Ghahramani. Learning dynamic Bayesian networks. *Lecture Notes in Computer Science*, 1387 :168–197, 1998.

[13] J. A. Gámez. Abductive inference in bayesian networks : a review. Technical report, Computer Sciences Department, Castilla-La Mancha University, Spain, July 2003.

[14] F. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in recursive graphical models by local computations. *Computational Statistics Quarterly*, 4 :269–282, 1990.

[15] F. V. Jensen. *Bayesian Networks and Decision Graphs.* Statistics for Engineering and Information Science. Springer-Verlag New York, Inc., 2001.

[16] F. V. Jensen and T. D. Nielsen. Bayesian networks and decision graphs. To be published.

[17] J. H. Kim and J. Pearl. A computational model for combined causal and diagnostic reasoning in inference systems. In *Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 190–193, 1983.

[18] S. L. Lauritzen. The em algorithm for graphical association models with missing data. *Compututational Statistics Data Analysis*, 19(2) :191–201, 1995.

[19] A. L. Madsen and F. V. Jensen. A computational model for combined causal and diagnostic reasoning in inference systems. In *Fourteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1998.

[20] K. Mochida and M. Nakagawa. Separating figures, mathematical formulas and japanese text from free handwriting in mixed online documents. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(7) :1173–1187, 2004.

[21] J. W. Myers, K. B. Laskey, and K. A. DeJong. Learning bayesian networks from incomplete data using evolutionary algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 458–465, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[22] J. Pearl. *Probabilistic reasoning in intelligent systems.* Morgan Kaufmann Publishers, San Mateo, California, 1988.

[23] J. Pearl. *Causality : models, reasoning and inference.* Cambridge University Press, San Mateo, California, 2000. ISBN 0-521-77362-8.

[24] Y. Sakakibara. Learning context-free grammars using tabular representations. *Pattern Recognition*, 38(2005) :1372–1383, Mar. 2004.

[25] T. Sang, P. Beame, and H. Kautz. Solving bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1–10, 2005.

[26] S. Smithies, K. Novins, and J. Arvo. A handwriting-based equation editor. In *Graphics Interface*, pages 84–91, 1999. citeseer.csail.mit.edu/smithies99handwritingbased.html.

[27] I. Tsamardinos, C. F. Aliferis, A. Statnikov, and L. E. Brown. Scaling-up bayesian network learning to thousands of variables using local learning techniques.

[28] A. Tulupyev and S. Nikolenko. Directed cycles in bayesian belief networks : Probabilistic semantics and consistency, checking complexity. In A. Gelbukh, Álvaro Albornoz, and H. Terashima-Marín, editors, *Fourth Mexican International Conference on Artificial Intelligence (MICAI 2005)*, volume 3789, pages 214 – 223. Springer Berlin / Heidelberg, 2005.

[29] N. Zhang and D. Pool. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5 :301–328, 1996.

Annexe A

Logiciel interpréteur

A.1 Description générale

Dans le cadre du projet de reconnaissance de formules mathématiques manuscrites réalisé à l'Université d'Aalborg, au Danemark, nous avons développé un outil logiciel qui constitue une première implémentation de l'interpréteur défini dans le chapitre applicatif (chapitre 5).

Cet outil logiciel, qui intègre la plupart des idées formulées dans l'application permet la reconnaissance d'une formule mathématiques à partir des vraisemblances fournies par un classifieur, propose des corrections de plusieurs manières et intègre une analyse du comportement de l'utilisateur.

Cette annexe a pour objectif de décrire les caractéristiques du logiciel.

A.2 Version de base

La version de base du logiciel interpréteur a été développé en Java, à l'aide de l'outil Eclipse. Elle utilise la librairie Java du logiciel Hugin, qui offre la possibilité de construire des réseaux, de les utiliser et des les apprendre. Les caractéristiques du logiciel Hugin sont précisées dans l'annexe 2.
La version de base de l'interpréteur est dotée d'une interface graphique qui permet de naviguer dans les différentes options de l'application.

Dans un premier temps, nous allons présenter les grandes lignes de la modélisation conceptuelle et physique de l'application (diagramme de cas d'utilisation, diagramme de composants du système et diagramme de classes).

Nous exposerons ensuite l'implémentation concrète des deux versions de l'algorithme résolvant les problèmes non contextuels (approche exhaustive et approche dialogue).

Cette version du logiciel est en mesure de trouver la phrase la plus probable, d'apprendre un réseau à partir d'une base de données, d'utiliser plusieurs fois le même réseau et enfin d'exploiter l'algorithme de correction a posteriori des erreurs.

A.2.1 Modélisation

Comme dans tout développement de logiciel, la première démarche àă entreprendre avant d'implémenter une application est d'évaluer les besoins des utilisateurs de cette application afin de s'assurer que cette dernière respectera les attentes de ceux qui l'utiliseront.

