RICHARDSON RIBEIRO

AVALIAÇÃO E DESCOBERTA DE POLÍTICAS DE AÇÃO PARA AGENTES AUTÔNOMOS ADAPTATIVOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

CURITIBA

2006

RICHARDSON RIBEIRO

AVALIAÇÃO E DESCOBERTA DE POLÍTICAS DE AÇÃO PARA AGENTES AUTÔNOMOS ADAPTATIVOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: *Sistemas Inteligentes*

Orientador: Prof. Dr. Fabrício Enembreck
Co-orientador: Prof. Dr. Alessandro L. Koerich

CURITIBA
2006

	Ribeiro, Richardson
R484a	Avaliação e descoberta de políticas de ação para agentes autônomos
2006	adaptativos / Richardson Ribeiro ; orientador, Fabrício Enembreck ;
	co-orientador, Alessandro L. Koerich. – 2006.
	95f. : il. ; 30 cm
	Dissertação (mestrado) -- Pontifícia Universidade Católica do Paraná,
	Curitiba, 2006
	Inclui bibliografia
	1. Inteligência artificial – Aplicações educacionais. 2. Sistemas especialistas
	(Computação). 3. Sistemas tutoriais inteligentes. 4. Teoria do aprendizado
	computacional. I. Enembreck, Fabrício. II. Koerich, Alessandro L.
	III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação
	em Informática Aplicada. IV. Título.
	CDD 20. ed. – 006.3
	006.33

Ribeiro, Richardson
Avaliação e Descoberta de Políticas de Ação para Agentes Autônomos Adaptativos. Curitiba, 2006. 95 p.

Dissertação (Mestrado) – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. Agentes Adaptativos 2. Aprendizagem Híbrida 3. Aprendizagem por Reforço 4. Avaliação de Políticas de Ação. I. Pontifícia Universidade Católica do Paraná. II. Centro de Ciências Exatas e de Tecnologia. III. Programa de Pós-Graduação em Informática Aplicada.

Aos meus queridos pais
Luis e Vânia,
meu dedicado irmão
Charlison,
minha carinhosa noiva
Adriana,
aos meus exemplos de profissionalismo
Fabrício e *Alessandro,*
e a *Deus* acima de tudo.

Agradecimentos

Sem dúvida, meus primeiros agradecimentos de forma especial são para meus orientadores Prof. Dr. Fabrício Enembreck e ao Prof. Dr. Alessandro L. Koerich que me receberam como orientando e me confiaram seus projetos sem mesmo conhecer meu trabalho. Agradeço pela amizade, comprometimento, disponibilidade e principalmente seus rigores científico que sem eles teria sido impossível chegar aonde se chegou. Obrigado pela paciência e compreensão nos momentos mais difíceis.

Aos amigos e companheiros do Laboratório de Sistemas Inteligentes – Lasin: Marcio, Helyane, Gustavo, Heverson, Daniele, André, Jaime, Leonardo, Haruo, entre outros, no qual juntos compartilhamos conhecimentos, alegrias e desesperos.

Ao meu pai, sempre presente e prestativo com seus conselhos, que inteligência. Nunca alcançarei sua cultura e seu conhecimento, meu exemplo de vida e superação, obrigado. A minha mãe, que sempre cuidou bem de mim com muito amor e carinho, obrigado. Ao meu irmão Charlison, obrigado pela atenção e favores prestados. Você teve grande contribuição na finalização desta dissertação.

A minha noiva Adriana, foi a pessoa mais motivadora para a realização desse trabalho. Essa dissertação é dedicada a ela, uma pessoa fantástica, que inteligência, eu prometo que agora é só mais o doutorado. Aos meus tios, Ivone e Dori, seus filhos e seus netos, obrigado pelo carinho e pela moradia nesta cidade.

Agradeço aos irmãos Douglas e Roger pela ajuda técnica ao trabalho, parabéns pela competência e organização. Aos meus orientadores de graduação Msc. Ivan Carlos Vicentin e Msc. Orestes Hacke e ao meu grande amigo Carlos Rafael Guerber. Sem esquecer ainda aquela que considero como minha segunda mãe, Dona Cisa.

Por fim, muito obrigado a todos da banca avaliadora que contribuíram com críticas e sugestões para as melhorias desta dissertação.

Sumário

Lista de Figuras iii
Lista de Tabelas v
Lista de Símbolos vi
Lista de Abreviaturas ix
Resumo x
Abstract xii
Capítulo 1- Introdução 1
1.1. Definição do Problema 3
1.2. Motivação 4
1.3. Hipóteses 5
1.4. Objetivos 5
1.5. Metodologia de Avaliação 6
1.6. Organização da Dissertação 6
Capítulo 2 - Agentes Autônomos Inteligentes 7
2.1. Considerações Iniciais 7
2.2. Agentes Inteligentes 7
2.3. Agentes Autônomos 10
2.4. Agentes Autônomos Adaptativos 11
2.5. Arquiteturas de Agentes 11
2.6. Técnicas de Aprendizagem para Agentes 13
2.6.1. Aprendizagem por Modelos 15
2.6.2. Aprendizagem Baseada em Regras 15
2.6.3. Aprendizagem por Reforço 16
2.6.4. Aprendizagem Baseada em Instâncias 16
2.7. Aprendizagem em Sistemas Multi-Agente 17
2.8. Agentes Adaptativos e Aprendizagem por Reforço 18
2.9. Aplicações de Agentes Autônomos Adaptativos 19
2.10. Considerações Finais 21
Capítulo 3 - Princípios de Aprendizagem por Reforço 22
3.1. Considerações Iniciais 22
3.2. Origens 22
3.3. Definições 23
3.4. Características da Aprendizagem por Reforço 24
3.5. Elementos Fundamentais da Aprendizagem por Reforço 25
3.6. Processos Markoviano 28
3.7. Principais Algoritmos de Aprendizagem por Reforço 29

3.7.1. Algoritmo R-Learning 29
3.7.2. Algoritmo H-Learning 30
3.7.3. Algoritmo $Q(\lambda)$ 32
3.7.4. Algoritmo Sarsa 32
3.7.5. Algoritmo Dyna 33
3.7.6. O Algoritmo Q-Learning 33
3.8. Medidas de Avaliação de Políticas 36
3.9. Técnicas de Estimação de Políticas Baseadas em Regressão 38
3.10. Métodos Híbridos 40
3.11. Considerações Finais 42
Capítulo 4 - Uma Metodologia para Avaliação de Políticas 43
4.1. Considerações Iniciais 43
4.2. Uma Nova Metodologia de Avaliação 43
4.3. Implementação e Simulação 46
4.3.1. Linguagem de Programação 47
4.3.2. Definição da Situação dos Estados 47
4.3.3. Desenvolvimento do Simulador 48
4.3.4. Descrição do Simulador 49
4.4. Utilização da Metodologia para Avaliar o Desempenho do Q-Learning 55
4.4.1. Número de Passos Necessários para Atingir a Melhor Eficiência 56
4.4.2. Política de Qualidade de Recompensas 57
4.4.3. Taxa de Aprendizagem 57
4.4.4. Fator de Desconto 58
4.5. Considerações Finais 59
Capítulo 5 - Avaliação e Descoberta de Políticas Utilizando Métodos Híbridos de Aprendizagem 60
5.1. Considerações Iniciais 60
5.2. Algoritmos de Regressão Baseados no KNN 60
5.3. O Método K-Learning 63
5.4. Algoritmos Utilizados nos Experimentos 65
5.4.1. Algoritmo Q-Learning 65
5.4.2. Algoritmo K-NN 66
5.5. Algoritmo Q-Learning *vs.* K-NN 69
5.6. Comparação dos Algoritmos Q-Learning, K-NN e K-Learning 70
5.7. Otimizando os Algoritmos de Aprendizagem 74
5.7.1. Algoritmo Q-Learning *vs.* Algoritmo Q-Learning com Heurística 74
5.7.2. Algoritmo K-NN *vs.* Algoritmo K-NN com Heurística 76
5.7.3. Algoritmo K-Learning *vs.* Algoritmo K-Learning com Heurística 77
5.8. Considerações Finais 81
Capítulo 6 - Conclusões Finais 82
Referências Bibliográficas 85
Apêndice A - A1. Algoritmo A^* (A *star*) 95

Lista de Figuras

Figura 1.1: Situação dos estados do ambiente de simulação.... 4
Figura 2.1: Esquema típico de um agente [RUS04] 9
Figura 3.1: Problema da AR [SUT92] 23
Figura 3.2: Algoritmo R-Learning [FAR00] 30
Figura 3.3: Algoritmo H-Learning [FAR99] 31
Figura 3.4: Algoritmo Q-Learning [MON04] 35
Figura 4.1: Pseudocódigo do algoritmo de avaliação de desempenho do algoritmo Q-Learning e de um algoritmo *X* 44
Figura 4.2: Pseudocódigo da métrica de acerto de uma política *P* 45
Figura 4.3: Ambiente de simulação com o melhor caminho traçado pelo algoritmo *A** 46
Figura 4.4: Nível de cores que representam a situação de cada estado 48
Figura 4.5: Tela do simulador 48
Figura 4.6: Melhor caminho aprendido pelo agente 50
Figura 4.7: Ações de uma política 50
Figura 4.8: Situações de erro 50
Figura 4.9: Gráfico do aprendizado do agente em relação ao número de trocas realizadas 51
Figura 4.10: Gráfico do aprendizado do algoritmo Q-Learning 51
Figura 4.11: Conjunto de todos os gráficos da aprendizagem do agente 52
Figura 4.12: Gráfico do aprendizado do algoritmo K-Learning 52
Figura 4.13: Valores e legendas das situações do ambiente 53
Figura 4.14: Tabela de aprendizagem do agente 53
Figura 4.15: Configurações da aprendizagem do agente 54
Figura 4.16: Critérios de parada do agente 54
Figura 4.17: Custos dos passos dos agentes 55
Figura 4.18: Tabela de comparação dos algoritmos 55
Figura 4.19: Configurações para os gráficos de eficiência 55
Figura 4.20: Gráfico comparativo da aprendizagem do algoritmo Q-Learning relacionando eficiência, número de estados e número de ciclos 57
Figura 4.21: Gráfico de eficiência das taxas de aprendizagem 58
Figura 4.22: Gráfico de eficiência dos fatores de desconto 59
Figura 5.1: Pseudocódigo do algoritmo que aproxima as instâncias e estima seu atributo- meta 62
Figura 5.2: Pseudocódigo da integração do algoritmo K-NN com o algoritmo Q-Learning 63
Figura 5.3: Gráfico comparativo dos algoritmos Q-Learning e K-NN 64
Figura 5.4: Pseudocódigo do método K-Learning 65

Figura 5.5: Eficiência do algoritmo K-NN em ambientes com 16 estados 67
Figura 5.6: Eficiência do algoritmo K-NN em ambientes com 25 estados 67
Figura 5.7: Eficiência do algoritmo K-NN em ambientes com 64 estados 68
Figura 5.8: Gráfico da superioridade de um arranjo com o K-NN 69
Figura 5.9: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 16 estados 70
Figura 5.10: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 25 estados 71
Figura 5.11: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 36 estados 71
Figura 5.12: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 49 estados 72
Figura 5.13: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 64 estados 72
Figura 5.14: Q-Learning *vs.* Q-Learning com heurística em ambientes com 25 estados 75
Figura 5.15: Q-Learning *vs.* Q-Learning com heurística em ambientes com 49 estados 75
Figura 5.16: Q-Learning *vs.* Q-Learning com heurística em ambientes com 64 estados 76
Figura 5.17: K-NN *vs.* K-NN com heurística em ambientes com 25 estados 76
Figura 5.18: K-NN *vs.* K-NN com heurística em ambientes com 49 estados 77
Figura 5.19: K-NN *vs.* K-NN com heurística em ambientes com 64 estados 77
Figura 5.20: K-Learning *vs.* K-Learning com heurística em ambientes com 25 estados 78
Figura 5.21: K-Learning *vs.* K-Learning com heurística em ambientes com 49 estados 78
Figura 5.22: K-Learning *vs.* K-Learning com heurística em ambientes com 64 estados 79
Figura 5.23: K-Learning $\varepsilon - Greedy$ com < 100% 80
Figura 5.24: K-Learning com $\varepsilon - Greedy$ em 100% *vs.* K-Learning com $\varepsilon - Greedy$ menor que 100% 81

Lista de Tabelas

Tabela 5.1: Exemplo de atributos e valores do Arranjo[i...n] gerados pelo agente com o algoritmo Q-Learning durante aprendizagem 61
Tabela 5.2: Comparativo dos algoritmos Q-Learning e K-NN 69
Tabela 5.3: Comparativo da eficiência média dos algoritmos Q-Learning e K-NN com o método K-Learning. 73
Tabela 5.4: Otimização do número de passos do método K-Learning 73

Lista de Símbolos

λ	Parâmetro do algoritmo $Q(\lambda)$
β	Parâmetro do algoritmo R-Learning
$\varepsilon - Greedy$	Ação aleatória com probabilidade em ε
α	Taxa de aprendizagem
γ	Fator de desconto para os reforços futuros
ρ	Recompensa média do reforço imediato
π	Política qualquer
$\pi(s,a)$	Política mapeada de estado-ações
$\pi*$	Política ótima qualquer
a	Determinada ação em um ambiente
a_t	Ação corrente
$a*$	Melhor ação
A	Conjunto finito de ações
$Arranjo[completo]$	Conjunto completo de todos os arranjos adquiridos pelo agente
$Arranjo[n\_ciclos]$	Conjunto de arranjo formado por um número *n* de ciclos realizados pelo agente
$Arranjo[ultimo\_ciclo]$	Arranjo formado pelo último ciclo de aprendizagem do agente
$Arranjo[ultimo\_ciclo \approx K-nn]$	Último arranjo adquirido pelo agente calculado sobre o último arranjo estimado pelo algoritmo K-NN
$A*$	Algoritmo *A*-Estrela
$g(n)$	Número de passos do agente até o estado final (objetivo)

$h(n)$	Função heurística
K	Número de vizinhos mais próximos
$CustMin \quad (A^*)$	Custo mínimo de um caminho até o estado final com o algoritmo A^*
$N(i,u)$	Número de vezes que a ação u foi executada no estado i
$O(n^2)$	Complexidade teórica do algoritmo A^*
$P(s' \mid s,a)$	Probabilidade condicional do próximo estado, dada o estado atual e ação
$p_{ij}(u)$	Probabilidade de ir de um estado i para um estado k executando a ação u
$PK(s,a)$	Tabela que armazena os valores aprendidos por uma determinada política de ações com o algoritmo K-NN
$PKI(s,a)$	Tabela que armazena os valores aprendidos por uma determinada política de ações com o algoritmo K-Learning
PK^*	Política ótima do algoritmo K-NN
PKI^*	Política ótima do algoritmo K-Learning
PQ^*	Política ótima do algoritmo Q-Learning
$PQ(s,a)$	Tabela que armazena os valores aprendidos por uma determinada política de ações com o algoritmo Q-Learning
$\hat{Q}$	Estimativa da função valor-ação
Q-table(s,a)	Tabela que armazena os valores aprendidos com o algoritmo Q-Learning
$Q(s,a)$	Função valor-ação
$Q : SxA \rightarrow V$	Mapeamento de uma função, onde V é valor de utilidade esperado ao se executar uma ação a no estado s
Q^*	Função valor-ação ótima
$Q(\lambda)$	Algoritmo de aprendizagem por reforço
$r_{s,a}$	Recompensa de um determinado estado-ação

$r(s_t, a_t)$	Reforço recebido após realizar a_t em s_t
$R(s,a) \rightarrow \Re$	Função de recompensa
s	Estado de um ambiente
s'	Próximo estado
s_t	Estado corrente do ambiente ou estado atual
s_{t+1}	Estado resultante ou novo estado
$s_t \rightarrow s_{t+1}$	Transição de estados
S	Conjunto finito de estados
Sit	Custo acumulado da situação de cada estado
(s,a)	Par estado/ação
t	Instante de tempo
T	Tempo final
$Tr(s,a)$	Traço de elegibilidade de um par estado-ação
$T : SxA \rightarrow \Pi(S)$	Função de transição de estados com distribuição de probabilidades
$T(\cdot)$	Função de probabilidade de transição
$V(s)$	Função valor-estado
$< S, A, T, R >$	Tupla composta por conjunto de estado, ação, transição de estados e função de recompensas

Lista de Abreviaturas

AR	Aprendizagem por Reforço
B2B	*Business-to-business*
B2C	*Business-to-consumer*
IBL	*Instance-based Learning*
K-NN	*K Nearest Neighbour* (*K* vizinhos mais próximos)
MAP	Metodologia de Avaliação de Políticas
MDP	*Markov Decision Process* (Processos Decisórios de Markov)
POMDP	*Partially Observable Markov Decision Process* (Processos Decisórios de Markov Parcialmente Observáveis
SMA	Sistemas Multi-Agente
TAC	*Trade Agent Competition*

Resumo

Esta dissertação apresenta uma nova metodologia, denominada MAP, para avaliar o desempenho do algoritmo Q-Learning e outras técnicas de estimação de políticas de ação para agentes autônomos adaptativos. A avaliação de políticas de ação produzidas por esses métodos de aprendizagem é uma tarefa complexa. Isso ocorre devido à falta de mecanismos genéricos que permitam medir o desempenho de um agente de aprendizagem sem demandar conhecimentos do domínio do problema e de longos processos de simulação, dificultando a compreensão dos resultados e a aplicação da técnica em diferentes ambientes. Na metodologia proposta, o desempenho de um agente é proporcional ao número de acertos produzidos pela sua política de ações em um dado ambiente. Uma política representa um espaço de estados onde existe um estado inicial, um estado final e um conjunto de transições de estados determinada por diferentes ações. Portanto, um acerto é obtido quando o agente encontra um caminho de menor custo entre o estado inicial e o estado final. Quando isso ocorre para todos os estados candidatos, pode-se dizer que uma política ótima foi descoberta. Dessa forma, a avaliação de uma política utiliza um algoritmo de resolução de problemas capaz de encontrar o caminho ótimo entre dois estados. Essa metodologia de avaliação permite observar o comportamento dos algoritmos em função do número de iterações, configuração do ambiente e valores dos parâmetros relacionados ao algoritmo de aprendizagem. MAP foi integrada a um sistema híbrido de aprendizagem visando diminuir a quantidade de iterações e o erro de um agente adaptativo. O método híbrido de aprendizagem combina aprendizagem por reforço e aprendizagem baseada em instâncias para a descoberta de políticas de ação. O método proposto, chama-se K-Learning e integra os algoritmos Q-Learning e K-NN. Verificou-se que o K-Learning pode ser superior ao Q-Learning em experimentos empíricos conduzindo o agente para uma política de boa qualidade mais rapidamente.

Palavras-Chaves: Agente Adaptativo, Avaliação de Políticas de Ação, Aprendizagem Híbrida, Aprendizagem por Reforço.

Abstract

This dissertation presents a new evaluation methodology namely EPM (*E*valuation *P*olicy *M*ethodology), that has been developed to evaluate the performance of the Q-Learning algorithm and other techniques to estimate policy of actions for adaptive autonomous agents. The evaluation of politics of action produced by these methods of learning is a complex task. This occurs due to the lack of generic mechanisms that allow measuring the performance of an agent of learning without needing the knowledge of the domain of the problem and of long simulation processes, making it difficult to understand the results and the application of the technique in different environments. In the proposed methodology, the performance of an agent is considered proportional to the number of successful hits produced for its action policy in a given environment. A policy represents a space of states with initial and target states where the transition between states is determined by the actions. Therefore, a correct decision is reached when the agent finds a way to reduce the cost between the initial and target states. When this occurs for all the candidate states, it can be said that an optimal policy has been discovered. The proposed evaluation methodology allows observing the behavior of the learning algorithm as a function of the number of iterations, configuration of environment and values of the parameters related to the algorithm. The EPM has been integrated to a learning hybrid system whose objective is to reduce the amount of errors and the number of iterations of an adaptive agent. The hybrid learning method combines Reinforcement Learning and Instance-based Learning algorithms to discover a policy of action that is evaluated and selected through the EPM. The proposed method, namely K-Learning, integrates the Q-Learning and K-Nearest Neighbors algorithm. Experiments have shown that the K-Learning method can be faster than the conventional Q-Learning in achieving a good policy of action.

Keywords: Adaptive Agents, Evaluation Action Policy, Hybrid Learning, Reinforcement Learning.

Capítulo 1

Introdução

Durante anos, procuramos entender como os seres humanos tomam decisões; isto é, como podemos ser capazes de compreender, prever e manipular um mundo muito maior e mais complexo do que nós mesmos. O campo da IA (Inteligência Artificial) vai ainda mais além. E ele tenta não apenas compreender, mas também construir entidades inteligentes [RUS95].

Pesquisas sobre agentes de software são realizadas por pesquisadores de diferentes áreas como IA, Sistemas Distribuídos, Interface Homem-Computador, Robótica, entre outras, que juntas têm como principais objetivos: atender aos novos requisitos exigidos por determinadas aplicações; facilitar a interação usuário/máquina; e construir sistemas inteligentes. Um agente de software pode maximizar uma medida de desempenho baseada nos reforços (recompensas ou punições) que recebe ao interagir com um ambiente geralmente parcialmente conhecido. Esse paradigma computacional de aprendizagem é denominado Aprendizagem por Reforço (AR) [RIB99].

Na literatura são encontrados diversos trabalhos que abordam AR e descrevem diferentes aplicações no uso de agentes inteligentes [RUS95] [TES95] [KAE96] [MIT97] [SUT98]. Métodos de AR tratam de situações onde um agente aprende por tentativa e erro ao atuar sobre um ambiente complexo [SUT98]. Desta maneira, não é necessária uma entidade externa que forneça exemplos ou um modelo de tarefas a executar. A única fonte de aprendizado é a própria experiência do agente, cujo objetivo formal é adquirir uma política de ações que maximize seu desempenho geral.

A avaliação de políticas de ação produzidas por esses métodos de aprendizagem é uma tarefa complexa. Isso ocorre devido à falta de mecanismos genéricos que permitam medir o desempenho de um agente de aprendizagem sem demandar conhecimentos do domínio do

problema e de longos processos de simulação. Além disso, as técnicas existentes não permitem que políticas de ação sejam avaliadas em diferentes ambientes. Políticas de ações correspondem a funções que mapeiam estados e ações. Um agente pode seguir várias políticas de ação, mas o objetivo da aprendizagem é encontrar a política que maximize a soma das recompensas futuras, isto é, o total de recompensas recebidas após a adoção dessa política.

Trabalhos como de Kononen [KON04], Monteiro e Ribeiro [MON04], Ernst et al. [ERN05], Ramon [RAM05] e Lev et al. [LEV02] utilizam avaliações de desempenho complexas, pois suas aplicações referem-se a domínios e classes distintas, e ainda não contam com uma metodologia de avaliação padronizada. Além disso, é observada uma certa carência de trabalhos que tratem de mecanismos genéricos de avaliação em problemas que envolvem AR. Dessa forma, é apresentada nesta dissertação uma nova metodologia genérica para a avaliação do desempenho de agentes autônomos adaptativos que utilizam algoritmos de AR. Essa metodologia permite que o desempenho de um agente seja estimado sem que longos períodos de simulação e conhecimento do domínio sejam necessários. Esta metodologia é chamada de Metodologia de Avaliação de Políticas (MAP).

Enquanto poucos trabalhos têm tratado o problema de metodologias eficazes de avaliação de políticas, muitos trabalhos têm sido propostos na tentativa de otimizar as tarefas de um agente de aprendizagem. Autores como Siedlecki e Skalanski [SIE89], Downing [DOW01], Rayan [RYA02] e Henderson et al. [HEN05], integram diferentes métodos de aprendizagem na esperança de que uma técnica possa corrigir a deficiência da outra, tornando os sistemas mais robustos e mais tolerantes a falhas.

Este trabalho apresenta também um método de aprendizagem que integra AR e aprendizagem baseada em instâncias na intenção de melhorar o desempenho[1] do agente. A união dessas técnicas forma um mecanismo híbrido de aprendizagem, denominado de método K-Learning. K-Learning também utiliza a metodologia MAP para selecionar políticas eficientes ao longo da aprendizagem e integra os algoritmos Q-Learning e *K* vizinhos mais próximos, geralmente chamado de K-NN. O algoritmo Q-Learning, proposto por Watkins [WAT92], permite estabelecer de maneira autônoma políticas de ação de maneira iterativa. Por outro lado, o algoritmo K-NN tem como característica a simplicidade e a habilidade em produzir classificações com alto grau de precisão após o aprendizado [AHA91]. Essa técnica é utilizada para estimar os valores da aprendizagem do agente gerados durante sua fase de

[1] O desempenho do agente é avaliado em termos de tempo de convergência, qualidade da política, número de iterações e ajustes de seus parâmetros de aprendizagem e ambiente.

treinamento. Este método híbrido tem como objetivo utilizar as melhores características de cada método: o K-NN permite que estados com características semelhantes tenham recompensas similares, antecipando recompensas e diminuindo o número de iterações necessárias ao Q-Learning. Por outro lado, o Q-Learning estabelece uma forma de atualização de recompensas que garante a descoberta da melhor política ao longo da aprendizagem do agente.

1.1. Definição do Problema

O problema consiste na avaliação do desempenho de um agente adaptativo que interage de forma autônoma e inteligente em ambientes parcialmente desconhecidos, aplicando uma metodologia capaz de descobrir e avaliar novas políticas de ação.

Ao longo desta dissertação, será utilizado o problema de simulação de tráfego como contexto para a avaliação da metodologia proposta. Neste problema o agente adaptativo fará a simulação de um motorista que tem por missão chegar ao seu destino de maneira a conquistar a maior recompensa possível em função das ações tomadas. O ambiente a ser simulado pelo agente representa uma malha viária, onde situações inesperadas de trânsito são apresentadas ao agente (congestionamentos, bloqueios parciais, entre outros).

Os estados do ambiente, impostos ao agente, podem assumir os seguintes valores: livre, pouco congestionado, congestionado, muito congestionado, bloqueado e desconhecido. Cada estado possui um valor que irá representar uma situação dentro do ambiente. Em cada estado o agente possui um conjunto de ações disponíveis. Essas ações são representadas pelos seguintes movimentos: para frente, para trás, para direita e para esquerda. Após cada movimento o agente recebe um novo estado apresentado pelo ambiente e novas ações poderão ser tomadas. O agente saberá se a sua ação foi positiva ou negativa por meio de recompensas atribuídas pelo ambiente, seguindo uma política de ações.

A Figura 1.1 apresenta um exemplo onde são representados os possíveis estados do ambiente. Cada cor representa um estado e sua situação. O agente será capaz de perceber somente os estados adjacentes ao estado atual e só poderá se movimentar sobre eles. A cada ação, novos estados adjacentes são percebidos pelo agente, podendo este então tomar novas decisões baseado na sua política de ações.

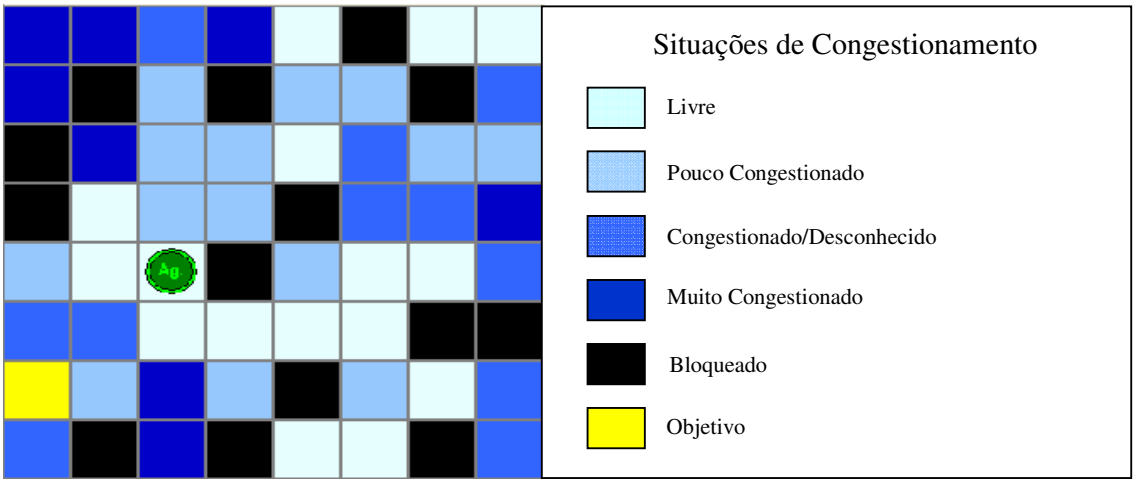

Figura 1.1: Situação dos estados do ambiente de simulação

1.2. Motivação

Estudos sobre agentes inteligentes vêm se intensificando devido às suas diversas aplicações como, por exemplo: simuladores de agentes, modelagem de agentes, adaptação de perfis de usuários, assistentes pessoais, agente de recuperação de informação, tecnologias de software móveis, agentes de mercado como mediadores e facilitadores, entre outros. Agentes autônomos representam entidades que interagem com o ambiente a fim de atingir seus objetivos. Este deve ter a capacidade de reunir informações do seu ambiente, tomar decisões baseadas nestas informações e iniciar uma ação especifica baseada nas decisões.

