Optimisation d'un tuteur intelligent à partir d'un jeu de données fixé

Lucie Daubigney[1, 3] Matthieu Geist[1] Olivier Pietquin[1,2]
(1) Equipe de recherche IMS, Supélec (Metz, France)
(2) UMI 2958, GeorgiaTech - CNRS (Metz, France)
(3) Equipe-projet Maia, Loria (Nancy, France)
prénom.nom@supelec.fr

RÉSUMÉ

Dans cet article, nous présentons une méthode générale pour optimiser un tuteur intelligent dans le domaine de l'acquisition d'une seconde langue. Plus particulièrement, le processus d'optimisation a pour but de trouver une stratégie qui propose la meilleure séquence de phases d'évaluation et d'enseignement afin de maximiser l'augmentation des connaissances de l'apprenant. La principale caractéristique de la méthode proposée est qu'elle est capable d'apprendre la meilleure stratégie à partir d'un jeu fixe de données, collectées à partir d'une stratégie définie à la main. Ainsi, aucun modèle, ni cognitif ni probabiliste de l'apprenant, n'est nécessaire. Seules sont requises des observations du comportement de l'apprenant alors qu'il interagit avec un système non-optimal. Pour ce faire, un algorithme de programmation dynamique approchée en mode hors-ligne est utilisé : l'algorithme LSPI (*Least Square Policy Iteration*). Des résultats obtenus avec des données simulées semblent prometteurs.

ABSTRACT

Optimization of a tutoring system from a fixed set of data

In this paper, we present a general method for optimizing a tutoring system with a target application in the domain of second language acquisition. More specifically, the optimisation process aims at learning the best sequencing strategy for switching between teaching and evaluation sessions so as to maximise the increase of knowledge of the learner in an adapted manner. The most important feature of the proposed method is that it is able to learn an optimal strategy from a fixed set of data, collected with a hand-crafted strategy. This way, no model (neither cognitive nor probabilistic) of learners is required but only observations of their behavior when interacting with a simple (non-optimal) system. To do so, a particular batch-mode approximate dynamic programming algorithm is used, namely the Least Square Policy Iteration algorithm. Experiments on simulated data provide promising results.

MOTS-CLÉS : Tuteurs intelligents, apprentissage par renforcement.

KEYWORDS: Tutoring systems, reinforcement learning.

Actes de la conférence conjointe JEP-TALN-RECITAL 2012, volume 1: JEP, pages 241–248,
Grenoble, 4 au 8 juin 2012.

1 Introduction

Le travail décrit dans cet article se place dans le cadre plus général d'un projet européen [1] qui vise à développer un tuteur intelligent pour l'aquisition d'une deuxième langue (particulièrement pour le français et l'allemand). Dans le cadre de ce projet, un jeu sérieux, intégré dans Second Life, et qui exploite un environnement de réalité virtuelle en 3 dimensions a été conçu (I-FLEG) (Amoia *et al.*, 2011). De la situation privilégiée dans laquelle se trouve l'apprenant lorsqu'un ordinateur l'assiste à acquérir de nouvelles connaissances, il est possible de tirer avantage de plusieurs caractéristiques. Parmi celles-ci, la personnalisation de la séquence d'apprentissage est à souligner et fera l'objet de cette contribution. De plus, le technologie basée sur le web facilite la collecte d'une grande quantité de données qui peuvent être utilisées pour optimiser le fonctionnement du tuteur intelligent.

