Un Algorithme Amélioré d'Itération de la Politique Approchée pour les Processus Décisionnels Semi-Markoviens Généralisés

Emmanuel Rachelson[1], Patrick Fabiani[1], and Frédérick Garcia[2]

[1] ONERA-DCSD
2, avenue Edouard Belin, F-31055 Toulouse, FRANCE
emmanuel.rachelson, patrick.fabiani@onera.fr,

[2] INRA-BIA
Chemin de Borde Rouge, F-31326 Castanet-Tolosan, FRANCE
fgarcia@toulouse.inra.fr

Résumé : La complexité des problèmes de décision dans l'incertain dépendant du temps provient souvent de l'interaction de plusieurs processus concurrents. Les Processus Décisionnels Semi-Markoviens Généralisés (GSMDP) consituent un formalisme efficace et élégant pour représenter à la fois les aspects de concurrence d'événements et d'actions et d'incertitude. Nous proposons un formalisme GSMDP étendu à un temps observable et un espace d'états hybride. Sur cette base, nous introduisons un nouvel algorithme inspiré de l'itération de la politique approchée afin de construire des politiques efficaces. Cet algorithme repose sur une exploration guidée par la simulation et utilise les techniques d'apprentissage à vecteurs supports. Nous illustrons cet algorithme sur un exemple et en proposons une version améliorée qui compense sa principale faiblesse.

1 Introduction

Les Processus Décisionnels Semi-Markoviens Généralisés (GSMDP) Younes & Simmons (2004) constituent un formalisme élégant pour décrire les problèmes de planification présentant les trois aspects d'incertitude, de temps continu et d'activités concurrentes. Imaginons, par exemple, devoir construire la stratégie d'exploitation d'un métro où les seules actions disponibles consistent à introduire ou retirer des rames de la circulation. L'objectif est de maximiser le nombre de passagers transportés tout en minimant le coût de fonctionnement. Les dates d'arrivée des passages, leurs montées et descentes et les retards des trains rendent stochastique le résultat de chaque action et de chaque événement en terme d'état final et de future date de décision. Par ailleurs, le flux des passagers et leurs destinations dépendent fortement de l'heure de la journée. D'autres exemples combinent les difficultés de la planification dans l'incertain, des contraintes temporelles fortes et de la concurrence des processus : la gestion du trafic au sol des avions dans un aéroport, la coordination d'un drone aérien avec la dunamique de son environnement résultant de l'interaction concurrente de processus exogènes (d'autres agents, l'évolution de l'environnement, de la mission, ...).

Nous commençons par présenter le formalisme GSMDP en section 2 et l'augmentons d'une variable temporelle continue afin de prendre en compte la dépendance explicite au temps. Ce formalisme est mis en lumière des approches MDP classiques, des techniques pour gérer les varaibles continues et des représentation des processus stochastiques concurrents. Puis (section 3 nous présentons un algorithme d'apprentissage par renforcement reposant sur la simulation développé pour exploiter la structure et les difficultés inhérentes aux GSMDP. La section 4 évalue cet algorithme en mettant en avant sa principale faiblesse. Nous en introduisons une version améliorée en section 5.

2 Processus Décisionnels Semi-Markoviens Généralisés

2.1 Temps, concurrence et incertitude

Les Processus Décisionnels de Markov (MDP) sont un modèle couramment utilisé pour représenter les problèmes de planification dans l'incertain. Un MDP se décompense en un quadruplet $< S, A, P, r >$, où S et A sont des espaces d'états et d'actions dénombrables, $P(s'|s,a)$ est une fonction de transition probabiliste entre états et $r(s,a)$ est une fonction de récompense pour la transition (s,a) qui permet de construire des critères et d'évaluer actions et politiques. Les solutions aux problèmes MDP sont généralement donnés sous la forme de *politiques Markoviennes* associant une action à entreprendre à tout état de S. On peut introduire des critères d'évaluation de ces politiques, comme le critère γ-pondéré ($V_\gamma^\pi(s) = E\left(\sum_{\delta=0}^{\infty} \gamma^\delta r(s_\delta, \pi(s_\delta))\right)$). Ces critères permettent la définition de la *fonction de valeur* V^π associée à une politique. Un résultat fondamental indique que pour toute politique dépendant de l'historique des états et actions, il existe une politique Markovienne au moins aussi bonne vis-à-vis d'un critère. Par conséquent, l'on peut chercher sans risque une solution optimale aux problèmes MDP dans l'espace restreint des politiques Markoviennes sans craindre de perdre en optimalité. Enfin, les algorithmes classiques comme l'*itération de la valeur* ou l'*itération de la politique* reposent sur le fait que la fonction de valeur V^* de la politique optimale obéit à l'équation d'optimalité de Bellman (équation 1) Bellman (1957).

$$V^*(s) \quad = \quad \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right] \tag{1}$$

Parmi les différentes approches de prise en compte du temps dans le cadre MDP on peut citer les Semi-MDP (Puterman (1994)), les Time-dependent MDP (Boyan & Littman (2001)) et plus récemment les XMDP (Rachelson *et al.* (2008b)). Par ailleurs, le traitement d'un temps continu peut s'inspirer des algorithmes conçus pour les problèmes à espaces d'états continus ou hybrides (eg. Hauskrecht & Kveton (2006), Feng *et al.* (2004)). Une discussion plus détaillée de la relation entre temps et MDP est présentée dans Rachelson *et al.* (2008a). Ces modèles reposent sur l'introduction d'une variable temporelle continue dans le model afin de représenter l'instationnarité dans la fonction de transition. Cependant, la tâche d'écriture des fonctions de transition et de durée pour de tels modèles demande un gros travail d'ingénierie. Par exemple, l'effet de l'action $RemoveTrain$ sur l'état global du problème du métro est le résultat de plusieurs processus concurrents : les arrivées de passagers, le passage des métro, leur mise en service ou leur retrait, etc. ; tous ces processus sont en compétition pour changer l'état du système et synthétiser leurs influences stochastiques concurrentes dans les fonctions de transition et de durée est un travail complexe.

