Tiago Matos

Uso de política abstrata estocástica na navegação robótica

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Tiago Matos

Uso de política abstrata estocástica na navegação robótica

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Área de concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Anna Helena Reali
Costa

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor com a anuência de seu orientador.

São Paulo, 31 de Outubro de 2011.

Assinatura do autor

Assinatura do orientador

Ficha Catalográfica

Matos, Tiago

Uso de política abstrata estocástica na navegação robótica/Tiago Matos. – ed. rev. – São Paulo, 2011. 78 p. DVD-ROM em Anexo.

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

1. Inteligência artificial. 2. Robótica. 3. Aprendizado Computacional Relacional. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais. II. t.

Agradecimentos

Este trabalho foi realizado com bolsa de Mestrado no País, concedida pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) através do processo nº 2009/04489-9.

Resumo

A maioria das propostas de planejamento de rotas para robôs móveis não leva em conta a existência de soluções para problemas similares ao aprender a política para resolver um novo problema; e devido a isto, o problema de navegação corrente deve ser resolvido a partir do zero, o que pode ser excessivamente custoso em relação ao tempo. Neste trabalho é realizado o acoplamento do conhecimento prévio obtido de soluções similares, representado por uma política abstrata, a um processo de aprendizado por reforço. Além disto, este trabalho apresenta uma arquitetura para o aprendizado por reforço simultâneo, de nome ASAR, onde a política abstrata auxilia na inicialização da política para o problema concreto, e ambas as políticas são refinadas através da exploração. A fim de reduzir a perda de informação na construção da política abstrata é proposto um algoritmo, nomeado X-TILDE, que constrói uma política abstrata estocástica. A arquitetura proposta é comparada com um algoritmo de aprendizado padrão e os resultados demonstram que ela é eficaz em acelerar a construção da política para problemas práticos.

Abstract

Most work in path-planning approaches for mobile robots does not take into account existing solutions to similar problems when learning a policy to solve a new problem, and consequently solves the current navigation problem from scratch, what can be very time consuming. In this work we couple a prior knowledge obtained from a similar solution to a reinforcement learning process. The prior knowledge is represented by an abstract policy. In addition, this work presents a framework for simultaneous reinforcement learning called ASAR, where the abstract policy helps start up the policy for the concrete problem, and both policies are refined through exploration. For the construction of the abstract policy we propose an algorithm called X-TILDE, that builds a stochastic abstract policy, in order to reduce the loss of information. The proposed framework is compared with a default learning algorithm and the results show that it is effective in speeding up policy construction for practical problems.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

Lista de Símbolos

Lista de Algoritmos

1	Introdução	1
1.1	Objetivos	3
1.2	Justificativa	3
1.3	Contribuição	5
1.4	Organização	6
2	Conceitos Fundamentais	7
2.1	Conceitos Básicos de Lógica de Primeira Ordem	7
2.2	Processo Markoviano de Decisão	9
2.3	Extração de política utilizando aprendizado por reforço	11
2.4	Processo Markoviano de Decisão Relacional	14
2.5	Conclusão do capítulo	15
3	Modelando e abstraindo um problema de navegação robótica com RMDP	16
3.1	Modelando um problema de navegação	16
3.2	Abstraindo um problema de navegação	21

4	Abstração relacional em árvore de decisão	22
4.1	Indução de árvores de decisão	23
4.2	Indução de árvores de decisão em lógica de primeira ordem	24
5	Abstraindo políticas	30
5.1	Gerando uma política abstrata a partir da política concreta	33
5.2	Utilizando a política abstrata para guiar o robô	34
6	Problemas da utilização da abstração relacional	36
6.1	Instanciação inadequada	37
6.2	Abstração inadequada	39
7	Explorando a construção e uso de políticas abstratas na Navegação Robótica	42
7.1	Gerando uma política abstrata estocástica	42
7.2	Política abstrata no aprendizado por reforço	46
7.3	Aprendizado por reforço simultâneo	48
8	Experimentos	53
8.1	Avaliando a aplicação das políticas abstratas	54
8.1.1	Mesmo ambiente e mesmo estado meta	57
8.1.2	Mesmo ambiente e diferentes metas	58
8.1.3	Novo ambiente e novas metas	59
8.1.4	Análise dos resultados	60
8.2	Avaliando a aplicação das políticas abstratas no aprendizado por reforço	60
8.2.1	Aceleração do aprendizado através da utilização de políticas abstratas	61
8.2.2	Aceleração do aprendizado através do aprendizado simultâneo	63
9	Conclusões	68

9.1	Sugestões de trabalhos futuros	70
Referências		72
Apêndice A – Abstração de política do ambiente da Figura 3.1		75
A.1	Estados da Figura 3.1	75
A.2	Ações abstratas do problema	75
A.3	Operadores de Refinamento	75
A.4	Indução da árvore	75
Apêndice B – Parâmetros utilizados nos experimentos		78
B.1	Mapas	78
B.2	Algoritmos de aprendizado	78
B.2.1	Q-Learning	78
B.2.2	Q-Learning para o nível abstrato	78

Lista de Figuras

2.1	Exemplo de um ambiente com 6 estados.	10
2.2	Ciclo de interação do agente com o ambiente no aprendizado por reforço.	11
3.1	Um exemplo de uma representação relacional para um problema de navegação robótica.	17
4.1	Exemplo de uma árvore de decisão binária.	23
4.2	Exemplo de uma árvore de decisão representada em lógica de primeira ordem	25
4.3	Construção de uma árvore de decisão em lógica de primeira ordem pelo algoritmo TILDE.	29
5.1	Exemplo de um ambiente com 6 estados.	31
5.2	Exemplo de uma política abstrata, para o problema da Figura 5.1, representada por uma FOLDT.	32
5.3	Caminhos existentes na árvore da Figura 5.2 que definem os estados abstratos da política abstrata.	33
6.1	Situação exemplo para a demonstração de problemas que ocorrem durante a instanciação da política abstrata no problema de navegação robótica.	38
6.2	Problema exemplo para a demonstração da abstração inadequada, onde a meta é atingir a localidade 17.	40
7.1	FOLDT representando a política abstrata para o problema de navegação robótica	43
7.2	Geração de uma folha nos algoritmos TILDE e X-TILDE.	44
7.3	FOLDT induzida pelo algoritmo X-TILDE utilizando os mesmos exemplos usados na indução da árvore da Figura 7.1.	45

7.4	Processo de aprendizado da política ótima π^* pelo algoritmo <i>Q-Learning</i>	47
7.5	Processo de aprendizado da política ótima π^* pelo algoritmo <i>Q-Learning</i> tendo a exploração guiada pela política abstrata $\hat{\pi}_a$. . .	47
7.6	Arquitetura de aprendizado por reforço simultâneo ASAR.	50
8.1	Representação relacional para o problema utilizado na indução da política abstrata estocástica.	54
8.2	Política abstrata estocástica induzida pelo algoritmo X-TILDE para o problema da Figura 8.1.	54
8.3	Problema alvo utilizado nos experimentos.	55
8.4	Distribuição cumulativa para os experimentos no mesmo ambiente e com a mesma meta.	57
8.5	Distribuição cumulativa para os experimentos no mesmo ambiente e metas distintas.	58
8.6	Distribuição cumulativa para os experimentos em um novo ambiente e novas metas.	59
8.7	Número médio de transições NT necessárias para alcançar a meta, com e sem a utilização da política abstrata π_a	62
8.8	Média de passos para alcançar a meta. 11 é a localidade inicial para o robô e a meta para o problema é alcançar a localidade 125.	66
8.9	Média de passos para alcançar a meta. 14 é a localidade inicial para o robô e a meta para o problema é alcançar a localidade 125.	67

Lista de Tabelas

3.1	Estados existentes para o ambiente ilustrado na Figura 3.1.	19
3.2	Tipos de regiões existentes para o problema de navegação robótica, os predicados utilizados para representar as pré-condições de aplicabilidade da ação e a representação da região.	20
3.3	Predicados representando a transição entre os diversos tipos de região existentes no problema de navegação robótica.	20
4.1	Exemplo de operadores de refinamento para a navegação robótica.	27
4.2	Testes candidatos derivados dos operadores de refinamento da Tabela 4.1, com A e B como variáveis já existentes.	28
5.1	Política abstrata existente na árvore da Figura 5.3 em uma forma tabular.	32
5.2	Política ótima para o RMDP da Figura 5.1 tendo como meta 16. .	34
7.1	Conjunto de estados definidos para o problema exemplo da Figura 5.1.	52
7.2	Conjunto de estados abstratos derivados.	52
8.1	Probabilidades associadas para as ações abstratas nas folhas com mais de uma ação abstrata como política.	55
8.2	Uso da política abstrata: média, desvio padrão e quantidade de episódios com sucesso para o mesmo ambiente e estado meta. . . .	57
8.3	Uso da política abstrata: média, desvio padrão e quantidade de episódios com sucesso para o mesmo ambiente e metas diferentes.	58
8.4	Uso da política abstrata: média, desvio padrão e quantidade de episódios com sucesso para o novo ambiente e metas diferentes. . .	59
8.5	Estados abstratos definidos para a tabela Q-Abstrata do problema alvo.	65
A.1	Estados da Figura 3.1.	76

A.2	Ações Abstratas definidas para o problema da Figura 3.1.	77
A.3	Operadores de refinamento utilizados na construção das árvores dos experimentos.	77
B.1	Parâmetros utilizados nos experimentos para o aprendizado no nível concreto.	78
B.2	Parâmetros utilizados nos experimentos para o aprendizado no nível abstrato.	78

Lista de Abreviaturas e Siglas

MDP *Markov Decision Process*

RL *Reinforcement Learning*

FOLDT *First Order Logical Decision Tree*

RMDP *Relational Markov Decision Process*

TDIDT *Top-down Induction of Decision Trees*

LPO *Lógica de Primeira Ordem*

Lista de Símbolos

$\mathbf{c}_i$	constante qualquer utilizada para representar objetos ($i \in \mathbb{N}$)
$\forall$	quantificador universal
$\exists$	quantificador existencial
θ	substituição na lógica de primeira ordem
$\mathbf{V}_i$	variável qualquer ($i \in \mathbb{N}$)
$\subseteq$	é subconjunto próprio
$\preceq_\theta$	é subsumido
$\text{vars}(\mathbf{A})$	conjunto de variáveis da sentença complexa $\mathbf{A}$
$\mathcal{S}$	conjunto discreto de estados
$\mathcal{A}$	conjunto discreto de ações
σ_i	um estado qualquer ($i \in \mathbb{N}$)
$\mathcal{A}_\sigma$	conjunto discreto de ações do estado $\sigma \in \mathcal{S}$
α	uma ação
$\mathcal{T}$	função de transição
r	função recompensa
σ_t	o estado que o sistema se encontra no instante t
α_t	a ação tomada pelo agente no instante t
π	política
π^*	política ótima
R_t	função de recompensas futuras
r_t	recompensa recebida no instante t
γ	fator de desconto dos reforços futuros
$\mathcal{G}$	conjunto de estados meta
$\mathcal{D}$	domínio de planejamento probabilístico
$\mathbf{P}$	problema de planejamento probabilístico
α_{ij}	ação que representa a transição da localidade i para a localidade j
Q^π	função valor estado-ação
a	taxa de aprendizagem
ϵ	fator de exploração da técnica ϵ -Greedy
$\mathcal{P}$	conjunto de predicados
ρ_i	predicado $i \in \mathbb{N}$ utilizado na representação relacional dos estados
ϕ_i	a aridade associada ao predicado ρ_i

$\mathcal{C}$	conjunto de constantes
$\mathcal{A}'$	conjunto de átomos representando ações na forma relacional
κ_i	um predicado utilizado para representar uma ação
ι_i	a aridade associada ao predicado κ_i
H_{κ_i}	uma sentença complexa representando um estado
A_{κ_i}	um átomo representando o nome e os argumentos da ação κ_i
B_{κ_i}	uma sentença complexa contendo a pré-condição da ação κ_i
$\mathcal{S}'$	conjunto de sentenças complexas
p_{κ_i}	probabilidade de que a aplicação da ação κ_i seja sucedida
$\mathbf{S}$	conjunto de sentenças complexas representando estados
$\mathbf{A}$	conjunto de sentenças atômicas representando ações
p	probabilidade de sucesso de uma ação
$\hat{\sigma}$	estado abstrato
$\hat{\mathcal{S}}$	conjunto de estados abstratos
$\hat{a}$	ação abstrata
$\hat{\mathcal{A}}$	conjunto de ações abstratas
nm	número mínimo de exemplos necessários para a divisão de um nó
$\mathcal{X}$	conjunto de variáveis
$\mathcal{BK}$	conjunto de cláusulas de Horn
$\mathcal{S}_{\hat{\sigma}}$	conjunto de estados concretos cobertos pelo estado abstrato $\hat{\sigma}$
$\mathcal{A}_{\hat{a}}$	conjunto de ações concretas cobertas por $\hat{a}$
Γ	conjunto de tarefas
$\hat{\pi}_a$	política abstrata
π_a	política concreta induzida pela política abstrata $\hat{\pi}_a$
π_r	política aleatória
ϵ_1	fator de exploração
ϵ_2	fator de escolha entre a política abstrata $\hat{\pi}_a$ e a política aleatória π_r
Υ	conjunto de todos os possíveis históricos de transição
$\hat{\pi}_{a_s}$	política abstrata induzida através do problema fonte
$\hat{\pi}_{a_t}$	política aprendida no nível abstrato através do problema alvo
ϵ_a	fator de escolha entre a política abstrata $\hat{\pi}_{a_s}$ e a política abstrata $\hat{\pi}_{a_t}$

Lista de Algoritmos

Algoritmo <i>Q-Learning</i>	12
Algoritmo TDIDT	23
Algoritmo TILDE	25
Algoritmo X-TILDE	45

1 Introdução

Neste trabalho é considerado a seguinte questão geral. Um agente define uma política de atuação que resolve um problema de decisão particular, como por exemplo, uma política que leva um agente robótico de um determinado lugar até um lugar alvo em um determinado edifício. O agente agora se depara com um problema similar: talvez o robô esteja agora num edifício adjacente e similar ao qual ele aprendeu a política; ou talvez um investidor do mercado de ações esteja se deparando com uma situação nas cotações dos ativos que se assemelha com situações que já ocorreram no passado. A intuição que temos é que a política aprendida em um problema pode ser utilizada na resolução de problemas similares; e é esta intuição que utilizamos para tomar a maioria das decisões em nossas vidas. Tendemos a comprar produtos de uma determinada marca utilizando como referência produtos anteriormente comprados da mesma marca; a interagir de um determinado modo com uma pessoa utilizando as experiências de convivência com ela; ou novamente no caso do investidor do mercado de ações que tende a fazer seu investimento utilizando como ferramenta de apoio a análise técnica.

É nesta intuição a respeito do reaproveitamento do aprendizado que inserimos este trabalho. Exploramos a intuição de que a generalização de problemas intimamente relacionados, mas resolvidos, podem produzir políticas que fazem uma boa decisão em muitos dos estados de um novo e não resolvido problema. Especificamente tratamos do reaproveitamento do aprendizado no domínio da navegação robótica.

Diversas soluções foram propostas visando a generalização do domínio do problema e a reutilização do conhecimento adquirido na resolução de tarefas similares.

Algumas propostas utilizam funções heurísticas para selecionar ações, assim como para guiar o processo de exploração no espaço de estados (BIANCHI; RIBEIRO; COSTA, 2007; BURKOV; CHAIB-DRAA, 2007; BIANCHI; RIBEIRO; COSTA, 2008; KNOX; STONE, 2010). A utilização de uma função heurística tem como objetivo reduzir a aleatoriedade na exploração durante o processo de aprendizado

e com isto reduzir o tempo de aprendizado da política ótima. A heurística pode ser derivada a partir da estimativa do modelo do ambiente (BIANCHI; RIBEIRO; COSTA, 2008).

Outras propostas tentam agregar estados similares em macroestados e utilizar macroações para transitar entre os macroestados; macroestados e macroações são então utilizados para resolver tarefas similares (DRUMMOND, 2002; SILVA; COSTA, 2009). A partir da resolução um problema fonte, os estados deste podem ser agrupados, em relação a uma função valor, em regiões (ou macroestados). Estes macroestados são então utilizados como exemplos para a resolução de um novo problema. Cada região identificada no novo problema vai ter os valores de seus estados atualizados utilizando os valores presentes em algum dos macroestados extraídos do problema fonte (DRUMMOND, 2002). A utilização dos macroestados visa portanto reduzir o tempo de aprendizado da política ótima atualizando a política para grupos de estados de um problema.

Ainda existem autores que propuseram dividir o problema de decisão em subtarefas, e a partir disto achar políticas que resolvam estas subtarefas. Estas políticas representadas numa forma abstrata são então utilizadas para resolver problemas similares, em conjunto com uma representação hierárquica dos estados agregados do problema (UTHER; VELOSO, 2002).

Há também trabalhos visando representar estados abstratos e ações na forma relacional, que é o caminho seguido neste trabalho. Representações relacionais facilitam a formulação de uma ampla coleção de tarefas relacionadas como um único domínio, possuindo com isto uma abstração natural entre estas tarefas relacionadas (KERSTING et al., 2007; DZEROSKI; RAEDT; DRIESSENS, 2001). Taylor e Stone (2009) fornecem uma vasta pesquisa na transferência do aprendizado utilizando descrições proposicionais dos estados. Cocora et al. (2006) propuseram aprender árvore de decisão relacional como estratégias abstratas de navegação a partir de exemplos de caminhos, e em vista disto uma política de navegação aprendida em um ambiente era diretamente aplicada em ambientes desconhecidos. Neste caso, as estratégias abstratas de navegação eram utilizadas diretamente para guiar o agente robótico na resolução de um novo problema, sendo que não era garantido que a meta do novo problema fosse atingida com desempenho ótimo, i. e., na menor distância possível. Baseado em exemplos experimentados pelo agente quando agia otimamente num problema de decisão, em (MADDEN; HOWLEY, 2004) aprendizes simbólicos proposicionais generalizavam a política ótima para outros problemas. Em (SHERSTOV; STONE, 2005) a relevância da ação é utilizada para reduzir o conjunto de ações a ser explorado no problema alvo. Por consequência

da redução do conjunto de ações a ser explorado, o aprendizado da política ótima do problema alvo é acelerado.

As propostas que representam estados e ações abstratos na forma relacional têm se mostrado muito promissoras, pois elas permitem explorar os aspectos estruturais do ambiente e permitem transferir o conhecimento em decorrência dos relacionamentos entre os objetos.

É através da representação relacional que pretendemos explorar a intuição do reaproveitamento do conhecimento. Acreditamos que, por representar o ambiente em função de objetos e relações entre eles, a representação relacional seja a técnica mais adequada para mapear semelhanças entre situações e explorar esta intuição de que situações semelhantes estruturalmente possuam soluções também semelhantes.

1.1 Objetivos

O objetivo principal do trabalho é:

- Apresentar uma arquitetura de aprendizado que utilize o aprendizado por reforço aplicado em um problema concreto em conjunto com uma política abstrata. Nesta arquitetura híbrida é aprendida a política ótima tanto no nível concreto quanto no nível abstrato do problema. Esta arquitetura tem como principal objetivo acelerar o processo de aprendizado de um problema concreto futuro.

Além disso, um objetivo secundário foi derivado, que é:

- Estender um algoritmo de abstração relacional para reduzir a perda de informação gerada no processo de geração da política abstrata. Com esta extensão é gerada uma política abstrata estocástica.

1.2 Justificativa

Em relação à descrição dos estados em lógica de primeira ordem, poucos exemplos são achados na literatura para o domínio da navegação robótica. Para conseguir descrever completamente a situação relacional na qual o agente se encontra, deve-se mapear completamente o ambiente. As propostas existentes (COCORA et al., 2006; LANE; WILSON, 2005) para a navegação robótica devem confiar apenas

em percepções locais e ambientes estruturados, onde políticas abstratas (quase) ótimas podem ser definidas. Em vista disto, é de grande importância o aprofundamento dos estudos em problemas de navegação robótica onde as políticas abstratas não são capazes de representar perfeitamente as políticas abstratas concretas.

Apesar da abstração relacional ter se apresentado como uma técnica promissora para o reaproveitamento de políticas, existem deficiências na política abstrata gerada. Entre essas deficiências está a baixa representatividade da política abstrata, gerada por um algoritmo de abstração relacional que não consegue diferenciar situações semelhantes estruturalmente, mas com políticas diferentes. Devido a este fato, estados que representam locais com a mesma semelhança estruturais (conectados com os mesmos tipos de localidades) vão ser representados pelo mesmo estado abstrato e por consequência vão receber a mesma política abstrata, mesmo que nessa situação os estados instanciados originais tivessem políticas totalmente diferentes. Em vista disto, é de grande interesse a proposta de técnicas que possam contornar estes problemas de representatividade da política abstrata. Essas técnicas podem ser tanto uma alteração dos algoritmos de abstração já existentes quanto a proposta de novos algoritmos de abstração relacional. No caso, optamos por propor uma alteração em um algoritmo de abstração já existente na literatura.

Por fim, vale ressaltar que o aprendizado por reforço fornece uma abordagem geral para o desenvolvimento de agentes que são capazes de otimizar o seu desempenho em ambientes estocásticos, em função de sua própria experiência. Entretanto, grande parte dos trabalhos em aprendizado por reforço não leva em conta, no processo de aprendizado de uma política para resolver um problema, soluções existentes para problemas similares. Por consequência disto, todo novo problema deve ser resolvido pelo aprendizado por reforço a partir do zero, o que pode ser muito demorado. Devido a isto, é de grande valor para a teoria do aprendizado por reforço a proposta de técnicas com o intuito de acelerar o processo de aprendizado de uma política ótima, utilizando conhecimento previamente adquirido.

Em vista da relevância aqui justificada dos termos tratados, nosso trabalho oferece contribuições importantes para a solução dos problemas ressaltados.

1.3 Contribuição

Existem três principais contribuições deste trabalho:

1. A primeira é a extensão de um algoritmo de abstração relacional. O algoritmo TILDE (BLOCKHEEL; RAEDT, 1998) constrói uma FOLDT (*First order logical decision tree*) que representa uma política abstrata, sendo que para cada estado abstrato é indicado apenas uma ação abstrata como política. Com a extensão proposta, a política para cada estado abstrato é dada por um conjunto de ações abstratas, tornando-a uma política estocástica. Esta política estocástica visa reduzir a perda de representatividade da política abstrata representada por uma FOLDT.
2. A segunda contribuição diz respeito ao acoplamento do conhecimento prévio adquirido, expresso por meio de uma política abstrata, a um processo de aprendizado por reforço, de modo a possibilitar o reaproveitamento do conhecimento prévio em uma nova tarefa de aprendizagem.
3. A terceira contribuição é a proposta de uma arquitetura híbrida de aprendizado onde a política para um problema é aprendida tanto no nível concreto quanto no nível abstrato, estendendo a contribuição anterior. Desta forma, o conhecimento expresso pela política abstrata adquirida na execução de um problema fonte pode ser aperfeiçoada com a experiência atual decorrente da solução do novo problema. Assim como ocorre no processo de utilização da política abstrata para guiar a exploração do ambiente no processo de aprendizado, a utilização da arquitetura híbrida acelera o processo de aprendizado da política concreta para o novo problema. A vantagem desta arquitetura é que as imperfeições que podem ocorrer no processo de abstração são contornadas pelo aprendizado concorrente da política abstrata, pois neste caso a política abstrata que está sendo aprendida é específica para o problema que está sendo resolvido. Como imperfeições entende-se as situações que não eram previstas no problema fonte utilizado para a indução da política abstrata. Neste caso, o nível abstrato da arquitetura vai combinar a política abstrata derivada do problema fonte com a política abstrata que está sendo aprendida no novo problema, de modo que este nível abstrato auxilie e agilize o aprendizado da política no nível concreto do problema.