Embora os métodos de AR e aprendizagem baseada em instâncias sejam comuns, é difícil encontrar na literatura estudos comparativos entre eles. Desta forma, não é trivial saber qual deles empregar em determinada aplicação. Boa parte dessa dificuldade advém da falta de uma metodologia de avaliação que permita analisar as vantagens, desvantagens e o desempenho desses métodos. Assim, neste trabalho se tentará preencher esta lacuna, propondo uma metodologia de avaliação que permita identificar os ajustes necessários aos parâmetros de algoritmos AR e que permita mapear o impacto de cada um deles no desempenho do agente.

Por último, diversos trabalhos vêm apresentando a integração de técnicas complementares, formando sistemas híbridos que visam eliminar deficiências e melhorar o desempenho do agente de aprendizagem [SIE89] [DOW01] [RYA02] [HEN05]. Dessa forma, duas técnicas de aprendizagem foram integradas neste trabalho, buscando melhorar o

desempenho do agente e mostrando que técnicas de AR e de aprendizagem baseada em instâncias podem ser usadas em conjunto.

1.3. Hipóteses

Uma política representa um espaço de estados onde existe um estado-inicial, um estado final e um conjunto de transições entre estados definidos por ações. Dessa forma, a avaliação de uma política pode ser realizada por meio de um algoritmo de resolução de problemas capaz de encontrar o caminho ótimo entre dois estados onde a função heurística corresponde aos valores retornados pela função de recompensa. Além disso, a quantidade de estados visitados também pode ser considerada. Utilizamos esses conceitos no desenvolvimento da metodologia MAP que emprega o conhecido algoritmo de busca *A** que garante uma solução completa e ótima [RUS94] [NIL98].

Ademais, algoritmos de AR e aprendizagem baseada em instância podem ser integrados para o desenvolvimento de métodos robustos de aprendizagem. Algoritmos de AR, como o Q-Learning, geralmente produzem boas políticas ao longo da aprendizagem enquanto que algoritmos de aprendizagem baseado em instância, como o K-NN podem definir funções discriminantes complexas. Dessa forma, estados que recebem recompensas similares podem ser identificados e uma técnica de unificação pode ser empregada de forma a antecipar valores de recompensas para iterações futuras. Esses conceitos foram utilizados no desenvolvimento do método de aprendizagem híbrida chamado de K-Learning, que será apresentado nos capítulos seguintes.

A comparação de diferentes métodos de estimação de políticas pode ser facilitada por um ambiente de simulação que utiliza uma metodologia unificada de avaliação. Além disso, um ambiente com essas características permite que os principais parâmetros dos algoritmos Q-Learning, K-NN e o método K-Learning sejam estudadas e, dessa forma, seus parâmetros de aprendizagem podem ser ajustados. Neste trabalho será apresentado um simulador que permite que todos esses parâmetros sejam manipulados durante a aprendizagem do agente a fim de encontrar os valores para os parâmetros que maximizem o desempenho.

1.4. Objetivos

Esta dissertação possui dois objetivos distintos. O primeiro é desenvolver uma metodologia genérica para a avaliação do desempenho de agentes autônomos adaptativos que

utilizam algoritmos de AR. Essa metodologia deve permitir que o desempenho de um agente seja estimado sem a necessidade de longos períodos de simulação e sem o conhecimento *a priori* do domínio de aplicação. Pretende-se utilizar a metodologia proposta para apontar os melhores valores para os parâmetros dos algoritmos e identificar quais dos métodos melhor se adapta ao problema proposto. O segundo objetivo é desenvolver um método híbrido de aprendizagem, unificando técnicas de AR com algoritmos de estimação baseada em instâncias. Esse método é utilizado no intuito de descobrir políticas de ação estimando os valores gerados pelo algoritmo de AR. Entretanto, este método deve utilizar uma metodologia eficiente de avaliação para a escolha de melhores políticas.

1.5. Metodologia de Avaliação

A metodologia MAP será utilizada para analisar a influência sobre o desempenho causado pela variação dos parâmetros dos algoritmos de aprendizagem considerando o ambiente descrito na Seção 1.2, sendo observados os parâmetros de aprendizagem de cada método. O algoritmo Q-Learning será avaliado considerando variações nos parâmetros: (i) taxa de aprendizagem representado por α; (ii) reforço recebido pela realização da ação a no estado s, indicado por R; (iii) um fator de desconto que determina o peso temporal relativo dos reforços indicado por γ e ainda os valores das recompensas para cada estado. Esses parâmetros têm se mostrado importantes para o sucesso da aprendizagem [MON04].

Com relação à aprendizagem baseada em instâncias serão analisados os parâmetros com grande influência na classificação dos novos exemplos: (i) a quantidade de K vizinhos mais próximos a serem utilizados; (ii) a escolha de diferentes atributos e conjuntos de instâncias a serem utilizadas. O método K-Learning será analisado quanto à seu desempenho utilizando os melhores parâmetros observados individualmente para o Q-Learning e o K-NN.

1.6. Organização da Dissertação

O presente trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta as principais descrições sobre agentes autônomos adaptativos. No Capítulo 3 são apresentados os conceitos sobre os paradigmas de AR. O Capítulo 4, por sua vez, apresenta a nossa metodologia para avaliação de políticas de ação. Já o Capítulo 5, discute os principais resultados obtidos após aplicação da metodologia e os resultados comparativos entre os métodos. Por último, as conclusões da pesquisa são apresentadas no Capítulo 6.

Capítulo 2

Agentes Autônomos Inteligentes

Neste capítulo será apresentada uma introdução sobre agentes autônomos inteligentes. Agentes autônomos são utilizados neste trabalho para simular e avaliar o desempenho de técnicas de aprendizagem de máquina. Ao longo desde capítulo, são revisados os principais conceitos sobre agentes autônomos e algumas de suas técnicas de aprendizagem.

2.1. Considerações Iniciais

A aprendizagem automática de agentes vem recebendo grande atenção por parte da comunidade da IA. Mesmo quando se trata de aplicações aparentemente simples, devido à complexidade da maioria dos sistemas de aprendizagem, torna-se muitas vezes difícil ou mesmo impossível prever quais comportamentos poderão levar um agente a obter os melhores resultados [FON00]. Em razão destas dificuldades, há o interesse em dotar os agentes inteligentes com alguma capacidade de auto-adaptação e aprendizagem que lhes permitam modificar seu comportamento em função da experiência adquirida e das possíveis alterações no ambiente onde eles atuam. Um agente é dito inteligente quando possui capacidade de compreender e adaptar-se a diferentes situações ou tarefas a ele atribuídas. Um agente inteligente interage de forma autônoma quando tem a capacidade de executar o controle sobre suas próprias ações em seu ambiente de interação. Os agentes que seguem estas definições e conseguem melhorar seu comportamento em função de suas ações anteriores são definidos como *agentes autônomos adaptativos*.

2.2. Agentes Inteligentes

A área de agentes é integrada por pesquisadores de diferentes áreas como, IA, sistemas distribuídos, interface humana computador e robótica, que juntos têm como principais objetivos: atender aos novos requisitos exigidos por determinadas aplicações; facilitar a interação usuário/máquina e; construir sistemas inteligentes.

A partir das definições observadas nos trabalhos de alguns pesquisadores é possível perceber que vários deles possuem diferentes opiniões para o termo agente [MAE94] [RUS95] [WOO95] [JEN96] [MUE97]. Dessa forma, não existe uma definição unânime dentre os pesquisadores sobre o conceito "agente", porém, muitas destas definições estudadas se complementam.

Em Russell e Norvig [RUS95] o termo agente é definido como tudo que pode perceber seu ambiente por meio de sensores e agir sobre ele por meio de atuadores. Realizando uma analogia simples, um humano possui diversos sensores como olhos, ouvidos e nariz e atuadores como os braços, pernas e boca. Um robô possui, por exemplo, câmeras de vídeo como sensores e motores como sendo os atuadores. Em um programa as percepções e ações são codificadas em uma seqüência de bits [OLI00].

Maes [MAE94] e Hendler [HEN96] consideram agentes inteligentes como programas de IA cuja finalidade é agir em diversos ambientes de importância para os humanos, sendo divididos em duas categorias: agentes físicos e agentes de informação. Os agentes físicos trabalham em um ambiente onde é difícil inserir um ser humano (ex. espaço) ou que seja perigoso (ex. núcleo de um reator nuclear). Os agentes de informação, ou *softbots* (*software robots*), atuam em um mundo virtual onde existe uma grande quantidade de informações espalhadas por diversos computadores (ex. Internet).

Wooldridge [WOO95] divide as aplicações de agentes em dois agrupamentos. O primeiro, sob uma notação mais fraca, na qual agentes compõem o hardware ou, geralmente, softwares dotados de *autonomia* para a realização de suas tarefas, *habilidades* para interagir com outros agentes e entidades e *reatividade* ao meio em que está inserido, a qual geralmente aumenta o grau de dinamismo e complexidade. Sob um contexto mais complexo, o segundo agrupamento se baseia em uma notação mais forte (utilizada por pesquisadores da IA), a qual define um agente como um software que, além das propriedades anteriores, implementam conceitos geralmente aplicados aos seres humanos, tais como o conhecimento, a crença, a intenção e a obrigação.

No geral, um agente é definido como uma entidade de software que exibe comportamentos autônomos e está situado em algum ambiente sobre o qual é capaz de realizar ações para alcançar seu próprio objetivo. O termo ambiente refere-se a uma representação do sistema estudado, onde os agentes são simulados. A Figura 2.1 apresenta um esquema típico entre um ambiente e o agente.

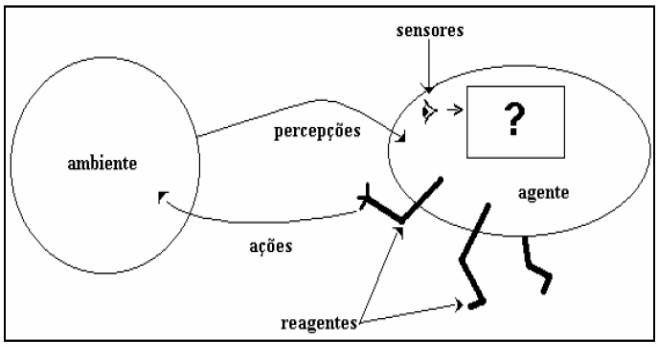

Figura 2.1: Esquema típico de um agente [RUS04]

Um agente é chamado de inteligente, quando, para cada seqüência de percepções possíveis, seleciona uma ação que venha maximizar sua medida de desempenho, dada a evidência fornecida pela seqüência de percepções e por qualquer conhecimento interno do agente [RUS04]. Um agente inteligente tem a capacidade de reagir racionalmente aos estímulos do ambiente no qual ele está inserido, sendo que seu grau de inteligência pode ser medido em função das ações que ele realiza. Neste contexto, inteligência é definida como a qualidade ou capacidade do agente de compreender e adaptar-se facilmente a diferentes situações.

Russel e Norvig [RUS95] afirmam que um agente computacional inteligente deve ter outros atributos que podem distingui-lo de meros programas, tais como operar sob controle autônomo, perceber seu ambiente, persistir por um período de tempo prolongado, adaptar-se a mudanças e ser capaz de assumir funções de outros agentes.

Wooldridge [WOO99] descreve algumas características típicas de agentes inteligentes:

- *Reação*: agentes devem perceber seu ambiente e responder oportunamente às mudanças que nele ocorrem;
- *Pró-atividade*: agentes não devem simplesmente atuar em resposta ao ambiente, devem exibir um comportamento oportunista e direcionado ao seu objetivo e tomar a iniciativa quando apropriado;
- *Sociabilidade*: agentes devem interagir, quando apropriado, com outros agentes artificiais ou humanos para completar suas próprias soluções de problemas ou ajudar outros com suas atividades.

E por fim, Bradshaw [BRA97] acrescenta ainda outras propriedades que os agentes devem possuir para se diferenciar de simples programas de computadores:

- *Capacidade de inferência*: agentes podem agir sobre especificações abstratas de tarefas utilizando um conhecimento anterior, conseguindo ir além das informações fornecidas, e devem possuir algum modelo de si próprio, de usuários, de situações de outros agentes;
- *Continuidade*: agentes que conseguem fazer persistir a sua identidade e estados durante longos períodos de tempo;
- *Adaptabilidade*: agentes são capazes de aprender e melhorar com a experiência;
- *Mobilidade*: agentes podem migrar de forma intencional de um determinado local para outro.

Assim, o objetivo de um agente é interagir com o ambiente para atingir seus objetivos devendo ter capacidade de reunir informações, tomar decisões baseadas nestas informações e iniciar uma ação especifica baseada nas decisões.

2.3. Agentes Autônomos

Agentes autônomos são sistemas computacionais que habitam em ambientes complexos e/ou dinâmicos e atuam de maneira autônoma no seu ambiente e, ao fazer isso, realizam um conjunto de objetivos ou tarefas para as quais foram designados [MAE95]. Essa característica de *autonomia* é definida como a capacidade do agente controlar suas próprias ações [FRA97]. As propriedades que permitem ao agente operar isoladamente, adicionadas às propriedades que o permitem decidir sobre o que fazer enquanto opera neste estado constituem a *independência*.

Agentes autônomos são conceituados como sistemas dotados da capacidade de interação independente e efetiva com o ambiente onde se encontra, regida pelo objetivo de concluir alguma tarefa [DAV95].

Um agente autônomo normalmente não é capaz de ter uma representação completa do mundo, vive num ambiente dinâmico e imprevisível e ainda deve fazer evoluir as suas competências. No entanto, certos agentes dotados de capacidade de aprendizagem e de adaptação podem melhorar o seu desempenho apesar destes obstáculos [ENE03]. Dessa forma, o agente deve almejar a execução da melhor ação possível processando as informações

provenientes dos sensores (dados de entrada), tomando alguma decisão e atuando no ambiente, modificando-o, para que suas metas sejam atingidas.

2.4. Agentes Autônomos Adaptativos

Um dos desafios da IA consiste em criar sistemas capazes de melhorar seu desempenho a partir de suas experiências. Esta capacidade de adaptação parece fundamental para os sistemas cujo comportamento é autônomo, pois também pode promover economia de recursos e aumentar a confiabilidade. As arquiteturas dos agentes autônomos que apresentam este comportamento são chamadas adaptativas [ENE03].

No início dos anos 90, certo cepticismo reinava sobre a utilização de mecanismos de aprendizagem em sistemas dinâmicos como aqueles dos agentes autônomos, devido à incipiência das pesquisas efetuadas no domínio da aprendizagem automática. Nesta época, algoritmos de aprendizagem necessitavam de recursos computacionais até então raros além de uma quantidade enorme de dados, como as redes neurais, por exemplo. Felizmente, as técnicas de aprendizagem e as tecnologias computacionais evoluíram. No entanto, nem todos os algoritmos de aprendizagem são indicados para um agente autônomo adaptativo, porque esses devem ter as seguintes características [ENE03]:

- a aprendizagem deve ser incremental;
- deve levar em conta o ruído;
- a aprendizagem não poderá ser supervisionada;
- eventualmente é necessário que o algoritmo permita a utilização de conhecimentos fornecidos pelo usuário e/ou por quem o desenvolveu.

Enembreck (2003) completa ainda, que certas formas de adaptação podem ser vistas como o aprendizado a partir da experiência. Neste caso, o agente irá melhorar à medida que o tempo passa. Isso significa que o agente deve aprender a escolher as boas ações nos bons momentos, com uma melhoria constante no mecanismo de seleção de ações.

2.5. Arquiteturas de Agentes

O principal problema a ser resolvido na pesquisa de agentes autônomos é o desenvolvimento de uma arquitetura de agentes que resulte em comportamento adaptativo, robusto e efetivo [MAE94]. *Comportamento adaptativo* significa que o agente melhora a

forma de alcançar seus objetivos durante o tempo; *comportamento robusto* significa que ele nunca falha totalmente e *efetivo* significa que o agente deve ser bem sucedido em alcançar seus objetivos.

Uma arquitetura de agente especifica como o agente pode ser decomposto para a construção de um conjunto de módulos e como estes devem interagir [WOO95]. O conjunto total dos módulos e suas interações proporcionam uma resposta para a questão de como os dados coletados e o estado interno do agente determinam as ações e futuros estados internos do mesmo. Uma arquitetura engloba técnicas e algoritmos que suportam esta metodologia. A arquitetura de um agente mostra como ele está implementado em relação às suas propriedades, sua estrutura e como os módulos que o compõem podem interagir, garantindo sua funcionalidade. Em suma, a arquitetura de um agente especifica sua estrutura e funcionamento.

Todas as propriedades que um agente apresenta devem estar implícitas na sua arquitetura. O conjunto de propriedades que um agente possui define o comportamento do agente. Como diferentes agentes podem possuir diferentes conjuntos de propriedades, têm-se então agentes com comportamentos diferentes, os quais são classificados de diferentes maneiras [BOG03].

Para que o sistema seja considerado agente, ele deve possuir algumas das seguintes características: autonomia, aprendizagem, comunicabilidade, comportamento adaptativo, confiabilidade, cooperatividade, degradação gradual, discurso, flexibilidade, inteligência, mobilidade, persistência, personalização, planejamento, pró-atividade ou orientação ao objetivo, reatividade, representatividade, responsabilidade, sociabilidade e veracidade [WOO96].

Quanto à forma de raciocínio, pelo menos três arquiteturas podem ser encontradas: Arquitetura deliberativa; Arquitetura reativa e Arquiteturas híbridas (utiliza ambas as abordagens). Várias aplicações destas arquiteturas podem ser encontradas em Wooldridge [WOO96]. A *arquitetura deliberativa* segue uma abordagem clássica e contém um modelo simbólico do mundo, onde as decisões são deliberadas pelo raciocínio lógico reconhecendo padrões existentes. Já a *arquitetura reativa* segue uma abordagem alternativa que suplanta limitações das arquiteturas simbólicas, pois reagem a eventos sem possuir um modelo simbólico formado do mundo [BRO86]; enquanto isso, as *arquiteturas híbridas* misturam os

componentes das arquiteturas deliberativas e reativas com o objetivo de torná-la mais adequada e funcional para a construção de agentes [WOO95].

Brooks [BRO86] foi o primeiro a propor uma arquitetura reativa chamada de arquitetura *subsumption*. Nos artigos de Brooks [BRO90] [BRO91a] [BRO91b] são apresentadas as principais idéias desta arquitetura como: (i) a inteligência real está situada no ambiente e não pode ser atingida por sistemas isolados do ambiente, como os sistemas especialistas e aqueles utilizados para provar teoremas; (ii) o comportamento inteligente é uma propriedade emergente de certos sistemas complexos que surge como resultado da interação de vários agentes com o ambiente e; (iii) um comportamento inteligente pode ser gerado sem uma representação explícita da realidade como propõe a IA simbólica tradicional. Para demonstrar a validade de suas idéias, Brooks construiu um robô cujo comportamento poderia ser considerado impressionante se tivesse sido realizado por sistemas de IA simbólicos.

2.6. Técnicas de Aprendizagem para Agentes

O aprendizado permite ao agente operar em ambientes inicialmente desconhecidos e se tornar mais competente do que seu conhecimento inicial sozinho poderia permitir [RUS04]. Agentes autônomos aprendem por tentativa e erro ao atuar sobre um ambiente dinâmico ou estático [SUT98]. Desta maneira, não é necessária uma entidade externa que forneça exemplos ou um modelo a respeito da tarefa a ser executada: a única fonte de aprendizado é a própria experiência do agente, cujo objetivo formal é adquirir uma política de ações que maximize seu desempenho geral.

Aprendizagem é a capacidade que um agente deve possuir para executar uma tarefa com maior eficiência do que em execuções anteriores. Sem a capacidade de aprendizagem o agente reagirá sempre da mesma maneira para um mesmo ambiente e uma mesma situação. O aprendizado em um agente pode ser dividido em quatro componentes conceituais [RUS04]:

- elemento de aprendizado;
- elemento de desempenho;
- um crítico e
- um gerador de problemas.

A distinção mais importante se dá entre o elemento de aprendizado e o elemento de desempenho. O elemento de desempenho é responsável pela execução de aperfeiçoamentos, isto é, ele utiliza realimentação do crítico sobre como o agente está funcionando e determina de que maneira o elemento de desempenho deve ser modificado para funcionar melhor no futuro. Já o elemento de desempenho é responsável pela seleção de ações externas, ou seja, ele recebe percepções e decide sobre ações.

Neste caso, o crítico é responsável por informar ao elemento de aprendizado como o agente está se comportando em relação a um padrão fixo de desempenho; isso é necessário porque as próprias percepções não oferecem nenhuma indicação do sucesso do agente. O gerador de problemas é responsável por sugerir ações que levarão à experiências novas e informativas, podendo sugerir execução de ações que talvez não fossem ótimas a curto prazo, mas que poderia levar a ações muito melhor a longo prazo.

Um agente autônomo adaptativo é dito inteligente se suas ações o levam a um estado melhor do que o anterior. Segundo Russell e Norvig [RUS04] para um agente ser racional ele depende de quatro fatores: (i) da medida de desempenho que define o critério de sucesso; (ii) do conhecimento anterior que o agente tem do ambiente; (iii) das ações que o agente pode executar; (iv) da seqüência de percepções do agente até o momento.

Para saber se um agente é racional, usa-se uma medida de desempenho que mede o sucesso do comportamento do agente. Quando um agente é inserido em um ambiente, ele gera uma seqüência de ações de acordo com as percepções que recebe. Se a seqüência das ações é desejada, isso quer dizer que o agente funcionou bem. Assim, não existe uma medida de desempenho fixa apropriada para todos os agentes, ficando a critério de cada projetista definir uma medida de desempenho ao agente [RUS04].

Neste projeto será usado como medida de desempenho a quantidade de interações necessárias até o agente conquistar seu objetivo, acrescentando o custo acumulado da situação de cada estado caracterizado como a seqüência de ações que permite ao agente alcançar o objetivo.

A seguir são descritos sucintamente algumas das principais técnicas de aprendizagem de agentes.

2.6.1. Aprendizagem por Modelos

Os agentes que utilizam a aprendizagem por modelo tentam encontrar relações de causa e efeito entre suas ações e os acontecimentos do ambiente. Em geral utilizam tanto os modelos probabilistas como os modelos lógicos nesta abordagem [ENE03].

Maes *apud* [ENE03], propôs uma rede de comportamento como modelo do procedimento de seleção das ações de um agente autônomo num ambiente dinâmico e imprevisível. Na rede de comportamento, cada nó representa um comportamento (ação), sendo composto por pré-condições, pós-condições, uma lista que adiciona predicados, uma lista que suprime predicados e uma energia de ativação. A partir de cada nó, três tipos de relações podem ocorrer:

- *antecessor*: de um nó para outros nós que satisfazem pré-condições do nó de origem. A ação deve ter pelo menos dois antecessores;
- *conflitante*: de um nó para outros nós que tornam impossível a satisfação das pré-condições. A ação deve ter uma relação conflituosa;
- *sucessor*: uma relação na direção contrária à relação antecessora.

O objetivo da rede é acumular uma energia em nós que representam as ações adequadas. A cada ciclo, o processo de transmissão de energia é realizado do seguinte modo: (i) todos os nós realizáveis (pré-condições satisfeitas) transmitem uma ativação aos sucessores; (ii) todos os nós não realizáveis transmitem uma ativação aos nós antecessores que tornam uma pré-condição falsa em verdadeira; (iii) todos os nós (realizáveis ou não) diminuem a ativação dos nós conectados por relações conflituosas. A rede guarda também um limiar mínimo que os nós devem respeitar. Quando um nó excede o limiar mínimo ele é suprimido. Ao final de um tempo a rede deve guardar apenas as ações que normalmente são dependentes umas das outras [ENE03]. A principal vantagen desta abordagem, conforme Mães, é que a rede pode modelar mudanças de objetivos do agente, trabalhando também em ambientes quantitativos e de ações múltiplas. Contudo, devido à propagação de valores de ativações, a tomada de decisão tende a exigir muito tempo de processamento.

2.6.2. Aprendizagem Baseada em Regras

Os primeiros sistemas classificadores utilizados em agentes autônomos são os sistemas baseados em regras, como por exemplo, na abordagem de Holland *apud* [ENE03].

Nestes sistemas, o projetista deve criar um conjunto de regras que descreve o comportamento do agente. Cada regra relaciona um conjunto de condições ou premissas a uma ação. Para que uma regra seja aplicável, as premissas devem ser verdadeiras no estado corrente do ambiente. Para aprender, o agente escolhe um conjunto de regras aplicáveis no estado corrente. Em seguida, utiliza uma medida de qualidade ligada a cada regra para escolher a melhor. Normalmente, o agente deve ter uma recompensa em função da sua ação, como nos sistemas de AR, aumentando a qualidade de todas as regras selecionadas previamente. Conseqüentemente, à medida que o tempo passa a prioridade das regras muda.

2.6.3. Aprendizagem por Reforço

A AR consiste em um paradigma computacional de aprendizagem em que um agente aprendiz procura maximizar uma medida de desempenho baseada nos reforços (recompensas ou punições) que recebe ao interagir com um ambiente desconhecido [RIB99]. AR não é definido como um conjunto de algoritmos de aprendizagem, mas como uma classe de problemas de aprendizagem. Todo algoritmo que resolver bem esse problema será considerado um algoritmo de AR [SUT98].

Devido este trabalho ser baseado em técnicas de AR, este paradigma é explanado em detalhes no próximo capítulo.

2.6.4. Aprendizagem Baseada em Instâncias

Os métodos de aprendizagem baseado em instâncias elaboram hipóteses diretamente a partir das próprias instâncias de treinamento [HAR68]. Existem diversos algoritmos que representam a aprendizagem baseada em instâncias como a família IBL (*Instance-based Learning*). Esses algoritmos são descendentes do algoritmo *Nearest Neighbour* (NN) (vizinho mais próximo), cuja principal técnica se baseia em classificar uma nova instância considerando a classe do seu vizinho mais próximo [AHA91].

Os algoritmos de aprendizado da família IBL armazenam instâncias de treinamento na memória como pontos em um espaço *n*-dimensional, definido pelos *n* atributos que as descrevem, e nunca mudam a representação desses pontos [AHA91].

Este método de aprendizagem é utilizado neste projeto na intenção de descobrir novas políticas de ações estimando os valores gerados durante a fase de aprendizagem pelo agente

Q-Learning. Assim, os algoritmos de aprendizagem baseadas em instâncias são descritos em detalhe no Capítulo 4.

2.7. Aprendizagem em Sistemas Multi-Agente

A aprendizagem multi-agente diferentemente da aprendizagem em agentes autônomos adaptativos, supõem que o conhecimento relevante não está disponível localmente em um único agente [MOD01]. Na aprendizagem multi-agente, os agentes aprendem a realizar uma tarefa que envolve mais do que um agente na sua execução.

Agentes de um SMA devem possuir algumas capacidades específicas para interagirem num mesmo ambiente. Eles devem ter conhecimento da sua existência e da existência dos outros agentes no ambiente e devem ser capazes de comunicar, possuindo para tanto, uma linguagem e um meio de comunicação específica. Cada agente deverá possuir conhecimentos e habilidades para executar uma determinada tarefa e devem, *a priori*, cooperar para atingir um objetivo global.

