Soluções eficientes para processos de decisão markovianos baseadas em alcançabilidade e bissimulações estocásticas

Felipe Martins dos Santos

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Programa: Ciência da Computação
Orientadora: Prof[a]. Dr[a]. Leliane Nunes de Barros

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES.

São Paulo, 07 de Fevereiro de 2014

Soluções eficientes para
processos de decisão markovianos
baseadas em alcançabilidade e
bissimulações estocásticas

Esta dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa realizada por Felipe Martins dos Santos em 09/12/2013. A versão original da dissertação encontra-se disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profa. Dra. Leliane Nunes de Barros (Orientadora) - IME-USP
- Prof. Dr. Felipe Werndl Trevisan - IME-USP
- Prof. Dr. Fábio Gagliardi Cozman - EP-USP

Agradecimentos

Em primeiro lugar, agradeço a Deus por ter me dado inspiração para começar o mestrado, persistência e amigos para superar os momentos mais difíceis e forças para terminá-lo. Agradeço também aos meus pais, João e Dilacy, que sempre apoiaram meus estudos e fizeram com que minha única preocupação fosse estudar para garantir um bom futuro. De forma especial, agradeço também a Regina, minha namorada, que sempre conheceu em detalhes cada um dos desafios que tive durante a pesquisa do mestrado e sempre me apoiou nesses momentos.

Deixo meu obrigado para alguns familiares como minhas primas Deisy e Flaviana, que sempre estiveram por perto dizendo palavras motivadoras. Agradeço também aos familiares e amigos que nem sempre puderam estar por perto, mas sei que sempre torceram por mim.

Das pessoas que conheci no IME, não posso deixar de agradecer à professora Leliane, que me aceitou como orientando, foi paciente nos momentos mais difíceis e me mostrou o que é ser um bom pesquisador. Além dela, também devo agradecimentos à professora Karina, que me auxiliou em determinados períodos da pesquisa; e ao Mijail, que me ajudou em momentos cruciais com implementações de algumas funcionalidades do projeto.

Outros amigos que tornaram essa caminhada menos árdua foram: Daniel Baptista, Daniel Delgado, Denis, Fabio, Ignasi, Luis, Ricardo Augusto, Ricardo Herrmann, Viviane e outros. Também devo agradecimentos aos meus professores do Mackenzie, especialmente aos professores Luis Tadeu Raunheitte, Ana Cláudia Rossi e Roberto Araújo, que me deram uma boa base em Ciência da Computação e me auxiliaram para que eu pudesse ingressar no programa de mestrado do IME. A todos, muito obrigado!

Resumo

Planejamento em inteligência artificial é a tarefa de determinar ações que satisfaçam um dado objetivo. Nos problemas de planejamento sob incerteza, as ações podem ter efeitos probabilísticos. Esses problemas são modelados como Processos de Decisão Markovianos (*Markov Decision Processes - MDPs*), modelos que permitem o cálculo de soluções ótimas considerando o valor esperado de cada ação em cada estado. Contudo, resolver problemas grandes de planejamento probabilístico, i.e., com um grande número de estados e ações, é um enorme desafio. MDPs grandes podem ser reduzidos através da computação de bissimulações estocásticas, i.e., relações de equivalência sobre o conjunto de estados do MDP original. A partir das bissimulações estocásticas, que podem ser exatas ou aproximadas, é possível obter um modelo abstrato reduzido que pode ser mais fácil de resolver do que o MDP original. No entanto, para problemas de alguns domínios, a computação da bissimulação estocástica sobre todo o espaço de estados é inviável.

Os algoritmos propostos neste trabalho estendem os algoritmos usados para a computação de bissimulações estocásticas para MDPs de forma que elas sejam computadas sobre o conjunto de estados alcançáveis a partir de um dado estado inicial, que pode ser muito menor do que o conjunto de estados completo. Os resultados experimentais mostram que é possível resolver problemas grandes de planejamento probabilístico com desempenho superior às técnicas conhecidas de bissimulação estocástica.

Palavras-chave: Planejamento Probabilístico, Processo de Decisão Markoviano, Bissimulação Estocástica, Análise de Alcançabilidade

Abstract

Planning in artificial intelligence is the task of finding actions to reach a given goal. In planning under uncertainty, the actions can have probabilistic effects. This problems are modeled using Markov Decision Processes (MDPs), models that enable the computation of optimal solutions considering the expected value of each action when applied in each state. However, to solve big probabilistic planning problems, i.e., those with a large number of states and actions, is still a challenge. Large MDPs can be reduced by computing stochastic bisimulations, i.e., equivalence relations over the original MDP states. From the stochastic bisimulations, that can be exact or approximated, it is possible to get an abstract reduced model that can be easier to solve than the original MDP. But, for some problems, the stochastic bisimulation computation over the whole state space is unfeasible.

The algorithms proposed in this work extend the algorithms that are used to compute stochastic bisimulations for MDPs in a way that they can be computed over the reachable set of states with a given initial state, which can be much smaller than the complete set of states. The empirical results show that it is possible to solve large probabilistic planning problems with better performance than the known techniques of stochastic bisimulation.

