Cristina Hohan Yu Chang

Aprendizado por Reforço Relacional com o algoritmo TG

Projeto de Formatura apresentado à disciplina PCS2050 - Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo

Área de Concentração:
Engenharia de Computação

Orientador:
Profa. Dra. Anna Helena Reali Costa

São Paulo

2010

Resumo

Inteligência Artificial (IA) é uma área de pesquisa que visa entender e criar métodos que simulem a inteligência humana e sua capacidade de resolver problemas. Entre os vários campos da IA, o aprendizado de máquina é aquele que se preocupa com a questão de como construir programas de computador que melhoram automaticamente com a experiência. Este trabalho apresenta os conceitos relacionados ao Aprendizado por Reforço Relacional (ARR), uma técnica de aprendizagem que combina o Aprendizado por Reforço (AR) padrão com a Programação em Lógica Indutiva (PLI) para permitir que o sistema de aprendizagem explore o conhecimento estrutural existente sobre o domínio de aplicação. AR interessa-se por como um agente deve executar ações em um ambiente de modo a maximizar alguma noção de recompensa a longo prazo; PLI, por sua vez, fornece um sistema de linguagem lógica com o qual o conhecimento adquirido pode ser armazenado e utilizado para ações futuras. O foco principal deste trabalho consiste no algoritmo TG, um algoritmo de aprendizado totalmente incremental de árvores de decisão de primeira ordem, e a integração deste algoritmo em um sistema ARR. São descritos o algoritmo TG e o sistema ARR-TG. Investigações são conduzidas na tarefa de planejamento no Mundo dos Blocos. Neste domínio simples é demonstrado que o ARR-TG permite empregar representações estruturais, abstrair a partir de objetivos específicos e explorar os resultados de fases anteriores de aprendizagem ao abordar novas situações.

ok

Abstract

Artificial Intelligence is an area of research that seeks to understand and create methods simulating human intelligence and ability to solve problems. Among the various fields of AI,Machine Learning is that one concerned with the question of how to construct computer programs that automatically improve with experience. This work presents the concepts related to Relational Reinforcement Learning (RRL), a learning technique that combines standard Reinforcement Learning (RL) with Inductive Logic Programming (ILP) to enable the learning system to exploit structural knowledge about the application domain. RL is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward; ILP, in turn, provides a logical language system with which the acquired knowledge is stored and used for future actions. The main focus of this project relies on the TG algorithm, a fully incremental first order decision tree learning algorithm, and the integration of this algorithm in the RRL system. We describe the TG algorithm and the RRL-TG system. We investigate the task of planning in the blocks world. Within this simple domain we demonstrate that RRL-TG allows us to employ structural representations, to abstract from specific goals pursued and to exploit the results of previous learning phases when addressing new situations.

Lista de Figuras

Lista de Abreviaturas e Siglas

AM	Aprendizagem de Máquina
FOLDT/ADLPO	*First Order Logical Decision Tree* (Árvore de Decisão Lógica de Primeira Ordem)
IA	Inteligência Artificial
ILP	*Inductive Logic Programming* (Programação Lógica Indutiva)
Java EE	*Java Enterprise Edition*
Java SE	*Java Standard Edition*
LPO	Lógica de Primeira Ordem
MSS	*Minimal Sample Size* (Tamanho Mínimo de Exemplos)
RL/AR	*Reinforcement Learning* (Aprendizado por Reforço)
RRL/ARR	*Relational Reinforcement Learning* (Aprendizado por Reforço Relacional)
TDIDT	*Top-Down Induction of Decision Tree*

Lista de Algoritmos

Sumário

1 *Introdução*

O conceito de Inteligência Artificial desde sempre populou o imaginário da humanidade. As primeiras referências ao conceito de máquinas inteligentes podem ser traçadas desde a mitologia grega, onde o deus Hefesto (ou Vulcano, na mitologia romana) morava em um palácio servido por robôs mecânicos (WIKIPéDIA, a). A área desperta muitas questões éticas e já deu origem a diversas ficções abordando essas discussões e tecendo as mais diversas possibilidades do futuro, tais como os filmes A.I.: INTELIGÊNCIA ARTIFICIAL (2001), EU, ROBÔ (livro: 1950, filme: 2004), e MATRIX (1999), entre muitos outros.

O termo IA em si foi cunhado apenas em 1956 por John McCarthy, Marvin Minsky e Claude Shannon, na primeira conferência dedicada ao assunto (WIKIPéDIA, b). Mas o que vem a ser IA? É toda entidade artificial que, de alguma forma consegue reproduzir a capacidade ou o processo de raciocínio dos seres humanos de solucionar problemas. Segundo Russel e Norvig (2003), para determinar se a máquina é inteligente ou não, ela deverá possui as seguintes capacidades (RUSSELL; NORVIG, 2003):

- **Processamento de linguagem natural** para habilitá-la a comunicar-se com sucesso.
- **Representação de conhecimento** para armazenar o conhecimento que sabe ou acredita saber.
- **Raciocínio automático** para fazer uso da informação armazenada a fim de responder questões ou chegar a novas conclusões.
- **Aprendizagem** para adaptar-se às novas circunstâncias e detectar e explorar padrões.

Atualmente a área de IA encontra-se fragmentada em diversos sub-campos, cada um abordando o assunto sob perspectivas diferentes e desenvolvendo métodos próprios, experimentando e aprimorando-os, fundindo-os ou incorporando-os com outros métodos para complementar falhas, melhorar desempenho, descobrir novos caminhos. Este trabalho foca-se na sub-área de Aprendizagem de Máquina (AM), mais especificamente, no

conceito de Aprendizado por Reforço (AR) e uma de suas variantes, o Aprendizado por Reforço Relacional (ARR) com o algoritmo TG (DRIESSENS; RAMON; BLOCKEEL, 2001).

De modo geral, a área de AM se dedica ao desenvolvimento de algoritmos e técnicas que permitam máquinas e programas a aprender e a melhorar seu desempenho na realização de tarefas, por meio da experiência. Dependendo da metodologia empregada, esses algoritmos são divididos em diversos paradigmas (SILVA, 2009).

Um desses paradigmas de aprendizado é o AR, como citado acima, que resulta de uma combinação de uma série de áreas, dentre elas: estatística, métodos computacionais, psicologia, teoria de controle ótimo. Sua idéia básica é recompensar uma ação do agente quando esta ação resulta em um estado desejável, ou aplicar uma penalidade, caso a ação leve o agente a um estado indesejável. O objetivo do agente é aprender qual sequência de ações resultaria na maior recompensa acumulada ao fim da tarefa.

A partir deste conceito, surgiu uma nova linha de pensamento, o ARR, que integra os conceitos de AR com os conceitos de *Inductive Logical Programming* (ILP), outro paradigma de AM. O princípio básico do ILP é gerar predicados descritivos dos estados a partir de exemplos e do conhecimento prévio do ambiente, preocupando-se mais no aprendizado de conceitos e ignorando o resto de AM.

O ARR combina técnicas de AR, especificamente o Q-learning (SUTTON; BARTO, 1998), com as técnicas de generalização do ILP, fornecendo uma linguagem de representação mais expressiva para representar estados, ações e funções Q. O ARR usa um algoritmo de regressão relacional que possui como entradas uma descrição de um estado, um objetivo e uma ação, devolvendo o valor Q correspondente como resultado e aprendendo, dessa forma, a função Q (DZEROSKI; RAEDT; BLOCKEEL, 1998).

O domínio de aplicação onde o funcionamento do sistema ARR é observado é o Mundo dos Blocos. No Mundo dos Blocos, os estados são representações de todas as combinações possíveis das posições dos blocos (um bloco sobre outro, ou sobre a chão), sendo eles conhecidos. As ações disponíveis são mover um bloco para outra posição, levando em conta se ele está desimpedido ou não (se é possível movê-lo ou não). Considere um mundo de três blocos (A,B e C), os estados possíveis e suas respectivas ações são: (1) todos os blocos sobre o chão - move(A,B), move(A,C),move(B,A), move(B,C), move(C,A), move(C,B); (2) bloco A sobre o bloco B, que está sobre o chão, bloco C sobre o chão - move(A,C), move(C,A), move(A,Chão); (3) bloco A sobre o bloco C, que está sobre o chão, bloco B sobre o chão - move(A,B), move(B,A), move(A,Chão); (4) bloco B sobre o bloco A,

que está sobre o chão, bloco C sobre o chão - move(B,C), move(C,B), move(B,Chão); e assim por diante. O objetivo do agente dentro deste domínio de aplicação é colocar um determinado bloco sobre outro bloco, especificados a priori.

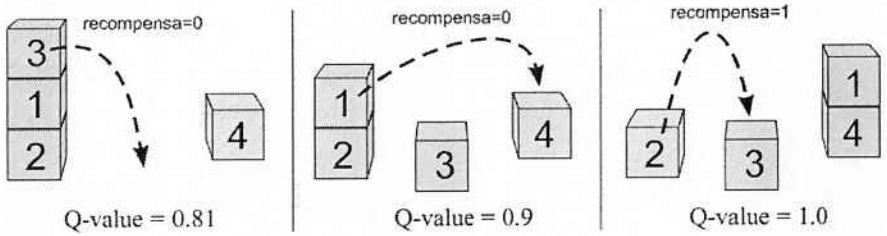

Figura 1.1: Exemplo de um episódio de aprendizado no Mundo dos Blocos, com quatro blocos e tendo on(2,3), indicando o bloco 2 sobre o bloco 3, como objetivo, com $\gamma = 0,9$.

1.1 Motivação

O paradigma de AR apresenta algumas limitações que muitas vezes o torna impraticável aos problemas do mundo real. Os algoritmos levam muito tempo para se chegar a uma solução ótima, além de não permitirem o aproveitamento do conhecimento adquirido para a execução de tarefas similares, que devem ser solucionadas do zero (LAZARIC, 2008). Outra desvantagem do AR é o uso de tabelas (Estados x Ações) que armazenam os respectivos valores Q, resultando em problemas de escalabilidade para espaços de estados muito grandes. O uso das técnicas de ILP permite generalizar os estados e objetivos específicos usando variáveis e descrevendo-os relacionalmente (DZEROSKI; RAEDT; BLOCKEEL, 1998).

Da incorporação dos métodos de ILP aos métodos de AR surge o ARR, cujo propósito provem de uma tentativa de superar as limitações que o AR apresenta. A primeira versão do paradigma de ARR teve como algoritmo base o TILDE (BLOCKEEL; RAEDT, 1998), um sistema que gera uma árvore de decisão de lógica de primeira ordem. Mais tarde esse algoritmo foi combinado com o algoritmo incremental G (CHAPMAN; KAELBLING, 1991), de onde surge o algoritmo TG (DRIESSENS; RAMON; BLOCKEEL, 2001), cujo objetivo é agilizar o processamento do ARR.

Neste projeto é feita uma pesquisa de campo sobre o assunto, o estudo, a implementação e testes do algoritmo TG e uma análise dos resultados obtidos, visando observar as vantagens que o algoritmo TG traz para o paradigma de ARR.

1.2 Objetivo

Este projeto objetiva descrever os conceitos relacionados ao paradigma de ARR e ao algoritmo TG, desde os motivos que levaram à sua concepção até a sua evolução ao estágio atual, implementar o sistema ARR-TG e aplicá-lo ao Mundo dos Blocos, observar seu funcionamento e desempenho, além de analisar os resultados obtidos com os testes. O projeto é composto pelas seguintes etapas:

- Estudo e compreensão dos conceitos relacionados;
- Implementação do algoritmo TG;
- Testes no Mundo dos Blocos, visando repetir e analisar os resultados do trabalho pioneiro (DRIESSENS, 2004; DRIESSENS; RAMON; BLOCKEEL, 2001) e entender melhor o sistema ARR-TG;

1.3 Organização do documento

O documento está organizado da seguinte forma: no Capítulo 2 são apresentados os conceitos fundamentais relacionados ao tema proposto, tendo em vista o Mundo dos Blocos. No Capítulo 3 é documentado o desenvolvimento do projeto, seus requisitos e o ambiente de desenvolvimento, com a definição do domínio onde o algoritmo TG será aplicado. No Capítulo 4 são descritos os testes aplicados ao algoritmo implementado e os resultados obtidos, incluindo também uma análise dos resultados obtidos. Finalmente, no Capítulo 5 são apresentadas as principais conclusões e sugestões de trabalhos futuros.

2 *Conceitos Fundamentais*

Neste capítulo são descritos os conceitos relacionados a este projeto, afim de fornecer uma base teórica necessária para compreender o objetivo deste projeto e os resultados obtidos.