La littérature des processus stochastiques fournit l'outil des Processus Semi-Markoviens Généralisés (GSMP) (Glynn (1989)) afin de représenter le déroulement concurrent de plusieurs processus stochastiques semi-Markoviens concurrents. Younes & Simmons (2004) a introduit les Processus Décisionnels Semi-Markoviens Généralisés (GSMDP) afin de représenter le problème de décision dans l'incertain où les actions sont concurrentes d'événements stochastiques et non-controlables. Un GSMDP décompose le problème en factorisant la fonction de transition globale par les différentes contributions des événements séparés. Cela fait des GSMDP un formalisme élégant et efficace afin de décrire la complexité des problèmes stochastiques dépendant du temps. Nous présentons une définition formelle des GSMDP au paragraphe suivant et nous concentrons sur les GSMDP à temps continu et observable dans le reste de l'article.

2.2 GSMDP

Partons du point de vue des processus stochastiques sans couche décisionnelle. Formellement, un GSMP Glynn (1989) est décrit par un ensemble d'états S et un ensemble d'événements E. A tout instant, le processus est dans un état s et il existe un sous-ensemble E_s d'événements qui sont dits *actifs*. Ces événements représentent les différents processus concurrents qui sont en compétition pour déclencher la prochaine transition. Comme dans le cadre des automates temporisés, à chaque événement actif e, on associe une horloge c_e représentant la durée avant que cet événement ne déclenche une transition. Cette durée serait le temps de séjour dans l'état s si e était l'unique événement actif. L'événement e^* possédant la plus petite horloge c_{e^*} (le premier à déclencher) emmène le processus global dans un nouvel état. La transition est alors décrite par le modèle de transition associé à l'événement en question : l'état suivant est tiré selon la distribution de probabilité $P_{e^*}(s'|s)$. Dans le nouvel état s', les événements n'appartenant pas à $E_{s'}$ sont désactivés (leurs horloges sont mises à $+\infty$) et les horloges des événements de E_{e^*} sont mises à jour de la façon suivante :

– Si $e \in E_s \setminus \{e^*\}$, alors $c_e \leftarrow c_e - c_{e^*}$
– Si $e \notin E_s$ or if $e = e^*$, tirer c_e selon $F_e(\tau|s')$

Le premier événement à déclencher emmène alors le processus dans un nouvel état où ces opérations sont reprises. Il est important de noter que le processus global d'évolution de l'état d'un GSMP n'est pas Markovien : connaître l'état courant n'est pas suffisant pour décrire la distribution de probabilité sur l'état suivant. Nielsen (1998) a montré qu'en introduisant les horloges dans l'espace d'état on récupérait la propriété de Markov pour les GSMP. Nous discutons de ce point au paragraphe suivant. Introduire un choix d'actions dans un GSMP permet de définir un GSMDP (Younes & Simmons (2004)). Dans un GSMDP, on identifie un sous-ensemble A d'événements *controlables* (les actions), les autres étant appelés incontrolables ou exogènes. Les actions peuvent être activées ou désactivées à discrétion et l'ensemble $A_s = A \cap E_s$ n'est jamais vide car il contient toujours au moins l'action "idle" a_∞ (dont l'horloge vaut en permanence $+\infty$) qui laisse le premier événement exogène emmener le processus dans un nouvel état. Tout comme dans le cas des MDP, chercher une stratégie de contrôle implique de définir des récompenses $r(s, e)$ ou $r(s, e, s')$ et d'introduire des politiques et critères.

2.3 Controler un GSMDP

Comme indiqué précédemment, la fonction de transition du processus global ne conserve pas la propriété de Markov à moins d'augmenter l'espace d'états. Dans le cadre MDP classique, cette propriété de Markov nous permet de montrer qu'il existe toujours une politique Markovienne (ne dépendant que de l'état courant) au moins aussi efficace que n'importe quelle politique histoire-dépendante (Puterman (1994)). En revanche, dans le cas des GSMDP, cela n'est plus possible et afin de définir des critères et de trouver des politiques optimales, nous avons en général besoin de permettre à la politique de dépendre de la totalité du *chemin d'exécution* du processus. Un chemin d'exécution Younes & Simmons (2004) de longueur n partant de l'état s_0 et allant dans l'état s_n est une séquence $\sigma = (s_0, t_0, e_0, s_1, \ldots, s_{n-1}, t_{n-1}, e_{n-1}, s_n)$ où t_i désigne le temps de séjour dans l'état s_i avant qu'e_i ne se déclenche. Comme dans Younes & Simmons (2004), nous définissons la valeur γ-pondérée de σ par :

$$V_\gamma^\pi(\sigma) = \sum_{i=0}^{n-1} \gamma^{T_i} \left(\gamma^{t_i} k(s_i, e_i, s_{i+1}) + \int_0^{t_i} \gamma^t c(s_i, e_i) dt \right) \quad (2)$$

où k et c sont les récompense instantanée et taux de récompenses classiques aux $SMDP$, et où $T_i = \sum_{j=0}^{i-1} t_j$. On peut alors définir la valeur espérée de la politique π dans l'état s comme l'espérance sur tous les chemins d'exécution possibles partant de s : $V_\gamma^\pi(s) = E_s^\pi \left[V_\gamma^\pi(\sigma) \right]$.

