Reinaldo Augusto da Costa Bianchi

Uso de Heurísticas para a Aceleração do Aprendizado por Reforço

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

São Paulo
2004

Reinaldo Augusto da Costa Bianchi

Uso de Heurísticas para a Aceleração do Aprendizado por Reforço

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

Área de concentração:
Sistemas Digitais

Orientadora:
Profa. Dra. Anna Helena Reali Costa

São Paulo
2004

Ficha Catalográfica

Bianchi, Reinaldo Augusto da Costa
Uso de Heurísticas para a Aceleração do Aprendizado por Reforço / Reinaldo Augusto da Costa Bianchi – São Paulo, 2004.
Edição Revisada. 174 p.

Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Inteligência Artificial. 2. Aprendizado Computacional
3. Robótica Móvel Inteligente. 4. Robôs.
I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t

Life can only be understood going backwards, but it must be lived going forwards.

(Kierkegaard)

Agradecimentos

Aos amigos e companheiros do Laboratório de Técnicas Inteligentes da Escola Politécnica da USP: Alexandre Simões, Fabrício Barth, Gustavo Lugo, Júlio Kawai, Jomi Hübner, Luís Ferreira Filho, Márcio Seixas, Maria das Graças Marietto, Rafael Pacheco e Valguima Odakura que colaboraram de inúmeras maneiras durante esse período. Em especial à Diana Adamatti, Regina Cantele e Valdinei Freire da Silva pela colaboração na finalização deste trabalho.

Aos professores Jaime Simão Sichman e José Jaime da Cruz da Escola Politécnica da USP, que estavam sempre dispostos a ajudar. Às professoras Leliane Nunes de Barros do Instituto de Matemática e Estatística da USP e Lúcia Franco da Universidade Federal de Itajubá. Ao Professor Renê Pegoraro, da Universidade Estadual Paulista. Aos professores do Centro Universitário da FEI, que incentivaram este projeto: Márcio Rillo, João Martino, Fabrício Leonardi, Flávio Tonidandel, Alessandro La Neve, Aldo Belardi e Devair Arrabaça.

Ao professor Carlos Henrique Costa Ribeiro, do Instituto Tecnológico de Aeronáutica, que acompanhou este trabalho sugerindo melhorias, discutindo o conteúdo e indicando os melhores caminhos a seguir.

À minha orientadora, Anna Helena Reali Costa, pela amizade.

À minha família, pelo suporte e incentivo constante.

À Mayra.

Resumo

Este trabalho propõe uma nova classe de algoritmos que permite o uso de heurísticas para aceleração do aprendizado por reforço. Esta classe de algoritmos, denominada "Aprendizado Acelerado por Heurísticas" (*"Heuristically Accelerated Learning"* – HAL), é formalizada por Processos Markovianos de Decisão, introduzindo uma função heurística $\mathcal{H}$ para influenciar o agente na escolha de suas ações, durante o aprendizado. A heurística é usada somente para a escolha da ação a ser tomada, não modificando o funcionamento do algoritmo de aprendizado por reforço e preservando muitas de suas propriedades.

As heurísticas utilizadas nos HALs podem ser definidas a partir de conhecimento prévio sobre o domínio ou extraídas, em tempo de execução, de indícios que existem no próprio processo de aprendizagem. No primeiro caso, a heurística é definida a partir de casos previamente aprendidos ou definida *ad hoc*. No segundo caso são utilizados métodos automáticos de extração da função heurística $\mathcal{H}$ chamados "Heurística a partir de X" (*"Heuristic from X"*).

Para validar este trabalho são propostos diversos algoritmos, entre os quais, o "*Q–Learning* Acelerado por Heurísticas" (*Heuristically Accelerated Q–Learning* – HAQL), que implementa um HAL estendendo o conhecido algoritmo *Q–Learning*, e métodos de extração da função heurística que podem ser usados por ele. São apresentados experimentos utilizando os algoritmos acelerados por heurísticas para solucionar problemas em diversos domínios – sendo o mais importante o de navegação robótica – e as heurísticas (pré-definidas ou extraídas) que foram usadas. Os resultados experimentais permitem concluir que mesmo uma heurística muito simples resulta em um aumento significativo do desempenho do algoritmo de aprendizado de reforço utilizado.

Abstract

This work presents a new class of algorithms that allows the use of heuristics to speed up Reinforcement Learning (RL) algorithms. This class of algorithms, called "Heuristically Accelerated Learning" (HAL) is modeled using a convenient mathematical formalism known as Markov Decision Processes. To model the HALs a heuristic function $\mathcal{H}$ that influences the choice of the actions by the agent during its learning is defined. As the heuristic is used only when choosing the action to be taken, the RL algorithm operation is not modified and many proprieties of the RL algorithms are preserved.

The heuristic used in the HALs can be defined from previous knowledge about the domain or be extracted from clues that exist in the learning process itself. In the first case, the heuristic is defined from previously learned cases or is defined *ad hoc*. In the second case, automatic methods for the extraction of the heuristic function $\mathcal{H}$ called "Heuristic from X" are used.

A new algorithm called Heuristically Accelerated Q–Learning is proposed, among others, to validate this work. It implements a HAL by extending the well-known RL algorithm Q–Learning. Experiments that use the heuristically accelerated algorithms to solve problems in a number of domains – including robotic navigation – are presented. The experimental results allow to conclude that even a very simple heuristic results in a significant performance increase in the used reinforcement learning algorithm.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

Lista de Símbolos

1 Introdução **1**

1.1 Domínio 2

1.2 Objetivos e contribuições 3

1.3 Histórico 4

1.4 Organização do trabalho 6

2 Aprendizado por Reforço **8**

2.1 O Problema do Aprendizado 8

2.2 Processos Markovianos de Decisão 9

2.3 MDPs e Programação Dinâmica 11

2.4 *Q–Learning* 15

2.5 O método das Diferenças Temporais – *TD-Learning* 17

2.6 SARSA 20

2.7 Jogos de Markov e o *Minimax–Q* 21

2.8 Processos Markovianos de Decisão Generalizados e o *Q–Learning* Generalizado 24

2.9 Discussão 26

3 Aceleração do aprendizado por reforço 28

3.1 Aceleração por Generalização 29

3.1.1 Generalização Temporal 29

3.1.2 Generalização Espacial 30

3.2 Aceleração por Abstração 32

3.2.1 Abstração Temporal 32

3.2.2 Abstração Espacial 32

3.3 Aceleração por Distribuição 37

3.3.1 *Q–Learning* Distribuído 40

3.3.2 Dyna–Q Multiagente 42

3.3.3 Otimização por Colônia de Formigas 44

3.4 Aceleração Baseada em Casos 49

3.5 Discussão 55

4 Uso de heurísticas para aceleração do aprendizado por reforço 56

4.1 Introdução 56

4.2 A função heurística $\mathcal{H}$ 60

4.3 "Aprendizado Acelerado por Heurísticas" como uma busca informada 67

4.4 Os métodos "Heurística a partir de X" 69

4.4.1 Extração da Estrutura 71

4.4.2 Construção da Heurística 72

4.5 O algoritmo *Q–Learning* Acelerado por Heurísticas 73

4.6 Resumo 78

5 Experimentos no domínio de navegação robótica 79

5.1 O domínio dos robôs móveis 79
5.2 Uso de métodos "Heurística a partir de X" em tempo de execução 80
5.2.1 Extração de estrutura 81
5.2.2 Construção da heurística 85
5.2.3 Discussão 85
5.3 Experimentos com o algoritmo HAQL 88
5.3.1 Navegação robótica em ambiente desconhecido 89
5.3.2 Navegação robótica em ambiente pouco modificado 93
5.3.3 Navegação robótica com reposicionamento da meta 100
5.3.4 Discussão 104
5.4 Simulação de um robô real utilizando o ambiente Saphira 104
5.5 Resumo 115

6 Utilização de heurísticas *a priori* para aceleração do aprendizado 116
6.1 O problema do Carro na Montanha utilizando HA–SARSA(λ) . . 116
6.2 O problema do Pêndulo Invertido utilizando HATD(λ) 128
6.3 O problema do Caixeiro Viajante utilizando HADQL 131
6.4 Futebol de Robôs utilizando HA–Minimax–Q 135
6.5 Resumo 142

7 Conclusão e trabalhos futuros 143

Anexos 148

A Estudo da convergência da política e da função valor 148

B Estudo da relação entre a qualidade da heurística e a evolução do aprendizado 152

C Evolução das visitas em um ambiente de grandes dimensões **155**

D Evolução da função valor **161**

Referências Bibliográficas **166**

Índice Remissivo **174**

Apêndices i

I Resumo sobre estatística **i**

I.1 O teste t de Student i

I.2 Erros médios ii

Lista de Figuras

2.1 Ciclo Percepção-Ação. 9

3.1 Exemplo de camadas de telhas em um CMAC para um problema unidimensional. 34

3.2 Exemplo das camadas de telhas em um CMAC. 34

3.3 Um exemplo de espaço de estados discretizado (esquerda) e a árvore-kd correspondente (MUNOS; MOORE, 2002, pg. 295). . . . 36

3.4 Refinamentos sucessivos de um espaço discretizado. (MUNOS; MOORE, 2002, pg. 300). 37

3.5 O espaço discretizado utilizando o critério da variância para o problema do carro na montanha (MUNOS; MOORE, 2002, pg. 302). 37

3.6 O grafo que identifica uma sub-tarefa (esquerda) e os valores da função que é a solução para a sub-tarefa (direita) (DRUMMOND, 2002). 49

3.7 Sala com paredes proposta por Drummond (2002), onde o quadrado marca a meta a atingir. 51

3.8 A função valor obtida utilizando *Q–Learning* apresenta grandes descontinuidades (área escura) e o grafo construído para a sala superior esquerda (DRUMMOND, 2002, p. 62). 51

3.9 Um ambiente com duas salas (DRUMMOND, 2002, p. 66). 52

3.10 A função valor para o ambiente da figura 3.9 (DRUMMOND, 2002, p. 67). 53

3.11 O gradiente da função valor da figura 3.10, o polígono extraído e o grafo construído (à esquerda). A expansão da serpente até a acomodação (à direita) (DRUMMOND, 2002, p. 71). 53

3.12 Representações gráficas de uma solução mostrando as sub-tarefas do problema da figura 3.7 (DRUMMOND, 2002, p. 63). 54

3.13 A extração e reutilização de uma função armazenada na base de casos. A função é rotacionada e ampliada para encaixar na nova sub-tarefa (DRUMMOND, 2002, p. 64). 54

4.1 Estando no estado S e desejando ir para o estado Sd, o valor de $H(s, a1)$ da ação que leva à $Sd = 0,21$ enquanto para as outras ações é nulo. 66

4.2 Se mover de s_1 diretamente para s_3 fizer parte da política ótima, então H é admissível quando $H(s_1, a_3) \geq H(s_1, a_2)$. 68

4.3 Esquema geral dos métodos "Heurística a partir de X". 71

4.4 O estado s_1 possui ambos os valores, máximo e mínimo, para a função valor-ação Q (a execução da ação a_2 em qualquer estado sempre recebe o reforço negativo mínimo e a execução da ação a_1 em s_1 recebe o reforço máximo). 77

5.1 Sala com paredes (representadas pelas linhas escuras) discretizada em uma grade de posições (representados pelas linhas mais suaves). 80

5.2 Política ótima para um robô móvel em um ambiente com 25 x 25 posições e algumas paredes. Setas duplas significam que, em uma mesma posição, não faz diferença qual das duas ações a tomar. . . 81

5.3 Estruturas geradas por quatro métodos de extração de estrutura ao final do centésimo episódio de treinamento. 83

5.4 A imagem da tabela $V(s_t)$ e o seu gradiente ao final do centésimo episódio de treinamento (células mais claras indicam maiores valores). 84

5.5 A imagem da tabela $V(s_t)$, o seu gradiente e o mapa criado ao final do centésimo episódio de treinamento para um ambiente com paredes delgadas. 86

5.6 Heurísticas construídas utilizando o método "Heurística a partir de X" a partir das estruturas geradas pelos métodos de extração de estrutura, apresentadas na figura 5.3 87

5.7 Salas com paredes (representadas pelas linhas claras) usadas por Drummond (2002), onde a meta a atingir se encontra em um dos cantos. 90

5.8 Resultado para a aceleração em um ambiente desconhecido, ao final do décimo episódio, utilizando "Heurística a partir de Exploração", com barras de erro (inferior em monolog). 91

5.9 Resultado do teste t de Student para um ambiente desconhecido (inferior em monolog). 92

5.10 O ambiente usado para definir a heurística (a) e o ambiente onde ela é utilizada (b). 94

5.11 A heurística gerada para o ambiente da figura 5.10-a. 96

5.12 A política ótima para o ambiente da figura 5.10-b. 97

5.13 Resultado para a aceleração em um ambiente modificado, ao final do décimo episódio, utilizando "Heurística a partir de Exploração" (inferior com barras de erro). 98

5.14 Resultado do teste t de Student para aceleração na 10.^{a} iteração, em um ambiente modificado. 99

5.15 Resultado do aprendizado quando a meta é reposicionada no 5000.^{o} episódio, utilizando "Heurística a partir de Exploração" no HAQL (em escala monolog na parte superior e ampliado na inferior). . . 101

5.16 Resultado do aprendizado quando a meta é reposicionada no 5000.^{o} episódio, utilizando "Heurística a partir de Exploração" no HAQL. Com barras de erro (inferior em monolog). 102

5.17 Resultado do teste t de Student para reposicionamento da meta (inferior em monolog). 103

5.18 A plataforma Saphira 8.0. A figura superior apresenta a tela do aplicativo (com a posição do robô no plano de referência) e a inferior, o monitor do simulador (apresentando a posição real do robô). 107

5.19 Localização Monte Carlo: os pontos vermelhos indicam a posição das partículas, cada uma definindo uma posição provável do robô. A incerteza na posição do robô é menor na imagem superior do que nas inferiores. 108

5.20 O robô atravessando uma parede – no plano de referência – devido ao erro de localização. 110

5.21 Número de visitas (branco indica um número maior de visitas) e o esboço do mapa criado utilizando o método "Estrutura a partir de Exploração" para o ambiente Saphira. 111

5.22 A heurística criada a partir do esboço do mapa apresentado na figura 5.21. 112

5.23 Resultado do uso da aceleração no quinto episódio usando o algoritmo HAQL no ambiente de simulação Saphira. 113

5.24 Caminhos percorridos pelo robô utilizando o *Q–Learning* (superior) e o HAQL (inferior) no ambiente de simulação Saphira 8.0. 114

6.1 Descrição do problema do Carro na Montanha. 117

6.2 Tabela da função valor para o problema do Carro na Montanha (superior em 3D, inferior em curvas de nível). 118

6.3 Número de passos necessários para atingir a meta para os algoritmos *Q–Learning*, SARSA(λ), HAQL e HAS(λ) para o problema do Carro na Montanha (inferior em monolog). 122

6.4 O reforço recebido para os algoritmos SARSA(λ), HAQL e HAS(λ) para o problema do Carro na Montanha (inferior com barras de erros). 123

6.5 Caminhos realizados no espaço de estados pelos algoritmos SARSA(λ) (superior) e HAS(λ) (inferior) para o problema do Carro na Montanha. 124

6.6 Caminhos finais realizados no espaço de estados para o problema do Carro na Montanha. 125

6.7 Comparação para o uso de vários valores de heurística no HAS(λ) para o problema do Carro na Montanha (Número de passos na figura superior e reforço recebido na inferior). 126

6.8 Comparação entre algoritmos SARSA(λ) e HAS(λ) para o problema do Carro na Montanha. A figura superior mostra o resultado com barras de erros e a inferior mostra o resultado do teste t de Student. 127

6.9 O problema do Pêndulo Invertido. 128

6.10 Comparação entre os algoritmos DQL, ACO e HADQL para o Problema do Caixeiro Viajante kroA100 (inferior com barras de erros). 134

6.11 O ambiente proposto por Littman (1994). A figura mostra a posição inicial dos agentes. 137

6.12 A heurística utilizada para o ambiente da figura 6.11. As setas indicam a ação a tomar. 137

6.13 Resultados do saldo de gols para os algoritmos Minimax–Q e HAMMQ contra um jogador com movimentação aleatória para o Futebol de Robôs de Littman (com barras de erro na figura inferior). 139

6.14 Resultados do saldo de gols para os algoritmos Minimax–Q e HAMMQ contra um agente utilizando o Minimax–Q (com barras de erro na figura inferior). 140

6.15 Resultados do teste t de Student entre os algoritmos Minimax–Q e HAMMQ, treinando contra um agente com movimentação aleatória (superior) e contra agente utilizando o Minimax–Q (inferior). 141

A.1 Evolução da diferença da função valor e do valor calculado a partir da política utilizando Programação Dinâmica. Gráfico Normalizado. 149

A.2 A convergência da política mais provável para a sala ao lado direito da figura 5.2. O eixo x mostra o episódio de treinamento e o y, a ação tomada (N, S, E, W). 151

B.1 Resultado do uso da aceleração com a heurística correta no centésimo episódio usando o algoritmo HAQL. 153

B.2 Resultado do uso da aceleração com uma heurística errada no centésimo episódio usando o algoritmo HAQL. 154

C.1 Planta do 2.^{o} andar da mansão de Bailicroft (SUMMERS, 2001), onde a área em preto corresponde às paredes e a em branco, ao espaço por onde o robô pode se mover. 156

C.2 Visitas a novos estados (ampliado na figura inferior). 157

C.3 Mapa criado a partir da exploração. 158

C.4 Resultado para a aceleração no 20.^{o} episódio utilizando "Heurística a partir de Exploração" com barras de erro (em monolog), para o ambiente Bailicroft. 159

C.5 Resultado do teste t de Student para a aceleração no 20.^{o} episódio no ambiente Bailicroft. 160

D.1 Evolução da raiz do erro quadrático médio (RMSE) entre a função valor do algoritmo (*Q–Learning* ou HAQL) e o valor V^*, em relação ao total de passos (ampliado na figura inferior). 162

D.2 Função valor gerada pelo *Q–Learning* (superior) e pelo HAQL (inferior) ao final do 20×10^8 passo, utilizando o reforço positivo igual a 10 ao atingir a meta. 163

D.3 Evolução da raiz do erro quadrático médio (RMSE) entre a função valor do HAQL e o V^*, para diversos valores de recompensa recebida ao atingir a meta no final de um episódio. 165

Lista de Tabelas

2.1 O algoritmo de Iteração de Valor (BERTSEKAS, 1987). 13

2.2 O algoritmo de Iteração de Política (BERTSEKAS, 1987, p. 199). . 14

2.3 O algoritmo *Q–Learning* (WATKINS, 1989). 18

2.4 O algoritmo Genérico $TD(\lambda)$ (SUTTON; BARTO, 1998). 21

2.5 O algoritmo Minimax–Q (LITTMAN, 1994). 23

2.6 Alguns modelos de MDPs Generalizados e suas especificações (SZEPESVÁRI; LITTMAN, 1996, Teorema 3). 25

3.1 O algoritmo *QS* (RIBEIRO, 1998, p. 53). 31

3.2 O algoritmo *Q–Learning Distribuído* (ROMERO; MORALES, 2000). 41

3.3 O algoritmo Ant Colony System (DORIGO; GAMBARDELLA, 1997). 48

4.1 As três hipóteses estudadas. 58

4.2 O meta-algoritmo "Aprendizado Acelerado por Heurísticas" . . . 59

4.3 O algoritmo HAQL. 77

5.1 Resultado do teste t de Student para aceleração no quinto episódio usando o algoritmo HAQL no ambiente de simulação Saphira. . . 112

6.1 A solução com o menor número de passos, o tempo médio para encontrá-la e o maior reforço recebido para o problema do Carro na Montanha. 120

6.2 Resultados para o problema do Pêndulo Invertido utilizando os algoritmos TD(λ) e HATD(λ). 130

6.3 Melhor resultado encontrado pelos algoritmos DQL, ACO e HADQL após 1000 iterações. 133

6.4 Média dos resultados encontrados pelos algoritmos DQL, ACO e HADQL após 1000 iterações. 133

6.5 Tempo médio (em segundos) para encontrar a melhor solução para os algoritmos DQL, ACO e HADQL, limitados a 1000 iterações. . 133

6.6 Média do saldo cumulativo de gols e do número de vitórias para todos os jogos realizados. 142

D.1 Erros entre a função valor gerada pelo HAQL e a função valor ótima V^*, em relação ao reforço recebido ao atingir a meta, ao final do 20×10^8 passo. 164

Lista de Abreviaturas

ACO *Ant Colony Optimization* – Otimização por Colônia de Formigas

ACS *Ant Colony System* – Sistema de Colônia de Formigas

AR Aprendizado por Reforço

DQL *Distributed Q–Learning* – *Q–Learning* Distribuído

HAL *Heuristically Accelerated Learning* – Aprendizado Acelerado por Heurísticas

HAQL *Heuristically Accelerated Q–Learning* – *Q–Learning* Acelerado por Heurísticas

IA Inteligência Artificial

IAD Inteligência Artificial Distribuída

ML *Machine Learning* – Aprendizado de Máquina

MDP *Markov Decision Process* – Processos de Decisão Markovianos

PD Programação Dinâmica

POMDP *Partially Observable Markov Decision Process* – Processos de Decisão Markovianos Parcialmente Observáveis

RNA Redes Neurais Artificiais

SMA Sistemas Multiagentes

TSP *Travelling Salesman Problem* – Problema do Caixeiro Viajante

VC Visão Computacional

Lista de Símbolos

α Taxa de aprendizagem

γ Fator de desconto para os reforços futuros

$\mathcal{S}$ Conjunto finito de estados

$\mathcal{A}$ Conjunto finito de ações

$\mathcal{T}$ Função de Transição de Estado

$\mathcal{R}$ Função de Recompensa

$\mathcal{H}$ Função Heurística

π Uma política

π^* A política ótima

Q Função Valor–Ação

$\hat{Q}$ Uma estimativa da Função Valor–Ação

Q^* A Função Valor–Ação ótima

V Função Valor

$\hat{V}$ Uma estimativa da Função Valor

V^* A Função Valor ótima

$\square$ c.q.d.

1 Introdução

Um dos objetivos da Inteligência Artificial (IA) é a construção de agentes autônomos inteligentes capazes de realizar tarefas complexas de forma racional, atuando em ambientes complexos do mundo real. Não são necessárias muitas considerações para concluir que qualquer sistema autônomo deve possuir capacidades de aprendizado que possibilitem a sua adaptação a eventos imprevistos, que surgem devido à falta de conhecimento prévio sobre ambientes complexos, a adaptação a mudanças no ambiente de atuação e a melhoria do desempenho do agente ao longo do tempo.

Aprendizado de Máquina

Mitchell (1997) define o Aprendizado de Máquina (*Machine Learning* – ML) como o campo da IA cujo interesse é a construção de programas de computadores que se aperfeiçoam automaticamente com a experiência. O aprendizado de máquina tem sido usado com sucesso em quase todas as áreas do conhecimento que utilizam computadores, como classificação e reconhecimento de padrões, controle, jogos, entre outros. Uma definição genérica para um agente autônomo que aprende é dada por Russell e Norvig (2004, p. 613):

"Um agente aprendiz é aquele que pode melhorar seu comportamento através do estudo diligente de suas próprias experiências".

O aprendizado de máquina pode ser classificado pela maneira na qual o agente interage com o ambiente em que atua para construir seu conhecimento em três classes: supervisionado, não supervisionado ou por reforço.

O aprendizado supervisionado envolve a existência de um supervisor que informa ao agente quão bem ele está atuando. De maneira geral, isto ocorre quando o resultado de uma ação pode ser validada de alguma maneira. Usualmente é apresentado ao agente um conjunto de treinamento, com os valores de entradas (o estado do ambiente e a ação do agente) e os resultados esperados. Com base

na diferença entre os resultados obtidos e os esperados, o agente pode verificar e corrigir o seu funcionamento.

Quando um agente não possui informações sobre os resultados esperados para as suas ações, o aprendizado é dito não supervisionado. Neste caso, o agente pode aprender relações entre os dados que ele percebe, agrupando conjuntos de dados similares ou prevendo resultados futuros com base em ações passadas, sem a ajuda de um supervisor.

No aprendizado por reforço (AR), um agente pode aprender a atuar de maneira bem sucedida em um ambiente previamente desconhecido utilizando a experimentação direta, por meio de tentativa e erro. Neste caso, o agente recebe uma avaliação incompleta sobre sua atuação (o chamado reforço). Este trabalho tem seu foco no aprendizado por reforço, que é o assunto do próximo capítulo.

1.1 Domínio

O principal domínio estudado neste trabalho é o dos agentes robóticos inteligentes que atuam de maneira eficiente e autônoma em ambientes finitos e Markovianos. Segundo Costa (2003), este domínio é uma área de pesquisa fascinante por diversas razões:

- É uma área de pesquisa multidisciplinar, envolvendo disciplinas como: Biologia, Ciência da Computação, Ciências Cognitivas, Engenharias, Filosofia, Física, Matemática, Psicologia e Sociologia.
- É uma área com um amplo espectro de aplicações comerciais: navegação de veículos autônomos, transporte de objetos, manipulação de objetos em lugares insalubres ou pouco acessíveis, vigilância e limpeza de grandes área, busca e resgate de pessoas em situação de perigo, aplicações em agricultura, como colheita e semeadura autônoma, tratamento da terra, entre outras.
- E pelo fato dos "robôs móveis serem, atualmente, a melhor aproximação de máquinas que imitam e emulam seres vivos. Assim, a Robótica Móvel Inteligente contribui grandemente em pesquisas em vida artificial, motivadas pela questão de o que seria vida e como entendê-la" (COSTA, 2003, p. 12).

Neste domínio, o número de interações necessárias para um agente aprender, geralmente, é muito grande. Quanto maior e mais complexo o ambiente, o número de ações ou a quantidade de agentes, maior é a capacidade computacional necessária para resolver um problema.

Além dos agentes robóticos inteligentes, outros domínios, cujos resultados também se aplicam à robótica, são estudados neste trabalho. Entre estes domínios estão o do Pêndulo Invertido e o do Carro na Montanha.

Para que um robô móvel possa aprender a atingir metas específicas em ambientes com exigências de atuação em tempo real, é necessário aumentar o desempenho do aprendizado por reforço. Assim, torna-se de grande interesse propostas de métodos que permitam a aceleração do AR.

1.2 Objetivos e contribuições

Objetivo

O objetivo deste trabalho é propor uma classe de algoritmos que permite o uso de heurísticas para aceleração do aprendizado por reforço. Esta classe de algoritmos, denominada "Aprendizado Acelerado por Heurísticas" (*"Heuristically Accelerated Learning"* – HAL), é formalizada utilizando o modelo de Processos Markovianos de Decisão.

As heurísticas utilizadas nos HALs podem ser definidas a partir de conhecimento prévio sobre o domínio ou extraídas, em tempo de execução, de indícios que existem no próprio processo de aprendizagem. No primeiro caso, a heurística é definida a partir de casos previamente aprendidos ou definida *ad hoc*. Dessa forma, a heurística é uma maneira de generalização do conhecimento que se tem acerca de um domínio. No segundo caso são utilizados métodos automáticos de extração da função heurística $\mathcal{H}$.

As principais contribuições deste trabalho são:

Contribuições

- A formalização da classe de algoritmos de "Aprendizado Acelerado por Heurísticas" com a introdução de uma função heurística $\mathcal{H}$ que influencia a escolha das ações e é atualizada durante o processo de aprendizado, preservando, no entanto, muitas das propriedades dos algoritmos de AR.
- A proposta de métodos automáticos de extração da função heurística $\mathcal{H}$,

a partir do domínio do problema ou do próprio processo de aprendizagem, chamados "Heurística a partir de X" (*"Heuristic from X"*). De maneira geral estes métodos funcionam em dois estágios: o primeiro retira da estimativa da função valor informações sobre a estrutura do domínio e o segundo encontra a heurística para a política usando as informações extraídas. Estes estágios foram denominados de Extração de Estrutura e Construção da Heurística, respectivamente.

Outras contribuições relevantes são:

- A comparação do uso da função heurística $\mathcal{H}$ pelos HALs com o uso de heurísticas em algoritmos de busca informada.
- A verificação de que as informações existentes no domínio ou em estágios iniciais do aprendizado permitem definir a função heurística $\mathcal{H}$. Entre os indícios existentes no processo de aprendizagem, os mais relevantes são a função valor em um determinado instante, a política do sistema em um determinado instante e o caminho pelo espaço de estados que o agente pode explorar.
- A proposta do algoritmo "*Q–Learning* Acelerado por Heurísticas" (*Heuristically Accelerated Q–Learning* – HAQL), que implementa um HAL estendendo o conhecido algoritmo *Q–Learning* de Watkins (1989).
- O teste do algoritmo *Heuristically Accelerated Q–Learning* em diversos domínios, incluindo o dos robôs móveis autônomos e o problema do Carro nas Montanha.
- A implementação e o teste de algoritmos acelerados por heurísticas baseados em algoritmos bem conhecidos na literatura de aprendizado por reforço como o SARSA(λ), o TD(λ) e o Minimax–Q, atuando em diferentes domínios e mostrando a generalidade dos HALs.

1.3 Histórico

Este trabalho teve início como uma pesquisa sobre o aprendizado em Sistemas Multiagentes, com o objetivo de estender a arquitetura de controle distribuída

proposta pelo autor em seu mestrado (BIANCHI, 1998).

Esta arquitetura, chamada ViBRA, utiliza uma abordagem multiagentes para integrar percepção, planejamento, reação e execução para resolver problemas onde manipuladores robóticos realizam tarefas de montagens visualmente guiadas.

A arquitetura ViBRA é definida como uma sociedade dinâmica de agentes autônomos, cada qual responsável por um comportamento específico. Os agentes comunicam-se através de uma rede descentralizada e totalmente conectada e são organizados segundo uma estrutura de autoridade e regras de comportamento. Esta arquitetura combina atividades de planejamento reativas (como evitar colisões) e deliberativas (como realizar uma montagem), atuando em ambientes complexos de montagem utilizando manipuladores robóticos (COSTA; BARROS; BIANCHI, 1998).

Uma das restrições na arquitetura ViBRA foi o uso de uma estrutura de autoridade predefinida e fixa. Uma vez estabelecido que um agente tem precedência sobre outro, o sistema se comporta sempre da mesma maneira, não se preocupando com questões de eficiência.

Para solucionar este problema, Costa e Bianchi (2000) estenderam a arquitetura ViBRA utilizando aprendizado por reforço para aprender a coordenar as ações dos diversos agentes, com o objetivo de minimizar o tempo de execução de uma tarefa de montagem. Para tanto, foi introduzido um agente de controle com capacidades de aprendizado na sociedade de agentes autônomos, permitindo a substituição da estrutura de autoridade fixa por uma dinâmica.

Nesta nova arquitetura, denominada L–ViBRA, o agente de controle utiliza o algoritmo *Q–Learning* para, com base em informações recebidas pelo sistema de percepção, criar um plano de montagem otimizado. O uso do *Q–Learning* na arquitetura L–ViBRA resultou em um sistema capaz de criar o plano de montagem otimizado desejado, mas que não era rápido suficiente na produção destes planos.

Para acelerar o aprendizado, o algoritmo de aprendizado utilizado no agente de controle foi substituído por uma adaptação do algoritmo de Inteligência de Enxame (*Swarm Intelligence*) chamado *Ant Colony System* (DORIGO; GAMBARDELLA, 1997). Esta nova arquitetura, denominada Ant–ViBRA, mostrou-se

eficiente na produção dos planos de montagem necessários, reduzindo o tempo de aprendizado (BIANCHI; COSTA, 2002a; BIANCHI; COSTA, 2002b).

Apesar do aprendizado em sistemas multiagentes não ser o foco principal deste trabalho, durante estes estudos surgiram as principais questões que este trabalho pretende investigar:

- Como acelerar o aprendizado por reforço?
- Como utilizar as informações existentes no próprio processo de aprendizado para acelerar o aprendizado?
- Como utilizar informações conhecidas *a priori* acerca de um problema para acelerar o aprendizado?
- Como reutilizar conhecimentos já aprendidos para acelerar o aprendizado?
- Como compartilhar o conhecimento entre diversos agentes para acelerar o aprendizado?
- Como combinar todas estas informações com o processo de aprendizado, sem perder as boas propriedades dos algoritmos de AR?

O restante deste trabalho pretende mostrar que o uso dos algoritmos de "Aprendizado Acelerado por Heurísticas" é uma resposta eficiente e elegante para estas questões.

1.4 Organização do trabalho

Este trabalho está organizado da seguinte maneira: no capítulo 2 é apresentada uma visão geral do aprendizado por reforço, mostrando os modelos utilizados para formalizar o problema de aprendizado e alguns dos principais algoritmos.

No capítulo 3 são resumidas algumas propostas para aumentar o desempenho dos algoritmos de aprendizado por reforço apresentadas nos últimos anos. No capítulo 4 é apresentado detalhadamente o "Aprendizado Acelerado por Heurísticas", os métodos "Heurística a partir de X" e o algoritmo *Heuristically Accelerated Q–Learning* – HAQL, propostos neste trabalho.

No capítulo 5 são apresentados alguns dos resultados obtidos para o domínio dos robôs móveis autônomos utilizando o *Heuristically Accelerated Q–Learning*. Por ser um domínio mais difundido, ele foi usado para explorar métodos "Heurística a partir de X" que podem ser usados em tempo de execução. No capítulo 6, são utilizadas heurística definidas *a priori* visando explorar outros algoritmos acelerados por heurísticas, atuando em diversos domínios.

Finalmente, no capítulo 7 é realizada uma discussão sobre os resultados e propostas de extensões deste trabalho.

2 Aprendizado por Reforço

O objetivo deste capítulo é apresentar uma visão geral do Aprendizado por Reforço (AR), permitindo a compreensão dos mecanismos básicos utilizados nesta área de pesquisa.

2.1 O Problema do Aprendizado

O Agente Aprendiz

Para o Aprendizado por Reforço (AR), um agente aprendiz é aquele que, a partir da interação com o ambiente que o cerca, aprende de maneira autônoma uma política ótima de atuação: aprende ativamente, por experimentação direta, sem ser ensinado por meio de exemplos fornecidos por um supervisor.

Um agente aprendiz interage com o ambiente em intervalos de tempos discretos em um ciclo de percepção-ação (figura 2.1): o agente aprendiz observa, a cada passo de iteração, o estado corrente s_t do ambiente e escolhe a ação a_t para realizar. Ao executar esta ação a_t – que altera o estado do ambiente – o agente recebe um sinal escalar de reforço $r_{s,a}$ (penalização ou recompensa), que indica quão desejável é o estado resultante s_{t+1}. O AR permite que o agente possa determinar, após várias iterações, qual a melhor ação a ser executada em cada estado, isto é, a melhor política de atuação.

Assim, o objetivo do agente aprendiz é aprender uma política ótima de atuação que maximize a quantidade de recompensa recebida ao longo de sua execução, independentemente do estado inicial. Esse problema pode ser modelado como um Processo Markoviano de Decisão, descrito a seguir.

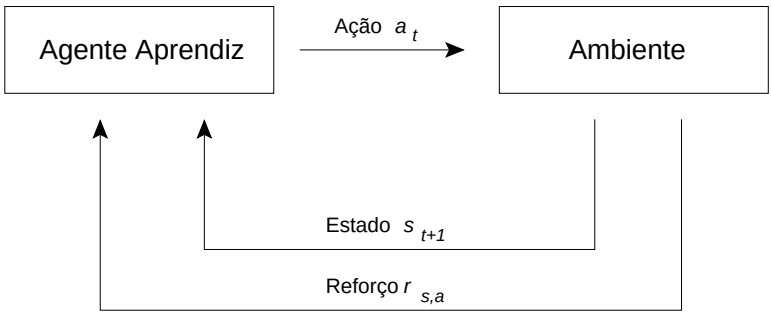


Figura 2.1: Ciclo Percepção-Ação.

2.2 Processos Markovianos de Decisão

A maneira mais tradicional para formalizar o Aprendizado por Reforço é utilizando o conceito de Processo Markoviano de Decisão (*Markov Decision Process* – MDP).

Por ser matematicamente bem estabelecido e fundamentado, este formalismo facilita o estudo do AR. Por outro lado, assume uma condição simplificadora – conhecida como Condição de Markov – que reduz a abrangência das soluções, mas que é compensada em grande parte pela facilidade de análise (RIBEIRO, 2002).

A Condição de Markov especifica que o estado de um sistema no próximo instante $(t + 1)$ é uma função que depende somente do que se pode observar acerca do estado atual e da ação tomada pelo agente neste estado (descontando alguma perturbação aleatória), isto é, o estado de um sistema independe da sua história. Pode-se ver que muitos domínios obedecem esta condição: problemas de roteamento, controle de inventário, escalonamento, robótica móvel e problemas de controle discreto em geral.

Markov Decision Process

Um Processo Markoviano de Decisão é aquele que obedece à Condição de Markov e pode ser descrito como um processo estocástico no qual a distribuição futura de uma variável depende somente do seu estado atual. Um MDP é definido formalmente (LITTMAN, 1994; KAELBLING; LITTMAN; MOORE, 1996; MITCHELL, 1997) pela quádrupla $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, onde:

- $\mathcal{S}$: é um conjunto finito de estados do ambiente.
- $\mathcal{A}$: é um conjunto finito de ações que o agente pode realizar.

- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$: é a função de transição de estado, onde $\Pi(\mathcal{S})$ é uma distribuição de probabilidades sobre o conjunto de estados $\mathcal{S}$. $T(s_t, a_t, s_{t+1})$ define a probabilidade de realizar a transição do estado s_t para o estado s_{t+1} quando se executa a ação a_t.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: é a função de recompensa, que especifica a tarefa do agente, definindo a recompensa recebida por um agente ao selecionar a ação a estando no estado s.

Resolver um MDP consiste em computar a política $\pi : \mathcal{S} \times \mathcal{A}$ que maximiza (ou minimiza) alguma função, geralmente a recompensa recebida (ou o custo esperado), ao longo do tempo.

Usando o MDP como formalismo, pode-se reescrever o objetivo do agente que aprende por reforço como: aprender a política ótima $\pi^* : \mathcal{S} \times \mathcal{A}$ que mapeia o estado atual s_t em uma ação desejada, de forma a maximizar a recompensa acumulada ao longo do tempo, descrevendo o comportamento do agente (KAELBLING; LITTMAN; MOORE, 1996).

Quando um agente não pode observar diretamente o estado do sistema em um determinado momento, mas consegue obter uma indicação sobre ele a partir de observações, o sistema é chamado Parcialmente Observável (*Partially Observable Markov Decision Process* – POMDP). POMDPs são extremamente importantes para a solução de problemas práticos, nos quais as observações são feitas com sensores imprecisos, ruidosos ou pouco confiáveis, que apenas indicam o estado completo do sistema.

Um MDP pode ser determinístico ou não-determinístico, dependendo da função de probabilidade de transição $T(\cdot)$. Caso $T(\cdot)$ especifique apenas uma transição válida para um par (estado, ação), o sistema é determinístico; caso a função defina um conjunto de estados sucessores potencialmente resultantes da aplicação de uma determinada ação em um estado, o sistema é chamado de não-determinístico. Um exemplo deste último pode ser dado para o domínio do Futebol de Robôs, no qual uma bola chutada em direção do gol pode entrar, pode bater no travessão ou pode ir para fora do campo. Outro exemplo é o do lançamento de uma moeda, no qual dois resultados são possíveis.

2.3 MDPs e Programação Dinâmica

Função Valor

Uma maneira de descrever a recompensa esperada acumulada é utilizar a função de custo esperado $V^{\pi}(s_t)$, gerada quando se segue uma determinada política π a partir de um estado inicial arbitrário s_t. Esta função, também chamada de Função Valor Cumulativo Esperado, pode ser calculada considerando-se que o intervalo de tempo no qual a recompensa é levada em conta para o cálculo da política é infinito (isto é, muito grande).

Na prática, as recompensas recebidas são amortizadas por um fator de desconto γ (onde $0 \leq \gamma < 1$), que determina o valor relativo da recompensa imediata em relação às possíveis recompensas futuras. Este modelo, que considera a recompensa que pode ser recebida a longo termo (chamada *Delayed Reward*), é denominado Modelo de Horizonte Infinito com Desconto. Nele, o valor cumulativo descontado é dado por:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}, \tag{2.1}$$

onde:

- r_{t+i} é a seqüência de reforços recebidos a partir do estado s_t, usando a política π de maneira repetida para selecionar ações e
- γ é o fator de desconto, com $0 \leq \gamma < 1$.

O objetivo do agente aprendiz é encontrar a política ótima estacionária π^* que maximiza $V^{\pi}(s)$, para todo estado $s \in S$:

$$\pi^* = argmax_{\pi} V^{\pi}(s), \forall s. \tag{2.2}$$

Programação Dinâmica

Uma maneira usual de se encontrar essa política é utilizando técnicas de Programação Dinâmica (PD) (BERTSEKAS, 1987). PD é uma coleção de técnicas que, apesar de conceitualmente simples, possui uma rigorosa base matemática. Um dos pontos fundamentais desta metodologia é o Princípio da Otimalidade de Bellman.

Esse princípio afirma que dada uma política ótima $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$ para um problema de controle, a política $\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\}$ também é ótima

para o sub-problema cujo estado inicial é s_i $(0 < i < N - 1)$.

Isto significa que seguir uma política ótima entre um estado inicial e um estado final, passando por um estado particular intermediário, é equivalente a seguir a melhor política do estado inicial até o intermediário, seguida da melhor política deste até o estado final.

Uma conseqüência imediata deste princípio é que, para definir a política de ações ótima de um sistema que se encontra no estado s_i, basta encontrar a ação $\mu_i^* = a_i$ que leva ao melhor estado s_{i+1} e, a partir deste, seguir a política ótima até o estado final. Esta definição permite a construção de diversos algoritmos recursivos para solucionar o problema utilizando PD.

Dois métodos de Programação Dinâmica muito utilizados para solucionar MDPs são o algoritmo de Iteração de Valor e o algoritmo de Iteração de Política.

Iteração de Valor

O algoritmo de Iteração de Valor (BERTSEKAS, 1987) apresentado na tabela 2.1 consiste no cálculo de maneira iterativa da função valor ótima V^* a partir da equação:

$$V(s_t) \leftarrow max_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V'(s_{t+1}) \right], \qquad (2.3)$$

onde:

- s_t é o estado atual,
- $\pi(s_t) = a_t$ é a ação realizada em s_t,
- s_{t+1} é o estado resultante ao se aplicar a ação a_t estando no estado s_t,
- V é a função valor e V' é o valor da função valor na iteração anterior.

Neste algoritmo, a política ótima π^* é encontrada ao final do processamento por meio da equação:

$$\pi^*(s_t) \leftarrow argmax_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V^*(s_{t+1}) \right]. \qquad (2.4)$$

Iteração de Política

Já o algoritmo de Iteração de Política (BERTSEKAS, 1987; KAELBLING; LITTMAN; MOORE, 1996, p. 198) manipula a política diretamente, ao invés

Repita:
Compute a função valor $V(s)$ por meio da equação:
$V(s) \leftarrow max_{\pi(s)}[r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s,\pi(s),s')V'(s')]$
Até que o valor de $V(s) = V'(s)$.
Calcule π^* por meio da equação:
$\pi^*(s) \leftarrow argmax_{\pi(s)}[r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s,\pi(s),s')V^*(s')]$.

onde: $s = s_t$ e $s' = s_{t+1}$.

Tabela 2.1: O algoritmo de Iteração de Valor (BERTSEKAS, 1987).

de encontrá-la indiretamente por meio da função valor ótima. Este algoritmo opera da seguinte maneira: após escolher uma política inicial estacionária $\pi^0 = \{a_0^0, a_1^0, \dots\}$, calcula-se V^{π^0}, resolvendo o sistema de equações lineares:

$$V^{\pi}(s_t) = r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}). \tag{2.5}$$

Após a obtenção de V^{π^0}, uma política π^1 é computada maximizando a soma do reforço $r(s,\pi(s))$ com o valor $V^{\pi}(s')$ do estado sucessor, descontado de γ:

$$\pi'(s_t) \leftarrow argmax_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}) \right]. \tag{2.6}$$

Este processo é repetido até que π^* seja encontrada, o que acontece quando $V^{\pi}(s_t) = V^{\pi'}(s_t)$ estabiliza e $\pi = \pi'$. A solução do primeiro sistema de equações lineares (equação (2.5)) é chamado de Passo da Avaliação da Política e a maximização da equação (2.6) é chamado Passo de Melhora da Política, pois nele a política é modificada de maneira a melhorar a função valor.

Diversos teoremas da área de Programação Dinâmica – que são conseqüência do Princípio da Otimalidade – garantem que, no momento em que nenhuma melhora no valor V for possível, a política é ótima (BERTSEKAS, 1987; KAELBLING; LITTMAN; MOORE, 1996, p. 198). O algoritmo de Iteração de Política completo é apresentado na tabela 2.2.

A política $\pi'(s_t)$ determinada segundo a equação (2.6) é chamada política Gulosa (*Greedy*) para $V^{\pi}(s_{t+1})$, pois escolhe sempre a ação que determina o maior retorno estimado (o máximo da função valor). Esta estratégia possui uma falha:

Inicialize $\pi \leftarrow \pi^0$ arbitrariamente.
Repita
  Passo 1: Compute a função valor $V^\pi(s)$ para a política π
    por meio da solução do sistema de equações lineares:
    $V^\pi(s) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s')$
  Passo 2: Melhore a política para cada estado:
    $\pi'(s) \leftarrow argmax_a [r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s')]$
  $\pi \leftarrow \pi'$
Até que $V^\pi(s) = V^{\pi'}(s)$.

onde: $s = s_t$, $s' = s_{t+1}$ e $a = a_t = \pi(s_t)$.

Tabela 2.2: O algoritmo de Iteração de Política (BERTSEKAS, 1987, p. 199).

uma amostra inicial ruim pode indicar uma ação que não é ótima, mas sub-ótima. Neste caso, o sistema estabiliza em um mínimo local.

Outras estratégias são possíveis, como exploração aleatória ou exploração Boltzmann. Na exploração aleatória conhecida como $\epsilon - Greedy$, o agente escolhe a melhor ação com probabilidade $1 - p$ e escolhe uma ação aleatória com probabilidade p. Na exploração Boltzmann, a escolha da ação a estando em um estado $s \in S$ obedece uma equação que depende de V e de outros parâmetros semelhantes aos usados em técnicas de Redes Neurais Artificiais, como a Têmpera Simulada (HAYKIN, 1999).

Nessas estratégias, o agente aprendiz sempre enfrenta um compromisso entre exploração, no qual o agente reúne novas informações sobre o sistema, e explotação, onde o agente utiliza as informações já conhecidas. Em uma boa estratégia, o uso de exploração e explotação deve ser balanceado.

Comparando os dois algoritmos apresentados nesta seção (tabelas 2.1 e 2.2), pode-se notar que a Iteração de Valor é muito mais rápida por iteração, já que não precisa resolver um sistema de equações lineares. Por outro lado, o método de Iteração de Política utiliza menos passos para chegar à solução. Outra diferença é que Iteração de Valor atua apenas sobre o estado, enquanto a Iteração de Política atua sobre o par estado-ação.

Finalmente, os algoritmos apresentam duas desvantagens. Em primeiro lugar, tanto a computação iterativa da equação (2.3) quanto a solução do sistema de equações lineares da equação (2.5), a cada iteração, são computacionalmente

custosas. Por exemplo, o algoritmo de Iteração de Política é de ordem exponencial (KAELBLING; LITTMAN; MOORE, 1996, p. 198). Em segundo lugar, é necessário que se conheça o modelo – $\mathcal{T}$ e $\mathcal{R}$ – do sistema. Para superar esses problemas foram propostos diversos métodos livres de modelos (*Model-Free*), apresentados nas seções a seguir.

2.4 *Q–Learning*

Tido como o mais popular algoritmo de aprendizado por reforço, o algoritmo *Q–Learning* foi proposto por Watkins (WATKINS, 1989; WATKINS; DAYAN, 1992) como uma maneira de aprender iterativamente a política ótima π^* quando o modelo do sistema não é conhecido. Além do fato de ser tipicamente de fácil implementação (KAELBLING; LITTMAN; MOORE, 1996, p. 253), Ribeiro (2002) expõe algumas razões para essa popularidade:

- Foi o pioneiro e é, até o momento, o único método de aprendizado por reforço usado para propósitos de controle minuciosamente estudado e com uma prova de convergência bem estabelecida.
- É uma extensão do conceito de aprendizado autônomo de Controle Ótimo, sendo a técnica mais simples que calcula diretamente uma política de ações, sem o uso de um modelo e sem um passo intermediário de avaliação de custo.

Além disso, o *Q–Learning* tem sido aplicado com sucesso em uma grande quantidade de domínios (MITCHELL, 1997).

Função Valor–Ação

O algoritmo *Q–Learning* propõe que o agente, ao invés de maximizar V^*, aprenda uma função de recompensa esperada com desconto Q, conhecida como Função Valor–Ação. Esta função de estimação Q é definida como sendo a soma do reforço recebido pelo agente por ter realizado a ação a_t no estado s_t em um momento t, mais o valor (descontado de γ) de seguir a política ótima daí por diante:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma V^*(s_{t+1}). \tag{2.7}$$

Reescrevendo esta equação na forma não-determinística, tem-se:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}). \tag{2.8}$$

A partir desta definição, e como conseqüência do Princípio da Otimalidade de Bellman, tem-se que:

$$\begin{aligned} V^*(s_t) &= \max_{a_t} \left[r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}) \right] \\ &= \max_{a_t} Q^*(s_t, a_t). \end{aligned} \tag{2.9}$$

Pode-se então concluir que:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}). \tag{2.10}$$

Outra conseqüência importante desta formulação é que a política ótima $\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t)$ pode ser obtida diretamente.

Seja $\hat{Q}_t$ a estimativa de $Q^*(s, a)$ no instante t. O algoritmo *Q–Learning* aproxima iterativamente $\hat{Q}$, isto é, os valores de $\hat{Q}$ convergem com probabilidade 1 para Q^*, desde que o sistema possa ser modelado como um MDP, a função de recompensa seja limitada ($\exists c \in \mathcal{R}; (\forall s, a), |r(s, a)| < c$), e as ações sejam escolhidas de maneira que cada par estado-ação seja visitado um número infinito de vezes (MITCHELL, 1997). A regra de aprendizado *Q–Learning* é:

$$\hat{Q}_{t+1}(s_t, a_t) \leftarrow \hat{Q}_t(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a_{t+1}} \hat{Q}_t(s_{t+1}, a_{t+1}) - \hat{Q}_t(s_t, a_t) \right], \tag{2.11}$$

onde:

- s_t é o estado atual,
- a_t é a ação realizada em s_t,
- $r(s_t, a_t)$ é o reforço recebido após realizar a_t em s_t,
- s_{t+1} é o novo estado,
- γ é o fator de desconto ($0 \leq \gamma < 1$) e

- α é a taxa de aprendizagem ($0 < \alpha < 1$), podendo ser definida por $\alpha = 1/(1 + visitas(s, a))$, sendo $visitas(s, a)$ o número de visitas já realizadas ao estado s, com ação a escolhida e realizada.

Uma propriedade importante deste algoritmo é que as ações usadas durante o processo iterativo de aproximação da função Q podem ser escolhidas usando qualquer estratégia de exploração (e/ou explotação). Desde que cada par (estado, ação) seja visitado muitas vezes, a convergência dos valores de $\hat{Q}$ para Q^* é garantida (MITCHELL, 1997) – porém essa convergência é extremamente lenta.

Uma estratégia para a escolha das ações bastante utilizada em implementações do *Q–Learning* é a exploração aleatória $\epsilon - Greedy$, na qual o agente executa a ação com o maior valor de Q com probabilidade $1 - \epsilon$ e escolhe uma ação aleatória com probabilidade ϵ. Neste caso, a transição de estados é dada pela seguinte Regra de Transição de Estados:

$$\pi(s_t) = \begin{cases} a_{random} & \text{se } q \leq \epsilon, \\ \arg\max_{a_t} \hat{Q}_t(s_t, a_t) & \text{caso contrário,} \end{cases} \tag{2.12}$$

onde:

- q é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e ϵ ($0 \leq \epsilon \leq 1$) é o parâmetro que define a taxa de exploração/explotação: quanto menor o valor de ϵ, menor a probabilidade de se fazer uma escolha aleatória.

- a_{random} é uma ação aleatória selecionada entre as ações possíveis de serem executadas no estado s_t.

O algoritmo *Q–Learning* completo é apresentado na tabela 2.3.

2.5 O método das Diferenças Temporais – *TD-Learning*

O método das Diferenças Temporais $TD(0)$ (SUTTON, 1988) foi um dos primeiros algoritmos de aprendizado por reforço desenvolvido a ter uma base matemática consistente. Foi proposto como uma versão adaptativa do algoritmo de Iteração

Inicialize $Q_t(s, a)$ arbitrariamente.
Repita:
Visite o estado s.
Selecione uma ação a de acordo com a regra de transição de estados.
Execute a ação a.
Receba o reforço $r(s, a)$ e observe o próximo estado s'.
Atualize os valores de $Q_t(s, a)$ de acordo com a regra de atualização:
$Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a)]$.
Atualize o estado $s \leftarrow s'$.
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$, $a = a_t$ e $a' = a_{t+1}$.

Tabela 2.3: O algoritmo *Q–Learning* (WATKINS, 1989).

de Política onde, ao invés de se computar a função valor por meio da resolução do sistema de equações (2.5), esta é determinada de maneira iterativa, por meio da equação:

$$V_{t+1}(s_t) \leftarrow V_t(s_t) + \alpha \left[r(s_t, a_t) + \gamma V_t(s_{t+1}) - V_t(s_t) \right], \tag{2.13}$$

onde:

- s_t é o estado atual,
- $r(s_t, a_t)$ é o reforço recebido após realizar a_t em s_t,
- s_{t+1} é o próximo estado,
- γ é o fator de desconto ($0 \leq \gamma < 1$), e
- α é a taxa de aprendizagem ($0 < \alpha < 1$).

Esta equação é similar à equação (2.3) usada no algoritmo de Iteração de Valor, na qual o valor de $T(s_t, a_t, s_{t+1})$ é retirado da interação com o mundo real e não de um modelo conhecido. Assim, os métodos de Diferença Temporal são uma combinação de Programação Dinâmica com os métodos de simulação de Monte Carlo (ROBBINS; MONRO, 1951). A principal propriedade deste método é que, se a taxa de aprendizagem α decair lentamente e se a política for mantida fixa, $TD(0)$ converge para o ótimo da função valor para esta política (SUTTON, 1988).

Para realizar a atualização das políticas Barto, Sutton e Anderson (1983) propuseram a arquitetura chamada *Adaptative Heuristic Critic* (AHC). Nela, dois componentes trabalham de forma alternada: primeiro se toma uma política fixa se computa a equação (2.13) de maneira iterativa, determinando a estimativa da função valor em um determinado instante. Em seguida, esta estimativa é utilizada para computar uma nova política.

O $TD(0)$ pode ser classificado como um caso especial de uma classe mais geral de algoritmos, chamados de Algoritmos das Diferenças Temporais – $TD(\lambda)$ (SUTTON, 1988). No $TD(\lambda)$, a atualização do valor do estado atual pode ser feita usando apenas o próximo estado ($\lambda = 0$) – recaindo no $TD(0)$ – até o uso de todos os estados futuros ($\lambda = 1$). Para tanto, a equação (2.13) é reescrita como:

$$V_{t+1}(u) \leftarrow V_t(u) + \alpha \left[r(s_t, a_t) + \gamma V_t(s_{t+1}) - V_t(s_t) \right] e(u), \tag{2.14}$$

onde:

- u é o estado que está sendo atualizado,
- s_t é o estado atual do ambiente,
- $r(s_t, a_t)$ é o reforço recebido após realizar a_t em s_t,
- s_{t+1} é o próximo estado do ambiente,
- γ é o fator de desconto ($0 \leq \gamma < 1$), e
- α é a taxa de aprendizagem ($0 < \alpha < 1$).
- $e(u)$ é a elegibilidade do estado u.

A elegibilidade de um estado define o grau em que foi visitado no passado recente. O uso deste termo na equação (2.14) permite que um reforço recebido seja usado para atualizar todos os estados recentemente visitados. Isto faz com que os estados mais próximos das recompensas sejam mais influenciados por elas. A elegibilidade pode ser definida como:

$$e(u) = \sum_{k=1}^{t} (\lambda\gamma)^{(t-k)} \delta_{s,s_k}, \tag{2.15}$$

onde:

- u é o estado que está sendo atualizado,
- t é o instante atual,
- λ é o fator de desconto para as diferenças temporais ($0 \leq \lambda \leq 1$),
- γ é o fator de desconto para os reforços futuros ($0 \leq \gamma < 1$) e
- δ_{s,s_k} vale 1 se $s = s_k$ e 0 caso contrário (Delta de Kronecker).

