Un calcul de Viterbi pour un Modèle de Markov Caché Contraint

Matthieu Petit, Christiansen Henning

Un calcul de Viterbi pour un Modèle de Markov Caché Contraint

Matthieu Petit * **Henning Christiansen**

Research group PLIS : Programming, Logic and Intelligent Systems
Department of Communication, Business and Information Technologies
Roskilde University, P.O Box 260, DK-4000 Roskilde, Denmark
{petit,henning}@ruc.dk

Résumé

Un modèle de Markov caché (MMC) est un modèle statistique permettant de représenter un processus de Markov dont l'état est non observable. Ce modèle a été/est largement utilisé pour adresser des problèmes liés à la reconnaissance vocale ou l'analyse de séquences en bio-informatique. L'algorithme de Viterbi permet de calculer la valeur la plus probable des états cachés du processus étant donné des données observables. Un MMC contraint étend ce cadre de travail en contraignant l'exécution d'un processus d'un MMC. Ce paradigme permet, de manière déclarative, de spécifier le comportement attendu d'un MMC.

Dans cet article, nous définissons un MMC contraint dans le cadre de la Programmation par Contraintes. Nous proposons une nouvelle version de l'algorithme de Viterbi pour ce cadre de travail. Plusieurs techniques issues de la Programmations par Contraintes sont utilisées pour accélérer la recherche de la valeur la plus probable des états cachés du processus. Une implémentation utilisant PRISM, un langage de programmation logique pour des modèles statistiques, est présentée.

Abstract

A hidden Markov model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with hidden states. This model has been widely used in speech recognition and biological sequence analysis. Viterbi algorithm has been proposed to compute the most probable value of these hidden states in regards to an observed data sequence. Constrained HMM extends this framework by adding some constraints on a HMM process run.

In this paper, we propose to introduce constrained HMMs into Constraint Programming. We propose new version of the Viterbi algorithm for this new framework. Several constraint techniques are used to reduce the search of the most probable value of hidden states of a constrained HMM. An implementation based on PRISM, a logic programming language for statistical modeling, is presented.

*Ce travail est supporté par le projet "Logic-statistic modeling and analysis of biological sequence data" financé par le programme NABIIT sountenu par "the Danish Strategic Research Council".

1 Introduction

Un modèle de Markov caché (MMC) est un modèle statistique permettant de représenter un processus de Markov pour lequel certains états sont non observables. Cependant, de l'information sur ces états cachés peut être déduite de données observées. En effet, la fréquence d'observation de ces données dépend de l'état du processus dans lequel celles-ci sont émises. Ce modèle a prouvé son utilité pour adresser de nombreux problèmes en reconnaissance vocal [9] et en analyse de séquences biologiques [3]. Cette large utilisation de ce modèle est en partie due aux nombreux algorithmes efficaces permettant :

- le calcul de la séquence la plus probable d'états cachés étant donné des données observées, appelé *chemin de Viterbi* ;
- de faire évoluer le MMC afin de refléter un ensemble de données par apprentissage.

Dans [10], une extension des MMC, appelée modèle de Markov caché contraint, a été proposée par Sato et al.. Ce modèle a pour objectif d'ajouter des contraintes sur les états cachés pouvant être visités et les données émises lors de l'exécution d'un processus de MMC. Cette approche étend l'expressivité de ce modèle en exprimant par des contraintes un ensemble

d'exécutions valides pour un MMC. Par exemple considérons un MMC composé de 100 états cachés. Un MMC contraint permet de spécifier que seulement un dixième de ces états doivent être utilisés lors de l'exécution d'un processus. Dans ce cadre de travail, un algorithme basé sur le calcul de vraisemblance maximale ajusté aux échecs a été proposé dans [10]. Cet algorithme permet par apprentissage de faire évoluer les différents paramètres d'un MMC contraint afin de refléter un ensemble de données observées.

Un certain nombre d'extensions probabilistes de la Programmation par Contraintes ont déjà été proposées. Ces extensions ont pour principal but la modélisation de paramètres incertains dans ce paradigme de programmation [13]. Le caractère aléatoire de ces évènements est modélisé par un choix probabiliste. Dans [4], une probabilité est associée à chaque contrainte d'un problème de satisfaction de contraintes. Cette probabilité caractérise la fréquence avec laquelle la contrainte va être ajouter au problème. La programmation concurrente par contraintes est étendu dans [2, 5] par des opérateurs de choix probabilistes. Ces opérateurs sont utilisés pour modéliser l'aléa dans l'exécution de ce type de processus. Dans [15, 11], Walsh et al. définissent un nouveau cadre de travail, appelé la programmation stochastique à contraintes, permettant de représenter comme un problème à contraintes des problèmes issus de la programmation stochastique. Dans [1], un modèle graphique est proposé par Dechter et al. pour modéliser la recherche de solutions de problèmes à contraintes comportant entre autre des paramètres probabilistes. Récemment dans [8, 7], nous avons défini des contraintes dites de choix probabiliste permettant de raisonner avec des choix probabilistes partiellement définis.

Dans cet article, nous proposons d'introduire les MMC contraints dans le cadre de la Programmation par Contraintes. Cela nous permet de formuler des problèmes classiques associés aux MMC comme des problèmes à contraintes. Plus précisément, nous proposons de modéliser le calcul du chemin de Viterbi comme un problème d'optimisation. Notre objectif est de bénéficier de différentes techniques issus de la Programmation par Contraintes pour effectuer efficacement ce calcul.

