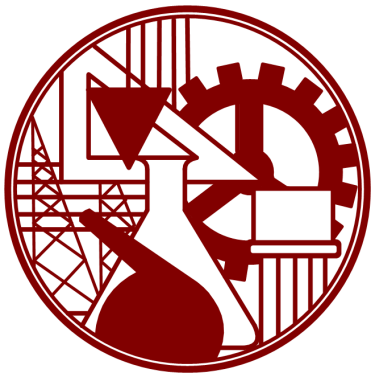
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de
Electrónica e Telecomunicações e de Computadores



Análise Funcional Comparativa de Algoritmos de Aprendizagem por Reforço

João Manuel Dionísio Pessoa
(Bacharel)

Dissertação de natureza científica realizada para obtenção do grau de
Mestre em Engenharia Informática e de Computadores

Júri:

Presidente:

Professor Coordenador Doutor Hélder Pita, ISEL - DEETC

Vogais:

Arguente: Professor Adjunto Doutor Paulo Trigo, ISEL - DEETC

Orientador: Professor Adjunto Doutor Luís Morgado, ISEL - DEETC

Setembro de 2011

Resumo

De entre todos os paradigmas de *aprendizagem* actualmente identificados, a *Aprendizagem por Reforço* revela-se de especial interesse e aplicabilidade nos inúmeros processos que nos rodeiam: desde a solitária *sonda* que explora o planeta mais remoto, passando pelo *programa especialista* que aprende a apoiar a decisão médica pela experiência adquirida, até ao *cão de brincar* que faz as delícias da criança interagindo com ela e adaptando-se aos seus gostos, é todo um novo mundo que nos rodeia e apela crescentemente a que façamos mais e melhor nesta área.

Desde o aparecimento do conceito de *aprendizagem por reforço*, diferentes métodos têm sido propostos para a sua concretização, cada um deles abordando aspectos específicos. Duas vertentes distintas, mas complementares entre si, apresentam-se como características chave do processo de aprendizagem por reforço: a obtenção de experiência através da *exploração* do espaço de estados e o *aproveitamento* do conhecimento obtido através dessa mesma experiência.

Esta dissertação propõe-se seleccionar alguns dos métodos propostos mais promissores de ambas as vertentes de *exploração* e *aproveitamento*, efectuar uma implementação de cada um destes sobre uma plataforma modular que permita a simulação do uso de *agentes inteligentes* e, através da sua aplicação na resolução de diferentes configurações de *ambientes padrão*, gerar estatísticas funcionais que permitam inferir conclusões que retractem entre outros aspectos a sua *eficiência* e *eficácia* comparativas em condições específicas.

Palavras-chave: agentes inteligentes, processos de decisão de Markov, aprendizagem por reforço, política avaliativa, política comportamental

Abstract

Of all the currently identified *learning* paradigms, *Reinforcement Learning* appears to be of special interest and applicability in the numerous processes that surround us: from the lonely *probe* exploring the most remote planet, through the *specialist program* that learns to support medical decision-making through gained experience, to the *toy dog* that delights the child interacting with it and adapting to its liking, it's a whole new world surrounding us beckoning us to do more and better in this area.

Since the *reinforcement learning* concept appeared, different methods have been proposed for its achievement, each of them approaching specific aspects. Two distinct, but complementary, strands appear as key features in the reinforcement learning process: getting experience through the *exploration* of state space and the *exploitation* of the knowledge gained through this same experience.

This thesis proposes to select some of the current most representative methods, in both *exploration* and *exploitation* strands, implement each of these on a modular platform that allows the simulation of *intelligent agents* and, through its application in solving different configurations of *standard environments*, generate functional statistics that allow conclusions to be inferred that reflect, amongst others, their comparative *efficiency* and *effectiveness* in specific conditions.

Keywords: intelligent agents, Markov decision processes, reinforcement learning, evaluation policy, behaviour policy

Agradecimentos

Ao Professor Doutor Luís Morgado, por ter assumido a orientação deste trabalho, pelo estímulo, colaboração e apoio que me proporcionou, e pela disponibilidade que sempre mostrou para analisar alternativas e discutir soluções. O seu profissionalismo e amizade foram os elementos-chave para a realização e bom terminus deste.

Ao ISEL - Instituto Superior de Engenharia de Lisboa, que ocupará sempre um lugar de destaque na minha formação, quer pela excelência de ensino, quer pelo excelente envolvimento profissional e humano de todos os docentes e colegas com quem tive o privilégio de estudar. Um destaque especial para o meu amigo e colega de curso Paulo Faustino, pelas discussões, opiniões e sugestões que comigo partilhou.

À minha família e amigos que sempre me apoiaram nesta etapa da minha vida particularmente trabalhosa. Uma menção especial para os meus pais que sempre fizeram tudo o possível para que eu prosseguisse sempre mais além na minha formação pessoal e académica. Esta é sem dúvida a concretização de mais um sonho que eles tinham para mim, e que de todo o coração lhes dedico.

Finalmente, às minhas adoradas filhas *Ana* e *Telma* que, sendo a razão da minha existência, continuamente me inspiraram a fazer mais e melhor. Que este trabalho possa servir, de alguma forma, de inspiração para elas, demonstrando que *com muito trabalho, persistência e dedicação tudo se consegue*.

Índice Geral

Índice de Figuras

Índice de Listagens

Glossário

AR	Aprendizagem por Reforço (*RL – Reinforcement Learning*)
DT	Diferença Temporal (*TD – Temporal Difference*)
GLIE	*Greedy in the Limit with Infinite Exploration*
HAQL	*Heuristic Accelerated Q-Learning*
HARL	*Heuristic Accelerated Reinforcement Learning*
HMM	*Hidden Markov Model*
HSMQ	*Hierarchical Semi-Markov Q-Learning*
MC	Monte Carlo (*Método*)
PD	Programação Dinâmica
PDM	Processos de Decisão de Markov (*MDP - Markov Decision Process*)
POMDP	*Partially Observable Markov Decision Process*
PSA	Plataforma de Simulação de Agentes
SARSA	*State-Action-Reward-State-Action*
USAF	*United States Air Force*
VDBE	*Value-Difference Based Exploration*

We cannot teach people everything;
We can only help them discover it within themselves

Galileu Galilei

1. Introdução

Um processo de *aprendizagem* pode ser definido de forma sintética como o modo como são adquiridos novos conhecimentos, se desenvolvem competências e se mudam comportamentos. De entre todos os paradigmas de aprendizagem actualmente identificados, a *aprendizagem por reforço* (AR) revela-se de especial interesse e aplicabilidade nos inúmeros processos que nos rodeiam. Esta traduz-se na prática por uma permanente interacção com o ambiente através de políticas de *tentativa e erro* que, maximizando um *sinal de reforço*, permite atingir um conjunto de objectivos pretendidos.

O processo de *aprendizagem por reforço* pode, assim, ser caracterizado como um processo de decisão *sequencial*, no qual sequências de transições entre *estados* e a observação dos sinais de *reforço* associados a cada transição determinam o resultado final. Os *Processos de Decisão de Markov* (MDP) apresentam-se como um suporte particularmente adequado para a representação deste tipo de processos. Todos os processos passíveis de serem modelados através de MDP designam-se de *Markovianos* e obedecem à *Propriedade de Markov*, segundo a qual o efeito de uma acção num estado depende apenas da própria acção e do estado actual do sistema, existindo uma completa independência entre a ocorrência de qualquer acção futura e os estados anteriormente percorridos.

Para um determinado tipo de problema que permita ser modelado por um processo *Markoviano* podem sempre existir diversas *Políticas Comportamentais*, isto é, conjuntos de acções decididas e executadas ao longo do tempo. No entanto, o objectivo da aprendizagem neste contexto será sempre o de encontrar uma *Política Óptima*, ou seja aquela que maximize o valor do *sinal de reforço* acumulado ao longo do tempo, este designado por *retorno*.

O valor do retorno pode reflectir, conforme a estratégia utilizada pelo algoritmo associado a este, significados distintos como seja uma simples soma dos reforços (recompensa pela acção) num determinado espaço temporal (*horizonte finito*), a média aritmética dos reforços (*retorno médio*) ou a soma dos reforços ponderados pela distância temporal entre estes (*horizonte infinito*).

Ainda na definição do processo de aprendizagem, devem-se ter em consideração duas vertentes distintas mas complementares entre si: a obtenção de experiência através da *Exploração* do espaço de estados e o *Aproveitamento* do conhecimento obtido através dessa mesma experiência de forma a obter o máximo de recompensa.

A utilização de idêntica política para ambas as vertentes define uma aprendizagem *On-Policy*, e a clara distinção entre a exploração através de uma política de *Avaliação* e o aproveitamento através de uma política *Comportamental* define uma aprendizagem *Off-Policy*.

Nesta dissertação iremos testar a *eficiência* e *eficácia* comparativas entre alguns dos diversos algoritmos propostos, aplicando-os como *políticas Avaliativas* e *Comportamentais* na resolução de problemas de AR.

1.1. Motivação

Ao abordar a resolução de um qualquer problema de decisão sequencial através dos algoritmos de aprendizagem inicialmente propostos *Q-Learning* (Watkins, 1989) e *Sarsa* (Rummery, et al., 1994), é possível constatar o surgimento de algumas dificuldades na compatibilização das respostas destes com uma solução adequada aos problemas reais, sobretudo devido ao esforço obtido (traduzido em *tempo* e/ou *recursos* gastos) apresentar-se frequentemente demasiado elevado para a resolução de muitos problemas relevantes.

No entanto, os últimos anos foram profícuos no aparecimento de diversas ideias para optimização dos algoritmos de aprendizagem existentes, ou mesmo de outras abordagens distintas, todas na procura da optimização da estratégia de aprendizagem visando a obtenção da política óptima através do menor esforço, ou seja, da forma mais rápida e económica possível em termos computacionais.

Perante o atrás exposto, faz sentido o crescente e assinalável esforço, por parte da comunidade académica, na tentativa de encontrar novos formas e técnicas que permitam normalizar os diversos métodos de teste e comparação qualitativa e quantitativa entre os diversos algoritmos (podendo-se tomar como referências (Abe, et al., 2002) ou (Dutech, et al., 2005) ou ainda (Neruda, et al., 2009)), no sentido de aferir as suas características em termos comparativos. Esta é a motivação principal desta dissertação.

1.2. Objectivos

Pelo atrás exposto, esta dissertação propõe-se seleccionar alguns dos algoritmos mais representativos, propostos ao longo das últimas duas décadas nas vertentes *exploratória* e de *aprendizagem*, e com base nestes:

- Realizar uma síntese teórica dos métodos seleccionados;
- Efectuar uma implementação de cada um sobre uma plataforma modular;

- Aplicar cada implementação na resolução de um ambiente *padrão*, comum a todos os algoritmos;
- Gerar estatísticas funcionais que permitam inferir conclusões que retractem, entre outros, a *eficiência* e *eficácia* comparativas entre os diversos algoritmos em condições específicas.

1.3. Organização da Dissertação

Para além deste capítulo de introdução, esta dissertação encontra-se organizada em mais cinco capítulos, num total de seis:

Capítulo 2: Aprendizagem por Reforço
Capítulo de enquadramento, onde são analisados os conceitos mais relevantes envolvidos na problemática da Aprendizagem por Reforço.

Capítulo 3: Políticas em Aprendizagem por Reforço
Selecção de Políticas *Avaliativas* ou *Comportamentais* classificadas como relevantes no âmbito desta temática, sua descrição, constituição e comportamento expectável.

Capítulo 4: Concretização experimental
Neste capítulo são descritas as opções tomadas, e as razões subjacentes a cada escolha, para todos os componentes envolvidos, como seja a plataforma de efectivação dos testes (PSA), os ambientes seleccionados, as configurações e métricas propostas, e ainda as configurações de Políticas que serão testadas.

Capítulo 5: Ensaios realizados
Capítulo onde são apresentados os resultados de desempenhos individuais e análise comparativa das configurações de *políticas* seleccionadas, e sua interpretação.

Capítulo 6: Conclusões
Conclusões inferidas dos resultados apresentados e vias de investigação passíveis de exploração em trabalhos futuros.

1.4. Convenções de escrita

No sentido de facilitar a leitura, ao longo da dissertação são utilizadas as seguintes convenções de escrita:

- As traduções de termos ou expressões originalmente em língua inglesa, na sua primeira ocorrência no texto, surgem seguidas da designação original, entre parêntesis e entre aspas;

- Optou-se por não traduzir alguns termos de língua inglesa de utilização generalizada, como "hardware" ou "software";
- Em relação às siglas utilizadas, dado o seu uso generalizado na comunidade técnico-científica, são mantidas tal como aparecem no original;
- O grafismo em itálico é utilizado para destacar termos individuais ou conjuntos de palavras no texto, como é o caso das designações de conceitos, que surgem com grafismo em itálico na primeira ocorrência que seja relevante para a sua descrição;
- Surgem igualmente com grafismo em itálico as expressões latinas como *a priori*.

2. Aprendizagem por Reforço

Segundo (Sutton, et al., 1998), a *Aprendizagem por Reforço* (AR) é um formalismo da *Inteligência Artificial* que permite a um *agente computacional* aprender a partir da interacção com o ambiente no qual se encontra inserido. Por sua vez, esta aprendizagem de uma *Política óptima* (ou quase óptima) para um determinado ambiente decorre do uso das *recompensas observadas* (Russell, et al., 2003).

A AR é sobretudo indicada quando se deseja obter, perante um determinado ambiente, um comportamento (*Política*) tendencionalmente óptimo do agente para alcançar o objectivo, sem prévio conhecimento da função que modela este mesmo comportamento. Para tal, o Agente interage directamente com o ambiente de forma a obter *informações*, que serão por sua vez processadas através de um *algoritmo* apropriado, a fim de executar as *acções* que levam o Agente a atingir os seus objectivos.

2.1. Características da Aprendizagem por Reforço

A *Aprendizagem por Reforço* apresenta algumas características intrínsecas que no seu conjunto se assumem como responsáveis pela sua diferenciação perante outras abordagens de aprendizagem. Podem-se destacar:

Orientação ao Objectivo: Existe um Agente que interage com o ambiente desconhecido tentando alcançar um objectivo. Consideram-se ambientes que exclusivamente cedem respostas perante acções efectuadas, não sendo necessário o conhecimento prévio de detalhes da modelação destes.

Aprendizagem por Interacção: Um Agente AR age sobre o ambiente e aguarda o retorno de um *valor de reforço*, que o ambiente devolve em resposta à *acção* realizada. O Agente irá aprender assimilando o valor de reforço obtido, memorizando a informação para tomadas de decisão posteriores

Retorno diferido: As acções tomadas, embora sejam o produto de uma decisão local no ambiente, podem ter um efeito diferido no tempo devendo levar à maximização do retorno total, isto é, a qualidade das acções tomadas é avaliada pelas soluções encontradas a longo prazo.

Exploração *versus* Aproveitamento: Este é um dilema que consiste em decidir quando o Agente deve aprender (*Exploração*) sobre o ambiente, ou quando deve utilizar a informação (*Aproveitamento*) já obtida até ao momento, para evoluir atrás do seu objectivo. Esta decisão consiste fundamentalmente numa escolha entre agir com base na informação disponível de momento, ou agir com a finalidade de obter novas informações sobre o ambiente, que permitam no futuro melhores níveis de desempenho. Agir para obter informação pode aumentar o desempenho de longo prazo, embora também possa provocar uma diminuição do desempenho a curto prazo. Assim, o Agente deve aprender quais as acções que maximizam os valores dos ganhos obtidos no tempo, mas simultaneamente deve agir de forma a atingir esta maximização, explorando acções ainda não executadas e/ou regiões do espaço de estados pouco visitadas.

2.2. Interacção entre Agente e Ambiente

Um Sistema de AR é essencialmente tipificado através de um Agente interagindo com o Ambiente, via *percepção* e *acção*, ou seja, após o Agente ter percepcionado (pelo menos parcialmente) a situação encontrada no ambiente, e com base nessas informações, selecciona uma acção a ser realizada. A acção seleccionada muda de alguma forma o ambiente, afectando o estado na tentativa de alcançar o seu objectivo. Estas mudanças são por sua vez comunicadas ao Agente através de um *sinal de reforço* e do próximo *estado*. O diagrama seguinte demonstra este ciclo:

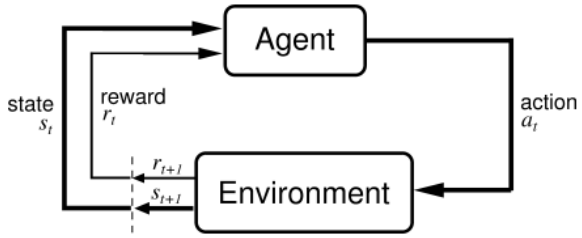


Figura 1 - *Interacção Agente-Ambiente em AR (Sutton, et al., 1998)*

Da análise da figura resulta de imediato de que a percepção do ambiente por parte do agente encontra-se em permanente evolução, através do *feedback* que o próprio ambiente debita perante cada acção executada sobre este. Tal implica que ao agente seja solicitada a contínua e frequente monitorização do ambiente e que reaja apropriadamente a este, através do conhecimento adquirido, de forma a conseguir o seu objectivo.

Num sistema de AR, o *ambiente* é caracterizado através de:

- Conjunto $\boldsymbol{S}$ de *estados* discretos do Agente, constituído pelo conjunto de combinações de variáveis de estado conhecidas deste;
- Conjunto $\boldsymbol{A}$ de *acções* discretas, passíveis serem seleccionadas pelo agente;
- Valor das *transições* de estado, que é apresentado ao agente através de um *sinal de reforço* $\boldsymbol{r}$

2.3. Factores variáveis na Aprendizagem por Reforço

Temos então um *agente* na senda constante do seu *objectivo*… mas que objectivo?

O objectivo final em AR será sempre o de encontrar uma *política* óptima $\boldsymbol{\pi}^*$, ou seja o mapeamento de *estados* em *acções* que maximize os sinais de *reforço* acumulados ao longo do tempo. Assim, o agente procurará escolher as sequências de acções que tendam a aumentar a soma dos valores de reforço.

No tipo de cenários apropriados à abordagem via AR, podem ser isolados alguns factores essenciais à modelação do problema de seguida apresentados.

2.3.1. Ambiente

Todos os sistemas de AR assimilam um mapeamento de situações em *acções* sobre um ambiente dinâmico. Assim, o ambiente no qual o agente se encontra inserido deve ser pelo menos *parcialmente observável*. No entanto, nos últimos anos a comunidade académica abordou, de uma forma crescente e intensa, diferentes generalizações associadas a este factor. A título informativo enunciamos duas das mais relevantes:

- **POMDP**: Um *Processo de Decisão de Markov Parcialmente Observável*: Uma generalização de um MDP onde o estado actual do sistema não é conhecido na sua totalidade.

- **HMM:** Um *Modelo Oculto (ou Escondido) de Markov*: Um modelo estatístico onde o sistema modelado é assumido como um processo de *Markov* com parâmetros desconhecidos.

2.3.2. Função de reforço

A função que determina o valor a ser devolvido pelo ambiente ao agente pode assumir diferentes graus de configuração e complexidade, adequados a cada tipo de problema, que intrinsecamente expressam o objectivo que o agente AR deve alcançar.

Dependendo da tipificação do problema modelado, também esta função pode ser adaptada às necessidades impostas fazendo variar algumas das suas características, como por exemplo:

- **Minimização de reforços**: Em situações de recursos limitados, onde o agente deva aprender a conservar estes em simultâneo com a perseguição do objectivo, porventura a maximização permanente da função de reforço pode não ser sempre possível.
- **Reforço no estado final:** Em tarefas episódicas, onde a recompensa (positiva) ou a penalidade (negativa) somente são atribuídas no estado final (terminal), o agente ao maximizar o reforço irá aprender quais os estados bons e quais aqueles que devem ser evitados.
- **Tempo mínimo**: Ao serem valorados negativamente todos os reforços associados às transições que não levem a um estado terminal (e valorando positivamente o atingir deste), o agente ao maximizar os valores de reforço aprende a seleccionar as acções que minimizem o tempo que leva a alcançar o objectivo.

2.3.3. Reforço versus Retorno/Utilidade

Tipicamente, o *reforço* assume a forma de um sinal escalar ($\boldsymbol{r}_{t+1}$) devolvido pelo ambiente ao agente, assim que uma *acção* tenha sido efectuada e uma *transição de estado* ($\boldsymbol{s}_t \rightarrow \boldsymbol{s}_{t+1}$) tenha ocorrido.

De uma forma geral, o *agente* deve maximizar a quantidade total de *reforços* recebidos, o que nem sempre significa maximizar o *reforço imediato*, mas sim o *reforço acumulado* ao longo da experiência do agente, também designado por **retorno** ou **utilidade**.

Uma *política* óptima $\boldsymbol{\pi}^*$ é aquela que produz o maior *retorno*. O *retorno* é uma medida da qualidade da *política* em uso e pode ser definido como uma função da sequência de valores de *reforço* obtidos até um tempo final $\boldsymbol{T}$, assumindo a forma de um simples somatório no caso mais básico:

$$R_T = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T \tag{2.1}$$

No entanto, existem aplicações onde esta formulação do *retorno* encontra dificuldades em ser aplicada, como sejam problemas onde se apresentem *espaços de estados muito grandes* (ou mesmo virtualmente sem limites, como sejam tarefas de controle contínuo), onde o cálculo do retorno tenderá para ∞**.** Assim, foi introduzido neste cálculo uma *taxa de amortização* (ou **factor de desconto**) $\boldsymbol{\gamma}$, o qual determina o grau de influência que têm os valores futuros sobre o retorno:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

assumindo a *taxa de amortização* $\boldsymbol{\gamma}$ valores no intervalo [0,1]**.** Desta forma, a partir de um *somatório de termos potencialmente infinitos* é produzido um *resultado finito.*

Da análise da fórmula apresentada, decorrem algumas conclusões acerca do comportamento dinâmico do valor de *retorno* em função da *taxa de amortização*:

- $\boldsymbol{\gamma = 1}$: Apresenta-se como um caso especial denominado de *reforços* (*recompensas*) *aditivos*. A visão do reforço abrange todos os estados futuros, dando a mesma importância a ganhos presentes ou futuros;
- $\boldsymbol{\gamma \approx 1}$: Apresenta-se significativa a relevância dos *reforços* atribuídos num futuro distante;
- $\boldsymbol{\gamma \to 0}$: Com o mesmo ritmo com que a taxa de amortização diminui e se aproxima de zero, aumenta também o ignorar dos *reforços* atribuídos num futuro distante. Desta forma, o *agente* tende a ficar com uma visão "míope", maximizando apenas os reforços imediatos.

2.3.4. Função-valor

Uma *função-valor* consiste no mapeamento do *estado*, ou par *estado-acção*, num valor que é obtido a partir do *reforço* actual e dos reforços futuros.

A função-valor pode assumir dois formatos específicos:

- Função *valor-estado* $\boldsymbol{V(s)}$ - Tem somente em consideração o *estado* $\boldsymbol{s}$;
- Função *valor-acção* $\boldsymbol{Q(s,a)}$ - Considera o par *estado-acção* $\boldsymbol{(s,a)}$.

Esta função assume especial relevância nos métodos de *Diferença Temporal* (DT) ao permitir a abstracção do modelo PDM do ambiente, não exigindo o conhecimento prévio do *modelo de transição* deste (distribuições de probabilidades das transições entre estados).

2.3.5. Política

Num cenário de AR, uma *política* $\boldsymbol{\pi}$ representa um mapeamento de estados $\boldsymbol{s}$ e acções $\boldsymbol{a}$ num valor $\boldsymbol{\pi(s,a)}$, o qual corresponde à probabilidade do agente seleccionar a respectiva acção $\boldsymbol{a \in A(S)}$ quando se encontrar no estado $\boldsymbol{s \in S}$.

Assim, uma política representa o *comportamento que o sistema AR assume para alcançar o objectivo.*

O *processo de aprendizagem* num sistema de AR pode ser expresso em termos de convergência para uma *política óptima* $\boldsymbol{\pi}^*$, o que por sua vez conduz à solução do problema de forma óptima. Uma mudança de política implica uma possível alteração às probabilidades de selecção de acções, e consequentemente o surgir de variações no comportamento do sistema.

O processo de aprendizagem pode ser decomposto em dois tipos base:

- **On-Policy** - Quando é usada a mesma *política*, tanto para o processo de *exploração* como para o de *aproveitamento* do conhecimento já obtido;
- **Off-Policy** - Quando são usadas *políticas* distintas para os processos de exploração (*avaliativa*) e de aproveitamento (*comportamental*).

2.4. Definições formais

Alguns conceitos formais devem ser conhecidos de forma a facilitar a modelação de um sistema de AR. Apresentamos de seguida uma breve descrição destes.

2.4.1. Propriedade de Markov

Se a probabilidade de transição de um estado **s** para um outro **s'** depender exclusivamente do estado **s** e da acção **a** seleccionada em **s**, então o estado corrente fornece informação suficiente para que o sistema de aprendizagem decida que acção deve ser tomada. Um sistema que possua esta característica satisfaz a *propriedade de Markov* (Bellman, 1957) .

Assim, considerando uma evolução discreta para $\boldsymbol{t} = 1, 2, 3, ...$, se a resposta do ambiente em $\boldsymbol{t+1}$ depender apenas do *estado* e *acção* seleccionada em $\boldsymbol{t}$, então a *dinâmica do ambiente* satisfaz a *propriedade de Markov*, sendo a probabilidade de transição para o estado $\boldsymbol{s'}$ dada pela expressão:

$$P^{a}_{s,s'} = P\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{2.3}$$

Onde $\boldsymbol{P}$ designa a probabilidade de uma acontecimento, e $\boldsymbol{P^{a}_{s,s'}}$ designa a *probabilidade de transição* para o estado $\boldsymbol{s'}$ após a selecção da acção $\boldsymbol{a}$ no estado $\boldsymbol{s}$. Simultaneamente, a *probabilidade de transição* satisfaz as seguintes condições:

1. $P^{a}_{s,s'} \geq 0, \forall s, s' \in S, \forall a \in A(s);$ (2.4)

2. $\sum_{s' \in S} P^{a}_{s,s'} = 1, \forall s \in S, \forall a \in A(s)$ (2.5)

Esta propriedade revela-se de fundamental importância na AR, uma vez que valores e decisões apenas dependem do estado actual. Assim, fica aberta a possibilidade de aplicação de métodos de *soluções incrementais*, onde podem ser obtidas soluções a partir do estado actual e para cada um dos estados futuros.

2.4.2. Processo de decisão de Markov (PDM)

Um *processo de decisão de Markov* é definido como (Bellman, 1957) :

- Um conjunto de estados $\boldsymbol{S}$;
- Um conjunto de acções $\boldsymbol{A(s)}$ possíveis em cada estado $\boldsymbol{s} \in \boldsymbol{S}$;
- Um conjunto de transições entre estados associadas às acções;
- Um conjunto de probabilidades $\boldsymbol{P}$ sobre o conjunto $\boldsymbol{S}$ que representa uma modelação das transições entre estados.

Dado um par estado-acção $(\boldsymbol{s}, \boldsymbol{a})$, a probabilidade de transição do estado $\boldsymbol{s}$ para o estado $\boldsymbol{s'}$ quando a acção $\boldsymbol{a}$ for seleccionada cumpre a *propriedade de Markov*. De forma análoga, dado um par *estado-acção* actual e um estado seguinte $\boldsymbol{s'}$, o valor esperado do *retorno* é:

$$R^{a}_{s,s'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.6}$$

Onde $\boldsymbol{R^{a}_{s,s'}}$ é o valor esperado de *retorno* $\boldsymbol{r_{t+1}}$, sempre que no instante $\boldsymbol{t}$ o estado $\boldsymbol{s_t}$ seja $\boldsymbol{s}$ e passe a ser $\boldsymbol{s'}$ no instante $\boldsymbol{t+1}$, quando o agente realiza a acção $\boldsymbol{a_t = a}$.