Cela permet de définir un critère d'évaluation des politiques. L'objectif est alors de trouver des politiques maximisant ce critère. Le principal problème réside dans le fait qu'il est complexe d'effectuer une recherche dans l'espace des politiques histoire-dépendantes. Par ailleurs, la méthode des variables supplémentaires est couramment utilisée afin de transformer des processus non-Markoviens en processus Markoviens. Elle consiste à enrichir l'espace d'état avec exactement assez de variables afin de pouvoir écrire la distribution sur l'état futur à partir de l'état courant. Dans Nielsen (1998), Nilsen augemente l'espace d'états avec les horloges et montre que cette opération suffit à rendre le processus Markovien. Nous noterons cet espace d'état (s, c) par commodité. Il est toutefois irréaliste de définir des politiques sur cet espace augmenté car les horloges contiennent intrinsèquement de l'information concernant le *futur* du processus et sont donc, par nature, inobservables. A partir de cette constatation, plusieurs options sont disponibles :

– Abandonner l'optimalité et de cherche des "bonnes" politiques au sein d'un ensemble restreint de politiques, *eg.* celles définies sur l'état naturel s uniquement.
– Chercher des hypothèses de représentation qui simplifient le GSMDP, rendent l'état naturel s Markovien de nouveau et se ramènent à un MDP.
– Construire des politiques optimales sur l'espace augmenté (s, c) puis inférer une politique sur les variables observables uniquement.
– Chercher un ensemble de variables *observables* permettant de conserver la propriété de Markov, *eg.* l'état naturel s, les durées d'activation τ_i et les états d'activation s_i de chaque e_i (état noté(s, τ, s_a)).

Younes & Simmons (2004) utilise la seconde option ci-dessus. Dans les paragraphes suivants, nous introduisons un nouvel algorithme d'apprentissage de politique fondé sur l'itération de la politique approchée guidée par la simulation, conçu pour gérer les grands espaces d'états des GSMDP à temps continu et observable et adapté aux trois autres options. Une discussion plus détaillée concernant les motivations de cette approche est présentée dans Rachelson *et al.* (2008a).

3 Itération de la politique approchée guidée par la simulation pour les GSMDP

3.1 Itération de la politique approchée

Notre algorithme s'inscrit dans la famille des méthodes d'Itération de la Politique Approchée (API). L'itétation de la politique est un algorithme de résolution des MDP qui explore l'espace des politiques de manière incrémentale en deux temps comme illustré sur la figure 3. Etant donné une politique π_n à l'étape n, la première étape consiste à calculer la valeur de π_n. La seconde étape effectue alors une itération de Bellman dans tous les états, améliorant ainsi la politique. Une propriété importante de l'itération de la politique est son bon comportement "anytime" : à chaque pas n de l'algorithme, la politique π_n est la meilleure politique considérée jusque là. L'itération de la politique converge généralement en moins d'itérations que l'itération de la valeur mais prend plus de temps car la phase d'évaluation est coûteuse en temps de calcul. Afin de traiter des cas réels, on autorise généralement l'évaluation de la politique à être approchée (comme dans Lagoudakis & Parr (2003)). Il y a peu de garanties théoriques sur la convergence et l'optimalité d'API, comme indiqué dans Munos (2003). L'idée de base de notre algorithme est de démarrer avec une politique définie sur tout l'espace d'états et de l'améliorer incrémentalement sur les états *pertinents* pour le problème courant.

$$\pi_{n+1}(s) = \underset{a \in A}{argmax} \left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_n}(s') \right] \tag{3}$$

3.2 L'algorithme

3.2.1 Exploration partielle de l'espace d'états.

Nous nous intéressons à des espaces d'états continus - éventuellement hybrides ; à moins de faire des hypothèses fortes sur la forme des fonctions de transition et de récompense (comme dans Boyan & Littman (2001) ou Younes & Simmons (2004)), il n'est pas possible de calculer exactement la valeur de π_n. A la place, nous échantillonnons l'espace d'états afin d'obtenir une évaluation de V^{pi_n}. Nous simulons N_0 fois la la politique π_n depuis l'état courant du processus et stockons les triplets état-date-récompense $(s_\delta, t_\delta, r_\delta)$. De cette manière, en effectuant la somme cumulée des récompenses depuis un état à la fin de la simulation, un chemin d'exécution fournit une fonction de valeur V_i sur les états traversés (equation 2). L'ensemble de ces fonctions de valeur issues de la simulation forme une base d'apprentissage $\{(s,v)\},\ s \in S, v \in \mathbb{R}$ à partir de laquelle on souhaite construire une fonction de valeur généralisée sur tous les états. Il est important de noter que le *principe de causalité* et l'observabilité du temps impliquent qu'il ne peut y avoir de boucles qu'à temps fixé (boucles instantanées) et que la probabilité d'une infinité de boucles instantanées est nulle. Par conséquent, les simulations s'arrêtent nécessairement en atteignant les états absorbants à $t = horizon$. Lorsque $N_0 \rightarrow \infty$, le sous-espace d'états exploré par les simulations tend vers le sous-espace d'états atteignable depuis s_0 et chaque état est parcouru un nombre de fois correspondant à la probabilité de visite de la chaîne de Markov stationnaire définie par les variables (s_0, c_0, π_n) Nielsen (1998). Par conséquent :

$$\forall s \in Reachable(s_0),\ \lim_{N_0 \rightarrow \infty} \frac{\sum_{i=1}^{N_0} V_i(s)}{N_0} = V^{\pi_n}(s) \tag{4}$$

Nous résumons cette approche en disant que nous laissons la politique courante guider l'échantillonage dans l'espace d'états afin d'atteindre une bonne estimation de la fonction de valeur sur les états atteignables.