A existência do agente no SMA precede e independe da existência de um problema global. Embora participe de uma comunidade, o agente é visto como uma entidade autônoma, direcionada por motivações e objetivos locais próprios [SAN97]. O ponto central nesta linha de pesquisa refere-se ao comportamento do agente em sociedade e de como este se posiciona tendo em vista concretizar objetivos de interesse individual ou coletivo. O que se busca em última instância é dotar um agente autônomo de estruturas e mecanismos de inferência e decisão que o capacitem, de forma descentralizada, a exibir um comportamento socialmente racional.

Segundo Gerhard [WEI96] a aprendizagem em sistemas multi-agente pode ser dividida em duas principais áreas: a aprendizagem isolada e a aprendizagem coletiva. Na *aprendizagem isolada*, o processo de aquisição do conhecimento pelo agente ocorre sem a influência dos demais agentes ou qualquer outro elemento da sua sociedade. Já na *aprendizagem coletiva*, o processo de aquisição do conhecimento tem influência direta com todos os elementos da sociedade em que o agente está inserido.

A aprendizagem em SMA está relacionada com a solução cooperativa de problemas dentro de um certo ambiente (BIT01). Esta tarefa está diretamente ligada ao aprendizado, onde muitos sistemas são distribuídos e deveriam ser adaptáveis ao ambiente que está em

constante mudança. A necessidade de integrar técnicas de aprendizagem nas arquiteturas de SMA tem fundamentado varias linhas de pesquisa na área [MAE95].

Stone e Veloso [STO96] completam ainda que, se um agente está aprendendo a conquistar habilidades para interagir com outros agentes em seu ambiente, e independentemente se os outros agentes estão ou não aprendendo simultaneamente, está aprendizagem é considerada aprendizagem multi-agente. Então, aprendizagem multi-agente inclui algumas situações na qual o agente aprende interagindo com outros agentes, mesmo se o comportamento dos outros agentes é estático.

Alguns dos principais trabalhos com aprendizagem multi-agente estão relacionados diretamente em *Robot soccer* (futebol de robôs) [KIT97a] [KIT97b] [VEL99] [VEL02] [WEL03] [HE06]. É importante ressaltar que neste trabalho não serão utilizadas técnicas de aprendizagem para SMA.

2.8. Agentes Adaptativos e Aprendizagem por Reforço

Em AR, um agente adaptativo interage com o ambiente em intervalos discretos de tempo em um ciclo de *percepção-ação*. A maneira mais tradicional para formalizar o AR é utilizando o conceito de MDP (Processos Decisórios de Markov), que pode ser definido formalmente por um conjunto de estados do ambiente; um conjunto de ações que o agente pode tomar; uma função de transição de estado; e uma função de reforço [MIT97].

Assim, no ciclo percepção–ação, o agente observa, a cada passo de iteração, o estado corrente s_t do ambiente e escolhe a ação a_t para executar. Ao executar esta ação a_t, o agente recebe um sinal escalar de reforço $r_{s,a}$ (punição ou recompensa), que indica quão desejável é o estado resultante s_{t+1}. Resolver um MDP consiste em computar a política $\pi : SxA$ que maximiza (ou minimiza) alguma função, geralmente a recompensa recebida (ou o custo esperado), ao longo do tempo.

Tido como o mais popular algoritmo de AR, o algoritmo Q–Learning foi proposto como uma maneira de aprender iterativamente a política ótima π^* quando o modelo do sistema não é conhecido [WAT92]. O algoritmo Q–Learning propõe que o agente aprenda uma função Q de recompensa esperada com desconto, conhecida como função valor–ação. Ele aproxima iterativamente $\hat{Q}$, a estimativa de $Q^*(s,a)$ no instante t, utilizando a seguinte regra de aprendizado apresentado na Equação 2.1:

$$\hat{Q}_{t+1}(s_t, a_t) \leftarrow \hat{Q}_t(s_t, a_t) + \alpha\left[r(s_t, a_t) + \gamma \max_{a_{t+1}} \hat{Q}_t(s_{t+1}, a_{t+1}) - \hat{Q}_t(s_t, a_t) \right] \quad (2.1)$$

na qual 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 $(0 \leq \gamma \leq 1)$; α é a taxa de aprendizagem $(0 \leq \alpha \leq 1)$.

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 ou investigação. Uma estratégia para a escolha das ações bastante utilizada em implementações do Q–Learning é a exploração aleatória que difere da $\varepsilon - Greedy$, na qual o agente executa a ação com o maior valor de Q com probabilidade $1 - \varepsilon$ e esta ação é escolhida aleatoriamente com probabilidade ε. Neste caso, a transição de estados é dada pela seguinte regra de transição de estados, conforme a Equação 2.2 [BIA04]:

$$\pi(s_t) = \begin{cases} a_{\,random} & se\ \ q > \varepsilon, \\ \arg\max_{at} \hat{Q}_t(s_t, a_t) & caso\ contrário \end{cases} \quad (2.2)$$

na qual q é um valor escolhido de maneira aleatória com distribuição de probabilidade uniforme sobre [0,1] e ε $(0 \leq \varepsilon \leq 1)$ é o parâmetro que define a taxa de exploração ou investigação; quanto menor o valor de ε, menor a probabilidade de se fazer uma escolha aleatória e a_{random} é uma ação aleatória selecionada entre as ações possíveis de serem executadas no estado s_t.

2.9. Aplicações de Agentes Autônomos Adaptativos

As aplicações com agentes inteligentes aparecem em diversas áreas, porém isso não implica que todos os problemas e sistemas sejam solucionados com agentes.

Segundo Reis [REI03], dentre as aplicações utilizando agentes destacam-se:

- Aplicações Industriais: aplicações na área das telecomunicações, controles de fábricas e controles de processos, distribuição de energia elétrica, controle de tráfego aéreo e sistemas de transportes;
- Agentes de Pesquisa de Informação: neste contexto, as aplicações mais conhecidas destacam-se a gestão de correios eletrônicos, recuperação e filtragem de informação na internet, filtragem de notícias e agendamento distribuído de reuniões;
- Comércio Eletrônico: Aplicações em mercados eletrônicos para B2C (*business-to-consumer*) e para B2B (*business-to-business*);
- Aplicações de Entretenimento: desenvolvimento de aplicações na área dos jogos e lazer. Dentre as aplicações de agentes neste domínio destacam-se os jogos, o desenvolvimento de personagens virtuais e histórias interativas;
- Aplicações Médicas: aplicações de agentes em controles de robôs e outros equipamentos hospitalares e sistemas de tratamento de pacientes distribuídos;
- Agentes para Simulação: a utilização de agentes em simuladores encontra-se cada vez mais generalizadas. As áreas de aplicações neste contexto incluem desde simuladores de condução, vôo, combate aéreo, futebol, produção e manufatura e diversas áreas da robótica;
- Controle de Robôs: a utilização de agentes no controle de robôs destina-se a obter uma navegação segura e eficiente e um funcionamento global versátil do robô. Na maioria das aplicações, o controle de robôs efetuado diretamente por humanos é, na prática, impossível e como tal o recurso a agentes autônomos é a solução mais aconselhável;
- Competições de Investigação Científica: uma área em grande expansão atualmente é a área das competições científicas. Neste tipo de competições, a investigação realizada por investigadores de uma dada área é comparada utilizando um problema ou plataforma comum. Dentre as competições mais significativas, merece destaque o RoboCup – Futebol Robótico [KIT97a] [KIT97b] [VEL02] e o TAC (Trade Agent Competition) [WEL03] [HE06].

2.10. Considerações Finais

Observou-se neste capítulo que muitas definições são atribuídas ao termo agente. Os agentes inteligentes são capazes de atuar de maneira autônoma em ambientes desconhecidos e/ou complexos, adaptando-se às tarefas para os quais foram designados, a fim de satisfazer seus objetivos. Os termos autonomia e adaptação são as características mais importantes dos agentes. Autonomia é a capacidade do agente executar o controle sobre suas próprias ações e adaptação é a capacidade de melhorar seu comportamento em função de experiências anteriores. Essas características formam os agentes autônomos adaptativos. A aprendizagem de um agente se dá através de tentativa e erro ao atuar sobre um ambiente. Assim, a fonte de aprendizagem do agente é a própria experiência, cujo objetivo formal é adquirir uma política de ações que maximize seu desempenho geral. Por fim, observou-se que diversas aplicações no uso de agentes vêm ocorrendo em diferentes áreas de estudos.

No próximo capítulo serão apresentados os principais conceitos sobre AR.

Capítulo 3

Princípios de Aprendizagem por Reforço

3.1. Considerações Iniciais

No capítulo anterior foi possível observar que a AR é um paradigma de aprendizagem que tenta solucionar problemas onde um agente recebe do ambiente um retorno por suas ações. Este tipo de aprendizagem vem sendo utilizado nesses últimos anos por diversos pesquisadores [GER95] [CRI96] [KAE96] [LIT96] [SUT98] [RIB99] [POR05] a fim de encontrar soluções para problemas de aprendizagem com o uso de agentes. É observado que a AR se difere de sistemas baseados em aprendizado supervisionado, devido à ausência da apresentação de pares de entrada e saída, presentes em sistemas supervisionados. Em sistemas baseados em reforços, a avaliação do sistema acontece simultaneamente ao aprendizado e precisa explicitamente explorar todo seu ambiente.

Um excelente método para resolver problemas que envolvam AR é o algoritmo Q-Learning proposto por Watkins [WAT92]. Este método busca iterativamente uma maneira de aprender uma política ótima quando o modelo do sistema não é conhecido.

Uma maneira de formalizar o AR é utilizando conceitos de Processos Decisórios de Markov. Esse formalismo inclui um conjunto de estados do ambiente, um conjunto de ações do agente, um conjunto de transições de estados e uma função de recompensas.

3.2. Origens

A AR é inspirada na aprendizagem infantil humana e na aprendizagem de animais. Uma criança costuma realizar ações aleatórias e de acordo com as respostas de seus pais (elogios ou reclamações), aprende quais daquelas ações são boas e quais são ruins.

Pavlov, um cientista russo, publicou em 1903 um artigo chamado “reflexo condicional” e a sua experiência ficou mundialmente conhecida como “o cachorro de Pavlov” [FON00]. No seu trabalho, ele tocava um sino toda vez que dava comida ao seu cachorro. Com o passar do tempo o cão começou a associar o som do sino com comida aprendendo que o som do sino estava relacionado com comida. Seguindo em sua pesquisa Pavlov ”ensinou“ ao cão que alguns dos sinais eram bons e outros ruins, então o cão começou evitar os “sinais ruins” e aumentou o seu interesse pelos ”sinais bons”. Esta é a essência do AR, ou seja, atribuir recompensas ao sistema quando a sua ação é boa e punições quando a ação do sistema resulta em algo indesejável.

3.3. Definições

A AR é um paradigma computacional de aprendizagem em que um agente aprendiz procura maximizar uma medida de desempenho baseada nos reforços (recompensas ou punições) que recebe ao interagir com um ambiente desconhecido [RIB99].

O agente atua em um ambiente formado por um conjunto de possíveis estados, e pode escolher ações dentro de um conjunto de ações possíveis, indicando o valor imediato da transição de estado resultante [RIB99]. A tarefa do agente é aprender uma política de controle (seqüência de ações) que maximize a soma esperada destes reforços, descontando (usualmente de modo exponencial) as recompensas ou punições proporcionalmente ao seu atraso temporal.

No problema de AR tem-se um agente, que atua em um ambiente. O agente percebe um conjunto discreto S de estados, e pode realizar um conjunto discreto A de ações. A cada instante de tempo t, o agente pode detectar seu estado atual s, e, de acordo com esse estado, escolher uma ação a ser executada, que o levará para um outro estado s'. Para cada par estado/ação, (s,a), há um sinal de reforço, $R(s,a) \rightarrow \Re$, que é dado ao agente quando ele executa a ação a no estado s. O problema da AR é ilustrado na Figura 3.1.

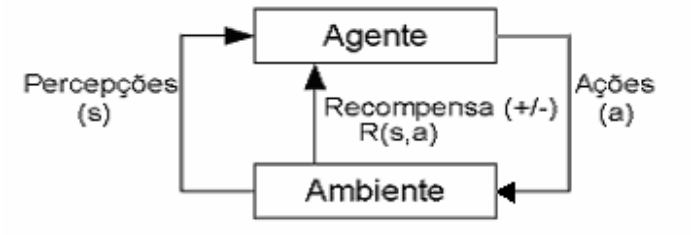

Figura 3.1: Problema da AR [SUT92]

O sinal de reforço é a base do aprendizado do agente, pois o reforço deve indicar o objetivo a ser alcançado pelo agente. O agente receberá uma recompensa positiva caso o seu novo estado seja melhor do que o seu estado anterior. Com isso, o reforço está mostrando ao agente que a sua meta é maximizar recompensas até o seu estado final.

Assim, o objetivo do método é levar o agente a escolher a seqüência de ações que tendem a aumentar a soma de valores de reforço, ou seja, é encontrar a política ótima, definida como o mapeamento de estados em ações que maximize as medidas do reforço acumuladas no tempo.

3.4. Características da Aprendizagem por Reforço

Algumas características que diferenciam a AR de outros métodos são descritas a seguir [SUT98]:

Aprendizado pela Interação: essa é a principal característica que define um problema de AR, onde um agente age no ambiente e aguarda pelo valor de reforço que o ambiente deve informar como resposta frente a ação tomada, assimilando através do aprendizado o valor de reforço obtido para tomar decisões posteriores.

Retorno Atrasado: um máximo valor de reforço que o ambiente envia para o agente não quer dizer necessariamente que a ação tomada pelo agente foi a melhor. Uma ação é produto de uma decisão local no ambiente, sendo seu efeito imediato de natureza local, enquanto que em um sistema de AR, busca-se alcançar objetivos globais no ambiente. Assim as ações tomadas devem levar a maximizar o retorno total, isto é, a qualidade das ações tomadas é vista pelas soluções encontradas em longo prazo.

Orientado a Objetivo: em AR, o problema tratado é considerado como um ambiente que dá respostas frente ações efetuadas, não sendo necessário conhecer detalhes da modelagem desse ambiente. Simplesmente, existe um agente que age dentro do ambiente desconhecido tentando alcançar um objetivo. O objetivo é, geralmente, otimizar algum comportamento dentro do ambiente.

Investigação x Exploração: em AR, agentes presenciam o dilema conhecido na literatura como "*the Exploration x Exploitation dilemm",* que consiste em decidir quando se deve aprender e quando não se deve aprender sobre o ambiente, mas usar a informação já obtida até o momento. Para que um sistema seja realmente autônomo, esta decisão deve ser tomada pelo próprio sistema.

A decisão é fundamentalmente uma escolha entre agir baseado na melhor informação de que o agente dispõe no momento ou agir para obter novas informações sobre o ambiente que possam permitir níveis de desempenho maiores no futuro. Isto significa que o agente deve aprender quais ações maximizam os valores dos ganhos obtidos no tempo, mas também, deve agir de forma a atingir esta maximização, explorando ações ainda não executadas ou regiões pouco visitadas do espaço de estados. Como ambas as formas trazem, em momentos específicos, benefícios à solução dos problemas, uma boa estratégia é mesclar estas formas [SUT98].

3.5. Elementos Fundamentais da Aprendizagem por Reforço

Conforme apresentado em Sutton e Barto [SUT98], o problema da AR apresenta cinco partes fundamentais a serem consideradas: o ambiente; a política; o reforço e o retorno; a função de reforço e a função valor-estado, descritos assim:

O Ambiente: todo sistema de AR aprende um mapeamento de situações e ações por experimentação em um ambiente. O ambiente no qual está inserido o sistema, deve ser pelo menos parcialmente observável através de sensores ou descrições simbólicas. Também é possível, entretanto, que toda informação relevante do ambiente esteja perfeitamente disponível. Neste caso, o agente poderá escolher ações baseadas em estados reais do ambiente.

A Política: uma política expressa pelo termo π, representa o comportamento que o sistema AR segue para alcançar o objetivo. Em outras palavras, uma política π é um mapeamento de estados s e ações a em um valor $\pi(s,a)$. Assim, se um agente AR muda a sua política, então as probabilidades de seleção de ações sofrem mudanças e conseqüentemente, o comportamento do sistema apresenta variações à medida que o agente vai acumulando experiência a partir das interações com o ambiente. Portanto, o processo de aprendizado no

sistema AR pode ser expresso em termos da convergência até uma política ótima $\pi^*(s,a)$ que conduza à solução do problema de forma ótima.

Reforço e Retorno: o reforço é um sinal do tipo escalar r_{t+1}, que é devolvido pelo ambiente ao agente assim que uma ação tenha sido efetuada e uma transição de estado $s_t \rightarrow s_{t+1}$ tenha ocorrido. Existem diferentes formas de definir o reforço para cada transição no ambiente, gerando-se funções de reforço que, intrinsecamente, expressam o objetivo que o sistema AR deve alcançar. O agente deve maximizar a quantidade total de reforços recebidos chamado de retorno, que nem sempre significa maximizar o reforço imediato a receber, mas o reforço acumulado durante a execução total.

De modo geral, o sistema AR busca maximizar o valor esperado de retorno, com isso, o retorno pode ser definido como uma função da seqüência de valores de reforço até um tempo T final. No caso mais simples é um somatório como aparece na Equação 3.1.

$$R = \sum_{k=0}^{T} r_{t+k+1} \tag{3.1}$$

Em muitos casos a interação entre agente e ambiente não termina naturalmente em um episódio (seqüência de estados que chegam até o estado final), mas continua sem limite, como por exemplo, em tarefas de controle contínuo. Para essas tarefas a formulação do retorno é um problema, pois $T = \infty$ e o retorno que se deseja também tenderá a infinito $R_T = \infty$.

Para este problema foi criada a taxa de amortização γ, a qual determina o grau de influência que têm os valores futuros sobre o reforço total. Assim, a expressão do retorno aplicando taxa de amortização é expressa pela Equação 3.2:

$$R = \sum_{k=0}^{\infty} y^k r_{t+k+1} \tag{3.2}$$

na qual, $0 \leq \gamma \leq 1$. Se $\gamma \rightarrow 0$, o agente tem uma visão baixa dos reforços, maximizando apenas os reforços imediatos. Se $\gamma \rightarrow 1$, a visão do reforço abrange todos os estados futuros

dando maior importância ao estado final, desde que a seqüência R seja limitada. Um sistema de AR faz um mapeamento de estados em ações baseado nos reforços recebidos.

Assim, o objetivo do AR é definido usando-se o conceito de função de reforço, a qual é uma função dos reforços futuros que o agente procura maximizar. Ao maximizar essa função, o objetivo será alcançado de forma ótima. A função de reforço define quais são os bons e maus eventos para os agentes.

Função de Reforço: as funções de reforço podem ser bastante complexas, porém existem pelo menos três classes de problemas freqüentemente usados para criar funções adequadas a cada tipo de problema, descritas assim:

i) *Reforço só no estado final:* Nesta classe de funções, as recompensas são todas zeros, exceto no estado final, em que o agente recebe uma recompensa real (ex: +1) ou uma penalidade (ex: -1). Como o objetivo é maximizar o reforço, o agente irá aprender que os estados correspondentes a uma recompensa são bons e os que levaram a uma penalidade devem ser evitados.

ii) *Tempo mínimo ao objetivo:* Funções de reforço nesta classe fazem com que o agente realize ações que produzam o caminho ou trajetória mais curta para um estado final. Toda ação tem penalidade (-1), sendo que o estado final é (0). Como o agente tenta maximizar valores de reforço, ele aprende a escolher ações que minimizam o tempo que leva a alcançar o estado final.

iii) *Minimizar reforços:* Nem sempre o agente precisa ou deve tentar maximizar a função de reforço, podendo também aprender a minimizá-las. Isto é útil quando o reforço é uma função para recursos limitados e o agente deve aprender a conservá-los ao mesmo tempo em que alcança o estado final.

Função Valor-Estado: define-se uma função valor-estado como o mapeamento do estado, ou par estado-ação, em um valor que é obtido a partir do reforço atual e dos reforços futuros. Se a função valor-estado considera só o estado s é indicada como $V(s)$, por outro lado, se é considerado o par estado-ação (s,a), então a função valor-estado é denotada como função valor-ação $Q(s,a)$.

3.6. Processos Markoviano

A maneira mais tradicional para formalizar a AR é utilizando o conceito de Processos Decisórios de Markov (*Markov Decision Process* - MDP). Por ser matematicamente bem estabelecido e fundamentado, este formalismo facilita o estudo da 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 [RIB02].

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 do sistema independe de 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.

Um MDP 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 pela quádrupla $< S, A, T, R >$ [LIT94] [MIT97], onde:

- S : é um conjunto finito de estados do ambiente;
- A : é um conjunto finito de ações que o agente pode realizar;
- $T : SxA \rightarrow \Pi(S)$: é a função de transição de estado, onde $\Pi(S)$ é uma distribuição de probabilidades sobre o conjunto de estados S, $T(s_{t+1}, s_t \mid a_t)$ define a probabilidade de realizar a transição do estado s_t para o estado s_{t+1} quando se executa a ação a_t ;
- $R : SxA \rightarrow \Re$: é a função de recompensas, que especifica a tarefa do agente, definindo a recompensa recebida por um agente ao selecionar a ação a estando no estado s .

Usando o MDP como formalismo, pode-se reescrever o objetivo do agente que aprende por reforço como: aprender a política $\pi^* : SxA$ 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 (KAE96).

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). POMDP são extremamente importantes para soluções de problemas práticos, nos quais as observações são feitas com sensores imprecisos, ruidosos ou poucos 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 ao gol pode entrar, pode bater no travessão ou pode ir para fora do campo. Outro exemplo é do lançamento de uma moeda, no qual dois resultados são possíveis.

3.7. Principais Algoritmos de Aprendizagem por Reforço

AR dispõe de vários métodos de aprendizagem. A seguir, são descritos brevemente alguns dos algoritmos mais comuns na literatura.

3.7.1. Algoritmo R-Learning

A técnica proposta por Schwartz [SCH93], chamada de R-Learning, maximiza a recompensa média a cada passo, ou seja, utiliza o modelo da recompensa média. O algoritmo R-Learning possui regra similar ao Q-Learning, sendo baseado na dedução de valores $R(s,a)$, e devendo escolher ações a em um estado s. A cada situação, o agente escolhe a ação que tem o maior valor R, exceto que em algumas vezes ele escolhe uma ação qualquer. Os valores de R são ajustados a cada ação, baseado na seguinte regra de aprendizagem, conforme indica a Equação 3.3.

$$R(s,a) \leftarrow (1-\alpha)R(s,a) + \alpha[r - \rho + eR(s')] \tag{3.3}$$

Esta regra difere da regra do Q-Learning, simplesmente por subtrair a recompensa média ρ do reforço imediato r e por não ter desconto γ para o próximo estado, $eR(s') = \max_a R(s',a')$. A recompensa média é calculada como:

$$\rho \leftarrow (1-\beta)\rho + \beta[r + eR(s') - eR(s)] \tag{3.4}$$

o ponto chave da Equação 3.4 é que ρ somente é atualizado quando uma ação não aleatória foi tomada, ou seja, $\max_a R(s,a) = R(s,a)$. A recompensa média ρ não depende de um estado particular, ela é uma constante para todo o conjunto de estados. A Figura 3.2 apresenta o algoritmo R-Learning, no qual se podem observar pequenas reestruturações nas equações de atualização de R e ρ, que melhoram o custo computacional.

Inicialize ρ e $R(s,a)$ arbitrariamente
Repita até condição de parada ser satisfeita:
  $s \leftarrow$ estado atual
  Escolha $a \in A(s)$
  Execute a ação *a*
  Observe os valores *s'* e *r*
  $R(s,a) \leftarrow R(s,a) + \alpha[r - \rho + \max_{a'} R(s',a') - R(s,a)]$
  se $R(s,a) = \max_a R(s,a)$ então
    $\rho \leftarrow \rho + \beta[r - \rho + \max_{a'} R(s',a') - \max_a R(s,a)]$

Figura 3.2: Algoritmo R-Learning [FAR00]

3.7.2. Algoritmo H-Learning

O algoritmo H-Learning foi proposto por Tadepalli e Ok [TAD94] na tentativa de otimizar a recompensa média sem utilizar descontos. O algoritmo H-Learning estima as probabilidades $P(s'|s,a)$ e os reforços $R(s,a)$ por contagem direta e atualiza os valores da recompensa esperada h utilizando a Equação 3.5, que segundo teorema provado por Bertsekas [BER87], converge para uma política ótima.

$$h(s) = \max_{u \in A(s)} \{r(s,a) + \sum_{s'=1}^{n} P(s'|s,a)h(s')\} - \rho \tag{3.5}$$

O algoritmo H-Learning pode ser observado na Figura 3.3:

Algoritmo *H-Learning*();

1 Se a estratégia de exploração sugere uma ação aleatória,
Então pegue uma ação aleatória para i,
Senão execute a ação $a \in_{best} (i)$

2 Faça k ser o estado resultante e r' a recompensa imediata recebida.

3 $N(i,a) \leftarrow N(i,a)+1$;

4 $N(i,a,k) \leftarrow N(i,a,k)+1$;

5 $P_{ik}(a) \leftarrow N(i,a,k)/N(i,a)$;

6 $r(i,a) \leftarrow r(i,a)+(r'-r(i,a))/N(i,a)$;

7 Se a ação executada $a \in_{best} (i)$ então
$T \leftarrow T+1$;
$\rho \leftarrow \rho+(r'-h(i)+h(k)-\rho)/T$;

8 Faça $H(i,u) = r(i,u) + \sum_{j=1}^{n} p_{ij}(u)h(j)$:
$U_{best}(i) \leftarrow \{\upsilon \mid H(i,\upsilon) = \max_{u \in U(i)} H(i,u)\}$;
$h(i) \leftarrow H(i,a) - \rho$, onde $a \in_{best} (i)$;
$i \leftarrow k$;
Até convergir ou MAX-Steps vezes;

9 Fim.

Figura 3.3: Algoritmo H-Learning [FAR99]

no qual, $N(i,u)$ é o número de vezes que a ação u foi executada no estado i e $N(i,u,j)$ o número de vezes que ela resultou no estado j, $p_{ij}(u)$ é a probabilidade de ir de um estado i para um estado k executando a ação u, $r(i,a)$ é a recompensa estimada por executar a ação a no estado i, $h(i)$ é a recompensa máxima esperada para o estado i e corresponde ao $eQ(s')$ no algoritmo Q-Learning, T é o número total de passos que uma ação aparentemente ótima foi executada e é inicializada com 0.

Todos os métodos em AR, exceto o H-Learning, tem um ou mais parâmetros, como por exemplo, o Q-Learning tem α e γ e o R-Learning tem α e β. A performance de todos estes algoritmos é sensível a estes parâmetros, e consequentemente fica necessário ajustá-los para obter um melhor desempenho [FAR99].

3.7.3. Algoritmo $Q(\lambda)$

O algoritmo $Q(\lambda)$ proposto por Peng e Williams [PEN96], é caracterizado por ser uma adaptação de uso de traços de elegibilidade para o algoritmo Q-Learning.

Traços de elegibilidade são registros temporários da ocorrência de um evento, como por exemplo, visita a um estado ou a execução de uma ação. O traço marca os parâmetros de memória associados aos eventos como estados elegíveis para passar por mudanças no aprendizado. Quando um passo de aprendizado ocorre, apenas os estados ou ações elegíveis recebem o crédito pela recompensa ou a culpa pelo erro.

Do ponto de vista teórico, traços de elegibilidade são como uma ponte entre os métodos de Monte Carlo [RUB81] e de Diferenças Temporais [SUT98], onde se enquadram os algoritmos Q-Learning e o Sarsa. Quando métodos de diferenças temporais são incrementados com traços de elegibilidade, eles produzem uma família de métodos atravessando um espectro que tem métodos de Monte Carlo em uma ponta e métodos de Diferenças Temporais na outra [RIB99]. Neste intervalo estão métodos que herdam vantagens de ambos os extremos, frequentemente apresentando melhor desempenho.

Métodos de Monte Carlo podem apresentar vantagens para lidar com processos não-*Markovianos*, porque não atualizam estimativas baseados em valores estimados anteriormente. A principal desvantagem destes métodos é a sua pesada carga computacional. Métodos usando traços de elegibilidade buscam, portanto, combinar a vantagem da rapidez relativa de aprendizado dos métodos de Diferenças Temporais e a capacidade de lidar com reforços atrasados ou observabilidade parcial dos métodos Monte Carlo [MON04].