Os valores de *probabilidade de transição* de estado $\boldsymbol{P^{a}_{s,s'}}$ e de *retorno* esperado $\boldsymbol{R^{a}_{s,s'}}$ determinam os aspectos mais importantes da dinâmica de um **PDM finito**, podendo este ser caracterizado como:

- Um ambiente com evolução probabilística, de acordo com um conjunto finito e discreto de *estados*;
- Para cada *estado* do ambiente, existe um conjunto finito de *acções* possíveis;
- Por cada *acção* executada, o agente recebe um *reforço* positivo ou negativo;

Assim, para grande parte dos problemas de AR é possível que a representação do *ambiente* tenha a forma de um PDM, desde que seja satisfeita a *propriedade de Markov*. No entanto, nem todos os algoritmos de AR necessitam de uma modelação PDM completa do ambiente, embora seja necessário *pelo menos ter-se a visão do ambiente como um conjunto de estados e acções* (Sutton, et al., 1998) .

2.4.3. Princípio da optimalidade

Num PDM, a *utilidade* (**retorno**) de um *estado* é a soma esperada de *recompensas* (**reforços**) descontadas, desse ponto em diante. Assim, existe uma relação directa entre a utilidade de um estado e a utilidade dos seus vizinhos (Bellman, 1957) :

A utilidade de um estado é a recompensa imediata correspondente a esse estado, somada à utilidade descontada esperada do próximo estado, assumindo que o agente escolhe a acção óptima:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P^a_{s,s'} U(s') \tag{2.7}$$

Na resolução desta equação (*Equação de Bellman*) utiliza-se o *algoritmo de iteração de valor* que, através da experiência real (ou simulada), actualiza a *utilidade* de cada estado a partir da *utilidade* dos seus vizinhos, até se chegar a um equilíbrio:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P^a_{s,s'} U_i(s') \tag{2.8}$$

2.5. Classes de métodos algorítmicos

Após a caracterização da AR através da referência das suas características especificas mais relevantes e formalização de alguns conceitos associados, serão neste capítulo sintetizados os *métodos algorítmicos* mais usuais empregues na resolução de problemas deste domínio.

2.5.1. Programação dinâmica (PD)

A *Programação Dinâmica* (PD) é constituída por uma colecção de algoritmos que podem obter políticas óptimas sempre que exista uma modelação perfeita do ambiente como PDM. A PD tem a vantagem de ser matematicamente bem fundamentada (Bellman, 1957), mas exige uma modelação precisa do ambiente sobre a forma de um PDM.

A modelação perfeita de um ambiente como PDM implica um grande custo computacional e por tal os algoritmos clássicos de PD são usados de forma limitada. Contudo, a PD fornece um bom padrão de comparação para o conhecimento de outros métodos utilizados na solução de problemas de AR.

2.5.2. Monte Carlo (MC)

Os métodos de *Monte Carlo* (MC) (Rubinstein, 1981) não exigem uma modelação completa do ambiente e apresentam-se relativamente simples em termos conceptuais. Estes assumem como base de trabalho a simulação de sequências e a média de *retornos* obtidos que convirjam para os valores desejados.
De forma a garantir a existência de um *valor de retorno* bem definido, os métodos MC são utilizados apenas em *tarefas episódicas*, isto é, a experiência é dividida em episódios que de alguma forma alcançam o *estado final*, sem ter em consideração as *acções* que foram seleccionadas. Assim, somente após a conclusão de um episódio é que o valor de retorno é obtido, permitindo a actualização da *função valor* e da respectiva *política de controlo*.

Por definição, os métodos MC exigem que o estado final do processo seja sempre alcançado, o que além de se poder revelar excessivamente lento, inviabiliza a abordagem de problemas cuja solução seja alcançável apenas de forma incremental. Em contrapartida, estes métodos não exigem uma descrição completa do ambiente, sendo suficientes algumas amostras da experiência (sequencias de *estados*, *acções* e sinais de *reforço*) obtidas a partir de uma interacção (real ou simulada) com o ambiente.

Efectivamente, a aprendizagem a partir da experiência é notável, uma vez que sem exigir um conhecimento *a priori* das dinâmicas do ambiente, pode levar a um comportamento óptimo. A modelação requerida apenas necessita gerar *transições de estado*, sem necessitar (ao invés da PD) de todo um conjunto de distribuições probabilísticas para todas as possíveis transições (*modelo de transição*).

Assim, os métodos MC são vocacionados para serem aplicados em cenários onde um simulador de ambiente se encontre disponível, ou cuja sua implementação se apresente simples.

2.5.3. Diferença Temporal (DT)

Os métodos de *diferença temporal* (DT) não exigem um modelo exacto do sistema (à imagem dos métodos MC) e permitem ser incrementais (tal como os métodos de PD). Estes procuram estimar valores de *utilidade* (retorno) para cada estado do ambiente (Sutton, et al., 1998), através dos ganhos oriundos das *transições* e de valores de *estados* sucessivos.

A aprendizagem ocorre directamente a partir da experiência, sem a necessidade de uma modelação completa do ambiente, com a vantagem de actualizar as estimativas para a *função-valor* a partir de outras estimativas já aprendidas em estados sucessivos (*bootstrap*), sem a necessidade de alcançar o estado final de um episódio antes da sua actualização. Assim, a avaliação de uma política é encarada como um problema de predição, isto é, estima-se a *função valor-estado* $\boldsymbol{V^{\pi}}$ sob a política $\boldsymbol{\pi}$.

Perante os outros dois métodos apresentados anteriormente, o método DT apresenta algumas vantagens relevantes:

- Não exige a modelação PDM do ambiente (não exige conhecimento prévio do *modelo de transição* do ambiente);
- Pode ser implementado de forma totalmente incremental para aplicações *on-line* (a sua actualização considera apenas o *estado* seguinte);
- Encontra-se garantida a convergência assimptótica para a resposta correta (embora as actualizações da *função-valor* não sejam obtidas a partir de dados reais, mas sim de valores aproximados);

- Os métodos DT são mais rápidos na sua convergência (do que os métodos MC) para tarefas estocásticas (Tsitsiklis, 1994).

A DT usa as transições obtidas para ajustar os valores dos *estados* observados, de forma a estes concordarem com as *equações de restrições* (que definem o ambiente). Assim, numa transição de estado $\boldsymbol{s \rightarrow s'}$, é aplicada a seguinte actualização à *utilidade*:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\big(R(s) + \gamma\, U^{\pi}(s') - U^{\pi}(s)\big) \tag{2.9}$$

sendo $\boldsymbol{\alpha}$ a *taxa de aprendizagem*, assumindo valores no intervalo $[0, 1]$.

A taxa de aprendizagem $\boldsymbol{\alpha}$ determina a velocidade com que o agente assimila informação, apresentando-se esta menor à medida que $\boldsymbol{\alpha \rightarrow 0}$. Se $\boldsymbol{\alpha}$ não for um parâmetro fixo mas sim uma *função* que diminui à medida que aumenta o número de vezes que o estado é visitado, então $\boldsymbol{U^{\pi}(s)}$ convergirá para o valor correcto (Russell, et al., 2003). A actualização somente envolve o sucessor observado $\boldsymbol{s'}$ (*e não todos os estados seguintes possíveis*), tendo implícito um cálculo pouco exigente em termos computacionais.

2.5.3.1. Aprendizagem função *valor-acção*

No entanto, ao considerarmos a *utilidade* de um estado em função do *modelo de transição* do ambiente, continuamos a necessitar de um conhecimento prévio deste, ou seja, uma modelação PDM.

Para um Agente DT, existe um método alternativo denominado *aprendizagem* $\boldsymbol{Q}$ (ou método *sem modelo*), que utiliza no processo de aprendizagem um substituto para a *utilidade*: uma *função valor-acção*:

$$U(s) = max_a\, Q(a, s) \tag{2.10}$$

sendo $\boldsymbol{Q(a, s)}$ o valor da execução da acção $\boldsymbol{a}$ no estado $\boldsymbol{s}$.

Através desta abordagem, um agente DT que utilize uma *função* $\boldsymbol{Q}$ não necessita de um *modelo de transição* do ambiente, quer para o *processo de aprendizagem*, quer para a *selecção de acções*.

A expressão para a *aprendizagem* $\boldsymbol{Q}$ de um agente DT, calculada sempre que uma *acção* $\boldsymbol{a}$ é executada no *estado* $\boldsymbol{s}$ e resulta no *estado* $\boldsymbol{s'}$, é a seguinte:

$$Q(a, s) \leftarrow Q(a, s) + \alpha\big(R(s) + \gamma\, max_{a'}\, Q(a', s') - Q(a, s)\big) \tag{2.11}$$

2.5.3.2. Grandes espaços de estados

Embora transcenda o âmbito deste trabalho, existem problemas de AR que originam *grandes espaços de estados* que por vezes se apresentam inviáveis para abordagens onde as *funções de utilidade* se encontrem representadas de uma forma discreta, como seja através de tabelas onde se encontram memorizados pares de associação entre *tuplos* de entrada e valores de saída, como é o caso de pares (*estado,valor*).

Uma das formas possíveis de abordagem a esta problemática será o uso de tipos de representação para a função que não seja sob a forma de uma tabela, nomeadamente uma *aproximação de função*.

Uma das principais abordagens propostas é o uso de uma *função de utilidade*, descrita como uma *função linear ponderada* de um conjunto de funções base (ou *características do ambiente*) (Russell, et al., 2003):

$$U_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s) \tag{2.12}$$

A utilização desta estratégia permite, entre outros aspectos, uma considerável compactação dos dados obtidos, além de possibilitar a generalização do conhecimento obtido a partir dos estados visitados, para estados ainda não visitados.

2.6. Conclusão

Referimos neste capítulo as *características específicas* e mais relevantes associadas à AR, formalizámos alguns dos *conceitos-chave* associados a esta temática e posteriormente sintetizámos os *métodos algorítmicos* mais usuais empregues na resolução de problemas deste domínio. Encontramo-nos agora preparados para enunciar ao longo do próximo capítulo, alguns dos *algoritmos* de maior relevância que foram sugeridos ao longo das últimas *duas décadas*, como soluções a empregar em problemas deste âmbito.

3. Métodos de Aprendizagem por Reforço

Um dos conceitos centrais da *Aprendizagem por Reforço* (AR) é o de *política*. Uma *política* num cenário de AR é a *componente responsável pelo comportamento que o sistema assume, de forma a alcançar o objectivo*.

Neste capítulo optou-se, para facilidade de compreensão, por fazer uma separação clara e distinta entre os mecanismos propostos para o processo de **exploração** do ambiente (denominados de *métodos de exploração*) e aqueles propostos para o processo de **aproveitamento** da informação já adquirida (denominados de *mecanismos de aprendizagem*). Cada um destes *componentes*, embora apresentados de uma forma distinta ao longo dos capítulos seguintes, são parte integrante de *políticas comportamentais* específicas.

Os *métodos de AR*, embora permanentemente adquiram novas valências devido ao seu carácter evolutivo, podem ser de uma forma geral caracterizados através de duas famílias distintas: aqueles que, através de *simulação da experiência*, actualizam um *modelo interno do mundo*, denominados de *métodos* **com modelo**; e aqueles que não recorrem a qualquer modelo interno, denominados *métodos* **sem modelo**.

Uma outra distinção usualmente empregue entre *métodos de aprendizagem* é a da aquisição de conhecimento ser proporcionada exclusivamente valorizando o *estado*, ou seja através de uma *função* **valor-estado**, ou ser proporcionada através da valorização do par *estado-acção* através de uma *função* **valor-acção**.

As funções de *exploração* e *aproveitamento* do ambiente podem ainda apresentar-se idênticas entre si, e a *estratégia* assume a denominação de **On-Policy**, ou distintas entre si assumindo a *estratégia* a denominação de **Off-Policy**.

3.1. Componente Exploratória

Num cenário de *aprendizagem activa*, onde um agente deve aprender o que fazer (ou seja, que *acçõ*es tomar através da interacção com o *ambiente*), a principal questão que se coloca a este é, em cada situação, conseguir resolver a dualidade entre *explorar* o ambiente ou *aproveitar* a informação já conseguida de forma a evoluir no caminho do *objectivo* definido.

Esta dualidade é resolvida ao nível do *método de exploração* utilizado com base no qual o *agente* opta entre escolher aquela *acção* que se lhe apresentar actualmente melhor avaliada (*aproveitamento da informação*) ou então, uma outra que possivelmente o encaminhe para um *estado* ainda pouco (ou nada) explorado (*exploração do ambiente*), de forma a conseguir angariar maior conhecimento.

Denomina-se de *agente sôfrego* (**greedy**) aquele que, como consequência de fazer experiências com pequenas variações se fixa na *política* actual, não aprendendo as *utilidades* dos outros *estados* e raramente convergindo para a *política óptima* do ambiente. Um esquema razoável que possa levar a um comportamento óptimo do agente terá sempre de ser *sôfrego no limite da exploração infinita*, **GLIE** (*Greedy in the Limit with Infinite Exploration*) (Russell, et al., 2003).

As estratégias possíveis de serem empregues pelos *métodos de exploração* são várias, variando estas entre dois extremos: a maximização da *exploração* e a maximização do *aproveitamento*. Alguns exemplos:

- **greedy**: Máximo aproveitamento (maximizar a *recompensa* no curto prazo);
- **ε-greedy**: Lugar à Exploração (com maior ou menor *peso probabilístico*);
- **softMax**: Probabilidade da selecção da *acção* em função do seu valor estimado;
- **HARL**: Associação de uma determinada *heurística* ao método exploratório.

Os diversos *métodos de exploração* seleccionados no âmbito deste trabalho pela sua relevância, enquadram-se nas estratégias intermédias apresentadas, *ε-greedy* e *softMax*, e tentam ser representativos das atuais abordagens e soluções propostas para esta problemática.

3.1.1. Métodos ε-greedy

Os três métodos apresentados neste grupo são variantes e enquadram-se todos numa estratégia onde a exploração se encontra presente, com maior ou menor *peso probabilístico*.

3.1.1.1. ε-greedy

O método de exploração *ε-greedy*[1] foi descrito inicialmente, e em simultâneo, no mesmo documento em que foi apresentado o mecanismo de aprendizagem *Q-Learning* (Watkins, 1989). Consiste numa tentativa de manter o equilíbrio entre as componentes de *exploração* e de *aproveitamento* do ambiente, através de uma escolha probabilística entre a *acção* que maximiza o valor da *função* $\boldsymbol{Q}$ para o estado actual $\boldsymbol{s}$, e uma qualquer acção de escolha puramente aleatória.

A sua definição é a seguinte:

[1] A implementação do método *ε-greedy* encontra-se descrita no apêndice 8.3

$$\boldsymbol{a_t} = \begin{cases} \boldsymbol{a_t^*} & \textit{com probabilidade } \mathbf{1} - \boldsymbol{\varepsilon} \\ \textit{acção aleatória com probabilidade } \boldsymbol{\varepsilon} \end{cases} \quad (3.1)$$

O método preconiza a selecção de uma qualquer *acção aleatória* com uma probabilidade $\boldsymbol{\varepsilon}$ ou, complementarmente, a selecção da *acção mais valorizada* $\boldsymbol{a_t^*}$ na *função* $\boldsymbol{Q}$ com uma probabilidade $(\mathbf{1}-\boldsymbol{\varepsilon})$. Assim, será expectável que uma variação do valor de $\boldsymbol{\varepsilon}$, definido no intervalo $[0,1]$, corresponda a uma variação directa (*no mesmo sentido*) da *componente exploratória.*

Será também de notar que, embora à partida a *componente exploratória* se encontre garantida e desejável, à medida que o algoritmo convirja assintoticamente para uma solução, e por esta componente se manter *constante*, esta possa passar a interferir crescentemente na estabilização desta mesma convergência. Por outro lado, ao ser garantida uma *componente exploratória* permanente, potenciamos a não estabilização sobre *máximos locais* mas sim o encontrar de *políticas óptimas.*

3.1.1.2. ε-greedy com ε decrescente

O método de exploração *ε-greedy com ε decrescente*[2] é uma variante directa do método *ε-greedy* e foi sugerido inicialmente no mesmo documento em que foi apresentado o antecessor (Watkins, 1989). Neste caso, o valor da probabilidade $\boldsymbol{\varepsilon}$ de escolha de uma *acção aleatória* (*exploração*) é fixa inicialmente num valor mas vai gradualmente decrescendo ao longo do tempo, privilegiando assim de uma forma crescente o *aproveitamento da informação* já adquirida.

A taxa $\boldsymbol{TRE}$ (*Taxa de Redução de Epsilon*) é fixa e definida inicialmente no intervalo [0,1]. Tem como significado a *razão inversa da taxa de decréscimo de* $\boldsymbol{\varepsilon}$ (e consequentemente, a componente *exploratória*). A título de exemplo: uma taxa $\boldsymbol{TRE} = 0{,}9$ significa que $\boldsymbol{\varepsilon}$ irá ser reduzido de 10% do seu valor actual a cada *mudança de estado* efectuada pelo agente.

Tenta-se desta forma diminuir de forma progressiva a *componente exploratória* simultaneamente com a evolução do agente no ambiente. No entanto, dever-se-á tomar em consideração que a efectividade deste decaimento na melhoria da convergência para uma solução encontra-se muito dependente da relação entre o valor da $\boldsymbol{TRE}$ aplicado e outros factores não controláveis inicialmente, como sejam: o *tamanho do espaço de estados* gerado pelo ambiente, o *comportamento do agente* quanto à velocidade e qualidade de aprendizagem, entre outros.

---

[2] A implementação do método *ε-greedy com ε decrescente* encontra-se descrita no apêndice 8.4

Por outro lado, ao anularmos a *componente exploratória* ao fim de um número arbitrário de interacções do agente, encontramo-nos a potenciar a possibilidade de estabilização sobre *máximos locais*, não obtendo assim a *política óptima* desejada.

3.1.1.3. ε-greedy VDBE-Boltzmann

O método de exploração *VDBE-Boltzmann*[3] *(Value-Difference Based Exploration)*, sendo outra derivação do *método de exploração ε-greedy* e um dos mais recentes a ser proposto (Michel, 2010), associa a cada estado $\boldsymbol{s}$ um $\boldsymbol{\varepsilon}$ (*probabilidade de exploração*) correspondente, valor este que se adapta à medida que o conhecimento sobre o meio ambiente (*função* $\boldsymbol{Q}$ associada) se altera.

A ideia base é o uso de uma *função valor-estado* que associa um valor de $\boldsymbol{\varepsilon}$ dinâmico a cada estado $\boldsymbol{s}$, de forma a que o agente se apresente *mais explorativo* nos estados em que o conhecimento acerca do ambiente seja menor, ou seja, onde a *variação do valor* da *função* $\boldsymbol{Q}$ se apresente maior ao ser actualizada. De forma análoga, à medida que as variações no conhecimento do ambiente em cada estado $\boldsymbol{s}$ são menores (ou mesmo nulas), também a taxa de exploração $\boldsymbol{\varepsilon}$ associada a este estado diminui (ou mesmo é anulada).

Este efeito obtêm-se através da adaptação de uma *distribuição de Boltzmann* para a variação dos valores estimados da *função* $\boldsymbol{Q}$ em cada estado $\boldsymbol{s}$, tal como de seguida apresentado (Michel, 2010):

$$f(s,a,\sigma) = \left| \frac{e^{\frac{Q_t(s,a)}{\sigma}}}{e^{\frac{Q_t(s,a)}{\sigma}} + e^{\frac{Q_{t+1}(s,a)}{\sigma}}} - \frac{e^{\frac{Q_{t+1}(s,a)}{\sigma}}}{e^{\frac{Q_t(s,a)}{\sigma}} + e^{\frac{Q_{t+1}(s,a)}{\sigma}}} \right|$$

$$f(s,a,\sigma) = \frac{1 - e^{\frac{-|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}}{1 + e^{\frac{-|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}} \tag{3.2}$$

$$\varepsilon_{t+1}(s) = \delta . f(s_t, a_t, \sigma) + (1 - \delta) . \varepsilon_t(s) \tag{3.3}$$

Este algoritmo tem associados dois parâmetros de configuração:

- O parâmetro de *qualidade de exploração* $\boldsymbol{\delta}$, que determina o *grau de influência de cada acção na taxa de exploração*. O seu valor encontra-se no intervalo [0,1] e o autor aconselha como valor o uso do *inverso do número de acções possíveis* para cada estado, devido sobretudo aos bons resultados obtidos por este experimentalmente.

[3] A implementação do método *VDBE-Boltzmann* encontra-se descrita no apêndice 8.5

- A *sensibilidade inversa* **σ**, que pode assumir valores no intervalo $]0, +\infty[$, por sua vez *modela o grau de exploração versus a variação do conhecimento* obtido, isto é, quanto menor o seu valor, maior se revela a influência da *taxa de exploração*, mesmo para variações pequenas do conhecimento adquirido, ou seja, da *função* $\boldsymbol{Q}$.

3.1.2. Métodos SoftMax

O único método seleccionado neste grupo é aquele que actualmente continua a ter maior expressão, como representativo da estratégia de probabilidade de selecção da *acção* em função do seu próprio valor estimado.

3.1.2.1. SoftMax

O método de exploração *SoftMax*[4], baseado na fórmula de distribuição de *Boltzmann*, foi inicialmente abordado num documento publicado pelo professor *Duncan Luce* em 1959 (Luce, 1959). No entanto foi somente sugerida a sua aplicação como método exploratório em 1994 conjuntamente com o Algoritmo SARSA (Rummery, et al., 1994). Consiste na atribuição de *probabilidades de selecção* específicas a cada uma das *acções* possíveis em cada *estado*, valores estes proporcionais ao peso relativo de cada uma perante o conjunto total destas.

A sua definição é a seguinte:

$$P(a)_t = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{k} e^{Q_t(b)/\tau}} \tag{3.4}$$

Este algoritmo tem associado um parâmetro designado por *temperatura* $\boldsymbol{\tau}$ (como analogia ao processo físico) que pode variar no intervalo $]0, +\infty[$. Valores superiores à unidade funcionam como "*Divisores*", ou seja, tendem a normalizar os pesos como todos de igual importância, potenciando a *exploração*. Valores inferiores à unidade tendem a funcionar como "*Multiplicadores*", colocando os Pesos associados sobre uma *curva exponencial* que apresenta uma evolução bastante acentuada. Acompanhando o decréscimo deste valor, é potenciada a selecção das *acções* mais valorizadas e, consequentemente, reduzida a *taxa exploratória*.

De notar que nesta abordagem é expectável um comportamento predominantemente *exploratório* de início, devido ao conhecimento do ambiente ser escasso e provocar uma distribuição uniforme de probabilidades de selecção de uma *acção*. No entanto, gradualmente o agente tenderá a potenciar o *aproveitamento* do ambiente (*sôfrego*), à medida que o conhecimento deste vai aumentando e são privilegiadas as *acções* com maior valor associado (*função* $\boldsymbol{Q}$).

---

[4] A implementação do método *SoftMax* encontra-se descrita no apêndice 8.6

3.1.3. Métodos Heurísticos

Neste capítulo apresentamos um único método representativo de uma classe de *métodos heurísticos*, caracterizada pela associação de uma determinada *heurística* à selecção da *acção*. Esta classe é caracterizada pela forma de associar uma *heurística genérica* ao *método exploratório*, e não pela tipificação da *heurística* utilizada, razão pela qual o método seleccionado é suficientemente representativo.

3.1.3.1. HAQL - Heuristic Accelerated Q-Learning

O método de *exploração HAQL*[5] (da família dos métodos **HARL** - *Heuristic Accelerated Reinforcement Learning*) consiste na associação de uma *heurística* ao método de selecção da próxima *acção* a realizar, num determinado estado (Bianchi, et al., 2004).

A função *heurística* a associar a um qualquer método deste tipo pode derivar do conhecimento entretanto obtido através da exploração do domínio, de pistas inferidas do processo de aprendizagem em si, ou mesmo de uma qualquer formulação prévia que consiga retractar, com um grau de fiabilidade suficiente, o comportamento ideal do agente perante o cenário proposto. Esta terá sempre como objectivo final *o contribuir para o acelerar do processo de aprendizagem*.

No caso específico do *HAQL*, a um *pré-determinado momento* é despoletada a alimentação da heurística com a informação previamente angariada durante as diversas interacções entretanto ocorridas com o ambiente. Esta informação será utilizada na construção de um percurso, seguindo uma estratégia de *Backtracking*.

Assim, o percurso é inicializado no estado final (o *objectivo*, anteriormente memorizado) e, percorrendo recursivamente todos os estados adjacentes conhecidos, valorizada em cada *estado* $\boldsymbol{s}$ aquela *acção* $\boldsymbol{a}$ que obtenha como *estado seguinte* aquele que se apresentar mais próximo do objectivo, sendo as restantes valorizadas a zero.

A definição do método de selecção é a seguinte:

$$\pi(s) = \begin{cases} \arg max_a \left[Q(s,a) + \xi H(s,a)\right] & se\ q \leq p, \\ a_{random} & caso\ contrário, \end{cases} \tag{3.5}$$

E da *heurística* proposta:

$$H(s,a) = \begin{cases} max_i\, Q(s,i) - Q(s,a) + \eta & se\ a = \pi^H(s), \\ 0 & caso\ contrário. \end{cases} \tag{3.6}$$

[5] A implementação do método *HAQL* encontra-se descrita no apêndice 8.7

Alguns parâmetros configuradores apresentam-se incluídos nesta *heurística*:

$\boldsymbol{q}$ e $\boldsymbol{p}$ são parâmetros arbitrários, sendo $\boldsymbol{q}$ um valor aleatório entre $[0,1]$, e $\boldsymbol{p}$ um valor pré-definido. Esta abordagem nada mais é do que a estratégia *ε-greedy* definida através do valor $\boldsymbol{\varepsilon}$ e por tal foi essa a implementação que se seguiu.

$\boldsymbol{\xi}$ é um parâmetro multiplicativo da influência do peso da heurística e que pode assumir valores entre $[0,1]$. Valores reduzidos originam pouca influência da heurística na selecção da *acção*.

Finalmente, $\boldsymbol{\eta}$ é um parâmetro aditivo ao valor da heurística, que pode assumir valores entre $[0,+\infty[$. Valores superiores originam maior predominância da heurística na selecção da *acção*.

De notar que a *heurística* somente começará a influenciar os resultados do método a partir do momento que tenha sido alimentada com informação, ou seja, quando a função *valor-ação* correspondente encontrar-se actualizada.

3.2. Componente de Aprendizagem

O *processo de aprendizagem*, ou seja, o modo como a informação sobre o ambiente é processada e representada para uso do *agente*, é definido através da forma como o algoritmo seleccionado para essa tarefa *valoriza e guarda os dados obtidos*.

Inúmeros tipos de *mecanismos de aprendizagem* (ou *políticas comportamentais*) encontram-se sugeridos, mas muitos deles enquadram-se como possíveis melhoramentos de mecanismos base previamente propostos. Por tal, foram definidas famílias de algoritmos onde são enquadrados todos aqueles algoritmos que, embora apresentem variações de estratégia, tenham características fundamentais semelhantes.

Foram seleccionadas quatro famílias de algoritmos, representativas das abordagens propostas e actualmente de maior relevância na resolução desta problemática. Todos estes enquadram-se nos métodos DT utilizando uma *função valor-acção*, de forma a serem independentes do *modelo de transição* do ambiente.

3.2.1. Mecanismos Q-Learning

Este tipo de mecanismo actualiza a função **Q** segundo a fórmula de *aprendizagem* $\boldsymbol{Q}$ para os Agentes DT. A fórmula proposta tem como principal característica a de, por cada actualização efectuada, não tomar em consideração o valor da *acção* seleccionada pelo *método de exploração*, mas sim o valor da acção mais valorizada

nesse mesmo estado. Desta forma são preconizadas políticas distintas, *avaliativa* e *comportamental*, definindo uma aprendizagem *Off-Policy*.

3.2.1.1. Q-Learning

O mecanismo de aprendizagem *Q-Learning*[6] foi descrito inicialmente em 1989 por *Watkins* da universidade de Cambridge, na sua tese de doutoramento (Watkins, 1989). Pode ser considerado um dos primeiros e mais relevantes mecanismos propostos, enquadrado especificamente na problemática AR. Este consiste em, a cada passo de evolução sobre o ambiente, o algoritmo actualizar a função *valor-ação* definidora do ambiente (*função* $\boldsymbol{Q}$) não com o valor da *acção* seleccionada pelo *método de exploração*, mas sim com o valor da acção mais valorizada no estado actual. Tal é equivalente a dizer que *o valor de um estado é directamente proporcional ao da acção com maior valor* que nele possa ser seleccionada.

