4
ESTUDO DE CASOS

O sistema RL-NFHP – *Reinforcement Learning* Neuro-Fuzzy Hierárquico Politree –, apresentado no capítulo 2, teve seu código computacional inteiramente reescrito para inserção de novos métodos mostrados no capítulo 3. Este novo modelo RL-NFHP modificado apresentado neste trabalho foi avaliado em três aplicações. Estas aplicações estão detalhadas em três seções: a primeira contempla uma aplicação *benchmark* simulada; a segunda uma aplicação simulada em robótica; e a última uma aplicação em um robô real.

A primeira aplicação, realizada no *benchmark* Carro na Montanha, visa variar diversas configurações de parâmetros do modelo RL-NFHP modificado a fim de observar o comportamento deste no simulador. Por fim, o modelo com o melhor desempenho obtido será comparado com modelos desenvolvidos por outros pesquisadores.

A segunda aplicação, mais complexa que a primeira, visa testar o modelo RL-NFHP modificado em um robô simulado baseado no robô Khepera para que este seja capaz de aprender a navegar em ambiente desconhecido, evitando obstáculos.

Por fim, a terceira aplicação é a adaptação e implantação do melhor modelo RL-NFHP modificado obtido em ambiente simulado (segunda aplicação) num robô real Lego *MindStorms* NXT, dotando-o de comportamento inteligente, capaz de navegar desviando de obstáculos para chegar a uma posição final estipulada.

4.1
Carro na montanha (*Mountain-Car Problem*)

O primeiro grupo de experimentos é realizado no problema do carro na montanha (Moore, 1991). Este *benchmark* é altamente relevante, uma vez que vem sendo usado por diferentes pesquisadores (Moore, 1991; Boyan & Moore, 1995; Gordon, 1995; Singh & Sutton, 1996; Jouffe, 1998; Figueiredo, 2003; Figueiredo, 2004; Figueiredo, 2005; Jong & Stone, 2006; Främling, 2008) com o objetivo de testar seus modelos de aproximação de funções.

O problema consiste em acelerar um carro ao topo de uma montanha (figura 4.1), entretanto, o carro não possui potência necessária para vencer a força da gravidade, quando parte do seu vale. A única forma possível de alcançar o objetivo é ganhar energia potencial movendo-se no sentido contrário ao alvo para conjugar aceleração da gravidade à sua própria aceleração.

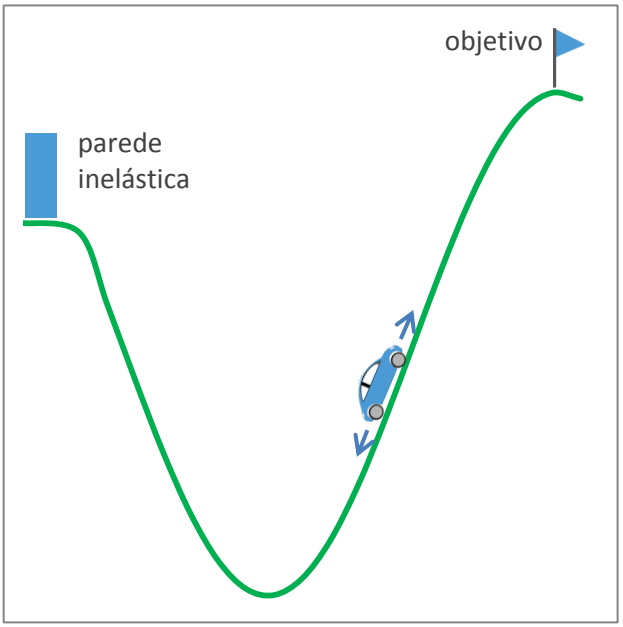

Figura 4.1: *Benchmark* Carro na Montanha.

Foi utilizado um simulador escrito em Java (seção 3.5) para realização dos experimentos. Desenvolveu-se uma interface gráfica simples, que poderia ser ativada ou desativada (menor custo computacional), para exibir o deslocamento do carro (círculo) no ambiente (figura 4.2). Demais informações relevantes (tais como: posição e velocidade inicinais e finais, episódio, número de passos, número máximo de células ativas e tamanho da estrutura) foram expostas na tela em forma de log ou gravadas em arquivos.

O problema possui duas variáveis de estados contínuas: a posição do carro $x \in [-1{,}2;0{,}5]$ e sua velocidade $v \in [-0{,}07;0{,}07]$, e três ações discretas: uma impulsão para a esquerda (F = -1); nenhuma impulsão (F = 0); e impulsão para a direita (F = 1). A montanha é representada pela função sen(3x), onde $x \in [-1{,}2;0{,}5]$.

A dinâmica do sistema é descrita pela eq. 4.1:

$$v_{t+1} = \min(0{,}07; \max(-0{,}07; v_t + 0{,}001F_t - 0{,}0025\cos(3x_t)))$$
$$x_t = \min(-1{,}2; x_t + v_{t+1}) \quad (4.1)$$

onde, o subscrito *t* indica o instante de tempo.

Quando x_t = -1,2 a velocidade é definida como v_t = 0 e o objetivo é atingido quando x_t = 0,5.

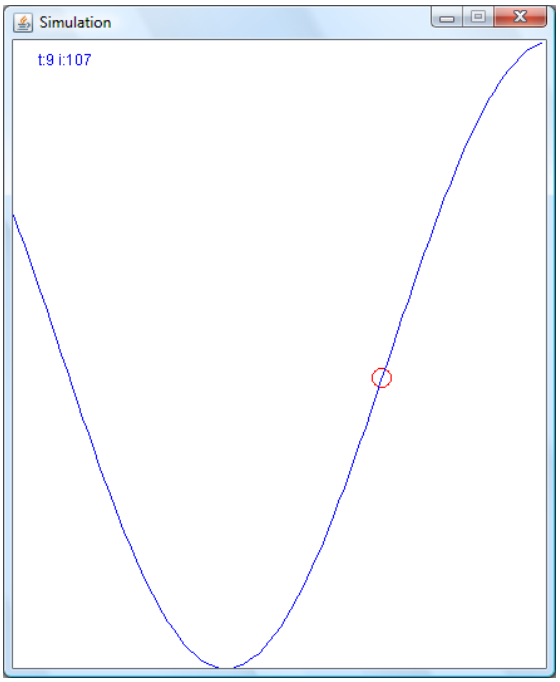

Figura 4.2: Tela de simulação do Carro na Montanha

No modelo RL-NFHP, cada célula criada possui 2 entradas normalizadas: a distância entre o carro e o objetivo $d_t^N \in [0; 1]$ e a velocidade do carro $v_t^N \in [-1; 1]$ dadas pela eq. 4.2:

$$d_t^N = \frac{0{,}5 - x_t}{1{,}7} \quad e \quad v_t^N = \frac{v_t}{0{,}07} \quad (4.2)$$

Em todos os experimentos realizados neste estudo de caso os valores de reforço foram calculados como funções de avaliação das variáveis do ambiente. Estas avaliações levaram em conta a distância normalizada entre o carro e o objetivo e a sua velocidade normalizada, segundo a eq. 4.3 proposta por Figueiredo (2003):

$$R_t = \begin{cases} k_1 e^{-d_t^N} + k_2 e^{-|v_t^N|} & se\ x_{t-1} \leq x_t \\ k_3 e^{-(1-d_t^N)} + k_4 e^{-|v_t^N|} & se\ x_{t-1} > x_t \end{cases} \quad (4.3)$$

onde: $k_1, k_2, k_3\ e\ k_4\ \in\ \mathbb{R}^*$.

O valor da função de avaliação relativa à distância cresce à medida que o carrinho se aproxima do objetivo ou quando ele se afasta do objetivo. Em ambas as formas a função de avaliação relativa à velocidade cresce quando o valor do módulo da velocidade também cresce.

Os experimentos descritos nas subseções seguintes visam alterar apenas um parâmetro a fim de verificar sua influência no tempo de aprendizado e no modelo RL-NFHP modificado final. Foram variados: a política de escolha das ações, a probabilidade de escolha aleatória, a função de crescimento, o *eligibility trace* cumulativo e o alfa de atualização de *Q(s,a)*. Por fim o melhor modelo RL-NFHP modificado foi comparado com modelos de outros pesquisadores.

4.1.1
Experimento 1: Política de escolha das ações

Inicialmente foram realizados testes no modelo RL-NFHP modificado com variações apenas das políticas de escolha das ações (seção 3.1):

- *Q-roulette* com probabilidade p = 1-ε de escolha da ação através de uma roleta baseada nos valores de Q(*s*,*a*) e com probabilidade p = ε de seleção aleatória.
- Política *DC-roulette* com probabilidade p = ε de escolha da ação baseada no número de visitas, onde as menos visitadas tem mais chances, e com probabilidade p = 1-ε de escolha da ação associada ao maior *Q*(*s*,*a*), ou seja, seleção gulosa.
- Política *Q+DC-roulette* com probabilidade p = 1-ε de escolha da ação por meio da roleta *Q-roulette* e com probabilidade p = ε pela *DC-roulette*.
- Política nova *Q-DC-roulette* com p = ε e p = 1-ε de seleção gulosa.
- Política *ε-greedy* com p = 1-ε de seleção gulosa e p = ε de escolha aleatória.

As demais variáveis foram mantidas fixas como está detalhado a seguir. O valor definido para o parâmetro γ da equação de atualização dos valores $Q(s,a)$ (eq. 2.17) foi de 1,0 e o $Q(s',a')$ de atualização foi a média ponderada das funções de valor das células participantes. O parâmetro ε associado a probabilidade de escolha aleatória de ação foi iniciado com 10% no momento de criação da célula. A taxa de incremento/decremento deste parâmetro quando a ação era respectivamente de punição ou prêmio, foi de 5%. Este valor foi definido heuristicamente por Figueiredo (2003), porém seguindo as recomendações feitas por Sutton (1996). Estes são valores típicos utilizados por outros pesquisadores (Sutton & Barto, 1998). O parâmetro *n* da função de crescimento utilizada no particionamento da célula (eq. 2.19), que também depende do número de ciclos e do número de épocas, teve seu valor fixado em 5 e os valores de incremento $\Delta\beta_1$ e decremento $\Delta\beta_2$ da variável de crescimento foram fixados em 1. Embora tenha sido implementado no código, não foi usado o parâmetro *alfa-cut* para os conjuntos fuzzy nos experimentos, pois não gerou mudanças significativas nos resultados em testes preliminares.

As condições iniciais foram alternadas entre (x_0, v_0) = (1,2;0) e (0,5;0), junto à parede e no vale com velocidade nula, respectivamente.

Foi utilizado o *early stopping* (seção 3.2) para definição do término do aprendizado. Neste experimento, este procedimento foi executado a cada 100 episódios, com validação através 200 ou menos passos para que o carro alcançasse o objetivo. Em todos os experimentos deste estudo de caso, foram realizados cerca de 1000 treinamentos com número máximo de 5000 épocas. Em cada época foi estabelecido um número máximo de passos para alcance do objetivo de 10000, ou seja, na prática um número ilimitado de passos. Treinamentos que não resultassem em aprendizado dentro de critérios adotados foram considerados sem sucesso e foram descartados. Os dados dos treinamentos que geraram aprendizado foram filtrados uma única vez, descartando aqueles que possuíam valores maiores que a média mais dois desvios padrões ou menores que a média menos dois desvios padrões.

O gráfico 4.1 mostra o número de treinamentos aproveitados em relação aos realizados. O treinamento é considerado aproveitado quando o modelo RL-NFHP

modificado termina atendendo ao critério de parada *early stopping* e seus testes são bem sucedidos.

Observa-se que a utilização das políticas *DC-roulette*, *Q-DC-roulette* e *ε-greedy* apresentaram os maiores aproveitamentos da fase de treino fazendo com que o modelo RLNFHP aprendesse. As políticas *Q-roulette* e *Q+DC-roulette* mostraram resultados insatisfatórios, aproveitamento menor que 25%, resultando num aprendizado geral mais lento devido ao maior número de treinamentos desperdiçados.

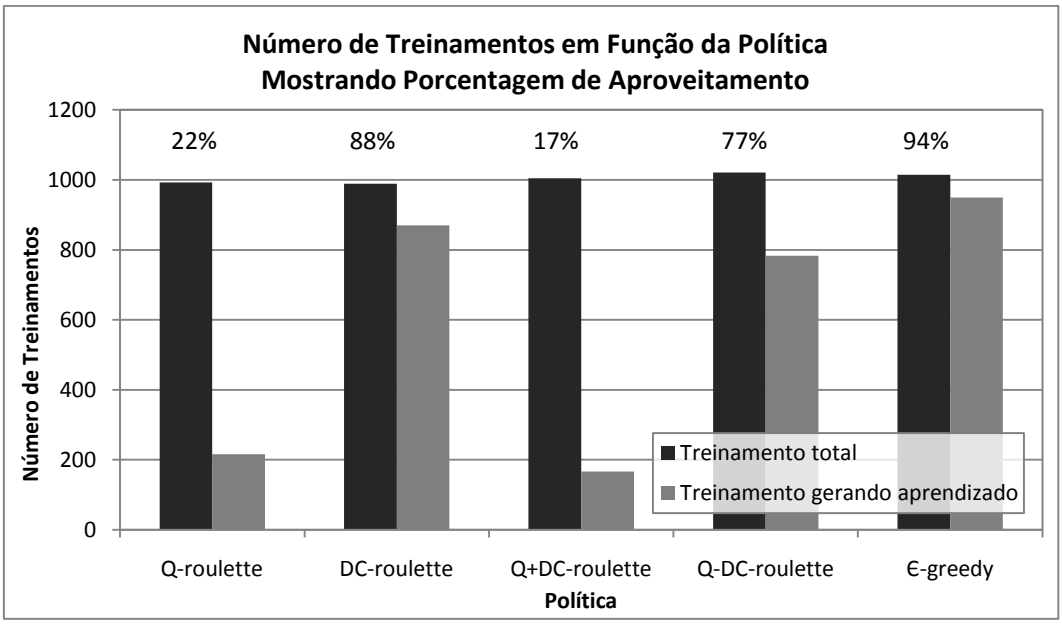

Gráfico 4.1: Aproveitamento do treinamento.

O gráfico 4.2 mostra o histograma da média do número de passos necessários para alcançar o objetivo utilizando a política *ε-greedy* após os treinamentos bem sucedidos.

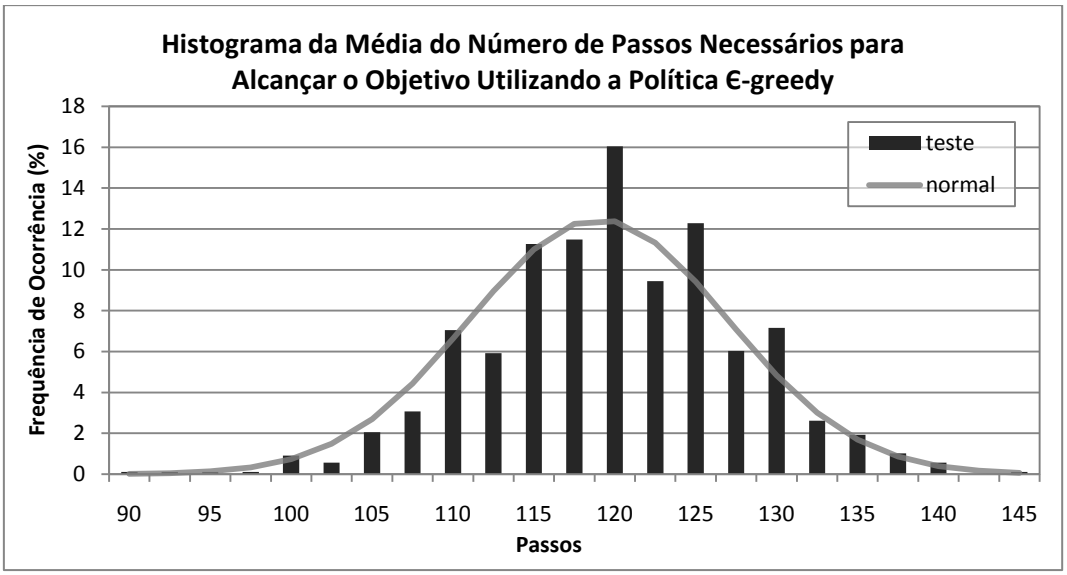

Gráfico 4.2: Histograma da média de passos.

Este histograma é semelhante em todas as políticas. Ele ilustra como a distribuição do número de passos médio para alcançar o objetivo partindo das diversas condições iniciais se assemelha a distribuição normal com média de ciclos de 119 com desvio padrão 8. Este comportamento é esperado uma vez que se está utilizando o critério de validação *early stopping* através de 200 ou menos passos. O valor do número máximo de passos para permitir a interrupção do treinamento, neste caso 200, está relacionado ao pior caso inicial, quando o carro está no vale, logo, na média, sempre se obtém números de passos menores quando o modelo aprendeu.

Os gráficos 4.3 a 4.7 ilustram a frequência de ocorrência do número de épocas necessárias para que ocorra aprendizado utilizando as políticas de escolha das ações *Q-roulette*, *DC-roulette*, *Q+DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

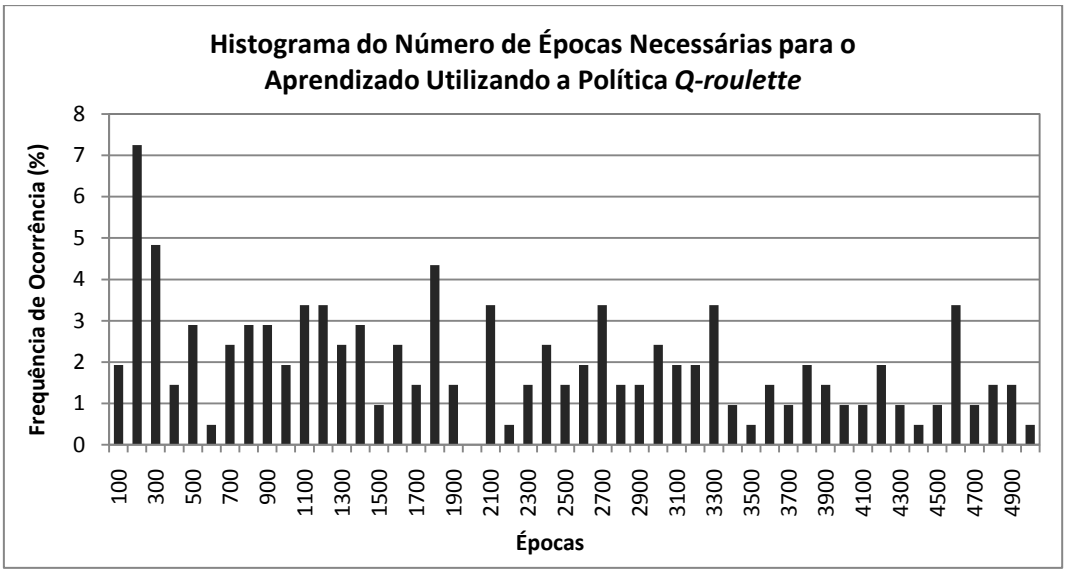

Gráfico 4.3: Histograma das épocas de treinamento usando *Q-roulette.*

Observa-se no gráfico 4.3 que a maior frequência de ocorrência de aprendizado ocorre com 200 épocas, ou seja, com 200 épocas de treinamento a estrutura RLNFHP possui uma resposta igual ou acima da desejada na validação (no caso 200 passos para alcançar o objetivo). Pode-se reparar também uma grande dispersão do número de épocas para treinamento semelhante a um ruído, devido à natureza aleatória da política usada.

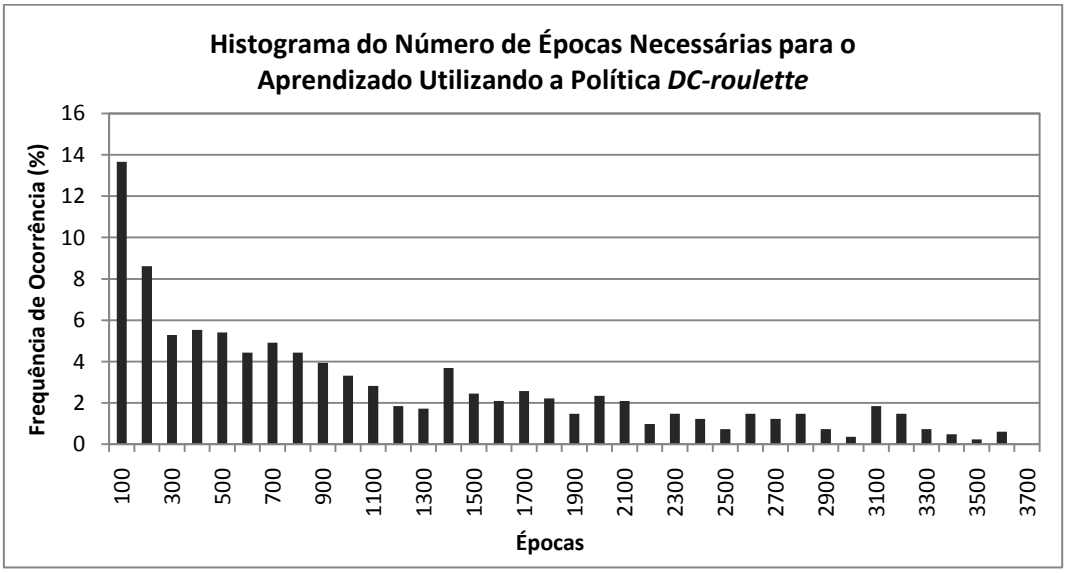

Gráfico 4.4: Histograma das épocas de treinamento usando *DC-roulette.*

O gráfico 4.4 mostra que o aprendizado ocorre na maioria das vezes em 100 épocas de treinamento e existe um decaimento da frequência de ocorrência de

aprendizado a medida que se aumentam o número de épocas, ou seja, geralmente ocorre aprendizado no começo do treinamento.

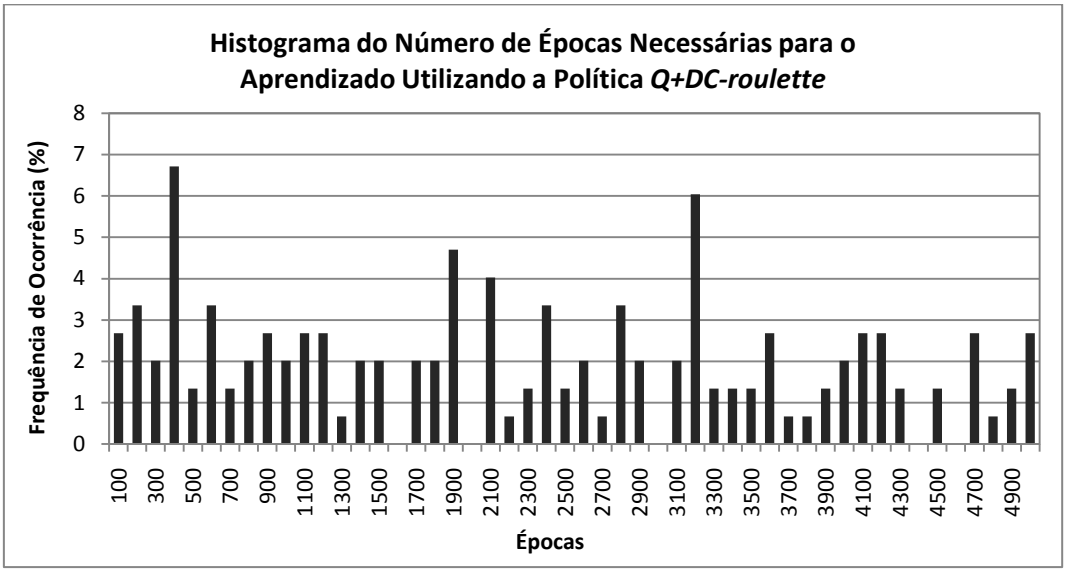

Gráfico 4.5: Histograma das épocas de treinamento usando *Q+DC-roulette.*

Observa-se no gráfico 4.5 que a maior frequência de ocorrência de aprendizado ocorre com 400 épocas, porém com grande dispersão semelhante ao que ocorre quando se usa a *Q-roulette*. Esta dispersão se deve ao uso associado com a política *Q-roulette*.

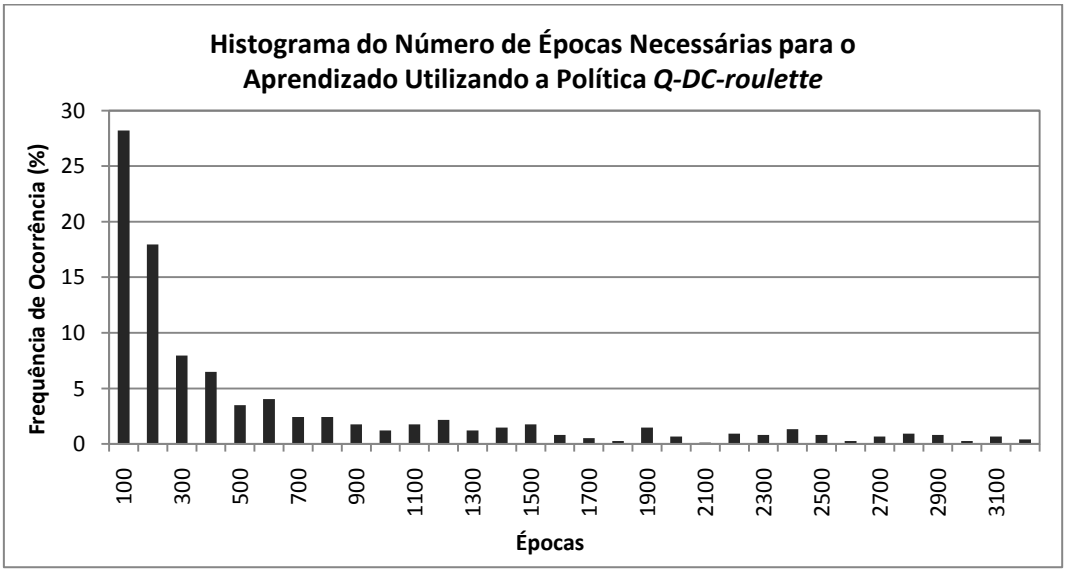

Gráfico 4.6: Histograma das épocas de treinamento usando *Q-DC-roulette.*

Constata-se pelo gráfico 4.6 que o aprendizado usando a política *Q-DC-roulette* ocorre na maioria das vezes no começo do treinamento, em 100

épocas, tão rápido quanto o uso da política *DC-roulette*, porém com menor dispersão do número de épocas para treinamento.

O desempenho no aprendizado ao utilizar-se a política *ε-greedy* é bastante similar ao uso do *Q-DC-roulette*, como pode ser visto no gráfico 4.7, no entanto a dispersão do número de épocas para treinamento é maior do que o último e se estende até 3700 épocas.

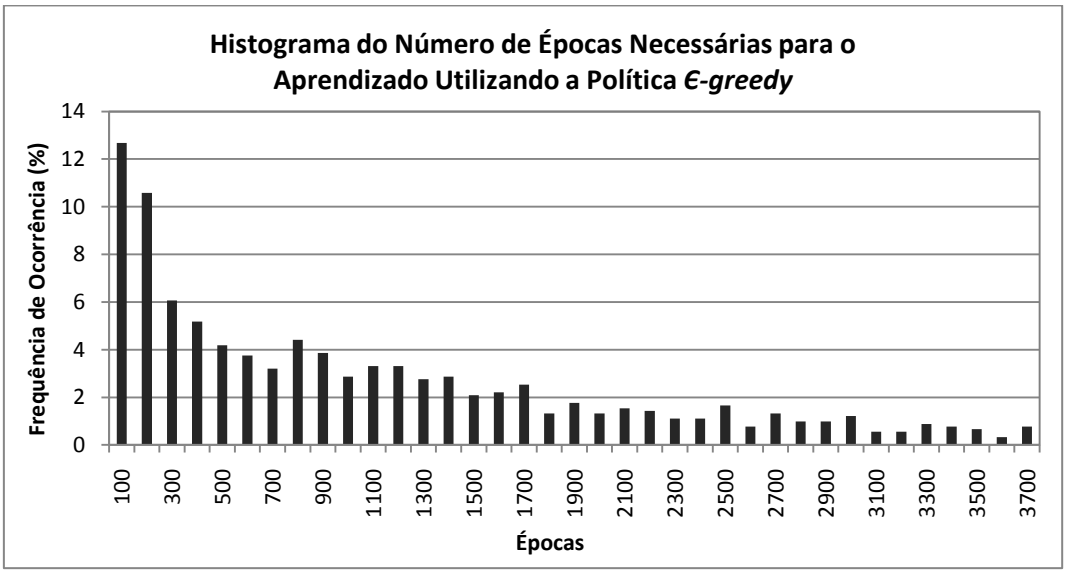

Gráfico 4.7: Histograma das épocas de treinamento usando *ε-greedy.*

As políticas que apresentaram melhor desempenho no número de épocas necessárias para treinamento foram: *DC-roulette*, *Q-DC-roulette* e *ε-greedy*. Todas obtiveram maior frequência de aprendizado em 100 épocas. Este número poderia ser menor, caso a validação fosse realizada em intervalos menores de épocas como será realizado nos próximos experimentos. Isto faria com que os gráficos 4.3 a 4.7 tivessem maior resolução no eixo das abscissas.

