PUBLICATIONS MATHÉMATIQUES ET INFORMATIQUES DE RENNES

JACQUES NICOLAS

Quelques travaux sur l'apprentissage automatique

Publications de l'Institut de recherche mathématiques de Rennes, 1992, fascicule 3
« Fascicule de didactique des mathématiques », , exp. n° 2, p. 1-13

<http://www.numdam.org/item?id=PSMIR_1992___3_A2_0>

QUELQUES TRAVAUX SUR L'APPRENTISSAGE AUTOMATIQUE

Jacques NICOLAS
IRISA - Université de Rennes 1

1 Introduction

L'apprentissage automatique, comme son nom ne l'indique peut-être pas assez clairement, vise à augmenter l'autonomie de machines ou de programmes en leur apportant des capacités d'évolution, ceci en leur permettant d'acquérir automatiquement un modèle de leur environnement. L'apprentissage automatique a démarré autour des perceptrons, ancêtres des réseaux neuronaux dans lesquels l'apprentissage se réduit au réglage automatique des paramètres d'une fonction d'identification. Durant les années 70, les études se sont affirmées vers l'apprentissage de modèles plus complexes : arbres de décision, règles et formules logiques. Les problématiques se sont affinées a des degrés de maturité divers. Nous exposons brièvement ici les principales, seules les deux premières formeront les articulations de l'exposé.

- L'apprentissage empirique a bénéficié des développements les plus nombreux. Il s'agit, étant donné des exemples et éventuellement des contre-exemples d'une classe d'objets (ou encore d'un concept, d'une situation), de trouver une définition générale de la classe dans un langage fixé (arbres de décision, règles,...).

- Une autre forme d'apprentissage à partir d'exemples s'est également développée, qualifiée d'analytique, particulièrement utile dans un cadre de résolution de problèmes. Le but est cette fois de spécialiser un concept trop général en cherchant sa restriction à un domaine caractérisé par les exemples. La notion d'explication y joue un rôle prépondérant.

- Enfin, lorsque l'on ne cherche pas la caractérisation d'une classe définie, mais plutôt une théorie cohérente expliquant un ensemble de faits observés, on s'attaque à un problème particulièrement difficile, dont la résolution passe par une communication interactive entre l'algorithme d'apprentissage et son utilisateur. Ce point a été surtout développé dans les études sur l'induction formelle.

- Parallèlement à ces travaux,une communauté de chercheurs s'est attaquée aux fondements de l'apprentissage, en étudiant les points communs et les limites théoriques des algorithmes. Ceci a conduit à la spécification de différents critères d'apprentissage et d'apprenabilité.

Nous commençons par un survol rapide des études sur l'apprentissage automatique, en introduisant progressivement les concepts essentiels de la discipline. Nous abordons ensuite le problème de la généralisation lorsque celle-ci est basée sur une hypothèse de recherche de similarités, problème central de l'apprentissage empirique. Un algorithme fondamental de recherche combinatoire est détaillé. Enfin, nous illustrons les principales recherches en apprentissage en développant un algorithme simple dans chaque cas.

2 Quelques définitions

2.1 Trois composantes d'un système d'apprentissage

L'apprentissage automatique a été défini par Michalski comme la capacité pour un système de construire ou de modifier les représentations de l'environnement utilisées par une tâche expérimentale. Il s'agit d'étudier, de modéliser un apprenant, sans s'imposer de contraintes a priori sur la manière d'y parvenir. Il faut donc le distinguer d'emblée des études en enseignement assisté par ordinateur, où l'apprenant n'a rien d'artificiel. La courte définition de Michalski met en valeur trois composantes essentielles de tout système d'apprentissage.