Dans le but de spécifier le logiciel, une analyse complète des besoins a été effectuée. Dans l'ordre, nous exposons d'abord un diagramme de cas d'utilisation pour évaluer les différents scénarios permis, puis un diagramme de composants pour évaluer l'environnement physique de notre application et enfin un diagramme de classes afin de mettre en place l'architecture logicielle de l'application.

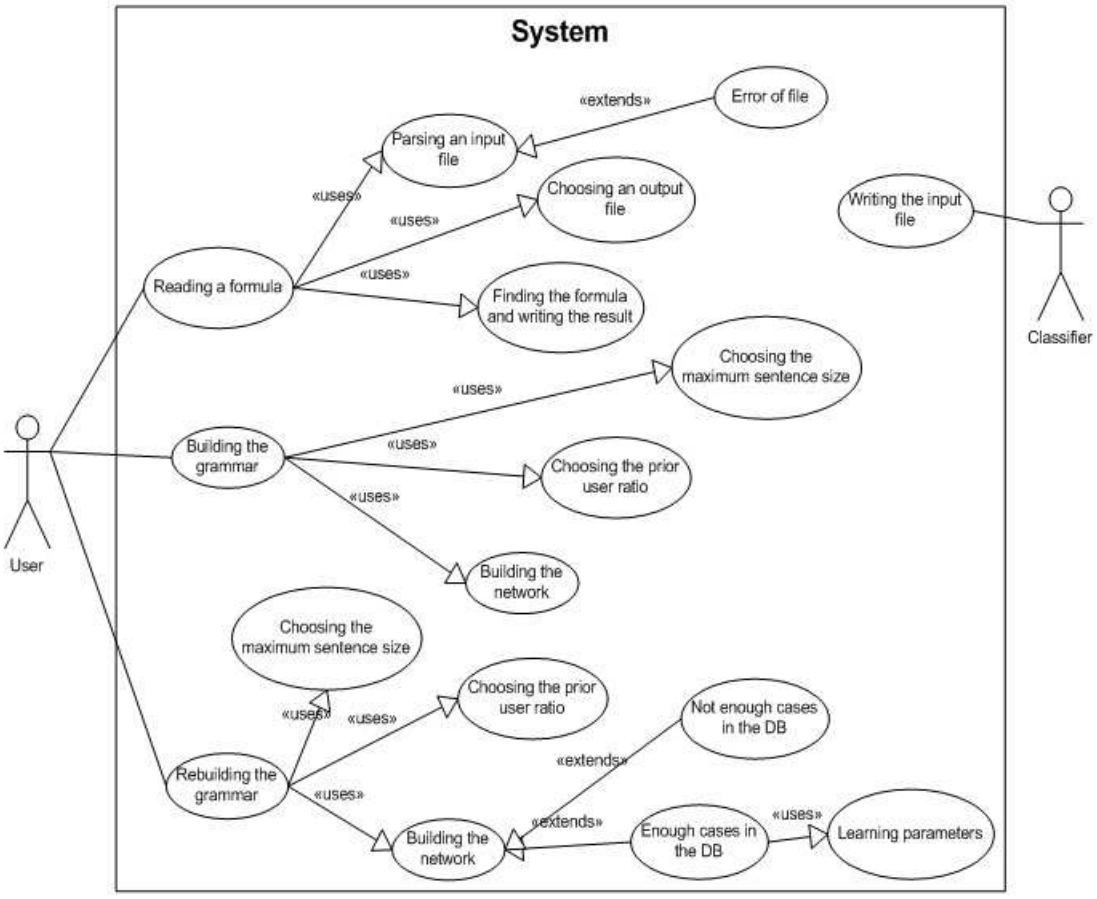


FIG. A.1 – Diagramme de cas d'utilisation du système

Ce schéma (A.1) illustre les services offerts par la version de base du logiciel interpréteur aux utilisateurs.

En effet, ce dernier a la possibilité, par l'application, de construire un réseau en choisissant la taille maximum de phrase que le réseau peut accepter. Il s'agit du nombre de noeuds "*Caractère*" du réseau.
Une fois ceci effectué, l'utilisateur peut calculer autant de fois qu'il veut la phrase la plus probable en choisissant un fichier en entrée contenant les vraisemblances du classifieur et en sélectionnant le répertoire dans lequel le fichier de résultats sera enregistré.
Pour finir, l'utilisateur peut également exécuter une opération d'apprentissage sur la base de données, décidant si la base de données doit être vidée après l'apprentissage ou non. La base de données est ici constituée des différentes phrases les plus probables validées par le réseau Bayésien construit au lancement du logiciel.

Le diagramme de la figure A.2 permet de distinguer différents composants (groupes d'activités) que le logiciel interpréteur met en oeuvre et de spécifier, pour certaines activités, les systèmes externes qu'elles utilisent. Il s'agit aussi d'identifier les différents flux entre les composants ainsi que leurs dépendances.