2.1 Aprendizado por Reforço

A idéia básica que rege o conceito de Aprendizado por Reforço tem como referência o entendimento humano do processo de aprendizado através da interação com o meio ambiente, por meio da percepção do ambiente, da tentativa, do experimento e da apreensão das consequências dessas ações. O exercício dessa relação de tentativa e efeito mostra a um aprendiz quais ações o levariam a atingir determinados objetivos de modo mais efetivo. Sem dúvida essas interações são a maior fonte de conhecimento sobre o meio ambiente para um ser vivo (SUTTON; BARTO, 1998).

O AR é uma abordagem computacional desse processo de aprendizagem, e seus algoritmos estão altamente relacionados com as técnicas de Programação Dinâmica, um método que soluciona problemas complexos dividindo-os em passos mais simples. Um agente é posicionado em um mundo sobre o qual não possui nenhuma informação prévia, e é encarregado de atingir uma determinada meta. Ao agente não são informadas quais ações devem ser tomadas, como é feito na maior parte das abordagens de AM, mas cabe ao agente explorar o ambiente com seus sensores e descobrir quais ações, ou sequência de ações, concedem maior recompensa, por meio da tentativa e erro. A Figura 2.1 mostra a interação do agente com o meio ambiente. A cada passo t da interação, o agente recebe uma entrada que descreve o estado atual s do ambiente, e então escolhe uma ação. A ação modifica o estado de acordo com a dinâmica do ambiente, e a qualidade da transição para este estado é comunicada ao agente por meio de um reforço r (LAZARIC, 2008).

Além do agente e do ambiente, existem mais quatro outros elementos importantes que definem um sistema AR:

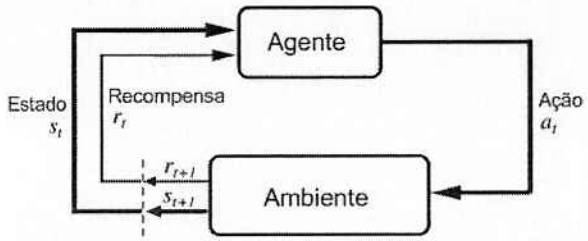

Figura 2.1: Modelo padrão de Aprendizado por Reforço (SILVA, 2009).

- uma política π: função que mapeia estados para ações e define o comportamento do agente;
- uma função de avaliação V: para cada estado, enquanto a função recompensa indica a melhor ação naquele estado, a função V especifica qual o melhor curso de ações ao longo do problema $V : S \rightarrow \Re$;
- um modelo do ambiente, definido por:
  - um conjunto de estados S
  - um conjunto de ações A
  - uma função de transição $\delta(s,a)$: $S \times A \rightarrow S$
  - uma função de recompensa r:$S \times A \rightarrow \Re$

A cada ponto no tempo, o ambiente pode estar em um dos estados s_t de S observado pelo agente, que seleciona uma ação $a_t = \pi(s_t) \in A$ a ser executada de acordo com a política π. O ambiente passa para um novo estado $s_{t+1} = \delta(s_t, a_t)$ observado pelo agente, que também recebe uma recompensa $r_t = r(s_t, a_t)$ como resultado da ação escolhida e executada. O objetivo do agente é encontrar uma política de atuação π^*:$S \rightarrow A$ que maximiza o valor da função $V^{\pi}(s)$ para todo estado $s \in S$.

2.1.1 Algoritmo Q-Learning

Uma das abordagens mais comuns ao AR é conhecida como Q-Learning, um algoritmo iterativo desenvolvido por Watkins em 1989 (WATKINS; DAYAN, 1992) para aprender a política ótima π^* sem necessidade de um modelo do ambiente (por exemplo, a função transição δ). Esse algoritmo possui uma função valor-ação que atribui um valor Q para cada par estado-ação permitindo uma aproximação à política ótima π^*, onde $r(s,a)$ é a

recompensa (ou reforço) recebida ao realizar a ação a no estado s, γ é o fator de desconto ($0 \leq \gamma < 1$) e $V^{\pi^*}(\delta(s,a))$ é o valor do próximo estado atingido.

$$\pi^*(s) = arg\max_a(Q^{\pi^*}(s,a)) \tag{2.1}$$

Com

$$Q^{\pi^*}(s,a) = r(s,a) + \gamma V^{\pi^*}(\delta(s,a)) \tag{2.2}$$

Antes de começar, os valores Q são inicializados com um valor fixo. Cada vez que o agente recebe uma recompensa, os valores Q são recalculados e corrigidos, baseados na nova informação para cada combinação de estado s de S, e ação a de A. Nesse valor Q fica implícito o quão bom é realizar determinada ação em determinado estado. Esta técnica é chamada de Programação Dinâmica.

A função Q pode ser aproximada em uma tabela de procura $\widehat{Q}$:

$$\widehat{Q}(s_t, a_t) \leftarrow r_t + \gamma \max_{a'} \widehat{Q}(s_{t+1}, a_{t+1}). \tag{2.3}$$

A escolha de uma ação a_i em um estado s baseia-se, por exemplo, na seguinte distribuição probabilística (DZEROSKI; RAEDT; BLOCKEEL, 1998):

$$Pr(a_i|s) = \frac{T^{-\widehat{Q}(s,a_i)}}{\sum_j T^{-\widehat{Q}(s,a_j)}}. \tag{2.4}$$

O parâmetro temperatura T controla a exploração do agente. Baixos valores de T fazem com que o agente prefira ações com altos valores de Q, enquanto que um valor T mais elevado faz com que o agente explore mais outras opções.

O algoritmo Q-Learning, descrito no Algoritmo 2.1, recebe como entrada informações sobre o estado atual e fornece como saída a ação a ser executada e que modifica o ambiente, e consequentemente, o estado. Com as informações sobre o estado atual, o algoritmo seleciona uma ação baseada na distribuição probabilística da função 2.4 e a executa. Ao executá-la, ele recebe uma recompensa imediata, observa o estado resultante de sua ação, e atualiza então a tabela de valores Q de acordo com a função 2.3.

O fato do Q-Learning armazenar os valores Q em uma tabela torna-o inadequado para

Algoritmo 2.1 Q-Learning

```
//entradas: informações sobre o estado atual
//saída: ação a ser executada
para cada s,a faça
  inicializa tabela Q(s,a) = 0
fim para
gera um estado inicial s0
repita
  seleciona uma ação a e a executa
  recebe uma recompensa r = r(s,a)
  observa o novo estado s'
  atualiza a tabela Q(s,a) como segue:
  Q(s,a) ← r + γ max_a' Q(s',a')
  s ← s'
até que s seja o estado objetivo
```

uso em grandes espaços de estados, devido ao uso excessivo de memória para armazená-la; além disso, quando o objetivo do problema é alterado, o agente deve reaprender a função Q do início, mesmo quando os objetivos são similares, o que leva a um certo desperdício de conhecimento prévio.

Esse modo de representar o conhecimento em tabelas é inadequado para problemas de planejamento. Por exemplo, no Mundo dos Blocos, os estados são representações de todas as combinações possíveis das posições dos blocos, como descritos na introdução. O maior problema dessa forma de representação é a presença obrigatória de todos os estados e ações possíveis, o que o torna inviável para situações onde o número de blocos é muito grande.

A figura 1.1 ilustra um exemplo do Q-Learning aplicado ao Mundo dos Blocos. O episódio ilustrado consiste de três pares estado-ação que são executados da esquerda para a direita. Quando o objetivo (on(2,3)) é alcançado, após a execução da última ação, os valores Q são calculados a partir da direita para a esquerda, como mostrado na figura. Mais detalhes sobre o Mundo dos Blocos são fornecidos no Capítulo 3.

2.2 Programação Lógica Indutiva

A importância do ILP reside na sua capacidade de generalizar e representar o conhecimento. O sistema ILP consiste em um aprendiz ao qual são fornecidos exemplos de treinamento, a partir dos quais ele extrai novas relações e regras usando outras relações conhecidas da sua base de conhecimento. O conhecimento é representado por uma linguagem de progra-

mação lógica, formalismo computacional que consiste no uso de lógica de primeira ordem para expressar o conhecimento (na forma de predicados), e a inferência para manipulá-lo e extrair hipóteses (LAVRAC; DZEROSKI, 1994).

O uso de um sistema ILP resolve o problema de escalabilidade do algoritmo Q-Learning em representar o Mundo dos Blocos, ao usar Lógica de Primeira Ordem (LPO) que permite uma descrição relacional do domínio. Por exemplo:

"O bloco A está sobre o bloco B"

Os objetos são os blocos A e B, enquanto que a relação entre os blocos é "está sobre". Os estados e as ações são representados por predicados e através de regras é possível atingir o outro estado. Nessa representação, apenas os estados e ações necessários aparecem. Além disso, é possível aproveitar o conhecimento adquirido para alcançar metas similares.

Os sistemas que utilizam esse tipo de linguagem de representação, são chamados de sistema de aprendizado relacional, e são constituídos por (SILVA, 2009):

- uma **linguagem formal** - que especifica as sentenças que podem ser utilizadas para expressar conhecimento;
- uma **semântica** - que especifica o significado da sentença na linguagem, tanto como entrada como saída;
- uma **teoria de raciocínio** - que é uma possível especificação não determinística de como uma resposta pode ser derivada de uma base de conhecimento.

A linguagem formal define os símbolos e como eles devem ser organizados. E esses são divididos em variáveis, termos, átomos, predicados, fatos e regras.

A **variável** é uma sequência de caracteres que indica um **termo**, e que pode ser uma variável ou uma constante. Os **predicados** são palavras que se referem as relações e os **átomos** são encontrados na forma p ou $p(t_1,...,t_n)$ onde p é um predicado e t_i denota um termo. Alguns exemplos de átomos são: Feliz(João) e JogoBasquete(Assis, Franca) (SILVA, 2009).

O **fato** é considerado um átomo como cláusula. As **regras** são cláusulas com dois literais que utiliza a forma a $\rightarrow$ b, em que a é conhecido como o corpo, conhecido também como premissa e b é a cabeça da regra, se referenciando a conclusão.

Por exemplo, supondo-se um conhecimento prévio dos átomos *pai(Y,X)*, Y é pai de X, e *feminino(X)*, X é do sexo feminino, pode-se inferir corretamente a hipótese *filha(X,Y)*, X é filha de Y, dada a seguinte regra (implicação).

$$filha(X,Y) \leftarrow feminino(X), pai(Y,X).$$

É de extrema importância especificar bem a base de conhecimento, uma vez que seus predicados especificam conhecimento geralmente válido para todos os exemplos de treinamento. Dada uma base adequada, o aprendizado será eficiente e, caso contrário, o aprendizado será difícil, senão impossível. Isso enfatiza a importância de se especificar corretamente os predicados importantes na base de conhecimento.

O foco deste projeto está nas Árvores de Decisão Lógica de Primeira Ordem (*First Order Logical Decision Tree - FOLDT*) como meios de representação do ambiente. De Blockeel e Raedt (1998), tem-se a seguinte definição para FOLDT:

Definição 2.1 (FOLDT) *Uma árvore de decisão lógica de primeira ordem é uma árvore de decisão binária no qual*

1. *os nós da árvore contêm um conjunto de literais, e*
2. *nós diferentes podem compartilhar variáveis, sob a seguinte restrição: a variável introduzida em um nó não deve aparecer no ramo à direita desse nó. Isso decorre do fato de que essa variável é existencialmente quantificada dentro da conjunção daquele nó. A sub-árvore à direita é relevante apenas quando essa conjunção falha, na qual situação qualquer referência a essa variável se torna sem sentido.*

2.3 Aprendizado por Reforço Relacional

O Aprendizado por Reforço Relacional (ARR) é uma técnica de aprendizado que combina AR com ILP. O objetivo do ARR é abstrair-se das identidades específicas de estados, ações e até mesmo de objetos do ambiente, e identificá-los apenas referindo-se a objetos por meio de suas propriedades e relações entre si. Ao definir o objeto pelas suas propriedades (forma, tamanho, cor) e não pela sua identidade (maçã, caixa, copo), qualquer objeto que possua essas propriedades é lidada corretamente pelo agente aprendiz.

Uma definição de ARR é dada como (DRIESSENS, 2004):

Definição 2.3 (Aprendizado por Reforço Relacional) *O Aprendizado por Reforço Relacional pode ser definido como segue:*

Dados:

1. *um conjunto de estados S, representado em formato relacional,*
2. *um conjunto de ações A, também representado em formato relacional,*
3. *uma função de transição* $\delta : S \times A \rightarrow S$ *desconhecida,*
4. *uma função recompensa* $r : S \times A \rightarrow \Re$,
5. *conhecimento prévio geralmente válido sobre o ambiente*

Encontrar *uma política ótima* $\pi^* : S \rightarrow A$ *que maximiza a função valor* $V^{\pi}(s_t)$ *para todo* $s_t \in S$.