Uma característica que facilita a computação da elegibilidade é que ela pode ser calculada a cada passo da execução do algoritmo, usando:

$$e(u) = \begin{cases} \gamma\lambda e(u) + 1 & \text{se } u = s_t \\ \gamma\lambda e(u) & \text{caso contrário.} \end{cases} \tag{2.16}$$

A execução do algoritmo $TD(\lambda)$ é computacionalmente mais custosa que o $TD(0)$. Porém, dependendo do valor de λ, o primeiro pode convergir mais rápido. Ribeiro (2002) afirma que "quase todos os algoritmos de AR atuais são baseados no método das Diferenças Temporais". Isto é facilmente visto no caso do *Q–Learning*, onde a regra de aprendizado (equação (2.11)) é um caso especial da regra de atualização do $TD(0)$ (equação (2.13)). Finalmente, um algoritmo genérico para o $TD(\lambda)$ é apresentado na tabela 2.4.

2.6 SARSA

Inicialmente chamado de *Q–Learning* Modificado, o algoritmo *SARSA* (SUTTON, 1996) propõe que a política seja aprendida em tempo de execução, estimando o valor de uma política ao mesmo tempo que a usa para interagir com o ambiente.

Para fazer isso, a regra de aprendizado usada no *SARSA* elimina a maximação das ações que existe no *Q–Learning* (equação (2.11)), sendo que a regra fica:

$$\hat{Q}_{t+1}(s_t, a_t) \leftarrow \hat{Q}_t(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \hat{Q}_t(s_{t+1}, a_{t+1}) - \hat{Q}_t(s_t, a_t) \right]. \tag{2.17}$$

Se a_{t+1} for escolhido segundo uma política gulosa, o algoritmo se torna igual ao *Q–Learning* e $a_{t+1} = \max_{a_{t+1}} \hat{Q}_t(s_{t+1}, a_{t+1})$. Porém, neste método, a ação a_{t+1} é escolhida de maneira aleatória (a partir de uma determinada distribuição de

Inicialize $V_t(s)$ arbitrariamente.
Inicialize $e(s) = 0$.
Repita:
  Visite o estado s e selecione uma ação a de acordo com uma política π.
  Execute a ação a.
  Receba o reforço $r(s, a)$ e observe o próximo estado s'.
  Atualize a elegibilidade $e(s) = e(s) + 1$.
  Para todos os estados u:
    Atualize os valores de $V_t(u)$ de acordo com a regra de atualização:
      $V_{t+1}(u) \leftarrow V_t(u) + \alpha[r(s, a) + \gamma V_t(s') - V_t(s)]e(u)$.
    Atualize $e(u) \leftarrow \gamma\lambda e(u)$.
  Atualize o estado $s \leftarrow s'$.
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$, $a = a_t$.

Tabela 2.4: O algoritmo Genérico $TD(\lambda)$ (SUTTON; BARTO, 1998).

probabilidades), alcançando um rendimento melhor que o *Q–Learning*, em casos onde o conjunto de ações é muito grande ou onde o ambiente apresenta apenas penalidades (SUTTON, 1996).

Para realizar o aprendizado *on-line*, a cada iteração estima-se $\hat{Q}^\pi$ a partir de π, ao mesmo tempo em que se modifica a distribuição de probabilidades que define a escolha da próxima ação (ou seja, π).

Finalmente, o nome do algoritmo advém do fato que a regra de aprendizado utiliza todos o elementos da quíntupla $\langle s_t, a_t, r, s_{t+1}, a_{t+1}\rangle$ – que define a transição de um par estado-ação para o próximo – para atualizar a estimativa de Q.

2.7 Jogos de Markov e o *Minimax–Q*

Markov Games

Os Jogos de Markov (*Markov Games* – MG) são uma extensão da teoria dos jogos para MDPs, que permite modelar agentes que competem entre si para realizar suas tarefas.

Um MG é definido de maneira similar a um MDP. Formalmente, um MG é, em sua forma geral, definido por (LITTMAN, 1994):

- $\mathcal{S}$: um conjunto finito de estados do ambiente.

- $\mathcal{A}_1 \ldots \mathcal{A}_k$: uma coleção de conjuntos $\mathcal{A}_i$ das possíveis ações de cada agente i.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_k \rightarrow \Pi(\mathcal{S})$: uma função de transição de estado que depende do estado atual e das ações de cada agente.
- $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_k \rightarrow \Re$: um conjunto de funções de recompensa, que especifica a recompensa recebida para cada agente i.

Resolver um MG consiste em computar a política $\pi : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_k$ que maximiza a recompensa recebida pelo agente ao longo do tempo.

Uma especialização dos Jogos de Markov muito estudada considera que existem apenas dois jogadores, chamados de agente e oponente, com objetivos diametralmente opostos. Esta especialização, chamada Jogo de Markov de soma zero, permite que se defina apenas uma função de recompensa, que o agente deseja maximizar e que o oponente tenta minimizar.

Zero Sum Markov Game

Um Jogo de Markov de soma zero entre dois jogadores é definido (LITTMAN, 1994) pela quíntupla $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$, onde:

- $\mathcal{S}$: é um conjunto finito de estados do ambiente.
- $\mathcal{A}$: é um conjunto finito de ações que o agente pode realizar.
- $\mathcal{O}$: é um conjunto finito de ações que o oponente pode realizar.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow \Pi(\mathcal{S})$: é a função de transição de estado, onde $\Pi(\mathcal{S})$ é uma distribuição de probabilidades sobre o conjunto de estados $\mathcal{S}$. $T(s_t, a_t, o_t, s_{t+1})$ define a probabilidade de realizar a transição do estado s_t para o estado s_{t+1} quando o agente executa a ação a_t e o oponente, a ação o_t.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow \Re$: é a função de recompensa, que especifica a recompensa recebida pelo agente quando este seleciona a ação a_t e o seu oponente, a ação o_t, estando no estado s_t.

A escolha da política ótima para um Jogo de Markov não é tarefa trivial porque o desempenho do agente depende de maneira crucial da escolha do oponente. A solução para este problema é bem conhecida, pelo algoritmo Minimax.

Inicialize $Q_t(s, a, o)$, $V_t(s)$ e $\pi_t(s, a)$.
Repita:
  Visite o estado s.
  Selecione uma ação a de acordo com a política $\pi_t(s, a)$.
  Execute a ação a.
  Observe a ação o do adversário, receba o reforço $r(s, a, o)$.
  Observe o próximo estado s'.
  Atualize os valores de $Q_t(s, a)$ de acordo com a regra de atualização:
    $Q_{t+1}(s, a, o) \leftarrow Q_t(s, a, o) + \alpha[r(s, a, o) + \gamma V_t(s') - Q_t(s, a, o)]$.
  Compute os valores de $\pi_t(s, \cdot)$ utilizando programação linear:
    $\pi_{t+1}(s, \cdot) \leftarrow \arg\max_{\pi_{t+1}(s,\cdot)} \min_{o_t \in O} \sum_{a'} Q_t(s, a', o')\pi_t(s, a')$.
  Atualize os valores de $V_t(s)$ de acordo com:
    $V_{t+1}(s) \leftarrow \min_{o'} \sum_{a'} Q_t(s, a', o')\pi(s, a')$.
  Atualize o estado $s \leftarrow s'$.
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$, $a = a_t$, $o = o_t$ e $a' = a_{t+1}$.

Tabela 2.5: O algoritmo Minimax–Q (LITTMAN, 1994).

O algoritmo Minimax (RUSSELL; NORVIG, 2004, capítulo 6) avalia a política de um agente em relação a todas possíveis ações do oponente, escolhendo sempre a política que minimiza o ganho do oponente e maximiza seu próprio ganho.

Minimax–Q

Para solucionar um Jogo de Markov, Littman (1994) propôs utilizar uma estratégia similar ao Minimax para a escolha da ação no *Q–Learning*, criando o Minimax–Q.

O Minimax–Q funciona de maneira essencialmente igual ao *Q–Learning* (tabela 2.5). A função valor–ação de uma ação a_t em um determinado estado s_t quando o oponente toma a ação o_t é dada por:

$$Q(s_t, a_t, o_t) = r(s_t, a_t, o_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, o_t, s_{t+1})V(s_{t+1}), \tag{2.18}$$

e o valor de um estado pode ser computado utilizando Programação Linear (STRANG, 1988) pela equação:

$$V(s_t) = \max_{a_t \in \Pi(\mathcal{A})} \min_{o_t \in \mathcal{O}} \sum_{a_t \in \mathcal{A}} Q(s_t, a_t, o_t)\pi_a. \tag{2.19}$$

Os Jogos de Markov onde os dois jogadores tomam suas ações em turnos

consecutivos são chamados de Alternantes (*Alternating Markov Games* – AMG). Neste caso, como o agente conhece a ação que foi tomada pelo oponente antes de necessitar escolher sua ação, a política se torna determinística, $\pi : \mathcal{S} \times \mathcal{A} \times \mathcal{O}$ e a equação (2.19) pode ser simplificada, resultando em:

$$V(s_t) = \max_{a_t \in \mathcal{A}} \min_{o_t \in \mathcal{O}} Q(s_t, a_t, o_t). \tag{2.20}$$

Finalmente, o algoritmo Minimax–Q tem sido estendido para tratar diversos domínios onde os Jogos de Markov se aplicam, como no Futebol de Robôs (LITTMAN, 1994; BOWLING; VELOSO, 2001; SAVAGAONKAR, 2002), busca de informações na Internet (KHOUSSAINOV; KUSHMERICK, 2003; KHOUSSAINOV, 2003) e alocação de canais em redes de computadores (SAVAGAONKAR, 2002), entre outros.

2.8 Processos Markovianos de Decisão Generalizados e o *Q–Learning* Generalizado

GMDP Szepesvári e Littman (1996) propõem um modelo geral para MDPs, chamado Processos Markovianos de Decisão Generalizados (*Generalized Markov Decision Process* – GMDP). O conceito básico introduzido pelos GMDPs é que a operação max_{a_t} que descreve a ação de um agente ótimo e a operação $\sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) V'(s_{t+1})$ que descreve o efeito do ambiente podem ser generalizadas. A grande contribuição dos GMDPs é que esta modelagem permite discutir uma grande variedade de modelos de programação dinâmica e de aprendizado por reforço de maneira unificada. Entre os diversos modelos estudados pelos autores estão: Iteração de Valor e Iteração de Política (seção 2.3), *Q–Learning* (seção 2.4), *Q–Learning* com espalhamento (RIBEIRO, 1998), *Q–Learning* sensível à exploração (JOHN, 1994) e sensível a risco (HEGER, 1994), SARSA (seção 2.6), Jogos de Markov (seção 2.7), Jogos de Markov com espalhamento (PEGORARO; COSTA; RIBEIRO, 2001), entre outros.

Um GMDP é definido pela quíntupla $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \bigoplus, \bigotimes \rangle$, onde:

- $\mathcal{S}$: é um conjunto finito de estados do ambiente.
- $\mathcal{A}$: é um conjunto finito de ações que o agente pode realizar.

Modelo	$(\bigotimes f)(s)$
MDPs valor descontado	$\max_a f(s,a)$
Jogos de Markov	$\max_A \min_b \sum_a A(a) f(s,a,o)$
Jogos de Markov Alternantes	$\max_a ou \min_b f(s,b)$

Modelo	$(\bigoplus g)(s,a)$
MDPs valor descontado	$\sum_{s'} T(s,a,s')g(s,a,s')$
Jogos de Markov	$\sum_{s'} T(s,a,o,s')g(s,a,o,s')$
Jogos de Markov Alternantes	$\sum_{s'} T(s,a,s')g(s,a,s')$

Tabela 2.6: Alguns modelos de MDPs Generalizados e suas especificações (SZEPESVÁRI; LITTMAN, 1996, Teorema 3).

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \Re$: é a função de recompensa.
- $\bigoplus : (\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Re) \to (\mathcal{S} \times \mathcal{A} \to \Re)$: é o operador que define o valor esperado a partir da função de transição. Este operador define como o valor do próximo estado deve ser usado para atualizar o estado atual.
- $\bigotimes : (\mathcal{S} \times \mathcal{A} \to \Re) \to (\mathcal{S} \to \Re)$: é o operador "maximização", que recebe uma função valor-ação que mapeia estados e ações em valores e retorna o valor da melhor ação de cada estado.

Por exemplo, se $\bigoplus$ for definido como $(\bigoplus S)(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) S(s_t, a_t, s_{t+1})$ onde $S : (\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Re)$ e $\bigotimes$ definido como $(\bigotimes Q)(s_t) = max_{a_t} Q(s_t, a_t)$ onde $Q : (\mathcal{S} \times \mathcal{A} \to \Re)$, tem-se o MDP descrito na seção 2.2. A equação (2.5), que é usada para solucionar um MDP, pode ser reescrita como:

$$V^{\pi}(s_t) = \bigotimes \bigoplus (r(s_t, \pi(s_t), s_{t+1}) + \gamma V^{\pi}(s_{t+1})). \tag{2.21}$$

O operador $\bigoplus$ é uma não-expansão se $\| \bigoplus f - \bigoplus g \| \leq \| f - g \|$ for válido para qualquer $f, g : \mathcal{S} \times \mathcal{A} \to \Re$ e $s \in \mathcal{S}$. Uma condição análoga define quando $\bigotimes$ é uma não-expansão. Szepesvári e Littman (1996) provam que se ambos os operadores $\bigoplus$ e $\bigotimes$ forem não-expansões e se $0 \leq \gamma < 1$, a solução para o GMDP $V^* = \bigoplus \bigotimes (R + \gamma V^*)$ existe e é única (SZEPESVÁRI; LITTMAN, 1996, Teorema 3). A tabela 2.6 apresenta alguns modelos e suas especificações como GMDPs.

De maneira similar, pode-se definir o algoritmo *Q–Learning* Generalizado utilizando os operadores $\bigoplus$ e $\bigotimes$. Seja a tupla $\langle s_t, a_t, s_{t+1}, r_t \rangle$ a experiência em um determinado instante t e $\hat{Q}_t(s_t, a_t)$ a estimativa da função Q, pode-se calcular o valor ótimo iterativamente por meio da equação de aprendizado:

$$\hat{Q}_{t+1}(s_t, a_t) \leftarrow \hat{Q}_t(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \bigotimes \hat{Q}_t(s_{t+1}) - \hat{Q}_t(s_t, a_t) \right]. \quad (2.22)$$

Finalmente, Szepesvari e Littman provaram que no algoritmo *Q–Learning* Generalizado o valor V_t converge com probabilidade unitária para o V^*, maximizando a recompensa acumulada esperada com desconto, se:

- s_{t+1} for selecionado de maneira aleatória de acordo com a distribuição de probabilidade de transição $T(s_t, a_t, \cdot)$,
- $\bigotimes$ for uma não-expansão,
- o reforço for limitado e
- a taxa de aprendizado decair. Uma taxa de aprendizado comumente usada é $\alpha_t(s_t, a_t) = 1/(1 + n_t(s_t, a_t))$, onde $n_t(s_t, a_t)$ é o número de visitas ao par estado-ação. O uso desta função requer que cada par estado-ação seja visitado infinitamente.

Este teorema pode ser estendido para os Jogos de Markov, *Q–Learning* com espalhamento e outros modelos.

2.9 Discussão

Este capítulo apresentou os conceitos básicos do Aprendizado por Reforço. Não foi o objetivo deste realizar uma extensa e completa revisão dos algoritmos de AR, que pode ser encontrada em obras como Kaelbling, Littman e Moore (1996), Sutton e Barto (1998) ou Ribeiro (2002). Assim, diversos assuntos importantes como os Processos Markovianos de Decisão Parcialmente Observáveis (*Partially Observable Markov Decision Process*) ou os métodos de simulação de Monte Carlo não foram abordados por não serem fundamentais para a compreensão deste trabalho.

Procurou-se descrever os algoritmos básicos de AR, ressaltando a boa propriedade de garantia, sob certas condições, do alcance da política (ou valor) ótimo para a atuação do agente. Por outro lado, foi mostrada a dificuldade de se alcançar tal otimalidade devido ao grande número de interações necessárias do agente com o ambiente, o que torna o processo de aprendizagem extremamente lento.

O próximo capítulo apresenta algumas maneiras comumente utilizadas para acelerar o aprendizado por reforço.

3 Aceleração do aprendizado por reforço

Para todos os algoritmos apresentados até o momento, o número de iterações necessárias para convergência, geralmente, é muito grande. Quanto maior e mais complexo o ambiente, maior o número de ações ou a quantidade de agentes, maior é a capacidade computacional necessária para resolver um problema. Para aumentar o escopo de aplicação dos métodos de AR, algumas propostas que melhoram os seus desempenhos foram apresentadas nos últimos anos.

Uma maneira de acelerar o aprendizado é melhorando o aproveitamento das experiências, por meio de generalizações temporais, espaciais ou das ações. Na generalização temporal, os resultados de uma experiência são distribuídos para estados executados anteriormente. Na generalização espacial, os resultados de uma experiência são distribuídos para vários estados segundo alguma medida de similaridade do espaço de estados.

Outra maneira de acelerar o aprendizado é com o uso de abstrações. Neste caso, pode-se realizar uma abstração temporal na forma de macro-ações, opções ou máquinas hierárquicas abstratas (*Hierarchic Abstract Machines* – HAMs) ou uma abstração estrutural, na qual estados são agregados de maneira que o tamanho efetivo (e a complexidade) do problema seja reduzido.

Neste trabalho é utilizado o termo generalização quando ocorre o espalhamento da experiência e abstração quando se realiza o aprendizado em um nível mais alto de representação. Um fator que gera confusão entre os termos é que a abstração e a generalização geralmente são usadas em conjunto.

Finalmente, outras maneiras possíveis de aceleração do aprendizado incluem a abordagem distribuída e a utilização de uma base de casos. Na abordagem distri-

buída, em vez de um único agente, tem-se diversos agentes aprendendo ao mesmo tempo; a utilização de uma base de casos reutiliza conhecimento sobre as soluções já encontradas. Algumas destas abordagens são discutidas neste capítulo.

3.1 Aceleração por Generalização

3.1.1 Generalização Temporal

Dyna

A arquitetura *Dyna* foi proposta por Sutton (1990) como uma maneira de encontrar uma política ótima por meio do aprendizado do modelo do ambiente. À medida que as ações são executadas, este algoritmo aprende iterativamente o modelo da função de Transição de Estado $\hat{T}(s_t, a_t, s_{t+1})$ e das recompensas $\hat{R}(s_t, a_t)$, usando a experiência e o modelo aprendidos para ajustar a política.

O funcionamento deste algoritmo é muito similar ao do *Q–Learning*. Ele opera em um ciclo que se repete até que algum critério de parada seja atingido, onde, dado uma tupla de experiência $\langle s_t, a_t, s_{t+1}, r_{s_t,a_t} \rangle$, o algoritmo:

- Atualiza os modelos de $\hat{T}_t(s_t, a_t, s_{t+1})$ e das recompensas $\hat{R}_t(s_t, a_t)$.
- Atualiza a política para o estado s_t em um instante t com base no modelo recém atualizado, usando a regra:

$$\hat{Q}_{t+1}(s_t, a_t) \leftarrow \hat{R}_t(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \hat{T}_t(s_t, a_t, s_{t+1}) \max_{a_{t+1}} \hat{Q}_t(s_t, a_{t+1}). \quad (3.1)$$

- Realiza k atualizações adicionais usando a regra acima em k pares de estado-ação escolhidos aleatoriamente.
- Atualiza o estado $s_t \leftarrow s_{t+1}$.

O algoritmo *Dyna* requer k vezes mais tempo para executar uma iteração que o *Q–Learning*, mas requer geralmente uma ordem de grandeza menor de passos para chegar à política ótima (KAELBLING; LITTMAN; MOORE, 1996).

Prioritized Sweeping

Uma melhoria natural que pode ser feita à metodologia implementada no *Dyna* foi proposta independentemente por duas técnicas similares chamadas *Queue-Dyna* (PENG; WILLIAMS, 1993) e *Prioritized Sweeping* (MOORE; ATKESON,

1993). Nestas técnicas, ao invés de escolher aleatoriamente os estados onde serão realizadas as k atualizações adicionais, escolhe-se de acordo com uma prioridade dinâmica, que indica a importância da atualização de um estado.

Para realizar as atualizações, cada estado s possui a informação de quais são seus predecessores (estados que possuam uma probabilidade de transição para s diferente de zero) e uma prioridade, inicialmente igual a zero. O funcionamento do algoritmo é similar ao *Dyna*, com a exceção de um passo adicional que é usado para atualizar o valor da prioridade de um estado: as prioridades dos estados que não necessitam de atualização são diminuídas e prioridades altas são atribuídas a estados predecessores que necessitem de atualizações.

A principal diferença entre o *Prioritized Sweeping* e o *Queue-Dyna* é que o primeiro mantém apenas a função valor V enquanto o último atualiza a função Q.

$Q(\lambda)$ e $SARSA(\lambda)$

Outros dois algoritmos bem conhecidos que utilizam generalização temporal são o $Q(\lambda)$ (WATKINS, 1989) e o $SARSA(\lambda)$ (RUMMERY; NIRANJAN, 1994), que estendem o *Q–Learning* e o SARSA adicionando a estes a propagação das atualizações existente no $TD(\lambda)$. Isto é feito ampliando o conceito da elegibilidade para incluir o par estado-ação:

$$e(u,v) = \begin{cases} \gamma\lambda e(u,v) + 1 & \text{se } u = s_t \text{ e } v = a_t, \\ \gamma\lambda e(u,v) & \text{caso contrário.} \end{cases} \tag{3.2}$$

Além disso, outras alterações são feitas na regra de atualização dos algoritmos para incluir o espalhamento do aprendizado.

3.1.2 Generalização Espacial

*QS–*Learning

A generalização espacial envolve a distribuição dos resultados de uma experiência para vários estados, segundo alguma medida de similaridade do espaço de estados. Um algoritmo que realiza este espalhamento é o QS, proposto por Ribeiro (1998). Nele, o algoritmo *Q–Learning* é combinado com o espalhamento espacial na função valor-ação. Assim, ao receber um reforço, outros pares valor-ação, que não estavam envolvidos na experiência também são atualizados. Isto é feito por meio da codificação do conhecimento que se tem sobre as similaridades no

Inicialize $Q_t(s, a)$ arbitrariamente.
Repita:
  Visite o estado s.
  Selecione uma ação a de acordo com a regra de transição de estados.
  Execute a ação a.
  Receba o reforço $r(s, a)$ e observe o próximo estado s'.
  Para todos os estados u e ações v:
    Atualize os valores de todos os $Q_t(u, v)$ de acordo com a regra:
    $Q_{t+1}(u, v) \leftarrow Q_t(u, v)$
      $+ \alpha(s, a)\sigma_t(s, a, u, v)[r(s, a) + \gamma \max_{a'} Q_t(s', a') - Q_t(u, v)]$.
  Atualize o estado $s \leftarrow s'$.
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$, $a = a_t$ e $a' = a_{t+1}$.

Tabela 3.1: O algoritmo *QS* (RIBEIRO, 1998, p. 53).

domínio em uma função de espalhamento $\sigma_t(s, a, u, v)$, com $0 \leq \sigma_t(s, a, u, v) \leq 1$.

A função $\sigma_t(\cdot)$ define o espalhamento: se $\sigma_t(s, a, u, v) = f_u(u, s)\delta(v, a)$, obtém-se o espalhamento no espaço de estados, com $f_u(u, s)$ definindo a similaridade entre os estados u e s; se $\sigma_t(s, a, u, v) = \delta(u, s)f_v(v, a)$, obtém-se o espalhamento no espaço de ações, com $f_v(v, a)$ definindo a similaridade entre as ações v e a no estado s (δ é o Delta de Kronecker). O caso onde $f_u(u, s) = \delta(u, s)$ e $f_v(v, a) = \delta(v, a)$ corresponde ao *Q–Learning* padrão, onde não existe espalhamento. O algoritmo QS é apresentado na tabela 3.1.

Ribeiro (1998) prova que se as condições de convergência do *Q–Learning* forem satisfeitas e se a função de espalhamento $\sigma_t(\cdot)$ decair mais rapidamente que a taxa de aprendizado $\alpha_t(s, a)$, o algoritmo *QS* converge para Q^*. Szepesvári e Littman (1996) mostram que este algoritmo pode ser tratado como um GMDP e também provam a sua convergência. Finalmente, o *QS* foi estendido por Ribeiro, Pegoraro e Costa (2002) para tratar Jogos de Markov (seção 2.7) de maneira similar à qual o algoritmo *Q–Learning* foi estendido pelo Minimax–*Q*. Este novo algoritmo foi chamado Minimax–*QS*.

3.2 Aceleração por Abstração

3.2.1 Abstração Temporal

A abstração temporal tem como principal característica o aprendizado em um nível onde a granularidade das ações é menor: ao invés de aprender utilizando ações básicas como "andar para frente um passo" ou "virar para esquerda", utiliza-se ações mais abstratas como "atravesse a sala de maneira ótima". Isto permite uma diminuição da complexidade do problema. Ribeiro (2000) argumenta que ao aprender leis em vez de ações de controle, a exploração é incrementada. Em seu trabalho um robô móvel utiliza seqüências de ações sem realimentação em vez de ações individuais, obtendo bons resultados.

3.2.2 Abstração Espacial

Para implementar um algoritmo de aprendizado por reforço é necessário escolher alguma representação para a função valor (ou valor-ação) e definir a operação apropriada da função de alteração. A representação tabular em uma matriz com uma entrada separada para cada valor é a maneira mais simples de representar estas funções.

Porém sua aplicação é difícil nos domínios com espaço de estados amplo devido não somente ao aumento requerido de memória com o aumento da dimensão do espaço, mas também o aumento no tempo necessário para o algoritmo convergir. A abstração espacial é realizada quando existe a necessidade de representar a função valor em domínios cujo espaço de estados é muito grande ou contínuo. O uso de aproximadores de funções como Redes Neurais Artificiais (RNA) para representar a avaliação do custo é um dos procedimentos mais comuns, tendo sido utilizado com sucesso por Tesauro (1995) no programa para jogar Gamão chamado TD-Gammon, por Scárdua, Cruz e Costa (2003) no controle de descarregadores de navios, entre outros (MITCHELL, 1997, p. 384).

TD-Gammon

Um dos mais famosos exemplos de um sistema baseado em aprendizado por reforço, o TD-Gammon utiliza uma RNA de Perceptrons Multi-Camadas (HAYKIN, 1999) para representar a função valor V e o algoritmo TD(λ) para modificar os pesos da rede neural, aprendendo a jogar gamão. A rede neural é treinada jo-

gando contra si mesma e recebendo apenas um reforço positivo ao ganhar um jogo. Sem utilizar conhecimento prévio do problema, o TD-Gammon aprendeu a jogar gamão com a mesma proficiência dos melhores jogadores humanos em apenas alguns dias. O sucesso do TD-Gammon incentivou o uso de Perceptrons Multi-Camadas como aproximador da função valor V, sendo esta técnica muito popular entre os pesquisadores da área de Aprendizado por Reforço.

CMAC

Outro modelo muito utilizado para representar a função valor é o chamado *Cerebellar Model Articulation Controller* (CMAC), que foi proposto por Albus (1975b) como um modelo funcional do cerebelo e como um método útil para a aproximação de funções (ALBUS, 1975a). Este modelo tem sido freqüentemente aplicado desde o final dos anos 80, principalmente no controle automático (GABRIELLI; RIBEIRO, 2003).

Um CMAC consiste de um conjunto de entradas com limites multidimensionais finitos que se sobrepõem, geralmente chamadas de telhas (*tiles*). Um vetor de entradas é contido em um número de telhas sobrepostas, bem menor que o número total de telhas. As telhas são dispostas geralmente em camadas deslocadas em relação às outras, com um intervalo fixo. Com isso o espaço de estados é eficientemente particionado entre pequenas regiões, sendo que o tamanho das telhas determina a resolução desta representação.

Por exemplo, um CMAC com entrada bidimensional típica possui um conjunto com C camadas de entradas. As telhas em cada camada são retangulares e organizadas em grade, de maneira a cobrirem todo o espaço de entradas sem se sobreporem. Qualquer entrada excita apenas um campo receptivo de cada camada. As camadas são, geralmente, idênticas em sua organização, mas deslocadas umas em relação às outras no hiperespaço de entradas. A figura 3.1 apresenta um exemplo para um problema com uma única entrada, com cinco camadas de telhas. Pode-se ver que um valor de entrada contínuo (por exemplo, $x = 10$) excita telhas em posições diferentes nas várias camadas (as telhas na quinta posição, da esquerda para a direita, nas camadas 1, 2 e 3 e as telhas na quarta posição para as camadas 4 e 5). A figura 3.2 apresenta um exemplo bidimensional com duas camadas de telhas.

Para um problema com N entradas, utiliza-se um CMAC N-dimensional com as telhas se sobrepondo no hiperespaço de ordem N (cubos para entradas tridi-

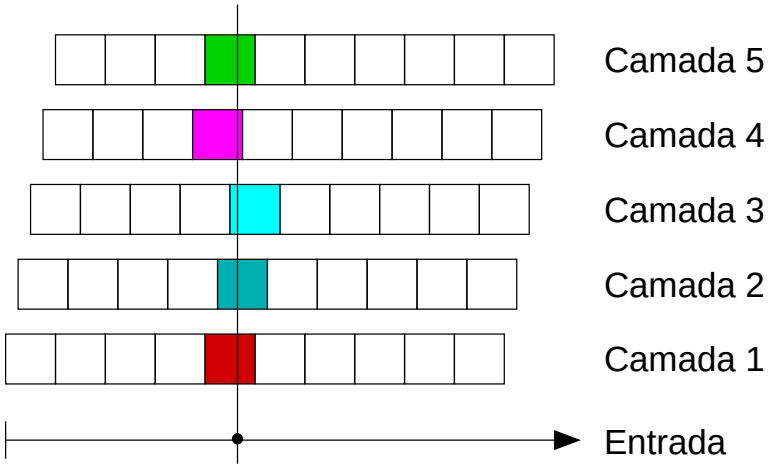


Figura 3.1: Exemplo de camadas de telhas em um CMAC para um problema unidimensional.

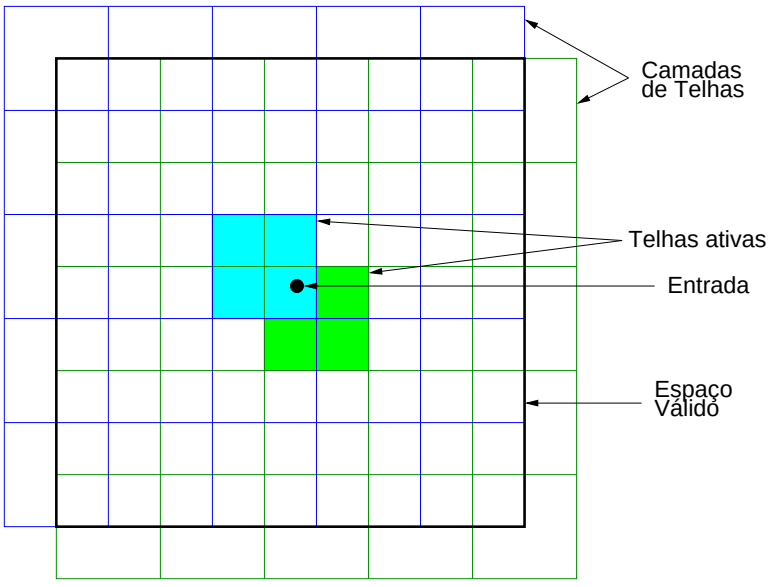


Figura 3.2: Exemplo das camadas de telhas em um CMAC.

mensionais, hipercubos para problemas com quatro entradas e assim por diante).

A resposta dada por um CMAC a uma determinada entrada é a média das respostas dadas pelas telhas excitadas por esse vetor de entrada, não sendo afetada por telhas não excitadas. O primeiro passo para computar a resposta é calcular, para cada entrada S, um vetor A de telhas excitadas pela entrada. Este mapeamento $S \rightarrow A$ faz com que dois vetores de entradas similares gerem um vetor A similar, generalizando a entrada.

Esta capacidade de generalização entre duas entradas similares permite a um CMAC aprender funções matemáticas utilizando um espaço de memória de tamanho razoável, pois o número de elementos do vetor A somados aumenta linearmente com o número de telhas, o qual é usualmente uma fração muito pequena do número total de estados.

O vetor A de um CMAC pode ser usado como uma tabela *look-up* grosseira que usa múltiplas telhas sobrepostas para cobrir o espaço de entrada, sendo mais eficiente no uso de memória quando comparado com as redes neurais por retropropagação ou o uso de tabelas. Sutton (1996) e Sutton e Barto (1998) apresentam diversos resultados utilizando os CMAC para generalização da entrada nos algoritmos *Q–Learning*, Q(λ) e SARSA.

Árvore-kd

Munos e Moore (2002) propõem a discretização não uniforme do espaço de estados utilizando árvores-kd (KNUTH, 1973) e triangulação de Kuhn (MOORE, 1992). Nesta abordagem, a representação da função valor $V(s)$ e da política é discretizada em uma grade de resolução variável, que é armazenada em uma árvore-kd. Neste tipo de árvore, a raiz cobre todo o espaço de estados, geralmente um retângulo (ou um hiper-retângulo em dimensões mais elevadas). Cada nó possui dois filhos que dividem o espaço de estados em duas células, exatamente na metade (figura 3.3-direita).

Para cada nó da árvore-kd é guardado o valor da função $V(s)$ nas quinas dos retângulos que dividem o espaço (figura 3.3-esquerda). Cada retângulo é dividido em dois triângulos (ou 6 pirâmides em três dimensões e $d!$ em d dimensões). Para se computar o valor em um determinado ponto do espaço é realizada uma interpolação linear baseada em triangulação de Kuhn.

Para resolver o problema de aprendizado Munos e Moore (2002) utilizam um

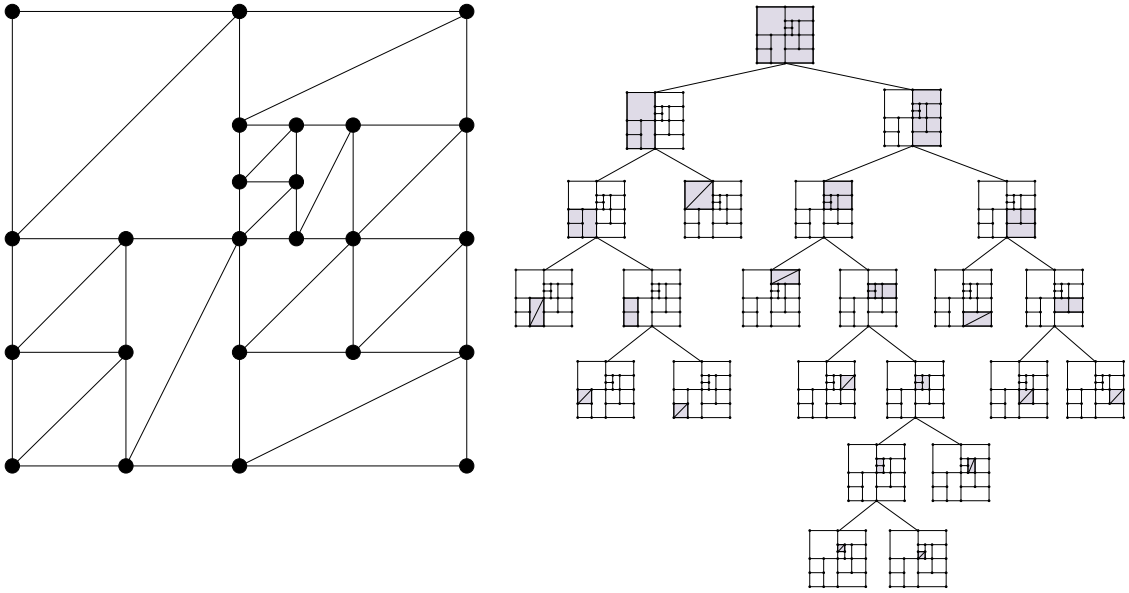


Figura 3.3: Um exemplo de espaço de estados discretizado (esquerda) e a árvore-kd correspondente (MUNOS; MOORE, 2002, pg. 295).

MDP onde o conjunto de estados Ξ armazena as quinas das células. Para cada quina $\xi \in \Xi$ e ação de controle $u \in U$ uma parte da trajetória de controle $x(t)$ pode ser calculada utilizando um método numérico. A integração se inicia no estado ξ, dura um tempo $\tau(\xi, u)$ e termina quando a trajetória entra em uma nova célula. O reforço recebido é definido pela integral:

$$R(\xi, u) = \int_0^{\tau(\xi,u)} \gamma^t r(x(t), u) dt. \tag{3.3}$$

A equação que define o MDP a ser resolvido é:

$$V(\xi) = \max_a \left[\gamma^{\tau(\xi,u)} \sum_{i=0}^{n} T(\xi, u, \xi_i) V(\xi_i) + R(\xi, u) \right], \tag{3.4}$$

onde n é o número de estados discretos e $T(\xi, u, \xi_i)$, a probabilidade de transição entre os estados.

A árvore que armazena a função valor inicia aproximando $V(s)$ a uma grade com estados grosseiros (*coarse*) e é refinada iterativamente (figura 3.4), com a divisão dos estados em dois, segundo algum critério. Para minimizar o erro de aproximação da função de valor, dois critérios podem ser usados para divisão das células: a diferença média do valor das quinas ou a variância do valor médio das quinas. Uma certa porcentagem das células (um parâmetro a ser definido)

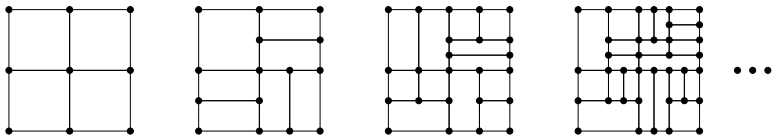


Figura 3.4: Refinamentos sucessivos de um espaço discretizado. (MUNOS; MOORE, 2002, pg. 300).

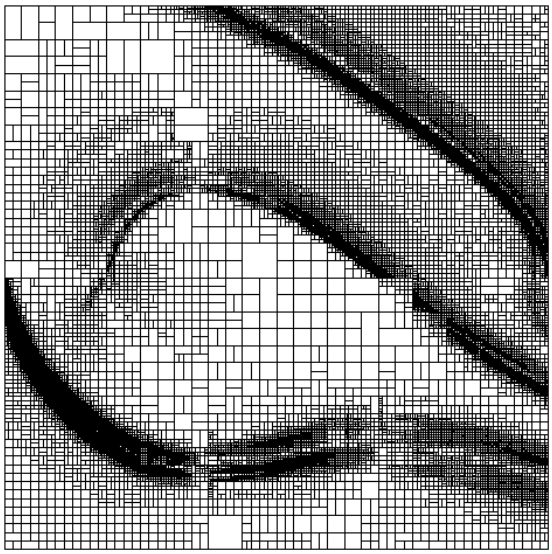


Figura 3.5: O espaço discretizado utilizando o critério da variância para o problema do carro na montanha (MUNOS; MOORE, 2002, pg. 302).

é dividida a cada iteração. O resultado deste algoritmo é uma representação onde estados próximos a mudanças bruscas na função valor são mais subdivididos (figura 3.5). Esta abordagem foi utilizada com sucesso em vários domínios, como o do carro na Montanha, o Pêndulo Invertido e o Ônibus Espacial.

Finalmente, Reynolds (2002, capítulo 5) realiza uma boa revisão dos aproximadores de funções e seus usos para a abstração em Aprendizado por Reforço.

3.3 Aceleração por Distribuição

Esta seção apresenta técnicas que visam a aceleração do aprendizado por reforço por meio de uma abordagem distribuída. As primeiras tentativas visando a distribuição do aprendizado por reforço foram feitas pelos pesquisadores da área de Inteligência Artificial Distribuída (IAD) e de Sistemas Multiagentes (SMA), que

passaram a utilizar, em vez de um único agente, diversos agentes aprendendo ao mesmo tempo (LITTMAN; BOYAN, 1993; WEIß, 1995; WEIß, 1999; SEN; WEIß, 1999).

Diversos autores apresentam taxonomias para os Sistemas Multiagentes sob a perspectiva dos pesquisadores de Aprendizado de Máquina. Stone e Veloso (2000) organizam os SMAs de acordo com o grau de heterogeneidade e a quantidade de comunicação existente entre os agentes, em três combinações: agentes homogêneos não comunicativos, agentes heterogêneos não comunicativos e agentes heterogêneos comunicativos. Tan (1997) estuda o aprendizado de agentes independentes em comparação com agentes cooperativos. Sen e Weiß (1999) distinguem duas categorias principais de SMAs, quanto ao grau de descentralização do aprendizado. Para eles, o aprendizado pode ser centralizado – caso ele seja realizado por um agente único e não necessita de interação com outros agentes – ou descentralizado, onde diversos agentes estão engajados no processo de aprendizado.

De maneira geral, todas as taxonomias têm em comum a divisão em algumas dimensões que podem incluir: comunicação, com agentes isolados/independentes ou cooperativos; método de distribuição do controle, com indivíduos benevolentes ou competitivos, controle estático ou dinâmico; a arquitetura dos agentes – homogêneos ou heterogêneos, reativos ou cognitivos. Como em diversas áreas de IA, esta também apresenta diversas taxonomias diferentes. Por exemplo, Sen e Weiß (1999) utilizam os termos isolado e concorrente como sinônimos (pg. 168), enquanto Banerjee, Sen e Peng (2001) utilizam concorrente como sinônimo de competitivo.

Neste trabalho, um SMA é classificado como independente quando os agentes atuam de maneira isolada e um indivíduo não está ciente da existência de outros agentes no sistema. Estas técnicas são geralmente a generalização das técnicas tradicionais de aprendizado por reforço, para execução distribuída por um número de agentes. Por outro lado, existem os sistemas onde os agentes podem se comunicar, interagindo durante o aprendizado. A comunicação pode ser feita de maneira direta ou indireta (através de um *blackboard* comum), pode ser de baixo ou alto nível, simples ou complexa. Tan (1997) compara a atuação de agentes independentes com cooperativos, mostrando que a cooperação através do

compartilhamento de informações como dados sensoriais, episódios ou as próprias políticas aprendidas levam a melhores resultados do que a atuação isolada dos agentes.

Quanto à heterogeneidade, os agentes podem ser redundantes ou especializados. Novamente, os sistemas com agentes redundantes são o resultado da generalização das técnicas tradicionais de aprendizado por reforço para vários agentes. Os agentes benevolentes comunicam-se com o objetivo de atingir a melhor atuação em comum, enquanto os competitivos também se comunicam, mas possuem objetivos diferentes uns dos outros.

Alvares e Sichman (1997) apresentam algumas características que diferenciam os SMA Reativos (SMAR) dos Cognitivos. Entre elas, podem ser citadas:

- Um SMAR não representa explicitamente o seu conhecimento, o ambiente, nem os outros agentes da sociedade. O conhecimento de um agente se manifesta através do seu comportamento, que se baseia no que é percebido a cada instante.
- Um SMAR não mantém um histórico de suas ações, sendo que a escolha das ações futuras independe do resultado de uma ação passada.
- A organização dos SMAR tem inspiração etológica, baseada na observação do comportamento de insetos como formigas (BONABEAU; DORIGO; THERAULAZ, 2000), abelhas (MATARIC, 1998), aranhas (BROOKS, 1986; BROOKS, 1991; BROOKS, 1996) e mais recentemente, na observação de mamíferos como ratos (JOSHI; SCHANK, 2003).
- Os sistemas reativos são geralmente formados por um grande número de agentes pouco complexos.

Atualmente, o termo Aprendizado por Reforço Multiagente (*Multi Agent Reinforcement Learning* – MARL) tem sido utilizado para descrever as técnicas de aprendizado por reforço distribuídas (TAN, 1997; PEETERS; VERBEECK; NOWÉ, 2003). A seguir, são apresentadas algumas destas técnicas, que podem ser caracterizadas como Sistemas Multiagentes Reativos por possuírem muitas de suas características.

3.3.1 *Q–Learning* Distribuído

DQL Um algoritmo de Aprendizado por Reforço Distribuído é o *Q–Learning Distribuído* (*Distributed Q–Learning* – DQL), proposto por Mariano e Morales (2000b, 2001). O DQL é uma generalização do algoritmo *Q–Learning* tradicional descrito na seção 2.4, na qual, em vez de um único agente, vários agentes independentes são usados para o aprendizado de uma única política.

No DQL, todos os agentes têm acesso a uma mesma cópia temporária da função de avaliação do par estado-ação $Q(s_t, a_t)$, chamada $Qc(s_t, a_t)$, que é usada para decidir qual ação executar. No início de um episódio, o algoritmo copia a tabela $Q(s_t, a_t)$ original para uma única tabela $Qc(s_t, a_t)$. Todos os agentes passam então a procurar a solução utilizando os valores armazenados na cópia. Ao realizarem suas ações, todos os agente atualizam a mesma tabela Qc de acordo com a equação (3.5):

$$Qc(s_t, a_t) \leftarrow Qc(s_t, a_t) + \alpha \left[\gamma \max_{a_{t+1}} Qc(s_{t+1}, a_{t+1}) - Qc(s_t, a_t) \right], \tag{3.5}$$

onde:

- s_t é o estado atual,
- a_t é a ação realizada em s_t,
- $r(s_t, a_t)$ é o reforço recebido após realizar a_t em s_t,
- s_{t+1} é o novo estado,
- γ é o fator de desconto ($0 \leq \gamma < 1$) e
- α é a taxa de aprendizagem ($0 < \alpha < 1$).

Estes agentes exploram opções diferentes em um ambiente comum e, quando todos os agentes terminam uma execução (encontrando o objetivo, por exemplo), suas soluções são avaliadas e todos os pares estado-ação utilizados na melhor solução recebem uma recompensa que modifica seus valores. Esta atualização é feita na tabela $Q(s_t, a_t)$ original, de acordo com a equação (3.6), que é similar à

Inicialize $Q(s, a)$ arbitrariamente.
Repita:
  Copie $Q(s, a)$ para $Qc(s, a)$.
  Repita (para m agentes):
    Inicialize o estado s.
    Repita:
      Selecione uma ação a.
      Execute a ação a.
      Receba o reforço $r(s, a)$ e observe o próximo estado s'.
      Atualize os valores de $Qc(s, a)$ de acordo com a regra:
        $Qc(s, a) \leftarrow Qc(s, a) + \alpha[\gamma \max_{a'} Qc(s', a') - Qc(s, a)]$
      Atualize o estado $s \leftarrow s'$.
    Até que s seja um estado terminal.
  Até que todos os m agentes atingido o estado terminal.
  Avalie as m soluções.
  Atribua a recompensa para todos os pares estado-ação
  presentes na melhor solução encontrada utilizando:
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$, $a = a_t$ e $a' = a_{t+1}$.

Tabela 3.2: O algoritmo *Q–Learning Distribuído* (ROMERO; MORALES, 2000).

regra de aprendizado do *Q–Learning* (equação (2.11)).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \quad (3.6)$$

Caso o algoritmo seja usado para um problema onde não existe um estado terminal definido, duas abordagens para determinar a parada podem ser utilizadas: primeiro, considerar que um episódio é formado por um número fixo de transições de maneira que, ao realizar todas as transições possíveis, o sistema se encontra no estado terminal; ou determinar previamente um estado a ser considerado como terminal. O algoritmo de DQL é apresentado na tabela 3.2.

O algoritmo DQL foi posteriormente estendido para problemas de otimização com multiplos objetivos. Estes problemas se caracterizam por não possuir uma solução ótima única, mas um conjunto de soluções alternativas igualmente eficientes, chamadas de conjunto de Pareto Ótimo. Este novo algoritmo foi chamado *Q–Learning* Distribuído Multi Objetivo (*Multiobjective Distributed Q–Learning* – MDQL) (MARIANO; MORALES, 2000a; ROMERO; MORALES, 2000).

O algoritmo DQL foi utilizado com sucesso em diversos problemas de otimização, como o dos robôs móveis e o dos robôs se movendo em ambientes afetados por forças externas (como, por exemplo, um avião que se move com uma força de vento agindo sobre ele) (MARIANO; MORALES, 2000b), enquanto o MDQL está sendo utilizado para problemas de otimização com múltiplos objetivos como o da distribuição de recursos hídricos para irrigação (MARIANO-ROMERO, 2001).

3.3.2 Dyna–Q Multiagente

M-Dyna-Q

Outra proposta de extensão distribuída para uma técnica de aprendizado por reforço foi feita por Weiß (2000). Ele propôs uma variante multiagente ao algoritmo Dyna–Q chamada M–Dyna–Q, onde múltiplos agentes comunicam-se para realizar planejamento conjunto, apresentar comportamentos reativos comuns e aprender conjuntamente.

Nesta proposta, a atividade dos agentes é realizada em ciclos de atividades de seleção de ação e de aprendizado. Cada ciclo é executado ou no modo hipotético ou no modo real, sendo que os agentes alternam entre os dois modos. O modo real (que é executado rapidamente) corresponde ao "comportamento reativo", enquanto o modo hipotético (mais lento) corresponde ao "comportamento deliberativo".

Os agentes decidem conjuntamente qual a próxima ação a ser tomada (resultante do próximo estado real ou de um novo estado hipotético). No caso de operar no modo real, esta ação é tomada com base na função valor distribuída entre os agentes, e no caso do modo hipotético, com base no modelo do ambiente associado aos agentes. Durante o aprendizado, os agentes ajustam tanto o modelo do mundo quanto as suas estimativas da função valor.

De acordo com a proposta M–Dyna–Q, cada agente $A_i \in Ag$ (onde $Ag = \{A_1, \cdots, A_n\}$ denota um conjunto finito de agentes disponíveis no MAS) mantém, para cada estado, uma estimativa da utilidade de cada ação para realização do objetivo. Assim, um agente A_i mantém para cada estado s e para cada uma de suas ações a_i^j a função valor–ação $\hat{Q}_i^j(s)$, que expressa a estimativa da qualidade da ação. A seleção da ação é realizada com base nesta estimativa. Se os agentes estão operando no modo real, cada agente A_i analisa o estado atual real s, identifica

e comunica aos demais agentes o conjunto $Ac_i^{poss}(s)$ das ações possíveis de serem realizadas imediatamente. A identificação da ação potencial pode ser feita por um agente independentemente dos outros agentes com base no seu conhecimento sobre s (inclusive de maneira paralela). A ação a ser realizada é então selecionada entre todas as ações anunciadas, seguindo uma política de seleção de ação dos agentes. Uma política proposta por Weiß (2000) é que a seleção de uma ação a_i^j é proporcional à estimativa da utilidade de todas as ações anunciadas:

$$\Pi(s, a_i^j) = \frac{e^{\hat{Q}_i^j(s)}}{e^{\sum_{a_i^j} \hat{Q}_i^j(s)}}. \tag{3.7}$$

onde:

- s é o estado atual,
- a_i^j é a ação realizada em s pelo agente i,
- $\hat{Q}_i^j$ é a estimativa da função valor–ação, e
- $\Pi : \mathcal{S} \times \mathcal{A}_i$ define a probabilidade de se executar a ação a_i^j quando o agente i se encontra no estado s.

No modo hipotético os agentes escolhem aleatoriamente um estado s entre todos os estados já visitados no modo real e selecionam uma ação entre todas as que já foram realmente executadas nesse estado de acordo com a equação (3.7), simulando atividades reais. Este modo de operação corresponde a executar o algoritmo Dyna–Q de um agente único nos estados já visitados, e tem a intenção de permitir aos agentes aprenderem sem desperdiçar comunicação entre eles.

O aprendizado em comum é realizado pelos agentes através do ajuste dos valores estimados de suas ações. Ao executar uma ação a_i^j no estado s, que resulta no estado s' um agente A_i recebe de todos os agentes que podem executar ações no estado s' a estimativa da utilidade do estado s'. A_i atualiza a sua estimativa $\hat{Q}_i^j(s)$ utilizando a equação:

$$\hat{Q}_i^j(s) \leftarrow \hat{Q}_i^j(s) + \alpha \left[r(s, a_i^j) + \gamma \max_{a_k^l} \hat{Q}_k^l(s') - \hat{Q}_i^j(s) \right], \tag{3.8}$$

onde $\max_{a_k^l} \hat{Q}_k^l(s')$ define uma estimativa de utilidade global para o estado s', que corresponde ao máximo de todas estimativas. Esta regra de aprendizado repre-

senta uma implementação multiagente direta do algoritmo *Q–Learning* padrão, onde o valor foi distribuído entre múltiplos agentes.

Finalmente, enquanto o aprendizado é realizado tanto no modo hipotético quanto no real, a atualização do modelo do mundo é realizada apenas no estado real, o que é sensato, pois permite modificar o modelo somente quando são observados os efeitos das experiências reais.

3.3.3 Otimização por Colônia de Formigas

Inteligência de Enxame

Nos últimos anos, o uso da inteligência de enxame (*Swarm Intelligence*) para resolver diversos tipos de problemas atraiu uma atenção crescente da comunidade de Inteligência Artificial (BONABEAU; DORIGO; THERAULAZ, 1999; BONABEAU; DORIGO; THERAULAZ, 2000; DORIGO, 2001).

Baseada na metáfora do inseto social para resolver problemas, a inteligência de enxame é uma abordagem que estuda a emergência da inteligência coletiva em grupos de agentes simples, enfatizando a flexibilidade, a robustez, a distribuição, a autonomia e as interações diretas ou indiretas entre agentes.

Os métodos mais comuns são baseados na observação do comportamento das colônias de formigas. Nestes métodos, um conjunto de agentes simples, chamados formigas (*Ants*), cooperam para encontrar soluções boas para problemas de otimização combinatória. Algoritmos de otimização e controle inspirados por estes modelos de cooperação se tornaram conhecidos nos últimos anos como Otimização por Colônia de Formigas (*Ant Colony Optimization* – ACO).

A inteligência de enxame pode ser vista como um novo paradigma para controle e otimização, e pode ser comparada ao paradigma das Redes Neurais Artificiais. "Uma colônia de formigas é um sistema 'conexionista', isto é, as unidades individuais estão conectadas de acordo com um determinado padrão"(BONABEAU; DORIGO; THERAULAZ, 2000). Algumas diferenças que podem ser notadas entre as RNAs e os algoritmos de enxame são: a mobilidade das unidades, que no ACO podem ser robôs móveis ou *softbots* que se movem na Internet; a natureza dinâmica do padrão de conectividade; o uso do ambiente como um meio de coordenação e de comunicação; e o uso do feromônio – hormônio deixado como rastro por formigas que descobrem trajetos novos, informando às outras formigas

se um trajeto é bom ou não – que facilita o projeto de sistemas de otimização distribuídos.

Pesquisadores estão aplicando técnicas da inteligência de enxame nos campos mais variados, dos sistemas da automação à gerência de processos de produção. Alguns deles são:

- Problemas de Roteamento (SCHOONDERWOERD et al., 1997): usando o paradigma da inteligência de enxame é possível inserir formigas artificiais nas redes de comunicações, de modo que possam identificar nós congestionados. Por exemplo, se uma formiga atrasar-se porque atravessou uma parte altamente congestionada da rede, as entradas correspondentes da tabela de roteamento serão atualizadas com um aviso. O uso de algoritmos de formigas em problemas das redes de comunicação ou do roteamento de veículos e logística é chamado Roteamento de Colônia de Formigas (*Ant Colony Routing* – ACR).

- Problemas de otimização combinatoriais, tais como o problema do caixeiro viajante (*Travelling Salesman Problem* – TSP) (DORIGO; GAMBARDELLA, 1997) e o problema da atribuição quadrática (MANIEZZO; DORIGO; COLORNI, 1995): as técnicas para resolver estes problemas foram inspiradas na maneira pela qual as formigas buscam alimento e são chamadas Otimização de Colônia de Formigas (*Ant Colony Optimization* – ACO).

- Problemas de otimização com múltiplos objetivos: Mariano e Morales (1999a) estenderam o algoritmo *Ant-Q* para tratar problemas que tenham um conjunto de soluções alternativas igualmente eficientes. O chamado *M0-Ant-Q* foi usado para modelar a distribuição de recursos hídricos para irrigação (MARIANO; MORALES, 1999b).

- Em diversos problemas de robótica, no aprendizado da coordenação de robôs (KUBE; ZHANG, 1994) e na alocação de tarefas de maneira adaptativa (KUBE; ZHANG, 1997). Em particular, o sistema Ant-ViBRA foi usado no planejamento de rotas visando minimizar a movimentação de manipuladores robóticos envolvidos em diferentes tarefas de montagem (BIANCHI; COSTA, 2002a).

Ant Colony System

O algoritmo de otimização por colônia de formigas mais conhecido é chamado *Ant Colony System* – ACS. Ele foi proposto inicialmente por Dorigo e Gambardella (1997) para resolver o Problema do Caixeiro Viajante (*Travelling Salesman Problem* – TSP), no qual diversas formigas viajam entre cidades e o trajeto mais curto é reforçado. O ACS pode ser interpretado como um tipo particular da técnica de aprendizagem distribuída por reforço, em particular uma abordagem distribuída aplicada ao *Q–Learning*. No restante desta seção o problema do caixeiro viajante é usado para descrever o algoritmo.

