Online Learning of Abstract Stochastic Policies with Monte Carlo

M. L. Koga, V. F. da Silva e A. H. R. Costa

***Abstract*— When an autonomous agent faces many different, but related tasks, obtaining abstract stochastic policies can be helpful because they can be used as transferred knowledge to accelerate learning in a new task throughout transfer learning from previously solved tasks. This paper presents a novel model-free algorithm for online building abstract stochastic policies for Relational Markov Decision Processes, within the reinforcement learning framework. This algorithm, AbsProb-RL, performs a search in the policy-space and uses Monte Carlo techniques to estimate the gradient of the value function of a policy. Results show that AbsProb-RL can effectively find optimal abstract policies, if given enough time.**

***Keywords*— Reinforcement Learning, Transfer learning, Relational Markov Decision Process, Monte Carlo, Abstraction.**

I. INTRODUÇÃO

Cada vez mais nos deparamos com máquinas autônomas na vida cotidiana e os esforços da área de Inteligência Artificial são justamente voltados a construir agentes (máquinas) inteligentes que possam resolver os mais diversos problemas. Nas tarefas de navegação robótica, o agente tem como objetivo achar o menor caminho para alcançar um local específico, sua meta. A capacidade de um agente de aprender e adaptar-se a diferentes situações através da interação com o ambiente é um dos desafios da área, que é atacado pelo aprendizado por reforço (AR) [1]. Nesse tipo de aprendizado, a agente aprendiz deve maximizar sua recompensa (numérica) descobrindo, através de uma estratégia de tentativa-e-erro com repetidas interações com o ambiente, quais são as melhores ações a serem tomadas em cada situação.

Um dos principais problemas do aprendizado por reforço é que o aprendizado pode ser lento e o agente pode demorar a encontrar uma política de ação ótima, pois como o aprendizado é baseado na interação do agente com o ambiente, dependendo da sua dimensão a exploração do ambiente todo pode levar muito tempo. Uma alternativa para se acelerar esse processo é usar o conhecimento adquirido na resolução de um problema anterior (fonte) e usá-lo no aprendizado de um outro problema similar (destino), ao invés de aprender a resolver o novo problema sem qualquer conhecimento prévio. A esse processo se dá o nome de transferência de conhecimento [2].

Para se realizar tal transferência, dois aspectos são importantes de serem considerados: qual o conhecimento a ser transferido e a representação utilizada para se descrever o problema. Quanto ao conhecimento a ser transferido, diversas opções têm sido exploradas, desde a construção de macroações (planos parciais) para transferência e reuso no aprendizado de novas políticas de atuação [3], a transferência baseada na função valor de cada estado [4] até mesmo construção de uma biblioteca de políticas, onde cada uma representa a solução de um problema anterior [5]. A transferência de políticas se mostra vantajosa por ter poucos pré-requisitos para poder ser transferida: apenas um mapeamento entre os estados das tarefas fonte e destino é necessário [6].

Já quanto à representação, em geral os modelos são simples e com pouca semântica associada, sendo muito atrelados ao problema em questão e tornando assim mais difícil transferir o conhecimento representado dessa maneira. Uma abordagem para tornar possível essa transferência é o uso de uma *representação relacional*, mais rica e generalizada [7]. Ela usa objetos e as relações entre eles para descrever o mundo, permitindo a*bstração*. Essa é uma abstração natural proveniente da representação escolhida e deve ser bem definida pelo projetista. Com ela, não só a descrição do problema como também sua solução pode ser generalizada, de modo a facilitar a sua aplicação em outras situações similares.

Nesse cenário, existem diversos trabalhos que exploram a *abstração de políticas* para transferência de conhecimento, que é o foco também deste trabalho. Um dos primeiros trabalhos a utilizar uma representação relacional com o aprendizado por reforço propôs o algoritmo TILDE [8] que, a partir de diversas experiências usando uma política ótima, induz uma árvore de decisão que representa uma política abstrata, usando os predicados para separar (e abstrair) os estados e apresentando a(s) ação(ões) mais frequentes em suas folhas. Matos *et al.* [9] estendeu esse algoritmo, para que fossem geradas políticas não-determinísticas, e usou essas políticas como conhecimento a ser transferido, mostrando que políticas não-determinísticas se comportam melhor na transferência (por serem mais flexíveis, já que o problema destino não é igual ao fonte). Já Beirigo *et al.* [10] explorou uma abordagem diferente para se construir uma política abstrata. Ao invés de induzí-la a partir de uma política ótima, construiu-a buscando direto nos espaços de estados e ações abstratos, usando uma adaptação do clássico algoritmo de AR, o Q-Learning [11]. Já o algoritmo AbsProb-PI [12] também constrói uma política realizando uma busca direta no espaço abstrato, mas no espaço de políticas e não de estados e ações. Resultados mostraram que essa abordagem é mais efetiva no aprendizado. Isso porque como estados abstratos podem agregar vários estados concretos, estados com diferentes valores são misturados. No entanto, o AbsProb-PI é um

M. L. Koga, Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brasil, mlk@usp.br

V. F. da Silva, EACH, Universidade de São Paulo, São Paulo, SP, Brasil, valdinei.freire@usp.br

A. H. R. Costa, Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brasil, anna.reali@poli.usp.br

algoritmo de planejamento, baseado em modelo, e não livre de modelo como são os de aprendizado por reforço.

O problema de navegação robótica pode ser descrito como um problema de decisão sequencial, pois é necessário que o agente tome uma série de decisões até que ele atinja seu objetivo (chegar a um lugar especificado). Além disso, é um problema probabilístico, pois o controle de um robô nunca é perfeito, então após executada uma ação, seu resultado é probabilístico. Esse tipo de problema pode ser modelado como um processo de decisão de Markov relacional (RMDP) [13].

O objetivo deste trabalho é o desenvolvimento do AbsProb-RL, um algoritmo de aprendizado por reforço, livre de modelo, para construção políticas estocásticas abstratas para a solução de RMDPs, políticas que generalizam o conhecimento adquirido. Esse algoritmo faz uma busca no espaço de políticas, ele começa com uma política arbitrária e a cada iteração ela é refinada. Além disso, para ser livre de modelo, ele usa técnicas Monte-Carlo para estimar *online* os valores necessários para o refinamento da política, ou seja, as estimativas são feitas a partir das repetidas *interações* do agente com o ambiente. Esse algoritmo é testado em domínios de robótica móvel. A principal motivação de se encontrar políticas no nível abstrato é possibilitar a transferência do conhecimento entre problemas similares, aproveitando-se da natural abstração da representação relacional, atuando apenas sobre classes de objetos e suas relações.

Este trabalho organiza-se da seguinte maneira: a seção II descreve os principais conceitos abordados e necessários para o entendimento do trabalho, enquanto a seção III descreve com mais detalhes as políticas estocásticas abstratas. A seção IV apresenta o algoritmo AbsProb-RL e a seção V mostra os experimentos realizados. Finalmente, na seção VI estão as conclusões finais.