Il a été montré assez tôt que la personnalisation de l'enseignement est très importante dans la relation entre enseignants et apprenants (Bloom, 1968). Idéalement, chaque apprenant devrait recevoir des cours adaptés qui lui permettaient d'obtenir le meilleur en fonction de ses capacités. C'est tout naturel de penser que cette situation pourrait être rencontrée avec des tuteurs intelligents, installés sur des ordinateurs personnels ou accessibles depuis l'Internet. Pourtant, la situation actuelle est loin d'être satisfaisante car les systèmes présentement commercialisés sont conçus pour un large public et non pour chaque apprenant. Pire encore, ils s'addressent à un étudiant moyen qui généralement n'existe même pas. C'est particulièrement vrai dans le contexte de l'aquisition d'une seconde langue où les erreurs sont très dépendantes de l'apprenant, à cause notamment de confusions lexicales et d'erreurs de prononciation liées à des causes culturelles et d'éducation. Il est donc important de concevoir des systèmes qui sont capables d'adapter leur comportement au profil particulier de chaque apprenant.

Nous allons supposer ici que le degré de liberté de l'interface est dans la séquence constituée d'une suite de choix entre phase d'enseignement et phase d'évaluation. Une phase d'enseignement aura pour but d'améliorer les connaissances d'un apprenant tandis qu'une phase d'évaluation aura pour objectif de quantifier ce savoir. Ainsi, étant donnée une situation (définie par rapport à l'historique des interactions avec l'apprenant), le système devra choisir quelle phase proposer ensuite. L'adaptation à l'apprenant intervient dans la séquence de décisions. Celle-ci diffère d'un apprenant à l'autre. Le problème d'adaptation du comportement du système à l'utilisateur peut ainsi être vu comme un problème de décisions séquentielles.

Dans cet article, nous proposons de résoudre ce problème en utilisant une méthode d'apprentissage automatique, l'apprentissage par renforcement (AR) (Sutton et Barto, 1998). Particulièrement, nous avons utilisé un algorithme d'AR en mode hors-ligne (appelé *Least Square Policy Iteration* (LSPI) (Lagoudakis et Parr, 2003)). Ces méthodes ont été récemment et avec succès utilisées dans le domaine des systèmes de dialogues parlés (Pietquin *et al.*, 2011b). Ce travail se démarque de précédentes tentatives d'utiliser l'AR dans le contexte des tuteurs intelligents (Beck *et al.*, 2000; Iglesias *et al.*, 2009) car l'apprentissage se fait en mode hors-ligne avec un jeu de données fixé. Seules les données collectées avec un système dont le comportement et les prises de décisions sont codées à la main ou bien provenant des traces d'utilisation de systèmes déjà déployés sont nécessaires. En conséquence, contrairement aux précédents travaux, la méthode proposée ici ne requière ni une modélisation de l'apprenant, ce qui évite donc les erreurs liées à l'imperfection du modèle, ni une interaction avec ce dernier durant l'apprentissage. Le fait de ne

1. ALLEGRO : www.allegro-project.eu, financé par le programme INTERREG-IVa et la Région Lorraine.

pas laisser interagir l'apprenant avec le tuteur pendant que celui-ci apprend le bon comportement permet de ne pas mettre l'apprenant face à des situations non maîtrisées (qui pourraient le lasser). En effet, durant l'apprentissage de la meilleurs séquence de décisions, la cohérence du comportement du tuteur n'est pas garantie.

Le reste de l'article est organisé comme suit : la section 2 présente l'apprentissage par renforcement de façon théorique. Ensuite, la section 3 montre comment le problème d'optimisation dans le cadre du tutorat remplit le paradigme de l'AR. Des résultats expérimentaux sont présentés dans la section 4 et montrent l'efficacité de la méthode. Enfin, la section 5 conclut le travail.

2 Apprentissage par renforcement

L'apprentissage par renforcement (AR) (Sutton et Barto, 1998) est un paradigme général d'apprentissage automatique qui a pour but de résoudre des problèmes de prise de décisions séquentielles. Dans ce cadre, un agent interagit avec un système qu'il essaie de contrôler. Le système est composé d'états et le contrôle consiste à exercer des actions sur le système. Après que chaque action a été effectuée par l'agent, le système passe d'un état à un autre et génère une récompense immédiate qui est visible par l'agent. Le but de l'agent est d'apprendre une correspondance entre les états et les actions qui vont lui permettre de maximiser un cumul de récompenses (récompenses à long terme). Pour cela, l'agent recherche la meilleure séquence d'actions et non les actions qui sont les meilleures localement.