- En parallèle de la tâche d'apprentissage se déroule toujours une tâche qui va effectivement utiliser la connaissance apprise. Il s'agira par exemple d'un système de reconnaissance de caractères, ou d'un bras de robot devant saisir des objets sur une chaîne d'assemblage, ou encore d'un programme d'intégration formelle. La nature de la connaissance utile va imposer des contraintes sur les hypothèses de base de l'apprentissage. Essentiellement, on distinguera les contextes où l'on souhaite *synthétiser* une connaissance, ceux où l'on souhaite *raffiner* une connaissance, et ceux où l'on ne cherche pas à construire une connaissance explicite, mais où un résultat de type *réflexe* est suffisant.
  L'existence de ces deux tâches entraîne le besoin de deux modules se chargeant de l'interface entre elles. Dans le sens utilisation → apprentissage, il faut un module *critique*, qui analyse les résultats de l'expérimentation en cours, afin de décider si la base de connaissances courante doit être modifiée et de générer éventuellement un nouveau problème d'apprentissage. Ceci est particulièrement utile dans un environnement de résolution de problèmes, pour lequel il faut extraire d'un ensemble d'essais, d'erreurs et de retour-arrière sur des choix, une évaluation individuelle des pas de résolution. Inversement, dans le sens apprentissage → utilisation, il faut un module *sélecteur*, capable d'induire de nouvelles expérimentations, d'agir sur les paramètres susceptibles de faire évoluer les connaissances.

- Le problème de la représentation des connaissances, central en intelligence artificielle, est a fortiori prépondérant dans ce domaine. Tous les programmes doivent réaliser un compromis entre la puissance d'expression théoriquement nécessaire pour représenter toutes les situations possibles et la complexité des opérations élementaires de traitement des expressions du langage choisi. Ainsi, passer d'une représentation propositionnelle à une représentation prédicative permet d'exprimer non seulement des propriétés sur des objets, mais également des relations entre ces objets. D'un autre côté, cela implique une complexité fortement accrue des opérations associées (unification, preuve,...).
  En fait, ce compromis n'est pas simplement nécessaire pour rendre le problème traitable. Plus fondamentalement, le choix d'une classe de langages correspond à un choix sur ce qu'il est souhaitable d'oublier. Un apprentissage "par coeur" est très rarement le but recherché. Choisir de ne pas tout exprimer, c'est aussi choisir ce qui peut présenter un intérêt vis à vis de l'application (ceci correspond à l'hypothèse psychologique de Whorf, qui suggère que les langages forment et restreignent les idées qui sont exprimables dans ce langage).

- Enfin, le coeur d'un système d'apprentissage est constitué d'un algorithme, permettant de sélectionner, de transformer ou de créer des connaissances à partir de connaissances

initiales et de l'analyse du fonctionnement du module utilisateur de ces connaissances. On parle souvent de mécanisme de *généralisation*, car il s'agit de généraliser la portée d'une connaisance par la prise en compte d'un nouveau cas (du point de vue des opérations appliquées, la *généralisation* correspond parfois à une spécialisation...). C'est cette partie que nous développerons tout particulièrement. Les algorithmes reposent presque tous sur la recherche d'un critère à maximiser ou à minimiser. La recherche d'un tel critère représente un travail important du point de vue des études en apprentissage. Nous introduisons dans la prochaine section un cadre général permettant de délimiter certaines des propriétés fondamentales recherchées.

2.2 Critères pour la généralisation

Mitchell a défini le problème de la généralisation comme une recherche dans un espace d'hypothèses possibles. C'est un présupposé important de la généralisation : on se donne a priori l'ensemble de toutes les solutions possibles.

Etant donnés

- un langage LI pour décrire les instances;
- un langage LG pour décrire les généralisations (descriptions de concepts);
- un prédicat d'appariement faisant correspondre instances et généralisations;
- un ensemble d'instances positives et éventuellement négatives (exemples et contre-exemples) d'un concept à apprendre;
- un critère de consistance

Déterminer
les généralisations faisant partie du langage considéré, consistantes avec les instances présentées.

Nous voulons développer deux points plus particulièrement : le prédicat d'appariement et le critère de consistance.