II. Conceitos Fundamentais

Os conceitos básicos envolvidos nesse trabalho – Processos de Decisão de Markov, sua extensão Relacional e Aprendizado por Reforço – são apresentados a seguir.

A. Processo de decisão de Markov (MDP)

Uma abordagem tradicional de formalização para problemas estocásticos de decisão sequencial consiste no uso de Processos de Decisão de Markov (MDP – Markov Decision Process). Um MDP pode ser definido formalmente pela quádrupla $\langle S, A, T, R, G, b^0 \rangle$ [14], onde:

- S é um conjunto finito de estados do ambiente;
- A é um conjunto finito de ações que o agente pode realizar;
- T: $S \times A \times S \rightarrow [0, 1]$ é a função de transição de estado; $T(s_t, a_t, s_t+1)$ define a probabilidade de realizar a transição do estado s_t para o estado s_{t+1} quando se executa a ação a_t em s_t;
- R: $S \times A \times S \rightarrow \mathbb{R}$ é a função de recompensa, com $r_t = R(s_t, a_t, s_{t+1})$;
- $G \subset S$ é um conjunto de estados-meta;
- $b^0 : S \rightarrow [0, 1]$ é a distribuição de probabilidades do estado inicial, indicando a probabilidade do agente iniciar um episódio em cada estado.

O conjunto de ações aplicáveis no estado $s \in S$ é denotado por $A(s)$. Dessa forma, uma transição do estado $i \in S$ para $j \in S$, decorrente da execução de alguma ação $a \in A(i)$, ocorre com probabilidade $T(i, a, j)$, e então uma recompensa $R(i, a, j)$ é recebida. A função recompensa muitas vezes não depende do estado destino, podendo ser descrita como $R(s_t, a_t)$ ou simplesmente depender apenas do estado no qual o agente se encontra, reduzindo-se para $R(s)$. Estamos interessados em MDPs *episódicos*, i.e., quando o agente alcança um estado $s \in G$, o episódio termina e um novo se inicia em algum estado inicial de acordo com b^0.

Assim, no ciclo percepção-ação, o agente aprendiz observa, a cada passo de iteração, o estado corrente s_t do ambiente e escolhe a ação a_t para realizar. Ao executar esta ação a_t, o agente recebe um reforço r_t, penalização ou recompensa, que indica quão desejável é o estado resultante s_{t+1}. A função de transição deve seguir a condição de Markov, i.e., as probabilidades de transição de um estado ao outro dependem exclusivamente do estado atual, e não do histórico de estados passados pelo agente.

Resolver um problema modelado por um MDP consiste em computar uma política π que especifica quais ações $a \in A$ devem ser executadas quando o agente está no estado $s \in S$. Se a política for determinística, tem-se que $\pi: S \rightarrow A$, ou seja, a política π é determinística se leva cada estado em S a uma única ação em A. Dada a política π e um fator de desconto $\gamma \in [0, 1]$ que desconta recompensas futuras, a função de valor de estado V^π: $S \rightarrow \mathbb{R}$, onde:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} (\gamma^t r_t | \pi, s_0 = s)\right] \forall s \in S \quad (1)$$

representa o valor de estar em um estado seguindo a política π, isto é, representa as recompensas esperadas. A política ótima π^* é a política que satisfaz a condição: $V^{\pi^*} \geq V^{\pi'}$, $\forall s \in S$ e $\forall \pi'$. A função valor ótima é denotada por V^*. Assim, o interesse consiste em computar uma política que melhor aproxime, ou, idealmente, iguale a política ótima π^*. Uma forma possível de resolver um MDP é utilizando um algoritmo de aprendizado por reforço. Para um MDP, sabe-se que uma política determinística é suficiente para otimalidade [14].

A descrição do MDP dada prevê que o agente possui observabilidade total, ou seja, ele sempre sabe em que estado se encontra e portanto, de acordo com a condição de Markov, a política ótima depende apenas do estado atual. Entretanto, existem muitos casos nos quais o ambiente é parcialmente observável, nos quais o agente não tem certeza de qual estado ele se encontra. Podemos definir problemas desse tipo como POMDPs (Processos de Decisão de Markov Parcialmente Observáveis). Os POMDPs são definidos tais quais os MDPs com o acréscimo de uma função O(s,o) que indica a probabilidade de se perceber a observação o num estado s. Assim sendo, num POMDP, ao realizar uma obervação, ao

invés do agente determinar qual o seu estado corrente, ele determina um estado de crença, que é uma distribuição de probabilidades sobre todos os estados possíveis. Na sua interação com o ambiente o agente percebe uma observação o_t, atualiza seu estado de crença indicando a probabilidade dele estar de fato em s_1, s_2, etc., executa então uma ação a_t (a transição $T(s_t,a_t,s_{t+1})$ ocorre de acordo com a função T inerente ao problema, ainda que o agente desconheça s_t e s_{t+1}) e uma uma recompensa r_t e uma nova observação o_{t+1} são percebidas pelo agente. Em POMDPs complexos pode ser difícil de se encontrar políticas ótimas aproximadas, na realidade trata-se de um problema PSPACE-difícil [15].

Pode-se utilizar diversas formas de representação dos estados e ações em um (PO)MDP. A representação relacional, descrita a seguir, é uma delas e é bastante efetiva por possibilitar abstrações do problema.

B. Processo de decisão de Markov Relacional (RMDP)

MDPs são essencialmente proposicionais, uma vez que cada estado tem que ser representado usando uma proposição separada, limitando sua expressividade por não conseguir representar adequadamente a estrutura embutida no problema e, assim, dificultando a generalização de políticas para diversos domínios com propriedades similares. Para aumentar a expressividade da representação proposicional, pode-se usar uma representação relacional, que permite evidenciar relações entre objetos e usar variáveis.

Um vocabulário relacional Σ é um conjunto de constantes e predicados. Na representação relacional, expressões da forma $p(t_1, t_2,..., t_m)$ são átomos, com p sendo um predicado de relação entre os termos t_i. Um termo pode consistir em uma variável (iniciando com letra maiúscula) ou uma constante (iniciando com letra minúscula, como c_1). Um conjunto de átomos forma uma conjunção. Átomos ou conjunções que não possuem variáveis são ditos *concretos.* Toda conjunção é considerada existencialmente quantificada. Uma base de Herbrand HB_Σ é o conjunto de todos os possíveis átomos concretos que podem ser formados com os elementos de Σ.

Com base neste conceito, foi proposto o MDP relacional, RMDP [13,[16]], definido pela tupla $\langle \Sigma, B, T, R, G, b^0 \rangle$, onde:

- $\Sigma = C \cup P_S \cup P_A$ é um conjunto de constantes C que representam os objetos do ambiente; predicados P_S usados para descrever as propriedades e relações entre objetos; e predicados P_A usados para descrever as ações.
- B é um conjunto de sentenças de lógica de primeira-ordem que representam uma base de conhecimento do domínio, usadas para restringir os estados (e ações) possíveis;
- Um conjunto de estados S é definido como o subconjunto do conjunto de todas as possíveis conjunções sobre os átomos da $HB_{P_S \cup C}$ que satisfaz as restrições de *B;*
- O conjunto de ações A um o subconjunto de $HB_{P_A \cup C}$, novamente satisfazendo *B;*
- T, R, G e b^0 são definidos como anteriormente, no MDP.