3.7.4. Algoritmo Sarsa

O algoritmo Sarsa é uma modificação do algoritmo Q-Learning que utiliza um mecanismo de iteração de política [SUT98]. A função de atualização do algoritmo Sarsa obedece a Equação 3.6.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)] \tag{3.6}$$

A forma procedimental do algoritmo Sarsa é similar a do algoritmo Q-Learning. Idealmente, o algoritmo Sarsa converge para uma política e valor de função de ação ótima. Assim que todos os pares estado-ação tenham sido visitados um número finito de vezes e a

política de escolha da próxima ação convirja, no limite, para uma política que utilize a melhor ação (ou seja, aquela que maximize a recompensa futura esperada).

Naturalmente, caso a ação escolhida a_{t+1} seja $\max a_{t+1} Q(s_{t+1}, a_{t+1})$, este algoritmo será equivalente ao do Q-Learning padrão. Entretanto, o algoritmo Sarsa admite que a_{t+1} seja escolhido aleatoriamente com uma probabilidade predefinida. Por eliminar o uso do operador *max* sobre as ações, este método é mais rápido que o Q-Learning para situações onde o conjunto de ações tenha cardinalidade alta.

3.7.5. Algoritmo Dyna

O termo Dyna foi introduzido em Singh [SIN96], e define uma técnica simples para integrar funções de aprendizado, planejamento e atuação.

O agente interage com o ambiente, gerando experiências reais. Estas experiências são utilizadas para melhorar diretamente as funções de valor e política de ações (através de algum método de AR) e aperfeiçoar um modelo do ambiente, que o agente pode usar para prever como o ambiente responderá a suas ações. As experiências originárias de simulação sobre este modelo são então utilizadas para melhorar as funções de valor e política de ações (planejamento sobre o modelo).

Após cada transição $s_t, a_t \rightarrow s_{t+1}, r_t$, o algoritmo Dyna armazena em uma tabela, para o valor de (s_t, a_t), a transição observada (s_{t+1}, r_t). Durante o planejamento, o algoritmo escolhe amostras aleatórias de pares estado-ação que foram experimentados anteriormente, ou seja, contidos no modelo. A seguir, realiza experiências simuladas nestes pares estado-ação selecionados. Finalmente, é aplicada uma atualização baseada em um método de AR sobre essas experiências simuladas, como se elas tivessem realmente ocorrido.

Tipicamente, o mesmo método de AR é utilizado tanto para o aprendizado a partir da experiência real quanto para o planejamento das experiências simuladas [MON04].

3.7.6. O Algoritmo Q-Learning

O algoritmo Q-Learning proposto por Watkins [WAT92] é o método mais popular utilizado para problemas em AR. Trata-se de um algoritmo que permite estabelecer de maneira autônoma e iterativa uma política de ações. Pode-se demonstrar que o algoritmo Q-Learning converge para um procedimento de controle ótimo, quando a hipótese de

aprendizagem de pares estado-ação Q for representada por uma tabela completa contendo a informação de valor de cada par. A convergência ocorre tanto em processos de decisão *Markovianos* determinísticos quanto não-determinísticos.

A idéia básica por trás do Q-Learning é que o algoritmo de aprendizagem aprenda uma função de avaliação ótima sobre todo o espaço de pares estado-ação SxA. A função Q fornece um mapeamento da forma $Q : SxA \rightarrow V$, onde V é o valor de utilidade esperada ao se executar uma ação a no estado s. Desde que o particionamento do espaço de estados do agente e o particionamento do espaço de ações não omitam informações relevantes nem introduzam novas, uma vez que a função ótima Q seja aprendida, o agente saberá precisamente que ação resultará na maior recompensa futura em uma situação particular s.

A função $Q(s,a)$, da recompensa futura esperada ao se escolher a ação a no estado s, é aprendida através de tentativa e erro, conforme Equação 3.7:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + y.\max_a Q(s',a) - Q(s,a)] \tag{3.7}$$

no qual α é a taxa de aprendizagem, *r* é a recompensa, ou custo, resultante de tomar a ação a no estado s, γ é o fator de desconto e o termo $\max_a Q(s',a)$ é a utilidade do estado s resultante da ação a, obtida utilizando a função Q que foi aprendida até o presente. A função Q representa a recompensa descontada esperada ao se tomar uma ação a quando visitando o estado s, e seguindo-se uma política ótima desde então.

O fator γ pode ser interpretado de várias formas: pode ser visto como uma taxa de gratificação, como uma probabilidade de ir para o próximo estado ou como um artifício matemático para evitar a soma infinita [LIT96]. A forma procedimental do algoritmo Q-Learning é apresentada na Figura 3.4.

```
Algoritmo Q-Learning();
1 Para cada s,a inicialize Q(s,a)=0;
2 Observe s;
3 Repita até que critério de parada seja satisfeito:
   Selecione ação a usando a política de ações atual;
    Execute a ação a;
      Receba a recompensa imediata r(s,a);
      Observe o novo estado s';
      Atualize o item Q(s,a) de acordo com a Equação
      3.7;
      s ← s';
4 Se critério de parada falso retorne ao passo 3;
5 Fim.
```

Figura 3.4: Algoritmo Q-Learning [MON04]

Uma vez que todos os pares estado-ação tenham sido visitados um número finito de vezes, garante-se que o método gerará estimativas Q_t que convergem para o valor de Q [WAT92]. Na prática, a política de ações converge para a política ótima em tempo finito, embora de forma lenta.

Uma característica do Q-Learning é a função valor-ação Q aprendida, que se aproxima diretamente da função valor-ação ótima $Q*$, sem depender da política que está sendo utilizada. Este fato simplifica bastante a análise do algoritmo e permite fazer testes iniciais da convergência. A política ainda mantém um efeito ao determinar quais pares estado-ação devem ser visitados e atualizados, porém, para que a convergência seja garantida, é necessário que todos os pares estado-ação sejam visitados continuamente e atualizados, por isso Q-Learning é um método *off-policy* [TSI94].

Dado os valores Q, existe uma política definida pela execução da ação a, quando o agente está em um estado s, que maximiza o valor $Q(s,a)$. Watkins [WAT92] demonstra que se cada par estado-ação for visitado um número suficientemente grande de vezes e a decrescer apropriadamente, as funções valoração Q irão convergir com probabilidade para $Q*$ e, conseqüentemente, a política irá convergir para uma política ótima.

A convergência do algoritmo Q-Learning não depende do método de exploração usado. Um agente pode explorar suas ações a qualquer momento, não existem requisitos para a execução de ações estimadas como as melhores. No entanto, para melhorar o desempenho do sistema é necessária, durante o aprendizado, a busca das ações que maximizam o retorno [MAR00].

Resumidamente, podem-se enumerar alguns dos aspectos mais importantes do algoritmo Q-Learning:

i) O objetivo do uso do algoritmo Q-Learning é achar uma regra de controle que maximize cada ciclo de controle;

ii) O uso do reforço imediato é indicado sempre que possível e necessário, desde que ele contenha informação suficiente que ajude o algoritmo a achar a melhor solução;

iii) O algoritmo Q-Learning é adotado quando o número de estados e ações a serem selecionados é finito e pequeno.

iv) O algoritmo Q-Learning foi o primeiro método de AR a possuir forte provas de convergência. É uma técnica simples que calcula diretamente as ações sem uso de modelo.

3.8. Medidas de Avaliação de Políticas

Medir o desempenho de um agente de aprendizagem é uma tarefa nada trivial. O desempenho de um agente pode ser medido conforme suas tarefas propostas e principalmente quanto ao objetivo almejado. Para isso, algumas medidas como velocidade de convergência para o comportamento ótimo ou quase-ótimo, reforços totais obtidos pelo agente e percentual de otimalidade (99% por exemplo) pode ser preferível em muitas aplicações. Entretanto um agente que tem garantia de eventual otimalidade, tende a apresentar uma taxa lenta de aprendizagem [KAE96].

Por outro lado, uma aprendizagem acelerada pode incluir muitas penalidades durante o período de aprendizagem. Caso um grande número de penalidades seja indesejável, talvez seja preferível adotar uma estratégia menos agressiva que obtenha reforços totais maiores, mas que talvez leve mais tempo para encontrar a política ótima [FRI02].

Ernst et al. [ERN05] cita em seu trabalho uma métrica de qualidade da solução produzida por um agente, como mecanismo para avaliar o desempenho dos algoritmos de AR. Para isso é utilizada uma política estacionária que calcula os valores esperados e os compara com algoritmos de regressão (*KD-tree*, *CART*, *Tree Bagging*, etc) sobre um conjunto de exemplos de estados iniciais escolhidos de um conjunto de vetores gerados pelo algoritmo Q-Learning. Assim, é calculada a média do valor esperado da política estacionária sobre o conjunto de estados iniciais.

No trabalho de Monteiro e Ribeiro [MON04] foi implementado um método para criar mapas cognitivos que consistia de um conjunto de partições de resolução variável, que dividia o ambiente em subáreas de diferentes tamanhos e que representava regiões sensorialmente homogêneas correspondentes a modelos locais que representam o conhecimento do agente sobre as regiões, e finalmente, combinava os paradigmas métricos e topológicos para analisar a performance dos algoritmos de aprendizagem baseados em reforço. Neste trabalho a tarefa do agente foi partir de um ponto de referência inicial e aprender uma trajetória de navegação de modo a atingir um ponto alvo, e ao mesmo tempo, desviar dos obstáculos do ambiente. Com isso, eram obtidos pelo agente, valores que serviam como base para os estudos das simulações dos algoritmos e posteriormente as avaliações com os algoritmos propostos pelo autor.

Ramon [RAM05] propôs o uso de árvores de decisão para encontrar uma aproximação da função Q para garantir a convergência em uma política ótima. É proposto um algoritmo de aprendizagem que gera uma nova estimativa Q' de uma estimativa precedente Q. O autor emprega três tipos diferentes para avaliação de performance. Primeiramente, há a quantidade $Q'(s,a)$ atualizado que se quer obter. Em seguida, há uma estimativa $\tilde{q}(s,a)$ de Q' que é medida durante a exploração executada pelo algoritmo. Posteriormente, $\tilde{q}(c)$ em que $c \subseteq S$xA, que é uma média da função $\tilde{q}$ sobre uma abstração c, onde S é o conjunto de estados e A o conjunto de ações. O algoritmo proposto pelo autor armazena na memória os valores da terceira opção de forma a gerar estatísticas. Assim, é anotada uma média de tempo obtida pelo agente até seu estado final e então relatada uma recompensa sempre depois do sucesso do episódio.

Observa-se no trabalho de Lev et al. [LEV02] o uso de uma função de valor heurístico, que avalia o desempenho do agente calculando os estados previstos por $\tilde{V}*$ e seleciona um operador que conduz ao estado mais promissor sendo $\tilde{V}*$ o valor ótimo da função.

Entretanto, estas políticas geralmente não possuem a eficiência esperada devido às diversidades das classes de problemas existentes. Além disso, elas requerem demasiado conhecimento do domínio para definir heurísticas relacionadas ao domínio do problema. Assim, podemos concluir que existe uma certa carência de trabalhos que tratem de mecanismos genéricos de avaliação em problemas que envolvem AR.

3.9. Técnicas de Estimação de Políticas Baseadas em Regressão

Técnicas de regressão foram adaptadas por alguns autores para a descoberta de políticas de ação. A tarefa de regressão tem como principal característica o mapeamento entre atributos de previsão e um atributo meta representado por uma variável de valores contínuos. A regressão é, portanto, uma generalização da tarefa de classificação onde o atributo meta é uma variável discreta.

Algoritmos de regressão se caracterizam por procurarem descobertas de fórmulas, e atuarem de forma semelhante aos que realizam aprendizado supervisionado. Na regressão procuram-se representações que mapeiem cada entidade para um valor numérico, ao invés de se procurar regras, frames ou árvores capazes de mapear cada entidade para uma classe [FAY96].

O problema de estimação de políticas utilizando técnicas de regressão consiste, portanto, na definição dos atributos de previsão (que geralmente descrevem um determinado estado e ações) e do atributo meta (que consiste no valor estimado de recompensa). Os exemplos de aprendizagem são gerados a partir das políticas intermediárias geradas iterativamente pelo sistema. Dessa forma, um algoritmo de regressão pode ser utilizado para estimar a partir de políticas passadas novos valores de recompensas para estados e ações, gerando novas políticas. É possível que esses algoritmos consigam estimar políticas semi-ótimas mais rapidamente que um algoritmo de AR convencional.

Portanto, regressão determina uma equação que relaciona matematicamente os valores de um conjunto de variáveis independentes, que são chamadas de explicativas, na tentativa de estabelecer o valor da variável dependente, ou variável de resposta [COE05].

Alguns exemplos de algoritmos de regressão como *Kd-Tree*, *Pruned Cart Tree*, *Tree Bagging*, *Totally Randomized Trees* são encontrados no trabalho de Ernst et al. [ERN05].

Alguns desses métodos produzem um modelo composto por uma única árvore de regressão a partir de um conjunto de treinamento, enquanto outros métodos constroem um conjunto de árvores de regressão. O autor caracteriza primeiramente os modelos que serão produzidos por estes métodos baseados em árvores e então explica como os diferentes métodos geram estes modelos.

O algoritmo *KD-tree* é um método de regressão construído a partir de um conjunto de treinamento que escolhe um ponto de um local médio para partição, então a árvore é

particionada em dois subconjuntos de mesma cardinalidade. O algoritmo *KD-tree* é representado como uma árvore binária, onde cada nó interno possui dois descendentes não importando a dimensionalidade k do espaço envolvido. Originalmente, esta árvore é apresentada em Bentley [BEN75], como uma alternativa a um problema mais genérico do que o geométrico das bases de dados espaciais.

O algoritmo *Cart-Tree* é um modelo de regressão não-paramétrico que estabelece uma relação entre as variáveis independentes (x), com uma única variável dependente, ou *meta* (*target*). O modelo é ajustado mediante sucessivas divisões binárias no conjunto de dados, para tornar os subconjuntos de dados da variável *meta* cada vez mais homogêneos [BRE84].

Para obter o número de divisões possíveis, este algoritmo considera, quando a variável explicativa tem K valores ordenados, $K-1$ divisões possíveis, quando a variável explicativa é nominal, com K categorias, consideram-se $2^{K-1}-1$ divisões possíveis.

Já o método de *Bagging*, consiste em utilizar múltiplas versões de um conjunto de treinamento. Cada versão é criada selecionando aleatoriamente $n' < n$ amostras do conjunto de treinamento D, com recobrimento. Cada um destes conjuntos é utilizado para treinar diferentes classificadores componentes e a decisão da classificação final é baseada no voto de cada componente [BRE96]. Assim, o método *Bagging* constrói os classificadores a partir de sucessivos e independentes conjuntos de amostras de dados. Estas amostras são geradas a partir de um conjunto de dados de treinamento. São extraídas aleatoriamente as n instâncias com substituição a partir do conjunto original, ou seja, ocorrendo à repetição de instâncias nas amostras.

Ernst et al. [ERN05] compara os algoritmos de *Cart* e *bagging*, e afirma que, o *Bagging* melhora a precisão do modelo produzido pela redução da variância, mas aumenta significativamente o tempo computacional.

E por fim, o autor utiliza um método chamado "*Totally Randomized Trees*" proposto por Geurts et al. [GEU06]. Árvores totalmente aleatórias correspondem a casos de árvores-extras, onde cada árvore é construída a partir de um conjunto completo de treinamento. Neste caso os testes em diferentes nós são escolhidos de maneira totalmente aleatória e independentemente dos valores resultantes dos elementos do conjunto de treinamento. Com isso, este algoritmo constrói árvores totalmente aleatórias sem observar o conjunto de treinamento, mas usando este conjunto de treinamento somente para remover os testes que conduzem a ramos vazios, decidindo quando parar o desenvolvimento de um ramo [GEU06].

3.10. Métodos Híbridos

Sistemas híbridos ou métodos híbridos de aquisição de conhecimento integram dois ou mais métodos diferentes para soluções de problemas. Osório [OSO99] aponta algumas das principais vantagens desses sistemas, descritos assim:

- A integração de duas técnicas complementares deseja permitir que uma complete as deficiências da outra de forma a obter um melhor desempenho;
- O uso de diferentes técnicas de aquisição e de representação de conhecimentos amplia a capacidade do sistema de adquirir novas informações, fazendo que esses sistemas não tenham apenas uma visão parcial limitada pelas imposições e restrições de um único método;
- Algumas tarefas complexas que usualmente não podem ser abordadas através da utilização de uma única técnica para a sua solução, podem ser quebradas em subproblemas, podendo ser tratados individualmente, chegando-se a uma solução global do problema;
- O processamento das informações pelos diferentes módulos irá usualmente produzir sistemas com uma maior performance, mais robustos, e com maior tolerância a falhas.

Diversos métodos como, indução de árvores de decisão, sistemas baseados em conhecimento, sistemas especialistas, sistemas baseado em regras nebulosas, aprendizagem por reforço, sistemas baseados em casos, algoritmos genéticos, redes neurais artificiais entre outros, vêem sendo usados de forma combinada, dando origem a múltiplos sistemas híbridos.

Rayan [RYA02] propõe um sistema híbrido chamado RACHEL, que incorpora técnicas de planejamento simbólico com AR, na intenção de produzir um sistema capaz de intensificar cada método, onde um tenta superar a deficiência de outro. O sistema usa uma nova representação de comportamento, o qual define estes comportamentos em termos das suas conseqüências desejada, mas deixando a implementação da política ser aprendida pelo AR. O sistema RACHEL forma um sistema de aprendizagem no qual é capaz de receber uma descrição abstrata do comportamento, construir planejamentos para descobrir regras a conquistar e aprender políticas concretas e ótimas escolhidas através da tentativa e erro.

No trabalho de Downing [DOW01], é proposta uma técnica híbrida chamada de *Reinforced Genetic Programming*. Este método combina AR com árvores baseadas em

programação genética. Esta técnica adiciona um novo elemento para um conjunto de funções da programação genética, que monitora as ações–seleções do ponto que fornece a ligação com AR. A programação genética é unida com alguns nós, onde, cujas ações são reforçadas de tal maneira que, o funcionamento sucessivo de algumas árvores exibe o melhoramento da performance dos ajustes das tarefas.

Em Henderson et al. [HEN05], um modelo híbrido que combina AR com aprendizagem supervisionada é proposto. A AR é utilizada neste caso para otimizar a média da recompensa entre a comunicação de sistemas que utilizam enormes conjuntos de dados e que possuem também grandes espaços de estados. Já a aprendizagem supervisionada é usada para restringir a política aprendida de uma parte do espaço de estados e modelar uma política com os dados que usa atualmente. Henderson et al. [HEN05] propõem a seguinte Equação: $Q_{hybrid}(s,a) = S_{data}(s,a)(Q_{data}(s,a) - U)$, na qual $Q_{hybrid}(s,a)$ é uma função que escolhe as ações para a política híbrida. U é usado para ajustar os valores dos estados penalizados não observados. $S_{data}(s,a)$ são os valores ajustados pela aprendizagem supervisionada e $Q_{data}(s,a)$ é a política definida pela AR.

Faria e Romero [FAR99], desenvolveram um método de navegação por sonares, denominado R'-Learning. Este método incorpora lógica *fuzzy* ao algoritmo R-Learning para navegação de robôs móveis em ambientes incertos. No algoritmo R'-Learning, as leituras dos sonares (que captura os valores do ambiente) são classificadas através de funções *fuzzy* para que valores muito próximos de outras classes não sejam considerados tão precisos quanto os demais.

É observado que, pouco ou quase nada, se tem encontrado sobre aplicações que integram os métodos de AR e aprendizagem baseada em instâncias. Assim, é aplicada neste estudo a união desses métodos, denominado de K-Learning, na intenção de permitir a descoberta de políticas ótima pelo agente de aprendizagem com um menor número de iterações. Os detalhes do método proposto e os resultados alcançados com este método estão descritos no Capítulo 5.

3.11. Considerações Finais

Neste capítulo foram estudadas as principais características da AR onde um agente deve aprender comportamentos interagindo em um ambiente através de tentativa e erro, ou seja, se uma ação é seguida de estados satisfatórios ou por uma melhoria no estado, então a tendência para se reproduzir esta ação é reforçada. Foi visto também que AR não é definido como um conjunto de algoritmos de aprendizagem, mas como uma classe de problemas de aprendizagem e que todo algoritmo que resolve bem esse problema é considerado um algoritmo de AR. Problemas de AR podem ser modelados através de Processos Decisórios de *Markov*, onde a probabilidade de transição de um estado s para um estado s' depende apenas do estado s e da ação a adotada em s, então o estado corrente fornece informação suficiente para o sistema de aprendizado decidir que ação deve ser tomada. Foram citados alguns dos principais algoritmos de AR, a saber; algoritmo R-Learning, algoritmo H-Learning, algoritmo $Q(\lambda)$, algoritmo Sarsa, algoritmo Dyna-Q e o algoritmo Q-Learning. O algoritmo Q-Learning proposto por Watklins é o método utilizado neste trabalho.

Verificou-se que uma das maiores dificuldades em AR consiste em avaliar a política de ações do agente, devido à complexidade de suas tarefas e a falta de conhecimento do domínio. Por fim, foram citados conceitos de sistemas híbridos e integração entre alguns métodos que tem como objetivo otimizar o desempenho da AR.

Nos capítulos seguintes serão apresentados alternativas a esses problemas propondo uma nova metodologia de avaliação de políticas assim como um método híbrido que integra os algoritmos Q-Learning e K-NN.

Capítulo 4

Uma Metodologia para Avaliação de Políticas

4.1. Considerações Iniciais

A avaliação das políticas de ação de um agente autônomo adaptativo é uma tarefa complexa devido à falta de mecanismos genéricos que permitam medir o desempenho de um agente de aprendizagem sem necessitar do conhecimento do domínio do problema. Além disso, as técnicas existentes não permitem que políticas de ação sejam avaliadas em diferentes ambientes. Neste capítulo discutiu-se o desenvolvimento de uma metodologia de avaliação que satisfaz esses requisitos. Apresentamos também as funcionalidades de um simulador desenvolvido para a avaliação da metodologia proposta.

4.2. Uma Nova Metodologia de Avaliação

A MAP considera o desempenho de um agente como proporcional ao número de acertos produzidos pela sua política de ações em um dado ambiente. Uma política representa um espaço de estados onde existe um estado inicial, um estado final e um conjunto de transições entre estados definidos por ações. Portanto, um acerto é obtido quando o agente encontra um caminho de menor custo entre o estado inicial e o estado final. Quando isso ocorre para todos os estados candidatos, pode-se dizer que uma política ótima foi descoberta. Dessa forma, a avaliação de uma política utiliza um algoritmo de resolução de problemas capaz de encontrar o caminho ótimo entre dois estados onde a função heurística corresponde aos valores retornados pela função de recompensa. Além disso, a quantidade de estados visitados também pode ser considerada.

Utilizamos como mecanismo de busca o algoritmo *A* Estrela (A^*), pois a busca pela melhor solução será ao mesmo tempo completa e ótima[2] [RUS95]. Uma vez que o A^* produz a melhor política possível para uma dada heurística, a avaliação (medida de qualidade) entre políticas descobertas por algoritmos diferentes de aprendizagem pode ser feita comparando estas políticas com a política produzida pelo A^*.

Pode-se assim afirmar, que esta metodologia é capaz de demonstrar a proximidade entre a política atual do agente e a política ótima. Um acerto ocorre quanto ele tem a capacidade de encontrar seu estado final com seus custos otimizados (menor caminho até o estado final). Utilizamos o tamanho (número de estados) e a soma do custo associado a cada estado (valor da função de recompensa) para avaliar o custo total de um caminho. O pseudocódigo que avalia o desempenho do algoritmo Q-Learning e de um algoritmo X[3] é apresentado pela Figura 4.1.

```
Algoritmo Avaliação_Desempenho(PQ, PX);
//PQ é a política de ações do Q-Learning e Q* sua política ótima;
//PX é a política de ações de um arranjo X e X* sua política ótima;
1 Inicialize estados_Q=0, estados_X=0, estados_A*=0;
2 Inicialize Q*=Política_ótima.Q, X*=Política_ótima.KNN;
3   Para cada estado(s) ∈ estado(S) aprendido pelo Q-Learning repita:
     Usando o algoritmo Avaliação_Política:
     Se estado_n.Q até estado_final.Q = estado_n.A* até estado_final.A*
       Então
         Política de ação Q*
       Senão
         Política de ação <> Q*
4   Para cada estado(s) ∈ estado(S) com o K-NN repita:
     Usando o algoritmo Avaliação_Política:
     Se estado_n.X até estado_final.X = estado_n.A* até estado_final.A*
       Então
         Política de ação X*
       Senão
         Política de ação <> X*
5 Retorne Avaliação_Política(PQ,PX);
6 Fim.
```

Figura 4.1: Pseudocódigo do algoritmo de avaliação de desempenho do algoritmo Q-Learning e de um algoritmo *X*

[2] Uma discussão mais detalhada sobre o algoritmo A^* pode ser encontrada no Anexo I.
[3] Algoritmo *X* representa os arranjos formado com o aprendizado do Q-Learning e serão estimados com o algoritmo K-NN.

O pseudocódigo da Figura 4.1 mostra como é avaliado o desempenho do algoritmo Q-Learning e de um algoritmo *X* qualquer *c*omparado com o algoritmo *A**. Para todos os estados do ambiente é aplicado o algoritmo `Avaliação_Política(P)`. O algoritmo avalia todos os estados e considera dois parâmetros em cada estado: i) o número de passos do estado *n* até o estado final; ii) o acúmulo da situação de cada estado *n* até o estado final. Assim, afirmamos que se a soma desses dois parâmetros de todos os estados for igual à soma de todos os estados do algoritmo *A**, os algoritmos Q-Learning e *X* possuem política ótima. O algoritmo *Avaliação_Política(P)* irá retornar o percentual de acerto dos algoritmos utilizados.

Para aplicar nossa metodologia na avaliação do desempenho dos algoritmos de aprendizagem, inserimos o agente em um ambiente parcialmente conhecido e ajustamos os parâmetros para cada ação possível aos estados candidatos. Em seguida, definimos os valores para os parâmetros de aprendizagem do agente. É possível analisar o desempenho dos algoritmos através da construção de um gráfico de aprendizagem[4].

O pseudocódigo da Figura 4.2 é utilizado em nossa pesquisa com a finalidade de medir a eficiência dos algoritmos Q-Learning, K-NN e do método K-Learning, discutidos nas seções seguintes. Os parâmetros do algoritmo A^* foram adaptados ao projeto em questão, onde $g(n)$ indica o número de passos que o agente precisa até conquistar o estado final e o $h(n)$ representa a heurística, que é o acúmulo dos congestionamentos (situações) de cada estado até o estado final. A função *custo* é utilizada para encontrar o melhor caminho de um estado *s* até o estado final a partir de uma dada política. Assim, o agente saberá o melhor caminho somando os custos do caminho com os custos dos valores de congestionamento.

```
Algoritmo Avaliação_Política(estado_final, P);
//P é uma política de ação qualquer de um algoritmo.
//PA* é a política ótima de ação do algoritmo A*.
1 Inicialize Acerto=0, Erro=0, CustoP=0, CustoA*=0;
2 Para cada s ∈ S
    CustoP ← custo(s, estado_final, P);
    CustoA* ← custo(s, estado_final, PA*);
      Se CustoP = CustoA*
         Acerto ← Acerto+1;
      Senão
          Erro ← Erro+1;
3 P ← (Acerto / (Acerto + Erro)) * 100;
```

Figura 4.2: Pseudocódigo da métrica de acerto de uma política *P*

[4] Os gráficos de aprendizagem relacionam o número de passos e a taxa de acerto percentual do agente. Os gráficos de aprendizagem estão detalhados na próxima seção.

A seguir é exemplificada a aplicação do algoritmo $A*$ ao problema proposto.

Considerando a Figura 4.3, onde o agente (representado na cor verde) deseja alcançar o estado final (representado na cor amarela), as cores em preto representam bloqueios e às demais cores níveis de congestionamento.