2.2.1 le prédicat d'appariement

Le rôle de ce prédicat M est de mettre en correspondance instances et généralisations, i.e. de spécifier quelles descriptions sont les généralisations de quelles instances. Il induit un ordre partiel sur le langage de généralisations, lié à la notion de généralité. Cette relation d'ordre (notée $\geq$, plus spécifique), est définie sur l'ensemble des couples de généralisations (x,y), si l'on note I l'espace des instances, par

$$x \geq y \Longleftrightarrow \{i \in I : M(x,i)\} \subseteq \{i \in I : M(y,i)\}$$

L'existence de cet ordre permet d'organiser la recherche dans l'espace des généralisations et d'obtenir des algorithmes plus efficaces que la recherche exhaustive. Cependant, la définition précédente, en extension, est peu intéressante en pratique. On recourt donc à une représentation unique des instances et des généralisations. Le langage des instances devient une restriction du

langage des généralisations et la relation de généralité un prolongement de la relation d'appariement

$$x \geq y \Longleftrightarrow M(g,s)$$

Le prédicat d'appariement sert également dans la définition du critère de consistance.
Pour simplifier et formaliser la discussion, nous considérons maintenant une forme logique pour le langage unique de représentation. Les descriptions des instances et des généralisations sont donc des formules logiques. On dispose également de connaissances a priori qui sont formalisées par un ensemble de formules T. On cherche à relier le prédicat M à la relation de dérivation logique $\vdash$. Deux cas se présentent, qui sont tous deux effectivement utilisés en apprentissage.

$$M(G,S) \iff T, G \vdash S \quad (1)$$

$$M(G,S) \iff T, S \vdash G \quad (2)$$

En ce qui concerne le raisonnement de type 1, on parlera d'abduction ou d'induction. Le point commun est que l'inférence est non valide, mais permet par contre de produire de nouveaux faits.
L'induction est le procédé qui consiste à passer d'un ensemble de faits à une loi impliquant ces faits. Une forme bien connue d'induction agit en généralisant les termes : de mortel(Socrate), mortel(Platon), mortel(Aristote), et d'un certain nombre de faits similaires (à condition que nulle part n'aparaisse des faits comme $\neg mortel(Dieu)$), on peut induire $\forall x\ mortel(x)$.
L'abduction est le procédé inférentiel consistant à passer des effets d'un phénomène à ses causes, autrement dit à remonter des conclusions aux prémisses. L'abduction cherche toujours à exploiter une connaissance préexistante. Elle est très commune, dès lors qu'il s'agit de diagnostic ou d'identification. Par exemple, de la définition préexistante $\forall x \forall y\ poison(x) \wedge homme(y) \wedge boit(x,y) \Rightarrow meurt(y)$ et du fait $homme(Socrate) \wedge boit(Socrate, ciguë) \wedge meurt(Socrate)$, on peut inférer par abduction $poison(ciguë)$.

Le raisonnement de type 2 correspond à *la déduction* et est particulièrement utilisé en apprentissage de concepts à partir d'exemples. L'idée sous-jacente est qu'une généralisation doit refléter les points communs de chacune des instances, et doit donc être impliquée par chacune d'entre elles. Ce type de généralisation est valide mais permet simplement de classer de nouvelles instances comme positives ou négatives, pas de générer de nouveaux faits. En reprenant notre exemple, des instances positives $philosophe(Socrate) \wedge petit(Socrate) \wedge grec(Socrate)$ et $philosophe(Platon) \wedge grand(Platon) \wedge grec(Platon)$, on déduira cette fois $\exists x\ philosophe(x) \wedge grec(x)$. Si l'on présente une nouvelle instance

$$philosophe(Aristote) \wedge moyen(Aristote) \wedge grec(Aristote),$$

celle ci sera déclarée positive parce qu'elle implique la généralisation.

2.2.2 le critère de consistance

Ce critère doit délimiter l'ensemble des formules correctes g vis-à-vis de la théorie T et des ensembles P et N d'instances positives et négatives. Nous nous contentons ici de proposer le critère le plus répandu, que l'on peut découper en deux sous-propriétés que nous nommons consistance faible et complétude ($\not\vdash$ signifie "non $\vdash$").

$$Complétude : \forall p \in P\ \ T, p \vdash g$$

$$Consistance faible : \forall n \in N\ \ T, n \not\vdash g$$

A ces critères, on ajoute la notion de maximisation. Les programmes se contentent pour la plupart de calculer une ou les généralisations les plus spécifiques. Mitchell a introduit également le calcul des généralisations les plus générales, et concu un algorithme simple exploitant tout l'intérêt d'un tel calcul. C'est cet algorithme que nous présentons dans la section suivante.