Conjunções de átomos lógicos representam estados. Assim, um exemplo de estado $s \in S$ seria:

$$s \equiv inRoom(r1) \wedge seeAdjRoom(r2)$$

indicando que o robô está na sala *r1*, a qual é conectada a sala *r2*, pois consegue enxergá-la. Estamos assumindo a condição de mundo fechado, i.e., todos os predicados omitidos da conjunção são falsos.

Da mesma forma, define-se o conceito de estados abstratos como conjunções de átomos lógicos que contêm variáveis no lugar de todas as constantes. Por exemplo, o estado abstrato

$$\sigma \equiv inRoom(R) \wedge seeAdjRoom(R')$$

representa estados nos quais o robô está na sala R, a qual é conectado a uma outra sala R'. Neste caso, o estado s seria uma instância do estado abstrato σ. Uma substituição θ atribui termos às variáveis de forma que s seja um dos possíveis estados concretos descritos por σ. Assim, para o exemplo acima, a instância s seria alcançada com a substituição $\theta = \{R/r1, R'/r2\}$. Um estado é denominado concreto se não contiver variáveis. Note que um estado abstrato pode representar mais de um estado concreto. Outro estado concreto s_2 alcançaria σ com a substituição $\theta_2 = \{R/r3, R'/r4\}$. Ao conjunto de todos os estados concretos possíveis de serem alcançados a partir do estado abstrato σ de S_σ.

Analogamente, ações abstratas podem ser representadas por átomos tendo variáveis como argumentos. Assim, $goToRoom(R)$ é uma ação abstrata que indica que o robô navega para uma sala R, e $goToRoom(r1)$ é uma ação concreta com a substituição $\theta = \{R/r1\}$. A_α é o conjunto de ações concretas abstraidas por uma ação abstrata α.

Dadas essas definições de estados e ações abstratos, define-se como S_{ab} o conjunto de todos os estados abstratos e A_{ab} o conjunto de todas as ações abstratas.

A tarefa de um agente para resolver um RMDP também é encontrar uma política, tal qual no MDP. No RMDP, pode-se também considerar apenas os estados e ações abstratas (S_{ab} e A_{ab}) para a solução. Nesse caso, diz-se que resolvemos um RMDP abstrato, e a sua solução será uma *política abstrata*. Além disso, ao invés de políticas abstratas determinísticas, aqui usamos políticas abstratas estocásticas. O uso de políticas abstratas estocásticas é explicado com mais detalhes na seção III.

C. Aprendizado por Reforço

Aprendizado por reforço é aprender o que fazer – i.e., mapear estados a ações – maximizando uma função reforço numérica. O agente não é ensinado sobre quais ações tomar, como a maioria das formas de aprendizado supervisionado, mas deve descobrir quais ações resultam em maiores somas de recompensas através de tentativas. Nos casos mais interessantes, as ações não afetam apenas a recompensa imediata, mas também os estados futuros e consequentemente as recompensas futuras. Essas duas características: tentativa-e-erro e recompensa futura são as mais importantes do aprendizado por reforço [1].

O aprendizado por reforço é, portanto, uma das maneiras de se encontrar a política ótima de um MDP, sem a necessidade de possuir o modelo completo dele, necessitando

apenas interagir com o ambiente e receber seus reforços mediante as ações tomadas. Por modelo completo, entende-se que o agente conhece *S*, *A*, *T*, *R*, *G* e b^0. No aprendizado por reforço, o agente pode desconhecer *T* e/ou *R*.

No aprendizado por reforço relacional, as técnicas de aprendizado são modificadas para tratar de RMDPs ao invés de MDPs. A principal diferença é que cada estado (inclusive abstrato) não é mais atômico e sim uma estrutura relacional, que permite o uso de variáveis e, assim, um estado abstrato pode representar diversos estados concretos [13].

III. Políticas abstratas estocásticas

Um RMDP é definido, portanto, com descrição relacional de estados e ações. Seja S_{ab} o conjunto de todos os estados abstratos e A_{ab} o conjunto de todas as ações abstratas, define-se uma política abstrata estocástica como $\pi_{ab}: S_{ab} \times A_{ab} \rightarrow [0,1]$, com $p(\alpha_i|\sigma) = \pi(\sigma, \alpha_i)$, $\sigma \in S_{ab}$, $\alpha_i \in A_{ab}^{\sigma}$, $A_{ab}^{\sigma} \subseteq A_{ab}$, e A_{ab}^{σ} é o conjunto de ações abstratas permitidas no estado abstrato σ. Ou seja, numa política abstrata estocástica, cada par estado abstrato-ação abstrata possui uma probabilidade associada e dado um estado abstrato, a soma das probabilidades de cada ação deve ser 1. Vale ressaltar que aqui estamos sempre falando de políticas sem memória, ou seja, somente o estado atual determina a ação a ser tomada, e não uma história.

Construir uma política abstrata para resolver um RMDP abstrato assemelha-se a se resolver um POMDP, pois em ambos os casos queremos encontrar as melhores ações para se executar em cada *agregação* de estados. Um estado abstrato é uma agregação de vários estados concretos, assim como uma observação parcial pode refletir vários estados também.

Levando em conta essa semelhança, políticas estocásticas são mais adequadas que determinísticas. As políticas determinísticas são um caso particular das estocásticas, onde para cada estado uma das ações possui probabilidade 1 e todas as outras 0. O fato é que uma política estocástica pode ser arbitrariamente melhor que uma determinística [17].

Para ilustrar essa afirmação, vamos analisar um simples exemplo. Considere um ambiente discreto com seis células dispostas em linha, como mostra a Fig. 1, com a meta na célula 4. Cada célula representa um estado e o agentepode realizar as seguintes ações: ir para a direita, para a esquerda ou ficar parado. A política determinística ótima para esse problema seria: se o agente estiver nas células 1,2 ou 3, ir para direita; se estiver na célula 4, parar; e se estiver nas células 5 ou 6, ir para a esquerda.

Considere que o agente possui apenas uma observação parcial do ambiente, estando apto apenas a perceber o que está imediatamente ao seu redor. Ou seja, ele tem as seguintes percepções:

- *Percepção na célula 1* = [parede acima, parede a esquerda, porta a direita, parede abaixo, não está na meta];
- *Percepção na célula 2* = [parede acima, porta a esquerda, porta a direita, parede abaixo, não está na meta];
- *Percepção na célula 3* = [parede acima, porta a esquerda, porta a direita, parede abaixo, não está na meta];
- *Percepção na célula 4* = [parede acima, porta a esquerda, porta a direita, parede abaixo, está na meta];
- *Percepção na célula 5* = [parede acima, porta a esquerda, porta a direita, parede abaixo, não está na meta];
- *Percepção na célula 6* = [parede acima, porta a esquerda, parede a a direita, parede abaixo, não está na meta].