Keywords: Probabilistic Planning, Markov Decision Processes, Stochastic Bisimulation, Reachability Analysis

Sumário

Lista de Abreviaturas

ADD	Algebraic Decision Diagram.
APRICODD	Approximate Policy Construction using Decision Diagrams.
BDD	Binary Decision Diagram.
BMDP	Bounded-parameter Markov Decision Process.
CSI	Context-Specific Independence.
CPT	Conditional Probability Table.
DBN	Dynamic Bayesian Network.
DNF	Disjunctive Normal Form.
ϵMRFS	ϵ-Model Reduction with Factored Splits.
IPPC	International Probabilistic Planning Competition.
IVI	Interval Value Iteration.
LRTDP	Labeled Real-Time Dynamic Programming.
MDP	Markov Decision Process.
MDPIP	Markov Decision Process with Imprecise Probabilities.
MDPST	Markov Decision Process with Set-Valued Transition.
MMFS	Model Minimization with Factored Splits.
MRFS	Model Reduction with Factored Splits.
PI	Policy Iteration.
RDDL	Relational Dynamic Influence Diagram Language.
Reach-ϵMRFS	Reachability-based ϵ Model Reduction with Factored Splits.
ReachMMFS	Reachability-based Model Minimization with Factored Splits.
ReachMRFS	Reachability-based Model Reduction with Factored Splits.
RTDP	Real-Time Dynamic Programming.
SPUDD	Stochastic Planning using Decision Diagrams.
sRTDP	Symbolic Real-Time Dynamic Programming.
SSP	Stochastic Shortest Path.
SSPLIT	Structure-based Split.
VI	Value Iteration.

Lista de Símbolos

$\vec{x}$	Vetor de atribuições para variáveis de estado.
A	Conjunto finito de ações.
B	Um bloco de uma partição.
G	Conjunto de estados meta.
M	Tupla que define um MDP.
$M_{\updownarrow}$	Tupla que define um BMDP.
P	Função que define probabilidades de transição.
$P_{\updownarrow}$	Probabilidade de transição dada por intervalo.
Q	Valor esperado de um estado ao aplicar uma ação.
R	Função que define recompensas para cada par de estado e ação.
$R_{\updownarrow}$	Função recompensa dada por intervalo.
S	Conjunto finito de estados.
V	Função Valor.
V^*	Função Valor Ótima.
V^t	Função Valor no estágio t.
V^π	Valor da política π.
$V_{\updownarrow}$	Função Valor Intervalar.
V^*_{opt}	Função Valor Ótima Otimista.
V^*_{pes}	Função Valor Ótima Pessimista.
X	Conjunto de variáveis de estado.
X_e	Conjunto de variáveis de estado essenciais.
$\mathcal{E}$	Erro máximo permitido.
$\mathcal{P}$	Uma partição de um conjunto.
$\mathcal{P}_{B_i}$	Uma partição de um bloco i.
$\mathcal{P}^a_f$	Partição baseada em uma função f e ação a.
$\mathcal{P}_{DD}$	Partição representada por ADD.
$\mathcal{P}^{S\|s_0}_{DD}$	Partição por alcançabilidade representada por ADD.
$\mathcal{G}$	Todos os nós de um ADD/BDD.
$\mathcal{V}$	Nós internos de um ADD/BDD.
$\mathcal{T}$	Nós terminais de um ADD/BDD.
ϵ	Constante usada para encontrar bissimulações estocásticas aproximadas.
γ	Fator de desconto.
π	Política.
π^*	Política Ótima.
π_{pes}	Política Ótima Pessimista.
π_{opt}	Política Ótima Otimista.
v_i	Rótulo de um bloco B_i em uma partição rotulada.

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Planejamento em inteligência artificial é a tarefa de determinar ações que satisfaçam um dado objetivo. A área de planejamento surgiu de investigações realizadas sobre os seguintes temas: busca em espaço de estados, prova de teoremas, teoria de controle e robótica. Desde então, planejamento é um dos temas centrais em inteligência artificial (Russel e Norvig, 2003).

Problemas de planejamento com características mais próximas de domínios reais devem lidar com a incerteza no efeito das ações. Em planejamento, os diferentes tipos de efeitos de ações definem o que é chamado de dinâmica de ações, que pode ser: determinística, não-determinística ou probabilística. Enquanto os modelos determinísticos não modelam incerteza no efeito das ações, os modelos não-determinísticos e probabilísticos modelam diferentes formas de incerteza para o efeito das ações. Formalmente, nos modelos determinísticos, a função de transição leva o sistema de um estado $s \in S$ para um estado $s' \in S$ ao executar uma ação $a \in A$, em que S é o conjunto de estados e A é o conjunto de ações. Nos modelos não-determinísticos, existem transições em que o agente pode alcançar um subconjunto $B \subseteq S$, sendo $|B| \geq 1$, e desta forma, o estado seguinte não é totalmente previsível e também não existe uma frequência associada a cada estado $s' \in B$. Nos modelos probabilísticos, também existem transições em que $|B| \geq 1$, porém essas transições levam o sistema de um estado s para um estado $s' \in B$ com uma probabilidade p, que permite expressar a frequência de cada estado $s' \in B$ ocorrer como resultado de uma ação $a \in A$.

Para lidar com ações que possuem efeitos probabilísticos em problemas de planejamento, adotou-se os Processos de Decisão Markovianos (*Markov Decision Process - MDPs*) (Puterman, 1994), por serem capazes de lidar com ações que possuem efeitos incertos e também por serem conhecidos algoritmos para calcular soluções da área da Pesquisa Operacional (Boutilier *et al.*, 1999).