1.4 Organização

O trabalho inicia com a apresentação dos conceitos envolvidos com a lógica de primeira ordem (Capítulo 2), técnica esta utilizada para representação dos estados, ações e políticas abstratas aprendidas. Após isto, são introduzidos os conceitos básicos envolvidos com Processos Markovianos de Decisão (MDP ou *Markov Decision Processes*) e Processos Markovianos de Decisão Relacional (RMDP ou *Relational Markov Decision Processes*). No Capítulo 3 é apresentado como um problema de navegação robótica móvel pode ser modelado por um RMDP.

O Capítulo 4 mostra como uma árvore de decisão representada em lógica de primeira ordem (FOLDT ou *First order logical decision tree*) pode ser utilizada para a abstração de políticas; e o Capítulo 5 apresenta como uma política abstrata pode ser induzida a partir de uma política concreta.

No Capítulo 6 são apresentados os problemas que podem surgir ao se utilizar a política abstrata para guiar a navegação robótica. Em vista destes problemas, o Capítulo 7 apresenta as propostas deste trabalho.

Os experimentos são apresentados no Capítulo 8, sendo eles divididos em três partes: a primeira parte (Seção 8.1) analisa o desempenho da extensão do algoritmo de abstração; a segunda parte (Seção 8.2.1) analisa a aceleração do aprendizado através da utilização da política abstrata; e a terceira (Seção 8.2.2) analisa o desempenho da arquitetura proposta neste trabalho, de nome ASAR, para acelerar o aprendizado.

2 Conceitos Fundamentais

Neste Capítulo são apresentados os conceitos fundamentais para o entendimento do presente trabalho. Inicialmente é realizada uma breve apresentação da Lógica de Primeira Ordem (LPO). Após isto, são apresentados o Processo Markoviano de Decisão (*Markov Decision Process* ou MDP) (PUTERMAN, 1994) e o aprendizado por reforço (*Reinforcement Learning* ou RL) (SUTTON; BARTO, 1998). O MDP pode ser utilizado para modelar problemas de decisão e o RL é uma técnica utilizada para a resolução de MDPs. Finalmente, na última parte deste Capítulo é apresentado como a LPO e o MDP podem ser utilizados para representar problemas de decisão. É dado o nome de Processo de Decisão Markoviano Relacional (*Relational Markov Decision Process* ou RMDP) (OTTERLO, 2009) para a união destas duas técnicas.

2.1 Conceitos Básicos de Lógica de Primeira Ordem

A lógica de primeira ordem (RUSSELL; NORVIG, 2003) visa utilizar objetos, relações e funções na representação do conhecimento. É suposto que o mundo a ser representado consiste de objetos com certas relações entre eles que podem ser ou não válidas. Os objetos constituem o domínio do modelo para a lógica de primeira ordem. Uma relação é um conjunto de tuplas de objetos inter-relacionados. Função é um tipo de relacionamento onde um único objeto é relacionado à entrada da função.

Símbolos são utilizados para representar objetos, funções e relações. Existem três tipos de símbolos na lógica de primeira ordem: símbolos de constantes, símbolos de predicados e símbolos de funções. Todos os símbolos são definidos por palavras minúsculas. Constantes ($c_1, c_2, c_3, \dots, c_n$) e funções são utilizadas para representar objetos. Predicados são utilizados para representar relações, sendo que deve existir uma aridade associada que fixa o número de argumentos. Por exemplo, o predicado `conn/2` pode simbolizar a relação de conexão entre duas

localidades (argumentos do predicado).

Termos são definidos como expressões lógicas que se referem a objetos e podem ser de três tipos: constantes, variáveis e funções. Variáveis são representadas por palavras maiúsculas. Termo básico é aquele que não possui variáveis.

Uma sentença atômica é formada a partir de um símbolo de predicado seguido por uma lista de termos entre parênteses. Elas podem ser reunidas através de conectivos lógicos para construir sentenças complexas. Por exemplo, utilizando uma representação relacional para o problema de navegação robótica e definindo as seguintes sentenças atômicas: $\text{in}(A)$ representa que o robô está na localidade A ; $\text{isCorridor}(A)$ simboliza que a localidade A é um corredor; $\text{isRoom}(B)$ simboliza que a localidade B é um cômodo; e $\text{conn}(A,B)$ simboliza que as localidades A e B são conexas. Estas sentenças atômicas podem ser reunidas pelo conectivo lógico $\wedge$ para construir a sentença $\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isCorridor}(A) \wedge \text{isRoom}(B)$ que simboliza todas as situações nas quais o agente se encontra em um corredor (representado pela variável A) conectado com um cômodo (representado pela variável B).

Relações envolvendo coleções de objetos podem ser expressas através da utilização de quantificadores. Os quantificadores existentes são o universal e o existencial. O quantificador universal ($\forall$) é utilizado para representação de regras gerais para uma coleção de objetos. O quantificador existencial ($\exists$) é utilizado para representar regras específicas sobre um elemento da coleção.

Para uma sentença lógica, uma substituição $\theta = \{V_1/t_1, V_2/t_2, \dots, V_n/t_n\}$ corresponde a um conjunto de pares **variável/termo** utilizados para a troca dos termos variáveis desta sentença por termos constantes ou outras variáveis. Por exemplo, utilizando a substituição $\theta = \{A/c1, B/c2\}$ na sentença complexa $\text{in}(A) \wedge \text{conn}(A,B)$ obtemos a sentença complexa $\text{in}(c1) \wedge \text{conn}(c1, c2)$. A sentença lógica gerada passa a ter apenas constantes como termos, especificando situações concretas.

Se A e B forem sentenças e existir uma substituição θ para a qual $B\theta \subseteq A$, é dito que A é θ -subsumida por B , denotado por $A \preceq_{\theta} B$.

O conjunto de variáveis de uma sentença complexa A é definido por $\text{vars}(A)$.

Estes conceitos serão usados na definição de um modelo de descrição relacional para problemas estocásticos (Seção 2.4). Para isso, é necessário a introdução de um modelo poderoso de descrição de problemas de decisão sequencial, chamado Processo Markoviano de Decisão.

2.2 Processo Markoviano de Decisão

É assumido que um problema de decisão pode ser modelado como um Processo Markoviano de Decisão (*Markov Decision Processes* ou MDP) (PUTERMAN, 1994). A cada passo (discreto) de tempo um agente observa o estado do sistema, escolhe uma ação que leva o sistema para um novo estado, retornando com isto uma recompensa para o agente. Para a finalidade deste trabalho um MDP pode ser definido por uma tupla $\langle \mathcal{S}, \mathcal{A}, T, r \rangle$, onde:

- $\mathcal{S}$ é um conjunto discreto de *estados*;
- $\mathcal{A} = \bigcup_{\sigma \in \mathcal{S}} \mathcal{A}_\sigma$ é um conjunto discreto de *ações*, onde $\mathcal{A}_\sigma$ é o conjunto de ações válidas no estado $\sigma \in \mathcal{S}$;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ é uma *função de transição* tal que $T(\sigma, \alpha, \sigma') = \mathbb{P}(\sigma_{t+1} = \sigma' | \sigma_t = \sigma, \alpha_t = \alpha)$, com $\sigma, \sigma' \in \mathcal{S}$ e $\alpha \in \mathcal{A}_\sigma$.
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ é uma *função recompensa*, tal que $r_{t+1} = r(\sigma, \alpha, \sigma')$ é a recompensa recebida ao atingir o estado σ' no passo de tempo $t + 1$, sendo escolhida anteriormente a ação α no estado σ , com $\sigma, \sigma' \in \mathcal{S}$ e $\alpha \in \mathcal{A}_\sigma$.

A tarefa do agente é achar uma *política*. Uma política π é uma função de $\mathcal{S}$ no conjunto potência de $\mathcal{A}$; ou seja, um estado é mapeado num *conjunto* de ações: $\pi : \mathcal{S} \mapsto 2^{\mathcal{A}}$. Se acaso a política mapear cada estado num *singleton* (ou seja, numa única ação), é dito que a política é *determinista*. Caso se deseje enfatizar que a política mapeia alguns estados num conjunto contendo mais de uma ação, é dito que a política é *não determinista*. Sempre que for permitido, o termo *política* será utilizado para se referir tanto a uma política determinista quanto não determinista. Uma *política determinista ótima* π^* (ou, de uma forma mais compacta, uma política ótima) é uma política determinista que maximiza uma função R_t de recompensas futuras $r_{t+1}, r_{t+2}, \dots$. Uma definição comum, que será utilizada aqui, é considerar a soma das *recompensas descontadas* $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, onde $\gamma \in [0, 1[$ é o fator de desconto.

É de particular interesse a subclasse de MDPs que são *episódicos*: existe um conjunto $\mathcal{G}$ de um ou mais estados meta, e quando a meta é atingida o problema é reiniciado em algum estado inicial escolhido de acordo com uma distribuição de probabilidade. Uma sequência de passos que leva o robô do estado inicial até a meta é um episódio. Um *domínio de planejamento probabilístico* é definido (PEREIRA; BARROS; COZMAN, 2008) por uma tupla $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, T \rangle$, e um *problema de planejamento probabilístico* como a tupla $\mathbf{P} = \langle \mathcal{D}, \mathcal{G} \rangle$.

Figura 2.1: Exemplo de um ambiente com 6 estados.

Considere o ambiente retratado na Figura 2.1. Cada forma geométrica é uma localidade do ambiente que o agente pode ocupar; e se duas localizações estão conectadas por uma aresta significa que o agente pode alcançar uma localidade através da outra. O estado do ambiente é representado pela localização do agente e por propriedades desta localização. Definindo $\mathcal{S} = \{\sigma_1, \dots, \sigma_6\}$, onde, por exemplo o σ_2 representa a configuração no qual o agente se encontra no triângulo com o número 2 (local 12). O conjunto de ações é $\mathcal{A} = \bigcup_{i=1}^6 A_{\sigma_i} = \bigcup_{i=1}^6 \{\alpha_{ij}, j \in \{1, \dots, 6\}\}$, onde por exemplo α_{12} significa que o agente escolheu ir para o triângulo 2 quando estava no local 11. Observe que o agente pode aplicar uma ação que continue no mesmo estado; este tipo de ação é denotada por α_{ii} . Assuma por simplicidade que o ambiente é determinista, portanto $T(\sigma_i, \alpha, \sigma_j)$ é igual a 1 se $\alpha = \alpha_{ij}$ e zero, no outro caso. Deste modo o domínio de planejamento $\mathcal{D}$ é completamente especificado. Agora suponha que se deseja aprender como chegar no estado σ_6 . Podemos definir $r(\sigma, \alpha, \sigma')$ para ser igual a 0 se $\sigma = \sigma' = \sigma_6$ e igual a -1 caso contrário; com a recompensa definida deste modo, o agente irá maximizar R_t se ele alcançar σ_6 na menor quantidade de passos e ficará neste estado após atingi-lo. Assim, a função recompensa implicitamente define $\mathcal{G} = \{\sigma_6\}$, e o problema de planejamento $\mathbf{P}$ é também especificado. Devido ao fato de que σ_2 possui duas ações que são ótimas¹, existem diversas possibilidades para a política ótima. Existe por exemplo $\pi^*(\sigma_2) = \alpha_{23}$ ou $\pi^*(\sigma_2) = \alpha_{24}$. Estas possibilidades podem ser codificadas em uma política não determinista como $\pi^*(\sigma_2) = \{\alpha_{23}, \alpha_{24}\}$. Para determinar uma ação particular a partir desta política não determinista, pode-se utilizar a aleatoriedade; por exemplo, pode-se escolher α_{23} ou α_{24} com probabilidade igual 1/2 para cada uma.

Dado um MDP, existe uma política determinista que maximiza as recompensas descontadas (PUTERMAN, 1994); existem diversos métodos para achar a política a partir de um MDP. Quando o MDP não é completamente especificado (ou seja, T não é conhecida), pode-se recorrer ao aprendizado por reforço

¹O local 16 pode ser atingido, a partir de 12, por meio de 13 ou de 14.

(*Reinforcement Learning* ou RL) (SUTTON; BARTO, 1998). Utilizando esta arquitetura é comum se definir uma *função valor estado-ação* $Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ onde para uma dada política π fornece uma estimativa para o valor esperado $\mathbb{E}_\pi[R_t | \sigma_t = \sigma, \alpha_t = \alpha]$. Esta função é aprendida diretamente pela interação com o ambiente: a cada passo de tempo o agente executa uma ação, observa o resultado e utiliza uma estratégia pré-especificada para atualizar as estimativas da função Q . A partir de uma boa estimativa da função Q , o agente pode maximizar R_t escolhendo $\alpha_t = \arg \max_\alpha Q(\sigma, \alpha)$.

A Seção 2.3 a seguir detalha como a função Q pode ser estimada através da interação com o ambiente e como a política pode ser extraída a partir desta função.

2.3 Extração de política utilizando aprendizado por reforço

O aprendizado por reforço (*Reinforcement Learning* ou RL) (SUTTON; BARTO, 1998) é uma técnica para a extração da política ótima que considera a não existência a priori do modelo completo do MDP. Essa consideração visa estender a aplicação da modelagem por MDPs para problemas onde não é possível de se ter de início a função de transição de estados T . Esse é um fato corrente para muitos dos problemas modelados por MDPs. No RL o aprendizado dessas funções ocorre através de ciclos de interação com o ambiente (ver Figura 2.2). Essas interações estão separadas entre si por intervalos de tempo discreto.

Figura 2.2: Ciclo de interação do agente com o ambiente no aprendizado por reforço. A cada instante t o agente, que se encontra no estado σ_t interage com o ambiente aplicando a ação α_t , transita para o estado σ_{t+1} e recebe uma recompensa $r(\sigma_t, \alpha_t, \sigma_{t+1})$.

A cada etapa t de interação com o ambiente o agente observa o estado σ_t em que se encontra. Para este estado o agente possui associado um conjunto $\mathcal{A}_\sigma$ de

ações que ele pode aplicar. Uma ação deste grupo $\alpha \in \mathcal{A}_\sigma$ é escolhida seguindo uma política de escolha que balanceia a exploração e exploração do ambiente. Se acaso a exploração do ambiente for escolhida, o agente deve escolher uma ação para o estado corrente através da política atualmente existente. A política existente é extraída da aproximação atual de uma função valor que quantifica o valor da situação do agente em frente à solução do problema. Caso a exploração seja escolhida, o agente seleciona uma ação qualquer do conjunto de ações disponíveis seguindo uma regra pré-determinada.

O objetivo do agente é aprender, através de sucessivas interações com o ambiente, a política ótima para a escolha de ações que atinja o seu objetivo, sendo esta política representada por uma função valor. A seguir é apresentado o algoritmo *Q-Learning* que extrai a política ótima π^* através do aprendizado da função valor estado-ação Q .

Algoritmo Q-Learning

Pelo motivo que no aprendizado por reforço é considerado que não existe o modelo completo do ambiente, a política ótima não pode ser obtida diretamente do MDP (por alguma das técnicas da programação dinâmica). Em vista disto, no algoritmo *Q-Learning* a política ótima π^* é extraída através da função Q^* , com $\pi^*(\sigma) = \arg \max_{\alpha} Q^*(\sigma, \alpha)$. Para cada estado σ e ação α do MDP, o valor de $Q(\sigma, \alpha)$ representa o valor acumulado da máxima recompensa descontada que pode ser obtida iniciando no estado σ , i. e., é a soma da recompensa obtida ao aplicar a ação α no estado σ com as recompensas futuras (descontadas por um fator γ) obtidas seguindo a política ótima.

Uma estimativa $\hat{Q}$ para a função ótima Q^* pode ser aprendida iterativamente. A cada instante t a estimativa $\hat{Q}_t(\sigma_t, \alpha_t)$ é atualizada conforme a equação 2.1, sendo que esta atualização é realizada por um fator $a \in [0, 1]$. O fator a é conhecido como taxa de aprendizagem. A atualização do valor de $\hat{Q}_t(\sigma_t, \alpha_t)$ é dado pela soma da recompensa $r(\sigma_t, \alpha_t, \sigma_{t+1})$ recebida no instante t com a máxima recompensa futura descontada, $\max \hat{Q}_{t-1}(\sigma_{t+1}, \alpha')$, do estado atingido σ_{t+1} , onde $\gamma \in [0, 1[$ é o fator de desconto de reforços futuros:

$$\hat{Q}_t(\sigma_t, \alpha_t) \leftarrow (1 - a) \cdot \hat{Q}_{t-1}(\sigma_t, \alpha_t) + a \cdot \left[r(\sigma_t, \alpha_t, \sigma_{t+1}) + \gamma \max \hat{Q}_{t-1}(\sigma_{t+1}, \alpha') \right]. \quad (2.1)$$

A máxima recompensa futura descontada é dada em função da estimativa

atual dos $Q(\sigma_{t+1}, \alpha'), \alpha' \in \mathcal{A}_{\sigma_{t+1}}$ para o estado σ_{t+1} atingido pela aplicação da ação α_t no estado σ_t .

Para o aprendizado da função $\hat{Q}$ é necessário que o agente percorra todo o espaço de estados do problema. Isso deve ser feito para a correta convergência da função $\hat{Q}$. Existem diversas técnicas para serem utilizadas no aprendizado por reforço para o balanceamento da exploração e exploração do ambiente. A técnica ϵ -Greedy (SUTTON; BARTO, 1998) balanceia a exploração/exploração probabilisticamente através de um fator $\epsilon \in [0, 1]$. Para um determinado estado σ , o agente executa a política α informada pela função Q , com $\pi(\sigma) = \arg \max_{\alpha} \hat{Q}(\sigma, \alpha)$, com probabilidade $1 - \epsilon$ (exploração) e escolhe aleatoriamente uma das ações disponíveis $\alpha \in A_{\sigma}$ com probabilidade ϵ (exploração).

O algoritmo *Q-Learning* é apresentado no Algoritmo 1. Inicialmente a função $\hat{Q}$ é inicializada com valores arbitrários para cada um dos pares de estado-ação (passo 2). Para a convergência da função $\hat{Q}$ é realizado um conjunto de episódios (passo 3). Em cada episódio de aprendizado o sistema é iniciado num estado inicial arbitrário (passo 4). Dentro de cada episódio o agente executa passos de interação com o ambiente até atingir o estado meta (passo 5). Para cada passo do episódio é escolhida a ação a ser tomada pelo agente, sendo que essa escolha é decidida pelo política ϵ -Greedy (passo 6). Com probabilidade $1 - \epsilon$ a ação vai ser dada pela política extraída da função Q , isto é $\pi(\sigma) = \arg \max_{\alpha} Q(\sigma, \alpha)$, e com probabilidade ϵ vai ser escolhida aleatoriamente uma ação $\alpha \in A_{\sigma}$. A ação α escolhida é aplicada (passo 7), o agente observa o estado σ_{t+1} atingido e a recompensa $r(\sigma_t, \alpha_t, \sigma_{t+1})$ obtida e, em função disto, atualiza o valor de $\hat{Q}_t(\sigma_t, \alpha_t)$ (passo 8). Um episódio é finalizado quando o estado meta for atingido (passo 10).

Algoritmo 1 Algoritmo Q-Learning

- 1: **function** Q-Learning:
 - 2: $\forall \sigma \in \mathcal{S}$ inicializa $\hat{Q}$ com valores arbitrários (ex. $\hat{Q}(\sigma, \alpha) = 0$)
 - 3: **Repeat** (para cada episódio)
 - 4: Inicializa σ_t
 - 5: **Repeat** para cada passo t do episódio
 - 6: Escolha α_t para σ_t usando como decisão o ϵ -Greedy
 - 7: Executa α_t e observa r e σ_{t+1}
 - 8: $\hat{Q}_t(\sigma_t, \alpha_t) \leftarrow (1-a) \cdot \hat{Q}_{t-1}(\sigma_t, \alpha_t) + a \cdot \left[r(\sigma_t, \alpha_t, \sigma_{t+1}) + \gamma \max_{\alpha'} \hat{Q}_{t-1}(\sigma_{t+1}, \alpha') \right]$
 - 9: $\sigma \leftarrow \sigma_{t+1}$
 - 10: **until** σ_t não é a meta
 - 11: **until** condição de término de aprendizado
 - 12: **end function**
-

2.4 Processo Markoviano de Decisão Relacional

A cada instante de tempo, o agente deve escolher entre a exploração do ambiente, para melhorar sua estimativa de Q , e a exploração do seu conhecimento corrente, visando com isto maximizar sua recompensa. Uma dificuldade deste empreendimento é que se acaso a meta, especificada pela função recompensa, for mudada, o agente deve aprender tudo do começo. De fato, o aprendizado por reforço sofre do problema da convergência lenta para a política ótima, além da inabilidade de reutilizar o conhecimento previamente aprendido.

De volta para o exemplo da Figura 2.1 e sendo $\mathbf{P}$ como definido anteriormente, e assumindo o modelo de recompensa descontada, pode-se facilmente observar que para o exemplo $Q^{\pi^*}(\sigma_4, \alpha_{45}) = -1 - \gamma$ e $Q^{\pi^*}(\sigma_4, \alpha_{43}) = -1 - \gamma - \gamma^2$. Esses valores indicam a estimativa do valor esperado para $Q^{\pi^*}(\sigma, \alpha)$, $\sigma \in \mathcal{S}$ e $\alpha \in \mathcal{A}_\sigma$, ao aplicar a ação α no estado σ e escolher as próximas ações seguindo a política ótima π^* para o problema.

Observe que α_{43} não é uma ação ótima, mas isto não é um problema para a obtenção do valor esperado para $Q(\sigma, \alpha)$. Q^{π^*} indica que o valor esperado de R_t é obtido através da escolha de uma ação qualquer (indicada no argumento), sendo a escolha das próximas ações dada pela política ótima π^* .

Como uma ilustração do desafio encontrado quando a meta é modificada, suponha que agora é desejado alcançar o estado σ_1 em vez de σ_6 . A simetria da figura deixa claro que o problema é muito similar ao que foi resolvido anteriormente, mas a função recompensa é modificada agora para $r(\sigma, \alpha, \sigma') = 0$ se $\sigma = \sigma' = \sigma_1$ e $r(\sigma, \alpha, \sigma') = -1$ caso contrário. Deve-se aprender a política Q^{π^*} do zero, visto que agora por exemplo $Q^{\pi^*}(\sigma_4, \alpha_{45}) = -1 - \gamma - \gamma^2 - \gamma^3$.