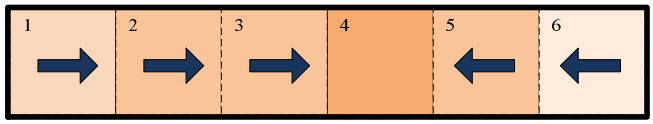

Figura 1 - Um ambiente discreto simples.

Com essas percepções, o agente não consegue distinguir os estados 2,3 e 5. Na realidade, o agente percebe apenas quatro situações distintas:

- *Situação 1* = σ_1 = [parede acima, parede a esquerda, porta a direita, parede abaixo, não está na meta];
- *Situação 2* = σ_2 = [parede acima, porta a esquerda, porta a direita, parede abaixo, não está na meta];
- *Situação 3* = σ_3 = [parede acima, porta a esquerda, porta a direita, parede abaixo, está na meta];
- *Situação 4* = σ_4 = [parede acima, porta a esquerda, parede a direita, parede abaixo, não está na meta].

Essas situações poderiam ser também quatro estados abstratos, dada a abstração adequada. Dadas essas quatro situações, qual política determinística resulveria (mesmo que subotimamente) o problema? Se o agente se encontrar na situação 2, nem ir para a direita nem ir para a esquerda resolve o problema para qualquer estado inicial. Nesse caso, uma política estocástica pode resolver, mesmo que subotimamente. Uma política estocástica abstrata possível seria:

- Se em σ_1, ir para direita com probabilidade 1;
- Se em σ_2, ir para direita com probabilidade 0.66 e ir para esquerda com probabilidade 0.33;
- Se em σ_3, parar com probabilidade 1;
- Se em σ_4, ir para esquerda com probabilidade 1.

Assim, essa política estocástica apresenta um valor médio de cada estado (considerando todos os estados igualmente prováveis) maior que qualquer política determinística.

Definida a política estocástica abstrata, a pergunta que surge é: como aplicar uma política estocástica abstrata π_{ab} a um problema concreto? O método utilizado por este trabalho é descrito a seguir. Quando o agente observa um estado $s \in S$, encontra-se o estado abstrato σ tal que $s \in S_{\sigma}$. Então, a política π_{ab} fornece um conjunto de ações abstratas, com uma probabilidade associada a cada ação. Uma ação abstrata α é escolhida probabilisticamente desse conjunto. Note que uma ação abstrata também pode ser mapeada para uma série de ações concretas. Uma ação concreta a é então escolhida aleatóriamente (com distribuição uniforme) do conjunto A_{α}, numa operação chamada de $concretizacao(\pi_{ab}(\sigma, \alpha))$. Essa

ação a é aplicada, um novo estado s' é observado e o processo se repete.

Um algoritmo para se construir uma política estocástica abstrata é detalhado na próxima seção.

IV. Proposta de Algoritmo

Para se construir uma política abstrata, que atua sobre uma abstração de estados e ações, uma busca direta no espaço de políticas é mais adequada [18]. Um dos mais famosos algoritmos com busca no espaço de políticas é o *Policy Iteration*, algoritmo de programação dinâmica para se resolver MDPs quando se possui o modelo completo do problema. O Policy Iteration possui basicamente dois passos, que se alternam iterativamente: avaliação e melhoramento da política atual. A avaliação da política é o cálculo do valor de cada estado (Eq. 1) e o melhoramento é a escolha da ação que vai maximizar o valor de cada estado (Eq. 2).

$$\pi_{i+1}(s) = \underset{a \in A}{argmax} \left\{ r(s) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi_i}(s') \right\} \forall s \in S \qquad (2)$$

O AbsProb-PI [12] é a adaptação desse algoritmo para o nível abstrato, sendo necessárias, obviamente, algumas mudanças. Ele deve aprender uma política abstrata probabilística π_{ab}. Para o passo de melhoramento da política, foi necessário redefinir a função de transição de estados (Eq. 3), definindo-a em dependência da política em análise:

$$T^{\pi_{ab}}(s,s') = \sum_{\alpha \in A_{ab}} \pi_{ab}(\alpha|\sigma_s) \sum_{a \in A_\alpha} \frac{1}{|A_\alpha|} T(s,a,s') \qquad (3)$$

onde A_{ab} é o conjunto de ações abstratas, A_α é o conjunto de ações concretas abstraídas pela ação abstrata α e σ_s é o estado abstrato que contém o estado concreto s e $\pi_{ab}(\alpha|\sigma_s)$ é o mesmo que $\pi_{ab}(\sigma_s,\alpha)$. Note que a função $T^{\pi_{ab}}$ apresenta as probabilidades de transição de cada estado s para cada estado s' quando se executa a política π_{ab}, i.e., a probabilidade de cada ação α ser executada pela política é multiplicada pela média das probabilidades de transição possiveis com ações subsumidas por α e então as probabilidades de cada ação são somadas.

Dadas essas definições, o algoritmo AbsProb-PI calcula o gradiente da função valor V, que indica a melhor direção (ação abstrata) para se mudar a política [19]. Inicialmente, uma política abstrata πab é arbitrariamente inicializada. Então o algoritmo refina iterativamente essa política, até alcançar algum critério de parada. Uma iteração do algoritmo, cujos cálculos estão na forma matricial e portanto usa representações vetoriais das funções, está detalhada no Algoritmo I, na qual: $\boldsymbol{I}$ é a matriz identidade; $\boldsymbol{T}^{\pi_{ab}}$ é a representação vetorial de $T^{\pi_{ab}}(s,s')$; $\boldsymbol{b^0}$ é a representação vetorial de $b^0(s)$, distribuição inicial de probabilidades de ocorrência de cada estado; $\boldsymbol{R}$ é a representação vetorial da função recompensa $R(s)$; S_{ab} é o conjunto de estados abstratos e $\boldsymbol{T}^{\pi_\alpha}$ é uma matriz de transições definida similarmente a $\boldsymbol{T}^{\pi_{ab}}$, mas cuja política escolhe uma ação abstrata α sempre.