Os gráficos 4.8 a 4.12 permitem uma análise da frequência de ocorrência do número de células do modelo RL-NFHP modificado após treinamento bem sucedido para a utilização das políticas de escolha das ações *Q-roulette*, *DC-roulette*, *Q+DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

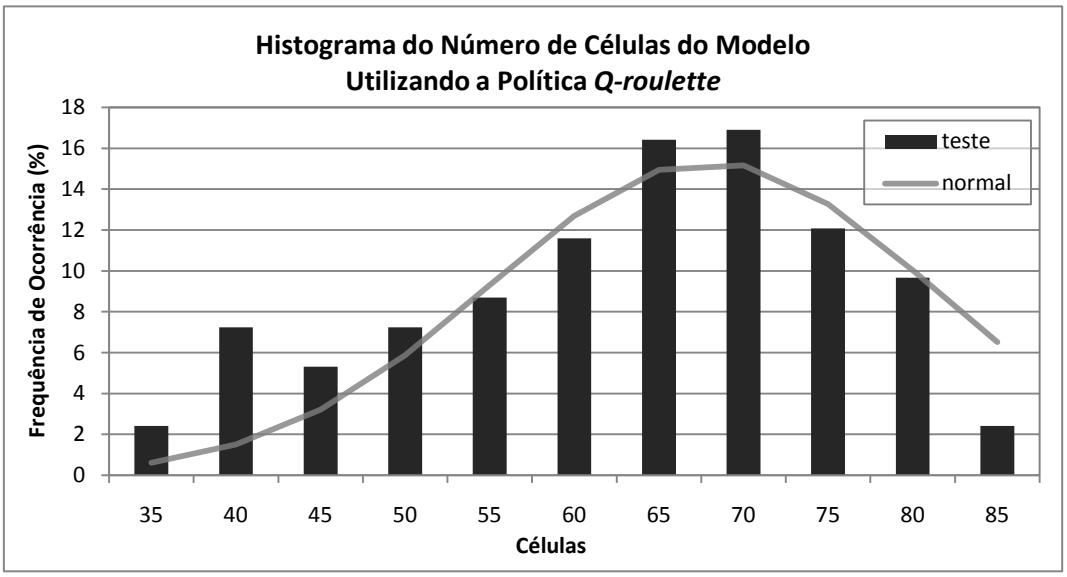

Gráfico 4.8: Histograma do número de células do modelo usando *Q-roulette.*

Para a política *Q-roulette*, gráfico 4.8, a distribuição se assemelha a uma distribuição normal com média de 68 células do modelo e desvio de 13, sendo 84 o número máximo de células alcançado após treinar o modelo.

Observa-se que a distribuição da frequência de ocorrência do número de células após o aprendizado, gráfico 4.9, assemelha-se a uma distribuição normal com média de 64 células do modelo e dispersão grande, com desvio padrão de 24 células, sendo 120 o número máximo de células alcançado para treinar o modelo.

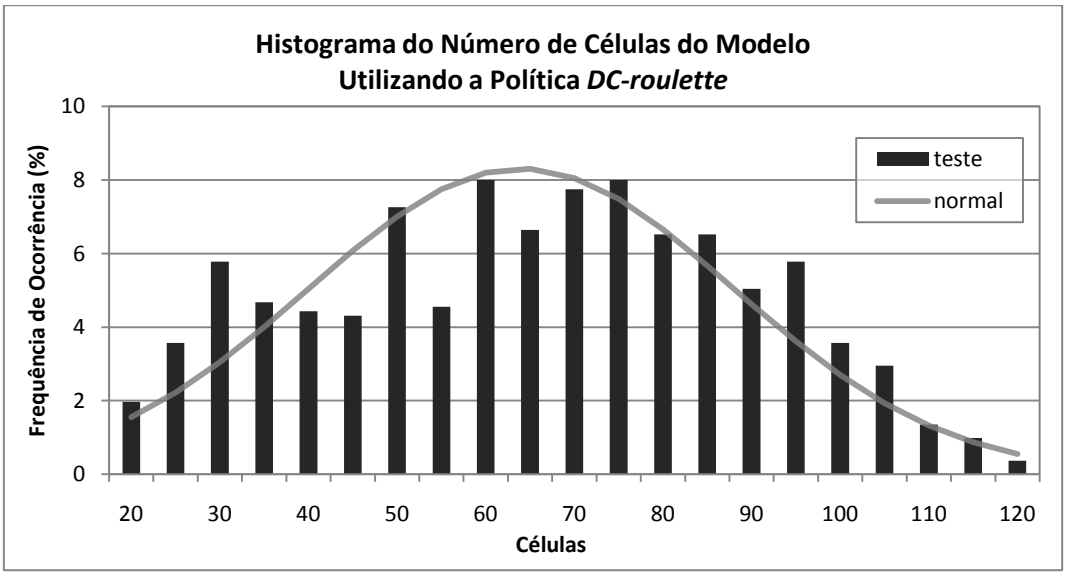

Gráfico 4.9: Histograma do número de células do modelo usando *DC-roulette.*

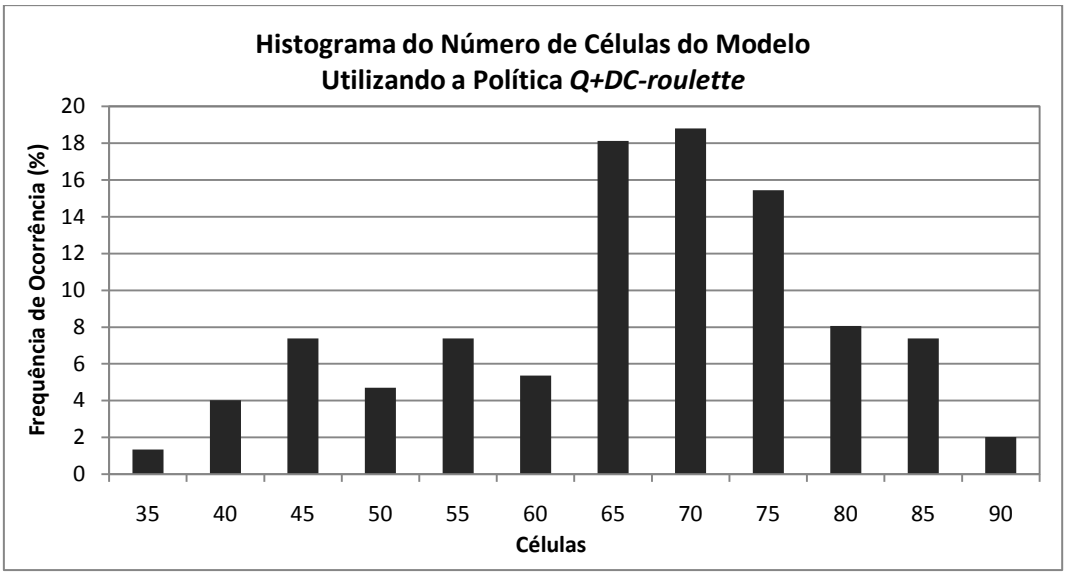

Gráfico 4.10: Histograma do número de células do modelo usando Q+*DC-roulette.*

No gráfico 4.10, verifica-se que a maior frequência de ocorrência do número de células do modelo é 70 com desvio padrão de 13. A distribuição assemelha-se um pouco com a obtida por meio da política *Q-roulette*, mostrando a forte influência desta roleta na composição *Q+DC-roulette*.

Pelo histograma do gráfico 4.11, pode-se dizer que, na utilização da política *Q-DC-roulette*, a maior frequência de ocorrência de células do modelo em um treinamento bem sucedido é a menor dentre todas as políticas, como será visto, de apenas 25 células, tendo uma cauda bastante longa estendendo-se até 160 células.

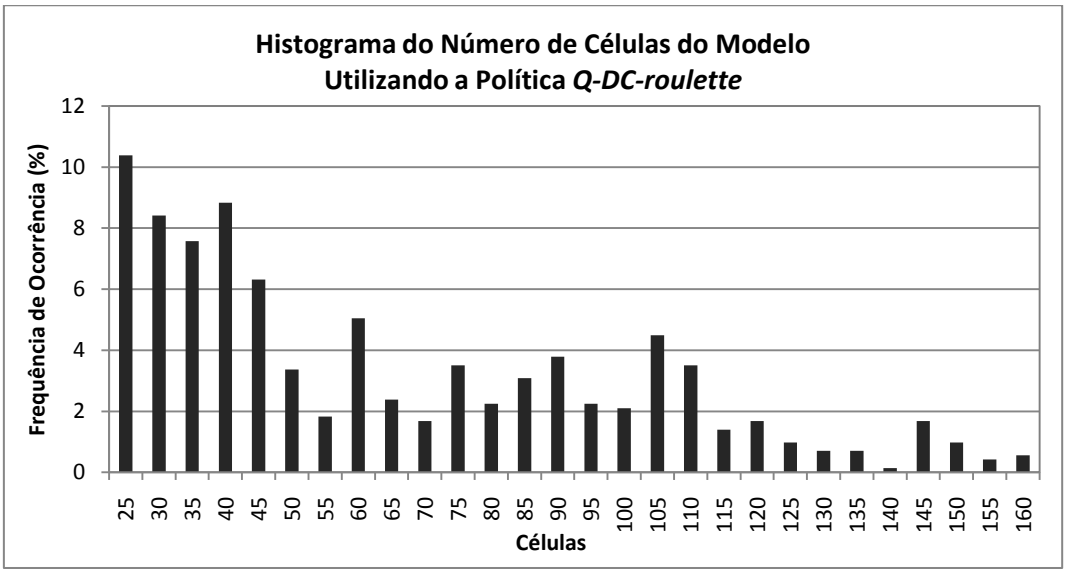

Gráfico 4.11: Histograma do número de células do modelo usando Q-*DC-roulette.*

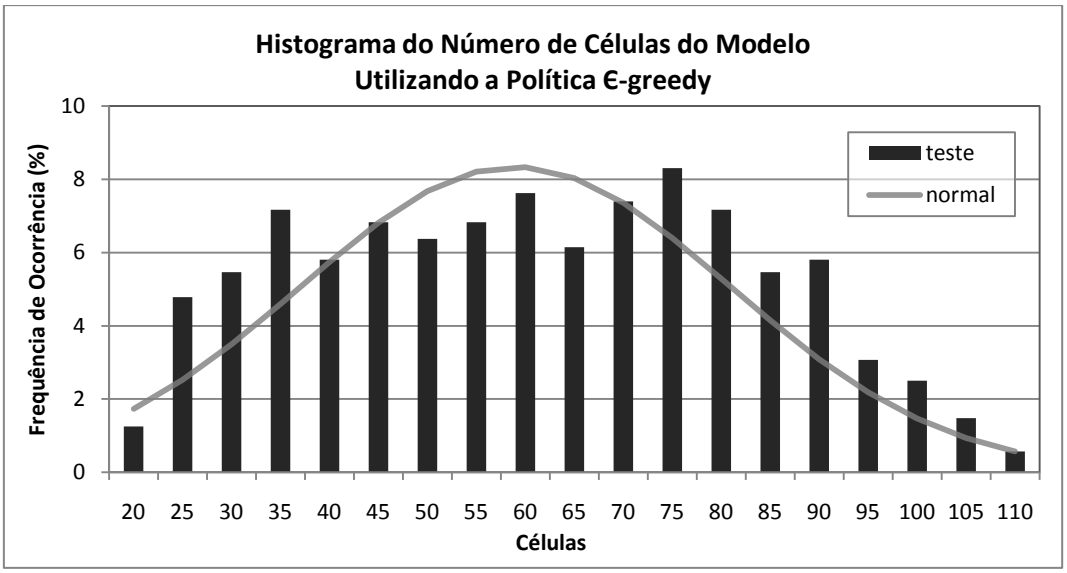

Gráfico 4.12: Histograma do número de células do modelo usando *ε-greedy.*

Para a política *ε-greedy*, gráfico 4.12, a distribuição se assemelha a uma distribuição normal com média de 59 células do modelo e dispersão grande, com desvio padrão de 22 células, sendo 110 o número máximo de células alcançado para treinar o modelo.

O histograma do número máximo de células ativas por episódio para a política *Q-DC-roulette*, gráfico 4.13, é similar ao das demais políticas testadas. Ele resume o ganho computacional alcançado devido à nova abordagem na programação. Em todos os ciclos desde as condições iniciais até o objetivo de todos os testes realizados nos modelos treinados, no máximo cinco células do modelo RL-NFHP modificado participaram ao mesmo tempo. Sendo, na maioria dos casos, apenas três células são responsáveis pela resposta do modelo.

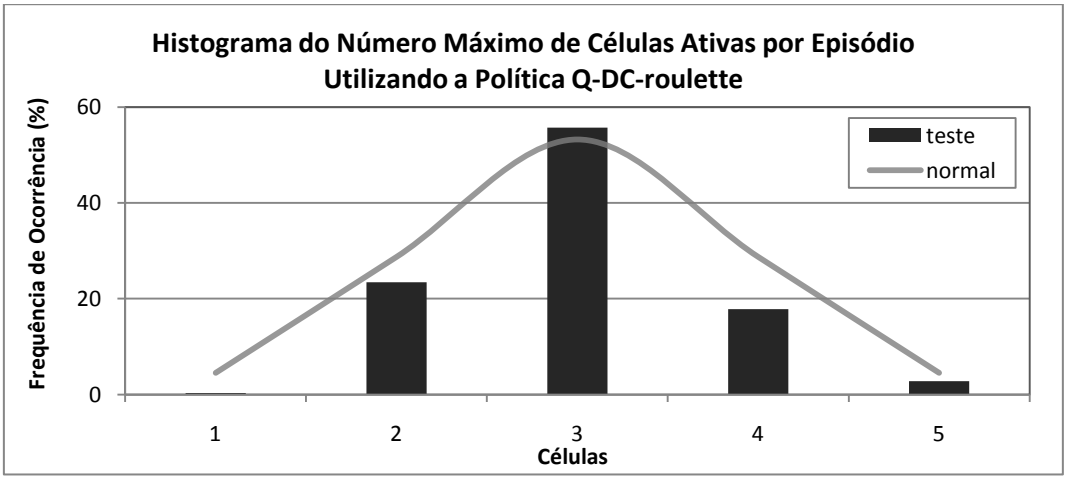

Gráfico 4.13: Histograma do número máximo de células ativas por episódio usando *Q-DC-roulette.*

A tabela 4.1 compara o desempenho das diversas políticas de escolha das ações no modelo RL-NFHP modificado em relação ao número de épocas no treinamento necessárias para aprendizado, percentagem de treinamentos resultantes em aprendizado e número de células da estrutura para o *benchmark* Carro na Montanha.

Tabela 4.1: Comparação entre as políticas no modelo RL-NFHP.

Política	Nº de Épocas para Treinamento	Aproveitamento dos Treinamentos (%)	Nº de Células
Q-roulette	200	22	68
DC-roulette	**100**	**88**	**64**
Q+DC-roulette	400	17	68
Q-DC-roulette	**100**	**77**	**25**
ε-greedy	**100**	**94**	**59**

Observa-se que as políticas que geram aprendizado mais rápido e com maior aproveitamento dos treinamentos são a *DC-roulette*, *Q-DC-roulette* e a *ε-greedy*. São elas também que possuem o menor tamanho de estrutura, destaque para a *Q-DC-roulette* que possui maior ocorrência de 25 células. O número de épocas necessárias para treinamento de 100 nos três casos se deve a falta de resolução. Os experimentos a seguir refinam a escala de treinamento.

4.1.2 Experimento 2: Variação da probabilidade de escolha aleatória

Nos testes do modelo RL-NFHP modificado conduzidos nesta subseção, foi alterada apenas a forma de variação do parâmetro ε associado à probabilidade de escolha aleatória. Este parâmetro foi iniciado com 10% no momento de criação da célula. A taxa de incremento/decremento de ε quando a ação era respectivamente de punição ou prêmio, foi de 5%/5% ou 40%/5%. Ou seja, quando uma ação é mal sucedida pretende-se aumentar bastante a possibilidade de escolha aleatória. A escolha desses valores visa observar influência de incrementos grandes e pequenos do ε no desempenho do modelo. Os demais

parâmetros foram mantidos fixos e as condições iniciais foram as mesmas mostradas no experimento 1 (subseção 4.1.1).

As políticas de escolha das ações utilizadas foram: *DC-Roulette*, *Q-DC- Roulette* e *ε-greedy*, pois obtiveram melhores desempenhos no número de épocas de aprendizado. Além disto, a política *ε-greedy* foi mantida, uma vez que é tradicionamente utilizada por divesos pesquisadores (Sutton, 1998; Figueiredo 2003).

Neste experimento, o *early stopping* foi testado a cada 10 episódios, ao invés de 100 como no experimento 1, aumentando a resolução dos histogramas de número de épocas de treinamento.

Os gráficos 4.14 a 4.16 mostram o histograma do número de épocas necessárias para treinamento usando a política *DC-roulette, Q-DC-roulette* e *ε-greedy* para a taxa de incremento da probabilidade de escolha aleatória de 5% e 40%. Estes gráficos foram truncados em 200 épocas para facilitar visualização. O número de épocas de aprendizado estende-se até 490 (com valores de frequência de ocorrência abaixo de 0,9%), 930 (com valores de frequência de ocorrência abaixo de 0,5%) e 290 (com valores de frequência de ocorrência abaixo de 0,9%), respectivamente aos gráficos das políticas *DC-roulette*, *Q-DC-roulette* e *ε-greedy*.

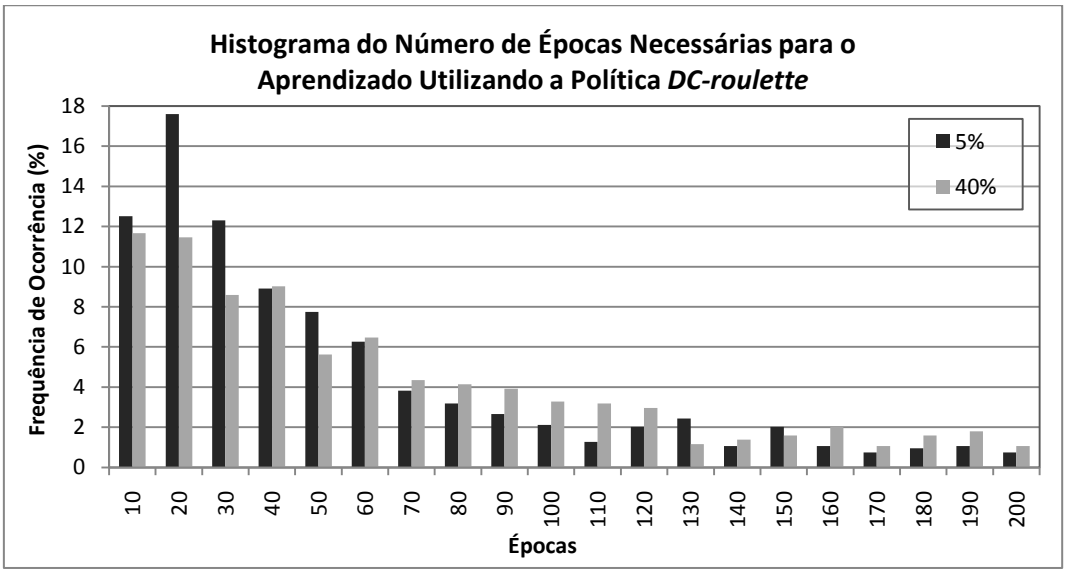

Gráfico 4.14: Histograma das épocas de treinamento usando *DC-roulette.*

Verifica-se, pelo gráfico 4.14, que o formato das distribuições para o incremento de escolha aleatória de 5% e 40% muda ligeiramente, decaindo mais rapidamente no caso de 5%. A mudança no incremento não altera o número de

épocas necessário para o aprendizado de maior número de ocorrência, 20 épocas nos dois casos.

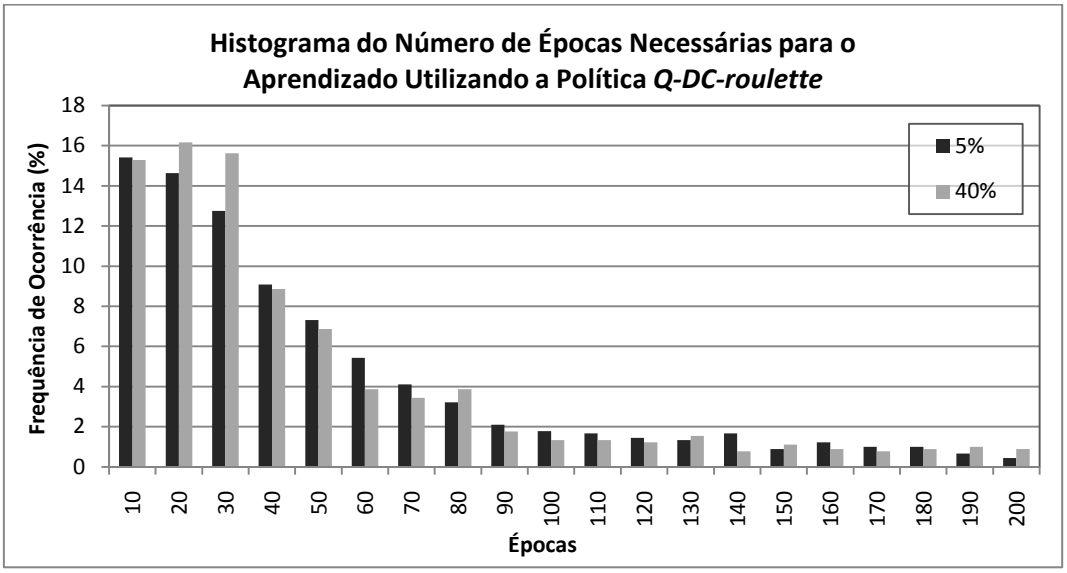

Gráfico 4.15: Histograma das épocas de treinamento usando *Q-DC-roulette.*

Observa-se, pelo gráfico 4.15, que o formato das distribuições, nos diferentes modelos, para a mudança do incremento de escolha aleatória se assemelham bastante. A diferença reside no fato do processo de aprendizado demorar ligeiramente mais para aprender. Um maior número de ocorrências se dá em 20 épocas, no caso de 40% do que as 10 épocas, em se usar 5%, apenas.

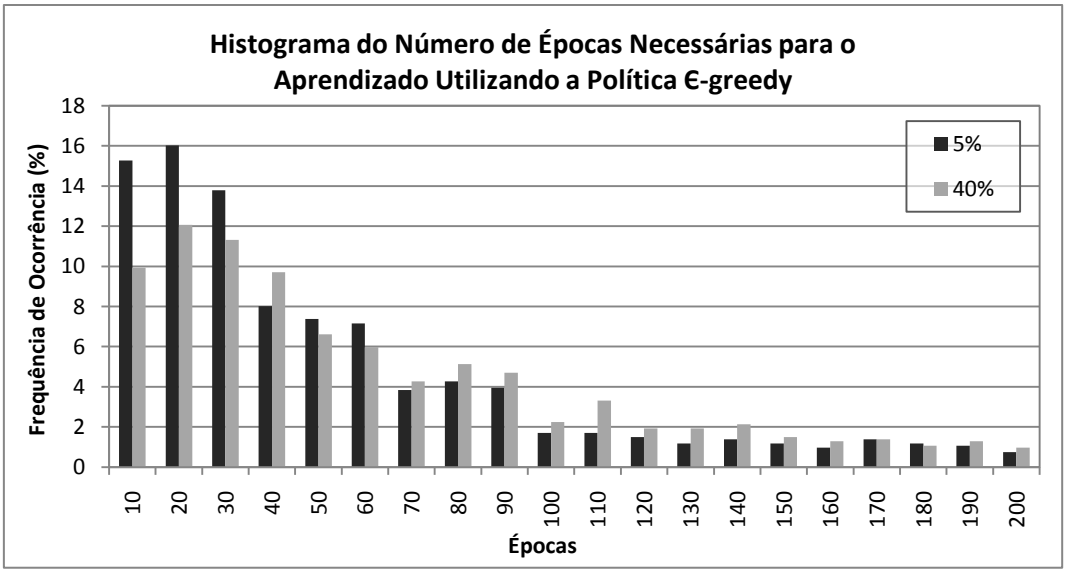

Gráfico 4.16: Histograma das épocas de treinamento usando *ε-greedy.*

O gráfico 4.16 mostra que o formato das distribuições para o incremento de escolha aleatória de 5% e 40% muda ligeiramente. No caso de 5% o aprendizado é mais frequente em poucas épocas do que no caso de 40%, maior número de ocorrência para 10, 20 e 30 épocas. A mudança no incremento não altera o número de épocas necessário para o aprendizado com maior número de ocorrências, 20 épocas nos dois casos. Por este gráfico, para um menor número de épocas, a diferenção entre os histogramas das taxas do incremento é maior do que quando o número de épocas cresce. Neste caso, a diferença dos dois incrementos testados parece menos significante.

O histograma do número de células do modelo após aprendizado usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy* para a taxa de incremento da probabilidade de escolha aleatória de 40% não varia significativamente em relação ao incremento de 5% mostrado no experimento 1 nos gráficos 4.9 a 4.12, respectivamente.

No geral, a maior variação da probabilidade de escolha aleatória durante o treinamento de 5% para 40% não ocasionou em mudança significativa no desempenho de treinamento como pode ser visto na tabela 4.2.

Tabela 4.2: Comparação entre as taxas de incremento da porcentagem aleatória de 5% e 40% para as diferentes políticas no modelo RL-NFHP.

Política	**Nº de Épocas para Treinamento**	
	5%	**40%**
DC-roulette	20	20
Q-DC-roulette	10	20
ε-greedy	20	20

4.1.3 Experimento 3: Função de crescimento

Nos testes do modelo RL-NFHP modificado conduzidos neste experimento, foi alterado apenas o parâmetro *n* da função de crescimento Φ (eq. 2.19) utilizada no particionamento da célula, que também depende do número de ciclos e do número de épocas. Este parâmetro assumiu os valores: 5, 20, 50 e 100.

Os demais parâmetros do modelo foram mantidos fixos e as condições iniciais foram as mesmas mostradas no experimento 1 (subseção 4.1.1). As políticas de escolha das ações utilizadas foram: *DC-Roulette*, *Q-DC- Roulette* e *ε-greedy*.

Os gráficos 4.17 a 4.19 mostram o histograma do número de épocas necessárias para treinamento usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy* para diferentes valores do parâmetro *n* da função de crescimento, respectivamente. A fim de permitir uma avaliação dos efeitos da mudança do valor de *n*, histogramas do número de épocas de treinamento foram plotados no mesmo diagrama. Para facilitar a visualização, os tradicionais histogramas em barras foram substituídos por linhas contínuas.

O gráfico 4.17 mostra que o menor número de épocas necessárias de treinamento ocorre para *n*=5, ou seja, 10 épocas. Para os demais valores de *n* o número de épocas é de 20. Porém quando *n*=20, o aprendizado ocorre mais rapidamente, maiores valores de frequência de ocorrência para baixas épocas (menores que 40).