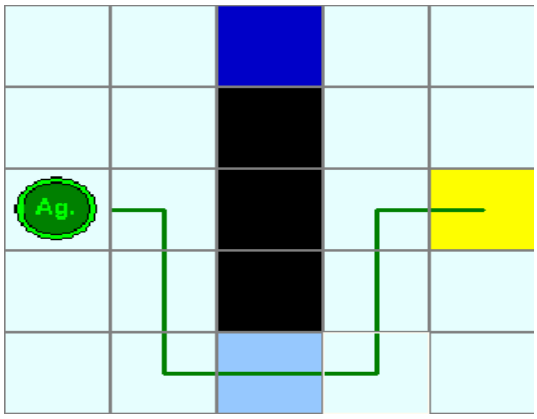

Figura 4.3: Ambiente de simulação com o melhor caminho traçado pelo algoritmo $A*$

O ambiente é constituído por um conjunto de estados, onde cada estado representa uma situação de congestionamento, bloqueio ou a meta. Simplificando a área de procura, como a Figura 4.3, o primeiro passo da busca consiste em marcar todos os estados a partir do estado final. Cada item na ordem representa um dos quadrados na grade e seu estado é registrado como acessível ou não-acessível. O caminho encontrado representa a seqüência de quadrados que devem ser tomados para ir da posição do agente até o estado final. Uma vez que o caminho é encontrado, o agente se move do centro de um quadrado ao centro do próximo e assim sucessivamente até que o estado final seja alcançado.

A qualidade do algoritmo $A*$ depende da qualidade da heurística estimada $h(n)$. Se h está próximo do custo real do caminho esperado, a eficiência do algoritmo será alta, mas por outro lado, se h possuir um valor não admissível, sua eficiência será ruim.

4.3. Implementação e Simulação

A avaliação da metodologia proposta e comparação entre algoritmos de descoberta de políticas tornou-se possível com a construção de um simulador de tráfego onde os experimentos podem ser configurados, controlados e os resultados podem ser graficamente

visualizados. As seções seguintes descrevem as principais características do simulador implementado.

4.3.1. Linguagem de Programação

A linguagem utilizada para o desenvolvimento do simulador é o C++. Optou-se por essa linguagem devido ao fato desta linguagem permitir a programação mais rápida e robusta. Outra característica que influencia sua utilização é o suporte a ambientes integrados de desenvolvimento que facilitam a programação. Foi utilizado o C++ Builder da Borland [BOR02], sendo um ambiente de desenvolvimento que facilita a criação dos itens da interface e implementação das funcionalidades do software.

4.3.2. Definição da Situação dos Estados

Para representar o ambiente de simulação (malha/mapa viária), os cruzamentos foram representados como estados e, a partir deles, o agente pode tomar direções como: norte, sul, leste ou oeste.

Para representar o tráfego entre cada cruzamento, foram utilizadas as seguintes condições de congestionamento de trânsito: livre, pouco congestionado, congestionado/desconhecido, muito congestionado e bloqueado, que são representadas por diferentes níveis de cores, mostrado na Figura 4.4.

Cada situação de congestionamento é representada por um valor. Os valores definidos através de experimentos para estas situações foram: -0,1 para livre; -0,2 para pouco congestionado; -0,3 para congestionado; -0,4 para muito congestionado; e -1 para bloqueado. Esses valores são os que apresentam melhor desempenho para a adaptação do agente durante iteração com ambiente. Para o estado final, o valor previamente definido foi 1.

A distância entre um estado e outro é definida com valor 0,1. Este valor representará o custo do agente até o estado final. Assim, se o agente passar por 7 estados para chegar até o estado final, seu custo será de 0,7.

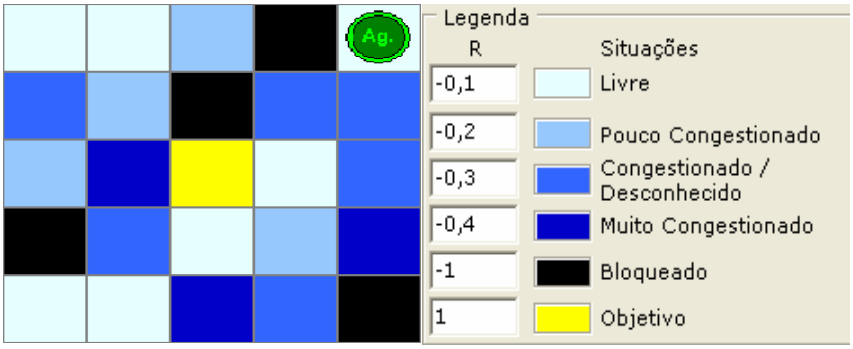

Figura 4.4: Nível de cores que representam a situação de cada estado

4.3.3. Desenvolvimento do Simulador

Desenvolvemos um simulador para demonstrar o desempenho dos algoritmos de descoberta de políticas de ação aplicando nossa nova metodologia de avaliação. A Figura 4.5 apresenta uma visão geral do simulador e seus principais componentes.

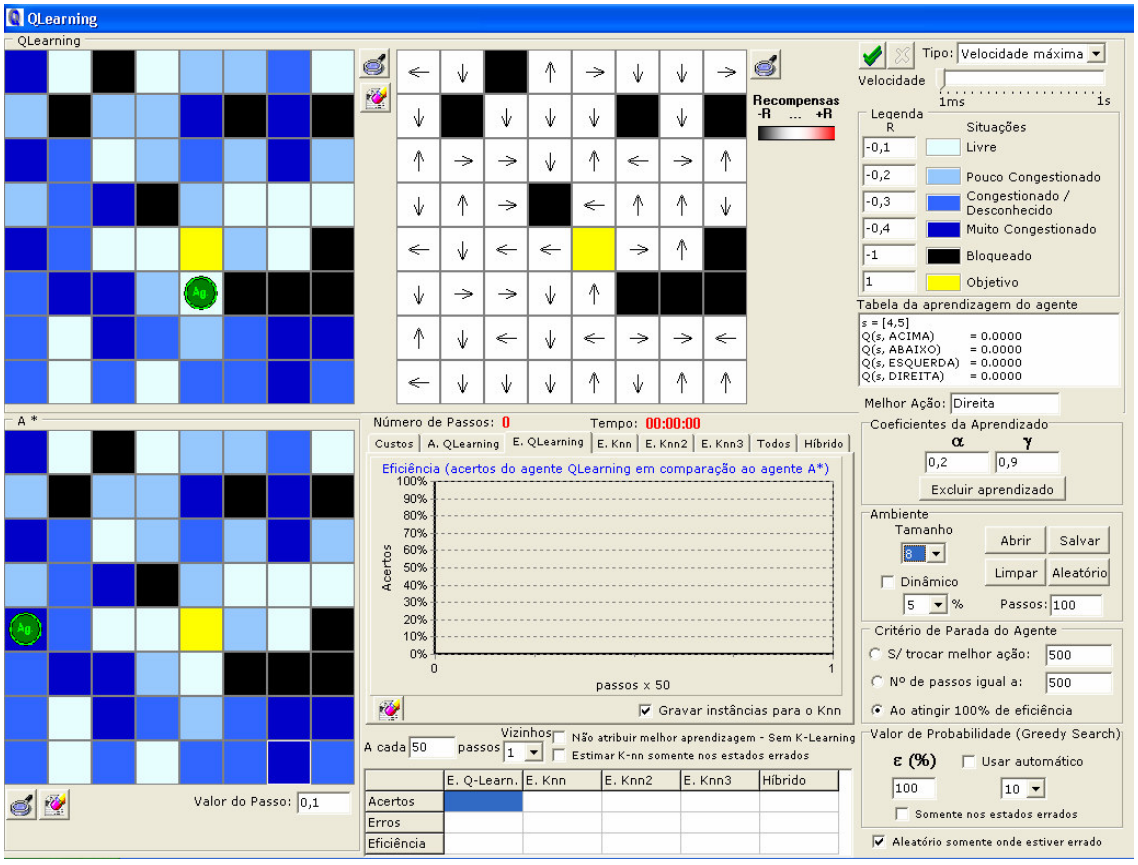

Figura 4.5: Tela do simulador

Na Figura 4.5 é possível observar 3 áreas. No canto superior esquerdo é representado o ambiente do algoritmo Q-Learning. Neste ambiente é possível representar as situações de congestionamento para cada estado, conforme a legenda. Logo abaixo está o ambiente que representa o algoritmo $A*$. Este ambiente irá demonstrar o melhor caminho que o agente pode fazer até a meta e será utilizado para medir a eficiência dos algoritmos. O último ambiente apresenta as iterações do agente, representando sua política de aprendizagem atual através de níveis de cores e através de setas que indicam o caminho do agente ao estado final após sua aprendizagem.

Outra importante funcionalidade do simulador são os gráficos de aprendizagem, onde é possível observar o comportamento do agente ao longo de suas iterações com o ambiente. A dimensão *Y* do gráfico representa a eficiência do agente em um número determinado de passos. A dimensão *X* corresponde ao número de passos realizados pelo algoritmo. Estes gráficos são descritos na subseção seguinte.

4.3.4. Descrição do Simulador

Nesta subseção é apresentada a descrição do simulador, detalhando cada item que compõe a interface.

Ambiente: Representa a área em que o agente se movimenta. Cada quadrado (estado) representa uma posição deste ambiente, contendo uma das cinco situações de congestionamento (livre, pouco congestionado, congestionado/desconhecido, muito congestionado e bloqueado), e ainda o estado final a ser alcançando pelo agente. A Figura 4.6 apresenta o ambiente utilizado para a simulação do agente e o seu melhor caminho obtido com a política atual. A Figura 4.7 representa um mapa das ações previstas por uma dada política.

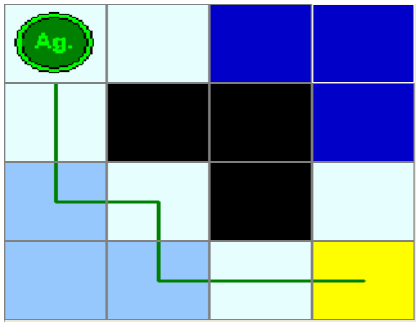

Figura 4.6: Melhor caminho aprendido pelo agente

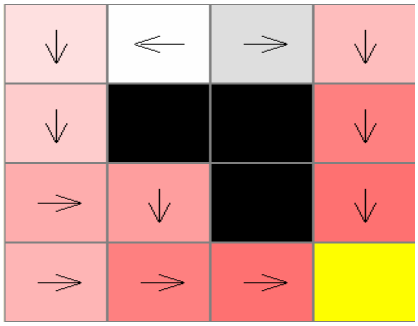

Figura 4.7: Ações de uma política

Situações de erros: A Figura 4.8 apresenta as linhas que representam os erros do agente. Quando a posição do ambiente é marcada com duas linhas diagonais em forma de "X", significa que, a partir desta posição, o agente não conseguiu encontrar o estado final. Quando a posição é marcada com uma linha diagonal, indica que o custo para ir desta posição até o estado final foi maior em comparação ao melhor caminho possível. Se a posição não estiver marcada com nenhuma linha, indica que o agente Q-Learning consegue alcançar o estado final com o mesmo custo calculado pelo $A*$.

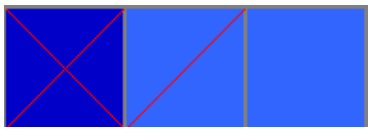

Figura 4.8: Situações de erro

Gráfico da troca de ação do agente: Este gráfico tem por objetivo indicar quando um estado trocou sua melhor ação em uma determinada quantidade de interações. Neste gráfico, a dimensão Y representa o número de vezes que o agente altera sua melhor ação para uma posição do ambiente a um número de ações realizadas. Com isto, é possível observar que quanto menos trocas o agente realiza mais próximo da política ótima o agente se encontra, sendo possível determinar a parada do agente quando o mesmo não realiza mais a troca da sua melhor ação em um determinado número de passos. A Figura 4.9 representa o exemplo de um gráfico com essas características

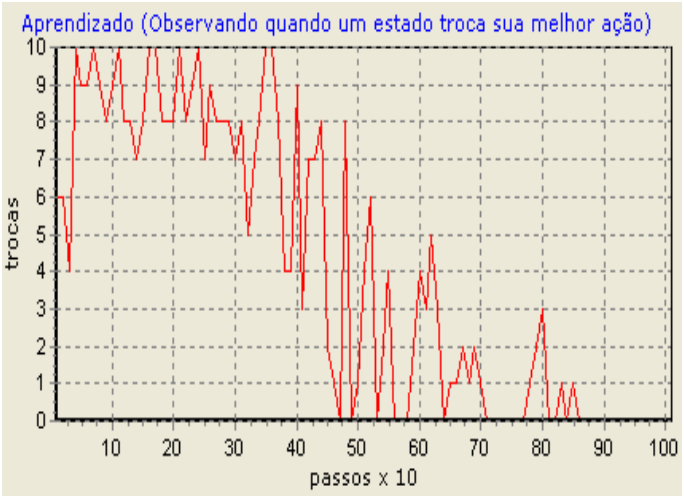

Figura 4.9: Gráfico do aprendizado do agente em relação ao número de trocas realizadas

Gráfico da aprendizagem do algoritmo Q-Learning: Este gráfico tem por objetivo demonstrar a aprendizagem do agente Q-Learning com sua política atual. Esta avaliação é feita através do número de acertos do agente. Este gráfico demonstrará a eficiência do agente em um número determinado de passos. A Figura 4.10 demonstra o gráfico com mil iterações.

Figura 4.10: Gráfico do aprendizado do algoritmo Q-Learning

Gráfico da aprendizagem dos algoritmos K-NN: Este gráfico tem por objetivo demonstrar a aprendizagem dos *n* arranjos[5] formados com a aprendizagem do algoritmo Q-Learning. Cada curva do gráfico representa um tipo de arranjo estimado pelo algoritmo K-NN. O

[5] Os arranjos estão descritos no Capítulo 5.

desempenho de cada arranjo é determinado pelo número de passos do agente. A Figura 4.11 demonstra o gráfico plotando todos os arranjos do algoritmo K-NN comparando com a curva da aprendizagem do Q-Learning.

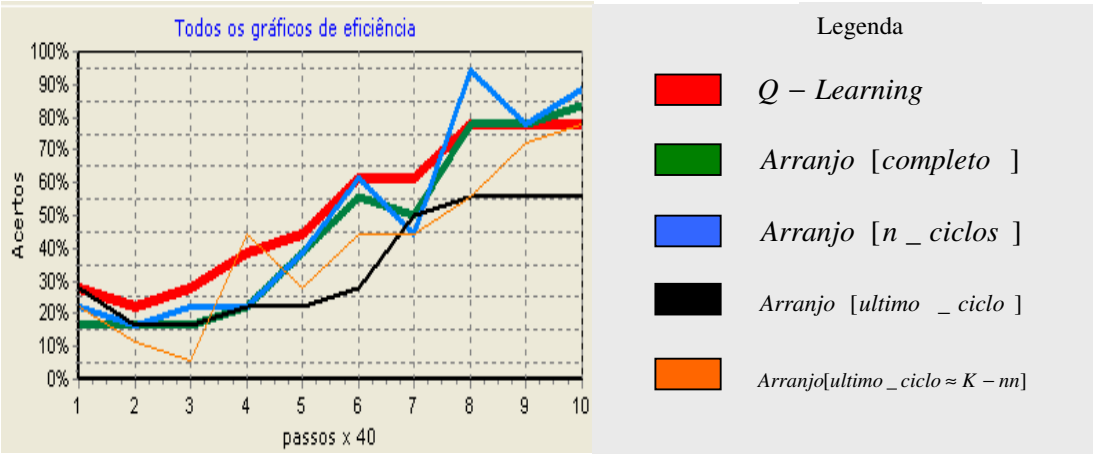

Figura 4.11: Conjunto de todos os gráficos da aprendizagem do agente

Gráfico da aprendizagem do algoritmo K-Learning: O gráfico 4.12 demonstra o desempenho da política atual do algoritmo K-Learning.

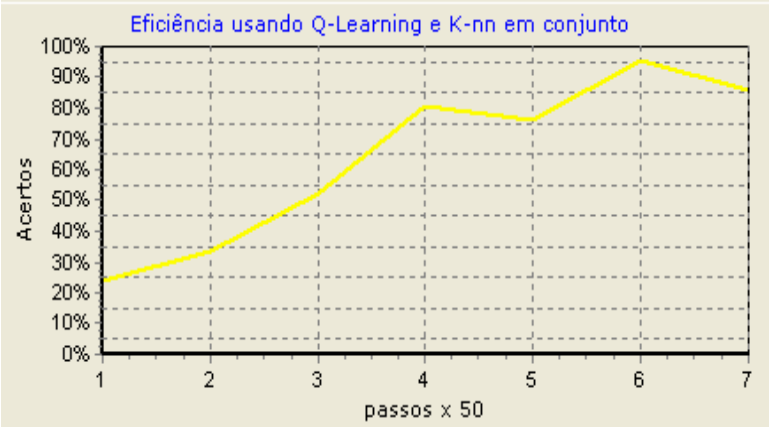

Figura 4.12: Gráfico do aprendizado do algoritmo K-Learning

Representação de valores e legendas das situações do ambiente: A legenda das situações e do estado final é apresentada através de cores. Os valores contidos na legenda são que apresentam os melhores resultados para o algoritmo. Esses valores são positivos quando utilizados pelo algoritmo $A*$, pois no algoritmo Q-Learning o agente procura a melhor

recompensa (maior valor), e no algoritmo $A*$ o agente procura o menor valor (somando a distância e a recompensa). Os valores contidos na legenda são os que conseguem melhor eficiência para o agente Q-Learning. A Figura 4.13 apresenta os valores e as situações do ambiente.

Figura 4.13: Valores e legendas das situações do ambiente

Tabela da aprendizagem do agente: A Figura 4.14 apresenta os valores da aprendizagem do agente adquiridos durante seu aprendizado. A variável s representa a posição atual do agente (coluna x linha), a variável Q representa a recompensa que o agente recebe ao realizar as ações "acima, abaixo, direita, esquerda", a partir da posição s.

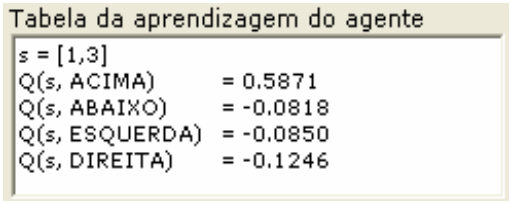

Figura 4.14: Tabela de aprendizagem do agente

Configurações da aprendizagem do agente: A Figura 4.15 representa um quadro com os coeficientes da aprendizagem do algoritmo Q-Learning. A variável α representa a constante de aprendizado. A variável γ representa o fator de desconto, e por fim, a variável ε representa a probabilidade de o agente escolher uma ação pelo maior valor esperado. O valor 50 da Figura 4.15 indica que, a probabilidade do agente realizar suas ações baseando-se em seu aprendizado ou aleatoriamente é de 50%.

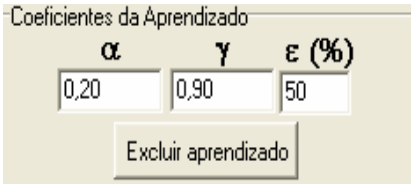

Figura 4.15: Configurações da aprendizagem do agente

Critérios de parada do agente: A Figura 4.16 apresenta os critérios de parada do agente. A opção "S/ trocar a melhor ação", indica que o agente irá interagir com o ambiente até o momento em que o número de passos sem trocar sua melhor ação atinja o valor determinado. A opção "Nº. de passos igual a", fará com que o agente interaja com o ambiente até atingir o número de passos igual ao valor determinado. E finalmente a opção "Ao atingir 100% de eficiência" faz com que o agente interrompa sua aprendizagem quando encontrar uma política ótima.

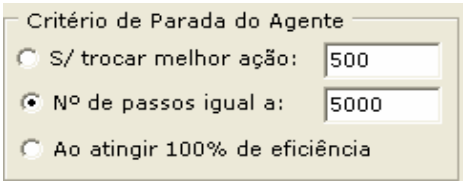

Figura 4.16: Critérios de parada do agente

Custos dos passos dos agentes: A Figura 4.17 apresenta os valores dos custos do agente para chegar até o estado final. Os itens "Distância", "Situações" e "Total" representam o custo para ir de uma posição selecionada até o estado final. O item "soma de todas as posições" representa a soma dos custos do agente para chegar até o estado final, partindo de cada posição possível do ambiente. O item "eficiência" representa a eficiência dos algoritmos. O algoritmo $A*$ sempre terá 100% de eficiência. A eficiência do algoritmo Q-Learning é definida comparando seu item "soma de todas as posições" com o do $A*$. A Equação 4.1 simplifica o algoritmo da Figura 4.2:

$$CustMin(A*) = \sum_{e \in E}^{n} 0{,}1 + \sum_{e \in E}^{n} Sit_e \tag{4.1}$$

no qual *CustMin(A*)* representa o custo mínimo de um caminho de tamanho *n* até o estado

final calculado pelo algoritmo $A*$, e é um estado pertencente ao caminho e *Sit* indica o custo da situação em um dado estado e. Naturalmente, quando um algoritmo produz custos idênticos ao $A*$, afirmamos que o agente encontrou a política ótima para todo seu espaço de estados.

	A*	QLearning
Distância:	0,300	0,300
Situações:	0,500	0,600
Total:	0,800	0,900
Soma de todas as posições:	15,300	15,500
EFICIÊNCIA:	100%	98,710%

Figura 4.17: Custos dos passos dos agentes

Tabela de comparação dos algoritmos: A Figura 4.18 representa o quadro que mostra os valores da eficiência dos algoritmos. São os mesmos valores apresentados nos gráficos de eficiência.

	Q-Learning	A[completo]	A[n_ciclos]	A[ultimo_ciclo	A[ult_ciclo~(K-nn)]
Acertos	19	19	19	18	18
Erros	2	2	2	3	3
Eficiência	90,48%	90,48%	90,48%	85,71%	85,71%

Figura 4.18: Tabela de comparação dos algoritmos

Configurações para os gráficos de eficiência: A Figura 4.19 mostra as configurações para os gráficos de eficiência. O campo "A cada X passos" representa o intervalo de passos em que será medida a eficiência dos algoritmos. O campo "Nº. Vizinhos p/ o Knn" indica o número de vizinhos que o algoritmo K-NN irá utilizar. O item "Executar em seqüência" faz com que os resultados do algoritmo K-NN sejam demonstrados simultaneamente.

A cada: 50 passos. Nº Vizinhos p/ o Knn: 1 Executar em seqüência

Figura 4.19: Configurações para os gráficos de eficiência

4.4. Utilização da Metodologia para Avaliar o Desempenho do Q-Learning

A partir da metodologia proposta, realizamos alguns experimentos preliminares que

têm como objetivo verificar a coerência dos valores encontrados. Os experimentos realizados com o algoritmo Q-Learning avaliaram sua eficiência considerando fatores como: o número de iterações necessárias para o agente atingir sua melhor eficiência; a política de qualidade de recompensas; variações na taxa de aprendizagem; fator de desconto e os valores das recompensas para as situações de congestionamento. Os valores obtidos são apresentados nos parágrafos seguintes.

Para obtenção dos resultados da eficiência dos algoritmos Q-Learning, K-NN e K-Learning, foram utilizadas 10 amostras diferentes para cada tipo de experimento com cada algoritmo. O aprendizado em cada amostra foi realizado 10 vezes pelos algoritmos, pois se observa que fazendo experimentos em um mesmo ambiente, com fatores iguais, podem ocorrer variações nas eficiências geradas pelos algoritmos. Isto ocorre porque as ações do agente são autônomas e os valores gerados durante sua aprendizagem são estocásticos. Portanto, a política de ações do agente pode variar de um experimento para outro. Com isso, as eficiências apresentadas ao longo deste trabalho representam a média de todos os experimentos gerados nas 10 amostras com 10 repetições em cada. Esse número de repetições foi suficiente para avaliar com precisão a eficiência média dos algoritmos, pois observamos que a partir deste número os resultados dos experimentos começavam a se repetir.

4.4.1. Número de Passos Necessários para Atingir a Melhor Eficiência

Foram realizados experimentos em ambientes de tamanho entre 16 e 64 estados, pois se observa em outros trabalhos que esses ambientes são os mais estudados pela comunidade científica [HAR96] [SUT98] [RIB99] [ABR01] [AND01].

Nos primeiros testes realizados, observou-se que para cada tamanho de ambiente testado, o agente alcançava a melhor eficiência em um número diferente de passos. A Figura 4.20 apresenta um comparativo de: eficiência, quantidade de estados e quantidade de passos. É possível observar que para ambientes menores que 25 estados são necessários aproximadamente 500 passos para obter sua melhor eficiência. Para ambientes de até 81 estados observamos que é necessário em torno de 5000 passos. Ambientes acima de 100 estados necessitam de um número superior a 20000 passos para atingir sua melhor eficiência.

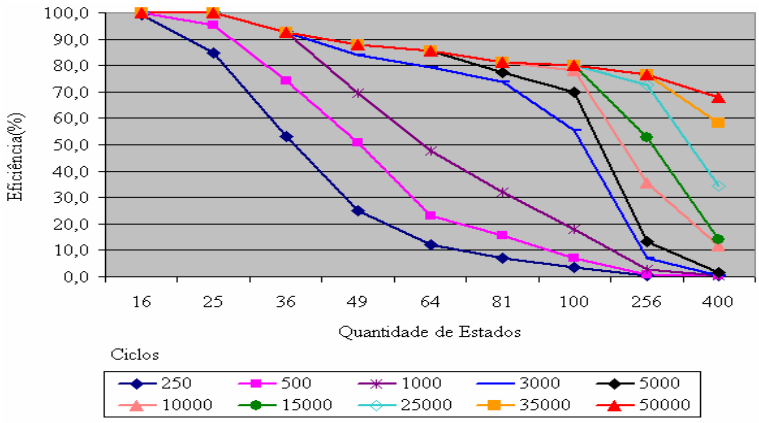

Figura 4.20: Gráfico comparativo da aprendizagem do algoritmo Q-Learning relacionando eficiência, número de estados e número de ciclos

A Figura 4.20 mostra que em ambientes com muitos estados o agente precisa de um número elevado de ciclos, devido a sua necessidade de visitar cada estado inúmeras vezes para realizar o aprendizado.

4.4.2. Política de Qualidade de Recompensas

Também foram utilizadas nos experimentos duas políticas de qualidade de recompensas no intuito de otimizar a qualidade da política de ações. As duas políticas de qualidade de recompensas são:

a) A recompensa é igual à soma dos valores dos estados vizinhos ao estado para qual o agente se moveu somada com o valor desse estado;

b) A recompensa é igual ao valor do estado para onde o agente se moveu.

Neste documento consideramos somente os resultados gerados pela segunda política de qualidade (b), pois verificamos que a primeira política (a) apresenta um desempenho em média 25% inferior comparada à segunda.

4.4.3. Taxa de Aprendizagem

Os experimentos para encontrar a melhor taxa de aprendizagem foram realizados em ambientes com estados e tamanhos diferentes utilizando fator de desconto com valor 0,9. Os melhores valores utilizados como taxa de aprendizagem para o algoritmo Q-Learning estão entre 0,10 e 0,20. Taxas superiores a 0,20 fizeram com que o agente, ao estabelecer uma

melhor ação em um determinado estado do ambiente, não efetuasse outras ações na busca de caminhos melhores.

Os valores inferiores a 0,10 tornaram a aprendizagem do agente mais demorada, sendo que o número de iterações apresentada na Figura 4.20 foi insuficiente para o agente alcançar sua melhor eficiência. A taxa de aprendizagem de 0,20 foi a que obteve melhores resultados e com menores iterações, sendo este utilizado para os demais experimentos. Observou-se também que quanto menor a taxa de aprendizagem, menores são as variações nas amostras. Para verificar as melhores taxas foram realizados experimentos com valores entre 0,05 e 0,85 em ambientes com até 64 estados.

A Figura 4.21 apresenta a eficiência das taxas de aprendizagem aplicada em ambientes com tamanho de até 64 estados.