> **MDP - Modelo Conceitual:** *Um MDP modela um sistema dinâmico em que um agente interage com um ambiente. O agente observa o estado atual, escolhe uma ação (com efeitos probabilísticos) e recebe uma recompensa pela ação escolhida no estado atual. Após executar essa ação, o sistema fornece um novo estado. O objetivo do agente é maximizar a recompensa acumulada ao longo de uma sequência de ações escolhidas.*

A solução de um MDP é uma *política*, i.e., uma função $\pi : S \mapsto A$ que associa uma ação a cada estado visitado no processo. Chamamos de *política ótima* aquela que garante uma recompensa esperada maior que qualquer outra política.

Os MDPs podem ser representados de forma enumerativa ou fatorada. Na representação enumerativa, cada estado é visto como uma caixa preta com um identificador único e as funções de transição probabilística e recompensa são definidas em termos desses identificadores de estados. Na representação fatorada, os estados são dados por atribuições booleanas sobre um conjunto X de variáveis de estado. Com isso, as funções de transição probabilística e recompensa são definidas em termos de variáveis de estado.

Um dos maiores desafios na área de planejamento probabilístico é resolver MDPs grandes, i.e., $|S|$ e $|A|$ são grandes. Essa dificuldade surge porque o número de estados em um MDP cresce

exponencialmente com o número de variáveis de estado. Isso pode tornar a computação da política ótima inviável, o que é conhecido como *maldição da dimensionalidade* (Bellman, 1957). Além do número de variáveis de estado, a dificuldade dos problemas pode aumentar se o domínio é *denso*, i.e., se o número de estados alcançáveis é maior do que ou igual à metade de S. Se o domínio não é denso, é chamado de *esparso*. Muitos algoritmos têm sido propostos para evitar a enumeração completa dos estados e explorar as características de domínios esparsos, por exemplo, explorando modelos fatorados (Hoey *et al.*, 1999) e usando informação do estado inicial, como acontece no algoritmo de programação dinâmica em tempo real (*Real-Time Dynamic Programming - RTDP*) (Barto *et al.*, 1993), que calcula uma *política ótima parcial*, que é uma política ótima definida sobre um subconjunto de S que contém apenas os estados alcançáveis dado um estado inicial s_0. Esse subconjunto é denotado por S_{alc}.

Uma outra abordagem para resolver MDPs grandes é reduzir o tamanho do MDP original, i.e., $|S|$. Para encontrar um MDP reduzido é necessário encontrar uma partição $\mathcal{P}$ de S em que cada bloco da partição representa um subconjunto $B_i \subseteq S$ que agrupa estados (fatorados) equivalentes de acordo com as funções de recompensa e transição probabilística. Isso é feito usando algoritmos de redução (*MRFS - Model Reduction with Factored Splits*) e minimização de modelos baseados em representações fatoradas (*MMFS - Model Minimization with Factored Splits*) (Givan *et al.*, 2003). As partições obtidas por esses algoritmos representam uma relação de equivalência que é conhecida como ***bissimulação estocástica exata***. Depois disso, um MDP enumerativo reduzido é obtido e pode ser resolvido com qualquer algoritmo para encontrar uma política ótima, por exemplo, Iteração de Valor (*Value Iteration - VI*) (Bellman, 1957). Contudo, o tamanho do MDP reduzido obtido de uma bissimulação estocástica também pode ser muito grande. Com isso, o conceito de bissimulação estocástica foi estendido para permitir aproximações.

Em uma ***bissimulação estocástica aproximada***, estados são agrupados pelo algoritmo de ϵ-redução de modelos (ϵ*MRFS*) (Dean *et al.*, 1997; Givan *et al.*, 2000) mesmo se não são considerados equivalentes. Como resultado, o modelo reduzido é um MDP com intervalos (*Bounded-parameter Markov Decision Process - BMDP*) (Dean *et al.*, 1997; Givan *et al.*, 2000), i.e., um MDP no qual as funções de recompensa e transição probabilística são dadas de forma aproximada por intervalos. Para resolver BMDPs, a solução mais básica é uma variação do algoritmo VI, chamada Iteração de Valor Intervalar (*Interval Value Iteration - IVI*) (Dean *et al.*, 1997; Givan *et al.*, 2000). Uma abordagem mais eficiente para resolver BMDPs é o RTDP Robusto (Buffet e Aberdeen, 2005), que permite a computação de soluções robustas atualizando somente os estados alcançáveis a partir de um estado inicial. Em outras palavras, o RTDP Robusto calcula uma política parcial que inclui apenas os estados alcançáveis a partir de um estado inicial s_0.

Apesar das vantagens de se usar RTDP e RTDP Robusto para resolver o modelo reduzido, o processo para encontrar a bissimulação estocástica (exata e aproximada) é feito sobre o conjunto completo de estados do MDP original, o que torna a computação inviável em MDPs grandes.