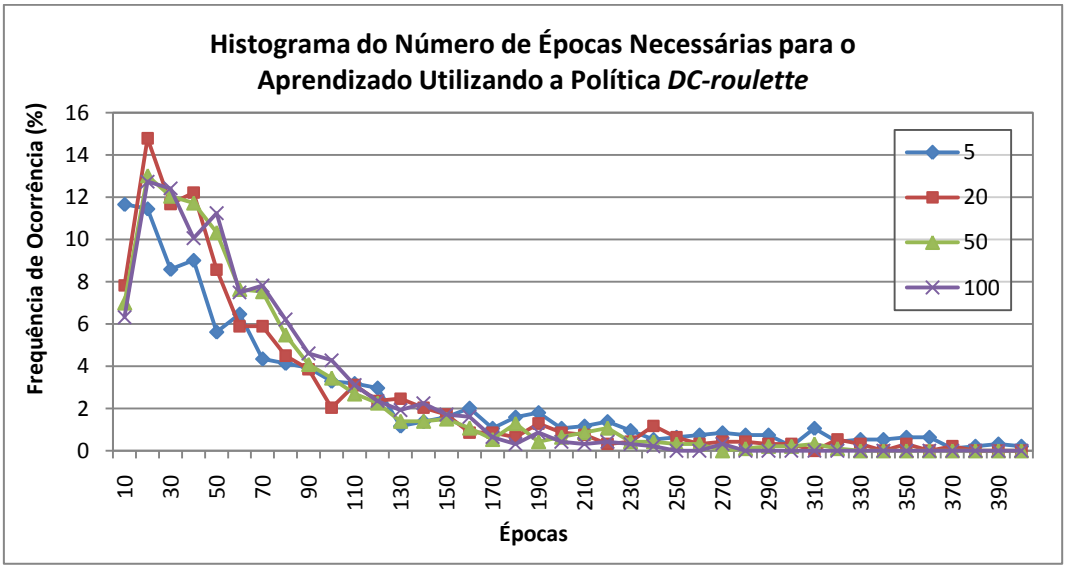

Gráfico 4.17: Histograma das épocas de treinamento usando *DC-roulette*.

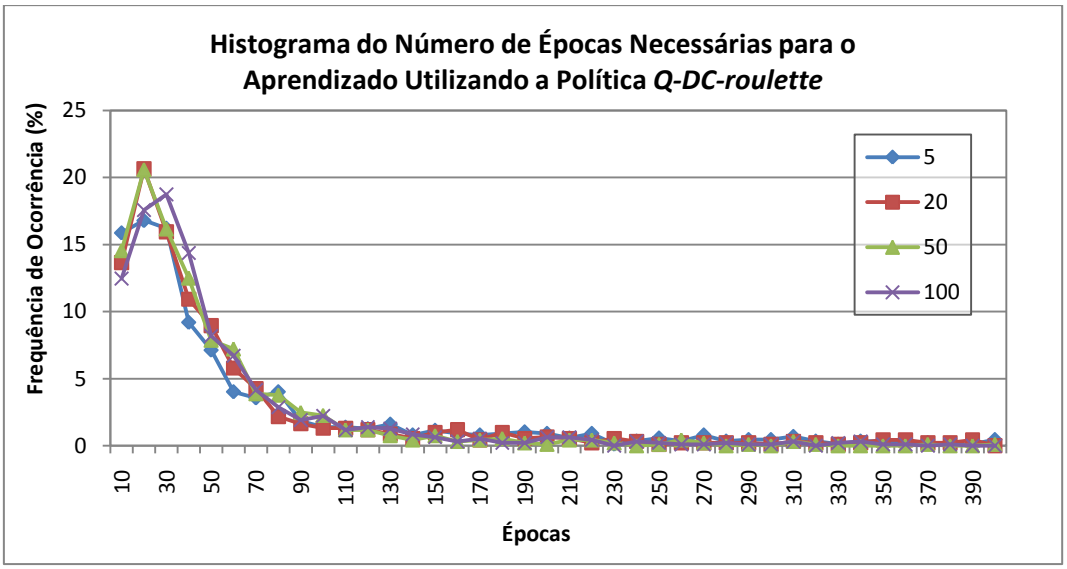

Gráfico 4.18: Histograma das épocas de treinamento usando *Q-DC-roulette*.

Observa-se pelo gráfico 4.18 que o número de épocas para treinamento com maior frequência de ocorrência é de 20, quando *n* é 5, 20 ou 50, e de 30 épocas, quando *n* é 100.

O gráfico 4.19 mostra que o menor número de épocas necessárias de treinamento, ou seja, 20 épocas, ocorre para *n*=5 e *n*=20. Para os demais valores de *n*, o número de épocas é de 30.

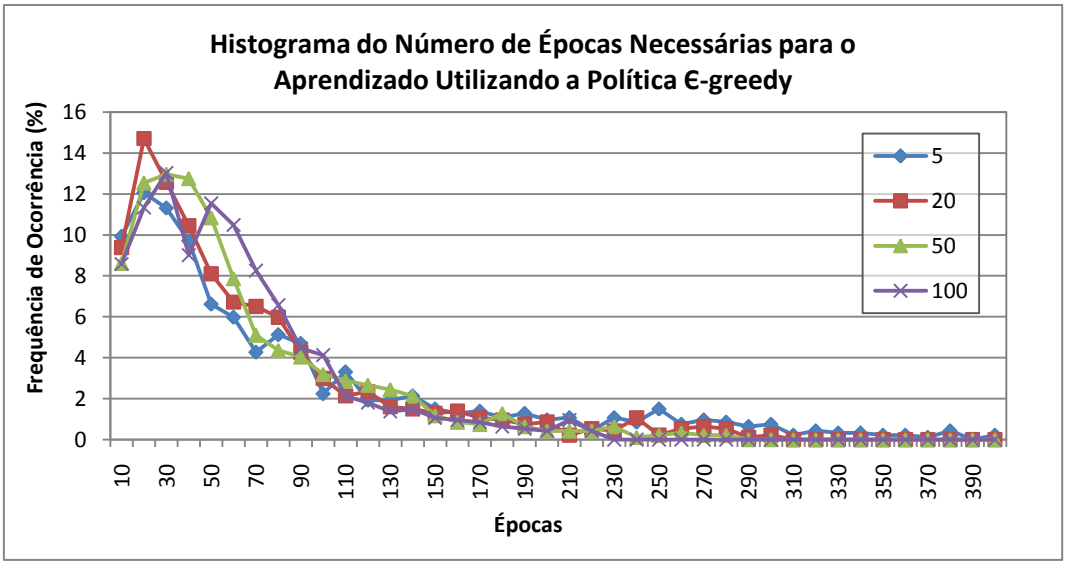

Gráfico 4.19: Histograma das épocas de treinamento usando *ε-greedy*.

Comparando o gráfico 4.18, com o gráfico 4.17 e 4.19, verifica-se que a distribuição das curvas para os diversos *n* para a política *Q-DC-roulette* é mais estreita, ou seja, possui menor desvio padrão, que as demais, *DC-roulette* e

ε-greedy. Além disto, a política *Q-DC-roulette*, possui os maiores valores de frequência de ocorrência para baixas épocas, indicando que, no geral, o modelo RL-NFHP modificado utilizando esta política aprende mais rapidamente.

Os gráficos 4.20 a 4.22 mostram o histograma do número de células do modelo após treinamento bem sucedido para diferentes valores do parâmetro *n* da função de crescimento usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

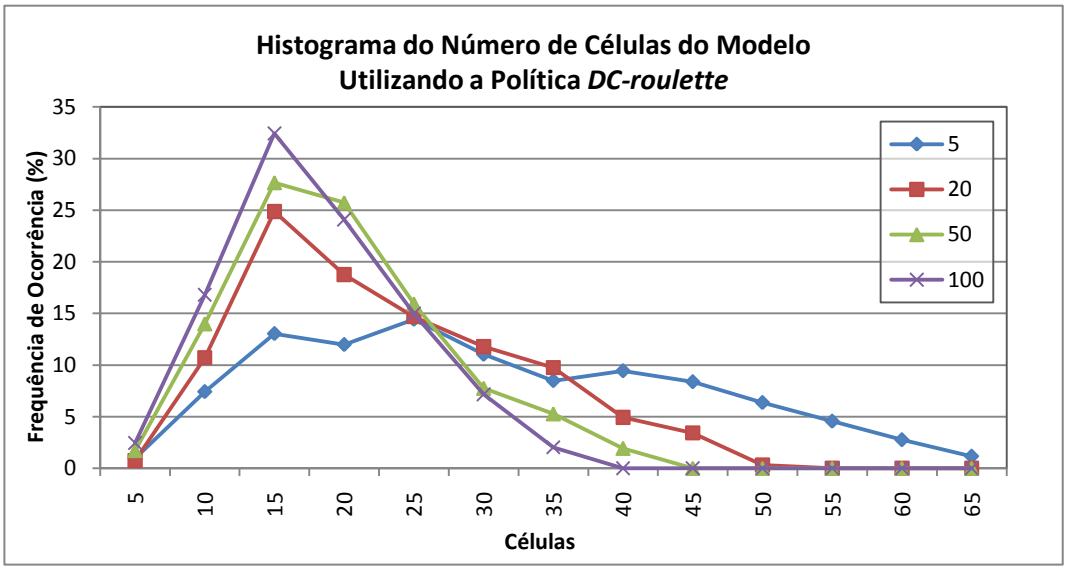

Gráfico 4.20: Histograma do número de células do modelo usando *DC-roulette.*

Utilizando-se a política *DC-roulette*, gráfico 4.20, a maior frequência de ocorrência de células do modelo é de 15 para *n* iguais a 20, 50 e 100. Quando *n* = 5 a distribuição torna-se mais dispersa, ou seja, com maior desvio padrão, e possui mediana igual a 26 células.

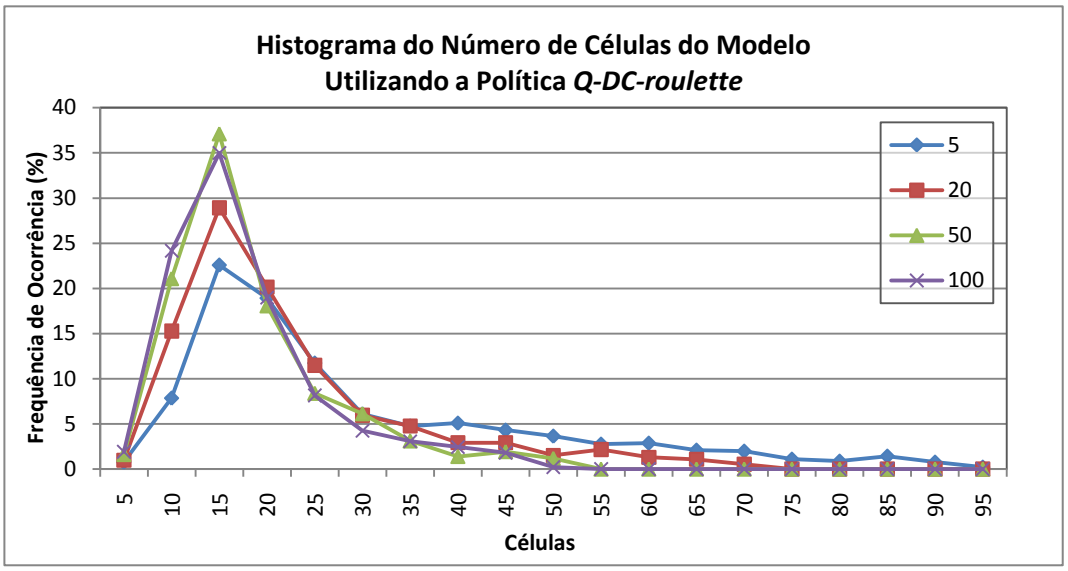

Gráfico 4.21: Histograma do número de células do modelo usando *Q-DC-roulette.*

A distribuição do número de células do modelo quando se utiliza a política *Q-DC-roulette*, gráfico 4.21, parece não se alterar significativamente com a variação do parâmetro *n*, sendo 15 o número de células do modelo de maior ocorrência.

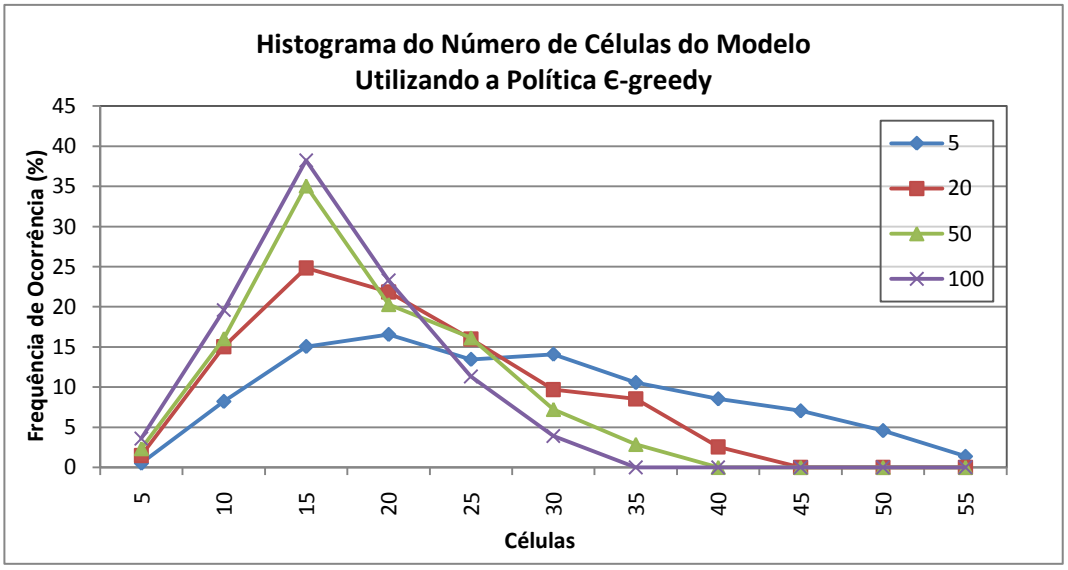

Gráfico 4.22: Histograma do número de células do modelo usando *ε-greedy.*

Observa-se pelo gráfico 4.22 que a maior frequência de ocorrência de células do modelo é de 15 para *n* iguais a 20, 50 e 100. Quando *n*=5 a distribuição torna-se mais esparramada com mediana igual a 24 células. Este comportamento se assemelha ao da política *DC-roulette*.

A tabela 4.3 traz um resumo da variação do número de épocas para treinamento e do número de células para diferentes políticas de escolha de ação e diferentes valores de *n*. No geral, o número de épocas de treinamento com maior probabilidade de ocorrência para as diferentes políticas de seleção de ação não se altera com a variação de *n*, sendo igual a 20. Para os casos *DC-roulette* e *ε-greedy*, o número de células do modelo tem probabilidade de ser ligeiramente maior e com maior dispersão quando se utiliza n=5 do que para valores maiores de n. O número de células sofre pouca influência com *n* e tem maior frequência de ocorrência de 15 para a política *Q-DC-roulette*.

Tabela 4.3: Comparação entre valores de *n* da função de crescimento para as diferentes políticas no modelo RL-NFHP.

Política	Nº de Épocas para Treinamento				Nº de Células			
	n=5	n=20	n=50	n=100	n=5	n=20	n=50	n=100
DC-roulette	10	20	20	20	25	15	15	15
Q-DC-roulette	20	20	20	30	15	15	15	15
ε-greedy	20	20	30	30	20	15	15	15

4.1.4 Experimento 4: *Elegibility Trace* Cumulativo

Nos testes do modelo RL-NFHP modificado conduzidos nesta subseção, foi alterado apenas o tamanho do rastro do *eligibility trace* cumulativo (seção 3.3): sem *eligibility*, dois passos para trás e cinco passos para trás.

Os demais parâmetros do modelo foram mantidos fixos e as condições iniciais foram as mesmas mostradas no experimento 1 (subseção 4.1.1). As políticas de escolha das ações utilizadas foram: *DC-Roulette*, *DC-Q-Roulette* e *ε-greedy*.

Os gráficos 4.23 a 4.25 mostram o histograma do número de épocas necessárias para treinamento sem eligibility, com dois passos para trás e quatro passos para trás usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente. A fim de facilitar a visualização, os tradicionais histogramas em barras foram substituídos por linhas contínuas como na subseção 4.1.3.

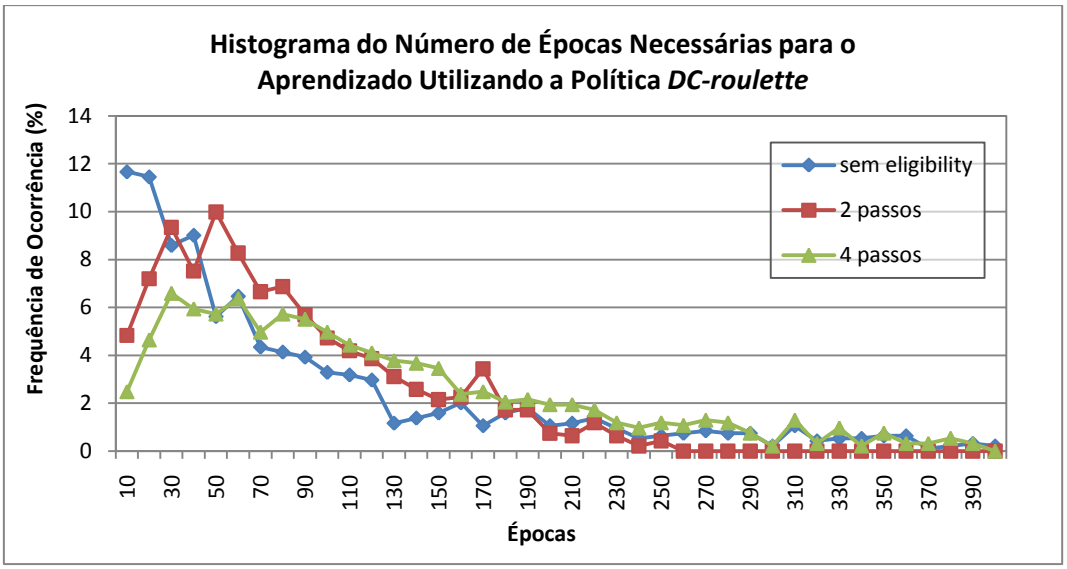

Gráfico 4.23: Histograma das épocas de treinamento usando *DC-roulette*.

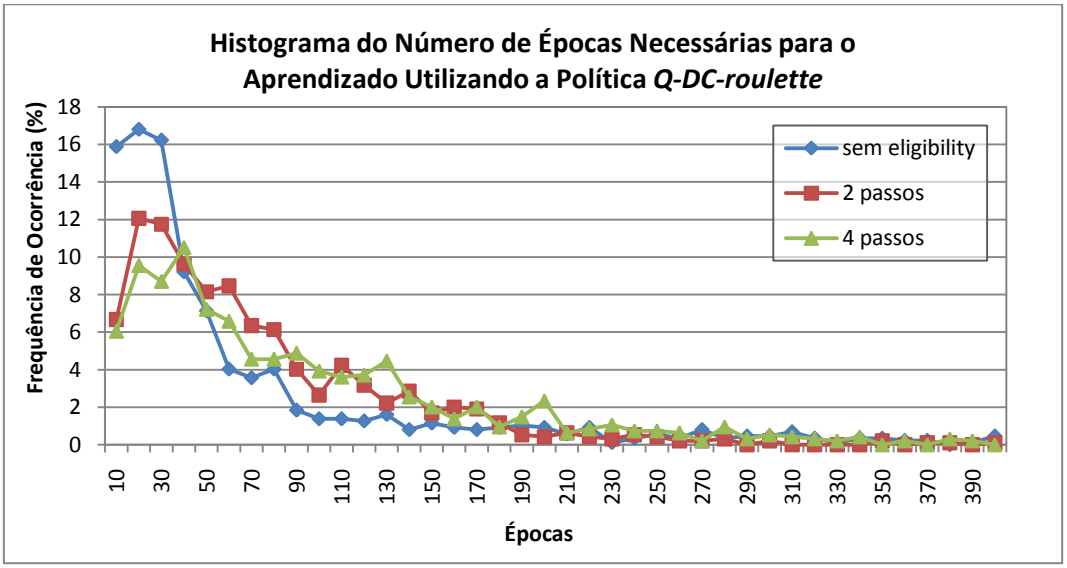

Gráfico 4.24: Histograma das épocas de treinamento usando *Q-DC-roulette*.

Ao contrário do que era esperado, observa-se, pelos gráficos 4.23 a 4.25, que em todas as políticas testadas (*DC-roulette*, *Q-DC-roulette* e *ε-greedy*), o aumento do *eligibility trace* retardou o aprendizado. Ou seja, as maiores frequências de ocorrência de aprendizado acontecem em um número maior de épocas à medida que se consideram mais passos para trás.

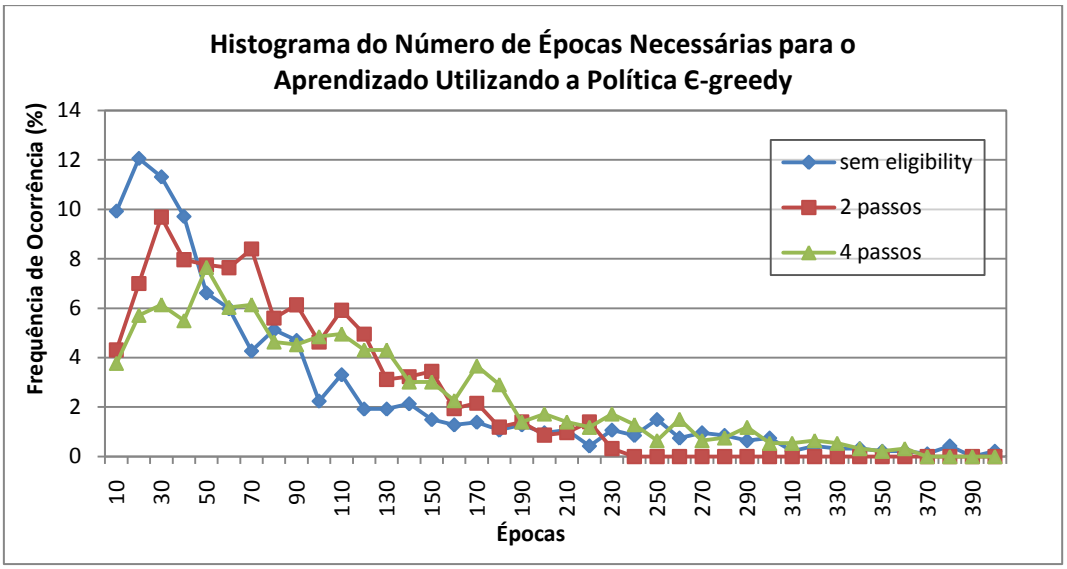

Gráfico 4.25: Histograma das épocas de treinamento usando *ε-greedy*.

A não utilização do *eligibility trace* trouxe melhores resultados: as maiores ocorrências de aprendizado acontecem em um número menor de épocas. Além disto, o histograma das épocas de treinamento é menos disperso quando não se utiliza o *eligibility trace*.

Este resultado é específico, não podendo ser estendido para o uso de qualquer *eligibility trace*. Como foi mencionado anteriormente, o *eligibility* implementado é cumulativo, desconsidera que o estado anterior possa ter sido particionado e que a ação possa ter sido escolhida de maneira aleatória. É bom reforçar que o *eligibility* aplicado a esse modelo não é trivial, pois a extrutura do modelo varia por meio de particionamentos sucessivos e a atualização de células que não aprenderam ainda pode causar problemas. Outros tipos de *eligibility trace* podem apresentar desempenho superior ao apresentado.

Os gráficos 4.26 a 4.28 mostram o histograma do número de células do modelo após treinamento bem sucedido sem *eligibility*, com dois passos para trás e quatro passos para trás usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

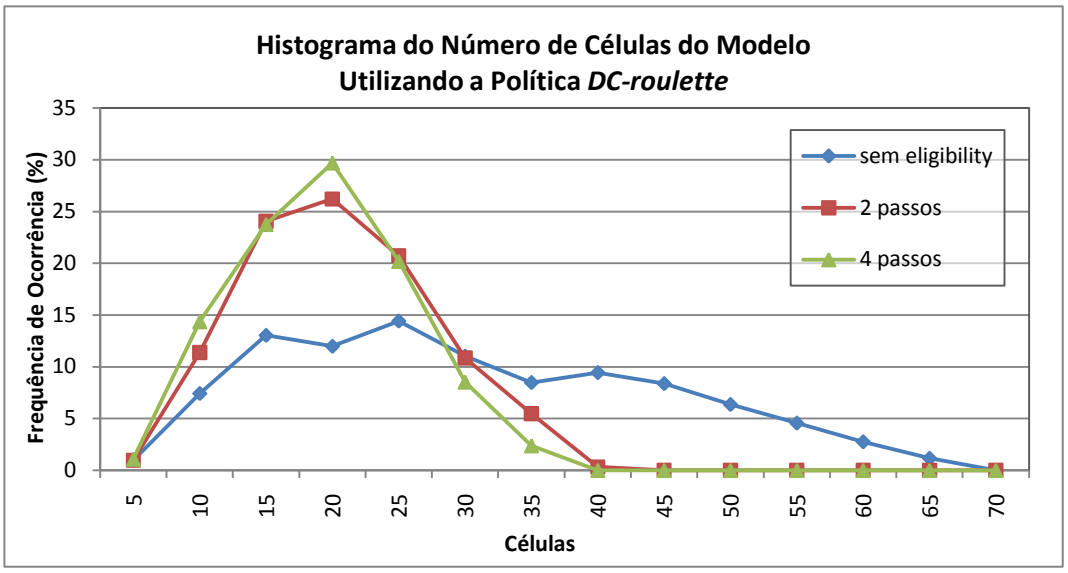

Gráfico 4.26: Histograma do número de células do modelo usando *DC-roulette.*

Observa-se que no aprendizado utilizando *eligibility trace*, com 2 ou 4 passos, para as políticas *DC-roulette* e *ε-greedy*, gráficos 4.26 e 4.28, tende a diminuir a dispersão do número de células do modelo no histograma quando comparado ao aprendizado sem *elegibility*. Isto torna a distribuição mais semelhante à distribuição normal e aumenta a probabilidade de existirem menos células no modelo após aprendizado. O número de células do modelo com maior frequência de ocorrência não altera significativamente ficando em torno de 20 a 25 células.

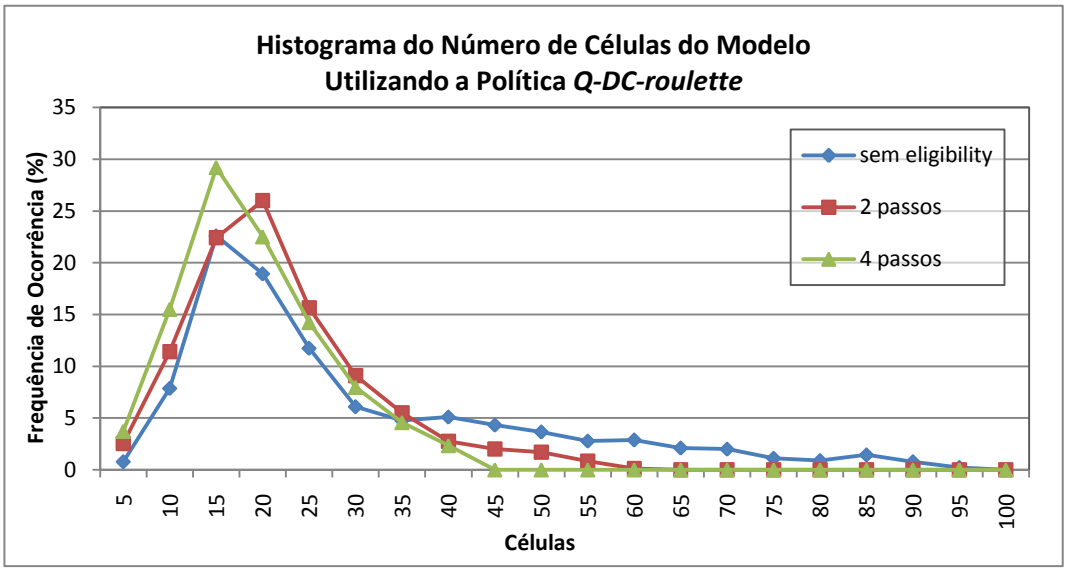

Gráfico 4.27: Histograma do número de células do modelo usando *Q-DC-roulette.*

O gráfico 4.27 mostra que a distribuição do número de células do modelo para a política *Q-DC-roulette* não se altera significativamente com o uso do *eligibility trace*, assim como o número de células de maior frequência entre 15 e 20.