A definição para o *mecanismo de aprendizagem* é a seguinte:

```
1) Initialize Q(s,a)
2) Repeat many times
   a) Pick start state
   b) Repeat each step to goal
      i)   Choose a based on Q(s,a)
      ii)  Do a, observe r,s'
      iii) Q(s,a) = Q(s,a) + α[r + γ.max_a' Q(s',a') − Q(s,a)]
      iv)  s = s'
   c) Until s terminal
```

Listagem 1 - *Pseudo-código do mecanismo Q-Learning (Watkins, 1989)*

De notar a utilização na *fórmula de aprendizagem* (fórmula 2.11) de dois parâmetros já previamente referidos: a *taxa de aprendizagem* $\boldsymbol{\alpha}$ (secção 2.5.3) e a *taxa de amortização* $\boldsymbol{\gamma}$ (secção 2.3.3).

3.2.1.2. Q-Learning λ (*eligibility traces*)

A base teórica do conceito de *eligibility traces* foi inicialmente apresentada por *Klopf* (Klopf, 1972) no âmbito de um trabalho sobre sistemas adaptativos para a USAF (*United States Air Force*). No entanto, a integração deste conceito na problemática de AR somente foi realizada em 1989 por *C.Watkins* (Watkins, 1989), ao incorporar este no mecanismo de aprendizagem *Q-Learning*[7].

[6] A implementação do mecanismo *Q-Learning* encontra-se descrita no apêndice 8.8
[7] A implementação do mecanismo *Q-Learning λ* encontra-se descrita no apêndice 8.9

O conceito de *eligibility traces* consiste no *registo temporário da ocorrência de um determinado evento*, quer seja a passagem por um *estado* ou a selecção de uma determinada *acção*. Este registo permite a sinalização de quaisquer parâmetros associados ao evento como elegíveis para posteriormente poderem sofrer alterações na sua definição. Assim, quando da ocorrência de um erro, este poderá ter a sua origem relacionada com os estados e/ou acções sinalizadas.

Este conceito, ao permitir encurtar a separação entre *eventos* e *informação apreendida*, de alguma forma unifica os métodos *Monte Carlo* (secção 2.5.2) e os de *Diferença Temporal* (secção 2.5.3), fazendo emergir a possibilidade da existência de métodos intermédios ou intercalares (Sutton, et al., 1998).

A definição para o *mecanismo de aprendizagem* é a seguinte:

```
1) Initialize Q(s,a) arbitrarily and e(s,a) = 0; for all s,a
2) Repeat (for each episode)
   a) Initialize s
   b) Choose a from s using policy derived from Q
   c) Repeat (for step of episode)
      i)  Take action a, observe r,s'
      ii) Choose a' from s' using policy derived from Q
      iii) δ ← r + γQ(s',a*) − Q(s,a)
      iv) e(s,a) ← e(s,a) + 1
      v) For all s,a :
         (1)  Q(s,a) ← Q(s,a) + αδe(s,a)
         (2)  If a' = a* then
            (a)  e(s,a) ← γλe(s,a)
         (3)  else
            (a)  e(s,a) ← 0
      vi) s ← s'; a ← a'
   d) until s terminal
```

Listagem 2 - *Pseudo-código do mecanismo Q-Learning λ (Watkins, 1989)*

Algumas características relevantes deste *mecanismo* de AR são de seguida referidas e explicadas.

Este tem associado um parâmetro denominado de *factor de rastreamento* $\boldsymbol{\lambda} \in [\mathbf{0}, \mathbf{1}]$ que tem uma relação inversa com a *taxa de decaimento da sinalização do evento*. Assim, quanto maior o seu valor, menor será a velocidade com que a sinalização do evento decai ao longo dos diversos passos de evolução do algoritmo.

Embora os eventos sejam sinalizados através de uma função *valor-acção* auxiliar $\boldsymbol{e}(\boldsymbol{s},\boldsymbol{a})$, esta sinalização é automaticamente incorporada na estrutura única de definição do ambiente (a *função* $\boldsymbol{Q}$), permitindo assim ser exercida uma influência transparente sobre todo o mecanismo base de aprendizagem.

Sendo este algoritmo a aplicação de um conceito sobre o mecanismo de aprendizagem *Q-Learning*, é natural que a *política* associada a este coincida com a do mecanismo em foco. Assim, em cada *estado* **s** existente, somente são sinalizados os eventos referentes à *acção* **a** que nesse estado *se encontre mais valorizada*, sendo as restantes acções possíveis mantidas sem qualquer sinalização.

A sinalização de um evento (selecção de determinada *acção* **a** num determinado *estado* **s**) é realizada de forma *incremental e unitária*, o que significa que eventos da mesma natureza que ocorram próximos no tempo irão *incrementar cumulativamente* a sua sinalização, tal como ilustrado na figura seguinte:

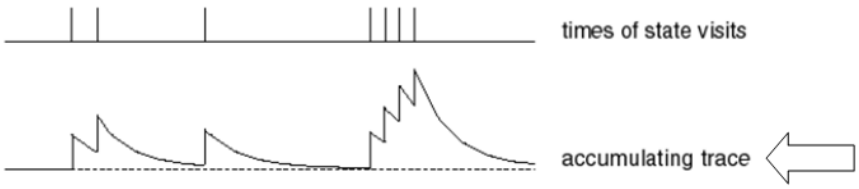


Figura 2 - *Sinalização de eventos de forma acumulada (Watkins, 1989)*

3.2.1.3. Q-Learning λ (*replacing*)

Em 1996, *Sutton* (Sutton, et al., 1996) apresenta uma alternativa ao conceito de *elegibility traces* (secção 3.2.1.2). Devido à sinalização de um evento no conceito original ser realizada de uma forma *incremental* (ou seja, *acumulativa*), constatou que para eventos da mesma natureza que ocorressem próximos no tempo, seria possível que estes ficassem sinalizados de forma excessiva (*valores superiores a* 1), e por consequência sobreavaliados pelo processo de aprendizagem.

Por tal, sugere como alternativa a substituição do processo de sinalização de eventos pela atribuição de um valor unitário *único*, *não acumulável*, demonstrando ainda que esta alteração se manifesta *mais rápida e origina informação de maior qualidade* durante o processo de aprendizagem[8].

Assim, a definição para o *mecanismo de aprendizagem*, sendo idêntica à do mecanismo original (secção 3.2.1.2), apresenta uma subtil mas relevante diferença, ao afectar com um valor *unitário não acumulável* a sinalização de um evento.

---

[8] A implementação do mecanismo *Q-Learning λ (replacing)* encontra-se descrita no apêndice 8.10

```
1) Initialize Q(s,a) arbitrarily and e(s,a) = 0; for all s,a
2) Repeat (for each episode)
   a) Initialize s
   b) Choose a from s using policy derived from Q
   c) Repeat (for step of episode)
      i) Take action a, observe r,s'
      ii) Choose a' from s' using policy derived from Q
      iii) δ ← r + γQ(s',a*) − Q(s,a)
      iv) e(s,a) ← 1
      v) For all s,a :
         (1)  Q(s,a) ← Q(s,a) + αδe(s,a)
         (2)  If a' = a* then
            (a)  e(s,a) ← γλe(s,a)
         (3)  else
            (a)  e(s,a) ← 0
      vi) s ← s'; a ← a'
   d) until s terminal
```

Listagem 3 - *Pseudo-código Q-Learning λ (replacing) (Sutton, et al., 1996)*

De notar que agora, sendo a sinalização de um evento (selecção de determinada *acção* **a** num determinado *estado* **s**) realizada de forma *unitária* e *não incremental*, eventos da mesma natureza que ocorram próximos no tempo serão sempre sinalizados com um *valor máximo e não acumulativo com sinalizações anteriores*, tal como ilustrado na figura seguinte:

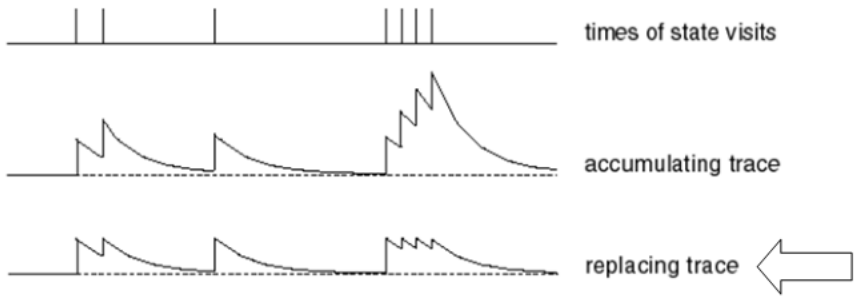


Figura 3 - *Sinalização de eventos de forma unitária (Sutton, et al., 1996)*

3.2.2. Mecanismos SARSA

Este é um tipo de mecanismo semelhante ao *QLearning*, ou seja, em cada evolução efectuada sobre o ambiente, é actualizada a *função* $\boldsymbol{Q}$ segundo a fórmula de *Aprendizagem* $\boldsymbol{Q}$ para os Agentes DT. No entanto, agora em cada actualização efectuada é sempre tomada em consideração o valor da *acção* seleccionada pelo *método de exploração*, sendo assim preconizadas políticas semelhantes *avaliativa* e *comportamental*, definindo uma aprendizagem *On-Policy*.

3.2.2.1. SARSA

O mecanismo de aprendizagem *SARSA*[9] (*State-Action-Reward-State-Action*) foi originalmente descrito por *Rummery* e *Niranjan*, da universidade de Cambridge (Rummery, et al., 1994), em Setembro de 1994. Em tudo semelhante ao mecanismo *QLearning* (proposto cinco anos antes) apresenta como principal diferença o facto de o algoritmo actualizar a função *valor-acção* (*função* ***Q***) com o valor da *acção* seleccionada pelo *método de exploração*, (e não com o valor da acção mais valorizada no estado actual).

Os autores constataram que o algoritmo *Q-Learning*, ao considerar exclusivamente as *acções* mais valorizadas no momento em determinado *estado* e desprezando as restantes, poderia apresentar alguns problemas. Por exemplo, a *sofreguidão* (*greedy*) na selecção de um caminho mais curto (solução *óptima*), desprezando outros que poderiam apresentar-se mais *seguros* por manterem a distância perante obstáculos perigosos a evitar.

Assim sendo, ao serem valorizadas todas as *acções* possíveis num determinado estado, o algoritmo *SARSA* tende a ser mais *conservador* (e também por consequência *mais lento*) na convergência para uma solução. No entanto, esta particularidade pode revelar-se interessante para alguns tipos de aplicações, sobretudo aquelas que coloquem em destaque factores como a *segurança*.

A seguinte figura (Sutton, et al., 1998) ilustra os conceitos enunciados: perante um ambiente onde entre a *origem* e o *objectivo* é colocado um obstáculo onde os *estados* são valorados (através do *retorno* associado a estes) de uma forma *dramaticamente negativa*, o mecanismo *Q-Learning* não hesitará em convergir para um caminho *óptimo*, sem ter em consideração a perigosidade da proximidade do "*penhasco*". Já o mecanismo *SARSA* terá em consideração esta proximidade e, embora *mais lentamente*, convergirá para o caminho mais *seguro*:

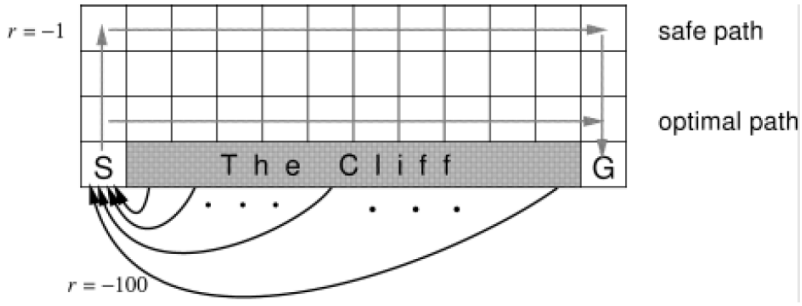


Figura 4 - *Caminho óptimo versus caminho seguro (Sutton, et al., 1998)*

[9] A implementação do mecanismo *SARSA* encontra-se descrita no apêndice 8.11

A definição para o *mecanismo de aprendizagem*, sendo idêntica à do mecanismo *Q-Learning* (secção 3.2.1.1), apresenta agora a diferença de valorizar *todas as acções possíveis* num determinado estado:

```
1) Initialize Q(s,a)
2) Repeat many times
   a) Pick start state
   b) Repeat each step to goal
      i)   Choose a based on Q(s,a)
      ii)  Do a, observe r,s'
      iii) Q(s,a) = Q(s,a) + α[r + γ.Q(s',a') − Q(s,a)]
      iv)  s = s'
   c) Until s terminal
```

Listagem 4 - *Pseudo-código para mecanismo SARSA (Rummery, et al., 1994)*

3.2.2.2. SARSA λ (*eligibility traces*)

O mecanismo de *eligibility traces*, inicialmente concebido por *Klopf* em 1972 (Klopf, 1972) e integrado na problemática de AR em 1989 por *C.Watkins* (Watkins, 1989), será por sua vez integrado com o algoritmo *SARSA*[10] (em simultâneo com a apresentação deste) em 1994 por *Rummery* e *Niranjan* (Rummery, et al., 1994).

Toda a sua justificação conceptual e características fundamentais apresentam-se idênticas às já referidas na descrição anterior para o mecanismo *Q-Learning* λ (secção 3.2.1.2), existindo ainda muitas semelhanças quanto às soluções encontradas para a sua implementação.

Excluindo a característica fundamental do mecanismo *SARSA* de valorizar todas as *acções* possíveis num determinado estado, também a definição para o *mecanismo de aprendizagem* correspondente apresenta-se em tudo idêntica ao enunciado para o mecanismo *Q-Learning* λ de seguida apresentado.

[10] A implementação do mecanismo *SARSA* λ encontra-se descrita no apêndice 8.12

```
1) Initialize Q(s,a) arbitrarily and e(s,a) = 0; for all s,a
2) Repeat (for each episode)
   a) Initialize s
   b) Choose a from s using policy derived from Q
   c) Repeat (for step of episode)
      i) Take action a, observe r,s'
      ii) Choose a' from s' using policy derived from Q
      iii) δ ← r + γQ(s',a') − Q(s,a)
      iv) e(s,a) ← e(s,a) + 1
      v) For all s,a :
         (1)  Q(s,a) ← Q(s,a) + αδe(s,a)
         (2)  e(s,a) ← γλe(s,a)
      vi) s ← s'; a ← a'
   d) until s terminal
```

Listagem 5 - *Pseudo-código para o mecanismo SARSA λ (Rummery, et al., 1994)*

De notar no algoritmo apresentado que, em cada *estado* s existente, são sinalizados os eventos referentes a qualquer *acção* a possível nesse estado que tenha sido seleccionada pela *política* associada.

3.2.2.3. SARSA λ (*replacing*)

No trabalho apresentado em 1996 por *Sutton* (Sutton, et al., 1996) aparece também a referência a esta variante de sinalização dos eventos aplicada ao mecanismo de aprendizagem SARSA λ[11]. Uma vez mais, por idênticas razões e de forma semelhante à discutida anteriormente para o mecanismo *Q-Learning λ (replacing)* (secção 3.2.1.3), a sinalização de eventos passa pela atribuição de um valor unitário *único*, *não acumulável*, tal como se poderá verificar pelo algoritmo do *mecanismo de aprendizagem* correspondente, de seguida apresentado.

[11] A implementação do mecanismo *SARSA λ (replacing)* encontra-se descrita no apêndice 8.13

```
1) Initialize Q(s,a) arbitrarily and e(s,a) = 0; for all s,a
2) Repeat (for each episode)
   a) Initialize s
   b) Choose a from s using policy derived from Q
   c) Repeat (for step of episode)
      i) Take action a, observe r,s'
      ii) Choose a' from s' using policy derived from Q
      iii) δ ← r + γQ(s',a') − Q(s,a)
      iv) e(s,a) ← 1
      v) For all s,a :
         (1)  Q(s,a) ← Q(s,a) + αδe(s,a)
         (2)  e(s,a) ← γλe(s,a)
      vi) s ← s'; a ← a'
   d) until s terminal
```

Listagem 6 - *Pseudo-código mecanismo SARSA λ (replacing) (Sutton, et al., 1996)*

3.2.3. Mecanismos Dyna-Q

Este é um tipo de mecanismo que acrescenta ao *QLearning* a capacidade de *planeamento*, através da contínua actualização de um *modelo do ambiente*, simulando sistematicamente um numero pré-determinado de *evoluções* sobre este.

3.2.3.1. Dyna-Q

A arquitectura *Dyna* (Sutton, 1990) pode ser caracterizada como uma extensão a mecanismos de AR que lhes permite acrescentar um *modelo interno do mundo*. Este *modelo* é definido como algo que se comporta como o próprio mundo: dado um *estado* $\boldsymbol{s}$ e uma *acção* $\boldsymbol{a}$, é suposto obter-se uma previsão da *recompensa* resultante $\boldsymbol{r}$ e do *próximo estado* $\boldsymbol{s}'$. Desta forma, sendo o *estado* observável, torna-se possível construir um *modelo* através do conjunto de exemplos adquiridos a partir das interacções com o mundo.

Nas arquitecturas *Dyna*, o *modelo do mundo* é utilizado como uma substituição directa do ambiente nos mecanismos de AR. Estes continuam na sua normal interacção com o mundo, mas adicionalmente as etapas de aprendizagem são também executadas sobre o *modelo*, utilizando os resultados previstos em vez dos reais, tal como ilustrado através da figura seguinte.

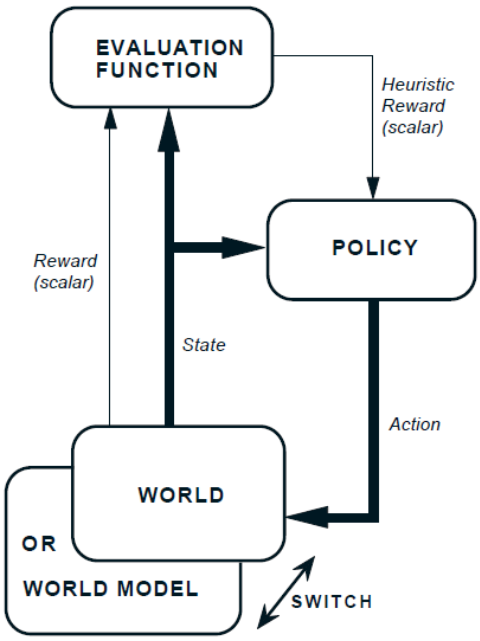


Figura 5 - *Arquitectura Dyna (Sutton, 1990)*

Assim, cada experiência real pode ser acompanhada por inúmeras experiências hipotéticas realizadas sobre o *modelo do mundo*, cada uma destas contribuindo adicionalmente para o processo de aprendizagem. O efeito acumulado destas experiências é a aproximação da política actual da *política óptima* perante o modelo existente: consegue-se assim uma forma de *planeamento*.

O mecanismo *Dyna-Q*[12] (Sutton, 1990) apresenta-se consequentemente como a combinação entre o mecanismo de aprendizagem *Q-Learning* e a ideia, originária das arquitecturas *Dyna*, de usar um modelo do mundo para gerar *experiência hipotética* através de *planeamento*.

A definição para o *mecanismo de aprendizagem* é de seguida apresentada.

[12] A implementação do mecanismo *Dyna-Q* encontra-se descrita no apêndice 8.14

```
1) Initialize Q(s,a) and Model(s,a) for all s,a
2) Repeat (for each episode)
   a) Initialize s
   b) Choose a from s using policy derived from Q
   c) Take action a, observe r,s'
   d) Q(s,a) = Q(s,a) + α[r + γ.max_a' Q(s',a') − Q(s,a)]
   e) Model(s,a) ← s',r (assuming a determinist environment)
   f) Repeat N times:
      i)  s ← random previously observed state
      ii) a ← random action previously taken in s
      iii) s',r ← Model(s,a)
      iv) Q(s,a) = Q(s,a) + α[r + γ.max_a' Q(s',a') − Q(s,a)]
```

Listagem 7 - *Pseudo-código para mecanismo Dyna-Q (Sutton, 1990)*

De notar a existência de um ciclo, integrado em cada episódio do mecanismo de aprendizagem, que promove a actualização da função *valor-acção* $\mathbf{Q}(\mathbf{s},\mathbf{a})$ através de diversas actualizações aleatórias deste com o *modelo do mundo* resultante da experiência entretanto adquirida. Este ciclo tem uma *variável de controlo* $\boldsymbol{N}$ que significa o *número de iterações* que, por cada episódio, são realizadas para actualização do *modelo*.

Quanto maior for o valor desta variável, melhor resultará o *planeamento* de acções futuras, devido ao numero superior de actualizações de que este usufruirá. No entanto, este valor terá relação directa com um *custo computacional* mais elevado. Pelo contrário, ao diminuirmos o número de interacções iremos limitar a qualidade do planeamento a gerar, obtendo no limite um puro mecanismo *Q-Learning*, sem qualquer tipo de planeamento, se reduzirmos as interacções a *zero*.

3.2.3.2. Dyna-Q com varrimento priorizado

Ao analisarmos um pouco mais atentamente as arquitecturas *Dyna*, apercebemo-nos de que as interacções executadas sobre o *modelo do mundo* foram inicialmente concebidas de forma aleatória, ou seja, sem qualquer critério subjacente na escolha do par *estado-acção* sobre o qual incidirá a simulação.

Este problema foi inicialmente abordado por *Peng* e *Williams* num trabalho publicado em 1992 (Peng, et al., 1992). No entanto, foi somente em 1993 que *Moore* e *Atkeson* (Moore, et al., 1993) analisaram este tema e propuseram uma alternativa, sugerindo adicionar ao mecanismo *Dyna-Q* uma forma de priorização na escolha dos pares *estado-acção* a envolver na simulação sobre o *modelo do mundo*, a qual designaram por *varrimento priorizado* (*prioritized sweeping*)[13].

[13] A implementação do mecanismo *DynaQ com varrimento priorizado* é descrita no apêndice 8.15

Assim, este mecanismo continua a apresentar a capacidade de *planeamento*, característica das arquitecturas *Dyna*, através da contínua actualização de um *modelo* do ambiente. No entanto, a simulação de diversas evoluções sobre esse mesmo ambiente não são agora puramente aleatórias mas sim guiadas através da alimentação e processamento de uma *fila* que contem os pares $(\boldsymbol{s},\boldsymbol{a})$ visitados, tendo estes associados individualmente uma *prioridade* proporcional à *variação do valor da função Q* (*Diferença Temporal*) respectiva.

A definição para o *mecanismo de aprendizagem* é de seguida apresentada.

1) Initialize $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})$, $\boldsymbol{Model}(\boldsymbol{s},\boldsymbol{a})$ for all $\boldsymbol{s},\boldsymbol{a}$ and $\boldsymbol{PQueue}$ to empty
2) Repeat (*for each episode*)
   a) Initialize $\boldsymbol{s}$
   b) Choose $\boldsymbol{a}$ from $\boldsymbol{s}$ *using policy derived from Q*
   c) Take action $\boldsymbol{a}$, observe $\boldsymbol{r},\boldsymbol{s}'$
   d) $\boldsymbol{Model}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{s}',\boldsymbol{r}$ *(assuming a determinist environment)*
   e) $\boldsymbol{p} \leftarrow |\boldsymbol{r} + \boldsymbol{\gamma}.\boldsymbol{max}_{\boldsymbol{a}'}\boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})|$
   f) If $\boldsymbol{p} > \theta$, insert $\boldsymbol{s},\boldsymbol{a}$ into $\boldsymbol{PQueue}$ with priority $\boldsymbol{p}$
   g) Repeat $\boldsymbol{N}$ times, while $\boldsymbol{PQueue}$ is not empty:
      i) $\boldsymbol{s},\boldsymbol{a} \leftarrow \boldsymbol{first}(\boldsymbol{PQueue})$
      ii) $\boldsymbol{s}',\boldsymbol{r} \leftarrow \boldsymbol{Model}(\boldsymbol{s},\boldsymbol{a})$
      iii) $\mathbf{Q}(\mathbf{s},\mathbf{a}) = \mathbf{Q}(\mathbf{s},\mathbf{a}) + \boldsymbol{\alpha}[\mathbf{r} + \boldsymbol{\gamma}.\mathbf{max}_{\mathbf{a}'}\mathbf{Q}(\mathbf{s}',\mathbf{a}') - \mathbf{Q}(\mathbf{s},\mathbf{a})]$
      iv) Repeat, for all $\bar{\boldsymbol{s}},\bar{\boldsymbol{a}}$ predicted to lead to $\boldsymbol{s}$:
         (1) $\bar{\boldsymbol{r}} \leftarrow \boldsymbol{predicted\ reward}$
         (2) $\boldsymbol{p} \leftarrow |\bar{\boldsymbol{r}} + \boldsymbol{\gamma}.\boldsymbol{max}_{\boldsymbol{a}'}\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) - \boldsymbol{Q}(\bar{\boldsymbol{s}},\bar{\boldsymbol{a}})|$
         (3) If $\boldsymbol{p} > \theta$, insert $\bar{\boldsymbol{s}},\bar{\boldsymbol{a}}$ into $\boldsymbol{PQueue}$ with priority $\boldsymbol{p}$

Listagem 8 - *Pseudo-código DynaQ - Prioritized Sweeping (Moore, et al., 1993)*

Uma variável importante a considerar neste algoritmo é $\boldsymbol{\theta}$, denominada de *limiar de prioridade* (*threshold*), que significa o *limiar* a partir do qual um determinado par *estado-acção* é inserido na *fila* se apresentar um valor superior de *prioridade*, para processamento posterior nas simulações sobre o *ambiente*. Um valor demasiado elevado desta variável poderá implicar uma selecção demasiado exigente dos pares *estado-acção* a colocar em *fila* para simulação, o que originará um escasso *planeamento*.

3.2.3.3. Dyna-H

Já em 2011 (*num trabalho ainda em evolução*) vários autores espanhóis (Santos, et al., 2011) apresentaram uma nova ideia conceptual: orientar a construção do *planeamento* intrínseco ao mecanismo *Dyna-Q*, associando a este uma determinada *Heurística* na selecção dos pares *estado-acção* para simulação[14].

[14] A implementação do mecanismo *Dyna-H* encontra-se descrita no apêndice 8.16

Esta solução apresenta como nítida vantagem o de integrar na sua concepção, à imagem do mecanismo anterior (*DynaQ – Prioritized Sweeping*), uma forma de orientar e optimizar o *planeamento* através de uma *função genérica* que, não dependendo de nenhum modelo explícito do ambiente, efectivamente retracte o comportamento implícito deste.

O modelo sugerido apresenta como principal mais-valia a integração de uma *função orientadora genérica* no mecanismo de *planeamento*, mas não fornece quaisquer soluções para a construção da *heurística* mais apropriada a aplicar para determinados tipos de ambiente.

A definição para o *mecanismo de aprendizagem* é de seguida apresentada.

