

Pense-bête VIP : Modèles basés sur les variables

Afshine AMIDI et Shervine AMIDI

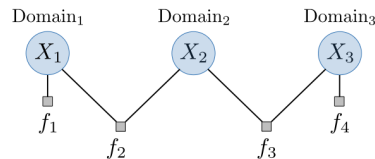
8 septembre 2019

Problèmes de satisfaction de contraintes

Dans cette section, notre but est de trouver des affectations de poids maximisants dans des problèmes impliquant des modèles basés sur les variables. Un avantage comparé aux modèles basés sur les états est que ces algorithmes sont plus commodes lorsqu'il s'agit de transcrire des contraintes spécifiques à certains problèmes.

Graphes de facteurs

□ **Définition** – Un graphe de facteurs, aussi appelé champ aléatoire de Markov, est un ensemble de variables $X = (X_1, \dots, X_n)$ où $X_i \in \text{Domain}_i$ muni de m facteurs f_1, \dots, f_m où chaque $f_j(X) \geq 0$.



□ **Arité** – Le nombre de variables dépendant d'un facteur f_j est appelé son arité.

Remarque : les facteurs d'arité 1 et 2 sont respectivement appelés unaire et binaire.

□ **Affectation de poids** – Chaque affectation $x = (x_1, \dots, x_n)$ donne un poids $\text{Weight}(x)$ défini comme étant le produit de tous les facteurs f_j appliqués à cette affectation. Son expression est donnée par :

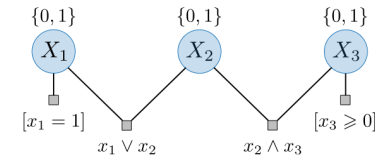
$$\text{Weight}(x) = \prod_{j=1}^m f_j(x)$$

□ **Problème de satisfaction de contraintes** – Un problème de satisfaction de contraintes (en anglais *constraint satisfaction problem* ou *CSP*) est un graphe de facteurs où tous les facteurs sont binaires ; on les appelle "contraintes".

$$\forall j \in \llbracket 1, m \rrbracket, \quad f_j(x) \in \{0, 1\}$$

Ici, on dit que l'affectation x satisfait la contrainte j si et seulement si $f_j(x) = 1$.

□ **Affectation consistante** – Une affectation x d'un CSP est dite consistante si et seulement si $\text{Weight}(x) = 1$, i.e. toutes les contraintes sont satisfaites.



Mise en ordre dynamique

□ **Facteurs dépendants** – L'ensemble des facteurs dépendants de la variable X_i dont l'affectation partielle est x est appelé $D(x, X_i)$ et désigne l'ensemble des facteurs liant X_i à des variables déjà affectées.

□ **Recherche avec retour sur trace** – L'algorithme de recherche avec retour sur trace (en anglais *backtracking search*) est utilisé pour trouver l'affectation de poids maximum d'un graphe de facteurs. À chaque étape, une variable non assignée est choisie et ses valeurs sont explorées par récursivité. On peut utiliser un processus de mise en ordre dynamique sur le choix des variables et valeurs et/ou d'anticipation (i.e. élimination précoce d'options non consistantes) pour explorer le graphe de manière plus efficace. La complexité temporelle dans tous les cas reste néanmoins exponentielle : $O(|\text{Domain}|^n)$.

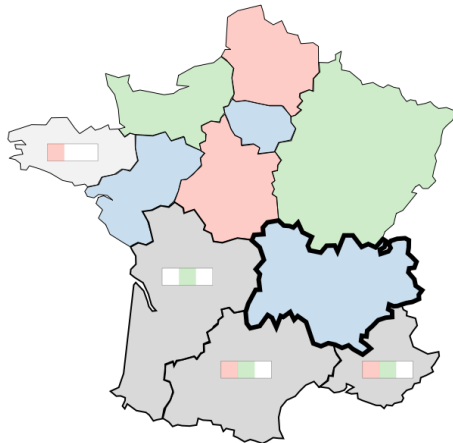
□ **Vérification en avant** – La vérification en avant (*forward checking* en anglais) est une heuristique d'anticipation à une étape qui enlève des variables voisines les valeurs impossibles de manière préemptive. Cette méthode a les caractéristiques suivantes :

- Après l'affectation d'une variable X_i , les valeurs non consistantes sont éliminées du domaine de tous ses voisins.
- Si l'un de ces domaines devient vide, la recherche locale s'arrête.
- Si l'on enlève l'affectation d'une valeur X_i , on doit restaurer le domaine de ses voisins.