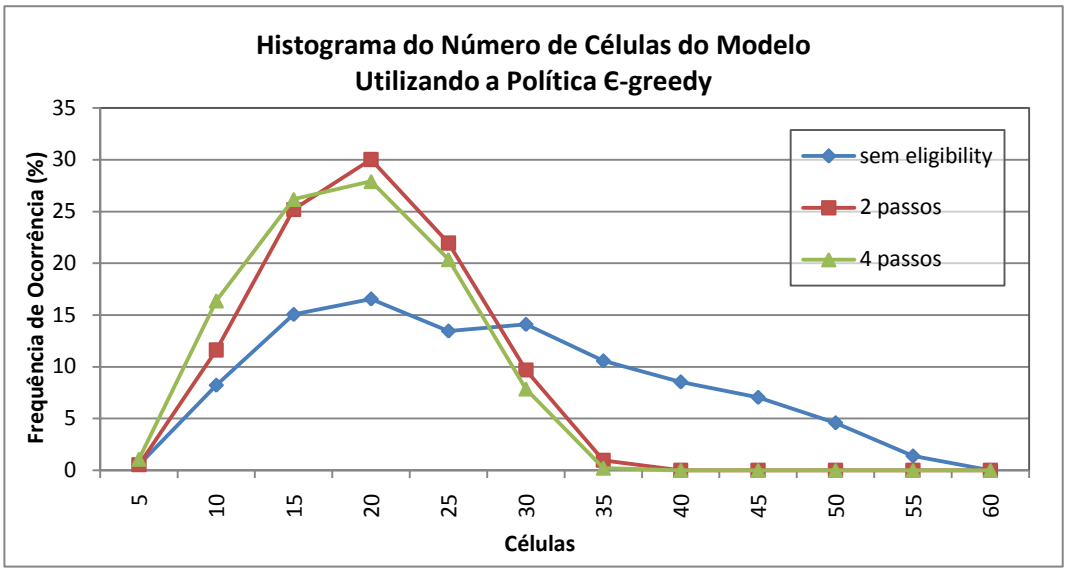

Gráfico 4.28: Histograma do número de células do modelo usando *ε-greedy.*

4.1.5
Experimento 5: Alfa de atualização de Q(s,a)

Nos testes do modelo RL-NFHP modificado realizados no experimento 5, foi alterado apenas o valor do parâmetro α da equação de atualização (eq. 2.17) da função de valor *Q* do par estado-ação. Este parâmetro assumiu os seguintes valores: razão entre o reforço da polipartição e o reforço total (r/R) e valores fixos de 10% e 50%.

Os demais parâmetros do modelo foram mantidos fixos e as condições iniciais foram as mesmas mostrados no experimento 1 (subseção 4.1.1). As políticas de escolha das ações utilizadas foram: *DC-Roulette*, *DC-Q-Roulette* e *ε-greedy*.

Os gráficos 4.29 a 4.31 mostram o histograma do número de épocas necessárias para treinamento para valores do parâmetro α da equação de atualização de *Q(s,a)* dados pela razão entre o reforço da partição e o reforço total (r/R), 10% e 50% usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*,

respectivamente. Com o intuito de facilitar a visualização, os tradicionais histogramas em barras foram substituídos por linhas contínuas como na subseção 4.1.3.

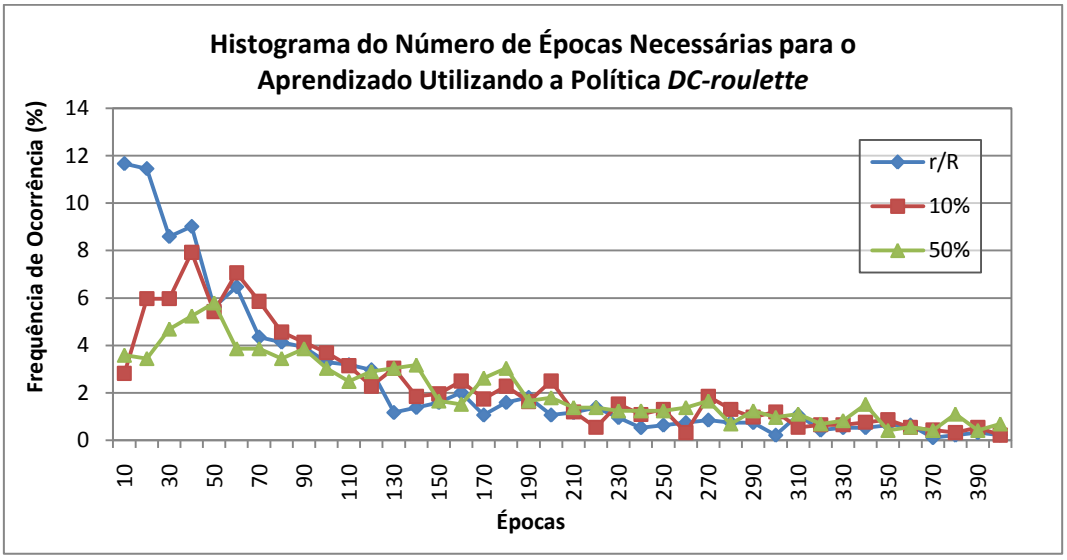

Gráfico 4.29: Histograma das épocas de treinamento usando *DC-roulette*.

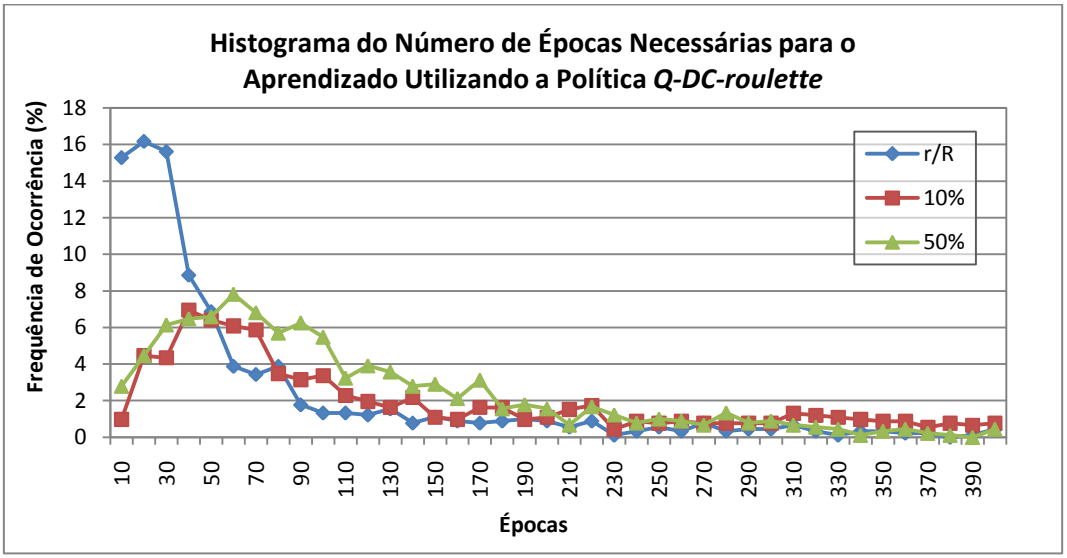

Gráfico 4.30: Histograma das épocas de treinamento usando *Q-DC-roulette*.

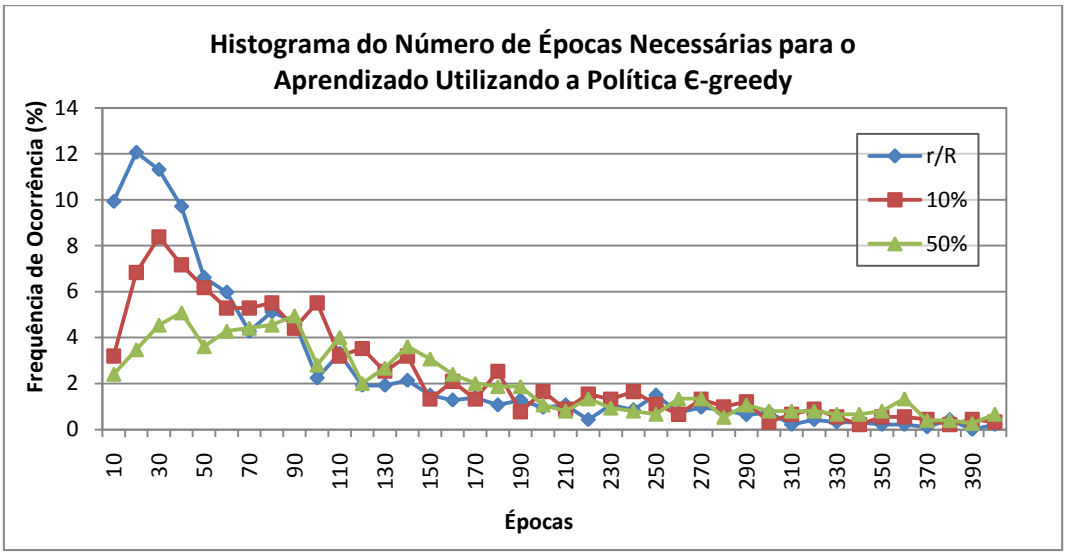

Gráfico 4.31: Histograma das épocas de treinamento usando *ε-greedy*.

Observa-se, pelos gráficos 4.29 a 4.31, que em todas as políticas testadas (*DC-roulette*, *Q-DC-roulette* e *ε-greedy*), o valor fixo do parâmetro α de atualização retardou o aprendizado. Isto é, as maiores frequências de ocorrência de aprendizado acontecem em um número maior de épocas para α de 10% e 50%. O histograma das épocas de treinamento para valores de α fixos, mostrou-se como uma distribuição mais dispersa que o dado pela razão entre o reforço da partição e o reforço total (r/R).

Os gráficos 4.32 a 4.34 mostram o histograma do número de células do modelo após treinamento bem sucedido para valores do parâmetro α da equação de atualização de *Q(s,a)* dados pela razão entre o reforço da partição e o reforço total (r/R), 10% e 50% usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

Observa-se que no aprendizado α de atualização de *Q(s,a)* fixo 10% e 50%, para todas as políticas (*DC-roulette*, *Q-DC-roulette* e *ε-greedy*), gráficos 4.32 a 4.34, tende a diminuir a dispersão do número de células do modelo no histograma comparado a α = r/R. Isto aumenta a probabilidade de realizar o aprendizado com um menor número de células no modelo. O número de células do modelo com maior frequência de ocorrência é em torno de 15 a 25 para α = r/R, 10 para α = 10% e 5 para α = 50%.

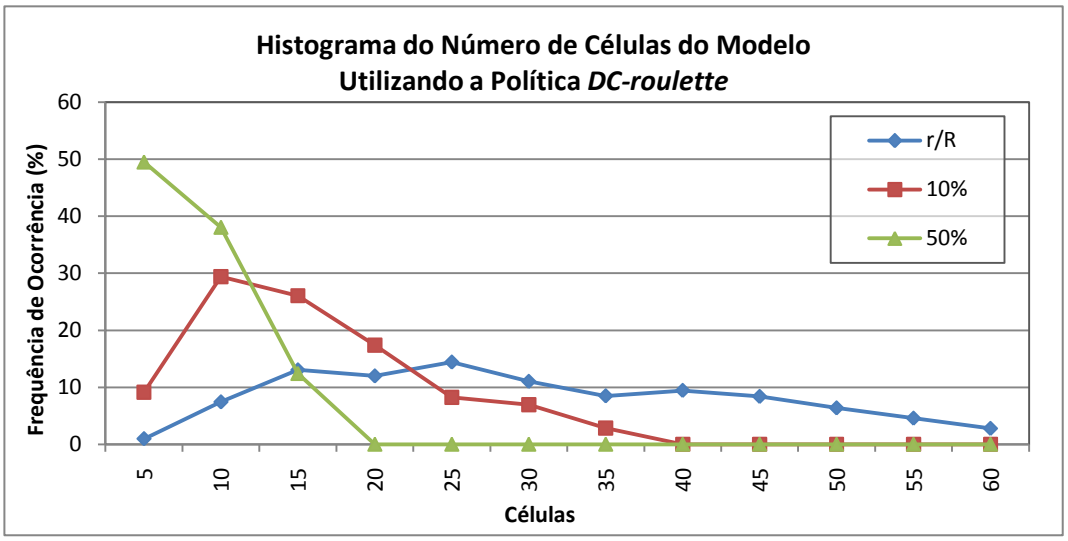

Gráfico 4.32: Histograma do número de células do modelo usando *DC-roulette.*

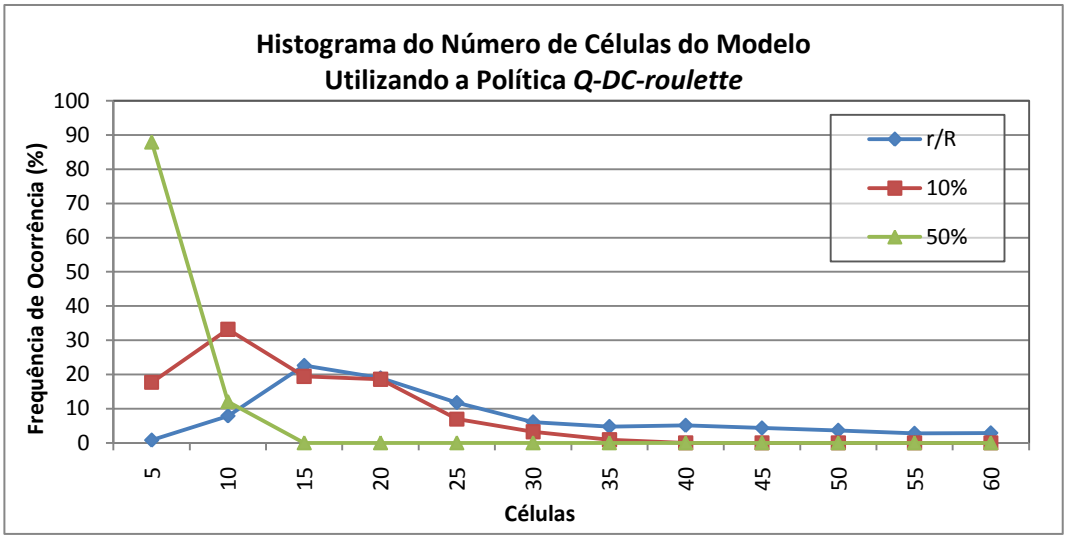

Gráfico 4.33: Histograma do número de células do modelo usando *Q-DC-roulette.*

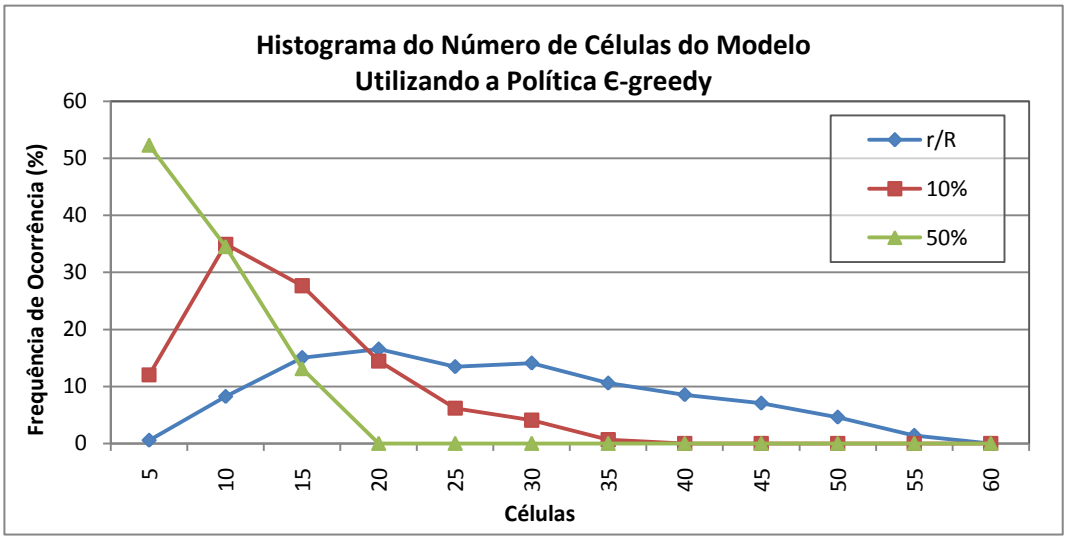

Gráfico 4.34: Histograma do número de células do modelo usando *ε-greedy*

A tabela 4.4 expõe um resumo do número de épocas para treinamento e do número de células, relacionados ao máximo de sucessos obtidos, para diferentes políticas de escolha de ação e diferentes valores de α de atualização da função valor *Q(s,a)*.

Tabela 4.4: Comparação entre valores do parâmetro α usado na atualização de Q(s,a) para as diferentes políticas no modelo RL-NFHP.

Política	**Nº de Épocas para Treinamento**			**Nº de Células**		
	α = r/R	α = 10%	α = 50%	α = r/R	α = 10%	α = 50%
DC-roulette	10	50	50	25	10	5
Q-DC-roulette	20	40	60	15	10	5
ε-greedy	20	30	40	20	10	5

Em todas as políticas de seleção de ação, o número de épocas de aprendizado, relacionado a maior probabilidade de ocorrência de sucesso, é maior e possui maior dispersão em α = 10% e α = 50%, do que para α = r/R. Em relação ao número de células, observa-se o comportamento oposto: maior probabilidade de número de células da estrutura e maior dispersão no caso de α = r/R.

Uma provável explicação para o comportamento do número de células do modelo RL-NFHP modificado e da quantidade de épocas de treinamento quando se utiliza α de 10% e 50% é a maior dificuldade no particionamento das células, gerando estruturas menores que não descrevem tão bem o espaço de estado do ambiente e, por isto, demoram mais a aprender (maior número de épocas de treinamento).

4.1.6
Experimento 6: Função de Atualização

Nos testes do modelo RL-NFHP modificado realizados no experimento 6, foi alterado apenas a forma de atualização da função valor *Q* (subseção 2.5.6). Dois testes forma realizados: no primeiro, durante o processo de aprendizado, a atualização foi feita com as duas equações 2.17 e 2.18, dependendo se o reforço da iteração seguinte era maior ($R_t < R_{t+1}$) ou menor ($R_t \geq R_{t+1}$) que o da iteração

anterior, respectivamente; posteriormente esta atualização passou a ser feita somente com a eq. 2.17, ou seja, sem a equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*).

Os demais parâmetros do modelo foram mantidos fixos e as condições iniciais foram as mesmas mostrados no experimento 1 (subseção 4.1.1). As políticas de escolha das ações utilizadas foram: *DC-roulette*, *Q-DC-roulette* e *ε-greedy*.

Os gráficos 4.35 a 4.37 mostram o histograma do número de épocas necessárias para treinamento para atualização da função valor *Q* com e sem a equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*) usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

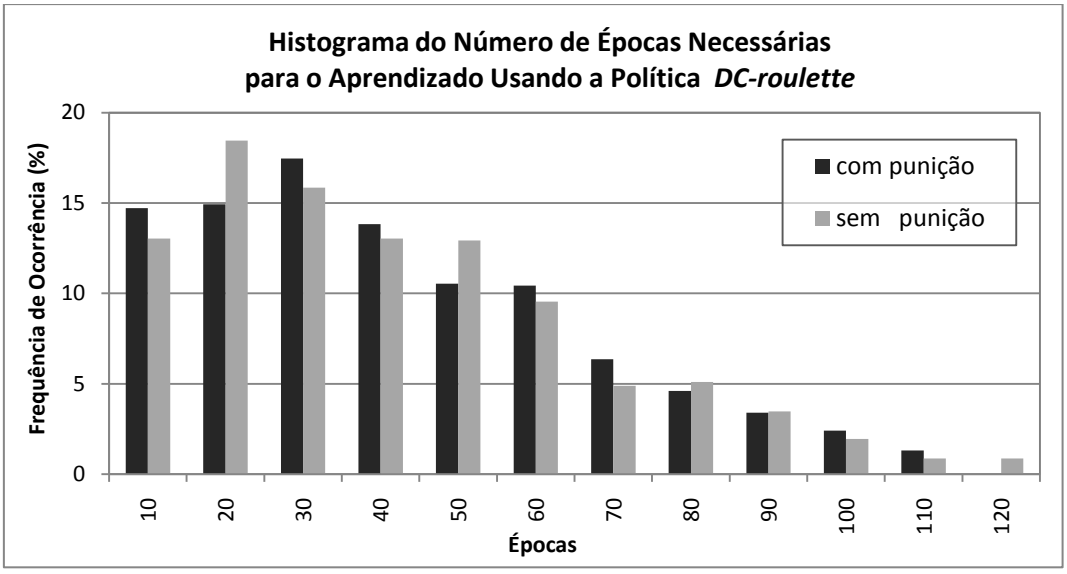

Gráfico 4.35: Histograma das épocas de treinamento usando *DC-roulette*.

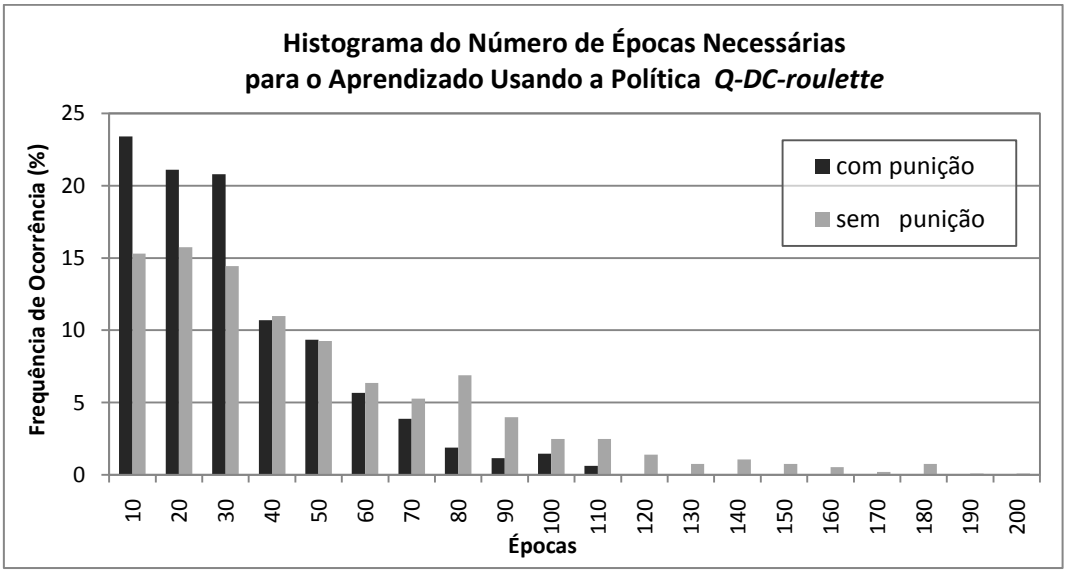

Gráfico 4.36: Histograma das épocas de treinamento usando *Q-DC-roulette*.

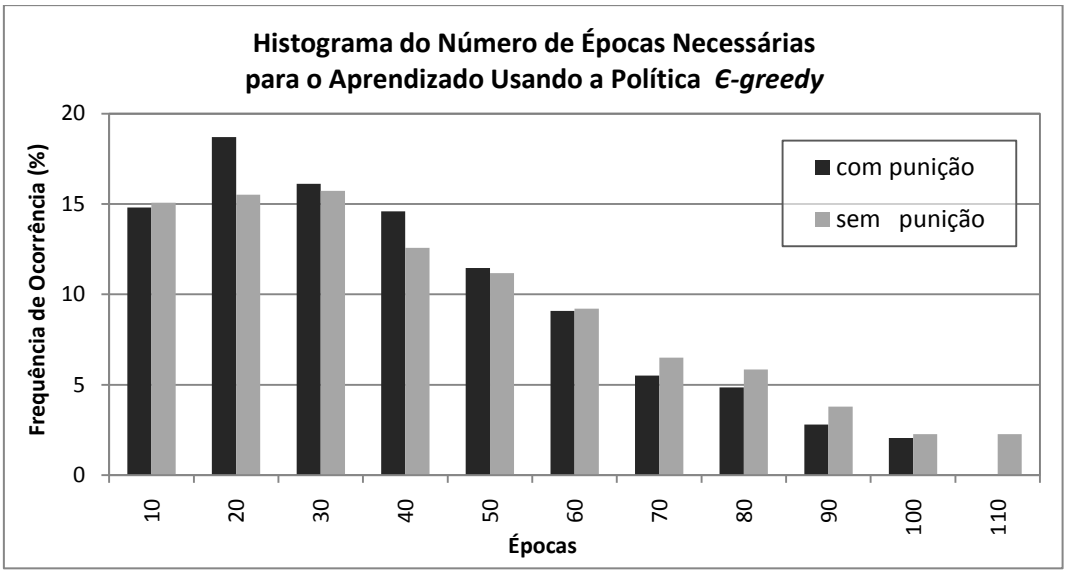

Gráfico 4.37: Histograma das épocas de treinamento usando *ε-greedy*.

O gráfico 4.35 revela que, para as políticas *DC-roulette*, a atualização sem o uso da equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*) possui aprendizado ligeiramente mais rápido do que com o uso da punição. A maior frequência de ocorrência de aprendizado acontece em 20 e 30 épocas, sem e com o uso da equação Q(s,*a*) = (1-*fp*)Q(s,*a*), respectivamente. Interessante notar que ambas possuem a mesma frequência de ocorrência de 47% de aprender em até 30 épocas.

Observa-se, pelos gráficos 4.36 e 4.37, que, para as políticas *Q-DC-roulette* e *ε-greedy*, a atualização com o uso da equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*) acelerou o aprendizado, sendo mais sensível para a política *Q-DC-roulette*. Para a política *Q-DC-roulette*, a maior frequência de aprendizado ocorre em 10 e 20 épocas, com e sem o uso da equação de punição, respectivamente. Já para a política *ε-greedy*, a maior frequência de ocorrência de aprendizado é um pouco mais elevada, 20 e 30 épocas. Ocorre um maior espalhamento do número de épocas necessárias para o aprendizado quando se utiliza somente a equação de atualização 2.17 para as duas políticas testadas. Vale ressaltar, que no caso da política *Q-DC-roulette*, a cauda da distribuição de ocorrência de aprendizado para a não utilização da equação de punição 2.18 é cerca de duas vezes maior que a distribuição ao se utilizar esta equação, chegando a 240 épocas.

Os gráficos 4.38 a 4.40 mostram o histograma do número de células do modelo após treinamento bem sucedido para atualização da função valor Q com e

sem a equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*) usando a política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, respectivamente.