1) Initialize $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})$ and $\boldsymbol{Model}(\boldsymbol{s},\boldsymbol{a})$ for all $\boldsymbol{s},\boldsymbol{a}$
2) Repeat (*for each episode*)
   a) Initialize $\boldsymbol{s}$
   b) Choose $\boldsymbol{a}$ from $\boldsymbol{s}$ *using policy derived from Q*
   c) Take action $\boldsymbol{a}$, observe $\boldsymbol{r},\boldsymbol{s}'$
   d) $\mathbf{Q}(\mathbf{s},\mathbf{a}) = \mathbf{Q}(\mathbf{s},\mathbf{a}) + \boldsymbol{\alpha}[\mathbf{r} + \boldsymbol{\gamma}.\mathbf{max}_{\mathbf{a}'}\mathbf{Q}(\mathbf{s}',\mathbf{a}') - \mathbf{Q}(\mathbf{s},\mathbf{a})]$
   e) $\boldsymbol{Model}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{s}',\boldsymbol{r}$ *(assuming a determinist environment)*
   f) Repeat **N** times:
      i) $\boldsymbol{a} \leftarrow \boldsymbol{h_a}(\boldsymbol{s},\boldsymbol{H})$
      ii) $\boldsymbol{if}\ \boldsymbol{s},\boldsymbol{a} \notin \boldsymbol{Model}$
         (1) $\boldsymbol{s} \leftarrow$ random previously observed *state*
         (2) $\boldsymbol{a} \leftarrow$ random *action* previously taken in $\boldsymbol{s}$
      iii) $\boldsymbol{s}',\boldsymbol{r} \leftarrow \boldsymbol{Model}(\boldsymbol{s},\boldsymbol{a})$
      iv) $\mathbf{Q}(\mathbf{s},\mathbf{a}) = \mathbf{Q}(\mathbf{s},\mathbf{a}) + \boldsymbol{\alpha}[\mathbf{r} + \boldsymbol{\gamma}.\mathbf{max}_{\mathbf{a}'}\mathbf{Q}(\mathbf{s}',\mathbf{a}') - \mathbf{Q}(\mathbf{s},\mathbf{a})]$
      v) $\mathbf{s} \leftarrow \mathbf{s}'$

Listagem 9 - *Pseudo-código para mecanismo Dyna-H (Santos, et al., 2011)*

Os autores sugerem, a título de exemplo, a definição de uma *heurística* válida exemplificativa do conceito: A *distância euclidiana a duas dimensões*, entre cada ponto e o objectivo (*goal*):

$$\boldsymbol{h_a}(\boldsymbol{s},\boldsymbol{H}) = \boldsymbol{argmax_a}\boldsymbol{H}(\boldsymbol{s},\boldsymbol{a}), \text{ sendo que: } \boldsymbol{H}(\boldsymbol{s},\boldsymbol{a}) = \|\boldsymbol{s}' - \boldsymbol{goal}\|^2$$

A escolha de uma *heurística* apropriada, que defina o comportamento ideal perante um determinado ambiente, não é uma questão trivial. Através do método de exploração HAQL (secção 3.1.3.1) apresentou-se um exemplo de uma heurística baseada numa estratégia de *Backtracking*, e na definição do *Dyna-H* dá-se como exemplo uma função que define a *distância euclidiana a duas dimensões* entre dois pontos. No entanto, quaisquer destas definições possivelmente poderão ser

complementadas por muitas outras, que se revelem tão ou mais apropriadas perante os ambientes que lhes sejam apresentados[15].

3.2.4. Mecanismos Hierárquicos (MaxQ)

O tipo de mecanismo *hierárquico* preconiza a decomposição de um problema numa série de *sub-tarefas* que, adaptadas a cada tipo de cenário, representam a solução *hierarquizada* do problema proposto. Os valores da *função* ***Q*** não são comuns a toda a solução mas sim individualizados por cada *tarefa* especifica. Este tipo de solução implica o desenho prévio de um *grafo de dependências entre tarefas*, sendo a raiz desse grafo o *objectivo final* a alcançar.

As abordagens *hierárquicas* aplicadas a problemas baseados em AR podem apresentar algumas vantagens substanciais (Dietterich, 1998):

- *Exploração mais eficaz*, ao permitir evoluir substancialmente o algoritmo através de altos níveis de abstracção;
- *Aprendizagem mais rápida*, ao reduzir os parâmetros a apreender, devido à redução do problema a um conjunto de sub-tarefas menores e específicas;
- *Reconfiguração mais rápida* perante novos problemas, devido ao possível reaproveitamento de políticas anteriormente apreendidas através de sub-tarefas já exploradas.

3.2.4.1. MaxQQ

Embora vários tipos de abordagens ao tema tenham sido experimentadas, aquela que foi alvo de investigação em 1998 por *T.Dietterich* (Dietterich, 1998) consiste na *prévia divisão de um problema*, de uma forma *manual*, num conjunto de *tarefas hierarquizadas*. Nesta hierarquia, cada *tarefa* é definida em termos de *estados objectivo* e *condições terminais* de execução, correspondendo a um MDP singular. O método consiste na tentativa de encontrar a *política óptima* para cada *tarefa* individual definida.

---

[15] O autor deste trabalho manteve contactos com a equipa responsável pela apresentação do mecanismo de aprendizagem ***Dyna-H***, nomeadamente com o *Professor Doutor José António*. Destes contactos saiu reforçada a ideia deste mecanismo apresentar como principal mais-valia a integração de uma *heurística genérica* na orientação do mecanismo de *planeamento* do *Dyna-Q*, e não o de propor qualquer formulação específica para a definição dessa mesma *heurística*.

De facto, verificou-se que o algoritmo *Dyna-H* em conjunto com a *heurística* sugerida (*distância euclidiana*) revelou-se, perante o ambiente proposto, com desempenhos marcadamente inferiores aos apresentados pelo mecanismo *DynaQ* (onde o planeamento é realizado de *forma aleatória* sobre o universo conhecido). A resposta (gentilmente obtida da equipa) e os *dados operacionais* gerados podem ser consultados no apêndice 8.1.

Esta abordagem, denominada de *política recursiva ideal*, foi alvo de vários estudos precedentes, tendo como pioneiro *Singh* (Singh, 1992), trabalhos posteriores de *Kaelbling* (Kaelbling, 1993), *Dayan e Hinton* (Dayan, et al., 1993), *Dean e Lin* (Dean, et al., 1995), e finalmente *Sutton e Singh* (Sutton, et al., 1998).

Dietterich no seu trabalho de 1998 introduz e desenvolve um método de *decomposição hierárquica*, integrado neste tipo de abordagem, denominado de MAXQ. Já em 2000, publica um novo documento (Dietterich, 2000) onde, fazendo uso do método de decomposição hierárquica MAXQ, é finalmente apresentado o *mecanismo de aprendizagem MAXQQ*[16], fazendo prova da sua convergência para uma *política* óptima e demonstrando experimentalmente de que este pode convergir muito mais rapidamente do que o mecanismo *Q-Learning*.

O autor tenta no seu trabalho explicar o conceito de *decomposição hierárquica* através de uma aplicação prática sobre um pequeno problema idealizado para o efeito, denominado o *problema do táxi* e que se encontra descrito no apêndice 8.2.

Através deste exemplo é ilustrada a decomposição de um problema de AR, aparentemente complexo de definir, em pequenas *tarefas objectivas que correspondem a rotinas a implementar e integrar no mecanismo de aprendizagem MAXQQ*, mecanismo esse que de forma genérica apresenta a seguinte definição:

```
1) Function MAXQQ(state s, subtask p) returns TotalReward
   a) Let TotalReward = 0
   b) While p is not terminated do
      i) Choose action a = π_x(s) according exploration policy π_x
      ii) Execute a
         (1) If a is primitive
            (a) Observe one-step reward r
         (2) else a is a subroutine
            (a) r := MAXQQ(s,a), which invokes subroutine a, and
                returns the total reward received while a executed.
      iii) TotalReward := TotalReward + r
      iv) Observe resulting state s'
      v) If a is primitive
         (1) V(a,s) := (1 – α).V(a,s) + α.r
      vi) else a is a subroutine
         (1) C(p,s,a) := (1 – α).C(p,s,a) + α.max_a'[V(a',s') + C(p,s',a')]
   c) Return TotalReward
```

Listagem 10 - *Função genérica do mecanismo MAXQQ (Dietterich, 2000)*

[16] A implementação do mecanismo *MAXQQ* encontra-se descrita no apêndice 8.17

O conhecimento adquirido em cada interacção do algoritmo com o ambiente, através da evocação de uma *primitiva* ou de uma *sub-tarefa*, integra com a função *valor-acção* que caracteriza o ambiente (*função* $\boldsymbol{Q}$) através da seguinte expressão:

$$\boldsymbol{Q^{\pi}(p,s,a) := V^{\pi}(a,s) + C^{\pi}(p,s,a)}$$

Uma sub-tarefa pode ser definida por um conjunto de acções *primitivas* e/ou que correspondam a *sub-tarefas*. A escolha da acção e a sua correspondente adequação à tarefa onde se encontra integrada é da responsabilidade da *política* associada a esta $\boldsymbol{\pi_x}$.

No algoritmo *MAXQQ*, a evocação de uma *acção primitiva* resume-se à sua execução e observação da *recompensa* que esta origina, enquanto a evocação de uma *sub-tarefa* corresponde a uma *chamada recursiva* ao próprio algoritmo *com esta como parâmetro*, observando posteriormente o total de *recompensas acumuladas* que as suas *acções* constituintes originaram durante a sua execução.

Duas estruturas auxiliares caracterizam o ambiente, perante o tipo de *acção* executada: A função *valor-acção* $\boldsymbol{V(a,s)}$ que representa a contribuição das *acções primitivas* (*folhas* do *grafo* gerado) para o conhecimento do ambiente, e a função *valor-acção individualizada* a cada tarefa $\boldsymbol{C(p,a,s)}$ que representa a contribuição das *sub-tarefas* para o conhecimento do ambiente.

3.2.4.2. HSMQ

O mecanismo de aprendizagem *HSMQ*[17] (*Hierarchical Semi-Markov Q-Learning*) enunciado por *Dietterich* em 2000 (Dietterich, 2000) é apresentado como uma simples extensão ao *Q-Learning*, exemplificativo da aplicação de mecanismos de *decomposição hierárquica* a problemas de AR. Toda a sua concepção e características fundamentais apresentam-se idênticas às já referidas na descrição anterior para o mecanismo *MAXQQ* (secção 3.2.4.1).

Embora apenas tenha sido citado dois anos após a apresentação do mecanismo *MAXQQ*, na realidade o *HSMQ* foi apenas incluído pelo autor a título demonstrativo de como é possível, sem decompor a função *valor-acção* que caracteriza o ambiente (*função Q*) nos seus diversos componentes oriundos das contribuições distintas das acções *primitivas* e *sub-tarefas*, integrar as diversas *tarefas hierarquizadas* num único algoritmo de AR.

[17] A implementação do mecanismo *HSMQ* encontra-se descrita no apêndice 8.18

Uma das consequências da *não decomposição* da função *valor-acção* que caracteriza o ambiente, ou seja, cada *sub-tarefa* aprender através da actualização da sua própria *função* $\boldsymbol{Q}$, é a deste mecanismo comparativamente com o *MAXQQ* se apresentar significativamente mais lento no processo de aprendizagem (Dietterich, 2000) .

A definição para o *mecanismo de aprendizagem* é apresentada de seguida.

```
2) Function HSMQ(state s, subtask p) returns TotalReward
   a) Let TotalReward = 0
   b) While p is not terminated do
      i) Choose action a = π_x(s) according exploration policy π_x
      ii) Execute a
         (1) If a is primitive
            (a) Observe one-step reward r
         (2) else a is a subroutine
            (a) r := HSMQ(s,a), which invokes subroutine a, and
                returns the total reward received while a executed.
      iii) TotalReward := TotalReward + r
      iv) Observe resulting state s'
      v) Q(p,s,a) := (1 – α).Q(p,s,a) + α[r + max_a' Q(p,s',a')]
   c) Return TotalReward
```

Listagem 11 - *Função genérica do mecanismo HSMQ (Dietterich, 2000)*

De notar que a actualização do conhecimento adquirido sobre o ambiente, resultado do processo de aprendizagem, é realizada no *HSMQ* exclusivamente sobre a função *valor-acção* (*função* $\boldsymbol{Q}$), sendo esta individualizada para cada *sub-tarefa* específica.

3.3. Conclusão

Tendo concluído a apresentação e respectiva caracterização funcional do conjunto de *métodos exploratórios* e *mecanismos de aprendizagem* seleccionados para análise no âmbito deste trabalho, coloca-se agora a necessidade de ser montado todo um *ambiente experimental* aferível, que permita a realização dos ensaios preconizados. Este será precisamente o tema do próximo capítulo.

4. Concretização experimental

Após a selecção de um conjunto de *algoritmos* representativos da investigação desenvolvida até à data no domínio das *políticas avaliativas* e *comportamentais* a aplicar na resolução de problemas de AR, e da sua interpretação e explicação em termos de *comportamento operacional* (capítulo 3), colocou-se a necessidade de escolha de uma *plataforma de desenvolvimento* para implementação destes, e de sobre esta serem desenvolvidos todos os pré-requisitos e aferições que se impunham, para suporte à geração de *estatísticas funcionais* adequadas.

4.1. Plataforma PSA

No âmbito deste trabalho utilizou-se a plataforma modular PSA (*Plataforma de Simulação de Agentes*), previamente desenvolvida em JAVA pelo *Prof. Doutor Luís Morgado* (Morgado, 2005) que, entre outras características, modulariza de forma exemplar todos os componentes necessários à implementação dos diversos algoritmos seleccionados, assim como à configuração dos ambientes adequados à aferição do desempenho das diversas *estratégias de aprendizagem*.

Ao utilizar a linguagem JAVA (Oracle, 2011) como suporte ao desenvolvimento, a PSA herda algumas características relevantes para a qualidade dos projectos desenvolvidos sobre esta, os quais importa enunciar:

- A *Portabilidade*, permitindo esta ser executada sobre um qualquer ambiente, para o qual exista um interpretador JAVA;
- A *Orientação a Objectos*, que permite a herança e a reutilização do código, de forma estruturada e optimizada;
- A possibilidade de *Modularidade* dos diversos componentes aplicacionais, através de estruturas nativas que possibilitam, entre outros, a definição de Interfaces;
- A *gestão Dinâmica* e *Transparente* dos recursos em uso, optimizando em tempo real a sua utilização e disponibilização, como seja a memória de trabalho.

A disponibilização desta plataforma de uma forma *aberta* pelo seu Autor possibilitou ainda a utilização da mesma como *infra-estrutura base*, permitindo o desenvolvimento de extensões programáticas a esta como sejam o módulo de geração de *estatísticas* e ainda os diversos *algoritmos de AR*, essenciais para o nosso estudo.

De notar que as características originais da PSA, como sejam a *modularidade* na construção de um agente adaptativo, a facilidade de *integração* nesta de diversas configurações de ambiente e ainda a possibilidade de *visualização em tempo real* da evolução de um agente sobre um determinado ambiente, foram mantidas em simbiose estreita com as novas extensões entretanto desenvolvidas.

4.1.1. Adaptação ao projecto

Nesta fase existiu a necessidade de adaptação da *plataforma* PSA seleccionada à realidade do projecto proposto. A disponibilidade das *fontes* (gentilmente fornecidas pelo seu autor) em conjunto com a permissão concedida por este para a sua alteração, permitiram entre outros um conhecimento profundo das estruturas programáticas utilizadas na implementação desta, bem como a sua reconfiguração em moldes úteis ao projecto. De todas as alterações realizadas, as que mais se destacaram foram:

- Reconfiguração da *main function, entry point* do projecto, de forma a poderem serem mantidas as características de *visualização da evolução do agente* sobre o ambiente, em conjunto com a possibilidade de geração de estatísticas úteis às análises específicas propostas. É neste módulo que são definidos todos os parâmetros configuradores das condições de *operação, registo e finalização* de cada ensaio algorítmico;
- Reconfiguração do *módulo de simulação*, de forma a suportar todas as operações adicionais impostas, sobretudo aquelas que contribuem para o novo módulo de estatística e ainda a manipulação das rotinas gráficas;
- Separação efectiva em dois pacotes distintos entre os *métodos de exploração* do ambiente e os *mecanismos de aprendizagem* por reforço. Reconfiguração destes para suportar a evocação de *parâmetros globais*, comuns a todos e passíveis de serem afectados externamente em *runtime*.

4.1.2. Implementação de algoritmos

Qualquer dos *métodos e mecanismos* propostos requereu uma implementação e adaptação sobre a plataforma PSA. Para tal, foram criados módulos separados para cada um dos componentes, seguindo de forma precisa os algoritmos (normalmente apresentados sobre a forma de *pseudo-código*) sugeridos pelos autores (capítulos 5 e 7). Estes programas, construídos em linguagem JAVA, foram idealizados de forma a implementar a *Interface* correspondente (*método de exploração* ou *mecanismo de aprendizagem*), previamente arquitectada de forma a fazer a ligação entre estes novos módulos e o comportamento do *agente virtual* disponibilizado pela plataforma.

4.1.3. Módulo de estatísticas

O *módulo de estatísticas* apresenta-se como um componente novo na plataforma PSA, e completamente desenvolvido de raiz. Requereu um estudo preliminar de quais os *dados relevantes a obter e registar* perante as boas práticas usuais, nomeadamente comparando com a documentação cedida pelos autores dos diversos algoritmos estudados.

Da análise efectuada, verificou-se que alguns elementos de aferição se destacaram como presenças assíduas, quando se pretende realizar uma *análise comparativa* entre algoritmos empregues em AR:

- A possibilidade de *repetições de sequências de aprendizagem* (conjuntos de *episódios*) com a mesma parametrização, de forma a *aferir médias operacionais* e não valores individualizados;
- A possibilidade de *limitar a um número fixo de episódios* cada sequência de aprendizagem, de forma a verificar o comportamento global de uma determinado algoritmo na sua *evolução assimptótica pretendida para uma solução óptima* (e não se este consegue atingir um qualquer limite arbitrário);
- O *registo de valores* como: valor mínimo, máximo e acumulado para as seguintes variáveis: número de *episódios* efectuados; *recompensas* recebidas e *desvios* (de convergência) obtidos.

4.2. Ambientes

A problemática envolvida na comparação de *modelos de decisão* distintos, utilizando *agentes autónomos*, não é recente tendo sido inicialmente abordada em 1990 por *Pollack* e *Ringuette* (Pollack, et al., 1990) ao terem sugerido um ambiente genérico de testes denominado *Tileworld*.

No original, o *Tileworld* consiste num ambiente *discreto* em grelha ocupado por *agentes, blocos, buracos* e *obstáculos*. O objectivo do(s) *agente*(s) traduz-se em ganhar(em) tantos pontos quantos conseguir(em), preenchendo os *buracos* empurrando os *blocos*. O agente pode-se mover em qualquer direcção (mesmo nas *diagonais*) mas os *obstáculos* devem ser evitados. O ambiente é *dinâmico*, isto é, os *buracos* e *blocos* podem aparecer e desaparecer de forma aleatória.

Devido ao *ambiente* original se apresentar demasiado complexo para o objectivo pretendido, foi defendida a utilização de uma *versão simplificada* deste (Schut, et al., 2000) por um outro trabalho académico de 2003 (Parsons, et al., 2003) onde foi abordada a mesma problemática.

A simplificação sugerida evidencia plenamente as características pretendidas para este trabalho, consistindo numa versão do *Tileworld* onde:

- O *ambiente* é *estático*, não sofrendo qualquer alteração ao longo do tempo;
- O *agente* é *único* e tem *perfeito conhecimento do estado* do mundo sem qualquer *custo* associado;
- Os *blocos* e *buracos* foram omitidos, mantendo-se somente os *obstáculos*;
- Existe um único *alvo* que o *agente* deve atingir para ter sucesso.

De notar que a *plataforma* PSA já tem em si incorporado um *sistema interpretador* de ficheiros de texto que, de uma forma visual e célere, permite construir e integrar *ambientes* do tipo preconizado.

4.2.1. Ambiente Base

Na secção anterior foram seleccionadas as principais características dos *ambientes* a conceber, que servirão como base para os *testes operacionais* a realizar sobre cada algoritmo em foco. O *ambiente base* seleccionado deverá simultaneamente obedecer às características encontradas e possibilitar, replicando-se resultados experimentais anteriormente publicados, a *aferição da fiabilidade da plataforma PSA e das implementações algorítmicas* sobre ela desenvolvidas no âmbito deste trabalho.

Dentro deste quadro, e para concepção dos *ambientes*, o ideal seria a selecção e utilização de quaisquer ambientes previamente estudados e publicados por um autor reconhecido. Para tal, foi seleccionado e implementado sobre a plataforma PSA, como *ambiente base* dos testes a realizar, uma versão simplificada do *Tileworld* publicada por (Sutton, 1990) e ainda (Sutton, et al., 1998) que obedece aos requisitos desejados: um *ambiente simples*, que se encontra publicado em simultâneo com alguns resultados de *testes replicáveis* efectuados sobre este, por um *autor reconhecido* no tema abordado e simultaneamente no meio académico.

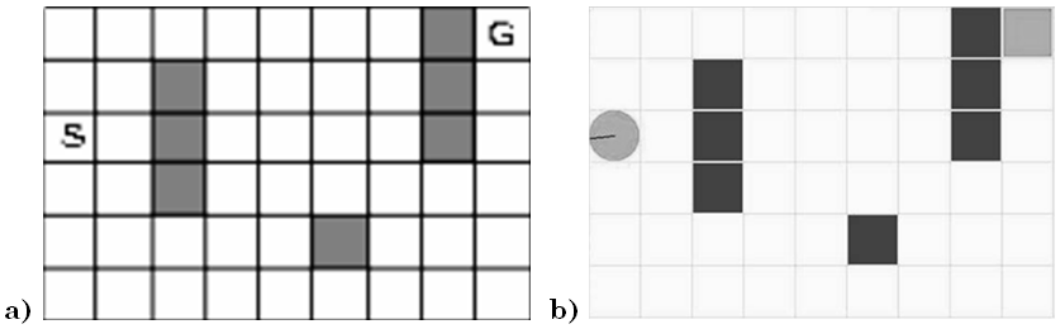


Figura 6 - *a) Ambiente Tileworld (Sutton, 1990) e b) equivalente sobre a PSA*

Os autores sugeriram este *ambiente* simplificado (**9x6**) num contexto de efectivação de testes sobre uma implementação do *mecanismo de aprendizagem Dyna-Q*, variando o *número de passos* utilizados por este para o *planeamento*, após cada episódio de interacção com o *ambiente*.

Os seguintes *parâmetros* foram utilizados por *Sutton* nos testes realizados:

Parâmetros base:
Objectivo:
- O Agente (S) atingir o Objectivo (G)

Acções possíveis:
- **4** (*Cima, Abaixo, Esquerda e Direita*)

Recompensas:
- **0** - Cada acção realizada
- **+1** - Ao atingir o Objectivo

Mecanismo de aprendizagem:
- **Dyna-Q**

Método exploratório:
- **ε-greedy**

Parâmetros de Aprendizagem:
- γ (*Taxa de Amortização*) = **0.95**
- α *(Taxa de Aprendizagem)* = **0.1**
- ε *(Taxa de Exploração)* = **0.1**

Variáveis:
Número de passos de planeamento:
- **0** (***Q-Learning*** *puro, sem qualquer planeamento associado*)
- **5**
- **50**

Listagem 12 - *Parâmetros de teste do mecanismo Dyna-Q (Sutton, 1990)*

Os resultados obtidos pelos autores, na sequência dos testes realizados, são de seguida apresentados.

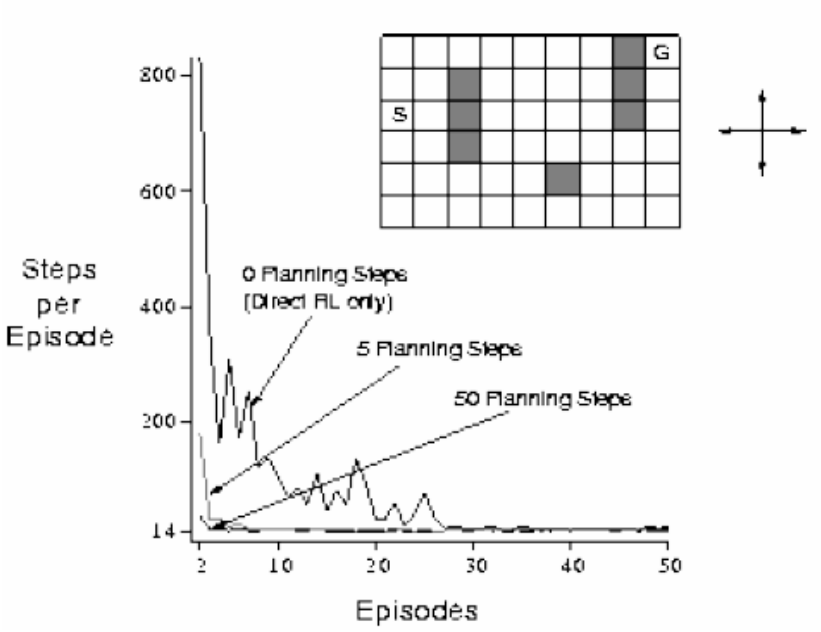


Figura 7 - *Resultados obtidos por Sutton (Sutton, 1990)*[18]

Após a configuração dos diversos algoritmos implementados sobre a plataforma PSA, de forma a existir uma *correspondência exacta entre os parâmetros* utilizados por esta e os aplicados por *Sutton* nos testes apresentados, procedeu-se à efectivação de idênticos testes sobre esta. Foi assim gerado um conjunto de estatísticas que, no final, permitiu a apresentação dos resultados obtidos.

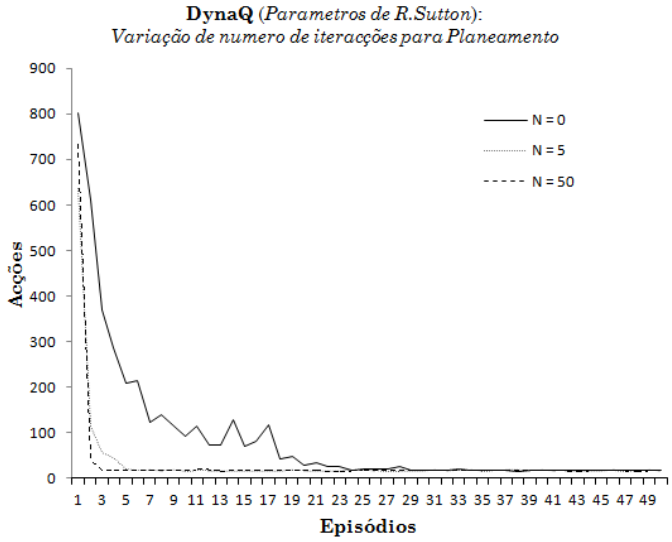


Figura 8 - *Resultados do Dyna-Q (parâmetros de Sutton) obtidos sobre a PSA*

[18] Resultados obtidos por (Sutton, 1990) relativos ao desempenho do mecanismo de aprendizagem DynaQ no contexto do ambiente base definido.

De certa forma, pode-se considerar notável o grau de correspondência conseguido entre os resultados obtidos por *Sutton* e aqueles originados nos testes realizados sobre a plataforma PSA. Conseguimos assim garantir que a *plataforma de testes* utilizada, assim como as *diversas implementações* realizadas sobre esta dos algoritmos em estudo, não diferem significativamente quanto aos resultados obtidos, relativamente aos alcançados por trabalhos previamente reconhecidos e publicados.

4.2.2. Ambiente Expandido

Um dos objectivos do trabalho efectuado foi adicionalmente conseguir verificar se os comportamentos que cada algoritmo apresenta na resolução do *ambiente base*, podem ser generalizados para outros ambientes com as mesmas características mas com *superior grau de complexidade*.

Para tal, foi necessário idealizar um ambiente que, mantendo as mesmas características encontradas para o *ambiente base*, apresentasse um relevante acréscimo no *grau de complexidade* em relação a este. A denominação atribuída a este foi de *ambiente expandido*, sendo apresentado de seguida.

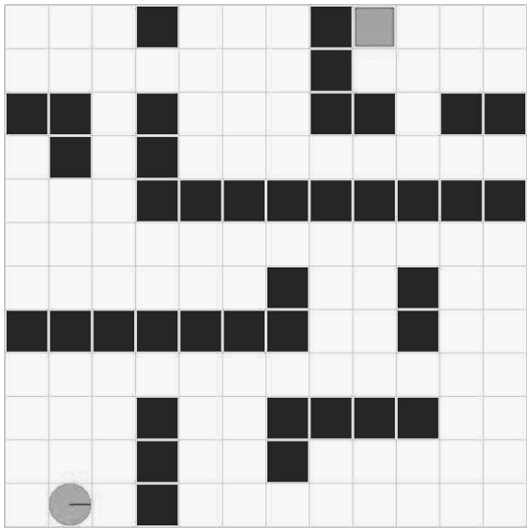


Figura 9 - *Ambiente Expandido, tipo Tileworld simplificado*

Na tentativa de se escalar em complexidade o *ambiente base*, aumentou-se a *área* (**12x12**) e o número de *obstáculos* presentes neste, além da *distância* inicial entre o *agente* e o *alvo*. No entanto, de forma a garantir que na perspectiva do agente, este *ambiente expandido* apresentará as mesmas características a superar do que o *ambiente base*, só que de uma forma incrementada, foi necessário verificar se o comportamento do *agente* seria similar para ambos os *ambientes*.