Neste trabalho, os algoritmos MRFS, MMFS e ϵ-MRFS (Dean *et al.*, 1997; Givan *et al.*, 2000, 2003), que são usados para encontrar bissimulações estocásticas, foram estendidos para serem realizados em duas fases (Figura 1.1): (1) análise de alcançabilidade realizada antes de computar a bissimulação estocástica; e (2) computação da bissimulação estocástica (exata ou aproximada) sobre S_{alc}. A Tabela 1.1 apresenta as diferentes maneiras de computar bissimulações estocásticas sobre S ou S_{alc} por meio de diferentes tomadas de decisão com base na Figura 1.1. Os algoritmos deste trabalho baseados em alcançabilidade são: ReachMRFS (Reachability-based MRFS), ReachMMFS (Reachability-based MMFS) e Reach-ϵMRFS (Reachability-based ϵMRFS). Os algoritmos propostos devolvem um MDP (ou BMDP) enumerativo que tem apenas estados alcançáveis a partir de um estado inicial s_0 e esses estados são agrupados em blocos que passam a ser vistos como estados abstratos. Finalmente, se a bissimulação estocástica é exata, o MDP reduzido pode ser resolvido com qualquer algoritmo que encontra soluções para MDPs. Se a bissimulação estocástica é aproximada, o BMDP pode ser resolvido usando algoritmos que encontram soluções para BMDPs, sendo que esses algoritmos são extensões dos algoritmos para MDPs.

Os resultados empíricos mostram que ao computar bissimulações estocásticas sobre S_{alc}, é pos-

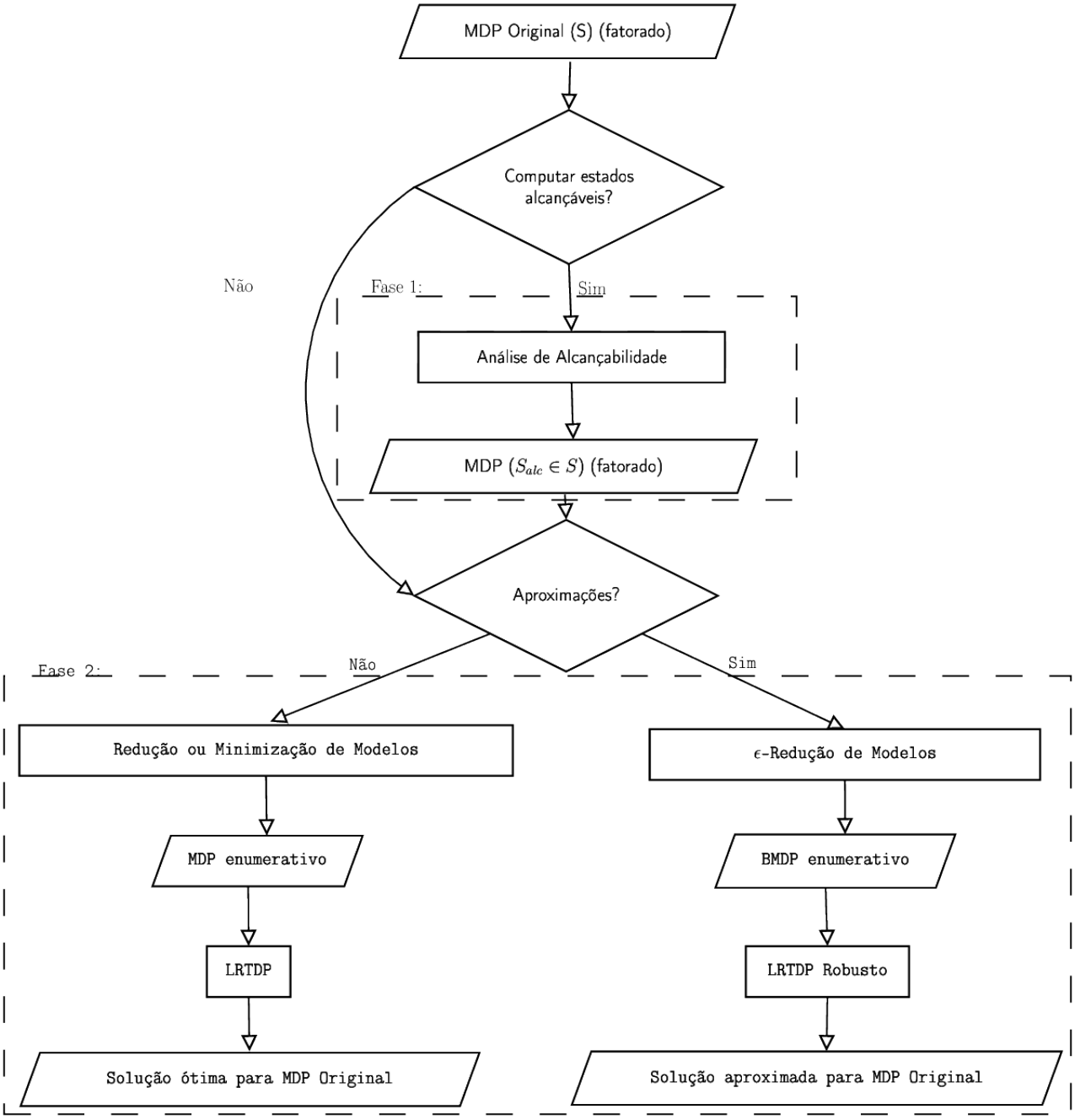


Figura 1.1: *Fluxograma que descreve como os problemas são resolvidos neste trabalho. Paralelogramos representam entrada/saída, retângulos correspondem aos algoritmos utilizados e losangos são as tomadas de decisão.*

sível reduzir e resolver problemas maiores do que ao calcular bissimulações estocásticas sobre S. Dentre os algoritmos propostos, ReachMRFS foi o que obteve melhor performance por realizar cálculos mais simples para encontrar bissimulações estocásticas exatas. Além das melhorias com o uso de alcançabilidade, foi demonstrado que o tempo gasto com a análise de alcançabilidade pode ser polinomial no número de estados. Para os problemas em que foi possível utilizar ReachMRFS foram encontradas 4 situações distintas:

1. problemas reduzidos apenas pela análise de alcançabilidade;
2. problemas reduzidos apenas pela bissimulação estocástica sobre S;
3. problemas reduzidos pela análise de alcançabilidade e bissimulação estocástica sobre S_{alc}; e

4. problemas que não puderam ser resolvidos.