3.2.2 Fonction de valeur généralisée.

Il n'est pas possible d'effectuer un nombre infini de simulations et notre base d'apprentissage a nécessairement une taille finie. Nous souhaitons donc généraliser cette information afin d'extrapoler la fonction de valeur sur tous les états. La base d'apprentissage sert alors à entraîner un régresseur qui généralise à l'ensemble de l'espace d'états. Plusieurs approches couplant apprentissage par renforcement et régression ont été développées récemment - certaines utilisant des arbres Ernst *et al.* (2005), des algorithmes évolutionnaires Whiteson & Stone (2006), des méthodes à noyaux Ormoneit & Sen (2002), etc. - mais peu ont

été couplées avec une approche de simulation de la politique. Nous nous sommes concentrés sur la régression à vecteurs supports (SVR) à cause de sa capacité à traiter les espaces de grande dimension auxquels nous sommes confrontés. Les SVR appartiennent à la famille des méthodes à noyaux. L'entraînement d'une SVR correspond à la recherche d'un hyperplan interpolateur dans un espace de dimension plus élevée que l'espace des échantillons. En pratique, les SVR profitent du *kernel trick* afin d'éviter d'exprimer explicitement la projection dans cet espace de grande dimension. Pour plus de détails sur les machines à vecteurs supports, nous renvoyons le lecteur à Vapnik *et al.* (1996). Dans notre cas, nous noterons $\tilde{V}_n(s)$ la fonction de valeur apprise pour la politique π_n.

3.2.3 Instanciation en ligne de la politique.

L'exploration guidée par la simulation et la généralisation sous forme de SVR sont des méthodes destinées à améliorer la phase d'évaluation de l'API. La dernière spécificité de notre algorithme concerne la phase d'optimisation. Pour des espaces d'états de grande taille ou continus, il n'est pas possible d'améliorer la politique partout. De fait, en général le calcul d'une politique complète n'est pas pertinent puisque la majorité des actions calculées ne sera pas utilisée pendant la simulation. Cette idée est à la base de l'itération de la politique *asynchrone*. En revanche, il peut être plus aisé de calculer en ligne la meilleure action à entreprendre dans chaque état visité (par rapport à la fonction de valeur construite à l'itération précédente). Dans un MDP classique, la phase d'optimisation consiste à résoudre l'équation 5 dans chaque état :

$$\pi_{n+1}(s) \leftarrow \arg\max_{a \in A} \tilde{Q}_{n+1}(s,a) \tag{5}$$
$$\text{avec : } \tilde{Q}_{n+1}(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)\tilde{V}_n(s,a)$$

Pour des espaces continus, calculer π_{n+1} nécessite alors de calculer des intégrales sur P et $\tilde{V}_n$. Nous ne souhaitons pas faire d'hypothèses sur la forme de ces fonctions et procéderons de nouveau par échantillonage pour l'évaluation de ces intégrales. Enfin, comme la fonction P elle-même n'est pas forcément connue du décideur et comme nous supposons disposer d'un simulateur du système, nous utiliserons ce simulateur afin d'effectuer cet échantillonage et de construire la fonction $\tilde{Q}_{n+1}(s,a)$ associée à l'exécution de l'action a dans l'état s sachant que l'on obtient $\tilde{V}_n$ après un coup (equation 6). A la fin de la phase d'évaluation, la fonction de valeur généralisée $\tilde{V}_n$ est stockée et la politique n'est pas optimisée. A la place, on réentre dans une phase de simulation où, à chaque état traversé, on calcule la meilleure action à partir des Q-valeurs, ce qui correspond à appliquer la politique π_{n+1} en ne la calculant que dans les états nécessaires. Le gain en terme d'exécution de l'algorithme est facilement illustrable dans le cas des espaces d'états discrets puisqu'alors on remplace $|S|$ evaluations des Q-valeurs pour la mise à jour de la politique par le nombre d'états parcourus pendant la simulation. C'est particulièrement utile puisqu'avec un temps observable, un état n'est jamais traversé deux fois (sauf boucles instantanées). Par conséquent, le calcul de $\tilde{Q}_{n+1}(s,a)$ se fait en simulant N_a fois l'application de l'action a en s et en observant les jeux de $\{(r_i, s'_i)\}$ comme dans l'équation 6. La politique renvoie alors l'action correspondant à la plus grande Q-valeur. Nous désignons cette instanciation en ligne de la politique "itération de la politique approchée en ligne".

$$\tilde{Q}_{n+1}(s,a) = \frac{1}{N_a} \sum_{i=1}^{N_a} \left[r_i + \tilde{V}_n(s'_i) \right] \tag{6}$$

Pour résumer, l'amélioration de la politique est effectuée en ligne ; en chaque état visité nous effectuons une itération de Bellman en utilisant la fonction de valeur généralisée issue de l'étape précédente puis appliquons l'action afin d'aller dans l'état suivant. Cette approche consiste à laisser la politique en cours de construction guider le choix du sous-ensemble d'états sur lequel on améliore les politiques successives.