□ **Variable la plus contrainte** – L'heuristique de la variable la plus contrainte (en anglais *most constrained variable* ou *MCV*) sélectionne la prochaine variable sans affectation ayant le moins de valeurs consistantes. Cette procédure a pour effet de faire échouer les affectations impossibles plus tôt dans la recherche, permettant un élagage plus efficace.

□ **Valeur la moins contraignante** – L'heuristique de la valeur la moins contraignante (en anglais *least constrained value* ou *LCV*) sélectionne pour une variable donnée la prochaine valeur maximisant le nombre de valeurs consistantes chez les variables voisines. De manière intuitive, on peut dire que cette procédure choisit en premier les valeurs qui sont le plus susceptible de marcher.

Remarque : en pratique, cette heuristique est utile quand tous les facteurs sont des contraintes.



L'exemple ci-dessus est une illustration du problème de coloration de graphe à 3 couleurs en utilisant l'algorithme de recherche avec retour sur trace couplé avec les heuristiques de MCV, de LCV ainsi que de vérification en avant à chaque étape.

□ **Arc-consistance** – On dit que l’arc-consistance de la variable X_l par rapport à X_k est vérifiée lorsque pour tout $x_l \in \text{Domain}_l$:

- les facteurs unaires de X_l sont non-nuls,
- il existe au moins un $x_k \in \text{Domain}_k$ tel que n'importe quel facteur entre X_l et X_k est non nul.

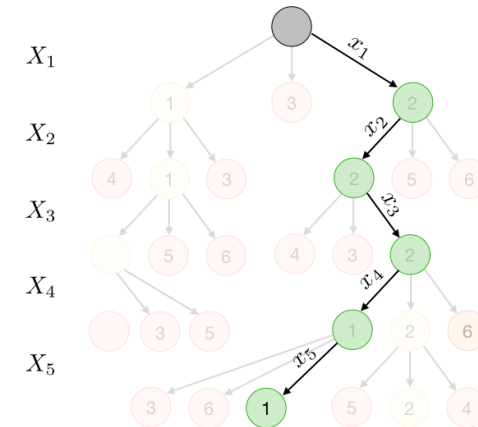
□ **AC-3** – L’algorithme d’AC-3 est une heuristique qui applique le principe de vérification en avant à toutes les variables susceptibles d’être concernées. Après l’affectation d’une variable, cet algorithme effectue une vérification en avant et applique successivement l’arc-consistance avec tous les voisins de variables pour lesquels le domaine change.

Remarque : AC-3 peut être codé de manière itérative ou récursive.

Méthodes approximatives

□ **Recherche en faisceau** – L'algorithme de recherche en faisceau (en anglais *beam search*) est une technique approximative qui étend les affectations partielles de n variables de facteur de branchement $b = |\text{Domain}|$ en explorant les K meilleurs chemins qui s'offrent à chaque étape. La largeur du faisceau $K \in \{1, \dots, b^n\}$ détermine la balance entre efficacité et précision de l'algorithme. Sa complexité en temps est de $O(n \cdot Kb \log(Kb))$.

L'exemple ci-dessous illustre une recherche en faisceau de paramètres $K = 2$, $b = 3$ et $n = 5$.



Remarque : $K = 1$ correspond à la recherche gloutonne alors que $K \rightarrow +\infty$ est équivalent à effectuer un parcours en largeur.

□ **Modes conditionnels itérés** – L'algorithme des modes conditionnels itérés (en anglais *iterated conditional modes* ou *ICM*) est une technique itérative et approximative qui modifie l'affectation d'un graphe de facteurs une variable à la fois jusqu'à convergence. À l'étape i , X_i prend la valeur v qui maximise le produit de tous les facteurs connectés à cette variable.

Remarque : il est possible qu'ICM reste bloqué dans un minimum local.

□ **Échantillonnage de Gibbs** – La méthode d'échantillonnage de Gibbs (en anglais *Gibbs sampling*) est une technique itérative et approximative qui modifie les affectations d'un graphe de facteurs une variable à la fois jusqu'à convergence. À l'étape i :

- on assigne à chaque élément $u \in \text{Domain}_i$ un poids $w(u)$ qui est le produit de tous les facteurs connectés à cette variable,
- on échantillonne v de la loi de probabilité engendrée par w et on l'associe à X_i .

Remarque : la méthode d'échantillonnage de Gibbs peut être vue comme étant la version probabiliste de ICM. Cette méthode a l'avantage de pouvoir échapper aux potentiels minimum locaux dans la plupart des situations.