Algoritmo I
Uma iteração do AbsProb-PI

1. Calcule a função valor $\boldsymbol{V}^{\pi_{ab}} = (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi_{ab}})^{-1} \boldsymbol{R}$;
2. Calcule o produto $\boldsymbol{C} = \gamma \boldsymbol{b}^0 (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi_{ab}})^{-1}$;
3. Para cada ação $\alpha \in A_{ab}$, calcule:
$$\Delta^{\alpha,\pi_{ab}} = (\boldsymbol{T}^{\pi_\alpha} - \boldsymbol{T}^{\pi_{ab}}) \boldsymbol{V}^{\pi_{ab}};$$
4. Para cada $\sigma \in S_{ab}$ e $\alpha \in A_{ab}$, calcule:
$$G(\sigma,\alpha) = \sum_{s \in S_\sigma} C(s) \Delta^{\alpha,\pi_{ab}}(s);$$
5. Para cada $\sigma \in S_{ab}$ ache a melhor direção
$$\alpha_\sigma^* = arg\ max_{\alpha \in A_{ab}} G(\sigma,\alpha)$$
6. Escolha um tamanho de passo δ e atualize a política:
$$\pi_{ab}(\sigma,\alpha) \leftarrow \begin{cases} (1-\delta)\pi_{ab}(\sigma,\alpha) + \delta\left(\frac{\varepsilon}{|A_{ab}|}(1-\varepsilon)\right), se\ \alpha \neq \alpha_\sigma^* \\ (1-\delta)\pi_{ab}(\sigma,\alpha) + \delta\frac{\varepsilon}{|A_{ab}|}, se\ \alpha = \alpha_\sigma^* \end{cases}$$

O passo 1 corresponde ao passo de avaliação da política abstrata atual, calculando seu valor. Ele é o equivalente do AbsProb-PI à equação 1 do Policy Iteration. O produto $\boldsymbol{C}$ (passo 2) contém a probabilidade de ocorrência de cada estado, considerando a política π_{ab} e a distribuição inicial, pois a política será avaliada baseada na distribuição inicial dos estados. Em seguida, calcula-se a diferença no valor de cada estado que uma política que usa principalmente a ação α causaria, e isso é feito para cada ação abstrata existente (passo 3). No passo 4, calcula-se o G para cada par estado-ação abstratos, somando o produto de $C(s)$ e $\Delta^{\alpha,\pi_{ab}}(s)$ somente dos estados subsumidos pelo σ da vez.

O valor G, calculado no passo 4, representa justamente o gradiente da função valor, cujo valor máximo indica qual é a ação abstrata cuja probabilidade de ocorrência deve ser aumentada na política para a obtenção de um melhor resultado (passo 5). A política é então atualizada nessa direção, usando um tamanho de passo δ (passo 6), que pode variar a cada iteração.

Esse algoritmo é baseado em modelo, necessitando possuir um modelo completo do problema para funcionar. A contribuição deste trabalho é apresentar o AbsProb-RL, uma versão livre de modelo para o AbsProb-PI. Para deixar esse algoritmo livre de modelo, é necessário avaliar a política (passo 1) e calcular G sem o conhecimento da função de transição T e reforço R completas (passos 2, 3 e 4). Para que isso seja possível, o AbsProb-RL usa técnicas *Monte-Carlo* para estimar os valores desconhecidos, semelhante ao que já foi feito para se encontrar políticas em POMDPs [20], [21].

O uso de Monte-Carlo para se estimar implica que o agente deve realizar diversas interações com o ambiente e o conjunto de experiências resultante dessas interações permite que ele estime os valores necessários. Isso significa que, para cada iteração do AbsProb-PI (passos de avaliação e melhoramento), o agente deve realizar diversos episódios com a política atual, um número suficiente para que ele possa estimar o valor dela para então sim poder realizar um passo de melhoramento.

O AbsProb-RL encontra-se descrito no Algoritmo 2. A ideia básica é a mesma do Policy Iteration: ele começa com uma política π_{ab} (passo 1) e a cada iteração deve avaliá-la e

melhorá-la. Para poder avaliá-la e calcular o gradiente do valor, um número $H(i)$ de episódios deve ser observado. Este número pode ser variável de acordo com a iteração e diversas funções H são testadas na seção V. Repare que, diferentemente do AbsProb-PI, este algoritmo é baseado na interação do agente com o ambiente (passos 5 a 7). Ao final de cada episódio, a estimativa $\hat{G}$ é atualizada (passo 9) com o valor G_e calculado em cada episódio (passo 8), seguindo uma taxa de aprendizado $\mu(i,h)$. Essa taxa pode ser fixa ou variável e diversos valores para a função μ também são explorados na seção V.

Após a execução de $H(i)$ episódios, a política é atualizada (passos 10 e 11) tal qual no AbsProb-PI, porém usando a estimativa $\hat{G}$ do gradiente. E então uma nova iteração se inicia, repetindo-se o processo.

ALGORITMO II
ABSPROB-RL

1. Inicialize arbitrariamente π_{ab}
2. **Para** cada iteração *i*
3. **Para** cada episódio $h \in H(i)$
4. **Para** cada passo *t* do episódio *h*
5. Observe *s* e encontre σ tal que $s \in S_\sigma$
6. Escolha ação $a = concretizacao(\pi_{ab}(\sigma,\alpha))$
7. Execute *a*, receba *r* e observe *s'*
8. G_e = *AtualizaEstimativaG*
9. $\hat{G} = \hat{G} + \mu(i,h)(G_e - \hat{G})$
10. Para cada $\sigma \in S_{ab}$ ache a melhor direção
$$\alpha_\sigma^* = arg\ max_{\alpha \in A_{ab}} \hat{G}(\sigma,\alpha)$$
11. Atualize a política:

$$\pi_{ab}(\sigma,\alpha) \leftarrow \begin{cases} (1-\delta)\pi_{ab}(\sigma,\alpha) + \delta\left(\frac{\varepsilon}{|A_{ab}|}(1-\varepsilon)\right), se\ \alpha \neq \alpha_\sigma^* \\ (1-\delta)\pi_{ab}(\sigma,\alpha) + \delta\frac{\varepsilon}{|A_{ab}|}, se\ \alpha = \alpha_\sigma^* \end{cases}$$

Falta definir o procedimento *AtualizaEstimativaG* (passo 8), que contém justamente como Monte-Carlo é usado para se estimar o valor do gradiente *G*. Para o cálculo de *G*, são necessárias estimativas de $\boldsymbol{V}^{\pi_{ab}}$, $\boldsymbol{T}^{\pi_{ab}}$, $\boldsymbol{T}^{\pi_\alpha}$, para toda ação $\alpha \in A_{ab}$, e $\boldsymbol{C}$. Basicamente, Monte Carlo usa todas as experiências do agente para estimar cada um desses valores. Uma experiência no passo *t* consiste dos valores $\langle s_t, a_t, r_t, s_{t+1}\rangle$ e um conjunto desses valores gera as estimativas necessárias.