3.2.4 Résumé de la méthode et implantation

L'algorithme *online Approximate Temporal Policy Iteration* (online-ATPI) est présenté à l'algorithme 1. On peut noter que s y désigne l'espace d'état que l'on considère observable pour la politique. Cela fait d'online-ATPI un algorithme adaptable selon le jeu de variables d'état que l'on décide de prendre (comme en section 2.3). Nous avons implanté une version d'online-ATPI sur l'état naturel du processus. Nos travaux futurs porteront notamment sur l'extension aux états (s, c) et (s, τ, s_a). Dans la version courante

de l'algorithme, la fonction de valeur généralisée est utilisable telle qu'elle alors que dans le cas de l'état (s, c) une étape supplémentaire est nécessaire afin de construire un estimateur des horloges.

Algorithm 1 Online-ATPI

main :
Input : π_0 or $\tilde{V}_0$, s_0
loop
  $TrainingSet \leftarrow \emptyset$
  for $i = 1$ to N_0 **do**
    $\{(s, v)\} \leftarrow$ simulate$(\tilde{V}, s_0)$
    $TrainingSet \leftarrow TrainingSet \cup \{(s, v)\}$
  end for
  $\tilde{V} \leftarrow$ TrainApproximator$(TrainingSet)$
end loop

simulate$(\tilde{V}, s_0)$ **:**
$ExecutionPath \leftarrow \emptyset$
$s \leftarrow s_0$
while horizon not reached **do**
  $action \leftarrow$ ComputePolicy$(s, \tilde{V})$
  $(s', r) \leftarrow$ GSMDPstep$(s, action)$
  $ExecutionPath \leftarrow ExecutionPath \cup (s', r)$
end while
convert execution path to value function $\{(s, v)\}$ (eqn 2)
return $\{(s, v)\}$

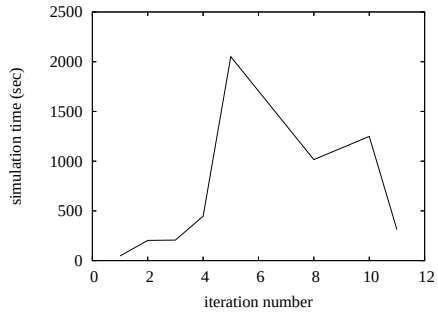
ComputePolicy$(s, \tilde{V})$ **:**
for $a \in A$ **do**
  $\tilde{Q}(s, a) = 0$
  for $j = 1$ to N_a **do**
    $(s', r) \leftarrow$ GSMDPstep(s, a)
    $\tilde{Q}(s, a) \leftarrow \tilde{Q}(s, a) + r + \gamma^{t'-t}\tilde{V}(s')$
  end for
  $\tilde{Q}(s, a) \leftarrow \frac{1}{N_a}\tilde{Q}(s, a)$
end for
$action \leftarrow \arg\max_{a \in A} \tilde{Q}(s, a)$
return $action$

4 Evaluation

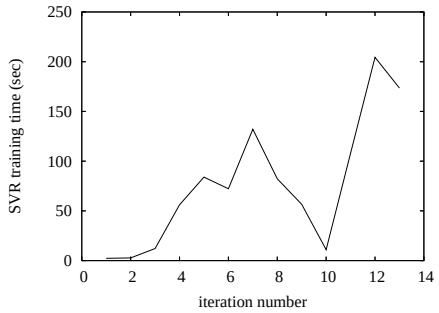
4.1 Le problème de contrôle du métro

Nous avons testé online-ATPI sur un exemple simple du problème de contrôle d'un métro ayant 4 rames et 6 stations. Dans ce problème, le but est d'optimiser le coût d'exploitation du réseau sur une journée. Les événements comme $trainMovement$ et les actions comme $addTrain$ coûtent de l'argent (pour la consommation d'énergie) tandis que chaque passager quittant le métro rapporte le prix d'un ticket. La formalisation du problème fait intervenir les événements concurrents :

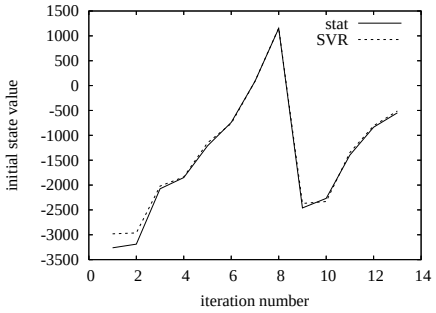
- $passengerArrival_i$: arrivée d'un passager en station i - exogène.
- $trainMovement_j$: mouvement du train numéro j, actif uniquement si le train est en service. Cet événement inclut également la montée et la descente des passagers lors de l'arrivée dans la nouvelle station - exogène.
- $addTrain_j$: met le train j en service à la station 1 - contrôlable.
- $removeTrain_j$: retire la rame j de la ligne, actif uniquement si la rame j est à la station 0 - contrôlable.
- a_∞ : NoOp - contrôlable.



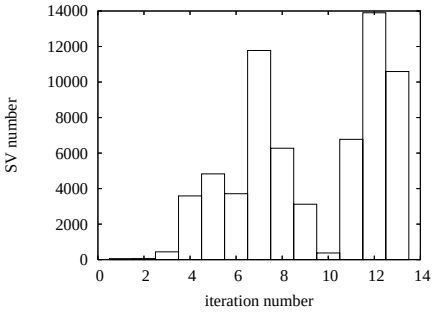
(a) Simulation Time



(b) SVR Training Time



(c) Policy quality



(d) Number of suport vectors

L'espace d'état de ce problème inclut un temps continu, des positions discrètes pour les trains et des variables booléennes redondantes (eg. $NoTrainAtStation_j$). C problème a au final 19 événements concurrents et un espace d'états hybride de dimension 22. En fonction de l'heure de la journée, les fonctions de transition de chaque événement décrivent l'évolution continue des flux de passagers, de leurs destinations et de leurs fréquences d'arrivée.