Para tal, optou-se por replicar o mesmo teste sugerido por *Sutton* a aplicar sobre o *ambiente base*, só que desta vez sobre o *ambiente expandido*, apresentando os resultados obtidos na figura seguinte.

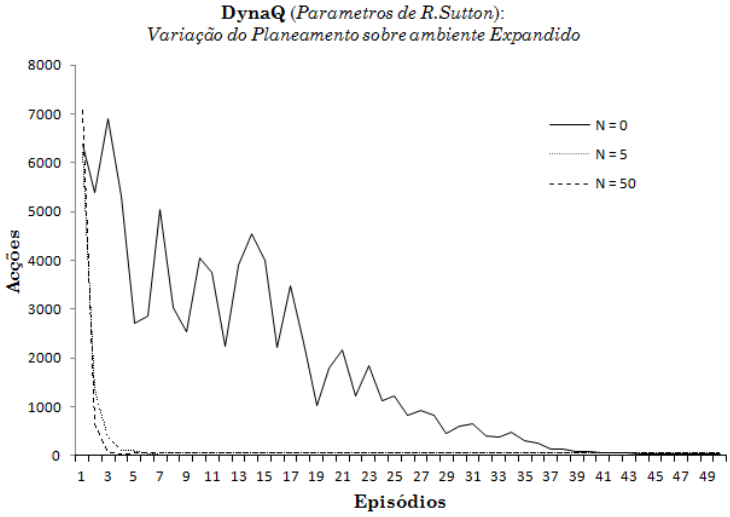


Figura 10 - *Resultados do Dyna-Q sobre ambiente expandido*

Como se pode verificar, o grau de correspondência conseguido entre os resultados obtidos para os *ambientes base* e *expandido* é significativo, existindo apenas um expectável aumento proporcional de *acções* e número de *episódios* realizados, de forma a conseguir convergir assintoticamente para uma *solução óptima*.

4.3. Aferição complementar

Uma das soluções apontadas por diversos autores ao longo de diversos trabalhos foi o de associar uma *heurística*, de forma a potenciar um *método de exploração* (Bianchi, et al., 2004), ou o *mecanismo de aprendizagem*, nomeadamente (a existir) ao nível do planeamento intrínseco a este (Santos, et al., 2011).

De forma a conseguir *aferir* uma vez mais os resultados obtidos através da *implementação algorítmica* ao nível da plataforma PSA, e aqueles publicados por um autor no âmbito dos testes produzidos e publicados por este, seleccionámos mais um dos documentos publicados, desta feita por *Bianchi* em 2004 (Bianchi, et al., 2004): A associação do *mecanismo QLearning* a uma *heurística* associada ao *método de exploração* (HAQL) (secção 3.1.3.1).

O autor apresenta no documento publicado um *ambiente de testes* (16x16), que foi integralmente replicado para utilização sobre a plataforma PSA.

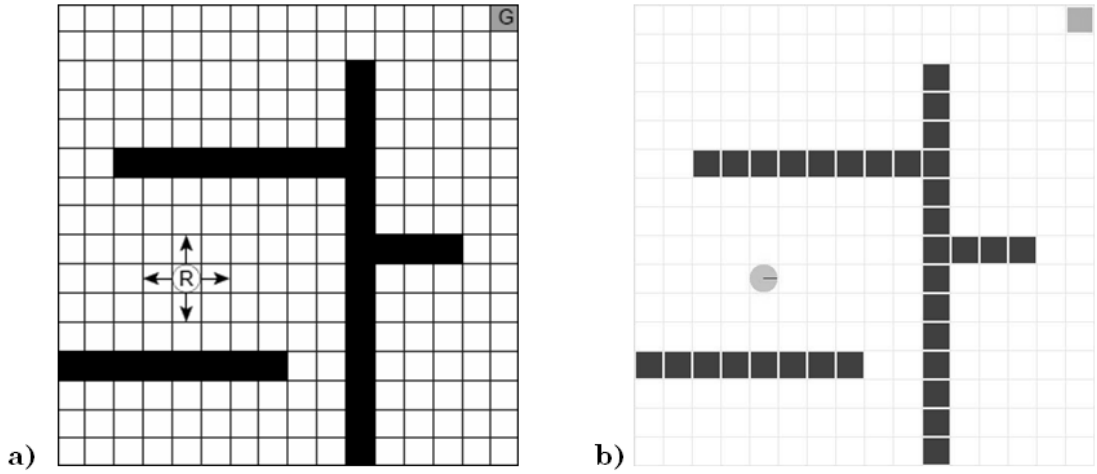


Figura 11 - *a) Ambiente de HAQL (Bianchi, et al., 2004) e b) equivalente sobre PSA*

Os seguintes *parâmetros* foram utilizados por *Bianchi* nos testes realizados:

Parâmetros base:

Objectivo:
- O Agente (R) atingir o Objectivo (G)

Acções possíveis:
- **4** (*Cima, Abaixo, Esquerda e Direita*)

Recompensas:
- **-1** - Cada acção realizada
- **+10** - Ao atingir o Objectivo

Mecanismo de aprendizagem:
- **QLearning**

Método exploratório:
- **ε-greedy (*Heuristic accelerated*)**

Parâmetros de Aprendizagem:
- γ (*Taxa de Amortização*) = **0.99**
- α *(Taxa de Aprendizagem)* = **0.1**
- ε *(Taxa de Exploração)* = **0.1**

Parâmetros de Exploração:
- ξ (*Factor multiplicativo do peso da heurística*) = **1**
- η *(Factor aditivo do peso da heurística)* = **1**
- ***bkE****(Episódio p/ Alimentação heurística)* = **10**

Listagem 13 - *Parâmetros de teste do HAQL (Bianchi, et al., 2004)*

Os resultados obtidos pelos autores, na sequência dos testes realizados, foram os de seguida apresentados.

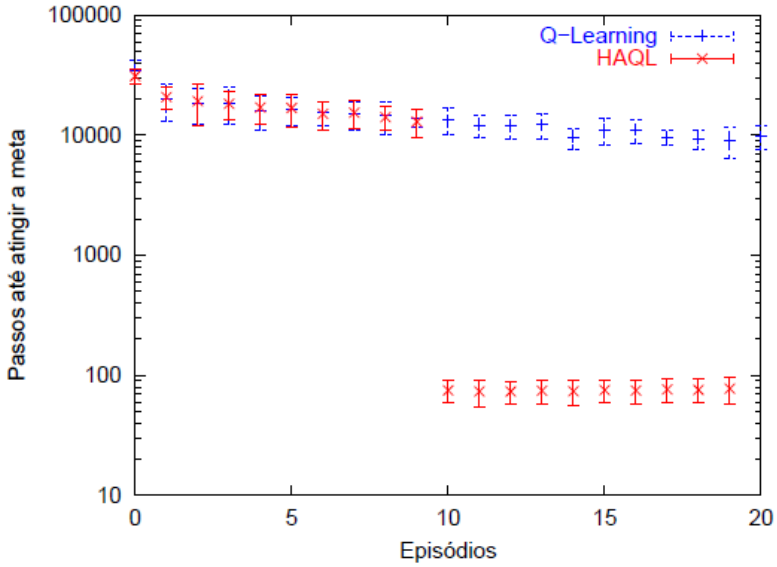


Figura 12 - *Resultados de testes realizados sobre o HAQL (Bianchi, et al., 2004)*

Os resultados obtidos via *algoritmo* implementado sobre a *plataforma* PSA, foram os de seguida apresentados.

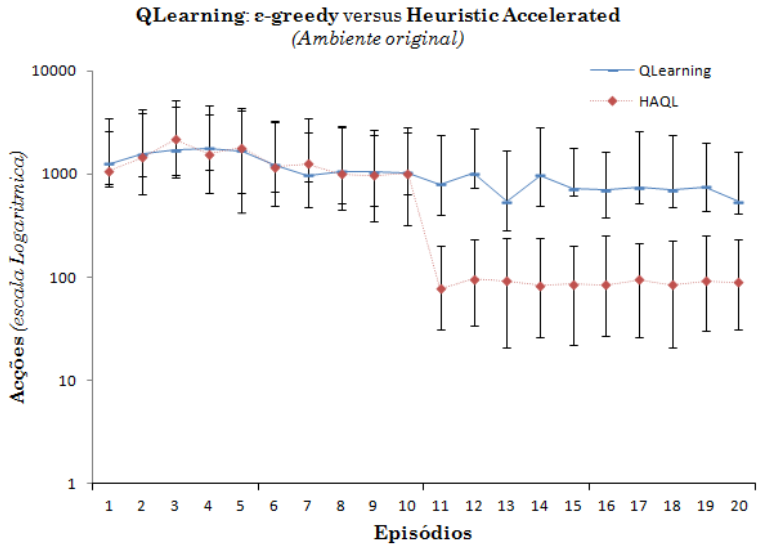


Figura 13 - *Resultados do HAQL (parâmetros de Bianchi) sobre a PSA*

Verifica-se, um *grau de correspondência significativo* entre os resultados obtidos[19] pelo *autor*, e aqueles obtidos via PSA. De salientar a queda abrupta e estabilização do *número de acções* necessárias para o *agente* atingir o objectivo, após a alimentação da *heurística* (episódio **10**) com os dados entretanto adquiridos (mecanismo de *Backtracking*).

[19] Detectou-se uma diferença, exclusivamente no *factor de escala*, entre os dois gráficos apresentados, o que originou uma tentativa de contacto com os autores. Infelizmente, não foi possível até à data obter qualquer esclarecimento adicional.

4.4. Parametrizações e estratégias experimentais

Embora os valores aplicados na parametrização dos diversos *algoritmos* possam por vezes depender localmente destes e do *ambiente* em foco, existem algumas *parametrizações* e *estratégias experimentais* que se apresentam comuns a todos os testes realizados no âmbito deste trabalho, de forma a *maximizar a uniformização* dos métodos e valores aplicados, e ainda incrementar a *consistência dos resultados* obtidos. São estas:

- Depois de ter sido previamente validada na sua *implementação e fiabilidade* de resposta, ir-se-ão realizar todos os ensaios sobre a *plataforma* **PSA** (secção 4.1) modificada, recaindo os testes primariamente sobre o *ambiente* ***base*** (secção 4.2.1) e posteriormente, de forma a verificar a coerência dos dados obtidos após o incremento do grau de dificuldade (escala), sobre o *ambiente* ***expandido*** (secção 4.2.2).

- Uma das estratégias predominantes na maior parte dos trabalhos que foram consultados, utilizada na comparação dos desempenhos apresentados pelos diversos algoritmos de AR, foi considerar a evolução do ***número de acções*** necessárias para atingir o objectivo, ao longo de um determinado número de e*pisódios* ocorridos (*a outra foi a de considerar o acumulado de recompensas obtidas*). Autores como *Sutton* (Sutton, 1990), *Rummery* (Rummery, et al., 1994) ou mesmo *Russell* (Russell, et al., 2003) defendem a utilização desta estratégia. Ao longo deste trabalho será precisamente esta a métrica usada no decorrer de todos os ensaios algorítmicos realizados.

- Não é prudente considerar como representativo de um determinado *teste algorítmico* os resultados obtidos através de um *único ensaio*, devido a este na sua *singularidade* poder de alguma forma debitar resultados que se afastem dos valores típicos em condições semelhantes. Por tal, quaisquer dos valores apresentados, oriundos de um qualquer ensaio efectuado (incluindo aqueles previamente debitados pelos testes de aferição da *plataforma PSA* e respectivos *ambientes*), são na realidade uma ***média aritmética*** de um número pré-determinado de *ensaios repetidos exactamente nas mesmas condições*, que se encontra configurável na respectiva plataforma.

- Nos ensaios realizados, embora alguns dos parâmetros sejam ajustados em função da especificidade do *algoritmo* ou do *ambiente*, outros irão ser mantidos fixos e comuns a cada ensaio. Seleccionados a partir do seu *uso recorrente* nos ensaios consultados, ou por *expressa recomendação* publicada

por um ou vários autores, ou mesmo por *reconhecido valor* com maior efectividade para as condições pretendidas (*quando aplicáveis no contexto*):

- No *ambiente*:
  - **Reforços** gerados:
    - Encontrar o *alvo* = **+10**
    - *Deslocação* ou *colisão* = **0**
  - Número de **acções possíveis**: **4** (*Norte, Sul, Este, Oeste*)
- No *método exploratório*:
  - $\boldsymbol{\varepsilon}$ *(Taxa de Exploração)* = **0.1**
  - $\boldsymbol{\tau}$ (*Temperatura*) = **0.05**
- No *mecanismo de aprendizagem*:
  - $\boldsymbol{\gamma}$ (*Taxa de Amortização*) = **0.95**
  - $\boldsymbol{\alpha}$ *(Taxa de Aprendizagem)* = **0.1**

Finalmente, resta referir que todos os *valores* e respectivos *gráficos* gerados e referidos no âmbito deste trabalho se encontram disponíveis para consulta, através de ficheiros guardados em formato *MS Excel (.xlsx)* (Microsoft, 2011) numa *directoria* anexa à estrutura do projecto desenvolvido.

4.5. Exclusões de ensaios

Nem todos os métodos (*exploratórios*), quer seja pela sua estrutura ou abordagem do problema, se encontram concebidos para serem utilizados em conjunto com todos os mecanismos (de *aproveitamento*) expostos. No entanto, e devido sobretudo à modularidade que foi imposta na implementação destes separando-os efectivamente do respectivo contexto de utilização, na realidade *não foi excluída do estudo qualquer possível combinação entre estes.*

Contudo existiram duas *exclusões* assumidas durante este trabalho:

- Tal como apresentado na secção 3.2.3.3, o *mecanismo de aprendizagem* **Dyna-H** consiste numa tentativa recente (Santos, et al., 2011) em associar uma *heurística genérica* ao *planeamento* realizado pelo *Dyna-Q*. No entanto, e tal como o próprio autor reconheceu (resposta *em apêndice* no secção 8.1), a escolha do tipo de *heurística* não é trivial nem foco do trabalho realizado.
  Efectivamente a heurística sugerida como exemplo (*distância euclidiana a duas dimensões entre dois pontos*), aplicada na resolução de *ambientes* semelhantes aos propostos neste trabalho (*tileworld simplificado*), revelou-se com piores desempenhos do que os apresentados pelo *Dyna-Q* original. *Por tal, não se justificou a inclusão do mecanismo **Dyna-H** em qualquer dos ensaios realizados.*

- A família de *mecanismos de aprendizagem hierárquicos* (secção 3.2.4) apresenta uma vantagem significativa na resolução de problemas de AR ao permitir a *decomposição de problemas complexos em sub-tarefas simplificadas e hierarquicamente organizadas*. A melhoria na *eficácia da exploração* e uma *aprendizagem e reconfigurações mais rápidas* são atributos adicionais a estes (Dietterich, 1998). No entanto, ambos os *ambientes* propostos para realização de testes comparativos entre todos os outros algoritmos, sendo *estáticos* e com um *único objectivo* directo associado, não apresentam complexidade suficiente para que este tipo de mecanismo se possa exprimir demonstrando todas as suas reais capacidades. *Por tal, embora citados e caracterizados, optou-se por não se realizar qualquer ensaio usando* ***mecanismos hierárquicos***.

4.6. Conclusão

Finalmente, e após a contextualização efectuada em termos de *ambientes programáticos* sobre o qual irão decorrer os ensaios preconizados, assim como o foco em *âmbitos específicos* e devidamente enquadrados, ir-se-ão efectuar e documentar os ensaios algorítmicos preconizados. Este é o assunto do próximo capítulo deste trabalho.

5. Resultados experimentais

Neste capítulo são apresentados os resultados experimentais obtidos dos diversos ensaios realizados. Estes, não sendo de natureza exaustiva na descrição do comportamento de cada algoritmo, pretendem sobretudo responder a algumas questões que se colocaram naturalmente ao tentarmos mensurar os níveis de *eficiência* e *eficácia* apresentados por cada um, comparativamente com similares.

5.1. Exploração ε-greedy

O *método de exploração* ε-greedy (secção 3.1.1.1) foi o primeiro a ser sugerido, num contexto de AR, integrado no *mecanismo de aprendizagem* Q-Learning (Watkins, 1989). No entanto surgiram propostas para outros métodos que, sem variarem na essência a estratégia utilizada, tentaram melhorar a resposta deste.

São exemplos disso o *ε-greedy com ε decrescente* (secção 3.1.1.2) e o *VDBE-Boltzmann* (secção 3.1.1.3), soluções propostas de melhoria no desempenho base, que aplicámos sobre os nossos *ambientes de testes* de forma a podermos comparar o seu comportamento. De destacar o ano recente de apresentação deste ultimo método (Michel, 2010), sinal inequívoco da actualidade deste tema.

Os seguintes *parâmetros* locais foram utilizados:

ε-greedy com ε decrescente:
- ***Tre*** (*Taxa de Redução de Epsilon*) = **0.9**

ε-greedy *VDBE-Boltzmann*:
- **δ** (*Qualidade de Exploração*) = **0.25**
- **σ** *(Sensibilidade inversa)* = **0.33**

De seguida são apresentados os resultados obtidos.

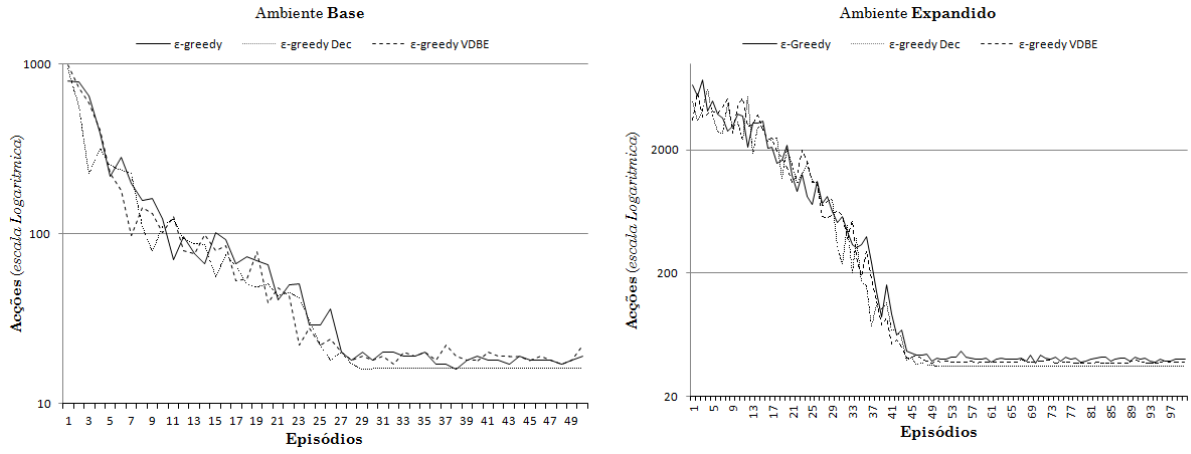


Figura 14 - *Q-Learning com método de exploração* ***ε****-greedy e variantes*

O primeiro comentário derivado da observação dos resultados obtidos é, sem dúvida, a constatação de, ao contrário do esperado, nenhum dos *métodos de exploração* sugeridos posteriormente ao original (sobretudo o mais recente *VDBE-Boltzmann*) se destacou de forma assinalável do comportamento do *método original* na resolução do tipo de *ambiente* proposto.

A curva assimptótica de *convergência* do algoritmo para um valor óptimo (ou *quase óptimo*) descrita pelo *ε-greedy* foi acompanhada quase na perfeição pelas curvas descritas pelos restantes dois *algoritmos*. Assim, a *velocidade de convergência dos algoritmos empregues foi idêntica para qualquer um*, com um comportamento idêntico para ambos os *ambientes*.

No entanto, existiu alguma diferença na estabilização destes sobre um *valor limite*. Como seria previsível, o método *ε-greedy com ε decrescente* ao esgotar a componente de exploração *estabiliza e mantém-se sobre um valor* que, embora na vizinhança, pode não atingir a *solução óptima*. Já o *VDBE-Boltzmann*, oscila nas proximidades de um valor próximo da *solução óptima*, mas sempre mais perto do que o *ε-greedy*, ou seja, com uma *eficácia ligeiramente superior*, tal como detalhado na figura seguinte.

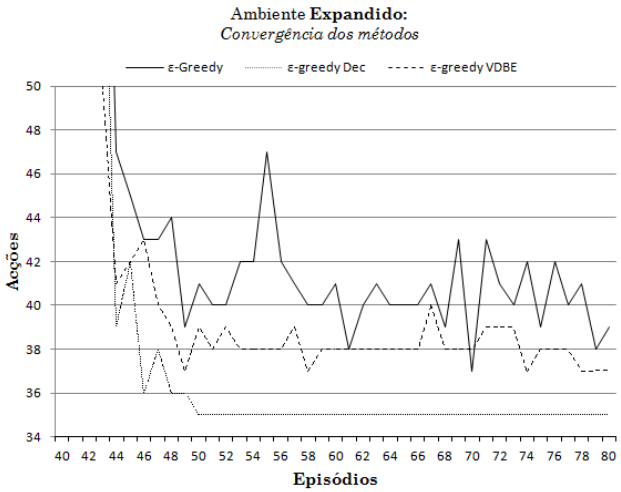


Figura 15 - *Convergência dos métodos ε-greedy e variantes*

5.2. Q-Learning versus SARSA

O *mecanismo de aprendizagem* SARSA (secção 3.2.2.1) surge aproximadamente cinco anos após a publicação do *mecanismo* QLearning (secção 3.2.1.1). A principal diferença entre estes reside no facto de o SARSA actualizar a *função* $\boldsymbol{Q}$ com o valor da *acção* seleccionada pelo *método de exploração*, e não com o valor da *acção mais valorizada* no estado actual (QLearning).

Esta maneira de actuar pode, como inúmeras vezes é citado, provocar na realidade com que o SARSA tenha uma *convergência mais lenta* do que o QLearning para (pelo menos) a vizinhança da *solução óptima*. Por outro lado, o SARSA foi apresentado em conjunto com o *método de exploração* Softmax (secção 3.1.2.1) e o QLearning com o *método* ε-greedy (secção 3.1.1.1), o que no entanto não implica que estes sejam os métodos que permitam melhor desempenho dos *mecanismos* a que se encontram associados.

De forma a podermos visualizar as questões apresentadas, foi realizado o ensaio de seguida apresentado, no qual cada um dos *mecanismos de aprendizagem* é associado a cada um dos *métodos de exploração* em foco.

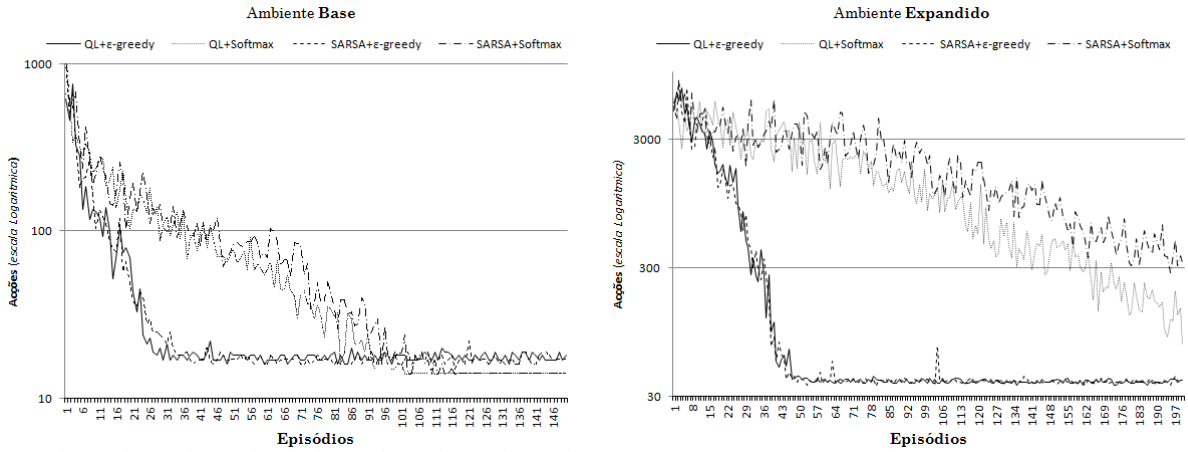


Figura 16 - *Q-Learning e SARSA associados a ε-greedy e Softmax*

Conforme esperado, a resposta a ambos os *ambientes* é idêntica. Mas ao observarmos mais atentamente os resultados obtidos, detectamos algumas características que se apresentam em evidência.

Nos casos estudados, independentemente do *mecanismo de aprendizagem*, o *método exploratório* dita a *eficiência* do algoritmo no seu conjunto. Neste caso, qualquer mecanismo associado ao ε-greedy revela-se mais *eficiente* ao necessitar de menos *episódios* (e menos *acções por episódio*) para convergir para uma vizinhança da *solução óptima*.

Também a resposta do mecanismo Q-Learning em relação ao SARSA, quando associados ambos a idêntico *método exploratório*, apresenta-se consistentemente mais *eficiente*, ou seja, com a necessidade de *um número menor de acções por episódio* para atingir o objectivo. No entanto, as curvas de aprendizagem apresentam-se semelhantes no seu contorno e evolução.

Finalmente, constata-se que qualquer dos *mecanismos de aprendizagem*, quando associados ao *método exploratório* Softmax, ao convergir *estabiliza e mantem-se sobre a solução óptima*. Por seu lado, quando associados ao método ε-greedy a componente *exploratória* encontra-se permanentemente presente, o que implica o algoritmo estabilizar a oscilar numa vizinhança da *solução óptima*. Este comportamento é ilustrado pelos resultados de seguida evidenciados.

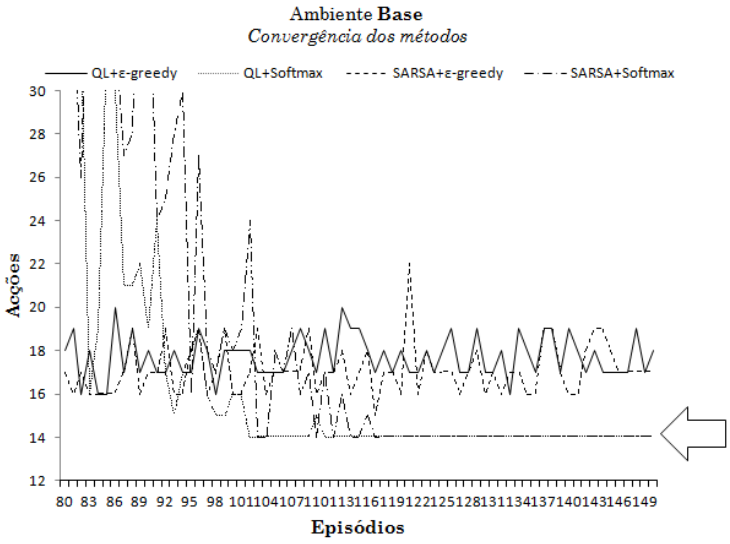


Figura 17 - *Estabilização do método SoftMax sobre a solução óptima*

Também a possível associação do *mecanismo* SARSA aos *métodos exploratórios* ε-greedy e Softmax pode apresentar resultados interessantes em termos de análise comportamental. Os resultados obtidos no ensaio realizado com estas configurações são de seguida apresentados.