Cada estado s_i é representado pela relação entre os blocos, e possui um conjunto de ações possíveis que podem ser realizadas pelo sistema. As pré-condições para que uma ação seja realizada é que o bloco a ser movido e o bloco, ou o espaço no chão, para onde ele está sendo movido estejam ambos desimpedidos. A pós-condição de cada ação é que o bloco que foi movido esteja sobre o bloco para onde ele foi movido. O objetivo do sistema (conjunto de estados metas) fica embutido na função recompensa, enquanto que neste projeto, a função de transição δ foi considerada como uma função determinística. O conhecimento prévio pode incluir informações como o conhecimento parcial sobre o efeito das ações, similaridade entre diferentes estados, entre outros.

O ARR foi desenvolvido para resolver os problemas que o AR apresenta, citados anteriormente. Ao invés de usar uma tabela de procura para armazenar os valores Q, o sistema ARR usa as informações coletadas sobre os valores Q de diferentes pares estado-ação em um algoritmo de regressão relacional que constrói uma função Q generalizada. Sua qualidade está no fato de usar uma representação relacional do ambiente e das ações disponíveis, e por usar um algoritmo de regressão relacional para construir a função Q. Esse algoritmo evita o uso de identidades específicas dos estados, ações e objetos na sua função, contando com estruturas e relações presentes no ambiente para definir semelhanças entre os pares estado-ação e prever seus correspondentes valores Q.

O algoritmo Q-RRL (algoritmo 2.3), que é um algoritmo *Q-Learning* adaptado para ARR, começa por inicializar a função Q e, depois, aprende episódios como qualquer outro

Algoritmo 2.2 Q-RRL

//entradas: informações sobre o estado atual
//saída: ação a ser realizada
inicializa a hipótese $\widehat{Q}_0$ da função Q
$e \leftarrow 0$
repita
  $Exemplos \leftarrow \emptyset$
  gera um estado inicial s_0
  $i \leftarrow 0$
  repita
    seleciona uma ação a_i para o estado s_i usando uma política derivada da hipótese $\widehat{Q}_e$ e a executa
    receba uma recompensa r_i
    observa o novo estado s_{i+1}
    $i \leftarrow i+1$
  até que s_i seja o estado objetivo
  para $j = i-1$ to 0 **faça**
    gera exemplo $x = (s_j, a_j, \widehat{q}_j)$ onde $\widehat{q}_j \leftarrow r_j + \gamma \max_a \widehat{Q}_e(s_{j+1}, a)$
    $Exemplos \leftarrow Exemplos \cup x$
  fim para
  atualiza $\widehat{Q}_e$ usando *Exemplos* e um algoritmo de regressão relacional para produzir $\widehat{Q}_{e+1}$
  $e \leftarrow e+1$
até que não exista mais episódios

algoritmo de Q-Learning. Na escolha de uma ação, o algoritmo usa a mesma função de probabilidade usada pelo Q-Learning padrão. Durante cada episódio de aprendizado, todos os estados encontrados e as respectivas ações tomadas são armazenadas, juntamente com as recompensas relacionadas com cada par estado-ação. No final de cada episódio, quando o sistema atinge seu objetivo, o sistema aproxima o valor Q para cada par estado-ação usando retro propagação da recompensa e a aproximação da função Q atual.

O algoritmo então fornece o conjunto de triplas (*estado, ação, valor Q*) para o mecanismo de regressão relacional que usará esse conjunto de exemplos para atualizar a estimativa da função Q, e assim procede com o próximo episódio.

Assim como no algoritmo 2.1, o algoritmo Q-RRL recebe como entrada o estado atual e fornece como saída a ação a ser executada.

Inicialmente o mecanismo de regressão relacional usado era o algoritmo TILDE, uma árvore de decisão lógica, que foi depois estendido com um algoritmo incremental para formar o algoritmo TG, ambos descritos a seguir.

2.3.1 TILDE

O TILDE é um sistema que gera uma árvore de decisão lógica de primeira ordem para realizar um processo de classificação, regressão ou agrupamento de dados, usando fatos ao invés de atributos. A figura 2.2 mostra a arquitetura geral do sistema.

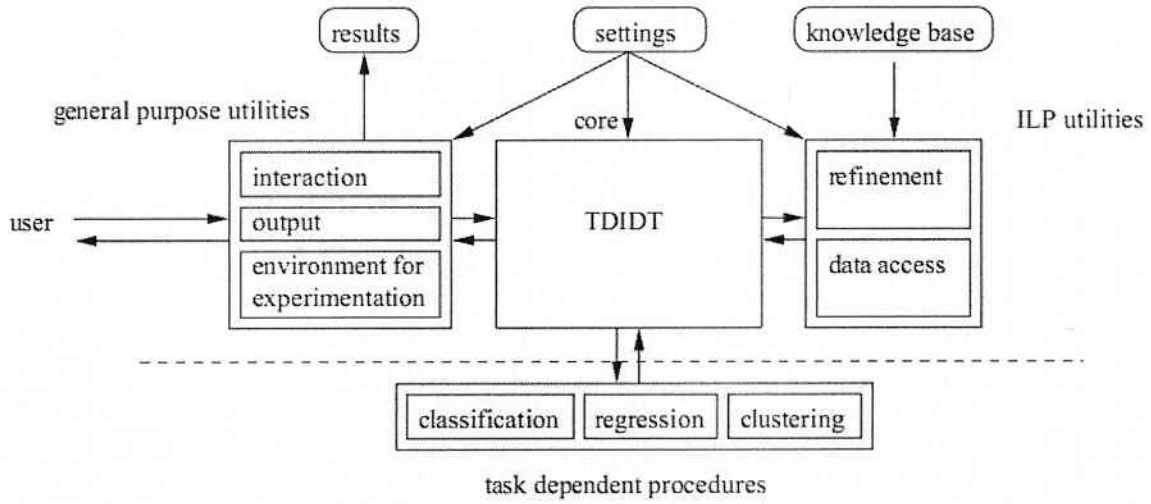

Figura 2.2: Arquitetura geral do algoritmo TILDE, onde as setas indicam a direção do fluxo da informação. Figura extraída de (BLOCKEEL; RAEDT, 1998)

.

O sistema é composto por um núcleo principal *core* que contém um algoritmo genérico TDIDT e três módulos auxiliares (BLOCKEEL; RAEDT, 1998):

- um módulo implementa a parte ILP do sistema (ILP *utilities*): contém o operador de refinamento a uma cláusula (para gerar o conjunto de testes que será implementado no nó), assim como todas as funções de acesso aos dados;
- um módulo que implementa todos os procedimentos relativos às três opções de operação: classificação, regressão e agrupamento de dados (TDIDT) *task dependent procedures*;
- um módulo que implementa toda a parte relativa à interface com o usuário (*General purpose utilities*).

O sistema recebe dois tipos de entrada: um arquivo de configuração (*settings*), especificando os parâmetros do sistema, e uma base de conhecimento (*Knowledge base*) contendo exemplos e conhecimento prévio, gerando um ou dois arquivos de saída, contendo os resultados do processo indutivo (*results*). As flechas indicam o fluxo de informação dentro do sistema.

Dentro do contexto de ARR (mostrado no algoritmo 2.2), o sistema TILDE é um dos algoritmos de regressão relacional que pode ser usado para processar os exemplos gerados pelo ARR. Cada exemplo recebido pelo TILDE é testado pelos nós da árvore até alcançar uma folha. Por não ser incremental, o sistema TILDE armazena todos os pares estado-ação encontrados (não apenas os gerados pelo episódio e) e o valor $\widehat{q}$ mais recente para cada par. Uma árvore de regressão relacional $\widehat{Q}_e$ é induzida dos exemplos (s,a,q) após cada episódio e. Essa árvore é então usada pelo ARR para selecionar ações no episódio $e+1$. Maiores detalhes sobre o algoritmo TILDE podem ser encontrados em (BLOCKEEL; RAEDT, 1998).

O uso do TILDE no ARR apresenta quatro problemas identificados (DRIESSENS; RAMON; BLOCKEEL, 2001):

- o sistema precisa manter o registro de uma quantidade sempre crescente de exemplos: para cada par estado-ação diferente encontrado, o valor Q é mantido.
- quando um par estado-ação é encontrado pela segunda vez, o novo valor Q precisa substituir o valor armazenado anteriormente, o que significa que o antigo exemplo precisa ser encontrado na base de conhecimento e ser substituído.

- a árvore é reconstruída do ponto de partida a cada episódio. Isso, assim como a substituição de exemplos redundantes, são procedimentos que tomam cada vez mais tempo de processamento à medida que o conjunto de exemplos cresce.

- as folhas da árvore deveriam identificar agrupamentos de pares estado-ação semelhantes, no sentido de que possuem o mesmo valor Q. Quando o valor Q de um par estado-ação é atualizado na folha, o valor Q de todos os outros pares existentes na mesma folha deveriam ser atualizados automaticamente, o que não acontece.

Para resolver esses problemas, é necessário um algoritmo de indução incremental. A próxima seção trata desse algoritmo.

2.3.2 TG

O algoritmo TG (DRIESSENS; RAMON; BLOCKEEL, 2001) descrito no algoritmo 2.4 é uma combinação do algoritmo TILDE (BLOCKEEL; RAEDT, 1998), que constrói uma árvore de primeira ordem de classificação e de regressão, e o algoritmo G (CHAPMAN; KAELBLING, 1991), que usa um número de valores estatísticos relativos ao desempenho de cada extensão possível em cada folha da árvore para construí-la incrementalmente.

Algoritmo 2.3 TG

```
//entradas: exemplos fornecidos pelo sistema ARR
//saída: árvore TG
inicializa criando uma árvore com uma folha simples e estatísticas vazias
para cada exemplo de treinamento que se tornar disponível faça
    aplica o exemplo sobre a árvore usando os testes dos nós internos até atingir uma
    folha
    atualiza as estatísticas da folha de acordo com o novo exemplo
    se as estatísticas na folha indicam que uma nova divisão é necessária, então
        gera um nó interno usando o teste indicado
        gera duas novas folhas com estatísticas vazias
    fim se
fim para
```

Assim como o TILDE, o TG usa uma linguagem de representação relacional para descrever os exemplos e os testes que podem ser usados na árvore de regressão.

O ARR passa todos os exemplos gerados em um episódio para o TG, que, um por um, testa os exemplos pelos nós da árvore até atingir uma folha. O valor Q da folha é atualizado com o novo valor (o valor Q mantido em cada folha é a média dos valores Q

dos exemplos que chegaram até ela). As estatísticas de cada folha consistem no número de exemplos nos quais cada teste possível falha ou tem sucesso, assim como a soma dos valores Q e a soma dos quadrados dos valores Q para cada um dos subconjuntos criados pelo teste. Essas estatísticas podem ser calculadas incrementalmente e são suficientes para verificar se algum teste é suficiente, isto é, se a variância dos valores Q dos exemplos da folha seria reduzida suficientemente dividindo a folha ao usar esse teste em particular. Um F-test padrão (SNEDECOR; COCHRAN, 1991) com nível de significância 0.001 é usado para fazer essa decisão. Se o teste é relevante, então a folha é dividida em duas sub-árvores, e cada folha nova recebe um valor Q baseado nas estatísticas obtidas do teste usado. Depois, esse valor é atualizado com os exemplos que chegam nessa nova folha. O algoritmo TG recebe como entradas os exemplos fornecidos pelo sistema ARR, e fornece como saída a árvore gerada com o conhecimento armazenado.

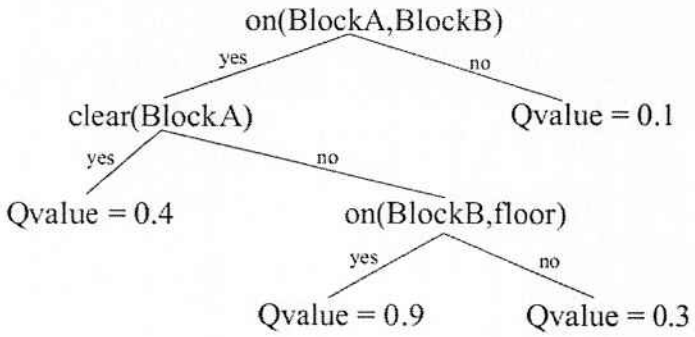

Figura 2.3: Uma árvore de regressão relacional

Por exemplo, supondo-se uma folha recém-criada, com zero estatísticas. A cada iteração do sistema, novos exemplos são gerados e entregue à árvore. Cada exemplo que chega em uma folha é armazenado e o valor Q da folha é atualizada com o valor Q do exemplo (o valor Q da folha é a média dos valores Q dos exemplos armazenados na folha).Após juntar um número mínimo de exemplos (*Minimal Sample Size* - MSS) na folha, o algoritmo TG aplica todos os testes possíveis a esse conjunto de exemplos, e verifica qual oferece a melhor divisão dos exemplos em dois subconjuntos. Aquele que oferecer a melhor divisão dos exemplos da folha é escolhido como teste dessa folha e duas novas folhas com zero estatísticas são criadas a partir dela.