2.3 L'algorithme d'élimination de candidats

L'idée de base de l'algorithme est de mettre à jour incrémentalement deux ensembles de généralisations S (plus spécifiques) et G (plus générales), "représentant" respectivement l'ensemble des instances positives et négatives. L'intérêt de retenir ces ensembles frontières est de n'avoir jamais à reconsidérer une instance déjà traitée. En effet une propriété importante de ces ensembles est la suivante :

$$\forall x, \exists s, \exists g,\ x \textit{ consistante} \iff s \in S \wedge g \in G \wedge s \vdash x \vdash g$$

debut

S=$\emptyset$; G=tout;

tant que il existe une instance et que S n'est pas réduit à un singleton égal à G

faire

lire(instance i);

cas type de i dans

positive $\rightarrow$

pour tout g $\in$ G faire si i $\not\vdash$ g alors supprimer g de G fait

S'=$\emptyset$;

pour tout s $\in$ S faire

si i $\not\vdash$ s alors supprimer s de S et construire S'= S'$\cup$généraliser(s,i,G) fait

pour tout s' $\in$ S', faire si $\forall$ s $\in$ S$\cup$S'$\backslash$s' s$\not\vdash$ s' alors ajouter s' à S fait

négative $\rightarrow$

pour tout s $\in$ S faire si i $\vdash$ s alors supprimer s de S fait

G'=$\emptyset$;

pour tout g $\in$ G faire

si i $\vdash$ g alors supprimer g de G et construire G'= G'$\cup$spécialiser(g,i,S) fait

pour tout g' $\in$ G', faire si $\forall$g $\in$ G$\cup$G'$\backslash$g' g'$\not\vdash$ g alors ajouter g' à G fait

fincas

fait

fin

Les opérations de spécialisation et de généralisation sont minimales et prennent en compte la partie de l'ensemble frontière en relation avec l'élément à modifier. Les deux ensembles frontières se contraignent ainsi mutuellement. On retiendra comme intérêt majeur de cet algorithme l'existence d'un critère d'arrêt de l'apprentissage (S=G=singleton), et l'utilisation possible à

tout moment d'une fonction de reconnaissance associée à S et G :
Etant donnée une nouvelle instance x de type inconnu, x est *positive* si $\forall s \in S$, $x \vdash s$, x est *négative* si $\forall g \in G$, $x \not\vdash g$, et x *fera progresser l'apprentissage* dans les autres cas.

3 Apprentissage de fonctions

C'est un des domaines les plus anciennement étudié de l'apprentissage. On y utilise des techniques d'optimisation du type poursuite de gradient (hill climbing) ou de l'apprentissage "par coeur", ou encore des techniques statistiques comme la régression. On part d'un modèle général de fonctions paramétrées $f : \vec{x} \rightarrow y$. On connaît un échantillon de valeurs correctes de la fonction ou on cherche a maximiser les valeurs de la fonction pour des vecteurs d'entrée donnés, formant l'ensemble des exemples. On cherche à apprendre des paramètres de la fonction modélisant au mieux les exemples observés.

3.1 Fonctions linéaires

Nous illustrons ici les techniques de type poursuite de gradient. L'idée consiste à affiner petit à petit les paramètres en faisant varier leur valeur de façon à minimiser l'écart $y - f(\vec{x})$. On ne peut garantir de s'arrêter que sur un optimum local.
L'exemple le plus connu d'apprentissage de ce type correspond aux perceptrons de Rosenblatt, particulièrement utilisés en reconnaissance des formes. Un perceptron est une machine calculant une fonction linéaire $y = \sum_i a_i x_i$, et ajustant les coefficients $\vec{a}$ en fonction des entrées $(\vec{x}, r)$, où r est dans sa version la plus élémentaire une variable indiquant le signe du résultat y (+1 si positif, -1 si négatif). Nous donnons un algorithme fort simple pour le perceptron, dû a Minsky et Papert.

debut

Choisir une valeur quelconque pour $\vec{a}$.