O conceito mais importante do ACS é o $\tau(r, s)$, chamado *feromônio*, que é um valor real positivo associado à aresta (r, s) de um grafo. É a contraparte existente no ACS dos valores Q utilizados no algoritmo de aprendizagem *Q–Learning*, e indica quão útil é mover-se para a cidade s estando na cidade r. Os valores $\tau(r, s)$ são atualizados em tempo de execução pelas formigas artificiais. O feromônio age como uma memória, permitindo que as formigas cooperem indiretamente.

Outro conceito importante é a *heurística* $\eta(r, s)$ associada à aresta (r, s), que representa uma avaliação heurística de quais movimentos são melhores. No TSP $\eta(r, s)$ é o inverso da distância δ de r a s, $\delta(r, s)$.

Uma formiga k posicionada na cidade r move-se para a cidade s usando a seguinte regra, chamada regra de transição do estado:

$$s = \begin{cases} \arg \max\limits_{u \in J_k(r)} \tau(r, u) \cdot \eta(r, u)^\beta & \text{se } q \leq q_0, \\ S & \text{caso contário,} \end{cases} \tag{3.9}$$

onde:

- β é um parâmetro que pesa a importância relativa do feromônio aprendido com relação aos valores heurísticos da distância ($\beta > 0$).
- $J_k(r)$ é a lista das cidades que faltam ser visitadas pela formiga k, onde r é a cidade atual. Esta lista é usada para restringir a uma única visita das formigas para cada cidade – uma restrição do enunciado do TSP.
- q é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e q_0 ($0 \leq q_0 \leq 1$) é o parâmetro que define a taxa de exploração/explotação, segundo uma política $\epsilon - Greedy$.

- S é uma variável aleatória selecionada de acordo com uma distribuição de probabilidade dada por:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,u)] \cdot [\eta(r,u)]^\beta}{\sum\limits_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^\beta} & \text{se } s \in J_k(r), \\ 0 & \text{caso contário.} \end{cases} \tag{3.10}$$

Esta regra de transição favorece o uso de arestas com uma quantidade grande de feromônio e que tenham um custo estimado baixo (no problema do caixeiro viajante, favorece a escolha da cidade mais próxima). A fim de aprender os valores do feromônio, as formigas atualizam os valores de $\tau(r,s)$ em duas situações: a etapa de atualização local e a etapa de atualização global.

A regra de atualização local do ACS é aplicada em cada etapa da construção da solução, quando as formigas visitam uma aresta e mudam seus níveis de feromônio usando a seguinte regra:

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s), \tag{3.11}$$

onde $0 < \rho < 1$ é um parâmetro, a taxa de aprendizagem.

O termo $\Delta\tau(r,s)$ pode ser definido como: $\Delta\tau(r,s) = \gamma \cdot \max_{z \in J_k(s)} \tau(s,z)$. Usando esta equação, a regra de atualização local torna-se similar à atualização do *Q–Learning*, sendo composta por um termo de reforço e pela avaliação descontada do estado seguinte (com γ como um fator de desconto). A única diferença é que o conjunto de ações disponíveis no estado s ($s \in J_k(s)$) é uma função da história prévia da formiga k.

Uma vez terminada uma excursão completa pelas formigas, o nível τ de feromônio é modificado usando a seguinte regra de atualização global:

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s), \tag{3.12}$$

onde:

- α é o parâmetro da deterioração do feromônio (similar ao fator de desconto no *Q–Learning*) e
- $\Delta\tau(r,s)$ é o reforço, geralmente o inverso do comprimento da melhor excursão. O reforço atrasado é dado somente ao melhor caminho feito pelas

Inicialize a tabela de feromônio, as formigas e a lista de cidades.
Repita:
    Repita (para m formigas):
        Coloque cada formiga em uma cidade inicial.
        Repita:
            Escolha a próxima cidade usando a regra de transição de estados.
            Atualize a lista J_k de cidades a serem visitadas pela formiga k.
            Aplique a atualização local aos feromônios de acordo com a regra:
                $\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s)$
        Até que todas as cidades tenham sido visitadas.
    Até que todas as formigas tenham completado uma excursão.
    Aplique a atualização global aos feromônios de acordo com a regra:
        $\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s)$
Até que algum critério de parada seja atingido.

Tabela 3.3: O algoritmo Ant Colony System (DORIGO; GAMBARDELLA, 1997).

formigas – somente as arestas que pertencem à melhor excursão receberão mais feromônios (reforço).

O sistema ACS trabalha da seguinte maneira: depois que as formigas são posicionadas em cidades iniciais, cada formiga realiza uma excursão. Durante a excursão, a regra de atualização local é aplicada e modifica o nível de feromônio das arestas. Quando as formigas terminam suas excursões, a regra de atualização global é aplicada, modificando outra vez o nível de feromônio. Este ciclo é repetido até que nenhuma melhoria seja obtida ou um número fixo de iterações seja alcançado. O algoritmo ACS é apresentado na tabela 3.3.

O procedimento genérico que define o funcionamento de sistemas de otimização baseados em colônia de formigas pode ser descrito como um meta-algoritmo (ou meta-heurística, como denominado pelos autores). Este procedimento envolve apenas três passos, repetidos sequencialmente até que algum critério de parada seja atingido (DORIGO; GAMBARDELLA, 1997, p. 8):

1. Gere formigas e ative-as: neste passo as formigas são geradas, alocadas nas posições iniciais e executam alguma tarefa, deixando um rastro de feromônio. Ao final deste passo as formigas "morrem".

2. Evapore o feromônio: ao término da tarefa das formigas, o sistema sofre

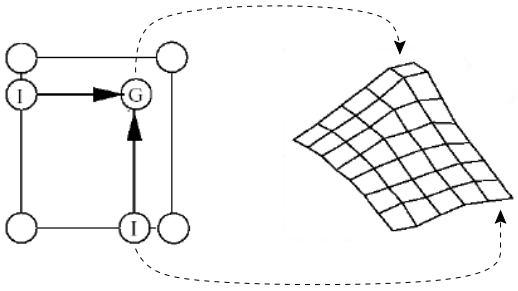


Figura 3.6: O grafo que identifica uma sub-tarefa (esquerda) e os valores da função que é a solução para a sub-tarefa (direita) (DRUMMOND, 2002).

uma atualização global com a evaporação de feromônio – serve para atenuar a memória dos caminhos menos usados.

3. Ações *daemon*: outras tarefas, como observar o comportamento das formigas, ou coleta de informações globais, podem ser executadas.

A partir desta descrição geral, diversos sistemas foram construídos, como o ACS, o *Ant–Q* (GAMBARDELLA; DORIGO, 1995), o *Ant System* e outros (DORIGO; CARO; GAMBARDELLA, 1999, p. 11).

3.4 Aceleração Baseada em Casos

Drummond (2002) propôs um sistema que realiza a aceleração do aprendizado por meio da reutilização de partes de soluções previamente encontradas. As suas principais contribuições consistem em uma maneira automática de identificar as sub-tarefas e uma proposta de como reutilizá-las em novos problemas. Em seu sistema, cada sub-tarefa é identificada por um grafo e a sua solução é uma função multidimensional no domínio dos números reais (figura 3.6). Os grafos são usados para indexar a base de casos previamente aprendidos, que guarda os valores das funções que solucionam cada sub-tarefa. As funções são aprendidas utilizando técnicas de aprendizado por reforço.

A composição de métodos que solucionam um problema é feita da seguinte maneira: uma vez identificadas as sub-tarefas que compõem o problema, o sistema busca na base de dados as funções que as solucionam e as reutiliza, construindo uma solução completa.

O sistema proposto por Drummond realiza três tarefas principais:

1. A identificação precoce das sub-tarefas de um problema, utilizando técnicas de processamento de imagens. Esta tarefa resulta na construção de um grafo que particiona o problema em sub-tarefas e as identifica.

2. A composição de uma nova solução a partir das sub-tarefas armazenadas na base de casos. Ao identificar as sub-tarefas de um novo problema, o sistema acessa a base de funções previamente aprendidas, indexada por grafos, e compõe uma solução para a nova tarefa.

3. A construção da base de casos. Ao finalizar o aprendizado de um problema, o sistema guarda na base de casos as funções que realizam cada sub-tarefa, indexadas por um grafo.

Por exemplo, no domínio onde um robô móvel pode se movimentar em um ambiente com paredes (figura 3.7), a função valor aprendida utilizando *Q–Learning* apresenta descontinuidades nos mesmos locais das paredes (figura 3.8). Estas descontinuidades existem porque o robô precisa andar uma distância extra para contornar a parede e encontrar um caminho para o estado meta. Pode-se notar que as descontinuidades da função valor na figura 3.8 são facilmente identificadas pela visão humana, indicando que estas características podem ser extraídas utilizando técnicas de processamento de imagens. No caso, Drummond propôs o uso da técnica de contornos ativos – também chamados de "Serpentes" (KASS; WITKIN; TERZOPOULOS, 1987) – usada em dezenas de aplicações em diversas áreas do processamento de imagens e da visão computacional, envolvendo desde detecção de bordas, cantos e movimento, até visão estéreo (FISHER, 2003; KERVRANN; TRUBUIL, 2002).

A técnica dos contornos ativos é aplicada ao gradiente da função valor (enfatizando suas descontinuidades), resultando eu uma curva que particiona a função valor nos pontos de maior descontinuidade, segmentando a função e definindo, assim, regiões correspondentes a sub-tarefas do sistema. Um grafo é então utilizado para representar cada sub-tarefa. Os nós do grafo correspondem aos vértices encontrados no polígono gerado pela curvado contorno ativo. Cada nó pode ainda conter uma marca: 'G' para um nó localizado perto do estado meta, 'I' para um estado perto da entrada de uma sala e 'O' para um estado perto de uma saída.

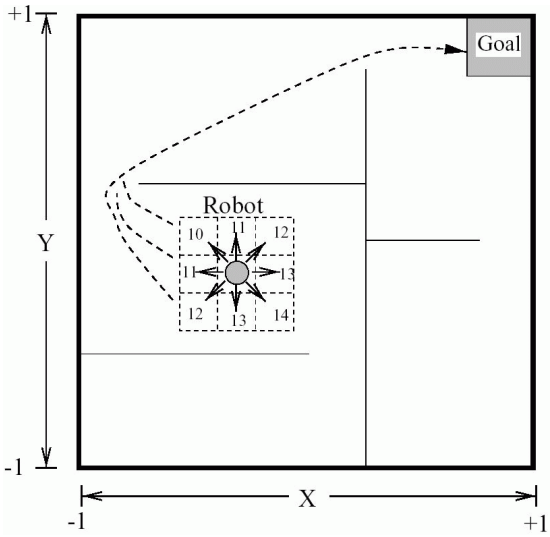


Figura 3.7: Sala com paredes proposta por Drummond (2002), onde o quadrado marca a meta a atingir.

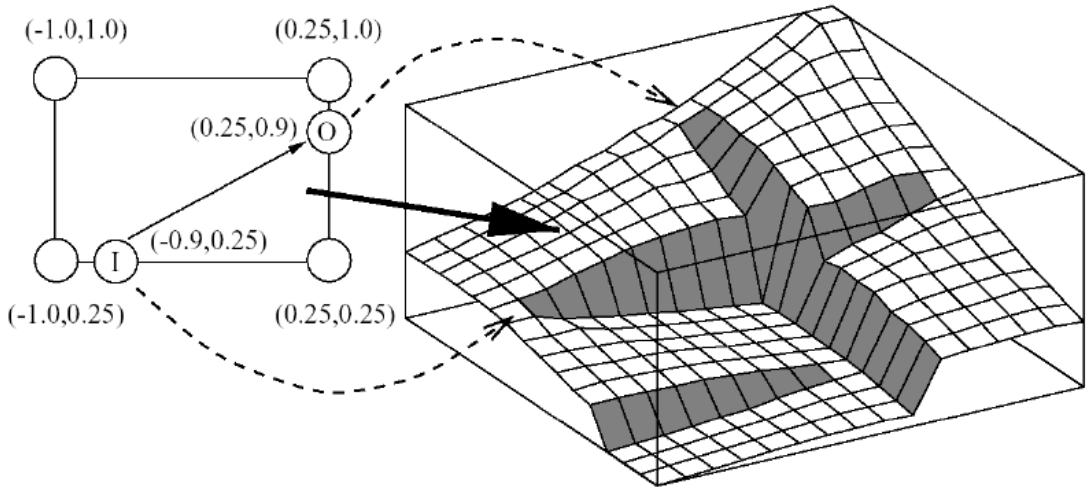


Figura 3.8: A função valor obtida utilizando *Q–Learning* apresenta grandes descontinuidades (área escura) e o grafo construído para a sala superior esquerda (DRUMMOND, 2002, p. 62).

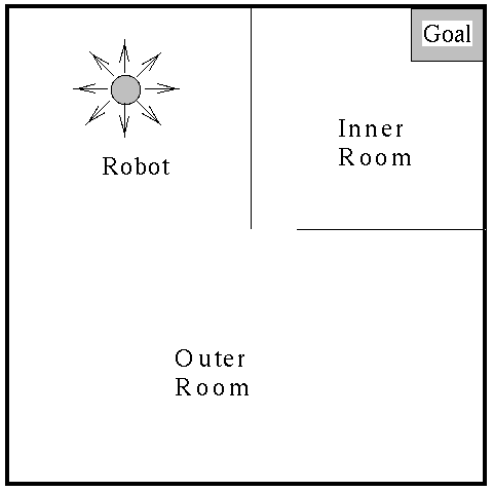


Figura 3.9: Um ambiente com duas salas (DRUMMOND, 2002, p. 66).

A figura 3.9 mostra um exemplo de ambiente e a figura 3.10 ilustra a função valor resultante do aprendizado realizado pelo robô da figura 3.9. A figura 3.11 mostra o gradiente da função valor, o polígono extraído pela curva do contorno ativo e o grafo construído (à esquerda). Na direita da mesma figura é mostrada a expansão da curva do contorno ativo até a acomodação.

Ao se encontrar todas as sub-tarefas que compõem um novo problema, o aprendizado pode ser acelerado utilizando informações armazenadas na base de casos. A figura 3.12 mostra o grafo que representa a solução para o problema da figura 3.7, onde as sub-tarefas estão separadas. A partir de cada grafo individual, o sistema busca, na base de casos, uma função valor que solucione a respectiva sub-tarefa.

Após encontrar todas as funções necessárias, elas são fundidas, formando uma solução para a nova tarefa. Para realizar esta composição, as funções guardadas na base de casos sofrem diversas transformações (como rotacões, ampliações e normalização dos valores) para encaixarem no espaço das novas sub-tarefas (figura 3.13).

Finalmente, esta composição é utilizada como uma aproximação inicial para a tabela função valor-ação, e o aprendizado é reinicializado. Dessa maneira, a composição não precisa ser a solução exata para o novo problema, bastando que esteja próxima da solução final para resultar em uma aceleração significativa do aprendizado.

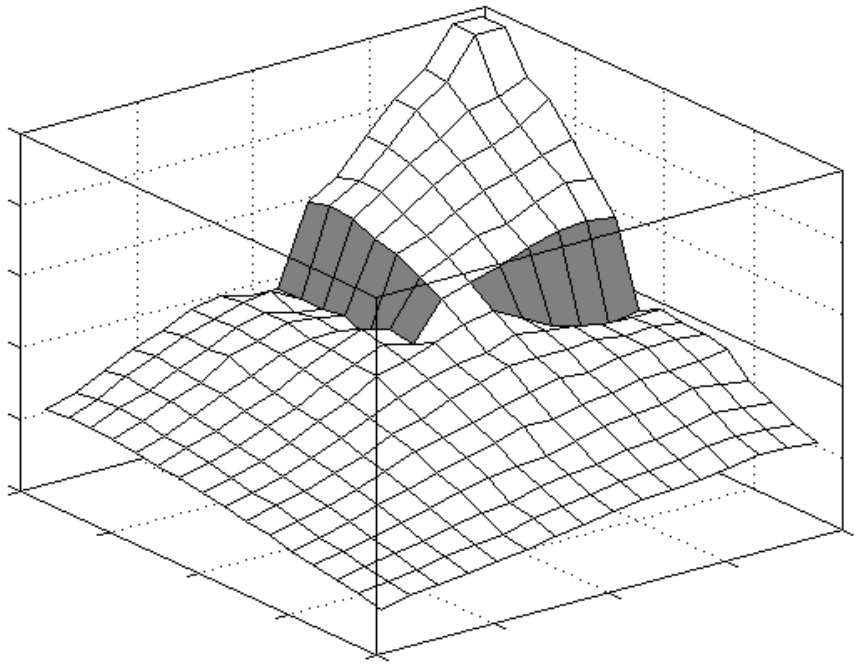


Figura 3.10: A função valor para o ambiente da figura 3.9 (DRUMMOND, 2002, p. 67).

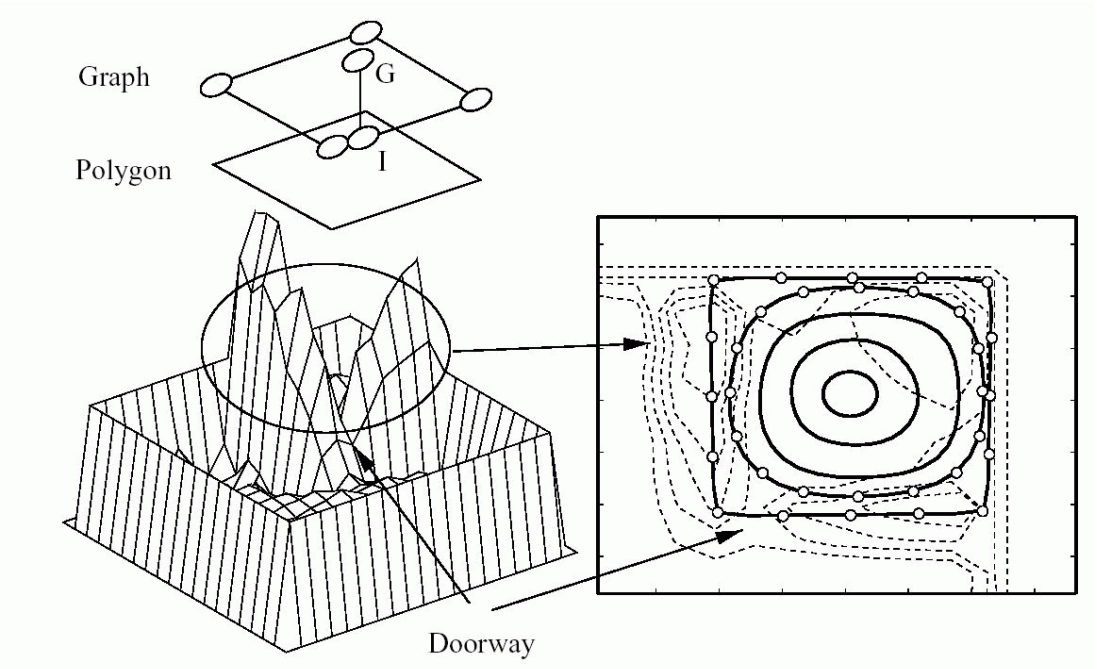


Figura 3.11: O gradiente da função valor da figura 3.10, o polígono extraído e o grafo construído (à esquerda). A expansão da serpente até a acomodação (à direita) (DRUMMOND, 2002, p. 71).

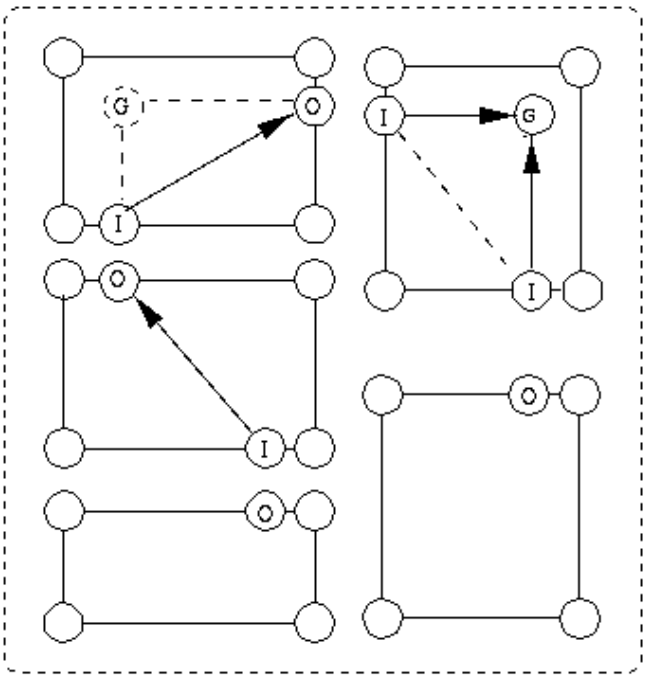


Figura 3.12: Representações gráficas de uma solução mostrando as sub-tarefas do problema da figura 3.7 (DRUMMOND, 2002, p. 63).

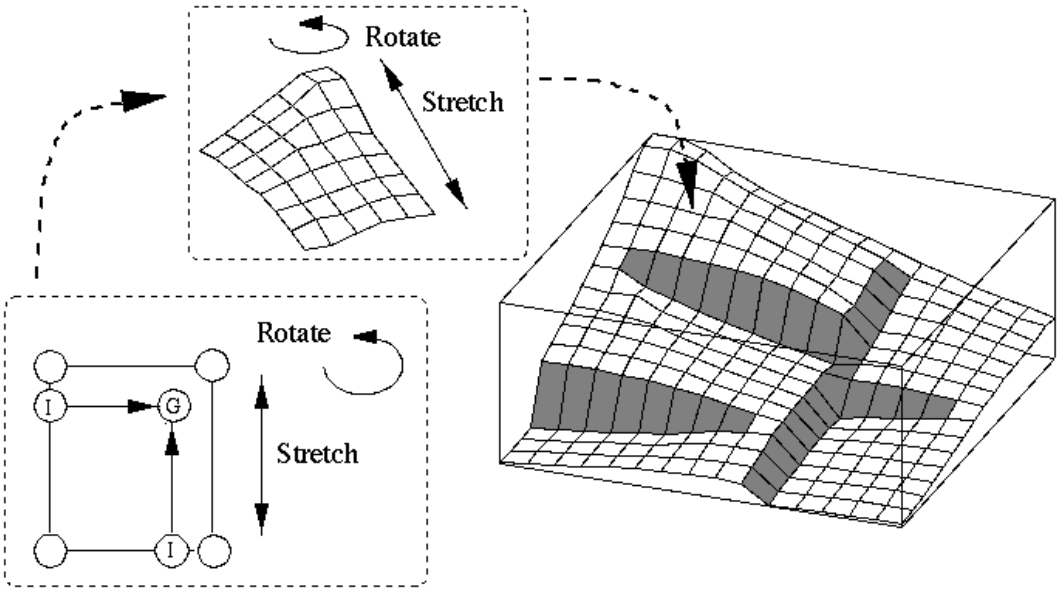


Figura 3.13: A extração e reutilização de uma função armazenada na base de casos. A função é rotacionada e ampliada para encaixar na nova sub-tarefa (DRUMMOND, 2002, p. 64).

A construção da base de casos é feita de maneira muito simples: dado um problema, o algoritmo *Q–Learning* é usado para solucioná-lo. Ao final do aprendizado, o problema é particionado em sub-tarefas. Finalmente, as soluções das sub-tarefas, que são os valores da função valor, são armazenadas, juntamente com os grafos que servirão de índice da base de casos.

3.5 Discussão

Neste capítulo foi apresentado um panorama da área de aceleração do aprendizado por reforço, sem a pretensão de ser completo. Especial atenção foi dada aos trabalhos que serviram de base e inspiração para esta tese, e nos quais muitas idéias tiveram origem. Entre os trabalhos que influenciaram esta proposta, dois tiveram maior peso: a otimização por Colônia de Formigas de Bonabeau, Dorigo e Theraulaz (2000) e a aceleração baseada em casos de Drummond (2002). O primeiro, por sugerir o uso de heurísticas em conjunto com o aprendizado e o segundo, por demonstrar que é possível a utilização de técnicas de processamento de imagens para obter informações sobre a função valor e reutilizar conhecimento previamente adquirido sobre o domínio.

Porém, ambos os trabalhos possuem desvantagens. No caso do trabalho de Bonabeau, Dorigo e Theraulaz (2000), um problema é a existência da lista J_k que guarda as cidades visitadas. Por utilizar esta lista como uma memória de estados passados, o ACO deixa de ser aplicável a um MDP. Além disso, os autores não exploram as possibilidades que o uso de uma heurística oferece. Já no trabalho de Drummond (2002), a utilização do conhecimento armazenado é feita de maneira muito complexa: para cada sub-tarefa, é guardada uma tabela com a função valor que a soluciona; e a composição de resultados prévios exige normalizações nos valores, transformações de escala assimétricas, cálculos de centros de área, etc.

Tomando como base este estudo da aceleração do aprendizado, uma nova proposta inspirada nas virtudes desses dois trabalhos é apresentada no próximo capítulo.

4 Uso de heurísticas para aceleração do aprendizado por reforço

Nos capítulos anteriores fez-se uma revisão da área de Aprendizado por Reforço e dos métodos usados para acelerar este aprendizado, visando definir o contexto no qual está inserida a proposta desta tese, apresentada neste capítulo.

4.1 Introdução

O principal problema abordado neste trabalho é o da aceleração do Aprendizado por Reforço visando seu uso em sistemas robóticos móveis e autônomos, atuando em ambientes complexos. Deseja-se um algoritmo de aprendizado que mantenha as vantagens dos algoritmos de AR, entre elas, a convergência para uma política estacionária, a livre escolha das ações a tomar e o aprendizado não supervisionado, minimizando sua principal desvantagem, o tempo que é necessário para o aprendizado.

Primeira Hipótese

A hipótese central deste trabalho é que existe uma classe de algoritmos de AR que permite o uso de heurísticas para abordar o problema da aceleração do aprendizado. Esta classe de algoritmos é aqui denominada "Aprendizado Acelerado por Heurísticas" (*Heuristically Accelerated Learning* – HAL).

O termo "heurística" tem sido utilizado com diversos significados em IA: foi o nome dado ao estudo de métodos automáticos para a descoberta de provas de teoremas nos anos 50, foi vista como regras para gerar boas soluções em sistemas especialistas e foi usada como o oposto do termo "algoritmo" nos anos 60 (RUSSELL; NORVIG, 1995, p. 94).

Russell e Norvig (2004) definem Heurística como uma técnica que melhora, no caso médio, a eficiência na solução de um problema. Segundo estes autores, "funções heurísticas são a forma mais comum de se aplicar o conhecimento adicional do problema a um algoritmo de busca", sendo, dessa forma, uma maneira de generalização do conhecimento que se tem acerca de um domínio.

De maneira mais formal, um algoritmo da classe HAL pode ser definido como um modo de solucionar um problema modelável por um MDP que utiliza explicitamente a função heurística $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ para influenciar o agente na escolha de suas ações, durante o aprendizado. $H_t(s_t, a_t)$ define a heurística que indica a importância de se executar a ação a_t estando no estado s_t.

A função heurística está fortemente vinculada à política: toda heurística indica que uma ação deve ser tomada em detrimento de outra. Dessa forma, pode-se dizer que a função heurística define uma "Política Heurística", isto é, uma política tentativa usada para acelerar o aprendizado.

Como a heurística é usada somente para a escolha da ação a ser tomada, esta classe de algoritmos difere dos algoritmos de AR propostos até então apenas pela maneira na qual a exploração é realizada[1]. Por não modificar o funcionamento do algoritmo de AR (por exemplo, sem mudar a atualização da função valor-ação Q), esta proposta permite que muitas das conclusões obtidas para algoritmos de AR continuem válidas para os HALs[2].

A utilização da função heurística feita pelos HALs explora uma característica importante de alguns dos algoritmos de AR: a livre escolha das ações de treino. A conseqüência disto é que uma heurística adequada acelera o aprendizado e que, se a heurística não for adequada, o resultado é um atraso que não impede o sistema de convergir para um valor estacionário ótimo – como pode ser visto na seção 5.3.2 e no anexo B – no caso dos sistemas determinísticos.

Uma outra característica importante dos HALs é que a função heurística pode ser modificada a cada iteração. Isto permite que a aceleração seja utilizada em um estágio precoce do aprendizado e modificada toda vez que mais informações

[1]Pode-se dizer que os algoritmos de aprendizado por reforço são um subconjunto dos algoritmos HAL, onde a influência da heurística é sempre nula.

[2]O ACO deixa de ser aplicável a um MDP ao inserir a lista J_k que age como uma memória. Como aqui esta lista não é utilizada, um problema resolvido por um algoritmo HAL continua sendo modelado por MDPs.

1. Existe uma classe de algoritmos de AR chamada HAL que: - Usa uma função heurística para influenciar a escolha das ações. - Usa uma função heurística fortemente vinculada à política. - Atualiza a função heurística iterativamente. 2. Informações existentes no domínio ou em estágios iniciais do aprendizado permitem definir uma política heurística que pode ser usada para acelerar o aprendizado. 3. Existe uma grande quantidade de métodos que podem ser usados para extrair a função heurística.

Tabela 4.1: As três hipóteses estudadas.

se tornem disponíveis.

Segunda Hipótese

Uma segunda hipótese a ser verificada é que as informações existentes no domínio ou em estágios iniciais do aprendizado permitem definir uma política heurística que pode ser usada para acelerar o aprendizado. Definir o que se entende por "situação inicial" é crucial para a validação desta hipótese. Neste trabalho, a situação inicial ocupa uma porcentagem pequena do tempo necessário para o término do aprendizado (10% do tempo, por exemplo), e corresponde à fase onde o aprendizado ocorre de maneira mais rápida.

A definição de uma situação inicial depende do domínio de aplicação. Por exemplo, no domínio de navegação robótica, pode-se extrair uma heurística útil a partir do momento em que o robô está recebendo reforços do ambiente: "após bater em uma parede, use como política heurística a ação que leva para longe da parede". Para indicar a validade desta hipótese, serão apresentados, na seção 5.2, os valores iniciais de algumas características que podem ser encontradas nos problemas estudados.

Terceira Hipótese

A última hipótese é que deve existir uma grande quantidade de métodos que podem ser usados para extrair a função heurística. Visto que existe uma grande quantidade de domínios nos quais o AR pode ser usado e uma grande quantidade de maneiras de extrair conhecimento de um domínio, esta hipótese é facilmente validada. De maneira genérica, os métodos de aprendizado da política heurística serão denominados "Heurística a partir de X" (*"Heuristic from X"*).

Esta denominação é feita pois, da mesma maneira que existem diversas técnicas para se estimar a forma dos objetos na área de Visão Computacional

Inicialize a estimativa da função valor arbitrariamente.
Defina uma função heurística inicial.
Repita:
    Estando no estado s_t selecione uma ação utilizando uma
        combinação adequada da função heurística com a função valor.
    Execute a ação a.
    Receba o reforço $r(s, a)$ e observe o próximo estado s'.
    Atualize a função heurística $H_t(s, a)$ utilizando um método
        "Heurística a partir de X" apropriado.
    Atualize os valores da função valor usada.
    Atualize o estado $s \leftarrow s'$.
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$ e $a = a_t$.

Tabela 4.2: O meta-algoritmo "Aprendizado Acelerado por Heurísticas"

(VC) chamada "Forma a partir de X"(*"Shape from X"*), aqui também existem diversas maneiras de estimar a heurística. No *"Shape from X"* as técnicas individuais são conhecidas como *"Shape from Shading"*, *"Shape from Texture"*, *"Shape from Stereo"*, entre outras (FAUGERAS, 1987; FISHER, 2003). A tabela 4.1 resume as três hipóteses que este trabalho pretende investigar.

O procedimento genérico que define o funcionamento dos algoritmos de "Aprendizado Acelerado por Heurísticas" pode ser genericamente como um meta-algoritmo. Este procedimento envolve quatro passos, repetidos seqüencialmente até que algum critério de parada seja atingido. Este meta-algoritmo é descrito na tabela 4.2.

Qualquer algoritmo que utilize heurísticas para selecionar uma ação, é uma instância do meta-algoritmo HAL. Dessa maneira, pode-se construir algoritmos da classe HAL a partir de qualquer algoritmo de aprendizado por reforço descrito no capítulo 2. Como exemplo, na seção 4.5 será apresentado o algoritmo "*Q–Learning* Acelerado por Heurísticas" (*Heuristically Accelerated Q–Learning* – HAQL), que estende o algoritmo *Q–Learning* (seção 2.4), e diversas outras extensões são apresentadas no 6.

A idéia de utilizar heurísticas em conjunto com um algoritmo de aprendizado já foi abordada por outros autores, como na abordagem de Otimização por Colônia de Formigas apresentada na seção 3.3.3. Porém, as possibilidades deste

uso ainda não foram devidamente exploradas. Em particular, o uso de heurísticas extraídas seguindo uma metodologia similar à proposta por Drummond (2002) parece muito promissora. A seguir é feita uma análise mais aprofundada de cada elemento do meta-algoritmo HAL.

4.2 A função heurística $\mathcal{H}$

A função heurística surge no contexto deste trabalho como uma maneira de usar o conhecimento sobre a política de um sistema para acelerar o aprendizado. Este conhecimento pode ser derivado diretamente do domínio ou construído a partir de indícios existentes no próprio processo de aprendizagem.

A função heurística é usada somente na regra de transição de estados que escolhe a ação a_t a ser tomada no estado s_t. Dessa maneira, pode-se usar, para esta classe de algoritmos, o mesmo formalismo utilizado no AR. Uma estratégia para a escolha das ações é a exploração aleatória $\epsilon - Greedy$, na qual, além da estimativa das funções de probabilidade de transição $\mathcal{T}$ e da recompensa $\mathcal{R}$, a função $\mathcal{H}$ também é considerada. A regra de transição de estados aqui proposta é dada por:

$$\pi(s_t) = \begin{cases} a_{random} & \text{se } q \leq \epsilon, \\ \arg\max_{a_t} \left[\mathsf{F}_t(s_t, a_t) \bowtie \xi H_t(s_t, a_t)^\beta \right] & \text{caso contrário,} \end{cases} \tag{4.1}$$

onde:

- $\mathcal{F} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ é uma estimativa de uma função valor que descreve a recompensa esperada acumulada. Por exemplo, se $\mathsf{F}_t(s_t, a_t) \equiv \hat{Q}_t(s_t, a_t)$ tem-se um algoritmo similar ao *Q–Learning*.

- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ é a função heurística, que influencia a escolha da ação. $H_t(s_t, a_t)$ define a importância de se executar a ação a_t estando no estado s_t.

- $\bowtie$ é uma função matemática, que deve operar sobre números reais e produzir um valor pertencente a um conjunto ordenado (que suporte a operação de maximização).

- ξ e β são variáveis reais usadas para ponderar a influência da função heurística.
- q é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e ϵ ($0 \leq \epsilon \leq 1$) é o parâmetro que define a taxa de exploração/explotação: quanto menor o valor de ϵ, menor a probabilidade de se fazer uma escolha aleatória.
- a_{random} é uma ação selecionada de maneira aleatória entre as ações executáveis no estado s_t.

Prova de Convergência

A primeira conseqüência desta formulação é que as provas de convergência existentes para os algoritmos de AR continuam válidas nesta abordagem. A seguir, são apresentados três teoremas que confirmam esta afirmação e limitam o erro máximo causado pelo uso de uma heurística.

Teorema 1 *Se a técnica de Aprendizado por Reforço na qual um HAL é baseado corresponde a uma forma generalizada do algoritmo* Q–Learning *(seção 2.8), então o valor* F_t *converge para o* F^**, maximizando a recompensa acumulada esperada com desconto, com probabilidade unitária para todos os estados* $s \in \mathcal{S}$*, desde que a condição de visitação infinita de cada par estado-ação seja mantida.*

Esboço da prova: Nos algoritmos HAL, a atualização da aproximação da função valor ao valor estacionário não depende explicitamente do valor da heurística. As condições necessárias para a convergência dos algoritmos *Q–Learning* generalizados que podem ser afetadas com o uso da heurística nos algoritmos HAL são as que dependem da escolha da ação. Das condições apresentadas na seção 2.8, a única que depende da ação escolhida é a necessidade de visitação infinita a cada par estado-ação. Como a equação (4.1) propõe uma estratégia de exploração ϵ*–greedy* que leva em conta a aproximação da função valor $\mathsf{F}(s_t, a_t)$ já influenciada pela heurística, a possibilidade de visitação infinita a cada par estado-ação é garantida e o algoritmo converge. □

A condição de visitação infinita de cada par estado-ação pode ser aceita na prática – da mesma maneira que ela é aceita para os algoritmos de aprendizado por reforço em geral – por meio do uso de diversas outras estratégias de visitação:

- Utilizando uma estratégia de exploração Boltzmann (KAELBLING; LITTMAN; MOORE, 1996) ao invés da exploração ϵ*–greedy*, proposta na equação (4.1).
- Intercalando passos onde se utiliza a heurística com passos de exploração.
- Iniciando cada novo episódio a partir dos estados menos visitados.
- Utilizando a heurística durante um período determinado de tempo, menor que o tempo total de exploração, permitindo que o agente ainda visite estados não apontados pela heurística.

Definição 1 *O erro causado na aproximação da função valor devido ao uso da heurística em um algoritmo de aprendizado por reforço é definido por*

$$L_H(s_t) = \mathsf{F}_t(s_t, \pi^*) - \mathsf{F}_t(s_t, \pi^H), \forall s_t \in S, \tag{4.2}$$

onde $\mathsf{F}_t(s_t, \pi^H)$ *é a estimativa da função valor calculada a partir da política indicada pela heurística,* π^H*.*

Teorema 2 *O erro causado pelo uso de uma heurística que faça com que a política usada coincida com a política ótima em um HAL atuando em um MDP determinístico, com conjuntos de estados e ações finitos, recompensas limitadas* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, fator de desconto* γ *tal que* $0 \leq \gamma < 1$ *e cuja função* $\bowtie$ *utilizada seja a soma, é nulo.*

Prova: Conseqüência direta da equação (4.2) pois, no caso em que a heurística faça com que a política usada coincida com a política ótima, tem-se que $\pi^H = \pi^*$, $\mathsf{F}_t(s_t, \pi_H) = \mathsf{F}_t(s_t, \pi^*)$ e, portanto, o erro $L_H(s_t) = 0$. □

O teorema apresentado a seguir define o limite superior para o erro $L_H(s_t), \forall s_t \in S$.

Teorema 3 *O erro máximo causado pelo uso de uma heurística limitada por* $h_{min} \leq H(s_t, a_t) \leq h_{max}$ *em um HAL atuando em um MDP determinístico, com conjuntos de estados e ações finitos, recompensas limitadas* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, fator de desconto* γ *tal que* $0 \leq \gamma < 1$ *e cuja função* $\bowtie$ *utilizada*

seja a soma, é limitado superiormente por:

$$L_H(s_t) \leq \xi\ [h^{\beta}_{max} - h^{\beta}_{min}]. \tag{4.3}$$

Prova: Esta prova será realizada em partes, pois $\mathsf{F}_t(s_t, a_t)$ pode ser definida de duas maneiras distintas.

Caso 1: $\mathsf{F}_t(s_t, a_t) \equiv Q_t(s_t, a_t)$.

A partir da definição da função valor-ação $Q(s_t, a_t)$ apresentada na equação (2.8) tem-se que:

$$Q_t(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) \hat{V}_t(s_{t+1}), \tag{4.4}$$

onde:

- s_t é o estado atual,
- a_t é a ação realizada em s_t,
- s_{t+1} é o estado resultante de se aplicar a ação a_t estando no estado s_t,
- $T(s_t, a_t, s_{t+1})$ é a função de probabilidade de transição e
- $\hat{V}_t(s_{t+1})$ é aproximação da função valor no instante t para o estado resultante da aplicação de a_t em s_t.

No caso da utilização de um HAL baseado na função $Q_t(s_t, a_t)$ e $\bowtie$ sendo a soma, a equação (4.1) fica:

$$\pi(s_t) = \begin{cases} \arg\max\limits_{a_t} \left[r(s_t, a_t) + \gamma \sum\limits_{s_{t+1} \in \mathcal{S}} T(s_t, a_t, s_{t+1}) \hat{V}_t(s_{t+1}) + \xi H_t(s_t, a_t)^{\beta} \right] & \text{se } q \leq p, \\ a_{random} & \text{c. c.} \end{cases} \tag{4.5}$$

Existe um estado z que causa o erro máximo: $\exists z \in S, \forall s \in S, L_H(z) \geq L_H(s)$. Para este estado z, considere a ação ótima $a = \pi^*(z)$ e a ação indicada pela heurística $b = \pi^H$. O uso da ação $a = \pi^*(z)$ tem como estado resultante x, e o uso de b resulta no estado y. Como a escolha da ação é feita através de uma

política ϵ-*Greedy*, b deve parecer ao menos tão boa quanto a:

$$Q_t(z,a) + \xi H_t(z,a)^\beta \leq Q_t(z,b) + \xi H_t(z,b)^\beta \tag{4.6}$$

Substituindo (4.4) na equação (4.6):

$$\begin{aligned} &r(z,a) + \gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x) + \xi H_t(z,a)^\beta \leq \\ &r(z,b) + \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y) + \xi H_t(z,b)^\beta \end{aligned}$$

$$\begin{aligned} r(z,a) - r(z,b) &\leq \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y) + \xi H_t(z,b)^\beta \\ &\quad - \left[\gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x) + \xi H_t(z,a)^\beta\right] \\ r(z,a) - r(z,b) &\leq \xi \left[H_t(z,b)^\beta - H_t(z,a)^\beta\right] \\ &\quad + \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y) - \gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x). \end{aligned} \tag{4.7}$$

O erro máximo é:

$$\begin{aligned} L_H(z) &= \mathsf{F}_t(z,\pi^*) - \mathsf{F}_t(z,\pi^H) \\ L_H(z) &= Q_t(z,a) - Q_t(z,b) \\ L_H(z) &= r(z,a) + \gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x) - \left[r(z,b) + \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y)\right] \\ L_H(z) &= r(z,a) - r(z,b) \\ &\quad + \gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x) - \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y). \end{aligned} \tag{4.8}$$

Substituindo a equação (4.7) em (4.8), tem-se:

$$\begin{aligned} L_H(z) &\leq \xi \left[H_t(z,b)^\beta - H_t(z,a)^\beta\right] \\ &\quad + \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y) - \gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x) \\ &\quad + \gamma \sum_{x \in \mathcal{S}} T(z,a,x)\hat{V}_t(x) - \gamma \sum_{y \in \mathcal{S}} T(z,b,y)\hat{V}_t(y) \\ L_H(z) &\leq \xi \left[H_t(z,b)^\beta - H_t(z,a)^\beta\right]. \end{aligned} \tag{4.9}$$

Finalmente, como a ação b é escolhida em detrimento da ação a, $H_t(z,b)^\beta \geq H_t(z,a)^\beta$. Como o valor de $\mathcal{H}$ é limitado por $h_{min} \leq H(s_t,a_t) \leq h_{max}$, conclui-se

que:

$$L_H(s_t) \leq \xi\ [h^{\beta}_{max} - h^{\beta}_{min}], \forall s_t \in S. \quad \square \tag{4.10}$$

Esta prova é semelhante à apresentada em (SINGH, 1994, p. 53), onde se demonstra que pequenos erros na aproximação da função valor não podem produzir resultados arbitrariamente ruins em um sistema baseado em iteração de política, quando as ações são selecionadas de maneira gulosa (*greedy*).

Caso 2: $\mathsf{F}(s_t, a_t) \equiv r(s_t, a_t) + \gamma \hat{V}(s_{t+1})$.

Os algoritmos de aprendizado por reforço baseados em iteração de política que utilizam diretamente a função valor $V(s_t)$ para computar a política, maximizam a soma do reforço $r(s_t, \pi(s_t))$ com o valor $V^{\pi}(s_{t+1})$ do estado sucessor (ou sua estimativa), descontado de γ (equação (2.6)):

$$\pi'(s_t) \leftarrow argmax_{\pi(s_t)} \left[r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}) \right].$$

Um HAL baseado nesta classe de algoritmos escolhe a política a ser seguida a partir da equação:

$$\begin{aligned} \pi'(s_t) \leftarrow & argmax_{\pi(s_t)} \Bigg[r(s_t, \pi(s_t)) \\ & + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V^{\pi}(s_{t+1}) + \xi H(s_t, \pi(s_t))^{\beta} \Bigg]. \end{aligned} \tag{4.11}$$

Pode-se notar que a equação (4.11) é similar à equação (4.5) e que todo argumento utilizado para provar este teorema no caso onde $\mathsf{F}(s_t, a_t) \equiv Q(s_t, a_t)$ também é válido para este caso. $\square$

Basta uma pequena manipulação algébrica para provar que o erro $L_H(s_t)$ possui um limite superior definido caso a função $\bowtie$ seja definida por uma das quatro operações básicas (trivial no caso da subtração). Para outras funções, esta prova pode não ser possível.

De maneira geral, o valor da heurística $H_t(s_t, a_t)$ deve ser maior que a variação entre os valores de $\mathsf{F}(s_t, a)$ para um mesmo $s_t \in S$, para poder influenciar a escolha das ações, e deve ser o menor possível, para minimizar o erro $L_H(s_t)$. Caso a função $\bowtie$ usada seja a operação de adição e $\xi = \beta = 1$, a heurística pode

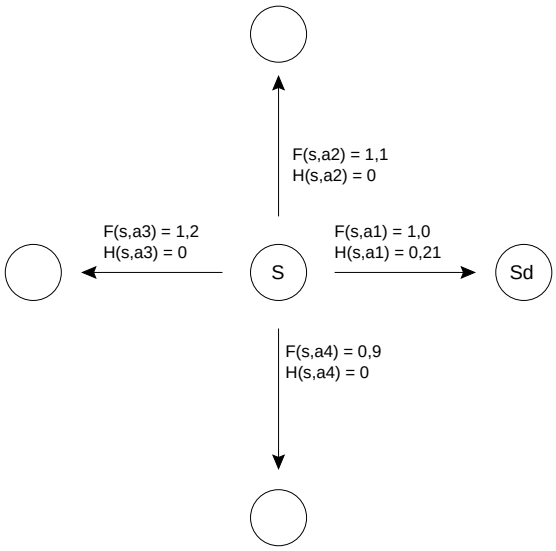


Figura 4.1: Estando no estado S e desejando ir para o estado Sd, o valor de $H(s, a1)$ da ação que leva à $Sd = 0,21$ enquanto para as outras ações é nulo.

ser definida como:

$$H_t(s_t, a_t) = \begin{cases} \max_a \left[\mathsf{F}_t(s_t, a)\right] - \mathsf{F}_t(s_t, a_t) + \eta & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{caso contrário.} \end{cases} \tag{4.12}$$

onde η é um valor pequeno e $\pi^H(s_t)$ é a heurística para a política obtida a partir de um método "Heurística a partir de X".

Por exemplo, se um estado s_t possui 4 ações possíveis, os valores de $\mathsf{F}(s_t, a)$ calculados para as ações são [1,0 1,1 1,2 0,9] e deseja-se que a ação selecionada seja a primeira, pode-se usar $\eta = 0,01$, resultando em $H(s_t, 1) = 0,21$ e igual a zero para as outras ações (a figura 4.1 mostra este exemplo). A heurística pode ser definida de maneira semelhante para outras definições da função $\bowtie$ e valores de ξ e β.

A função $\bowtie$ é o último item introduzido pela formulação apresentada na equação (4.1). Apesar de poder ser utilizada qualquer função que opere sobre números reais (pois tanto a estimativa da função valor quanto a heurística possuem valores reais) e que produza valores pertencentes a um conjunto ordenado, as mais úteis são as operações de adição e de multiplicação. O uso da operação de adição é particularmente interessante, pois permite uma análise da influência dos valores de $\mathcal{H}$ de maneira similar àquela que é feita em algoritmos de busca

informada, como o A^* (HART; NILSSON; RAPHAEL, 1968; RUSSELL; NORVIG, 2004).

Finalmente, a multiplicação também pode ser usada no lugar da função $\bowtie$. Por exemplo, a regra de transição de estado do *Ant Colony System* (seção 3.3.3) utiliza a multiplicação, onde $H(s_t, a_t)$ é ponderada pela constante β. Porém, o uso da multiplicação pode ser problemático em casos nos quais a função de estimação $\mathsf{F}(s_t, a_t)$ (seja ela a função valor-ação Q ou o feromônio τ) pode assumir valores positivos e negativos. Neste caso, ao multiplicar $\mathsf{F}(s_t, a_t)$ por uma heurística positiva, não se pode ter certeza se a importância da ação irá aumentar ou diminuir, pois se o valor de $\mathsf{F}(s_t, a_t)$ for negativo a multiplicação diminui a importância de se executar esta ação.

4.3 "Aprendizado Acelerado por Heurísticas" como uma busca informada

Dado um espaço de hipóteses, o aprendizado de máquina pode ser visto como uma tarefa de busca pela hipótese que melhor explica os dados observados (MITCHELL, 1997, pg. 14). Por exemplo, uma RNA do tipo Perceptron realiza uma busca que define os pesos que melhor ajustam uma função aos exemplos de treinamento. (MITCHELL, 1997, pg. 81).

O aprendizado por reforço pode ser visto como uma tarefa de busca, no espaço de estados e ações (s, a), cujo objetivo é encontrar a política que otimiza a função valor. A introdução da heurística no aprendizado por reforço, proposta neste trabalho, permite a criação de novos algoritmos que passam a realizar uma busca informada.

Admissibilidade

A primeira análise que se pode fazer acerca do "Aprendizado Acelerado por Heurísticas" é quanto à admissibilidade da função heurística $\mathcal{H}$. Uma heurística h é admissível se ela nunca superestimar o custo real h^* para atingir o objetivo. Heurísticas admissíveis são sempre otimistas, já que sempre estimam que o custo de resolver um problema é menor (ou, no máximo, igual) que o custo real.

No caso dos algoritmos HAL, a heurística H indica que uma ação deve ser tomada em detrimento de outra. Para ser admissível, os valores de $H_t(s_t, a_t)$ que

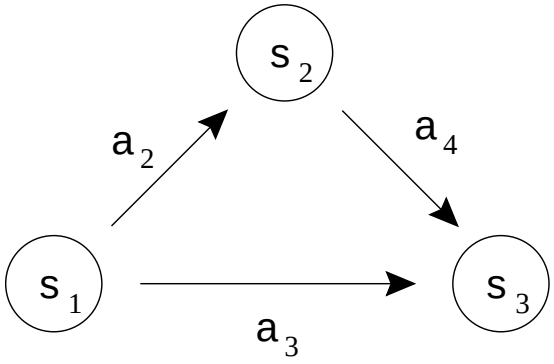


Figura 4.2: Se mover de s_1 diretamente para s_3 fizer parte da política ótima, então H é admissível quando $H(s_1, a_3) \geq H(s_1, a_2)$.

levam a estados que não pertencem à política ótima devem ser menores que os dos $\mathcal{H}$ que levam a estados desejados (pois o algoritmo maximiza a função de estimativa de recompensa acumulada).

Por exemplo, supondo que um sistema esteja no estado s_1, no qual a ação a_2 leva o sistema para o estado s_2 e a ação a_3, para s_3. Suponha ainda que no estado s_2, a ação a_4 leva ao estado s_3. Neste sistema, ir de s_1 diretamente para s_3 faz parte da política ótima (e passar por s_2 aumenta o custo). Neste caso, para H ser admissível, $H(s_1, a_3) \geq H(s_1, a_2)$. A figura 4.2 ilustra este caso.

Formalmente,

$$\text{se } \pi^*(s_t) = \arg\max_{a_t} H(s_t, a_t), \ \mathcal{H} \text{ é admissível.}$$

Otimalidade

Pode-se provar que, no caso de h ser admissível, um algoritmo de busca informada é ótimo. Isto é, ele encontra a solução de melhor qualidade quando existem diversas soluções (a prova é uma conseqüência do Teorema de Bellman). Mas, para um algoritmo da classe HAL, a admissibilidade de H não é suficiente para garantir que a solução encontrada seja ótima. Além disso, é necessário que $|H(s, \pi^*)| > |\max_{a \in \mathcal{A}} \mathsf{F}(s, a)|$ (isto é, o módulo de H da ação ótima deve ser maior que o módulo de F de todas as possíveis ações). Isto significa que, para a heurística forçar a execução de um caminho em um determinado momento, indiferentemente do valor da função de estimativa da recompensa acumulada, H deve ser maior que a recompensa esperada para qualquer ação neste momento, o que ocorre se a heurística usada for a dada na equação (4.12).

Quanto ao comportamento do algoritmo de aprendizado em função do valor da heurística, três casos limites bem conhecidos para os algoritmos de busca informada são interessantes:

- Caso $h \equiv h^*$, isto é h estima perfeitamente o custo h^*, a busca informada converge diretamente para a solução. Isto também acontece no HAL, caso $\pi^H \equiv \pi^*$ e $|H(s, \pi^*)| > |\max_{a \in \mathcal{A}} \mathsf{F}(s, a)|$.

- Caso $h << h^*$, isto é, h subestima em demasia o custo de uma solução, um algoritmo de busca informada executa uma busca mais exaustiva, desperdiçando esforços computacionais. No caso dos algoritmos aqui propostos, o sistema passa a funcionar como se não houvesse a heurística, dependendo apenas do aprendizado para encontrar a solução.

- Quando $h >> h^*$, h superestima o custo h^*, um algoritmo de busca informada deixa de encontrar soluções ótimas pois deixa de expandir caminhos interessantes. No caso dos algoritmos HAL ocorrerá um atraso na convergência. Mas como a heurística é utilizada em conjunto com o aprendizado, o sistema pode aprender a superar heurísticas ruins e encontrar o caminho ótimo, desde que a condição de visitação infinita de cada par estado-ação seja mantida (teorema 1).

Completeza

Finalmente, os algoritmos HAL podem ser analisados quanto a sua completeza. Um algoritmo de busca é completo quando se pode garantir que ele encontra uma solução, caso ela exista. Um algoritmo da classe HAL só não é completo quando existe infinitos nós nos quais $\mathsf{F}(\mathsf{s}, \mathsf{a}) \bowtie H(s, a) \geq \mathsf{F}(s_G, \cdot)$, onde s_G é o estado meta. A única ocasião na qual esta condição pode ocorrer quando H é admissível, é se existirem nós com fator de ramificação infinito (infinitas ações podem ser tomadas) ou existir um caminho com custo finito mas um número infinito de estados (RUSSELL; NORVIG, 1995, p. 100).

4.4 Os métodos "Heurística a partir de X"

Uma das questões fundamentais que este trabalho tenta responder é a de como descobrir, em um estágio inicial do aprendizado, a política que deve ser usada, acelerando o aprendizado. A partir da definição da classe de algoritmos HAL e

da análise da função heurística feita na seção anterior, esta questão traduz-se em como definir a função heurística em uma situação inicial.

A terceira hipótese levantada no início deste capítulo é a da existência de uma grande quantidade de métodos e algoritmos que podem ser usados para definir a função heurística. Esses métodos podem ser divididos em duas classes:

- a classe dos métodos que usam o conhecimento existente sobre o domínio para inferir uma heurística. Estes métodos usam conhecimento *a priori* do domínio para definir a heurística de maneira *ad hoc*, ou reutilizam casos previamente aprendidos, que se encontram armazenados em uma base de casos.

- a classe dos métodos que utilizam indícios que existem no próprio processo de aprendizagem para inferir, em tempo de execução, uma heurística. Sem conhecimento anterior, esta classe de métodos define automaticamente a heurística a partir da observação da execução do sistema durante o processo de aprendizagem. Entre o indícios que podem ser observados os mais relevantes são: a política do sistema em um determinado instante, função valor em um determinado instante e o caminho pelo espaço de estados que o agente pode explorar.

Excluindo a definição das heurísticas de maneira *ad hoc*, os métodos "Heurística a partir de X" funcionam geralmente em dois estágios. O primeiro, que retira da estimativa da função valor F informações sobre a estrutura do domínio e o segundo, que encontra a heurística para a política – em tempo de execução ou a partir de uma base de casos – usando as informações extraídas de F. Estes estágios foram aqui denominados de Extração de Estrutura (*Structure Extraction*) e Construção da Heurística (*Heuristic Composition*), respectivamente (ver figura 4.3).

Os métodos que utilizam uma base de casos construída previamente necessitam de uma fase preliminar, durante a qual o aprendizado é executado sem o uso de heurísticas e que permite a construção da base de casos. O principal problema destes métodos é encontrar características do sistema que permitam indexar a base de casos e que possam ser extraídas de uma situação inicial. Outro problema consiste na adaptação do caso prévio para a situação atual.