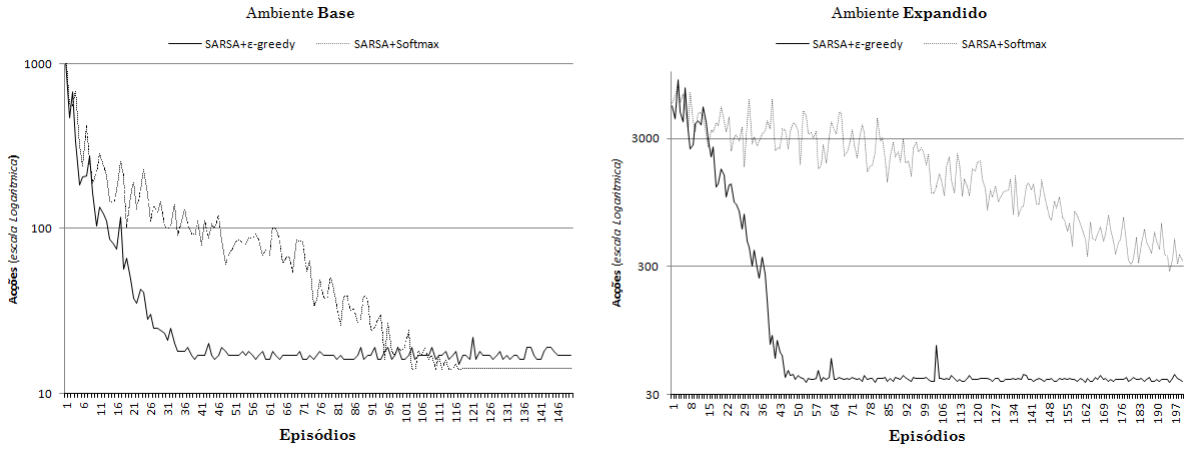


Figura 18 - *SARSA associado aos métodos ε-greedy e Softmax*

Uma vez mais se nota a relação entre a *eficiência* e *eficácia* obtidas e o *método exploratório* associado. O *mecanismo* SARSA apresenta uma nítida maior *eficiência* quando associado ao *método* ε-greedy, ao necessitar significativamente de menos *episódios* e *acções* para convergir para a vizinhança de uma solução óptima. No entanto, apresenta uma maior *eficácia* quando associado a *método* Softmax, ao convergir e estabilizar mais perto da *solução óptima*.

5.3. Exploração Heurística HAQL

A ideia de associar uma *heurística* genérica a um *método exploratório* foi publicada em 2004 por *Reinaldo Bianchi* (Bianchi, et al., 2004), sugerindo a associação de uma estratégia de *backtracking* ao método ε-greedy, originando o HAQL (secção 3.1.3.1).

De facto, todo o documento coloca o foco na *forma de associar uma qualquer heurística ao método exploratório*. No entanto, a própria estratégia sugerida revelou-se bastante promissora pelos resultados apresentados. Esta consiste em, num *determinado momento* ser despoletada a alimentação da *heurística* com a informação previamente angariada durante as diversas interacções entretanto ocorridas com o *ambiente*.

A *heurística* sugerida é baseada na construção de um percurso inverso a partir das coordenadas do objectivo identificado. Percorrendo o *espaço de estados* já conhecido, a partir do *estado final* obtido, são privilegiadas as *acções* em cada estado que obterão como destino um *estado* mais próximo do estado final. Tal pode ser ilustrado a partir do exemplo utilizado por Bianchi (Bianchi, et al., 2004) com recurso a um *ambiente* também já citado neste trabalho num contexto de aferição complementar à plataforma PSA (secção 4.3). O exemplo apresenta o *ambiente* acompanhado de uma *matriz* que representa graficamente uma possível configuração, numa situação de pleno conhecimento da totalidade do *espaço de estados*, para as *acções* privilegiadas em cada *estado*, após a alimentação da *heurística*.

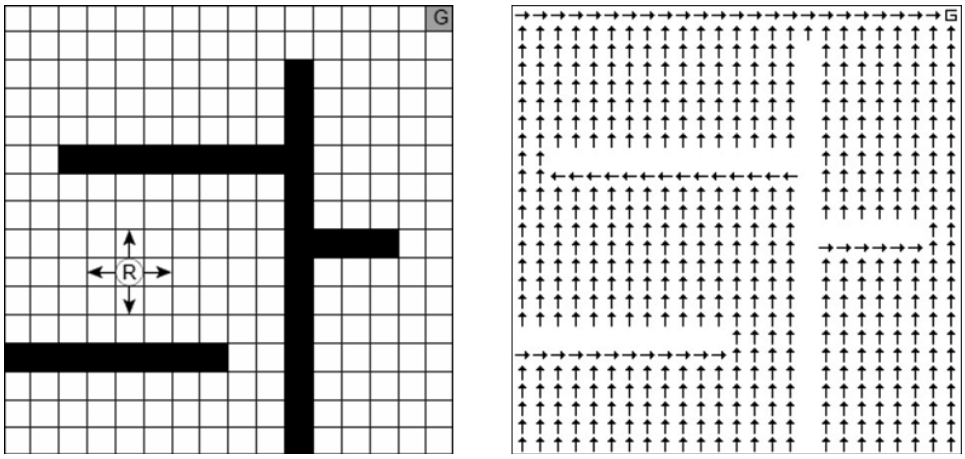


Figura 19 - *Ambiente e propagação heurística associada (Bianchi, et al., 2004)*

Assim, a pergunta que se coloca é a seguinte: *em que momento deve ser alimentada a heurística* com o conhecimento já adquirido? Prematuramente de forma a acelerar logo de início o algoritmo na sua convergência, ou mais tarde de forma à heurística debitar uma informação mais correcta? E que outros efeitos poderão ocorrer ao variar o *episódio* onde é despoletada esta operação?

De forma a tentar responder a estas perguntas, realizou-se o ensaio de seguida descrito, associando o *mecanismo* Q-Learning ao *método* HAQL com a seguinte *parametrização*:

HAQL:
- ξ (*Factor multiplicativo peso heurística*) = **1**
- η (*Factor aditivo peso heurística*) = **1**

Variáveis:
Episódio de activação da heurística (***Bke***):
- **--** (***Q-Learning*** *puro, sem heurística associada*)
- **1, 5 e 15**

Os resultados obtidos foram os de seguida apresentados.

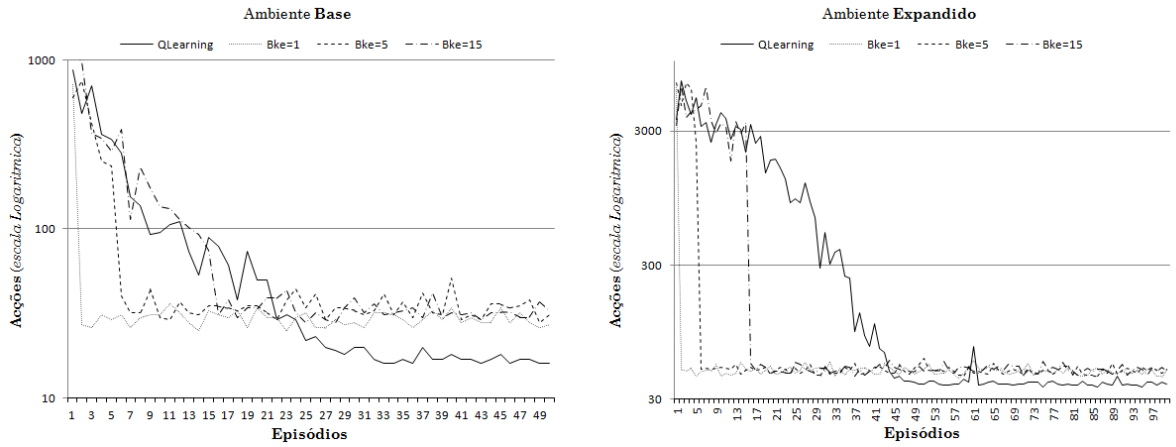


Figura 20 - *HAQL variando o episódio de activação da heurística*

Observando os resultados obtidos é notória a queda abrupta de *acções* necessárias para o *agente* atingir o *objectivo*, logo no episódio seguinte à alimentação da *heurística*. Esta queda ocorre com a *mesma inclinação*, independentemente de qual o *episódio* onde a heurística é activada, e deve-se ao envolvimento desta no processo exploratório, que agora consegue *priorizar em cada estado a acção que mais aproxima o agente* do objectivo pretendido, devido à informação angariada no *primeiro episódio* ser suficiente.

Mas qual será o comportamento do algoritmo após estabilizar na sua *convergência*? Olhemos atentamente um segmento do gráfico originário do *ambiente expandido*.

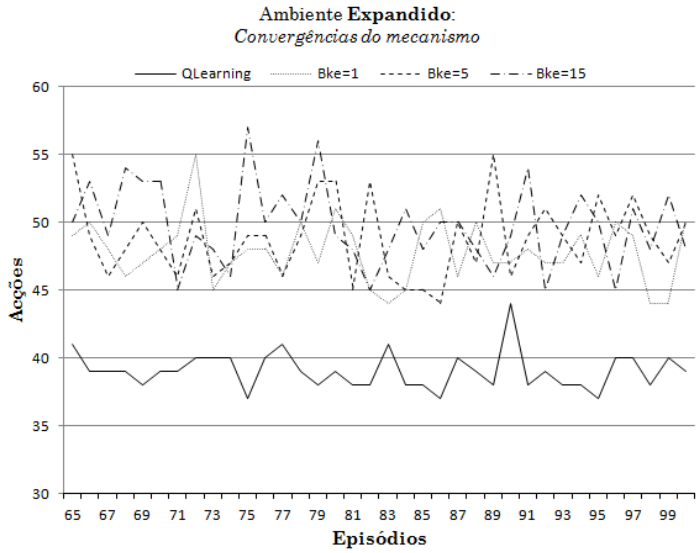


Figura 21 - *Estabilização do HAQL*

Aparentemente, a informação adicional do ambiente que o mecanismo possa aprender a partir do *primeiro episódio* (ou seja, logo após conhecer *onde se encontra o objectivo*) não influencia o *grau de eficácia* do algoritmo, pois este irá sempre *estabilizar a oscilar numa vizinhança da solução óptima*, de forma idêntica qualquer que seja o *episódio* onde a *heurística* tenha sido activada.

No entanto, será de referir que esta estabilização apresenta-se sempre menos *eficaz*, ou seja mais distante da *solução óptima*, do que o Q-Learning (sem qualquer heurística associada ao *método exploratório*). Tal é consequência do *erro*, por pequeno que seja, que a *estratégia de backtracking* pode adicionar e se encontra sempre presente, mesmo após a estabilização da convergência para uma solução.

5.4. Eventos elegíveis

O conceito de *eventos elegíveis* (*eligibility traces*) consiste no *registo temporário da ocorrência de um determinado evento*, quer seja a passagem por um *estado* ou a selecção de uma determinada *acção* (secção 3.2.1.2). Esta *sinalização* foi inicialmente proposta como sendo de *ordem cumulativa* (Watkins, 1989), o que foi posteriormente contestado por outro autor (Sutton, et al., 1996), afirmando este que desta forma seria possível que os eventos ficassem sinalizados de forma excessiva e por consequência *sobreavaliado*s pelo processo de aprendizagem. Sugere então como alternativa a substituição do processo de sinalização de eventos pela atribuição de um valor unitário *único*, *não acumulável* (secção 3.2.1.3).

Uma vez mais, este conceito aparece na tentativa de melhorar o desempenho do *mecanismo de aprendizagem*, desta feita assinalando a passagem por um determinado *estado* e, desta forma, elegendo este temporariamente como alvo preferencial de possível aprendizagem futura.

No entanto, deve-se ter em consideração que este conceito impõe uma *carga de processamento adicional não desprezável* ao mecanismo de aprendizagem ao qual se acopla. De facto, por cada *episódio* realizado é imposta uma actualização a todos os *estados* conhecidos do *agente*, o que corresponde ao decaimento gradual das sinalizações previamente atribuídas.

No ensaio agora realizado quisemos testar o desempenho comparativo do algoritmo (Q-Learning) sem o mecanismo associado e associando o mecanismo, sem e com a modificação sugerida posteriormente, sobre os *ambientes* seleccionados.

Foi utilizada a seguinte *parametrização*:

Q-Learning λ:
- ***λ*** (*Factor de rastreamento*) = **0.01**

Os resultados obtidos são de seguida apresentados.

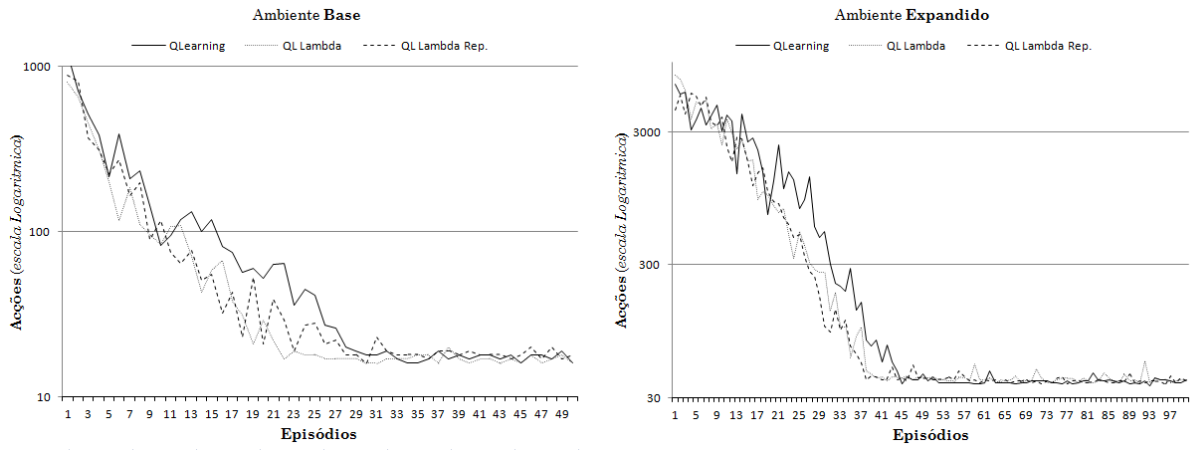


Figura 22 - *Q-Learning, QL λ e QL λ (replacing)*

Como primeira análise, e perante *comportamentos semelhantes* para ambos os *ambientes*, podemos constatar que o mecanismo Q-Learning tendo associado o conceito de *eligibility traces* consegue efectivamente, para *ambas as variantes*, ter maior *eficiência* do que o mecanismo original, ao necessitar de menos *acções* por *episódio* para o *agente* atingir o objectivo, e conseguir convergir para uma *vizinhança da solução óptima* num menor número de *episódios*.

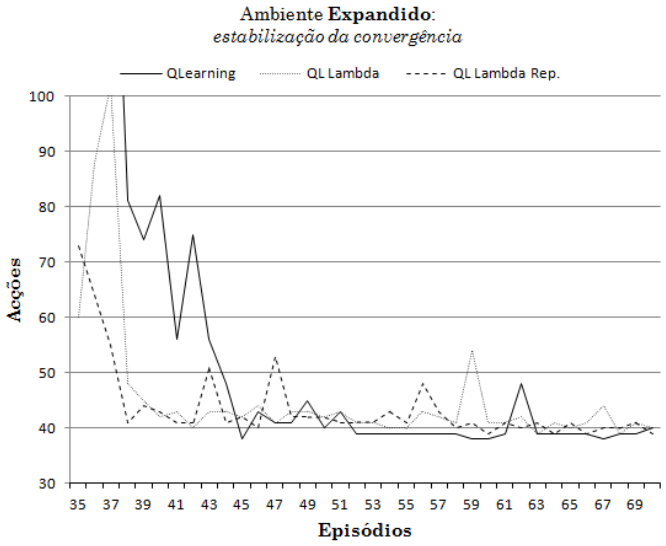


Figura 23 - *Maior eficiência do mecanismo com elegibility traces*

Ao nível da convergência, qualquer das soluções ao estabilizar permanece a *oscilar* da mesma forma *na vizinhança da solução óptima*, independentemente de ser o *mecanismo* Q-Learning original ou o acoplado com qualquer das variantes do método.

5.5. Planeamento no Dyna-Q

O mecanismo *Dyna-Q* (secção 3.2.3.1) apresenta-se como a combinação entre o mecanismo de aprendizagem *Q-Learning* e a ideia, originária das arquitecturas *Dyna*, de usar um modelo do mundo para gerar *experiência hipotética* e adquirir *planeamento*.

Dever-se-á ter em consideração, na comparação com outro tipo de *mecanismos de aprendizagem*, que no Dyna-Q a actualização em cada *episódio* do modelo do mundo interno ao mecanismo impõe uma *carga de processamento adicional não desprezável*, afectando um número pré-determinado de *estados* já conhecidos do *agente*.

Sendo a *simulação* efectuada sobre um *número pré-definido de estados* por cada *episódio* ocorrido, idealizou-se este ensaio para determinar, sobre os *ambientes* do tipo proposto, qual seria o melhor compromisso (à imagem do teste anterior idealizado por *Sutton* (secção 4.2.1)) entre o *número de simulações* a efectuar por *episódio* e a *eficiência* ganha por parte do algoritmo com a variação deste valor.

Os resultados obtidos no ensaio são apresentados de seguida.

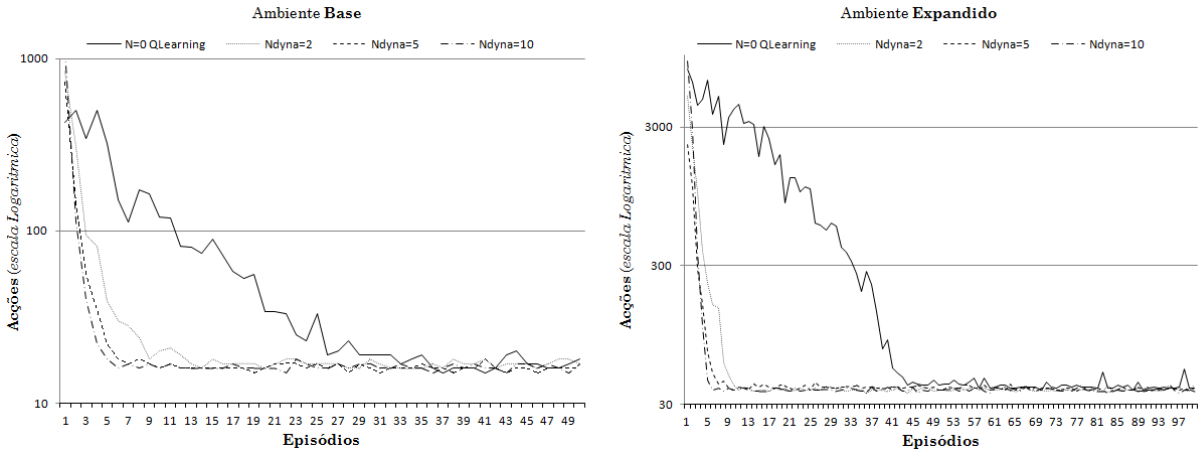


Figura 24 - *Dyna-Q variando o número de simulações por episódio*

Com um *comportamento idêntico* para ambos os ambientes, pode-se desde logo observar o ganho notável de *eficiência* na passagem do mecanismo Q-Learning puro (Dyna-Q *sem planeamento*) para o *mecanismo* Dyna-Q com planeamento, qualquer que seja o *número de simulações* efectuadas. O número de *acções* necessárias para o agente atingir o objectivo cai muito rapidamente para muito próximo da *solução óptima*, ficando a *oscilar na vizinhança desta*.

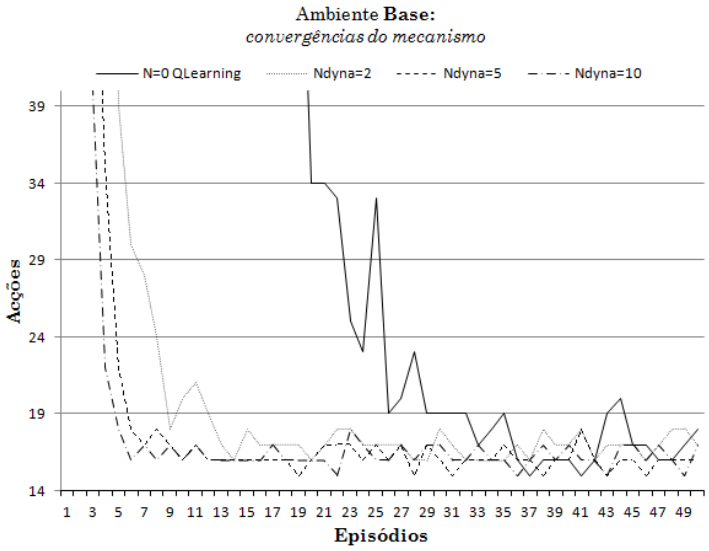


Figura 25 - *Dyna-Q a oscilar na vizinhança da solução óptima*

No entanto, e para os *ambientes* em foco, embora se note existir um ganho acentuado ao incrementar o *número de simulações até cinco por episódio*, a partir da vizinhança desse valor os ganhos em *eficiência* apresentam-se bastante diminutos face ao custo em termos de processamento adicional.

5.6. Varrimento priorizado

Em 1993 *Moore* e *Atkeson* propuseram uma alternativa à forma de planeamento do Dyna-Q, sugerindo adicionar a este mecanismo uma forma de priorização na escolha dos pares *estado-acção* a envolver na simulação sobre o *modelo do mundo*, denominada de *varrimento priorizado* (*Prioritized Sweeping*) (secção 3.2.3.2).

Este mecanismo continuaria a apresentar a capacidade de *planeamento*, característica das arquitecturas *Dyna*, através da contínua actualização de um *modelo* do ambiente. No entanto, a simulação de diversas evoluções sobre esse mesmo ambiente não seriam agora puramente aleatórias mas sim guiadas através da alimentação e processamento de uma *fila* que contem os pares $(\boldsymbol{s}, \boldsymbol{a})$ visitados, tendo estes associados individualmente uma *prioridade* proporcional à *variação do valor da função* $\boldsymbol{Q}$ (*Diferença Temporal*) respectiva.

Intrínseco ao algoritmo encontra-se o *limiar de prioridade* (*threshold*) $\boldsymbol{\theta}$ que significa o limite a partir do qual um determinado par *estado-acção* é inserido na *fila* para processamento posterior nas simulações sobre o *ambiente*, se apresentar um valor superior de *prioridade*. Um valor demasiado elevado desta variável poderá implicar uma selecção demasiado exigente dos pares *estado-acção* a colocar em *fila* para simulação, o que originará um escasso *planeamento*.

De forma a testar o comportamento do algoritmo perante a variação do *nível limite* $\boldsymbol{\theta}$ para inserção na fila de processamento, e sobretudo a efectividade da ideia original de *priorizar os pares estado-acção* a processar nas diversas simulações, foi realizado um ensaio do qual são apresentados de seguida os resultados obtidos.

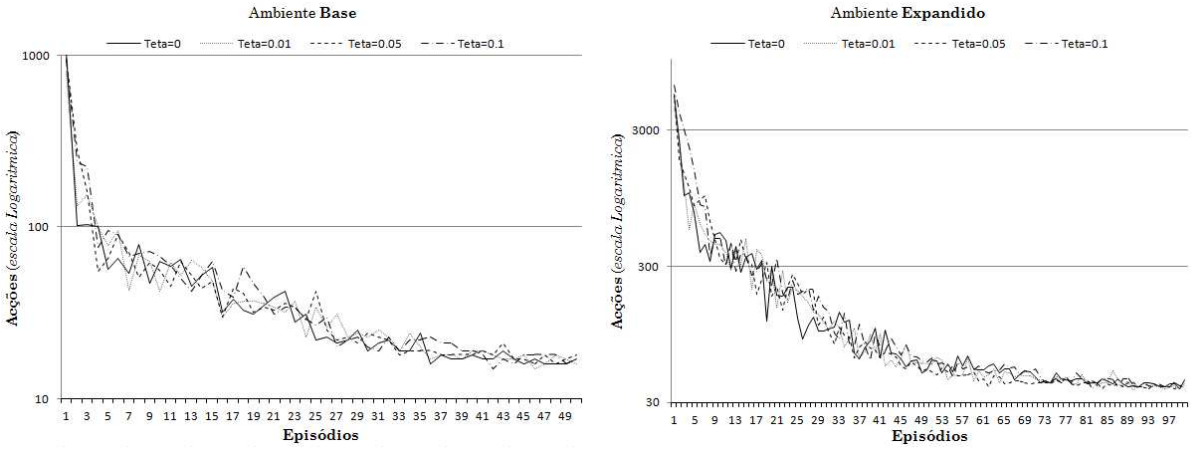


Figura 26 - *Dyna-Q com Prioritized Sweeping, variando o limiar de prioridade*

Surpreendentemente, para ambos os ambientes propostos a variação do *nível limite* $\boldsymbol{\theta}$ para inserção na fila de processamento demonstra muito pouca influência no comportamento do algoritmo. Por outro lado, este mecanismo apresenta uma *eficiência* consistentemente inferior ao Dyna-Q original (secção 5.5), como se pode constatar pelas respostas apresentadas no seguinte ensaio comparativo.

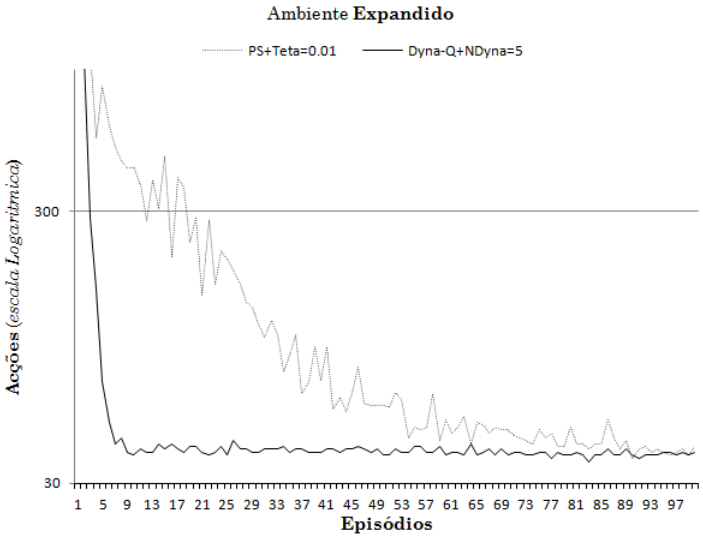


Figura 27 - *Dyna-Q sem e com Prioritized Sweeping*

A razão para tão fraco desempenho relativo encontra-se no facto de, se observarmos a evolução da *fila de processamento*, mesmo colocando-a a aceitar todos os *pares estado-acção* gerados (ou seja, com $\boldsymbol{\theta} = 0$), esta apresenta-se a *maior parte das vezes vazia a seguir às actualizações do modelo*. Assim, enquanto este algoritmo por vezes se *autolimita no número de simulações a efectuar*, o Dyna-Q faz sempre o número máximo de simulações permitido.

Foi exactamente a esta conclusão que também chegou (Kølle, 2003) ao monitorar a *fila de processamento* através de experiência idêntica efectuada sobre um pequeno labirinto **10x10**, com o algoritmo configurado para efectuar **cinco** simulações base por episódio, tal como apresentado na figura seguinte.

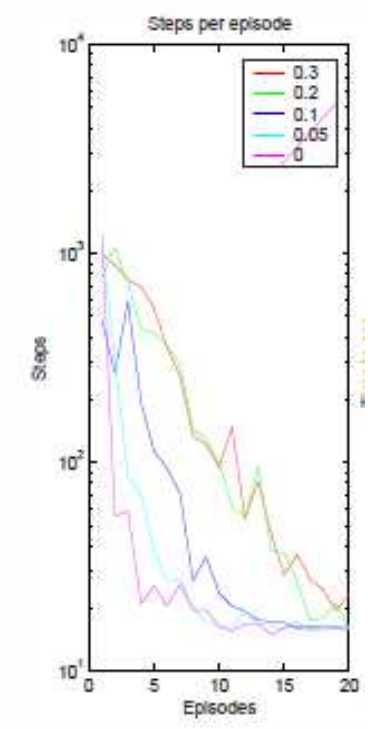


Figura 28 - *Dyna-Q c/ PS, variando o limiar de prioridade (Kølle, 2003)*

Como se observa, uma vez mais é visível a semelhança do traçado gráfico obtido por *Kølle* e aquele obtido através da plataforma PSA para ambiente idêntico (*base*), variando o valor atribuído a $\boldsymbol{\theta}$.

5.7. HAQL versus Dyna-Q

Face aos dados obtidos através dos ensaios anteriormente realizados, um dos pontos em destaque foi a rapidez de convergência do HAQL, algoritmo que sem qualquer esforço adicional de *planeamento* consegue concorrer com o Dyna-Q na *eficiência* apresentada, associando o método exploratório a uma *heurística* baseada em *Backtracking*.