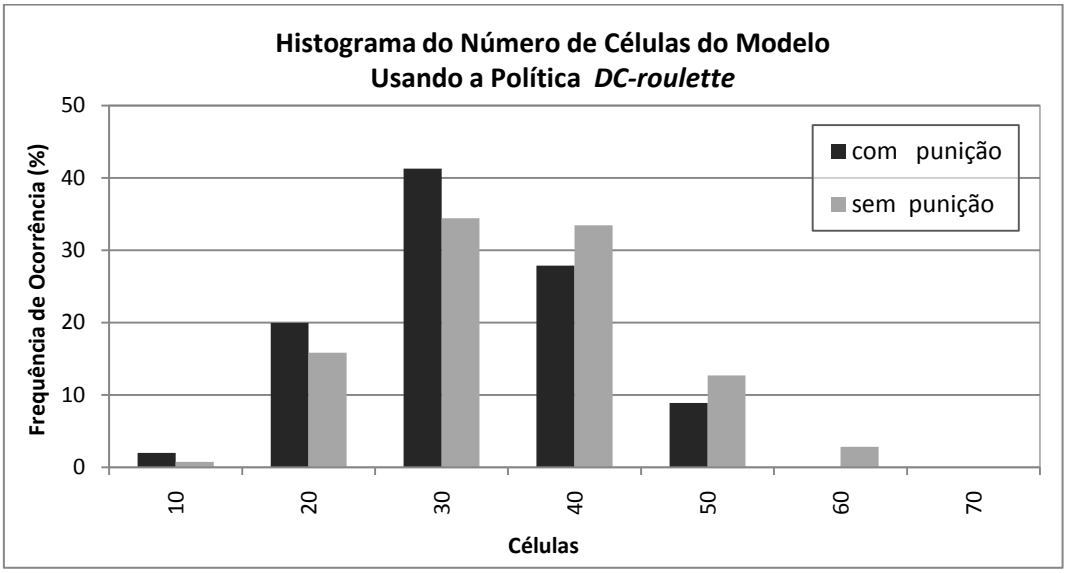

Gráfico 4.38: Histograma do número de células do modelo usando *DC-Roulette*.

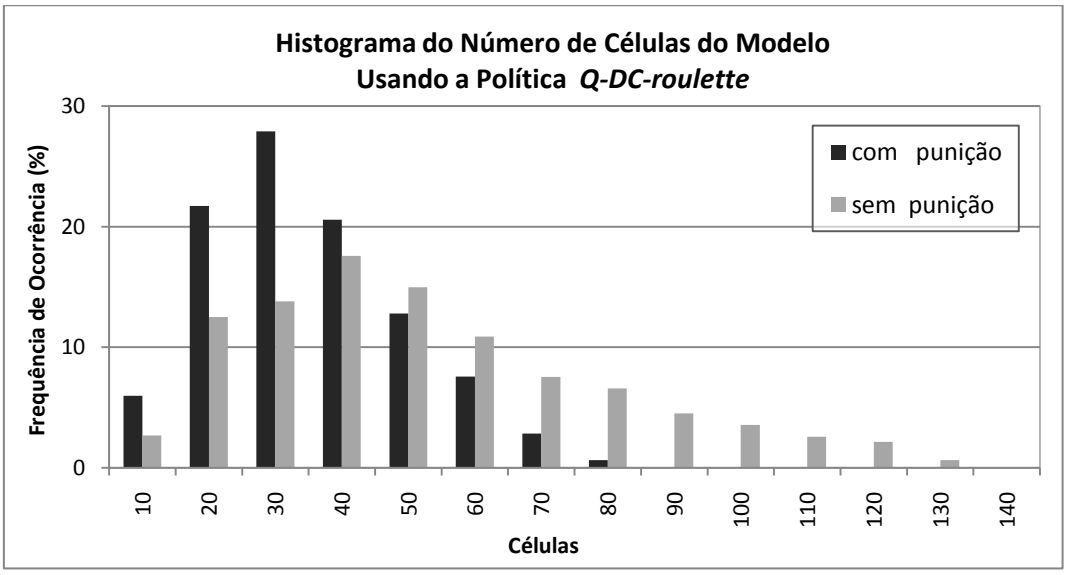

Gráfico 4.39: Histograma do número de células do modelo usando *Q-DC-Roulette*.

Observa-se, pelos gráficos 4.38, 4.39 e 4.40, para as política *DC-roulette*, *Q-DC-roulette* e *ε-greedy*, que o modelo treinado com a equação de punição, geralmente possui menos células. A maior frequência de células do modelo, com e sem o uso da equação de punição, acontece em 30 e 35 para as políticas *DC-roulette* e *ε-greedy*, e 30 e 40 células para *Q-DC-roulette*, respectivamente. Vale ressaltar que, na utilização da política *Q-DC-roulette*, ocorre um maior

espalhamento do número de células – maior desvio padrão – quando se utiliza somente a equação de atualização 2.17, a cauda da distribuição para a não utilização da equação de punição 2.18 é bem maior que a distribuição ao se utilizar esta equação, chegando a 130 células.

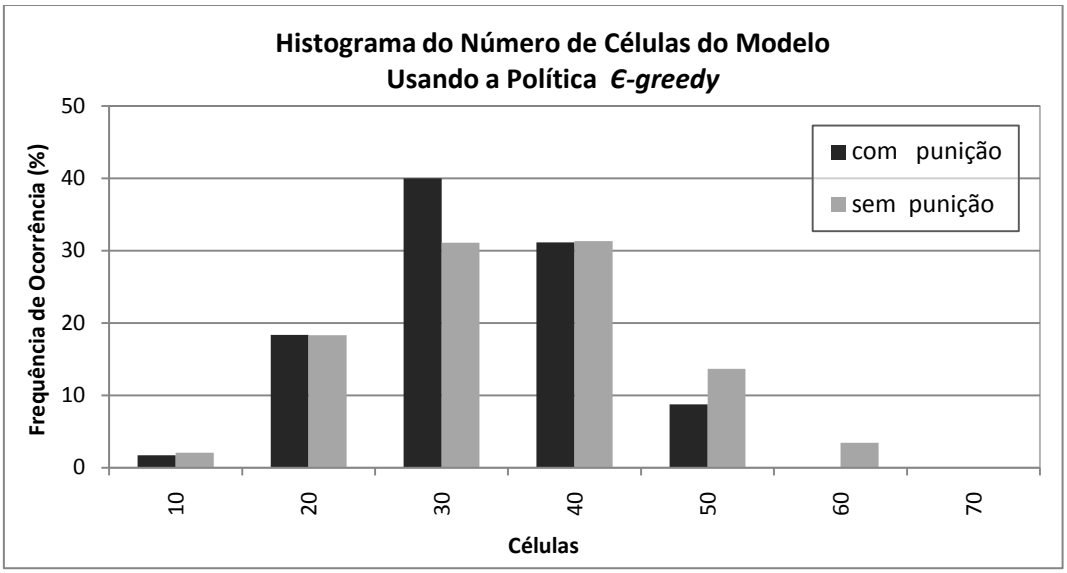

Gráfico 4.40: Histograma do número de células do modelo usando *ε-greedy.*

A tabela 4.5 expõe um resumo da variação do número de épocas para treinamento e do número de células com relação à maior probabilidade de ocorrência para diferentes políticas de escolha de ação e atualização da função valor *Q(s,a)* com e sem a equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*).

Tabela 4.5: Comparação entre as formas de atualização de Q(s,*a*), com e sem a equação de punição, para as diferentes políticas no modelo RL-NFHP.

Política	Nº de Épocas para Treinamento		Nº de Células	
	com a eq. de punição	sem a eq. de punição	com a eq. de punição	sem a eq. de punição
DC-roulette	30	20	30	35
Q-DC-roulette	10	20	30	40
ε-greedy	20	30	30	35

A influência da inserção da equação de punição Q(s,*a*) = (1-*fp*)Q(s,*a*) é mais visível para a política *Q-DC-roulette*. Ao se utilizá-la, tanto o número de épocas de treinamento quanto o de células do modelo treinado possui uma frequência de ocorrência maior em menores valores com o uso da equação de punição. Nota-se

também um maior espalhamento com o uso desta política de seleção associada a não utilização da punição. Neste caso, ocorre maior probabilidade de criação de modelos menores ao se utilizar a equação de punição. A inclusão desta equação ajuda na atualização da função valor *Q*, pois explicita que aquela combinação de ações para determinado conjunto de células ativas não gerou bons resultados – reforço do ambiente menor na iteração seguinte. Os testes com as políticas *DC-roulette* e *ε-greedy* se mostraram inconclusivos.

4.1.7
Experimento 7: Comparação entre modelos

Os testes conduzidos no experimento 7 visaram comparar o modelo RL-NFHP apresentado neste trabalho com diferentes modelos baseados em RL desenvolvidos por outros pesquisadores: Neural-Q-Learning, CMAC Q-Learning, FQL e RL-NFHB.

Os modelos Neural, CMAC (Miller et al., 1990) e FQL (Jouffe, 1998) possuem as seguintes configurações e características: o modelo Neural, implementado segundo Lin (1992), possui uma rede neural para cada ação discreta e as saídas dessas redes são as funções de valor Q associadas a cada uma dessas ações. Cada rede possui uma única camada escondida com quatro neurônios e um neurônio na camada de saída. As funções de ativação da camada escondida e de saída são tangente hiperbólica e linear, respectivamente. Os pesos das sinapses foram atualizados segundo o algoritmo de *backpropagation*, com taxas de aprendizado iguais a 0,0001 e 0,001 para a primeira e segundas camadas, respectivamente, e *momentum* igual a 0,5; o modelo CMAC possui 343 *grids*; o modelo FQL baseado em SIF (Sistema de Inferência Fuzzy) possui 5 funções de pertinência triangulares distribuídas no universo de discurso das variáveis de estado posição e velocidade. Os parâmetros RL usados neste modelo possuem os seguintes valores: $\lambda = 0,9$, $\vartheta_i = 0,01$, $\kappa = 0,5\vartheta_i$, $\phi = 0,2$, $\varphi = 0,5$.

No modelo RL-NFHB (*Reinforcement Learning Neuro-Fuzzy Hierarchical BSP*), proposto por Figueiredo (2003), o valor definido para o parâmetro γ da equação de atualização dos valores *Q* (eq. 2.17) foi de 1,0, a política *non-greedy* usada foi a de selecionar aleatoriamente a ação e o valor inicial do parâmetro ε (da política *ε*-greedy) foi estabelecido em 0,1 no momento de criação

da célula. A taxa de incremento/decremento deste parâmetro quando a ação era respectivamente de punição ou prêmio, foi de 5%. Vale a pena lembrar que este valor representa a probabilidade de se realizar a seleção da ação utilizando-se métodos *non-greedy* (por exemplo, escolha aleatória da ação) e (1-ε) representa a probabilidade de se usar a ação que tiver associada ao maior *Q*. O carro partia aleatoriamente, a cada episódio, de qualquer posição pertencente ao intervalo [-1,2;0,5] com qualquer velocidade entre [-0,07; 0,07]. O conjunto de ações possíveis associadas às células é igual a {-10; -5; -1; 0; 1; 5; 10}, após a defuzzificação o valor de saída da estrutura têm os seus limites superior e inferior fixados em 1 e -1, respectivamente.

Através das análises dos resultados dos experimentos anteriores, ajustaram-se os parâmetros do modelo RL-NFHP modificado. O valor definido para o parâmetro γ da equação de atualização dos valores *Q(s,a)* (eq. 2.17) foi de 1,0 e o *Q(s',a')* de atualização foi a média ponderada das funções de valor pelos alfas das células participantes. O parâmetro ε associado a probabilidade de escolha aleatória de ação foi iniciado com 10% no momento de criação da célula. A taxa de incremento/decremento deste parâmetro quando a ação era respectivamente de punição ou prêmio, foi de 5%. O parâmetro *n* da função de crescimento utilizada no particionamento da célula, que também depende do número de ciclos e do número de épocas, teve seu valor fixado em 5.

As condições iniciais foram alternadas entre (x_0, v_0) = (1,2;0) e (0,5;0), junto à parede e no vale com velocidade nula, respectivamente.

A política de escolha das ações utilizadas foi a política *Q-DC-Roulette* com probabilidade p = ε de escolha da ação por meio da nova roleta *Q-DC-roulette* apresentada neste trabalho e probabilidade p = 1-ε de seleção gulosa.

O conjunto de ações possíveis associadas às células é igual a {-1;0;1}, e a ação executada pelo agente após o processo de defuzzificação é submetido a uma função limitadora. Essa função tem por objetivo tornar o valor de saída discreto nos seguintes casos: se o valor na saída da estrutura estiver entre 0,1 e 1, o valor da ação será 1; se estiver entre -0,1 e -1 o valor será -1, e se o valor estiver entre -0,1 e 0,1 permanecerá o calculado na defuzzificação. Esta função limitadora proposta por Figueiredo (2003) tem por objetivo permitir a comparação com modelos existentes, que têm por característica saídas discretas.

Esta função limitadora foi criada para contornar um aspecto menos favorável do modelo RL-NFHP: sua dificuldade para solucionar adequadamente problemas que exijam saídas discretas. Este fato deve-se à generalização inerente ao particionamento fuzzy do espaço de entrada (Figueiredo, 2003). O comportamento aprendido pelo agente deve responder adequadamente a um domínio que corresponde a um número significativo de estados do ambiente. Dessa forma, o agente não consegue ter um comportamento discreto.

Outra possibilidade de contornar este problema é a utilização de ações acima dos limites do problema e saturar as saídas do modelo RL-NFHP como sugerido também por Figueiredo (2003). Nesse caso utilizou-se um conjunto de ações possíveis associadas às células igual a {-10; -5; -1; 0; 1; 5; 10} para o *benchmark* carro na montanha que permite ações entre -1 e 1. Após a defuzzificação, o valor de saída da estrutura têm os seus limites superior e inferior fixados em 1 e -1, respectivamente.

A tabela 4.6 compara os resultados obtidos nas fases de aprendizado e de teste dos modelos Neural-Q-Learning, CMAC Q-Learning, FQL, RL-NFHB e RL-NFHP. O modelo RL-NFHP o exposto por Figueiredo (2003) está identificado como RL-NFHP, enquanto o modelo RL-NFHP^{++} é o desenvolvido neste trabalho.

Tabela 4.6: Comparação entre modelos para o *benchmark* Carro na Montanha.

Modelo	Característica	Fase de Aprendizado	Fase de Teste
Neural-Q-learning	4	1724	2189
CMAC Q-learning	343	262	85
FQL	25	112	61
RL-NFHB	0	131	71
RL-NFHP	0	91	69
RL-NFHP^{++}	**0**	**136**	**63**

A primeira coluna da tabela 4.6 indica o modelo, a segunda coluna indica a característica, ou seja, o número de neurônios na camada escondida para o caso do Neural Q-Learning, o número de *grids* para o caso CMAC e o número de regras para o FQL. A terceira e quarta colunas indicam o desempenho obtido para cada modelo em termos de números de passos.

É importante ressaltar que o número de características dos modelos RL-NFHB, RL-NFHP e RL-NFHP^{++} foram considerados zero, porque os

conjuntos fuzzy iniciais, definidos pela primeira célula, serão subdivididos um número de vezes não determinado no início do processo de aprendizado. Apesar de alguns parâmetros terem que ser ajustados no modelo RL-NFHP, estes também não são considerados como características.

Segundo Jouffe (1998), a explicação para a Rede Neural ter o pior resultado deve-se ao fato de que, além do aprendizado dos valores Q, ela também precisa aprender a identificar os estados, assim como nos modelos propostos nesta tese. Porém as modificações feitas nos pesos da rede pelo algoritmo *backpropagation* afetam toda a rede, prejudicando o aprendizado. Nos modelos CMAC e FQL os estados são fixados a priori (*grids* e regras) e as modificações das funções de valor que ocorrem para o caso CMAC e FQL afetam somente as regras ou os *grids* que estão ativos em cada passo do aprendizado.

A percepção sobre o conjunto de estados ativos pelo modelo CMAC é discreta (muitos estados vizinhos ativam os mesmos conjuntos de *grids*); já no caso dos modelos FQL, RL-NFHB, RL-NFHP e RL-NFHP^{++} a percepção sobre o conjunto de estados é contínua. A generalização introduzida no espaço de entrada pelas regras fuzzy é mais suave do que a introduzida pelos *grids* do modelo CMAC. Por esse motivo os resultados dos modelos FQL, RL-NFHB, RL-NFHP e RL-NFHP^{++} apresentam resultados superiores.

No entanto, apesar do resultado do FQL ser ligeiramente melhor que os resultados obtidos pelos modelos desenvolvidos neste trabalho, ele parte, assim como o CMAC, de informações prévias (regras e conjuntos fuzzy) relativas ao processo de aprendizado. Em relação ao modelo FQL, os modelos RL-NFHB, RL-NFHP e RL-NFHP^{++} utilizam uma função de reforço um pouco mais elaborada, pois no modelo FQL a função de reforço é diferenciada em 3 funções relacionadas a partes do espaço de estados, e em uma delas utiliza uma função baseada em uma função linear entre -1 e 1 para atribuir diferentes reforços dependendo do estado.

A tabela 4.7, mostra a comparação entre os modelos RL-NFHP: o exposto por Figueiredo (2003) identificado como RL-NFHP e o modelo RL-NFHP^{++} desenvolvido neste trabalho. A primeira coluna representa o modelo, a segunda, o número de épocas necessário no treinamento para o aprendizado, a terceira, o número de células do modelo após treinamento (tamanho da estrutura) e a quarta e

a quinta, a média de passos desde a posição inicial até o objetivo durante o treinamento e durante o teste, respectivamente.

Tabela 4.7: Comparação entre modelos RL-NFHP para o *benchmark* Carro na Montanha.

Modelo	Número de Épocas	Número de Células	Média de Passos	
			Fase de Aprendizado	Fase de Teste
RL-NFHP	5000	96	91	69
RL-NFHP^{++}	**540**	**23**	**136**	**63**

O modelo RL-NFHP^{++} treinou cerca de 10 vezes mais rápido que a sua versão original RL-NFHP, além de aprensentar desempenho ligeiramente melhor nos testes, observado pelo menor número de passos para cumprimento do objetivo.

A estrutura do modelo RL-NFHP^{++} é bem menor que a do RL-NFHP, apresentando apenas 23 células. Este fato acarreta em um processamento computacional mais rápido na utilização pelo agente do modelo treinado.

A discrepância em relação ao número médio de passos da fase de treino da tabela 4.7 ocorre devido às condições iniciais e ao conjunto de ações possíveis adotados. O modelo RL-NFHP utiliza a configuração de condições iniciais e de ações possíveis igual a mostrada para o modelo RL-NFHB, enquanto este trabalho utiliza posições iniciais no pico e no vale com velocidades nulas e ações possíveis {-1;0;1}. Embora se tenha utilizado outro ajuste de parâmetros e a implementação computacional tenha sido inteiramente refeita. Isto revela o quão robusto são o modelo RL-NFHP e sua versão modificada.

Os gráficos 4.41 e 4.42 mostram a posição e velocidade do carro na fase de teste para o modelo RL-NFHP modificado, exposto na tabela 4.7, para as posições iniciais -0,5 (vale da montanha, pior ponto de partida) e -1,2 (junto à parede, melhor ponto de partida) com velocidade nula, respectivamente.

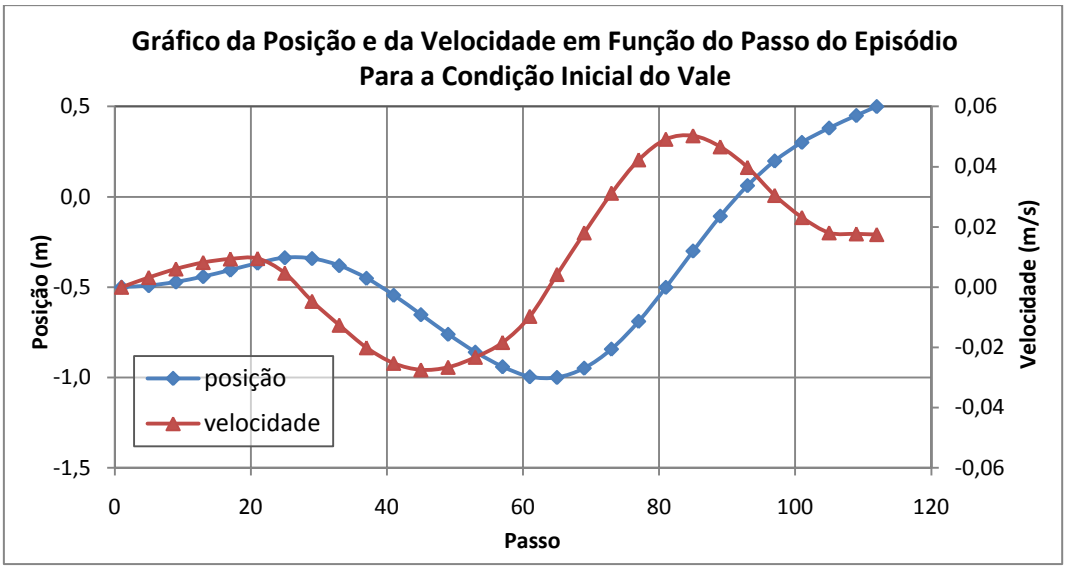

Gráfico 4.41: Posição e velocidade do carro ao longo do episódio iniciando no vale.

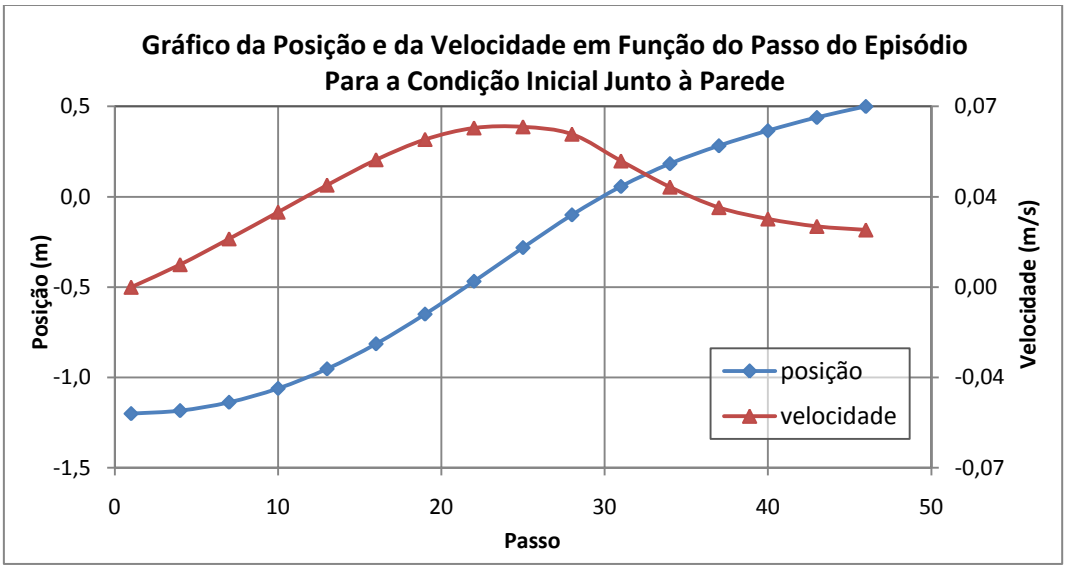

Gráfico 4.42: Posição e velocidade do carro ao longo do episódio iniciando junto à parede.

Pode-se observar no gráfico 4.41 que, quando o carro parte do vale da montanha (posição -0,5), ele necessita oscilar de um lado para outro para aumentar o suficiente a sua velocidade, de modo a adquirir energia potencial para atingir o objetivo que é o pico da montanha (posição 0,5). Por outro lado, quando o carro parte junto à parede (posição -1,2), gráfico 4.42, ele já possui energia potencial suficiente para atingir o objetivo, necessitando apenas acelerar.

A figura 4.3 mostra o particionamento do estado (distância-velocidade) após aprendizado do modelo RL-NFHP modificado de 23 células mostrado anteriormente. O eixo das abscissas é composto pela velocidade normalizada

[-1;1], enquanto o eixo das ordenadas representa a distância normalizada [0;1]. A estrutura em árvore das células deste mesmo modelo RL-NFHP modificado treinado encontra-se na figura 4.4. Observando a árvore (figura 4.4) é possível entender melhor o particionamento do estado. A título de exemplo, a célula 2 representa o quadrado com vértices {(-1;0),(-1;0,5),(0;0,5),(0;0)} e a célula 13 o quadrado com vértices {(0;0,25),(0;0,5),(0,5;0,5),(0,5;0,25)}.

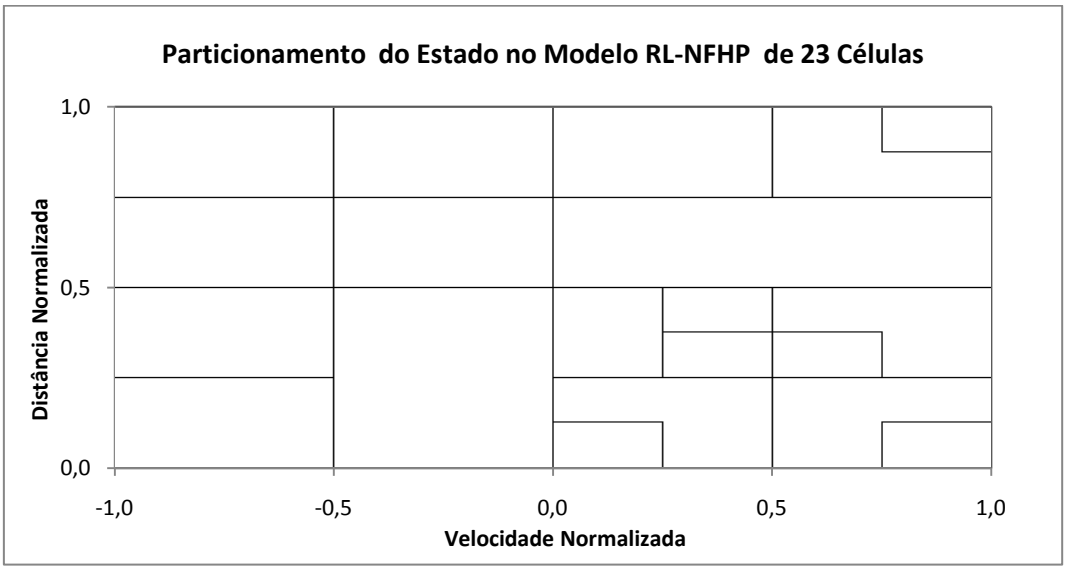

Figura 4.3: Particionamento do estado no modelo RL-NFHP modificado de 23 células.

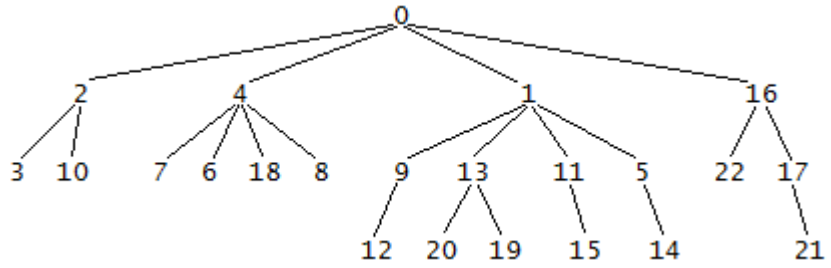

Figura 4.4: Árvore de células do modelo RL-NFHP modificado de 23 células.

O modelo não subdividiu o espaço de estados da mesma forma, ele especificou mais apenas algumas regiões. Constata-se que foi necessário um maior particionamento da região de velocidade normalizada positiva [0;1], deslocamento para a direita, e de distância normalizada baixa [0;0,5], a direita do vale da montanha, esperado como sendo o comportamento mais difícil. Nesta região o agente deve aprender que mesmo se dirigindo para o objetivo (velocidade positiva), caso ele não tenha energia suficiente para transpor a montanha, deve se

dirigir para a direção oposta (velocidade negativa) a fim de oscilar e ganhar esta energia.