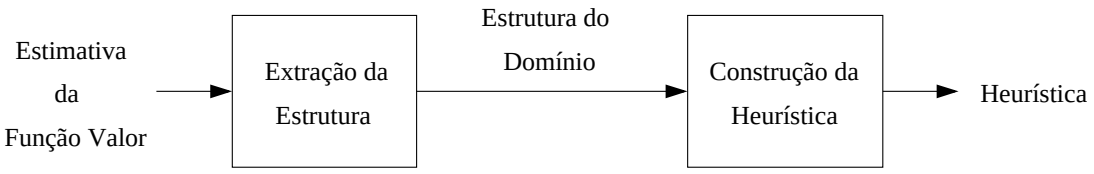


Figura 4.3: Esquema geral dos métodos "Heurística a partir de X".

Já os métodos que definem automaticamente a heurística a partir da observação da execução do sistema durante o processo de aprendizagem têm como principal problema a determinação do momento no qual as informações extraídas são suficientes para a construção de uma heurística adequada, permitindo o início da aceleração.

4.4.1 Extração da Estrutura

Neste trabalho são estudados quatro métodos diferentes para a extração, em tempo de execução, das informações sobre a estrutura existente no problema a ser resolvido: a partir da Política, a partir da função valor, a partir dos Reforços Negativos e a partir de Exploração. A estrutura que se deseja conhecer depende do domínio – pode ser o conjunto de estados permitidos no problema ou mesmo a função de transição de estado – e geralmente reflete restrições existentes no mundo real.

O primeiro método – "Estrutura a partir da Política" – utiliza o algoritmo de Iteração de Política (seção 2.3) para calcular, a partir da política $\pi_t(s_t)$ em um instante t, a função valor $V_t^{\pi}(s_t)$. Isto é feito porque existem indícios de que a política converge mais rapidamente que a função valor (ver anexo A), e por isso deve gerar informações de melhor qualidade que o uso direto de $V_t(s_t)$.

A partir de V_t^{π}, é calculado o gradiente da função valor, ∇V_t^{π}. No caso de um ambiente bidimensional (como no domínio da navegação robótica), este passo corresponde a extrair as bordas que marcam os locais onde existe uma grande variação no valor de $V_t^{\pi}(s_t)$, indicando que algum atributo do ambiente impede a execução de uma ou mais ações. No caso da navegação robótica, as bordas podem indicar as paredes por onde o robô não pode passar. Finalmente, o resultado ∇V_t^{π} é binarizado, utilizando-se um algoritmo de limiarização. A matriz resultante corresponde a uma aproximação do mapa do ambiente.

O segundo método – "Estrutura a partir da Função Valor" – é similar ao primeiro, porém é usada diretamente a função valor $V_t(s_t)$ para encontrar a estimativa do mapa do ambiente.

O método "Estrutura a partir dos Reforços Negativos" constrói uma tabela que guarda apenas os reforços negativos recebidos em cada estado, chamada $R^-(s_t)$. A limiarização desta tabela gera um mapa das posições nas quais o agente recebe mais reforços negativos, o que geralmente reflete os obstáculos em um ambiente. A utilização de uma estrutura separada para as premiações e para penalizações recebidas durante o aprendizado foi proposta por Koenig e Simmons (1996) e também utilizada por Braga (1998) no domínio da navegação robótica. Vale notar que uma tabela que guarde os reforços positivos, chamada R^+, contém informações sobre as ações a executar e também pode ser utilizada para a definição de uma heurística.

O quarto método – "Estrutura a partir de Exploração" – constrói iterativamente uma estimativa da função de probabilidade de transição $\hat{T}_t(s_t, a_t, s_{t+1})$, anotando o resultado de todas as ações realizadas pelo agente. Esta estatística é similar à realizada pelos algoritmos Dyna e *Prioritized Sweeping*, apresentados na seção 3.1.1. No caso de um robô móvel, ao tentar se mover de uma posição para a seguinte, anota-se se esta ação obteve sucesso. Não conseguir realizar uma ação indica a existência de um obstáculo no ambiente. Com o passar do tempo este método gera o mapa do ambiente e ações possíveis em cada estado. Estes quatro métodos são comparados na seção 5.2.

4.4.2 Construção da Heurística

Um novo método automático para compor a heurística em tempo de execução, a partir da estrutura do domínio extraída, é proposto neste trabalho e é chamado "Retropropagação de Heurísticas" (*Heuristic Backpropagation*). Ele propaga, a partir de um estado final, as políticas corretas que levam àquele estado. Por exemplo, ao chegar no estado terminal, define-se a heurística como composta pelas ações que levam de estados imediatamente anteriores a este estado terminal. Em seguida, propaga-se esta heurística para os antecessores dos estados que já possuem uma heurística definida.

Teorema 4 *Para um MDP determinístico cujo modelo é conhecido, o algoritmo de Retropropagação de Heurísticas gera a política ótima.*

Prova: Como este algoritmo é uma simples aplicação do algoritmo de Programação Dinâmica (BERTSEKAS, 1987), o próprio teorema de Bellman prova esta afirmação. □

A Retropopagação de Heurísticas é um algoritmo muito semelhante ao algoritmo de Programação Dinâmica. No caso onde o ambiente é totalmente conhecido, ambos funcionam da mesma maneira. No caso onde apenas parte do ambiente é conhecida, a retropopagação é feita apenas para os estados conhecidos. No exemplo de mapeamento robótico, o modelo do ambiente (mapa) é construído aos poucos. Neste caso, a retropopagação pode ser feita apenas para as partes do ambiente já mapeadas.

A combinação dos métodos de extração de estrutura propostos com a retropropagação das políticas gera quatro métodos "Heurística a partir de X" diferentes: a "Heurística a partir da Política" ($\mathcal{H}$-de-π), a "Heurística a partir da Função Valor" ($\mathcal{H}$-de-V), a "Heurística a partir dos Reforços Negativos" ($\mathcal{H}$-de-R^-) e "Heurística a partir de Exploração" ($\mathcal{H}$-de-E).

Como exemplo de um método baseado em casos para a construção da heurística, pode ser citado a "Reutilização de Políticas", que utiliza as informações extraídas por qualquer um dos métodos propostos – o gradiente da função valor, por exemplo – para encontrar, em uma base de casos previamente construída, uma política que possa ser usada como heurística para um novo problema. Este método, inspirado no trabalho de Drummond (2002), pode ser aplicado em uma grande quantidade de problemas (novamente, Drummond (2002) apresenta resultados nesta direção).

4.5 O algoritmo *Q–Learning* Acelerado por Heurísticas

HAQL Por ser o mais popular algoritmo de AR e possuir uma grande quantidade de dados na literatura para a realização de uma avaliação comparativa, o primeiro algoritmo da classe HAL a ser implementado é uma extensão do algoritmo *Q*–

Learning (seção 2.4). Este novo algoritmo é denominado "*Q–Learning* Acelerado por Heurísticas" (*Heuristically Accelerated Q–Learning* – HAQL).

Para sua implementação, é necessário definir a regra de transição de estados e o método a ser usado para atualizar a heurística. A regra de transição de estados usada é uma modificação da regra $\epsilon - Greedy$ padrão do *Q–Learning* que incorpora a função heurística como uma somatória simples (com $\beta = 1$) ao valor da função valor–ação. Assim, a regra de transição de estados no HAQL é dada por:

$$\pi(s_t) = \begin{cases} a_{random} & \text{se } q \leq \epsilon, \\ \arg\max_{a_t} \left[\hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t)\right] & \text{caso contrário.} \end{cases} \tag{4.13}$$

O valor da heurística usada no HAQL é definido instanciando-se a equação (4.12). A heurística usada é definida como:

$$H(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + \eta & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{caso contrário.} \end{cases} \tag{4.14}$$

A convergência deste algoritmo é garantida pelo teorema 1. Porém, o limite superior para o erro pode ser melhor definido. A seguir são apresentados dois lemas conhecidos, válidos tanto para o *Q–Learning* quanto para o HAQL.

Lema 1 *Ao se utilizar o algoritmo HAQL para a solução de um MDP determinístico, com um conjunto de estados e ações finitos, recompensas limitadas* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, fator de desconto* γ *tal que* $0 \leq \gamma < 1$*, o valor máximo que* $Q(s_t, a_t)$ *pode assumir é igual á* $r_{max}/(1 - \gamma)$*.*

Prova: A partir da equação (2.1) que define o valor cumulativo descontado para o Modelo de Horizonte Infinito e da definição da função valor–ação Q (equação (2.7)), tem-se:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \tag{4.15}$$

$$\begin{aligned} Q^*(s_t, a_t) &= r_t + \gamma V^*(s_{t+1}) \\ &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i}. \end{aligned} \tag{4.16}$$

onde r_{t+i} é a seqüência de reforços recebidos a partir do estado s_t, usando a política π de maneira repetida para selecionar ações e γ é o fator de desconto, com $0 \leq \gamma < 1$.

Assumindo que, no melhor caso, todos os reforços recebidos $r_{t+i} = r_{max}$ em todos os passos, tem-se que:

$$\begin{aligned} \max Q(s_t, a_t) = &\ r_{max} + \gamma r_{max} + \gamma^2 r_{max} + \ldots + \gamma^n r_{max} \\ = &\ \sum_{i=0}^{n} \gamma^i r_{max} \end{aligned} \tag{4.17}$$

Finalmente, no limite, quando $n \to \infty$, tem-se:

$$\begin{aligned} \max Q(s_t, a_t) = &\ \lim_{n\to\infty} \sum_{i=0}^{n} \gamma^i r_{max} \\ = &\ \frac{r_{max}}{1-\gamma} \ \square \end{aligned} \tag{4.18}$$

Caso o reforço positivo seja atribuído apenas ao chegar no estado final, $r_t \leq r_{max}$ e não existem reforços em $t \geq t+1$, conclui-se que $\forall(s_t, a_t), \max Q(s_t, a_t) \leq r_{max}$.

Lema 2 *Ao se utilizar o algoritmo HAQL para a solução de um MDP determinístico, com um conjunto de estados e ações finitos, recompensas limitadas* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, fator de desconto* γ *tal que* $0 \leq \gamma < 1$*, o valor mínimo que* $Q(s_t, a_t)$ *pode assumir é igual a* $r_{min}/(1-\gamma)$*.*

Prova: Assumindo que, no pior caso, todos os reforços recebidos $r_{t+i} = r_{min}$ em todos os passos, pode-se concluir que:

$$\begin{aligned} \min Q(s_t, a_t) = &\ r_{min} + \gamma r_{min} + \gamma^2 r_{min} + \ldots + \gamma^n r_{min} \\ = &\ \sum_{i=0}^{n} \gamma^i r_{min} \end{aligned} \tag{4.19}$$

No limite quando $n \rightarrow \infty$

$$\begin{aligned} \min Q(s_t, a_t) &= \lim_{n\to\infty} \sum_{i=0}^{n} \gamma^i r_{min} \\ &= \frac{r_{min}}{1-\gamma} \ \square \end{aligned} \tag{4.20}$$

Teorema 5 *Ao se utilizar o algoritmo HAQL para a solução de um MDP determinístico, com um conjunto de estados e ações finitos, recompensas limitadas* $(\forall s_t, a_t)\ r_{min} \leq r(s_t, a_t) \leq r_{max}$*, fator de desconto* γ *tal que* $0 \leq \gamma < 1$*, o erro máximo na aproximação da função valor–ação causado pelo uso de uma heurística é limitado superiormente por*

$$L_H(s_t) = \xi \ \left[\frac{r_{max} - r_{min}}{1-\gamma} + \eta\right]. \tag{4.21}$$

Prova: A partir da equação (4.14), tem-se que:

$$h_{min} = 0 \qquad \text{quando } a_t \neq \pi^H(s_t), \text{ e} \tag{4.22}$$

$$h_{max} = \max_{a_t} \hat{Q}(s_t, a_t) - \hat{Q}(s_t, \pi^H(s_t)) + \eta \qquad \text{quando } a_t = \pi^H(s_t). \tag{4.23}$$

O valor da heurística será máximo quando tanto o $\max \hat{Q}(s_t, a_t)$ quanto o $\min \hat{Q}(s_t, a_t)$, $\forall s_t \in S, a_t \in A$ se encontram no mesmo estado s_t. Neste caso

$$h_{max} = \frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma} + \eta. \tag{4.24}$$

Substituindo h_{max} e h_{min} no resultado do teorema 3, tem-se:

$$\begin{aligned} L_H(s_t) &= \xi \ [h^{\beta}_{max} - h^{\beta}_{min}] \\ &= \xi \ \left[\frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma} + \eta - 0\right] \\ &= \xi \ \left[\frac{r_{max} - r_{min}}{1-\gamma} + \eta\right] .\square \end{aligned} \tag{4.25}$$

A figura 4.4 apresenta um exemplo de configuração onde ambos o $\max Q(s_t, a_t)$ e o $\min Q(s_t, a_t)$, se encontram no mesmo estado s_1. Nela, o estado s_2 é o estado terminal; mover para s_2 gera uma recompensa r_{max} e qualquer outro movimento gera uma recompensa r_{min}.

O algoritmo HAQL completo é apresentado na tabela 4.3. Pode-se notar que as únicas modificações se referem ao uso da função heurística para a escolha

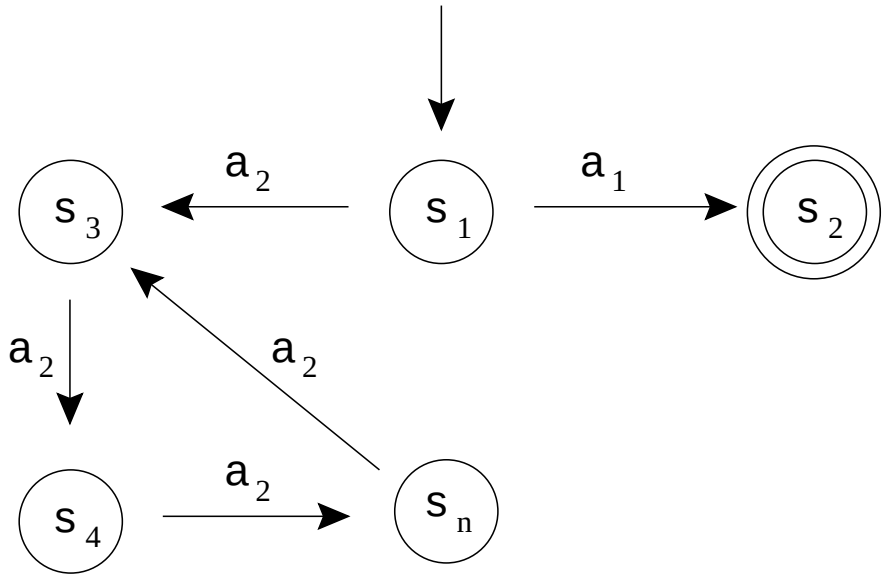


Figura 4.4: O estado s_1 possui ambos os valores, máximo e mínimo, para a função valor-ação Q (a execução da ação a_2 em qualquer estado sempre recebe o reforço negativo mínimo e a execução da ação a_1 em s_1 recebe o reforço máximo).

Inicialize $Q(s, a)$ arbitrariamente.
Repita:
  Visite o estado s.
  Selecione uma ação a utilizando a regra de transição de estado.
  Receba o reforço $r(s, a)$ e observe o próximo estado s'.
  Atualize os valores de $H_t(s, a)$ utilizando um método "$\mathcal{H}$ a partir de X".
  Atualize os valores de $Q(s, a)$ de acordo com a regra de atualização:
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$.
  Atualize o estado $s \leftarrow s'$.
Até que algum critério de parada seja atingido.

onde: $s = s_t$, $s' = s_{t+1}$, $a = a_t$ e $a' = a_{t+1}$.

Tabela 4.3: O algoritmo HAQL.

da ação a ser executada e a existência de um passo de atualização da função $H_t(s_t, a_t)$.

Apesar da função $H_t(s_t, a_t)$ poder ser extraída utilizando qualquer método "Heurística a partir de X", um bom método acelera a computação e aumenta a generalidade deste algoritmo. Em conjunto com o HAQL foi utilizado o método de "Heurística a partir de Exploração" descrito na seção 4.4 para definir a heurística usada. Os resultados da implementação completa deste algoritmo serão apresentados no próximo capítulo.

4.6 Resumo

Neste capítulo foi apresentada a classe de algoritmos denominada "Aprendizado Acelerado por Heurísticas" (*Heuristically Accelerated Learning*), os métodos "Heurística a partir de X" e o algoritmo *Heuristically Accelerated Q–Learning*.

No próximo capítulo são apresentados experimentos utilizando o *Heuristically Accelerated Q–Learning*, que demonstram a eficiência desta proposta.

5 Experimentos no domínio de navegação robótica

Neste capítulo são apresentados estudos realizados com alguns métodos de "Heurística a partir de X" e resultados obtidos utilizando o *Heuristically Accelerated Q–Learning* no domínio dos robôs móveis autônomos, atuando em um ambiente simulado. Os métodos "Heurística a partir de X" estudados são aqueles que estimam a heurística a partir de informações do próprio processo de aprendizado, em tempo de execução.

5.1 O domínio dos robôs móveis

Navegação Robótica

O problema de navegação em um ambiente onde existam obstáculos, com o objetivo de alcançar metas em posições específicas, é uma das tarefas básicas que um robô móvel autônomo deve ser capaz de realizar.

Devido ao fato de que a aprendizagem por reforço requer uma grande quantidade de episódios de treinamento, o algoritmo *Heuristically Accelerated Q–Learning* – HAQL foi testado inicialmente em um domínio simulado. Em todos os experimentos realizados neste capítulo (exceto o da seção 5.4) foi utilizado o domínio onde um robô móvel pode se mover em um ambiente discretizado em uma grade com N x M posições possíveis e pode executar uma entre quatro ações: N, S, E e W (Norte, Sul, Leste e Oeste, respectivamente). O ambiente possui paredes, que são representadas por posições para as quais o robô não pode se mover (ver figura 5.1). Este domínio – chamado *Grid World* – é bem conhecido, tendo sido usado nos experimentos de Drummond (2002), Foster e Dayan (2002) e Kaelbling, Littman e Moore (1996), entre outros.

Um treinamento consiste em uma série de episódios executados em seqüência,

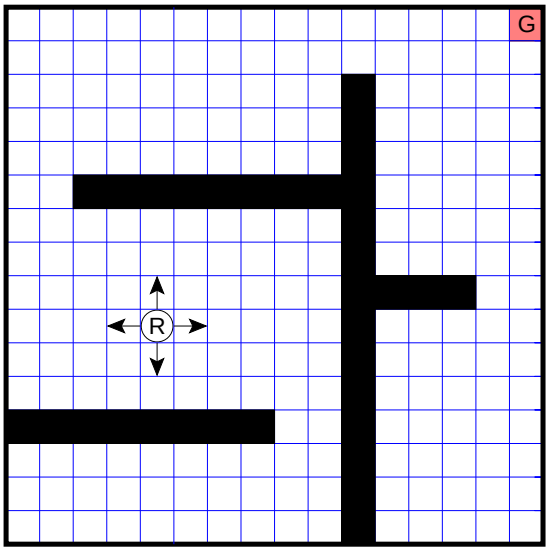


Figura 5.1: Sala com paredes (representadas pelas linhas escuras) discretizada em uma grade de posições (representados pelas linhas mais suaves).

que se iniciam com o posicionamento do robô em uma posição escolhida de maneira aleatória e terminam quando o robô encontra o estado meta (MITCHELL, 1997, p. 376).

Existem diversos algoritmos bem conhecidos que permitem computar a política ótima para este ambiente. A figura 5.2 apresenta o resultado do algoritmo de Iteração de Política (tabela 2.2). O problema foi solucionado em 38 iterações, demorando 811 segundos.

A maioria dos experimentos apresentados neste capítulo foi codificada em Linguagem C++ (compilador GNU g++) e executada em um Microcomputador Pentium 3-500MHz, com 256MB de memória RAM e sistema operacional Linux. O único caso no qual foi usado outra configuração é o da seção 5.4, onde foi utilizado um Microcomputador Pentium 4m-2.2GHz com 512MB de memória RAM e a mesma configuração de software.

5.2 Uso de métodos "Heurística a partir de X" em tempo de execução

Esta seção possui dois objetivos correlatos. Primeiro, apresentar indícios que permitam aceitar como válida a segunda hipótese proposta no capítulo 4, de que as

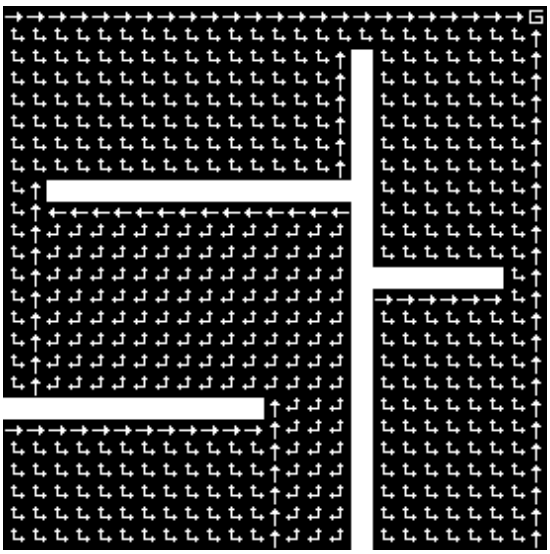


Figura 5.2: Política ótima para um robô móvel em um ambiente com 25 x 25 posições e algumas paredes. Setas duplas significam que, em uma mesma posição, não faz diferença qual das duas ações a tomar.

informações existentes nos estágios iniciais do treinamento permitem definir uma heurística que pode ser usada para acelerar o aprendizado. O segundo objetivo é determinar qual dos métodos de extração de estrutura apresentados na seção 4.4 é mais eficiente para o problema de navegação de um robô móvel, podendo ser usado com bons resultados a partir dos primeiros episódios de treinamento.

Para tanto, foram realizados experimentos, utilizando o algoritmo *Q– Learning*, baseados no domínio apresentado na figura 5.1, com os seguintes parâmetros: $\gamma = 0,99$, $\alpha = 0,1$ e a taxa de exploração/explotação igual a 0,1. Os reforços utilizados foram: 10 para quando o robô chega ao estado meta, que se encontra no canto superior direito, e -1 quando realiza uma ação.

5.2.1 Extração de estrutura

Foram utilizados os quatro métodos de extração de estrutura descritos na seção 4.4, que resultaram em um esboço do mapa do ambiente:

- "Estrutura a partir da Política": utiliza o algoritmo de Iteração de Política (seção 2.3) para calcular, a partir da política $\pi_t(s_t)$ em um instante t, a função valor $V_t^{\pi}(s_t)$. A partir de V_t^{π}, é calculado o gradiente da função valor, ∇V_t^{π}. Finalmente, é realizada uma limiarização no resultado ∇V_t^{π},

esboçando um mapa do ambiente.

- "Estrutura a partir da Função Valor": similar ao primeiro, porém usa diretamente a função valor $V_t(s_t)$ para encontrar o mapa do ambiente.

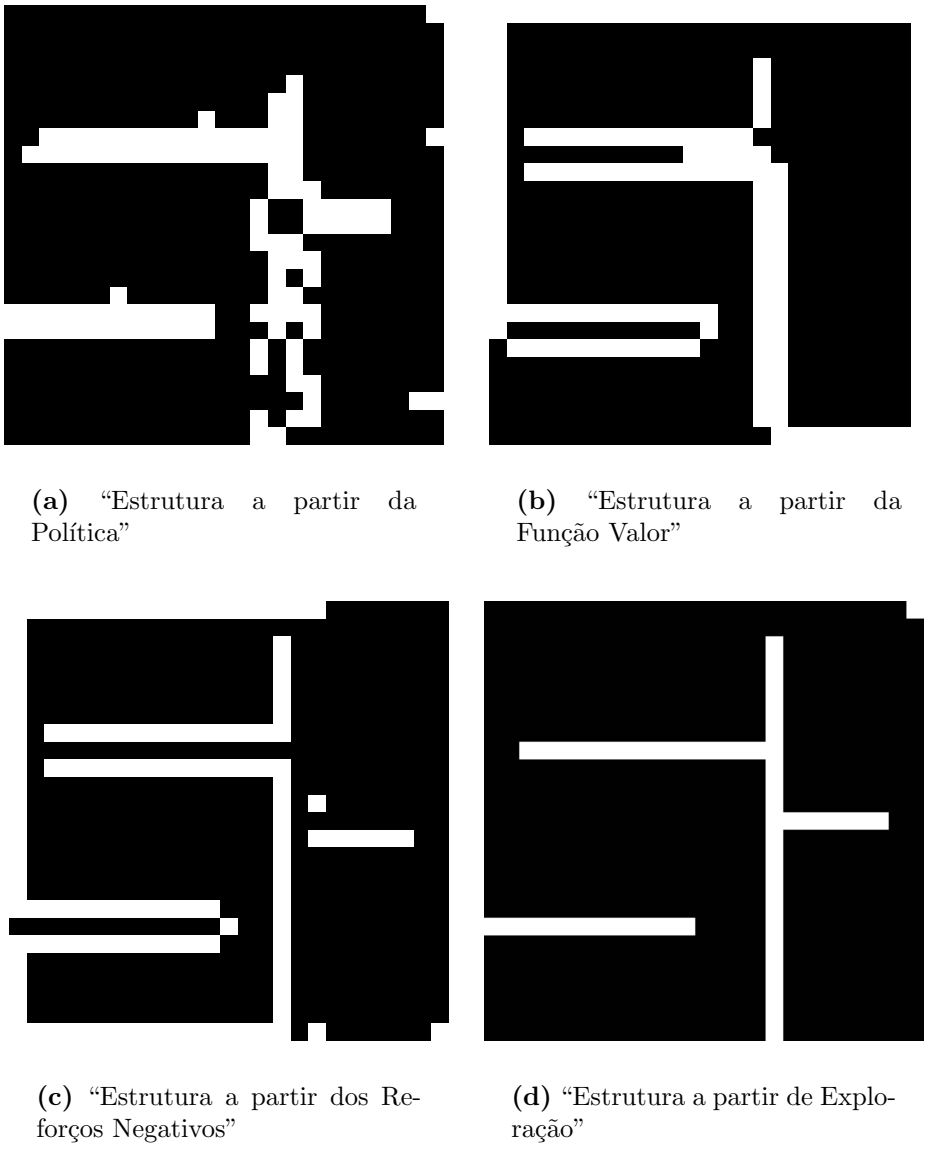
- "Estrutura a partir dos Reforços Negativos": é construída uma tabela que guarda os reforços negativos recebidos em cada posição, chamada $R^-(s_t)$. Novamente, utilizando limiarização, este método gera um mapa das posições que o robô deve evitar, o que geralmente reflete os obstáculos em um ambiente.

- "Estrutura a partir de Exploração": constrói iterativamente uma estimativa da função de probabilidade de transição $\hat{T}_t(s_t, a_t, s_{t+1})$, anotando o resultado de todas as experiências realizadas pelo robô. No caso do mundo de grade com paredes espessas (ocupam pelo menos uma posição na grade), este método gera o esboço do mapa do ambiente, por meio da anotação das posições visitadas.

As estruturas geradas por estes quatro métodos ao final do centésimo episódio de treinamento são mostradas na figura 5.3.

A figura 5.3-a mostra o esboço do mapa do ambiente construído utilizando o método de extração da estrutura a partir da política. Pode-se ver que este método reconhece bem os obstáculos, pois estes receberam reforços negativos e a função valor $V_t^\pi(s_t)$ nestas posições tem um valor menor que nas posições à sua volta.

A figura 5.3-b apresenta o resultado para o método "Estrutura a partir da Função Valor". Neste caso é interessante observar os resultados dos passos intermediários do algoritmo: a figura 5.4-a mostra a função valor $V_t(s_t)$ ao final do 100.^{o} episódio e a figura 5.4-b mostra o gradiente da função valor, ∇V_t. Pode-se ver que as partes mais claras na figura 5.4-b, que correspondem a uma maior diferença entre posições vizinhas na tabela $V(s_t)$, coincidem com as posições próximas às paredes. O resultado deste método não parece tão bom quanto o primeiro, pois não consegue encontrar os obstáculos onde o gradiente da função valor é pequeno, por exemplo, ele não encontra a parede localizada à direita da figura. Para solucionar este problema é necessário diminuir o valor limite usado na limiarização e utilizar outras técnicas de processamento de imagens para



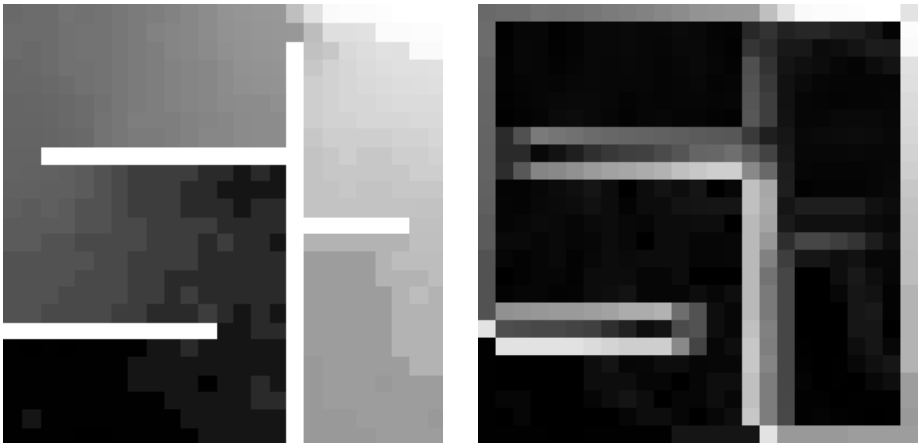
(a) "Estrutura a partir da Política"

(b) "Estrutura a partir da Função Valor"

(c) "Estrutura a partir dos Reforços Negativos"

(d) "Estrutura a partir de Exploração"

Figura 5.3: Estruturas geradas por quatro métodos de extração de estrutura ao final do centésimo episódio de treinamento.



(a) V_t **(b)** ∇V_t

Figura 5.4: A imagem da tabela $V(s_t)$ e o seu gradiente ao final do centésimo episódio de treinamento (células mais claras indicam maiores valores).

garantir que todas as bordas de interesse sejam encontradas, como no trabalho realizado por Drummond (2002).

A figura 5.3-c mostra que o método "Estrutura a partir dos Reforços Negativos" produz bons resultados. Pode-se ver que nela estão marcadas as posições vizinhas a todas as paredes. O único problema deste método é que, ao aprender a política ótima, o agente passa a receber menos reforços negativos. Por este motivo, a parede localizada à direita da figura só está marcada na parte de baixo.

O resultado do método "Estrutura a partir de Exploração" é apresentado na figura 5.3-d. Como é possível ver, para domínios determinísticos e estáticos, este método constrói perfeitamente o mapa do ambiente, bastando anotar quais pares estado-ação já foram visitados. No caso de um domínio não determinístico e propenso a erros de posicionamento, é necessário verificar quais estados foram menos visitados, realizando um processo similar à limiarização em relação ao número de visitas (realizado na seção 5.4).

Estes resultados, que são obtidos de maneira muito simples quando as paredes do ambiente são espessas, podem ser estendidos para o caso das paredes serem delgadas (elas não ocupam uma posição na grade, apenas impedem a transição entre duas posições adjacentes). Este caso é mais comum em domínios onde o que impede a transição entre os estados não é necessariamente um obstáculo físico

existente no ambiente, mas um impedimento do domínio.

Como exemplo da validade destes métodos para problemas com paredes delgadas, a figura 5.5 mostra o resultado do método de extração da estrutura a partir da função valor para um ambiente idêntico ao da figura 5.2, onde, porém, as paredes não ocupam uma posição na grade, mas apenas impedem a transição para a posição localizada à direita ou acima. Nota-se que o mesmo problema – não localizar a parede à direita – ocorre novamente aqui. O mesmo pode ser feito para os outros métodos estudados neste capítulo.

5.2.2 Construção da heurística

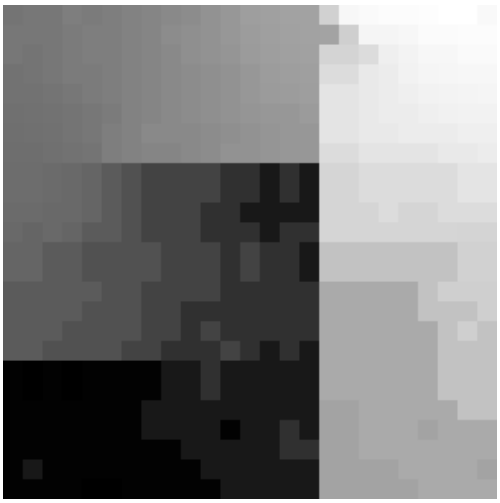
Para construir uma heurística para a política a partir das estruturas extraídas foi utilizada a "Retropropagação de Heurísticas", proposta na seção 4.4. Como este método depende apenas da informação que recebe, a qualidade da heurística gerada depende apenas da qualidade do método de extração da estrutura utilizado. A figura 5.6 mostra a heurística construída a partir do mapa gerado pelos quatro métodos de extração de estrutura (mostrados na figura 5.3).

Do mesmo modo que os mapas gerados pela "Estrutura a partir da Política", "Estrutura a partir da Função Valor" e "Estrutura a partir dos Reforços Negativos" (mostrados nas figuras 5.3-a, 5.3-b e 5.3-c) não correspondem perfeitamente a estrutura real, as heurísticas geradas (figuras 5.6-a, 5.6-b e 5.6-c) também não são ótimas. Apesar disso, a seção 5.3 mostrará que mesmo estas heurísticas produzem bons resultados na aceleração do aprendizado. Já o uso da retropropagação de heurísticas com o método "Estrutura a partir de Exploração" produz a política ótima (figura 5.6-d), igual a apresentada na figura 5.2.

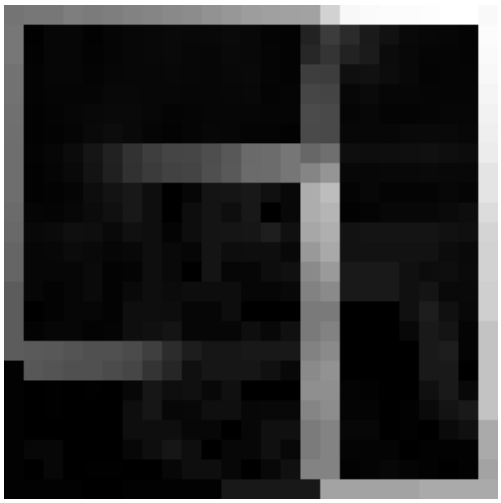
5.2.3 Discussão

Os resultados obtidos por estes experimentos permitiram concluir que tanto a extração da estrutura a partir da política quanto a partir da função valor necessitam de processamentos adicionais, pois o uso apenas do gradiente faz com que paredes onde a função valor é similar dos dois lados (por exemplo, uma parede muito distante da meta) não sejam localizadas.

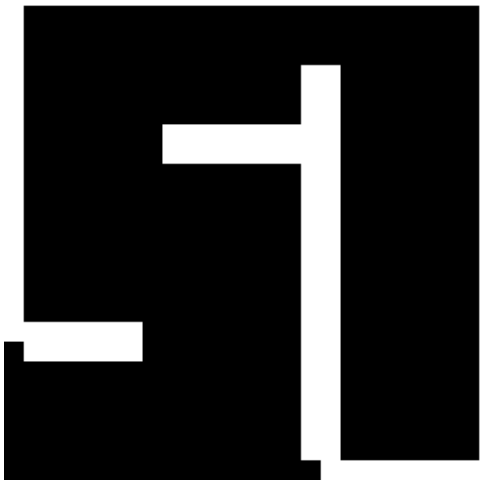
Além disso, o método de extração da estrutura a partir da política tem a



(a) V_t

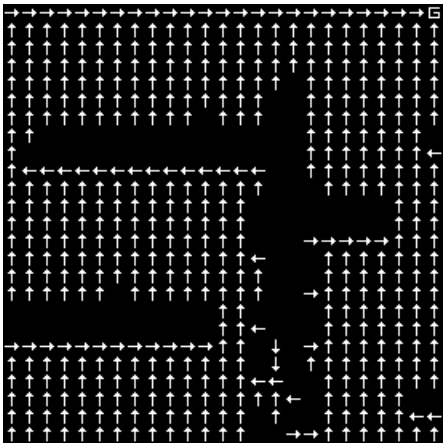


(b) ∇V_t

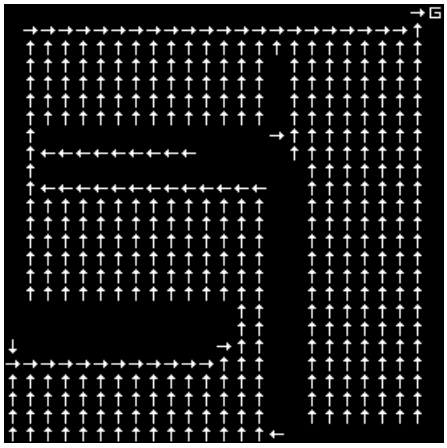


(c) Gradiente Binarizado de V_t

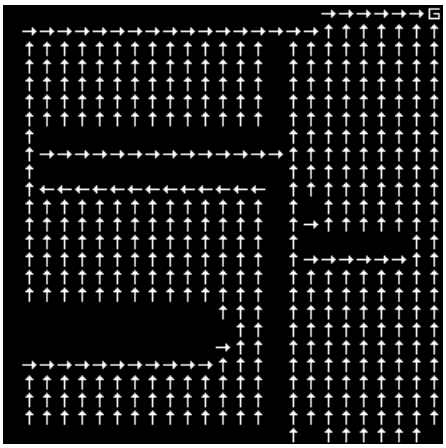
Figura 5.5: A imagem da tabela $V(s_t)$, o seu gradiente e o mapa criado ao final do centésimo episódio de treinamento para um ambiente com paredes delgadas.



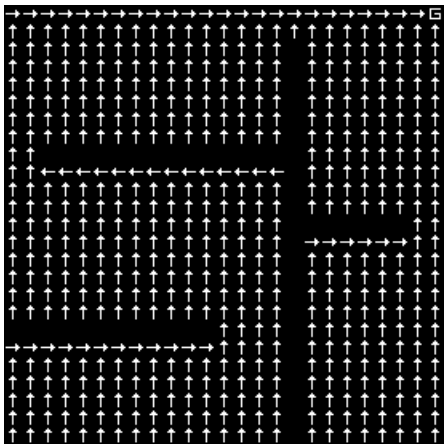
(a) "Heurística a partir da Política"



(b) "Heurística a partir da Função Valor"



(c) "Heurística a partir dos Reforços Negativos"



(d) "Heurística a partir de Exploração"

Figura 5.6: Heurísticas construídas utilizando o método "Heurística a partir de X" a partir das estruturas geradas pelos métodos de extração de estrutura, apresentadas na figura 5.3

desvantagem de precisar calcular a função valor utilizando programação dinâmica, o que torna o tempo necessário para a construção do esboço do mapa do ambiente a partir da política muito alto (da ordem de 45 segundos para o ambiente 25x25 da figura 5.1, utilizando a configuração descrita). Finalmente, existe a necessidade de se conhecer as probabilidades de transição para resolver o sistema de equações lineares utilizando programação dinâmica, que pode ser descoberto a partir de exploração, mas torna todo o processo mais complicado.

Tanto os métodos de extração de estrutura a partir dos reforços negativos quanto por exploração produziram bons resultados. O primeiro tem como vantagem poder ser usado diretamente a partir da tabela R^-, integrando-se facilmente com a maioria dos algoritmos de aprendizado mostrados neste trabalho. Porém, o método de extração a partir de exploração permite a construção da função de transição (usada para gerar a heurística) melhor e mais rapidamente que todos os outros em um domínio determinístico. Assim, este método será utilizado nos experimentos apresentados a seguir.

5.3 Experimentos com o algoritmo HAQL

Para realizar um primeiro teste do algoritmo *Heuristically Accelerated Q-Learning* foram realizados três experimentos utilizando o método "Heurística a partir de Exploração" no domínio dos robôs móveis: navegação robótica em ambiente desconhecido, navegação robótica em um ambiente pouco modificado e navegação com reposicionamento da meta. Estes experimentos são tradicionais na área de robótica móvel, sendo usados por diversos pesquisadores, como Drummond (2002).

O valor da heurística usada no HAQL é definido a partir da equação (4.14) como sendo:

$$H(s_t, a_t) = \begin{cases} \max_a \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + 1 & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{caso contrário.} \end{cases} \tag{5.1}$$

Este valor é calculado apenas uma vez, no início da aceleração. Em todos os episódios seguintes, o valor da heurística é mantido fixo, permitindo que o aprendizado supere indicações ruins (caso $H(s_t, a_t)$ fosse recalculado a cada episódio,

uma heurística ruim dificilmente seria superada).

Para efeitos comparativos, os mesmos experimentos também são desenvolvidos utilizando o algoritmo *Q–Learning*. Os parâmetros utilizados tanto no *Q–Learning* quanto no HAQL foram os mesmos: taxa de aprendizado $\alpha = 0,1$, $\gamma = 0,99$ e a taxa de exploração/explotação igual a 0,1. Os reforços utilizados foram: 10 para quando o robô chega ao estado meta e -1 ao executar qualquer ação.

Os resultados mostram a média de 30 treinamentos em nove configurações diferentes para o ambiente onde os robôs navegam – uma sala com diversas paredes – mostrados na figura 5.7. O tamanho do ambiente é de 55 por 55 posições e a meta se encontra inicialmente no canto superior direito.

5.3.1 Navegação robótica em ambiente desconhecido

Neste experimento, um robô deve aprender a política que o leva à meta ao ser inserido em um ambiente desconhecido, em uma posição aleatória. Foi realizada a comparação entre o *Q–Learning* e o HAQL utilizando o método "Heurística a partir da Exploração" para acelerar o aprendizado a partir do 10.^{o} episódio de aprendizado.

Inicialmente o HAQL apenas extrai a estrutura do problema, sem fazer uso da heurística. Assim, entre o primeiro e o nono episódio, o HAQL se comporta do mesmo modo que *Q–Learning*, com o algoritmo de extração de estrutura sendo executado ao mesmo tempo. Ao final do nono episódio, a heurística a ser usada é construída utilizando a "Retropropagação de Heurísticas", e os valores de $H(s_t, a_t)$ são definidos a partir da equação (5.1).

A partir da análise do número de novos estados visitados a cada episódio, foi possível concluir que a partir do 10.^{o} episódio nenhum novo estado era visitado neste domínio (o anexo C analisa a evolução das visitas às posições em um ambiente de maior dimensão). Este resultado reflete o fato do robô ser recolocado em uma posição aleatória ao início de um episódio e do ambiente ser pequeno. Com base nesta análise, foi escolhido o 10.^{o} episódio para o início da aceleração, visando permitir ao agente explorar o ambiente e a criação uma boa heurística. O resultado (figura 5.8) mostra que, enquanto o *Q–Learning* continua a buscar pela

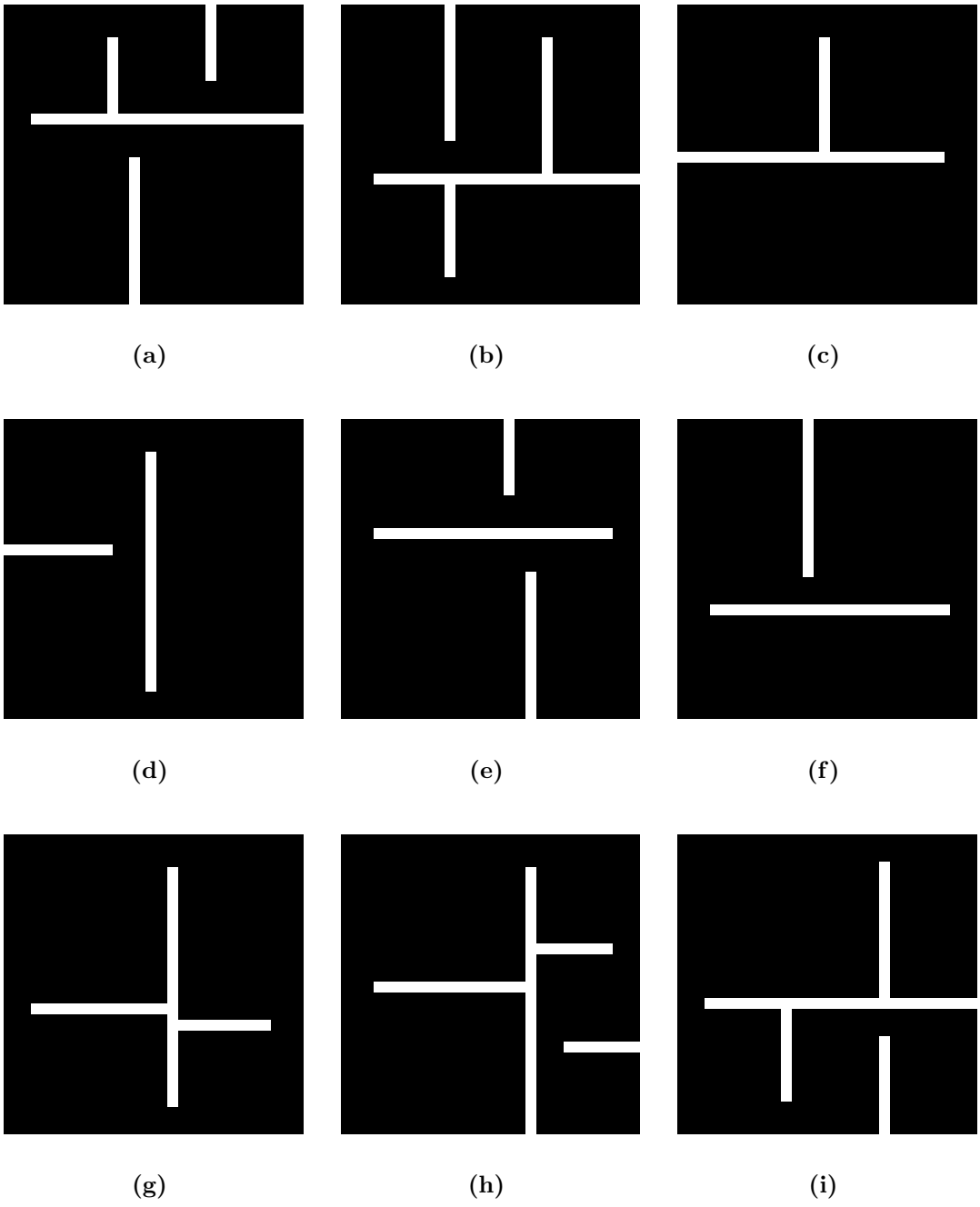


Figura 5.7: Salas com paredes (representadas pelas linhas claras) usadas por Drummond (2002), onde a meta a atingir se encontra em um dos cantos.

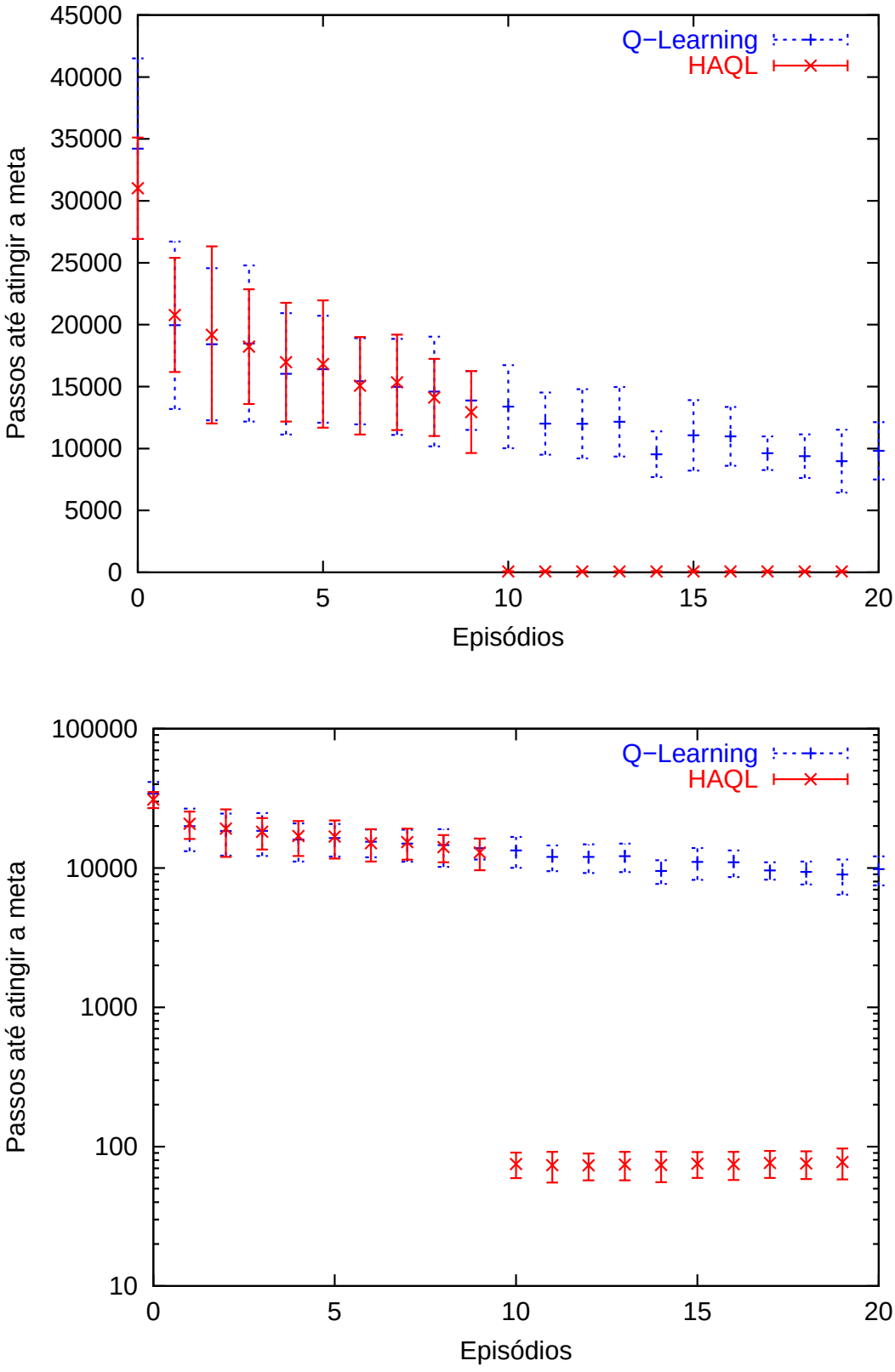


Figura 5.8: Resultado para a aceleração em um ambiente desconhecido, ao final do décimo episódio, utilizando "Heurística a partir de Exploração", com barras de erro (inferior em monolog).

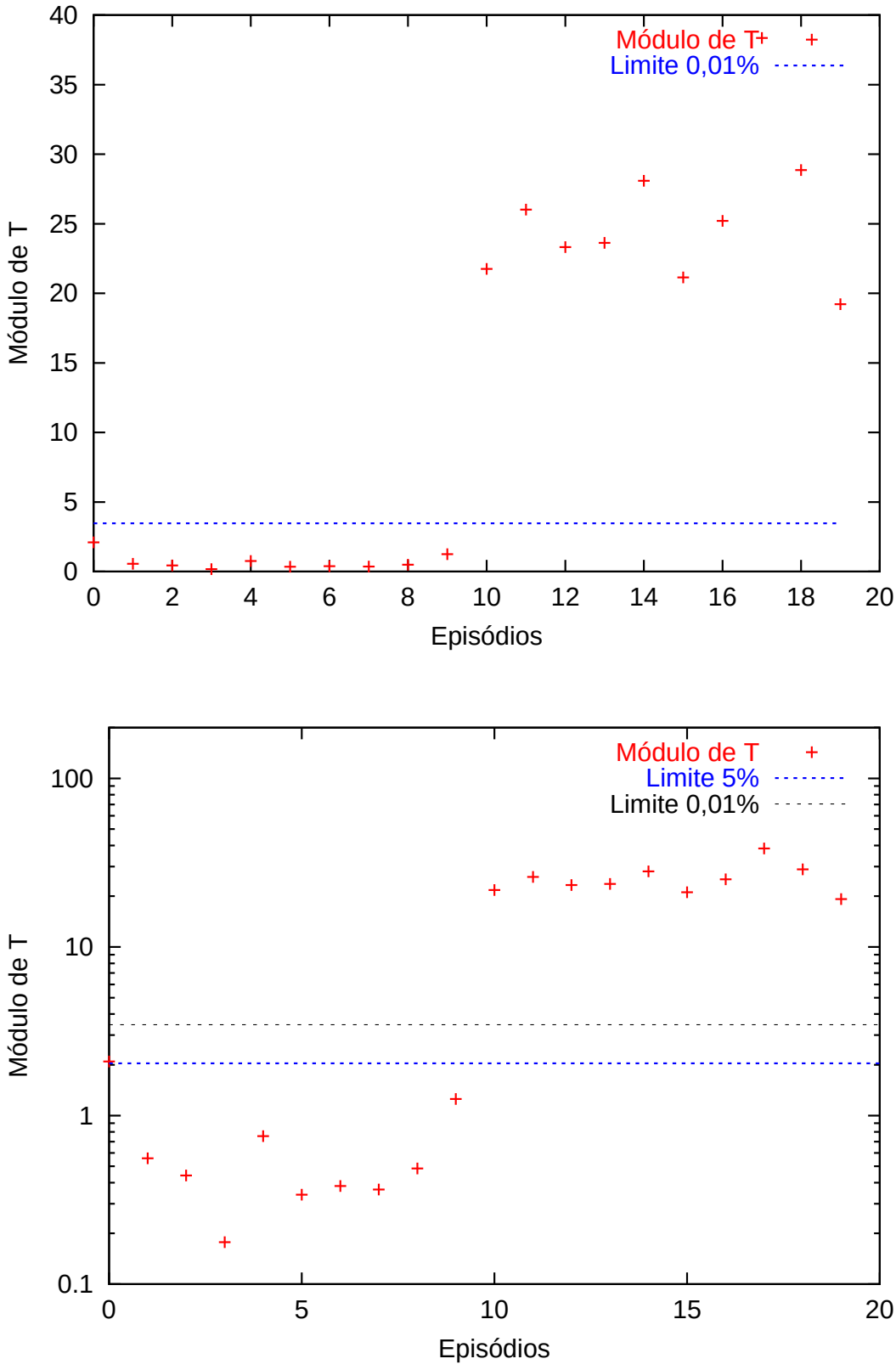


Figura 5.9: Resultado do teste t de Student para um ambiente desconhecido (inferior em monolog).

melhor política, o HAQL passa a utilizar a política ótima imediatamente após a aceleração, executando o número mínimo de passos para chegar à meta a partir do 10.^{o} episódio.

Finalmente, foi utilizado o teste t de Student (SPIEGEL, 1984) – que também é conhecido como *T-Test* – para verificar se a hipótese de que o uso do algoritmo HAQL acelera o aprendizado é válida[1].

Foi calculado o valor do módulo de T para cada episódio utilizando os mesmos dados apresentados na figura 5.8, permitindo a comparação entre o resultado do *Q–Learning* e o HAQL. A figura 5.9 apresenta o resultado, mostrando que a partir da 10.^{a} iteração os algoritmos são significativamente diferentes, com nível de confiança maior que 0,01%.

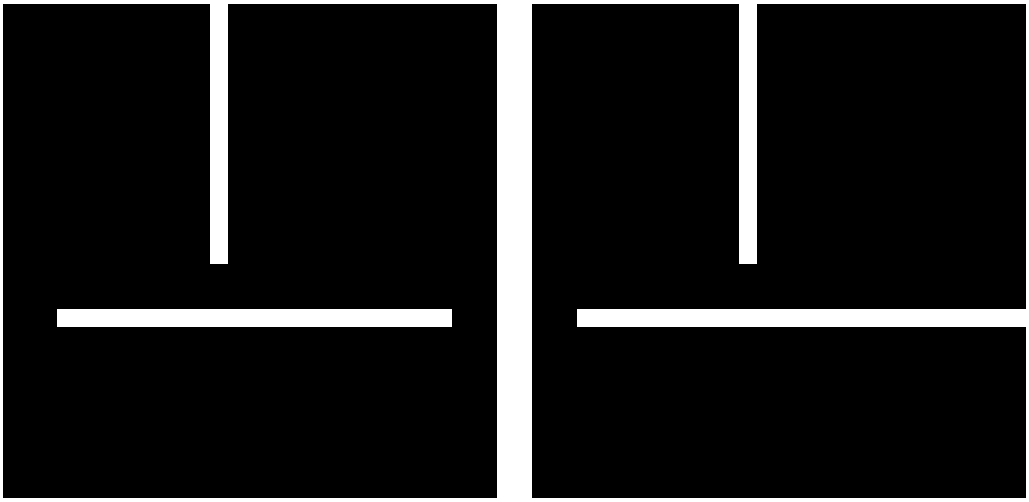
5.3.2 Navegação robótica em ambiente pouco modificado

A figura 5.6 mostra que a heurística criada pelo método "Heurística a partir da X" nem sempre corresponde à política ótima, já que o esboço do mapa do ambiente usado pode não ser perfeito e, algumas ações apontadas como boas pelas heurísticas, na verdade não o são. Caso a extração da estrutura falhe ao localizar uma parede completamente, a heurística criada pode fazer com que o robô fique tentando se mover para uma posição inválida.