Assim, realizou-se um ensaio especificamente para comparar comportamentos entre os dois algoritmos em foco, do qual são apresentados de seguida os resultados obtidos.

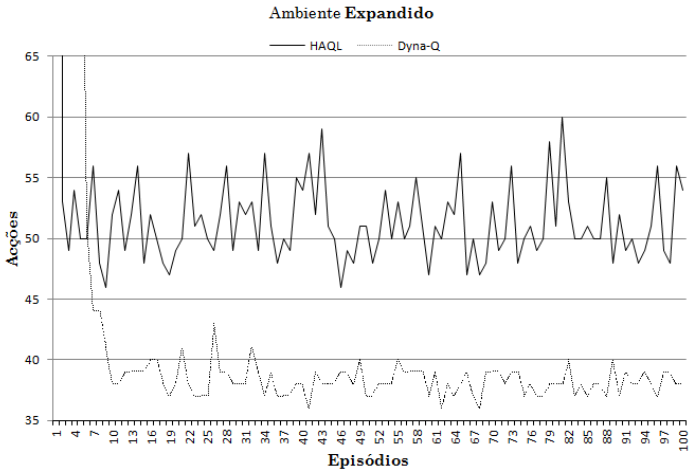


Figura 29 - *HAQL versus Dyna-Q sobre o ambiente Expandido*

De notar que, o HAQL *sem qualquer tipo de planeamento e recorrendo a uma única interacção de forma a alimentar a sua heurística*, consegue de imediato convergir para uma região fronteira da solução óptima, ficando a oscilar nesta. Por sua vez, o Dyna-Q demora um pouco mais a convergir e a apresentar comportamento semelhante, mas apresenta-se mais *eficaz*, ao ficar a oscilar mais próximo da solução óptima. Tal deve-se sobretudo ao erro imposto pela *heurística* associada ao HAQL após a sua convergência.

5.8. Análise de Resultados

Uma das primeiras características que podemos destacar é a extraordinária importância do *mecanismo de aprendizagem* Q-Learning, que se encontra, directamente ou sobre a forma de inúmeras variantes, como base de influência sobre um grande número de outros algoritmos apresentados posteriormente.

Associar alguns mecanismos ou características adicionais a um algoritmo base foi um dos caminhos encontrados na linha de evolução desta temática: a sinalização de eventos ocorridos (*Elegibility Traces*) sobre um determinado *par estado-acção* de forma a privilegiar este temporariamente na escolha de futuras evoluções sobre o ambiente, ou o associar de uma adequada *heurística* (*HAQL, Dyna-H*) à *exploração* ou mesmo ao *mecanismo de aprendizagem* de forma a potenciar estes, foram algumas das formas estudadas no âmbito deste trabalho.

No entanto, na maior parte deste tipo de soluções foi observado que os ganhos em termos de *eficiência* e/ou *eficácia* perante o algoritmo original, na prática muitas vezes se apresentavam diminutos. Tal facto muitas vezes indicia a existência de um potencial de crescimento associado ao mecanismo apresentado, requerendo investigação posterior adicional, ou pelo menos ser considerado como mais uma

possível linha de investigação futura a seguir. Nesta classe enquadram-se, por exemplo, possíveis estudos sobre novas e adequadas *heurísticas*, passíveis de caracterizar a dinâmica de *ambientes* específicos, ou novas formas de optimização do equilíbrio entre as *necessidades de exploração* e o imperativo de *aproveitamento da informação* obtida.

Uma outra linha de evolução explorada no âmbito deste trabalho foi a associação de *planeamento* (*Dyna-Q* - secção 3.2.3) à aprendizagem, e ainda o desenvolvimento de diversas técnicas adicionais na perspectiva de adquirir melhorias na implementação deste (*Dyna-H, Prioritized Sweeping*).

De facto, a associação de *planeamento ao processo de aprendizagem* revelou-se significativamente produtiva em ganhos de *eficiência*, comparando com grande parte dos algoritmos estudados que não usufruíam desta extensão. Há que ter no entanto em consideração que esta associação implica também uma *carga de processamento adicional* que é imposta, ao serem efectuadas sistematicamente simulações sobre o *espaço de estados* conhecido.

Quanto às técnicas adicionais estudadas, apresentadas como possíveis melhorias dos algoritmos com *planeamento* associado, na realidade observou-se uma vez mais que os ganhos em termos de *eficiência* e/ou *eficácia* perante o algoritmo original na resolução dos ambientes propostos, muitas vezes se apresentavam diminutos ou mesmo inexistentes. Também aqui, é possível a existência de necessidade de investigação posterior adicional.

Finalmente, propusemos um ensaio comparativo entre um mecanismo sem qualquer tipo de *planeamento* mas conduzido por uma *heurística* associada baseada em técnicas de *backtracking* (HAQL), e um mecanismo base com *planeamento* associado (Dyna-Q). Entre ambos constatou-se um excelente desempenho em termos de *eficiência*, convergindo ambos muito cedo para a vizinhança de uma solução óptima.

Analisando mais detalhadamente este ensaio, o mecanismo com uma *heurística associada* (HAQL) apresentou resultados que convergiram mais rapidamente para a vizinhança de uma *solução óptima*. Por outro lado, o mecanismo com *planeamento associado* (Dyna-Q), ou seja aquele que apresenta maior *carga de processamento* intrínseca, foi aquele que demonstrou maior *eficácia* ao convergir, permanecendo a oscilar mais perto da *solução óptima*.

Numa hipotética situação de abordagem sobre um *ambiente dinâmico* e mais *complexo*, a dependência da convergência de um mecanismo de uma *heurística* adquirida sobre um único alvo (HAQL) poder-se-á revelar problemática em termos de reconfiguração do *modelo interno* do ambiente. Neste cenário, o *planeamento* associado ao Dyna-Q poderá ser um instrumento valioso para uma mais rápida reconfiguração dos valores descritivos do mundo.

6. Conclusões

Desde 1989, quando *Watkins* apresentou o algoritmo de Q-Learning no âmbito da sua tese de doutoramento, a investigação efectuada sobre o tema de *Aprendizagem por Reforço* é intensa e profícua em ideias, tanto na crescente complexidade dos problemas possíveis de abordar, como na engenhosidade associada aos diversos tipos de soluções propostas para resolução destes.

Tanto ao nível dos *mecanismos de aprendizagem*, como ao nível dos *métodos exploratórios* associados a estes, os algoritmos estudados neste trabalho, não podendo ser considerados uma selecção exaustiva, são representativos de todo um esforço de desenvolvimento sobre esta temática, realizado pela comunidade científica ao longo das últimas duas décadas.

No decurso do trabalho realizado conseguiram-se isolar alguns dos principais algoritmos mais representativos da AR propostos ao longo das últimas duas décadas, nas vertentes *exploratória* e de *aprendizagem*. Realizou-se uma síntese teórica e uma implementação prática sobre a plataforma PSA de cada um destes e, cumprindo os objectivos enunciados, efectuaram-se diversos ensaios sobre *ambientes padrão* propostos que permitiram aferir para cada um, os graus comparativos de *eficiência* e *eficácia* apresentados.

Podemos destacar sobretudo o grande potencial demonstrado pela *associação de uma heurística apropriada ao método de exploração*, tendo sempre em consideração a dificuldade intrínseca ao processo de definição de uma *heurística* que caracterize correctamente as características relevantes de *ambientes* específicos.

Um destaque adicional para o potencial demonstrado pela associação, aos *mecanismos de aprendizagem*, de *planeamento* com base num *modelo interno* do mundo, embora o ganho adquirido imponha um custo computacional acrescido que por vezes possa não ser tolerável.

6.1. Trabalho futuro

O trabalho efectuado apresentou uma significativa abrangência dos temas em estudo, obrigando de certa forma a limitar o âmbito deste, e deixando por resolver de uma forma mais estruturada vário caminhos e possíveis soluções.

Diferentes direcções para *desenvolvimento futuro* se perspectivaram como relevantes e promissoras, decorrentes do trabalho realizado. Alguns dos caminhos revelaram-se porventura mais promissores no âmbito deste, e pela sua relevância merecem um especial destaque, sendo sintetizados nas próximas alíneas.

6.1.1. Plataforma PSA

A plataforma PSA (secção 4.1) revelou-se um instrumento fundamental para a realização do trabalho apresentado. Foi possível, devido à disponibilidade total das suas fontes, uma *evolução qualitativa* desta, nomeadamente ao nível da estabilização das interfaces com *mecanismos de aprendizagem* e *métodos exploratórios*, e ainda uma extensão para elaboração de *estatísticas*.

A qualidade conceptual desta plataforma permite-nos afirmar que, tanto em termos *didácticos* como ainda *ferramenta auxiliar de investigação*, esta poderia ser uma alternativa específica para a temática de AR, como contraponto à utilização de ferramentas genéricas, como seja o *MatLab* (MathWorks, 2011).

Para tal, a PSA poderia evoluir para novos patamares através da reconfiguração e desenvolvimento de novos módulos que, de uma forma estruturada, enriqueceriam as funcionalidades apresentadas por esta.

Algumas das melhorias sugeridas são apresentadas de seguida.

- Reforço da **estandardização**
  Para que novas extensões à PSA não se revelem como "acréscimos não estruturados", deveria existir um esforço na *normalização e documentação das interfaces* com agentes, ambientes e mecanismos. Adicionalmente, a *configuração* de valores estruturais (como seja os *reforços devolvidos* ou as *acções possíveis*) deveria revelar-se associada às estruturas respectivas, acessível globalmente e programaticamente.

- Controlo **temporal**
  Deveriam ser acrescentadas à PSA funções que, tendo em consideração os *ambientes programáticos heterogéneos* sobre os quais ela é colocada a funcionar, permitissem contabilizar o esforço desenvolvido por *unidades de tempo* efectivamente associadas a este, e simultaneamente desenvolver estatísticas sobre os valores obtidos.

- Suporte a **ambientes complexos**
  O suporte actualmente existente na PSA para incorporação de *ambientes*, embora simples e potente no seu uso e definição, encontra-se limitado nalgumas das suas vertentes que poderiam ser potenciadas, como sejam:

  - Suporte a **ambientes dinâmicos**, onde de uma forma estruturada todos os recursos possam *evoluir no tempo*, por iniciativa automática (pré-programada), por interacção com os outros recursos (ex: *uma porta a ser aberta, um bloco empurrado, etc*…), ou por acção externa (intervenção programática);
  - Suporte à **precedência**, onde seja possível definir de uma forma estruturada objectivos escalonados no tempo e obrigatoriamente precedidos de um conjunto de outras tarefas realizadas.
  - Suporte a **ambiente contínuos** (ou com muito *grandes espaços de estados* associados), onde a sua definição não seja realizada de uma forma discreta mas sim através, por exemplo, de uma *função de utilidade* composta por várias *funções características do ambiente*.

- Suporte **multi-Agente**
  Embora seja uma temática não directamente abordada neste trabalho, a possibilidade de *interacção, coexistência e partilha de conhecimento entre agentes* sobre um qualquer ambiente, é potencialmente enriquecedora das capacidades de investigação sobre a plataforma PSA e encontra-se na linha das correntes de investigação actuais sobre a temática de AR.

6.1.2. Mecanismos hierárquicos

Referidos no âmbito deste trabalho, o aparecimento dos *mecanismos hierárquicos* significou uma efectiva evolução qualitativa dos algoritmos aplicados a cenários de AR. Ao permitirem a *decomposição de problemas complexos em sub-tarefas simplificadas e hierarquicamente organizadas*, estes conseguem codificar e resolver problemas que, se abordados através dos mecanismos desprovidos dessa capacidade, muito dificilmente conseguiriam ser resolvidos de uma forma satisfatória. Pela sua potencialidade e características associadas, deveriam ser alvo de um estudo estruturado e comparativo somente dedicado a esta temática.

A melhoria na *eficácia da exploração* e uma *aprendizagem e reconfigurações mais rápidas* são atributos adicionais aos *mecanismos hierárquicos*. Embora as funções actualmente implementadas na PSA relativas à *gestão de ambientes* possam ser consideradas algo limitadoras neste âmbito, no ponto anterior foram preconizados mecanismos de evolução desta que permitiriam integrar este tipo de *mecanismos*, além de potenciar a criação de ambientes suficientemente complexos que permitissem o mecanismo alvo de estudo expressar-se em toda a sua potencialidade.

Uma possível linha adicional de investigação poderia ainda ser a possível associação entre este tipo de *mecanismos* e outros tipos de soluções já estudadas, nomeadamente a inclusão nestes de *planeamento* e/ou *heurísticas* descritoras do comportamento do ambiente.

6.1.3. Heurísticas

Foram várias as situações ao longo deste estudo onde se deparou a necessidade de que as *heurísticas* aplicadas demonstrassem maior eficácia na descrição do ambiente (*Dyna-H* (secção 3.2.3.3), *HAQL* (secção 3.1.3.1)). Esta é uma área relevante para trabalho futuro.

A escolha de uma *heurística* apropriada, que defina o comportamento ideal perante um determinado *ambiente,* não é uma questão trivial e requer um conhecimento profundo das variáveis que caracterizam esse mesmo ambiente. Este é um capítulo em aberto, quer em termos de selecção dos tipos de ambientes mais apropriados para caracterizar com este tipo de abordagem, quer em termos de parâmetros *definidores* desses mesmo ambientes que possam constituir uma *heurística* apropriada.

6.1.4. Ambientes contínuos

A caracterização de *ambientes contínuos* (ou com muito *grandes espaços de estados* associados) não pode ser realizada de forma *discreta*, como seja através de *tabelas*, e requer uma abordagem alternativa. Uma das formas possíveis é o uso de um tipo de representação baseado em aproximação de funções (Russell, et al., 2003).

Uma boa aproximação seria o uso de uma *função de utilidade*, ou seja, uma função linear ponderada de um conjunto de funções base (ou *características do ambiente*). A utilização de uma estratégia deste tipo pode permitir ao *agente de aprendizagem* uma considerável *compactação dos dados* obtidos e uma *generalização do seu conhecimento*.

Embora sendo uma área aparentemente mais árdua de abordar pela sua especificidade, o considerável potencial de utilidade associado a esta estratégia faz com que toda a investigação que decorra sobre este tema possa possivelmente ser aproveitada para enriquecimento de muitas outras áreas transversais e contribuintes da AR.

6.1.5. Ambientes de recursos limitados

Todo o trabalho desenvolvido decorreu em geral com a premissa de ter como objectivo a *maximização do conjunto de reforços retornados pelo ambiente* ao longo do tempo, de forma a alcançar rapidamente o *objectivo*. No entanto, esta estratégia somente é valida se os recursos nomeadamente computacionais de que dispomos forem muito grandes ou mesmo *ilimitados*.

O que aconteceria se os nossos recursos fossem limitados perante o desafio que nos fosse proposto? E se esses mesmos recursos necessitassem de *tempo* e/ou *certas acções de recuperação* quando atingissem níveis críticos? Qual a estratégia que se deveria aplicar? Seria esta estratégia constante ao longo do tempo?

Estas e outras questões permanecem em grande parte em aberto e deverão ser alvo de investigação adicional. *Agentes* com *recursos limitados*, na realidade retractam a maior parte das situações com que nos deparamos no mundo que nos rodeia, e são aqueles que numa última análise devemos ter em maior consideração na aplicação prática das nossas ideias.

6.2. Considerações finais

A área de AR tem sofrido uma contínua e crescente evolução ao longo dos últimos anos, com uma tendência crescente para *expansão dos domínios aos quais esta pode ser aplicada*. Sozinha ou em conjunto com outras técnicas, a sua relevância é inquestionável num mundo que se encontra de uma forma gradual a transformar processos decisórios onde a intervenção humana predomina, em processos *automáticos* e *autónomos*.

Este trabalho serviu, de alguma forma, para aferir o *estado da arte* de algumas das soluções actualmente propostas no âmbito da AR, e comparar estas entre si de uma forma uniforme e estandardizada. Dos resultados apurados e do caminho percorrido resultaram algumas sugestões de possíveis linhas de investigação futura que, pelo seu interesse e enquadramento, poderão de alguma forma ajudar a contribuir para a evolução desta temática.

7. Bibliografia

Abe Naoki [et al.] Empirical Comparison of Various Reinforcement Learning Strategies for Sequential Targeted Marketing. - Yorktown Heights, NY : I.B.M. T. J. Watson Research Center, 2002.

Bellman R. E. Dynamic Programming. - Princeton, New Jersey : Princeton University Press, 1957.

Bianchi Reinaldo A.C., Ribeiro Carlos H.C. e Costa Anna H.R. Heuristically Accelerated Q-Learning: A New Approach to Speed Up Reinforcement Learning. - São Paulo : Brasil, 2004.

Dayan Peter e Hinton Geoffrey E Feudal Reinforcement Learning. - San Francisco : Morgan Kaufmann, 1993.

Dean Thomas e Lin Shieu-Hong Decomposition Techniques for Planning in Stochastic Domains. - Providence, Rhode Island : Department of Computer Science, Brown University, 1995.

Dietterich Thomas G An Overview of MAXQ Hierarchical Reinforcement Learning. - Corvallis, Oregon : Oregon State University, 2000.

Dietterich Thomas G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. - Corvallis : Department of Computer Science, Oregon State University, 2000.

Dietterich Thomas G. The MAXQ method for hierarchical reinforcement learning. - Corvallis, Oregon : Department of Computer Science, Oregon State University, 1998.

Dutech Alain, Edmunds Tim e Kok Jelle Reinforcement Learning Benchmarks and Bake-offs II. - Vancouver, BC Canada : workshop at the 2005 NIPS conference, 2005.

Kaelbling L.P. Hierarchical reinforcement learning: Preliminary results. - San Francisco : Morgan Kaufmann, 1993.

Klopf A.H. Brain function and adaptive systems. A heterostatic theory. - Bedford, Massachusetts : Air Force Cambridge Research Laboratories, 1972.

Kølle Anders Christian The Nature of Learning - A study of Reinforcement Learning Methodology. - Copenhagen : Department of Computer Science, University of Copenhagen, 2003.

Luce R.Duncan Individual Choice Behavior: A Theoretical Analysis. - New York : John Wiley & Sons, 1959.

MathWorks MatLab [Online]. - MathWorks, 2011. - http://www.mathworks.com.

Michel Tokic Adaptive epsilon-greedy exploration in reinforcement learning based on value differences. - Weingarten : Germany, 2010.

Microsoft Excel (Office) [Online]. - Microsoft, 2011. - http://office.microsoft.com/en-us/excel/.

Moore Andrew W. e Atkeson Christopher G. Prioritized sweeping: Reinforcement Learning with less data and less time. - Cambridge : MIT Artificial Intelligence Laboratory, 1993.

Morgado Luís Integração de Emoção e Raciocinio em Agentes inteligentes. - Lisboa : FCUL, 2005.

Neruda Roman e Slusny Stanislav Performance Comparison of Two Reinforcement Learning Algorithms for Small Mobile Robots. - Prague : Academy of Sciences of the Czech Republic, 2009.

Oracle JAVA [Online]. - Oracle, 2011. - http://www.java.com.

Parsons Simon D. e Simari Gerardo I. On the Problem of Comparing Two Models for Rational Decision Making in Autonomous Agents. - New York : Department of Computer and Information Science Brooklyn College, 2003.

Peng J. e Williams R.J. Efficient Search Control in Dyna. - Massachusetts : College of Computer Science, Northeastern University, 1992.

Pollack Martha e Ringuette Marc Introducing the tileworld: experimentally evaluating agent architectures. - Menlo Park, CA : Center for the Study of Language and Information, 1990.

Rubinstein R. Y. Simulation and the Monte Carlo Method. - New York : John Wiley & Sons, Inc., 1981.

Rummery G.A. e Niranjan M. Online Q-Learning using connectionist systems. - Cambridge : Tech. report CUED/F-INFENG/TR166, Cambridge University, 1994.

Russell Stuart e Norvig Peter Artificial Intelligence: A moderne approach. - New Jersey : Prentice Hall, 2003.

Santos Matilde, Martin H. Jose Antonio e Lopez Victoria Dyna-H: a heuristic planning reinforcement learning algorithm applied to role-playing game strategy decision systems. - Madrid : Complutense University of Madrid, 2011.

Schut Martijn e Wooldridge Michael Intention reconsideration in complex environments. - New York : Proceedings of the fourth international conference on Autonomous agents, ACM, 2000.

Singh Satinder Pal Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. - Massachusetts : Department of Computer Science, University of Massachusetts, Amherst, 1992.

Sutton Richard S. e Barto Andrew G. Reinforcement Learning: An Introduction. - Cambridge, Massachusetts : MIT Press, 1998.

Sutton Richard S. e Singh Satinder P. Reinforcement Learning with Replacing Elegibility Traces. - Cambridge : Dept. of Computer Science, MIT, 1996.

Sutton Richard S. Integrated Architectures for Learning, Planning, and Reacting based on Approximating Dynamic Programming. - San Franscisco : GTE Laboratories Incorporated, 1990.

Sutton Richard S., Singh Satinder e Precup Doina Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales. - Massachusetts : Journal of Artificial Intelligence Research, University of Massachusetts, Amherst, 1998.

Tsitsiklis J. N. Asynchronous stochastic approximation and Q-learning. - Boston : Kluwer Academic Publishers, 1994.

Watkins C.J.C.H. Learning from Delayed Rewards. - Cambridge : Cambridge University, 1989.

8. Apêndices

Sob a forma de apêndices apresenta-se, entre outros, o resumo do código mais relevante utilizado na implementação dos diversos algoritmos em estudo sobre a plataforma PSA.

8.1. Dyna-H e uma "*fraca*" heurística

Em conjunto com a equipa responsável pelo desenvolvimento do mecanismo de aprendizagem *Dyna-H* (Santos, et al., 2011) foi possível confirmar que, aplicando este algoritmo em conjunto com a *heurística* proposta de *distância euclidiana entre dois pontos* sobre o *Ambiente Expandido* (secção 4.2.2), o mecanismo de aprendizagem *DynaQ* apresenta efectivamente um desempenho superior:

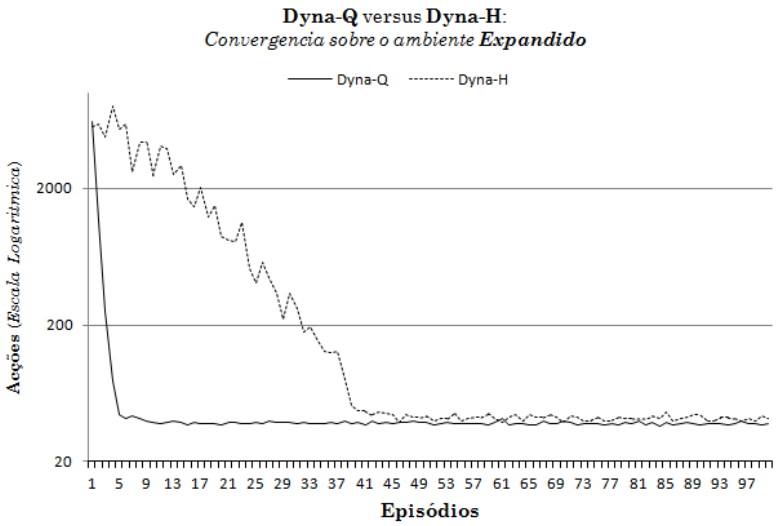


Figura 30 - *Dyna-Q versus Dyna-H, sobre o Ambiente Expandido*

Estes dados foram encontrados em simultâneo, quer no âmbito deste trabalho através da *plataforma PSA* (secção 4.1), quer pela equipa responsável pelo *Dyna-H* através de simulações efectuadas via a ferramenta *MatLab* (MathWorks, 2011).

-----Original Message-----
From: **"José Antonio Martín H." (jamartinh@fdi.ucm.es)**
Sent: Wednesday, August 24, 2011 4:11 PM
To: Joao Manuel
Subject: **Re: Dyna-H**

Hola Joao.
...

Una cosa que se me ocurre es que el heuristico utilizado (la distancia euclidea) no sea una buena información para el tipo de laberintos con los que estás probando.

El método es extremadamente sensible a la función H porque es un método basado en información heurística como lo es el A* que también puede verse afectado si se utiliza un heurístico no adecuado al problema.

He probado el laberinto de 12x12 y he comprobado que Dyna-Q converge más rápido que Dyna-H.

Una opción interesante para investigar es que tipo de heurísitco funciona en este caso.

Si encuentro algún otro detalle te aviso.

Un saludo,
Jose.

Figura 31 - *Resposta da equipa responsável pelo Dyna-H (Santos, et al., 2011)*

8.2. O problema do táxi

O autor (Dietterich, 2000) tenta no seu trabalho sobre o *mecanismo de aprendizagem MAXQQ*, explicar o conceito de *decomposição hierárquica* MAXQ através de uma aplicação prática sobre um pequeno problema idealizado para o efeito, denominado o *problema do táxi*, de seguida descrito.

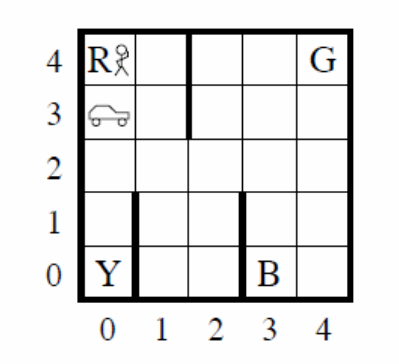


Figura 32 - *Problema do táxi: Decisão de Markov (Dietterich, 2000)*

Neste pequeno mundo definido sobre uma grelha 5x5, existe um *táxi* colocado numa qualquer quadrícula, e um *passageiro* que, encontrando-se colocado numa qualquer das quatro posições especiais (designadas por *R, G, B e Y*) pretende ser transportado pelo *táxi* para uma outra das posições designadas.