O maior interesse aqui é abstrair os detalhes de uma política previamente construída e utilizar esta política abstrata em um novo problema. A política abstrata vai ser construída em uma linguagem relacional.

A idéia chave do Processo de Decisão Markoviano Relacional (*Relational Markov Decision Process* - RMDP) (OTTERLO, 2009) é utilizar a lógica de primeira ordem em conjunto com o MDP. A lógica de primeira ordem provê o MDP com representações relacionais para os seus estados e ações, sendo que são utilizados termos básicos nestas representações. Sendo $\mathcal{P} = \{\rho_1/\phi_1, \dots, \rho_n/\phi_n\}$ um conjunto de predicados ρ_i com suas aridades ϕ_i , $\mathcal{C} = \{c_1, \dots, c_k\}$ um conjunto de constantes, e $\mathcal{A}' = \{\kappa_1/\iota_1, \dots, \kappa_m/\iota_m\}$ um conjunto de ações κ_i com suas aridades ι_i onde $\kappa_i = H_{\kappa_i} \xleftarrow{p_{\kappa_i}: A_{\kappa_i}} B_{\kappa_i}$. Nesta definição, A_{κ_i} é uma sentença atômica que

representa o nome e os argumentos da ação κ_i ; B_{κ_i} contém a pré-condição para a ação, i. e., um conjunto de sentenças atômicas que devem ser satisfeitas para ser possível a aplicação da ação κ_i ; H_{κ_i} contém a pós-condição da aplicação da ação κ_i , i. e., as sentenças atômicas que serão válidas após a aplicação sucedida da ação; e p_{κ_i} indica a probabilidade de que a aplicação da ação seja sucedida.

Considerando $\mathcal{S}'$ como um conjunto formado por todas as sentenças sobre $\mathcal{P}$ e $\mathcal{C}$, $\mathbf{A}$ o conjunto de todas as sentenças atômicas sobre $\mathcal{A}'$ e $\mathcal{C}$; e sendo $\mathbf{S}$ um subconjunto de $\mathcal{S}'$. A partir destas definições, a diferença entre RMDP e MDP consiste apenas na definição, com representação relacional, de $\mathcal{S}$ e $\mathcal{A}$, i. e., os estados e ações são representados numa linguagem baseada na lógica de primeira ordem ($\mathcal{S} = \mathbf{S}$ e $\mathcal{A} = \mathbf{A}$). Nesta linguagem, não são utilizadas funções na representação de objetos. Para um estado $\sigma \in \mathcal{S}$ existe um conjunto correspondente de ações aplicáveis $\mathcal{A}_\sigma \subseteq \mathcal{A}$ que são calculadas a partir das pré-condições das ações. Uma ação κ_i é aplicável num estado σ se suas pré-condições são satisfeitas.

A utilização de RMDPs oferece muitas possibilidades para abstração devido à forma estruturada das sentenças que descrevem os estados e as ações. A abstração pode ser realizada através da criação de sentenças complexas que utilizam variáveis em seus termos.

Do mesmo modo que nos MDPs, técnicas de programação dinâmica e aprendizado por reforço podem ser utilizadas para resolver RMDPs, e com isto extrair uma política ótima. O método mais apropriado a utilizar depende de se ter ou não o modelo completo do RMDP.

2.5 Conclusão do capítulo

Neste Capítulo foram fornecidos os conceitos básicos necessários para introduzir as duas ferramentas fundamentais utilizadas no presente trabalho: o aprendizado por reforço e o RMDP.

Com base nestes, propomos uma modelagem utilizando RMDP para o problema de navegação robótica e uma abordagem baseada em aprendizado por reforço para reusar políticas em problemas similares e mesmo para adaptá-las aos novos problemas.

3 Modelando e abstraindo um problema de navegação robótica com RMDP

Um problema de navegação robótica pode ser elegantemente representado utilizando um RMDP. Na representação por RMDP, o problema é dado em função de objetos e relações entre estes. Na navegação robótica os objetos representam as localidades do ambiente. As relações entre objetos correspondem às conexões existentes entre localidades e aos tipos de localidades possíveis.

De uma maneira semelhante, sentenças complexas podem ser utilizadas para agrupar estados semelhantes de um problema. Essas sentenças complexas vão definir os estados abstratos, i. e., sentenças complexas que representam grupos de estados. O agrupamento de estados é realizado através da utilização de variáveis nos termos das sentenças complexas. Os estados pertencentes a um agrupamento são todos aqueles que podem ser obtidos através de uma substituição θ na sentença complexa que define o agrupamento de estados.

A seguir é apresentado como um RMDP pode ser utilizado para representar um problema concreto (Seção 3.1) e como, através da utilização de sentenças complexas com variáveis, os problemas podem ser abstraídos (Seção 3.2).

3.1 Modelando um problema de navegação

Na representação de um problema através de um RMDP são utilizados constantes para representar os objetos, sendo o nome destas formadas por uma palavra iniciada em letra minúscula seguida por números, e. g., $11, 12, \dots, 1n$. Por exemplo, considere o ambiente simples indicado na Figura 3.1 onde existem 19 localidades no total, i. e., $\mathcal{C} = \{11, 12, \dots, 119\}$.

Os tipos de localidades são definidos por quatro predicados unários, sendo eles: `isCorridor/1` indica que a localidade (argumento do predicado), é um

corredor; **isRoom/1** indica que a localidade é um cômodo; **isCenter/1** indica que a localidade se encontra no centro do ambiente; e **isNearDoor/1** que indica que a localidade está próxima de uma porta. A partir das combinações destes quatro predicados unários, o ambiente pode ser dividido em quatro tipos de regiões:

- **Centro de corredor** (uma região de um corredor que não está próxima de uma porta): formada pelos predicados **isCorridor/1** e **isCenter/1**, e. g. $\text{isCorridor}(l1) \wedge \text{isCenter}(l1)$ indica que $l1$ é o centro de um corredor.
- **Porta de corredor** (uma região de um corredor que está próxima a uma porta): formada pelos predicados **isCorridor/1** e **isNearDoor/1**, e. g. $\text{isCorridor}(l5) \wedge \text{isNearDoor}(l5)$.
- **Centro de um cômodo** (uma região de um cômodo que não está próxima de uma porta): formada pelos predicados **isRoom/1** e **isCenter/1**, e. g. $\text{isRoom}(l2) \wedge \text{isCenter}(l2)$.
- **Porta de cômodo** (uma região de um cômodo próxima a uma porta): formada pelos predicados **isRoom/1** e **isNearDoor/1**, e.g. $\text{isRoom}(l4) \wedge \text{isNearDoor}(l4)$.

Cada localidade do ambiente vai pertencer a apenas um tipo de região, conforme pode ser observado na Figura 3.1.

Figura 3.1: Um exemplo de uma representação relacional para um problema de navegação. Quadrados simbolizam centro de cômodos, e. g. $(\text{isRoom}(l2) \wedge \text{isCenter}(l2))$; triângulos simbolizam regiões de um cômodo próximas de portas, e. g., $(\text{isRoom}(l4) \wedge \text{isNearDoor}(l4))$; círculos simbolizam regiões de corredores próximas de portas, e. g., $(\text{isCorridor}(l5) \wedge \text{isNearDoor}(l5))$; e pentágonos negros simbolizam centros de corredores, e. g., $(\text{isCorridor}(l1) \wedge \text{isCenter}(l1))$

A localização do robô é dada por um predicado unário especial `in/1`. Por exemplo, `in(18)` indica que o robô está na localidade 18. Já a conexão entre localidades é representada pelo predicado binário `conn/2`, e. g. `conn(13,16)` indica que as localidades 13 e 16 estão conectadas.

Os estados são representados por sentenças complexas. Para a formação das sentenças complexas são utilizadas as sentenças que definem os tipos de localidades, a sentença atômica de localização e a sentença atômica de conexão entre localidades. A Tabela 3.1 apresenta todos os 19 estados definidos para o ambiente ilustrado na Figura 3.1.

Nesta representação é considerada a existência de um conjunto $\mathcal{A}'$ de ações especializadas que correspondem aos deslocamentos do robô de uma localidade para outra, e. g., `goto_RC_RD(12,14)` corresponde ao deslocamento do centro de um cômodo 12 para uma região de um cômodo 14 próxima de uma porta; `goto_RD_CD(14,15)` corresponde ao deslocamento de uma região 14 de um cômodo próxima a uma porta para uma região 15 de um corredor próxima a uma porta; etc.

O conjunto de ações $\mathcal{A}'$ pode ser descrito através de uma ação genérica `goto_XX_YY/2` onde `XX` representa as pré-condições para a localidade origem, i. e., a localidade onde o robô se encontra; e `YY` as pré-condições para o destino do deslocamento. Por exemplo, para representar a transição de 12 para 14 deve-se ter `XX = RC (isRoom(12)  $\wedge$  isCenter(12))` e assim por diante; do mesmo modo, para a transição de 110 para 111, deve-se ter `YY = CC (isCorridor(111)  $\wedge$  isCenter(111))`. As representações para as pré-condições de aplicabilidade da ação para cada uma das regiões estão representadas na Tabela 3.2.

A partir da definição anterior para as condições requisitos, a Tabela 3.3 indica todas as ações que podem ser obtidas através da substituição de `XX` e `YY` pelos quatro tipos de regiões existentes.

As pré-condições básicas para a obtenção de uma ação é que o robô se encontre na localidade origem e que as localidades origem e destino estejam conectadas. Por exemplo, para a ação `goto_RC_RD(12,14)`, além das pré-condições de `XX (isRoom(12)  $\wedge$  isCenter(12))` e `YY (isRoom(14)  $\wedge$  isNearDoor(14))`, deve-se ter `in(12)  $\wedge$  conn(12,14)`.

Se a aplicação de uma ação ocorrer com sucesso, o robô atinge a localidade desejada (no exemplo anterior 14) com probabilidade p , caso contrário, o robô irá permanecer no mesmo local (12) com probabilidade $1 - p$.

Tabela 3.1: Estados existentes para o ambiente ilustrado na Figura 3.1.

Estados
$\text{in}(11) \wedge \text{isCorridor}(11) \wedge \text{isCenter}(11) \wedge \text{conn}(11,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(12) \wedge \text{isRoom}(12) \wedge \text{isCenter}(12) \wedge \text{conn}(12,14) \wedge \text{isRoom}(14) \wedge \text{isNearDoor}(14)$
$\text{in}(13) \wedge \text{isRoom}(13) \wedge \text{isCenter}(13) \wedge \text{conn}(13,16) \wedge \text{isRoom}(16) \wedge \text{isNearDoor}(16)$
$\text{in}(14) \wedge \text{isRoom}(14) \wedge \text{isNearDoor}(14) \wedge \text{conn}(14,12) \wedge \text{isRoom}(12) \wedge \text{isCenter}(12) \wedge \text{conn}(14,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15) \wedge \text{conn}(15,14) \wedge \text{isRoom}(14) \wedge \text{isNearDoor}(14) \wedge \text{conn}(15,16) \wedge \text{isRoom}(16) \wedge \text{isNearDoor}(16) \wedge \text{conn}(15,11) \wedge \text{isCorridor}(11) \wedge \text{isCenter}(11) \wedge \text{conn}(15,18) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18)$
$\text{in}(16) \wedge \text{isRoom}(16) \wedge \text{isNearDoor}(16) \wedge \text{conn}(16,13) \wedge \text{isRoom}(13) \wedge \text{isCenter}(13) \wedge \text{conn}(16,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(17) \wedge \text{isRoom}(17) \wedge \text{isNearDoor}(17) \wedge \text{conn}(17,19) \wedge \text{isRoom}(19) \wedge \text{isCenter}(19) \wedge \text{conn}(17,18) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18)$
$\text{in}(18) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18) \wedge \text{conn}(18,17) \wedge \text{isRoom}(17) \wedge \text{isNearDoor}(17) \wedge \text{conn}(18,110) \wedge \text{isCorridor}(110) \wedge \text{isCenter}(110) \wedge \text{conn}(18,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(19) \wedge \text{isRoom}(19) \wedge \text{isCenter}(19) \wedge \text{conn}(19,17) \wedge \text{isRoom}(17) \wedge \text{isNearDoor}(17) \wedge \text{conn}(19,112) \wedge \text{isRoom}(112) \wedge \text{isNearDoor}(112)$
$\text{in}(110) \wedge \text{isCorridor}(110) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18) \wedge \text{isCenter}(110) \wedge \text{conn}(110,18) \wedge \text{conn}(110,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115) \wedge \text{conn}(110,111) \wedge \text{isCorridor}(111) \wedge \text{isCenter}(111)$
$\text{in}(111) \wedge \text{isCorridor}(111) \wedge \text{isCenter}(111) \wedge \text{conn}(111,110) \wedge \text{isCorridor}(110) \wedge \text{isCenter}(110)$
$\text{in}(112) \wedge \text{isRoom}(112) \wedge \text{isNearDoor}(112) \wedge \text{conn}(112,19) \wedge \text{isRoom}(19) \wedge \text{isCenter}(19) \wedge \text{conn}(112,113) \wedge \text{isRoom}(113) \wedge \text{isNearDoor}(113)$
$\text{in}(113) \wedge \text{isRoom}(113) \wedge \text{isNearDoor}(113) \wedge \text{conn}(113,117) \wedge \text{isRoom}(117) \wedge \text{isCenter}(117) \wedge \text{conn}(113,112) \wedge \text{isRoom}(112) \wedge \text{isNearDoor}(112)$
$\text{in}(114) \wedge \text{isRoom}(114) \wedge \text{isNearDoor}(114) \wedge \text{conn}(114,117) \wedge \text{isRoom}(117) \wedge \text{isCenter}(117) \wedge \text{conn}(114,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115)$
$\text{in}(115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115) \wedge \text{conn}(115,114) \wedge \text{isRoom}(114) \wedge \text{isNearDoor}(114) \wedge \text{conn}(115,116) \wedge \text{isRoom}(116) \wedge \text{isNearDoor}(116) \wedge \text{conn}(115,119) \wedge \text{isCorridor}(119) \wedge \text{isCenter}(119) \wedge \text{conn}(115,110) \wedge \text{isCorridor}(110) \wedge \text{isCenter}(110)$
$\text{in}(116) \wedge \text{isRoom}(116) \wedge \text{isNearDoor}(116) \wedge \text{conn}(116,118) \wedge \text{isRoom}(118) \wedge \text{isCenter}(118) \wedge \text{conn}(116,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115)$
$\text{in}(117) \wedge \text{isRoom}(117) \wedge \text{isCenter}(117) \wedge \text{conn}(117,114) \wedge \text{isRoom}(114) \wedge \text{isNearDoor}(114) \wedge \text{conn}(117,113) \wedge \text{isRoom}(113) \wedge \text{isNearDoor}(113)$
$\text{in}(118) \wedge \text{isRoom}(118) \wedge \text{isCenter}(118) \wedge \text{conn}(118,116) \wedge \text{isRoom}(116) \wedge \text{isNearDoor}(116)$
$\text{in}(119) \wedge \text{isCorridor}(119) \wedge \text{isCenter}(119) \wedge \text{conn}(119,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115)$

Tabela 3.2: Tipos de regiões existentes para o problema de navegação robótica, os predicados utilizados para representar as pré-condições de aplicabilidade da ação e a representação da região.

Região	Predicados utilizados	Representação
Centro de Corredor	isRoom/1 e isCenter/1	RC
Porta de Corredor	isRoom/1 e isNearDoor/1	RD
Centro de Cômodo	isCorridor/1 e isCenter/1	CC
Porta de Cômodo	isCorridor/1 e isNearDoor/1	CD

Tabela 3.3: Predicados representando a transição entre os diversos tipos de região existentes no problema de navegação robótica. A primeira coluna **XX** indica o tipo de localidade da origem e a segunda coluna **YY** o tipo de localidade do destino. A terceira coluna indica o predicado construído através da combinação dos tipos indicados na primeira e na segunda coluna.

XX	YY	Predicado da ação
RC	RC	goto_RC_RC/2
RC	RD	goto_RC_RD/2
RC	CC	goto_RC_CC/2
RC	CD	goto_RC_CD/2
RD	RC	goto_RD_RC/2
RD	RD	goto_RD_RD/2
RD	CC	goto_RD_CC/2
RD	CD	goto_RD_CD/2
CC	RC	goto_CC_RC/2
CC	RD	goto_CC_RD/2
CC	CC	goto_CC_CC/2
CC	CD	goto_CC_CD/2
CD	RC	goto_CD_RC/2
CD	RD	goto_CD_RD/2
CD	CC	goto_CD_CC/2
CD	CD	goto_CD_CD/2

3.2 Abstraindo um problema de navegação

A utilização de RMDPs possibilita a abstração do conhecimento por meio da definição de estados e ações abstratos. Um *estado abstrato* $\hat{\sigma}$ é definido por uma sentença complexa que possui pelo menos um termo com variável. O conjunto de estados abstratos é $\hat{\mathcal{S}}$. Um determinado estado concreto $\sigma \in \mathcal{S}$ é uma instância de $\hat{\sigma}$ se existir uma determinada substituição $\theta = \{V_1/t_1, \dots, V_k/t_k\}$ tal que $\hat{\sigma}\theta = \sigma$, onde V_i é uma variável e t_i uma constante. Por exemplo, dado o estado abstrato $\hat{\sigma} = (\text{in}(L) \wedge \text{isRoom}(L) \wedge \text{isNearDoor}(L) \wedge \text{conn}(L, L_2) \wedge \text{isRoom}(L_2) \wedge \text{isCenter}(L_2))$ e a substituição $\theta = \{L/11, L_2/12\}$, então o estado $\sigma = \hat{\sigma}\theta = (\text{in}(11) \wedge \text{isRoom}(11) \wedge \text{isNearDoor}(11) \wedge \text{conn}(11, 12) \wedge \text{isRoom}(12) \wedge \text{isCenter}(12))$ é uma instância de $\hat{\sigma}$, assim como $\sigma = \hat{\sigma}\theta = (\text{in}(13) \wedge \text{isRoom}(13) \wedge \text{isNearDoor}(13) \wedge \text{conn}(13, 14) \wedge \text{isRoom}(14) \wedge \text{isCenter}(14))$ também é para a substituição $\theta = \{L/13, L_2/14\}$.

Estados abstratos podem ser utilizados para representar um conjunto de estados básicos que possuem semelhanças estruturais entre si, i. e., um estado abstrato representa estados que possuem tipos de objetos semelhantes e relações semelhantes entre estes. Do mesmo modo, define-se *ação abstrata* $\hat{a} \in \hat{\mathcal{A}}$ aquela que possui pelo menos uma variável como parâmetro. Ações abstratas também são utilizadas para agrupar as ações especializadas. Por exemplo, no ambiente mostrado na Figura 3.1, a ação abstrata $\text{goto_CD_RD}(15, L)$ representa tanto a ação concreta $\text{goto_CD_RD}(15, 14)$ quanto $\text{goto_CD_RD}(15, 16)$.

O Capítulo 4 a seguir, apresenta como uma política definida em termos de estados e ações abstratos pode ser utilizada na construção de uma árvore de decisão representada em lógica de primeira ordem (*First Order Logical Decision Tree* - FOLDT). Posteriormente no Capítulo 5 é apresentado como uma FOLDT pode ser induzida para representar uma política abstrata para um problema de navegação robótica.

4 Abstração relacional em árvore de decisão

Árvore de decisão é uma das estruturas mais populares para a representação do conhecimento e dos padrões estruturais encontrados em dados obtidos pelas técnicas de aprendizagem de máquina (MITCHELL, 1997; WITTEN; FRANK, 2005). Este tipo de estrutura recebe como entrada uma determinada situação descrita por um conjunto de atributos e, em função desta, devolve uma decisão que prediz um valor para esta situação.

Neste tipo de estrutura, os nós internos têm a função de testar o conjunto de atributos da situação de entrada. Quando a árvore utiliza em sua estrutura apenas testes binários, ela é chamada de árvore binária. Neste caso, cada nó interno possui apenas dois filhos: filho esquerdo quando o teste do nó for válido e filho direito no outro caso.

Os valores preditos nas folhas da árvore para a situação de entrada podem ser tanto discretos quanto contínuos. No caso de predição de valores discretos a árvore é chamada de árvore de classificação ou diagramas de decisão binária (BDD); e no caso de valores contínuos ela é chamada de árvore de regressão ou diagramas de decisão algébrica (ADD) (RUSSELL; NORVIG, 2003).

Para retornar uma decisão é realizada uma sequência de testes. Os testes a serem realizados correspondem aos nós internos da árvore. Cada ramo de um nó corresponde a uma saída para o teste realizado, sendo nomeado com a saída possível. Cada folha da árvore corresponde à decisão a ser tomada, caso a folha seja alcançada.

Um exemplo de uma árvore de decisão binária¹ se encontra na Figura 4.1. Esta árvore representa a tomada de decisão para se jogar uma partida de tênis. Cada situação é formada por um conjunto de quatro atributos: chovendo, úmido, ventando e ensolarado. Os testes possíveis para essa árvore correspondem a esses atributos. Existe dois ramos possíveis para os testes: sim e não. Nas folhas da

¹Adaptado de (MITCHELL, 1997).

árvore estão as classificações possíveis para a situação de entrada. Dois tipos de classificação para as situações são possíveis: JOGAR e NÃO_JOGAR.

Figura 4.1: Exemplo de uma árvore de decisão binária.

Suponha que se deseja obter a decisão, utilizando a árvore da Figura 4.1, para a situação descrita pelos atributos {úmido = não, chovendo = não, ventando = sim, ensolarado = sim}. A decisão é tomada através da aplicação dos testes da raiz até uma folha da árvore. Inicialmente o teste do nó raiz (úmido) é aplicado e, devido ao fato deste ser inválido para esta situação, o próximo teste a ser realizado vai ser o ramo negativo deste (filho direito). O mesmo processo é repetido para o teste ventando, chegando com isto na decisão JOGAR.

4.1 Indução de árvores de decisão

A construção de uma árvore de decisão pode ser feita utilizando a abordagem dividir-e-conquistar. Nesta abordagem a árvore vai sendo construída através da criação de sucessivos subconjuntos que decidem as diversas situações de entrada. A partir de um conjunto inicial E de exemplos, o teste a ser escolhido, entre aqueles definidos para a sua divisão, é aquele que melhor divide este conjunto segundo um determinado critério de qualidade. O teste escolhido é utilizado para a divisão do conjunto original E em dois subconjuntos: E_1 e E_2 . O conjunto E_1 corresponde aos exemplos no qual o teste é válido e o segundo conjunto E_2 corresponde aos exemplos no qual ele é inválido, com $E_1 \cap E_2 = \emptyset$ e $E_1 \cup E_2 = E$. O processo de escolha-divisão é repetido recursivamente para cada um dos subconjuntos gerados. Considera-se aqui que o conjunto E de exemplos utilizados corresponde a um conjunto de situações, sendo que cada situação descrita é definida por um conjunto de atributos.