Plan de l'article. L'article suit l'organisation suivante : section 2 introduit la notion de MMC contraint à l'aide d'un exemple. La section 3 décrit les pré-requis nécessaires concernant les MCC pour une bonne compréhension de l'article. Dans la section 4, les MCC contraints sont formellement décrits dans le cadre de la Programmation par Contraintes et un calcul de Viterbi est présenté dans ce cadre de travail section 5. Une première implémentation de ce modèle probabiliste, basé sur PRISM, est décrite section 6. Finalement, la section 7 en présentant nos travaux futures à ce sujet conclut l'article.

2 Exemple illustratif

Dans cette section, le cadre de travail des MMC contraints est illustré sur un MMC abstrait. Ce MMC est présenté par FIGURE 1. Le modèle de Markov est composé de quatre états cachés a, b, c et d. Chacun de ces états peut émettre le chiffre 1 ou 2. Les probabilités de transiter d'un état à un autre ou d'émettre 1 ou 2 étant donné un état sont représentées par des arcs étiquetés. Une exécution du processus commence dans l'état *start*, est composée d'une séquence d'états cachés et d'émissions et se termine lorsque l'état *end* est atteint. L'algorithme de Viterbi permet de calculer la séquence la plus probable d'états cachés étant donné une séquence d'émissions connue. Par exemple, le chemin de Viterbi pour la séquence d'émissions 1 1 1 2 est $start\, a\, a\, b\, d\, end$. Un algorithme proposé par Viterbi dans [14] issu de la programmation dynamique permet d'effectuer ce calcul efficacement. Cet algorithme sera détaillé dans la section 3

Des contraintes peuvent être ajoutées sur ce modèle pour spécifier des conditions devant être satisfaites par celui-ci. Par exemple, l'exécution du MMC abstrait peut être contraint à satisfaire : si l'état a a été visité lors de l'exécution, alors l'état d ne peut plus être visité. Afin de représenter cette contrainte, une approche possible consiste à modifier la structure initial du MMC. En effet, en dupliquant les états b et c, il est possible d'interdire pour les chemins ayant transité par a de visiter d. L'approche suivie dans cet article consiste simplement à prendre en compte ces contraintes sur le MMC sans changer la structure initiale du modèle. Le MMC abstrait peut-être étendu par la contrainte que pour toutes exécutions du MMC abstrait $(start\, s_1 \dots s_n\, end\, , e_1 \dots e_n)$

$$s_i = a \implies \forall j > i, s_j \neq d.$$

où $start\, s_1 \dots s_n\, end$ est une séquence d'états cachés et $e_1 \dots e_n$ une séquence d'émissions. Cependant, notons que le calcul Viterbi doit prendre en compte le fait que certaines des exécutions peuvent ne satisfaire pas cette contrainte. Ainsi par exemple, le chemin de Viterbi $start\, a\, a\, b\, d\, end$ calculé pour la séquence d'émissions 1 1 1 2 ne satisfait pas la contrainte énoncée ci-dessus. Pour le MMC abstrait, le chemin de Viterbi pour 1 1 1 2 et satisfaisant la contrainte de ne pas visiter d après avoir visité a est $start\, a\, a\, a\, b\, end$

Les MMC contraints permettent de modéliser différents problèmes tout en conservant la structure initiale du MMC. Il est possible par exemple de contraindre

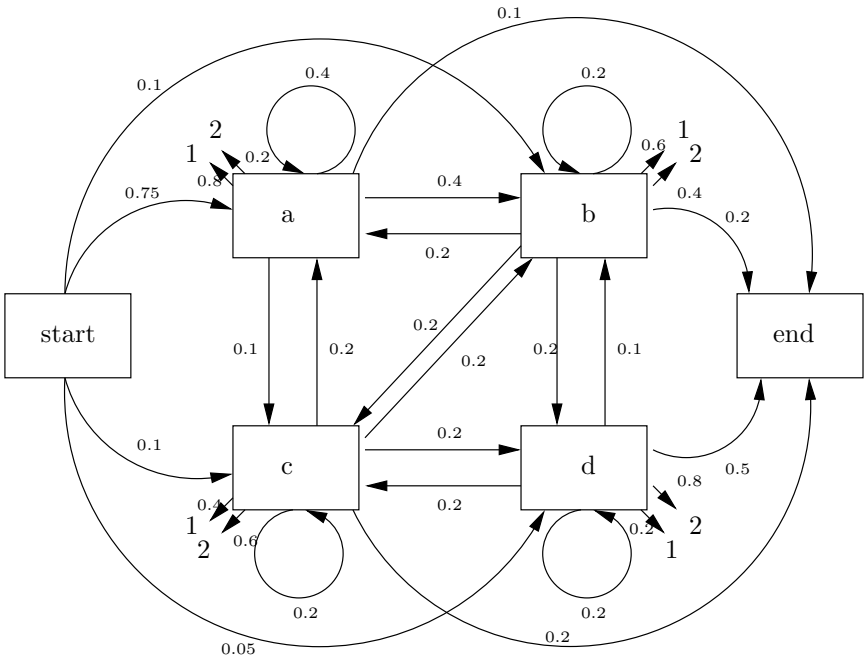

FIGURE 1 – Un MMC abstrait

la somme des émissions à être inférieures à une valeur donnée ou contraindre le nombre de fois qu'un état du modèle est visité à ne pas dépasser une borne maximale. Ces contraintes peuvent exprimer ainsi : pour toutes exécutions $(starts_1 \dots s_n end, e_1 \dots e_n)$

$$\sum_{i=1}^{n} e_i < valeur_1$$

ou

$$cardinality\_atmost(valeur_2, [s_1, \dots, s_n], s).$$

Cependant, ce nouveau cadre de travail nous oblige à adapter les algorithmes classiques associés aux MMC. En effet, ce calcul doit prendre compte le fait que certaines exécutions pouvent ne pas satisfaire le problème à contraintes.