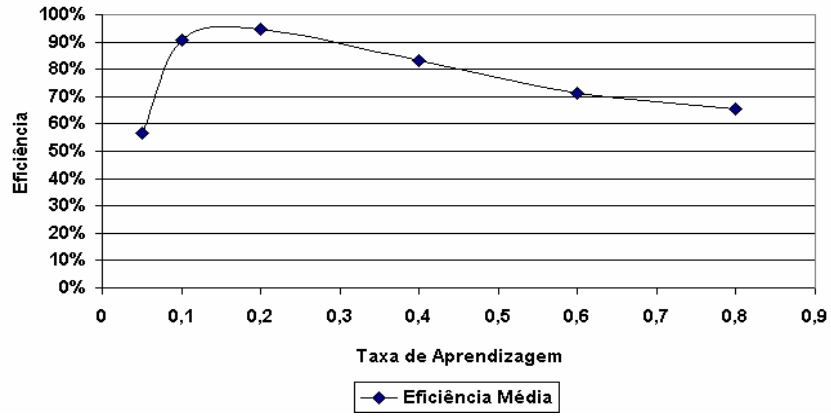

Figura 4.21: Gráfico de eficiência das taxas de aprendizagem

4.4.4. Fator de Desconto

Os experimentos alterando o fator de desconto foram realizados em ambientes com estados e tamanhos diferentes. Os melhores valores para o fator de desconto estão entre 0,85 e 0,95 conforme apresentado na Figura 4.22. Valores inferiores a 0,85 mostraram-se ineficientes para o algoritmo, tendo baixa relevância no aprendizado do agente, e assim este ao invés de escolher ações que o levam até o estado final, escolhem ações que apenas o levam para estados com condições de congestionamento menores. Valores superiores a 1,1 apresentam relevância excessiva no uso das recompensas dos estados vizinhos, e assim o agente não procura os caminhos com menores níveis de congestionamento, mas sim os caminhos mais curtos até o estado final.

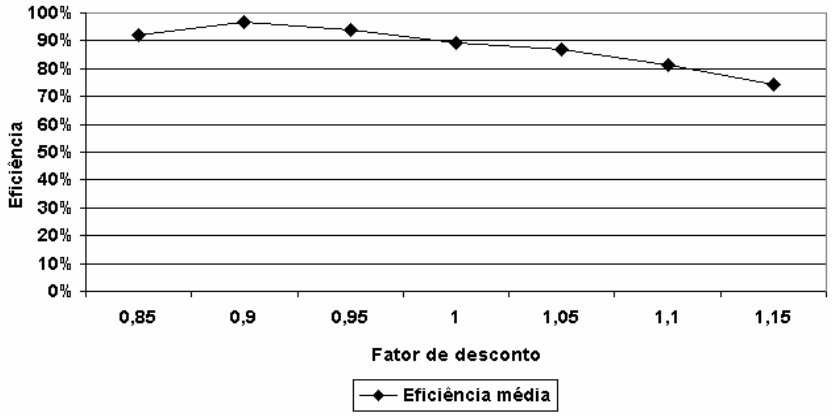

Figura 4.22: Gráfico de eficiência dos fatores de desconto

4.5. Considerações Finais

Neste capítulo discutimos o desenvolvimento de uma nova metodologia de avaliação de algoritmos para descoberta de políticas de ação chamada MAP. A metodologia proposta foi integrada a um ambiente simulador que possibilita a avaliação de algoritmos de AR e, principalmente, a configuração dos parâmetros de cada algoritmo. A MAP foi utilizada para a avaliação do algoritmo Q-Learning para que os resultados encontrados pudessem ser comparados com os valores dos parâmetros mais utilizados na literatura. A metodologia mostrou-se robusta e capaz de avaliar precisamente a qualidade de uma política de ações. Os conceitos utilizados na MAP sugerem que os valores de qualidade obtidos podem ser utilizados no desenvolvimento de métodos híbridos de aprendizagem como uma heurística em busca de melhores políticas. Esses conceitos são utilizados no próximo capítulo que apresenta o desenvolvimento de métodos que integram AR e aprendizagem por instâncias e também a utilização da MAP para comparar o desempenho destes algoritmos.

Capítulo 5

Avaliação e Descoberta de Políticas Utilizando Métodos Híbridos de Aprendizagem

5.1. Considerações Iniciais

Neste capítulo utilizamos alguns dos conceitos estudados nos Capítulos 3 e 4 para desenvolver e avaliar um método híbrido que integra duas técnicas utilizadas em agentes autônomos adaptativos: AR (algoritmo Q-Learning) e Aprendizagem Baseada em Instâncias (algoritmo K-NN). A integração destes métodos tem por objetivo aproveitar as vantagens de cada algoritmo: o K-NN permite que estados com características semelhantes tenham recompensas similares, podendo antecipar valores de recompensas e diminuir o número de iterações necessárias ao Q-Learning. Por outro lado, o Q-Learning garante que uma política ótima seja encontrada ao longo das iterações. Além disso, a aprendizagem do agente deve ser dirigida por uma função objetiva de qualidade a ser maximizada ao longo das iterações com o ambiente. Neste trabalho utilizamos a metodologia MAP para garantir uma evolução rápida da aprendizagem para políticas de boa qualidade.

5.2. Algoritmos de Regressão Baseados no KNN

Os métodos de aprendizagem baseados em instâncias elaboram hipóteses diretamente a partir das próprias instâncias de treinamento. Dessa forma, o algoritmo K-NN [AHA91], armazena instâncias de treinamento na memória como pontos no espaço *n*-dimensional, definido pelos *n* atributos que os descrevem. Quando uma nova instância precisa ser classificada, a classe mais freqüente dentre os *K* vizinhos mais próximos é escolhida. Acreditamos que o algoritmo K-NN pode ser utilizado para estimar os valores da tabela de aprendizagem de um agente que utiliza o Q-Learning. O algoritmo K-NN deve receber como

entrada um conjunto de instâncias geradas a partir de uma política de ações durante sua fase de aprendizagem. Desejamos com isso, gerar um novo conjunto de estados de aprendizagem, na tentativa de aproximar o agente de sua política ótima. Isso pode ser feito definindo como um arranjo o conjunto de instâncias geradas durante um ciclo de passos realizado pelo algoritmo Q-Learning. É interessante notar que para cada estado do ambiente podemos gerar quatro tuplas (uma para cada ação) que representam os valores aprendidos pelo agente. Sendo assim, o nosso espaço de estados nunca muda e é independente do número de passos realizado pelo agente. Dessa forma, cada instância de treinamento do arranjo possui os seguintes atributos:

- atributos para representação do estado na forma das recompensas esperadas para as ações: norte, sul, oeste e leste;
- uma ação e
- recompensa para esta ação.

Tabela 5.1: Exemplo de atributos e valores do Arranjo[i...n] gerados pelo agente com o algoritmo Q-Learning durante aprendizagem

Estado (x,y)	**R Norte**	**R Sul**	**R Oeste**	**R Leste**	**Ação Escolhida**	**R Ação Escolhida**
(0,0)	-0.535	-0.987	-0.651	0.327	Norte	-0.523
(0,0)	-0.535	-0.987	-0.651	0.327	Sul	-0.933
(0,0)	-0.535	-0.987	-0.651	0.327	Oeste	-0.651
(0,0)	-0.535	-0.987	-0.651	0.327	Leste	0.328
(0,1)	-0.112	1	0.35	-0.9	Norte	-0.122
. . .	. . .	. . .	. . .	. . .	. . .	. . .

A Tabela 5.1 apresenta alguns exemplos de instâncias de treinamento. Esses vetores serão comparados através de uma função de similaridade (co-seno) e em seguida, as instâncias mais próximas serão encontradas. O pseudocódigo da Figura 5.1 apresenta como as instâncias do conjunto de treinamento gerado pelo algoritmo Q-Learning são aproximados e estimados com o algoritmo K-NN.

```
Algoritmo estimar_instância(atributo-meta, K);
//Am é somatória dos atributos-meta mais próximos de K vizinhos;
//Atributo-meta é o valor estimado que vai ser usado como recompensa ao
  agente;
1 Inicialize K=número de vizinhos mais próximo, AM=0;
2 Para todo arranjo(A) de instância gerado pelo Q-Learning faça:
3  Selecione uma instância do conjunto;
4  p ← instância selecionada;
5    Para as demais instâncias faça:
       Com o algoritmo K-NN encontre as K instâncias mais próximas de p;
       Selecione as K instâncias mais próximas;
       Am ← somatória do Atributo-meta das instâncias mais próximas de p;
          Atualize o Atributo-meta de p com Am/K;
   Volte ao passo 3 até que todas as instâncias tenham sido selecionadas;
6 Retorne (Atributo-meta);
7 Fim.
```

Figura 5.1: Pseudocódigo do algoritmo que aproxima as instâncias e estima seu atributo-meta

A partir de um arranjo de instância qualquer gerado pelo Q-Learning, é selecionada uma instância *p* e encontrado seus *K* vizinhos mais próximos, utilizando o algoritmo K-NN com a Equação 5.1 (co-seno). O atributo-meta de *p* e de suas instâncias mais próximas serão somados e divididos por *K*. Esse valor estimado será a nova recompensa fornecida a tabela de aprendizagem do agente com o algoritmo Q-Learning.

Arranjo de instâncias pode ser gerado a partir dos valores da tabela de aprendizagem do agente com o algoritmo Q-Learning em diferentes formas. Quando o agente encerra um ciclo de aprendizagem ou encerra sua iteração, o arranjo com as instâncias é gerado e o valor do atributo-meta é estimado[6] com o algoritmo K-NN. Esses arranjos podem ser definidos da seguinte forma:

- Como conjunto completo de todos os arranjos adquiridos pelo agente ao longo da execução do Q-Learning, ou seja, as instâncias usadas são aquelas geradas desde a primeira iteração do agente com o ambiente. Essa forma é denominada de K-NN $Arranjo[completo]$;
- Como um conjunto de arranjos formados por um número N de ciclos realizados pelo agente, onde cada ciclo realizado pelo agente gera um novo arranjo. Essa maneira leva o nome de K-NN $Arranjo[n\_ciclos]$;

[6] Atributo meta consiste no valor estimado por recompensas.

- Como o arranjo formado pelo último ciclo de aprendizagem do agente, agora chamado de K-NN $Arranjo[ultimo\_ciclo]$;
- E finalmente, na tentativa de encontrar a melhor maneira no uso das instâncias é utilizado o último arranjo adquirido pelo agente calculado sobre o último arranjo estimado pelo algoritmo K-NN, formalizado pelo nome de K-NN $Arranjo[ultimo\_ciclo \approx K-nn]$.

A seguir é apresentado na Figura 5.2 o pseudocódigo do algoritmo que representa a integração do K-NN com o Q-Learning.

```
Algoritmo integra_aprendizagem(PQ,PK);
1 Para um dado ambiente inicie PK(s,a)=0;
2 Após a interação do agente Q-Learning no ambiente, receba a
  política de ações Q;
3 Com a política de ações faça:
 a) Crie um vetor para cada ação em cada estado do ambiente, com os
    atributos: estado, ação e recompensa para a ação neste estado;
 b) Selecione um vetor A e compare com os demais vetores B
    utilizando o cálculo da distância do co-seno, e selecione um
    número K determinado de vetores mais similares ao vetor
    comparado A.
 c) PK(s,a) recebe a média calculada para o atributo recompensa
    para os K vetores selecionados, onde s e a são o estado e
    ação indicados no vetor A.
 d) Repita os passos de 'b' a 'c' para cada vetor criado.
4 Retorne PK(s,a);
5 Fim.
```

Figura 5.2: Pseudocódigo da integração do algoritmo K-NN com o algoritmo Q-Learning

5.3. O Método K-Learning

O método descrito anteriormente pode gerar políticas intermediárias, entretanto, não é garantido que essas políticas apresentem uma boa qualidade. Em alguns casos, estados que possuem valores de recompensa inadequados poderão sofrer modificações que os aproximam de um conjunto de recompensas corretas. Entretanto, também pode haver o efeito contrário: estados que possuem altas recompensas podem se tornar menos importantes. Isso ocorre porque todos os estados terão seus valores de recompensa modificados, mesmo aqueles estados cujas recompensas sejam adequadas. O efeito prático disso pôde ser observado nos experimentos onde a eficiência dos métodos se alternava durante a fase de aprendizado do agente, conforme ilustrado na Figura 5.3.

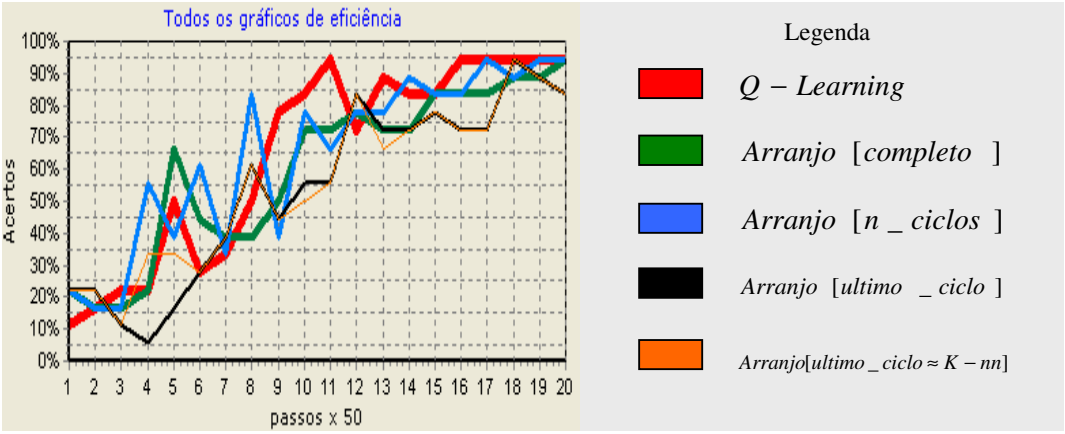

Figura 5.3: Gráfico comparativo dos algoritmos Q-Learning e K-NN

O K-Learning é um método híbrido de aprendizagem que surgiu da união dos algoritmos Q-Learning e K-NN e tenta resolver esse problema, ou seja, sempre melhorar o desempenho do agente e descobrir novas políticas de ação durante sua fase de aprendizagem. O K-Learning funciona da seguinte forma: a aprendizagem do algoritmo Q-Learning é armazenada em uma tabela de nome *PQ(s,a)*[7] e a aprendizagem do K-NN em uma tabela *PK(s,a)*[8]. Os melhores valores dessas tabelas durante a fase de aprendizagem são armazenados em uma nova tabela, denominado como *PKI(s,a)*[9].

Este método foi criado em função dos resultados obtidos em experimentos, onde se observou que a eficiência dos métodos se alternava durante a fase de aprendizagem do agente. Em muitos casos, o desempenho do K-NN tornava as recompensas descobertas pelo Q-Learning menos interessantes, pois estados que produziam acertos passavam a produzir erros[10]. Para resolver este problema, o K-Learning modifica apenas as recompensas de estados que conduziam a erros de aprendizagem. Quando um dos métodos é superior ao outro, o agente passa a utilizar a aprendizagem do mesmo. Isso faz o agente encontrar políticas de ação que maximizam seu desempenho e diminuem a quantidade de passos necessários.

[7] *PQ(s,a)* é a tabela que armazena os valores aprendidos com a política do algoritmo Q-Learning.
[8] *PK(s,a)* é a tabela que armazena os valores aprendidos com a política do algoritmo K-NN.
[9] *PKI(s,a)* é a tabela de aprendizagem do método K-Learning.
[10] A metodologia MAP é utilizada para definir acertos e erros do agente e foi discutida em detalhes na Seção 4.2.

```
Algoritmo KLearning(PQ,PK,PKI);
1 Para um dado ambiente inicialize PQ(s,a)=0, PK(s,a)=0, PKI(s,a)=0;
2 Para cada iteração usando o algoritmo Avaliação_Política repita:
  Enquanto Condição de Parada != Verdadeiro faça:
     Se PQ(s,a) < PK(s,a)
       Então
          PKI(s,a) ← PQ(s,a);
       Senão
         Continue aprendizagem usando PQ(s,a);
2 Retorne (PQ,PKI);
3 Fim.
```

Figura 5.4: Pseudocódigo do método K-Learning

A Figura 5.4 mostra que para cada iteração o desempenho dos algoritmos Q-Learning e K-NN são comparadas. O algoritmo *Avaliação_Política* foi definida na Seção 4.2 e faz parte da metodologia MAP. Quando o desempenho de um dos algoritmos é superior, essa aprendizagem passa para um a tabela `PKI`, que representa a aprendizagem do método K-Learning. Os resultados desse método híbrido de aprendizagem são apresentados nos parágrafos seguintes. Dessa forma o agente pode encontrar políticas de ação que maximizam seu desempenho e diminuem a quantidade de passos necessários.

5.4. Algoritmos Utilizados nos Experimentos

A seguir são descritos os algoritmos e parâmetros utilizados nos experimentos utilizando a metodologia MAP.

5.4.1. Algoritmo Q-Learning

Para o algoritmo Q-Learning, os valores definidos para cada situação de congestionamento representarão as recompensas que o agente irá receber como reforço para o seu aprendizado. Para o aprendizado do agente com o algoritmo Q-Learning, várias escolhas precisam ser feitas na tentativa de fazer com que o agente chegue o mais próximo possível de uma política ótima para o ambiente simulado. Parâmetros como: recompensas, fator de desconto, taxa de aprendizagem e a porcentagem de probabilidade de escolha da melhor ação irão determinar a eficiência do agente Q-Learning.

A política de recompensas inicialmente definida determina que, ao realizar uma ação, o agente recebe como recompensa a soma dos valores das situações vizinhas ao estado para o qual ele se moveu, ainda somada com o valor da situação desse estado. Após alguns

experimentos realizados, a política de recompensas foi alterada da seguinte forma: o agente ao realizar uma ação, recebe como recompensa apenas o valor da situação do estado para onde ele se moveu. Os demais valores dos parâmetros necessários ao Q-Learning foram definidos de acordo com os experimentos discutidos na Seção 4.4.

5.4.2. Algoritmo K-NN

Para o aprendizado com o algoritmo K-NN, serão utilizadas instâncias do aprendizado do Q-Learning. Arranjos de instâncias serão adquiridos em intervalos de passos do Q-Learning. Previamente, este intervalo é definido com valor de 50 passos, ou seja, a cada 50 iterações do agente Q-Learning o algoritmo K-NN recebe vetores (Tabela 5.1) que representam as recompensas que o agente adquiriu para cada ação em cada estado do ambiente. A partir desses vetores, o algoritmo K-NN irá medir a similaridade entre eles utilizando a distância co-seno, apresentada na Equação 5.1, selecionando os vetores que possuírem valores mais parecidos:

$$\cos\theta = \cos(\vec{d}_j, \vec{q}) = \frac{\vec{d} \cdot \vec{q}}{\left|\vec{d}\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{i} (d_i \cdot q_i)}{\sum_{i=1}^{i} d_i^{\,2} \cdot \sum_{i=1}^{i} q_i^{\,2}} \tag{5.1}$$

no qual $\vec{d}$ e $\vec{q}$ representam os vetores da tabela *PK(s,a)*, d e q são os elemento de cada vetor e i é o índice da posição de cada elemento. Antes de comparar o desempenho do Q-Learning, K-NN e K-Learning são necessários estabelecer os melhores parâmetros para o K-NN como feito anteriormente. Para isso são descritos nos parágrafos a seguir alguns experimentos realizados.

Os experimentos sobre o algoritmo K-NN foram realizados em ambientes de 16 estados com 200 iterações, 25 estados com 500 iterações e ambientes de 64 estados com 3000 iterações. Os demais parâmetros utilizados serão os melhores encontrados nos experimentos com o algoritmo Q-Learning, citados anteriormente.

Nos experimentos a seguir, foi avaliada a eficiência média[11] do algoritmo K-NN quanto ao número de vizinhos utilizados.

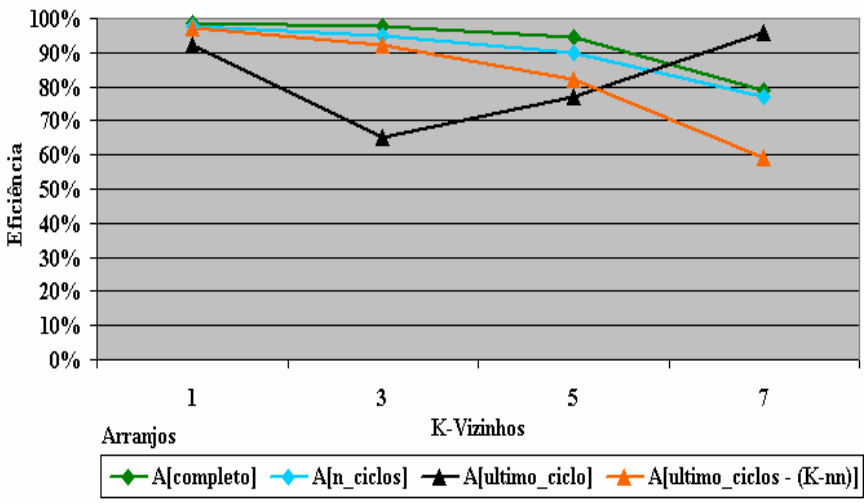

Figura 5.5: Eficiência do algoritmo K-NN em ambientes com 16 estados

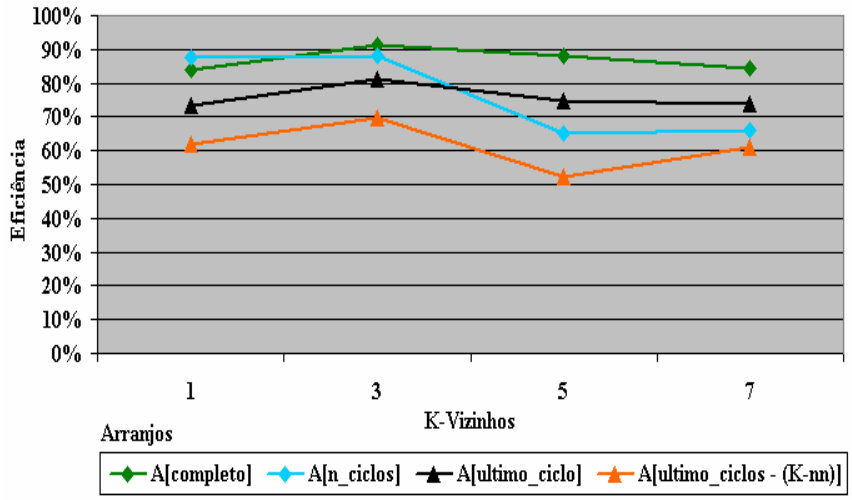

Figura 5.6: Eficiência do algoritmo K-NN em ambientes com 25 estados

[11] A eficiência média de cada algoritmo é calculada pela somatória da eficiência final de cada amostra dividido pelo número total de amostras.

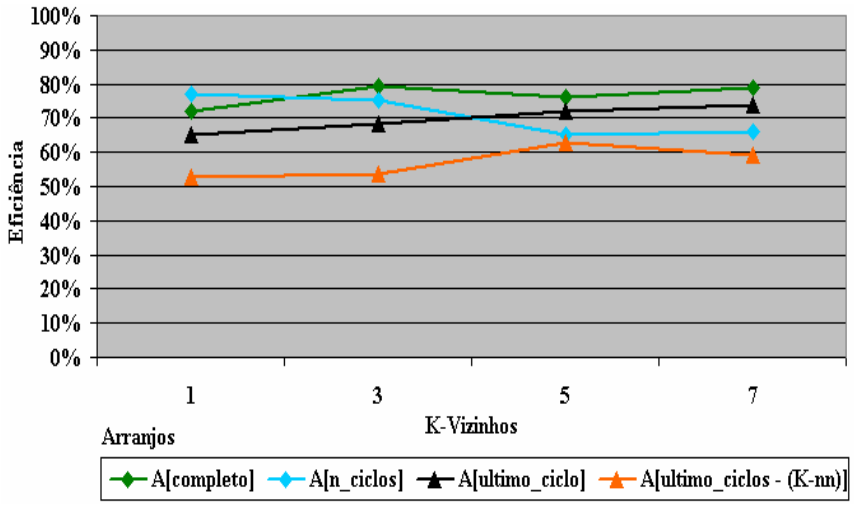

Figura 5.7: Eficiência do algoritmo K-NN em ambientes com 64 estados

As Figuras 5.5, 5.6 e 5.7 demonstram que o algoritmo K-NN obtém em geral maior eficiência quando realiza seu aprendizado utilizando todos os conjuntos de instâncias adquiridas a partir do Q-Learning (K-NN $Arranjo[completo]$). No entanto, para este tipo de aprendizado, quanto maior o número de vizinhos utilizados menor foi sua eficiência.

Para o tipo de aprendizado do K-NN onde é utilizado um número de conjunto de últimas instâncias do agente Q-Learning (K-NN $Arranjo[n\_ciclos]$), a eficiência foi maior em alguns experimentos e menor em outros, não mantendo uma regularidade e diminuindo a confiabilidade da técnica, pois ela não consegue garantir eficiência média de melhor qualidade.

No tipo de aprendizado utilizando somente o último conjunto de instâncias de aprendizado do agente Q-Learning (K-NN $Arranjo[ultimo\_ciclo]$, a média da eficiência foi a menor comparada com os outros tipos. No entanto, apresentou as melhores eficiências utilizando 7 vizinhos. Já no tipo de aprendizado utilizando o último conjunto de instâncias de aprendizado adquirido pelo agente Q-Learning e pelo agente K-NN ($Arranjo[ultimo\_ciclo \approx K-nn]$), a média de eficiência apresentou-se inferior a todos os outros métodos de aprendizado, e também diminuiu com um número maior de vizinhos.

Portanto, o tipo de aprendizado que apresenta melhor desempenho aos demais arranjos é o tipo de aprendizado com todos os conjuntos de instâncias do agente Q-Learning $Arranjo[completo]$, utilizando apenas 3 vizinhos mais próximos. Assim, esse tipo de

aprendizado será adotado ao longo do Capítulo 5 para os demais experimentos. A seguir, é apresentado um comparativo da eficiência dos algoritmos Q-Learning e o K-NN.

5.5. Algoritmo Q-Learning *vs.* K-NN

Os experimentos para comparação da eficiência do algoritmo Q-Learning e o algoritmo K-NN foram realizados em ambientes de tamanho entre 16 a 64 estados com 1000 iterações. A quantidade de amostra para todos os experimentos foi descrito na Seção 4.4.

A Tabela 5.2 mostra que o algoritmo Q-Learning é superior ao algoritmo K-NN. A eficiência média dos algoritmos utilizados neste trabalho é calculada pela somatória da eficiência final de cada amostra dividido pelo número total de amostras.

Tabela 5.2: Comparativo dos algoritmos Q-Learning e K-NN

Algoritmo Q-Learning		**Algoritmo K-NN**	
Quantidade de estados	**Eficiência (%)**	**Quantidade de estados**	**Eficiência (%)**
16	98	16	96
25	94	25	88
64	86	64	79

Porém, nos experimentos realizados, notou-se que em algumas amostras, a eficiência do algoritmo K-NN foi maior que a do Q-Learning, e em alguns casos, conseguindo alcançar a política ótima antes do agente Q-Learning, como apresentado no gráfico da Figura 5.8 pelo $Arranjo[completo]$.

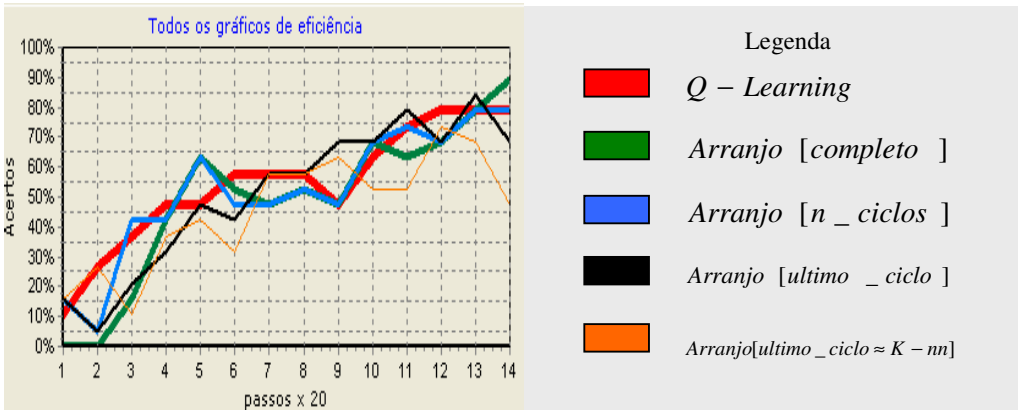

Figura 5.8: Gráfico da superioridade de um arranjo com o K-NN

Assim, se em alguns momentos o algoritmo K-NN consegue otimizar a eficiência do algoritmo Q-Learning, pode-se utilizar estes dois algoritmos em conjunto para o aprendizado em busca da política ótima. Neste tipo de aprendizado, durante a interação do agente no ambiente, a base de conhecimento utilizada seria a do algoritmo que estivesse com a maior eficiência (*PQ(s,a)* ou *PK(s,a)*), sendo aplicado então, o algoritmo Q-Learning na base mais eficiente. Como visto nas seções anteriores, este é o conceito-chave do algoritmo K-Learning. Os experimentos a seguir comparam a eficiência dos algoritmos Q-Learning e K-NN e K-Learning.