Transformations sur les graphes de facteurs

□ **Indépendance** – Soit A, B une partition des variables X . On dit que A et B sont indépendants s'il n'y a pas d'arête connectant A et B et on écrit :

$A \text{ et } B \text{ indépendants} \iff A \perp B$

Remarque : l'indépendance est une propriété importante car elle nous permet de décomposer la situation en sous-problèmes que l'on peut résoudre en parallèle.

□ **Indépendance conditionnelle** – On dit que A et B ont conditionnellement indépendants par rapport à C si le fait de conditionner sur C produit un graphe dans lequel A et B sont indépendants. Dans ce cas, on écrit :

$A \text{ et } B \text{ cond. indép. par rapport à } C \iff A \perp\!\!\!\perp B|C$

□ **Conditionnement** – Le conditionnement est une transformation visant à rendre des variables indépendantes et ainsi diviser un graphe de facteurs en pièces plus petites qui peuvent être

traitées en parallèle et utiliser le retour sur trace. Pour conditionner par rapport à une variable $X_i = v$, on :

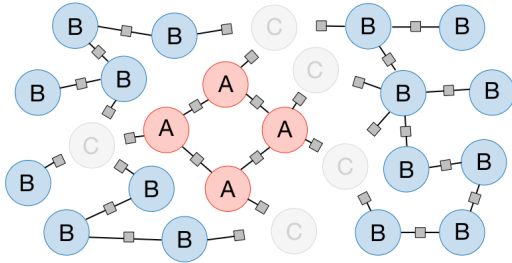
- considère toutes les facteurs f_1, \dots, f_k qui dépendent de X_i
- enlève X_i et f_1, \dots, f_k
- ajoute $g_j(x)$ pour $j \in \{1, \dots, k\}$ défini par :

$$g_j(x) = f_j(x \cup \{X_i : v\})$$

□ **Couverture de Markov** – Soit $A \subseteq X$ une partie des variables. On définit $\text{MarkovBlanket}(A)$ comme étant les voisins de A qui ne sont pas dans A .

□ **Proposition** – Soit $C = \text{MarkovBlanket}(A)$ et $B = X \setminus (A \cup C)$. On a alors :

$$A \perp\!\!\!\perp B | C$$



□ **Élimination** – L'élimination est une transformation consistant à enlever X_i d'un graphe de facteurs pour ensuite résoudre un sous-problème conditionné sur sa couverture de Markov où l'on :

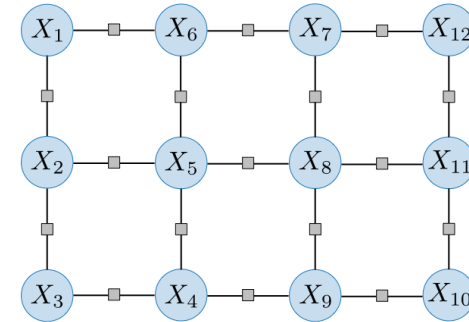
- considère tous les facteurs $f_{i,1}, \dots, f_{i,k}$ qui dépendent de X_i
- enlève X_i et $f_{i,1}, \dots, f_{i,k}$
- ajoute $f_{\text{new},i}(x)$ défini par :

$$f_{\text{new},i}(x) = \max_{x_i} \prod_{l=1}^k f_{i,l}(x)$$

□ **Largeur arborescente** – La largeur arborescente (en anglais *treewidth*) d'un graphe de facteurs est l'arité maximum de n'importe quel facteur créé par élimination avec le meilleur ordre de variable. En d'autres termes,

$$\text{Treewidth} = \min_{\text{orderings}} \max_{i \in \{1, \dots, n\}} \text{arity}(f_{\text{new},i})$$

L'exemple ci-dessous illustre le cas d'un graphe de facteurs ayant une largeur arborescente égale à 3.



Remarque : trouver le meilleur ordre de variable est un problème NP-difficile.

Réseaux bayésiens

Dans cette section, notre but est de calculer des probabilités conditionnelles. Quelle est la probabilité d'un événement étant donné des observations ?

Introduction

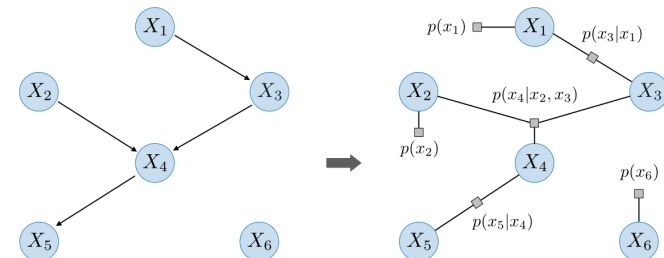
□ **Explication** – Supposons que les causes C_1 et C_2 influencent un effet E . Le conditionnement sur l'effet E et une des causes (disons C_1) change la probabilité de l'autre cause (disons C_2). Dans ce cas, on dit que C_1 a expliqué C_2 .