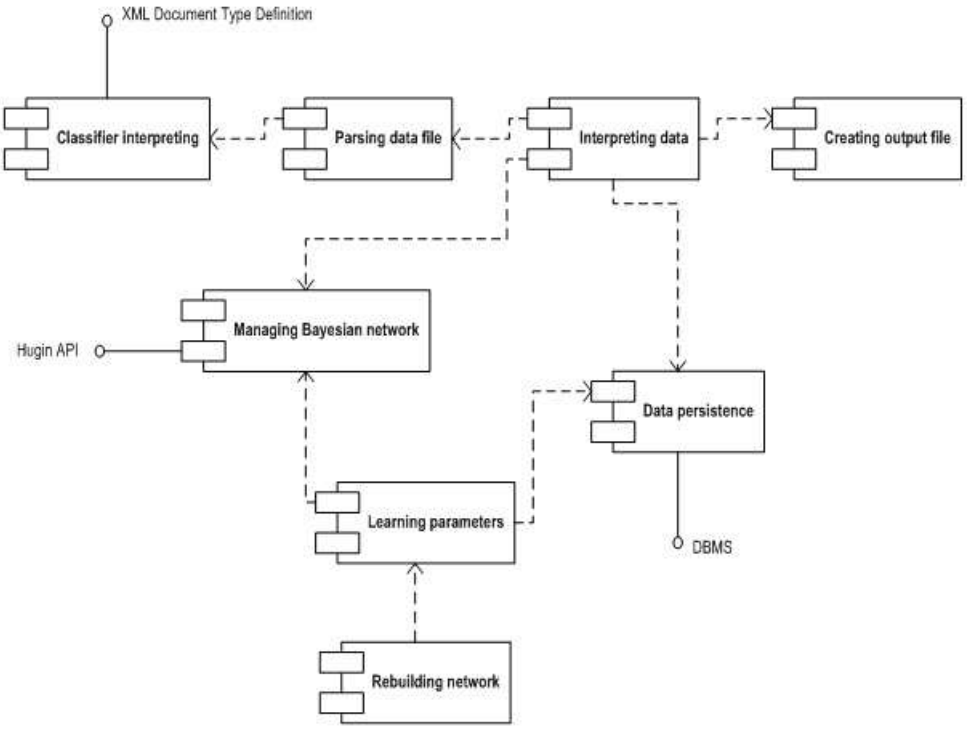


FIG. A.2 – Diagramme de composants du système

Le fichier en entrée contenant les vraisemblances du classifieur est écrit en XML (la *document type definition* est exposée plus loin). Le logiciel utilise également une base de données pour enregistrer les cas qui, en l'espèce, est un ensemble de fichiers textes. Enfin, le logiciel fait appel à l'API Java de Hugin pour gérer le réseau Bayésien utilisé.

L'architecture du logiciel est divisé en trois blocs (*packages*) : l'ensemble des classes définissant l'interface, l'ensemble des classes gérant l'interface et l'ensemble des classes offrant des services et des fonctions appelés par le gestionnaire d'interface.
L'architecture de l'ensemble des classes offrant des services est exposée dans la figure A.3. Le pattern d'architecture utilisé est un pattern orienté objet dont les objets principaux sont : les fichiers en entrée, les caractères définis dans chaque fichier en entrée, les réponses de l'interpréteur pour chaque fichier en entrée, l'interpréteur lui-même, le réseau Bayésien et l'information contenue dans la base de données.