2.1 Processus décisionnels de Markov

Pour résoudre le problème d'apprentissage par renforcement décrit ci-dessus, le paradigme des Processus décisionnels de Markov (PDM) est traditionnellement utilisé. Un PDM est défini par un n-uplet $\{S, A, R, P, \gamma\}$, où S est l'ensemble constitué de tous les états possibles, A l'ensemble d'actions, R la fonction de récompense, P l'ensemble des probabilités de transisitions markoviennes et γ est le facteur d'actualisation (pondérant les récompenses futures). Une stratégie ou une politique π est une application de S dans A. Etant donnée une politique π, chaque état de S peut être associé à une valeur ($V^\pi : S \rightarrow \mathbb{R}$) définie comme étant l'espérance de la somme des récompenses pondérées obtenues par l'agent sur un horizon infini en partant de l'état s et en suivant la politique π :

$$V^\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k r_k | s_0 = s, \pi]. \quad (1)$$

Une fonction sur une paire état-action peut être définie : $Q^\pi : S \times A \rightarrow \mathbb{R}$. Cela ajoute un degré de liberté dans le choix de la première action choisie :

$$Q^\pi(s,a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi]. \quad (2)$$

La fonction $Q^*(s,a)$ est la fonction Q optimale associée à la politique optimale π^*. Cette dernière est celle qui maximise le valeur de chaque état (ou de chaque paire état-action) : $\pi^* = \text{argmax}_\pi V^\pi = \text{argmax}_\pi Q^\pi$. La politique optimale est *gloutonne* par rapport à la fonction Q

optimale : $\pi^*(s) = \text{argmax}_{a \in A} Q^*(s,a)$. La programmation dynamique (Bellman, 1957) a pour but de calculer la politique optimale, en utilisant la fonction Q comme intermédiaire, dans le cas où les probabilités de transition ainsi que la fonction de récompense sont connues. Particulièrement, l'algorithme d'itération de la politique calcule la politique optimale de façon itérative. Une politique initiale est arbitrairement choisie : π_0. A l'itération k, la politique π_{k-1} est évaluée, c'est-à-dire que la fonction Q qui lui est associée, $Q^{\pi_{k-1}}(s,a)$, est calculée. Pour cela, la propriété de Markov sur les probabilités de transition est utilisée pour réécrire l'équation 2 :

$$\begin{aligned} Q^{\pi}(s,a) &= E_{s'|s,a}[R(s,a,s') + \gamma Q^{\pi}(s', \pi(s'))] \\ &= T^{\pi} Q^{\pi}(s,a) \end{aligned} \tag{3}$$

Cette équation est appelée l'équation d'évaluation de Bellman et T^{π} est l'opérateur associé. L'opérateur T^{π} est linéaire et l'équation 3 définit ainsi un système linéaire qui peut être résolu de manière exacte. La politique est ensuite améliorée en utilisant le fait que π_k est gloutonne par rapport à $Q^{\pi_{k-1}}$:

$$\pi_k(s) = \underset{a \in A}{\text{argmax}}\, Q^{\pi_{k-1}}(s,a) \tag{4}$$

Les étapes d'évaluation et d'amélioration sont répétées jusqu'à ce que π_k converge vers π^* (qui peut être démontré comme se produisant après un nombre fini d'itérations, quand $\pi_k = \pi_{k-1}$).