Cette série d'expériences utilise le moteur de simulation multi-modèles VLE Quesnel *et al.* (2007), s'appuyant sur les principes du formalisme DEVS Zeigler *et al.* (2000). Nous avons développé des extensions pour les GSMP et GSMDP à la plate-forme VLE et les avons utilisées pour les tâches de simulation. Cela a permis de tirer parti de de l'interopérabilité des modèles DEVS. Plus précisément, cela permet la simulation de modèles hétérogènes couplés, *ie.* si un événement ou un système externe interagit avec la simulation de notre GSMDP mais n'est pas écrit comme un GSMP lui-même, alors le couplage DEVS permet la simulation *sure* du système, garantissant les propriétés des simulations de systèmes à événements discrets. Cet aspect n'a pas d'impact direct sur l'algorithme mais étend son champ d'application aux systèmes à événements discrets en général. Nous avons utilisé des valeurs de $N_0 = 20$ simulations par itération de la politique en prenant $N_a = 15$ échantillons par action (Cf. algorithme 1). La SVR est une ϵ-SVR standard, adaptée de la bibliothèque C++ LIBSVM Chang & Lin (2001), utilisant des noyaux Gaussiens avec $\sigma^2 = 20$. A l'issue des N_0 simulations, la base d'apprentissage contenait environ 45000 points. Ces tests ont été effectués sur un processeur simple coeur cadencé à 1,7GHz avec 884Mo de RAM.

4.2 Discussion

Les figures 1(a) à 1(d) présentent les résultats de l'algorithme lorsque ce dernier est initialisé avec une politique par défaut qui fait tourner tous les trains en permanence. Les figures 1(a) et 1(d) illustrent le fait que le nombre de vecteurs supports dans la SVR est le facteur handicapant pour la vitesse de simulation. Cela pourrait être amélioré au prix d'un peu de perte en précision en utilisant d'autres techniques comme les ν-SVR. Toutefois, puisque les N_0 simulations génèrent toujours environ le même nombre d'échantillons d'apprentissage, le nombre de vecteurs supports est borné.

En premier lieu, entre les itérations 1 et 8, la valeur espérée de l'état initial augmente, comme on pouvait l'espérer d'un algorithme type itération de la politique. Cette croissance n'est pas linéaire et dépend de la structure du problème. Si la politique courante emmène la simulation vers des états qui sont "loin" des états déjà explorés (états pour lesquels $\tilde{V}$ peut être une estimation erronée) et qui génèrent des récompenses

très mauvaises, il peut arriver que la valeur de l'état inital chute pour une itération. C'est le contrecoup de l'exploration partielle de l'espace d'états et de l'interpolation : les très bonnes ou très mauvaises régions de l'espace d'états peuvent être découvertes tard dans les itérations (ou ne pas être découvertes du tout).

Deuxièmement, on peut remarquer qu'online-ATPI trouve rapidement (en moins de dix itérations) une politique qui rend le métro rentable. Il existe donc une telle politique malgré les contraintes économiques dures imposées au problème (la meilleure politique "à la main" testée avait une valeur de zéro pour s_0).

Enfin, ce même résultat illustre la principale faiblesse de cette première version de l'algorithme. Après avoir trouvé une bonne politique π_8, la valeur de l'état initial s'effondre à l'itération suivante ce qui est en contradiction à l'objectif de monotonie de l'évolution de la fonction de valeur. Ce phénomène est duû au fait qu'hors de la région explorée par la simulation à la dernière itération (hors de la région où les échantillons sont denses), la SVR surestime la valeur espérée à cause de l'absence d'échantillons dans ces régions (puisque la dernière bonne politique a justement évité ces zones). Par conséquent, lors de l'étape d'amélioration suivante, à chaque fois que le calcul d'une Q-valeur demande la valeur espérée en un état hors de la dernière zone explorée, cette valeur est trop élevée et la SVR n'agit plus comme une heuristique admissible (elle entraîne la politique vers les zones de faibles récompenses). A partir de cette politique biaisée, le processus d'amélioration reprend, comme on le voit aux itérations 9 à 14. Le paragraphe 5 présente une modification de l'algorithme pour résoudre ce problème.

5 Eviter la dégradation de la politique

Afin de conserver l'intérêt d'online-ATPI et la monotonie de l'évolution de la fonction de valeur, nous devons pouvoir garantir que nous ne surestimerons pas la valeur espérée d'un état pour une politique donnée. L'idée principale que nous développons consiste à garder trace des optimisations précédentes de la politique, en particulier sur les états qui ne sont plus parcourus par la simulation, en définissant des régions de confiance dans l'espace d'état relatives aux différentes étapes de l'optimisation.