De acordo com as situações observadas, a situação ideal para aplicar ReachMRFS se compararmos com MRFS é a situação 3. Apesar disso, a situação 1 também pode trazer muitas vantagens em MDPs esparsos.

Computar estados alcançáveis?	Aproximações?	Algoritmo
Não	Não	MRFS ou MMFS
Não	Sim	ϵ-MRFS
Sim	Não	ReachMRFS ou ReachMMFS
Sim	Sim	Reach-ϵMRFS

Tabela 1.1: *Algoritmos para computar bissimulações estocásticas obtidos através das diferentes decisões apresentadas na Figura 1.1.*

Condições	**Probabilidade de $X_iY'_j = 1$**
Se a célula X_iY_j está viva e tem entre 2 e 3 vizinhos vivos; ou célula X_iY_j não está viva e tem exatamente 3 vizinhos vivos; ou a ação setX_iY_j foi executada.	$1 - ruido(X_i, Y_j)$
Caso contrário.	$ruido(X_i, Y_j)$

Tabela 1.2: *Probabilidade de uma variável $X_iY'_j$ ser verdadeira em um estado seguinte nas instâncias do domínio Game of Life.*

estado	ação ($\pi(s)$)
$s_0 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_1 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$	set(X_2, Y_2)
$s_2 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_3 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$	set(X_2, Y_2)
$s_4 : \neg X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_5 : \neg X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$	set(X_2, Y_2)
$s_6 : \neg X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_7 : \neg X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$	noop
$s_8 : X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_9 : X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$	set(X_2, Y_2)
$s_{10} : X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_{11} : X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$	noop
$s_{12} : X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	set(X_2, Y_2)
$s_{13} : X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$	noop
$s_{14} : X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$	noop
$s_{15} : X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$	noop

Tabela 1.3: *Os 16 estados da instância de Game of Life, descrita por 4 variáveis de estado, e um mapeamento de política ótima para cada estado.*

1.1 Exemplo: Game of Life

Considere o domínio de planejamento probabilístico *Game of Life*, baseado no jogo de mesmo nome proposto por John H. Conway em 1970 (Gardner, 1970). Nesse domínio, é dada uma grade contendo $x \times y$ células biológicas. Cada célula é representada por uma variável de estado e pode estar viva (1) ou morta (0) de acordo com a vida existente nas células vizinhas e com o ruído da própria célula, i.e., uma medida de probabilidade que interfere na forma como essa célula vive ou morre. Em planejamento, diferentemente do que ocorre no *Game of Life* original, o agente pode interferir a qualquer momento dando vida a células específicas com o intuito de maximizar o número de células vivas na grade ao longo das gerações, i.e., diferentes instantes de tempo. Uma ação nesse domínio permite tornar a célula da posição X_iY_j viva ($set(X_i, Y_j)$), o que implica em $x \times y$ ações (uma para cada célula). Existe ainda a ação *noop* que não modifica o valor de nenhuma célula.

Considere uma instância de *Game of Life* com $x = y = 2$, sendo as variáveis de estado que representam as células: X_1Y_1, X_1Y_2, X_2Y_1 e X_2Y_2. Para essa instância, a probabilidade de uma célula biológica X_iY_j estar viva na geração seguinte depende de diferentes configurações da grade conforme é apresentado na Tabela 1.2. A Tabela 1.3 mostra os 16 estados e a política ótima em cada um deles.

Para essa instância de *Game of Life*, ao se obter uma bissimulação estocástica com a técnica de redução de modelos, o número de estados no MDP reduzido é apenas 5 conforme é ilustrado na Tabela 1.4. Além disso, ainda é possível perceber que a política ótima se mantém a mesma para cada estado que foi agrupado.

estado	estado abstrato	ação ($\pi(B_i)$)
$s_0 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	B_0	set(X_2, Y_2)
$s_1 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$ $s_2 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$ $s_4 : \neg X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$ $s_8 : X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	B_1	set(X_2, Y_2)
$s_3 : \neg X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$ $s_5 : \neg X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$ $s_6 : \neg X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$ $s_9 : X_1Y_1 \wedge \neg X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$ $s_{10} : X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$ $s_{12} : X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge \neg X_2Y_2$	B_2	set(X_2, Y_2)
$s_7 : \neg X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$ $s_{11} : X_1Y_1 \wedge \neg X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$ $s_{13} : X_1Y_1 \wedge X_1Y_2 \wedge \neg X_2Y_1 \wedge X_2Y_2$ $s_{14} : X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge \neg X_2Y_2$	B_3	noop
$s_{15} : X_1Y_1 \wedge X_1Y_2 \wedge X_2Y_1 \wedge X_2Y_2$	B_4	noop

Tabela 1.4: *Bissimulação estocástica encontrada pelo algoritmo de redução de modelos para uma instância de Game of Life com 16 estados.*