Tant que entrée$(\vec{x}, r)$

faire

calculer $y = r \sum_i a_i x_i$

si $y < 0$ alors $\vec{a}$ devient $\vec{a} + r\vec{x}$ fsi.

fait

fin

Le théorème de convergence du perceptron nous dit que si le problème est effectivement soluble (plus précisément, s'il existe un vecteur unitaire $\vec{b}$, tel que pour tout exemple $(\vec{x}, r)$, on ait $y > s > 0$), alors l'algorithme précédent converge en un nombre fini d'étapes (le nombre d'étapes est borné par $1/s^2$).

3.2 Fonctions hiérarchisables

Nous présentons un des travaux de Samuel afin d'illustrer ce que peut être un apprentissage "par coeur", non réduit à une simple mémorisation.
Samuel a travaillé dans le contexte du jeu de dames. Le but du programme est d'acquérir le niveau d'un bon joueur par apprentissage sur un grand nombre de parties. Il utilise pour celà différentes stratégies, dont l'une consiste à se servir de fonctions d'évaluation pour juger de la force d'une position donnée du jeu. Cette fonction d'évaluation repose sur différents attributs calculés sur le damier (nombre de dames, nombre de mouvements possibles,...). Elle est mémorisée dans un système de tables d'indexation hiérarchisé appelées tables de signature. On suppose donc donnés un ensemble d'exemples de couples $(\vec{x}, y)$ comme précédemment, ainsi qu'une classification sur les attributs. A tout noeud de la classification est associée une table de signature dont les entrées sont les feuilles du noeud et la sortie les valeurs d'une variable qui servira d'entrée pour le noeud suivant (sauf la variable finale, qui donnera la valeur y de la fonction recherchée). On se fixe les domaines de valeur pour tous les noeuds. Le but de l'algorithme est de remplir les valeurs de sortie de chaque table. A chaque noeud, on peut associer deux ensembles de variables : celles en entrée de la table associée, V_{in}, et celles n'appartenant pas à la descendance du noeud, V_{out}. On considère alors les vecteurs $\vec{V_{out}}$ des valeurs de la fonction pour l'ensemble des valeurs de $\vec{V_{in}}$. L'idée générale est de regrouper ces vecteurs en n groupes, n étant le cardinal du domaine de la variable de sortie. Samuel caractérisait chaque vecteur par un indice, les indices étant regroupés ensuite en n valeurs d'intervalle. Biermann a perfectionné la technique en appliquant une méthode classique en analyse de données de regroupement : la méthode des centres mobiles ou nuées dynamiques. L'algorithme partitionne un ensemble en un nombre fixé n de classes par une série convergente d'une séquence de choix de n centres d'attractions (par exemple, des éléments de l'ensemble) suivi d'une allocation des élements de l'ensemble au centre le plus "proche".

3.3 Pour aller plus loin

Il existe beaucoup d'autres travaux qu'on peut rattacher à ce domaine et nous ne citerons que les principales orientations pour conclure cette partie.
Une première approche consiste à utiliser des méthodes statistiques pour évaluer les paramètres de fonctions complexes. Ainsi, l'apprentissage de fonctions quadratiques a été exploré a l'aide de classificateurs bayésiens.
L'approche connexioniste quant à elle, (ou neuronale, du fait du souci d'une partie des auteurs de modéliser les neurones humains) est une extension des perceptrons où la fonction est calculée de façon distribuée selon un réseau de dépendances donné entre éléments du calcul. La technique principale d'apprentissage utilisée est la rétro-propagation sur les paramètres de la fonction (les "poids"), depuis la sortie du réseau, de l'écart constaté entre la valeur prédite et la valeur souhaitée de la fonction.
Enfin, nous signalerons une branche aux résultats encore peu nombreux, qui s'intéresse à la (re-)découverte de lois empiriques à partir de résultats d'expérimentations. On se donne un ensemble de fonctions élémentaires (addition, soustraction, multiplication, division, sinus,...). Le "paramétrage" consiste alors à trouver progressivement une combinaison de ces fonctions conduisant à une fonction constante sur les résultats d'expérimentations. On utilise essentiellement des techniques combinatoires et heuristiques, mais aussi statistiques (régression).