2.2 Programmation dynamique approchée

Bien qu'elle paraisse très intéressante, la méthode proposée ci-dessous est difficilement applicable à des situations réelles pour deux raisons. Tout d'abord, il est supposé que les probabilités de transition ainsi que la fonction de récompense sont connues. Cela est rarement le cas, surtout avec des systèmes qui interagissent avec des humains car cela reviendrait à modéliser leur comportement. Le plus souvent, seulement des exemples d'interations sont disponibles, au travers de collectes de données et de traces d'utilisation qui constituent des trajectoires dans l'espace état-action. Mais il devient alors nécessaire d'apprendre à partir d'un jeu de données fixé. Deuxièmement, l'itération de la politique suppose que la fonction Q puisse être exactement représentée et que sa valeur puisse être stockée dans un tableau pour chaque paire état-action, pour qu'ainsi l'expression 3 représente un système d'équations.

Cependant, pour des problèmes réels, les espaces d'état et/ou d'action sont souvent trop grands (et même parfois continus) pour que cette hypothèse tienne. La programmation dynamique approchée (PDA) a pour but d'estimer la politique optimale à partir de trajectoires lorsque l'espace d'état est trop grand pour une représentation tabulaire. La fonction Q est alors approchée par une représentation paramétrique $\hat{Q}_\theta(s,a)$ tandis que la connaissance du modèle est remplacée par une base d'exemples de transitions. Dans cet article, une approximation linéaire de la fonction Q est choisie : $\hat{Q}_\theta(s,a) = \theta^T \phi(s,a)$ où $\theta \in \mathbb{R}^p$ est un vecteur de paramètres et $\phi(s,a)$ un jeu de *fonctions de base* (ou *attributs*). Toutes les fonctions qui peuvent se mettre sous cette forme définissent un *espace d'hypothèses* $\mathscr{H} = \{\hat{Q}_\theta | \theta \in \mathbb{R}^p\}$. Chaque fonction Q peut se projeter sur cet espace à l'aide d'un opérateur de projection Π défini tel que :

$$\Pi Q = \underset{\hat{Q}_\theta \in \mathscr{H}}{\text{argmin}} \|Q - \hat{Q}_\theta\|^2. \tag{5}$$

Le but des algorithmes de PDA est de cacluler le meilleur jeu de paramètres θ étant donné les fonctions de base.

2.2.1 *Least-Squares Policy Iteration*

Least-Squares Policy Iteration (LSPI) est un algorithme de PDA (Lagoudakis et Parr, 2003). LSPI est inspiré de méthodes d'itération sur la politique et alterne phase d'évaluation avec phase d'amélioration de la politique. La phase d'amélioration est la même que celle décrite précédemment (la politique est gloutonne par rapport à la fonction Q évaluée) mais la phase d'évaluation doit apprendre une représentation approximative de la fonction Q en utilisant des échantillons. Dans LSPI, cela est fait en utilisant une version *off-policy* de l'algorithme *Least-Squares Temporal Differences* (LSTD) (Bradtke et Barto, 1996), c'est-à-dire une version dans laquelle la politique évaluée n'est pas celle qui a généré les données.

L'objectif est de trouver une approximation Q_θ de Q. Seulement, les valeurs de la fonction ne sont pas directement observables. Le problème est donc plus difficile à réoudre qu'un problème de régression. La fonction Q étant le point fixe de l'opérateur de Bellman, il pourrait paraître raisonnable de chercher Q_θ tel que $Q_\theta \approx TQ_\theta$. Toutefois, il n'y a aucune raison que l'espace d'hypothèse soit stable par application de l'opérateur de Bellman. LSTD consiste donc à calculer le point fixe de $Q_\theta = \Pi T Q_\theta$, qui existe bien.