5.1 Conserver une trace de la politique globale

Afin de ne pas "oublier" les optimisations passées, nous conservons un historique des actions optimisées au cours des itérations. En pratique, nous construisons un classifieur portant sur les paires (s, a) (un régresseur dans le cas des actions continues) qui généralise les actions trouvées à leur voisinage. Cela soulève deux problèmes : d'une part il faut pouvoir décider quand remplacer une action par une autre dans le classifieur, d'autre part il faut préciser la notion de voisinage afin de décider quand utiliser la politique optimisée à la place de la politique initiale. Nous apportons une réponse unique à ces deux problèmes en définissant une fonction indiquant où nous devons nous fier à notre interpolation ou non. En d'autres termes, nous avons besoin de définir une *fonction de confiance* qui indique si le classifieur a reçu suffisamment d'échantillons autour de l'état courant pour apprendre une action pertinente du point de vue statistique et fournir une solution raisonnable. Cette notion de confiance est cruciale pour l'algorithme amélioré d'online-ATPI que nous développerons plus en détail au paragraphe suivant. Finalement la modification de l'algorithme consiste à maintenir une base d'échantillons $actionDB$ composée de $x = (s, a)$, conservant ainsi trace de tous les états explorés. A l'itération n, les N_0 simulations fournissent un nouveau jeu d'actions optimisées sur le sous-espace d'état traversé par la simulation $actionDB'$. Nous retirons d'$actionDB$ tout échantillon x tel que $confidence(actionDB', x) = true$, rajoutons les échantillons d'$actionDB'$ dans $actionDB$ et enfin réentraînons le classifieur $\tilde{\pi}$ sur cette base d'apprentissage mise à jour. En pratique, nous utilisons des classifieurs à vecteurs supports pour les mêmes raisons que nous avons avancées pour l'usage des SVR. Pendant la simulation, nous utilisons la fonction de confiance de nouveau afin de déterminer si une action a été visitée à la dernière itération. Si c'est le cas, nous utilisons $\tilde{\pi}$, sinon π_0.

5.2 Quand faire confiance à l'évaluation par SVR de la fonction de confiance ?

Toutefois, conserver trace des optimizations précédentes ne nous garantit pas de ne pas sur-estimer la valeur espérée dans les régions mal explorées. Losqu'une étape d'amélioration nous entraîne loin des états explorés à la dernière itération, la SVR forunit une valeur qui tend (lorsqu'on s'éloigne des zones explorées) vers la moyenne des valeurs des points explorés. La fonction de confiance que nous avons définie au paragraphe précédent peut être utilisée ici pour définir la confiance dans l'estimation SVR. Nous fixons ainsi

un seuil sur la valeur de confiance en dessous duquel nous considérons la SVR comme non-fiable. Cela ne garantit pas que la SVR soit une heuristique admissible pour la zone de l'état courant mais garantit que sa valeur est proche de la fonction de valeur réelle. Afin de définir cette fonction de confiance sur la valeur, comme précédemment, nous construisons une base d'apprentissage $valueDB$ des échantillons (s, v) de la dernière itération. Ces échantillons correspondent à l'évaluation de la denrière politique calculée. Lorsque la fonction $ComputePolicy$ demande la valeur d'un état s', si $confidence(valueDB, s') = vrai$, alors nous utilisons la valeur de $\tilde{V}$. Sinon, nous effectuaons N_1 simulations à partir de s' en utilisant $\tilde{\pi}$ (ou π_0 si $\tilde{\pi}$ n'est pas fiable) afin d'enrichir la base $valueDB$ et d'adapter $\tilde{V}$. Nous arrêtons ces simulations dès qu'elles réentrent dans la zone de confiance sur la valeur (ce qui permet de réduire le temps de simulation en raccourcissant les trajectoires et le temps d'apprentissage en évitant les échantillons inutiles). Cela fournit une estimation de $V(s')$, met à jour la fonction globale $\tilde{V}$ et étend sa région de confiance associée.

On cherche à conserver la monotonie de l'évolution de la fonction de valeur en cas de régresseur parfait. Supposons que nous sommes en train d'effectuer l'itération $n + 1$. Nous voulons montrer que $V^{\pi_{n+1}}(s) \geq V^{\pi_n}(s)$. Par simplicité, nous négligeons l'erreur d'approximation dans la région de confiance. Si s est à l'intérieur de la région de confiance de $valueDB$, alors le résultat est immédiat. Si s est hors de cette région, alors nous pouvons décomposer tout chemin d'exécution commençant en s en un chemin hors zone e_{out} et une premier état à rencontrer de nouveau une zone de confiance s_{in}. Nous avons alors $V^{\pi_{n+1}}(s) = V(e_{out}) + V(s_{in})$, avec $V(e_{out}) = V^{\pi_n}(e_{out})$ et $V(s_{in}) \geq V^{\pi_n}(s_{in})$. Par conséquent $V^{\pi_{n+1}}(s) \geq V^{\pi_n}(s)$.

En pratique, il existe plusieurs différentes manières de définir la fonction de confiance. Nous nous sommes tournés vers les estimateurs de densité locaux et les SVM mono-classe, cependant d'autres travaux issus de l'apprentissage statistique pourraient s'appliquer ici. Ce sujet concerne encore les recherches futures. Néanmoins, il est rapide de noter qu'une fonction de confiance très sélective engendrera de très grandes bases d'apprentissage qui fourniront à leur tour une estimation très fine de la fonction de valeur et de la politique, tandis que des fonctions de confiance plus lâches réduiront le nombre de simulations à effectuer en déteriorant la qualité de l'estimateur. Ainsi, la fonction de confiance définit la granularité de nos fonctions de valeur et politiques. Enfin, dans le cadre ϵ et ν-SVR, les échantillons pour la régression arrivent incrémentalement ce qui oriente le choix du regrésseur vers des méthodes d'incremental batch SVR learning. Ce dernier point est important pour l'implantation d'*online-ATPI* car les premières expériences avec l'algorithme amélioré ont montré que le réentraînement ralentissait beaucoup le processus global d'apprentissage de la politique optimale. L'algorithme amélioré d'*online-ATPI* est présenté à l'algorithme 2.