O conjunto de regras que traduz o conhecimento linguístico do modelo RL-NFHP modificado exposto nas figuras 4.3 e 4.4 é:

Se $x_1 \in {}^0\rho_1$ e $x_2 \in {}^0\rho_2$ então
{
  Se $x_1 \in {}^2\rho_1$ e $x_2 \in {}^2\rho_2$ então
  {
    Se $x_1 \in {}^3\rho_1$ e $x_2 \in {}^3\rho_2$ então $y = {}^3a_1$
    Se $x_1 \in {}^3\mu_1$ e $x_2 \in {}^3\rho_2$ então $y = {}^3a_2$
    Se $x_1 \in {}^3\rho_1$ e $x_2 \in {}^3\mu_2$ então $y = {}^3a_3$
    Se $x_1 \in {}^3\mu_1$ e $x_2 \in {}^3\mu_2$ então $y = {}^3a_4$
  }
  Se $x_1 \in {}^2\mu_1$ e $x_2 \in {}^2\rho_2$ então
  {
    Se $x_1 \in {}^{10}\rho_1$ e $x_2 \in {}^{10}\rho_2$ então $y = {}^{10}a_1$
    Se $x_1 \in {}^{10}\mu_1$ e $x_2 \in {}^{10}\rho_2$ então $y = {}^{10}a_2$
    Se $x_1 \in {}^{10}\rho_1$ e $x_2 \in {}^{10}\mu_2$ então $y = {}^{10}a_3$
    Se $x_1 \in {}^{10}\mu_1$ e $x_2 \in {}^{10}\mu_2$ então $y = {}^{10}a_4$
  }
  Se $x_1 \in {}^2\rho_1$ e $x_2 \in {}^2\mu_2$ então $y = {}^2a_3$
  Se $x_1 \in {}^2\mu_1$ e $x_2 \in {}^2\mu_2$ então $y = {}^2a_4$
}
Se $x_1 \in {}^0\mu_1$ e $x_2 \in {}^0\rho_2$ então
{
  Se $x_1 \in {}^4\rho_1$ e $x_2 \in {}^4\rho_2$ então
  {
    Se $x_1 \in {}^7\rho_1$ e $x_2 \in {}^7\rho_2$ então $y = {}^7a_1$
    Se $x_1 \in {}^7\mu_1$ e $x_2 \in {}^7\rho_2$ então $y = {}^7a_2$
    Se $x_1 \in {}^7\rho_1$ e $x_2 \in {}^7\mu_2$ então $y = {}^7a_3$
    Se $x_1 \in {}^7\mu_1$ e $x_2 \in {}^7\mu_2$ então $y = {}^7a_4$
  }
  Se $x_1 \in {}^4\mu_1$ e $x_2 \in {}^4\rho_2$ então
  {
    Se $x_1 \in {}^6\rho_1$ e $x_2 \in {}^6\rho_2$ então $y = {}^6a_1$
    Se $x_1 \in {}^6\mu_1$ e $x_2 \in {}^6\rho_2$ então $y = {}^6a_2$
    Se $x_1 \in {}^6\rho_1$ e $x_2 \in {}^6\mu_2$ então $y = {}^6a_3$
    Se $x_1 \in {}^6\mu_1$ e $x_2 \in {}^6\mu_2$ então $y = {}^6a_4$
  }
  Se $x_1 \in {}^4\rho_1$ e $x_2 \in {}^4\mu_2$ então
  {
    Se $x_1 \in {}^{18}\rho_1$ e $x_2 \in {}^{18}\rho_2$ então $y = {}^{18}a_1$
    Se $x_1 \in {}^{18}\mu_1$ e $x_2 \in {}^{18}\rho_2$ então $y = {}^{18}a_2$
    Se $x_1 \in {}^{18}\rho_1$ e $x_2 \in {}^{18}\mu_2$ então $y = {}^{18}a_3$
    Se $x_1 \in {}^{18}\mu_1$ e $x_2 \in {}^{18}\mu_2$ então $y = {}^{18}a_4$
  }

Se $x_1 \in {}^{4}\mu_1$ e $x_2 \in {}^{4}\mu_2$ então
{
Se $x_1 \in {}^{8}\rho_1$ e $x_2 \in {}^{8}\rho_2$ então $y = {}^{8}a_1$
Se $x_1 \in {}^{8}\mu_1$ e $x_2 \in {}^{8}\rho_2$ então $y = {}^{8}a_2$
Se $x_1 \in {}^{8}\rho_1$ e $x_2 \in {}^{8}\mu_2$ então $y = {}^{8}a_3$
Se $x_1 \in {}^{8}\mu_1$ e $x_2 \in {}^{8}\mu_2$ então $y = {}^{8}a_4$
}
}
Se $x_1 \in {}^{0}\rho_1$ e $x_2 \in {}^{0}\mu_2$ então
{
Se $x_1 \in {}^{1}\rho_1$ e $x_2 \in {}^{1}\rho_2$ então
{
Se $x_1 \in {}^{9}\rho_1$ e $x_2 \in {}^{9}\rho_2$ então
{
Se $x_1 \in {}^{12}\rho_1$ e $x_2 \in {}^{12}\rho_2$ então $y = {}^{12}a_1$
Se $x_1 \in {}^{12}\mu_1$ e $x_2 \in {}^{12}\rho_2$ então $y = {}^{12}a_2$
Se $x_1 \in {}^{12}\rho_1$ e $x_2 \in {}^{12}\mu_2$ então $y = {}^{12}a_3$
Se $x_1 \in {}^{12}\mu_1$ e $x_2 \in {}^{12}\mu_2$ então $y = {}^{12}a_4$
}
Se $x_1 \in {}^{9}\mu_1$ e $x_2 \in {}^{9}\rho_2$ então $y = {}^{9}a_2$
Se $x_1 \in {}^{9}\rho_1$ e $x_2 \in {}^{9}\mu_2$ então $y = {}^{9}a_3$
Se $x_1 \in {}^{9}\mu_1$ e $x_2 \in {}^{9}\mu_2$ então $y = {}^{9}a_4$
}
Se $x_1 \in {}^{1}\mu_1$ e $x_2 \in {}^{1}\rho_2$ então
{
Se $x_1 \in {}^{13}\rho_1$ e $x_2 \in {}^{13}\rho_2$ então $y = {}^{13}a_1$
Se $x_1 \in {}^{13}\mu_1$ e $x_2 \in {}^{13}\rho_2$ então $y = {}^{13}a_2$
Se $x_1 \in {}^{13}\rho_1$ e $x_2 \in {}^{13}\mu_2$ então
{
Se $x_1 \in {}^{20}\rho_1$ e $x_2 \in {}^{20}\rho_2$ então $y = {}^{20}a_1$
Se $x_1 \in {}^{20}\mu_1$ e $x_2 \in {}^{20}\rho_2$ então $y = {}^{20}a_2$
Se $x_1 \in {}^{20}\rho_1$ e $x_2 \in {}^{20}\mu_2$ então $y = {}^{20}a_3$
Se $x_1 \in {}^{20}\mu_1$ e $x_2 \in {}^{20}\mu_2$ então $y = {}^{20}a_4$
}
Se $x_1 \in {}^{13}\mu_1$ e $x_2 \in {}^{13}\mu_2$ então
{
Se $x_1 \in {}^{19}\rho_1$ e $x_2 \in {}^{19}\rho_2$ então $y = {}^{19}a_1$
Se $x_1 \in {}^{19}\mu_1$ e $x_2 \in {}^{19}\rho_2$ então $y = {}^{19}a_2$
Se $x_1 \in {}^{19}\rho_1$ e $x_2 \in {}^{19}\mu_2$ então $y = {}^{19}a_3$
Se $x_1 \in {}^{19}\mu_1$ e $x_2 \in {}^{19}\mu_2$ então $y = {}^{19}a_4$
}
}
Se $x_1 \in {}^{1}\rho_1$ e $x_2 \in {}^{1}\mu_2$ então
{
Se $x_1 \in {}^{11}\rho_1$ e $x_2 \in {}^{11}\rho_2$ então $y = {}^{11}a_1$
Se $x_1 \in {}^{11}\mu_1$ e $x_2 \in {}^{11}\rho_2$ então $y = {}^{11}a_2$
Se $x_1 \in {}^{11}\rho_1$ e $x_2 \in {}^{11}\mu_2$ então
{

Se $x_1 \in {}^{15}\rho_1$ e $x_2 \in {}^{15}\rho_2$ então $y = {}^{15}a_1$
Se $x_1 \in {}^{15}\mu_1$ e $x_2 \in {}^{15}\rho_2$ então $y = {}^{15}a_2$
Se $x_1 \in {}^{15}\rho_1$ e $x_2 \in {}^{15}\mu_2$ então $y = {}^{15}a_3$
Se $x_1 \in {}^{15}\mu_1$ e $x_2 \in {}^{15}\mu_2$ então $y = {}^{15}a_4$
}
Se $x_1 \in {}^{11}\mu_1$ e $x_2 \in {}^{11}\mu_2$ então $y = {}^{11}a_4$
}
Se $x_1 \in {}^{1}\mu_1$ e $x_2 \in {}^{1}\mu_2$ então
{
Se $x_1 \in {}^{5}\rho_1$ e $x_2 \in {}^{5}\rho_2$ então
{
Se $x_1 \in {}^{14}\rho_1$ e $x_2 \in {}^{14}\rho_2$ então $y = {}^{14}a_1$
Se $x_1 \in {}^{14}\mu_1$ e $x_2 \in {}^{14}\rho_2$ então $y = {}^{14}a_2$
Se $x_1 \in {}^{14}\rho_1$ e $x_2 \in {}^{14}\mu_2$ então $y = {}^{14}a_3$
Se $x_1 \in {}^{14}\mu_1$ e $x_2 \in {}^{14}\mu_2$ então $y = {}^{14}a_4$
}
Se $x_1 \in {}^{5}\mu_1$ e $x_2 \in {}^{5}\rho_2$ então $y = {}^{5}a_2$
Se $x_1 \in {}^{5}\rho_1$ e $x_2 \in {}^{5}\mu_2$ então $y = {}^{5}a_3$
Se $x_1 \in {}^{5}\mu_1$ e $x_2 \in {}^{5}\mu_2$ então $y = {}^{5}a_4$
}
}
Se $x_1 \in {}^{0}\mu_1$ e $x_2 \in {}^{0}\mu_2$ então
{
Se $x_1 \in {}^{16}\rho_1$ e $x_2 \in {}^{16}\rho_2$ então $y = {}^{16}a_1$
Se $x_1 \in {}^{16}\mu_1$ e $x_2 \in {}^{16}\rho_2$ então
{
Se $x_1 \in {}^{22}\rho_1$ e $x_2 \in {}^{22}\rho_2$ então $y = {}^{22}a_1$
Se $x_1 \in {}^{22}\mu_1$ e $x_2 \in {}^{22}\rho_2$ então $y = {}^{22}a_2$
Se $x_1 \in {}^{22}\rho_1$ e $x_2 \in {}^{22}\mu_2$ então $y = {}^{22}a_3$
Se $x_1 \in {}^{22}\mu_1$ e $x_2 \in {}^{22}\mu_2$ então $y = {}^{22}a_4$
}
Se $x_1 \in {}^{16}\rho_1$ e $x_2 \in {}^{16}\mu_2$ então $y = {}^{16}a_3$
Se $x_1 \in {}^{16}\mu_1$ e $x_2 \in {}^{16}\mu_2$ então
{
Se $x_1 \in {}^{17}\rho_1$ e $x_2 \in {}^{17}\rho_2$ então $y = {}^{17}a_1$
Se $x_1 \in {}^{17}\mu_1$ e $x_2 \in {}^{17}\rho_2$ então $y = {}^{17}a_2$
Se $x_1 \in {}^{17}\rho_1$ e $x_2 \in {}^{17}\mu_2$ então $y = {}^{17}a_3$
Se $x_1 \in {}^{17}\mu_1$ e $x_2 \in {}^{17}\mu_2$ então
{
Se $x_1 \in {}^{21}\rho_1$ e $x_2 \in {}^{21}\rho_2$ então $y = {}^{21}a_1$
Se $x_1 \in {}^{21}\mu_1$ e $x_2 \in {}^{21}\rho_2$ então $y = {}^{21}a_2$
Se $x_1 \in {}^{21}\rho_1$ e $x_2 \in {}^{21}\mu_2$ então $y = {}^{21}a_3$
Se $x_1 \in {}^{21}\mu_1$ e $x_2 \in {}^{21}\mu_2$ então $y = {}^{21}a_4$
}
}
}

onde:

- x_1 é a entrada relativa à posição e x_2 a entrada relativa à velocidade;
- ${}^i\rho_1$, ${}^i\rho_2$, ${}^i\mu_{1,}$ ${}^i\mu_2$ são as funções de pertinência *baixo* e *alto* que definem a partição da célula i;
- ${}^i a_1$, ${}^i a_2$, ${}^i a_3$, ${}^i a_4$ são as ações associadas as 4 polipartições.

O modelo RL-NFHP modificado foi capaz de aprender automaticamente sem que houvesse necessidade da definição prévia de regras ou de conjuntos fuzzy, e gerou resultados linguisticamente interpretáveis.

4.2
Aplicação simulada em robótica (*Khepera*)

O simulador do robô com movimento diferencial tipo Khepera foi implementado com base no simulador desenvolvido por Ostergard (2000) e nas equações de Dudek e Jenkin (2000).

O ambiente simulado é constituído por um tabuleiro de 1000x1000 mm limitado por paredes, contendo um robô de raio 80 mm, podendo ou não ter um obstáculo central octagonal conforme mostrado na figura 4.5.

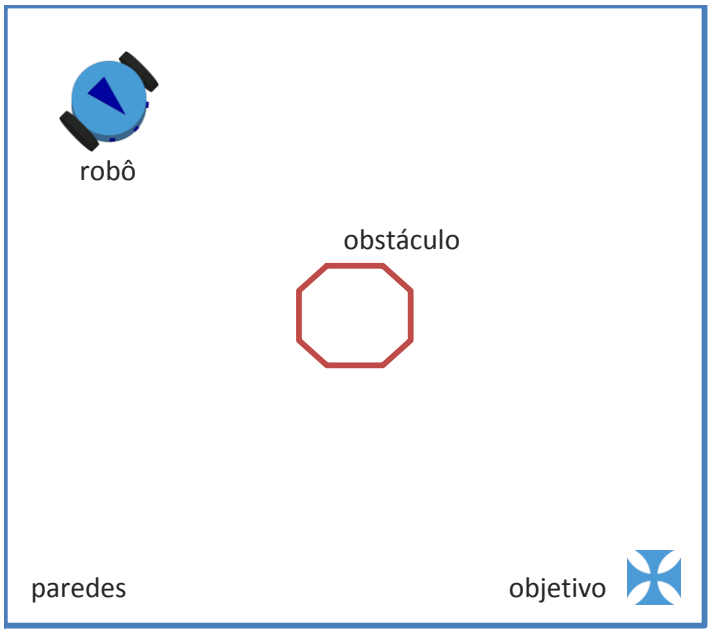

Figura 4.5: Ambiente de simulação do robô.

Para facilitar o processamento, o robô Khepera adotado na simulação possui apenas 3 sensores ultrassônicos (esquerda, frontal e direita) para detecção de proximidade de uma parede ou um obstáculo. O robô está dotado de um sensor de localização que fornece as coordenadas (x,y) de sua posição central no ambiente e de um sensor de rotação que fornece o ângulo θ entre a frente do robô e o eixo x.

A figura 4.6 mostra o mecanismo de detecção de um obstáculo pelo robô Khepera (em cinza) simulado. Nesta figura, a detecção do obstáculo (octaedro roxo) ocorre por meio dos sensores ultrassônicos frontal e lateral esquerdo (paralelepípedos azuis), que fornecem um valor de leitura proporcional a interseção da área de alcance de seus arcos (verde) com o obstáculo. Esta figura é meramente ilustrativa, e assim sendo, as dimensões do robô e os ângulos de abertura, raios de alcance e disposição dos sensores não podem ser utilizados para cálculo.

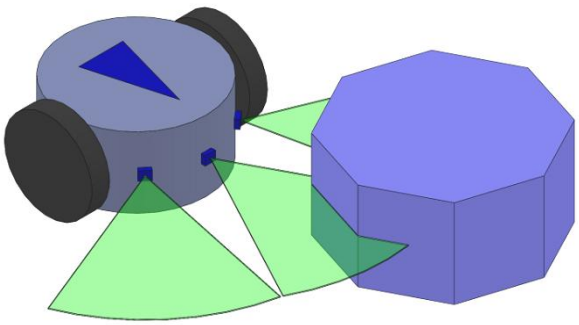

Figura 4.6: Detecção de obstáculo.

Para locomoção o robô Khepera possui 2 servo-motores independentes para mover as rodas da direita e da esquerda, cujas potências podem variar entre -20 e 20.

O simulador fornece a localização do robô, através das coordenadas da posição no ambiente (x,y). Com estas coordenadas e sabendo a posição final desejada (x_f,y_f) pode-se calcular a distância euclidiana entre o robô e o alvo por meio da eq. 4.3.

$$d = \sqrt{(x_f - x)^2 + (y_f - y)^2} \tag{4.3}$$

O ângulo mais importante para este problema específico é o ângulo entre o robô e a posição objetivo final do robô (α). Este ângulo pode ser calculado a partir do ângulo de rotação θ, das coordenadas do robô e da posição final desejada (figura 4.7) conforme eq. 4.4.

$$\alpha = \begin{cases} cos^{-1}(cosseno) & se\ seno \geq 0 \\ -cos^{-1}(cosseno) & se\ seno < 0 \end{cases}$$

$$cosseno = \frac{(x_f - x)\cos(\theta) + (y_f - y)\sin(\theta)}{d} \quad (4.4)$$

$$seno = \frac{(x_f - x)\sin(\theta) - (y_f - y)\cos(\theta)}{d}$$

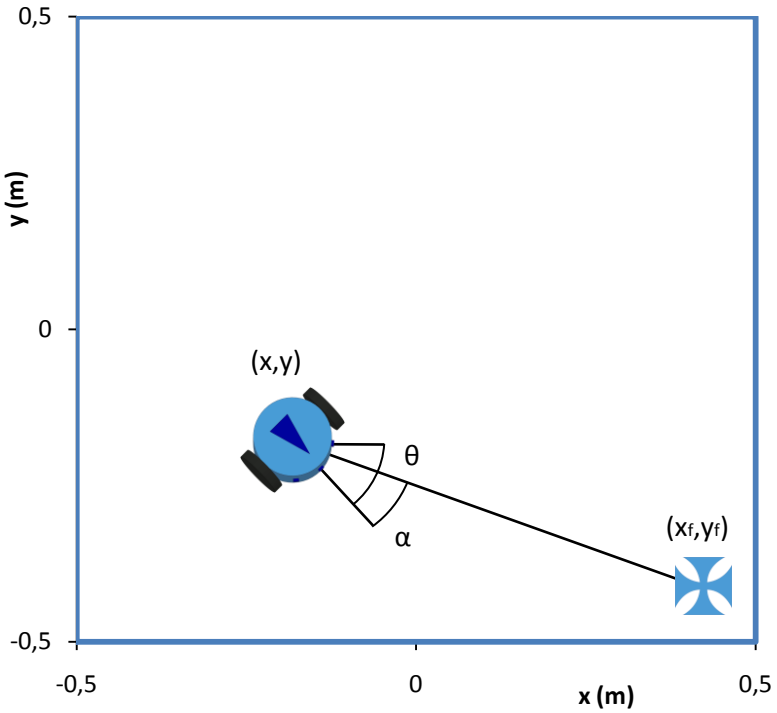

Figura 4.7: Identificação dos ângulos de interesse.

As 5 variáveis de entrada normalizadas do controlador do robô Khepera são:

- distância euclidiana entre o robô e o alvo d ;
- ângulo entre o robô e o objetivo α ;
- sensor frontal s_f ;
- sensor da esquerda s_l ;
- sensor da direita s_r .

O movimento do robô no espaço de simulação é definido pela posição (x,y) (orientação euclidiana) e pelo ângulo θ (orientação angular). A rotação $\Delta\theta$ e a translação do robô em relação ao eixo x, Δx, e ao eixo y, Δy, são calculadas com base nas velocidades angulares dos motores direito e esquerdo conforme eq. 4.5 (Dudek and Jenkin, 2000).

- Se $V_l = V_r$ (movimento sem rotação):

$$\Delta\theta = 0$$
$$\Delta x = V_r \Delta t \cos(\theta)$$
$$\Delta y = V_r \Delta t \sin(\theta)$$

- Se $V_l \neq V_r$:

$$\Delta\theta = \omega_{ICC}\Delta t$$
$$\Delta x = R_{ICC}[\cos(\Delta\theta)\sin(\theta) + \sin(\Delta\theta)\cos(\theta) - \sin(\theta)] \quad (4.5)$$
$$\Delta y = R_{ICC}[\sin(\Delta\theta)\sin(\theta) - \cos(\Delta\theta)\cos(\theta) + \cos(\theta)]$$

$$V_l = \omega_l R_{wheel} \quad e \quad V_r = \omega_r R_{wheel}$$
$$R_{ICC} = R_{robot}\frac{(V_r + V_l)}{(V_r - V_l)}$$
$$\omega_{ICC} = \frac{(V_r - V_l)}{2R_{robot}}$$

onde: ω_l é a velocidade angular da esquerda, ω_r a velocidade angular da direita, R_{wheel} o raio da roda, R_{robot} o raio do robô e Δt o passo de tempo da simulação.

A saída do controlador são as potências aplicadas aos atuadores, neste caso os motores das rodas esquerda e direita, tendo seus valores limitados entre –20 (velocidade angular máxima para trás) e 20 (velocidade angular máxima para frente).

A função de reforço do ambiente deve ser composta por parcelas relativas à distância angular, à distância euclidiana e à proximidade de obstáculos. Foram utilizadas diferentes funções de reforço nos experimentos do Khepera.

Os experimentos do robô Khepera simulado conduzidos nas subseções a seguir visam estudar a influência do uso das políticas de escolha das ações *Q-DC-roulette* e *ε-greedy* e do método de poda da estrutura no modelo RL-NFHP

modificado para o caso sem obstáculo; e, por meio destas informações, produzir o melhor modelo RL-NFHP modificado para o caso com obstáculos.

4.2.1
Experimento 1: Sem obstáculo

Inicialmente foram realizados testes no modelo RL-NFHP modificado no simulador Khepera sem obstáculos (figura 4.8) para duas políticas de escolha das ações: *Q-DC-roulette* e *ε-greedy*.

As demais variáveis foram mantidas fixas como será detalhado a seguir. O valor definido para o parâmetro γ da equação de atualização dos valores Q(s,*a*) (eq. 2.17) foi de 1,0 e o Q(s',*a*') de atualização foi a média ponderada das funções de valor pelos Ω_is das células participantes. O parâmetro ε associado a probabilidade de escolha aleatória de ação foi iniciado com 10% no momento de criação da célula. A taxa de incremento/decremento deste parâmetro quando a ação era respectivamente de punição ou prêmio, foi de 5%. O parâmetro *n* da função de crescimento utilizada no particionamento da célula (eq. 2.19) teve seu valor fixado em 200, o valor de incremento $\Delta\beta_1$ da variável de crescimento foi 0,1 e de decremento $\Delta\beta_2$ 0,2, não permitindo valores de β inferiores a zero.

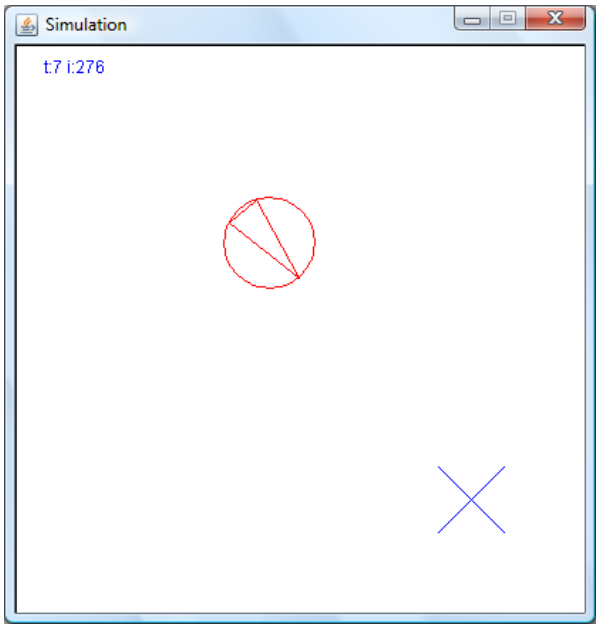

Figura 4.8: Ambiente de simulação do robô Khepera sem obstáculo.

As condições iniciais foram alternadas entre (x_0, y_0, θ_0) = (-0,3;-0,3;1,5), (0,3;0,3;-1,5) e (-0,3;0,3;-0,8), nos arredores dos três vértices afastados do objetivo. A posição final desejava foi (x_f, y_f) = (0,3;-0,3).

As ações possíveis, velocidades angulares das rodas da esquerda e da direita, foram: (20,20), (-10,10), (10,-10), (-10,-10), correspondentes a andar para frente, rotacionar no sentido trigonométrico, no sentido anti- trigonométrico e se deslocar para trás, respectivamente.

A função de reforço utilizada foi a soma ponderada de exponenciais da distância e ângulo e tem seu valor reduzido à medida que o robô se aproxima de obstáculos mostrada na eq. 4.6.

$$R = (k_1 e^{-(d)} + k_2 e^{-|\alpha|})[1 - (0{,}6\, s_f + 0{,}2\, s_l + 0{,}2\, s_r)] \tag{4.6}$$

onde, *d* é a distância euclidiana entre o robô e o alvo; α o ângulo entre o robô e o objetivo; s_f o sensor frontal; s_l o sensor da esquerda; s_r o sensor da direita; e k_1 e k_2 são constantes maiores do que 1 utilizadas para adequar os valores de reforço à estrutura do modelo.

Foi utilizado o método de *early stopping* (seção 3.2) para definir a o término do aprendizado. Este procedimento foi executado a cada 20 episódios, com validação através de 250 ou menos passos para que o robô alcançasse o objetivo. Foram realizados cerca de 1000 treinamentos com número máximo de 5000 épocas.

Os dados dos treinamentos que geraram aprendizado foram filtrados uma única vez, descartando aqueles que possuíam valores de número de épocas de treinamento ou número de células maior que a média mais dois desvios padrões ou menores que a média menos dois desvios padrões.

O gráfico 4.43 mostra o número de épocas necessárias para treinamento através das políticas *Q-DC-roulette* e *ε-greedy*. Por este gráfico observa-se o modelo RL-NFHP, na maioria das vezes com quase 20% da frequência de ocorrência, é treinado em apenas 40 épocas e dificilmente demora mais de 400 épocas, menos de 10% dos treinamentos, quando utiliza-se a política *Q-DC-roulette*. Já o uso da política *ε-greedy* torna em geral o treinamento mais lento, com média de 290 épocas, e mais esparramado, com desvio padrão de 172 épocas.

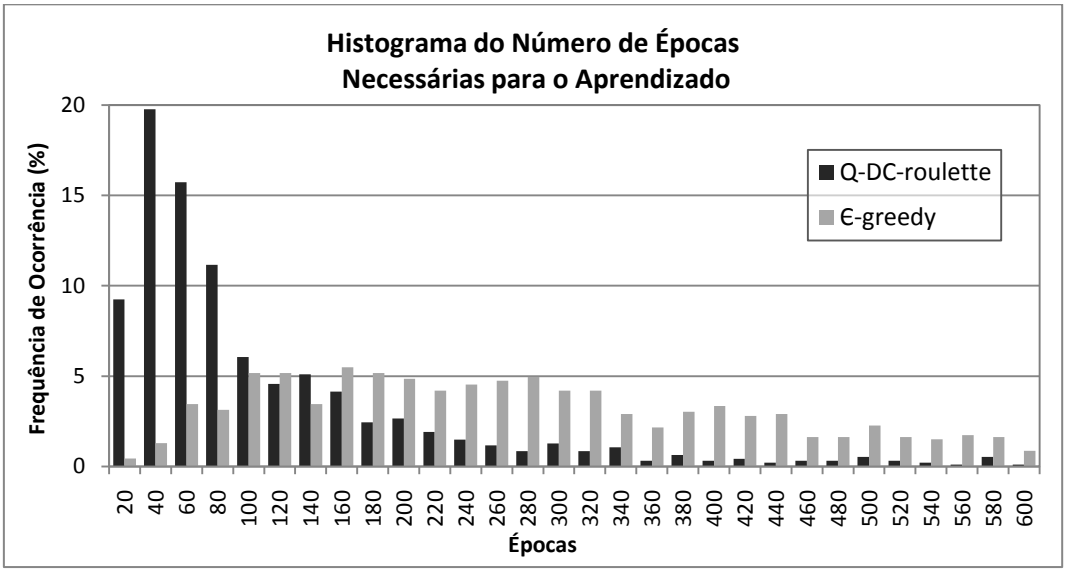

Gráfico 4.43: Histograma das épocas de treinamento.