En pratique, T^π n'est pas connu (les probabilités de transition de sont pas connues) mais un jeu de N transitions $\{(s_j, a_j, r_j, s'_j)_{1 \leq j \leq N}\}$ est disponible. Le problème de point fixe précédent s'exprime alors ainsi :

$$\theta_\pi = \underset{\theta}{\operatorname{argmin}} \sum_{j=1}^{N} C_j^N(\theta, \theta_\pi), \tag{6}$$

$$C_j^N(\theta, \theta_\pi) = (r_j + \gamma \hat{Q}_{\theta_\pi}(s'_j, \pi(s'_j)) - \gamma \hat{Q}_\theta(s_j, a_j))^2.$$

Grâce à la paramétrisation linéaire, une solution analytique peut être calculée :

$$\theta_\pi = (\sum_{j=1}^{N} \phi_j \Delta\phi_j^\pi)^{-1} \sum_{j=1}^{N} \phi_j r_j \tag{7}$$

$$\text{avec } \phi_j = \phi(s_j, a_j)$$

$$\text{et } \Delta\phi_j^\pi = \phi(s_j, a_j) - \gamma\phi(s'_j, \pi(s'_j)).$$

L'algorithme LSPI fonctionne ainsi comme suit. Une politique initiale π_0 est choisie. Ensuite, à l'itération k (avec $k > 1$), la fonction Q de la politique π_{k-1} est estimée en utilisant LSTD et π_k est gloutonne par rapport à cette fonction Q estimée. L'algorithme se termine quand un critère d'arrêt est atteint, par exemple quand la différence entre deux politiques consécutives est inférieure à une certaine valeur.

3 Le comportement du tuteur vu comme un PDM

Ainsi que présentée dans l'introduction, la personnalistion d'un tuteur intelligent peut se voir comme équivalente à un problème de décisions séquentielles dans lequel l'agent doit alterner entre phases d'enseignement et phases d'évaluation. Par exemple, dans le cadre de l'acquisition d'une seconde langue, le tuteur peut choisir de proposer un exercice de grammaire, suivi d'un exercice de conjugaison et seulement ensuite proposer une évaluation à l'apprenant sur les deux notions précedemment enseignées. L'utilisation de l'apprentissage par renforcement pour

résoudre ce problème d'optimisation a déjà été proposé dans (Beck *et al.*, 2000) et (Iglesias *et al.*, 2009). Le travail présenté dans cette publication diffère des précédents car il se propose d'utiliser une méthode qui apprend une stratégie optimale à partir de données fixées. Ainsi, aucune interaction avec l'apprenant n'est requise durant l'apprentissage et n'importe quel système peut être amélioré en utilisant simplement des traces d'utilisation. Pour trouver la séquence optimale de décisions à l'aide de l'apprentissage par renforcement, il faut traduire le problème de tutorat dans le cadre du paradigme des processus décisionnels de Markov et ainsi définir un espace d'état, un espace d'action et une fonction de récompense (les probabilités de transition ne sont pas connues mais l'information qu'elles apportent est remplacée par un jeu de données provenant de traces d'utilisation du système).

En ce qui concerne les actions, elles sont au nombre de deux : commencer une phase d'évaluation ou commencer une phase d'enseignement. La représentation de l'état doit contenir des informations sur le contexte de l'interaction, c'est-à-dire l'information suffisante mais nécessaire pour prendre une décision. Ici l'espace d'état est défini comme un vecteur à deux dimensions : la première dimension est le taux de bonnes réponses que l'apprenant a déjà fournies (valeur continue entre 0 et 1) et la deuxième dimension est le nombre de phases d'enseignement que le système a déjà proposées (valeur entière). Il est à noter que cette représentation hybride (continue/discrète) de l'espace d'état est totalement différente de ce qui a été proposé dans différents travaux et qu'elle nécessite une approximation de la fonction de valeur. Le but pour le système est ici de tirer le meilleur de chaque apprenant et non de lui faire atteindre un taux de réussite fixé, en-dessous duquel on considère que l'apprenant a échoué (par exemple, dans (Iglesias *et al.*, 2009) ce taux est fixé à 90% ; la progression de l'apprenant n'est pas prise en compte s'il ne l'atteint jamais au cours de l'apprentissage). Enfin, la récompense est fournie par le taux de bonnes réponses de l'apprenant après une phase d'évaluation (une récompense est obtenue après chaque phase d'évaluation, aucune après celle d'enseignement). A nouveau, le but est de maximiser le cumul de ces taux en fonction des capacités de l'apprenant. Pour cela, le système doit proposer la séquence qui fait augmenter le plus rapidement les connaissances de l'élève puisque son but est de maximiser la séquence de récompenses.