Neste domínio podem ser isoladas seis *acções primitivas*:

- Quatro acções de *navegação* (de *um* quadrado) do táxi (N,S,E e W)
- *Apanhar* o Passageiro (*Pickup*)
- *Colocar* o Passageiros (*Putdown*)

Definiram-se os seguintes valores de *recompensa*:

- **-1** por qualquer acção realizada pelo táxi (incluindo *navegação* não válida)
- **+20** adicionais se *colocar* o passageiro na posição certa
- **-10** adicionais se tentar *apanhar* um passageiro não existente, ou *colocar* este numa posição não designada

Perante este cenário, podemos agora tentar construir uma *decomposição hierárquica* MAXQ. Para tal necessitamos de identificar um conjunto de *sub-tarefas* que acreditemos serem necessárias para a resolução do problema. Como proposta de trabalho temos então:

- *Navigate(t)* – objectivo: *mover* o *táxi* da sua posição corrente para uma das quatro posições designadas, indicada no parâmetro **t**;
- *Get()* – objectivo: *mover* o *táxi* da sua posição corrente para a posição actual do *passageiro*, e *apanhar* este;

- *Put()* – objectivo: *mover* o *táxi* da sua posição corrente para a posição destino do *passageiro*, e *colocar* este;
- *Root()* – objectivo: problema finalizado

O passo final será o de definir, por cada *tarefa*, quais as *sub-tarefas* e/ou *acções primitivas* que devem ser evocadas por esta, para conseguir o seu objectivo. Tal consegue-se através do desenho de um *grafo de tarefas*, tal como a imagem seguinte ilustra:

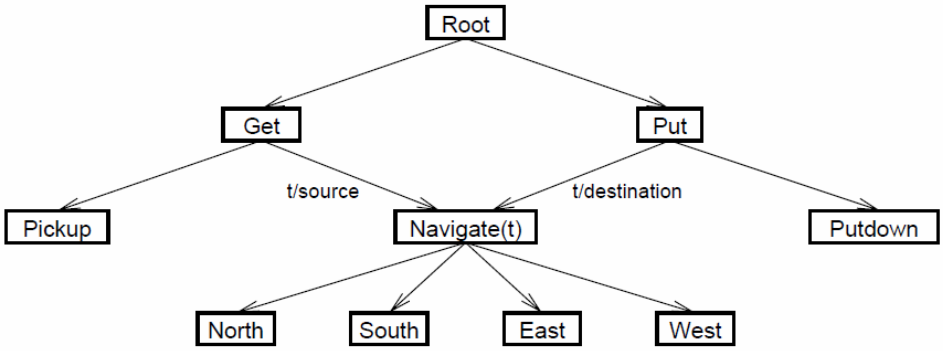


Figura 33 - *Grafo de tarefas para o problema do táxi (Dietterich, 2000)*

Assim, decompôs-se um problema de AR aparentemente complexo de definir, em pequenas *tarefas* objectivas que correspondem a rotinas a implementar e integrar num *mecanismo de aprendizagem hierárquico.*

8.3. Exploração ε-greedy

```
public class MetEpsilonGreedy<E,A> implements IAprendExplore<E,A>
{
   /** Acções definidas */
   protected A[] accoes;

   -----
   public A selAccao( E s, boolean ecran )
   {
      double epsilon = mecanismo.getParametro("Epsilon");

      if(rand.nextDouble() > epsilon)
         return getAccaoMax(s);
      else
         return getAccaoAleat();
   }
   -----
   /**
    * Obter Acção com máximo valor no estado indicado
    * @param   s  Estado actual
    * @return  Acção com valor maximo associado
    */
   protected A getAccaoMax( E s )
   {
      A maxAc = null;
      double maxVal = Double.NEGATIVE_INFINITY;
      double acVal;

      // Evitar a polarização sobre uma mesma Acção, em caso de empate de valor
      A[] laccoes = accoes;
      Collections.shuffle(Arrays.asList(laccoes));

      for(A a : laccoes) {
         acVal = mecanismo.getValMatrizQ(s, a);
         if(acVal > maxVal) {
            maxVal = acVal;
            maxAc = a;
         }
      }
      return maxAc;
   }

   /**
    * Obter acção aleatória
    * @return  Acção aleatória do grupo de possíveis
    */
   protected A getAccaoAleat( )
   {
      return accoes[rand.nextInt(accoes.length)];
   }
```

8.4. Exploração ε-greedy com ε decrescente

```
public class MetEpsilonGreedyDecreasing<E,A> extends MetEpsilonGreedy<E,A>
implements IAprendExplore<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por defeito. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */
   protected double TREPSILON = 0.9;   // Taxa de redução Gradual de Epsilon

   -----
   public void actMecanismo( E s, double _Qt0, double _Qt1)
   {
      double trepsilon = TREPSILON;

      // Memorização de Epsilon Original
      if (_memEpsilon == Double.NEGATIVE_INFINITY)
         _memEpsilon = mecanismo.getParametro("Epsilon");

      // Decrementar de forma constante a Exploração (Epsilon)
      double epsilon = mecanismo.getParametro("Epsilon");
      mecanismo.setParametro("Epsilon", epsilon * trepsilon);

      super.actMecanismo(s, _Qt0, _Qt1);
   }
```

8.5. Exploração ε-greedy VDBE-Boltzmann

```
public class MetEpsilonGreedyVDBE<E,A> extends MetEpsilonGreedy<E,A>
implements IAprendExplore<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por defeito. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */
   protected double DELTA = 1.0 / (double) accoes.length;  //[0,1]Parametro qual. Expl.
   protected double SIGMA = 0.33;   //]0,+oo[ Sensibilidade Inversa

   // Matriz Modelo para associação de Valores de Epsilon a Estados
   protected MatrizEAV<E, A, Double> _epsilonS = null;

   -----
   public A selAccao( E s )
   {
      double epsilon = _epsilonS.get(s, _aRef);

      if(rand.nextDouble() > epsilon)
         return getAccaoMax(s);
      else
         return getAccaoAleat();
   }

   public void actMecanismo( E s, double _Qt0, double _Qt1)
   {
      double delta = DELTA;
      double sigma = SIGMA;

      // Valor Espilon inicial para o estado em análise
      double epsilon0 = _epsilonS.get(s, _aRef);

      double deltaQ = Math.exp((- (Math.abs(_Qt1 - _Qt0))) / sigma);
      double Fsas = (1.0 - deltaQ) / (1.0 + deltaQ);
      double epsilon1 = delta * Fsas + (1.0 - delta) * epsilon0;

      // Actualizacao do Modelo
      _epsilonS.set(s, _aRef, epsilon1);

      super.actMecanismo(s, _Qt0, _Qt1);
   }
```

8.6. Exploração SoftMax

```
public class MetSoftMax<E,A> extends MetEpsilonGreedy<E,A>
implements IAprendExplore<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por defeito. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */
   protected double TAU = 0.01;  // Temperatura

   -----
   // Estratégia de selecção de acção "SOFTMAX"
   public A selAccao( E s )
   {
        double tau = TAU;
        double prob[] = new double[accoes.length];

        /**
       * Calculo de peso relativo do valor Q de cada acção, em
       * relação ao conjunto de todas as acções possiveis no Estado
       */
        double sumProb = 0;
        int ptaction = 0;

        for(A a : accoes) {
            prob[ptaction] = Math.exp(mecanismo.getValMatrizQ(s, a) / tau);
            sumProb += prob[ptaction++];
        }
        for (ptaction = 0; ptaction < prob.length; ptaction++) {
            prob[ptaction] = prob[ptaction] / sumProb;
        }

      /**
       * Selecção aleatoria de uma Acção, ponderada pelo seu peso
       * relativo em relação ás restantes
       */
        double rndValue = Math.random();
        double offset = 0;

        ptaction = 0;
        for(A a : accoes) {
            if ((rndValue >= offset) && (rndValue <= (offset + prob[ptaction]))) {
                return a;
            }
            offset += prob[ptaction++];
      }
        return null;
   }
```

8.7. Exploração HAQL

```
public class MetHeuristicAcceleratedQL<E,A> extends MetEpsilonGreedy<E,A>
implements IAprendExplore<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por omissão. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */

   protected double CSI = 1;          // [0,1] Parametro mult. influencia peso da Heur.
   protected double ETA = 0.1;        // [0,...] Parametro potenciador da Heurística
   protected double BKTEPISODIO = 10; // ]1,+n[ Episodio para calculo de BackTracking

   // Matriz para calculo da Heurística
   protected MatrizEAV<E, A, Double> _heuristic = null;

   -----
   public A selAccao( E s, boolean ecran )
   {
      double epsilon = mecanismo.getParametro("Epsilon");

      A aSel = null;
      if(rand.nextDouble() > epsilon)
         aSel = getAccaoHeuristica(s);
      else
         aSel = getAccaoAleat();

      // Somente para actualizacao da MatrizQ em ecran...
      if (ecran) return (aSel);

      // Memoria de caminhos percorridos
      if ((_lastS != null)&&(!_lastS.equals(s)))  // Se não existiu conflito...
         _sBacktrack.set(_lastS, _lastA, s);
      _lastS = s;
      _lastA = aSel;

      return (aSel);
   }

   -----
   // Selecção da Acção de maior valor, tendo em consideração a Heurística associada
   private A getAccaoHeuristica( E s )
   {
      double csi = CSI;

      A maxAc = null;
      double maxVal = Double.NEGATIVE_INFINITY;
      double acVal;

      // Evitar a polarização sobre uma mesma Acção, em caso de empate de valor
      A[] laccoes = accoes;
      Collections.shuffle(Arrays.asList(laccoes));

      for(A a : laccoes) {
         acVal = mecanismo.getValMatrizQ(s, a) + csi * _heuristic.get(s, a) ;
         if(acVal > maxVal) {
            maxVal = acVal;
            maxAc = a;
         }
      }
      return maxAc;
   }
```

```
/**
 * Actualização da Heurística (Mecanismo de BackTracking).
 * alvoS contem um estado final (ou já valorado heuristicamente).
 * Pretende-se que todos os estados que tenham como Destino este,
 * valorizem heuristicamente a respectiva acção
 */

private void actualizaHeuristica(E alvoS)
{
   double eta = ETA;

   // Teste de sanidade
   if (alvoS == null)
      return;

   // Procurar todos os estados (ainda não tratados) que tenham estado final=alvoS
   Vector <E> estados = _sBacktrack.getEstados();
   for(E sx : estados) {
      if (!_flagTrack.get(sx, _aRef)){   // Estado ainda não tratado?
         Vector <A> laccoes = _sBacktrack.getAccoes(sx);
         for(A ax : laccoes) {
            E SDest = _sBacktrack.get(sx, ax);
            if (SDest.equals(alvoS)){

               // Actualização da Heurística
               A aMax = getAccaoMax(sx);
               for(A acS : accoes) {
                  if (acS != ax)
                     _heuristic.set(sx, acS, 0.0);
               }
               _heuristic.set(sx, ax, (mecanismo.getValMatrizQ(sx, aMax)
                             - mecanismo.getValMatrizQ(sx, ax) + eta));

               // Chamada recursiva para propagação da actualização
               _flagTrack.set(sx, _aRef, true);
               actualizaHeuristica(sx);
            }
         }
      }
   }
}
```

8.8. Aprendizagem Q-Learning

```
public class MecQLearning<E,A> implements IAprendRef<E,A>
{
   public MatrizEAV<E,A,Double> matrizQ = null; // MatrizQ para o Ambiente actual
   protected A[] accoes; // Acções definidas

   -----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      // A Acção a contabilizar na actualização da Matriz Q
      an = getAccaoMecanismo (sn, an);

      double alfa = getParametro("Alfa");
      double gama = getParametro("Gama");

      double qsa = matrizQ.get(s, a);
      double qsnan = matrizQ.get(sn, an);

      // Actualizar matriz Q: (1 - alfa) * qsa + alfa * (r + gama * qsnan)
      double novoQsa = qsa + alfa * (r + gama * qsnan - qsa);
      matrizQ.set(s, a, novoQsa);

      // Actualizar Método de Exploração
      metodo.actMecanismo(s, qsa, novoQsa);
   }

   -----
   /**
    * Obter Acção a considerar para actualização do Mecanismo
    * @param   sn Estado seguinte
    * @param   an  Acção escolhida do estado seguinte
    * @return  Acção seleccionada para actualização
    */
   protected A getAccaoMecanismo (E sn, A an)
   {
      // QLearning: Actualiza Matriz Q com Acção valor máximo estado-alvo (Off-Policy)
      return getAccaoMax(sn);
   }

   -----
   /**
    * Obter Acção com máximo valor no estado indicado
    * @param   s  Estado actual
    * @return  Acção com valor máximo associado
    */
   protected A getAccaoMax( E s )
   {
      A maxAc = null;
      double maxVal = Double.NEGATIVE_INFINITY;
      double acVal;

      // Evitar a polarização sobre uma mesma Acção, em caso de empate de valor
      A[] laccoes = accoes;
      Collections.shuffle(Arrays.asList(laccoes));

      for(A a : laccoes) {
         acVal = getValMatrizQ(s, a);
         if(acVal > maxVal) {
            maxVal = acVal;
            maxAc = a;
         }
      }
      return maxAc;
   }
```

8.9. Aprendizagem Q-Learning λ

```
public class MecQLearningLambda<E,A> extends MecQLearning<E,A>
implements IAprendRef<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por defeito. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */
   protected double LAMBDA = 0.05;  // [0,1] Factor de rastreamento

   // Rastreamento do numero de visitas/actualizações de um par (s,a)
   protected MatrizEAV<E,A,Double> _matrizN = new MatrizEAV<E,A,Double>(0.0);

   -----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      double lambda = LAMBDA;

      // A Acção a contabilizar na actualização da Matriz Q
      an = getAccaoMecanismo (sn, an);

      double alfa = getParametro("Alfa");
      double gama = getParametro("Gama");

      double qsa = matrizQ.get(s, a);
      double qsnan = matrizQ.get(sn, an);

      // Parametro auxiliares
      double delta = r + gama * qsnan - qsa;

      // Actualizar mecanismo de rastreamento
      actualizarRastreio(s,a);

      // Para actualização do Metodo de Exploração
      double novoQsa = qsa + alfa * delta * _matrizN.get(s, a);

      // Actualizar matriz Q em todos os Estados/Acções
      Vector <E> lestados = matrizQ.getEstados();
      for(E sx : lestados) {
         Vector <A> laccoes = matrizQ.getAccoes(sx);
         for(A ax : laccoes) {
            double qsx = matrizQ.get(sx, ax);
            double nsx = _matrizN.get(sx, ax);
            double novoQsx = qsx + alfa * delta * nsx;
            matrizQ.set(sx, ax, novoQsx);

            // Actualização do Valor de Rastreamento
            nsx = gama * lambda * getValorRastreamento(sx, ax, nsx);
            _matrizN.set(sx, ax, nsx);
         }
      }

      // Actualização do método de exploração
      metodo.actMecanismo(s, qsa, novoQsa);
   }
```

```
/** ********** CONFIGURADORES DO MECANISMO *************** */

   /**
    * Obter Valor a considerar para actualização do Mecanismo de Rastreamento
    * @param   sx    Espaço em análise
    * @param   ax    Acção em análise
    * @param   nsx   Valor actual do Mecanismo de Rastreamento para o par em foco
    * @return  Valor para actualização
    */
   protected double getValorRastreamento (E sx, A ax, double nsx)
   {
      A aMax = getAccaoMax( sx );
      return ((ax == aMax) ? nsx : 0.0 );
   }

   /**
    * Actualizar matriz de rastreamento - Accumulating Traces
    * @param   s  Estado actual
    * @param   a  Acção actual
    */
   protected void actualizarRastreio (E s, A a)
   {
      double nsa = _matrizN.get(s, a) + 1;
      _matrizN.set(s, a, nsa);
   }
```

8.10. Aprendizagem Q-Learning λ (replacing)

```
public class MecQLearningLambdaReplacing<E,A> extends MecQLearningLambda<E,A>
implements IAprendRef<E,A>
{

   -----
   /**
    * Actualizar matriz de rastreamento - Replacing Traces
    * @param   s  Estado actual
    * @param   a  Acção actual
    */
   protected void actualizarRastreio (E s, A a)
   {
      Vector <A> laccoes = matrizQ.getAccoes(s);
      for(A ax : laccoes) {
         double nsa = ((ax == a) ? 1.0 : 0.0) ;
         _matrizN.set(s, ax, nsa);
      }
   }
```

8.11. Aprendizagem SARSA

```
public class MecSarsa<E,A> extends MecQLearning<E,A> implements IAprendRef<E,A>
{

   -----
   /**
    * Obter Acção a considerar para actualização do Mecanismo
    * @param   sn  Estado seguinte
    * @param   an  Acção escolhida do estado seguinte
    * @return  Acção seleccionada para actualização
    */
   protected A getAccaoMecanismo (E sn, A an)
   {
      // SARSA: Actualiza Matriz Q com valor da Acção seleccionada (ON-Policy)
      return an;
   }
```

8.12. Aprendizagem SARSA λ

```
public class MecSarsaLambda<E,A> extends MecQLearningLambda<E,A>
implements IAprendRef<E,A>
{

   -----
   /** ********** CONFIGURADORES DO MECANISMO ************** */

   /**
    * Obter Acção a considerar para actualização do Mecanismo
    * @param   sn Estado seguinte
    * @param   an Acção escolhida do estado seguinte
    * @return  Acção seleccionada para actualização
    */
   protected A getAccaoMecanismo (E sn, A an)
   {
      return an;
   }

   /**
    * Obter Valor a considerar para actualização do Mecanismo de Rastreamento
    * @param   sx    Espaço em análise
    * @param   ax    Acção em análise
    * @param   nsx   Valor actual do Mecanismo de Rastreamento para o par em foco
    * @return  Valor para actualização
    */
   protected double getValorRastreamento (E sx, A ax, double nsx)
   {
      return (nsx);
   }
```

8.13. Aprendizagem SARSA λ (replacing)

```
public class MecSarsaLambdaReplacing<E,A> extends MecSarsaLambda<E,A>
implements IAprendRef<E,A>
{

   -----
   /**
    * Actualizar matriz de rastreamento - Replacing Traces
    * @param   s  Estado actual
    * @param   a  Acção actual
    */
   protected void actualizarRastreio (E s, A a)
   {
      Vector <A> laccoes = matrizQ.getAccoes(s);
      for(A ax : laccoes) {
         double nsa = ((ax == a) ? 1.0 : 0.0) ;
         _matrizN.set(s, ax, nsa);
      }
   }
```

8.14. Aprendizagem Dyna-Q

```
public class MecDynaQ<E,A> extends MecQLearning<E, A> implements IAprendRef<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por defeito. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */
   protected double NDYNA = 10; //200; // [0,+n[ Numero Interacções efectuadas DYNA

   // Matrizes para definição de Modelo do Ambiente
   protected MatrizEAV<E, A, E> _modT;
   protected MatrizEAV<E, A, Double> _modR;

   -----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      double ndyna = NDYNA;

      // Actualizar da MatrizQ
      actualizarDiferido(s, a, sn, an, r);

      // Actualizar modelo
      _modT.set(s, a, sn);
      _modR.set(s, a, r);

      // Simulação interna
      Vector<E> estados = _modT.getEstados();

      for(int n = 0; n < (int) ndyna; n++) {
         E sr = estados.elementAt(_rand.nextInt(estados.size()));
         Vector<A> accoes = _modT.getAccoes(sr);
         A ar = accoes.elementAt(_rand.nextInt(accoes.size()));

         E srn = _modT.get(sr, ar);
         double rr = _modR.get(sr, ar);
         A arn = getAccaoMax(srn);

         // Actualizar da MatrizQ
         actualizarDiferido(sr, ar, srn, arn, rr);
      }
   }
```

8.15. Aprendizagem Dyna-Q com varrimento priorizado

```
public class MecDynaQPrioritizedSweep<E,A> extends MecDynaQ<E,A>
implements IAprendRef<E,A>
{
   /**
    * Variáveis contempladas no Algoritmo mas utilizadas somente com valores
    * por defeito. Poderão posteriormente ser transformadas em Parâmetros
    * globais (do Mecanismo).
    */
   protected double TETA = 0.01;    // [0, +n] Threshold (Limite) que insere na Queue

   // Para implementação de PQueue
   Comparator<ParEstadoAccao> _comparator = new ParEstadoAccaoComparator();
   PriorityQueue<ParEstadoAccao> _pQueue = null;

   -----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      double teta = TETA;
      double ndyna = NDYNA;

      double gama = getParametro("Gama");

      ParEstadoAccao<E,A> pea = null;

      // Actualizar modelo
      _modT.set(s, a, sn);
      _modR.set(s, a, r);

      // Calculo de Prioridade
      double qsa = matrizQ.get(s, a);
      an = getAccaoMax(sn);
      double qsnan = matrizQ.get(sn, an);
      double priority = Math.abs(r + gama * qsnan - qsa);

      // Verificação para inserção na Queue
      if (priority > teta){
         pea = new ParEstadoAccao();
         pea.prioridade = priority;
         pea.estado = s;
         pea.accao = a;
         _pQueue.add(pea);
      }

      // Processamento dos pares (s,a) em Queue
      for(int n = 0; n < (int) ndyna; n++) {
         if (_pQueue.size() == 0)
            break;

         pea = _pQueue.remove();
         s = (E) pea.estado;
         a = (A) pea.accao;

         // Actualização da MatrizQ actual
         sn = _modT.get(s, a);
         r = _modR.get(s, a);
         an = getAccaoMax(sn);
         super.actualizarDiferido(s, a, sn, an, r);

         // Estado origem (s) passa a ser destino/alvo (na pesquisa)
         a = getAccaoMax(s);
         qsa = matrizQ.get(s, a);
```

```
      // Para todos os pares (sOri,aOri) que tenham destino=s (modT(sOri,aOri) -> s)
       Vector<E> estados = _modT.getEstados();
       for (E sOri : estados) {
          Vector<A> accoes = _modT.getAccoes(sOri);
          for (A aOri : accoes) {
             E sDest = _modT.get(sOri, aOri);

             // O destino de (sOri,aOri) é o estado s ?
             if (sDest.equals(s)){
                r = _modR.get(sOri,aOri);
                double qsOri = matrizQ.get(sOri, aOri);

                priority = Math.abs(r + gama * qsa - qsOri);

                // Verificação para inserção na Queue
                if (priority > teta){
                   pea = new ParEstadoAccao();
                   pea.prioridade = priority;
                   pea.estado = sOri;
                   pea.accao = aOri;
                   _pQueue.add(pea);
                }
             }
          }
       }
    }
}
```

8.16. Aprendizagem Dyna-H

```
public class MecDynaQHeuristic<E,A> extends MecDynaQ<E,A> implements IAprendRef<E,A>
{
   // Variável auxiliar para memorização do Estado Alvo
   protected E _alvo = null;

   -----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      double ndyna = NDYNA;

      // Actualizar Modelo
      super.actualizarDiferido(s, a, sn, an, r);

      // Actualizar modelo
      _modT.set(s, a, sn);
      _modR.set(s, a, r);

      // Simulação interna
      Vector<E> estados = _modT.getEstados();
      for(int n = 0; n < (int) ndyna; n++) {
         E sr = s;
         A ar = Hgreedy(sr);

         if ((ar == null) || (_modT.get(sr, ar) == null)){
            sr = estados.elementAt(_rand.nextInt(estados.size()));
            Vector<A> accoes = _modT.getAccoes(sr);
            ar = accoes.elementAt(_rand.nextInt(accoes.size()));
         }
         E srn = _modT.get(sr, ar);
         double rr = _modR.get(sr, ar);
         A arn = getAccaoMax(srn);

         // Actualização do Modelo
         super.actualizarDiferido(sr, ar, srn, arn, rr);
         s = srn;
      }
   }

   -----
   /**
    * Obter Acção através de Heurística associada
    * @param   s  Estado actual
    * @return  Acção com valor maximo associado
    */
   protected A Hgreedy( E s )
   {
      // Alvo ainda não angariado...
      if (_alvo == null)
         return null;

      // Coordenadas do Alvo
      Coordenada sAlvo = (Coordenada) _alvo;
      int XAlvo = sAlvo.getX();
      int YAlvo = sAlvo.getY();

      // Heurística: Acção que dê origem a um Estado com MAIOR distância do Alvo
      A maxAc = null;
      double maxVal = Double.NEGATIVE_INFINITY;

      Vector <A> laccoes = _modT.getAccoes(s);

      // Evitar a polarização sobre uma mesma Acção, em caso de empate de valor
      Collections.shuffle(laccoes);
```

```
      for(A ax : laccoes) {
         Coordenada sDest = (Coordenada) _modT.get(s, ax);
         int XDest = sDest.getX();
         int YDest = sDest.getY();

         // Distância Euclidiana a duas dimensões, entre as coordenadas
         double acVal = Math.sqrt(Math.pow((double)(XAlvo - XDest), 2) +
                        Math.pow((double)(YAlvo - YDest), 2));

         // Selecção da MAIOR distância ao Alvo!
         if(acVal > maxVal) {
            maxVal = acVal;
            maxAc = ax;
         }
      }
      return (maxAc);
   }
```

8.17. Aprendizagem MaxQQ

```
public class MecMaxQQ<E,A> extends MecHSMQ<E, A> implements IAprendRef<E,A>
{

   /**
    *  MatrizV, modelo do Ambiente associado a Acções Primitivas
    */
   public MatrizEAV<E,A,Double> _matrizV = null;

   /**
    * Quanto ao modelo associado a acções SubTasks, este seria implementado
    * através de matrizesC, cada uma delas associada individualmente a uma
    * subtask (p):
    * -> C(p,s,a)
    * No entanto, ao não existirem SubTasks no nosso ambiente, esta ver-se-á
    * reduzida a uma única MatrizC, comum a toda a solução:
    * C(p,s,a) -> C(s,a)
    */
   public MatrizEAV<E,A,Double> _matrizC = null;

   -----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      /**
       * A MatrizV representa a contribuição das Acções Primitivas (Folhas)
       * para o conhecimento do Ambiente. Um olhar mais atento revela-nos que
       * esta contribuição nada mais é do que a formula de actualização Q
       * onde o novo conhecimento é exclusivamente cedido pela integração da
       * recompensa retornada:
       * V(a,s) = (1 - alfa) * V(a,s) + alfa * r
       */
      double alfa = getParametro("Alfa");
      double Vsa = _matrizV.get(s, a);
      double novoVsa = (1 - alfa) * Vsa + alfa * r;
      _matrizV.set(s, a, novoVsa);

      /**
       * A MatrizC representa a contribuição das SubTasks para o conhecimento
       * do Ambiente. Esta componente somente faz sentido se existir o
       * tratamento de SubTasks (o que não é o caso no nosso exemplo). No
       * entanto, tem implícita a integração do conhecimento decorrente da
       * observação do estado seguinte, o que não pode ser descurado:
       * C(p,s,a) = (1 - alfa) * C(p,s,a) + alfa * maxa'[V(a',s') + C(p,s',a')]
       * Com as simplificações decorrentes do ambiente, ficamos com:
       * C(s,a) = (1 - alfa) * C(s,a) + alfa * maxa'[V(a',s') + C(s',a')]
       */
      double Csa = _matrizC.get(s, a);
      double novoCsa = (1 - alfa) * Csa  + alfa * getValorMaxVC(sn);
      _matrizC.set(s, a, novoCsa);

      /**
       * No algoritmo, a actualização do valor da MatrizQ associadaàSubTask
       * actual é dado pela Formula:
       * Q(p,s,a) = V(s,a) + C(p,s,a)
       * No nosso caso, sendo as matrizes Q e C únicas para a solução, a
       * formula fica reduzida a:
       * Q(s,a) = V(s,a) + C(s,a)
       */
      double qsa = matrizQ.get(s, a);
      double novoQ = novoVsa + novoCsa;
      matrizQ.set(s, a, novoQ);

      // Actualizar Metodo de Exploração
      metodo.actMecanismo(s, qsa, novoQ);
   }

   -----
```

```
/**
 * Obter Valor da Acção com maximo valor no estado indicado
 * @param   s  Estado actual
 * @return  Valor da Acção com valor maximo associado (Matrizes V e C)
 */
protected double getValorMaxVC( E s )
{
   double maxVal = Double.NEGATIVE_INFINITY;
   double acVal;

   // Evitar a polarização sobre uma mesma Acção, em caso de empate de valor
   A[] laccoes = accoes;
   Collections.shuffle(Arrays.asList(laccoes));

   for(A a : laccoes) {
      acVal  = _matrizC.get(s, a);
      acVal += _matrizV.get(s, a);
      if(acVal > maxVal) {
         maxVal = acVal;
      }
   }
   return maxVal;
}
```

8.18. Aprendizagem HSMQ

```
public class MecHSMQ<E,A> extends MecQLearning<E, A> implements IAprendRef<E,A>
{
   /**
    * No algoritmo original, a matriz Q com a imagem do Ambiente actual não é
    * única para toda a solução, mas sim em numero igual ás SubTaks (p)
    * existentes, cada uma destas com uma Matriz associada:
    * -> Q(p,s,a)
    */

-----
   public void actualizar( E s, A a, E sn, A an, double r )
   {
      /**
       * No caso de uma SubTask, a função acumularia as diversas recompensas
       * numa variável local (TotalReward), a qual seria o seu valor de
       * retorno. Enquanto a SubTask não cumprisse as suas condições de
       * términus, seria executada ciclicamente uma sequencia de:
       *    - Selecção da Acção (evocação do método de exploração)
       *    - Tratamento da Acção (execução de primitiva ou evocação de subTask)
       *    - Acumulação da Recompensa (obtida ou retornada)
       *    - Actualização do Modelo (Matriz Q da SubTask)
       *
       * Neste caso especifico (cenário minimalista), esta rotina
       * apresenta-se como SubTask única, evocada ciclicamente pelo próprio
       * mecanismo de simulação, sendo-lhe já passados como parâmetros todos
       * os dados acerca da Acção seleccionada (Primitiva), sua execução e
       * recompensa associada.
       *
       * Não existindo SubTasks adicionais, não existirá também qualquer
       * evocação recursiva e a matriz Q com a informação de Ambiente é única
       * e comum a todo o Algoritmo. Tal permite-nos utilizar a actualização
       * da MatrizQ comum implementada pelo algoritmo QLearning (Super).
       */

      /**
       * Actualização da MatrizQ.
       * De notar que não existindo quaisquer SubTasks adicionais, Q(p,s,a)
       * fica reduzido a Q(s,a), fazendo com que o algoritmo se converta num
       * QLearning (considerando o parâmetro Gama unitário (= 1)):
       */
      setParametro("Gama",1.0);
      super.actualizar(s, a, sn, an, r);
   }
```