Neste experimento, um robô deve aprender a política que o leva à meta, ao ser inserido em um ambiente desconhecido, em uma posição aleatória, de maneira similar ao experimento anterior. A única diferença é que, neste experimento, é realizada uma pequena modificação no ambiente ao final da nona iteração. Com isso espera-se verificar o comportamento do HAL quando o ambiente é ligeiramente modificado e simular o uso de uma heurística que não seja completamente correta.

Novamente, entre o primeiro e o nono episódio, o HAQL apenas extrai a estrutura do problema, sem fazer uso da heurística. Ao final do nono episódio, a heurística a ser usada é construída, utilizando a "Retropropagação de Heurísticas", e os valores de $H(s_t, a_t)$ são definidos a partir da equação (5.1). A partir do 10.^{o} episódio de aprendizado esta heurística é usada para acelerar o

[1] O apêndice I apresenta um resumo sobre o teste t de Student.



(a) Ambiente até o final do 9.^{o} episódio

(b) Ambiente após o início do 10.^{o} episódio

Figura 5.10: O ambiente usado para definir a heurística (a) e o ambiente onde ela é utilizada (b).

aprendizado.

Para este experimento foi utilizado apenas uma configuração do ambiente, mostrado na figura 5.10. A figura 5.10-a mostrado o ambiente até o final do 9.^{o} episódio, onde existem passagens à direita e à esquerda da parede horizontal. Ao início do décimo episódio a passagem à direita é fechada, como pode ser observado na figura 5.10-b.

A heurística gerada para este problema utilizando o método "Heurística a partir de Exploração" é mostrada na figura 5.11. Destaca-se que ela é muito parecida com a política ótima para o ambiente com a passagem da parede horizontal direita fechada, mostrada na figura 5.12, que é gerada pelo HAQL ao final do 100^o episódio.

A figura 5.13 apresenta o resultado do uso do HAQL com a heurística criada para o ambiente com a parede incompleta (média para 30 treinamentos). Pode-se notar que no momento da aceleração (que é o mesmo no qual ocorre a modificação no ambiente) ocorre uma piora no desempenho do agente HAQL, mas o agente logo aprende a desconsiderar heurística no estado onde ela não está correta, resultando em uma aceleração (o anexo B mostra outros resultados do uso de uma

heurística não apropriada).

O resultado do teste t de Student, utilizando-se os dados apresentados na figura 5.13, é apresentado na figura 5.14. Pode ser observado que a partir da 60.ª iteração a maioria dos resultados é significativamente diferente devido à aceleração, com nível de confiança maior que 5%.

Uma maneira de comparar o custo computacional dos algoritmos HAQL e *Q–Learning* é computando a área abaixo das curvas da figura 5.13, que representa o número de passos total realizados pelo agente até o término de um dado episódio e reflete o custo computacional dos algoritmos.

A partir do resultado do teste t de Student, foi definindo o 60^o episódio como ponto a partir do qual o HAQL passa a apresentar resultados significativamente melhores que o *Q–Learning*. O número de passos que o HAQL levou para atingir o final desse episódio é 429.396 ± 35.108 passos e o tempo de execução é 0,34 ± 0,04 segundos (média para 30 treinamentos).

Sabendo que a partir deste episódio o agente que utiliza o algoritmo HAQL não mais leva mais que 1000 ± 1000 passos para atingir a meta, foi tomado este valor como limite máximo para o número de passos e, analisando os resultados para cada episódio, verificou-se que o *Q–Learning* só passa a apresentar um resultado qualitativamente similar a partir do 1000^o episódio.

Foi computado então o número de passos total que um agente executando o algoritmo *Q–Learning* executa até atingir o 1000^o episódio. O resultado encontrado mostra que é necessário 2.360.358 ± 28.604 passos e 1,25 ± 0,02 segundos (média de 30 treinamentos, utilizando a mesma configuração computacional) para este agente apresentar soluções semelhantes.

Este resultado mostra qualitativamente que, mesmo existindo um aumento no número de passos executados nos episódios próximos ao momento da aceleração do aprendizado, o HAQL é computacionalmente mais eficiente que o *Q–Learning*. Em particular, o tempo de execução de cada algoritmo mostra que utilizar o método "Heurística a partir de Exploração" não aumenta significantemente o custo computacional do HAQL.

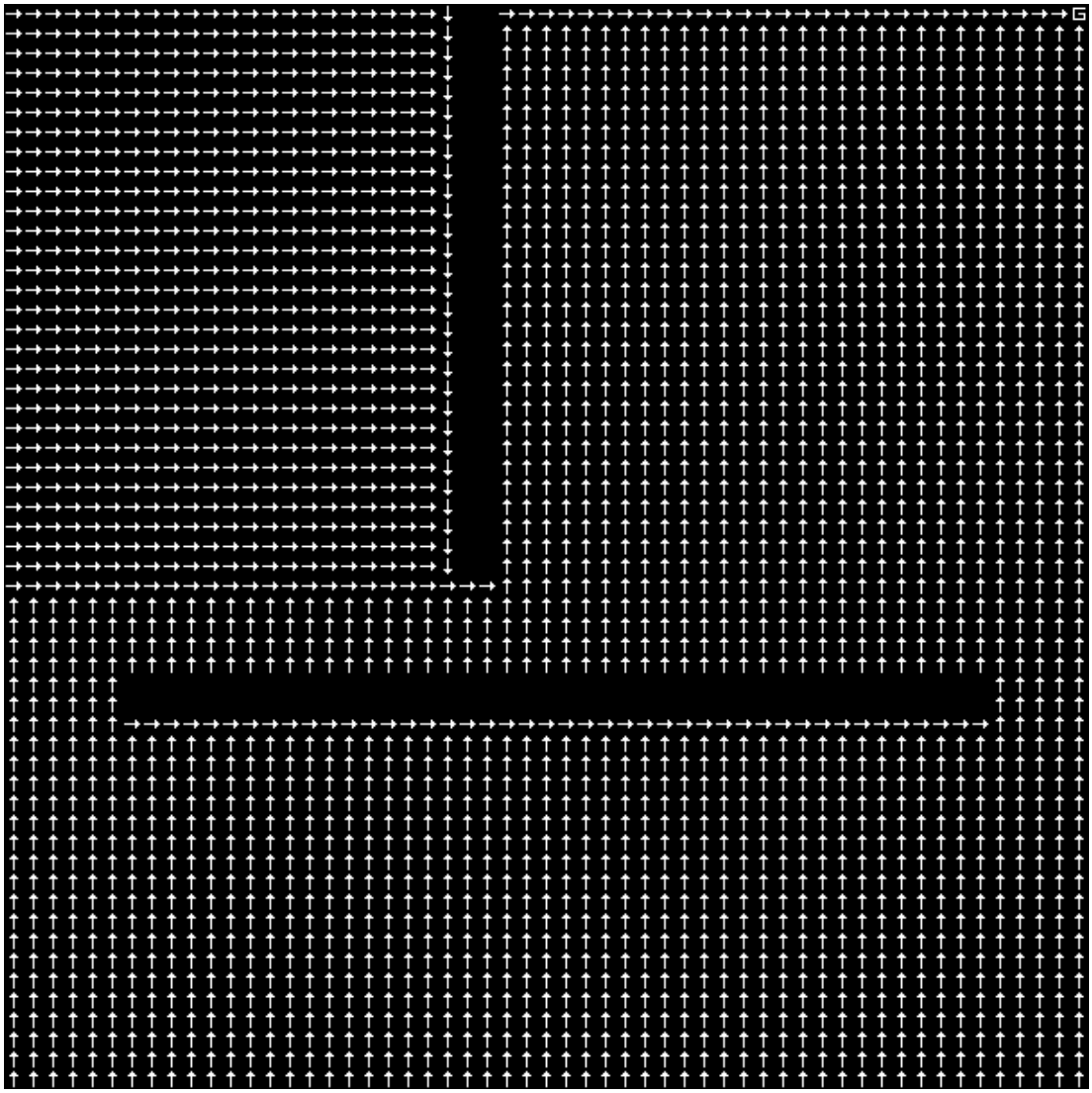


Figura 5.11: A heurística gerada para o ambiente da figura 5.10-a.

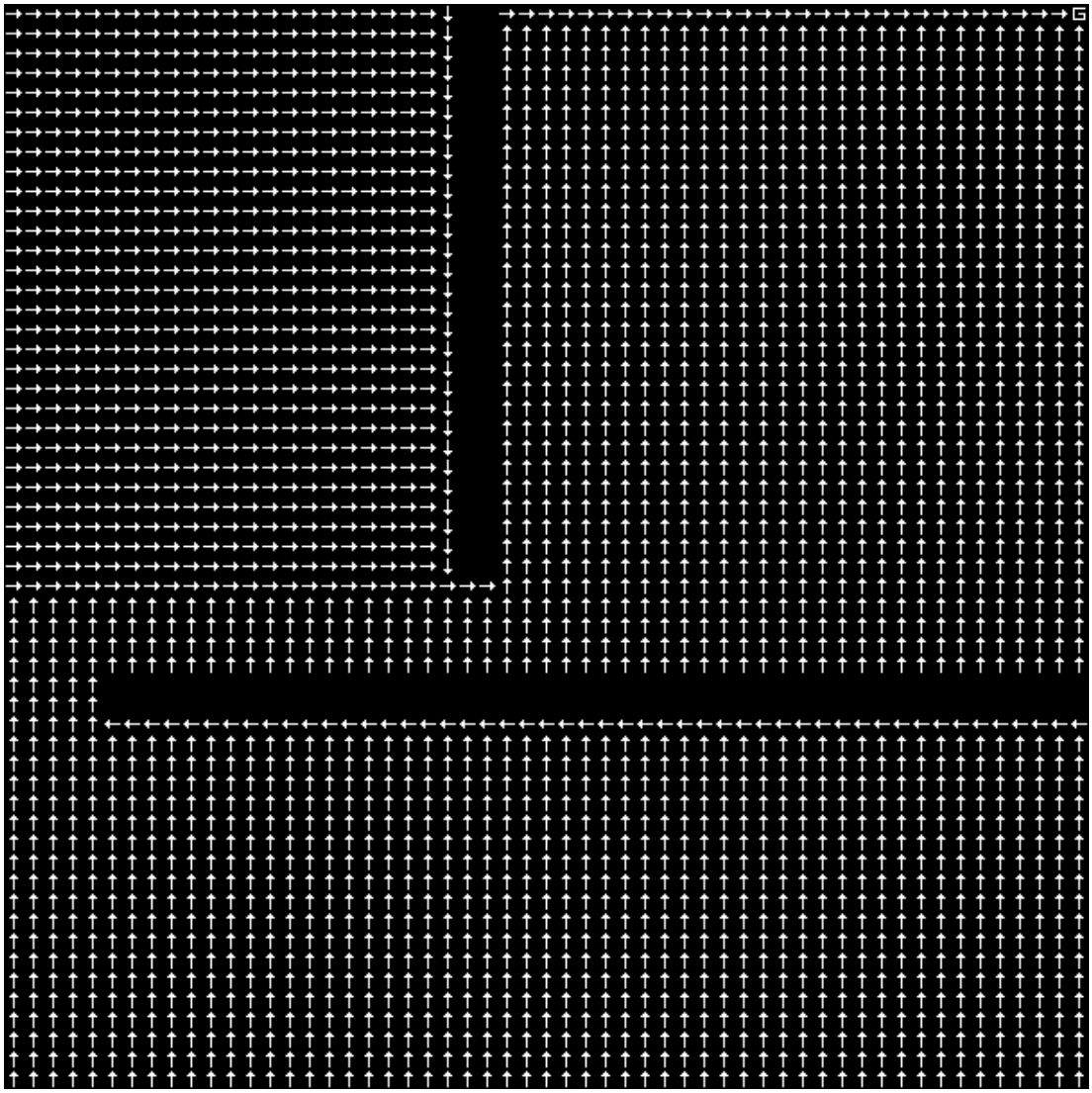


Figura 5.12: A política ótima para o ambiente da figura 5.10-b.

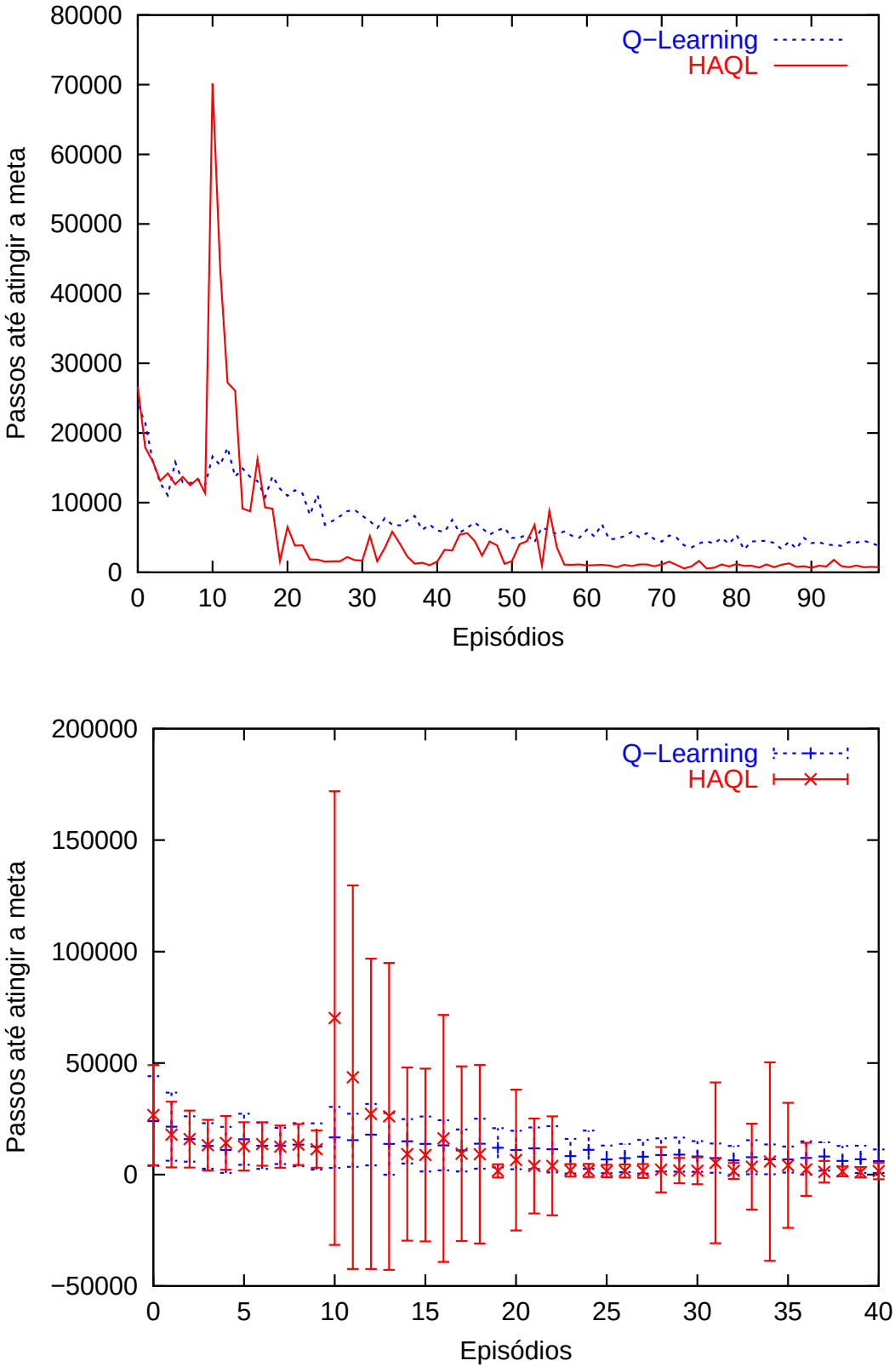


Figura 5.13: Resultado para a aceleração em um ambiente modificado, ao final do décimo episódio, utilizando "Heurística a partir de Exploração" (inferior com barras de erro).

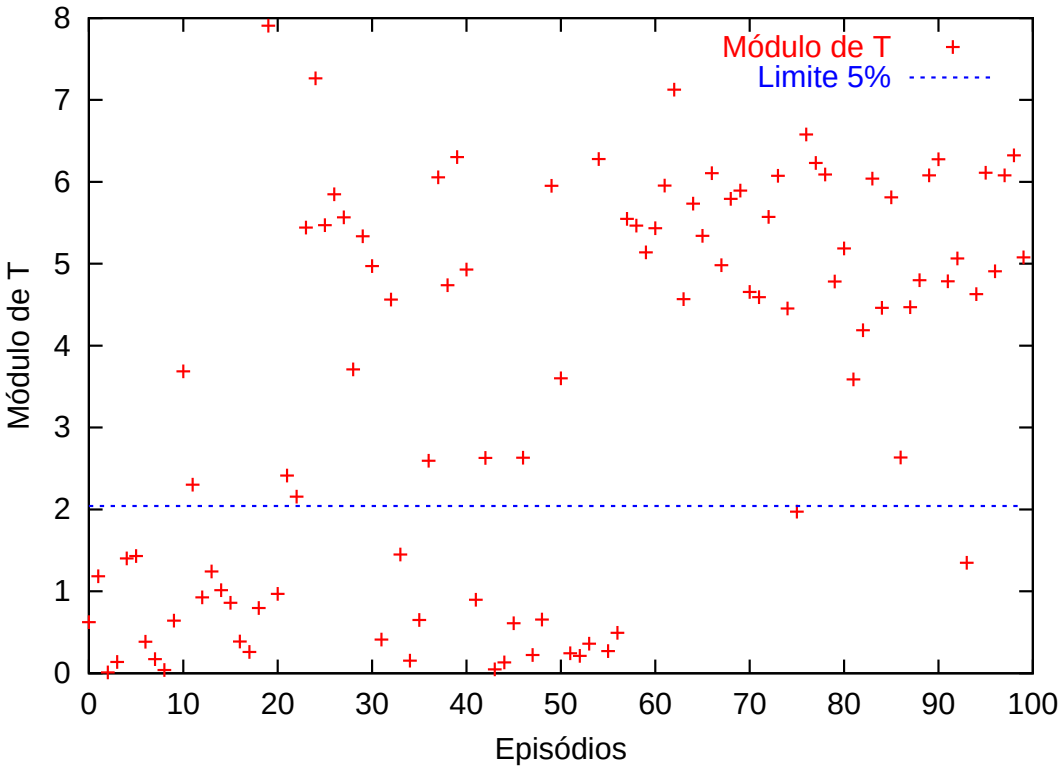


Figura 5.14: Resultado do teste t de Student para aceleração na 10.ª iteração, em um ambiente modificado.

5.3.3 Navegação robótica com reposicionamento da meta

O objetivo deste teste é verificar o resultado da aceleração do aprendizado quando a meta é apenas reposicionada em um mesmo ambiente, durante o processo de aprendizado.

Novamente, o HAQL inicialmente apenas extrai a estrutura do problema (entre o primeiro e o 4999.^{o} episódio), comportando-se como o *Q–Learning* com o algoritmo de extração de estrutura sendo executado ao mesmo tempo.

Ao final do 4999.^{o} episódio a meta é reposicionada do canto superior direito para o canto inferior esquerdo. Com isso, ambos os algoritmos devem encontrar a nova posição da meta. Como os algoritmos estão seguindo a política aprendida até então, ocorre uma piora no desempenho, pois existe a necessidade de se executar um grande número de passos para atingir a nova posição da meta.

Assim que o robô controlado pelo HAQL chega à meta (ao final do 5000.^{o} episódio), a heurística a ser usada é construída utilizando a "Retropropagação de Heurísticas" a partir da estrutura do ambiente (que não foi modificada) e da nova posição da meta, e os valores de $H(s_t, a_t)$ são definidos. Esta heurística é então utilizada à partir do 5001.^{o} episódio, resultando em um melhor desempenho em relação ao *Q–Learning*, como mostrado nas figuras 5.15 e 5.16.

Como esperado, pode-se observar que o HAQL tem um desempenho similar ao *Q–Learning* até o 5000.^{o} episódio. Neste episódio, tanto o *Q–Learning* quanto o HAQL levam da ordem de 1 milhões de passos para encontrar a nova posição da meta (já que a política conhecida os leva a uma posição errada).

Após o reposicionamento da meta, enquanto o *Q–Learning* necessita reaprender toda a política, o HAQL executa sempre o mínimo de passos necessários para chegar à meta. A figura 5.16 mostra ainda que o desvio que ocorre em cada episódio é pequeno, ocorrendo devido ao fato do agente ser posicionado em estados iniciais diferentes a cada treinamento.

Novamente foi usado o teste t de Student para verificar a validade do resultado. Foi calculado o valor do módulo de T para cada episódio utilizando os mesmos dados apresentados na figura 5.16. No resultado, mostrado na figura 5.17, sobressai que a partir da 5001.^{a} iteração os resultados são significativamente diferentes

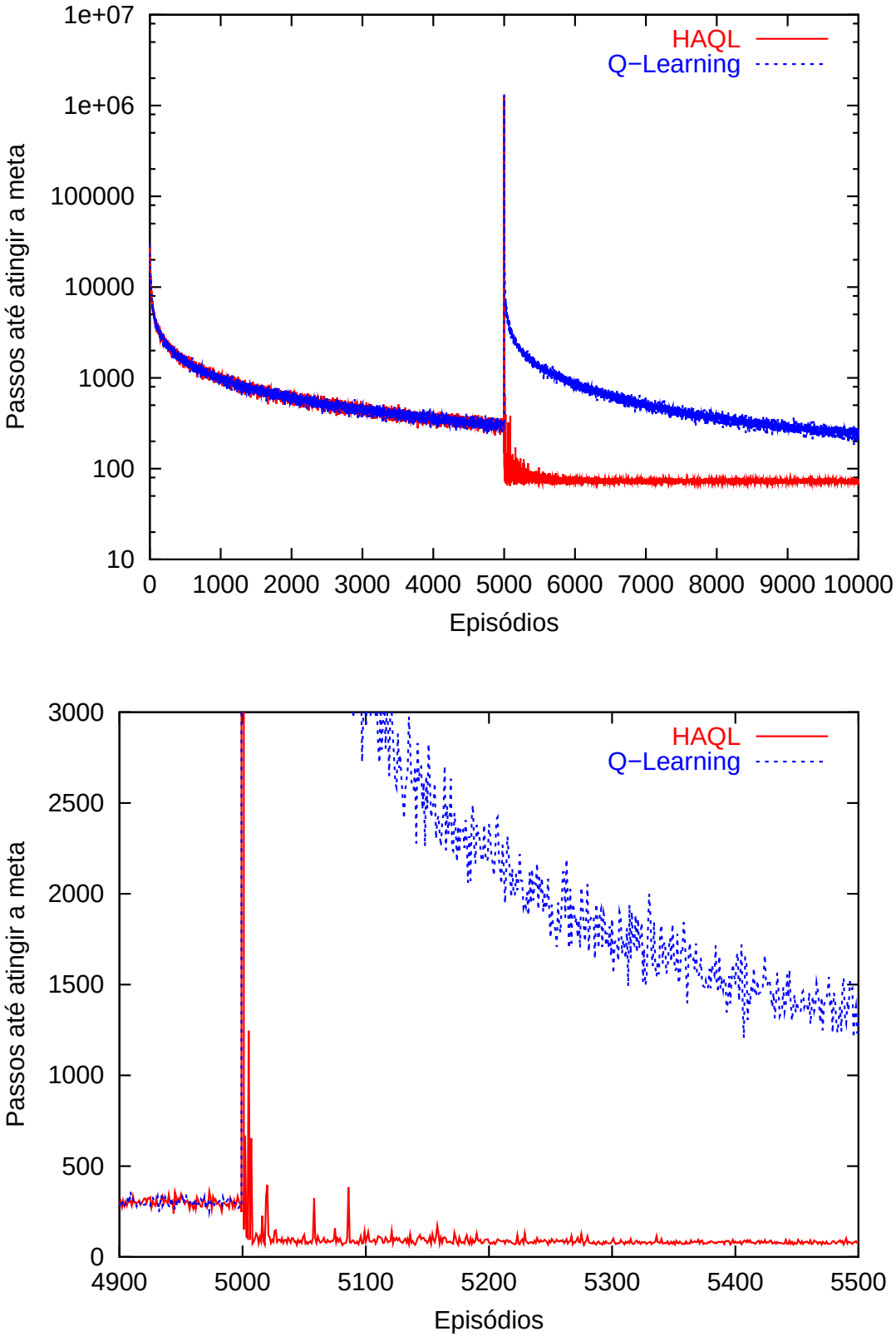


Figura 5.15: Resultado do aprendizado quando a meta é reposicionada no 5000.º episódio, utilizando "Heurística a partir de Exploração" no HAQL (em escala monolog na parte superior e ampliado na inferior).

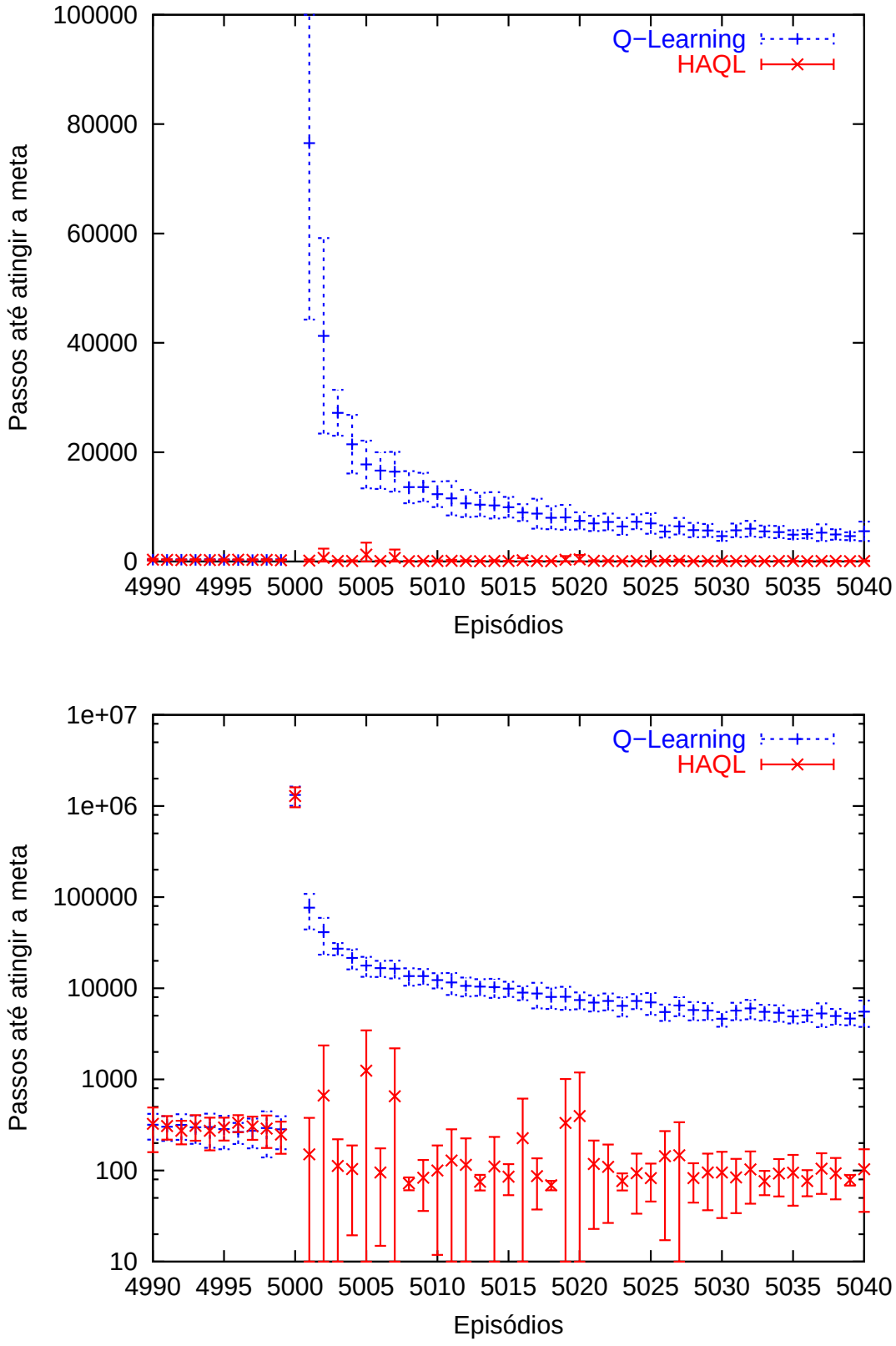


Figura 5.16: Resultado do aprendizado quando a meta é reposicionada no 5000.º episódio, utilizando "Heurística a partir de Exploração" no HAQL. Com barras de erro (inferior em monolog).

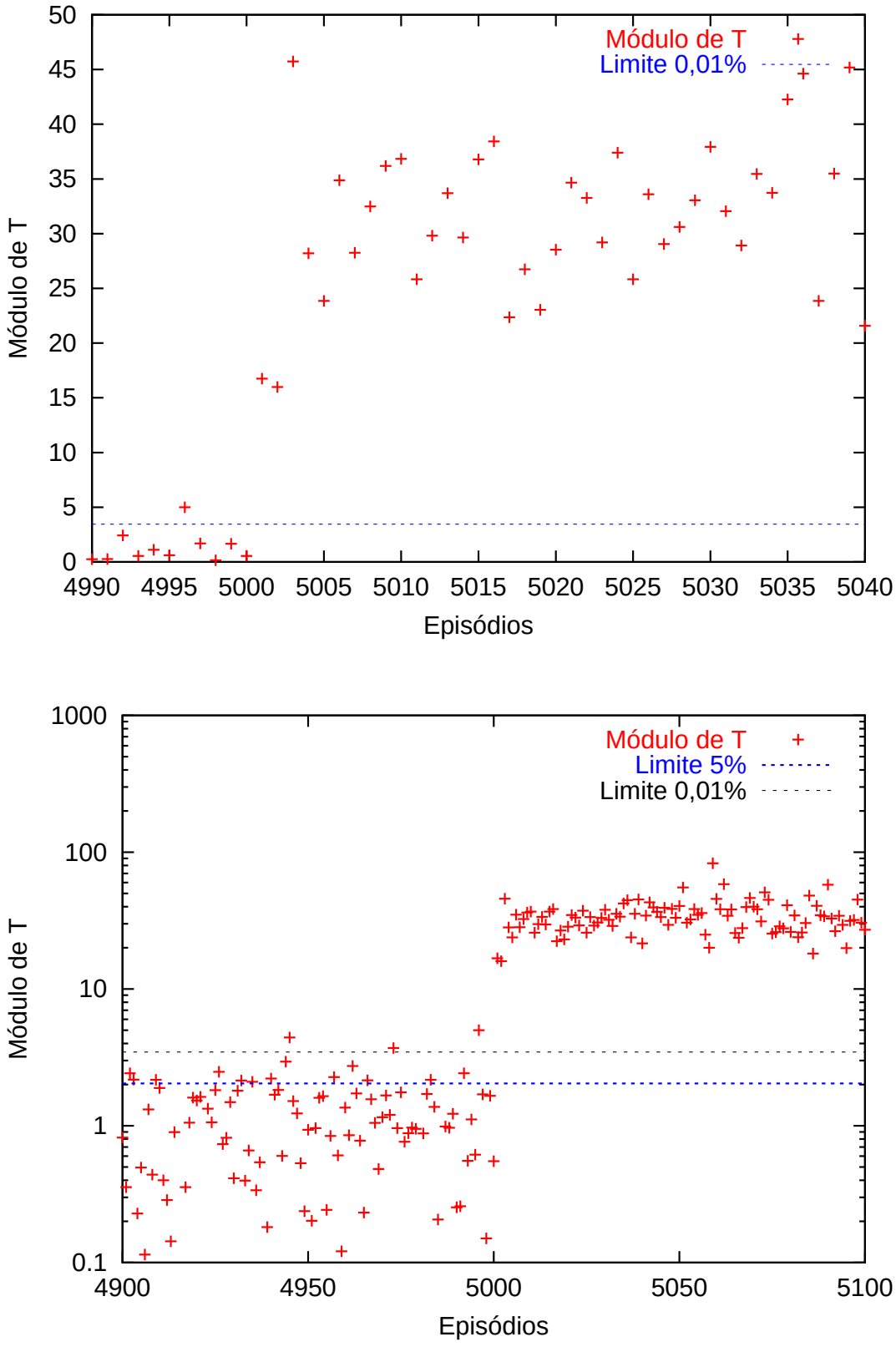


Figura 5.17: Resultado do teste t de Student para reposicionamento da meta (inferior em monolog).

devido à aceleração, com nível de confiança maior que 0,01%.

5.3.4 Discussão

Esta seção apresentou uma comparação entre o algoritmo *Q–Learning* e o HAQL no domínio dos robôs móveis autônomos. Pode-se verificar que o uso da heurística pelo algoritmo HAQL acelera o aprendizado, utilizando informações adquiridas durante o processo de aprendizagem.

Muitas vezes uma pequena modificação na configuração do domínio torna inútil uma política aprendida, exigindo que o *Q–Learning* reinicie o aprendizado. As sub-seções 5.3.2 e 5.3.3 mostraram que o HAQL permite a reutilização de conhecimento adquirido durante o processo de aprendizagem. Esta reutilização é feita de maneira simples, na própria política, e não na estimativa da função valor (como realizado por Drummond (2002)).

Uma característica interessante a ser destacada é que, utilizando o algoritmo HAQL, mesmo que a heurística não seja completamente correta, ainda assim pode ocorrer um ganho no desempenho do aprendizado (como mostrado na sub-seção 5.3.2) devido à parte correta da heurística.

Finalmente, experimentos discutindo a evolução das visitas às posições em um ambiente de maior dimensão e a evolução da função valor ao se utilizar a aceleração são apresentados nos anexos C e D, respectivamente.

5.4 Simulação de um robô real utilizando o ambiente Saphira

O objetivo desta seção é verificar o desempenho do *Heuristically Accelerated Q–Learning* ao ser aplicado na simulação de um robô real, com dinâmica mais complexa que a do mundo de grades, imerso em um ambiente não determinístico e sujeito a erros de posicionamento. Para tanto, foi utilizada a plataforma Saphira 8.0 (KONOLIGE; MYERS, 1996; ACTIVMEDIA ROBOTICS, 2001) controlando um robô móvel Pioneer 2DX, fabricado pela ActivMedia Robotics.

O Saphira, projetado originalmente para controlar o robô Flakey da SRI Internacional, é hoje uma plataforma de desenvolvimento completa para sistemas

de controle de diversos tipos de robôs móveis. É composta por diversos módulos: um servidor capaz de controlar um robô real ou simulado, um simulador que permite estudar diversos modelos de robôs e diversas bibliotecas de funções. A partir da versão 8.0, o Saphira passou a ser baseado na biblioteca ARIA, que contém a maior parte do controle do robô encapsulada em classes. Enquanto o Saphira facilita a tarefa de criação dos aplicativos, a biblioteca ARIA provê o acesso de baixo nível aos sensores e atuadores dos robôs.

A arquitetura do Saphira é baseada no modelo cliente–servidor, utilizando o protocolo TCP–IP, onde o servidor é responsável por gerenciar as tarefas de baixo nível de um robô real ou simulado e os usuários criam aplicativos clientes que se comunicam com o servidor para controlar o robô. Sua biblioteca provê um conjunto de rotinas de comunicação que possibilita controlar o robô, receber informações dos sensores, construir clientes com interfaces gráficas e para a simulação dos robôs.

O módulo de simulação da plataforma Saphira utiliza modelos de erros próximos aos reais para os sensores – sonares, laser e *encoders* dos motores – de maneira que, de modo geral, se um programa funciona no simulador, também funcionará para controlar o robô real. Ele permite a construção de ambientes simples onde o robô pode navegar.

Uma das competências fundamentais que um robô deve possuir para navegar em um ambiente real é a capacidade de determinar sua própria posição dentro de um plano de referência.

O plano de referência para navegação pode ser definido como sendo um sistema cartesiano de coordenadas onde posições são armazenadas. Considerando que uma jornada inicie em uma posição conhecida e que a velocidade e direção de navegação do robô sejam precisamente medidas, a posição do robô pode ser obtida por meio da integração, no tempo, do movimento do robô. Este processo é conhecido como *Dead Reckoning*.

É muito raro que um sistema de navegação de um robô móvel baseado exclusivamente em *dead reckoning* funcione para longas distâncias. Como o sistema de referência não está fixo no mundo real, os odômetros são capazes de determinar movimentos ocorridos no sistema de referência, mas não aqueles do próprio sistema. Assim, problemas como escorregamentos das rodas geram erros na loca-

lização do robô.

O módulo servidor da plataforma Saphira usa um mapa global, acessível somente ao simulador, para representar o ambiente com os objetos nele contidos e realizar a simulação, e um mapa local, acessível ao aplicativo cliente, que contém apenas uma parte do ambiente, em coordenadas centradas no robô.

A localização do robô acessível ao cliente é a do plano de referência: ele tem conhecimento de sua posição inicial e determina sua postura (posição e orientação) utilizando *dead reckoning*, compensando os erros odométricos através de técnicas de Localização Monte Carlo.

A Localização Monte Carlo (*Monte Carlo Localization* – MCL) (FOX; BURGARD; THRUN, 1999; THRUN et al., 2001) é uma família de algoritmos baseados em filtros de partículas, que utilizam uma abordagem probabilística para estimar a posição real do robô, a partir de dados de sensores. Estes algoritmos utilizam N partículas (amostras) distribuídas em torno da posição mais provável do robô. Cada partícula define uma postura para o robô e sua incerteza associada. A postura do robô é uma combinação, ponderada pela incerteza, das posturas das amostras.

A MCL atualiza a posição do robô gerando novas amostras, que aproximam a nova posição do robô, e utilizando dados sensorias para estimar a incerteza de uma determinada amostra com relação a sua postura. A cada atualização, uma determinada quantidade de partículas é retirada do conjunto de amostras, com uma probabilidade determinada pelas suas incertezas.

A figura 5.18 mostra o ambiente de simulação Saphira 8.0. Na imagem superior pode-se ver o aplicativo, que controla o robô móvel. Os quadrados que aparecem em volta do robô são resultados da leitura dos sensores de ultrassom, sendo que os quadrados verdes indicam que a distância resultante é maior que o limite de confiança ou que não houve retorno; os azuis são leituras recentes e os pretos leituras a uma distância muito pequena. Os pontos vermelhos em volta do robô indicam as posições das partículas usadas pela localização Monte Carlo e a seta verde indica a orientação provável do robô. Na imagem inferior encontra-se o monitor do simulador, que apresenta a postura real do robô em um determinado instante. A figura 5.19 mostra em detalhe as partículas usadas pela localização Monte Carlo.

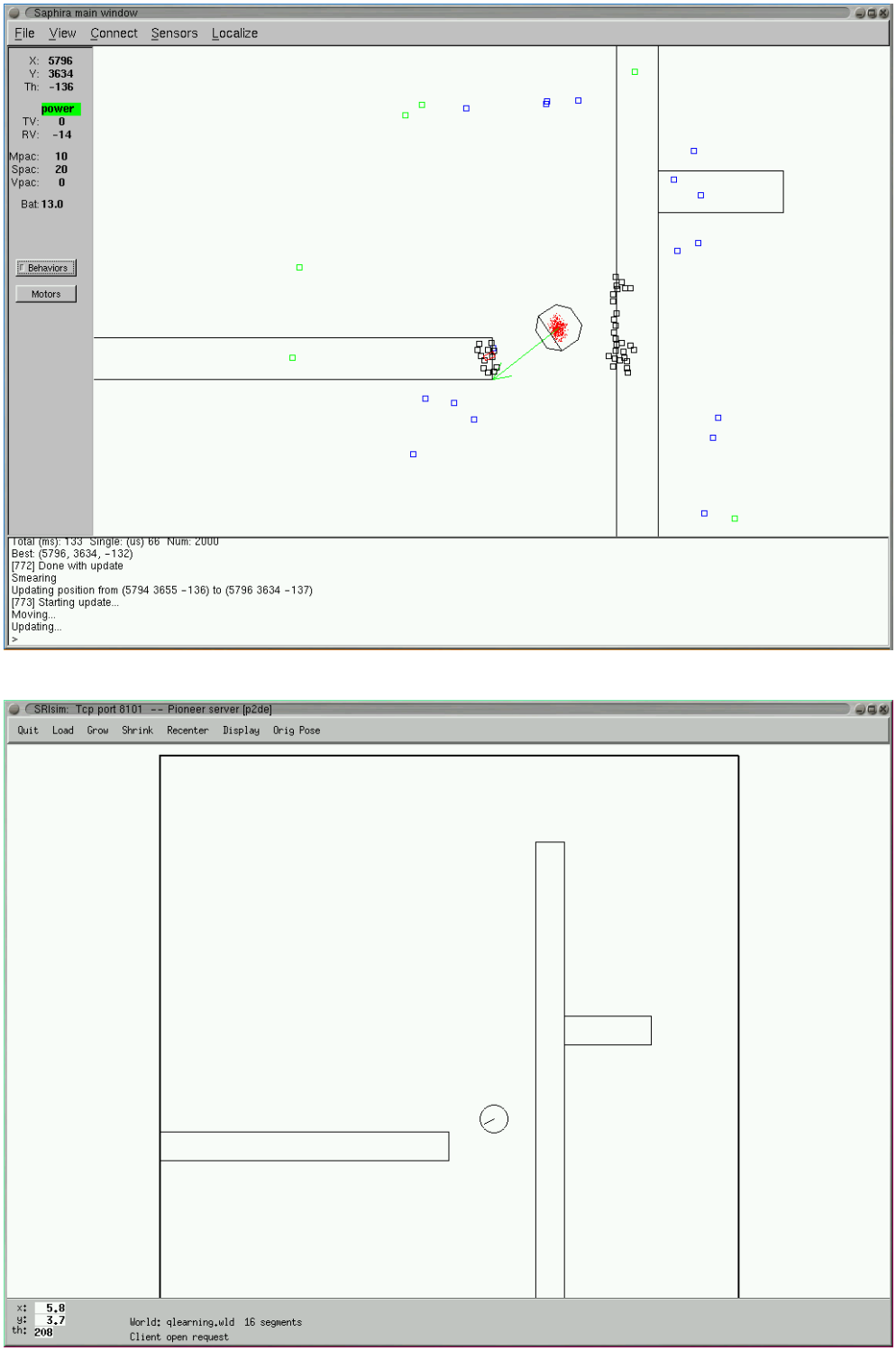


Figura 5.18: A plataforma Saphira 8.0. A figura superior apresenta a tela do aplicativo (com a posição do robô no plano de referência) e a inferior, o monitor do simulador (apresentando a posição real do robô).

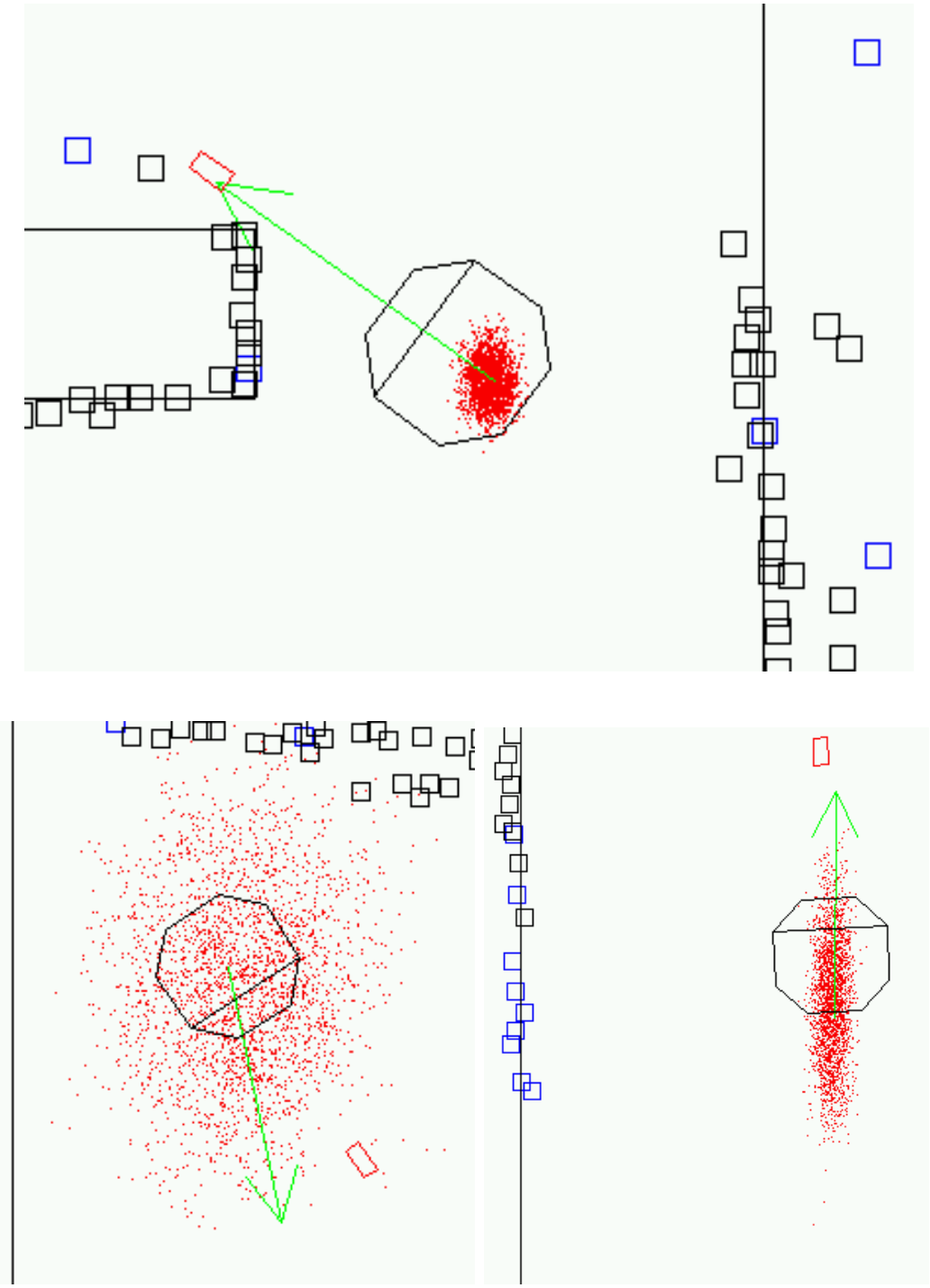


Figura 5.19: Localização Monte Carlo: os pontos vermelhos indicam a posição das partículas, cada uma definindo uma posição provável do robô. A incerteza na posição do robô é menor na imagem superior do que nas inferiores.

O simulador Saphira permite simular diversos tipos de erros. A figura 5.18 apresenta um erro de posicionamento, onde o robô se encontra em uma posição ligeiramente acima da posição estimada. Já a figura 5.20 apresenta o robô atravessando uma parede no plano de referência, fato que ocorre devido a erros de localização (na verdade o robô real se encontra um pouco distante da parede).

O robô simulado neste trabalho foi o Pioneer 2DX, fabricado pela ActivMedia Robotics. Este robô tem dimensões de 44 x 38 centímetros e pesa 9 Kg. Possui 2 motores funcionando em modo diferencial – um robô da classe cinemática (2, 0) – acoplados às rodas, que têm 19 centímetros de diâmetro. Odômetros presentes no eixo do motor permitem medidas com uma resolução de 500 pulsos por revolução da roda. O robô é dotado ainda de seis sonares frontais e dois laterais, que permitem uma abrangência de 180 graus.

Para realizar o aprendizado, o ambiente foi particionado em células, cada uma aproximadamente do tamanho do robô. O ambiente utilizado – um quadrado de 10×10 metros – foi discretizado em uma grade de 20 x 20 células. A orientação do robô foi particionada em 16 valores discretos, que englobam 20 graus cada. Assim, a variável de estado do robô (x, y, θ), usada pelo aprendizado, é uma representação grosseira do estado real do robô. Foram definidas quatro ações possíveis de serem executadas pelo robô: mover para frente ou para trás, por uma distância correspondente ao tamanho do robô, e girar, no próprio eixo, 20 graus no sentido horário ou anti-horário.

Neste domínio, foi realizada a comparação entre o *Q–Learning* e o HAQL utilizando o método "Heurística a partir da Exploração" para acelerar o aprendizado a partir do 5.º episódio. O ambiente utilizado nos testes pode ser visto na parte inferior da figura 5.18.

Para extrair a informação sobre a estrutura do ambiente foi utilizado o método "Estrutura a partir de Exploração": toda vez que passa por uma célula, o robô anota a passagem, computando as visitas às células. Ao final da quinta iteração, esta computação é limiarizada, resultando em um esboço do mapa do ambiente, usado para criar a heurística por meio da "Retropropagação de Heurísticas". A figura 5.21 apresenta um dos esboços de mapas criados. Como pode ser visto, o mapa não corresponde exatamente ao ambiente, resultando em uma heurística que não indica exatamente a política ótima. Além disso, a heurística criada

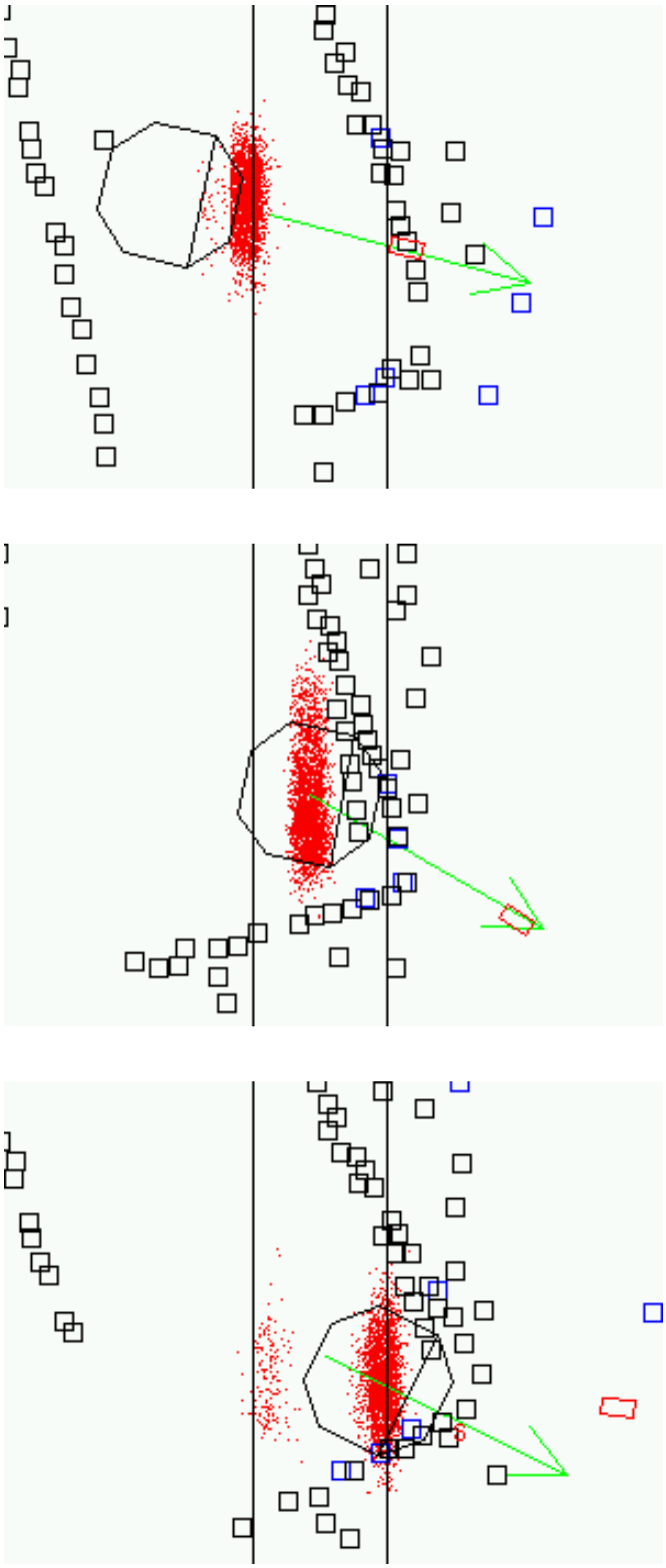
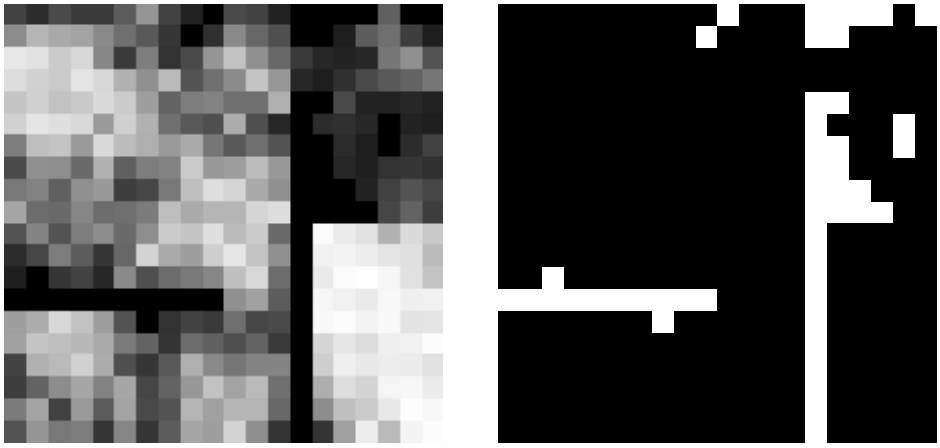


Figura 5.20: O robô atravessando uma parede – no plano de referência – devido ao erro de localização.



(a) Número de visitas **(b)** Mapa

Figura 5.21: Número de visitas (branco indica um número maior de visitas) e o esboço do mapa criado utilizando o método "Estrutura a partir de Exploração" para o ambiente Saphira.

leva o robô muito próximo às paredes, permitindo que ele fique imobilizado (ver figura 5.22).

Os parâmetros utilizados tanto no *Q–Learning* quanto no HAQL foram os mesmos: taxa de aprendizado $\alpha = 0,1$, $\gamma = 0,99$ e a taxa de exploração/explotação igual a 0,1. A localização Monte Carlo utilizou 10.000 partículas. Os reforços utilizados foram: 1000 para quando o robô chega ao estado meta e -10 quando executa uma ação. O robô inicia um episódio de treinamento em uma postura (posição e orientação) aleatória e a meta é chegar no canto superior direito do ambiente (definido pela região $9250 \leq x, y \leq 9750$).

Os resultados, que mostram a média de 30 treinamentos, podem ser visto na figura 5.23. Sobressai nesta figura a grande variação nos resultados do algoritmo *Q–Learning*. Isto ocorre porque, como ele explora aleatoriamente o ambiente, os erros de localizacao se acumulam e o robô se perde. Isto não acontece com o uso da heurística pois o número de passos é menor, diminuindo o erro acumulado. Na tabela 5.1 é apresentado o número de passos (média e desvio padrão) até se atingir a meta para o *Q–Learning* (na 2^a coluna) e para o HAQL (3^a coluna) e o resultado do teste t de Student para este experimento (4^a e 5^a coluna). Estes resultados mostram que, a partir da $5.^a$ iteração, os dois algoritmos atuam de maneira significativamente diferente, com nível de confiança próximo a 0,01%.

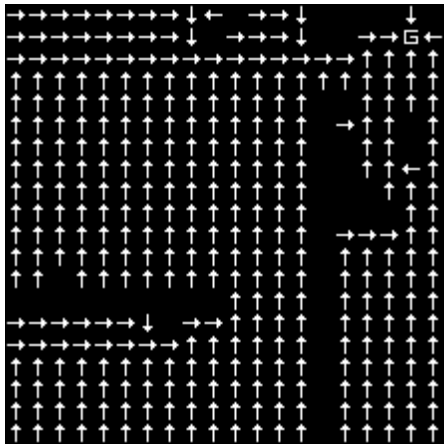


Figura 5.22: A heurística criada a partir do esboço do mapa apresentado na figura 5.21.

Episódio	Q–Learning	HAQL	Módulo de T	Nível de Confiança
5	7902 ± 8685	65 ± 51	4.942	0,01%
6	10108 ± 10398	55 ± 43	5.295	0,01%
7	8184 ± 12190	72 ± 61	3.644	0,02%
8	8941 ± 8367	75 ± 71	5.803	0,01%
9	8747 ± 9484	63 ± 55	5.015	0,01%

Tabela 5.1: Resultado do teste t de Student para aceleração no quinto episódio usando o algoritmo HAQL no ambiente de simulação Saphira.