4 Expériences

Nous n'avions pas de données disponibles au moment de la rédaction de cet article ; nous avons donc simulé des interactions entre le système et l'apprenant. Le modèle de l'apprenant est inspiré de (Corbett et Anderson, 1994). Le modèle est basé sur un jeu de probabilités qui simulent le fait que les connaissances d'un apprenant augmentent ou non après avoir suivi une phase d'enseignement et qui simulent des réponses à des questions. Il est important de garder à l'esprit que le modèle a seulement été utilisé pour générer des données concordantes avec notre représentation d'état mais qu'il n'est pas explicitement pris en compte pour élaborer la stratégie d'apprentissage. Du point de vue de l'apprentissage par renforcement, tout se passe comme si les données étaient générées par des utilisateurs réels. Générer des données avec un modèle a aussi l'avantage de tester les stratégies de façon statistiquement cohérente. Les données prennent la forme de traces d'interactions (une interaction étant simplement une décision du sytème suivie de la réaction de l'apprenant). Pour obtenir les données, l'utilisateur simulé doit interagir avec un système initial dont le comportement est défini par une politique codée à la main (ici le choix des actions est totalement aléatoire : des phases d'évaluation et d'enseignement alternent avec une probabilité de 50%). L'algorithme LSPI présenté section 2.2.1 est appliqué ensuite sur des

jeux de données de tailles différentes. Les résultats sont présentés figure 1.

Sur cette figure, la récompense cumulée obtenue par l'apprenant en utilisant la politique apprise par le système est tracée en fonction du nombre d'interactions contenues dans le jeu de données d'entraînement utilisé par l'algorithme LSPI. Le but de cette expérience est d'identifier le nombre d'interactions requis pour apprendre une politique dont les performances sont supérieures à une simple politique définie à la main. Les performances des politiques apprises sont comparées à celles des politiques aléatoires utilisées pour la collecte de données et à celles issues d'une politique codée à la main qui alterne des phases d'enseignement et d'évaluation.

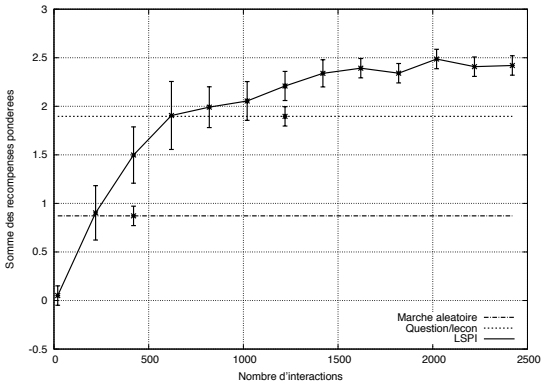

FIGURE 1 – Résultats

Il apparaît clairement sur la figure qu'en utilisant 500 interactions (une interaction n'étant pas une session entière de tutorat mais simplement une décision suivie par la réaction de l'apprenant, ce qui fait de 500 un nombre plutôt bas), la politique apprise est meilleure qu'une politique aléatoire. Ensuite, après 1000 intéractions, la politique apprise devient meilleure que la politique codée à la main. Il est donc possible d'utiliser les traces d'un système existant pour apprendre des politiques optimales.

De façon à mesurer la reproductibilité de l'apprentissage (c'est-à-dire la sensibilité à la composition du jeu de données), LSPI a été appliqué 100 fois et les politiques apprises ont été testées 1000 fois sur les apprenants simulés. L'intervalle de confiance à 95% a aussi été caclulé. Il montre que les résultats ne sont que peu sensibles à l'aléat dans les données. C'est assez important car dans des applications réelles, il n'est pas possible de contrôler la qualité des données puisque seules des traces d'utilisation sont disponibles. Après 1500 interactions, l'intervalle de confiance ne varie plus beaucoup.