1.2 Principais Contribuições

Os algoritmos de redução, minimização e ϵ-redução de modelos foram propostos com o intuito de resolver MDPs grandes. Apesar disso, nos trabalhos anteriores relacionados à bissimulação estocástica, os maiores problemas envolviam 20 variáveis (Kim e Dean, 2002), ou seja, no máximo 2^{20} estados considerando que as variáveis de estado são booleanas. Contudo, na forma como esses algoritmos foram propostos, resolver problemas maiores não era uma tarefa viável. Neste trabalho, foram propostas algumas melhorias que permitem que MDPs maiores sejam reduzidos e resolvidos. Dentre as melhorias estão:

1. Utilizar a informação do estado inicial com o objetivo de encontrar S_{alc}, e em seguida, computar uma bissimulação estocástica (exata ou aproximada) sobre S_{alc}. Essa técnica foi capaz de resolver problemas com um número de variáveis até 7 vezes maior que os problemas resolvidos com a técnica tradicional que computa bissimulações estocásticas.

2. Eliminação de partições repetidas durante a computação da bissimulação estocástica para os algoritmos MRFS e ReachMRFS (mais adiante é explicado porque apenas esses algoritmos permitem essa melhoria).

3. Comparação empírica entre o algoritmo proposto, ReachMRFS e o algoritmo MRFS. Os resultados mostram que RechMRFS é mais eficiente que MRFS em problemas esparsos.

4. Comparação empírica entre ReachMRFS, ReachMMFS e Reach-ϵMRFS nos domínios utilizados na última competição internacional de planejamento probabilístico (IPPC-2011). Os resultados mostram ReachMRFS como o melhor entre os três algoritmos para qualquer tipo de problema.

5. Comparação empírica entre o LRTDP para o MDP original e para o MDP reduzido. Os resultados empíricos mostram que resolver o MDP reduzido com o LRTDP é mais vantajoso em problemas densos.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma.

- O Capítulo 2 apresenta os MDPs enumerativos, fatorados e como resolver esses MDPs com diferentes algoritmos.
- O Capítulo 3 apresenta os BMDPs e como resolvê-los com variações dos algoritmos para MDPs enumerativos
- O Capítulo 4 apresenta conceitos da bissimulação estocástica exata e algoritmos para computação dessa relação de equivalência que permite visualizar subconjuntos de estados do MDP original como estados abstratos em um MDP enumerativo reduzido.
- O Capítulo 5 estende os conceitos apresentados no Capítulo 4 com a bissimulação estocástica aproximada, a partir da qual é possível obter BMDPs enumerativos reduzidos.
- O Capítulo 6 apresenta as melhorias propostas para os algoritmos de redução, minimização e ϵ-redução de modelos que são usados originalmente para encontrar bissimulações estocásticas.
- O Capítulo 7 apresenta os problemas usados para benchmark e resultados empíricos obtidos com os experimentos.
- O Capítulo 8 apresenta as conclusões e possíveis trabalhos futuros.

Capítulo 2

MDP - Processo de Decisão Markoviano

Um Processo de Decisão Markoviano (*Markov Decision Process - MDP*) (Puterman, 1994) modela um sistema dinâmico em que um agente interage com um ambiente. O agente observa o estado atual s e escolhe uma ação que possui efeitos probabilísticos e com a qual está associada uma função de recompensa. Ao executar essa ação, o agente alcança um novo estado e obtém uma recompensa pela ação ter sido executada no estado s. O objetivo do agente é maximizar a recompensa ao longo de uma sequência de ações escolhidas. Neste trabalho, os MDPs são usados com as seguintes suposições:

- conjunto de estados finito;
- ambientes completamente observáveis: o agente é capaz de observar o ambiente por completo, sem perdas de informações e sem ruídos em seus sensores;
- ações estocásticas: ao executar uma ação em um dado estado, um conjunto de estados pode ser alcançado de acordo com uma distribuição de probabilidades;
- tempo implícito e discreto: as ações não possuem duração de tempo e o tempo é considerado discreto, ou seja, cada estado visitado (ou escolha de ação, i.e., tomada de decisão) é associada a um instante de tempo;
- presença de um único agente;
- ambientes em que as mudanças ocorrem apenas com as ações do agente, i.e., não existem eventos que estão fora do controle do agente, também conhecidos como eventos exógenos; e
- planejamento *off-line*, i.e., a política ótima é calculada e depois executada no ambiente.

2.1 MDP - Definições

Definição 1. *Um MDP (Puterman, 1994) é uma tupla $M = (S, A, P, R, \gamma)$, em que:*

- *S é um conjunto finito de estados que podem ser observados no decorrer do tempo.*
- *A é um conjunto finito de ações que o agente pode executar. Além disso, $A(s)$ define as ações aplicáveis em s, i.e., é um subconjunto de A que especifica quais ações podem ser executadas pelo agente quando o sistema se encontra no estado $s \in S$.*
- *$P : S \times A \times S \mapsto [0, 1]$ é uma medida de probabilidade condicional para cada transição. Ou seja, $P(s'|s, a)$ é a probabilidade de mudar do estado $s \in S$ para o estado $s' \in S$ ao executar a ação $a \in A(s)$. A soma das probabilidades de cada transição possível a partir de um par (s, a) é dada por: $\sum_{s' \in S} P(s'|s, a) = 1$. Uma forma de visualizar cada $P(.|., a)$ é como uma matriz de transição probabilística em que cada linha representa um estado $s \in S$, cada coluna*

representa um estado $s' \in S$ e cada entrada (s, s') tem a probabilidade $P(s'|s, a)$. Desta forma, cada matriz tem $|S|^2$ entradas. Com isso, dizemos que uma matriz de transição é densa se pelo menos 50% das $|S|^2$ entradas têm probabilidades maiores do que 0 (Tabela 2.1); caso contrário, tem-se uma matriz de transição esparsa (Tabela 2.2).