O gráfico 4.44 expõe o número de células do modelo após treinamento. Usando a política *ε-greedy*, obtêm-se na média 520 células, com desvio padrão de 162, enquanto para a política *Q-DC-roulette* estes valores são maiores, com média de 624 células e desvio padrão de 231. O treinamento por meio da política *Q-DC-roulette* pode chegar ao máximo número de 1000 células permitido para o modelo. Isto encarece computacionalmente, pois refina demasiadamente o espaço de estados aumentando o número de cálculos, e deve ser evitado.

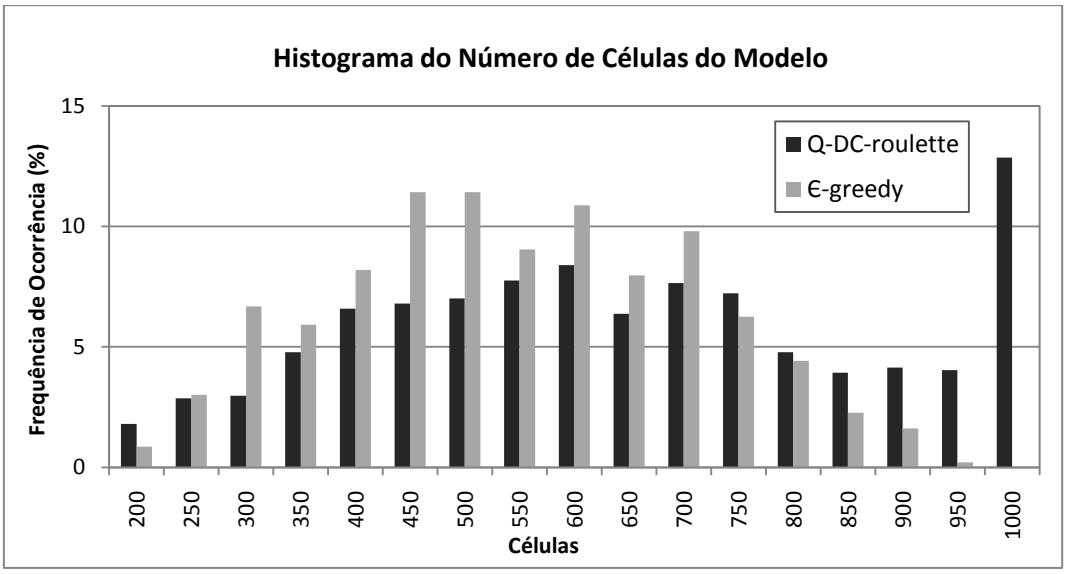

Gráfico 4.44: Histograma do número de células do modelo.

O gráfico 4.45 mostra o histograma do número máximo de células ativas por episódio para a política *Q-DC-roulette* e *ε-greedy*. Em todos os ciclos desde as condições iniciais até o objetivo de todos os testes realizados nos modelos treinados, no máximo cinco células do modelo RL-NFHP modificado participaram ao mesmo tempo e com frequência de ocorrência insignificante (0,11%). Sendo, na maioria dos casos, apenas duas a três células são responsáveis pela resposta do modelo. Este desempenho é similar ao apresentado no gráfico 4.13 para o problema do carro na montanha. Ele revela o ganho computacional alcançado devido à nova abordagem na programação, mesmo em problemas mais complexos como o do robô Khepera.

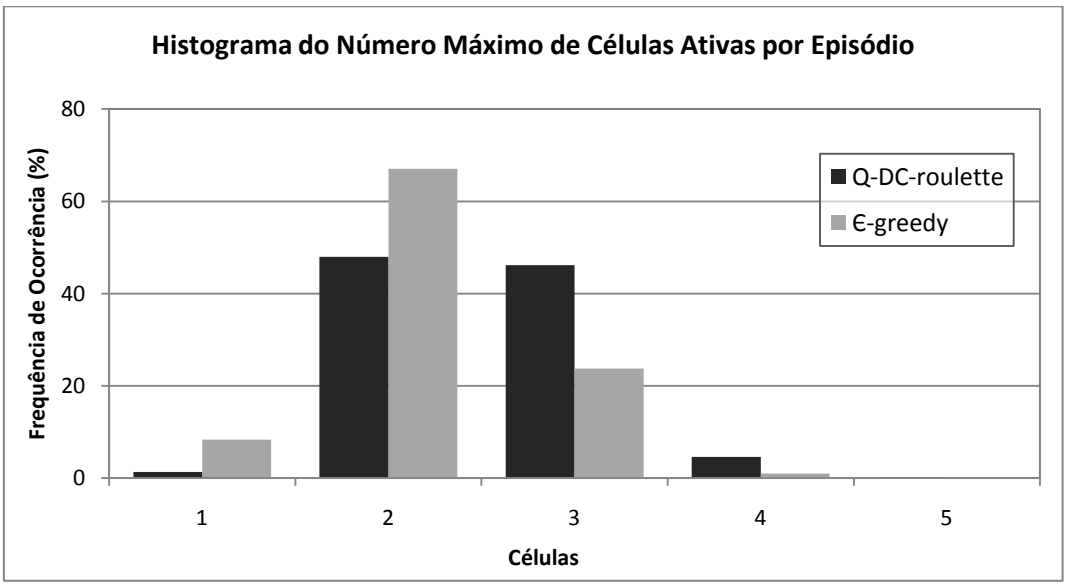

Gráfico 4.45: Histograma do número máximo de células ativas por episódio durante a fase de testes.

A tabela 4.8 expõe a comparação entre as políticas *Q-DC-roulette* e *ε-greedy* com relação ao número de épocas para treinamento e ao número de células com maior probabilidade de ocorrência.

Tabela 4.8: Comparação entre as políticas de seleção no modelo RL-NFHP.

Política	Nº de Épocas para Treinamento	Nº de Células
Q-DC-roulette	40	624
ε-greedy	290	520

A utilização da política *Q-DC-roulette* tem maior probabilidade de aprender com menor número de épocas e de produzir maiores estruturas RL-NFHP que a *ε-*

greedy. Vale ressaltar que foi observado empiricamente que as estruturas maiores geralmente estão relacionadas com maior número de épocas de aprendizado, diferentemente do que pode erradamente sugerir a tabela 4.8, ou seja, para a política *Q-DC-roulette*, os modelos com número de células de 624 provavelmente demoram mais do que 40 épocas para aprender.

4.2.2 Experimento 2: Poda da estrutura

Nos testes do modelo RL-NFHP modificado realizados no experimento 2, foi testada a poda da estrutura conforme procedimento exposto na seção 3.4.

Os demais parâmetros do modelo foram mantidos fixos e as condições iniciais foram as mesmas mostrados no experimento 1 (subseção 4.2.1) para o simulador Khepera sem obstáculo. A política de escolha das ações utilizada foi a *Q-DC-Roulette* por ter apresentado melhor desempenho na fase de treinamento no experimento 1.

O gráfico 4.45 mostra o histograma do número de épocas necessárias para treinamento usando a política *Q-DC-roulette* com poda da estrutura e sem este procedimento. Por este gráfico observa-se o modelo RL-NFHP modificado sem poda, na maioria das vezes é treinado em apenas 40 épocas e dificilmente demora mais de 400 épocas. Já o uso do método de poda atrasa o treinamento, que passa a ser realizado na maior parte das vezes em 80 épocas, além de ter maior variância, com desvio padrão de 186 épocas.

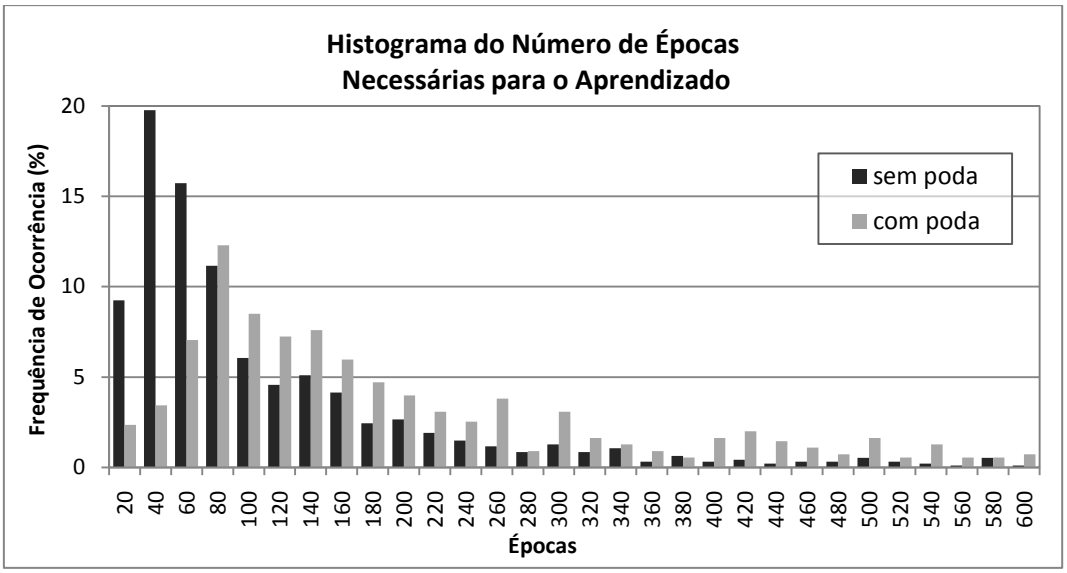

Gráfico 4.46: Histograma das épocas de treinamento usando *Q-DC-roulette* com e sem o método de poda.

O gráfico 4.46 expõe o histograma do número de células do modelo após treinamento bem sucedido usando a política *Q-DC-roulette* com poda da estrutura e sem este procedimento, respectivamente. Constata-se que o número de células do modelo treinado é indiscutivelmente inferior com a utilização do método de poda, em geral 50 células. A não utilização da poda cria um histograma de distribuição normal com média de 624 células e desvio padrão de 231 (gráfico 4.38). Verifica-se que o treinamento sem poda gera na maioria das vezes mais de 600 células, maior do que uma ordem de grandeza em relação à utilização da poda.

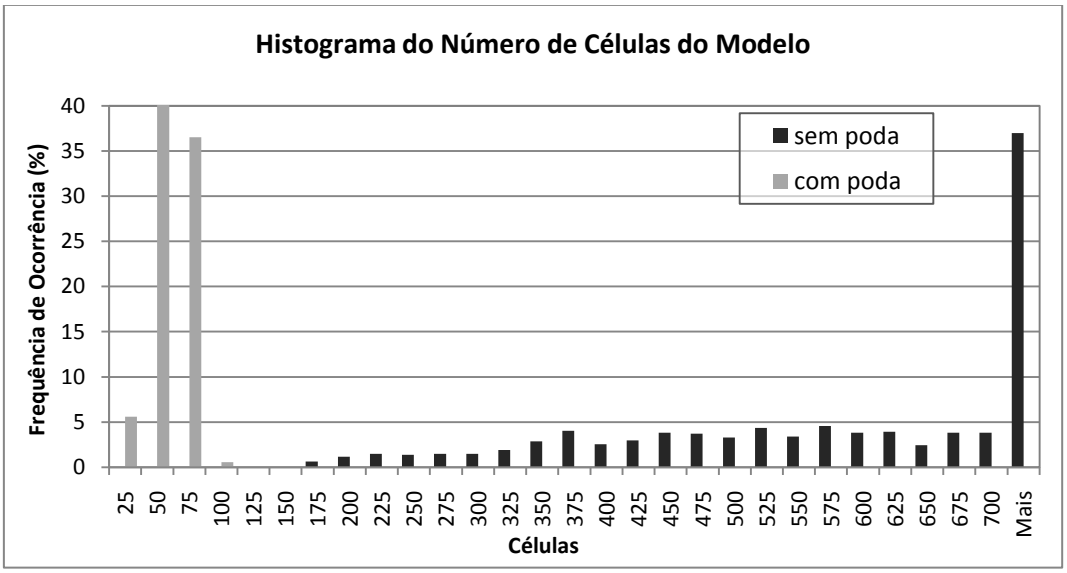

Gráfico 4.47: Histograma do número de células do modelo usando *Q-DC-roulette* com e sem o método de poda.

A tabela 4.9 expõe a comparação entre a utilização ou não do método de poda da estrutura com relação ao número de épocas para treinamento e ao número de células com maior probabilidade de ocorrência para a política *Q-DC-roulette*.

Tabela 4.9: Comparação entre o uso ou não do método de poda da estrutura para a política *Q-DC-roulette* no modelo RL-NFHP.

Método de poda	Nº de Épocas para Treinamento	Nº de Células
Sim	40	50
Não	80	624

Constata-se, como já esperado, que o número de células do modelo treinado é menor para o uso do método de poda. A poda das células recentemente criadas e não visitadas, gera uma dificuldade extra, além dos critérios de particionamento (subseção 2.4.7), para o crescimento da estrutura. Isto evita o refinamento demasiado do espaço de estados, o que encareceria os cálculos computacionais, como ocorre com frequência não desprezível para a não utilização do método de poda.

Um aspecto menos favorável ao modelo RL-NFHP modificado, assim como o original, é o fato de a estrutura do RL-NFHP poder alcançar tamanhos maiores do que o necessário devido às dificuldades de aprendizado inseridas pelo próprio RL somadas às do modelo RL-NFH. Isto acarreta em uma maior dificuldade do aprendizado, aumentando o tempo computacional e gerando regras mais especializadas do que o necessário, ou seja, uma estrutura com mais células.

4.2.3
Experimento 3: Com obstáculo

Foram executados diversos aprendizados do modelo RL-NFHP modificado no simulador Khepera com obstáculos (figura 4.9), mais complexo que o exposto na subseção anterior.

As variáveis foram definidas como na subseção 4.2.1. Utilizou-se a política de escolha de ações *Q-DC-roulette* que apresentou melhores resultados para a simulação sem obstáculo.

As condições iniciais foram alternadas entre (x_0, y_0, θ_0) = (-0,3;-0,3;0,52), (0,3;0,3;1,57), (-0,3;0,0;0,35), (-0,2;0,3;2,44), (-0,34;0,2;0,87), (-0,4;0,4;-0,785) e (0,0;0,3;3,14). A posição final desejada foi (x_f, y_f) = (0,3;-0,3).

As ações possíveis, velocidades angulares das rodas da esquerda e da direita, foram: (20;20), (-10;10), (10;-10), (-10;-10), correspondentes a andar para frente, rotacionar no sentido trigonométrico, no sentido anti-trigonométrico e andar para trás, respectivamente.

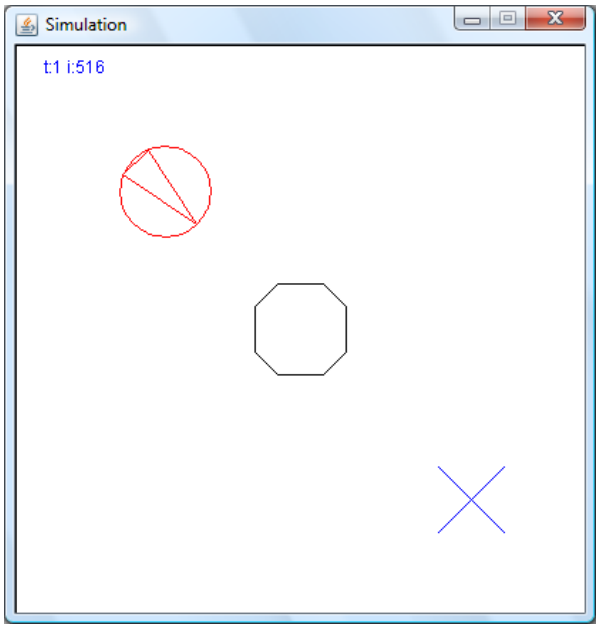

Figura 4.9: Ambiente de simulação do robô Khepera com obstáculo.

A função de reforço utilizada, dada pela eq. 4.7, foi bem mais complexa que a do caso sem obstáculo. Esta função está dividida em duas partes: uma próxima ao obstáculo, para valores médios e altos dos sensores ultrassônicos; e outra longe deste.

Na parte relativa à proximidade com o obstáculo, é interessante o reforço R não levar em consideração o ângulo α entre a frente do robô e o objetivo, pois o agente deve ser capaz de desviar deste obstáculo, e para isto deve alterar a sua direção. No geral, o reforço diminui à medida que a leitura dos sensores ultrassônicos aumenta remetendo a uma punição proporcional à proximidade com o obstáculo. Existe ainda uma parcela de recompensa/punição que induz à variação da direção, ou ângulo, conforme o robô vá de encontro ao obstáculo.

Na parte da equação relativa a regiões do ambiente sem obstáculo, o reforço leva em conta a distância à posição desejada d e o ângulo α. As diminuições desta distância e deste ângulo aumentam o valor de recompensa. Com o intuito de agilizar o deslocamento do robô, existe um incremento da recompensa caso este venha caminhando rumo à posição desejada, ou seja, diminuindo sua distância.

- Se $s_f > 0,4$ ou $s_l > 0,3$ ou $s_r > 0,3$

  $R = k_1 e^{-k_3(d)}$

  - Se $s_f > 0,4$

    $R = R(1 - 0,9s_f)$
  - Se $s_l > 0,3$

    $R = R(1 - 0,5s_l)$
  - Se $s_r > 0,3$

    $R = R(1 - 0,5s_r)$ (4.7)
  - Se $s_{f\,(t-1)} < s_{f\,(t)}$

    $R = k_5 R \mid \alpha_{(t-1)} - \alpha_{(t)} \mid$
- Caso contrário

  $R = k_1 e^{-k_3(d)} + k_2 e^{-k_4|\alpha|}$
  - Se $d_{(t-1)} > d_{(t)}$

    $R = 1,3R$

onde: o subscrito t refere-se ao passo presente, enquanto *t*-1 ao passo passado e k_1, k_2, k_3, k_4, k_5 são constantes maiores do que 1 utilizadas para adequar os valores de reforço à estrutura do modelo.

Um aspecto menos favorável do modelo RL-NFHP é o fato da função de reforço dever ser mais elaborada do que as funções requeridas por outros modelos NF baseados em RL, pois, sem esta informação, a solução dispenderia mais tempo para ser encontrada, uma vez que o sistema necessita, paralelamente ao aprendizado do comportamento do agente, aprender a identificar seus domínios.

O melhor modelo RL-NFHP modificado demorou 320 épocas para treinar e gerou uma estrutura de 283 células. Este modelo obtido apresentou um número médio de passos para cumprimento do objetivo na fase de testes de 257, partindo das 12 posições iniciais mostradas na figura 4.10 e chegando a posição final estipulada.

Como esperado, o modelo RL-NFHP modificado gera mais regras linguísticas, ou seja, maior número de células, quanto mais complexo é o problema. Este fato pode ser observado para o caso Khepera: no caso sem obstáculo foi construída uma estrutura menor – de 25 células (subseção 4.2.2) – do que no experimento com obstáculos – estrutura de 283 células –. Isto se deve à maior complexidade nas posições próximas ao obstáculo. Pelo mesmo motivo, o número de ciclos necessários para o aprendizado, no caso sem obstáculo, consequentemente também foi menor – 40 épocas em relação a 320, respectivamente.

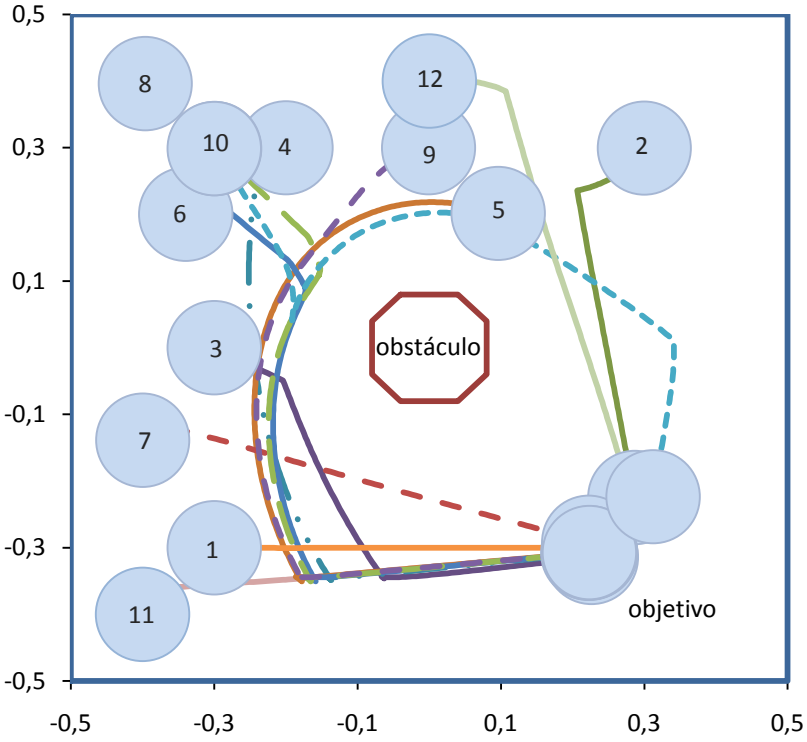

Figura 4.10: Caminho percorrido após aprendizado para posições e ângulos iniciais de 1 a 12.

A figura 4.10 mostra o caminho percorrido pelo robô nos testes realizados após o aprendizado para diferentes ângulos e posições iniciais (círculos numerados de 1 a 12) e chegando a posição final desejada com tolerância de metade do raio do robô.

As posições testadas foram: (x_0, y_0, θ_0) = (-0,3;-0,3;0,52), (0,3;0,3;1,57), (-0,3;0,0;0,35), (-0,2;0,3;2,44), (0,1;0,2;2,79), (-0,34;0,2;0,87), (-0,4;-0,14;1,40), (-0,4;0,4;-0,785), (0,0;0,3;3,14), (-0,3;0,3;2,09), (-0,4;-0,4;0,0) e (0,0;0,4;0,0), respectivamente aos percursos 1 a 12.

Constata-se, pela figura 4.10, que o robô cumpriu seu objetivo em todas as condições iniciais testadas. Ressalta-se que mesmo partindo de pontos que não foram explicitamente usados durante a fase de aprendizado, o robô consegue chegar à posição desejada. Estes testes também se estenderam a ângulos diferentes. Portanto, como esperado, o robô conseguiu aprender, criando uma base de conhecimento capaz de generalizar.

Observa-se que, para todas as posições iniciais testadas, este agente com comportamento inteligente, quando não existe obstáculo a sua frente, anda diretamente rumo à posição alvo, sem fazer curvas desnecessárias. Este comportamento foi observado em casos simples, como nas condições iniciais 1, 7 e 11, e mais complicados, como 6 e 8, onde o robô desvia do obstáculo central e depois aponta sua frente para a posição objetivo e segue um percurso praticamente em linha reta.

O agente realizou alguns caminhos desvantajosos, ou seja, com maior percurso. Para as condições iniciais 5 e 9, o robô preferiu contornar o obstáculo central no sentido trigonométrico, ao invés de anti-trigonométrico. Isto se deve aos ângulos iniciais atribuídos de 160º e 180º, que colocavam a frente do robô apontada naquela direção e induziam seu comportamento. Posições iniciais semelhantes, porém com ângulo de valores absolutos menores, como a 12, permitem que o robô tome a decisão correta.

As condições iniciais 4 e 10 tiveram por objetivo verificar o comportamento do robô nas condições mais adversas (o obstáculo no meio do caminho e frente do robô contra a parede). Mesmo assim, o robô apresentou ótimos resultados, girando sua frente, aproximando-se do obstáculo central, detectando este obstáculo e o contornando no sentido trigonométrico.

O gráfico 4.47 exibe as distâncias euclidianas normalizadas (d) e angulares normalizadas (α), enquanto o gráfico 4.48 mostra as distâncias e leituras dos sensores normalizados (s_f , s_l , s_r) em relação ao passo do episódio para a condição inicial (x_0, y_0, θ_0) = (-0,4;0,4;-0,785). Estes gráficos têm por objetivo exemplificar o comportamento típico do robô.

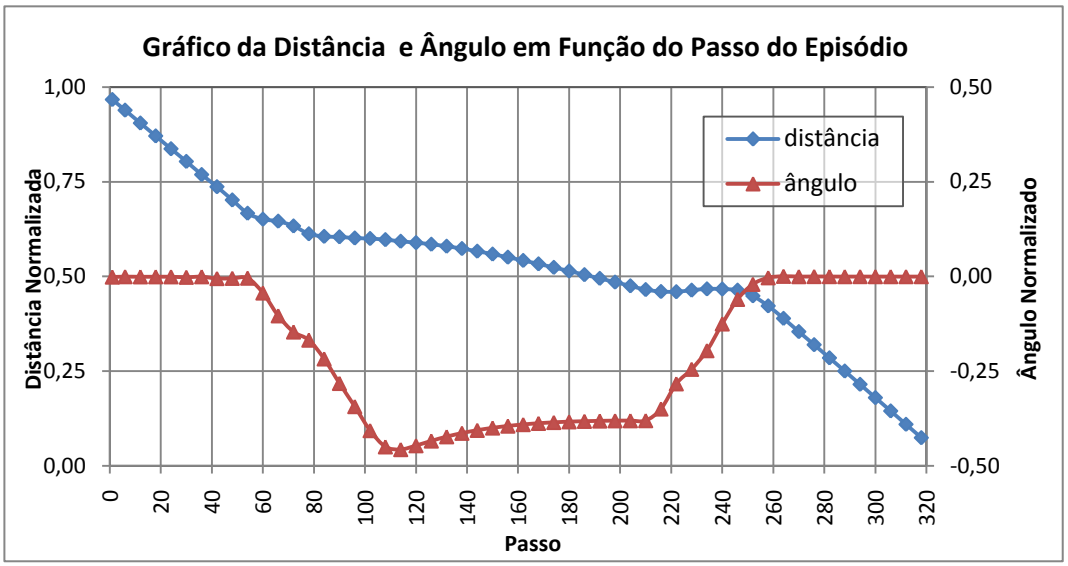

Gráfico 4.48: Distância e ângulo do robô durante um episódio para a condição inicial 8.

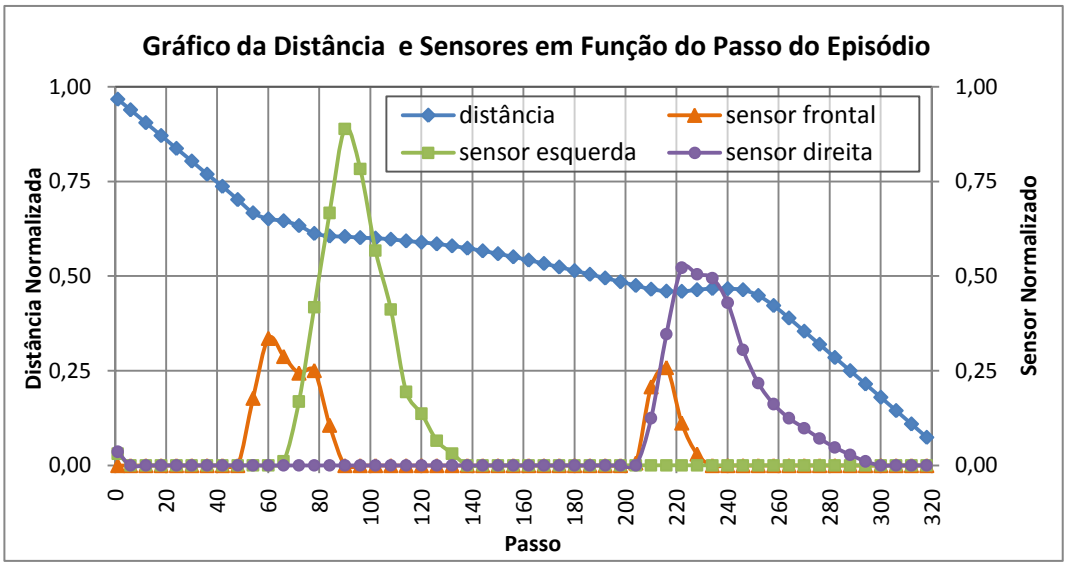

Gráfico 4.49: Distância e sensores do robô durante um episódio para a condição inicial 8.