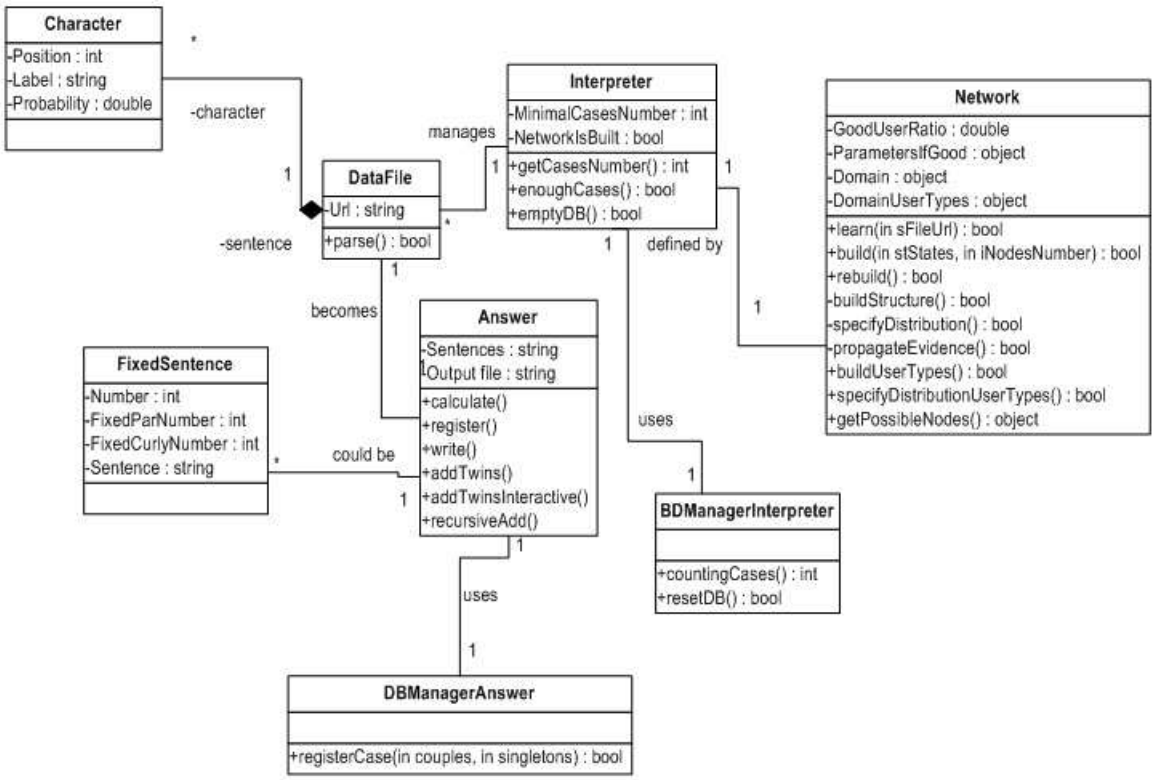


FIG. A.3 – Diagramme de classes du système

Enfin, dans la figure A.4, se trouve le contenu du fichier de *document type definition* utilisé pour les fichiers XML contenant les vraisemblances du classifieur. Un exemple simple de fichier en entrée est donné en A.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT sentence (character*)>
<!ELEMENT character (a, b, blank)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
<!ELEMENT blank (#PCDATA)>
<!– a, b et blank sont des réels –>
```

FIG. A.4 – *Document type definition* des fichiers XML en entrée

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sentence SYSTEM "classifier.dtd">
<sentence>
<character>
<b>0.69</b>
<a>0.31</a>
</character>
<character>
<a>0.9</a>
<b>0.1</b>
<blank>0</blank>
</character>
</sentence>
```

FIG. A.5 – Exemple de fichier XML d'entrée définissant une phrase de deux caractères

A.2.2 Algorithmique des problèmes non contextuels

Comme mentionné antérieurement, la version de base du logiciel interpréteur utilise l'algorithme a posteriori pour la correction des fautes.
Cependant, il y a deux approches de cet algorithme : la version complète qui utilise tel quel l'algorithme, définie dans la section "Approche exhaustive" et la version interactive qui essaie d'améliorer l'efficacité du système, en demande l'avis de l'utilisateur. La version interactive est décrite dans la section "Approche dialogue".

Le problème de la version complète est qu'elle génère toutes les possibilités. Même pour une phrase réduite contenant quelques parenthèses et accolades manquantes, le nombre de possibilités est grand et croît de manière exponentielle en fonction du nombre de caractères manquants que nous essayons de gérer.

Afin de diminuer cette complexité, il est possible de mettre le système dans un mode interactif, suscitant ainsi l'avis de l'utilisateur et générant une seule phrase par exécution.

FIG. A.6 – Fenêtre principale de la première version de l'interpréteur

Approche exhaustive

Pour voir les potentialités de l'application, examinons les les réponses fournies lorsque l'utilisateur écrit une phrase telle que $\{\{(a*a+b:\{\{(a*a+b)\}\}, \{\{(a*a)+b\}\}, \{\{(a*a)\}+b\}, \{\{(a*a)\}\}+b, \{\{(a)*a+b\}\}, \{\{(a)*a\}+b\}, \{\{(a)*a\}\}+b, \{\{(a)\}*a+b\}, \{\{(a)\}*a\}+b, \{\{(a)\}\}*a+b$.

Dans cet exemple pourtant simple, le nombre de réponses possibles générées est déjà considérable. Ceci prouve que le nombre de possibilités croît exponentiellement et qu'il dépend bien du nombre de caractères manquants que le logiciel interpréteur accepte de prendre en compte (ce nombre est spécifié au préalable avant construction du réseau), du nombre effectif de caractères manquants et du nombre de positions où il est possible d'ajouter un caractère.

Approche interactive

Dans ce mode, pour chaque caractère manquant, le système génère une solution à la suite de l'autre, dans l'ordre des différentes positions où il est possible de rajouter un caractère. Il demande à l'utilisateur de faire un choix après chaque proposition.