3 Pré-requis sur les Modèles de Markov Cachés

Dans cette section, nous proposons d'introduire les pré-requis nécessaires sur les MMC pour une bonne compréhension du reste de l'article. Une description de l'algorithme de Viterbi est donnée.

3.1 Modèle de Markov Caché

Afin de simplifier la définition formelle de ce modèle, nous nous limitons dans cet article à des MMC n'émettant qu'une seule lettre par transition et n'ayant qu'un unique état initial. La généralisation de ce modèle à des modèles pouvant émettre un nombre arbitraire de lettres (zéro ou plus) et ayant plusieurs états initiaux est simple. L'ordre du processus de Markov est aussi limité au premier ordre et seulement des processus discrets sont considérés.

Définition 1. Un modèle de Markov caché (MMC) est un 4-uplet $\langle S, A, T, E \rangle$ où

- $S = \{s_1, \dots, s_n\}$ est un ensemble d'états parmi lesquels sont distingués : un état initial et des états finaux ;
- A est un ensemble fini de symboles appelés émissions ;
- T est un ensemble de probabilités de transition $\{p(s_i; s_j)\}$ représentant (en excluant l'état initial pour s_j et les états finaux pour s_i) la probabilité de transiter de s_i à s_j. De plus, pour chaque $s_i \in S$, $\sum_{s_j \in S} p(s_i; s_j) = 1$.
- E est un ensemble de probabilités d'émission $\{p(s_i; e_j)\}$ représentant (en excluant l'état initial et les états finaux pour s_i) la probabilité d'émettre e_j de l'état caché s_i. De plus, pour chaque état s_i (en excluant l'étant initial et les états finaux), $\sum_{e_j \in A} p(s_i; e_j) = 1$.

Les données observées lors de l'exécution d'un modèle Markovien caché prennent la forme de séquences d'émissions. Une exécution de ce modèle est composée d'une séquence d'états cachés et d'émissions. Étant donné une séquence d'émissions, une séquence d'états

cachés permettant cette émission est appelée un *chemin.*

Définition 2 (Exécution d'un processus de MMC)**.** Soit $\langle S, A, T, E\rangle$ un modèle de Markov caché, une exécution d'un processus de MMC est composée par une séquence $start\, s_1\, s_2 \ldots s_k\, end$ d'états cachés et une séquence $e_1\, e_2 \ldots e_k$ d'émissions satisfaisant les conditions suivantes :

- $p(start; s_1) \neq 0$ (probabilité de transiter vers s_1 de l'état initial) ;
- $\forall i > 1, p(s_{i-1}; s_i) \neq 0$ (probabilité de transiter entre deux états cachés) ;
- $\forall i, p(s_i; e_i) \neq 0$ (probabilité d'une émission d'un état caché).

Dans le cadre des MMC, plusieurs algorithmes ont été proposés pour donner efficacement une réponse à plusieurs types de questions :

1. Quelle est la probabilité d'observer une séquence d'émissions ?
2. Quel est le chemin le plus probable associé à une séquence d'émissions ?
3. Est-il possible d'adapter les paramètres du MMC afin que ceux-ci reflètent un ensemble de données observées ?

Dans cet article, nous nous focalisons à donner une réponse à la deuxième question dans le cadre des MMC contraints.

3.2 Calcul du chemin de Viterbi pour un MMC

L'algorithme Viterbi [14] est un algorithme issu de la programmation dynamique permettant de déterminer un chemin le plus probable associé à une séquence d'émissions $e_1 \ldots e_k$ pour un MMC. Cet algorithme se base sur un calcul itératif pour chaque état caché s_l du chemin le plus probable atteignant cet état à l'étape i de l'exécution d'un processus Markovien. On note par la suite cette probabilité $p(e_1 \ldots e_i)_{s_l}$. Cette itération est effectuée chaque étape de l'exécution du processus Markovien. L'algorithme est donc de complexité linéaire par rapport la taille de la séquence d'émissions.

Soit $\langle S, A, T, E\rangle$ un modèle de Markov et $e_1 \ldots e_n$ une séquence d'émissions. La probabilité du chemin le plus probable atteignant l'état s_l à l'étape i peut être calculée récursivement selon la formule suivante :

$$p(e_1 \ldots e_i)_{s_l} = \max_{s_k \in S} (p(e_1 \ldots e_{i-1})_{s_k} \cdot p(s_k; s_l) \cdot p(s_l; e_i)) \quad (1)$$

Si l'état caché s_m permet de maximiser $p(e_1 \ldots e_{i-1})_{s_k} \cdot p(s_k; s_l) \cdot p(s_l; e_i)$, le chemin le plus probable atteignant s_l à l'étape i est composé du chemin le plus probable atteignant s_m à l'étape $i-1$ et de la transition entre s_m et s_l.

Basé sur le calcul présenté ci-dessus, l'algorithme de Viterbi est décrit par l'**Algorithme** 1. Il prend comme entrées un MMC $HMM = \langle S, A, T, E\rangle$ et une séquence d'émissions $e_1 \ldots e_n$, et calcule le chemin de Viterbi $Viterbi\_path$ et la probabilité de ce chemin p_{max}.