4 Apprentissage d'arbres de décision

C'est probablement la technique d'apprentissage automatique la plus diffusée. Un arbre de décision est un arbre dont les noeuds sont étiquetés par un attribut (taille, couleur,...), les branches par des valeurs possibles de cet attribut (1.5, rouge,...), et les feuilles par les noms de classes d'objets (espèce, maladie,...). Les arbres sont utilisés en identification d'objets dont on connait les valeurs d'attributs mais pas la classe. On parcourt l'arbre depuis la racine en prenant à chaque noeud la branche correspondant à la valeur d'attribut observée sur l'objet. La feuille d'arrivée donne la classe de l'objet.
Nous présentons un des algorithmes les plus simples utilisé pour la construction automatique d'arbres de décision : ID3, conçu par Quinlan. De façon préliminaire, constatons tout d'abord les faits suivants : 1) un attribut engendre une partition des objets selon les valeurs qu'ils prennent pour ce descripteur. 2) A un ensemble d'objets dont on connaît les classes, on peut associer un arbre "canonique", qui comporte un nombre minimal de noeuds binaires pour départager les différentes classes. Ceci correspond à la notion d'entropie, ou encore de quantité d'information associée à l'ensemble (si c est le nombre total d'éléments et c_i le nombre d'éléments dans la classe i, on peut la calculer par la formule $log_2(c) - 1/c \sum_{i=1}^{nbdeclasses} p_i log_2(p_i)$)

debut

L'arbre de décision initial est vide

On choisit un sous-ensemble SE de taille fixée de l'ensemble des instances.

Tant que il reste des exemples (objets) mal classés

faire

développer l'arbre { *pour chaque attribut, calculer la moyenne des entropies des éléments de la partition correspondante. choisir l'attribut qui minimise cette moyenne (entropie résiduelle)*

Classer les instances restantes avec l'arbre construit

fait

fin

L'algorithme n'a pas de convergence théorique, mais une bonne convergence pratique. Il s'agit d'une recherche heuristique pour laquelle il est difficile de dégager un critère global à optimiser. Il existe de nombreux raffinements de cette version de base, tant au niveau du choix de l'indice de sélection des attributs que des stratégies d'élagage de l'arbre pendant ou après sa construction.

5 Apprentissage en résolution de problèmes

De façon formalisée, un résolveur de problèmes peut se concevoir comme un programme de recherche de combinaisons d'opérateurs permettant de transformer un état initial donné du monde

sur lequel agissent les opérateurs en un état but donné. Chaque opérateur n'est applicable que sous certaines conditions. Prenons l'exemple de l'intégration formelle. Un des opérateurs est l'intégration par parties. Une précondition pratique d'utilisation de l'opérateur pourrait être : une des fonctions du produit de fonctions est trigonométrique. Un état initial correspond à une intégrale donnée, le but est d'obtenir la supression du signe d'intégration dans la formule, les états intermédiaires sont les formules transformées à chaque étape.
Nous présentons deux techniques d'apprentissage dans ce contexte, qui reflètent les deux types de connaissances qui peuvent être apprises dans ce domaine. La première, l'apprentissage par recherche d'explications, vise à apprendre les préconditions d'utilisation des opérateurs. La seconde, l'apprentissage de macro-opérateurs, vise la constitution de nouveaux opérateurs, formés par composition des opérateurs élémentaires disponibles.

5.1 Apprentissage par recherche d'explications

L'apprentissage par recherche d'explications (Explanation Based Learning) est appliqué dans les situations où on possède déjà une définition (théorique) des concepts à apprendre, mais où cette définition est peu utile car trop complexe à utiliser en pratique. La résolution de problèmes est un contexte idéal où ce genre de situation existe (différence entre la maîtrise théorique et pratique d'un outil...).
Techniquement, l'algorithme d'apprentissage est une recherche des préconditions minimales de validité d'un calcul. La version de base de l'algorithme est la suivante (il s'agit plus d'un découpage en tâches que d'une succession d'étapes).

debut

Etablissement de la preuve d'un but concret (dont toutes les variables sont instanciées), correspondant au calcul sur un exemple;

Généralisation de la preuve sur un but variabilisé, en conservant la même structure de l'arbre de preuve (mêmes règles d'inférence);

Simplification de la preuve généralisée en ne la développant que jusqu'à un certain niveau d'abstraction, correspondant à la sélection des conditions utiles, opérationnelles.