Cada treinamento com 10 episódios utilizando o algoritmo *Q–Learning* levou em média 24 horas para ser concluído, enquanto o HAQL levou em média 1 minuto para realizar cada episódio. Os testes foram realizados em um Notebook Pentium 4m-2.2GHz com 512MB de memória RAM e sistema operacional Linux.

Finalmente, a figura 5.24 mostra os caminhos percorridos pelo robô utilizando o *Q–Learning* (superior) e o HAQL (inferior). Nas duas figuras, geradas com o resultado do quinto episódio de treinamento de ambos os algoritmos (portanto, com a heurística atuando no HAQL), o robô inicia no canto esquerdo inferior e deve atingir a meta localizada no canto direito superior. Pode-se notar que, utilizando o *Q–Learning*, o robô caminha aleatoriamente até encontrar a meta, executando 12081 passos para atingi-la, enquanto que, ao usar o HAQL, ele se dirige quase certeiramente para o alvo, executando 86 passos. Neste exemplo pode-se ver claramente que a ação da heurística é a de limitar a exploração no espaço de busca, direcionando a navegação do robô no ambiente, mas ainda permitindo pequenas explorações.

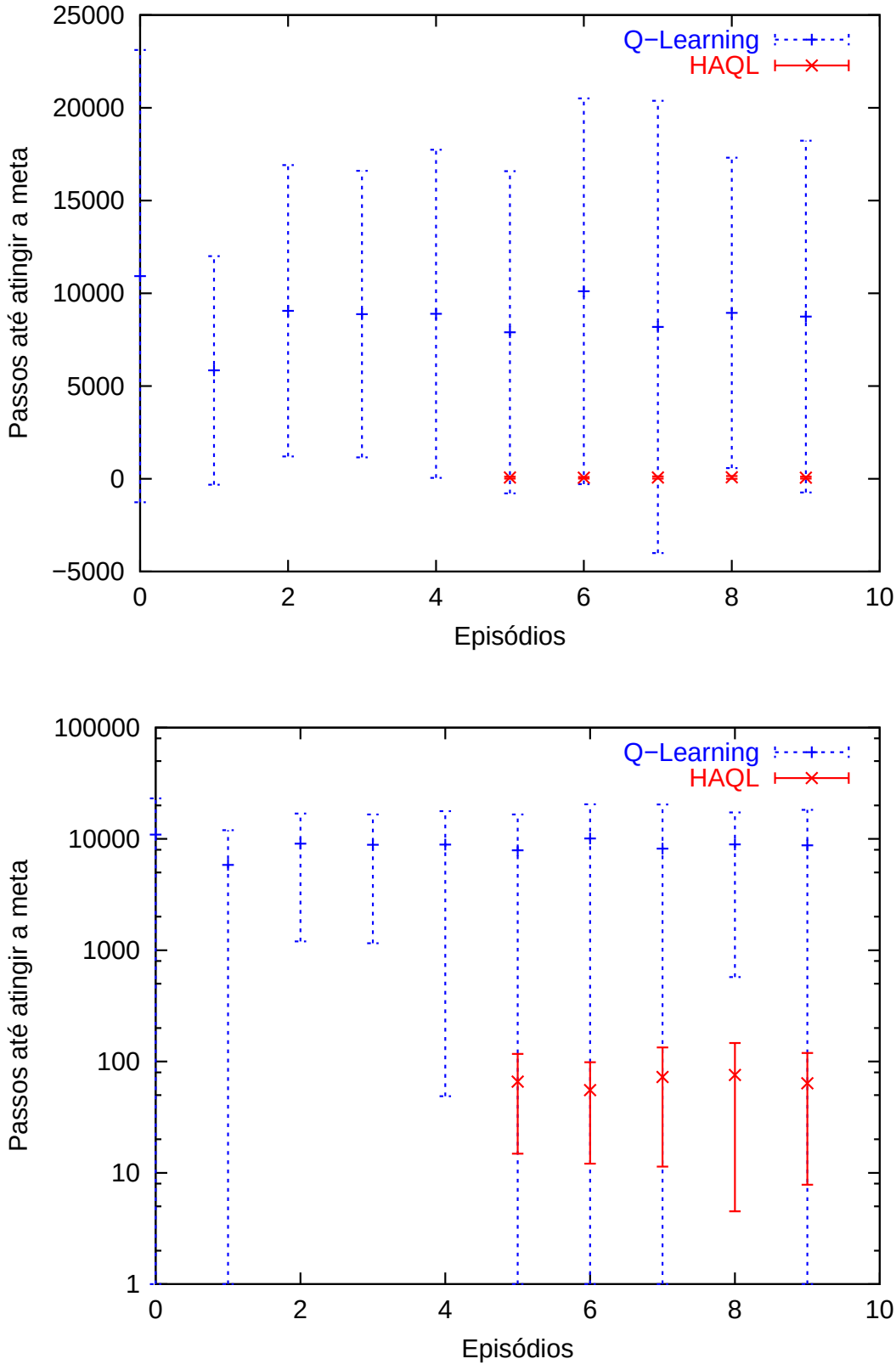


Figura 5.23: Resultado do uso da aceleração no quinto episódio usando o algoritmo HAQL no ambiente de simulação Saphira.

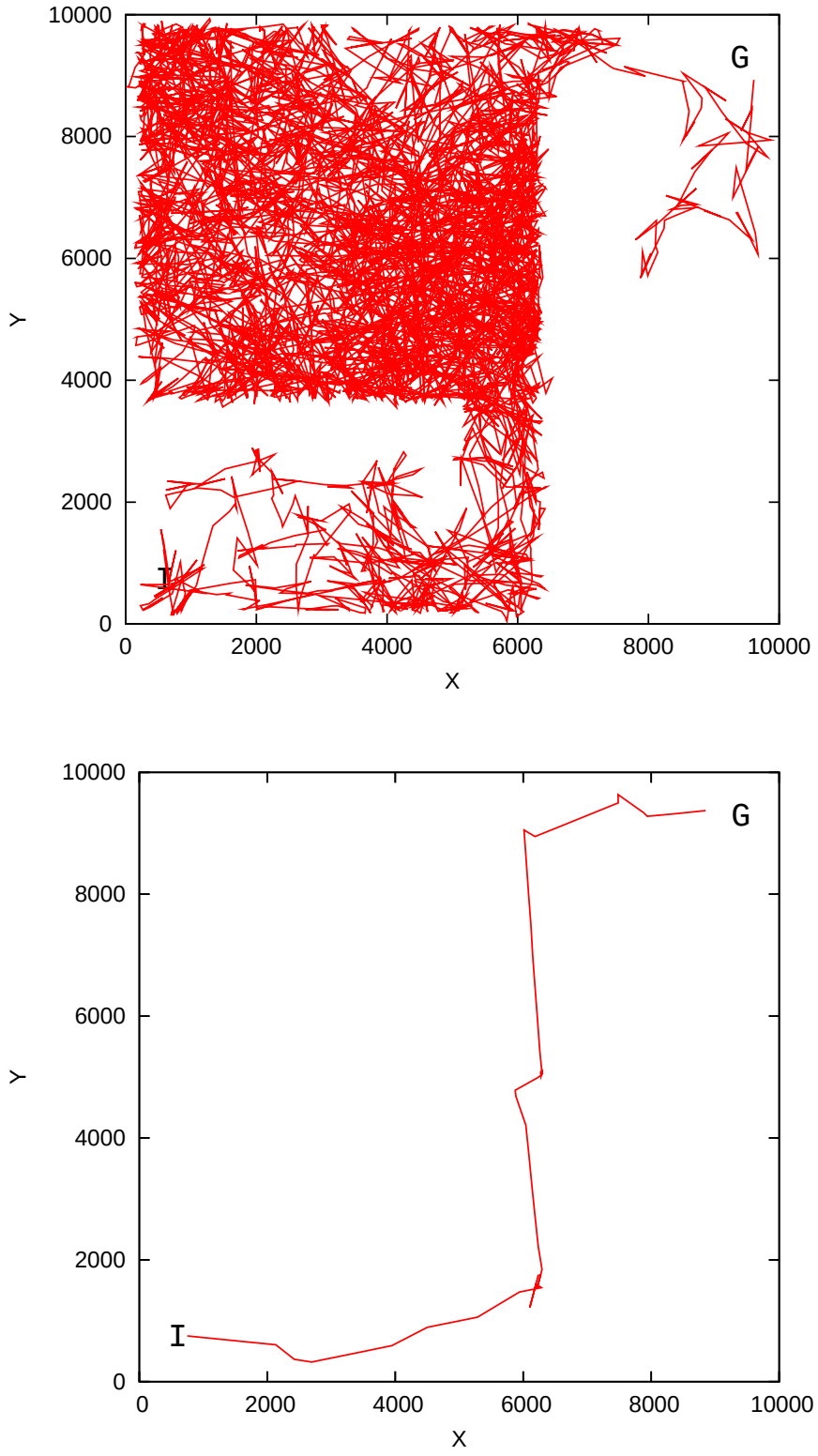


Figura 5.24: Caminhos percorridos pelo robô utilizando o *Q–Learning* (superior) e o HAQL (inferior) no ambiente de simulação Saphira 8.0.

5.5 Resumo

Os resultados obtidos neste capítulo indicam que o algoritmo *Heuristically Accelerated Q–Learning* pode ser usado com vantagens no domínio dos robôs móveis autônomos. Entre os métodos de composição de heurísticas apresentados neste trabalho, o que utiliza exploração para obter informações sobre a estrutura do ambiente foi o que permitiu chegar aos melhores resultados, mesmo em ambientes não determinísticos.

O próximo capítulo mostra que outros algoritmos da classe de "Aprendizado Heuristicamente Acelerado" podem ser criados e usados com heurísticas simples em diversos domínios importantes para a área de IA e controle.

6 Utilização de heurísticas *a priori* para aceleração do aprendizado

Neste capítulo são apresentados experimentos realizados com o objetivo de verificar se a abordagem baseada em heurísticas pode ser aplicada a outros algoritmos de Aprendizado por Reforço e se ela é independente do domínio. Além disso, as heurísticas usadas foram definidas com base apenas no conhecimento *a priori* do domínio, sendo mais simples que as estudadas no capítulo anterior.

Todos os novos algoritmos apresentados neste capítulo são similares ao *Heuristically Accelerated Q–Learning*, no sentido que eles utilizam a heurística apenas para influenciar a escolha das ações, através da soma de um valor H à função valor usada pelo algoritmo. As diferenças ocorrem devido ao funcionamento do algoritmo que está sendo acelerado, e serão detalhadas nas próximas seções.

6.1 O problema do Carro na Montanha utilizando HA–SARSA(λ)

O domínio do "Carro na Montanha" (*Mountain-Car Problem*), proposto por Moore (1991), tem sido usado por pesquisadores para testar modelos de aproximação de funções (SUTTON, 1996; MUNOS; MOORE, 2002), aceleração do aprendizado (DRUMMOND, 2002) entre outros problemas.

O problema a ser resolvido é o do controle de um carro que deve parar no topo de uma montanha, representada pela função $sen(3 \cdot x), x \in [-1.2, 0.5]$ (figura 6.1). O sistema possui duas variáveis de entrada contínuas, posição $x_t \in [-1.2, 0.5]$ e velocidade $v_t \in [-0.07, 0.07]$ do carro. Atua-se sobre o carro através de três ações

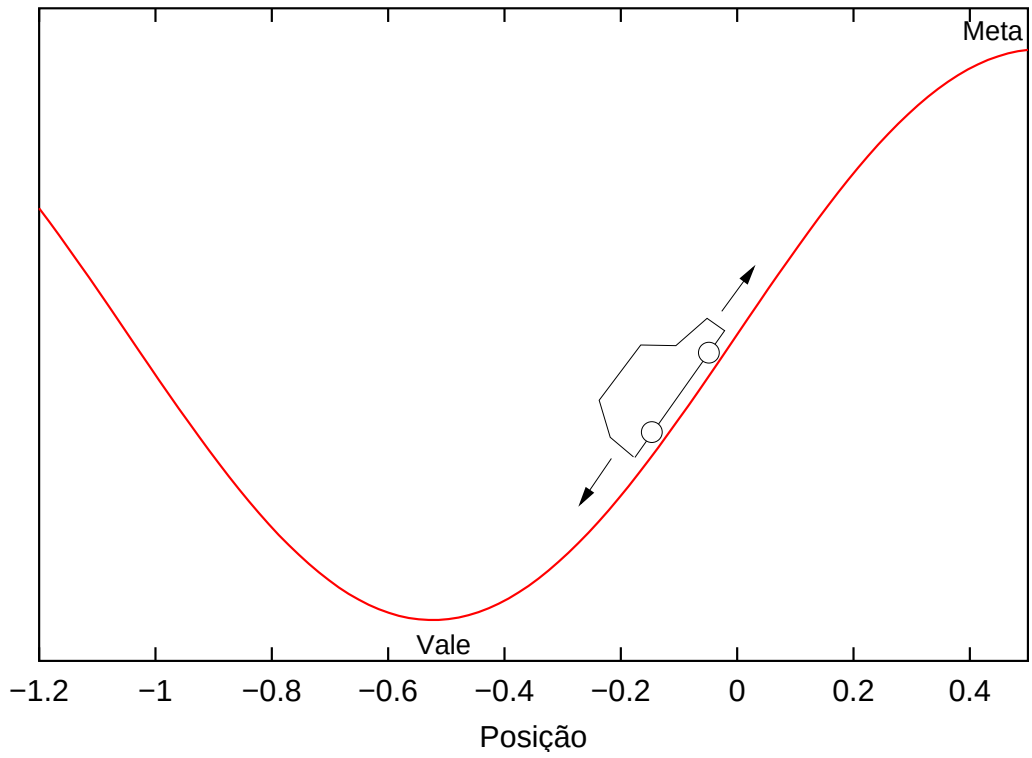


Figura 6.1: Descrição do problema do Carro na Montanha.

discretas: impulsão máxima para a frente ($F = 1$), impulsão máxima para trás ($F = -1$) ou nenhuma impulsão ($F = 0$).

O que torna o problema mais relevante é que o carro não possui potência suficiente para vencer a força da gravidade, tendo que se mover inicialmente no sentido contrário à meta para poder aumentar sua velocidade, "pegando impulso para subir a ladeira". A dinâmica do sistema é:

$$x_{t+1} = \min(0.5, \max(-1.2, x_t + v_{t+1})) \tag{6.1}$$

$$v_{t+1} = \min(0.07, \max(-0.07, v_t + 0.001F_t - 0.0025cos(3x_t))) \tag{6.2}$$

Este domínio tem como característica a não linearidade, a não existência de partições ou obstáculos retos, o que gera uma função valor onde as fronteiras são curvas (figura 6.2).

Para solucionar este problema foram utilizados quatro algoritmos:

- O *Q–Learning*, descrito na seção 2.4.

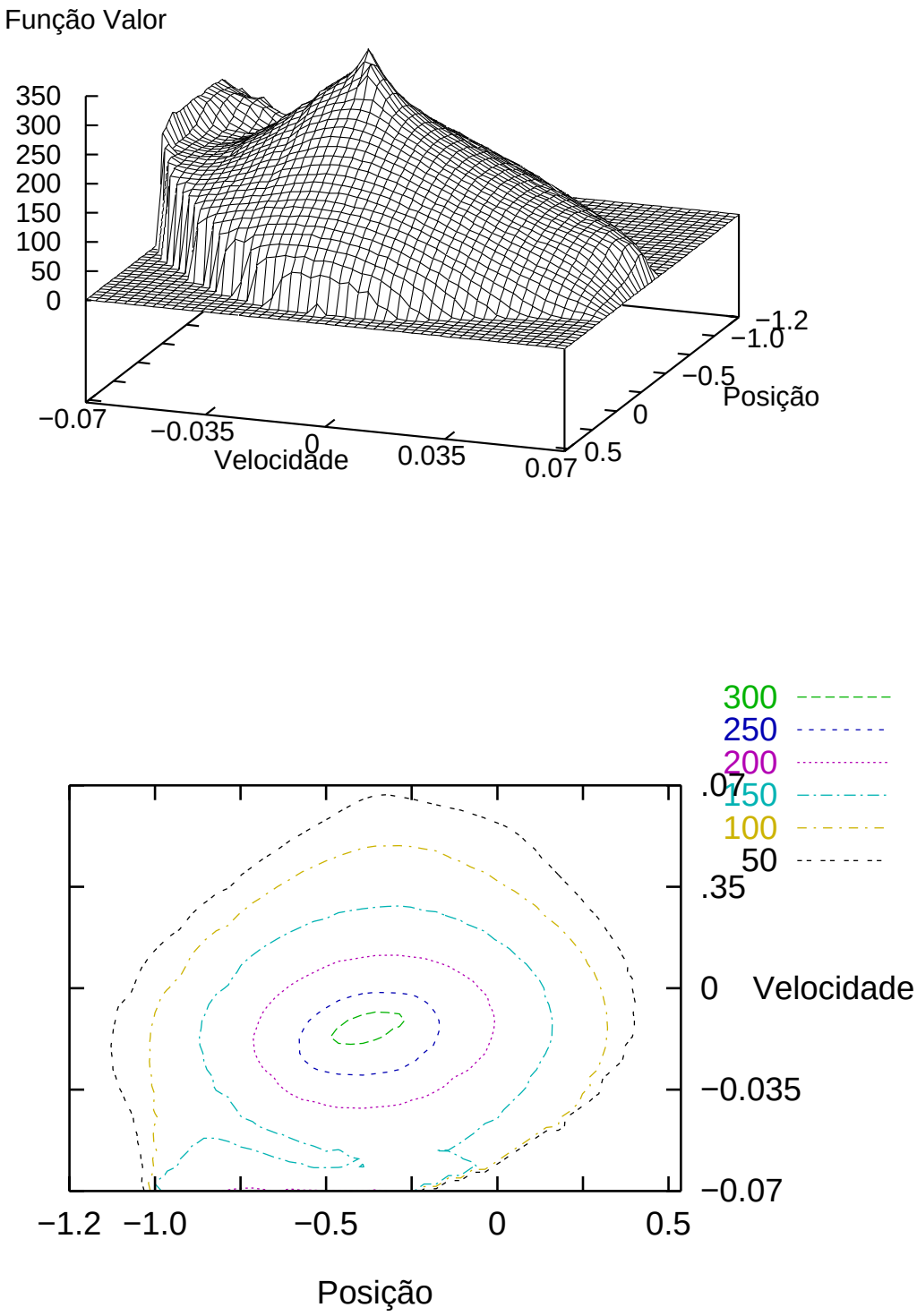


Figura 6.2: Tabela da função valor para o problema do Carro na Montanha (superior em 3D, inferior em curvas de nível).

- O *Heuristically Accelerated Q–Learning – HAQL*, proposto neste trabalho e descrito na seção 4.5.
- O SARSA(λ), descrito na seção 3.1.1.
- O *Heuristically Accelerated* SARSA(λ) – HAS(λ), que implementa uma versão acelerada por heurísticas para o algoritmo SARSA(λ).

Devido ao fato das variáveis de entrada serem contínuas, foi utilizado um aproximador de funções do tipo CMAC (seção 3.2.2) com 10 camadas e 8 posições de entrada para cada variável para representar a função valor-ação (nos quatro algoritmos).

A heurística utilizada foi definida a partir de uma regra simples: sempre aumente o módulo da velocidade. Assim, independente da posição onde se encontra o carro, se a velocidade for negativa, a heurística indica que se deve executar impulsão máxima para trás, se for positiva, impulsão para frente. A heurística foi armazenada em uma tabela que discretiza o problema em uma grade de 100 posições por 100 velocidades, e foi utilizada desde o instante inicial do treinamento. O valor da heurística usada tanto para o HAQL quanto para o HAS(λ) é definido a partir da equação (4.14) como sendo:

$$H(x_t, v_t, F_t) = \begin{cases} \max_f \hat{Q}(x_t, v_t, f) - \hat{Q}(x_t, v_t, F_t) + \eta & \text{se } v_t > 0 \text{ e } F_t = +1. \\ \max_f \hat{Q}(x_t, v_t, f) - \hat{Q}(x_t, v_t, F_t) + \eta & \text{se } v_t < 0 \text{ e } F_t = -1. \\ 0 & \text{caso contrário.} \end{cases} \tag{6.3}$$

Como, porém, a tabela Q inicia com valores iguais a zero, tem-se os valores da heurística a serem usados durante todo o processamento são:

$$H(x_t, v_t, F_t) = \begin{cases} +\eta & \text{se } v_t > 0 \text{ e } F_t = +1. \\ +\eta & \text{se } v_t < 0 \text{ e } F_t = -1. \\ 0 & \text{caso contrário.} \end{cases} \tag{6.4}$$

Os parâmetros utilizados foram quase os mesmos para os quatro algoritmos: taxa de aprendizado $\alpha = 0,5$, $\gamma = 0,99$ e $\lambda = 0,9$. A taxa de exploração/explotação foi definida como igual a 0,1. O valor da heurística η usado

Algoritmo	Número de passos médio na solução	Tempo (segundos)	Maior Reforço Recebido
Q-Learning	430 ± 45	31 ± 3	-3396 ± 560
SARSA(λ)	110 ± 13	23 ± 10	45 ± 41
HAQL	115 ± 1	7 ± 5	-78 ± 12
HAS(λ)	114 ± 1	4 ± 1	-85 ± 12

Tabela 6.1: A solução com o menor número de passos, o tempo médio para encontrá-la e o maior reforço recebido para o problema do Carro na Montanha.

foi igual a 100. O reforço utilizado foi de -10 ao executar uma impulsão (tanto $F = -1$ quanto $F = 1$), -1 quando não se executou nenhuma impulsão ($F = 0$) e 1000 ao chegar no estado meta. Os experimentos foram codificados em Linguagem C++ (compilador GNU g++) e executados em um Microcomputador AMD K6-II 500MHz, com 288MB de memória RAM e sistema operacional Linux.

A tabela 6.1 apresenta, para os quatro algoritmos, a solução com o menor número de passos encontrada e o tempo médio para encontrá-la, e o maior reforço recebido em um episódio (média de 30 treinamentos limitados a 500 episódios). Pode-se notar que o *Q–Learning* apresenta o pior desempenho nos três critérios de comparação, conforme o esperado.

Quanto ao número de passos para encontrar a meta os algoritmos SARSA(λ) HAQL e HAS(λ) apresentam resultados semelhantes. Mas apesar do SARSA(λ) encontrar uma solução com menor número de passos que o *Q–Learning*, o tempo para encontrar esta solução ainda é alto se comparado com o dos algoritmos HAQL e HAS(λ). Quanto ao maior reforço recebido em um episódio de treinamento, o SARSA(λ) apresentou um resultado ligeiramente melhor que o dos algoritmos acelerados.

A figura 6.3 mostra a evolução do número de passos necessários para atingir a meta dos quatro algoritmos (média de 30 treinamentos) para o carro iniciando sempre na posição 0.5 (no vale) e com velocidade nula. É possível verificar que, conforme esperado, o *Q–Learning* apresenta o pior desempenho (o início de sua evolução não é apresentado pois registra valores superiores a 30.000 passos) e os algoritmos acelerados pela heurística os melhores, com o SARSA(λ) entre eles. Esta figura permite que se infira o motivo pelo qual o tempo necessário para o algoritmo SARSA(λ) encontrar a solução com menor número de passos (apresentada na tabela 6.1) é maior que o tempo dos algoritmos acelerados: o

menor número de passos que tanto o HAQL quanto o HAS(λ) executam no início do treinamento.

É interessante notar no início da evolução o algoritmo HAS(λ) apresenta um desempenho pior que o apresentado após o quinquagésimo episódio. Isto ocorre porque o carro, estando com velocidade inicial igual a zero, não é influenciado pela heurística a tomar uma decisão para qual lado seguir. Mas se ele mover-se em direção à montanha, não vai conseguir chegar no cume pois não tem velocidade inicial. Assim, o que deve ser aprendido é que, estando na posição inicial, com velocidade igual a zero, a ação a ser tomada é ir para a direção contrária à da montanha. Durante este aprendizado, o aprendiz recebe mais reforços negativos (figura 6.4), e aprende que a primeira ação a ser executada deve ser uma impulsão para trás. A figura 6.4 mostra ainda a média dos reforços recebidos pelos algoritmos SARSA(λ), HAQL e HAS(λ) (não é mostrado o Q-Learning pois este não tem a média superior a -10.000)

Os caminhos efetuados pelo carro no espaço de estados (posição $\times$ velocidade) quando controlado pelos diferentes algoritmos podem ser vistos nas figuras 6.5 e 6.6. Na figura 6.5-superior pode-se ver como o SARSA(λ) explora o ambiente ao início do treinamento, convergindo para o caminho de menor custo. Ao se comparar com o HAS(λ) (figura 6.5-inferior), nota-se que a grande vantagem do algoritmo acelerado é a de não realizar uma exploração tão intensa. Esta figura apresenta ainda um caminho no qual a primeira ação tomada foi ir em direção à montanha, ocorrida no vigésimo episódio. A figura 6.6 mostra os melhores caminhos dos quatro algoritmos. Apesar de serem diferentes, os caminhos dos algoritmos SARSA(λ), HAQL e HAS(λ) têm um número de passos próximos e recebem a mesma quantidade de reforços negativos, bem menor que o caminho encontrado pelo *Q–Learning*.

A figura 6.7-superior mostra a relação entre o valor de η utilizado pela heurística e o número de passos para atingir a meta. Pode-se notar que quanto maior o valor de η, menor o número de passos médio a cada episódio. Já a figura 6.7-inferior mostra a relação entre o valor de η e o reforço recebido. Novamente, quanto maior o valor da heurística, maior o reforço médio recebido.

Finalmente, foi utilizado o teste t de Student para verificar a hipótese de que o algoritmo HAS(λ) é mais rápido que o SARSA(λ). Para cada episódio é

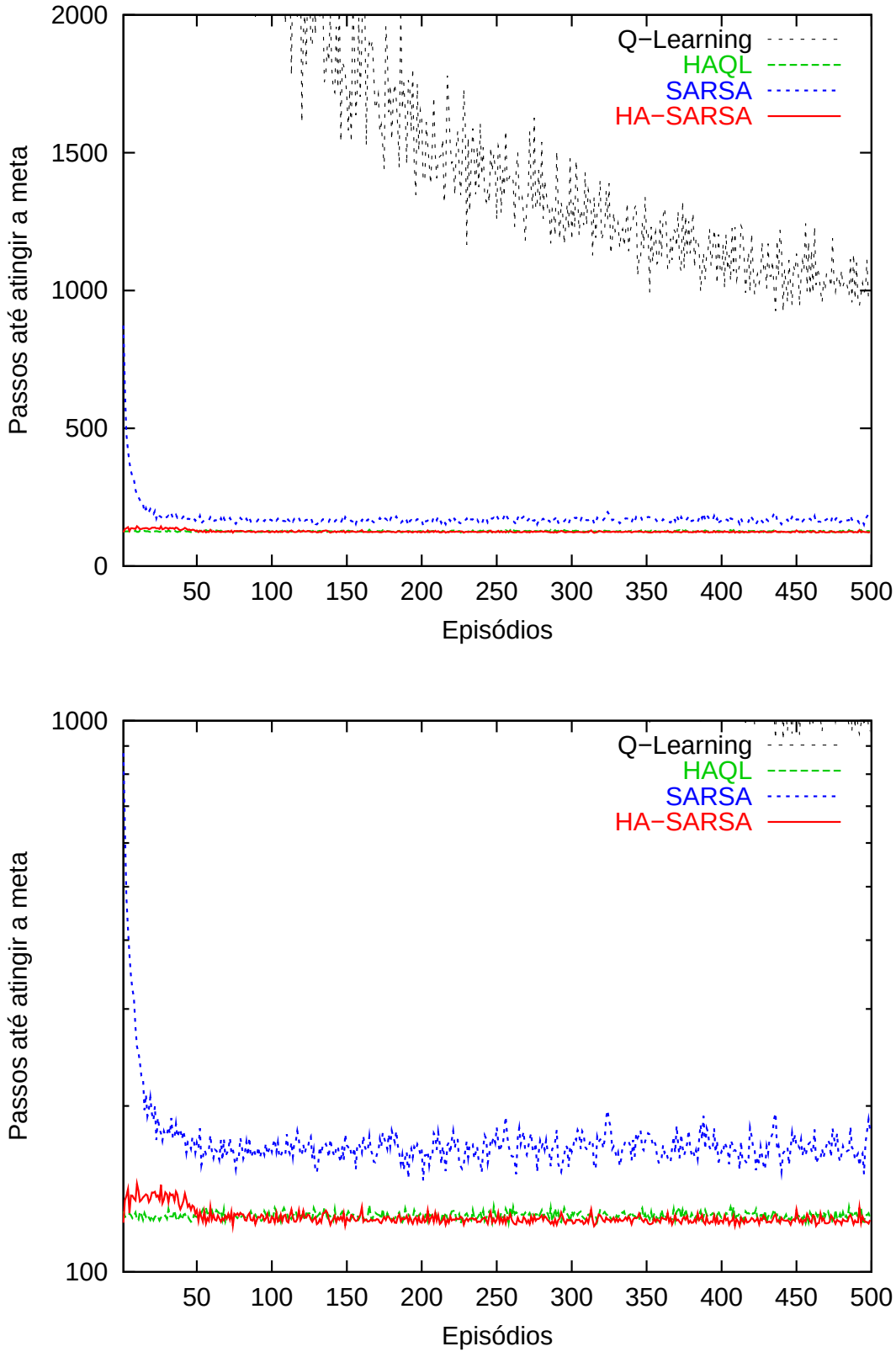


Figura 6.3: Número de passos necessários para atingir a meta para os algoritmos *Q–Learning*, SARSA(λ), HAQL e HAS(λ) para o problema do Carro na Montanha (inferior em monolog).

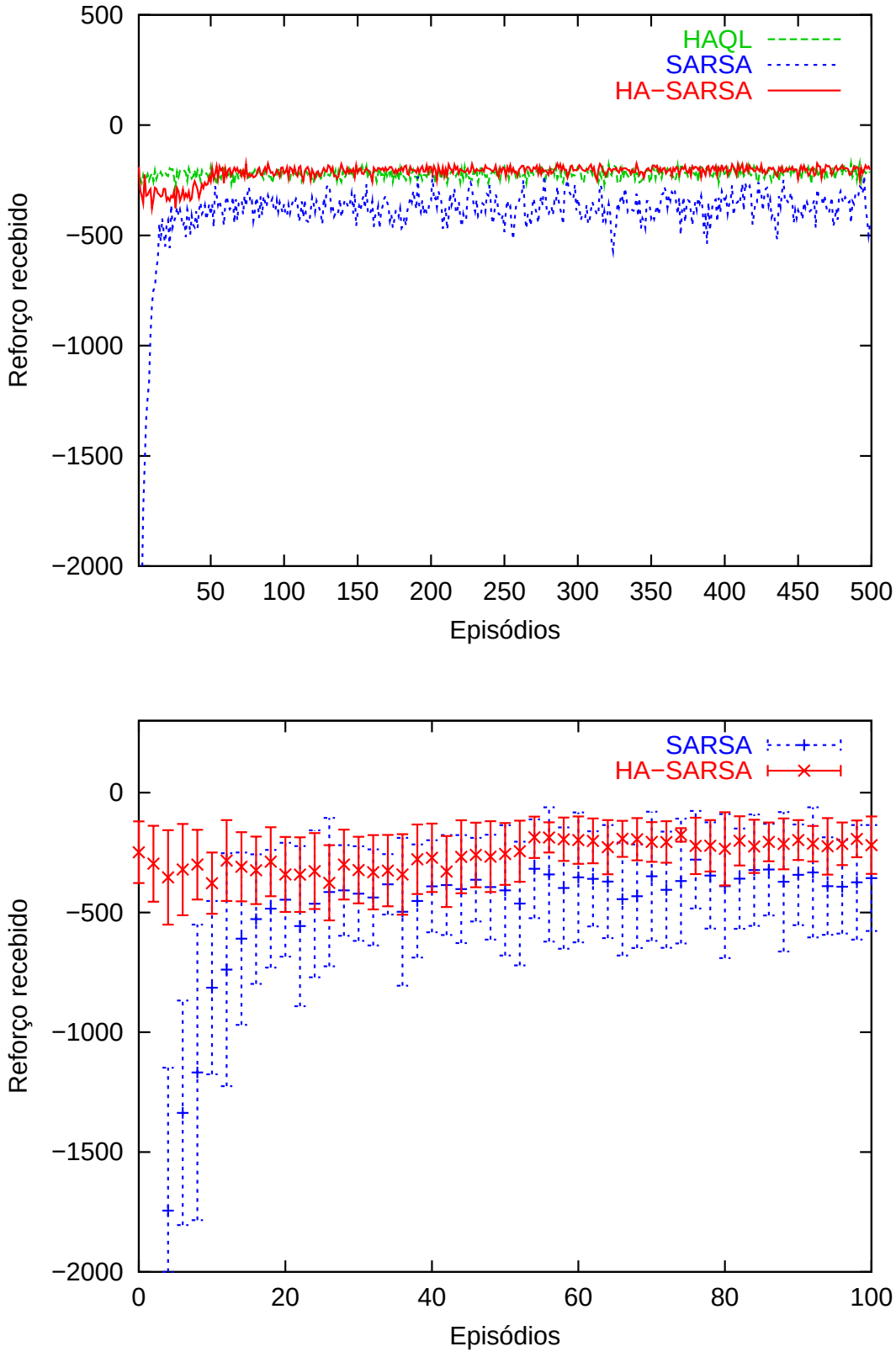


Figura 6.4: O reforço recebido para os algoritmos SARSA(λ), HAQL e HAS(λ) para o problema do Carro na Montanha (inferior com barras de erros).

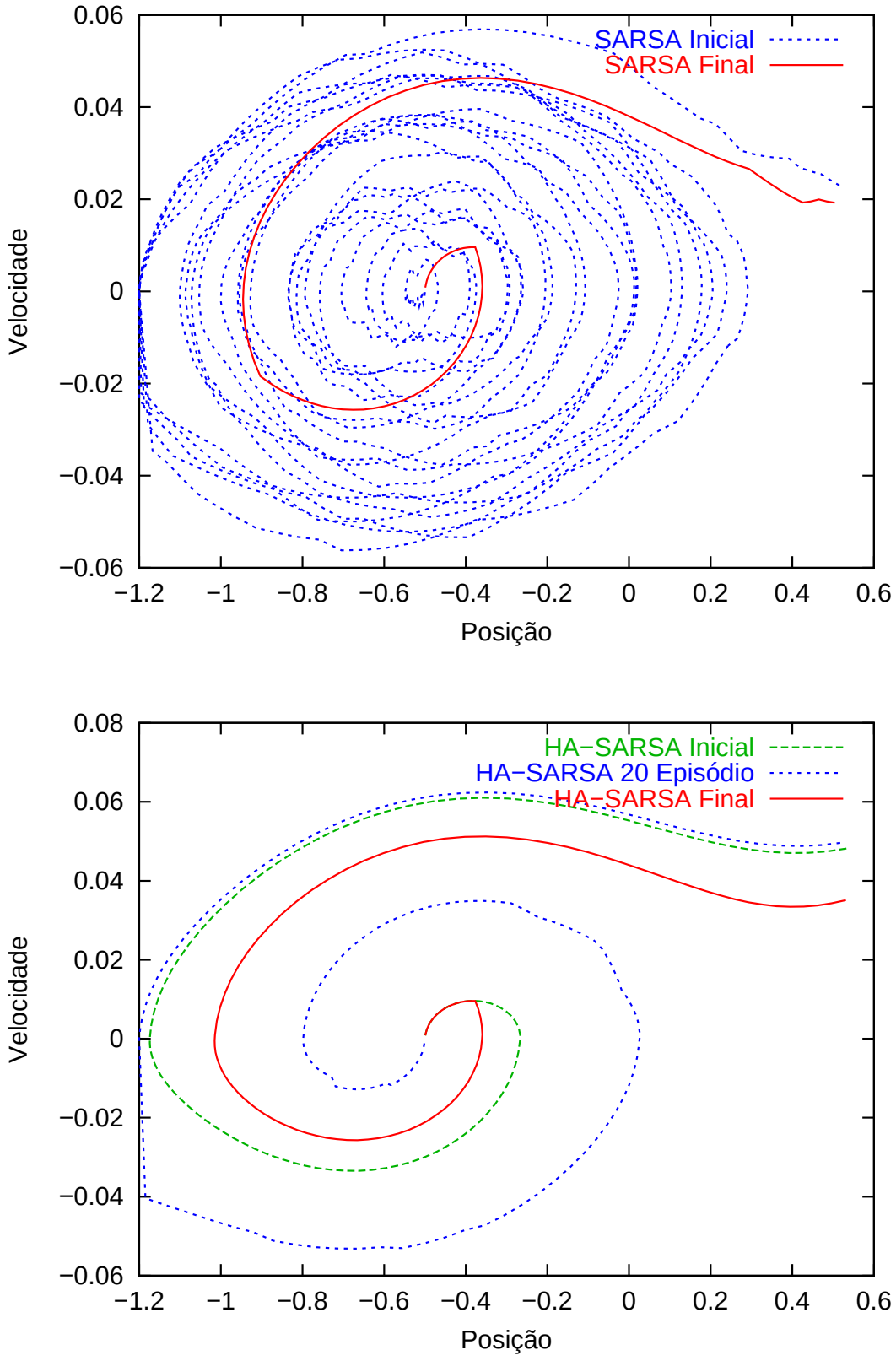


Figura 6.5: Caminhos realizados no espaço de estados pelos algoritmos SARSA(λ) (superior) e HAS(λ) (inferior) para o problema do Carro na Montanha.

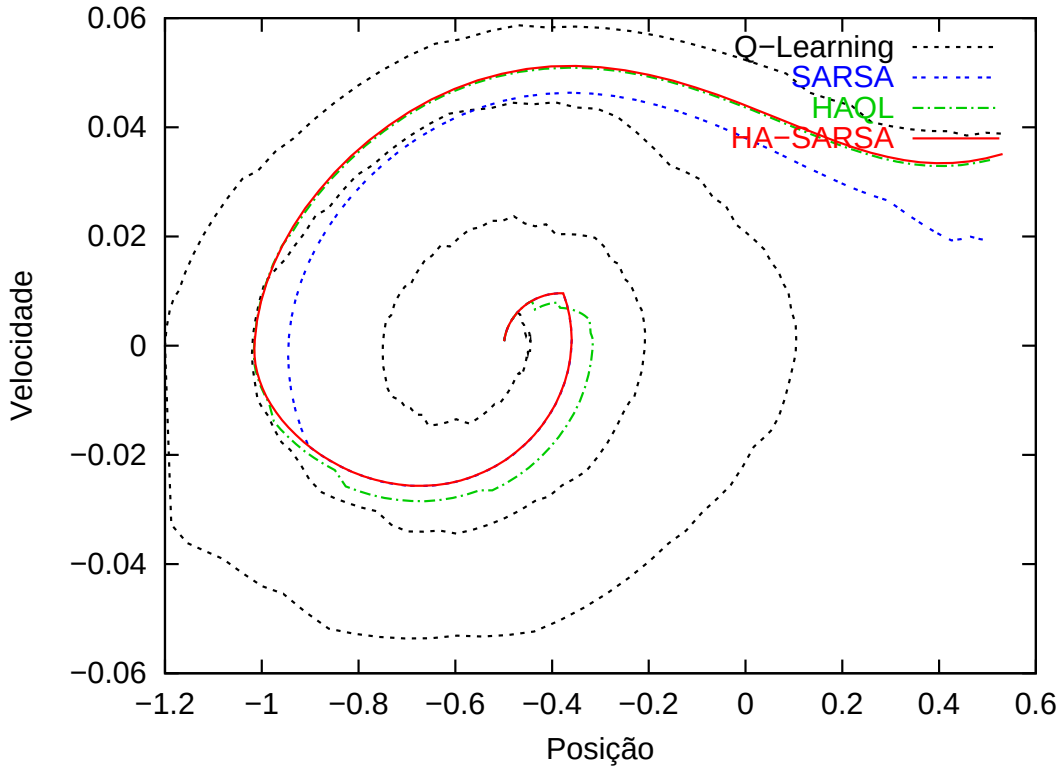


Figura 6.6: Caminhos finais realizados no espaço de estados para o problema do Carro na Montanha.

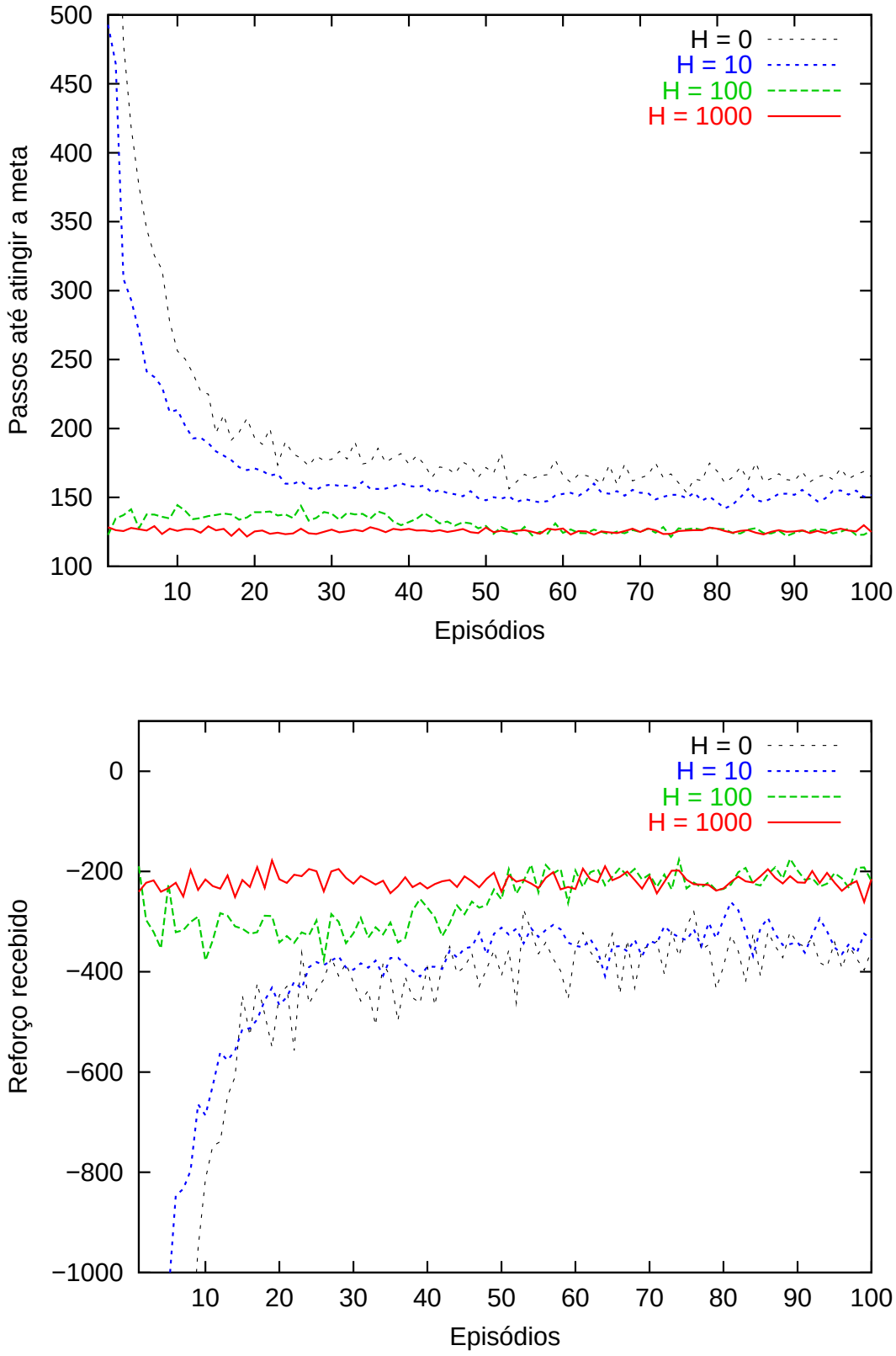


Figura 6.7: Comparação para o uso de vários valores de heurística no HAS(λ) para o problema do Carro na Montanha (Número de passos na figura superior e reforço recebido na inferior).

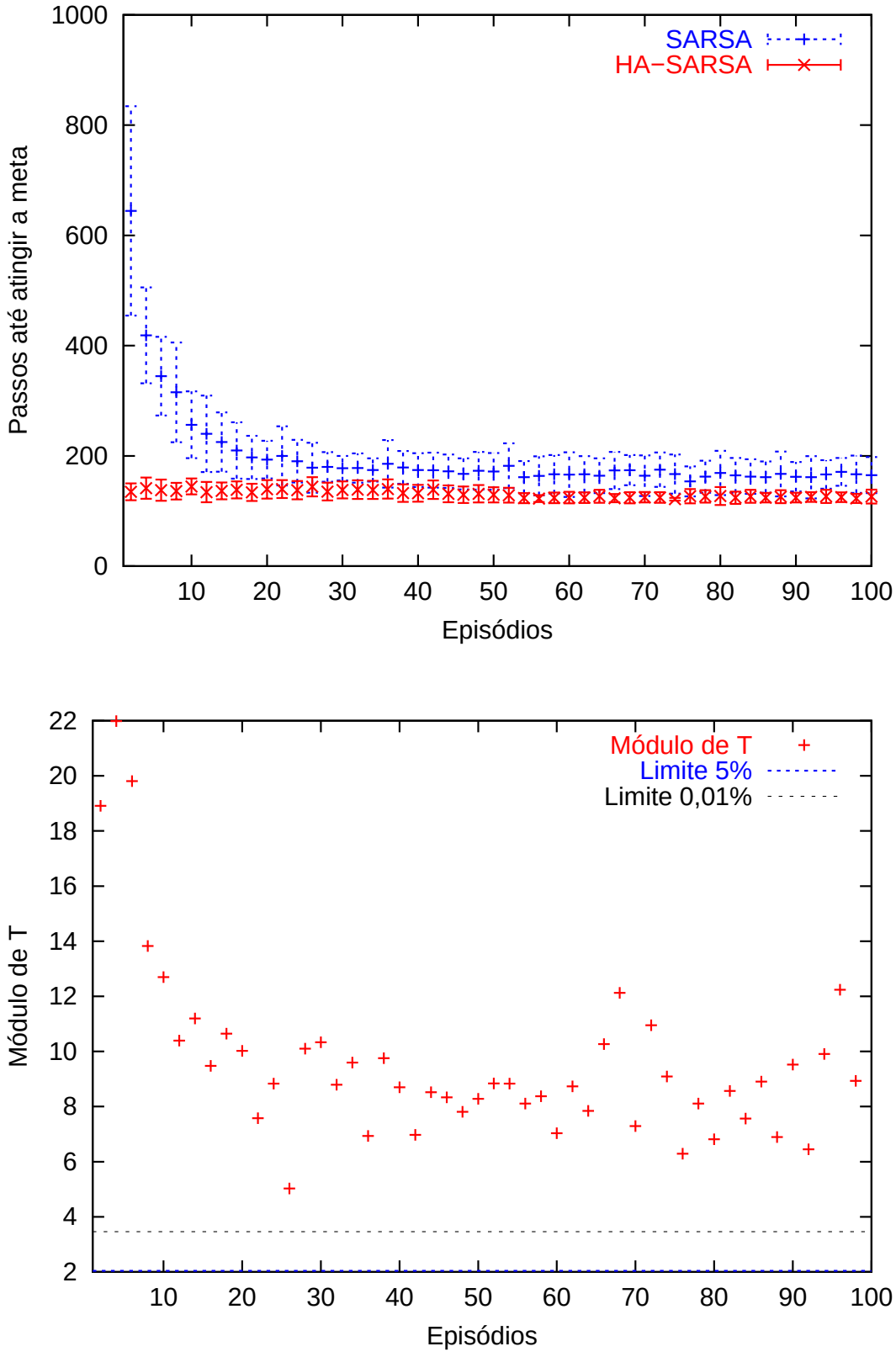


Figura 6.8: Comparação entre algoritmos SARSA(λ) e HAS(λ) para o problema do Carro na Montanha. A figura superior mostra o resultado com barras de erros e a inferior mostra o resultado do teste t de Student.

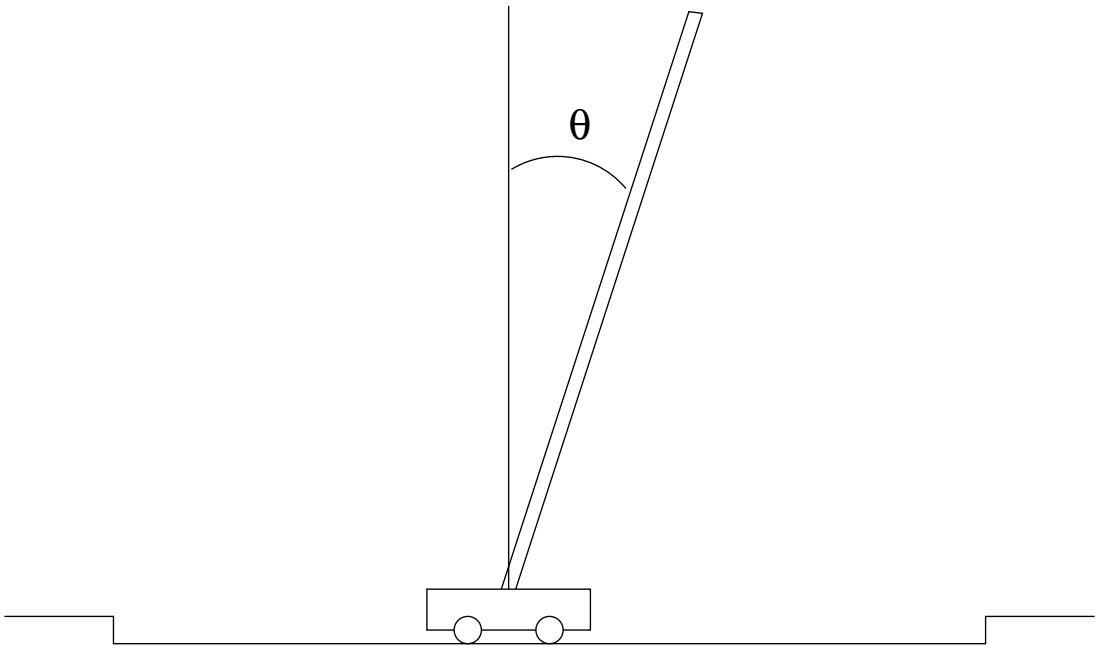


Figura 6.9: O problema do Pêndulo Invertido.

calculado o valor do módulo de T utilizando o número de passos para atingir a meta, apresentados na figura 6.8-superior. Os resultados, mostrados na figura 6.8-inferior, são significativamente diferentes devido à aceleração, com nível de confiança maior que 0,01%.

6.2 O problema do Pêndulo Invertido utilizando HATD(λ)

Um problema muito semelhante ao do carro na montanha é o controle do Pêndulo Invertido (OGATA, 1998, p. 71) (também conhecido como problema do *Cart-Pole*), sendo um dos sistemas clássicos usados como *benchmark* em controle.

O sistema é constituído por uma base móvel sobre a qual oscila uma haste rígida, que pode se deslocar ao longo de um trilho (unidimensional) limitado em ambos os lados. O objetivo de controle é manter a haste na posição vertical, através da aplicação de forças que movem a base para frente ou para trás. O comportamento oscilante da haste tenta reproduzir o problema de estabilização freqüente em situações típicas, tais como o controle da trajetória de um projétil, ou do movimento de um satélite.

As variáveis de estado deste problema são contínuas: a posição do centro da base $y_t \in [-2.4, 2.4]$ metros, o ângulo entre a haste e a vertical, $\theta_t \in [-12, 12]$ graus, a velocidade da base $\dot{y}_t$ e a velocidade angular do pêndulo $\dot{\theta}_t$. Admitindo que o ângulo e a velocidade angular são pequenos ($\theta \equiv sin(\theta), cos(\theta) \equiv 1$), a dinâmica é dada pelas equações (OGATA, 1998, p. 699):

$$\ddot{\theta}_{t+1} = \frac{(M+m)\ g\ \theta_t - F}{M\ l} \tag{6.5}$$

$$\ddot{y}_{t+1} = \frac{F - m\ g\ \theta_t}{M} \tag{6.6}$$

onde:

- M é a massa da base.
- m é a massa do pêndulo.
- l é a distância entre o centro de massa do pêndulo e a base (metade da altura do pêndulo).
- F é a força utilizada em cada ação.
- g é a aceleração da gravidade.

Para solucionar este problema foram utilizados dois algoritmos:

- O TD(λ), descrito na seção 2.5.
- O *Heuristically Accelerated* TD(λ) – HATD(λ), que implementa uma versão acelerada por heurísticas para o TD(λ).

A heurística utilizada foi similar à utilizada na seção anterior: se o pêndulo estiver caindo para a frete, mova a base para a frente; se estiver caindo para trás, mova para trás. Esta heurística, similar à usada por um ser humano ao tentar equilibrar o pêndulo invertido, foi definida como:

$$H(y_t, \theta_t) = \begin{cases} +\eta & \text{se } \theta_t > 0, \forall y_t \\ -\eta & \text{se } \theta_t < 0, \forall y_t \end{cases} \tag{6.7}$$

Esta heurística influencia a escolha das ações, que é dada por:

$$A = \frac{1}{1 + e^{V(y_t,\theta_t)+H(y_t,\theta_t)}} \tag{6.8}$$

Algoritmo	Treinamentos bem sucedidos	Episódios	Passos
TD(λ)	48	103 ± 58	411393 ± 366272
HATD(λ)	47	78 ± 31	330387 ±229681
Módulo de T		2,612	1,288
Confiança		1%	15%

Tabela 6.2: Resultados para o problema do Pêndulo Invertido utilizando os algoritmos TD(λ) e HATD(λ).

Se o arredondamento de A for igual a zero, a força é aplicada no sentido positivo; caso o arredondamento seja igual a um, a força é aplicada no sentido contrário.

O objetivo de controle é equilibrar o pêndulo perto da posição vertical durante 100.000 instantes de tempo, mantendo o ângulo no limite $-12^o \leq \theta \leq 12^o$. Uma falha ocorre se o pêndulo ultrapassar este limite, se a base atingir o final do trilho em qualquer um dos lados ou se for atingido um limite de 300 episódios de treinamento. Após uma falha, o pêndulo é reinicado na posição vertical e um novo episódio é iniciado.

Neste experimento a massa da base foi definida como 1 quilograma e a do pêndulo, 100 gramas. O tamanho do pêndulo é de 1 metro e a força utilizada em cada ação é de ± 10 Newtons. A atualização das variáveis ocorre a cada 0,02 segundos. Os parâmetros utilizados para o aprendizado foram os mesmos para os dois algoritmos: taxa de aprendizado $\alpha = 0,5$, $\gamma = 0,95$ e $\lambda = 0,9$. O valor da heurística η usado foi igual a 10. O reforço utilizado foi de -1 quando ocorre uma falha.

A tabela 6.2 apresenta os resultados obtidos. Pode-se ver que tanto o número de episódios de treinamento quanto o número de passos necessários para aprender a equilibrar a haste é menor no caso do HATD(λ). O teste t de Student foi utilizado para verificar se a hipótese que o algoritmo HATD(λ) é mais rápido que o TD(λ) é válida. O resultado indica que os algoritmos diferem significativamente, sendo o HATD(λ) mais rápido que o TD(λ) devido a aceleração, com nível de confiança maior que 1% para o número de episódios de treinamento. Porém o teste t rejeita a existência de uma diferença significativa para o número de passos, sendo, no entanto, o valor obtido para o módulo de T (1,288) próximo ao valor da probabilidade crítica de 10% de confiança (que é 1,299). Foi realizada uma média de 50 épocas de treinamento.

Os experimentos foram codificados em Linguagem C++ (compilador GNU g++), baseado no código fonte disponível em Sutton e Barto (1998), e executados em um Microcomputador AMD K6-II 500MHz, com 288MB de memória RAM e sistema operacional Linux.

6.3 O problema do Caixeiro Viajante utilizando HADQL

Dados um conjunto com n cidades $C = \{c_1, c_2, \cdots, c_n\}$ e as distâncias $d_{i,j}$ entre elas, o Problema do Caixeiro Viajante (*Travelling Salesman Problem* – TSP) consiste em encontrar o caminho que visita todas as cidades em C exatamente uma vez e cujo comprimento total seja o menor possível. Este problema de otimização estudado por todas as disciplinas ligadas à computação pode ser classificado em dois tipos: o TSP simétrico, onde a distância da cidade i para a cidade j é igual à da cidade j para a cidade i e o assimétrico (ATSP), onde esta distância pode ser diferente.

Para solucionar este problema foram utilizados três algoritmos:

- O *Q–Learning* Distribuído – DQL, descrito na seção 3.3.1.
- A Otimização por Colônia de Formigas – ACO, descrita na seção 3.3.3.
- O *Heuristically Accelerated Distributed Q–Learning* – HADQL, que implementa uma versão modificada e acelerada por heurísticas para o DQL.