Algorithme 1 : Algoritme de Viterbi

Entrée : HMM,$e_1 \ldots e_n$
Sortie : $Viterbi\_path$,p_{max}

$(Paths$,$Probas) \leftarrow$ `initialize(`e_1, HMM`)`;
$E \leftarrow \{e_1\, e_2, \ldots, e_1 \ldots e_n\}$;
for $e_1 \ldots e_i \in E$ **do**
  foreach $s_l \in S$ **do**
    $(p(e_i)_{s_l}, s_m) \leftarrow$
    $\max_{s_k \in S}(p(e_1 \ldots e_{i-1})_{s_k} \cdot p(s_k; s_l) \cdot p(s_l; e_i))$
    `update_path(`s_l,s_m,$Paths$,New_Paths`)`;
    `update_proba(`s_l,$Probas$,New_Probas`)`;
  end
  $Paths \leftarrow New\_Paths$;
  $Probas \leftarrow New\_Probas$;
end
$(Viterbi\_path, p_{max}) \leftarrow$
`most_probable(`$Paths$,$Probas$`)`;
return $Viterbi\_path, p_{max}$

L'algorithme Viterbi itère de e_1 à $e_1 \ldots e_n$. Cet algorithme calcule pour chaque état caché du modèle $s_l \in S$ le chemin le plus probable atteignant cet état à l'étape i et la probabilité de ce chemin partiel. Pendant l'itération, les différents chemins ainsi que leurs probabilités respectives sont stockés respectivement dans la structure de données $Paths$ ou $Probas$. Cette structure de données associe à chaque état caché le chemin le plus probable atteignant cet état ou la probabilité de celui-ci. L'équation 1 permet de déduire la valeur de ce chemin ainsi que sa probabilité au regard des valeurs calculées à l'étape précédente de l'itération. La mise à jour des deux structures données est effectuée respectivement par update_path$(s_l, s_m, Paths, New_Paths)$ et update_proba$(s_l, Probas, New_Probas)$. À la fin de l'itération, l'appel à most_probable$(Paths, Probas)$ permet le calcul de p_{max} la probabilité maximale de $Proba$. De l'état caché associé à cette probabilité, le chemin Viterbi $Viterbi\_path$ peut-être déduit.

4 Un modèle de Markov caché contraint

Un MMC contraint a pour objectif de restreindre l'ensemble des exécutions possibles pour un MMC par des contraintes posées sur la séquence d'états cachés

et/ou d'émissions associée à une exécution d'un processus.

Définition 3 (Modèle de Markov Caché Contraint)**.** Soit $\langle S, A, T, E\rangle$ un modèle de Markov caché, un MMC contraint est défini par un 5-tuplet $\langle S, A, T, E, C\rangle$ où C est un ensemble de contraintes associant une exécution du MMC à $\{vrai, faux\}$. Une exécution de MMC contraint R est *valide* si la valeur de $C(R)$ est $vrai$. Sinon, cette exécution est dite *non-valide*.

La définition d'un MMC contraint est volontairement abstraite. Cela nous permet de ne pas avoir à nous placer dans un langage à contraintes particulier. Toutefois, les processus de Markov considérés dans cet article sont discrets. Différentes contraintes issus d'un langage à contraintes sur les domaines finis [12] peuvent par exemple être posées sur l'exécution d'un processus MMC contraint. Notons cependant que l'exécution d'un MMC ne correspond pas à la résolution d'un problème à contraintes classique. En effet, l'ensemble des contraintes posées lors une exécution du modèle dépend des transitions et des émissions composant cette exécution. Par exemple, si on considère la contrainte que l'état d ne peut-être visité si l'état a déjà été visité. La contrainte que d ne peut plus être visité va être posé au cours de l'exécution si a est visité.

Introduire les MMC contraints dans le cadre de la Programmation par Contraintes autorise à ne pas devoir décrire explicitement l'ensemble des exécutions valides et non-valides du modèle. Une exécution valide ou non-valide est détectée par la résolution du problème à contraintes associé à celle-ci. En contrepartie, ce cadre de travail introduit une nouvelle complexité dans le calcul de Viterbi certains chemins pouvant ne pas satisfaire le problème à contraintes. Afin de modéliser ce calcul comme un problème classique de recherche de solutions, nous proposons de modéliser cette recherche comme une recherche dans un arbre, appelé arbre probabiliste.

Définition 4 (Arbre probabiliste)**.** Soit $\langle S, A, T, E, C\rangle$ un MMC contraint. Un arbre probabiliste est 4-uplet $\langle Noeuds, Arcs, CP, Probabilitées\rangle$. Un élément de $Noeuds$ représente un couple état caché émission (en excluant le noeud initial et les noeuds finaux). Un élément de $Arcs$ représente une transition vers un état caché ainsi que l'émission produite de cet état. Chaque noeud N de $Noeuds$ est étiqueté par un problème de satisfaction de contraintes (CSP) composant CP. Chaque variable de ce CSP représente les possibles exécutions du modèle Markovien passant par ce noeud. Chaque arc N_iN_{i+1} de $Arcs$ est étiqueté par la probabilité de transiter de N_i à N_{i+1}.

Un chemin de l'arbre allant de la racine à une feuille correspond à l'exécution d'un MMC. La probabilité de cette exécution est calculée en multipliant les différentes probabilités associées aux arcs composant le chemin. L'exécution du MMC contraint est valide si le CSP associé à la feuille atteinte par cette exécution est satisfaisable.