O método TDIDT (*Top-down Induction of Decision Tree*) (QUINLAN, 1986) utiliza a abordagem dividir-e-conquistar para a construção de uma árvore de decisão. Neste método a árvore de decisão é construída da raiz até as folhas. O

algoritmo deste método se encontra no Algoritmo 2. O algoritmo possui como entrada (passo 1), para a construção da árvore, um conjunto de exemplos E . Para este conjunto de exemplos é gerado, através do procedimento `GENERATE_TESTS`, um conjunto de testes candidatos T para a divisão deste (passo 2). O teste τ que melhor divide os exemplos, de acordo com um determinado critério de qualidade, é escolhido (procedimento `OPTIMAL_SPLIT` no passo 3). O teste escolhido particiona o conjunto E em subconjuntos (passo 4). Se o critério de parada (procedimento `STOP_CRIT(E,  $\epsilon$ )` no passo 5) de construção da árvore for válido, é criada na árvore uma folha contendo a decisão que melhor representa a informação existente no conjunto E (passo 6). No caso de uma árvore de classificação, a decisão corresponde à classe que representa o maior número de exemplos existentes no conjunto E ; e no caso de uma árvore de regressão, a decisão corresponde à média dos valores dos exemplos existentes no conjunto E . Caso o critério de parada não seja válido, para cada subconjunto E_j da partição induzida é realizada uma nova chamada à função `TDIDT` (passos 8 e 9). As subárvores t_j retornadas por cada uma destas chamadas são utilizadas, em conjunto com o teste τ , na geração de um nó interno da árvore que possui como ramos cada uma das t_j (passo 11).

Algoritmo 2 Algoritmo de indução de árvore de decisão TDIDT.

```

1: function TDIDT ( $E$ : conjunto de exemplos): retorna uma árvore de decisão
2:    $T = \text{GENERATE\_TESTS}(E)$ 
3:    $\tau = \text{OPTIMAL\_SPLIT}(T, E)$ 
4:    $\epsilon = \text{partição induzida em } E \text{ por } \tau$ 
5:   if STOP_CRIT( $E, \epsilon$ ) then
6:     return leaf(INFO( $E$ ))
7:   else
8:     for all  $E_j$  in  $\epsilon$  do
9:        $t_j = \text{TDIDT}(E_j)$ 
10:    end for
11:    return inode( $\tau, (j, t_j)$ )
12:  end if
13: end function

```

4.2 Indução de árvores de decisão em lógica de primeira ordem

A programação lógica indutiva (ILP ou Inductive Logic Programming) (RUSSELL; NORVIG, 2003) é uma técnica de aprendizado de hipóteses. As hipóteses são representadas por um conjunto de regras. As regras por sua vez são representadas pela lógica de primeira ordem, sendo que são utilizadas variáveis nesta representação.

O conceito envolvido com as árvores de decisão representadas em lógica de

primeira ordem (FOLDT ou *First Order Logical Decision Tree*) é combinar a programação lógica indutiva com árvores de decisão (BLOCKEEL; RAEDT, 1998). Neste caso, a árvore possui em sua estrutura sentenças complexas representando os testes. Basicamente, existem as seguintes diferenças estruturais nas árvores de decisão representadas em lógica de primeira ordem:

- Os testes existentes nos nós internos da árvore são sentenças complexas. Neste caso, deve existir na sentença complexa pelo menos um termo que seja uma variável.
- Pode ocorrer compartilhamento de variáveis entre os nós da árvore. A restrição neste compartilhamento é de que uma variável que aparece pela primeira vez no teste de um determinado nó deve aparecer apenas no ramo positivo deste (ramo onde o teste foi válido, correspondendo ao filho esquerdo do nó). Uma variável L introduzida em um nó é quantificada existencialmente dentro do teste (conjunção de sentenças atômicas) deste nó. O ramo negativo (onde o teste é inválido) é relevante apenas quando o teste falha (não existe algum L que satisfaça o teste), sendo que neste caso não tem sentido referenciar novamente a variável L .

Suponha uma situação hipotética na qual se deseja classificar uma localidade do ambiente em um dos quatro tipos principais de regiões indicados na Tabela 3.2. A localidade do ambiente é representada por uma sentença complexa que descreve as suas características principais. Um exemplo de uma árvore de decisão, representada em lógica de primeira ordem que classificaria adequadamente as localidades se encontra na Figura 4.2. Para esta árvore, a localidade descrita pela sentença complexa $\text{isRoom}(11) \wedge \text{isCenter}(11)$ seria classificada como RC; e a localidade descrita por $\text{isCorridor}(12) \wedge \text{isDoor}(12)$ como CD.

Figura 4.2: Exemplo de uma árvore de decisão representada em lógica de primeira ordem para classificar as localidades em quatro tipos de regiões: centro de corredores (CC), localidades de corredores próximas a portas (CD), centro de cômodos (RC) e localidades de cômodos próximas a portas (RD).

O algoritmo TILDE (BLOCKEEL; RAEDT, 1998) é uma extensão para a lógica de primeira ordem do algoritmo TDIDT visto na seção 4.1. Ele constrói uma árvore de decisão em lógica de primeira ordem utilizando a metodologia *top-down*. Para a construção da árvore de decisão em lógica de primeira ordem o algoritmo TILDE parte de uma sentença complexa que seja sempre verdadeira. Essa sentença é definida por um especialista. No caso do domínio de navegação robótica o robô sempre vai estar em uma determinada localidade ($\text{in}(\mathbf{A})$). Essa sentença complexa é considerada o nó raiz da árvore (DZEROSKI; RAEDT; DRIESSENS, 2001). O principal motivo da existência deste nó é o reaproveitamento de variáveis entre os nós da árvore. O nó raiz é considerado um fornecedor inicial de variáveis para os outros nós. Além desta consideração, algumas modificações são realizadas nos procedimentos do algoritmo TDIDT. O algoritmo TILDE, com as modificações nos procedimentos, é apresentado no Algoritmo 3.

Algoritmo 3 Algoritmo de indução de árvore de decisão representada em lógica de primeira ordem TILDE.

```

1: function TILDE (E: conjunto de exemplos): retorna uma FOLDT
2:   T = GENERATE_TESTS_FOL(E)
3:    $\tau$  = OPTIMAL_SPLIT_FOL(T,E)
4:    $\epsilon$  = partição induzida em E por  $\tau$ 
5:   if STOP_CRIT(E, $\epsilon$ ) then
6:     return leaf(INFO_FOL(E))
7:   else
8:     for all  $E_j$  in  $\epsilon$  do
9:        $t_j$  = TILDE( $E_j$ )
10:    end for
11:    return inode( $\tau, (j, t_j)$ )
12:  end if
13: end function

```

O procedimento GENERATE_TESTS_FOL, de geração de testes candidatos para a indução da árvore, é modificado para gerar sentenças complexas como testes. As sentenças complexas são criadas a partir de um conjunto de operadores de refinamento (BLOCKEEL; RAEDT, 1998). Esse conjunto de operadores de refinamento deve ser definido por um especialista.

Cada operador de refinamento é definido por um conjunto de predicados e para cada predicado existe associado um conjunto de termos pertencentes a este. Os termos podem ser tanto variáveis quanto constantes. No caso dos termos que serão variáveis é associado um modo para sua criação. São definidos três modos de geração das variáveis: modo '+', modo '-' e modo '+-'. O modo '+' indica que a variável já deve existir no caminho da raiz até o nó antecessor do nó atual. O modo '-' indica que a variável não deve existir no caminho da raiz até o nó atual,

sendo que ela deve ser criada neste nó. No modo '+-' ambos os modos anteriores podem ser utilizados para a criação da variável. Além disto, cada operador define também a quantidade n de vezes que ele poderá ser utilizado para gerar testes candidatos para a indução da árvore. Por exemplo, um operador com $n = 1$ só poderá ser utilizado uma única vez para gerar nós de teste no caminho da árvore, i. e., ele só pode gerar testes candidatos para um nó se não foi utilizado em nenhum dos nós antecessores deste.

Um exemplo hipotético de um conjunto de operadores de refinamento para o problema de navegação robótica se encontra na Tabela 4.1. Nesta tabela cada linha corresponde a um operador de refinamento. A primeira coluna indica os predicados e os modos que definem a sentença complexa do teste. Na segunda coluna está indicada a quantidade de vezes que cada operador de refinamento pode ser utilizado. Por exemplo, no caso do terceiro operador de refinamento (`isCenter(+)`) foi definido que a variável a ser utilizada para a construção do teste pode ser tanto uma existente nos antecessores do nó atual da árvore quanto uma variável nova a ser criada neste nó. Já o operador `conn(+,-)` indica que o primeiro termo do argumento deve usar uma variável existente e o segundo termo, não.

Tabela 4.1: Exemplo de operadores de refinamento para a navegação robótica. Para cada operador de refinamento são definidos os predicados, modos dos termos e a quantidade de vezes que o operador de refinamento pode ser utilizado.

Definição dos Operadores de Refinamento	
Predicados e modos dos termos	Quantidade de vezes a utilizar
<code>isRoom(+)</code>	2
<code>isCorridor(-)</code>	2
<code>isCenter(+)</code>	2
<code>conn(+,-)</code>	4

Supondo, por exemplo, que se deseja gerar os testes candidatos para um determinado conjunto de exemplos existentes em um determinado nó da árvore. Nesta situação hipotética existem duas variáveis a serem utilizadas para a geração dos testes (variáveis estas que já foram criadas em nós anteriores da árvore): variável A e variável B. Os operadores de refinamento a serem utilizados para a geração dos testes candidatos são aqueles informados na Tabela 4.1. A partir desta situação exemplo, os testes gerados por este conjunto de operadores de refinamento estão indicados na Tabela 4.2.

O algoritmo TILDE pode induzir tanto árvores de classificação quanto árvores de regressão. Em função disto, os outros procedimentos do algoritmo TDIDT são

Tabela 4.2: Testes candidatos derivados dos operadores de refinamento da Tabela 4.1, com A e B como variáveis já existentes.

Operador	Testes gerados pelo operador
isRoom(+)	isRoom(A) isRoom(B)
isCorridor(-)	isCorridor(C)
isCenter(+/-)	isCenter(A) isCenter(B) isCenter(C)
conn(+, -)	conn(A, C) conn(B, C)

dados em função do tipo de árvore que se quer induzir.

Para o procedimento `OPTIMAL_SPLIT` no caso de classificação a escolha do melhor teste candidato para a divisão de um nó é realizada através do *default gain ratio* GR (QUINLAN, 1993) e no caso da regressão a escolha é dada em função da redução da variância. O *default gain ratio* escolhe como teste de divisão de um nó da árvore aquele que reduz a entropia dos conjuntos. No caso da regressão o melhor teste é aquele que possui a maior diferença entre as médias dos valores dos dois subconjuntos criados pela divisão. Maximizar a diferença é equivalente a minimizar a variância entre os subconjuntos.

O procedimento `STOP_CRIT` indica a parada na divisão dos nós no processo de indução da árvore. A parada de divisão dos nós ocorre em vista de duas condições. Estas condições dependem do tipo de árvore que está sendo induzida.

Para a árvore de classificação ocorre a parada da divisão quando: os exemplos de um conjunto são todos pertencentes a mesma classe; ou não existe nenhuma divisão que permita o mínimo de ganho de informação ao mesmo tempo que deixe os dois subconjuntos com um número mínimo de exemplos (**nm**).

No caso da regressão essas duas condições são: nenhuma divisão pode ser encontrada que reduza a variância entre os subconjuntos ao mesmo tempo que deixe os dois subconjuntos com um número mínimo de exemplos (**nm**); ou nenhuma divisão reduz significativamente a variância (indicada por um F-test).

Uma folha, contendo uma decisão, é gerada se acaso o critério de parada for válido (procedimento `leaf(INFO_FOL(E))` do Algoritmo 3). Essa decisão é criada a partir das informações dos exemplos existentes na folha. As informações utilizadas vão depender do tipo de árvore que está sendo induzida.

Nas árvores de classificação a folha irá conter a classe que melhor representa os exemplos nela existentes. Se acaso existir mais de uma classe que pode representar

os exemplos existentes na folha, é escolhida para representar os exemplos aquela que engloba o maior número de exemplos existentes. Por exemplo, suponha que durante a criação de uma folha em uma árvore existem duas classes (**C1** e **C2**) representando os exemplos desta folha. A classe **C1** representa 70% dos exemplos existentes na folha e **C2** os 30% restantes. Neste caso, a classe **C1** engloba o maior número de exemplos da folha e, em vista disto, é escolhida para representar os exemplos. No caso de uma árvore de regressão a folha vai conter a média dos valores dos exemplos nela existentes.

Suponha como exemplo que se deseja construir uma árvore de classificação para os exemplos da Figura 4.3.a. Tendo para reutilização de variáveis (nó pai definido pelo TILDE) a sentença complexa $\text{in}(A) \wedge \text{conn}(A,B)$ e utilizando os operadores de refinamento da Tabela 4.1; os testes candidatos gerados para este caso se encontram na Tabela 4.2. Utilizando o procedimento **OPTMAL_SPLIT**, o teste que melhor divide os exemplos é escolhido no conjunto de testes candidatos. Como exemplo hipotético, suponha que o procedimento **OPTMAL_SPLIT** apresentou **isCenter(A)** como o melhor teste. Neste caso um nó interno é criado dividindo o conjunto de exemplos inicial em dois subconjuntos (Figura 4.3.b): um subconjunto **E1** contendo os exemplos de **E** no qual o teste **isCenter(A)** é válido e outro **E2** no qual o teste é inválido. O processo de construção é repetido para cada um dos subconjuntos de exemplos criados, **E1** e **E2**.

Figura 4.3: Construção de uma árvore de decisão em lógica de primeira ordem pelo algoritmo TILDE. (a) Árvore contendo apenas o conjunto inicial de exemplos **E**. (b) Árvore depois da escolha de um teste (**isCenter(A)**) para divisão dos exemplos em dois outros conjuntos **E1** e **E2**. **E1** contém os exemplos de **E** onde o teste **isCenter(A)** é válido. Do mesmo modo, **E2** contém os exemplos onde **isCenter(A)** é falso.

5 Abstraindo políticas

Por meio da representação de um problema por um RMDP é possível gerar uma política abstrata que possa ser utilizada em problemas similares. Com essa política abstrata é possível transferir o conhecimento para a resolução de problemas similares.

Introduzindo as seguintes definições:

- $\mathcal{X}$ um conjunto de variáveis,
- $\mathcal{BK}$ (*Background Knowledge*) um conjunto de cláusulas de *Horn*,
- Um estado abstrato $\hat{\sigma} \in \hat{\mathcal{S}}$ é uma sentença complexa sobre $\mathcal{P}$, $\mathcal{C}$, $\mathcal{X}$ e $\mathcal{BK}$; uma ação abstrata $\hat{\alpha} \in \hat{\mathcal{A}}$ é uma sentença atômica sobre $\mathcal{A}'$, $\mathcal{C}$ e $\mathcal{X}$,
- $\mathcal{S}_{\hat{\sigma}}$ é o conjunto de todos os estados concretos cobertos pelo estado abstrato $\hat{\sigma}$, com $\mathcal{S}_{\hat{\sigma}} = \{\sigma \in \mathcal{S} \mid \sigma \preceq_{\theta} \hat{\sigma}\}$,
- Do mesmo modo, $\mathcal{A}_{\hat{\alpha}} = \{\alpha \in \mathcal{A} \mid \alpha \preceq_{\theta} \hat{\alpha}\}$.

Utilizando o problema exemplo da Figura 2.1, reproduzido na Figura 5.1 por comodidade, pode-se definir os estados abstratos como a seguir. Primeiro estado, $\hat{\sigma}_1 = \{\text{in}(\text{A}) \wedge \text{conn}(\text{A}, \text{B}) \wedge \text{isRoom}(\text{A}) \wedge \text{isNearDoor}(\text{A})\}$ representa todos os estados nos quais o robô está numa região de um cômodo próxima a uma porta, i. e. $\mathcal{S}_{\hat{\sigma}_1} = \{\sigma_2, \sigma_5\}$. Segundo estado, $\hat{\sigma}_2 = \{\text{in}(\text{A}) \wedge \text{conn}(\text{A}, \text{B}) \wedge \text{isNearDoor}(\text{A})\}$, representa todos os estados nos quais o robô está numa localidade próxima a uma porta, i. e., $\mathcal{S}_{\hat{\sigma}_2} = \{\sigma_2, \sigma_3, \sigma_4, \sigma_5\}$. Terceiro, $\hat{\sigma}_3 = \{\text{in}(\text{A}) \wedge \text{conn}(\text{A}, \text{B}) \wedge \text{isRoom}(\text{A}) \wedge \text{isCenter}(\text{A})\}$ representa todos os estados nos quais o robô está no centro de um cômodo, i. e. $\mathcal{S}_{\hat{\sigma}_3} = \{\sigma_1, \sigma_6\}$.

Uma política abstrata é um conjunto de regras que mapeia um estado abstrato num conjunto de ações abstratas. Ela captura uma generalização de políticas ótimas (ou subótimas) de acordo com as características estruturais do problema. Define-se uma política abstrata (OTTERLO, 2004) como uma lista de ações abstratas na forma $\hat{\sigma} \rightarrow \{\hat{\alpha}_i\}_i$ para $\hat{\sigma} \in \hat{\mathcal{S}}$, $\hat{\alpha}_i \in \hat{\mathcal{A}}$.

Figura 5.1: Exemplo de um ambiente com 6 estados. Quadrados simbolizam centro de cômodos, e. g. (`isRoom(l1) ∧ isCenter(l1)`); triângulos simbolizam regiões de um cômodo próximas de portas, e. g. (`isRoom(l2) ∧ isNearDoor(l2)`); e círculos simbolizam regiões de corredores próximas de portas, e. g. (`isCorridor(l3) ∧ isNearDoor(l3)`).

Considerando o RMDP para o exemplo da Figura 5.1; uma política abstrata poderia ser:

$$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A) \xrightarrow{1.0} \text{goto_RC_RD}(A,B) \quad (5.1)$$

$$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isNearDoor}(A) \xrightarrow{1.0} \text{goto_RD_CD}(A,B) \quad (5.2)$$

$$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isNearDoor}(A) \xrightarrow{1.0} \text{goto_CD_RD}(A,B) \quad (5.3)$$

$$\text{in}(A) \wedge \text{conn}(A,B) \xrightarrow{1.0} \text{goto_CC_CD}(A,B) \quad (5.4)$$

Particularmente, esta é uma política determinística, já que define, com probabilidade 1, uma única ação abstrata para cada estado abstrato. Deve-se notar que a ordem na lista é importante para a definição da política abstrata. As regras 5.2, 5.3 e 5.4 podem ser aplicadas no estado σ_2 , mas apenas a regra 5.2 fornece um conjunto de ações válidas para este estado. A regra 5.2 quando aplicada no estado σ_2 sugere o conjunto $\{\alpha_{23}, \alpha_{24}\}$; as regras 5.3 e 5.4 sugerem um conjunto vazio $\{\}$.

Essa mesma política abstrata pode ser representada através de uma FOLDT. Neste caso as regras estão implicitamente ordenadas na estrutura da árvore. Os estados abstratos correspondem aos caminhos existentes na árvore e as ações abstratas são as folhas. Uma política abstrata equivalente à política abstrata anteriormente citada é representada pela FOLDT da Figura 5.2.

Figura 5.2: Exemplo de uma política abstrata, para o problema da Figura 5.1, representada por uma FOLDT.

A árvore da Figura 5.2 fornece quatro ações abstratas como política para os estados abstratos existentes nela. Para o estado abstrato $\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A)$ formado pela conjunção das expressões complexas no caminho 1 (Figura 5.3) a árvore fornece a ação abstrata $\text{goto_RC_RD}(A,B)$. Todos os estados abstratos existentes nesta árvore e suas respectivas políticas estão indicadas na Tabela 5.1.

Tabela 5.1: Política abstrata existente na árvore da Figura 5.3 em uma forma tabular. Cada linha da tabela contém um caminho da Figura 5.3, o estado abstrato correspondente e a respectiva ação abstrata indicando a política para o estado abstrato.

Caminho	Estado abstrato	Política abstrata
1	$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A)$	$\text{goto_RC_RD}(A,B)$
2	$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isNearDoor}(A)$	$\text{goto_RD_CD}(A,B)$
3	$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isNearDoor}(A)$	$\text{goto_CD_RD}(A,B)$
4	$\text{in}(A) \wedge \text{conn}(A,B)$	$\text{goto_CC_CD}(A,B)$

Figura 5.3: Caminhos existentes na árvore da Figura 5.2 que definem os estados abstratos da política abstrata.

5.1 Gerando uma política abstrata a partir da política concreta

A partir da definição completa de um RMDP, uma política abstrata ótima pode ser obtida através da programação dinâmica simbólica (*Symbolic dynamic programming*) (OTTERLO, 2004; KERSTING; OTTERLO; RAEDT, 2004). No presente trabalho uma política abstrata é aprendida pela indução de uma árvore de decisão lógica (BLOCKHEEL; RAEDT, 1998) a partir de exemplos gerados com o uso de uma política concreta previamente definida. O conjunto de estados abstratos $\hat{\mathcal{S}}$ é formado pela união de todas as conjunções existentes no caminho da raiz até uma folha; e o conjunto das ações abstratas $\hat{\mathcal{A}}$ é o conjunto de todas as ações abstratas existentes nas folhas.

Uma política abstrata pode ser induzida através de um conjunto de exemplos E . Os exemplos são formados por pares de estado-ação concretos retirados a partir de um conjunto de experiências de navegação gerado com a aplicação da política ótima. Dado um estado meta no problema de navegação robótica, a política concreta ótima para o RMDP pode ser definida. Um conjunto de sequências que levam o robô até a meta pode ser gerado a partir da política ótima e de um conjunto de estados iniciais. Cada um dos pares de cada uma das sequências

é inserido no conjunto de exemplos E. A tarefa portanto é induzir uma política abstrata utilizando este conjunto E gerado.

Suponha que se deseja induzir uma política abstrata a partir do problema exemplo da Figura 5.1. Neste problema todas as ações que podem ser aplicadas num estado têm uma certa probabilidade (p) de sucesso. Se acaso a aplicação da ação falhar (probabilidade $1 - p$) o robô continua na localidade em que se encontrava e, caso contrário, transita para a localidade esperada. A meta neste caso é chegar na localidade 16. O RMDP que modela este problema é resolvido por uma das técnicas apresentadas no Capítulo 2, ou por um especialista, e uma política é extraída.

A política ótima π^* para o problema exemplo da Figura 5.1 pode ser visualizada na Tabela 5.2, onde para cada estado do RMDP (primeira coluna) está indicado sua respectiva política ótima (segunda coluna).

Tabela 5.2: Política ótima para o RMDP da Figura 5.1 tendo como meta 16.