Os testes podem ser dos seguinte tipos:

- equal(X,Y) o bloco X é igual ao bloco Y, no sentido de que bloco X é o bloco Y

- on(X,Y) o bloco X está diretamente sobre o bloco Y
- above(X,Y) o bloco X está na pilha de blocos que estão sobre o bloco Y
- clear(X) o bloco X está livre

A figura 2.3 fornece um exemplo de uma árvore de regressão de primeira lógica.

Depois de construída a árvore na fase de aprendizagem, ela passa a ser usada como base de conhecimento do agente para a escolha de ações futuras.

Por ser incremental, o algoritmo TG não precisa reconstruir a árvore a cada episódio, e não precisa guardar os exemplos gerados durante os vários episódios.

3 Desenvolvimento do Projeto

Neste capítulo é apresentado como o projeto foi implementado, o seu ambiente de desenvolvimento e os requisitos mínimos para sua implantação.

3.1 Ambiente de Desenvolvimento

O projeto foi inteiramente desenvolvido com o uso dos softwares *open source* ECLIPSE e NETBEANS; ambos são ambientes de desenvolvimento integrado (IDE) que suportam a tecnologia Java. Além disso, o cálculo das estatísticas de cada folha é auxiliado pelo uso da biblioteca *Commons Math* que pode ser encontrada em: http://commons.apache.org/math/. A renderização da interface do programa utiliza uma versão adaptada da classe *GraphicsUtil* que pode ser obtida em: http://www.win.tue.nl/ sthite/sp/GraphicsUtil.java. Para a geração de gráficos, foi utilizada um pacote Java chamado *Plot*, obtida em: http://homepage.mac.com/jhuwaldt/java/Packages/Plot/PlotPackage.html.

3.2 Requisitos para Implantação

Para executar o projeto, é necessário a instalação do *Java Runtime Environment (JRE)* versão 6 e do IDE Eclipse, além de uma configuração mínima de 1Gb de memória RAM.

3.3 Domínio do Projeto

O processo de ARR com TG neste projeto é aplicado ao Mundo dos Blocos. O Mundo dos Blocos é um clássico problema de planejamento frequentemente utilizado dentro de IA, devido às suas características simples e claras. O Mundo dos Blocos consiste em:

- uma superfície lisa sobre a qual ficam os blocos. Neste caso, a superfície lisa é

denominada chão e é representado no programa por "F".

- um conjunto de blocos idênticos identificados por letras ou números.
- os blocos podem ser movidos para o chão ou para cima de outro bloco, deste que ele esteja desimpedido de se mover, e que o espaço para o qual está sendo movido esteja livre.
- apenas um bloco pode ser movido por vez.

Os estados são representados por fatos, ou predicados, da seguinte forma:

- on(X,Y) descreve que o bloco X está sobre Y, podendo Y assumir o papel de um bloco ou do chão;
- clear(X), indicando que sobre o bloco X não existe nenhum outro bloco.

O objetivo do robô é calcular uma política ótima π^* que o permita atingir um determinado objetivo (por exemplo, on(A,B)) pelo meio mais rápido. Cada configuração dos blocos no mundo constitui um estado s diferente, e cada ato de movimentar um bloco de lugar para outro constitui uma ação a, que altera o ambiente. Cada ação realizada muda o estado do ambiente, e o robô aprendiz assim prossegue até atingir o estado objetivo. Uma vez atingido o estado objetivo, ele recebe uma recompensa r de valor 1. As ações que resultam em uma estado que não o estado objetivo, recebem valor 0. Todas as etapas percorridas pelo robô desde o estado inicial até o estado objetivo e o conjunto de triplas estado-ação-recompensa (s,a,r) gerados por essas etapas caracterizam um episódio.

Os testes nos nós da árvore (do algoritmo ARR) podem ser dos seguintes tipos:

- on(X,Y) testa se o bloco X está sobre o bloco Y;
- clear(X) testa se o bloco X está livre;
- above(X,Y) testa se, na pilha de blocos que está sobre Y, está X;
- equal(X,Y) testa se X e Y são o mesmo bloco.

3.4 O Programa

Neste projeto foi desenvolvido um programa implementando o sistema ARR-TG tendo como domínio de aplicação o Mundo dos Blocos. Neste programa é possível configurar o Mundo dos Blocos com a quantidade de blocos desejada (identificados por números), configurar o objetivo, o fator de desconto γ, a temperatura T (lembrando que valores altos favorecem a exploração, enquanto que valores baixos dão preferência às ações conhecidamente recompensadoras naquele estado), a recompensa (valor fornecido ao se atingir o objetivo) e o tamanho mínimo de exemplos (MSS), além de, para melhor visualização do funcionamento do programa, oferecer a opção de configurar o tempo entre uma ação e outra.

É possível executar o programa em três modos: passo a passo, para observar as ações tomadas pelo sistema; por episódio, para executar um episódio inteiro; e em ciclos, para executar o algoritmo n vezes. A figura 3.1 ilustra a interface do programa.

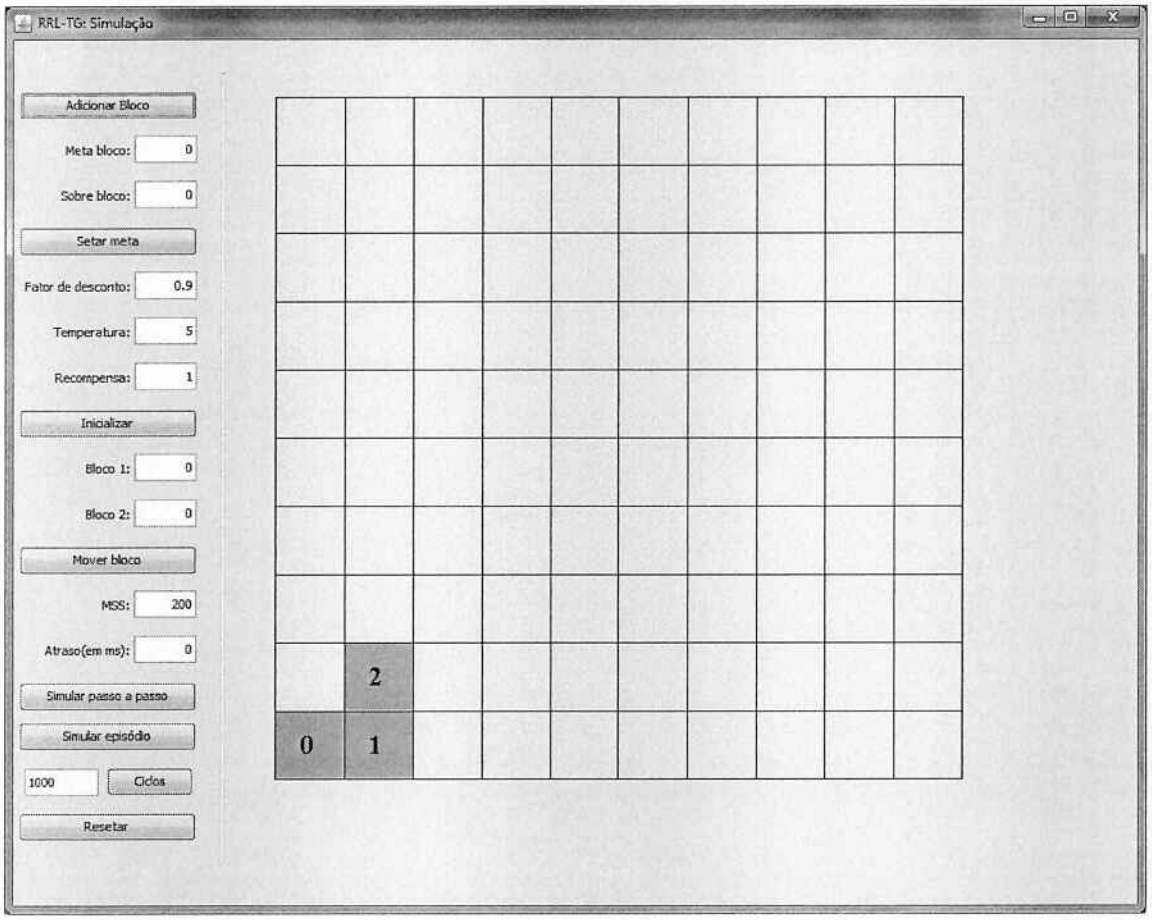

Figura 3.1: Interface do programa de ARR-TG, indicando que os blocos 0 e 1 estão sobre o chão e que o bloco 2 está sobre o bloco 1.

A figura 3.2 mostra a arquitetura geral do sistema. O domínio de aplicação é o ambiente a ser explorado pelo agente, no caso, o Mundo dos Blocos. O ARR-TG interage com o ambiente e o conhecimento prévio para a escolha de suas ações. No conhecimento prévio está a árvore gerado pelo algoritmo TG, que serve como banco de consulta na escolha de uma ação a ser realizada. Nele também estão as pré e as pós condições das ações, que foram descritos dentro da seção 2.3. Os parâmetros fornecidos ao sistema são: o fator de desconto γ, a Temperatura T, a recompensa r e o MSS.

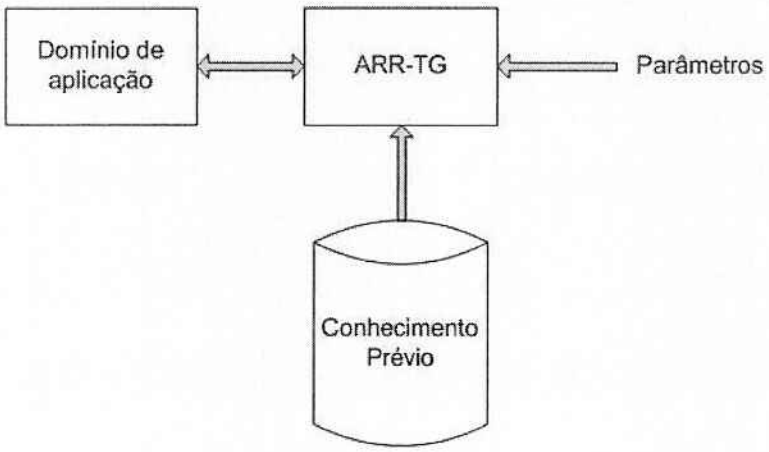

Figura 3.2: Arquitetura geral do Sistema.

4 *Testes e análise*

Neste capítulo estão a descrição dos testes realizados, os parâmetros usados e as análises dos resultados obtidos pela implementação do algoritmo TG em um sistema ARR.

4.1 Configuração do Experimento

Para todos os testes realizados, a fim de observar o desempenho do algoritmo sob diferentes situações, foram usados os seguintes parâmetros: fator de desconto $\gamma = 0.9$ (eq. 2.3), temperatura $T = 5$ com fator de decaimento de 0,95 (eq. 2.4) [1] e recompensa $r = 1$ (eq. 2.3).

Duas séries de testes são realizadas, descritas a seguir.

4.1.1 Número de Blocos Fixos

A primeira série de testes compara a curva de aprendizado do algoritmo TG em um espaço de estados fixo (DRIESSENS; RAMON; BLOCKEEL, 2001). São três espaços testados: com 3 blocos, com 4 blocos e com 5 blocos.

Para os testes de espaço fixo foram realizados 1000 episódios de treino. A cada episódio de treino, são gerados aleatoriamente 10000 episódios de testes que se utilizam de uma política *greedy* para a escolha de ações - isso remove a influência da estratégia de exploração nos resultados. Uma recompensa de 1 é dada se o estado objetivo é alcançado no número mínimo de passos, 0 no caso contrário. Neste trabalho, o número mínimo de passos foi configurado como o número mínimo de passos para se atingir o objetivo no pior cenário (na referência (DRIESSENS; RAMON; BLOCKEEL, 2001) do teste, não está claro como esse valor é configurado). O parâmetro MSS (pg. 26)foi configurado para o valor 200.