O *Heuristically Accelerated Distributed Q–Learning* implementado é uma versão que modifica o DQL em três aspectos. Primeiro, diferente do DQL, o HADQL utiliza uma única tabela Q para todos os agentes (não existe a tabela cópia Qc). Com isso, todos os agentes atualizam a função valor-ação durante a solução do problema. Aliado a esta modificação, é dado apenas reforço negativo instantâneo aos agentes, definido como a distância entre as cidades, $r = -d_{i,j}$.

O terceiro aspecto que torna o HADQL diferente do DQL é o uso da heurística, que é definida como o inverso da distância vezes uma constante η:

$$H[c_i, c_j] = \frac{\eta}{d_{i,j}}$$

Os parâmetros utilizados foram os mesmos para os três algoritmos: taxa de aprendizado $\alpha = 0,1$, $\gamma = 0,3$ e a taxa de exploração/explotação de 0,1. Para o ACO foi usado β igual a dois e para o HADQL utilizou-se $\eta = 10^8$. Estes parâmetros correspondem aos definidos por Gambardella e Dorigo (1995) e que foram utilizados tanto por Mariano e Morales (2000b) quanto Bianchi e Costa (2002a) em seus experimentos. Os algoritmos foram codificados em Linguagem C++ (compilador GNU g++) e executados em um Microcomputador AMD Athlon 2.2MHz, com 512MB de memória RAM e sistema operacional Linux.

Os testes foram realizados utilizando problemas do caixeiro viajante padronizados, disponíveis na TSPLIB (REINELT, 1995). Esta biblioteca, que foi utilizada para *benchmark* tanto por Gambardella e Dorigo (1995) quanto por Mariano e Morales (2000b), disponibiliza problemas de otimização padronizados como o do caixeiro viajante, de carga e descarga de caminhões e problemas de cristalografia. Os resultados para a média de 30 épocas de treinamentos com 1.000 episódios são apresentados nas tabelas 6.3, 6.5, 6.4.

A tabela 6.3 apresenta o melhor resultado encontrado pelos algoritmos DQL, ACO e HADQL após 1000 iterações. Pode-se notar que o HADQL encontra soluções melhores que os outros dois algoritmos para todos os problemas. O mesmo ocorre para a média dos melhores resultados encontrados (tabela 6.4). A figura 6.10 mostra a evolução da média dos resultados encontrados pelos algoritmos DQL, ACO e HADQL para o problema kroA100, onde se vê que o HADQL converge mais rapidamente para a solução.

O pior desempenho em relação ao menor caminho encontrado pelo algoritmo DQL em relação aos outros é causado pela não utilização de todas as experiências executadas em cada episódio, uma vez que o reforço é atribuído somente ao melhor caminho encontrado ao final de cada episódio.

O tempo médio para encontrar estas soluções, mostrado na tabela 6.5 indica que o DQL é o algoritmo mais rápido. O significado deste resultado é que o DQL estabiliza antes, porém em uma solução de pior qualidade. Uma comparação entre os tempos médios para encontrar a melhor solução entre o ACO e HADQL mostra que não existe diferença significativa entre eles. Vale notar que pequenas melhorias podem ocorrer em qualquer momento, pois ambos algoritmos realizam exploração $\epsilon - Greedy$. Por este motivo, a variação nos resultados é muito grande,

Problema	Número de Cidades	Tipo	Solução Conhecida	DQL	ACO	HADQL
berlin52	52	TSP	7542	15424	8689	7824
kroA100	100	TSP	21282	105188	25686	24492
kroB100	100	TSP	22141	99792	27119	23749
kroC100	100	TSP	20749	101293	24958	22735
kroD100	100	TSP	21294	101296	26299	23839
kroE100	100	TSP	22068	102774	26951	24267
kroA150	150	TSP	26524	170425	33748	32115
ry48	48	ATSP	14422	26757	16413	15398
kro124	100	ATSP	36230	122468	45330	42825
ftv170	171	ATSP	2755	18226	4228	3730

Tabela 6.3: Melhor resultado encontrado pelos algoritmos DQL, ACO e HADQL após 1000 iterações.

Problema	DQL	ACO	HADQL
berlin52	16427 ± 540	8589 ± 139	7929 ± 61
kroA100	108687 ± 2474	26225 ± 542	25114 ± 353
kroB100	105895 ± 2949	27337 ± 582	24896 ± 463
kroC100	105756 ± 2710	25985 ± 737	23585 ± 361
kroD100	104909 ± 2293	26188 ± 533	24441 ± 274
kroE100	108098 ± 2652	26723 ± 557	25196 ± 359
kroA150	179618 ± 3397	35816 ± 1001	33532 ± 603
ry48	29562 ± 1131	16285 ± 195	15538 ± 58
kro124	127911 ± 2485	46394 ± 599	43482 ± 322
ftv170	19278 ± 373	4532 ± 104	3982 ± 98

Tabela 6.4: Média dos resultados encontrados pelos algoritmos DQL, ACO e HADQL após 1000 iterações.

Problema	DQL	ACO	HADQL
berlin52	7 ± 3	12 ± 7	11 ± 6
kroA100	37 ± 13	89 ± 50	73 ± 43
kroB100	44 ± 17	85 ± 44	88 ± 47
kroC100	51 ± 27	82 ± 48	89 ± 38
kroD100	47 ± 21	98 ± 39	74 ± 39
kroE100	48 ± 22	80 ± 43	80 ± 45
kroA150	91 ± 42	267 ± 136	294 ± 154
ry48	6 ± 3	9 ± 6	3 ± 4
kro124	62 ± 25	89 ± 42	95 ± 43
ftv170	113 ± 73	317 ± 122	333 ± 221

Tabela 6.5: Tempo médio (em segundos) para encontrar a melhor solução para os algoritmos DQL, ACO e HADQL, limitados a 1000 iterações.

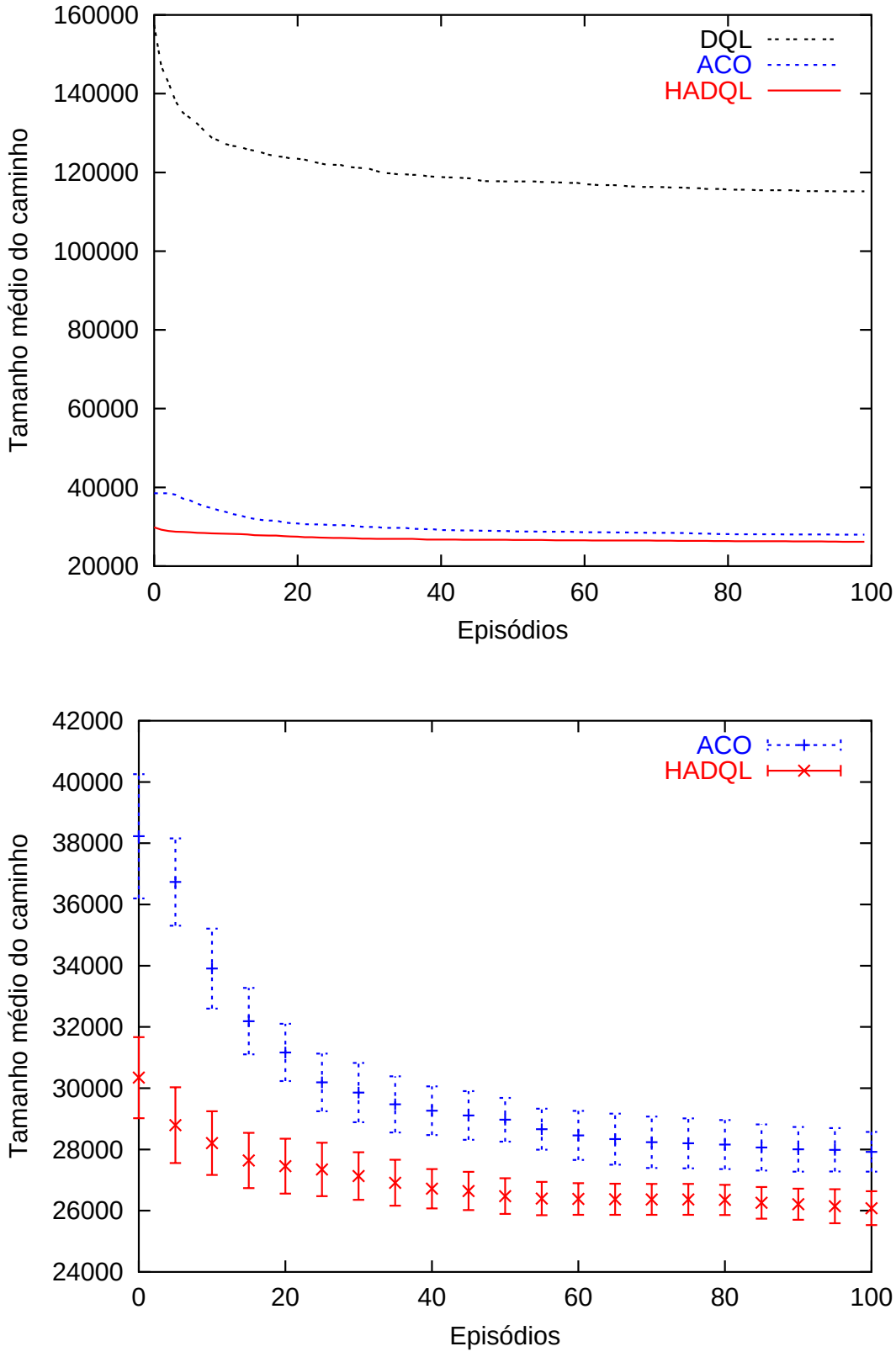


Figura 6.10: Comparação entre os algoritmos DQL, ACO e HADQL para o Problema do Caixeiro Viajante kroA100 (inferior com barras de erros).

sendo refletido no erro da medida.

Finalmente, foi utilizado o teste t de Student para verificar se é verdadeira a hipótese de que o algoritmo HADQL produz soluções melhores que o ACO. Os resultados mostraram que o aprendizado é significativamente diferente devido à aceleração, com nível de confiança maior que 0,01%, para todos os problemas. O mesmo teste aplicado ao tempo médio para chegar ao melhor resultado mostrou que não existe diferença significativa na velocidade de execução dos dois algoritmos.

6.4 Futebol de Robôs utilizando HA–Minimax–*Q*

O futebol de robôs foi proposto por diversos pesquisadores (KITANO et al., 1995; KITANO et al., 1997; KIM; VADAKKEPAT; VERNER, 1998; SANDERSON, 1997) para criar um novo desafio que possibilite a realização de experimentos reais para o desenvolvimento e testes de robôs que apresentem comportamento inteligente, cooperando entre si para a execução de uma tarefa, além de serem extremamente motivantes para possibilitar o surgimento de um espírito de ciência e tecnologia nas jovens gerações.

O futebol de robôs tem sido adotado como um problema padrão por dezenas de pesquisadores, visto que o desenvolvimento de times de robôs envolve muito mais que integração de técnicas de IA. Segundo Kraetzschmar et al. (1999), "dispositivos mecatrônicos, hardware especializado para o controle de sensores e atuadores, teoria de controle, interpretação e fusão sensorial, redes neurais, computação evolutiva, visão, aprendizado por reforço, e sistemas multiagentes são exemplos de campos envolvidos nesse desafio".

O domínio do Futebol de Robôs tem adquirido importância cada vez maior na área de Inteligência Artificial pois possui muitas das características encontradas em outros problemas reais complexos, desde sistemas de automação robóticos, que podem ser vistos como um grupo de robôs realizando uma tarefa de montagem, até missões espaciais com múltiplos robôs (TAMBE, 1998).

Para a realização dos experimentos foi utilizado o domínio introduzido por

Littman (1994), que propôs um Jogo de Markov com soma zero entre dois agentes modelado a partir do jogo de futebol de robôs.

Neste domínio, dois jogadores chamados A e B competem em uma grade de 4 x 5 células, apresentada na figura 6.11. Cada célula pode ser ocupada por um dos jogadores, que podem executar uma ação a cada jogada. As ações permitidas indicam a direção de deslocamento do agente – Norte, Sul, Leste e Oeste – ou mantém o agente imóvel.

A bola sempre acompanha um dos jogadores (ela é representada por um círculo em torno do agente na figura 6.11). Quando um jogador executa uma ação que o levaria para a célula ocupada pelo adversário, ele perde a posse da bola e permanece na mesma célula em que se encontrava. Quando o jogador executa uma ação que o levaria para fora da grade, ele permanece imóvel.

Quando o jogador que tem a posse da bola entra no gol adversário, a jogada termina e o seu time marca um ponto. Ao início de uma jogada os agentes retornam à posição inicial, mostrada na figura 6.11, e a posse da bola é determinada aleatoriamente, sendo que o jogador com a bola realiza o primeiro movimento (visto que nesta implementação as jogadas são alternadas entre os dois agentes).

Para solucionar este problema foram utilizados dois algoritmos:

- O Minimax–Q, descrito na seção 2.7.
- O *Heuristically Accelerated Minimax–Q* – HAMMQ, que implementa uma versão acelerada por heurísticas para o algoritmo Minimax–Q.

A heurística utilizada foi definida a partir de uma regra simples: se possuir a bola, dirija-se ao gol adversário (a figura 6.12 mostra as ações indicadas como boas pela heurística para o jogador A da figura 6.11). Note que ela não leva em conta a posição do adversário, deixando a tarefa de como desviar do mesmo para ser definida pelo aprendizado. O valor da heurística usada é definido a partir da equação (4.14) como sendo:

$$H(s_t, a_t, o_t) = \begin{cases} \max_a \hat{Q}(s_t, a, o_t) - \hat{Q}(s_t, a_t, o_t) + 1 & \text{se } a_t = \pi^H(s_t), \\ 0 & \text{caso contrário.} \end{cases} \quad (6.9)$$

Os parâmetros utilizados foram os mesmos para os dois algoritmos: taxa de

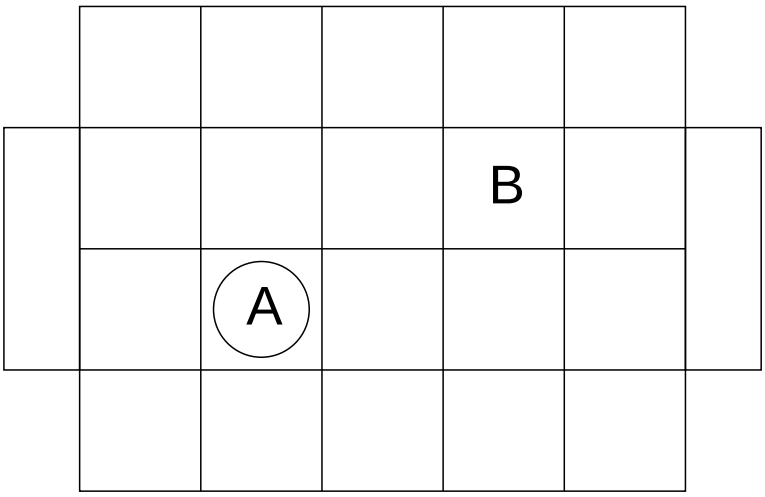


Figura 6.11: O ambiente proposto por Littman (1994). A figura mostra a posição inicial dos agentes.

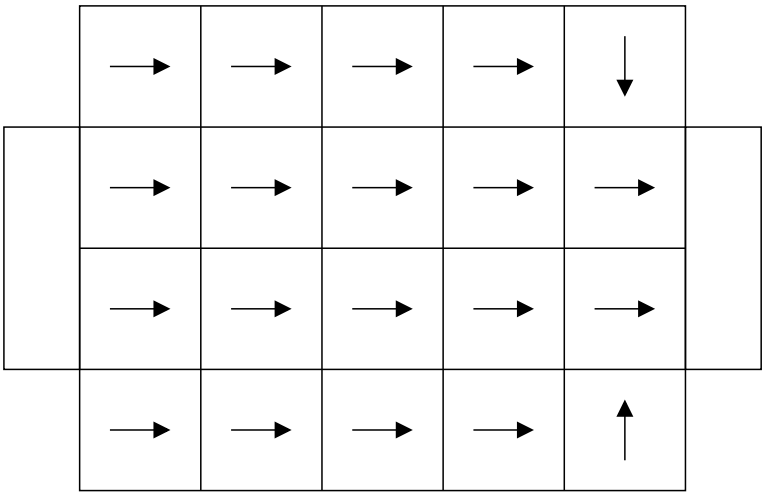


Figura 6.12: A heurística utilizada para o ambiente da figura 6.11. As setas indicam a ação a tomar.

aprendizado é iniciada com $\alpha = 1,0$ e tem um decaimento igual a $0,9999954$ por ação executada. A taxa de exploração/explotação foi definida como igual a 0,2 e o fator de desconto $\gamma = 0,9$ (estes parâmetros são iguais aos usados por Littman (1994)). O valor da heurística η usado foi igual a 1. O reforço utilizado foi 1000 ao chegar ao gol e -1000 ao ter um gol marcado pelo adversário. Os valores da tabela Q foram iniciados aleatoriamente, sendo que $0 \leq Q(s_t, a_t, o_t) \leq 1$. Os experimentos foram codificados em Linguagem C++ (compilador GNU g++) e executados em um Microcomputador AMD K6-II 500MHz, com 288MB de memória RAM e sistema operacional Linux.

Para cada algoritmo foram realizados 30 treinamentos, cada um consistindo de 500 partidas. Cada partida é composta por 10 jogadas. Uma jogada termina quando um gol é marcado por qualquer um dos agentes ou quando um determinado limite de tempo – 50 iterações – é atingido.

A figura 6.13 mostra o resultado do aprendizado para os dois algoritmos jogando contra um agente com movimentação aleatória. Ela mostra, para cada partida, a média do saldo de gols marcados pelo agente aprendiz contra o oponente. É possível verificar que, conforme esperado, o Minimax–Q apresenta um desempenho pior que o HAMMQ no início do aprendizado e que, conforme as partidas são jogadas, o desempenho dos dois algoritmos se torna semelhante.

A figura 6.14 mostra o resultado do aprendizado para os dois algoritmos jogando contra um agente que também aprende, utilizando o Minimax–Q. Neste caso é possível ver que com maior clareza o HAMMQ leva vantagem durante o início do aprendizado e que depois da 300.^{a} partida o desempenho se torna semelhante.

A figura 6.15-superior mostra o resultado do teste t de Student para o aprendizado dos algoritmos Minimax–Q e HAMMQ contra um agente com movimentação aleatória (utilizando os dados mostrados na figura 6.13). Nela é possível ver que o aprendizado acelerado é melhor que o Minimax–Q até a 150.^{a} partida, a partir da qual o resultado é semelhante, com um nível de confiança maior que 5%. A figura 6.15-inferior apresenta o mesmo resultado para o aprendizado contra agente utilizando o Minimax–Q.

Finalmente, a tabela 6.6 mostra a média do saldo de gols e do número de vitórias ao final de 500 partidas para todos os treinamentos realizados. A soma

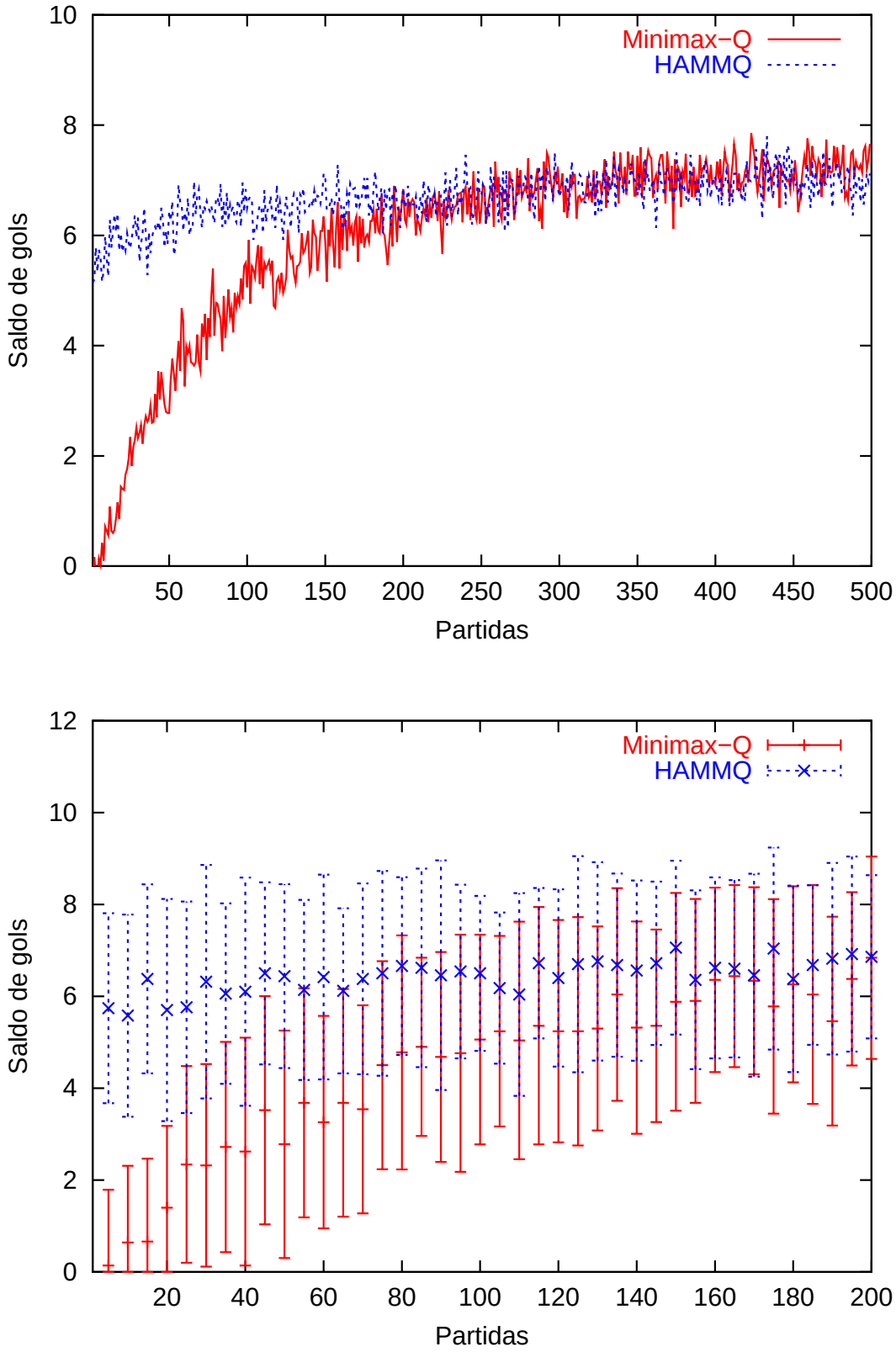


Figura 6.13: Resultados do saldo de gols para os algoritmos Minimax–Q e HAMMQ contra um jogador com movimentação aleatória para o Futebol de Robôs de Littman (com barras de erro na figura inferior).

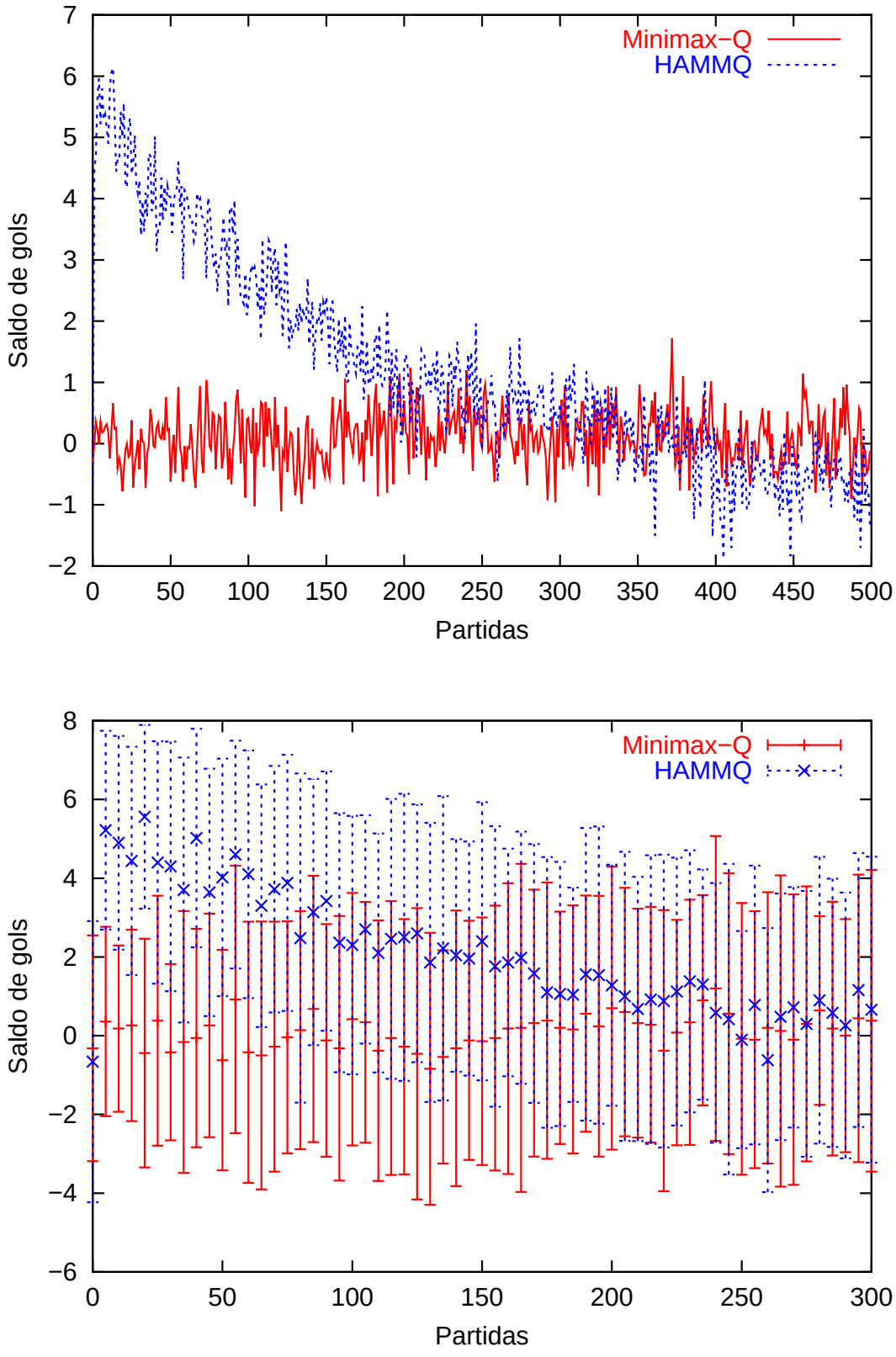


Figura 6.14: Resultados do saldo de gols para os algoritmos Minimax–Q e HAMMQ contra um agente utilizando o Minimax–Q (com barras de erro na figura inferior).

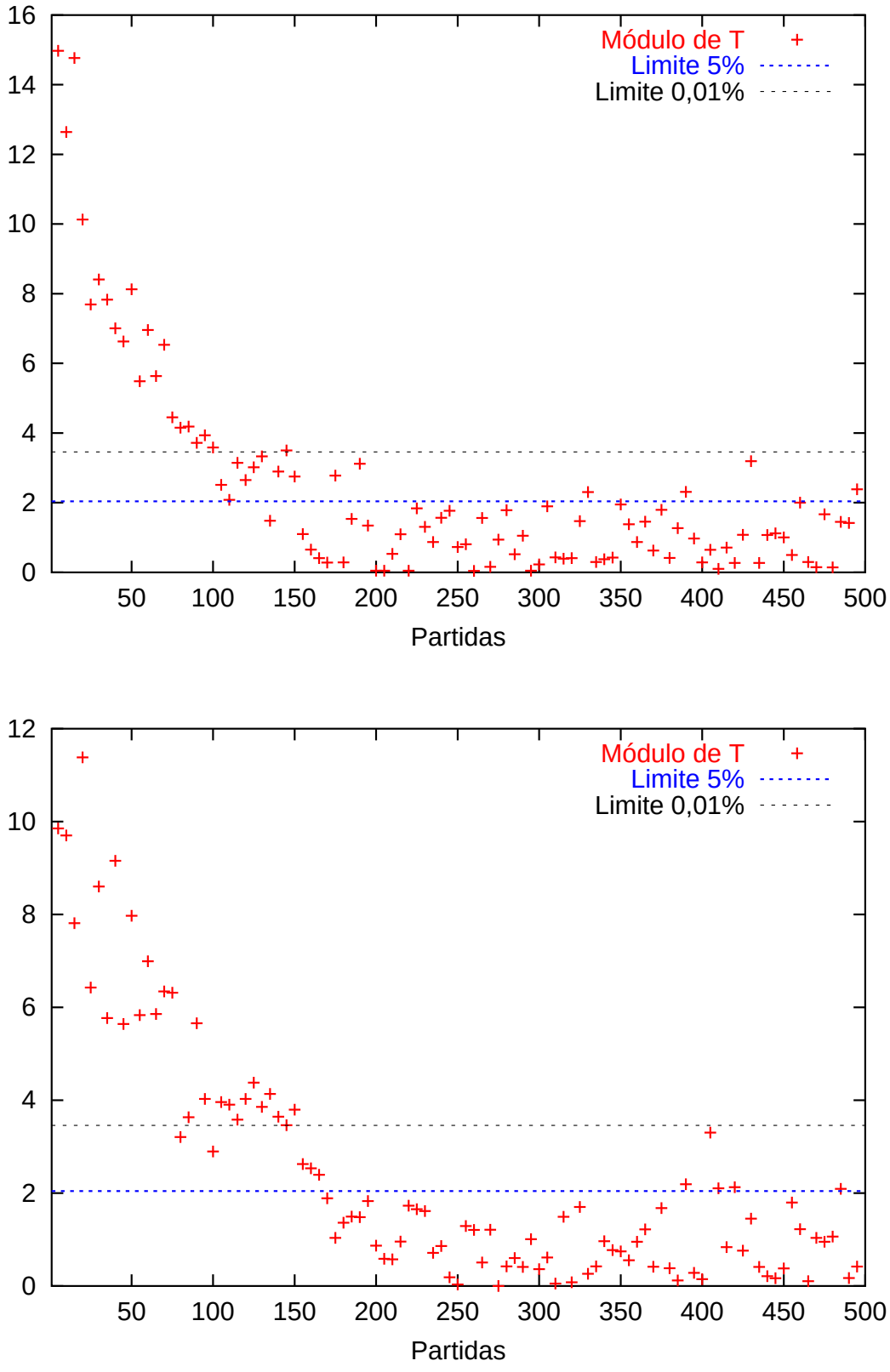


Figura 6.15: Resultados do teste t de Student entre os algoritmos Minimax–Q e HAMMQ, treinando contra um agente com movimentação aleatória (superior) e contra agente utilizando o Minimax–Q (inferior).

Partidas	Saldo de gols cumulativo
Minimax–Q × Aleatório	(3401 ± 647) × (529 ± 33)
HAMMQ × Aleatório	(3721 ± 692) × (496 ± 32)
Minimax–Q × Minimax–Q	(2127 ± 430) × (2273 ±209)
HAMMQ × Minimax–Q	(2668 ± 506) × (2099 ± 117)
Partidas	Vitórias
Minimax–Q × Aleatório	(479 ± 6) × (11 ± 4)
HAMMQ × Aleatório	(497 ± 2) × (1 ± 1)
Minimax–Q × Minimax–Q	(203 ± 42) × (218 ± 42)
HAMMQ × Minimax–Q	(274 ± 27) × (144 ± 25)

Tabela 6.6: Média do saldo cumulativo de gols e do número de vitórias para todos os jogos realizados.

do número de vitórias dos dois agentes não é igual ao total de partidas pois existem algumas onde ocorre o empate. Sobressai nesta tabela que, devido ao maior número de gols feitos pelo HAMMQ no início do aprendizado, este vence mais partidas tanto contra o agente aleatório quanto contra o agente que aprende utilizando o Minimax–Q.

6.5 Resumo

Este capítulo mostrou que abordagem baseada em heurísticas pode ser aplicada a outros algoritmos de Aprendizado por Reforço, obtendo bons resultados mesmo utilizando heurísticas simples retiradas do conhecimento *a priori* do domínio.

Outro resultado significativo é que os HALs podem ser aplicados em outros domínios, além do robôs móveis autônomos.

O próximo capítulo apresenta as conclusões deste trabalho e indica trabalhos futuros.

7 Conclusão e trabalhos futuros

Este trabalho propôs uma classe de algoritmos que permite o uso de heurísticas para aceleração do aprendizado por reforço. Os resultados obtidos indicam que esta classe de algoritmos, denominada "Aprendizado Acelerado por Heurísticas" (*Heuristically Accelerated Learning* – HAL), permite a extensão dos algoritmos existentes de aprendizado por reforço, tornando-os mais eficientes.

A principal vantagem do uso da heurística é que esta limita o espaço de busca que o agente aprendiz explora. Neste sentido, a heurística funciona, no aprendizado, de maneira similar aos algoritmos de busca informada: indica o melhor caminho a seguir, reduzindo a busca. Os HALs apresentam ainda a vantagem de, por utilizarem a heurística em conjunto com o aprendizado, poderem aprender a superar heurísticas ruins e encontrar o melhor caminho (desde que sejam mantidas as condições descritas no teorema 1). No caso do uso de heurísticas inadequadas, pode ocorrer um atraso no aprendizado (veja anexo B) ou mesmo uma aceleração, após uma piora pontual no desempenho do agente aprendiz (como mostrado na sub-seção 5.3.2).

As principais contribuições deste trabalho foram:

- A formalização da classe de algoritmos de "Aprendizado Acelerado por Heurísticas" com a introdução de uma função heurística $\mathcal{H}$ que influencia a escolha das ações e é atualizada durante o processo de aprendizado (seção 4.2).

- A proposta de métodos automáticos de extração da função heurística $\mathcal{H}$, a partir do domínio do problema ou do próprio processo de aprendizagem, chamados "Heurística a partir de X" (*"Heuristic from X"*) (seção 4.4).

Foram estudados quatro destes métodos (seção 5.2), sendo que o chamado "Heurística a partir de Exploração" produziu os melhores resultados.

- Foram propostos e estudados cinco algoritmos de aprendizado por reforço acelerados por heurísticas:
  - O algoritmo *Heuristically Accelerated Q–Learning* – HAQL, que implementa um HAL estendendo o algoritmo *Q–Learning* (seção 4.5).
  - O *Heuristically Accelerated* SARSA(λ) – HAS(λ), que implementa uma versão acelerada por heurísticas para o algoritmo SARSA(λ) (seção 6.1).
  - O *Heuristically Accelerated* TD(λ) – HATD(λ), que implementa um HAL baseado no TD(λ) (seção 6.2).
  - O *Heuristically Accelerated Distributed Q–Learning* – HADQL que implementa uma versão modificada e acelerada por heurísticas para o *Q–Learning Distribuído* (seção 6.3).
  - O *Heuristically Accelerated Minimax–Q* – HAMMQ, que implementa uma versão acelerada por heurísticas para o algoritmo Minimax–Q (seção 6.4).

  Todos os algoritmos propostos apresentaram um desempenho significativamente melhor que os algoritmos originais.

- Foram realizados testes em diversos domínios bem conhecidos na literatura de Aprendizado por Reforço, permitindo verificar a generalidade de atuação dos HALs. Foram estudados os seguintes domínios:
  - Navegação no domínio dos robôs móveis (capítulo 5).
  - O Carro na Montanha (seção 6.1).
  - O Pêndulo Invertido (seção 6.2).
  - O Problema do Caixeiro Viajante (seção 6.3).
  - O Futebol de Robôs simulado (seção 6.4).

Outras contribuições deste trabalho que devem ser citadas foram:

- A verificação do fato que muitas das propriedades dos algoritmos de AR também são válidas para os HALs (seção 4.2). Foi mostrado que as provas de convergência existentes para os algoritmos de AR correspondentes a uma forma generalizada do algoritmo *Q–Learning* continuam válidas nesta abordagem (teorema 1) e foi calculado o erro máximo causado pelo uso de uma heurística (teorema 3).

- A comparação do uso da função heurística $\mathcal{H}$ pelos HALs com o uso de heurísticas em algoritmos de busca informada (seção 4.3).

- A verificação que as informações existentes no sistema em uma situação inicial permitem definir a função heurística $\mathcal{H}$ (capítulo 5). Entre os indícios existentes no processo de aprendizagem, os mais relevantes são a função valor em um determinado instante, a política do sistema em um determinado instante e o caminho pelo espaço de estados que o agente pode explorar.

A partir dos resultados apresentados nos capítulos 5 e 6 pode-se concluir que as três hipóteses apresentadas no capítulo 4 são válidas:

- **1ª Hipótese:** Existe uma classe de algoritmos de AR chamada HAL que:
  - Usa uma função heurística para influenciar a escolha das ações.
  - Usa uma função heurística fortemente vinculada à política.
  - Atualiza a função heurística iterativamente.

- **2ª Hipótese:** Informações existentes no domínio ou em estágios iniciais do aprendizado permitem definir uma heurística que pode ser usada para acelerar o aprendizado.

- **3ª Hipótese:** Existe uma grande quantidade de métodos que podem ser usados para extrair a função heurística.

Algumas das questões levantadas na seção 1.3 foram respondidas: o uso da heurística nos HALs permite, de maneira eficiente e elegante:

- Acelerar o aprendizado por reforço (capítulos 5 e 6).

- Utilizar as informações existentes no próprio processo de aprendizado para acelerar o aprendizado (capítulo 5).
- Utilizar informações conhecidas *a priori* acerca de um problema para acelerar o aprendizado (capítulo 6).
- Combinar todas estas informações com o processo de aprendizado, sem perder as propriedades dos algoritmos de AR (seção 4.2).

Duas questões levantadas na seção 1.3 foram estudadas superficialmente e ainda exigem maior atenção. Elas são as duas primeiras tarefas que se pretende realizar como extensão deste trabalho:

1. Estudar métodos que permitam reutilizar conhecimentos aprendidos *a priori* para acelerar o aprendizado. Neste caso, o que foi aprendido em um treinamento anterior é guardado em uma base de casos construída previamente. O principal problema destes métodos é encontrar características que permitam indexar uma base de casos e que possam ser extraídas de uma situação inicial. Outro problema consiste na adaptação do caso prévio para a situação atual.
2. Estudar maneiras de compartilhar o conhecimento entre diversos agentes para acelerar o aprendizado. No momento que um agente em um SMA define uma heurística, esta pode ser informada aos outros agentes do sistema, permitindo uma melhora global do desempenho. As questões a serem estudadas envolvem problemas típicos dos sistemas multiagentes, como a credibilidade de um agente e a heterogeneidade do agentes (a heurística de um agente pode não ser boa para outro).

Além disso, outras extensões deste trabalho se mostram de grande interesse:

1. Os métodos que definem automaticamente a heurística a partir da observação da execução do sistema durante o processo de aprendizagem têm como principal problema a determinação do momento no qual as informações extraídas são suficientes para a construção de uma heurística adequada, permitindo o início da aceleração. Por este motivo, neste trabalho a aceleração foi iniciada sempre a partir de um episódio pré-definido. Um dos

principais trabalhos futuros é o estudo de maneiras de automatizar o início da aceleração, criando métodos completamente automáticos que, após permitirem a exploração do ambiente de maneira satisfatória, passam a usar a heurística para acelerar o aprendizado por reforço.

2. Realizar um estudo mais aprofundado sobre o comportamento da função heurística. Especialmente, qual a relação dela com a função de estimativa $\mathsf{F}(s_t, a_t)$ usada, o que permite inferir quais os valores de H usar para um determinado problema.

3. Estudar outros métodos "Heurística a partir de X" que usam base de casos ou não. Estudar métodos que utilizam as técnicas propostas por Foster e Dayan (2002) e por Drummond (2002) – entre outras técnicas de visão computacional – para particionar a função valor.

4. Estudar maneiras de se definir a heurística antes de atingir o estado meta, usando a aceleração por regiões do espaço de estados. Por exemplo, se um robô está em uma sala que já foi suficientemente explorada, pode-se definir imediatamente uma heurística que o leve para fora da sala.

5. Estudar o uso dos algoritmos de aprendizado por reforço acelerados por heurísticas em domínios mais complexos e dinâmicos. No domínio dos robôs móveis pode-se realizar experimentos com obstáculos que se movam, portas que abrem e fecham e outras mudanças dinâmicas no ambiente.

6. Comparar os resultados obtidos pelos algoritmos acelerados por heurísticas com outros algoritmos de aprendizado por reforço.

7. Estudar heurísticas que envolvam o tempo. Apesar de ter sido definida como dependente do tempo, nos exemplos apresentados neste trabalho o valor da heurística é definido no momento do início da aceleração e não é mais modificado. Ignorou-se casos nos quais mais informações podem ser extraídas com o passar do tempo, como o de um robô que explora partes diferentes de um ambiente em cada episódio.

8. Implementar outros algoritmos de aprendizado por reforço acelerados por heurísticas. O uso da heurísticas parece promissor em algoritmos como o QS (RIBEIRO, 1998) e o Minimax–QS (PEGORARO; COSTA; RIBEIRO, 2001).

Anexo A – Estudo da convergência da política e da função valor

Este anexo apresenta uma experiência realizada com o objetivo de verificar se a política é um melhor indicativo para a heurística que a função valor. Isto significa que poderia-se extrair uma heurística – a ser usada para acelerar o aprendizado – melhor e mais cedo a partir da política do que a partir da função valor.

Para tanto, foi realizada, para o domínio descrito na figura 5.2, uma comparação da evolução de dois valores: o valor da função valor e o da função valor calculada a partir da política, utilizando Programação Dinâmica – PD.

Ao final de cada episódio (isto é, ao atingir o alvo), foram calculadas as seguintes diferenças:

- $\Delta V = \sum_{s_t \in S} |V_t(s_t) - V^*(s_t)|$, isto é, a diferença da função valor, que foi definida como a soma dos módulos das diferenças entre o $V(s_t)$ de cada estado e o $V^*(s_t)$.
- $\Delta V^\pi = \sum_{s_t \in S} |V_t^\pi(s_t) - V^{\pi^*}(s_t)|$ é similar ao primeiro item, sendo que a função valor é calculada a partir da política utilizando a equação (2.5):

$$V^\pi(s_t) = r(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} T(s_t, \pi(s_t), s_{t+1}) V^\pi(s_{t+1}).$$

Os resultados normalizados são mostrados na figura A.1. Pode-se ver que o valor calculado a partir da política por PD converge antes da função valor (que converge próximo ao 200.000.^{o} episódio de treinamento). Isto indica que a política em um estado é um bom atributo para se utilizar para definição de uma política heurística.

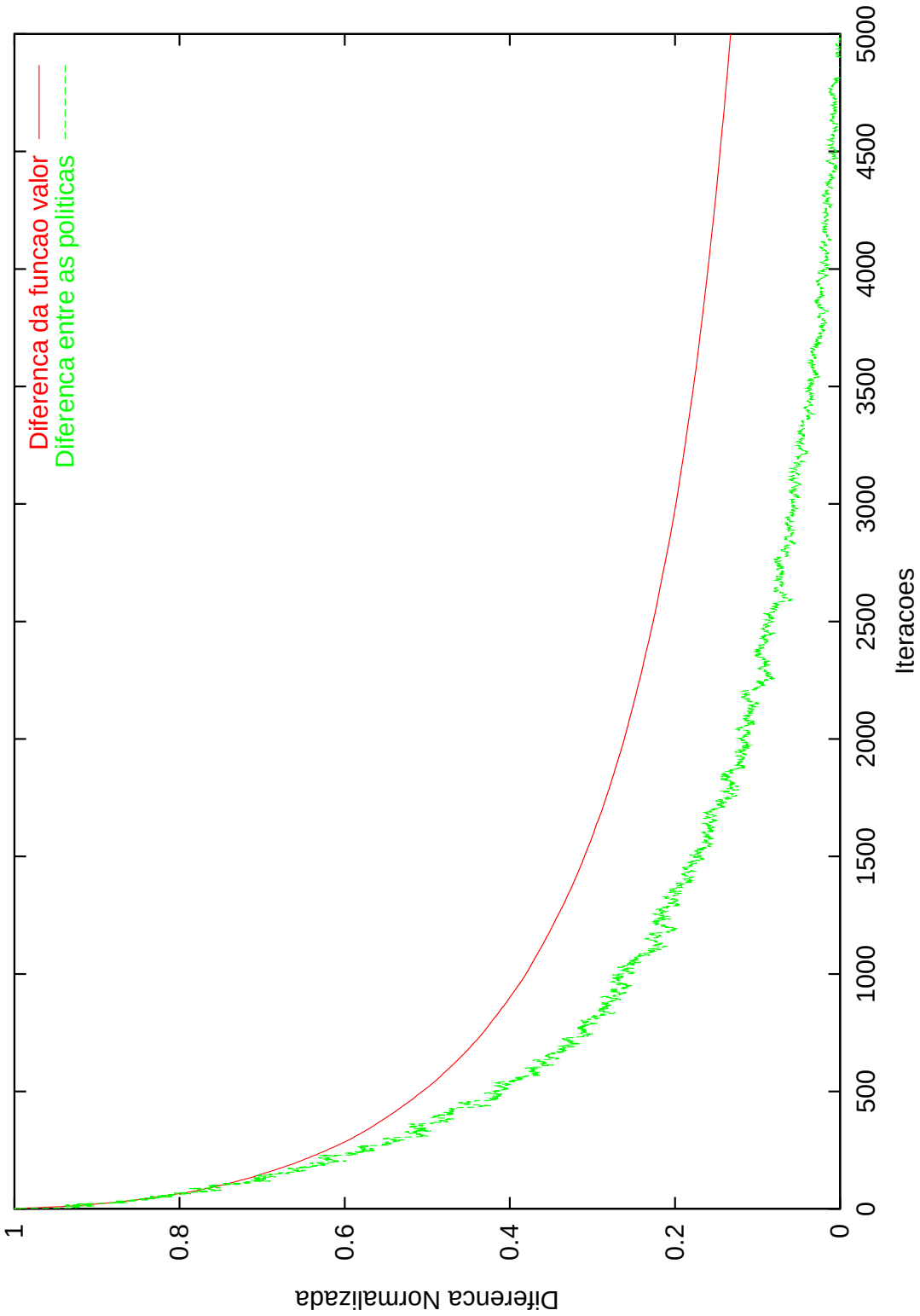


Figura A.1: Evolução da diferença da função valor e do valor calculado a partir da política utilizando Programação Dinâmica. Gráfico Normalizado.

Finalmente, comprovou-se que a política mais provável de uma determinada partição do ambiente converge rapidamente. A figura A.2 mostra o valor da política mais provável para a sala ao lado direito da figura 5.2. O eixo x mostra o episódio de treinamento e o y, a ação tomada (uma entre N, S, E, W). Pode-se ver que a convergência (para a ação N) ocorre rapidamente. Isto indica que uma estratégia que utilize a estimativa da política mais provável pode gerar boas heurísticas.

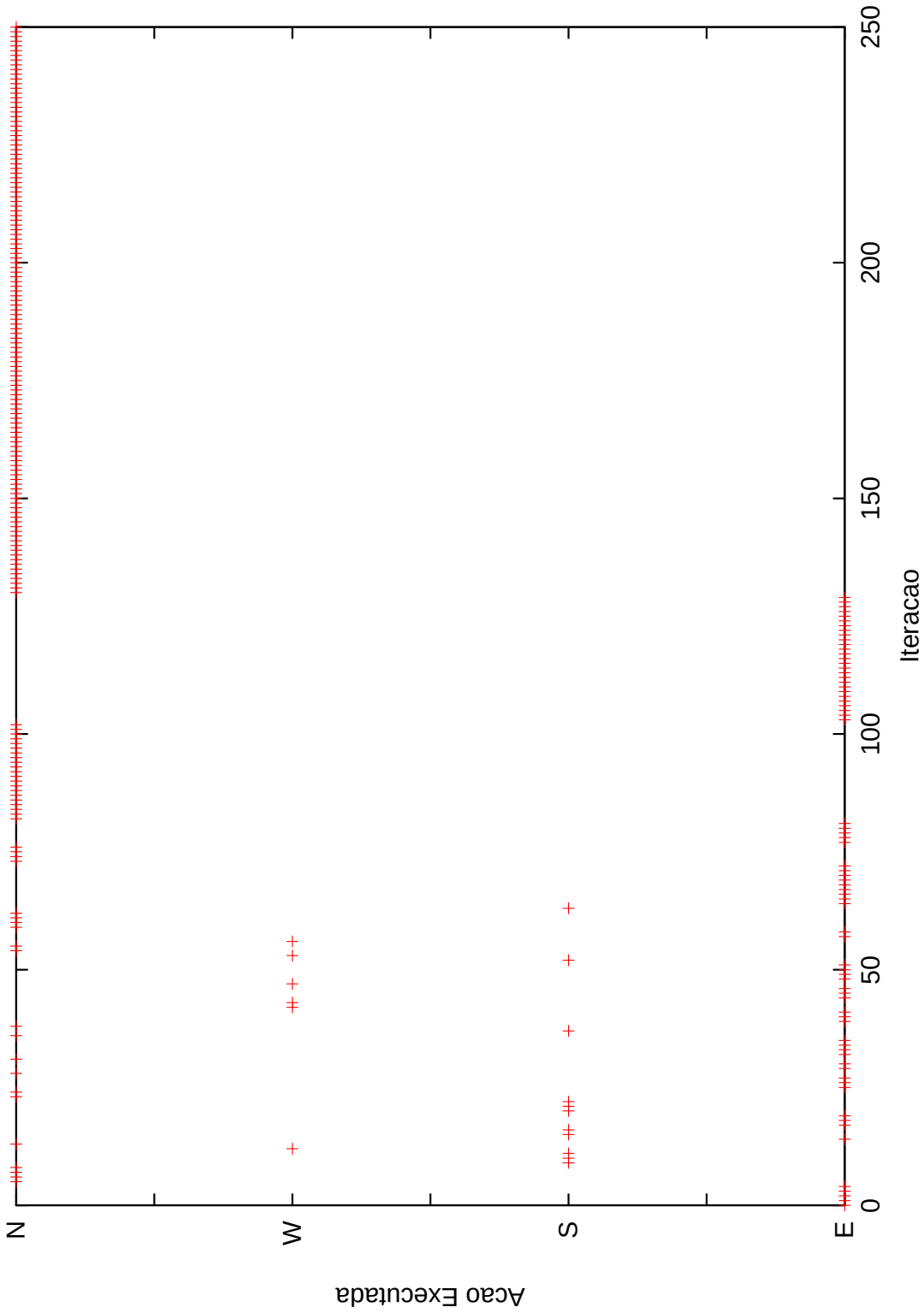


Figura A.2: A convergência da política mais provável para a sala ao lado direito da figura 5.2. O eixo x mostra o episódio de treinamento e o y, a ação tomada (N, S, E, W).

Anexo B – Estudo da relação entre a qualidade da heurística e a evolução do aprendizado

Este anexo tem por objetivo mostrar que mesmo o uso de uma heurística muito ruim não prejudica o funcionamento dos HAQL.

Neste estudo, a heurística é definida como:

$$H(s_t, a_t) = \begin{cases} 1 & \text{se } a_t = \pi_t^*, \\ 0 & \text{caso contrário.} \end{cases} \tag{B.1}$$

A figura B.1 mostra o resultado do uso da aceleração ao final do centésimo episódio de treinamento para o HAQL, descrito na seção 5.3, utilizando uma heurística que define a política ótima. Nota-se que, após a aceleração, o aprendizado converge imediatamente para a política ótima. Uma característica interessante é que, utilizando estes valores para a função heurística (1 se $a_t = \pi_t^*$ e zero caso contrário), mesmo que seja usada uma política errada para acelerar a convergência, é encontrada uma solução para o problema, com o aprendizado corrigindo esta informação errada. A figura B.2 apresenta o resultado do uso da aceleração com uma política errada, definida como a heurística totalmente contrária à política ótima (caso a ótima indique ir para a esquerda, a heurística usada indica ir para a direita, e assim por diante).

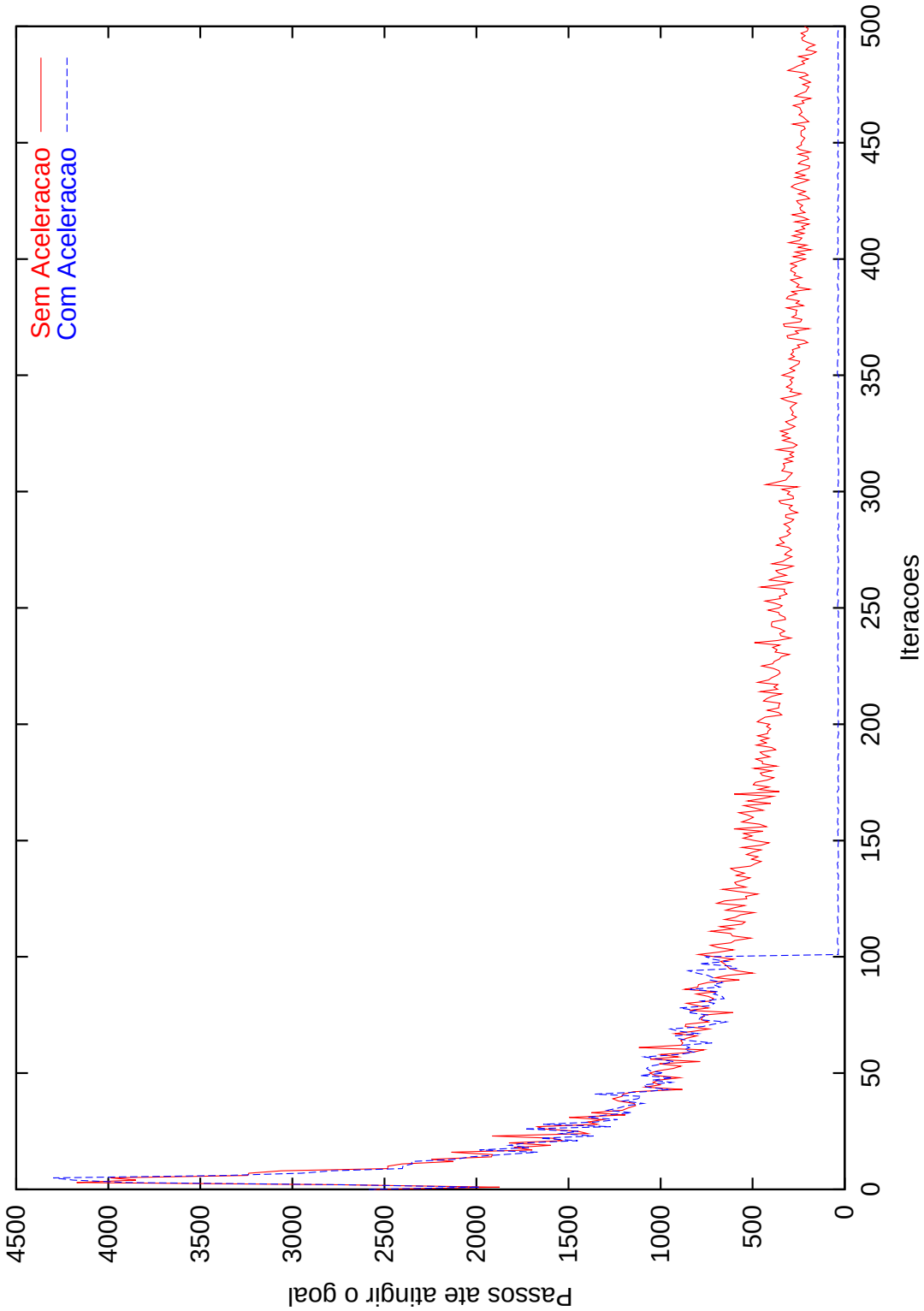


Figura B.1: Resultado do uso da aceleração com a heurística correta no centésimo episódio usando o algoritmo HAQL.