Estado	Política ótima
σ_1	goto_RC_RD(11,12)
σ_2	{goto_RD_CD(12,13), goto_RD_CD(12,14)}
σ_3	goto_CD_RD(13,15)
σ_4	goto_CD_RD(14,15)
σ_5	goto_RD_RC(15,16)
σ_6	goto_RC_RC(16,16)

A partir do conjunto de estados do problema e da política π^* é definido um conjunto Γ de tarefas $\langle \sigma_0, \sigma_g \rangle$ onde $\sigma_0 \in \mathcal{S}$ e $\sigma_g \in \mathcal{G}$, e. g. $\Gamma = \{\langle \sigma_1, \sigma_6 \rangle, \langle \sigma_3, \sigma_6 \rangle, \langle \sigma_2, \sigma_6 \rangle\}$. A política ótima π^* é utilizada para resolver cada uma destas tarefas e todos os estados concretos visitados (e as respectivas ações concretas tomadas) são utilizados para formar o conjunto E.

5.2 Utilizando a política abstrata para guiar o robô

Uma política abstrata $\hat{\pi}_a$ pode induzir uma política π_a em um RMDP concreto. Dado um estado concreto σ , deve-se achar o primeiro estado abstrato $\hat{\sigma}$ da política abstrata com $\sigma \preceq_{\theta} \hat{\sigma}$. No caso de uma política abstrata indicada numa lista de regras, deve-se levar em conta a ordem das regras; já para o caso de utilização de uma FOLDT não ocorre ambiguidade neste mapeamento.

Cada nó da FOLDT possui um teste lógico, que pode tanto ter êxito ou falhar quando aplicado a um estado concreto. Se o teste tiver êxito, deve-se analisar a

subárvore esquerda; caso contrário, deve-se analisar a subárvore direita. As folhas da FOLDT induzida pelo algoritmo TILDE possui uma única ação abstrata que deve ser tomada no estado abstrato definido pela conjunção de todos os testes percorridos.

A política abstrata pode representar um conjunto de ações concretas. Para fornecer uma ação concreta deve-se escolher aleatoriamente (com probabilidade uniforme) uma ação concreta no conjunto de ações fornecidas pela política abstrata.

Para obter um conjunto de ações deve-se realizar uma substituição θ nas variáveis do par abstrato $(\hat{\sigma}, \hat{\alpha})$. Entretanto pode ocorrer que a substituição θ gere um conjunto de ações concretas $A\theta$ que não pertencem ao conjunto de ações executáveis no estado concreto observado, i. e., $A\theta \cap A_\sigma = \emptyset$. Neste caso deve-se escolher uniformemente entre todas as ações concretas no conjunto A_σ .

Como exemplo, considere novamente o problema exemplo da Figura 5.1 e a política abstrata $\hat{\pi}_a$ (indicada na Figura 5.2 para a meta de alcançar a localidade 16). Se acaso o robô estiver no estado σ_{l5} e a política abstrata for utilizada para indicar uma ação, as seguinte substituições são válidas:

- $\theta = \{A/l5, B/l6\}$ e $(\text{in}(l5) \wedge \text{conn}(l5, l6) \wedge \text{isRoom}(l5))$ é VERDADEIRO.
- $\theta = \{A/l5, B/l3\}$ e $(\text{in}(l5) \wedge \text{conn}(l5, l3) \wedge \text{isRoom}(l5))$ é VERDADEIRO.
- $\theta = \{A/l5, B/l4\}$ e $(\text{in}(l5) \wedge \text{conn}(l5, l4) \wedge \text{isRoom}(l5))$ é VERDADEIRO.

Assim, a árvore fornece a política abstrata $\hat{\pi}_a = \{\text{goto_RD_CD}(A, B)\}$ e $\hat{\alpha} = \text{goto_RD_CD}(A, B) = \{\text{goto_RD_CD}(15, 16)\}$, o que resulta na indicação de $\alpha = \text{goto_RD_CD}(15, 16)$ para ser aplicada em σ_{l5} .

Deve-se observar que o algoritmo TILDE constrói uma FOLDT que contém apenas uma única ação abstrata na folha. Infelizmente, informações importantes são perdidas durante o processo de abstração. Devido a isto, a aplicação da política abstrata pode causar alguns problemas. No próximo capítulo são discutidos estes problemas.

6 Problemas da utilização da abstração relacional

A vantagem da utilização da política abstrata aprendida na resolução de um novo problema pode ser limitada entre dois extremos. De um lado se encontra a situação mais otimista onde o uso da política abstrata é de grande utilidade, sempre conseguindo levar o agente até a meta do problema. Neste caso $\forall \sigma_0 \in \mathcal{S}$ a meta $\sigma_g \in \mathcal{G}$ é sempre atingida. No outro extremo está o caso onde a utilização da política abstrata não consegue levar o agente até a meta.

No extremo mais otimista a política abstrata não apresenta nenhum problema de indeterminismo ou ambiguidade e pode ser aplicada diretamente no novo problema.

Indeterminismo ocorre quando a ação pode ser instanciada num conjunto de ações concretas. Um problema que pode ocorrer é haver ações no conjunto de ações instanciadas que levam o agente a situações (estados) completamente distintas em relação à meta.

Problemas de ambiguidade podem ser entendidos quando dois (ou mais) estados concretos completamente distintos são representados pelo mesmo estados abstratos e limitados pela mesma política abstrata. Neste caso a política abstrata, quando utilizada, vai ser correta para apenas um dos estados concretos.

No caso mais otimista a política abstrata é também a ótima para o novo problema. Um exemplo deste extremo foi apresentado no trabalho de Kersting et al. (2007), onde no domínio do Mundo dos Blocos foi aprendida a política abstrata ótima, para qualquer quantidade de blocos, na tarefa de empilhar um determinado bloco em cima de outro.

Já no extremo mais pessimista pode ser considerado que a política abstrata aprendida sempre vai indicar como política a pior ação possível, não levando o agente até a meta. Nesta situação, suponha que uma política tenha sido induzida a partir de um determinado problema de navegação robótica e ela vai ser aplicada

em outro problema. Este novo problema possui como política ótima a situação inversa à do primeiro problema, i. e., as ações ótimas a serem aplicadas nos estados do problema não pertencem ao conjunto de ações fornecidas pela política abstrata.

Como exemplo desta situação, suponha que no problema fonte (utilizado para induzir a política abstrata), na situação na qual o agente se encontra em uma CD (região do corredor próxima a uma porta), a ação ótima a ser aplicada fosse ir para uma RD (região de um cômodo próxima de uma porta); agora, já para o problema atual, a ação ótima seria ir para outra CD; e o mesmo ocorreria para todas as outras situações do problema. Isto implica que a utilização da política abstrata (induzida através de exemplos do problema fonte) não conseguiria levar, no problema atual, o agente até a meta.

Entre estes dois extremos existe a situação na qual a política abstrata não traz nenhuma vantagem em sua utilização em um novo problema. Neste caso, o conjunto de ações $\hat{\mathcal{A}}_\sigma$ fornecidas pela política abstrata para cada estado σ do problema é o mesmo conjunto de ações $\mathcal{A}_\sigma$ aplicáveis no estado, i. e., $\hat{\mathcal{A}}_\sigma = \mathcal{A}_\sigma$. Em vista deste fato, a política abstrata é equivalente a uma escolha aleatória de ações.

A seguir são apresentados mais detalhadamente os problemas que a abstração relacional pode provocar na navegação robótica. Basicamente devido a estes problemas decorre que a política abstrata, quando aplicada na navegação robótica, pode não ser ótima e pode não levar o agente até a meta do problema.

6.1 Instanciação inadequada

Conforme observado por Kersting et al. (2007), quando a política abstrata é instanciada, ela se torna não determinista. A árvore de decisão apresenta como política, para um estado, uma ação abstrata. A instanciação desta política abstrata pode apresentar um conjunto de ações concreta como política. Devido a este fato, deve-se escolher uma ação deste conjunto. Como não existe uma diferenciação entre estas ações, a escolha entre elas deve ser aleatória. Em função disto a política abstrata pode não ser ótima para o problema de navegação robótica.

Por exemplo, considere o problema apresentado pela Figura 6.1, e sendo a política abstrata dada pelo seguinte conjunto de regras:

$$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isCorridor}(A) \wedge \text{isRoom}(B) \xrightarrow{1.0} \text{goto_CD_RD}(A,B) \quad (6.1)$$

$$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isCorridor}(B) \xrightarrow{1.0} \text{goto_RD_CD}(A,B) \quad (6.2)$$

$$\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isRoom}(B) \xrightarrow{1.0} \text{goto_RD_RC}(A,B) \quad (6.3)$$

Suponha que o robô esteja em l1 e a meta do problema seja chegar em 17. Nesta situação, a instanciação da política abstrata ($\text{goto_CD_RD}(A,B)$) fornecida pelo conjunto de regras apresentará três ações concretas como política: $\text{goto_CD_RD}(11,12)$, $\text{goto_CD_RD}(11,13)$ e $\text{goto_CD_RD}(11,14)$. Destas ações a escolha correta seria ir para 14, pois é a localidade que leva para a meta do problema. Devido ao fato da política abstrata não conseguir diferenciar estas situações, a política abstrata para o problema de navegação não é ótima.

Figura 6.1: Situação exemplo para a demonstração de problemas que ocorrem durante a instanciação da política abstrata no problema de navegação robótica.

A meta para este caso é chegar na localidade 17.

O problema apresentado sempre pode ocorrer na navegação robótica quando a política abstrata fornecer, quando instanciada, mais de uma ação como política, sendo estas completamente distintas em relação à meta do problema. Este é um problema inerente da navegação robótica, pois não é possível de se diferenciar situações distintas existentes nos estados, como por exemplo a diferença existente entre as localidades 12, 13 e 14 da Figura 6.1. Isto não ocorre no problema do mundo dos blocos, pois os estados abstratos conseguem separar as diferentes situações distintas possíveis e a política, mesmo não determinista, é ótima (KERS-
TING et al., 2007).

6.2 Abstração inadequada

A abstração relacional tende a agrupar os estados de um problema em função das semelhanças estruturais destes, de modo que os estados semelhantes são representados pelo mesmo estado abstrato. Já a política dada a partir de um MDP que modela um problema é dada em função do valor das recompensas descontadas, i. e., a política para um estado é a ação que maximiza as recompensas futuras. A política a partir das recompensas descontadas pode ser vista como uma medida indireta da distância até a meta. Em função destes dois fatos a abstração relacional tende a agrupar estados que possuem semelhanças estruturais, mas que possuem políticas completamente distintas.

Devido a este agrupamento inadequado, a política apresentada para um determinado estado abstrato vai possuir problemas de ambiguidade. Durante a construção da árvore de decisão em lógica de primeira ordem (FOLDT) a ação abstrata (classe) que representa os exemplos de uma folha é escolhida em função de sua representatividade (ver Seção 4.2). No algoritmo TILDE apenas uma única ação abstrata é escolhida para representar os exemplos da folha. Neste caso, vai ser escolhida uma das ações abstratas que representa apenas uma das distintas situações existentes. Em vista disto, nenhuma outra informação é preservada a respeito das outras situações.

Por exemplo, suponha que num determinado problema de navegação (indicado na Figura 6.2.a) existam dois estados distintos σ_{11} e σ_{15} (respectivamente Figuras 6.2.b e 6.2.c). Em ambos o robô está em um CD conectado com três RD e um outro CD. A meta para este problema é atingir a localidade 17. No caso do estado σ_{11} , a ação ótima a ser tomada é `goto_CD_RD(11,14)`; já para o estado σ_{15} a ação ótima é `goto_CD_CD(15,11)`.

Durante a indução da FOLDT estas duas situações distintas vão ser representadas pelo mesmo estado abstrato, sendo que possuem políticas completamente distintas. Em vista deste fato, vão produzir uma política abstrata ambígua. Um exemplo (estado σ_{11} e ação `goto_CD_RD(11,14)`) vai indicar que a política abstrata deve indicar a transição de uma localidade CD para uma localidade RD; e o outro exemplo (estado σ_{15} e ação `goto_CD_CD(15,11)`) vai indicar que a política abstrata deve indicar a transição de uma localidade CD para outra localidade CD.

Esses dois exemplos vão contribuir igualmente para a escolha da ação abstrata representativa. Na indução de uma FOLDT pelo algoritmo TILDE apenas uma única ação abstrata vai ser escolhida para representar os exemplos existentes na

Figura 6.2: Problema exemplo para a demonstração da abstração inadequada, onde a meta é atingir a localidade 17. (a) Distintos estados, $\sigma_{l1} = \{\text{in}(11) \wedge \text{conn}(11,12) \dots\}$ e $\sigma_{l5} = \{\text{in}(15) \wedge \text{conn}(15,11) \dots\}$ vão ser representados pelo mesmo estado abstrato. No estado σ_{l1} (Figura 6.2.(b)) a ação ótima é `goto_CD_RD(11,14)` e no estado σ_{l5} (Figura 6.2.(c)) é `goto_CD_CD(15,11)`.

folha. Neste caso a ação abstrata a ser escolhida vai ser aquela que representar o maior número de exemplos. Em função disto, toda informação existente no RMDP que gerou os exemplos de indução vai ser perdida. Mais especificamente, toda a informação a respeito dos estados distintos (σ_{l1} e σ_{l5}) vai ser perdida.

Por consequência do problema de abstração temos que a política abstrata pode não conseguir guiar o agente até a meta do problema. Por não conseguir guardar todas as informações relevantes, a política abstrata não vai conseguir diferenciar situações distintas que ocorrem no problema no qual ela está sendo aplicada.

A política abstrata não consegue guardar todas as informações a respeito dos estados, sendo os estados abstratos apenas uma representação parcial dos

estados do problema concreto. Devido a esta representação parcial, a política abstrata não consegue apresentar uma decisão adequada para alguns dos estados do problema no qual a política abstrata está sendo aplicada (OTTERLO, 2009), sendo este um problema inerente a diversos domínios, incluindo o da navegação robótica.

Por exemplo, considerando novamente o problema da Figura 6.2, onde a meta é atingir a localidade 17; e a política abstrata indicada pelo conjunto de regras apresentado na Seção 6.1 (regras 6.1, 6.2 e 6.3).

Considere agora que a política, representada pelo conjunto de regras, vai ser aplicada para guiar o agente até a meta do problema. Para a abstração relacional as situações (estados) nas quais o agente se encontra na localidade 11 e na localidade 15 são idênticas, i. e., são representadas pelo mesmo estado abstrato. A política abstrata, representada pelo conjunto de regras, não consegue distinguir entre essas duas situações completamente distintas. No caso da localidade 11 a ação ótima é `goto_CD_RD(11,14)`; e `goto_CD_CD(15,11)` na localidade 15. Para ambas as situações a regra 6.1 vai fornecer como política a ação abstrata `goto_CD_RD(A,B)`.

Na localidade 11 esta ação abstrata é apropriada, pois no conjunto de ações concretas (`{goto_CD_RD(11,12), goto_CD_RD(11,13), goto_CD_RD(11,14)}`), instanciadas a partir desta ação abstrata, existe a ação (`goto_CD_RD(11,14)`) que leva até a meta do problema.

Já para o caso da localidade 15 esta ação abstrata fornecida pelo conjunto de regras é inapropriada. Ao ser instanciada num conjunto de ações concretas (`{goto_CD_RD(15,16), goto_CD_RD(15,18), goto_CD_RD(15,19)}`) a ação abstrata não consegue fornecer uma ação que leva até a meta 17 do problema. Para esta política abstrata o agente vai ficar preso no conjunto de estados $\{\sigma_5, \sigma_6, \sigma_8, \sigma_9\}$.

O Capítulo 7 a seguir apresenta as propostas deste trabalho para contornar os problemas apresentados neste presente Capítulo. A primeira proposta é a de um algoritmo para a construção de uma política abstrata estocástica, visando com isto reduzir a perda de informação no processo de abstração. A segunda proposta é a de um algoritmo que combina o aprendizado no nível concreto com uma indicação de preferência de política dada por uma política abstrata derivada para um outro problema similar. A terceira e última proposta oferece uma arquitetura que propicia a combinação do aprendizado tanto no nível concreto quanto no nível abstrato para acelerar o aprendizado.

7 Explorando a construção e uso de políticas abstratas na Navegação Robótica

Este Capítulo apresenta as propostas do trabalho. A primeira proposta (Seção 7.1) tem como intuito aperfeiçoar o algoritmo de construção da política abstrata TILDE visando com isto reduzir a perda de informação ocorrida no processo de abstração. Na modificação proposta (nomeada X-TILDE) é construída uma política abstrata estocástica. A segunda proposta (Seção 7.3) tem o objetivo realizar o aprendizado tanto no nível concreto quanto no abstrato, visando através desta combinação acelerar o aprendizado.

7.1 Gerando uma política abstrata estocástica

O processo de generalização provoca a perda de algumas informações importantes para resolver o problema, devido ao fato de que ele agrupa diferentes ações a serem aplicadas no mesmo estado abstrato. Por exemplo, no ambiente da Figura 3.1 onde a meta é atingir a localidade 12, o estado abstrato que agrupa conjuntamente os estados concretos definidos pelas localizações 15 e 18 conduzirá para a mesma folha da árvore (representada pela terceira folha da esquerda para a direita na Figura 7.1). Entretanto, os exemplos de treinamento fornecidos quando o robô está na localidade 15 indicam que a ação concreta ótima seria `goto_CD_RD(15,14)`, e quando o robô está na localidade 18 a ação concreta ótima seria `goto_CD_CD(18,15)`. Neste caso, durante a geração da folha, o algoritmo TILDE escolhe a ação abstrata que representa o maior número de exemplos, e devido a isto a distinção entre estas situações é perdida.

Ainda relacionado com o mesmo problema, se a ação abstrata gerada deste modo é usada para controlar diretamente a navegação robótica, ocorrerão casos nos quais a meta nunca será atingida. Para ilustrar este caso, considere novamente o ambiente apresentado na Figura 3.1 e a política abstrata apresentada na

Figura 7.1: FOLDT representando a política abstrata para o problema de navegação robótica, induzida pelo algoritmo TILDE usando os predicados indicados no Apêndice A e exemplos extraídos do problema da Figura 3.1.

Figura 7.1: esta política abstrata nunca irá dirigir o robô para a localidade alvo 12, pois sempre que ele chegar na localidade 18, a política abstrata irá guiá-lo para a localidade 17, ao invés da localidade 15 (que leva para a meta).

Devido a este fato foi proposto neste trabalho uma modificação no algoritmo de abstração TILDE que conseguisse contornar os problemas previamente mostrados. A modificação, chamada X-TILDE, habilita a construção de uma árvore que representa uma política abstrata estocástica. Cada folha contém com isto um conjunto de ações abstratas. Neste trabalho são propostas duas versões para o algoritmo X-TILDE: o P-TILDE (*Probabilistic TILDE*) no qual é associada uma probabilidade para a escolha de cada ação abstrata pertencente ao conjunto de ações abstratas de uma folha; e o ND-TILDE (*Nondeterministic TILDE*) no qual não existe preferência na escolha de uma ação abstrata do conjunto de ações abstratas de uma folha. Em Kersting et al. (2007) também foram notados alguns destes problemas que foram levantados em nosso trabalho, e para contornar-los, foi proposto naquele trabalho a utilização de uma estrutura auxiliar e algumas heurísticas no processo de utilização da política abstrata.

No nível da folha, o algoritmo TILDE pode se deparar com duas situações: i) as ações de todos os exemplos de treinamento correspondente ao mesmo estado abstrato indicam a mesma ação abstrata; ou ii) as ações de alguns dos exemplos sugerem outras ações abstratas. No primeiro caso, a escolha é trivial, enquanto no segundo caso, o algoritmo TILDE escolhe para associar com a folha a ação abstrata que ocorre com a maior frequência.

É aqui proposta a modificação do algoritmo TILDE para que menos in-

formação seja perdida. Ao invés de optar por associar uma única ação abstrata para cada folha da árvore (passo 6 do Algoritmo 3), propõe-se associar um conjunto de ações para cada folha (conforme indica na Figura 7.2). Nesta modificação o algoritmo P-TILDE mantém na folha da árvore as ações abstratas e suas respectivas frequências de ocorrência; essas frequências são utilizadas sempre na escolha da ação abstrata a ser usada, ação esta decorrente da utilização da política abstrata. De outro modo, o algoritmo ND-TILDE mantém na folha apenas as ações abstratas derivadas dos exemplos de treinamento, sendo que estas ações abstratas possuem chances iguais de serem escolhidas como política abstrata para um determinado estado abstrato.

Figura 7.2: Geração de uma folha nos algoritmos TILDE e X-TILDE. (a)

Número de exemplos e suas respectivas ações abstratas candidatas para a terceira folha (da esquerda para a direita) da FOLDT da Figura 7.1. (b) Folha gerada pelo algoritmo TILDE utilizando apenas a ação abstrata goto_CD_RD(A,D). (c) Folha gerada pelo algoritmo P-TILDE, com as duas ações abstratas, goto_CD_RD(A,D) e goto_CD_CD(A,C), e suas respectivas frequências de ocorrência. No caso do algoritmo ND-TILDE não há valor de frequência associado a cada uma das ações abstratas.

Os problemas previamente citados são contornados ao se utilizar o algoritmo X-TILDE. A política abstrata estocástica aprendida fornece duas ações abstratas como política para o estado abstrato que agrupa os estados definidos pelas localizações 15 e 18: goto_CD_RD(A,D) e goto_CD_CD(A,C), conforme apresentado na Figura 7.2.c. A árvore completa induzida pelo algoritmo X-TILDE utilizando os mesmos exemplos usados para a construção da árvore da Figura 7.1 é apresentada na Figura 7.3.

Ao utilizar a árvore da política abstrata para fornecer uma ação concreta, existem três maneiras distintas para o robô decidir qual ação concreta utilizar, sendo que esta escolha vai depender do algoritmo utilizado para a indução da FOLDT:

- TILDE: escolhe uniformemente entre todas as ações concretas definidas pela substituição θ na ação abstrata sugerida pela folha da FOLDT;

Figura 7.3: FOLDT induzida pelo algoritmo X-TILDE utilizando os mesmos exemplos usados na indução da árvore da Figura 7.1.

- P-TILDE: primeiro escolhe probabilisticamente entre todas as ações abstratas existentes na folha da FOLDT; e depois escolhe uniformemente entre todas as ações concretas definidas pela substituição θ na ação abstrata escolhida probabilisticamente;
- ND-TILDE: primeiro escolhe uniformemente entre todas as ações abstratas existentes na folha da FOLDT; e depois escolhe uniformemente entre todas as ações concretas definidas pela substituição θ na ação abstrata escolhida.