[1] A escolha de uma temperatura elevada inicial serve para garantir a exploração do ambiente pelo algoritmo

4.1.2 Número Variável de Blocos

Na segunda série de testes, o número de blocos é variado entre 3 e 5 blocos durante os episódios de treino, isto é, cada episódio de 3 blocos é seguido por um de 4 blocos, que por sua vez é seguido por um de 5, e assim por diante (DRIESSENS, 2004). A cada 100 episódios de treino, são gerados aleatoriamente 200 episódios de testes, que também se utilizarão de uma política *greedy* para a escolha de ações. No total serão 1000 episódios de treino. Os testes serão repetidos para valores de MSS diferentes, a seguir: 30, 50, 100, 200 e 500. A atribuição de recompensa segue a mesma estratégia da série de testes anterior.

4.2 Resultados Obtidos

Em relação aos resultados dos testes originais (DRIESSENS, 2004; DRIESSENS; RAMON; BLOCKEEL, 2001), os resultados obtidos neste projeto apresentaram algumas diferenças. Tais variações podem ser atribuídas a vários fatores, uma vez que alguns parâmetros usados nos testes originais são desconhecidos. No primeiro teste, o parâmetro MSS é desconhecido, assim como γ (fator de desconto), incerto em ambos os testes. A primeira influência na seleção dos testes nas folhas, enquanto que o segundo influência na convergência do algoritmo. O gráfico da figura 4.1 mostra o resultado do teste original (DRIESSENS; RAMON; BLOCKEEL, 2001) para o espaço fixo de 3, 4 e 5 blocos. A árvore correspondente não se encontrava disponível no mesmo. O gráfico da figura 4.2 mostra os resultados obtidos para o espaço variável com MSS de 30, 50, 100, 200 e 500, enquanto que a figura 4.3 mostra a árvore correspondente para MSS = 200. Essas figuras representam os resultados esperados pelo projeto.

Os gráficos das figuras 4.4, 4.6, 4.8, 4.10, 4.12, 4.14, 4.16, 4.18 mostram a recompensa média recebida em cada conjunto de testes. Pode-se considerar que a ordenada Y mostra a porcentagem de acertos do algoritmo nos testes à medida que a árvore vai sendo construída.

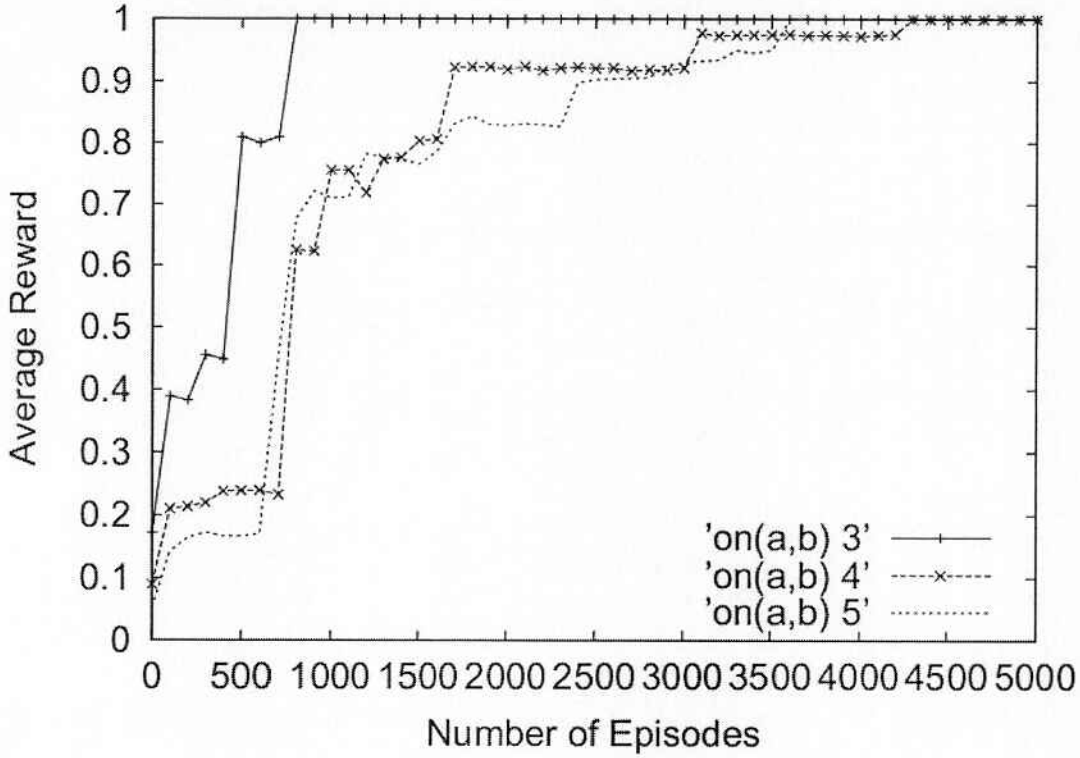

Figura 4.1: Curvas de aprendizado para os espaços fixo de 3, 4 e 5 blocos.

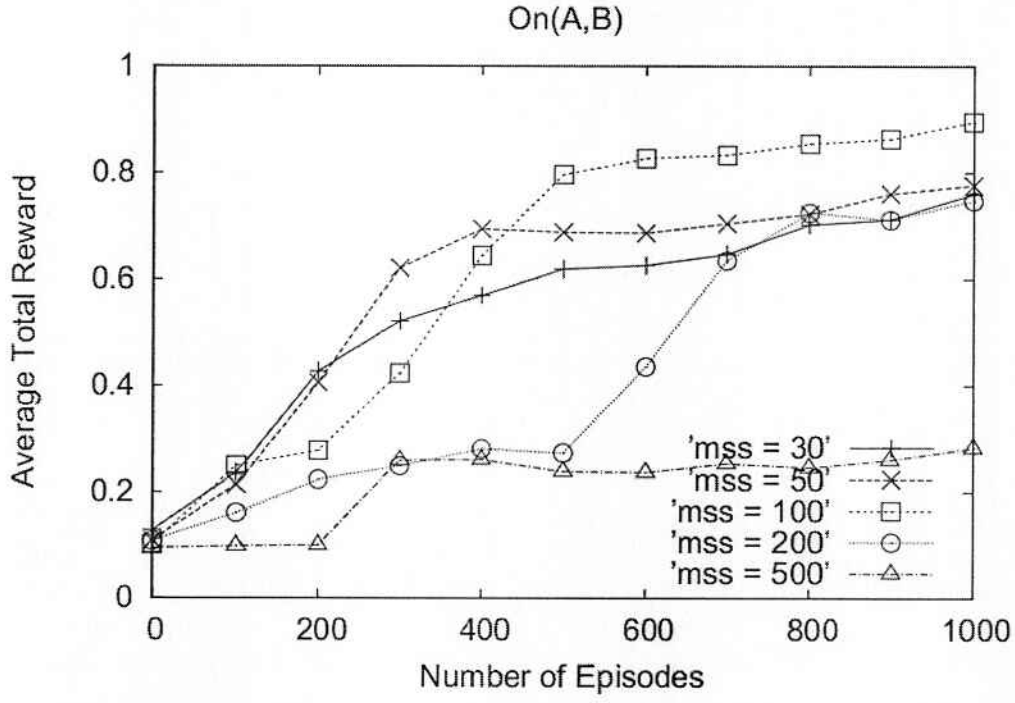

Figura 4.2: Curva de aprendizado para o espaço variável com MSS nos valores de 30, 50, 100, 200 e 500.

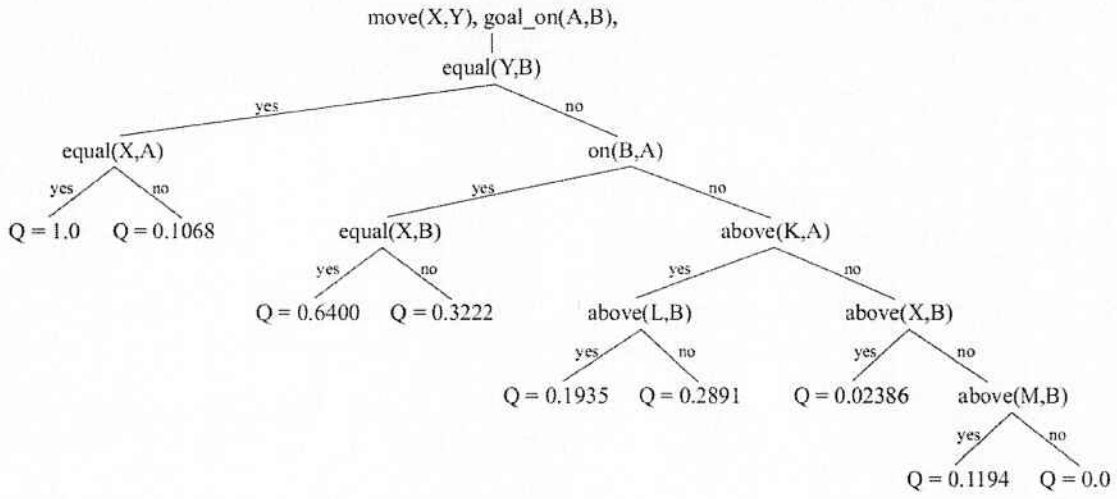

Figura 4.3: Árvore gerada para MSS = 200.

4.2.1 Número Fixo de Blocos

A seguir estão os gráficos obtidos na primeira série de testes. Pode-se observar que a maior quantidade de blocos no ambiente implica em uma velocidade de convergência mais lenta do algoritmo, uma vez que aumenta a quantidade de estados possíveis e, consequentemente, o nível de dificuldade de se atingir o objetivo. A mudança do valor da recompensa média pode ser atribuída a dois fatores: criação de um teste relevante na árvore, ou atualização dos valores Q da árvore que favoreçam a escolha de uma determinada ação.

O fato de um teste ser criado na árvore nem sempre implica em uma melhora na curva de aprendizado. Dependendo do teste escolhido, o sistema permanece no mesmo patamar da curva de aprendizado. Isso acontece porque às vezes o teste escolhido não é logicamente relevante. Por exemplo, o primeiro teste da figura 4.13 (*equal(X,A)*), no momento que é criado, é logicamente irrelevante para o agente. Isso porque ele testa se o bloco que está sendo movido é o bloco A (do objetivo). Se é o bloco A, ele pode estar sendo movido para cima de qualquer outro bloco, além do bloco B. Se não for, o teste não diz muito a respeito sobre o quão bom é a ação de mover outros blocos além do A. Em suma, o teste sozinho não é relevante ao aprendizado do sistema. Agora, na figura 4.5, o primeiro teste criado é *equal(Y,B)*. Esse teste verifica se o bloco Y (para onde o X está sendo movido) é o bloco B (do objetivo). Esse teste já muda a curva de aprendizado porque já direciona melhor a escolha de uma ação, uma vez que neste teste fica implícito o objetivo do agente (mover X para cima do bloco B).

Outro fator que afeta a curva de aprendizado do sistema é o valor Q nas folhas da árvore. Como explicitado anteriormente, o valor Q das folhas é a média dos valores Q dos exemplos que chegam na folha, e são esses valores Q das folhas que ditam a escolha de uma ação em um determinado estado. Esse valor Q das folhas é atualizado a cada episódio de treino. Pode acontecer que, no momento da criação de duas novas folhas, o valor Q inicial de uma folha seja maior que o da folha irmã. Tomando o teste *on(X,A)* na figura 4.7 como exemplo. Depois de verificar que o bloco Y não é o bloco B (no primeiro teste, *equal(Y,B)*), o agente verifica se o bloco X que está sendo movido é o bloco B (*equal(X,B)*). O valor Q do ramo da esquerda (0,4458) é menor que o valor Q do ramo da direita (0,46283).

Isso faz sentido, uma vez que informa ao agente que é uma idéia ruim ficar movendo o bloco B. Se ele move o bloco B para cima do bloco A, no mínimo mais duas ações terão de ser feitas para que o objetivo seja atingido. Supondo que inicialmente o valor do ramo esquerdo é maior que o valor do ramo direito; isso faz com que haja uma maior preferência por mover o bloco B (o que é ruim) de lugar e, consequentemente, o agente não aprende todo o potencial daquele teste (e a curva de aprendizado não é afetada). Mas como o valor Q dos ramos é atualizado a cada episódio de treino, a tendência é o valor Q do ramo esquerdo diminuir, e o do ramo direito, aumentar (uma vez que mover o bloco B pode resultar em um maior número de ações e portanto, uma recompensa média menor). Assim que o valor do ramo direito ultrapassar o do ramo esquerdo, o agente aprende corretamente que a ação de mover o bloco B não é boa, e passa a priorizar outras ações. A oscilação observada no gráfico da figura 4.7 é causada por essa mudança de valores nos ramos direito e esquerdo do teste *equal(X,B)*, quando esses valores ainda não se estabilizaram.