5.6. Comparação dos Algoritmos Q-Learning, K-NN e K-Learning

Os experimentos para comparação da eficiência dos algoritmos Q-Learning, K-NN e o método híbrido K-Learning foram realizados em ambientes de tamanho entre 16 a 64 estados. As Figuras 5.9 a 5.13 apresentam a comparação dos três algoritmos em ambientes e estados com situações diferentes.

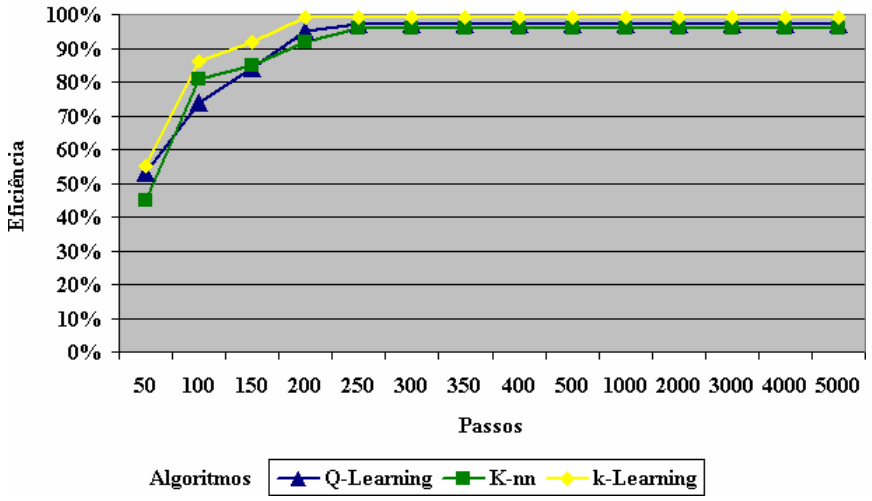

Figura 5.9: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 16 estados

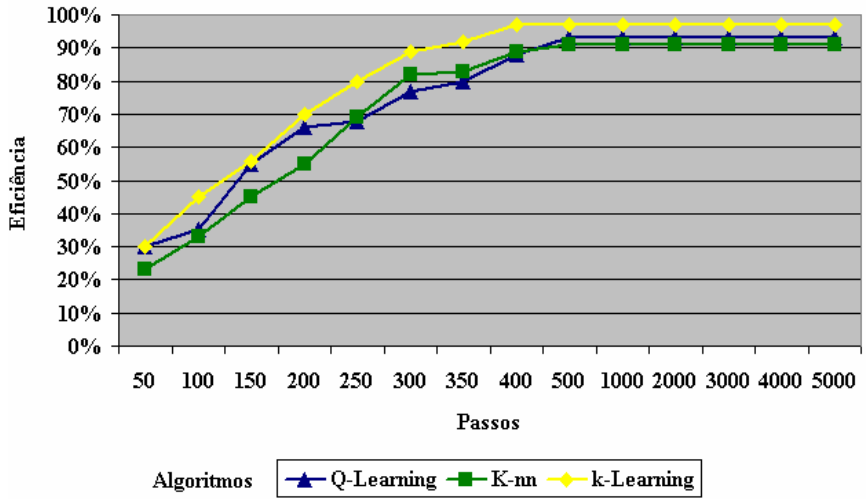

Figura 5.10: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 25 estados

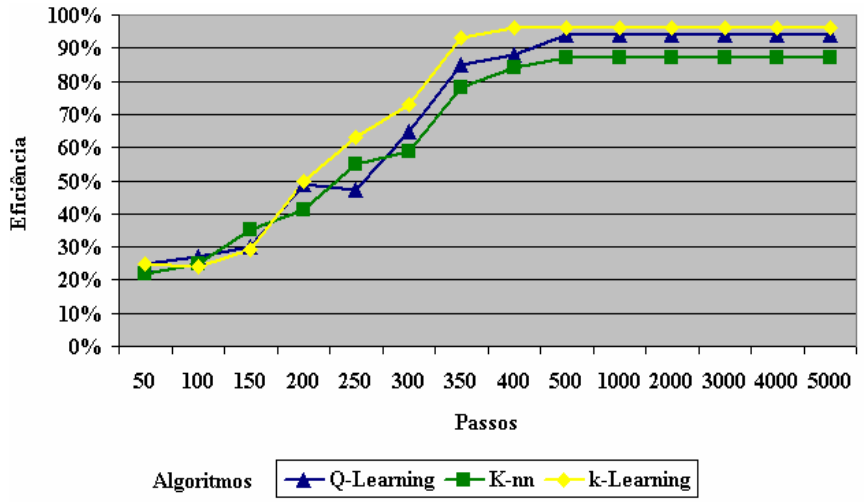

Figura 5.11: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 36 estados

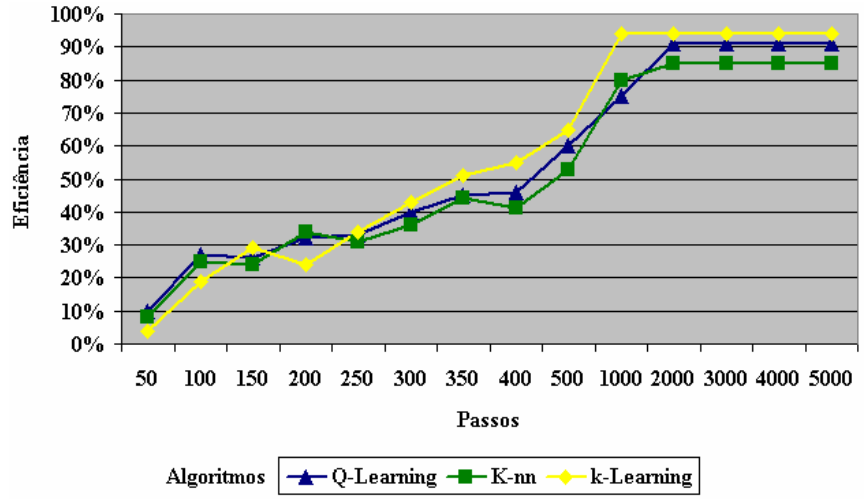

Figura 5.12: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 49 estados

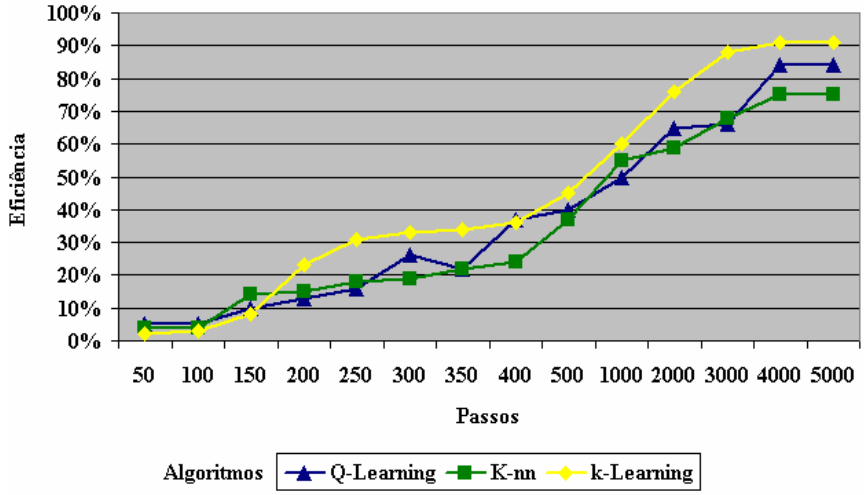

Figura 5.13: Eficiência dos algoritmos Q-Learning, K-NN e K-Learning em ambientes com 64 estados

As Figuras 5.9 a 5.13 apresentam que, de um modo geral, o método K-Learning apresenta eficiência superior aos algoritmos Q-Learning e K-NN durante a aprendizagem do agente. Geralmente, o método K-Learning obtém desempenho superior em cada fase (ciclo) do aprendizado do agente. Isso fez com que ele diminuísse o número de passos necessários para o agente encontrar uma boa política de ação. Em ambientes com 16 estados houve em média uma diminuição de 18% do número de passos. Para ambientes de 25 estados a redução foi em média de 20%. Para ambientes de 36 estados a redução foi em média de 13%. Para

ambientes de 49 estados a redução foi em média de 15%. E finalmente, para o ambiente de 64 estados o método K-Learning conseguiu otimizar em média 12% os passos. Portanto, o método K-Learning converge mais rapidamente para uma boa política de ações e diminui o número de iterações.

A Tabela 5.3 apresenta a diferença média de eficiência dos algoritmos Q-Learning e K-NN em relação ao método K-Learning. A quantidade de amostras usada nos experimentos é a mesma descrita na Seção 4.4.

Tabela 5.3: Comparativo da eficiência média dos algoritmos Q-Learning e K-NN com o método K-Learning.

Quantidade de Estados	**Q-Learning (%)**	**K-NN (%)**
16	- 2	- 3
25	- 4	- 6
36	- 2	- 9
49	-3	-8
64	- 7	- 16

É possível observar que o método híbrido é superior tanto na eficiência média quanto ao número de iterações necessárias para encontrar uma boa política. Isso mostra que esse método consegue se adaptar mais rapidamente ao problema proposto. O seu bom desempenho é decorrente de técnicas de estimação que conseguem aproximar os valores aprendidos pelo agente aos valores de uma boa política de ações. A Tabela 5.4 apresentada a percentagem de superioridade do método K-Learning em número de passos.

Tabela 5.4: Otimização do número de passos do método K-Learning

Quantidade de Estados	**Otimização do número de passos (%)**
16	18
25	20
36	13
49	15
64	12

5.7. Otimizando os Algoritmos de Aprendizagem

Utilizamos também algumas heurísticas que permitiram melhorar o desempenho dos algoritmos de aprendizagem estudados nas seções anteriores, fazendo com que políticas de ação fossem descobertas mais rapidamente.

A primeira heurística utilizada para aumentar o desempenho dos algoritmos foi evitar a visita do agente aos estados aprendidos depois de um número *X* de iterações. Assim, o agente passou a buscar os estados com recompensas mais baixas, fazendo que a soma das recompensas finais fossem acumuladas mais rapidamente. As Figuras 5.14 a 5.22 apresentam uma comparação dos algoritmos Q-Learning, K-NN e K-Learning com e sem a aplicação da heurística.

A segunda heurística consiste em usar a estratégia de exploração aleatória - $\varepsilon - Greedy$, onde o agente passa a escolher ações aleatórias com probabilidade em ε $(0 \leq \varepsilon \leq 1)$ no qual ε é o parâmetro que define a taxa de exploração, ou seja, quanto menor o valor de ε, menor a probabilidade de se fazer uma escolha aleatória. Esta estratégia é aplicada somente ao método híbrido K-Learning, devido a sua melhor adaptação com o $\varepsilon - Greedy$. As Figuras 5.23 e 5.24 apresentam a eficiência da heurística com o K-Learning.

5.7.1. Algoritmo Q-Learning *vs.* Algoritmo Q-Learning com Heurística

A aplicação da primeira heurística no algoritmo Q-Learning fez seu desempenho aumentar em média de 10% nos três ambientes. Nos ambientes com 25 estados o método melhorou o desempenho do algoritmo em 6%. Para os ambientes com 49 estados o desempenho aumentou em 7%. E finalmente em ambientes com 64 estados a heurística otimizou o algoritmo em 16%.

As Figuras 5.14, 5.15 e 5.16 apresentam a evolução do algoritmo Q-Learning usando heurística.

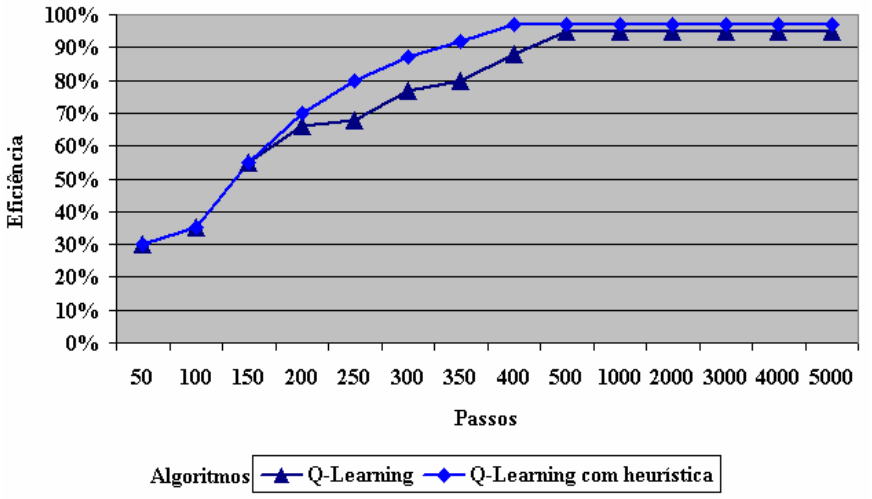

Figura 5.14: Q-Learning *vs*. Q-Learning com heurística em ambientes com 25 estados

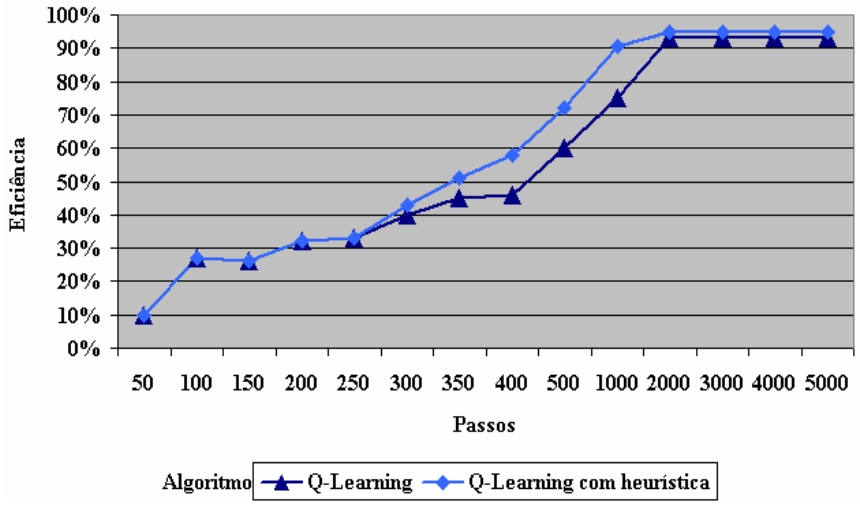

Figura 5.15: Q-Learning *vs*. Q-Learning com heurística em ambientes com 49 estados

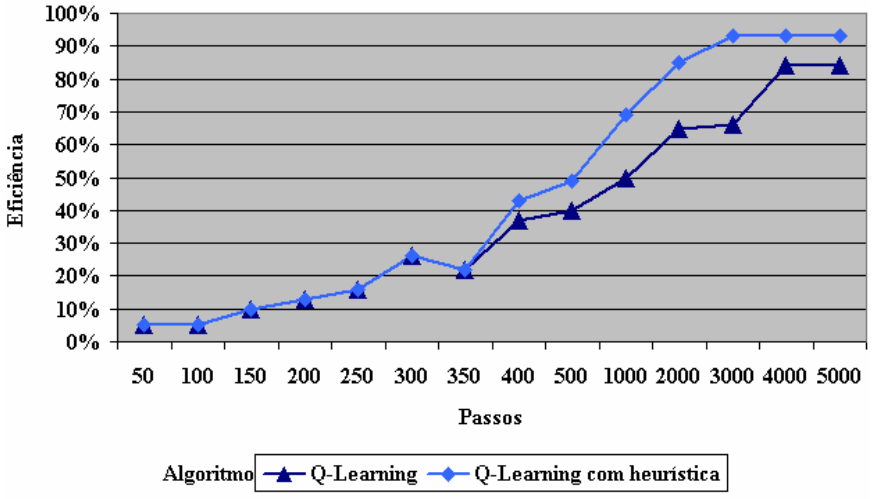

Figura 5.16: Q-Learning *vs*. Q-Learning com heurística em ambientes com 64 estados

5.7.2. Algoritmo K-NN *vs.* Algoritmo K-NN com Heurística

O algoritmo K-NN com a primeira heurística conseguiu aumentar em média 13% o desempenho nos três ambientes. Esta heurística otimizou o desempenho do algoritmo em 7% nos ambientes de 25 estados, 11% em ambientes de 49 estados e 22% em ambientes com 64 estados.

As Figuras 5.17, 5.18 e 5.19 apresentam o comparativo do algoritmo K-NN com e sem heurística.

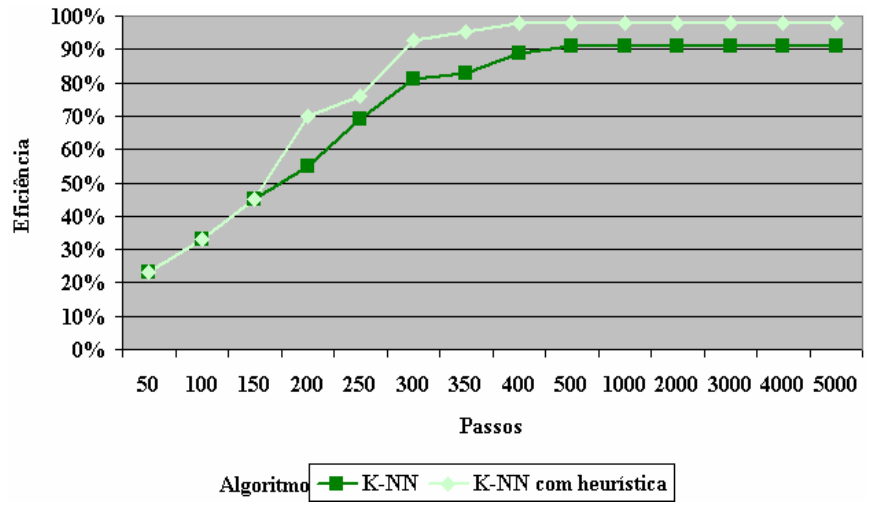

Figura 5.17: K-NN *vs.* K-NN com heurística em ambientes com 25 estados

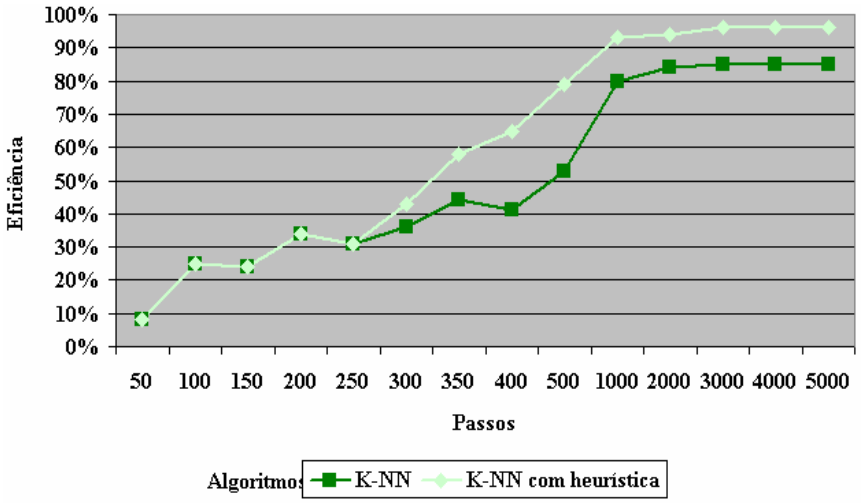

Figura 5.18: K-NN *vs.* K-NN com heurística em ambientes com 49 estados

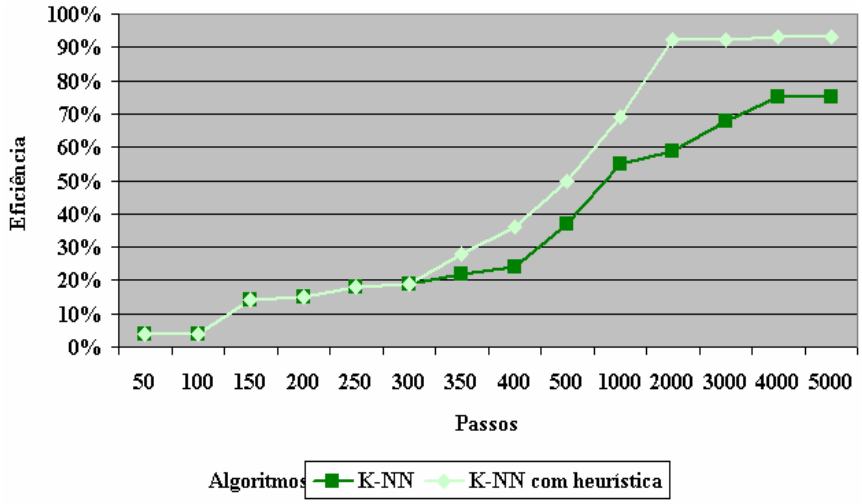

Figura 5.19: K-NN *vs.* K-NN com heurística em ambientes com 64 estados

5.7.3. Algoritmo K-Learning *vs.* Algoritmo K-Learning com Heurística

E finalmente para o algoritmo K-Learning, a primeira heurística conseguiu aumentar seu desempenho em média de 6% nos três ambientes. Em ambientes com 25 estados o desempenho aumentou em 3%, ambientes de 49 estados o desempenho aumentou em 7% e 8% foi o aumento em ambientes com 64 estados.

As Figuras 5.20, 5.21 e 5.22 apresentam a evolução do K-Learning com heurística.

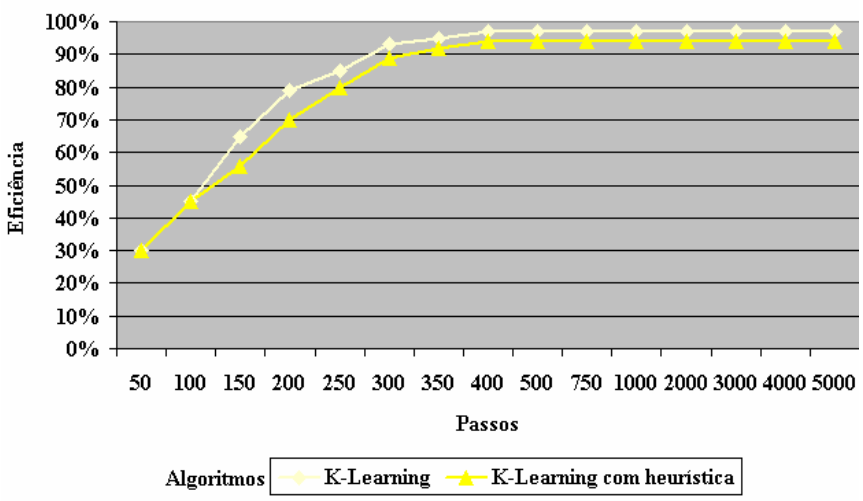

Figura 5.20: K-Learning *vs.* K-Learning com heurística em ambientes com 25 estados

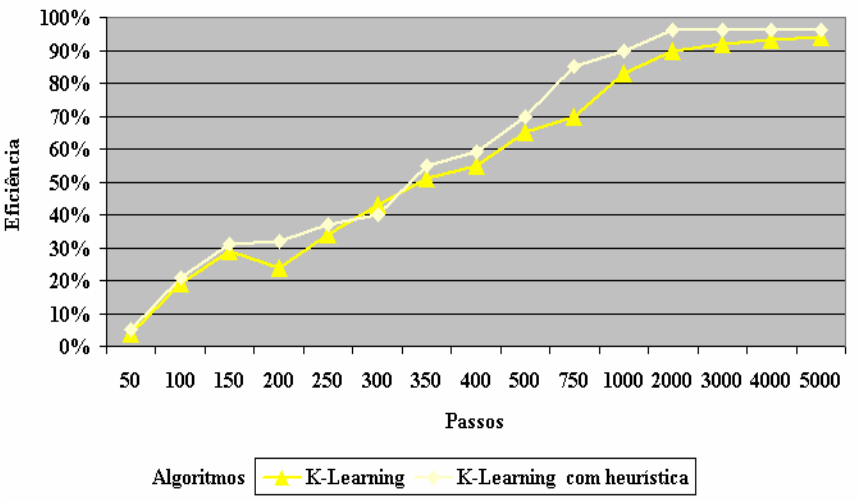

Figura 5.21: K-Learning *vs.* K-Learning com heurística em ambientes com 49 estados

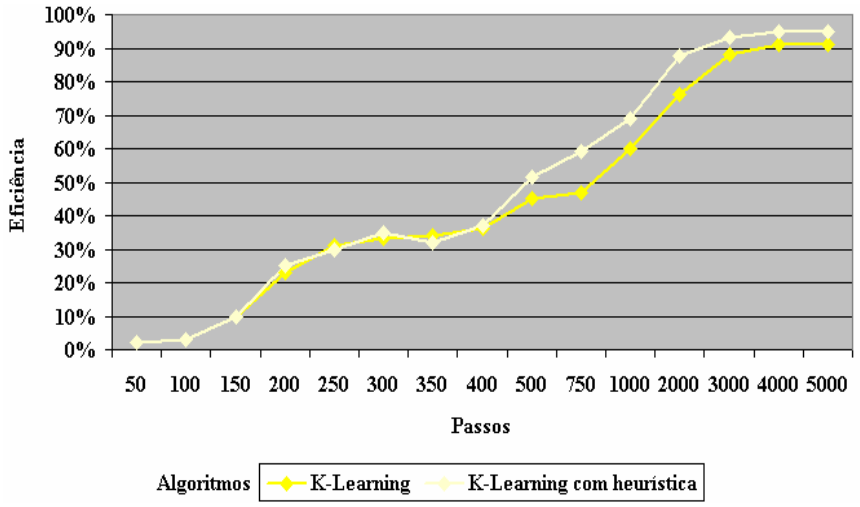

Figura 5.22: K-Learning *vs.* K-Learning com heurística em ambientes com 64 estados

Assim, pode-se observar que esta heurística é eficiente aos algoritmos de aprendizagem com algumas observações; i) o método é eficiente somente após um número *X* de iterações. Isso acontece porque os estados visitados inicialmente possuem recompensas pequenas, devido às poucas iterações do agente no começo da aprendizagem; ii) é necessário um número de iterações até que as recompensas sejam maiores. Os algoritmos com heurística apresentam melhor desempenho a partir de um número razoável de iterações; iii) é utilizado junto com este método o algoritmo K-NN estimando somente os estados com recompensas incorretas.

A segunda heurística - estratégia de exploração aleatória $\varepsilon - Greedy$ garantiu aos algoritmos encontrar políticas ótimas de ação ($PQ*$, $PK*$, $PKI*$), maximizando a soma das recompensas finais realizando escolha de ações de maneira aleatória. A estratégia $\varepsilon - Greedy$ foi aplicada somente quando o desempenho dos algoritmos estacionaram em uma política diferente da ótima.

Os resultados apresentados com a estratégia $\varepsilon - Greedy$ são do algoritmo K-Learning, pois este algoritmo é o que apresenta melhor desempenho com esta estratégia. Os algoritmos Q-Learning e K-NN precisaram de um número de iterações muito maior do que apresentado nos experimentos das seções anteriores.

A Equação 5.2 apresenta a estratégia $\varepsilon - Greedy$ aplicada ao método K-Learning:

$$PKI(s,a) \leftarrow PKI(s,a) + \alpha[r + y.(A_{random}(\varepsilon)(\max_a PKI(s',a) - PKI(s,a)))] \quad (5.2)$$

onde $PKI(s,a)$ é o valor aprendido pelo método K-Learning com ação aleatória $A_{random}(\varepsilon)$ em um estado s.