Si l'utilisateur est d'accord avec la position proposée pour le caractère manquant en cours de lecture, le logiciel enregistre la modification de la phrase et remplace la phrase originelle fausse par une nouvelle qui contient un caractère manquant en moins. Le logiciel essaie alors de résoudre le problème suivant si tant est qu'il y en a encore à résoudre.
Si l'utilisateur n'est pas d'accord, une deuxième possibilité est offerte pour le caractère en cours de lecture via un changement de position du caractère et cela jusqu'à ce que la position qui convient à l'utilisateur soit trouvée.

Dans la plupart des cas, la phrase que l'utilisateur essaye d'écrire est trouvée plus vite que dans la version exhaustive. En effet, demander à l'utilisateur de faire des choix permet d'obtenir rapidement et sans équivoque la phrase corrigée que celui-ci désirait écrire.

En A.7, se trouve une capture d'écran de la fenêtre interactive qui demande l'avis de l'utilisateur.

A.3 Extension

Le logiciel présenté dans la section précédente a été étendu afin d'intégrer le type d'utilisateur. En effet, comme il a été expliqué dans le chapitre 5, on considère le comportement de l'utilisateur à travers l'existence de plusieurs types d'utilisateurs.

Le logiciel étendu utilise un second réseau Bayésien afin de calculer la probabilité du type d'utilisateur, en positionnant les vraisemblances sur les noeuds "*Caractère*" de ce réseau, pour chaque formule écrite.

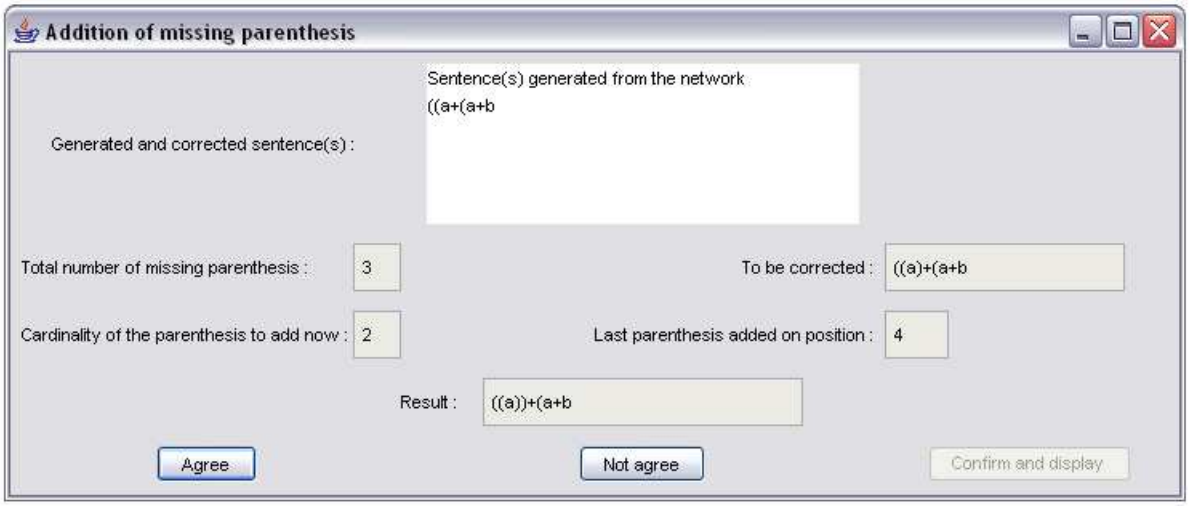


FIG. A.7 – Fenêtre interactive pour l'approche dialogue

Plus les phrases écrites par l'utilisateur sont correctes, plus ce ratio augmente, signifiant ainsi que l'utilisateur est de type "Bon". Le nouveau ratio obtenu est alors utilisé pour mettre à jour la grammaire, qui de cette façon, évolue.

Concernant le ratio de type d'utilisateur, notons que lors de la première utilisation du réseau, quand l'utilisateur veut construire ou apprendre un nouveau réseau, le logiciel l'interroge sur le ratio de base qu'il désire. La valeur par défaut est de 0.9 pour le type d'utilisateur "Bon".

Les phrases enregistrées dans la base de données sont les phrases non corrigées, ce qui implique que des phrases fausses peuvent y être trouvées et que la grammaire peut évoluer dynamiquement lorsqu'on effectue l'opération d'apprentissage.

Annexe B

Logiciel Hugin

B.1 Présentation

Dans le cadre de l'application de reconnaissance de texte développée dans ce travail, dont l'étude est faite dans le chapitre 5, nous avons utilisé le logiciel Hugin, dont la licence nous a été temporairement accordée par l'Université d'Aalborg. Ce logiciel nous a permis d'une part de mener à bien cette étude et de tester nos résultats. D'autre part, il a été utilisé pour mettre au point l'outil logiciel interpréteur, défini dans l'annexe 1.