Analisando a árvore da figura 4.5, podemos observar que ela está logicamente correta. O primeiro teste juntamente com o primeiro teste do ramo esquerdo da árvore informa ao agente que mover o bloco X = A para cima do bloco Y = B resulta em um valor Q alto (Q = 1) e que portanto, é a melhor ação a se realizar, enquanto que mover qualquer outro bloco que não o bloco A para cima do bloco B é uma ação que resulta em uma recompensa menor, e portanto, não tão atrativa. Do lado direito, o teste *on(X,A)* juntamente com o teste *equal(Y,B)* informa ao agente que mover o bloco X, que está sobre o bloco A, para o bloco Y que não é o bloco B (ou seja, desobstruir o bloco A sem obstruir o bloco B, ambos do objetivo) também é uma ação interessante de ser realizada, por possuir o segundo maior valor Q da árvore. Por outro lado, se X não está sobre A, então a ação correspondente não diminui nem aumenta a distância do estado para o estado objetivo,

mas ainda assim, é preferível à ação que move um bloco qualquer que não o A para cima do bloco Y (no teste *equal(X,A)*).

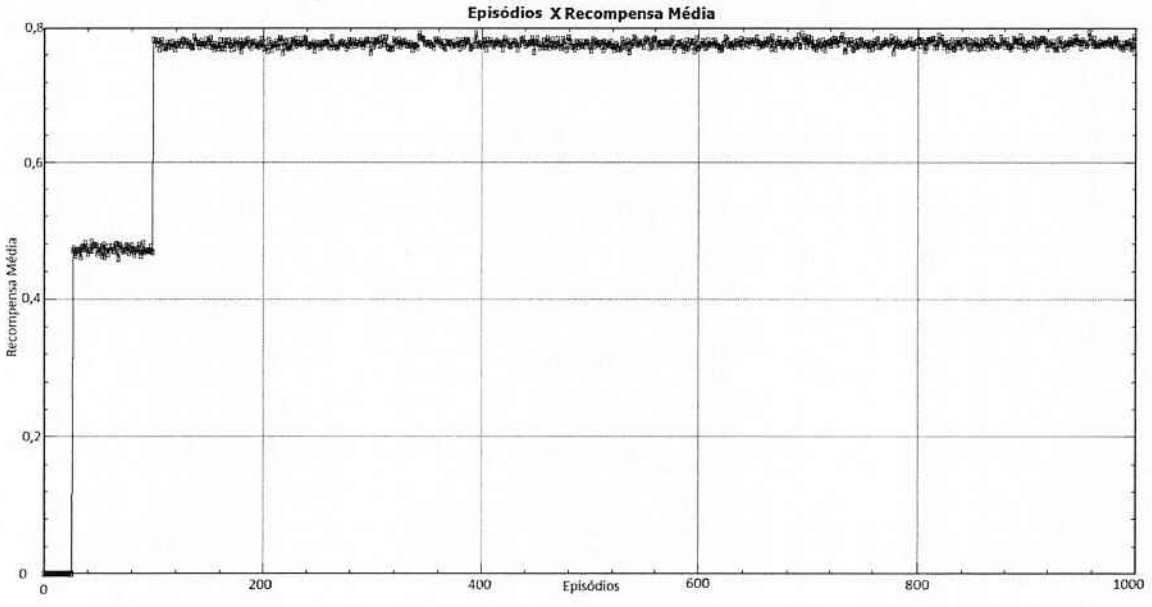

Figura 4.4: Curva de aprendizado para o espaço fixo de 3 blocos.

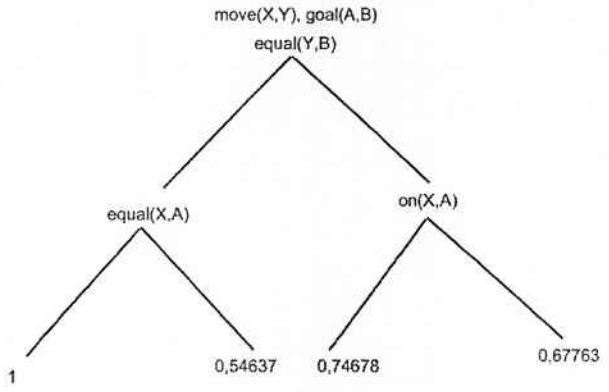

Figura 4.5: Respectiva árvore gerada após 1000 episódios de treino para o espaço fixo de 3 blocos.

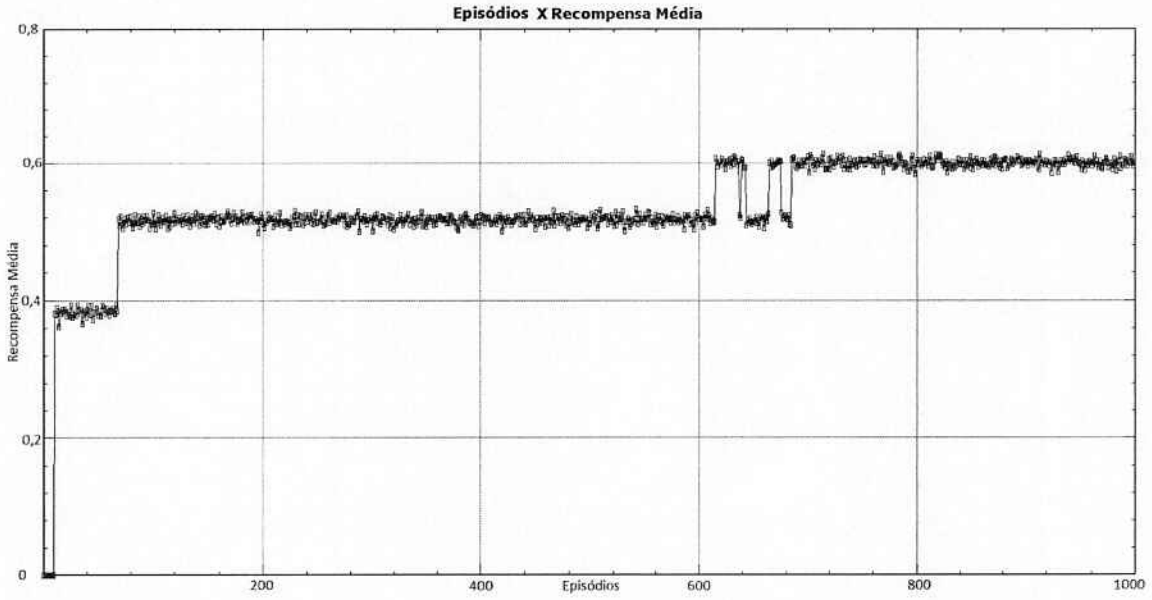

Figura 4.6: Curva de aprendizado para o espaço fixo de 4 blocos.

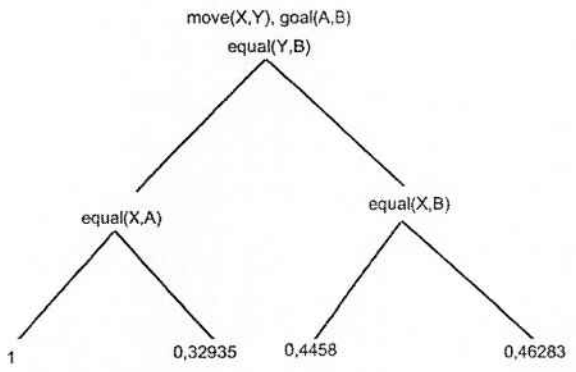

Figura 4.7: Respectiva árvore gerada após 1000 episódios de treino para o espaço fixo de 4 blocos.

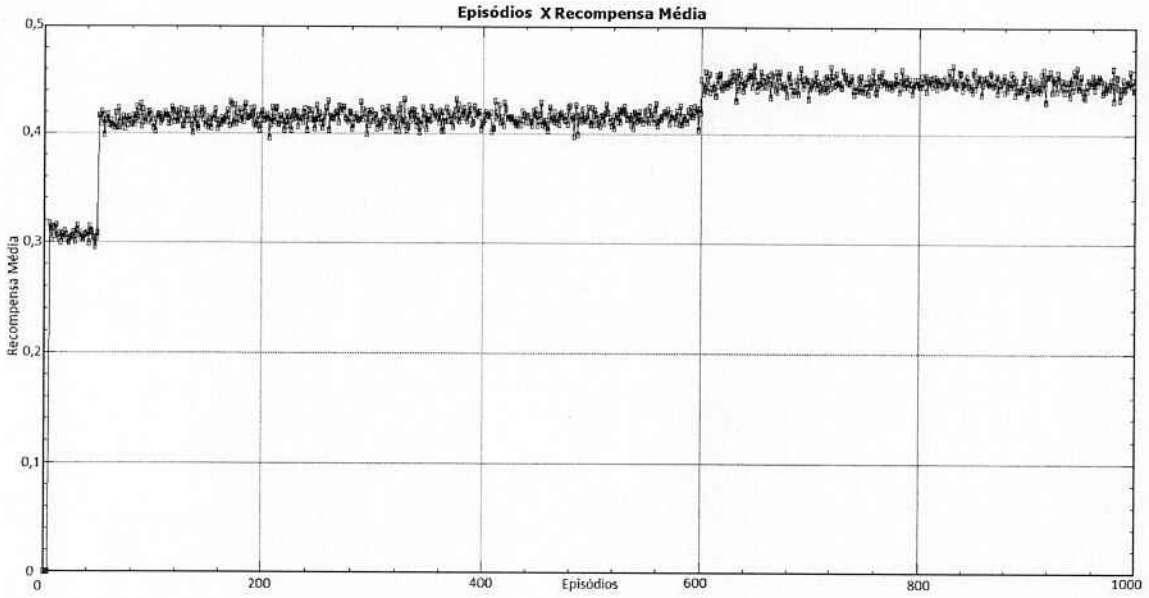

Figura 4.8: Curva de aprendizado para o espaço fixo de 5 blocos.

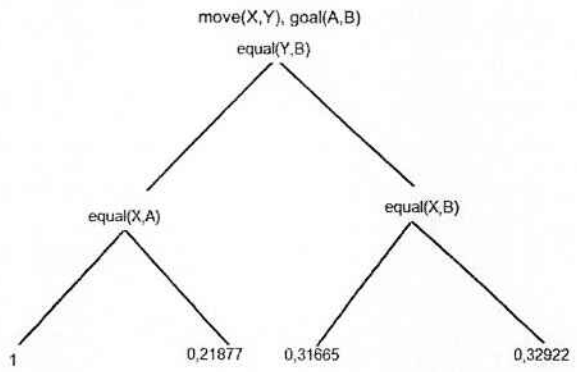

Figura 4.9: Respectiva árvore gerada após 1000 episódios de treino para o espaço fixo de 5 blocos.

4.2.2 Número Variável de Blocos

Considerando que o MSS controla a velocidade com que os testes são criados, valores baixos fazem com que os nós sejam criados mais rapidamente, e consequentemente, a árvore resultante seja maior. Valores altos de MSS diminuem a chance de se escolher um teste ruim nos nós iniciais, mas também causa lentidão no aprendizado do agente. É de se esperar que no início, o aprendizado do sistema para valores de MSS baixos seja mais rápido do que para valores altos, mas vá diminuindo com o tempo, no passo que, para valores altos de MSS, o aprendizado vá aumentando com o tempo até convergir para um certo valor.

Os gráficos das figuras 4.10, 4.12, 4.14, 4.16, 4.18, mostram que para valores altos de MSS, o aprendizado do sistema é relativamente mais lento se comparado com os outros de valor MSS menores. Com relação às árvores geradas, a única que se encontra incoerente é a da figura 4.19, no teste *equal(X,B)*. O valor Q do ramo esquerdo deveria ser menor que a do ramo direito, para desencorajar o agente de mover o bloco B, e no caso, o valor é maior. A tendência do sistema é corrigir esse valor com mais episódios de treino, então supõe-se que a árvore seja corrigida nos passos posteriores.