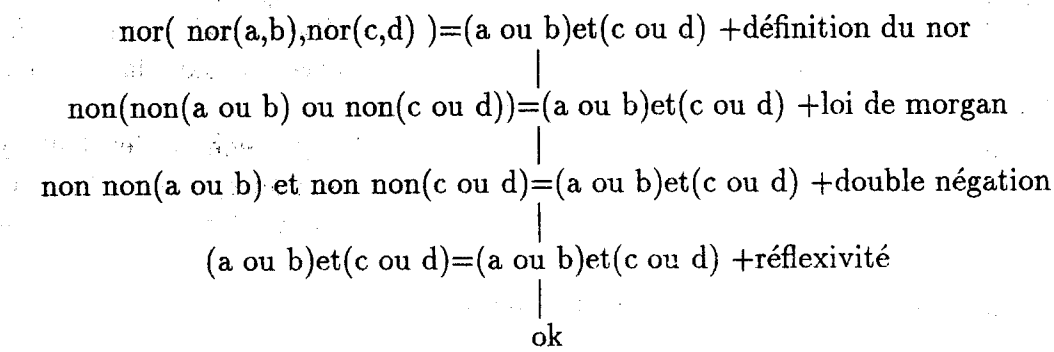
fin

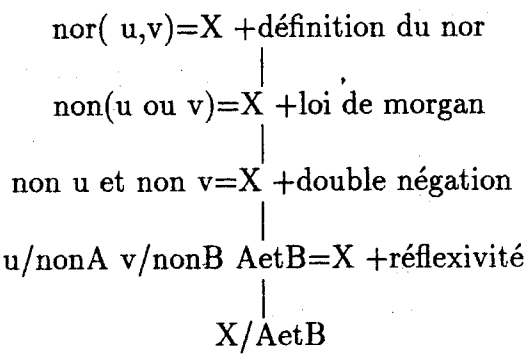
Voici un exemple d'utilisation, un programme apprenti de conception de circuits électroniques. Supposons qu'un concepteur doive implémenter un circuit réalisant la fonction logique

$$out(a, b, c, d) = (a \vee b) \wedge (c \vee d).$$

Il propose la solution pratique suivante, visant à minimiser le nombre de composants différents nécessaires : $out(a, b, c, d) = nor(nor(a, b), nor(c, d))$. L'utilisation de l'apprentissage par recherche d'explications permet de généraliser cet exemple de conception, pour produire une règle de conception pratique, s'appliquant dans un plus grand nombre de cas. Il faut pour cela que le programme dispose d'une théorie de l'égalité de fonctions logiques, capable d'expliquer pourquoi le circuit proposé par le concepteur est correct vis a vis de la spécification (lois de morgan, double négation,...). Il faut également préciser le niveau d'abstraction souhaité dans

la preuve à généraliser. Ici, on pourra se contenter d'un niveau où les fonctions logiques correspondent à des composants électroniques élémentaires. Nous donnons ci-dessous les deux arbres de preuve produits.





La règle de conception correspondant à l'arbre de preuve généralisé est :

Pour réaliser $A \wedge B$ *construire le circuit* $nor(\neg A, \neg B)$

5.2 Apprentissage de macro-opérateurs

Une autre façon d'améliorer les performances d'un résolveur de problèmes est d'augmenter l'ensemble initial des opérateurs applicables par de nouveaux opérateurs, les macro-opérateurs, correspondant à des séquences d'opérateurs élémentaires.
Nous illustrons ce type d'études avec le travail d'Iba, fondé sur l'existence d'une fonction heuristique d'évaluation pour estimer si l'application d'un opérateur permet de s'approcher de la solution. Ces fonctions heuristiques sont en général faciles à obtenir, mais ont un comportement non monotone sur l'espace de recherche. Nous prenons l'exemple du jeu de taquin (eight puzzle) pour éclaircir le propos.
Le jeu peut être formalisé de la manière suivante. Chaque état est une matrice 3x3 dont les éléments sont numérotés de 0 à 8. En général, le but est une matrice dont les éléments forment une matrice ordonnée. L'état initial est donné dans une configuration quelconque. Il y a une seule opération élémentaire d'échange, qui consiste à échanger l'élément 0 avec un élément adjacent dans la matrice, en ligne ou en colonne. La stratégie choisie place les éléments séquentiellement de 1 à 8 à leur place définitive.
Une bonne fonction d'évaluation pour ce problème est la fonction rendant le vecteur (A,B,C), où A est le nombre d'éléments mal placés, B la distance de Manhattan entre la position actuelle du prochain élément à placer et sa position finale, et C la distance entre la position actuelle du