- *$R : S \times A \mapsto \mathbb{R}$ é uma função que associa uma recompensa para cada par (s, a).*
- *$\gamma \in [0, 1]$ é o fator de desconto, que é utilizado para se encontrar as soluções em MDPs de horizonte infinito, ou seja, aqueles em que o agente deve tomar decisões ao longo do tempo sem um prazo finito para o término de suas tarefas. Neste trabalho são utilizados MDPs de horizonte infinito. Existem ainda MDPs de horizonte finito e horizonte indefinido, chamados de SSP (Stochastic Shortest Path).*

$P(.\|., a_0)$	s'_0	s'_1	s'_2	s'_3
s_0	0.9025	0.0475	0.0475	0.0025
s_1	0.0475	0.9025	0.0025	0.0475
s_2	0.035	0.015	0.665	0.285
s_3	0.0025	0.0475	0.0475	0.9025

Tabela 2.1: *Exemplo de matriz de transição densa para uma ação a_0 em um MDP que tem $|S| = 4$.*

$P(.\|., a_1)$	s'_0	s'_1	s'_2	s'_3
s_0	0.95	0	0.05	0
s_1	0	1	0	0
s_2	0.05	0	0.95	0
s_3	0	0	0	1

Tabela 2.2: *Exemplo de matriz de transição esparsa para uma ação a_1 em um MDP que tem $|S| = 4$.*

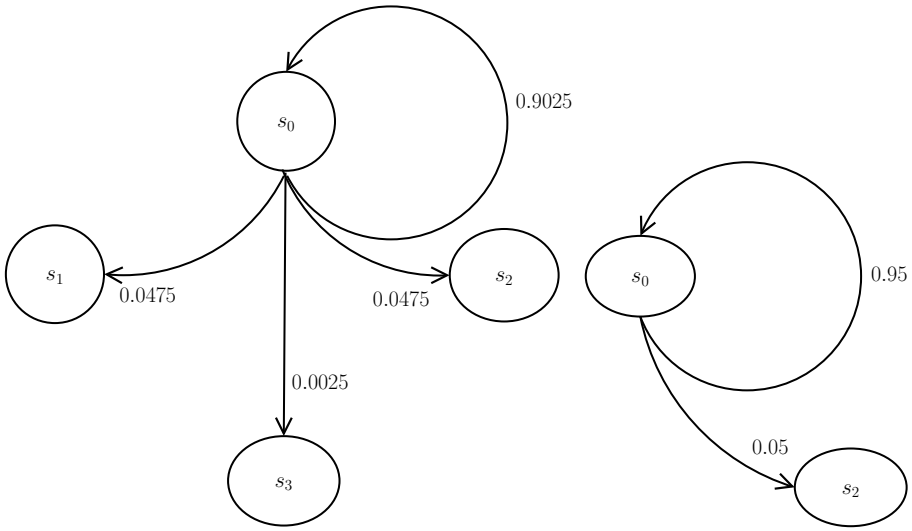
O MDP apresentado nessa seção (Definição 1) é conhecido também como *MDP enumerativo*, por enumerar todos os estados em S e tratá-los como caixas pretas, i.e., não há como aproveitar a representação interna de cada estado através de variáveis de estado. Além disso, a palavra 'Markoviano' é utilizada no nome do modelo porque a probabilidade do sistema ir para o estado $s' \in S$ depende apenas do estado atual $s \in S$ e não de todos os estados que foram visitados anteriormente (Russel e Norvig, 2003).

A Figura 2.1 apresenta dois exemplos diferentes de funções de transição quando um estado s_0 é a origem e diferentes ações são aplicadas. O MDP do exemplo possui $S = \{s_0, s_1, s_2, s_3\}$ e $A = \{a_0, a_1\}$.

Devido à incerteza nas transições de estado, a solução de um MDP não pode ser uma sequência de ações como é feito em soluções de problemas de planejamento com modelos determinísticos. Ao invés disso, a solução para MDPs é um mapeamento $\pi : S \mapsto A$ chamado de política, que informa qual ação $a \in A$ o agente deve executar em um determinado estado $s \in S$. Note que π é estacionária, i.e., as ações especificadas pela política para cada estado são sempre as mesmas. Em problemas de horizonte infinito, as políticas devem ser estacionárias pois a ação a ser executada independe do momento em que é executada (Puterman, 1994). Além disso, pode-se ter uma política definida apenas para um subconjunto de S, chamada de *política parcial* (Barto *et al.*, 1993).