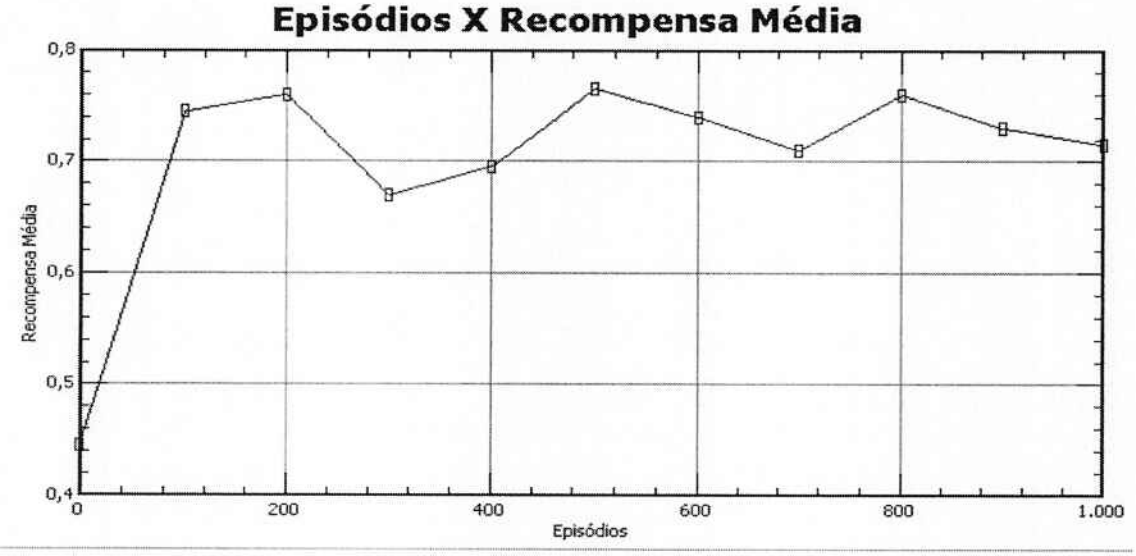

Figura 4.10: Curva de aprendizado para o espaço variável com MSS=30.

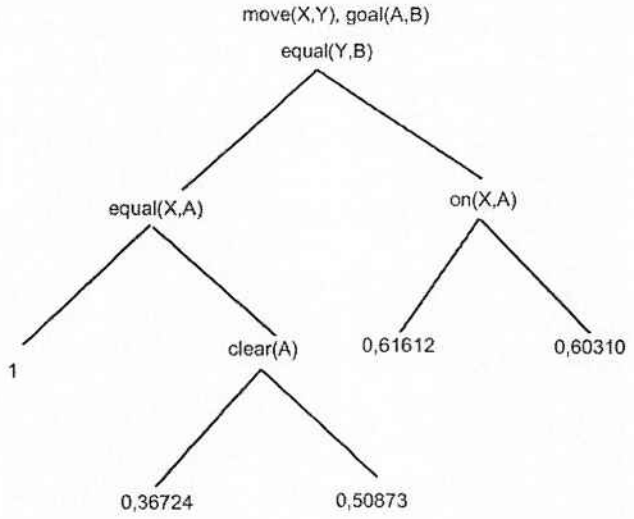

Figura 4.11: Respectiva árvore gerada após 1000 episódios de treino para o espaço variável com MSS=30.

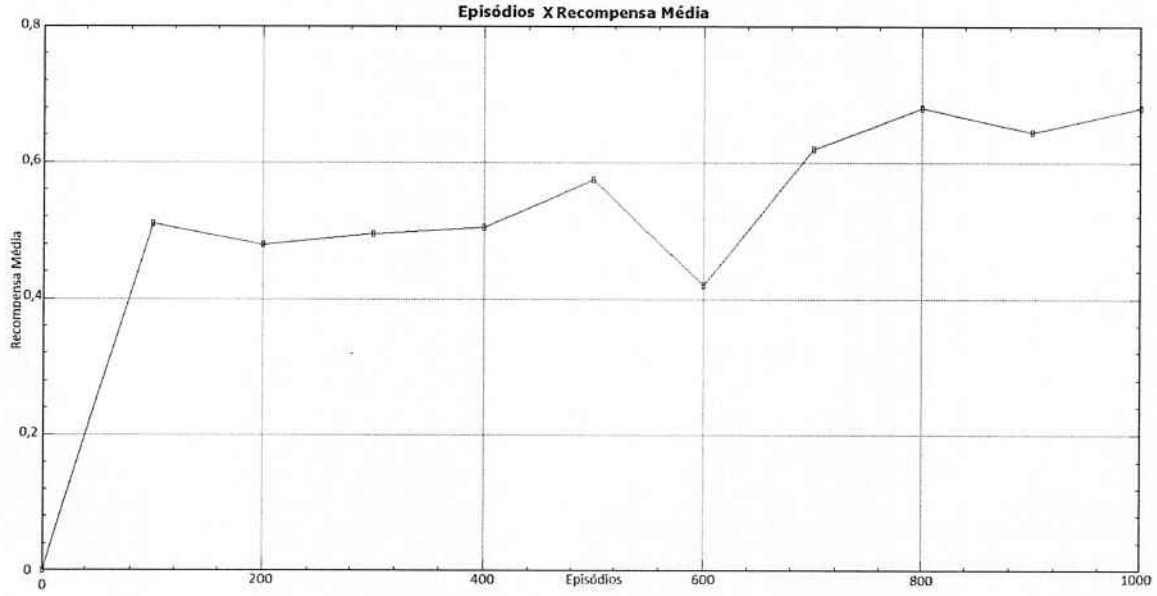

Figura 4.12: Curva de aprendizado para o espaço variável com MSS=50.

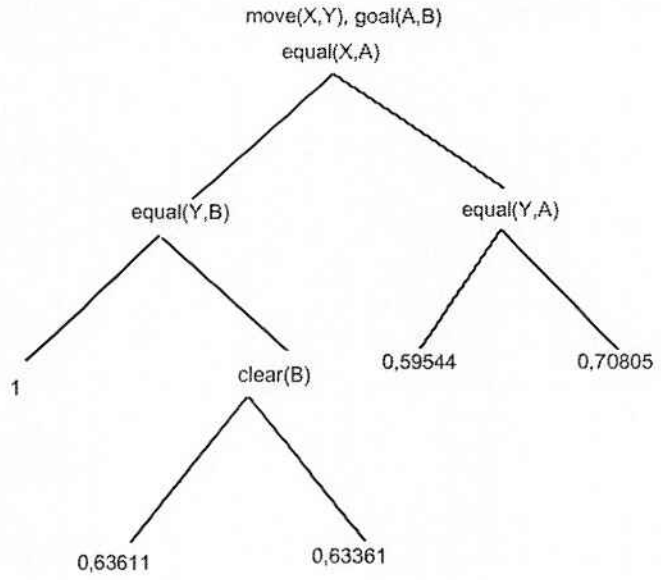

Figura 4.13: Respectiva árvore gerada após 1000 episódios de treino para o espaço variável com MSS=50.

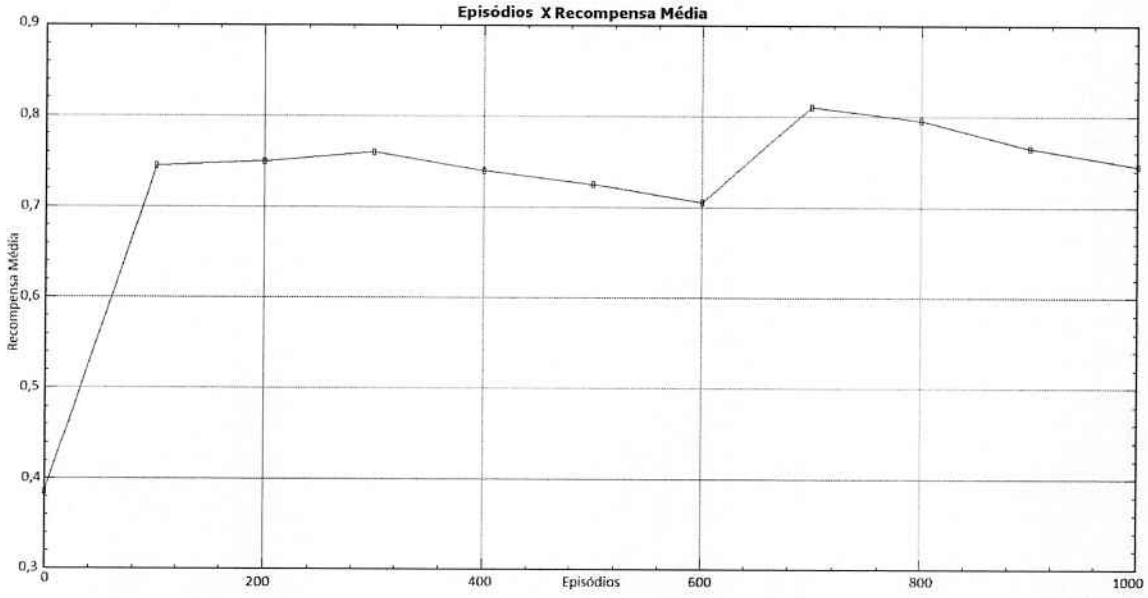

Figura 4.14: Curva de aprendizado para o espaço variável com MSS=100.

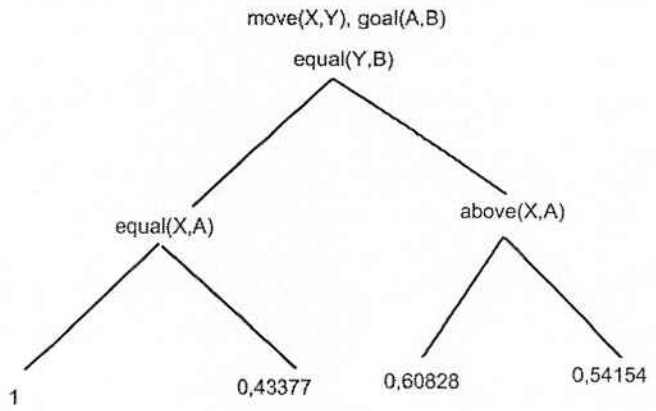

Figura 4.15: Respectiva árvore gerada após 1000 episódios de treino para o espaço variável com MSS=100.

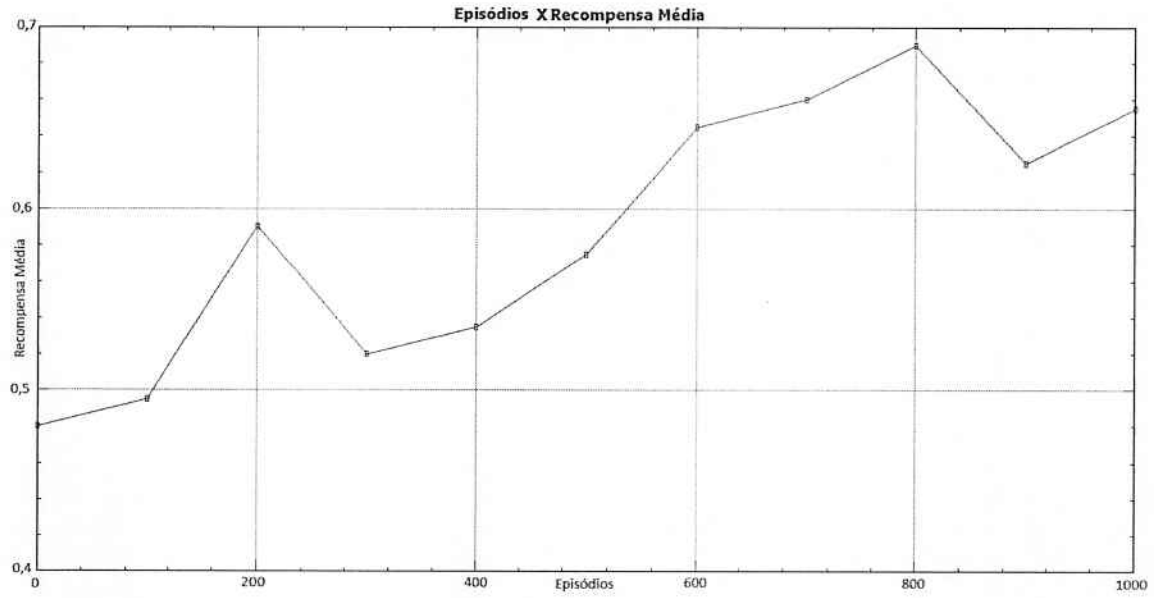

Figura 4.16: Curva de aprendizado para o espaço variável com MSS=200.