O Algoritmo 4 apresenta o algoritmo X-TILDE. Os procedimentos em comum com o algoritmo TILDE foram descritos no Capítulo 4.2. O algoritmo X-TILDE constrói uma FOLDT a partir de um conjunto de exemplos de treinamento E . Os testes candidatos são criados (passo 2 do algoritmo) a partir de um conjunto de operadores de refinamento previamente definidos por um especialista. Cada operador de refinamento gera um conjunto de sentenças complexas como testes candidatos para a divisão do conjunto de exemplos E . O melhor teste para dividir o conjunto de exemplos E é aquele que reduz a entropia. O teste ótimo é escolhido (passo 3) utilizando o *default gain ratio* (QUINLAN, 1993). Se acaso a partição induzida em E indicar que a divisão deve parar (procedimento **STOP_CRIT** no passo 5), uma folha é gerada. As folhas da árvore (passo 6) contêm sentenças atômicas que representam ações abstratas. Se mais de uma sentença atômica representa os exemplos remanescentes em E , o algoritmo TILDE escolheria (procedimento **leaf** do Algoritmo 3) aquela que representasse o maior número de exemplos, enquanto o algoritmo X-TILDE gera um conjunto de ações (procedimento **X_leaf** do Algoritmo 4). A variante ND-TILDE gera nas folhas uma política não de-

terminista, i. e., ND-TILDE associa a uma folha o conjunto de todas as ações abstratas indicadas pelos exemplos em E para o estado abstrato correspondente à respectiva folha. No caso do algoritmo P-TILDE cada ação abstrata tem associada a frequência em que aparece nos exemplos de E , sendo esta frequência utilizada na escolha da ação abstrata. Caso a partição induzida em E não indique que o processo de divisão deva parar, para cada partição induzida (passo 8) a função X-TILDE é recursivamente chamada (passo 9). Um nó interno é gerado utilizando τ como teste, e os dois filhos serão as subárvores induzidas por cada chamada ao X-TILDE (passo 9).

Algoritmo 4 Algoritmo de indução de árvore de decisão representada em lógica de primeira ordem X-TILDE.

```

1: function X-TILDE ( $E$ : conjunto de exemplos): retorna uma FOLDT
2:    $T = \text{GENERATE\_TESTS\_FOL}(E)$ 
3:    $\tau = \text{OPTIMAL\_SPLIT\_FOL}(T, E)$ 
4:    $\epsilon = \text{partição induzida em } E \text{ por } \tau$ 
5:   if STOP_CRIT( $E, \epsilon$ ) then
6:     return X_leaf(INFO_FOL( $E$ ))
7:   else
8:     for all  $E_j$  in  $\epsilon$  do
9:        $t_j = \text{X-TILDE}(E_j)$ 
10:    end for
11:    return inode( $\tau, (j, t_j)$ )
12:  end if
13: end function

```

7.2 Política abstrata no aprendizado por reforço

O processo de obtenção de uma política ótima através do aprendizado por reforço tem com deficiência o fato de ser custoso em relação ao tempo. Isso ocorre devido ao fato de que a obtenção da política ótima deve ser feita através de uma interação aleatória com o ambiente, i. e., para garantir a obtenção da política ótima deve-se garantir que todos os estados do problema sejam visitados um número suficiente de vezes. Para conseguir isto, deve-se recorrer a uma escolha aleatória de ações na exploração do ambiente.

O aprendizado da política ótima pelo algoritmo *Q-Learning* é esquematizado na Figura 7.4. Neste processo de aprendizado, a ação aplicada pelo agente é dada por uma escolha (probabilística) entre a ação indicada pela política π_Q obtida até o momento (com probabilidade $1 - \epsilon_1$) e uma ação indicada por uma política aleatória π_r , i. e., uma ação escolhida aleatoriamente (com probabilidade ϵ_1).

Devido a este fato foi proposto neste trabalho utilizar um conhecimento prévio

Figura 7.4: Processo de aprendizado da política ótima π^* pelo algoritmo *Q-Learning*.

obtido no processo de exploração do ambiente. O conhecimento prévio é representado através de uma política abstrata. Através da utilização deste conhecimento prévio a aleatoriedade da escolha de ações durante o aprendizado da política é reduzida e uma redução no tempo gasto para a obtenção da política ótima é esperada.

Para garantir a convergência do algoritmo de aprendizado *Q-Learning* a escolha de uma ação através política abstrata $\hat{\pi}_a$ é alternada com a escolha de uma ação através da política aleatória π_r . A Figura 7.5 apresenta o esquema do processo de aprendizado guiado pela política abstrata. O fator ϵ_1 continua sendo utilizado para balancear a exploração/exploração do ambiente, i. e., com probabilidade ϵ_1 realiza-se a exploração do ambiente e com probabilidade $1 - \epsilon_1$ realiza-se a exploração do ambiente. Na exploração do ambiente a política abstrata $\hat{\pi}_a$ vai ser escolhida para apresentar uma ação com probabilidade $1 - \epsilon_2$ e a política aleatória π_r vai ser escolhida para apresentar uma ação com probabilidade ϵ_2 .

Figura 7.5: Processo de aprendizado da política ótima π^* pelo algoritmo *Q-Learning* tendo a exploração guiada pela política abstrata $\hat{\pi}_a$.

Portanto, a escolha de uma ação pelo agente para a interação com o ambiente é dada por:

$$\pi_c(\sigma) = \begin{cases} \pi_a(\sigma) & \text{com probabilidade } \epsilon_1(1 - \epsilon_2), \\ \pi_r(\sigma) & \text{com probabilidade } \epsilon_1\epsilon_2, \\ \pi_Q(\sigma) & \text{com probabilidade } 1 - \epsilon_1, \end{cases} \quad (7.1)$$

onde: π_c é uma política determinista aplicada pelo agente no estado concreto σ ; π_a é uma política determinista induzida de $\hat{\pi}_a(\hat{\sigma})$ com $\sigma \preceq_{\theta} \hat{\sigma}$, π_r é a política aleatória e $0 \leq \epsilon_1 \leq 1$ e $0 \leq \epsilon_2 \leq 1$ são parâmetros que definem a alternância da exploração/explotação no processo de aprendizado.

7.3 Aprendizado por reforço simultâneo

Considere um problema de decisão concreto, que pode ser chamado de *problema fonte*, para o qual a política ótima foi calculada e uma política abstrata foi gerada para posterior utilização. Tendo agora um outro problema de decisão, o *problema alvo*, no qual a política abstrata vai ser instanciada para sua resolução. A esperança neste caso é que, ao utilizar a política abstrata, seja obtida uma política razoável para o problema alvo. Entretanto, não existem garantias disto. Devido a isto, a política inicial deve ser refinada pela exploração: deve-se aplicar o aprendizado por reforço no problema alvo concreto. Do ponto de vista do aprendizado por reforço, o atrativo desta idéia é a aceleração do aprendizado da política, visto que a política abstrata fornece uma iniciativa razoável para o processo.

Entretanto, apenas aplicar o aprendizado por reforço no problema alvo concreto não é o melhor uso possível dos dados de exploração disponíveis. Um dos desafios para o processo de abstração é capturar toda a informação que seja útil do problema fonte; provavelmente no problema alvo algumas recomendações da política abstrata estarão incorretas para alguns estados.

A idéia, portanto, é utilizar o aprendizado por reforço para refinar a política abstrata. Devido ao fato da política abstrata ser mais simples (possui uma menor quantidade de estados e ações) que a política concreta alvo, é esperado que a convergência para a política ótima seja mais rápida no nível abstrato. A partir disto pode-se ter uma arquitetura na qual o nível abstrato fornece uma política inicial para o nível concreto; o nível abstrato é rapidamente refinado, e continua a fornecer a exploração para o nível concreto; e o nível abstrato é finamente sintonizado para o problema alvo. O desafio nesta situação é criar uma arquitetura de

aprendizado onde exista a garantia de que este processo convirja para o ótimo no nível concreto. Neste trabalho é proposta uma arquitetura deste tipo, chamada ASAR, onde o aprendizado ocorre tanto no nível concreto quanto no abstrato e que garante a convergência da política para a ótima no nível concreto.

Devido ao fato de estarem sendo tratados problemas de planejamento, pode-se considerar que cada tarefa é representada por uma tupla $\langle \sigma_0, \sigma_g \rangle$, onde $\sigma_0 \in \mathcal{S}$ e $\sigma_g \in \mathcal{G}$. Define-se a partir disto

$$\begin{aligned} \Upsilon(\pi, \langle \sigma_0, \sigma_g \rangle) &= \{s_1 s_2 \dots s_n \text{ para algum } n | \\ &\quad s_1 = \sigma_0, s_n = \sigma_g, \\ &\quad T(s_i, a_i, s_{i+1}) \neq 0 \text{ para alguma } a_i \\ &\quad \text{na qual } \pi(s_i) \neq \emptyset\}, \end{aligned}$$

como o conjunto de todos os possíveis históricos de transição que começam no estado σ_0 , e, seguindo a política π , alcançam em algum momento o estado meta σ_g .

Por exemplo, com $\mathbf{P}$ e π^* (Tabela 5.2) definidos para o exemplo da Figura 5.1, tem-se $\Upsilon(\pi^*, \langle \sigma_3, \sigma_6 \rangle) = \{\sigma_3 \sigma_5 \sigma_6\}$ e $\Upsilon(\pi^*, \langle \sigma_2, \sigma_6 \rangle) = \{\sigma_2 \sigma_3 \sigma_5 \sigma_6, \sigma_2 \sigma_4 \sigma_5 \sigma_6\}$.

Nas discussões a seguir π^* é a política ótima para o conjunto de tarefas $\{\langle \sigma_0, \sigma_g \rangle\}$ no qual a meta $\sigma_g \in \mathcal{G}$ é fixada e $\sigma_0 \in \mathcal{S}$; π_r é uma política aleatória que para cada estado σ , escolhe uma ação com distribuição uniforme sobre o conjunto $\mathcal{A}_\sigma$; π_a é uma política (determinista) induzida no MDP concreto por uma política abstrata (não determinista) $\hat{\pi}_a$. Também se define π_Q como a política (determinista e concreta) que utiliza as estimativas da função Q para escolher uma ação de acordo com $\pi_Q(\sigma) = \arg \max_\alpha Q(\sigma, \alpha)$.

A interação do robô com o ambiente claramente ocorre no nível concreto, entretanto a experiência $\langle \sigma_t, \alpha_t, \sigma_{t+1}, r_{t+1} \rangle$ é utilizada em ambos, abstrato e concreto, para que os valores possam ser atualizados em ambos os níveis. Ambos os processos de aprendizado, abstrato e concreto, seguem a estratégia de exploração/explotação ϵ -greedy.

No caso do nível abstrato temos:

$$\hat{\pi}_a(\hat{\sigma}) = \begin{cases} \hat{\pi}_{a_s}(\hat{\sigma}) & \text{com probabilidade } \epsilon_a, \\ \hat{\pi}_{a_t}(\hat{\sigma}) & \text{com probabilidade } 1 - \epsilon_a, \end{cases} \quad (7.2)$$

onde: $\hat{\pi}_{as}$ é uma política abstrata induzida dos exemplos derivados do uso da política concreta no problema fonte, $\hat{\pi}_{at} = \arg \max_{\alpha} Q(\hat{\sigma}, \hat{\alpha})$ é a política aprendida no nível abstrato, e $0 \leq \epsilon_a \leq 1$ é um parâmetro que define a alternância entre a exploração/explotação no nível abstrato para a arquitetura proposta.

No nível concreto temos:

$$\pi_c(\sigma) = \begin{cases} \pi_a(\sigma) & \text{com probabilidade } \epsilon_1(1 - \epsilon_2), \\ \pi_r(\sigma) & \text{com probabilidade } \epsilon_1\epsilon_2, \\ \pi_Q(\sigma) & \text{com probabilidade } 1 - \epsilon_1, \end{cases} \quad (7.3)$$

onde: π_c é uma política determinista aplicada pelo agente no estado concreto σ ; π_a é uma política determinista induzida de $\hat{\pi}_a(\hat{\sigma})$ com $\sigma \preceq_{\theta} \hat{\sigma}$, π_r é a política aleatória e $0 \leq \epsilon_1 \leq 1$ e $0 \leq \epsilon_2 \leq 1$ são parâmetros que definem a alternância da exploração/explotação no nível concreto da arquitetura proposta.

De modo resumido, conforme ilustra a Figura 7.6, tem-se que, no nível concreto, aprende-se utilizando a política π_a e gradualmente ela é substituída pela política π_Q , enquanto utiliza a política π_r cada vez menos, apenas para garantir a exploração e convergência para a política ótima. No nível abstrato, aprende-se utilizando a política $\hat{\pi}_{as}$ induzida no problema fonte e gradualmente ela é substituída pela política $\hat{\pi}_{at}$ aprendida no nível abstrato do problema alvo, conforme o processo de aprendizado se desenvolve.

Figura 7.6: Arquitetura de aprendizado por reforço simultâneo ASAR.

Quando o agente inicia o processo de aprendizado, π_Q é igual a π_r , pois a função Q é usualmente inicializada com o mesmo valor para todos os pares estado-ação. Neste caso, para alguma tarefa $\langle \sigma_0, \sigma_g \rangle$ o agente irá considerar todas as transições em $\Upsilon(\pi_r, \langle \sigma_0, \sigma_g \rangle)$. Por outro lado temos $\Upsilon(\pi_a, \langle \sigma_0, \sigma_g \rangle) \subset$

$\Upsilon(\pi_r, \langle \sigma_0, \sigma_g \rangle)$, e se assumirmos que a política π_a foi aprendida considerando uma tarefa similar, então as ações que o agente irá considerar possuem a propriedade que costumava ser boa para resolver um problema similar. Devido a isto, espera-se que seja obtido um desempenho inicial melhor, i. e., um alto valor de R_t nos primeiros episódios.

O motivo pelo qual é necessário a mudança gradual para a política π_Q é que talvez seja possível ter $\Upsilon(\pi^*, \langle \sigma_0, \sigma_g \rangle) \subset \Upsilon(\pi_a, \langle \sigma_0, \sigma_g \rangle)$, i.e. π_a pode levar a uma política não ótima, ou no pior caso pode não resolver a tarefa. Com isto, a convergência do aprendizado por reforço pode ser adaptada para provar a convergência para o valor ótima, visto que sempre existe a garantia da exploração completa.

Para o aprendizado da política no nível abstrato deve-se definir o espaço de estados abstratos $\hat{\mathcal{S}}$. Isto é necessário devido à utilização de uma tabela para representar a função Q utilizada pelo algoritmo *Q-learning*. Para a construção do conjunto $\hat{\mathcal{S}}$ adotou-se um método sistemático de geração, onde o espaço de estados abstratos é gerado a partir do espaço de estados concretos $\mathcal{S}$.

Neste processo, para cada estado $\sigma \in \mathcal{S}$ é realizada uma substituição das constantes de σ por variáveis. Por exemplo, a Tabela 7.1 contém todos os estados definidos para o problema concreto exemplo da Figura 5.1. A partir do estado $\sigma_1 = \text{in}(11) \wedge \text{isRoom}(11) \wedge \text{isCenter}(11) \wedge \text{isRoom}(12) \wedge \text{isNearDoor}(12) \wedge \text{conn}(11,12)$, através da substituição das constantes $\{11,12\}$ por variáveis $\{A,B\}$ é obtido o estado abstrato $\hat{\sigma}_1 = \text{in}(A) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{conn}(A,B)$; e assim por diante. A lista completa de todos os estados abstratos que podem ser obtidos para o problema da Figura 5.1 utilizando este método se encontra na Tabela 7.2. Note que, neste problema, $\hat{\mathcal{S}} = \{\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3\}$ e que $\mathcal{S} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$ e, portanto, $|\hat{\mathcal{S}}| < |\mathcal{S}|$, o que mostra que o aprendizado no nível abstrato lida com um problema menos complexo.

Tabela 7.1: Conjunto de estados definidos para o problema exemplo da Figura 5.1.

Estado	Definição do estado
σ_1	$\text{in}(l1) \wedge \text{isRoom}(l1) \wedge \text{isCenter}(l1) \wedge \text{isRoom}(l2) \wedge \text{isNearDoor}(l2) \wedge \text{conn}(l1,l2)$
σ_2	$\text{in}(l2) \wedge \text{isRoom}(l2) \wedge \text{isNearDoor}(l2) \wedge \text{isRoom}(l1) \wedge \text{isCenter}(l1) \wedge \text{isCorridor}(l3) \wedge \text{isNearDoor}(l3) \wedge \text{isCorridor}(l4) \wedge \text{isNearDoor}(l4) \wedge \text{conn}(l2,l1) \wedge \text{conn}(l2,l3) \wedge \text{conn}(l2,l4)$
σ_3	$\text{in}(l3) \wedge \text{isCorridor}(l3) \wedge \text{isNearDoor}(l3) \wedge \text{isRoom}(l2) \wedge \text{isNearDoor}(l2) \wedge \text{isCorridor}(l4) \wedge \text{isNearDoor}(l4) \wedge \text{isRoom}(l5) \wedge \text{isNearDoor}(l5) \wedge \text{conn}(l3,l2) \wedge \text{conn}(l3,l4) \wedge \text{conn}(l3,l5)$
σ_4	$\text{in}(l4) \wedge \text{isCorridor}(l4) \wedge \text{isNearDoor}(l4) \wedge \text{isRoom}(l2) \wedge \text{isNearDoor}(l2) \wedge \text{isCorridor}(l3) \wedge \text{isNearDoor}(l3) \wedge \text{isRoom}(l5) \wedge \text{isNearDoor}(l5) \wedge \text{conn}(l4,l2) \wedge \text{conn}(l4,l3) \wedge \text{conn}(l4,l5)$
σ_5	$\text{in}(l5) \wedge \text{isRoom}(l5) \wedge \text{isNearDoor}(l5) \wedge \text{isRoom}(l6) \wedge \text{isCenter}(l6) \wedge \text{isCorridor}(l3) \wedge \text{isNearDoor}(l3) \wedge \text{isCorridor}(l4) \wedge \text{isNearDoor}(l4) \wedge \text{conn}(l5,l6) \wedge \text{conn}(l5,l3) \wedge \text{conn}(l5,l4)$
σ_6	$\text{in}(l6) \wedge \text{isRoom}(l6) \wedge \text{isCenter}(l6) \wedge \text{isRoom}(l5) \wedge \text{isNearDoor}(l5) \wedge \text{conn}(l6,l5)$

Tabela 7.2: Conjunto de estados abstratos derivados.

Estado abstrato	Definição do estado abstrato
$\hat{\sigma}_1$	$\text{in}(A) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{conn}(A,B)$
$\hat{\sigma}_2$	$\text{in}(A) \wedge \text{isRoom}(A) \wedge \text{isNearDoor}(A) \wedge \text{isRoom}(B) \wedge \text{isCenter}(B) \wedge \text{isCorridor}(C) \wedge \text{isNearDoor}(C) \wedge \text{isCorridor}(D) \wedge \text{isNearDoor}(D) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D)$
$\hat{\sigma}_3$	$\text{in}(A) \wedge \text{isCorridor}(A) \wedge \text{isNearDoor}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{isCorridor}(C) \wedge \text{isNearDoor}(C) \wedge \text{isRoom}(D) \wedge \text{isNearDoor}(D) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D)$

8 Experimentos

Duas etapas foram realizadas nos experimentos para avaliar o desempenho das propostas do trabalho. Numa primeira etapa as políticas abstratas geradas pelo algoritmo X-TILDE foram aplicadas para guiar o robô num problema de navegação robótica. Esta etapa visou avaliar o desempenho destas políticas para transferir o conhecimento entre problemas e corresponde à primeira etapa dos experimentos (Seção 8.1). Após esta etapa, a política abstrata mais promissora, i. e., a que apresentava o melhor desempenho, foi utilizada em conjunto com o algoritmo de aprendizado *Q-Learning*. Esta etapa (Seção 8.2) visava analisar o desempenho do algoritmo de aprendizado por reforço guiado por uma política abstrata. Num primeiro momento, o conjunto de experimentos (Seção 8.2.1) visou analisar o desempenho das políticas abstratas para acelerar o aprendizado de uma política no nível concreto. Num segundo momento desejou-se avaliar o desempenho dessas políticas quando elas são refinadas para o problema alvo. A arquitetura ASAR foi usada com uma política abstrata do problema fonte no aprendizado do problema alvo e da nova política abstrata (Seção 8.2.2).

Em todas as etapas foram utilizados dois mapas. O primeiro (Figura 8.1)¹ foi utilizado tanto para a indução das políticas abstratas quanto para validação dos resultados. A árvore de decisão representada em lógica de primeira ordem (FOLDT) construída pelo X-TILDE é apresentada na Figura 8.2². O segundo mapa (Figura 8.3) foi utilizado para análise do desempenho dos algoritmos. As configurações utilizadas nos algoritmos de indução e de aprendizado, respectivamente, estão indicadas nos Apêndices A e B.

¹Este mapa é o mesmo da Figura 3.1, reproduzido aqui para maior clareza do texto.

²Mesma FOLDT da Figura 7.3

Figura 8.1: Representação relacional para o problema utilizado na indução da política abstrata estocástica. Quadrados simbolizam centro de cômodos, e. g. $(\text{isRoom}(12) \wedge \text{isCenter}(12))$, triângulos simbolizam portas de cômodos, e. g. $(\text{isRoom}(14) \wedge \text{isNearDoor}(14))$, círculos simbolizam portas de corredores, e. g. $(\text{isCorridor}(15) \wedge \text{isNearDoor}(15))$, e pentágonos pretos simbolizam centros de corredores, e. g. $(\text{isCorridor}(11) \wedge \text{isCenter}(11))$.

Figura 8.2: Política abstrata estocástica induzida pelo algoritmo X-TILDE para o problema da Figura 8.1.

8.1 Avaliando a aplicação das políticas abstratas

Em todos os experimentos desta etapa foi utilizada a mesma FOLDT. Os exemplos utilizados para a indução da FOLDT foram gerados pela aplicação da política ótima do problema, onde a meta era conseguir navegar de qualquer localidade L_i para a localidade meta 12 do problema da Figura 8.1. A árvore induzida pelo algoritmo X-TILDE está presente na Figura 8.2. No caso do ND-TILDE é feita uma escolha uniforme entre todas as ações abstratas existentes numa folha. Para o P-TILDE a escolha entre as ações abstratas é dada probabilisticamente. As probabilidades para cada uma das folhas com mais de uma ação abstrata são

Figura 8.3: Problema alvo utilizado nos experimentos. Na primeira etapa dos experimentos são escolhidos aleatoriamente pares de localizações inicial e meta ($\langle \sigma_0, \sigma_g \rangle$). Na segunda etapa dos experimentos a localização inicial é 11 e a meta é a localidade 126. Já para a terceira etapa dos experimentos as localizações iniciais são 11 e 14; e a meta é a localidade 125.

apresentadas na Tabela 8.1.

Tabela 8.1: Probabilidades associadas para as ações abstratas nas folhas com mais de uma ação abstrata como política.

Folha (da esq. para a dir.)	Ações abstratas e probabilidades			
3 ^a	goto_CD_RD(A,D)	0.58	goto_CD_CD(A,C)	0.42
6 ^a	goto_RD_RC(A,D)	0.65	goto_RD_CD(A,C)	0.35
7 ^a	goto_RD_RC(A,D)	0.66	goto_RD_RD(A,D)	0.34

Os experimentos foram divididos em três partes: na primeira parte foi comparado o reaproveitamento do conhecimento no mesmo domínio e na mesma tarefa utilizados para a construção da política abstrata; na segunda foi considerado o mesmo domínio, mas foram definidas outras localidades como metas; e finalmente, na terceira parte, foi considerada a capacidade da política abstrata como meio de transferência do conhecimento de um problema fonte para guiar o robô em um novo ambiente.