$\boldsymbol{V}^{\pi_{ab}}$ é estimado conforme a Eq. 1, só que com tempo finito ao invés de infinito.

$$\boldsymbol{T}^{\pi_{ab}}(s,s') = p(s_{t+1} = s' | s_t = s, \pi_{ab}), \quad (4)$$

ou seja, é a probabilidade de transição entre estados concretos dado que a política π_{ab} é seguida. No caso das experiências do agente, a política π_{ab} é sempre a política seguida. Então essas probabilidades são estimadas através da contagem simples de todas as transições que o agente experimentar. Analogamente, para cada ação abstrata $\alpha \in A_{ab}$, temos

$$\boldsymbol{T}^{\pi_\alpha}(s,s') = p(s_{t+1} = s' | s_t = s, a_t \in A_\alpha), \quad (5)$$

e a cada passo, uma das funções $\boldsymbol{T}^{\pi_\alpha}$ tem sua estimativa atualizada, dependendo da ação que foi realizada naquele passo. E finalmente, o produto $\boldsymbol{C}$, que tem sua definição reproduzida novamente aqui:

$$\boldsymbol{C} = \gamma \boldsymbol{b}^0 (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi_{ab}})^{-1}. \quad (6)$$

Sabemos que:

$$(\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi_{ab}})^{-1} = I + \gamma \boldsymbol{T}^{\pi_{ab}} + \gamma^2 (\boldsymbol{T}^{\pi_{ab}})^2 + \cdots = \sum_{t=0}^{\infty} \gamma^t (\boldsymbol{T}^{\pi_{ab}})^t \quad (7)$$

E também podemos dizer que a distribuição de probabilidades iniciais de cada estado multiplicada pela função de transição *t* vezes representa a probabilidade de ocorrência de cada estado no passo *t:*

$$\boldsymbol{b}^0 (\boldsymbol{T}^{\pi_{ab}})^t (s) = p(s_t = s | \pi_{ab}) \quad (8)$$

Então, podemos definir $C(s)$ como a probabilidade de ocorrência de cada estado *s* dada a política π_{ab}, descontados pelo fator de desconto γ. E essa probabilidade é facilmente estimada através do conjunto de experiências do agente

$$C(s) = \sum_{t=0}^{\infty} \gamma^t p(s_t = s | \pi_{ab}) \quad (9)$$

Assim sendo, o procedimento *AtualizaEstimativaG* atualiza as estimativas de todas as probabilidades descritas, gerando ao final de cada episódio uma estimativa :

$$G_e(\sigma,\alpha) = \sum_{s \in S_\sigma} C(s) \sum_{s' \in S} \left(T^{\pi_\alpha}(s,s') - T^{\pi_{ab}}(s,s')\right) V^{\pi_{ab}}(s') \quad (10)$$

Esse é o algoritmo AbsProb-RL, que usa Monte-Carlo para transformar o AbsProb-PI num algoritmo livre de modelo e com aprendizado *online.* A sua efetividade é testada na seção V, na qual ele é submetido a uma série de experimentos.

V. EXPERIMENTOS

Nesta seção os experimentos realizados são descritos. Todos os experimentos foram feitos num ambiente simulado de navegação robótica.

A. Domínio da navegação robótica

Modelamos o ambiente de navegação robótica dividindo-o em regiões discretas que são descritas unicamente com um conjunto de predicados e um conjunto de objetos. Cada uma dessas regiões representa um estado. Elas são divididas em salas e corredores e os predicados $inRoom(ri)$ e $inCorridor(ci)$ indicam se o agente se encontra numa sala ou corredor, respectivamente. Com um alcance de visão de duas células, o agente pode também ver: salas adjacentes, corredores adjacentes, portas, marcadores (perto ou longe) e espaços vazios; e os predicados $seeAdjRoom(ri)$, $seeAdjCorridor(ci)$, $seeDoor(di)$, $seeMarkerNear(mi)$,

$seeMarkerFar(mi)$ e $seeEmptySpace$ indicam, respectivamente, se o que o agente pode ver numa região. O agente observa salas ou corredores adjacentes somente se está ao lado de uma porta.

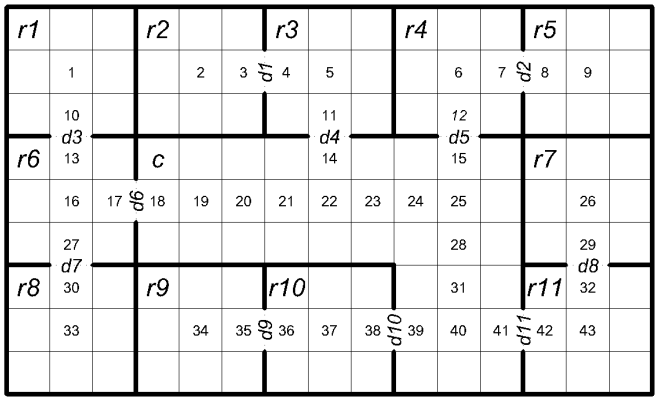

Figura 2 - Ambiente de navegação robótica. Linhas grossas representam paredes e as células numeradas são estados.

Finalmente, o agente também possui a habilidade de sentir a direção e distância para a meta com três predicados: $nearGoal$, $appGoal(X)$, $awayGoal(X)$. O primeiro indica se o agente está a uma distância Manhattan de no máximo 5 da meta; os outros indicam se o objeto X que o agente está vendo está mais perto da meta ($appGoal$) ou mais longe ($awayGoal$) relativamente ao próprio agente. Os objetos podem ser salas(ri), corredores(ci), portas(di) e marcadores(mi), $i \in \mathbb{N}$.

Todos os experimentos usam um mapa com 43 regiões (estados concretos). A Fig. 2 mostra esse mapa, e o estado-meta pode ser escolhido entre qualquer estado dentro de uma sala. Por exemplo, considere a meta no estado 16. O estado 12 seria descrito como:

$$s_{12} = inRoom(r4) \wedge seeAdjCorridor(c) \wedge appGoal(c) \wedge seeEmptySpace.$$

O estado abstrato correspondente,

$$\sigma_1 = inRoom(X) \wedge seeAdjCorridor(Y) \wedge appGoal(Y) \wedge seeEmptySpace,$$

também engloba os estados 11 e 42, i.e., $S_{\sigma_1} = \{11,12,42\}$. Dependendo da posição da meta, o conjunto de predicados descreve diferentemente cada estado, gerando abstrações diferentes. Os experimentos foram rodados considerando as seguintes metas: 5, 16, 34 e 43.

As ações são descritas pelos predicados:

$P_A = \{gotoDoorAppGoal(di), gotoRoomAppGoal(ri), goToCorridorAppGoal(ci), goToMarkerAppGoal(mi), goToEmptyAppGoal, gotoDoorAwayGoal(di), gotoRoomAwayGoal(ri), goToCorridorAwayGoal(ci), goToMarkerAwayGoal(mi), goToEmptyAwayGoal\}$

e seus nomes são autoexplicativos, indicando que o agente deve andar em direção ao objeto especificado.

A distribuição de probabilidades para estado inicial é uniforme para todos os estados.