□ **Graphe orienté acyclique** – Un graphe orienté acyclique (en anglais *directed acyclic graph* ou *DAG*) est un graphe orienté fini sans cycle orienté.

□ **Réseau bayésien** – Un réseau bayésien (en anglais *Bayesian network*) est un DAG qui définit une loi de probabilité jointe sur les variables aléatoires $X = (X_1, \dots, X_n)$ comme étant le produit des lois de probabilités conditionnelles locales (une pour chaque nœud) :

$$P(X_1 = x_1, \dots, X_n = x_n) \triangleq \prod_{i=1}^n p(x_i | x_{\text{Parents}(i)})$$

Remarque : les réseaux bayésiens sont des graphes de facteurs imprégnés de concepts de probabilité.



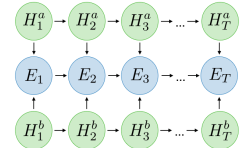
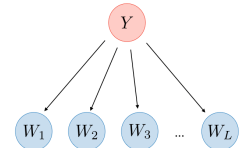
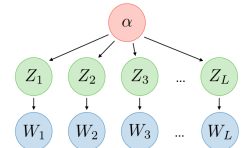
□ **Normalisation locale** – Pour chaque $x_{\text{Parents}(i)}$, tous les facteurs sont localement des lois de probabilité conditionnelles. Elles doivent donc vérifier :

$$\sum_{x_i} p(x_i | x_{\text{Parents}(i)}) = 1$$

De ce fait, les sous-réseaux bayésiens et les distributions conditionnelles sont consistants.

Remarque : les lois locales de probabilité conditionnelles sont de vraies lois de probabilité conditionnelles.

□ **Marginalisation** – La marginalisation d'un nœud sans enfant entraîne un réseau bayésien sans ce nœud.

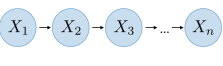
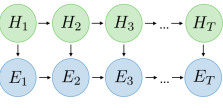
HMM factoriel	$H_t^o \sim_{o \in \{a,b\}} p(H_t^o H_{t-1}^o)$ $E_t \sim p(E_t H_t^a, H_t^b)$		Suivi de plusieurs objets
Bayésien naïf	$Y \sim p(Y)$ $W_i \sim p(W_i Y)$		Classification de document
Allocation de Dirichlet latente (LDA)	$\alpha \in \mathbb{R}^K$ distribution $Z_i \sim p(Z_i \alpha)$ $W_i \sim p(W_i Z_i)$		Modélisation de sujet

Programmes probabilistes

□ **Concept** – Un programme probabiliste rend aléatoire l'affectation de variables. De ce fait, on peut imaginer des réseaux bayésiens compliqués pour la génération d'affectations sans avoir à écrire de manière explicite les probabilités associées.

Remarque : quelques exemples de programmes probabilistes incluent parmi d'autres le modèle de Markov caché (en anglais hidden Markov model ou HMM), HMM factoriel, le modèle bayésien naïf (en anglais naive Bayes), l'allocation de Dirichlet latente (en anglais latent Dirichlet allocation ou LDA), le modèle à blocs stochastiques (en anglais stochastic block model).

□ **Récapitulatif** – La table ci-dessous résume les programmes probabilistes les plus fréquents ainsi que leur champ d'application associé :

Programme	Algorithme	Illustration	Exemple
Modèle de Markov	$X_i \sim p(X_i X_{i-1})$		Modélisation du langage
Modèle de Markov caché (HMM)	$H_t \sim p(H_t H_{t-1})$ $E_t \sim p(E_t H_t)$		Suivi d'objet

Inférence

□ **Stratégie générale pour l'inférence probabiliste** – La stratégie que l'on utilise pour calculer la probabilité $P(Q | E = e)$ d'une requête Q étant donnée l'observation $E = e$ est la suivante :

- Étape 1 : on enlève les variables qui ne sont pas les ancêtres de la requête Q ou de l'observation E par marginalisation
- Étape 2 : on convertit le réseau bayésien en un graphe de facteurs
- Étape 3 : on conditionne sur l'observation $E = e$
- Étape 4 : on enlève les nœuds déconnectés de la requête Q par marginalisation
- Étape 5 : on lance un algorithme d'inférence probabiliste (manuel, élimination de variables, échantillonnage de Gibbs, filtrage particulaire)