Selon [1], Hugin est l'outil probablement le plus connu et le plus utilisé commercialement. Les principales fonctions offertes par cet outil sont les suivantes :

- Construction de bases de connaissances fondées sur des réseaux bayésiens ou des diagrammes d'influence.
- Développement de réseaux bayésiens "orientés objets".
- Apprentissage de structure et de paramètres.

Hugin est principalement fourni sous forme d'un environnement graphique mais aussi sous forme d'un environnement de développement permettant de piloter l'ensemble des fonctions de définition, d'inférence et d'apprentissage à partir d'une application Java, C ou Visual Basic.
Il est édité par la société Hugin Expert A/S, société danoise créée en 1989 et basée à Aalborg au Danemark. Son principal domaine d'application est le diagnostic dans le domaine médical.

B.2 Construction de modèles

La création d'un réseau bayésien avec Hugin s'effectue sous un environnement graphique simple et assez intuitif. Cette interface permet de gérer plusieurs types de noeuds : noeud discret, noeud continu, noeud d'utilité et noeud de décision. Cette création de modèles présente cependant certaines contraintes :

- Elle ne permet de gérer que des noeuds continus gaussiens.
- Un noeud continu ne peut pas être parent d'un noeud discret.

– On ne peut pas utiliser dans le même modèle des noeuds continus et des noeuds d'utilité ou de décision.

En général, autant pour un noeud discret que pour un noeud continu, la construction du réseau s'effectue grâce à un éditeur graphique qui permet de définir à la fois l'architecture du modèle et les tables de probabilités d'un noeud conditionnellement à ses parents. La particularité des noeuds continus est que leur distribution est gaussienne. La figure B.1 montre la capture d'un environnement graphique Hugin "standard".

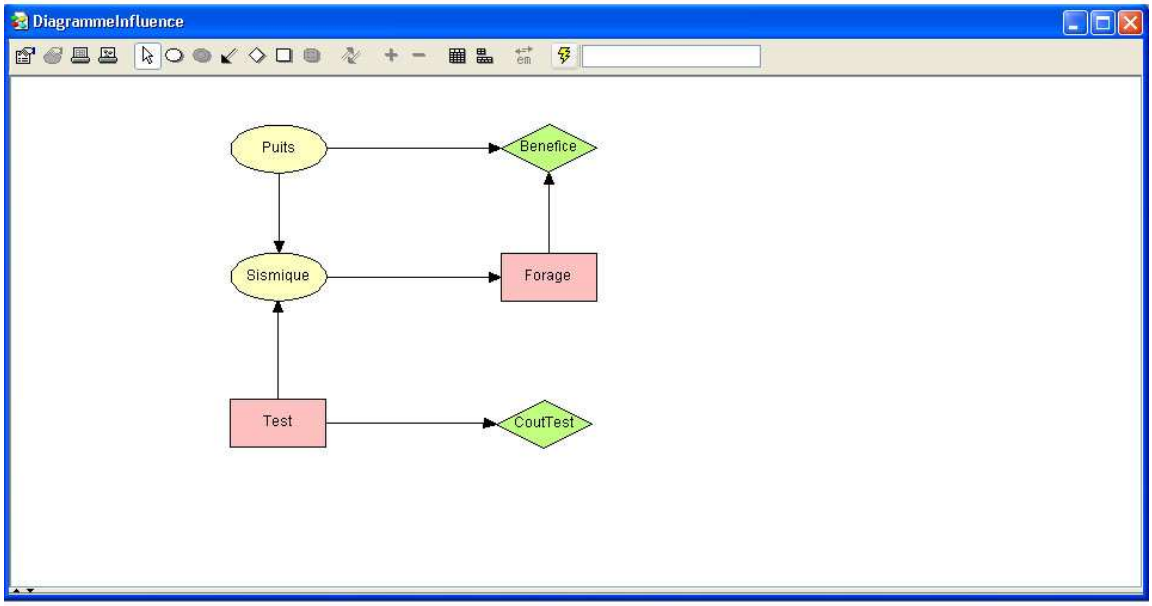


FIG. B.1 – Capture d'écran d'un environnement graphique standard Hugin

En ce qui concerne le diagramme d'influence, il s'agit d'un réseau bayésien auquel on a ajouté des noeuds de décision et d'utilité. Il est illustré par un exemple qui décrit une prise de décision dans le domaine du forage pétrolier, cité dans [1].

Supposons qu'un ingénieur doive choisir ou non de creuser à un certain point. Il ne connaît pas la quantité de pétrole éventuellement présente. Le puits peut être sec, humide, ou inhibé de pétrole. Pour avoir une information complémentaire, l'ingénieur décide de faire une mesure d'écho sismique sur le puits.
Le résultat du test indiquera que la structure du terrain est fermée (ce qui est un bon signe de présence de pétrole), ouverte (moyen), ou sans structure (présence de pétrole improbable).
La structure des coûts est établie de la manière suivante : le test sismique coûte 10000$, creuser coûte 70000$, la recette attendue si le puits est imbibé est de 270000$, de 120000$ s'il est humide, et de 0$ s'il est sec. Enfin, bien entendu, si l'ingénieur décide de ne pas creuser, la recette attendue est nulle.