Para cada uma das partes do experimento foram comparadas três abordagens. A primeira, nomeada **RANDOM**, é a inexistência da transferência do conhecimento, onde o robô é guiado por uma política aleatória $\pi_r(\sigma)$. Na segunda, nomeada **P-TILDE**, a transferência do conhecimento é dada pela política abstrata estocástica construída pelo algoritmo P-TILDE. E na última parte do experimento, nomeada **ND-TILDE**, é considerado a transferência do conhecimento dada pelo uso da política abstrata estocástica construída pelo algoritmo ND-TILDE.

A utilização da política abstrata em uma nova tarefa pode levar a situações que não existiam no problema utilizado para a indução da política abstrata. Devido a isto, a política abstrata pode não sugerir uma ação adequada para esta nova situação. Uma ocorrência deste problema se dá no segundo conjunto de experimentos no qual, para o estado abstrato $(\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isCenter}(B) \wedge \text{isCorridor}(B))$, é fornecida $\text{goto_CD_CC}(A,B)$ como ação abstrata. Esta ação abstrata é apropriada em quatro dos estados alvo (12, 13, 19 e 117), mas não é a ação correta no último (118). Nesta situação, o algoritmo de abstração não recebeu, durante a indução da política abstrata, um exemplo que cobrisse este caso. Toda vez que a localidade 115 for alcançada, a política abstrata irá guiar o robô para um centro de corredor (110 ou 119), e a meta correta (118) nunca será alcançada.

O mesmo tipo de problema ocorre no terceiro conjunto de experimentos. Neste caso, a política abstrata se depara com uma nova situação na qual, para o estado abstrato $(\text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isCenter}(B) \wedge \text{isCorridor}(B) \wedge \text{conn}(A,C) \wedge \text{isNearDoor}(C) \wedge \text{isCorridor}(C))$, a ação abstrata $\text{goto_CD_CC}(A,D)$ deve ser considerada para ser possível de se alcançar todos os centros de cômodos do ambiente. A política abstrata carece desta informação e, devido a isto, o ambiente é dividido em duas regiões distintas: a primeira englobando a maioria das localidades na parte superior do mapa e a segunda, as inferiores. Ambas as regiões são conectadas pelas localidades 112 e 133, mas não existe um caminho ligando uma localidade de uma região a outra localidade da outra região, i. e., existem apenas caminhos ligando localidades da mesma região.

Sendo N o número de transições necessárias para atingir a meta, então, para cada experimento, pode ser apresentada a distribuição cumulativa de N , i. e., $P(N \leq t)$, a média $\mu(N)$, o desvio padrão $\sigma(N)$ e o número E_{succ} de episódios no qual o robô conseguiu atingir com sucesso a meta, com o correspondente número total de episódios E_{tot} .

O objetivo da reutilização da política é minimizar a média $\mu(N)$ do número de passos necessários para se atingir a meta. Em vista disto, esse parâmetro foi utilizado para comparar o desempenho das abordagens.

Devido ao fato de que N vai depender da distância relativa de cada par $\langle \sigma_0, \sigma_g \rangle$ utilizado em cada episódio, foi utilizada como medida de desempenho a razão $\mu(N)_{X-TILDE} / \mu(N)_{RANDOM}$. Para esta medida, um número menor que um indica uma vantagem da política abstrata em relação à política aleatória.

Nos itens a seguir são apresentados os desempenhos obtidos para cada uma

Figura 8.4: Distribuição cumulativa para os experimentos no mesmo ambiente e com a mesma meta.

das políticas abstratas em cada uma das partes do experimento.

8.1.1 Mesmo ambiente e mesmo estado meta

Nesta parte foi comparado o desempenho das políticas abstratas em função da política aleatória nas mesmas condições que foram utilizadas para gerar os exemplos de treinamento para a indução da FOLDT. Para a medida do desempenho foram escolhidos cinco estados iniciais, e para cada um destes foram realizados 2000 episódios. A Figura 8.4 apresenta a distribuição cumulativa para esta parte do experimento. A Tabela 8.2 apresenta a média μ , o desvio padrão σ e a quantidade E_{succ} de episódios onde a meta foi atingida. Nesta parte do experimento foram realizados $E_{tot} = 10000$ episódios no total.

Tabela 8.2: Uso da política abstrata: média, desvio padrão e quantidade de episódios com sucesso para o mesmo ambiente e estado meta.

Política	μ	σ	E_{succ}
RANDOM	113.19	106.94	9986
P-TILDE	77.64	77.10	9998
ND-TILDE	68.76	64.91	9999

O desempenho para as políticas abstratas geradas pelos algoritmos P-TILDE e ND-TILDE, respectivamente, foram:

$$\frac{\mu(N)_{P-TILDE}}{\mu(N)_{RANDOM}} = 0.69 \text{ e } \frac{\mu(N)_{ND-TILDE}}{\mu(N)_{RANDOM}} = 0.60.$$

Figura 8.5: Distribuição cumulativa para os experimentos no mesmo ambiente e metas distintas.

8.1.2 Mesmo ambiente e diferentes metas

Esta segunda parte do experimento foi feita tendo em vista um primeiro exemplo de generalização e reaproveitamento. O mesmo ambiente utilizado (Figura 8.1) para a indução da política abstrata foi utilizado, porém ocorreu a variação da meta. Foram escolhidos como estados-meta os centros de cômodos restantes deste ambiente (13, 19, 117, 118); para cada meta foram realizados 2000 episódios com estados iniciais escolhidos aleatoriamente. A quantidade de episódios no total foi $E_{tot} = 18000$. A Figura 8.5 apresenta a distribuição cumulativa para esta parte do experimento. Os resultados para esta parte estão indicados na Tabela 8.3.

Tabela 8.3: Uso da política abstrata: média, desvio padrão e quantidade de episódios com sucesso para o mesmo ambiente e metas diferentes.

Política	μ	σ	E_{succ}
RANDOM	68.27	74.45	17992
P-TILDE	51.45	58.30	17999
ND-TILDE	48.11	51.59	18000

O desempenho para as políticas abstratas geradas pelos algoritmos P-TILDE e ND-TILDE, respectivamente, foram:

$$\frac{\mu(N)_{P-TILDE}}{\mu(N)_{RANDOM}} = 0.75 \text{ e } \frac{\mu(N)_{ND-TILDE}}{\mu(N)_{RANDOM}} = 0.70.$$

Figura 8.6: Distribuição cumulativa para os experimentos em um novo ambiente e novas metas.

8.1.3 Novo ambiente e novas metas

Nesta parte do experimento foi escolhido um conjunto de pares de estado inicial e estado meta. Os estados-meta foram escolhidos aleatoriamente entre os centro de cômodos deste novo ambiente. Os estados iniciais foram escolhidos aleatoriamente entre as localidades do ambiente. Para cada par $\langle \sigma_0, \sigma_g \rangle$ foram realizados 2000 episódios. A quantidade de episódios no total foi $E_{total} = 8000$. A Figura 8.6 apresenta a distribuição cumulativa para esta parte do experimento. A Tabela 8.4 apresenta os resultados para esta parte.

Tabela 8.4: Uso da política abstrata: média, desvio padrão e quantidade de episódios com sucesso para o novo ambiente e metas diferentes.

Política	μ	σ	E_{succ}
RANDOM	177.97	172.73	7971
P-TILDE	138.84	176.39	7901
ND-TILDE	120.49	155.25	7974

O desempenho para as políticas abstratas geradas pelos algoritmos P-TILDE e ND-TILDE, respectivamente, foram:

$$\frac{\mu(N)_{P-TILDE}}{\mu(N)_{RANDOM}} = 0.78 \text{ e } \frac{\mu(N)_{ND-TILDE}}{\mu(N)_{RANDOM}} = 0.68.$$

8.1.4 Análise dos resultados

Pelos resultados obtidos em cada uma das partes deste experimento, pode-se notar que a utilização das políticas abstratas apresenta desempenho melhor do que uma política aleatória. Isto pode ser entendido como uma transferência, mesmo que discreta, de conhecimento entre problemas semelhantes estruturalmente. Entre as duas políticas abstratas utilizadas, pode-se observar que o desempenho apresentado pela política construída pelo algoritmo ND-TILDE é superior ao da política construída pelo algoritmo P-TILDE.

8.2 Avaliando a aplicação das políticas abstratas no aprendizado por reforço

Os experimentos realizados nesta etapa visam demonstrar a eficácia das políticas abstratas, geradas pelos algoritmos propostos, para acelerar o aprendizado de uma política ótima para um determinado problema. Inicialmente um problema de navegação robótica, nomeado problema fonte, é resolvido e uma política extraída deste. A tarefa do robô para este problema era conseguir navegar de qualquer localidade Li para a localidade meta 12 do ambiente apresentado na Figura 8.1. Esta política aprendida foi abstraída, pelo algoritmo ND-TILDE, na política abstrata estocástica $\hat{\pi}_{as}$ apresentada na Figura 8.2. O algoritmo ND-TILDE foi utilizado nesta etapa dos experimentos por apresentar um melhor desempenho do que o P-TILDE, conforme pôde ser verificado no conjunto de experimentos anterior.

A estratégia ϵ -greedy foi utilizada para a convergência da função $\hat{Q}$. Para acelerar o aprendizado, a política abstrata gerada foi utilizada em substituição à exploração aleatória da estratégia ϵ -greedy. Na estratégia ϵ -greedy a política informada pela função $\hat{Q}$, $\pi_Q(\sigma) = \arg \max_{\alpha} \hat{Q}(\sigma, \alpha)$, é escolhida com uma probabilidade $1 - \epsilon$ para apresentar uma ação para um estado σ . Com uma probabilidade ϵ é utilizada uma política aleatória $\pi_r(\sigma)$, i. e., é feita uma escolha aleatória de uma ação $\alpha \in A_{\sigma}$.

Essa substituição tem como objetivo fazer uma exploração guiada pela política abstrata. Através desta exploração guiada, apenas as ações mais relevantes (apresentadas pela política abstrata) são escolhidas para serem aplicadas, visando com isto ter uma redução significativa dos passos executados no aprendizado da política ótima. Para garantir a convergência da função $\hat{Q}$, a utilização da exploração guiada é alternada com a política $\pi_r(\sigma)$.

No primeiro conjunto de experimentos (Seção 8.2.1) a política abstrata aprendida através do problema fonte foi utilizada para guiar a exploração no problema alvo, sendo que não era realizado nenhum refinamento desta política para o problema alvo. Já no segundo conjunto de experimentos (Seção 8.2.2) a política abstrata foi refinada pelo aprendizado por reforço. O refinamento da política abstrata foi realizado através da utilização da arquitetura ASAR.

8.2.1 Aceleração do aprendizado através da utilização de políticas abstratas

Os experimentos desta etapa focaram em aprender a política ótima para o problema de navegação apresentado na Figura 8.3. Neste experimento foi considerada a tarefa $task_1 = \langle \sigma_{l1}, \sigma_{l26} \rangle$, sendo que para a análise foi considerada a média do número de transições (ou passos) NT necessárias para se alcançar o estado meta em cada episódio. Essa tarefa foi repetida 1000 vezes e a média foi calculada para cada episódio.

Nesta etapa foram comparadas duas abordagens. A primeira é uma utilização direta do algoritmo *Q-Learning* com a técnica ϵ -greedy, onde a política aleatória π_r é utilizada como estratégia de exploração. A segunda é a utilização do algoritmo *Q-learning* com a técnica ϵ -greedy onde a política de exploração é dada pela alternância entre a política aleatória $\pi_r(\sigma)$ e a política abstrata $\pi_a(\sigma)$ induzida pelo problema fonte. Portanto, a política $\pi_c(\sigma)$ no nível concreto é dada por:

$$\pi_c(\sigma) = \begin{cases} \pi_a(\sigma) & \text{com probabilidade } \epsilon_1(1 - \epsilon_2), \\ \pi_r(\sigma) & \text{com probabilidade } \epsilon_1\epsilon_2, \\ \pi_Q(\sigma) & \text{com probabilidade } 1 - \epsilon_1. \end{cases}$$

Para aproveitar toda a vantagem da informação disponível na política abstrata, considerou-se que a ação abstrata sugerida pela política abstrata não deveria ser sugerida novamente por π_a no mesmo episódio. Se durante um episódio o agente estiver num estado σ e se a política abstrata sugerir a ação $\alpha \in \mathcal{A}_\sigma$, então na próxima vez, durante o mesmo episódio, que o agente se encontrar no estado σ e utilizar π_a para escolher uma ação a ser tomada, o conjunto de ações concretas possíveis sugeridas será reduzido a $\mathcal{A}_\sigma - \{\alpha\}$.

Considerou-se que para a $task_1$ a política abstrata pode guiar de uma forma subótima o agente robótico até o estado meta. Em vista disto, inicia-se $\epsilon_1 = 1$ e $\epsilon_2 = 0$ e, para cada episódio esses fatores são: ϵ_1 decrementado de 0.05 e ϵ_2 incrementado de 0.06. Isto foi feito visando aproveitar nos episódios iniciais, onde

a política π_Q não possui nenhuma informação relevante, o máximo possível das informações existentes na política abstrata π_a .

Os resultados dos experimentos para os primeiros 10 episódios estão indicados na Figura 8.7, onde *com* π_a corresponde a abordagem que utiliza a política abstrata na exploração e *sem* π_a corresponde a abordagem que utiliza diretamente o algoritmo *Q-Learning* com a exploração dada por π_r .

Figura 8.7: Número médio de transições NT necessárias para alcançar a meta, com e sem a utilização da política abstrata π_a .

A estratégia que utiliza π_a inicia o processo de aprendizado com uma média de transições menor e gradualmente vai decrescendo este valor devido ao processo de aprendizado. Já a estratégia que utiliza apenas π_r começa com um alto valor para a média de transições. Esta diferença no começo da curva de aprendizado mostra que a política abstrata pode transferir conhecimento entre problemas, o que implica num desempenho inicial melhor.

Durante o processo de aprendizado a distância entre as curvas é reduzida (aproximadamente no quinto episódio). Isto ocorre devido ao fato de que π_a é gradualmente substituída por π_r durante o processo de aprendizado (através do incremento de ϵ_2) para garantir a convergência da estimativa da função Q .

8.2.2 Aceleração do aprendizado através do aprendizado simultâneo

Os experimentos desta etapa focaram em aprender a política ótima para o problema de navegação apresentado na Figura 8.3. Duas tarefas foram executadas pelo robô: $task_1 = \langle \sigma_1, \sigma_{25} \rangle$ e $task_2 = \langle \sigma_4, \sigma_{25} \rangle$. Num primeiro momento, $task_1$, desejou-se analisar o desempenho dos algoritmos para uma situação na qual a política abstrata fornecida conseguia guiar o robô, mesmo que de uma forma não ótima, para a meta da tarefa. Desta forma era esperado que o desempenho dos algoritmos que utilizam a política abstrata para guiar a exploração, conforme os resultados apresentados na seção 8.1, seria melhor. Num segundo momento, $task_2$, analisou-se o desempenho para a situação na qual a política não consegue guiar o robô até a meta para o problema. Neste caso, era esperado que o desempenho do algoritmo que utilizasse a política abstrata π_a fornecida fosse pior do que um que utilizasse uma política aleatória π_r para guiar o robô na exploração.

As tarefas eram repetidamente resolvidas aplicando o aprendizado por reforço. Cada tarefa era formada por 4000 passos, onde cada passo é uma transição de um estado para outro. Toda vez que o robô atingia a meta, ele era reposicionado na localidade inicial.

Para cada uma das tarefas foram comparadas três abordagens. A primeira, nomeada **ALET**, é uma utilização direta do algoritmo *Q-learning* com a técnica ϵ -greedy, onde a política aleatória π_r é utilizada como estratégia de exploração. Neste caso, $\pi_Q(\sigma)$ é utilizada com probabilidade $1 - \epsilon_1$ enquanto $\pi_r(\sigma)$ com probabilidade ϵ_1 . A segunda, nomeada **ND**, é a utilização do algoritmo *Q-learning* com a técnica ϵ -greedy onde a política de exploração é dada pela alternância entre a política aleatória $\pi_r(\sigma)$ e a política abstrata $\pi_a(\sigma)$ induzida pelo problema fonte. Portanto, a política $\pi_c(\sigma)$ no nível concreto é dada por:

$$\pi_c^2(\sigma) = \begin{cases} \pi_a^2(\sigma) & \text{com probabilidade } \epsilon_1(1 - \epsilon_2), \\ \pi_r^2(\sigma) & \text{com probabilidade } \epsilon_1\epsilon_2, \\ \pi_Q^2(\sigma) & \text{com probabilidade } 1 - \epsilon_1. \end{cases}$$

A terceira abordagem, nomeada **QSA**, utiliza a arquitetura ASAR que alterna no nível abstrato entre a política abstrata $\pi_{as}(\sigma)$ aprendida de um problema similar com uma política $\pi_{at}(\sigma)$ que está sendo aprendida no novo problema. Enquanto as abordagens **ALET** e **ND** aplicam o algoritmo *Q-learning* apenas no nível concreto, a **QSA** utiliza o aprendizado por reforço tanto no nível concreto quanto no abstrato. Para este caso, a política $\pi_c^3(\sigma)$ no nível concreto é dada por:

$$\pi_c^3(\sigma) = \begin{cases} \pi_{a_s}^3(\sigma) & \text{com probabilidade } \epsilon_1(1 - \epsilon_2)\epsilon_a, \\ \pi_{a_t}^3(\sigma) & \text{com probabilidade } \epsilon_1(1 - \epsilon_2)(1 - \epsilon_a), \\ \pi_r^3(\sigma) & \text{com probabilidade } \epsilon_1\epsilon_2, \\ \pi_Q^3(\sigma) & \text{com probabilidade } 1 - \epsilon_1. \end{cases}$$

Na construção da tabela Q-abstrata para o problema da Figura 8.3 foram utilizados os estados abstratos indicados na Tabela 8.5. Estes estados foram criados de acordo com o descrito na Seção 7.3. A utilização destes estados abstratos reduz o espaço de estados ($|\mathcal{S}| = 33$ e $|\hat{\mathcal{S}}| = 12$). Para as constantes ϵ foram utilizados os valores: $\epsilon_1 = 0.5$, $\epsilon_2 = 0.3$ e $\epsilon_a = 0.7$.

Para aproveitar toda a vantagem da informação disponível na política abstrata, considerou-se que a ação abstrata sugerida pela política abstrata não deveria ser sugerida novamente por π_a no mesmo episódio. Se durante um episódio o agente estiver num estado σ e se a política abstrata sugerir a ação $\alpha \in \mathcal{A}_\sigma$, então na próxima vez, durante o mesmo episódio, que o agente se encontrar no estado σ e utilizar π_a para escolher uma ação a ser tomada, o conjunto de ações concretas possíveis sugeridas será reduzido a $\mathcal{A}_\sigma - \{\alpha\}$.

Os resultados para as duas tarefas ($task_1$ e $task_2$) estão apresentadas respectivamente pelas Figuras 8.8 e 8.9. Cada tarefa foi repetida 40 vezes. A cada 250 passos a política obtida até o momento foi avaliada. Na avaliação eram realizados 30 episódios de validação. Portanto, cada ponto do gráfico é a média da avaliação de 40 tarefas (sendo cada ponto a média do total de 1200 medidas). Em cada episódio de validação, apenas a política $\pi_Q(\sigma)$ obtida até o momento foi utilizada para guiar o robô. A distância horizontal entre dois pontos do gráfico corresponde a um intervalo de 250 passos.

No caso da primeira tarefa, a abordagem **ND** converge em aproximadamente 3250 passos, enquanto **QSA** converge em aproximadamente 2750. Para esta tarefa a abordagem **ALET** não permitiu a convergência no limite de 4000 passos imposto. Para a segunda tarefa, observa-se que **ND** inicialmente tem um desempenho degradado, em relação à **ALET**, devido ao efeito da política π_a^2 que não leva em conta as novas características do problema alvo. Entretanto, este baixo desempenho é gradualmente corrigido através do aprendizado, levando **ND** a ter um desempenho melhor do que **ALET**.

A abordagem **QSA** tem um desempenho visivelmente melhor sobre **ALET** e **ND** em ambas as tarefas, devido ao efeito da combinação do aprendizado tanto no nível concreto quanto no nível abstrato. Particularmente devido ao fato desta

abordagem conseguir refinar a política no nível abstrato, o que faz com que a política abstrata seja melhorada ao longo do tempo e um número menor de passos seja necessário para a obtenção da política ótima no nível concreto.

Tabela 8.5: Estados abstratos definidos para a tabela Q-Abstrata do problema alvo.

Estados abstratos
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D) \wedge \text{conn}(A,E) \\ & \wedge \text{isCorridor}(A) \wedge \text{isNearDoor}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \\ & \wedge \text{isRoom}(C) \wedge \text{isNearDoor}(C) \wedge \text{isCorridor}(D) \wedge \text{isCenter}(C) \\ & \wedge \text{isCorridor}(E) \wedge \text{isNearDoor}(E) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D) \wedge \text{isCorridor}(A) \\ & \wedge \text{isNearDoor}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{isCorridor}(C) \\ & \wedge \text{isNearDoor}(C) \wedge \text{isCorridor}(D) \wedge \text{isCenter}(C) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D) \\ & \wedge \text{isCorridor}(A) \wedge \text{isNearDoor}(A) \wedge \text{isCorridor}(B) \wedge \text{isNearDoor}(B) \\ & \wedge \text{isCorridor}(C) \wedge \text{isNearDoor}(C) \wedge \text{isRoom}(D) \wedge \text{isNearDoor}(D) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D) \wedge \text{isRoom}(A) \\ & \wedge \text{isNearDoor}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{isCorridor}(C) \\ & \wedge \text{isNearDoor}(C) \wedge \text{isRoom}(D) \wedge \text{isCenter}(D) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D) \wedge \text{isCorridor}(A) \\ & \wedge \text{isCenter}(A) \wedge \text{isCorridor}(B) \wedge \text{isNearDoor}(B) \wedge \text{isCorridor}(C) \\ & \wedge \text{isNearDoor}(C) \wedge \text{isCorridor}(D) \wedge \text{isNearDoor}(D) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{conn}(A,D) \wedge \text{isRoom}(A) \\ & \wedge \text{isNearDoor}(A) \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{isRoom}(C) \\ & \wedge \text{isNearDoor}(C) \wedge \text{isRoom}(D) \wedge \text{isCenter}(D) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{isRoom}(A) \wedge \text{isNearDoor}(A) \\ & \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{isRoom}(C) \wedge \text{isCenter}(C) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{isRoom}(A) \wedge \text{isNearDoor}(A) \\ & \wedge \text{isRoom}(B) \wedge \text{isCenter}(B) \wedge \text{isCorridor}(C) \wedge \text{isNearDoor}(C) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A) \\ & \wedge \text{isRoom}(B) \wedge \text{isNearDoor}(B) \wedge \text{isRoom}(C) \wedge \text{isNearDoor}(C) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{conn}(A,C) \wedge \text{isCorridor}(A) \wedge \text{isCenter}(A) \\ & \wedge \text{isCorridor}(B) \wedge \text{isNearDoor}(B) \wedge \text{isCorridor}(C) \wedge \text{isNearDoor}(C) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isCorridor}(A) \wedge \text{isCenter}(A) \wedge \text{isCorridor}(B) \\ & \wedge \text{isNearDoor}(B) \end{aligned}$
$\begin{aligned} & \text{in}(A) \wedge \text{conn}(A,B) \wedge \text{isRoom}(A) \wedge \text{isCenter}(A) \wedge \text{isRoom}(B) \\ & \wedge \text{isNearDoor}(B) \end{aligned}$

Figura 8.8: Média de passos para alcançar a meta. O resultado é uma média de 40 repetições da tarefa. Cada execução da tarefa foi limitada em 4000 passos. A cada 250 passos, a política obtida até o momento foi avaliada pela execução de 30 episódios de validação. Portanto, cada ponto corresponde a uma média de 1200 episódios de validação. 11 é a localidade inicial para o robô e a meta para o problema é alcançar a localidade 125.