B. Resultados

Para averiguar o funcionamento do AbsProb-RL, realizamos experimentos no ambiente de navegação robótica, sempre rodando 100000 episódios, sendo que cada episódio possui um número máximo de 500 passos. O tamanho de passo δ usado foi fixo, com $\delta = 0{,}05$. Avaliamos então o valor da política π resultante a cada episódio, sendo o valor da política V^{π} definido como na Eq. 11.

$$V^{\pi} = \sum_{s \in S} b^0(s) \left\{ E\left[\sum_{t=0}^{\infty} (\gamma^t r_t | \pi, s_0 = s) \right] \right\} \quad (11)$$

No primeiro experimento, usamos uma taxa de aprendizado $\mu = 1/h$. Isso significa que a estimativa $\hat{G}$ é a média dos $H(i)$ episódios de cada iteração i do AbsProb-RL e que ela é reinicializada a cada iteração. Isso porque o valor de G depende da política atual, então a cada vez que essa política é mudada (ou seja, a cada iteração) a estimativa deve ser refeita. A função $H(i) = 20i$ indica que a política é atualizada a cada n episódios e esse número aumenta de 20 em 20. Ou seja, a primeira atualização ocorre após 20 episódios, a segunda após mais 40 (novos) episódios, e assim por diante. Esse número é crescente para que o agente possa ter estimativas melhores dos valores com o passar do tempo, pois com o passar do tempo é cada vez mais difícil também encontrar a direção para melhorar a política. Três valores diferentes foram testados para a função H: incremento de 20, 50 e 100.

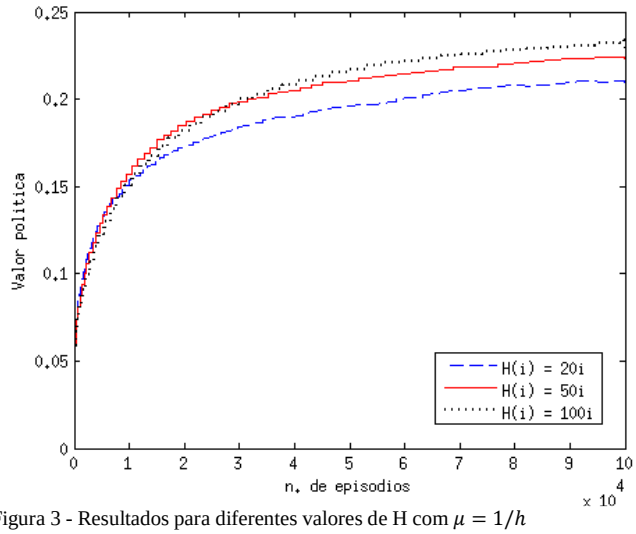

Figura 3 - Resultados para diferentes valores de H com $\mu = 1/h$

A Fig. 3 ilustra os resultados do primeiro experimento. Foram consideradas quatro diferentes metas e a média de 10 execuções para cada uma é mostrada. Percebe-se que valores menores no incremento do número de episódios apresentam um aprendizado ligeiramente mais rápido no início, mas que acaba sendo mais lento após certo ponto. H_3 tem início mais lento que H_1, mas após 10000 episódios o valor de sua política cresce mais rápido. H_2 se encontra entre essas duas curvas. Entretanto, nenhuma das curvas chegou ao valor ótimo de $V^{\pi*} = 0.30$, calculado com o AbsProb-PI após os 100000 episódios.

Para tentarmos melhorar o desempenho do AbsProb-RL, o segundo experimento investigou diferentes formas para a taxa de aprendizado μ. A primeira forma é a que foi usada no experimento anterior, a *média por iteração*. Uma alternativa que mostrou ser bem mais eficiente, é usar uma *média global*, e não por iteração. Nesse caso, $\mu = \frac{1}{ih}$. Isso significa que a estimativa dos valores das políticas das iterações passadas não é esquecida; apesar de cada iteração possuir uma política diferente, as políticas são similares entre si, então uma estimativa já inicializada se mostra ser melhor que uma inicialização em zero.

A Fig. 4 ilustra os resultados deste segundo experimento, desta vez com a média de 10 execuções para apenas uma única meta (estado 16) e com $H(i) = 50i$. A média por iteração apresenta resultado similar ao experimento anterior, não chegando ao valor ótimo. Já a média global teve um desempenho muito melhor, alcançando eventualmente valores muito próximos ao valor ótimo $V^{\pi*} = 0.362$.

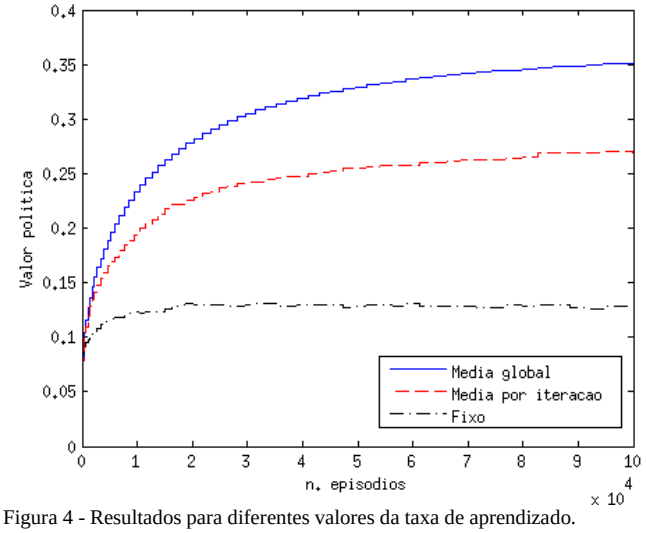

Figura 4 - Resultados para diferentes valores da taxa de aprendizado.

Visto que $\mu = 1/ih$ apresentou melhor resultado, o terceiro experimento fixa essa taxa de aprendizado para novamente avaliar a variação do tamanho da janela de episódios *H*. Além das funções H já definidas anteriormente, como a média global já carrega algum conhecimento prévio, foram testados também valores fixos para H: a política é sempre atualizada a cada 20 ou 50 episódios. As médias de 20 execuções para a tarefa de chegar na meta 16 são mostradas na Fig. 5.

Todas as curvas convergem para o mesmo ponto após os 100000 episódios, mas aqui estamos mostrando apenas até o episódio 50000 para uma melhor visualização. Nota-se que as curvas que possuem *H* crescente sempre crescem também. Isso porque a cada atualização da política, mais episódios são usados para a estimativa do gradiente e portanto ela é mais apurada, garantindo que o passo de melhoramento da política sempre tenha a direção correta (como sempre acontece no algoritmo baseado em modelo).

Já com *H* fixo, a estimativa do gradiente é sempre feita com o mesmo número de episódios, não ficando mais apurada com o tempo. Por essa razão, as curvas com *H* fixo são bastante ruidosas, pois nem sempre a direção correta é a escolhida. Apesar disso, pelo fato de *H* não aumentar, muito mais passos de atualização são feitos e no geral vemos que as curvas apresentaram sempre um valor superior as versões incrementais. Vale notar também que a curva $H(i) = 50$ possui menos quedas que a curva $H(i) = 20$, pois faz uma estimativa melhor do gradiente.