Hugin permet de modéliser le problème grâce au diagramme d'influence suivant. La première décision est d'effectuer ou non le test sismique. Si on décide de faire ce test, le résultat obtenu

sera fonction de la configuration réelle du puits, avec une certaine incertitude.

A partir du résultat du test sismique, on décidera de creuser ou non. Le diagramme d'influence permet de guider la décision, car il indique l'utilité espérée de chaque décision. La figure B.2 montre la capture d'écran d'un diagramme d'influence sous Hugin.

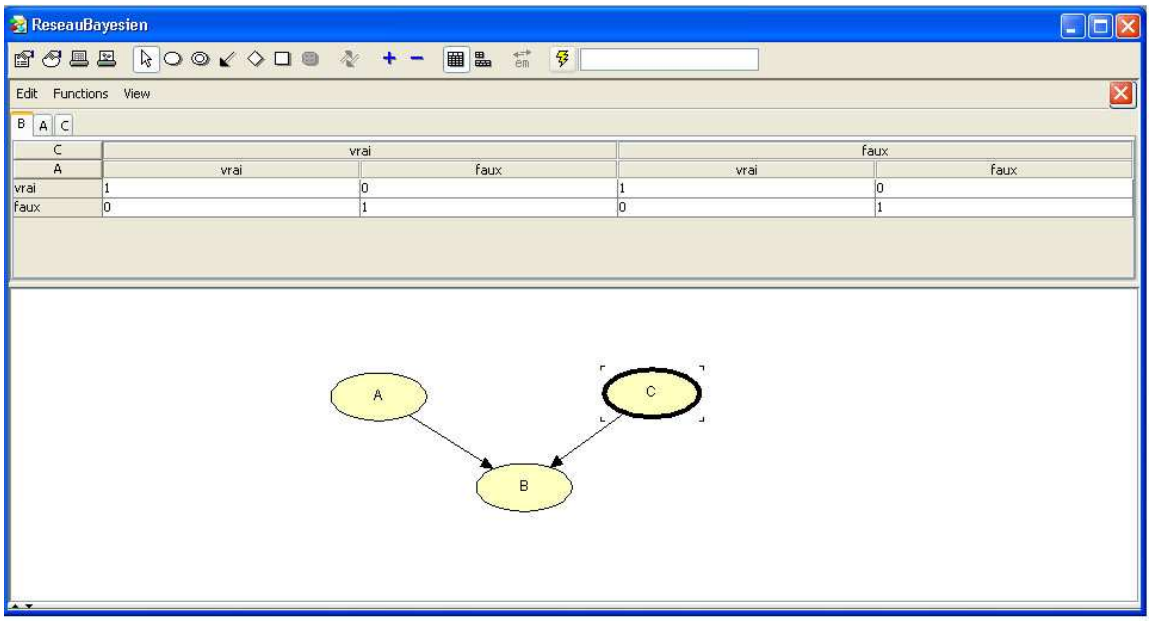


FIG. B.2 – Capture d'écran d'un diagramme d'influence sous Hugin

B.3 Inférence

L'inférence dans Hugin s'effectue grâce au calcul d'un arbre de jonction sur le réseau. Le mode le plus simple d'inférence consiste à entrer des observations dans le réseau, simplement en cliquant sur la valeur observée.
Hugin permet aussi de saisir des vraisemblances, grâce à la fonction "*Insert likelihood*" (figure B.3).

Le type d'inférence standard, c'est à dire le calcul de la probabilité des noeuds non observés conditionnellement aux observations, s'appelle la propagation "Sum normal" dans Hugin. Il utilise l'algorithme de somme-propagation de Jensen-Olesen.
Hugin offre d'autres modes d'inférence, comme le mode "*Max normal*" qui correspond à l'algorithme de maximum-propagation et permet de trouver la configuration du réseau la plus probable, ayant effectué certaines observations.