Dans la suite, seule une version de l'arbre dédiée au calcul de Viterbi est considérée. En effet, les données émises sont connues pour ce calcul. Donc, étant donné une séquence d'émissions, seule une restriction de l'arbre probabiliste pour cette séquence sera considérée. De plus, comme la séquence d'émissions est de taille finie, cela nous permet de raisonner sur un arbre de recherche de taille finie. Cette restriction d'un arbre probabiliste PT à une séquence d'émissions $e_1 \dots e_n$ est notée $PT_{|e_1 \dots e_n}$.

Comme exemple, la FIGURE 2 représente les deux premières étapes de l'arbre probabiliste du MMC abstrait contraint à satisfaire : pour toutes exécutions du MCC $(start\, s_1 \dots\ s_n end, e_1 \dots e_n)$

$$s_i = a \implies \forall j > i, s_j \neq d.$$

étant la séquence d'émissions 1 1 1 2. Dans cet arbre, les noeuds sont étiquetés par des contraintes posées sur les différents états cachés représentant les exécutions possibles du processus. Des variables S_1, S_2, S_3 et S_4 sont utilisées pour représenter l'état caché choisi à l'étape 1, 2, 3 et 4. Dans ce contexte, une affectation de ces variables représente le choix d'un chemin. Le calcul du chemin de Viterbi pour les MMC contraints doit chercher une affectation de ces variables la plus probable satisfaisant le problème à contraintes associées à cette affectation.

5 Calcul de Viterbi pour les MMC contraints

L'algorithme présenté dans la sous-section 3.2 permet le calcul du chemin de Viterbi pour un MMC classique. Ce calcul est effectué linéairement par rapport à la taille de la séquence d'émissions. Basé sur l'équation 1, le calcul du chemin de Viterbi ne nécessite que de se remémorer des différents chemins les plus probables atteignant chaque état caché pour chaque étape du processus. Cependant, le cadre des MMC contraints introduit une nouvelle complexité dans ce calcul. En effet, il est nécessaire de prendre compte la validité du chemin suivi, i.e. prendre compte la satisfiabilité des contraintes associées à ce chemin. Dans le pire des cas, le calcul de Viterbi pour une séquence d'émissions $e_1 \dots e_n$ dans le contexte des MMC contraints nécessitent de tester la satisfiabilité des CSP associés à chacun des chemins $PT_{|e_1 \dots e_n}$ et de choisir le

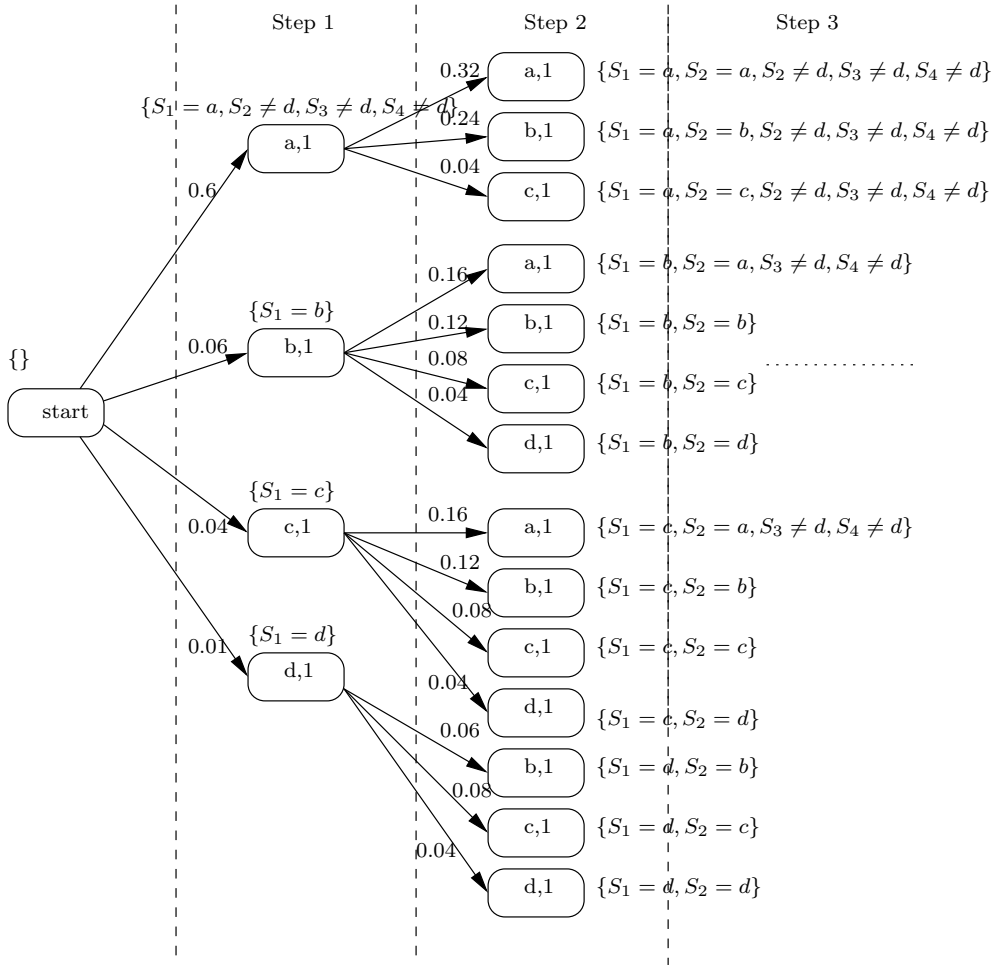

FIGURE 2 – Arbre probabiliste associé à l'observation 1 1 1 2 pour le MMC abstrait

plus probable parmi ceux-ci. Toutefois, l'introduction de ce modèle dans le cadre de la Programmation par Contraintes nous permet de bénéficier de différentes techniques existantes permettant de réduire l'espace de recherche associé au calcul du chemin de Viterbi pour les MMC contraints. Dans cette section, nous proposons donc d'exprimer le calcul de Viterbi comme un problème d'optimisation et utilisons certaines techniques permettant de réduire quand cela est possible la complexité de ce calcul.