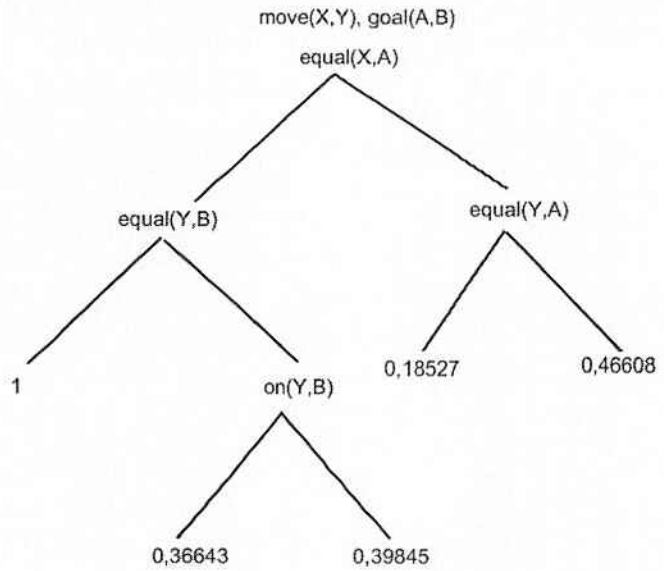

Figura 4.17: Respectiva árvore gerada após 1000 episódios de treino para o espaço variável com MSS=200.

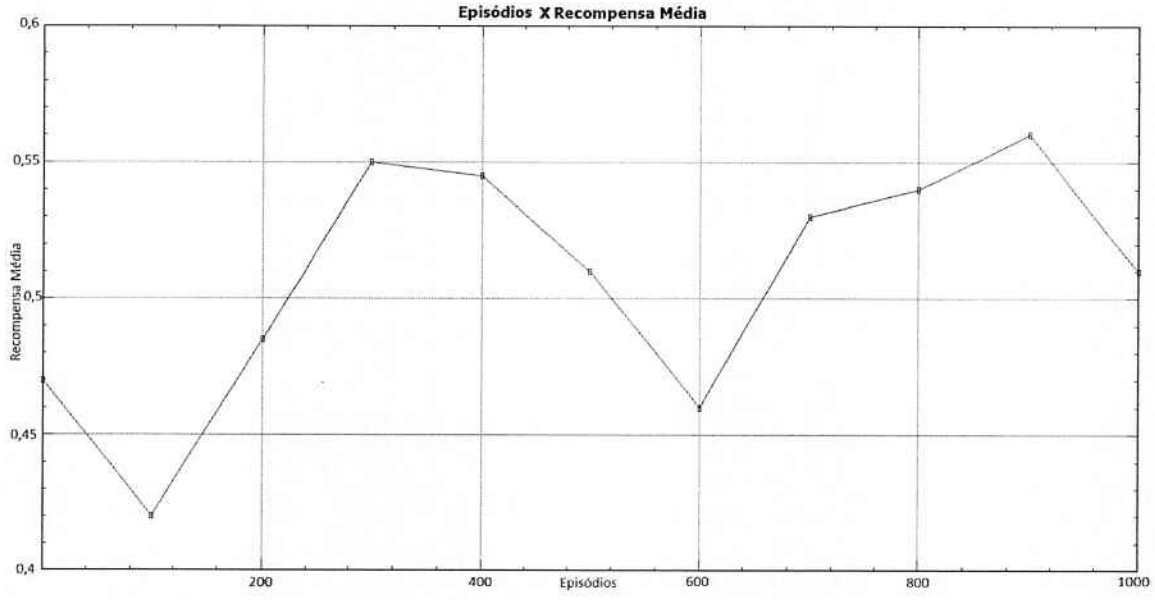

Figura 4.18: Curva de aprendizado para o espaço variável com MSS=500.

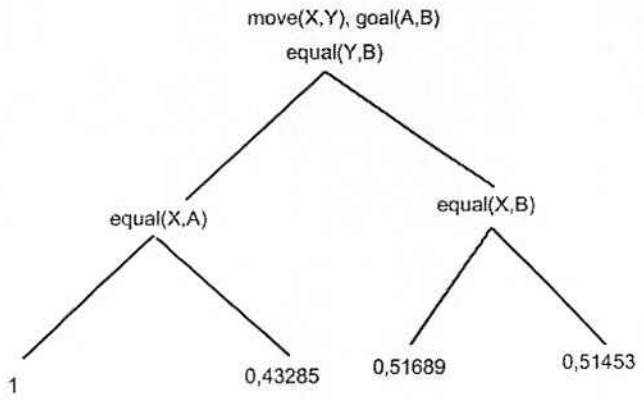

Figura 4.19: Respectiva árvore gerada após 1000 episódios de treino para o espaço variável com MSS=500.

5 Conclusões e Trabalhos Futuros

A escolha dos testes nos nós da árvore é altamente influenciada pelos exemplos fornecidos pelo sistema ARR. Devido à alta exploração no início do algoritmo, existe uma grande probabilidade de se fornecer exemplos que induzam à escolha de testes errados para a árvore. A figura 5.1 mostra o gráfico de um exemplo de aprendizado desastroso, e a respectiva árvore gerada se encontra na figura 5.2. No gráfico da figura 5.1 pode-se observar que a partir do 400^o episódio a recompensa média recebida decai a zero. Isso acontece porque a árvore gerada está incoerente, ao priorizar ações que na prática não minimizam a distância entre o estado e o estado objetivo (nos testes *equal(Y,B)* e *equal(X,A)* ao escolher ações que interagem com os blocos que não sejam nem A, nem B. Outro teste incoerente é *equal(Y,B)* e *on(X,B)*. O primeiro teste verifica se Y, o bloco destino de X, é o bloco B do objetivo. O segundo teste verifica se o bloco X, que está sendo movido, está sobre B. Se X está sobre B e está sendo movido para cima do bloco Y, que é o bloco B, então esta ação não faz muito sentido. Por outro lado, a verificação de que X não está sobre B está correta, e mover X para cima do bloco B é uma ação válida, mas devido ao baixo valor Q correspondente, é pouco atrativa. Isso acaba fazendo com que o sistema dê preferência por mover blocos não relacionados ao objetivo, causando assim a baixa recompensa média recebida.

Um problema encontrado no algoritmo TG é sua incapacidade de corrigir erros gerados no início da árvore (quando testes ruins são escolhidos). O algoritmo tenta corrigir esses erros construindo uma árvore maior, o que acaba atrasando o aprendizado do sistema. A escolha de testes é influenciada principalmente pelo MSS. Esse parâmetro pode ser um meio efetivo de se reduzir a probabilidade de se selecionar testes ruins no topo da árvore, mas pode também introduzir uma série de efeitos colaterais indesejáveis. Valores altos de MSS reduzem a velocidade de aprendizado do algoritmo TG, e consequentemente, o aprendizado do sistema ARR, principalmente nos estágios posteriores de aprendizado, devido à quantidade de dados coletados em cada folha. Valores baixos aceleram o aprendizado mas, como consequência, a árvore gerada é maior por gerar mais nós na tentativa

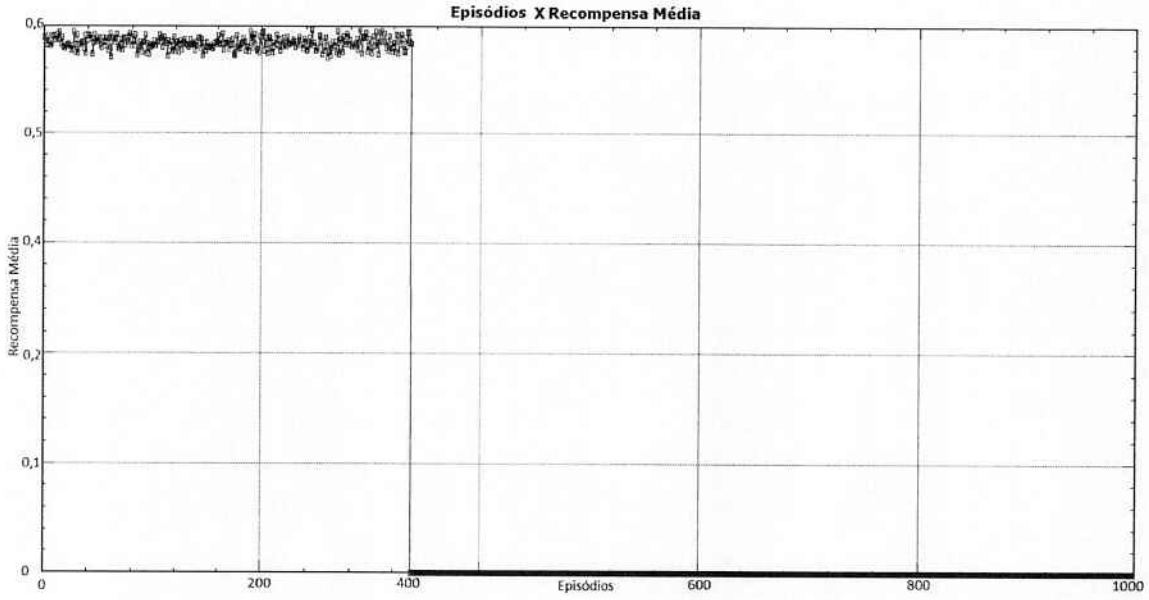

Figura 5.1: Curva de aprendizado para o espaço fixo com 3 blocos e MSS = 200.

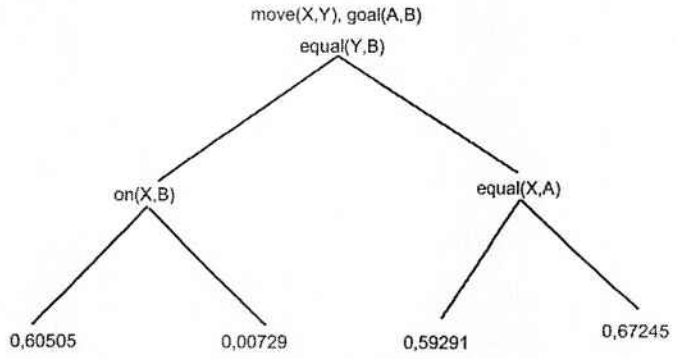

Figura 5.2: Respectiva árvore gerada para o espaço fixo com 3 blocos e MSS = 200.

de corrigir os erros iniciais.

Uma proposta de trabalho futuro seria implementar um MSS dinâmico, que seja reduzido a cada nível da árvore. Outra proposta seria permitir armazenar estatísticas sobre os testes em cada nó, não apenas nas folhas, e permitir que o algoritmo TG altere um teste escolhido quando ficar claro que um erro foi cometido.

Referências Bibliográficas

BLOCKEEL, H.; RAEDT, L. D. Top-down induction of logical decision trees. In: *Artificial Intelligence*. [S.l.: s.n.], 1998.

CHAPMAN, D.; KAELBLING, L. P. Input generalization in delayed reinforcement learning: an algorithm and performance comparisons. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, p. 726–731, 1991.

DRIESSENS, K. Relational reinforcement learning. In: *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course ACAI 2001 and Agent Links 3rd European Agent Systems Summer School (EASSS 2001), volume 2086 of Lecture Notes in Computer Science*. [S.l.: s.n.], 2004.

DRIESSENS, K.; RAMON, J.; BLOCKEEL, H. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In: *Proceedings of the 13th European Conference on Machine Learning*. [S.l.]: Springer-Verlag, 2001. p. 97–108.

DZEROSKI, S.; RAEDT, L. D.; BLOCKEEL, H. Relational reinforcement learning. In: *Machine Learning*. [S.l.]: Morgan Kaufmann, 1998. p. 7–52.

LAVRAC, N.; DZEROSKI, S. *Inductive Logic Programming*: Techniques and applications. New York: Ellis Horwood, 1994.

LAZARIC, A. Knowledge transfer in reinforcement learning. *Tese de Doutorado*, Politecnico di Milano, Milão, p. 18, 2008.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence*: A modern approach. 2. ed. New Jersey: Prentice Hall, 2003.

SILVA, R. R. da. Aprendizado por reforço relacional para o controle de robôs sociáveis. *Tese de Mestrado*, ICMC-USP, São Carlos, 2009.

SNEDECOR, G.; COCHRAN, W. *Statistical Methods*. 8. ed. [S.l.]: Wiley-Blackwell, 1991.

SUTTON, R.; BARTO, A. *Reinforcement Learning*: An introduction. Cambridge: The MIT Press, 1998.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. *Machine Learning*, v. 8, n. 3, p. 279–292, May 1992.

WIKIPéDIA. *Hefesto — Wikipédia*. Online; acessado em 16 nov. 2009. Disponível em: <http://pt.wikipedia.org/wiki/Hefesto>.

WIKIPéDIA. *Timeline — Wikipédia*: A brief history of artificial intelligence. Online; acessado em 16 nov. 2009. Disponível em: <http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/BriefHistory>.