Esta estratégia garante ao método híbrido de aprendizagem encontrar políticas de ação ótima $PKI*$ em qualquer ambiente, pois nos estados incorretos o agente executará ações de maneira aleatória, maximizando a soma das recompensas finais.

A Figura 5.23 apresenta a evolução do método usando o $\varepsilon - Greedy$ com valor diferente de 100% em uma política de ação qualquer em ambientes com 25 estados. Os pontos anotados na figura (passos 500 e 550) informam o momento do uso da estratégia $\varepsilon - Greedy$.

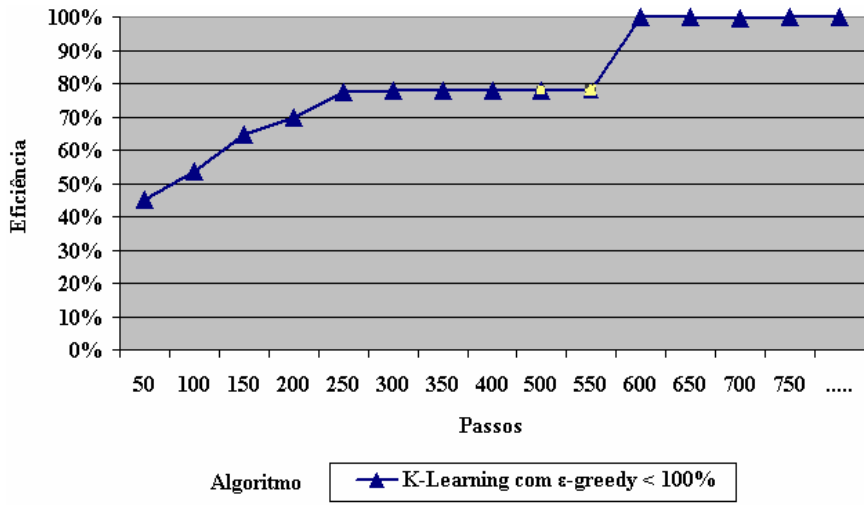

Figura 5.23: K-Learning $\varepsilon - Greedy$ com < 100%

A estratégia $\varepsilon - Greedy$ foi aplicada somente quando o desempenho do K-Learning estacionou em uma política diferente de $PKI*$, ou seja, desempenho menor que 100%.

Nos experimentos realizados, a estratégia $\varepsilon - Greedy$ não conseguiu bons resultados com o método K-Learning quando usado desde as primeiras iterações. A estratégia $\varepsilon - Greedy$ com o método K-Learning somente consegue boas políticas quando usado depois da sua aprendizagem estar estabilizada. Isso acontece, porque inicialmente o método K-Learning não possui bom desempenho devido aos seus valores de aprendizado ainda serem

baixos (recompensas menores). Neste caso, quando o agente utiliza a política corrente ele possivelmente cometerá erros na atualização das recompensas.

A Figura 5.24 apresenta o comparativo do método K-Learning com $\varepsilon - Greedy$ igual e diferente de 100% em toda a aprendizagem.

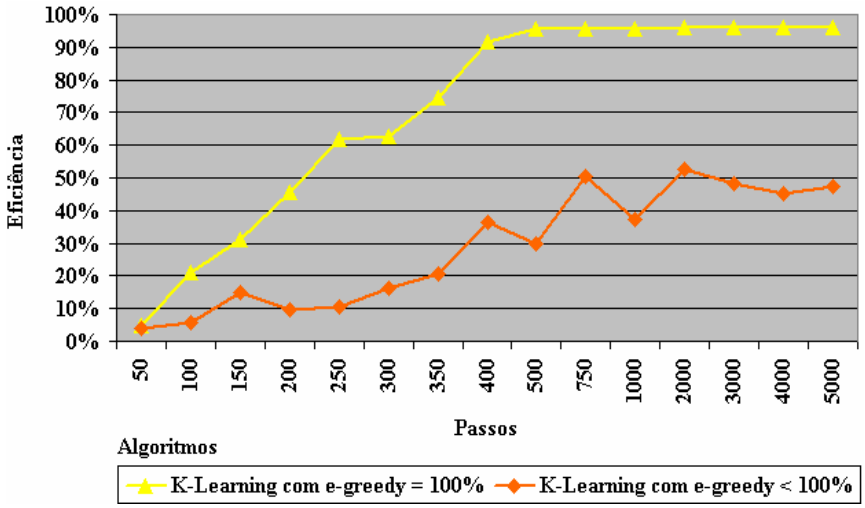

Figura 5.24: K-Learning com $\varepsilon - Greedy$ em 100% *vs.* K-Learning com $\varepsilon - Greedy$ menor que 100%

5.8. Considerações Finais

Neste capítulo foram apresentados os principais resultados dos algoritmos baseados em instâncias e do método K-Learning. O algoritmo K-NN nos permitiu estimar novos valores que são gerados pelo algoritmo Q-Learning, em qualquer fase de aprendizagem, na intenção de descobrir novas políticas de ação. Esses valores estimados pelo K-NN não garantem que esses novos valores irão convergir mais rapidamente para uma boa política. Isso ocorre porque novos valores de recompensas podem alterar os estados aprendidos corretamente pelo Q-Learning anteriormente. O método híbrido consegue encontrar boas políticas de ação, justamente por utilizar os melhores valores do Q-Learning e estimar corretamente valores de recompensas que até então se apresentavam incorretos. Apresentamos também alguns métodos que otimizaram o desempenho dos algoritmos de aprendizagem utilizados nos experimentos.

Capítulo 6

Conclusão

Este trabalho apresentou duas contribuições principais: a primeira é o desenvolvimento de uma metodologia genérica – MAP, que é capaz de avaliar a qualidade de políticas descobertas por agentes autônomos adaptativos que utilizam algoritmos de AR, possibilitando que o desempenho de um agente seja estimado sem a necessidade de conhecer o domínio. A segunda é o desenvolvimento de um método híbrido de aprendizagem, unificando técnicas de AR com algoritmos de aprendizagem baseada em instâncias, permitindo que boas políticas fossem descobertas em um número razoável de passos.

Técnicas de AR e algoritmos baseados em instâncias podem ser utilizados conjuntamente na construção de métodos híbridos inteligentes para satisfazer determinadas classes de problemas. Esses algoritmos podem encontrar soluções ótimas para problemas onde agentes autônomos interagem com diferentes ambientes.

O uso de agentes autônomos adaptativos é importante em problemas onde exemplos de comportamentos para suas ações não estão disponíveis. No problema proposto nesta dissertação, o agente conseguiu de maneira satisfatória interagir com um ambiente de simulação de tráfego viário para aprender políticas de movimentação. Além disso, foram propostos algoritmos alternativos de descoberta de políticas.

Para medir o desempenho do agente, desenvolvemos uma nova metodologia de avaliação – MAP; que permitiu avaliar precisamente o desempenho dos algoritmos de aprendizagem. A metodologia permite observar em detalhes o desempenho dos algoritmos em qualquer ciclo de aprendizagem. Isso foi fundamental para a construção do método K-Learning, pois pudemos deduzir que estes algoritmos poderiam ser usados paralelamente, observando que o desempenho dos algoritmos Q-Learning e K-NN se alternavam constantemente.

Os experimentos realizados com MAP mostram que a metodologia é robusta em ambientes parcialmente conhecidos e complexos e pode auxiliar na determinação adequada dos parâmetros de algoritmos de AR. Isso é importante porque esses fatores têm grande relevância no desempenho da aprendizagem do agente.

O algoritmo K-Learning usado em conjunto com técnicas de estimação, consegue convergir rapidamente para uma boa política de ação. Isso ocorre porque os estados que ainda foram pouco explorados pelo Q-Learning, recebem valores que podem levar o agente a encontrar políticas intermediárias. Dessa forma, o problema de exploração repetida de todos os estados do ambiente observado no Q-Learning é reduzido significativamente.

Experimentos realizados com o algoritmo K-NN mostram que mesmo tendo um custo computacional mais elevado, seus resultados são satisfatórios, pois ele consegue estimar valores encontrando soluções superiores ao do Q-Learning no decorrer de sua aprendizagem.

O método híbrido K-Learning integrou a forma robusta de aprendizagem do Q-Learning com a maneira eficaz de estimar os valores do aprendizado do algoritmo K-NN descobrindo novas políticas de ação. Este método híbrido de aprendizagem consegue otimizar o agente em termos de convergência, número de iterações e qualidade da política de ações.

Os resultados observados com MAP e o bom desempenho do método K-Learning, nos motivam a realizar novas pesquisas em ambientes maiores e mais complexos, como por exemplo, ambientes dinâmicos. Alguns experimentos preliminares neste tipo de ambiente mostram que o método é robusto para este tipo de problema.

Apesar dos resultados obtidos serem encorajadores, algumas diretivas futuras são merecedoras de investigações para responder algumas questões como:

- Comparar diferentes formas a atualização da política atual. A atualização da política corrente poderia ser feita de forma parcial (apenas com os melhores valores de recompensa) e não integralmente como é feito atualmente.
- Uma função heurística poderia ser usada para a aceleração da AR [BIA04], indicando a escolha da ação tomada e limitando dessa forma, o espaço de busca do agente.
- O uso de uma arquitetura multi-agente poderia ser indicado para a exploração de regiões mais distantes do estado final em que as recompensas dos estados são menores. Alguns desses conceitos podem ser encontrados em Almeida et al. [ALM04].

- Uma outra questão consiste em avaliar os algoritmos em ambientes dinâmicos e também em ambientes com ruído. Como o K-Learning utiliza resultados de algoritmos diferentes, *a priori*, ele deverá ser mais robusto em situações onde os valores de recompensa variam inexplicavelmente (ruído), pois sabe-se que o K-NN é um método menos susceptível a ruídos nos dados de aprendizagem.
- Finalmente, o uso de vários agentes poderia ser usado para explorar ambientes maiores, onde cada agente ficaria responsável em explorar determinadas regiões de um certo ambiente. Assim, a aprendizagem de cada agente seria propagada para os demais agentes, diminuindo o número de iterações até encontrar uma boa política de ações.

Mesmo com os bons resultados alcançados pelo método K-Learning, esses estudos complementares são importantes para a evolução deste método, pois a diversidade de experimentos a serem produzidos é enorme, o que faz surgir novos cenários a serem estudados. MAP e K-Learning poderão ser utilizados em classes de problemas similares ao apresentado neste trabalho, pois se mostram robustos em experimentos realizados com ambientes de estados com diferentes níveis de congestionamentos e bloqueios.

Referências Bibliográficas

[ABR01] ABRAMSON, M.; WECHSLER, H. *Competitive reinforcement learning for combinatorial problems*. In INNS-IEEE International Joint Conference on Neural Networks (IJCNN), 2001, Washington, DC, USA, v. 4, p. 2333-2338.

[AHA91] AHA, D. W.; KIBLER, D.; ALBERT, M. K. *Instance-based Learning Algorithms*, Machine Learning 6(1), 1991, p. 37-66.

[ALM04] ALMEIDA, A.; RAMALHO, G. L.; SANTANA, H. P.; TEDESCO, P.; MENEZES, T. R.; CORRUBLE, V.; CHEVALEYRE, Y. *Recent Advances on Multi-Agent Patrolling*. In Advances in Artificial Intelligence – SBIA 2004. LNAI 3171, p. 474-483. Berlin: Springer.

[AND01] ANDRE, D.; RUSSELL, S. J. *State Abstraction for Programmable Reinforcement Learning Agents.* EECS Department, University of California, Berkeley, 2001. Technical Report, UCB/CSD-01-1156.

[BEN75] BENTLEY, J. L. *Multidimensional binary search trees used for associative searching*. Communications of the ACM, 1975, v. 18, n. 9, p. 509-517.

[BER87] BERTSEKAS, D. P. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[BIA04] BIANCHI, R. A. C. *Uso de heurísticas para a aceleração do aprendizado por reforço*. Tese de Doutorado - ESC POLITECNICA, Universidade de São Paulo, São Paulo, 2004.

[BIT01] BITTENCOURT, G. *Inteligência Artificial – Ferramentas e Teorias*. Segunda Edição. Editora da UFSC – Florianópolis, 2001.

[BOG03] BOGO, L. H. *Criação de Comunidades Virtuais a Partir de Agentes Inteligentes: Uma Aplicação em e-learning*, dissertação de mestrado, Universidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia de Produção, 2003.

[BOR02] BORLAND SOFTWARE CORPORATION, *Borland C++ Builder,* Enterprise Suite, version 6.0, 2002.

[BRA97] BRADSHAW, J. M. *An Introduction to software Agents*. In: Bradshaw, J. M. (Ed.). Software Agents. Massachussetts: MIT Press 1997.

[BRE84] BREIMAN, L.; FRIEDMAN, J.H.; OLSHEN, R.A.; STONE, C.J. *Classification and Regression Trees*. Wadsworth International: Califórnia – USA, 1984.

[BRE96] BREIMAN, L. *Bagging predictors*. Machine Learning, 1996, 24(2): p. 123–140.

[BRO86] BROOKS, R. A. *A robust layered control system for a mobile robot*. IEEE Journal of Robotics and Automation., 1986, vol. RA-2, p. 14-23.

[BRO90] BROOKS, R. A. *Elephants don't play chess*. Maes, P., editor, Designing Autonomous Agents, pp. 3-15. The MIT Press: Cambridge, MA, 1990.

[BRO91a] BROOKS, R. A. *Intelligence without reason*. Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), 1991, p. 569-595, Sydney, Australia.

[BRO91b] BROOKS, R. A. *Intelligence without representation*. Artificial Intelligence, 1991, 47:139-159.

[COE05] COELHO, P. S. S. *Um Sistema Para Indução de Modelos de Predição Baseados em Árvores*. COPPE/UFRJ, Tese de Doutorado – Universidade Federal do Rio de Janeiro, COPPE, 2005.

[COR90] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. *Introduction to Algorithms*. McGraw-Hill, second edition, 1990.

[CRI96] CRITES, R. H.; BARTO, A.G. *Improving Elevator Performance Using Reinforcement Learning*. Advances in Neural Information processing Systems 8. MIT Press, Cambridge, MA, 1996, p. 1017–1023.

[DAV95] DAVIDSSON, P. *On the Concept of Concept in the Context of Autonomous Agents*, WOCFAI 95: Proceedings of the Second World Conference on the Fundamentals of Artificial Intelligence, July, Paris - France. 1995, p. 85-96.

[DOW01] DOWNING, K. L. *Reinforced Genetic Programming Genetic Programming and Evolvable Machines*. Genetic Programming and Evolvable Machines, vol. 2, no. 3, September 2001, Springer, p. 259-288(30).

[ENE03] ENEMBRECK, F. *Contribution à la conception d'agentes assistants personnels adaptatifs*, Thèse de Docteur. Universite de Technologie de Compiegne U. F. R. de Sciences Et Technologie, 2003.

[ERN05] ERNST, D.; GEURTS, P.; WEHENKE, L. *Tree-Based Batch Mode Reinforcement learning*. In Journal of Machine Learning Research, vol. 6, 2005, p. 503-556.

[FAR99] FARIA, G.; ROMERO, R. A. F. *Explorando o Potencial de Algoritmos de Aprendizado com Reforço em Robôs Móveis*. Proceedings of the IV Brazilian Conference on Neural Networks - IV Congresso Brasileiro de Redes Neurais. p. 237-242, July 20-22, 1999 - ITA, São José dos Campos - SP – Brasil.

[FAR00] FARIA, G.; ROMERO, R. A. F. *Incorporating Fuzzy Logic to Reinforcement Learning*. In: FUZZIEEE 2000, San Antonio-Texas, Proceedings of the 9th IEEE International Conference of Fuzzy Systems, vol. 1, 2000, p. 847-851.

[FAY96] FAYYAD, U. M.; PIATETSKY, G.; SMYTH, P. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, Cambridge MA, 1996.

[FON00] FONSECA, J. M. M. R. *Protocolos de Negociação com Coligações em Sistemas Multiagentes – Uma aplicação À Gestão Distribuída de Recursos*. Tese de Doutorado. Universidade de Nova Lisboa, Faculdade de Ciências e Tecnologia. 2000.

[FRA97] FRANKLIN, S.; GRAESSER, A. *Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents*, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, published as Intelligent Agents III, Springer-Verlag, 1997, p. 21-35.

[FRI02] FRISKE, L. M. *Utilização de opções para o controle autônomo de robôs móveis*. Dissertação de Mestrado - ITA, São José dos Campos - SP, 2002.

[GER95] TESAURO, G. *Temporal Difference Learning and TD-Gammon Communications of the ACM*, 1995, vol. 38, no. 3, p. 58-68.

[GEU06] GEURTS, P.; ERNST, D.; WEHENKEL, L. *Extremely randomized trees*. Published online in Machine Learning, March 2006, 40 pages.

[HAR68] HART, P. E.; NILSSON, N. J.; RAPHAEL, B. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs*, IEEE Trans. on Systems Science and Cybernetics, 1968, vol. SSC-4, no. 2, p. 100-107.

[HAR96] HARMON, M. E.; HARMON, S. S. *Reinforcement learning: A Tutorial*, 1996. URL: http://citeseer.ist.psu.edu/harmon96reinforcement. Acessado em 10/02/2006.

[HE06] HE, M., ROGERS, A.; LUO, X.; JENNINGS, N. R. *Designing a successful trading agent for supply chain management*. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, 2006, p. 1159-1166.

[HEN96] HENDLER, J. A. *Intelligent Agents: Where AI Meets Information Technology*. IEEE Expert, 1996, v. 11, n. 6, p. 20-23.

[HEN05] HENDERSON, J.; LEMON, O.; GEORGILA, K. *Hybrid reinforcement/supervised learning for dialogue policies from COMMUNICATOR data*. In Proc. IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems, Edinburgh, Scotland, 2005, p. 68-75.

[JEN96] JENNINGS, N. R.; WOOLDRIDGE, M. J. *Software Agents*. *IEEE Review, 1996,* 42(1): p. 17-20.

[KAE96] KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. *Reinforcement learning: A survey*. Journal of Artificial Intelligence Research, 1996, v. 4, p. 237-285.

[KIT97a] KITANO, H.; ASADA, M.; CORADESCHI, S.; KUNIYOSHI, Y.; NODA, I.; OSAWA, E. *RoboCup: The Robot World Cup Initiative*, Proceedings of the 1st International Conference on Autonomous Agent (Agents' 97), Marina del Ray, California. The ACM Press, 1997, p. 5-8.

[KIT97b] KITANO, H.; TAMBE, M.; STONE, P.; VELOSO, M.; NODA, I.; OSAWA, E.; ASADA, M. *The RoboCup Synthetic Agent Challenge*. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), San Francisco, CA, 1997, p. 24-29.

[KON04] KONONEN, V. *Hybrid Model for Multiagent Reinforcement Learning*. Proceedings of the International Joint Conference on Neural Networks 2004 (IJCNN-2004), p. 1793–1798, Budapest, Hungary.

[LEV02] LEVNER, I.; BULITKO, V.; MADANI, O.; GREINER, R. *Performance of Lookahead Control Policies in the Face of Abstractions and Approximations*. Publisher: Springer-Verlag, v. 2371/2002, p. 299-307.

[LIT94] 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.

[LIT96] LITTMAN, M. L.; KAELBLING, L.P. *Reinforcement Learning: A Survey*, Journal of intelligence Research 4, 1996, p. 237-285.

[MAE94] MAES, P. *Modeling Adaptive Autonomous Agents*, Artificial Life Journal, edited by C. Langton, 1994, v. 1, n. 1, 2, p. 135-162, MIT Press.

[MAE95] MAES, P. *Artificial Life Meets Entertainment: Lifelike Autonomous Agents*, Communications of ACM, 1995, v. 38, n. 11 (Nov), p. 108-114.

[MAR00] MARIETTO, M. B. *Definição Dinâmica de Estratégias Instrucionais em Sistemas de Tutoria Inteligente: Uma Abordagem Multi-agentes na WWW*, Tese de Doutorado, ITA, Brasil, 2000.

[MIT97] MITCHELL, T. *Machine learning*. New York: McGraw Hill, 1997.

[MOD01] MODI, P. J.; SHEN, W. *Collaborative Multiagent Learning for Classification Tasks*. In Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press. Montreal - Quebec, Canada, 2001, p. 37-38.

[MON04] MONTEIRO, S. T.; RIBEIRO, C. H. C. *Desempenho de algoritmos de aprendizagem por reforço sob condições de ambiguidade sensorial em robótica móvel*. Sba Controle & Automação, July/Sept. 2004, vol. 15, no. 3, p.320-338.

[MUE97] MUELLER, J. P.; WOOLDRIDGE, M.; JENNINGS, N. R. *Intelligent Agents III. Agent Theories, Architectures, and Languages*. ECAI'96 Workshop, Budapest, Hungary. Series: Lecture Notes in Computer Science, v. 1193, 1997, p. 401, Springer Verlag, Berlin.

[NIL98] NILSSON, N. J. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, 1998.

[OLI00] OLIVEIRA, R. B. T. *O processo de extração de conhecimento de base de dados apoiado por agentes de software*. Dissertação de Mestrado. São Carlos: ICMC/USP, 2000.

[OSO99] OSÓRIO, F. S.; VIEIRA, R. *Tutorial Sistemas Híbridos Inteligentes*. XIX Congresso da S.B.C. Enia'99 – Encontro nacional de inteligência artificial. Rio de Janeiro, 1999.

[PEN96] PENG, J.; WILLIAMS, R. J. *Incremental multi-step Q-Learning*. W. W. Cohen e H. Hirsh (eds.), Proceedings of the Eleventh International Conference on Machine Learning, 1996, p. 226-232. San Francisco: Morgan Kaufmann.

[POR05] PORTA, J. M.; CELAYA, E. *Reinforcement Learning for Agents with Many Sensors and Actuators Acting in Categorizable Environments*. Journal of Artificial Intelligence Research, v. 23, 2005, p. 79-122.

[RAM05] RAMON, J. *On the convergence of reinforcement learning using a decision tree learner*. Proceedings of ICML-2005 workshop on Rich Representation for Reinforcement Learning, Bonn, Germany (Driessens, K. and Fern, A., van Otterlo, M., eds.), 2005, p. 1-6.

[REI03] REIS, L. P. *Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico*. Tese de Doutoramento, Faculdade de Engenharia da Universidade do Porto, Junho de 2003.

[RIB99] RIBEIRO, C. H. C. *A Tutorial on reinforcement learning techniques.* Supervised Learning track tutorials of the 1999 International Joint Conference on Neuronal Networks. Washington: INNS Press.

[RIB02] RIBEIRO, C. H. C. *Reinforcement learning agents*. Artificial Intelligence Review, 2002, v. 17, p. 223-250.

[RUB81] RUBINSTEIN, R. Y. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., New York, USA, 1st edition, 1981.

[RUS95] RUSSELL, S.; NORVIG, P. *Artificial Intelligence, a Modern Approach*, New Jersey: Prentice Hall International, 1995.

[RUS04] RUSSELL, S.; NORVIG, P. *Inteligência Artificial*, Tradução da segunda edição. Rio de Janeiro: Elsevier, 2004.

[RYA02] RYAN, M. R. K. *Hierarchical Reinforcement Learning: A Hybrid Approach*. PhD Thesis, University of New South Wales, School of Computer Science and Engineering, 2002.

[SAN97] SANTOS, C. L. R.; SICHMAN, J. S. *Significado e Representação de Organizações em Sistemas Multi-Agentes: Uma Análise Preliminar*, In: Encontro Nacional de Inteligência Artificial (ENIA), Brasília, 1997. Anais. Porto Alegre: SBC, p. 263-272.

[SCH93] SCHWARTZ, A. *A reinforcement Learning Method for Maximizing Undiscounted Rewards*. In Proceedings of the Tenth International Conference on Machine Learning, 1993. p. 298-305, Amherst, Massachusetts. Morgan Kaufmann.

[SIE89] SIEDLECKI, W.; SKLANSKY, J. *A note on Genetic Algorithms for Large-Scale Selection, Pattern Recognition Letters*, 1989, vol. 10, p. 335-347.

[SIN96] SINGH, S. P.; SUTTON, R. S. *Reinforcement learning with replacing eligibility traces*. Machine Learning, 1996, no. 22, p.123-158.

[STO96] STONE, P.; VELOSO, M. *Towards Collaborative and Adversarial Learning: a Case Study in Robotic Soccer*. International Journal of Human-Computer Studies, v. 48,

issue 1. Evolution and learning in multiagent systems. Academic Press, Inc. Duluth, MN, USA, 1996, p. 83-104.

[SUT92] SUTTON, R. S. *Reinforcement learning architectures*, Proceedings ISKIT'92 International Symposium on Neural Information Processing, Fukuoka, Japan, 1992.

[SUT98] SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. A Bradford book, The MIT Press, London, England, 1998.

[TAD94] TADEPALLI, P.; OK, D. *A reinforcement learning method for optimizing undiscounted average reward*. Technical Report, 1994. Department of Computer Science, Oregon State University.

[TES95] Tesauro, G. *Temporal difference learning and TD-Gammon*, Communications of the ACM, 1995, vol. 38 (3), p. 58-68

[TSI94] TSITSIKLIS, J. N. *Asynchronous Stochastic Approximation and Q-Learning*. Machine Learning, 1994, vol. 16, no. 3: p.185-202.

[VEL99] VELOSO, M.; STONE, P.; BOWLING, M. *Anticipation as a key for collaboration in a team of agents: a case study in robotic soccer*. In Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II, vol. 3839, 1999, p.134-141, G.T. McKee and P.S. Schenker, editors. Bellingham, Wash.: SPIE Press.

[VEL02] VELOSO, M. *Robot soccer: A Multi-Robot challenge*. In Multi-Robot Systems: From Swarms to Intelligent Automata. Kluwer, 2002.

[WAT92] WATKINS, C. J. C. H.; DAYAN, P. *Q-Learning, Machine Learning*, vol. 8(3), 1992, p.279-292.

[WEI96] WEISS, G.; SEN, S. *Adaptation and Learning in Multiagent Systems*, Lecture Notes in Artificial Intelligence. Berlin, Germany: Springer-Verlag, 1996, vol. 1042, p. 1-21.

[WEL03] WELLMAN, M. P.; CHENG, S.; REEVES, D. M.; LOCHNER, K. M. *Trading Agents Competing: Performance, Progress, and Market Effectiveness.* IEEE Intelligent Systems, 2003, vol. 18, no. 6, p. 48-53.

[WOO95] WOOLDRIDGE, M. J.; JENNINGS, N. R. *Intelligent agents: Theory and practice*. Knowledge Engineering Review 10(2), 1995, p. 115-152.

[WOO96] WOOLDRIDGE, M.; JENNINGS, N. R. *Software Agents*, IEE Review January, 1996, p. 17-20.

[WOO99] WOOLDRIDGE, M. *Intelligent agents*. InWeiss, G., ed., *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

Apêndice A

A1. Algoritmo A^* (A *star*)

Segundo Russell e Norvig [RUS04], a forma mais amplamente conhecida da busca pela melhor escolha de um caminho é chamada de busca A^* ou algoritmo do A^* (*A star*). Ela avalia os estados combinando $g(n)$, o custo para alcançar cada nó, e $h(n)$, o custo para ir do estado atual até o estado final.

A função expressa na Equação A.1 estima os valores do algoritmo [HAR68].

$$f(n) = g(n) + h(n) \tag{A.1}$$

Tendo em vista que $g(n)$ fornece o custo do caminho desde o estado inicial até o estado n, e que $h(n)$ é o custo estimado do caminho de custo mais baixo de n até o estado final, então, $f(n)$ é o custo estimado da solução de custo mais baixo passando por n. Desse modo, se for necessário encontrar a solução de custo mais baixo, uma opção razoável será experimentar primeiro o estado com o menor valor de $g(n)+h(n)$. Na verdade, essa estratégia é mais que apenas razoável: desde que a função heurística $h(n)$ satisfaça a certas condições, a busca A^* será ao mesmo tempo completa e ótima.

Nilsson [NILS98] prova que, se $h(n)$ é uma sub-estimativa de valor, então o algoritmo A^* garante encontrar o caminho mínimo entre a origem e o destino. A complexidade algorítmica do A^* é $O(n^2)$, sendo a mesma do algoritmo de Dijkstra [COR90].