Observa-se na figura 4.10 e nos gráficos 4.47 e 4.48 que o robô inicia na condição 8 de frente para o obstáculo e move-se em direção a este até o passo 50 aproximadamente. A partir daí, ele o percebe por meio do sensor frontal e passa a contorná-lo no sentido trigonométrico – diminuição do valor do sensor frontal e aumento da leitura do sensor da esquerda – até o passo 220. Em seguida, ele realiza uma rotação em torno de seu próprio eixo até o passo 250 – note que neste intervalo não existe variação na distância –. Por fim, o robô segue diretamente

para a posição desejada – repare que o sensor da direita percebe a parede inferior do ambiente durante este percurso.

Neste problema do robô Khepera, tem-se 5 entradas, formando um espaço de dimensão 5. A figura 4.11 objetiva expor o particionamento do espaço de estado mostrando relações entre duas entradas. Esta figura mostra o particionamento da distância (d) em relação ao sensor frontal (s_f), ao lateral direito (s_d) e ao ângulo (α), todos normalizados.

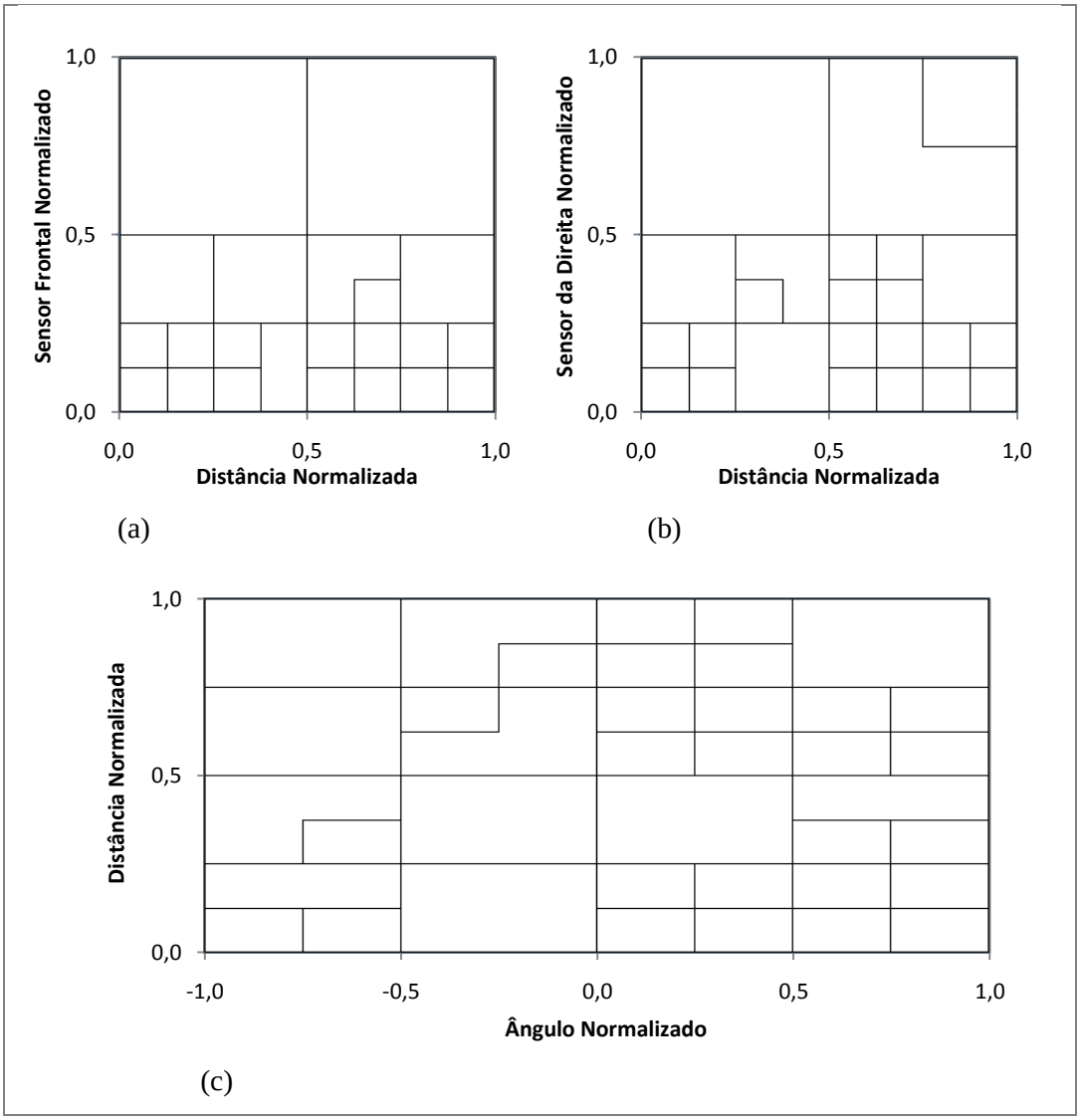

Figura 4.11: Particionamento do estado no modelo RL-NFHP modificado de 238 células.

Através das figuras 4.11.a e 4.11.b, verifica-se que os sensores frontal e da direita possuem maior refinamento para valores abaixo de 0,5 e para distâncias maiores que 0,5. É nesta região do espaço de estado que o robô se depara com o obstáculo durante o treinamento e deve tomar as mais difíceis decisões.

Ainda pelas figuras 4.11.a e 4.11.b, nota-se que sensor da direita apresenta maior particionamento que o frontal. Por possuir comportamento similar ao da direita, o sensor da esquerda não foi mostrado na figura 4.11. Talvez esta diferença dos sensores laterais com relação ao frontal seja devido ao fato de o robô ser capaz de desviar de obstáculos e de se locomover paralelamente a uma parede. Os sensores laterais devem identificar estas duas características distintas, enquanto o frontal apenas relaciona-se ao fato de desviar.

Pode-se observar pela figura 4.11.c que o ângulo possui maior refinamento para valores positivos. Este comportamento se deve provavelmente às condições iniciais apresentadas que estimularam uma maior visitação aos espaços de estado correspondentes a ângulos positivos. Esta maior frequência de visitação tende a induzir particionamentos.

Pode-se concluir que o modelo RL-NFHP modificado demonstrou ser capaz de aprender o comportamento desejado, mesmo quando submetido a um problema bem mais complexo que o *benchmark* carro na montanha (seção 4.1). O caso Khepera é um problema maior e com 5 variáveis de entradas contínuas, o que dificulta a capacidade de aprendizado, devido ao alto número de consequentes.

O modelo RL-NFHP modificado utilizando a nova política de seleção de ações *Q-DC-roulette*, o método de poda da estrutura e o método de *early stopping* para determinação do fim do treinamento acarretou em aceleração de aprendizado e obtenção de modelos menores em relação ao modelo original hierárquico neuro-fuzzy RL-NFHP.

O aprendizado foi obtido interagindo diretamente com o ambiente, sem que houvesse qualquer processamento *off-line* e sem usar qualquer outra informação que pudesse beneficiar o aprendizado.

Outro aspecto a destacar é o fato de que o RL-NFHP modificado dispensou a análise prévia das variáveis de entrada, não requerendo ordenação nem priorização destas entradas. Isto porque as células de sua estrutura foram elaboradas com a capacidade de suportar todas as entradas simultaneamente. Isto é bastante importante, pois, na maioria dos problemas baseados em sistemas neuro-fuzzy, é inviável avaliar o comportamento do agente através da utilização individual das entradas. Além disso, a análise prévia das entradas aumenta o tempo computacional para o aprendizado do agente.

4.3
Aplicação real em robótica (Lego *MindStorms* NXT)

Visando analisar a influência de incertezas e ruídos nas leituras dos sensores, variação entre os comandos enviados e executados pelos atuadores e possibilidade de treinamento simulado, foi realizado, neste trabalho, a implementação do modelo RL-NFHP modificado em um robô real. O melhor modelo criado a partir do simulador do robô Khepera (subseção 4.2.3) foi adaptado e testado em um ambiente real com um robô Lego *MindStorms* NXT. Este robô com comportamento inteligente foi capaz de desviar de obstáculos para chegar a uma posição final estipulada.

O ambiente real, mostrado na figura 4.12, é constituído por um tabuleiro de 2000x2000 mm limitado por paredes de 15 cm de altura (material: MDF), contendo um obstáculo central octagonal de 26 cm de diâmetro externo (material: papel panamá) e um robô Lego *MindStorms* NXT.

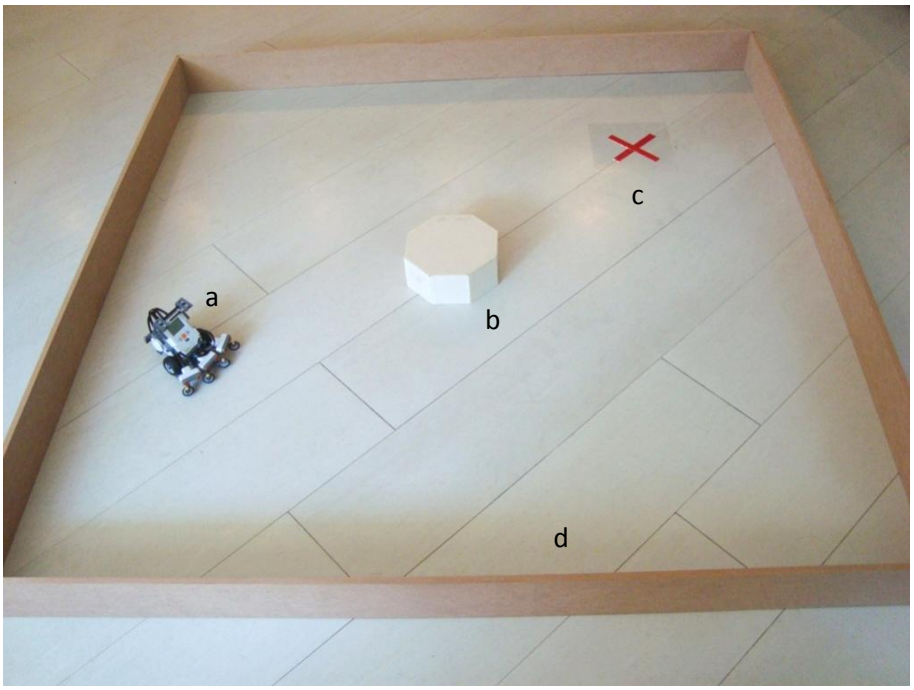

Figura 4.12: Cenário real mostrando: (a) o robô; (b) o obstáculo central; (c) a posição final desejada ou objetivo; (d) as paredes laterais.

O robô Lego *MindStorms* NXT (figura 4.13), utilizado neste trabalho, possui aproximadamente 25 cm de diâmetro (espaçamento entre rodas de 11 cm e

raio da roda de 2,5 cm) e está dotado com um *brick* NXT de comando, 3 sensores ultrassônicos (esquerda, frontal e direita) – para detecção de proximidade de uma parede ou um obstáculo – e dois servo-motores (esquerdo e direito). A estrutura básica de locomoção do robô foi baseada na construção sugerida pela Lego para o TriBot (figura 4.14): um triciclo dotado de dois servo motores ligados cada um a uma roda para deslocamento e direcionamento do robô, e uma terceira roda de apoio em eixo livre. Outras pequenas rodas auxiliares, posicionadas em eixo livre a frente dos sensores ultrassônicos, evitam leituras incorretas – decorrentes da aproximação excessiva destes com obstáculos – e facilitam o deslizar da lateral do robô contra o obstáculo.

Figura 4.13: Robô Lego *MindStorms* NXT montado para avaliação prática do modelo RL-NFHP, constituído por: (a) *brick* NXT; (b) 3 sensores ultrassônicos; (c) 2 servo-motores.

O *brick* NXT de comando foi lançado pela Lego em 2006 nos kits Lego *MindStorms* NXT. Ele possui dois microprocessadores: um principal de 32 bits (AT91SAM7S256), com 48MHz e 256k de memória flash e 64k de memória RAM; e outro secundário de 8 bits (ATmega48), 4MHz e 4k de memória flash e 512b de RAM, um adaptador Bluetooth interno de 26MHz e uma entrada USB 1.1 (12Mbit/s). Ambos permitem conexão sem fio ao computador para transmissão de

programas ou controle remoto do dispositivo. A programação aceita diversas linguagens e a empresa Lego disponibiliza o firmware em código aberto.

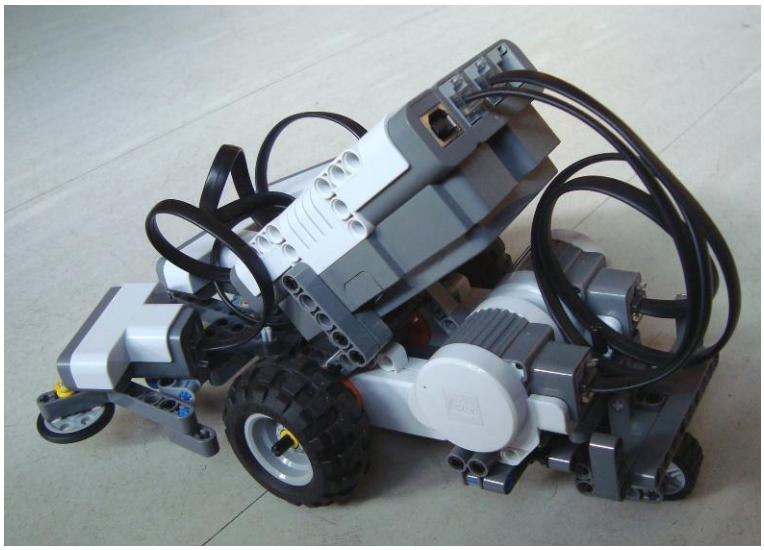

Figura 4.14: Detalhe da estrutura do TriBot do robô.

O sensor ultrassônico compõe a visão do robô, a qual funciona com o mesmo princípio científico que permite aos morcegos se localizar. O sensor percebe obstáculos calculando o tempo de reforço de uma onda sonora refletida pelo objeto que esteja a uma distância menor que 255 cm. Entretanto na prática esta distância não deve ser superior a 80 cm devido a flutuações na leitura do sensor. A incerteza nesta leitura é de ±3 cm.

Os servo-motores funcionam como entradas e saídas de dados. Eles atuam nas rodas da esquerda e direita – produzindo o movimento do robô – com potências entre -100 e 100. Cada servo-motor possui um encoder que fornece a rotação do eixo do motor em graus, com incerteza de ±1º.

A localização por coordenadas (x, y, θ) do centro do robô é estimada por meio da eq. 4.5, para as velocidades angulares ω_l e ω_r dadas pela eq. 4.8 e fornecida a posição inicial (x_0, y_0, θ_0). Ressalta-se que esta forma de cálculo da localização acumula erros, como será mostrado adiante, e não é válida quando eventualmente ocorre deslizamento das rodas sobre o piso.

$$\omega_l = \frac{\Delta\theta_l}{\Delta t} \quad \text{e} \quad \omega_r = \frac{\Delta\theta_r}{\Delta t} \tag{4.8}$$

onde: $\Delta\theta_l$ é a variação angular da esquerda lida pelo encoder, $\Delta\theta_r$ a variação angular da direita e Δt o passo de tempo entre cada comando fornecido para os atuadores (neste trabalho utilizou-se Δt = 20 ms).

Através desta localização do robô e das coordenadas da posição final desejada (x_f,y_f) = (-0,55;-0,55) pode-se calcular a distância euclidiana d (eq. 4.3) e o ângulo (em radianos) entre o robô e a posição objetivo final do robô α (eq. 4.4). Com isto, obtém-se as 5 variáveis de entrada normalizadas do controlador do robô Lego *MindStorms* NXT, análogas ao do simulador Khepera:

- distância euclidiana entre o robô e o alvo d ;
- ângulo entre o robô e o objetivo α ;
- sensor frontal s_f ;
- sensor da esquerda s_l ;
- sensor da direita s_r .

A saída do controlador são as potências aplicadas aos atuadores, neste caso os setrvo-motores das rodas esquerda e direita, tendo seus valores limitados entre –60 (velocidade angular máxima para trás) e 60 (velocidade angular máxima para frente).

O código do controlador responsável pelo comportamento inteligente do agente não foi embarcado. Ele foi mantido no computador e suas instruções transferidas para o robô por meio de cabo USB. A Lego possibilita a comunicação através de *bluethooth*, porém na ocasião este dispositivo não estava à disposição. Esta medida foi necessária uma vez que a simplicidade do bloco de processamento NXT do robô Lego MindStorms NXT impedia o embarque da inteligência. Com isto foram executados concomitantemente dois programas (seção 3.5): o programa principal (Main.java) e um programa de interface computador-robô (Interface5Entries.vi). Este programa de interface foi escrito em LabVIEW® 8.2 devido à facilidade desta linguagem em se comunicar com equipamentos externos ao computador. O programa de interface foi responsável pela leitura do ambiente por meio dos sensores do robô, adaptação destas leituras transformando-as em entradas para o controlador e execução dos comandos do controlador nos atuadores do robô. A troca de informações entre estes dois programas foi realizada através de arquivos de leitura e escrita dos valores dos sensores (arquivo_sensores.txt) e dos comandos do controlador

(arquivo_comandos.txt). A permissão para a leitura e escrita destes arquivos foi dada por um terceiro arquivo (flag.txt) que funcionou como um semáforo, mostrando a vez de cada programa.

Vale ressaltar que o programa de interface precisou adaptar ajustando a escala do modelo RL-NFHP modificado treinado no simulador (subseção 4.2.3) para controle do robô Lego MindStorms NXT, transformando entradas e aplicando ganhos. Para adequar os valores de potência para as rodas comandados pelo controlador RL-NFHP treinado no simulador que estão entre -20 a 20, foi introduzido um ganho de 3. Isto por que o trabalho aqui apresentado foi realizado em etapas, o simulador construído não foi desenvolvido visando o robô real. Logo, passou a existir uma diferença razoável entre o simulador implementado e o robô.

O melhor modelo RL-NFHP modificado de 283 células obtido na simulação foi utilizado diretamente no robô real. O robô apresentou um número médio de passos para cumprimento do objetivo na fase de testes de 81, partindo das 12 posições iniciais mostradas na figura 4.15 e chegando a posição final estipulada. Todo o teste foi documentado em vídeo (CD em anexo).

O menor número médio de passos para o cumprimento do objetivo por parte do robô Lego *MindStorms* NXT deve-se, principalmente, a diferença entre as dimensões do robô real e do simulado (mesmo levando em consideração os fatores de escala) e a rotação das rodas. Uma mesma potência comandada pelo controlador pode gerar rotações diferentes devido a efeitos de desbalanceamento. Este desbalanceamento é intrínseco ao robô e também é produzido pelo peso do cabo USB de transmissão de comandos, o qual foi observado diversas vezes durante os testes.

A figura 4.15 mostra o caminho percorrido pelo robô real nos testes para diferentes ângulos e posições iniciais (círculos numerados de 1 a 12) e chegando a posição final desejada com tolerância igual ao raio do robô. Nesta figura, o percurso traçado foi estimado através da localização do robô, obtida pela eq. 4.5, ao longo dos episódios de teste. O teste foi considerado bem sucedido quando a distância do robô ao objetivo, calculada por meio da eq. 4.3 ou visualmente, era inferior ao seu raio. Foram testadas algumas posições equivalentes as da simulação (subseção 4.2.3) e outras foram acrescentadas.

As posições testadas foram: (x_0, y_0, θ_0) = (0,6;0,0;0,0), (-0,6;0,6;1,57), (0,6;0,0;0,35), (0,4;0,6;2,44), (-0,2;0,4;2,79), (0,7;0,4;0,87), (0,8;-0,3;1,40), (0,6;0,6;-0,785), (0,0;0,6;3,14), (-0,3;0,3;2,09), (0,6;-0,6;0,0) e (0,0;0,8;0,0), respectivamente aos percursos 1 a 12. Observa-se que os eixos de coordenadas de localização foram dispostos na ordem inversa para se adequarem a figura 4.12 e a ordenada da posição desejada é negativa, diferentemente da simulação do Khepera (subseção 4.2.3).

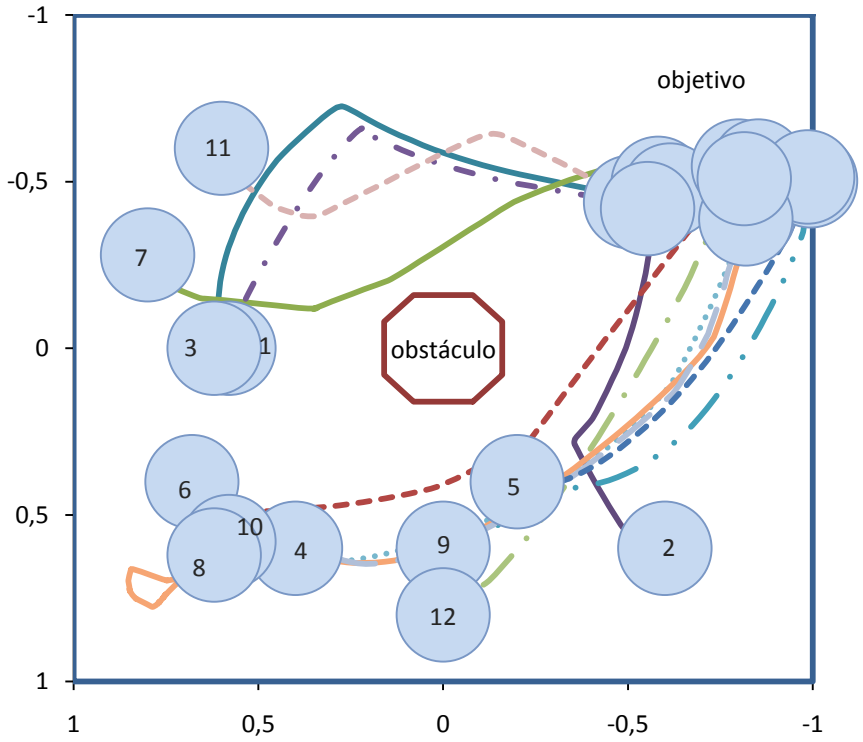

Figura 4.15: Caminho percorrido pelo robô real
para posições e ângulos iniciais de 1 a 12.

Constata-se, pela figura 4.15, que o robô cumpriu seu objetivo em todas as condições iniciais testadas. Ressalta-se que mesmo utilizando o modelo treinado em ambiente simulado com as dificuldades expostas na seção 4.1 e partindo de pontos que não foram explicitamente usados durante a fase de aprendizado, o robô consegue chegar à posição desejada. Estes testes também se estenderam a ângulos diferentes. Portanto, como almejado, o robô conseguiu aprender na simulação, criando uma base de conhecimento, e foi capaz de generalizar para o problema real.

Nota-se que, diferentemente do que ocorre para o teste simulado (figura 4.10), o agente não realiza trajetórias retas, destaque para o caminho

sinuoso da condição inicial 11. Este fato provavelmente deve-se ao desbalanceamento do robô, como referido anteriormente.

Por outro lado, alguns caminhos da figura 4.15 foram similares aos da figura 4.10 como os para as condições iniciais 2, 3 e 12, revelando coerência de comportamento.

A condição inicial 10 teve por objetivo verificar o comportamento do robô nas condições mais adversas (o obstáculo no meio do caminho e frente do robô contra a parede). Mesmo assim, o robô apresentou ótimos resultados, girando sua frente, aproximando-se do obstáculo central, detectando este obstáculo e o contornando no sentido trigonométrico.

A figura 4.16 mostra caminhos não ideais percorridos pelo robô real nos testes para diferentes ângulos e posições iniciais (círculos numerados de 1 a 5). As posições testadas foram: (x_0, y_0, θ_0) = (0,7;-0,8;0,0), (0,0;-0,7;0,0), (0,7;0,0;-3,14), (0,4;0,2;1,57) e (0,4;0,4;-2,36), respectivamente aos percursos 1 a 5.

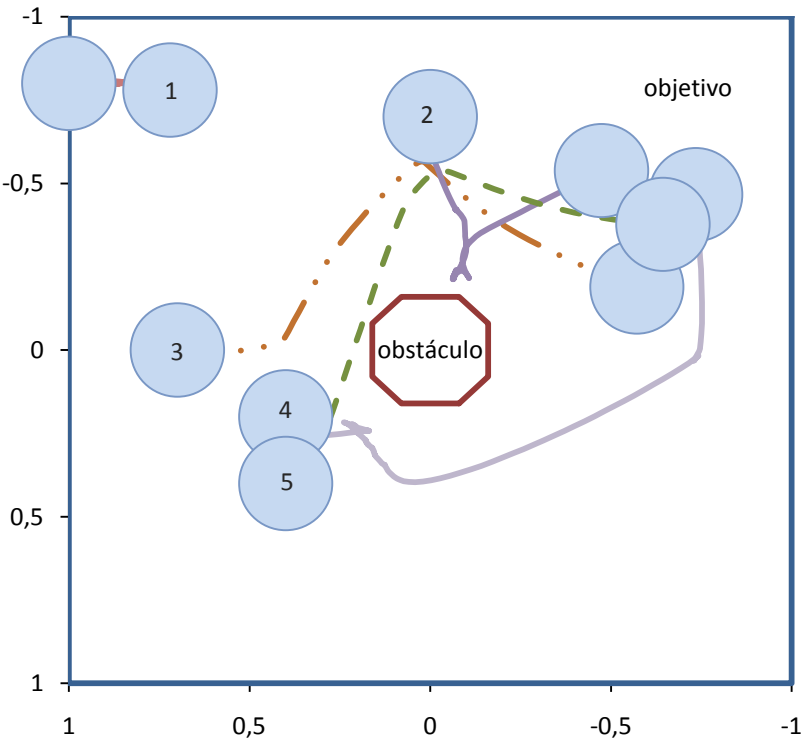

Figura 4.16: Caminho percorrido pelo robô real
para posições e ângulos iniciais de 1 a 5.

Na condição inicial 1, o robô começa voltado para a parede, se aproxima mais e, tentando rotacionar, colide insistentemente contra a parede, não

conseguindo completar seu objetivo. Este comportamento provavelmente se deve a estrutura do robô real ser diferente a do simulado. No simulado, o diâmetro do robô é igual ao espaçamento entre as rodas, enquanto no Lego *MindStorms* NXT o robô não é circular e o diâmetro do círculo que o circunscreve é bem maior. Este fato pode ocasionar em colisões com obstáculos ao rotacionar sobre o próprio eixo, colisões inexistentes na simulação. Além disto, a roda de apoio em eixo livre na estrutura TriBot (figura 4.14), por diversas vezes, atrapalhou movimentos para trás.

As posições iniciais 2 e 4, mostram comportamentos indevidos com partes do caminho rumo a posição desejada refeitas. Provavelmente trata-se de pares estado-ação pouco explorados durante o processo de treinamento no simulador ou ruído nos sensores ultrassônicos. Mesmo com estes pequenos percalços, o robô cumpriu sua tarefa.

As posições iniciais 1, 3 e 5 expõe o que já foi dito anteriormente: a incerteza na estimativa da posição do robô. Na posição 1, devido às colisões com a parede, ocorre o deslizamento das rodas em relação ao piso. Nestas condições, a eq. 4.5 não é mais válida, e a estimativa da localização mostra erradamente o robô dentro da parede no término do percurso 1. Na posição 3, pelos dados coletados e plotados na figura 4.16, o robô aparentemente não chega a posição desejada, porém o vídeo deste caso revela o contrário. Já para a 5, parece que o robô colide com o obstáculo central, o que não ocorre.

Pode-se concluir que o modelo RL-NFHP modificado demonstrou ser capaz de aprender o comportamento desejado no simulador e, com algumas adaptações, ser utilizado para equipar um agente real dotando-o de comportamento inteligente. O uso de simuladores realistas evita o dispêndio de tempo de treinamento. Modelos treinados somente no simulador podem gerar resultados satisfatórios quando adaptados aos problemas reais. Além disso, a normalização das entradas permitiu que o modelo respondesse adequadamente a mudanças nos limites das variáveis de entrada.

Vale ressaltar que o experimento conduzido com o robô Lego *MindStorms* NXT gerou desempenho acima do esperado, uma vez que o aprendizado do par estado-ação foi feito apenas no simulador e o modelo já pronto utilizado em um robô e ambiente ligeiramente diferentes.