5.1 Énoncé du problème

Considérons un MMC contraint $\langle S, A, T, E, C\rangle$, et une séquence d'émissions $e_1 \dots e_n$. Le calcul du chemin de Viterbi a pour objectif de déterminer la séquence d'états cachés $start\, s_1 \dots s_n end$ pour laquelle l'exécution $(start\, s_1 \dots s_n\, end, e_1 \dots e_n)$ est la plus probable et tel que le CSP associé à cette exécution soit satisfaisable.

Dans les termes de la Programmation par Contraintes, le calcul du chemin de Viterbi consiste à chercher une affectation $(s_1, \dots, s_n)$ des variables $S_1, \dots, S_n$ à valeurs dans $S \times \dots \times S$ tel que

$$\begin{array}{rccl} f: & S \times \dots \times S & \rightarrow & [0;1[\\ & (s_1, \dots, s_n) & \mapsto & p(start; s_1) \cdot p(s_1; e_1) \dots \\ & & & p(s_n; e_n) \cdot p(s_n; end) \end{array}$$

soit maximisée et le problème à contraintes associé à ce chemin soit satisfaisable.

5.2 Algorithme

L'algorithme de recherche du chemin de Viterbi pour un MMC contraint est donné par l'**Algorithme**2. Il a pour entrée un modèle de Markov caché contraint $Ctr\_HMM = \langle S, A, T, E, C\rangle$ et une séquence d'émissions $e_1 \dots e_n$, et calcule le chemin de Viterbi $Viterbi\_path$ et la probabilité de ce chemin p_{max}.

Algorithme 2 : Algorithme de recherche du chemin de Viterbi

Entrée : Ctr_HMM,$e_1 \dots e_n$
Sortie : $Viterbi\_path$,p_{max}

$p_{max} = 0$;
$hs \leftarrow [S_1, S_2, \dots, S_n]$;
`search_Viterbi`(hs,pr,$partial\_path$) {
if $hs = \emptyset$ **then**
  $p_{max} \leftarrow pr$;
  $Viterbi\_path \leftarrow partial\_path$;
end
$S_i \leftarrow$`first_element`(hs);
foreach $s \in \mathrm{dom}(S_i)$ **do**
  $s_p \leftarrow$ `previous_state`($partial\_path$);
  $pr \leftarrow pr * p(s_p; s) * p(s; e_i)$;
  if $pr > p_{max}$ **then**
    $partial\_path \leftarrow$`add`($partial\_path, s$);
    if `check_sat`($C(partial\_path, e_1 \dots e_i)$) **then**
      $hs \leftarrow$ `remove`(S_i, hs);
      `search_Viterbi`(hs,pr,$partial\_path$)
    end
  end
end
}

Le calcul est basé sur un appel itératif de $search\_Viterbi$. Cette itération permet le parcours en profondeur d'abord de l'arbre probabiliste associé à Ctr_HMM étant donné la séquence d'émissions $e_1 \dots e_n$. Ce parcourt prend la forme de l'affectation successive des variables S_i représentant le choix d'un état caché à l'étape i. L'affection partielle de ces variables est stockée dans $partial\_path$. La recherche est effectuée pour chaque valeur possible de la variable S_i. La variable pr est utilisée pour stocker la probabilité d'atteindre s à l'étape i. Le parcours de l'arbre peut-être interrompu dans deux situations :

- quand la probabilité de sélectionner ce chemin partiel est inférieure à la courante meilleure solution ;
- quand le problème à contraintes associé au noeud atteint est détecter comme non-satisfaisable.

La variable p_{max} est utilisée pour se remémorer de la courant meilleur solution. Le test de la satisfiabilité du CSP associé au noeud atteint est effectué par une procédure de décision représentée par l'appel de check_sat. Lorsque l'ensemble des variables hs réduit à $\emptyset$, alors une feuille de l'arbre est atteinte. Dans ce cas, une nouvelle solution optimale vient d'être trouvée et les variables $Viterbi\_path$ et p_{max} sont donc mises à jour.

Terminaison et complexité.

Comme l'ensemble des variables est fini, la recherche effectuée dans l'arbre probabiliste termine comme l'arbre est d'hauteur et de largeur finie. Cependant dans le pire des cas, l'algorithme doit tester la satisfiabilité de chacun des CSP associé aux feuilles de l'arbre. Or le nombre de ces feuilles croit exponentiellement par rapport à la longueur de la séquence d'émissions. Cependant, les techniques de cohérences partielles permettent de détecter dès que possible que le chemin partiel ne menant à aucune exécution du modèle valide.