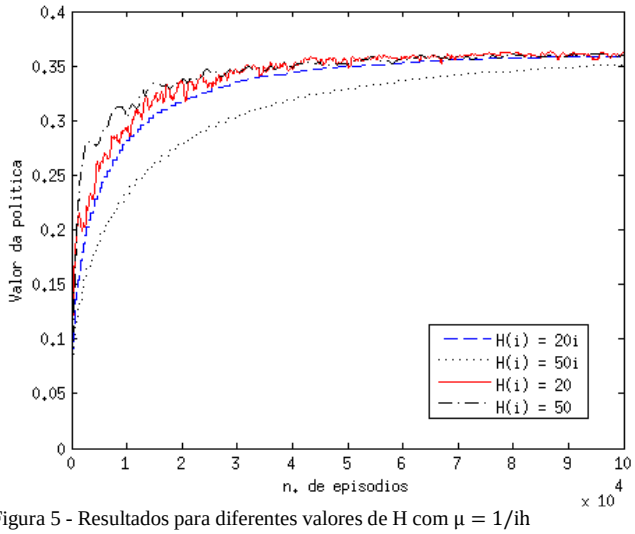

Figura 5 - Resultados para diferentes valores de H com $\mu = 1/ih$

VI. CONCLUSÃO

Diante do exposto, pode-se concluir que é possível construir uma política estocástica abstrata de modo *online*, livre de modelo. Este trabalho apresentou um novo algoritmo, AbsProb-RL, que permite a construção de uma política nesses moldes através de técnicas Monte Carlo para o cálculo do gradiente da função valor de uma política. Resultados mostram que o AbsProb-RL efetivamente alcança resultados similares ao algoritmo equivalente dependente de modelo, ainda que o processo possa ser lento. Conclui-se também que para se estimar os valores do gradiente, é mais efetivo usar valores similares já conhecidos para acelerar essa estimativa.

Uma vantagem de se obter uma política abstrata, já demonstrada em trabalhos anteriores, é acelerar o aprendizado de uma nova tarefa, através da transferência de conhecimento. Trabalhos futuros devem integrar o AbsProb-RL com a transferência de conhecimento, possibilitando assim um aprendizado *online* e simultâneo de ambas políticas concreta e abstrata.

AGRADECIMENTOS

Agradecemos o apoio da FAPESP (proc. n. 12/02190-9, proc. n. 11/19280-8) e da CNPq (proc. n. 311058/2011-6).

REFERÊNCIAS

[1] R.S. Sutton and A.G. Barto.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA. 1998.

[2] M. E. Taylor and P. Stone. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10, p. 1633-1685, 2009.

[3] L. Torrey, J. Shavlik, T. Waker and R. Maclin. *Relational macros for transfer in reinforcement learning*. 2008. In: *Proc. of the 17th Int. Conf. on Inductive Logic Programming (ILP '07)*, 2008.
[4] M. E. Taylor, P. Stone and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, vol. 8, n. 1, p. 2125-2167, 2007.
[5] F. Fernández, F. and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proc. of the 5th international joint conference on Autonomous agents and multiagent systems (AAMAS 2006)*, p.720-727, 2006.
[6] F. Fernández, F. and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, vol. 58, n. 7, p. 866-871. Elsevier, 2010.
[7] E. F. Morales. Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-agent systems (AORC 2003)*, p. 15-26, 2003.
[8] H. Blockeel and L. D. Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, vol. 101, n. 1-2, p. 285-297, 1998.
[9] T. Matos, Y. Bergamo, V. F. da Silva and A. H. R. Costa. Stochastic Abstract Policies for Knowledge Transfer in Robotic Navigation Tasks. In: *MICAI 2011 – Mexican International Conference on Artificial Intelligence*, 2011.
[10] R. L. Beirigo, F. A. Pereira, M. L. Koga, T. Matos, V. F. da Silva and A. H. R. Costa. Avaliação de Políticas Abstratas na Transferência de Conhecimento em Navegação Robótica. In *Proc. of VI Workshop de Tecnologias Adaptativas*, 2012.
[11] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.
[12] V. F. da Silva, F. A. Pereira and A. H. R. Costa. Finding memoryless probabilistic relational policies for inter-task reuse. In *Proc. of 14th Int. Conf. on Information Processing and Management of Uncertainty in Knowlegde-Based Systems (IPMU 2012)*, 2012.
[13] M. van Otterlo. *The Logic of Adaptative Behaviour*, IOS Press, Amsterdam, 2009.
[14] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.
[15] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, Pearson Education, Inc., 2nd. edition, 2003.
[16] K. Kesting, M. van Otterlo, and L. D. Raedt. Bellman goes relational. In *Proc. of 21th Int. Conf. on Machine Learning*, p. 465–472, 2004.
[17] S. P. Singh, T. Jaakkola and M. I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proc. of the eleventh international conference on machine learning (ICML 2011)*, vol. 31, p. 37, 1994.
[18] L. Li, T. J. Walsh and M. L. Littman. Towards a unified theory of state abstraction for MDPs. In: *9th Int. Symp. Artificial Int. and Mathematics*. p. 531-539, 2006.
[19] X. R. Cao. A sensitivity view of markov decision processes and reinforcement learning. In: *Modeling, Control and Optimization of Complex Systems*, 2002.
[20] T. Jaakkola, S. P. Singh and M. I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. *Advances in Neural Information Processing Systems 7: Proceedings of the 1994 Conference*, vol. 7, p. 345. MIT Press, 1995.
[21] J. K. Willians and S. Singh. Experimental results on learning stochastic memoryless policies for partially observable markov decision processes. In *Proc. of the 1998 conference on Advances in neural information processing systems II*, p. 1073-1079, 1999.

Marcelo Li Koga graduou-se em Engenharia Elétrica com ênfase em Computação (2010) e atualmente é estudante de Mestrado em Inteligência Artificial, ambos pela Escola Politécnica da Universidade de São Paulo (EPUSP). Atualmente seus interesses de pesquisa se concentram nas áreas de Aprendizado por Reforço e Transferência de Conhecimento.

Valdinei Freire da Silva é Professor Doutor do curso de Sistemas de Informação da Escola de Artes, Ciências e Humanidades da Universidade de São Paulo (USP, 2011). Possui Doutorado em Engenharia Elétrica (USP em co-tutela com IST-UTL, 2009) e é formado em Engenharia da Computação (USP, 2002). Seus interesses em pesquisa abrangem vários aspectos de Aprendizado de Máquina e Tomadas de Decisões Sequenciais em ambientes probabilísticos.

Anna Helena Reali Costa é Professora Titular do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo (USP, 2011). Possui Doutorado em Engenharia Elétrica (USP, 1994) e Mestrado em Engenharia (FEI, 1989). É formada em Engenharia Elétrica (FEI, 1983). De 1985 a 1988 e de 1991 a 1992 foi pesquisadora na Universidade de Karlsruhe, Alemanha. Em 1998 e 1999 foi pesquisadora visitante na Carnegie Mellon University, nos Estados Unidos. Seus interesses em pesquisa abrangem um grande espectro da Inteligência Artificial, incluindo particularmente o Aprendizado de Máquinas, Robótica Inteligente e Visão Computacional.