prochain élément à placer et la position actuelle du 0. Les valeurs de la fonction d'évaluation sont comparées lexicographiquement. On cherche à la minimiser.
L'apprentissage s'effectue simplement par détection de minima locaux de la fonction d'évaluation, puis construction de macro-opérateurs correspondant aux séquences détectées par extraction et variabilisation des positions variantes (plus l'élément 0).
Sur notre exemple, considérons la séquence de mouvements élémentaires suivante, utilisée lors d'une résolution.

$$\begin{pmatrix} 1 & 5 & 4 \\ 7 & 0 & 2 \\ 8 & 6 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 5 & 4 \\ 7 & 2 & 0 \\ 8 & 6 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 5 & 0 \\ 7 & 2 & 4 \\ 8 & 6 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 5 \\ 7 & 2 & 4 \\ 8 & 6 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 5 \\ 7 & 0 & 4 \\ 8 & 6 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 5 \\ 7 & 4 & 0 \\ 8 & 6 & 3 \end{pmatrix}$$

La fonction d'évaluation vaut (6,2,1) pour la première matrice et (5,2,1) pour la dernière.

Un minimum local est atteint à l'avant dernière matrice avec la valeur (4,2,2). Le macro-opérateur correspondant fait passer de la matrice $\begin{pmatrix} 1 & 5 & 4 \\ 7 & 0 & 2 \\ 8 & 6 & 3 \end{pmatrix}$ à la matrice $\begin{pmatrix} 1 & 2 & 5 \\ 7 & 0 & 4 \\ 8 & 6 & 3 \end{pmatrix}$, ce qui conduit à l'apprentissage d'un macro-opérateur effectuant une permutation circulaire d'éléments:

$$\begin{pmatrix} B & C \\ 0 & A \end{pmatrix} \longrightarrow \begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$$

Bibliographie

[1] Machine Learning Journal, Kluwer Academics Pub.
La revue du domaine

[2] P. Cohen, E. FeigenBaum "The Handbook of Artificial Intelligence, Vol 3"
La première compilation des études sur l'apprentissage

[3] R. Michalski, J. Carbonell, T. Mitchell "Machine Learning, an Artificial Intelligence Approach, Vol 1" Tioga, Palo Alto CA 1983

[4] R. Michalski, J. Carbonell, T. Mitchell "Machine Learning, an Artificial Intelligence Approach, Vol 2" Morgan Kaufman, Los Altos CA 1986

[5] R. Michalski, Y. Kodratoff "Machine Learning, an Artificial Intelligence Approach, Vol 3"
Morgan Kaufman, San Mateo CA 1990
Des recueuils d'articles pour le chercheur chevronné avec beaucoup de références bibliographiques

[6] J. Shavlik, T. Dietterich "Readings in Machine Learning"
Morgan Kaufman, San Mateo CA 1990
Une très bonne sélection d'articles "classiques" du domaine, avec des introductions explicatives.

[7] A. Biermann, J. Fairfield, T. Beres "Signature Table Systems and Learning"
IEE trans. on Systems, Man, and Cybernetics, vol SMC-12, No 5, sept 1982.

[8] R. Quinlan "Induction of Decision trees" Machine Learning, 1, 1986, pp 81-106

[9] T. Mitchell "Generalization as search", Artificial Intelligence 18, p. 203-226, 1982

[10] G. Iba "A Heuristic Approach to the Discovery of Macro-Operators"
Machine Learning, 3, 1989, pp 285-317

Table des matières