5.3 Réduction de l'arbre par partage de sous-arbre

Par définition, un processus de Markov n'a pas besoin d'avoir une information complète du passé pour effectué une transition. Dans le cadre dans lequel nous nous sommes placés, celui des MMC du premier ordre, la probabilité de transition dépend uniquement de l'état caché atteint à l'étape précédente. En regardant l'arbre probabiliste associé à un MMC contraint, cela peut se traduire par des sous-arbres similaires issus d'un même état caché pour une étape donnée du processus. Considérons de nouveau l'arbre probabiliste $PT_{|1\,1\,1\,2}$, on peut remarquer que les CSP associés respectivement au chemin partiel $start\,b\,a$ et $start\,c\,a$ sont identiques. Cette affirmation nous permet donc de déduire que les sous-arbres associés à ces deux noeuds seront les mêmes. Or la probabilité de sélectionner le chemin partiel $start\,b\,a$ est de $0,0096$ alors que la la probabilité de sélectionner le chemin partiel $start\,c\,a$ est de $0,0064$. Comme le calcul de Viterbi recherche le chemin le plus probable, cela veut dire que le sous-arbre issu du chemin partiel ayant la plus faible probabilité ($startca$) ne va pas mener à une solution optimale.

Plus formellement, considérons l'arbre probabiliste $PT_{|e_1 \dots e_n}$, un chemin partiel de cet arbre $start\,s_1\,s_2 \dots s_j$ (resp. $start\,s'_1\,s'_2 \dots s_j$) atteignant le noeud N_i (resp. N'_i), C_i (resp. C'_i) le problème à contraintes associé à ce noeud et p_i (resp. p'_i) la probabilité d'atteindre ce noeud. Notons par P et P' les CSPs $\langle \{s_1, \dots, s_j, S_{j+1}, \dots, S_n\}, S \times \dots \times S, C_i \rangle$ and $\langle \{s'_1, \dots, s'_j, s_j, S_{j+1}, \dots, S_n\}, S \times \dots \times S, C'_i \rangle$. Alors, si

$$\mathrm{sol}(P') \subseteq \mathrm{sol}(P) \text{ et } p_i \geq p'_i$$

où sol est une fonction associant à un CSP l'ensemble de ces solutions, alors le sous-arbre associé au noeud N'_i peut-être retiré de l'arbre de recherche. Notons que la recherche d'inclusion de solutions de deux sous-CSPs est un problème difficile et coûteux à résoudre dans le cas général. Cependant pour certains sous-ensemble de problèmes à contraintes, il est possible de mettre en place des tests d'inclusion efficaces adaptés au MMC contraint considéré.

6 Implémentation

Dans cette section, nous montrons que le langage de programmation PRISM permet d'implémenter le cadre des MMC contraints. En effet, l'implémentation de PRSIM est basée sur B-Prolog. Il nous permet de combiner les différents résolveurs de contraintes de B-Prolog avec PRISM. Nous illustrons cette implémentation sur l'exemple du MMC abstrait donné FIGURE 1.

6.1 Une brève introduction à PRISM

PRISM [10] est un système développé par Sato et al. permettant dans un cadre déclaratif la définition de différents modèles probabiliste-logiques. Ce système étend Prolog par l'ajout de variables aléatoires discrètes. La déclaration

```
values(lettre, [a,c,g,t])
```

permet la création de la variable aléatoire `lettre` à valeurs dans `[a,c,g,t]`. Par défaut, cette variable aléatoire a une distribution uniforme. Un appel `msw(lettre,X)` pendant l'exécution d'un programme correspond au choix aléatoire d'une valeur pour `X` selon la définition de `lettre`.

La déclaration des variables aléatoires peut être paramétrée, comme

```
values(lettre(_), [a,c,g,t,*])
```

qui est un raccourci pour définir un nombre infini de variables aléatoires dont le nom peut s'unifier au terme `letter(_)`. Cet ensemble de déclarations peut-être utilisé pour définir une transition d'un état à un autre pour un modèle de Markov simple.

```
s(S):- s(-,S).
s(*,[]).
s(X,[L|Ls]):-
      X\= *,
      msw(letter(X),L),
      s(L,Ls).
```

Dans ce modèle, '-' est utilisé pour représenter le début de l'exécution du processus et '*' la fin.

Un programme PIRSM peut-être utilisé de différentes manières. Le système comprend entre autre des prédicats permettant la génération d'échantillon d'exécutions du modèle probabiliste. Ce langage inclut aussi des algorithmes d'apprentissage permettant de faire évoluer les paramètres d'un modèle afin de représenter le plus vraisemblablement un ensemble de données. Pour finir, une version généralisée de l'algorithme de Viterbi au modèle probabiliste-logique est disponible. Cela permet entre autre d'effectuer ce calcul pour différents modèles probabilistes comme les modèles de Markov cachés, les réseaux bayésien discret ...

6.2 Une exemple de MMC contraint avec PRISM

Cette sous-section présente l'implémentation du MMC abstrait donné par la FIGURE 1. Ce modèle est contraint à safistaire la condition suivante : pour chaque exécution du MMC $(start\, s_1 \dots s_n\, end, e_1 \dots e_n)$