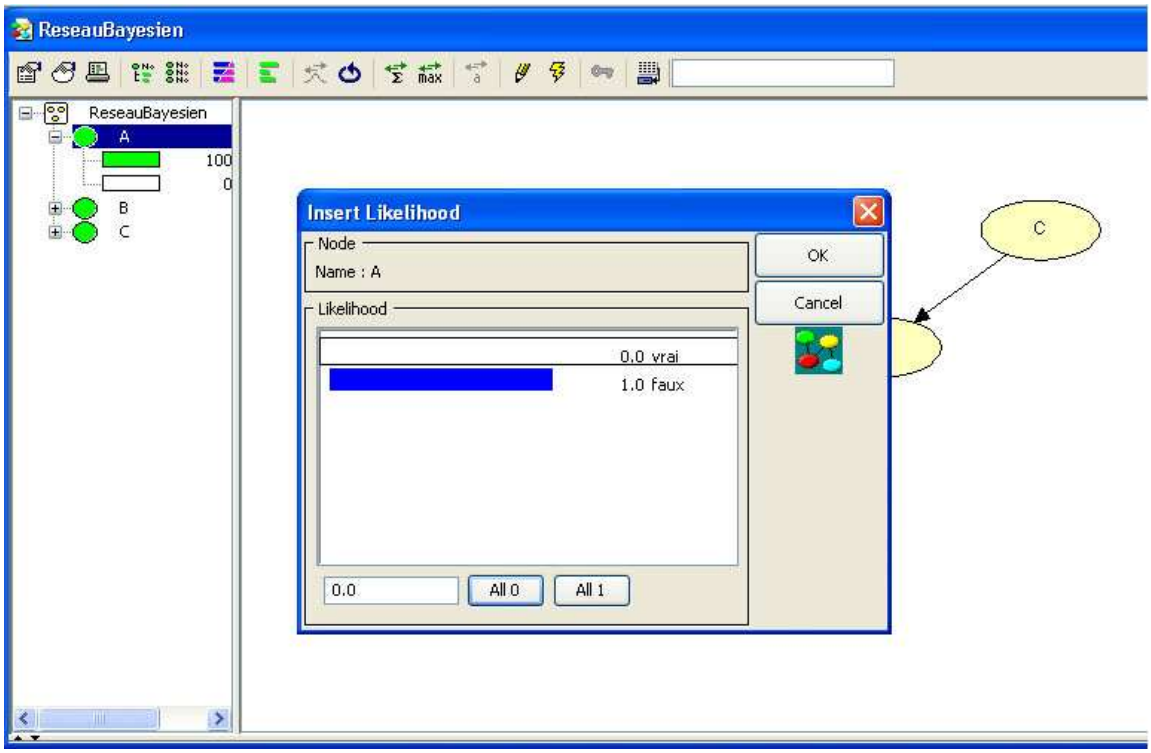


FIG. B.3 – Capture d'écran d'une inférence sous Hugin

B.4 Apprentissage

Hugin permet l'apprentissage de paramètres mais aussi de structure à partir de deux algorithmes. Cette fonction est présentée comme un assistant, en plusieurs étapes :

- Acquisition des données = choix d'un fichier ou d'une table de base de données.
- Prétraitement des données = sélection des entrées, discrétisation, etc.
- Contraintes structurelles = ici l'utilisateur peut spécifier manuellement les dépendances ou indépendances connues entre les variables.
- Apprentissage : choix de l'algorithme PC ou NPC.
- Résolution des incertitudes = l'utilisateur est sollicité dans le cas où certaines liens, ou certaines orientations des liens n'ont pu être établies par l'algorithme.
- Sélection des liens = l'utilisateur peut visualiser la signification de chacun des liens, et sélectionner ceux qui dépassent un certain seuil.
- Distribution a priori = si une information sur la distribution des données est connue, on peut l'indiquer à ce stade, ainsi que le nombre d'exemples sur lesquels cette information a été obtenue.
- Apprentissage EM = c'est la dernière étape, au cours de laquelle les tables de probabilités du réseau sont apprises grâce à l'algorithme d'Expectation Maximisation.

L'apprentissage des paramètres, c'est-à-dire des tables de probabilités peut s'effectuer à tout moment sur un réseau existant. Deux options existent pour cet apprentissage :

- L'apprentissage séquentiel, aussi appelé adaptation, permet de modifier la distribution du réseau à partir de chaque exemple observé.
- L'apprentissage global permet de recalculer les tables de probabilités du réseau à partir d'un ensemble d'exemples.

Signalons enfin que Hugin peut également être utilisé pour générer des bases de cas aléatoires à partir d'un réseau entièrement défini.

B.5 Compléments

Une fonctionnalité intéressante de Hugin est sa possibilité de gérer des réseaux imbriqués, appelés "réseaux orientés objets". Il s'agit d'insérer une instance d'un réseau déjà créé au sein d'un nouveau réseau, en le représentant par un seul noeud.

Hugin offre également une API, c'est-à-dire une interface programmeur, complète. Cette API est disponible en C/C++, Java et Visual Basic. C'est la version Java que nous avons utilisée pour le développement de notre outil logiciel. Une documentation complète quoique parfois élémentaire est disponible avec ce logiciel, sous licence d'utilisation stricte.

B.6 Observations

Hugin est aujourd'hui l'un des produits les plus robustes et les plus simples à utiliser pour construire des réseaux bayésiens. Il dispose d'algorithmes puissants et est très facile à intégrer dans des applications existantes.