5 Conclusion

Dans cette contribution, une méthode pour optimiser un tuteur intelligent qui aide à l'acquisition d'une seconde langue a été proposée. Le problème d'optimisation est d'abord exprimé comme un problème de décisions séquentielles qui peut être résolu grâce à un algorithme d'apprentissage

par renforcement. Puisque le comportement des apprenants est difficile à prédire, une méthode sans modèle est préférable. La méthode choisie n'utilise donc que des traces d'interactions entre l'apprenant et le système à optimiser. Les performances de la stratégie d'interactions apprise dépassent celles des stratégies basiques utilisées pour collecter les données.

Dans le futur, les collectes de données avec des étudiants réels commenceront en utilisant l'environnement virtuel I-FLEG. Ainsi, une perspective immédiate est de collecter des traces d'utilisation de ce système pour apprendre des stratégies optimales d'enseignement. Nous souhaiterions aussi utiliser d'autres algorithmes d'AR (Geist et Pietquin, 2010) capables d'utiliser l'incertitude sur l'estimation des paramètres de la fonction Q (Geist et Pietquin, 2011) de façon à améliorer la stratégie en ligne (pendant que le système est utilisé), grâce à des stratégies d'exploration qui évitent de perturber l'apprenant. Cette méthode a déjà été appliquée avec succès à des gestionnaires de dialogues parlés (Pietquin *et al.*, 2011a; Daubigney *et al.*, 2011).

Références

Amoia, M., Gardent, C. et Perez-Beltrachini, L. (2011). A serious game for second language acquisition. *In Proceedings of the Third International Conference on Computer Aided Education (CSEDU 2011)*, Noordwijkerout (The Netherlands).

Beck, J. E., Woolf, B. P. et Beal, C. R. (2000). ADVISOR : A machine learning architecture for intelligent tutor construction. *In Proceedings of the National Conference on Articial Intelligence*, pages 552–557, Menlo Park, CA. MIT Press.

Bellman, R. (1957). *Dynamic Programming*. Dover Publications, sixth édition.

Bloom, B. S. (1968). Learning for mastery. *Evaluation comment*, 1(2):1–5.

Bradtke, S. J. et Barto, A. G. (1996). Linear Least-Squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57.

Corbett, A. T. et Anderson, J. R. (1994). Knowledge tracing : Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278.

Daubigney, L., Gasic, M., Chandramohan, S., Geist, M., Pietquin, O. et Young, S. (2011). Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system. *In Proceedings of the Twelfth Annual Conference of the International Speech Communication Association (Interspeech 2011)*.

Geist, M. et Pietquin, O. (2010). Kalman Temporal Differences. *Journal of Artificial Intelligence Research (JAIR)*, 39:483–532.

Geist, M. et Pietquin, O. (2011). Managing Uncertainty within the KTD Framework. *In Proceedings of the Workshop on Active Learning and Experimental Design (AL&E collocated with AISTAT 2010)*, Journal of Machine Learning Research Conference and Workshop Proceedings, Sardinia (Italy). 12 pages - to appear.

Iglesias, A., Martinez, P., Aler, R. et Fernandez, F. (2009). Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. *Applied Intelligence*, 31(1):89–106.

Lagoudakis, M. G. et Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

Pietquin, O., Geist, M. et Chandramohan, S. (2011a). Sample Efficient On-line Learning of Optimal Dialogue Policies with Kalman Temporal Differences. *In International Joint Conference on Artificial Intelligence (IJCAI 2011)*, Barcelona, Spain. to appear.

Pietquin, O., Geist, M., Chandramohan, S. et Frezza-Buet, H. (2011b). Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization. *ACM Transactions on Speech and Language Processing*. accepted for publication - 24 pages.

Sutton, R. S. et Barto, A. G. (1998). *Reinforcement Learning : An Introduction*. MIT Press.