□ **Algorithme progressif-rétrogressif** – L'algorithme progressif-rétrogressif (en anglais *forward-backward*) calcule la valeur exacte de $P(H = h_k | E = e)$ pour chaque $k \in \{1, \dots, L\}$ dans le cas d'un HMM de taille L . Pour ce faire, on procède en 3 étapes :

- Étape 1 : pour $i \in \{1, \dots, L\}$, calculer $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1}) p(h_i | h_{i-1}) p(e_i | h_i)$
- Étape 2 : pour $i \in \{L, \dots, 1\}$, calculer $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1}) p(h_{i+1} | h_i) p(e_{i+1} | h_{i+1})$
- Étape 3 : pour $i \in \{1, \dots, L\}$, calculer $S_i(h_i) = \frac{F_i(h_i) B_i(h_i)}{\sum_{h_i} F_i(h_i) B_i(h_i)}$

avec la convention $F_0 = B_{L+1} = 1$. À partir de cette procédure et avec ces notations, on obtient

$$P(H = h_k | E = e) = S_k(h_k)$$

Remarque : cet algorithme interprète une affectation comme étant un chemin où chaque arête $h_{i-1} \rightarrow h_i$ a un poids $p(h_i | h_{i-1}) p(e_i | h_i)$.

□ **Échantillonnage de Gibbs** – L’algorithme d’échantillonnage de Gibbs (en anglais *Gibbs sampling*) est une méthode itérative et approximative qui utilise un petit ensemble d’affectations (particules) pour représenter une loi de probabilité. Pour une affectation aléatoire x , l’échantillonnage de Gibbs effectue les étapes suivantes pour $i \in \{1, \dots, n\}$ jusqu’à convergence :

- Pour tout $u \in \text{Domain}_i$, on calcule le poids $w(u)$ de l’affectation x où $X_i = u$
- On échantillonne v de la loi de probabilité engendrée par $w : v \sim P(X_i = v | X_{-i} = x_{-i})$
- On pose $X_i = v$

Remarque : X_{-i} veut dire $X \setminus \{X_i\}$ et x_{-i} représente l’affectation correspondante.

□ **Filtrage particulaire** – L’algorithme de filtrage particulaire (en anglais *particle filtering*) approxime la densité postérieure de variables d’états à partir des variables observées en suivant K particules à la fois. En commençant avec un ensemble de particules C de taille K , on répète les 3 étapes suivantes :

- Étape 1 : proposition - Pour chaque particule $x_{t-1} \in C$, on échantillonne x avec loi de probabilité $p(x|x_{t-1})$ et on ajoute x à un ensemble C' .
- Étape 2 : pondération - On associe chaque x de l’ensemble C' au poids $w(x) = p(e_t|x)$, où e_t est l’observation vue à l’instant t .
- Étape 3 : échantillonnage - On échantillonne K éléments de l’ensemble C' en utilisant la loi de probabilité engendrée par w et on les met dans C : ce sont les particules courantes x_t .

Remarque : une version plus coûteuse de cet algorithme tient aussi compte des particules passées à l’étape de proposition.

□ **Maximum de vraisemblance** – Si l’on ne connaît pas les lois de probabilité locales, on peut les trouver en utilisant le maximum de vraisemblance.

$$\max_{\theta} \prod_{x \in \mathcal{D}_{\text{train}}} p(X = x; \theta)$$

□ **Lissage de Laplace** – Pour chaque loi de probabilité d et affectation partielle $(x_{\text{Parents}(i)}, x_i)$, on ajoute λ à $\text{count}_d(x_{\text{Parents}(i)}, x_i)$ et on normalise ensuite pour obtenir des probabilités.

□ **Espérance-maximisation** – L’algorithme d’espérance-maximisation (en anglais *expectation-maximization* ou *EM*) est une méthode efficace utilisée pour estimer le paramètre θ via l’estimation du maximum de vraisemblance en construisant de manière répétée une borne inférieure de la vraisemblance (étape E) et en optimisant cette borne inférieure (étape M) :

- Étape E : on évalue la probabilité postérieure $q(h)$ que chaque point e vienne d’une partition particulière h avec :

$$q(h) = P(H = h | E = e; \theta)$$

- Étape M : on utilise la probabilité postérieure $q(h)$ en tant que poids de la partition h sur les points e pour déterminer θ through via le maximum de vraisemblance.