6 Conclusion

Nous avons introduit une nouvelle méthode de recherche de politique sur des GSMDP à espace d'états de grande dimension. Notre méthode s'appuie sur l'exploration partielle de l'espace d'états, guidée par la politique courante, et vise à conserver dans la mesure du possible la monotonie d'évolution de la fonction de valeur d'un algorithme d'itération de la politique. Cette démarche permet de définir un algorithme spécifique d'itération de la politique temporelle approchée (ATPI) que nous nommons *online-ATPI*. Dans sa version courante, online-ATPI utilise extensivement les méthodes de SVR et SVC afin de généraliser des échantillons discrets à l'espace d'état continu ou hybride. Ce nouvel algorithme repose de plus sur une notion de granularité dans la recherche de la politque, au travers de l'usage de *fonctions de confiance* définissant la pertinence de l'usage des estimateurs SVM. Le travail en cours et futur s'axera sur l'évaluation plus en détail de l'algorithme amélioré.

Références

BELLMAN R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

BOYAN J. & LITTMAN M. (2001). Exact solutions to time dependent MDPs. *Advances in Neural Information Processing Systems*, **13**, 1026–1032.

CHANG C.-C. & LIN C.-J. (2001). *LIBSVM : a library for support vector machines*. Software available at `http://www.csie.ntu.edu./tw/~cjlin/libsvm`.

ERNST D., GEURTS P. & WEHENKEL L. (2005). Tree-based batch mode reinforcement learning. *JMLR*, **6**, 503–556.

FENG Z., DEARDEN R., MEULEAU N. & WASHINGTON R. (2004). Dynamic programming for structured continuous markov decision problems. In *20th Conference on Uncertainty in AI*, p. 154–161.

GLYNN P. (1989). A GSMP formalism for discrete event systems. *Proc. of the IEEE*, **77**.

HAUSKRECHT M. & KVETON B. (2006). Approximate linear programming for solving hybrid factord MDPs. In *9th Int. Symp. on AI and Math.*

Algorithm 2 Modified online-ATPI

main :
Input : π_0 or $\tilde{V}_0$, s_0
loop
  $valueDB$.reset(), $newActionDB$.reset()
  for $i = 1$ to N_0 **do**
    σ.reset()
    $mainProcess.s \leftarrow s_0$
    while horizon not reached **do**
      $a =$bestAction(s), activateEvent(a)
      $(s', r) \leftarrow mainProcess$.step(), σ.add(s, a, r)
    end while
    $valueDB$.convertExecutionPathToValueFunction(σ)
    $newActionDB$.add(σ)
  end for
  $\tilde{V} \leftarrow$ TrainSVR$(valueDB)$, $\tilde{\pi} \leftarrow$ TrainSVC$(actionDB)$
end loop

bestAction(s) **:**
for $a \in A_s$ **do**
  $\tilde{Q}(s, a) = 0$
  for $j = 1$ to N_a **do**
    $\tilde{Q}(s, a) = \tilde{Q}(s, a) +$ simulateWithStop(s, a)
  end for
  $\tilde{Q}(s, a) = \frac{\tilde{Q}(s,a)}{N_a}$
  return $\arg\max_{a \in A} \tilde{Q}(s, a)$
end for

simulateWithStop(s, a) **:**
activateEvent(a), $(s', r) \leftarrow mainProcess$.clone().step()
$Q \leftarrow r,\ s \leftarrow s'$
while horizon not reached & confidence(s)=false **do**
  $a = \tilde{\pi}(s)$, activateEvent(a)
  $(s', r) \leftarrow mainProcess$.clone().step()
  $Q \leftarrow Q + r$, $s \leftarrow s'$
end while
$Q = Q + \tilde{V}(s)$
$valueDB.update()$, $\tilde{V} \leftarrow$ TrainSVR$(valueDB)$
return Q

LAGOUDAKIS M. & PARR R. (2003). Least-squares policy iteration. *JMLR*, **4**, 1107–1149.

MUNOS R. (2003). Error bounds for approximate policy iteration. In *Int. Conf. on Machine Learning*.

NIELSEN F. (1998). GMSim : a tool for compositionnal GSMP modeling. In *Winter Simulation Conference*.

ORMONEIT D. & SEN S. (2002). Kernel-based reinforcement learning. *Machine Learning*, **49**, 161–178.

PUTERMAN M. (1994). *Markov Decision Processes*. John Wiley & Sons, Inc.

QUESNEL G., DUBOZ R., RAMAT E. & TRAORE M. (2007). VLE - A Multi-Modeling and Simulation Environment. In *Moving Towards the Unified Simulation Approach, Proc. of the 2007 Summer Simulation Conf.*, p. 367–374.

RACHELSON E., FABIANI P., GARCIA F. & QUESNEL G. (2008a). Une approche basée sur la simulation pour l'optimisation des processus décisionnels semi-markoviens généralisés (english version). In *Conf. Francophone sur l'Apprentissage Automatique*.

RACHELSON E., GARCIA F. & FABIANI P. (2008b). Extending the bellman equation for MDP to continuous actions and continuous time in the discounted case. In *10th Int. Symp. on AI and Math.*

VAPNIK V., GOLOWICH S. & SMOLA A. (1996). Support vector method for function approximation, regression estimation and signal processing. *Advances in Neural Information Processing Systems*, **9**, 281–287.

WHITESON S. & STONE P. (2006). Evolutionary function approximation for reinforcement learning. *JMLR*, **7**, 877–917.

YOUNES H. & SIMMONS R. (2004). Solving generalized semi-markov decision processes using continuous phase-type distributions. In *AAAI*.

ZEIGLER B. P., KIM D. & PRAEHOFER H. (2000). *Theory of modeling and simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.