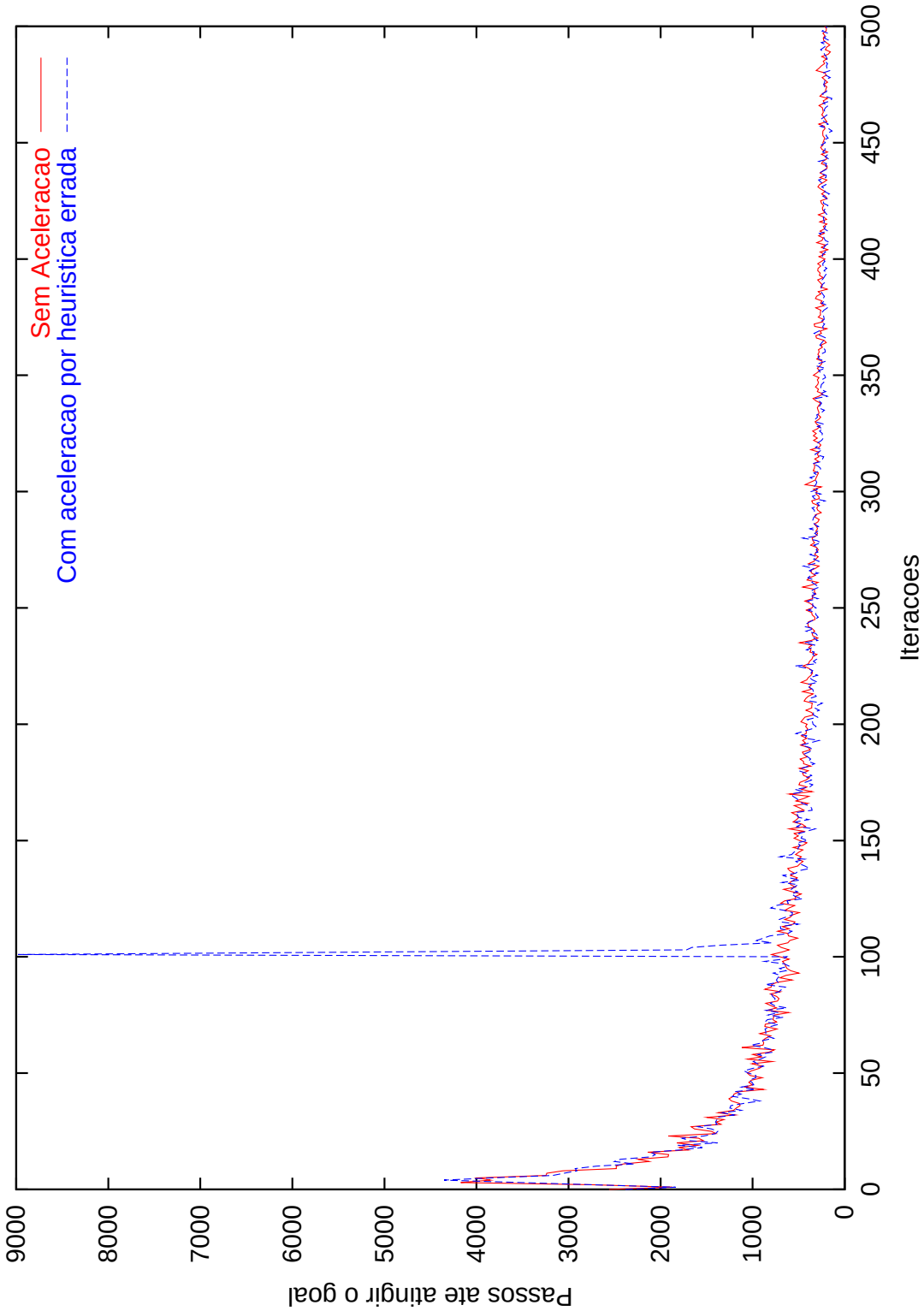


Figura B.2: Resultado do uso da aceleração com uma heurística errada no centésimo episódio usando o algoritmo HAQL.

Anexo C – Evolução das visitas em um ambiente de grandes dimensões

Como os testes realizados nas sub-seções 5.3.1 e 5.3.3 convergiram quase imediatamente para a solução utilizando o algoritmo HAQL, optou-se por mais um teste, em um ambiente de grandes dimensões (570 linhas por 800 colunas) e maior complexidade, que permitisse avaliar a evolução da visitação do agente robótico a novos estados. O ambiente escolhido, mostrado na figura C.1, corresponde ao 2.^{o} andar da mansão de Bailicroft (SUMMERS, 2001), onde a área em preto corresponde às paredes e a em branco, ao espaço por onde o robô pode se mover. A meta encontra-se no canto inferior direito da sala inferior direita (marcado com um 'G' na figura C.1).

A figura C.2 mostra a quantidade de visitas a novas posições (aquelas que não tinham sido visitadas anteriormente) ao final de cada episódio de treinamento (média para 30 treinamentos) e a figura C.3 mostra os estados visitados ao final de um episódio (branco indica posições visitadas) e o número de passo executados, para uma determinada seqüência de treinamento.

Estes resultados mostram que a exploração é realizada quase completamente no início da execução, e que a partir de um determinado momento (o 15.^{o} episódio), a visita a novos estados é mínima. A partir destes resultados, pode-se definir o momento ideal para iniciar a aceleração: o momento onde a quantidade de novos estados visitados em um episódio decair para próximo de zero.

Finalmente, o resultado da uso do HAQL com aceleração ao final do vigésimo episódio pode ser visto na figura C.4. Foi realizado apenas o experimento de navegação em um ambiente desconhecido conforme descrito na sub-seção 5.3.1, pois o reposicionamento da meta também é solucionado em apenas um episódio. O teste t de Student para este experimento (figura C.5) mostra que a partir da 20.^{a}

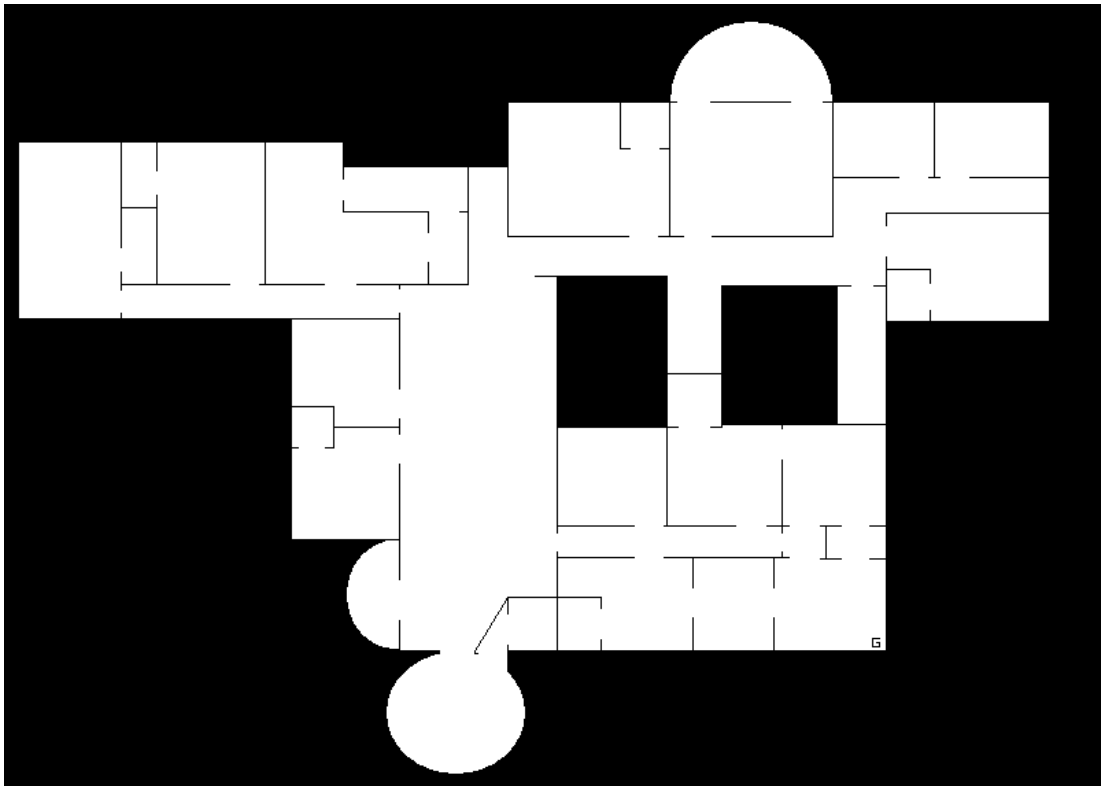


Figura C.1: Planta do 2.º andar da mansão de Bailicroft (SUMMERS, 2001), onde a área em preto corresponde às paredes e a em branco, ao espaço por onde o robô pode se mover.

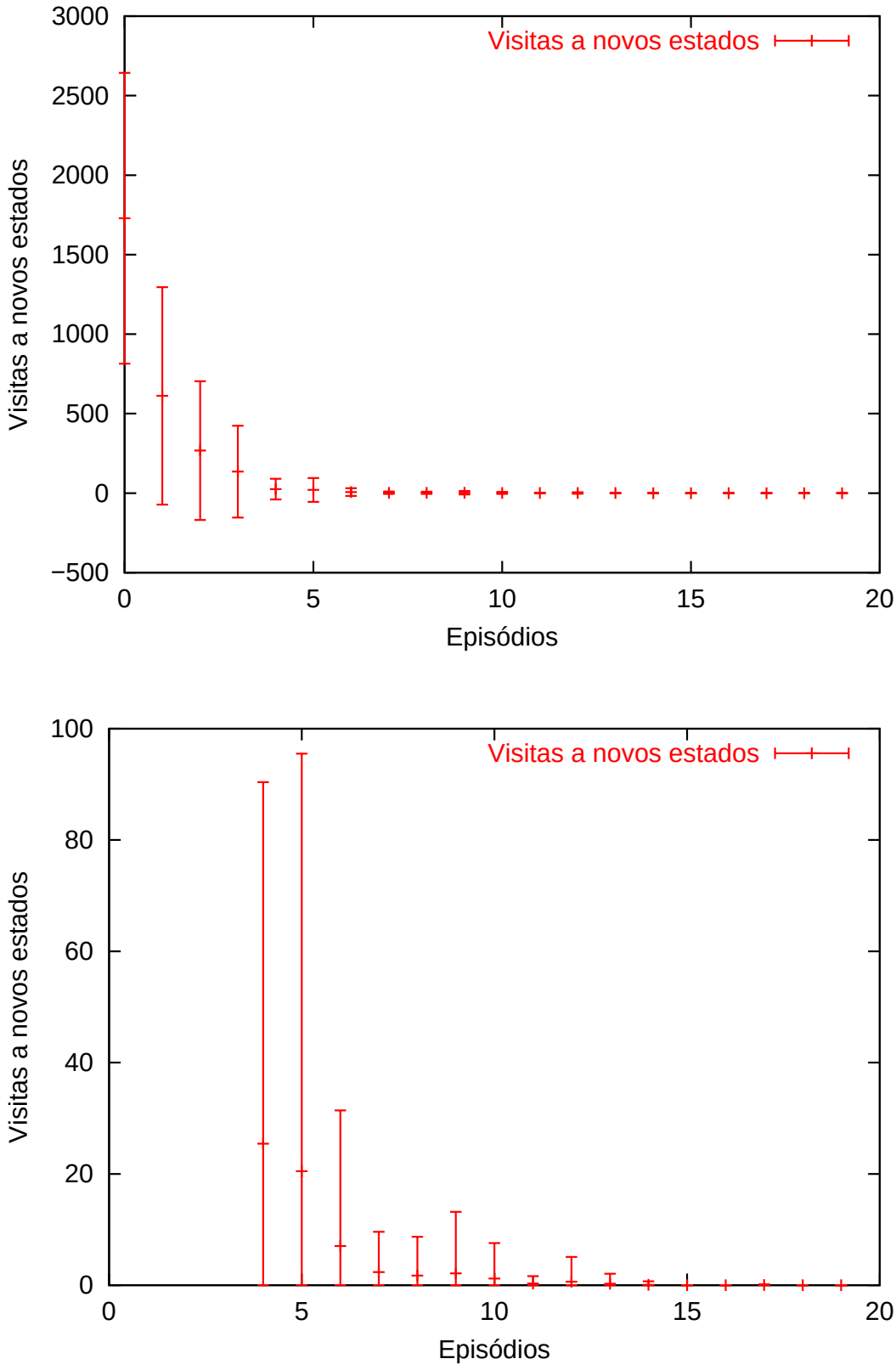
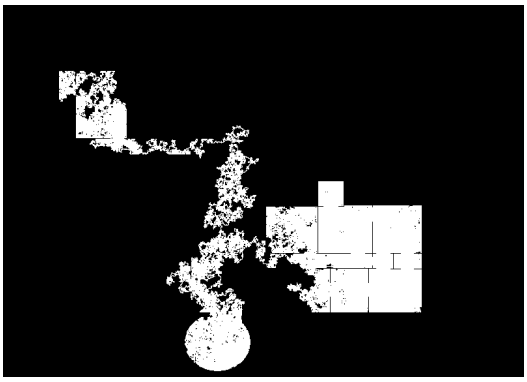
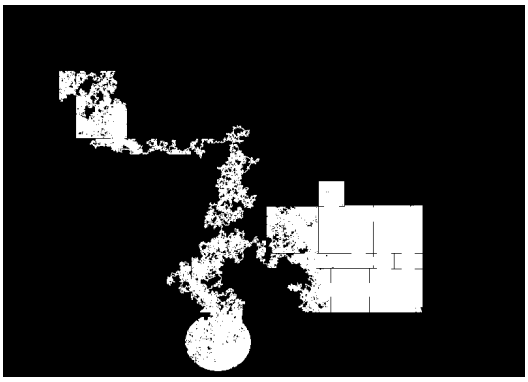


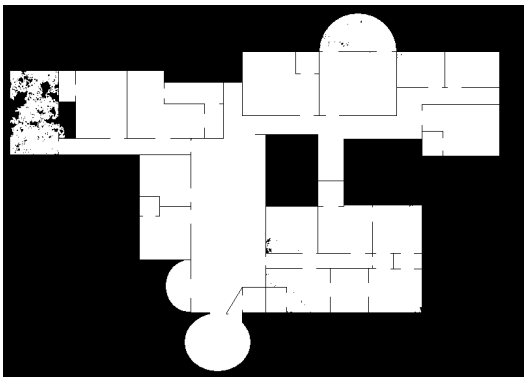
Figura C.2: Visitas a novos estados (ampliado na figura inferior).



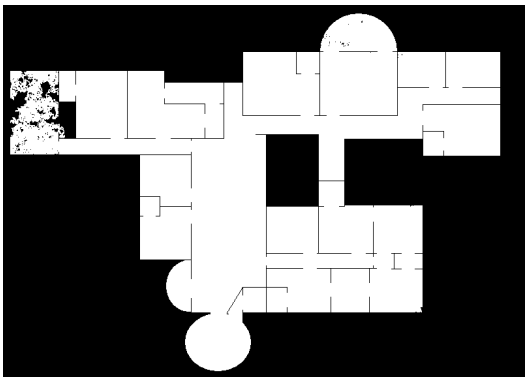
(a) 1.^{o} episódio – 876.887 passos.



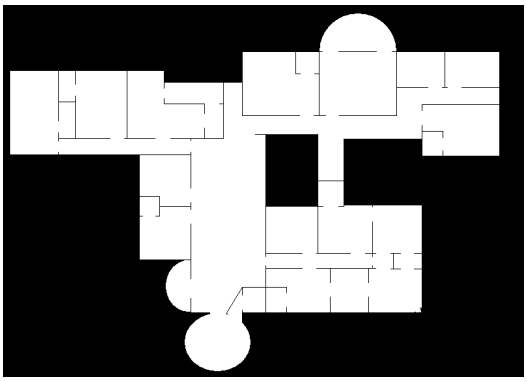
(b) 2.^{o} episódio – 5.163.329 passos.



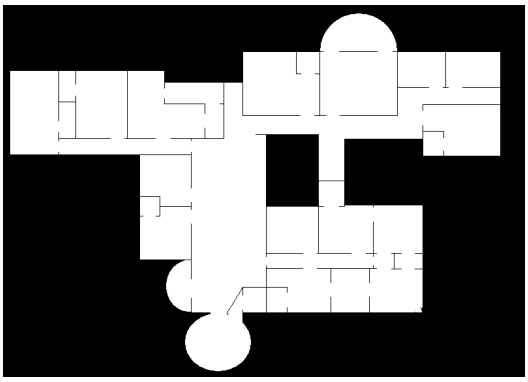
(c) 3.^{o} episódio – 5.220.071 passos.



(d) 4.^{o} episódio – 4.743.443 passos.



(e) 5.^{o} episódio – 1.350.375 passos.



(f) 6.^{o} episódio – 565.181 passos.

Figura C.3: Mapa criado a partir da exploração.

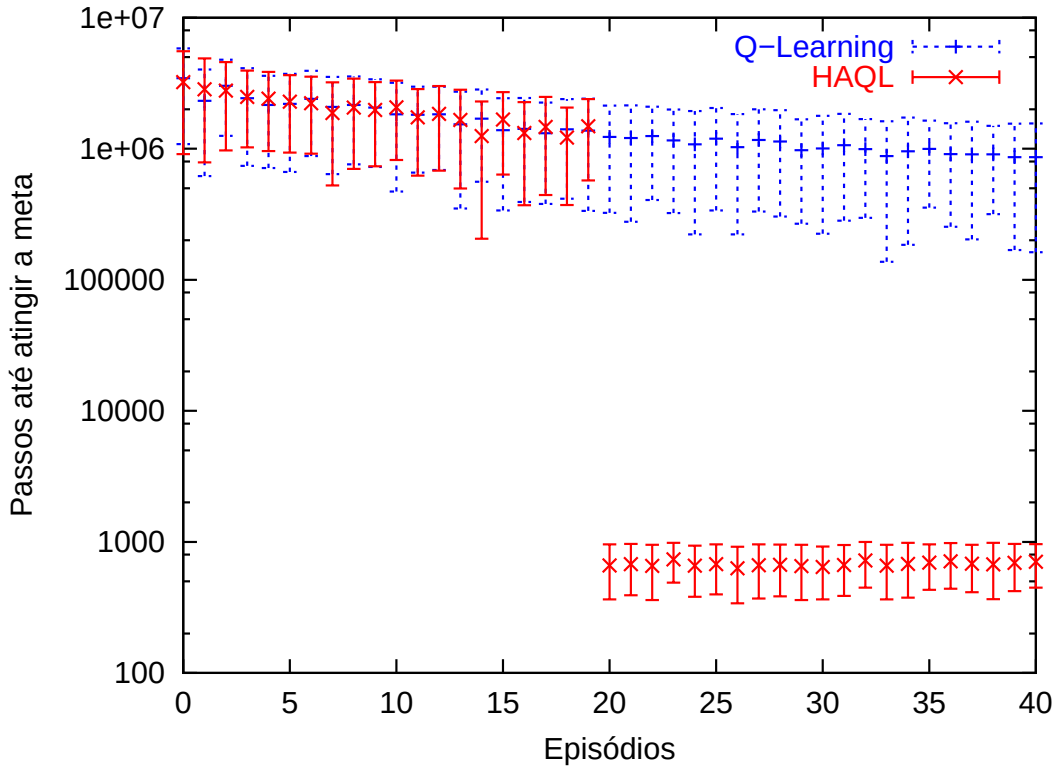


Figura C.4: Resultado para a aceleração no 20.º episódio utilizando "Heurística a partir de Exploração" com barras de erro (em monolog), para o ambiente Bailicroft.

iteração os resultados são significativamente diferentes, com nível de confiança maior que 0,01%.

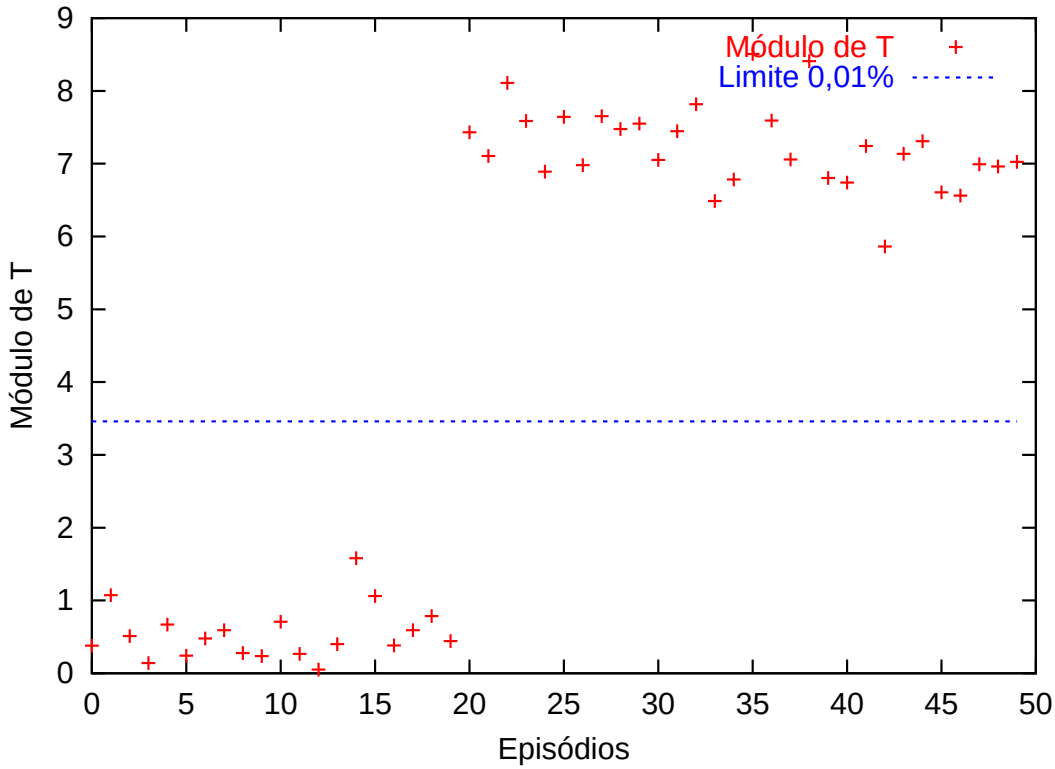


Figura C.5: Resultado do teste t de Student para a aceleração no 20.º episódio no ambiente Bailicroft.

Anexo D – Evolução da função valor

Uma preocupação que surgiu durante este trabalho foi como a utilização da heurística modifica a evolução da função valor. Assim, utilizando a mesma configuração das experiências descritas na seção 5.3, foi calculada a raiz quadrada do erro quadrático médio (*Root Mean Squared Error* – RMSE)[1] entre a função valor encontrada pelo HAQL e a função valor ótima V^* (computada *a priori*) a cada passo do treinamento. Este erro foi calculado para o HAQL com aceleração ao final do décimo episódio e também para o *Q–Learning* sem aceleração.

Pode-se ver na figura D.1 que o erro RMSE decai mais rapidamente para o aprendizado sem a aceleração. Como, após a aceleração, o HAQL executa menos passos a cada episódio, possibilitando menos atualização dos valores de V, esta figura apresenta o aprendizado em relação ao número total de passos executados pelo agente para possibilitar uma comparação entre a quantidade de atualizações da função valor-ação Q.

Esta diferença é resultado da menor exploração realizada pelo HAQL. Ao utilizar a aceleração, o agente demora a receber os reforços negativos, pois ele executa um menor número de vezes as ações que levam às posições ruins (como a uma parede).

A figura D.2 mostra as funções valores gerada pelo *Q–Learning* e pelo HAQL ao final do passo 20×10^8. Pode-se ver que na função valor gerada pelo HAQL, a partir de uma determinada distância da meta, o valor começa a aumentar novamente pois as posições estão muito distante da meta (o valor da recompensa e o da constante γ não permite que o reforço ganho ao final do episódio influencie o valor de V nesses estados) e por executarem um menor número de ações.

Existe uma relação entre o valor do reforço ganho ao final do episódio e

[1] Definição dos tipos de erros pode ser encontrada no apêndice I.

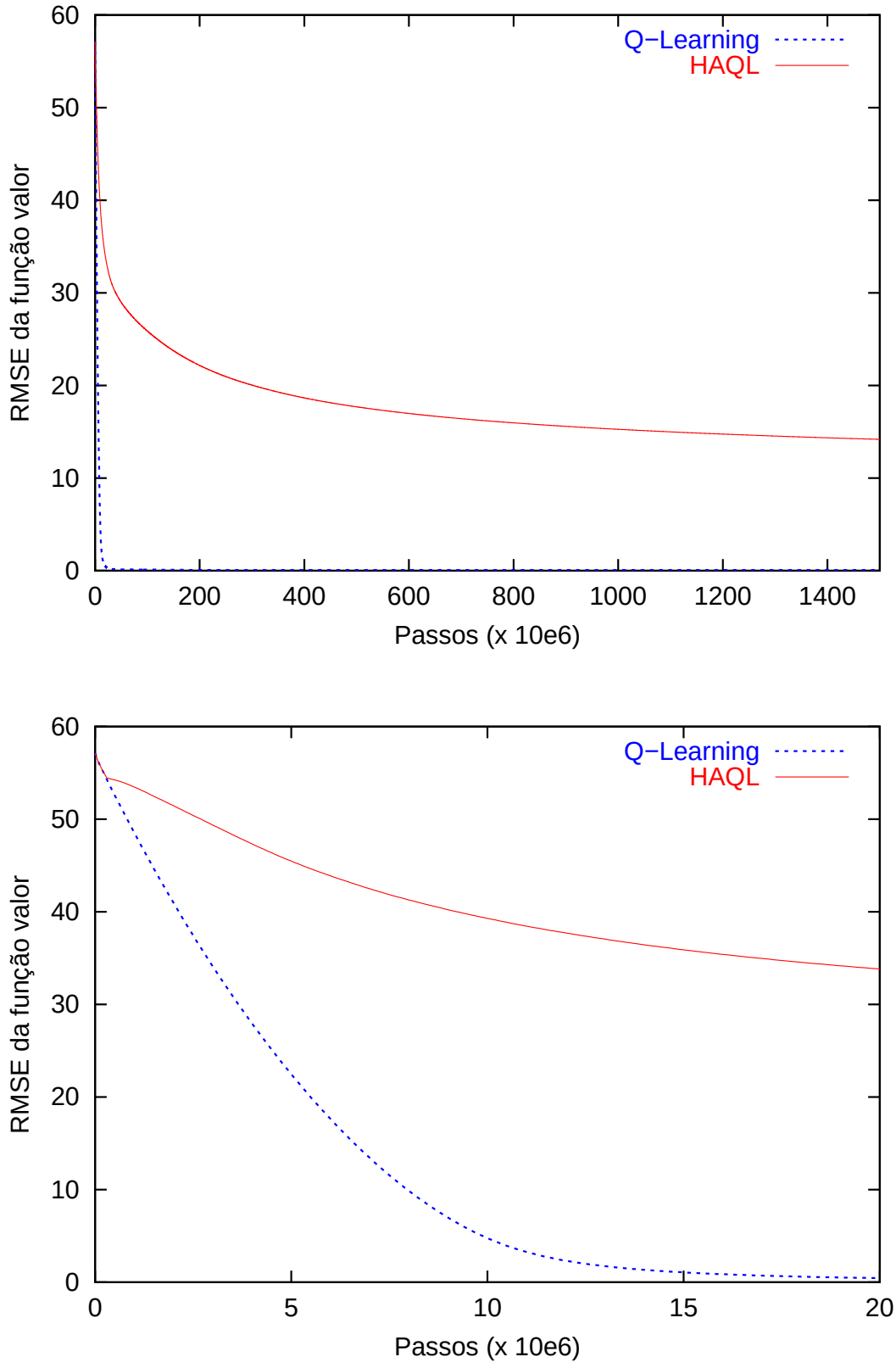


Figura D.1: Evolução da raiz do erro quadrático médio (RMSE) entre a função valor do algoritmo (*Q–Learning* ou HAQL) e o valor V^*, em relação ao total de passos (ampliado na figura inferior).

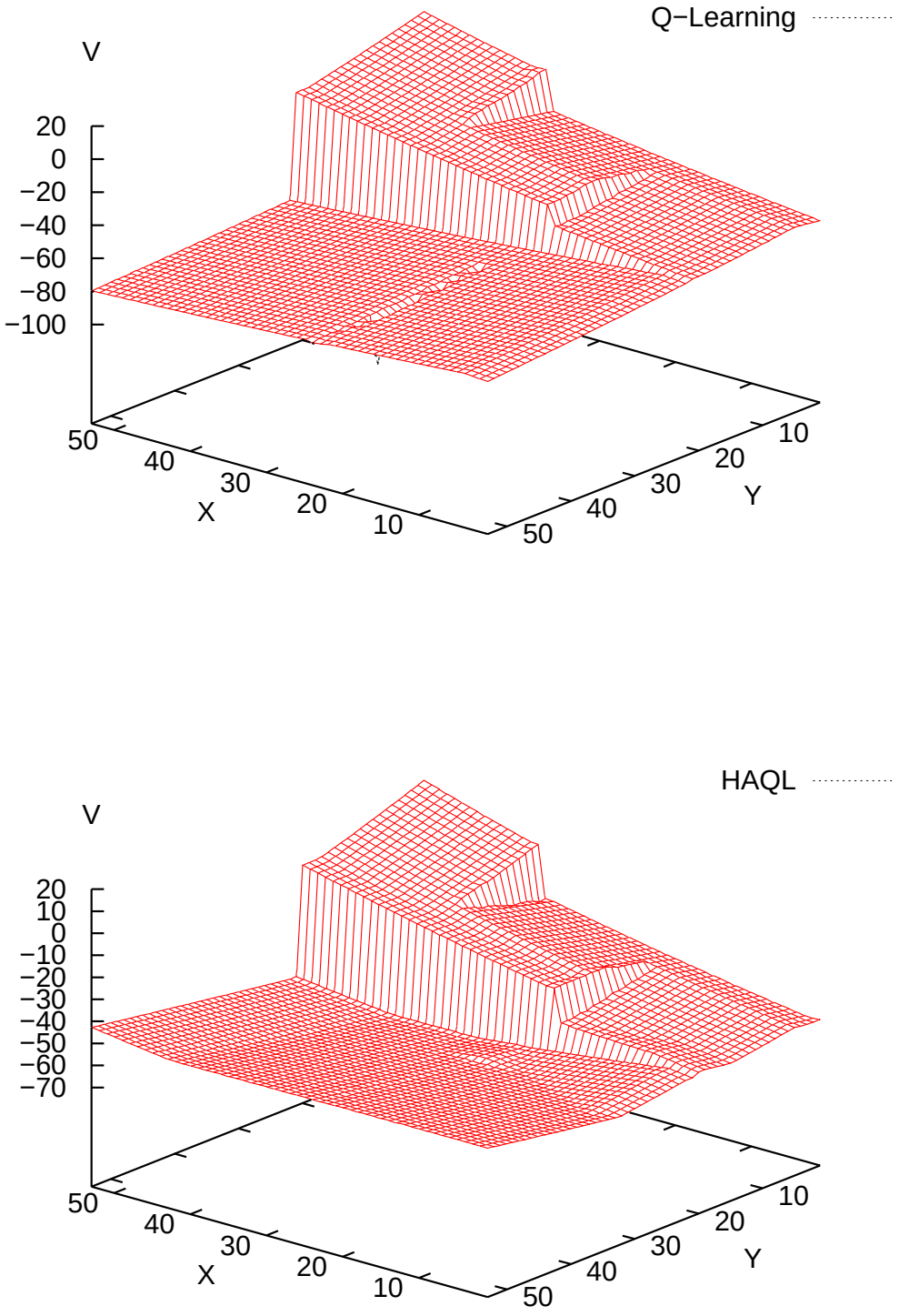


Figura D.2: Função valor gerada pelo *Q–Learning* (superior) e pelo HAQL (inferior) ao final do 20×10^8 passo, utilizando o reforço positivo igual a 10 ao atingir a meta.

Reforço	Raiz do Erro Quadrático Médio	Erro Percentual Médio
10	13.5	8.46
25	11.7	5.85
50	8.3	3.28
75	5.6	1.49
100	3.7	2.14
1000	7.1	0.26

Tabela D.1: Erros entre a função valor gerada pelo HAQL e a função valor ótima V^*, em relação ao reforço recebido ao atingir a meta, ao final do 20×10^8 passo.

as diferenças na evolução da função valor, apresentada na figura D.3. Pode-se verificar que, quanto maior a recompensa recebida ao encontrar a meta, menor a diferença entre as funções valores de Q e de HAQL. Finalmente, a tabela D.1 mostra o erro RMSE e o erro percentual médio (*Mean Percentual Error* – MPE) entre as funções valores geradas pelo *Q–Learning* e o HAQL ao final do passo 20×10^8.

Finalmente, apesar da função valor convergir mais lentamente para o V^* com o uso da heurística (pela restrição na exploração do espaço de estados imposta pela heurística), vale lembrar que a política encontrada no processo de aprendizagem rapidamente se aproxima do adequado.

Caso seja de interesse, além da política ótima para um determinado problema, a função valor V^*, pode-se utilizar um algoritmo que após encontrar a política ótima π^* utilizando o HAQL, calcula V^* a partir da equação (2.1) de maneira similar à retropropagação da política – executando uma propagação do valor cumulativo descontado de γ. Este método é uma maneira interessante de unir Programação Dinâmica e AR, utilizando PD para calcular a política e o custo rapidamente depois da exploração inicial, e o Aprendizado por Reforço para dar conta das eventuais modificações que ocorram em um ambiente dinâmico.

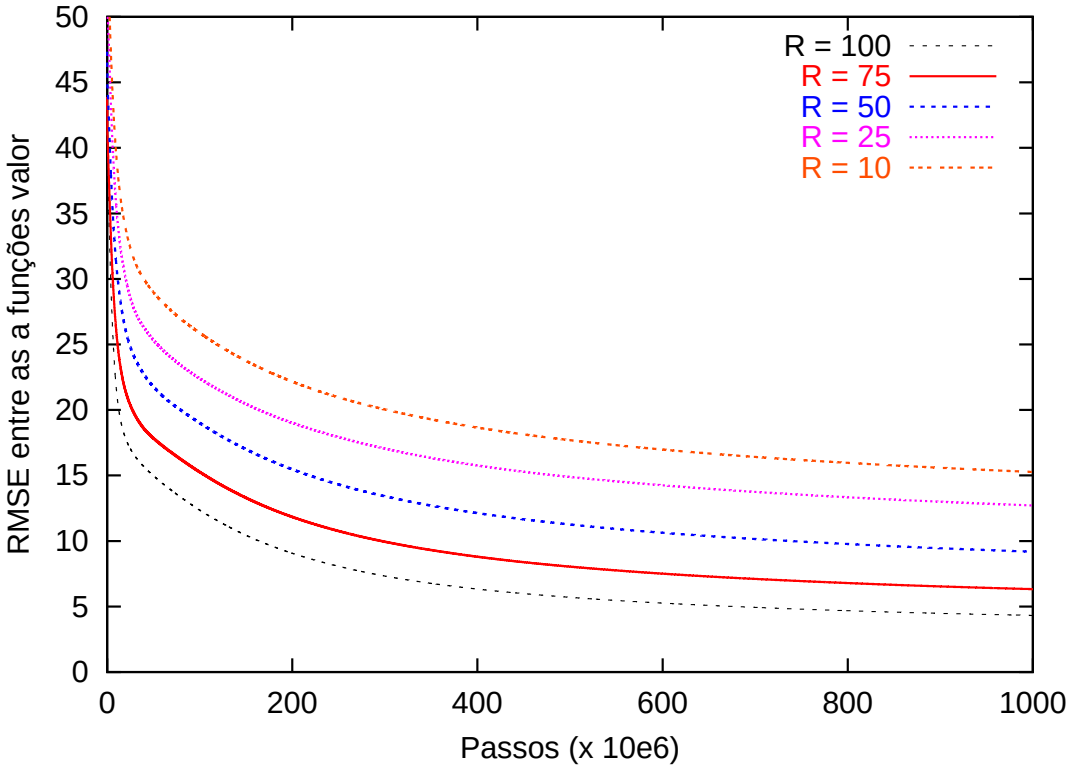


Figura D.3: Evolução da raiz do erro quadrático médio (RMSE) entre a função valor do HAQL e o V^*, para diversos valores de recompensa recebida ao atingir a meta no final de um episódio.

Referências Bibliográficas

ACTIVMEDIA ROBOTICS. *Saphira's Manual*. Menlo Park, CA, 2001. Version 8.0a.

ALBUS, J. S. Data storage in the cerebellar model articulation controller. *Trans. of the ASME, J. Dynamic Systems, Measurement, and Control*, v. 97, n. 3, p. 228–233, 1975.

ALBUS, J. S. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Trans. of the ASME, J. Dynamic Systems, Measurement, and Control*, v. 97, n. 3, p. 220–227, 1975.

ALVARES, L. O.; SICHMAN, J. S. Introdução aos sistemas multiagentes. In: MEDEIROS, C. M. B. (Ed.). *XVI Jornada de Atualização em Informática / XVII Congresso da Sociedade Brasileira de Computação*. Brasília: Universidade de Brasília - UnB, 1997. p. 1–37.

BANERJEE, B.; SEN, S.; PENG, J. Fast concurrent reinforcement learners. In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence - IJCAI'01*. Seattle, Washington: IJCAI, 2001. v. 2, p. 825–830.

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, n. 13, p. 834–846, 1983.

BERTSEKAS, D. P. *Dynamic Programming: Deterministic and Stochastic Models*. Upper Saddle River, NJ: Prentice-Hall, 1987.

BIANCHI, R. A. C. *Uma Arquitetura de Controle Distribuída para um Sistema de Visão Computacional Propositada*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 1998. Mestrado em Engenharia Elétrica.

BIANCHI, R. A. C.; COSTA, A. H. R. Comparing distributed reinforcement learning approaches to learn agent coordination. In: *Advances in AI: 8th. Ibero–American Conference on AI – IBERAMIA'2002*. Berlin, Heidelberg: Springer Verlag, 2002. (Lecture Notes in Artificial Intelligence), p. 575–584.

BIANCHI, R. A. C.; COSTA, A. H. R. Ant–Vibra: a swarm intelligence approach to learn task coordination. In: *Advances in AI: 16th. Brazilian Symposium on AI – SBIA'2002*. Berlin, Heidelberg: Springer Verlag, 2002. (Lecture Notes in Artificial Intelligence), p. 195–204.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press, 1999.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. Inspiration for optimization from social insect behaviour. *Nature 406 [6791]*, 2000.

BOWLING, M. H.; VELOSO, M. M. Rational and convergent learning in stochastic games. In: *IJCAI*. [S.l.: s.n.], 2001. p. 1021–1026.

BRAGA, A. P. S. *Um agente autônomo baseado em aprendizagem por reforço direcionado à meta*. Dissertação (Mestrado) — Escola de Engenharia de São Carlos – Universidade de São Paulo, 1998. Mestrado em Engenharia Elétrica.

BROOKS, R. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, v. 1, p. 1–10, 1986.

BROOKS, R. Prospects for human level intelligence for humanoid robots. In: *Proceedings of the First International Symposium on Humanoid Robots (HURO96)*. Tokyo, Japan: [s.n.], 1996.

BROOKS, R. A. Intelligence without representation. *Artificial Intelligence*, v. 47, p. 139–159, 1991.

COSTA, A. H. R. *Robótica Móvel Inteligente: progressos e desafios*. São Paulo: Escola Politécnica da Universidade de São Paulo, 2003. Tese de Livre Docência em Engenharia Elétrica.

COSTA, A. H. R.; BARROS, L. N.; BIANCHI, R. A. C. Integrating purposive vision with deliberative and reactive planning: An engineering support on robotics applications. *Journal of the Brazilian Computer Society*, v. 4, n. 3, p. 52–60, April 1998.

COSTA, A. H. R.; BIANCHI, R. A. C. L–vibra: Learning in the vibra architecture. *Lecture Notes in Artificial Intelligence*, v. 1952, p. 280–289, 2000.

DORIGO, M. Ant algorithms and swarm intelligence. *Proceedings of the Seventeen International Joint Conference on Artificial Intelligence, Tutorial MP-1*, 2001.

DORIGO, M.; CARO, G. D.; GAMBARDELLA, L. M. Ant algorithms for discrete optimization. *Artificial Life*, v. 5, n. 3, p. 137–172, 1999.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, 1997.

DRUMMOND, C. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, v. 16, p. 59–104, 2002.

FAUGERAS, O. *Three-Dimensional Computer Vision – A Geometric ViewPoint*. Cambridge, MA: MIT Press, 1987.

FISHER, R. *On-Line Compendium of Computer Vision*. [s.n.], 2003. Disponível em: <http://www.dai.ed.ac.uk/CVonline/>.

FOSTER, D.; DAYAN, P. Structure in the space of value functions. *Machine Learning*, v. 49, n. 2/3, p. 325–346, 2002.

FOX, D.; BURGARD, W.; THRUN, S. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, n. 11, p. 391–427, 1999.

GABRIELLI, L. H.; RIBEIRO, C. H. C. Generalização cmac para aprendizagem por reforço em times de robôs. In: *6o. Simpósio Brasileiro de Automação Inteligente – 6o. SBAI*. Baurú, SP: SBA, 2003. p. 434–438.

GAMBARDELLA, L.; DORIGO, M. Ant–Q: A reinforcement learning approach to the traveling salesman problem. *Proceedings of the ML-95 – Twelfth International Conference on Machine Learning*, p. 252–260, 1995.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, v. 4, n. 2, p. 100–107, 1968.

HAYKIN, S. *Neural Networks - A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice-Hall, 1999.

HEGER, M. Consideration of risk in reinforcement learning. In: *Proceedings of the Eleventh International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, 1994. p. 105–111.

JOHN, G. H. When the best move isn't optimal: Q–learning with exploration. In: *Procs. of the 12th National Conference on Artificial Intelligence*. Seattle, WA: AAAI Press/MIT Press, 1994.

JOSHI, S.; SCHANK, J. Of rats and robots: a new biorobotics study of social behavior in norway rat pups. In: *2nd International Workshop on the Mathematics and Algorithms of Social Insects*. Atlanta, Georgia: [s.n.], 2003.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, v. 4, p. 237–285, 1996.

KASS, M.; WITKIN, A.; TERZOPOULOS, D. Snake: active contour model. *Int. J. Computer Vision*, v. 1, p. 321–331, 1987.

KERVRANN, C.; TRUBUIL, A. Optimal level curves and global minimizers of cost functionals in image segmentation. *Journal of Mathematical Imaging and Vision*, v. 17, n. 2, p. 157–174, 2002.

KHOUSSAINOV, R. Playing the web search game. *IEEE Intelligent Systems*, v. 18, 2003.

KHOUSSAINOV, R.; KUSHMERICK, N. Automated index management for distributed Web search. In: *Proceedings of the Eleventh ACM International Conference on Information and Knowledge Management (CIKM 2003)*. New Orleans, LA, USA: ACM Press, New York, USA, 2003. p. 386–393.

KIM, J. H.; VADAKKEPAT, P.; VERNER, I. M. Fira robot world cup initiative and research directions. *FIRA Documents – www.fira.net*, 1998.

KITANO, H. et al. Robocup: The robot world cup initiative. In: *Procs. of the Workshop on Entertainment and AI/Alife, IJCAI*. Montreal, Canada: IJCAI Press, 1995.

KITANO, H. et al. Robocup: A challenge problem for ai. *AI Magazine*, v. 18, n. 1, p. 73–85, 1997.

KNUTH, D. *The Art of Computer Programming*. Reading, Mass.: Addison-Wesley, 1973.

KOENIG, S.; SIMMONS, R. G. The effect of representation and knowledge on goal–directed exploration with reinforcement–learning algorithms. *Machine Learning*, v. 22, p. 227–250, 1996.

KONOLIGE, K.; MYERS, K. The saphira architecture for autonomous mobile robots. In: *AI-based Mobile Robots: Case studies of successful robot systems*. Canbridge, MA: MIT Press, 1996.

KRAETZSCHMAR, G. K. et al. The Ulm Sparrows: Research into sensorimotor integration, agency, learning, and multiagent cooperation. *Lecture Notes in Computer Science*, v. 1604, p. 459–465, 1999.

KUBE, C. R.; ZHANG, H. Collective robotics: from social insects to robots. *Adaptive Behavior*, v. 2, p. 189–218, 1994.

KUBE, C. R.; ZHANG, H. Task modelling in collective robotics. *Autonomous Robots*, v. 4, p. 53–72, 1997.

LITTMAN, M.; BOYAN, J. *A Distributed Reinforcement Learning Scheme for Network Routing*. [S.l.], 1993. 6 p.

LITTMAN, M. L. Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the 11th International Conference on Machine Learning (ICML-94)*. New Brunswick, NJ: Morgan Kaufmann, 1994. p. 157–163.

MANIEZZO, V.; DORIGO, M.; COLORNI, A. Algodesk: an experimental comparison of eight evolutionary heuristics applied to the QAP problem. *European Journal of Operational Research*, v. 81, p. 188–204, 1995.

MARIANO, C. E.; MORALES, E. MOAQ: an Ant–Q algorithm for multiple objective optimization problems. In: BANZHAF, W. et al. (Ed.). *Proceedings of the GECCO'99 – Genetic and Evolutionary Computation Conference*. San Francisco: Morgan Kaufmann, 1999. p. 894–901.

MARIANO, C. E.; MORALES, E. *A Multiple Objective Ant–Q Algorithm for the Design of Water Distribution Irrigation Networks*. Jiutepex, Mexico, 1999. Technical Report HC-9904.

MARIANO, C. E.; MORALES, E. MDQL: A reinforcement learning approach for the solution of multiple objective optimization problems. In: *PPSN/SAB Workshop on Multiobjective Problem Solving from Nature (MPSN)*. Paris, France: [s.n.], 2000.

MARIANO, C. E.; MORALES, E. A new distributed reinforcement learning algorithm for multiple objective optimization problems. *Lecture Notes in Artificial Intelligence*, v. 1952, p. 290–299, 2000.

MARIANO, C. E.; MORALES, E. DQL: A new updating strategy for reinforcement learning based on Q–learning. In: RAEDT, L. D.; FLACH, P. A. (Ed.). *EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5–7, 2001, Proceedings*. Berlin Heidelberg: Springer, 2001. (Lecture Notes in Computer Science, v. 2167), p. 324–335. ISBN 3-540-42536-5.

MARIANO-ROMERO, C. E. Aprendizaje por refuerzo en la solución de problemas de optimización con múltiples objetivos: Aplicación al diseño de redes de riego. In: BUSTAMANTE, W. O.; CARMONA, V. M. R. (Ed.). *XI Congreso Nacional de Irrigación, Simposio 10, Modelación Hidroagrícola. Guanajuato, Gto., México, 19–21 de Septiembre de 2001, Proceedings*. Guanajuato, Mexico: ANEI, 2001. p. 1–13.

MATARIC, M. J. Using communication to reduce locality in distributed multiagent learning. *JETAI*, v. 10, n. 3, p. 357–369, 1998.

MITCHELL, T. *Machine Learning*. New York: McGraw Hill, 1997.

MOORE, A. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In: BIRNBAUM, L.; COLLINS, G. (Ed.). *Machine Learning: Proceedings of the Eighth International Conference*. 340 Pine Street, 6th Fl., San Francisco, CA 94104: Morgan Kaufmann, 1991.

MOORE, A. W.; ATKESON, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, v. 13, p. 103–130, 1993.

MOORE, D. W. *Simplical Mesh Generation with Applications*. Tese (Doutorado) — Cornell University, 1992. Report no. 92-1322.

MUNOS, R.; MOORE, A. Variable resolution discretization in optimal control. *Machine Learning*, v. 49, n. 2/3, p. 291–323, 2002.

NEHMZOW, U. *Mobile Robotics: A Practical Introduction*. Berlin, Heidelberg: Springer-Verlag, 2000.

OGATA, K. *Engenharia de Controle Moderno*. 3. ed. São Paulo: LTC, 1998.

PEETERS, M.; VERBEECK, K.; NOWÉ, A. Reinforcement learning techniques for cooperative multi-agent systems. In: DUTECH, A. (Ed.). *6rd European Workshop on Reinforcement Learning, EWRL'03*. Nancy, France: [s.n.], 2003.

PEGORARO, R.; COSTA, A. H. R.; RIBEIRO, C. H. C. Experience generalization for multi-agent reinforcement learning. In: *SCCS'2001 – XXI International Conference of the Chilean Computer Science Society*. Los Amitos, CA: IEEE Computer Society Press, 2001. p. 233–239.

PENG, J.; WILLIAMS, R. J. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, v. 1, n. 4, p. 437–454, 1993.

REINELT, G. *TSPLIB95*. [S.l.], 1995. Technical Report. Universität Heidelberg. Disponível em: <ftp://softlib.rice.edu/pub/tsplib>.

REYNOLDS, S. I. *Reinforcement Learning with Exploration*. Tese (Doutorado) — University of Birmingham, Birmingham, 2002.

RIBEIRO, C. H. C. Embedding a priori knowledge in reinforcement learning. *Journal of Intelligent and Robotic Systems*, v. 21, p. 51–71, 1998.

RIBEIRO, C. H. C. On the use of option policies for autonomous robot navigation. In: MONARD, M. C.; SICHMAN, J. S. (Ed.). *Advances in AI: International Joint Conference, Proceedings, 7th. Ibero-American Conference on AI; 15th. Brazilian Symposium on AI; IBERAMIA-SBIA'2000*. Berlin, Heidelberg: Springer Verlag, 2000. (Lecture Notes in Artificial Intelligence, v. 1952), p. 369–378.

RIBEIRO, C. H. C. Reinforcement learning agents. *Artificial Intelligence Review*, v. 17, p. 223–250, 2002.

RIBEIRO, C. H. C.; PEGORARO, R.; COSTA, A. H. R. Experience generalization for concurrent reinforcement learners: the minimax-qs algorithm. In: CASTELFRANCHI, C.; JOHNSON, W. L. (Ed.). *International Joint Conference on Autonomous Agents and Multi-Agent Systems AAMAS'2002*. Bologna, Italy: [s.n.], 2002. p. 1239–1245. ISBN 1-58113-480-0. The Association for Computing Machiney.

ROBBINS, H.; MONRO, S. A stochastic approximation method. *Annals of Mathematical Statistics*, v. 22, p. 400–407, 1951.

ROMERO, C. E. M.; MORALES, E. A new approach for the solution of multiple objective optimization problems based on reinforcement learning. In: CAIRO, O.; SUCAR, L. E.; CANTU, F. J. (Ed.). *MICAI'2000*. Acapulco, México: Springer–Verlag, 2000. p. 212–223.

RUMMERY, G.; NIRANJAN, M. *On-line Q-learning using connectionist systems*. 1994. Technical Report CUED/F-INFENG/TR 166. Cambridge University, Engineering Department.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall, 1995.

RUSSELL, S.; NORVIG, P. *Inteligência Artificial*. 2. ed. Rio de Janeiro: Elsevier, 2004.

SANDERSON, A. Micro-robot world cup soccer tournament (mirosot). *IEEE Robotics and Automation Magazine*, Dec, p. 15, 1997.

SAVAGAONKAR, U. *Network Pricing using Sampling Techniques for Markov Games*. Tese (Doutorado) — Purdue University, West Lafayette, Indiana, 2002.

SCÁRDUA, L. A.; CRUZ, J. J. da; COSTA, A. H. R. Controle Ótimo de descarregadores de navios utilizando aprendizado por reforço. *Controle & Automação, SBA*, v. 14, n. 4, p. 368–376, 2003.

SCHOONDERWOERD, R. et al. Ant–based load balancing in telecommunications networks. *Adapt. Behav.*, v. 5, p. 169–207, 1997.

SEN, S.; WEIß, G. Learning in multiagent systems. In: WEIß, G. (Ed.). *Multiagent Systems*. Cambridge, MA: MIT press, 1999. p. 259–298.

SINGH, S. P. *Learning to solve Markovian Decision Processes*. Tese (Doutorado) — University of Massachusetts, Amherst, 1994.

SPIEGEL, M. R. *Estatística*. 2. ed. São Paulo: McGraw-Hill, 1984.

STONE, P.; VELOSO, M. M. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, v. 8, n. 3, p. 345–383, 2000.

STRANG, G. *Linear algebra and its applications*. 3. ed. San Diego: Harcourt, Brace, Jovanovich, 1988.

SUMMERS, L. *Harry Potter and the Paradigm of Uncertainty*. [S.l.]: Yahoo e-groups, 2001.

SUTTON, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, v. 3, n. 1, 1988.

SUTTON, R. S. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In: *Proceedings of the 7th International Conference on Machine Learning*. Austin, TX: Morgan Kaufmann, 1990.

SUTTON, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: TOURETZKY, D. S.; MOZER, M. C.; HASSELMO, M. E. (Ed.). *Advances in Neural Information Processing Systems*. [S.l.]: The MIT Press, 1996. v. 8, p. 1038–1044.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

SZEPESVÁRI, C.; LITTMAN, M. L. *Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms*. Brown University, Providence, Rhode Island 02912, 1996. CS-96-11.

TAMBE, M. Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, v. 12, 1998.

TAN, M. Multi-agent reinforcement learning: Independent vs. cooperative learning. In: HUHNS, M. N.; SINGH, M. P. (Ed.). *Readings in Agents*. San Francisco, CA, USA: Morgan Kaufmann, 1997. p. 487–494.

TESAURO, G. Temporal difference learning and td-gammon. *Communications of the ACM*, v. 38, p. 58–67, 1995.

THRUN, S. et al. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, v. 128, p. 99–141, 2001.

WATKINS, C. J. C. H. *Learning from Delayed Rewards*. Tese (Doutorado) — University of Cambridge, 1989.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. *Machine Learning*, v. 8, p. 279–292, 1992.

WEIß, G. Distributed Reinforcement Learning. *Robotics and Autonomous Systems*, v. 15, p. 135–14, 1995.

WEIß, G. (Ed.). *Distributed Artificial Intelligence Meets Machine Learning: Learning in Multi-Agent Environments*. Berlin Heidelberg: Springer, 1999. (Lecture Notes in Computer Science, v. 1237). ISBN 3-540-62934-3.

WEIß, G. A multiagent variant of Dyna-Q. In: *Proceedings of the Fourth International Conference on Multi-Agent Systems – ICMAS*. Los Alamitos: IEEE Computer Society, 2000. p. 461–462.

Índice Remissivo

Ant Colony Optimization, 44
Ant Colony System, 45
Aprendizado Acelerado por Heurísticas, 56
Aprendizado por Reforço, 8

Condição de Markov, 9

Dyna, 29
Dyna Multiagente, 42

Função Valor, 11

HAQL, 73
Heurísticas a partir de X, 69
Heuristically Accelerated Learning, 56

Inteligência de Enxame, 44
Iteração de Política, 12
Iteração de Valor, 12

Jogos de Markov, 21
Jogos de Markov de soma zero, 22

Markov Decision Process, 9
Minimax–Q, 23
Modelo de Horizonte Infinito, 11

Otimização por Colônia de Formigas, 44

Princípio da Otimalidade de Bellman, 11
Prioritized Sweeping, 29
Problema do Aprendizado por Reforço, 8
Processos Markovianos de Decisão, 9
Processos Markovianos de Decisão Generalizados, 24
Processos Markovianos de Decisão Parcialmente Observável, 10

Q(λ), 30
Q–Learning, 15
Q–Learning Distribuído, 40
Q–Learning Generalizado, 25
QS, 30
Queue-Dyna, 29

Retropropagação de Heurísticas, 72

SARSA, 20
SARSA(λ), 30

TD–Learning, 17

Apêndice I – Resumo sobre estatística

Neste apêndice é apresentado um resumo simples contendo noções básicas usadas de estatística neste trabalho.

I.1 O teste *t* de Student

O teste t de Student (SPIEGEL, 1984; NEHMZOW, 2000), que também é conhecido como *T-Test*, permite verificar se os resultados de duas experiências diferem significativamente. Ele é amplamente utilizado em todas as ciências e permite decidir se um algoritmo de controle exibe um desempenho melhor que outro.

Para estabelecer se dois valores médios μ_x e μ_y são diferentes, o valor T é calculado segundo a equação:

$$T = \frac{\mu_x - \mu_y}{\sqrt{(n_x - 1)\sigma_x^2 + (n_y - 1)\sigma_y^2}} \sqrt{\frac{n_x n_y (n_x + n_y - 2)}{n_x + n_y}}, \tag{I.1}$$

onde n_x e n_y são o número de pontos, μ_x e μ_y as médias e σ_x e σ_y o desvio padrão dos pontos nos experimentos x e y.

Se o módulo de T for maior que a constante t_α, a hipótese H_0 que $\mu_x = \mu_y$ é rejeitada, significando que os experimentos diferem significativamente. A constante t_α define a probabilidade que o teste tem de estar errado, chamado nível de confiança. Esta probabilidade é geralmente definida como 5%. O valor de t_α depende do número de experimentos e pode ser encontrado na maioria dos livros de estatística.

I.2 Erros médios

Os erros utilizados neste trabalho para comparar dois conjuntos de dados foram:

- A raiz quadrada do erro quadrático médio (*Root Mean Squared Error* – RMSE), definida como:

$$RMSE = \sqrt{\frac{1}{N}\sum_{s_t \in S} [V_1(s_t) - V_2(s_t)]^2},$$

onde N é o número de estados em S, $V_1(\cdot)$ e $V_2(\cdot)$ os valores a serem comparados.

- O erro percentual médio (*Mean Percentual Error* – MPE), definido como:

$$MPE = 100 \cdot \sqrt{\frac{1}{N}\sum_{s_t \in S} \left[\frac{V_1(s_t) - V_2(s_t)}{V_1(s_t)}\right]^2},$$

onde N é o número de estados em S, $V_1(\cdot)$ e $V_2(\cdot)$ os valores a serem comparados.