$$s_i = a \implies \forall j > i, s_j \neq d.$$

Le code suivant représente le programme PRISM définissant ce MMC contraint.

```
% Transition état-état
values(start,[a,b,c,d]).
values(trans(_),[a,b,c,d,end]).

% Transition état-émission
values(emit(_),[1,2]).

% Définition des distributions de probabilités
set_params:-
   set_sw(begin,[0.75,0.1,0.1,0.05]),
   set_sw(trans(a),[0.4,0.4,0.1,0,0.1]),
   set_sw(trans(d),[0,0.1,0.2,0.2,0.5]),
   set_sw(emit(a),[0.8,0.2]),
   set_sw(emit(b),[0.6,0.4]),
   set_sw(emit(c),[0.4,0.6]),
   set_sw(emit(d),[0.2,0.8]).

% Initialisation du Processus
abstract_ctr(Emissions) :-
   set_params,
   msw(begin,State), % Choix du 1er état
   SeenA in 0..1,
   abstract_ctr(State,SeenA,Emissions).

% Fin de l'exécution
abstract_ctr(end,_SeenA,[]).

% Etat a non visité
abstract_ctr(State,SeenA,[Letter|Rest]) :-
   State \== end,
   var(SeenA), % a non visité
   msw(emit(State),Letter), % Emission
   msw(trans(State),New_State),% Transition
   New_State #= a # => SeenA#=1
   abstract_ctr(State_New,SeenA,Rest).

% Etat a visité
abstract_ctr(State,SeenA,[Letter|Rest]) :-
   State \== end,
   msw(emit(State),Letter) % Emission
   msw(trans(State),New_State),% Transition
   SeenA #= 1 #=> New_State #\=d),
   abstract_ctr(State_New,SeenA,Rest).
```

Une exécution du MMC contraint a lieu par l'appel au prédicat `abstract_ctr/1`.

```
?- abstract_ctr(S).
```

Une exécution valide a été suivie lorsque la requête réussit

```
?- abstract_ctr(S).
S = [1,1,1,2]
yes
```

Cette requête échoue lorsque le chemin choisi ne satisfait pas la contrainte :

```
?- abstract(S).
false
```

Les transitions entre états cachés du modèle sont modélisées par l'appel à

```
msw(trans(State),New_State)
```

La génération d'une émission étant donné un état caché est effectuée à l'aide de

```
msw(emit(State),Letter)
```

La variable `SeenA` est utilisée pour savoir si l'état caché `a` a déjà été visité. Le système PRISM permet le calcul du chemin de Viterbi pour ce type modèle par l'appel de

```
?- viterbi(abstract_ctr([1,1,1,2]),P,Path).
P = 0.00098304
Path = [start,a,a,a,b,end]
yes
```

Dans cette implémentation, nous utilisons l'algorithme de Viterbi générique de PRISM pour différents modèles probabilistes. Des techniques de cohérences d'intervalles utilisés par le résolveur de contraintes sur les domaines finis de B-Prolog permettent réduire dès que possible l'arbre de recherche. Toutefois, cette implémentation n'intègre pas la technique le partage de sous-arbres proposée sous-section 5.3.

7 Travaux futurs

Dans cet article, nous avons proposé les MMC contraints dans le cadre de la Programmation Contraintes. Cette approche nous permet de modéliser le calcul du chemin de Viterbi comme un problème d'optimisation. L'introduction de ce modèle s'est voulue générique, au sens où ce modèle n'est pas restreint à un type particulier de résolveur de contraintes. Une première implémentation basée sur le système PRISM et utilisant le résolveur de contraintes sur les domaines finis de B-Prolog est proposée. Cette implémentation offre un cadre déclaratif pour la définition de MMC contraints. Toutefois, de nombreuses améliorations peuvent être apportées à l'algorithme proposé. En restant dans le cadre de la programmation à contraintes sur les domaines finis, ce calcul de Viterbi pour les MMC contraints peut bénéficier de nombreux résultats obtenus dans le cadre de la recherche And/Or pour des modèles graphiques [1] et utiliser les différents algorithmes de cohérence locale pour les CSP pondérés [6]. La réduction de l'arbre de recherche par partage de sous-arbre est un problème intéressant que nous désirons développer.

Références

[1] R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 171(2-3) :73–106, February 2007.

[2] A. Di Pierro and H. Wiklicky. On probabilistic CCP. In *Proceedings of the Joint Conference on Declarative Programming*, pages 225–234, Grado, Italy, 1997.

[3] R. Durbin, S.R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[4] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems : a probabilistic approach. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, LNCS, pages 97–104, Granada, Spain, November 1993.

[5] V. Gupta, R. Jagadeesan, and V.A. Saraswat. Probabilistic concurrent constraint programming. In *Proceedings of the International Conference Conference on Concurrency Theory*, LNCS, pages 243–257, Warsaw, Poland, 1997. Springer.

[6] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1–2) :1–26, 2004.

[7] M. Petit. *Test statistique structurel par résolution de contraintes de choix probabiliste (in french)*. Phd thesis, Université de Rennes 1, France, Juillet 2008.

[8] M. Petit and A. Gotlieb. Boosting probabilistic choice operators. In *Proceedings of the International Conference on Principles and Pratice of Constraint Programming*, LNCS, pages 559–573, Providence, USA, September 2007.

[9] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2) :257–286, 1989.

[10] T. Sato and Y. Kameya. New advances in logic-based probabilistic modeling by prism. In *Probabilistic Inductive Logic Programming*, pages 118–155, 2008.

[11] S. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming : a scenario-based approach. *Constraints*, 11(1) :53–80, 2006.

[12] P. Van Hentenryck, V.A. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(fd). *Constraint Programming*, 910 :293–316, May 1995.

[13] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments : A survey. *Constraints*, 10(3) :253–281, July 2005.

[14] A.J. Viterbi. Error bound for convutional codes and an asymptotically optimum algorithm. *Transactions on Information Theory*, 13(2) :260–269, 1967.

[15] T. Walsh. Stochastic constraint programming. In *Proceedings of the European Conference on Artificial Intelligence*, pages 111–115, Lyon, France, July 2002. IOS Press.