Figura 8.9: Média de passos para alcançar a meta. O resultado é uma média de 40 repetições da tarefa. Cada execução da tarefa foi limitada em 4000 passos. A cada 250 passos, a política obtida até o momento foi avaliada pela execução de 30 episódios de validação. Portanto, cada ponto corresponde a uma média de 1200 episódios de validação. 14 é a localidade inicial para o robô e a meta para o problema é alcançar a localidade 125.

9 Conclusões

Este trabalho apresentou a proposta de uma nova arquitetura, chamada ASAR, para o aprendizado por reforço onde ocorre simultaneamente o aprendizado tanto no nível concreto quanto no nível abstrato. Nesta arquitetura híbrida o nível abstrato é utilizado para guiar a exploração do ambiente e acelerar o aprendizado. O nível abstrato possui um componente herdado de um problema de decisão anteriormente resolvido. Para a reutilização do conhecimento no nível abstrato, este faz uso de uma política abstrata representada na forma de uma FOLDT.

A FOLDT é construída pelo algoritmo X-TILDE. Esse algoritmo foi proposto neste trabalho para contornar os problemas apresentados pelo algoritmo TILDE original. O algoritmo TILDE constrói a FOLDT contendo apenas uma ação abstrata em cada folha da árvore, ou seja, constrói uma política determinista. O algoritmo X-TILDE propõe uma modificação no algoritmo TILDE de forma que a FOLDT passa a ter um conjunto de ações abstratas para cada folha da árvore. Através desta modificação a FOLDT passa a representar uma política abstrata estocástica. A construção da FOLDT pelo algoritmo X-TILDE reduz a perda de informação provocada no processo de abstração. Ao manter a informação sobre situações distintas ocorridas no problema utilizado para a indução da FOLDT, o algoritmo X-TILDE possibilita a resolução de uma quantidade maior de tarefas.

A qualidade da política abstrata gerada pelo algoritmo TILDE depende da estrutura do problema e da representação relacional escolhida. Considerando que o algoritmo TILDE não fornece uma política que possa resolver a maioria das tarefas apresentadas, os experimentos mostraram que a nossa proposta X-TILDE pode resolver uma variedade maior de problemas (Seção 8.1). Os experimentos realizados demonstraram que a política gerada pelo X-TILDE possui um desempenho melhor do que uma política aleatória. Isto pode ser visto como transferência do aprendizado entre problemas. O conhecimento do primeiro problema foi abstraído/inserido na política abstrata e utilizado no segundo problema para reduzir a quantidade de passos necessários para se atingir a meta. Nestes experimentos foram utilizados problemas considerados semelhantes, i. e., ambientes

que possuíam algumas semelhanças estruturais.

Nos experimentos realizados o desempenho do algoritmo P-TILDE é pior que o desempenho do algoritmo ND-TILDE. Acreditamos que isto ocorra porque o algoritmo P-TILDE classifica os estados baseando-se no conjunto de exemplos de indução, mas tende a agrupar os estados em relação às descrições relacionais. Em vista disto, a probabilidade associada à escolha de uma ação como política vai estar atrelada à frequência dos exemplos de indução e, na maioria dos casos, pode não corresponder a mesma distribuição de frequências do problema alvo que está sendo tratado. Pelo fato de que o algoritmo ND-TILDE não leva em consideração a distribuição de ocorrência das ações, os resultados mostraram que uma proposta mais conservadora é melhor. Por outro lado, a política aleatória, que é a mais conservadora existente, teve um desempenho pior. O conjunto de ações $\mathcal{A}_{ND-TILDE}$ fornecidas pela política do algoritmo ND-TILDE é um subconjunto das ações $\mathcal{A}_{ALET}$ fornecidas pela política aleatória, i. e., $\mathcal{A}_{ND-TILDE} \subseteq \mathcal{A}_{ALET}$. Devido a isto, o desempenho obtido pela política do algoritmo ND-TILDE, na maioria dos casos, tende a ser melhor que o desempenho obtido utilizando a política aleatória. Para achar um compromisso melhor entre a abordagem conservadora (ND-TILDE) e a abordagem probabilística (P-TILDE) deve-se avaliar cuidadosamente a política abstrata. Uma vez que a propriedade de Markov não é mais válida no domínio relacional da navegação robótica, o aprendizado do gradiente da política pode ser uma boa alternativa (GOSAVI, 2009).

A motivação principal na reutilização da política em navegação robótica é conseguir guiar o robô na resolução de problemas similares. Se a política abstrata aprendida não for perfeita, o agente deve aprender a partir do conhecimento que foi transferido. Neste caso, a política abstrata pode ser utilizada como uma heurística para guiar a exploração nos algoritmos de aprendizado por reforço. Esta combinação visa acelerar o processo do aprendizado de uma política ótima na tarefa alvo.

O principal objetivo deste trabalho foi a combinação da política abstrata com um algoritmo de aprendizado por reforço para acelerar o aprendizado da política concreta. Apesar da nossa proposta X-TILDE ser mais robusta para a transferência do conhecimento, ela não garante a resolução de qualquer tarefa, podendo ocorrer que algumas das metas não consigam ser atingidas. A eficácia da política abstrata vai depender da experiência obtida na tarefa fonte utilizada para sua indução, e pode falhar em situações que não ocorreram na tarefa fonte. Em vista deste fato foi nesta combinação que se deu a principal contribuição deste trabalho, com a proposta da arquitetura ASAR para a atualização simultânea

das políticas abstrata e concreta onde é levado em conta o conhecimento prévio, os dados advindos da exploração, e que garante a convergência para a política concreta ótima.

Os experimentos realizados na Seção 8.2.2 conseguiram mostrar a eficiência de nossas propostas para acelerar o aprendizado da política concreta em um novo problema. Em um primeiro momento mostrou-se que a utilização da política abstrata estocástica construída pelo algoritmo X-TILDE tem um desempenho melhor em comparação à utilização de apenas uma política aleatória. Esse desempenho é melhor até mesmo em situações nas quais a política abstrata estocástica não conseguia guiar o agente até a meta do problema. Em segundo lugar mostrou-se que a nossa arquitetura proposta, que efetua o aprendizado tanto no nível concreto quanto no abstrato, consegue resolver situações que não existiam na tarefa fonte. Com isto, o desempenho da arquitetura proposta foi superior tanto em situações que a política abstrata estocástica conseguia resolver quanto em situações nas quais ela não conseguia resolver.

As contribuições deste trabalho são importantes para a teoria de abstração, visto que elas mostram como a abstração pode ser útil na prática. Além disto, é de grande valor para a teoria do aprendizado por reforço, pois o trabalho mostra como acelerar o aprendizado utilizando políticas previamente obtidas em conjunto com a abstração destas. Neste processo ocorre uma significativa redução do número de passos necessários para se obter a política ótima.

9.1 Sugestões de trabalhos futuros

Algumas extensões interessantes deste trabalho podem ser realizadas como trabalhos futuros, relacionadas tanto com o processo de abstração quanto com o processo de utilização da política abstrata.

Em relação ao processo de abstração, pode ser investigado como um conjunto de políticas obtidas de problemas menores pode ser utilizado para a indução de uma única política abstrata que englobe as informações distintas existentes em cada um destes problemas menores. Cada política obtida de um problema menor poderia ter uma situação não representada pelas outras e, quando combinadas, poderiam produzir uma política abstrata que resolvesse todos os problemas menores. A produção de uma política abstrata a partir de várias outras menores tem o atrativo de que a resolução em separado de cada um dos problemas menores tende a ser menos custosa do que a resolução de um problema maior que repre-

sente todas as situações presentes nos problemas menores. Um problema que pode ocorrer neste caso é a superespecialização (*overfitting*) da política abstrata. Neste caso a política abstrata poderia acabar representando todas as ações existentes no domínio de navegação robótica. Devido a isto, não vai haver grande abstração, o que pode comprometer a transferência de conhecimento entre problemas.

Outra extensão interessante que poderia ser feita seria a combinação da abstração relacional, que agrega os estados em função da sua informação estrutural, com trabalhos no qual a agregação de estados se dá em função da função valor (DRUMMOND, 2002). A abstração relacional, devido a sua forma de agregação de estados, tende a não distinguir estados distintos em relação à política. Em função disto, a abstração relacional poderia ser combinada com técnicas de agregação de estados que utilizam a função valor, visando com isto melhorar o desempenho da abstração relacional.

Além destas extensões, um conjunto de experimentos poderia ser realizado visando analisar o desempenho das propostas em problemas maiores e em outros domínios. Em particular, em relação aos experimentos conduzidos neste trabalho, o desempenho das propostas poderia ser analisado em função de uma variação dos valores ϵ_1 , ϵ_2 e ϵ_a e do decaimento destes parâmetros ao longo dos episódios, analisando mais profundamente o impacto causado pela dinâmica do chaveamento entre políticas durante o aprendizado.

Referências

- BIANCHI, R. A. C.; RIBEIRO, C. H. C.; COSTA, A. H. R. Heuristic selection of actions in multiagent reinforcement learning. In: *IJCAI*. [S.l.: s.n.], 2007. p. 690–695.
- BIANCHI, R. A. C.; RIBEIRO, C. H. C.; COSTA, A. H. R. Accelerating autonomous learning by using heuristic selection of actions. *Journal of Heuristics*, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 2, p. 135–168, 2008. ISSN 1381-1231.
- BLOCKEEL, H.; RAEDT, L. D. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, v. 101, p. 285–297, May 1998. ISSN 0004-3702.
- BURKOV, A.; CHAIB-DRAA, B. Adaptive play q-learning with initial heuristic approximation. In: *ICRA*. [S.l.: s.n.], 2007. p. 1749–1754.
- COCORA, A. et al. Learning relational navigation policies. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.: s.n.], 2006.
- DRUMMOND, C. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, v. 16, p. 59–104, February 2002. ISSN 1076-9757.
- DZEROSKI, S.; RAEDT, L. D.; DRIESSENS, K. Relational reinforcement learning. *Machine Learning*, v. 43, p. 7–52, 2001.
- GOSAVI, A. Reinforcement learning: A tutorial survey and recent advances. *INFORMS JOURNAL ON COMPUTING*, v. 21, n. 2, p. 178–192, 2009. Disponível em: <<http://joc.journal.informs.org/cgi/content/abstract/ijoc.1080.0305v1>>.
- KERSTING, K.; OTTERLO, M. V.; RAEDT, L. D. Bellman goes relational. In: BRODLEY, C. E. (Ed.). *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*. Banff, Alberta, Canada: ACM, 2004. p. 465–472.
- KERSTING, K. et al. Learning to transfer optimal navigation policies. *Advanced Robotics: Special Issue on Imitative Robots*, v. 21, n. 13, p. 1565–1582, 2007.
- KNOX, W. B.; STONE, P. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In: *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*. [S.l.: s.n.], 2010.
- LANE, T.; WILSON, A. Toward a topological theory of relational reinforcement learning for navigation tasks. In: *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*. [S.l.: s.n.], 2005.

- MADDEN, M. G.; HOWLEY, T. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, Kluwer Academic Publishers, Norwell, MA, USA, v. 21, p. 375–398, June 2004. ISSN 0269-2821. Disponível em: <<http://portal.acm.org/citation.cfm?id=1013650.1013676>>.
- MITCHELL, T. M. *Machine Learning*. 1. ed. [S.l.]: McGraw-Hill Science/Engineering/Math, 1997. Hardcover. ISBN 0070428077.
- OTTERLO, M. V. *The logic of adaptive behavior : knowledge representation and algorithms for the Markov decision process framework in first-order domains*. Tese (Doutorado) — University of Twente, Enschede, May 2009. Disponível em: <<http://doc.utwente.nl/58987/>>.
- OTTERLO, M. van. Reinforcement learning for relational MDPs. In: Nowe, A.; Lenaerts, T.; Steenhaut, K. (Ed.). *Proceedings of the Machine Learning Conference of Belgium and the Netherlands, BeNeLearn '04*. Brussels: Brussels, 2004. p. 138–145.
- PEREIRA, S. do L.; BARROS, L. de; COZMAN, F. Strong probabilistic planning. In: GELBUKH, A.; MORALES, E. (Ed.). *MICAI 2008: Advances in Artificial Intelligence*. [S.l.]: Springer Berlin / Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5317). p. 636–652.
- PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1994. ISBN 0471619779.
- QUINLAN, J. R. Induction of decision trees. *Machine Learning*, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1, p. 81–106, 1986. ISSN 0885-6125.
- QUINLAN, J. R. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 1-55860-238-0.
- RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A modern approach*. [S.l.]: Prentice Hall, 2003.
- SHERSTOV, A. A.; STONE, P. Improving action selection in MDP's via knowledge transfer. In: *Proceedings of the Twentieth National Conference on Artificial Intelligence*. [S.l.: s.n.], 2005.
- SILVA, V. F. d.; COSTA, A. H. R. Compulsory Flow Q-Learning: an RL algorithm for robot navigation based on partial-policy and macro-states. *Journal of the Brazilian Computer Society*, scielo, v. 15, p. 65 – 75, 09 2009. ISSN 0104-6500.
- SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.
- TAYLOR, M. E.; STONE, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, v. 10, n. 1, p. 1633–1685, 2009.

UTHER, W. T. B.; VELOSO, M. M. Ttree: Tree-based state generalization with temporally abstract actions. In: *In Proceedings of SARA-2002*. [S.l.: s.n.], 2002. p. 260–290.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. 2. ed. [S.l.]: Morgan Kaufmann, 2005. Paperback. (Morgan Kaufmann series in data management systems). ISBN 0120884070.

Apêndice A – Abstração de política do ambiente da Figura 3.1

A.1 Estados da Figura 3.1

Os estados para o problema da Figura 3.1 estão listados na Tabela A.1.

A.2 Ações abstratas do problema

As ações abstratas utilizadas estão indicadas na Tabela A.2.

A.3 Operadores de Refinamento

Os operadores de refinamento estão indicados na Tabela A.3.

A.4 Indução da árvore

Número mínimo de exemplos para a divisão de um nó $\mathbf{nm} = 2$.

Nó raiz da árvore = $\text{in}(\mathbf{A})$.

Aplicando o algoritmo TILDE (Algoritmo 3), resulta na política abstrata mostrada na Figura 7.1.

Tabela A.1: Estados da Figura 3.1.

Estados
$\text{in}(11) \wedge \text{isCorridor}(11) \wedge \text{isCenter}(11) \wedge$ $\text{conn}(11,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(12) \wedge \text{isRoom}(12) \wedge \text{isCenter}(12) \wedge$ $\text{conn}(12,14) \wedge \text{isRoom}(14) \wedge \text{isNearDoor}(14)$
$\text{in}(13) \wedge \text{isRoom}(13) \wedge \text{isCenter}(13) \wedge$ $\text{conn}(13,16) \wedge \text{isRoom}(16) \wedge \text{isNearDoor}(16)$
$\text{in}(14) \wedge \text{isRoom}(14) \wedge \text{isNearDoor}(14) \wedge \text{conn}(14,12) \wedge \text{isRoom}(12)$ $\wedge \text{isCenter}(12) \wedge \text{conn}(14,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15) \wedge \text{conn}(15,14) \wedge$ $\text{isRoom}(14) \wedge \text{isNearDoor}(14) \wedge \text{conn}(15,16) \wedge \text{isRoom}(16) \wedge$ $\text{isNearDoor}(16) \wedge \text{conn}(15,11) \wedge \text{isCorridor}(11) \wedge \text{isCenter}(11) \wedge$ $\text{conn}(15,18) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18)$
$\text{in}(16) \wedge \text{isRoom}(16) \wedge \text{isNearDoor}(16) \wedge \text{conn}(16,13) \wedge \text{isRoom}(13)$ $\wedge \text{isCenter}(13) \wedge \text{conn}(16,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(17) \wedge \text{isRoom}(17) \wedge \text{isNearDoor}(17) \wedge \text{conn}(17,19) \wedge \text{isRoom}(19)$ $\wedge \text{isCenter}(19) \wedge \text{conn}(17,18) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18)$
$\text{in}(18) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18) \wedge \text{conn}(18,17) \wedge$ $\text{isRoom}(17) \wedge \text{isNearDoor}(17) \wedge \text{conn}(18,110) \wedge \text{isCorridor}(110) \wedge$ $\text{isCenter}(110) \wedge \text{conn}(18,15) \wedge \text{isCorridor}(15) \wedge \text{isNearDoor}(15)$
$\text{in}(19) \wedge \text{isRoom}(19) \wedge \text{isCenter}(19) \wedge \text{conn}(19,17) \wedge \text{isRoom}(17)$ $\wedge \text{isNearDoor}(17) \wedge \text{conn}(19,112) \wedge \text{isRoom}(112) \wedge \text{isNearDoor}(112)$
$\text{in}(110) \wedge \text{isCorridor}(110) \wedge \text{isCorridor}(18) \wedge \text{isNearDoor}(18) \wedge$ $\text{isCenter}(110) \wedge \text{conn}(110,18) \wedge \text{conn}(110,115) \wedge \text{isCorridor}(115) \wedge$ $\text{isNearDoor}(115) \wedge \text{conn}(110,111) \wedge \text{isCorridor}(111) \wedge \text{isCenter}(111)$
$\text{in}(111) \wedge \text{isCorridor}(111) \wedge \text{isCenter}(111) \wedge$ $\text{conn}(111,110) \wedge \text{isCorridor}(110) \wedge \text{isCenter}(110)$
$\text{in}(112) \wedge \text{isRoom}(112) \wedge \text{isNearDoor}(112) \wedge \text{conn}(112,19) \wedge \text{isRoom}(19)$ $\wedge \text{isCenter}(19) \wedge \text{conn}(112,113) \wedge \text{isRoom}(113) \wedge \text{isNearDoor}(113)$
$\text{in}(113) \wedge \text{isRoom}(113) \wedge \text{isNearDoor}(113) \wedge \text{conn}(113,117) \wedge \text{isRoom}(117)$ $\wedge \text{isCenter}(117) \wedge \text{conn}(113,112) \wedge \text{isRoom}(112) \wedge \text{isNearDoor}(112)$
$\text{in}(114) \wedge \text{isRoom}(114) \wedge \text{isNearDoor}(114) \wedge \text{conn}(114,117) \wedge \text{isRoom}(117)$ $\wedge \text{isCenter}(117) \wedge \text{conn}(114,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115)$
$\text{in}(115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115) \wedge \text{conn}(115,114) \wedge$ $\text{isRoom}(114) \wedge \text{isNearDoor}(114) \wedge \text{conn}(115,116) \wedge \text{isRoom}(116) \wedge$ $\text{isNearDoor}(116) \wedge \text{conn}(15,119) \wedge \text{isCorridor}(119) \wedge$ $\text{isCenter}(119) \wedge \text{conn}(15,110) \wedge \text{isCorridor}(110) \wedge \text{isCenter}(110)$
$\text{in}(116) \wedge \text{isRoom}(116) \wedge \text{isNearDoor}(116) \wedge \text{conn}(116,118) \wedge \text{isRoom}(118)$ $\wedge \text{isCenter}(118) \wedge \text{conn}(116,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115)$
$\text{in}(117) \wedge \text{isRoom}(117) \wedge \text{isCenter}(117) \wedge \text{conn}(117,114) \wedge \text{isRoom}(114)$ $\wedge \text{isNearDoor}(114) \wedge \text{conn}(117,113) \wedge \text{isRoom}(113) \wedge \text{isNearDoor}(113)$
$\text{in}(118) \wedge \text{isRoom}(118) \wedge \text{isCenter}(118) \wedge$ $\text{conn}(118,116) \wedge \text{isRoom}(116) \wedge \text{isNearDoor}(116)$
$\text{in}(119) \wedge \text{isCorridor}(119) \wedge \text{isCenter}(119) \wedge$ $\text{conn}(119,115) \wedge \text{isCorridor}(115) \wedge \text{isNearDoor}(115)$

Tabela A.2: Ações Abstratas definidas para o problema da Figura 3.1.

Ações Abstratas
goto_RC_RD(A,B)
goto_RC_RD(A,C)
goto_RD_RC(A,B)
goto_RD_RC(A,C)
goto_RD_CD(A,B)
goto_RD_CD(A,C)
goto_RD_RD(A,B)
goto_RD_RD(A,C)
goto_CD_RD(A,B)
goto_CD_RD(A,C)
goto_CD_RD(A,D)
goto_CD_RD(A,E)
goto_CD_CC(A,B)
goto_CD_CC(A,C)
goto_CD_CC(A,D)
goto_CD_CC(A,E)
goto_CC_CD(A,B)
goto_CC_CD(A,C)
goto_CC_CD(A,D)
goto_CD_CD(A,B)
goto_CD_CD(A,C)
goto_CD_CD(A,D)
goto_CD_CD(A,E)
goto_CC_CC(A,B)
goto_CC_CC(A,C)
goto_CC_CC(A,D)

Tabela A.3: Operadores de refinamento utilizados na construção das árvores dos experimentos.

Definição dos Operadores de Refinamento	
Predicados e modos dos termos	Quantidade de vezes a utilizar
isRoom(+X) $\wedge$ isCenter(+X)	10
isCorridor(+X) $\wedge$ isCenter(+X)	10
isRoom(+X) $\wedge$ isNearDoor(+X)	10
isCorridor(+X) $\wedge$ isNearDoor(+X)	10
conn(+X,+Y) $\wedge$ isRoom(+Y) $\wedge$ isCenter(+Y)	1
conn(+X,+Y) $\wedge$ isCorridor(+Y) $\wedge$ isCenter(+Y)	1
conn(+X,+Y) $\wedge$ isRoom(+Y) $\wedge$ isNearDoor(+Y)	1
conn(+X,+Y) $\wedge$ isCorridor(+Y) $\wedge$ isNearDoor(+Y)	1

Apêndice B – Parâmetros utilizados nos experimentos

B.1 Mapas

Probabilidade de sucesso na transição entre localidades $p = 0.9$.

B.2 Algoritmos de aprendizado

B.2.1 Q-Learning

Tabela B.1: Parâmetros utilizados nos experimentos para o aprendizado no nível concreto.

Parâmetro	Valor
γ	0.9
a	0.3
ϵ_1	0.5
ϵ_2	0.3
ϵ_a	0.7

B.2.2 Q-Learning para o nível abstrato

Tabela B.2: Parâmetros utilizados nos experimentos para o aprendizado no nível abstrato.

Parâmetro	Valor
γ	0.9
a	0.1