Seja t a t-ésima decisão tomada pelo agente, também chamada de *estágio* de um MDP. Além disso, considere que s_t é o estado visitado no estágio t. O objetivo do agente em MDPs de horizonte infinito é maximizar a recompensa acumulada esperada descontada ao longo de uma sequência de estados visitados com uma política π, com $t \to \infty$. Ou seja, maximizar a seguinte expressão:

$$E_\pi[\gamma^t R(s_t, \pi(s_t))]. \tag{2.1}$$



(a) $P(.|s_0, a_0)$ (b) $P(.|s_0, a_1)$

Figura 2.1: *Um exemplo de função de transição num MDP genérico. Aplicando-se a ação a_0 no estado s_0, a função $P(.|s_0, a_0)$ leva aos estados s_1, s_2 e s_3, com probabilidades 0.0475, 0.0475 e 0.0025; e com probabilidade 0.9025, o agente permanece no estado s_0.*

Para avaliar uma política π, é necessário definir a função valor para políticas, que é uma função $V^\pi : S \mapsto \mathbb{R}$ que para cada $s \in S$, estima o ganho esperado da ação $\pi(s)$ que foi selecionada. A função valor para políticas é dada por (Puterman, 1994):

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s'). \tag{2.2}$$

Uma política π^* é ótima se para todo $s \in S$ e para toda política π, $V^{\pi^*}(s) \geq V^\pi(s)$. Ou seja, uma política ótima maximiza a função valor em S e é denotada para cada $s \in S$ por $V^*(s)$ (Puterman, 1994).

Uma política gulosa π^* que é obtida através de $V^*(s)$ é uma política ótima. Seja $Q(s, a)$ a função que representa o valor esperado de um estado $s \in S$ ao executar a ação $a \in A$ e seguir a política gulosa. O valor $V^*(s)$ é obtido através da Equação 2.4 de forma que ao expandir Q, obtém-se a Equação 2.5, conhecida como o *Princípio da Otimalidade de Bellman* ou *Equação de Bellman* (Bellman, 1957):

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \tag{2.3}$$

$$V^*(s) = \max_{a \in A(s)} \{Q(s, a)\} \tag{2.4}$$

$$V^*(s) = \max_{a \in A(s)} \{R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s')\}. \tag{2.5}$$

A política ótima para cada estado $s \in S$ é obtida da seguinte maneira:

$$\pi^*(s) = \arg \max_{a \in A(s)} \{R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s')\}. \tag{2.6}$$

Pela natureza recursiva da função valor, costuma-se utilizar programação dinâmica nas implementações dos algoritmos por ser uma técnica eficiente para resolver problemas com essas características. Os algoritmos de programação dinâmica buscam a cada iteração aperfeiçoar a função valor até que a mesma convirja para o valor ótimo $V^*(s)$ (Equação 2.7).

2.2 MDP - Soluções

Um algoritmo que resolve um MDP é um algoritmo de otimização que deve encontrar uma política que maximize a função valor. Nas seções seguintes são apresentados alguns algoritmos com essa finalidade.

2.2.1 Iteração de Política

O algoritmo de iteração de política (*Policy Iteration - PI*) (Howard, 1960) é um algoritmo que inicialmente define uma política π aleatória e a cada iteração tenta aperfeiçoá-la. Na fase de aperfeiçoamento da política, o algoritmo resolve um sistema de equações lineares a cada iteração, o que torna as iterações do algoritmo computacionalmente custosas (Ghallab *et al.*, 2004). A resposta fornecida pelo PI (Algoritmo 2.2.1) é uma política ótima π^*.

Algoritmo 2.2.1: IteraçãoDePolítica (M) (Howard, 1960)

Entrada: M: um MDP
Saída: π^*: uma política ótima

1 *Selecione qualquer* $\pi' \neq \emptyset$;
2 **enquanto** $\pi' \neq \pi$ **faça**
3 $\quad \pi \leftarrow \pi'$;
4 $\quad$ **para cada** $s \in S$ **faça**
5 $\quad\quad$ *Resolva o sistema de equações lineares para determinar* $V^\pi(s)$;
6 $\quad\quad V^\pi(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s')$;
7 $\quad$ **fim**
8 $\quad$ **para cada** $s \in S$ **faça**
9 $\quad\quad$ **se** $\exists a \in A(s)$ tal que $V^\pi(s) < R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s')$ **então**
10 $\quad\quad\quad \pi'(s) \leftarrow a$;
11 $\quad\quad$ **fim**
12 $\quad\quad$ **senão**
13 $\quad\quad\quad \pi'(s) \leftarrow \pi(s)$;
14 $\quad\quad$ **fim**
15 $\quad$ **fim**
16 **fim**
17 **retorna** π.

O PI se torna uma solução inviável em problemas grandes por ter que atualizar $V^\pi(s)$ para todo $s \in S$ em cada iteração (Linhas 5-16) (Ghallab *et al.*, 2004).

2.2.2 Iteração de Valor

O algoritmo de iteração de valor (*Value Iteration - VI*) (Bellman, 1957) é um algoritmo que encontra V^* e se utiliza da técnica de programação dinâmica para realizar essa tarefa. Esse algoritmo tem garantias de encontrar uma política ótima; apesar disso, o algoritmo demora para convergir por ter que atualizar os valores de todo conjunto de estados em cada iteração. A forma como a atualização do conjunto de estados é realizada pelo VI é conhecida como *programação dinâmica síncrona*.

O VI realiza uma inicialização qualquer para $V^0(s)$, e em seguida, busca melhores valores consultando os ganhos esperados para estados seguintes de acordo com a equação a seguir, que é conhecida como *atualização de Bellman* (Bellman, 1957):

$$V^{t+1}(s) = \max_{a \in A(s)} \{R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^t(s')\}. \tag{2.7}$$

O critério de parada utilizado pelo VI é a inequação a seguir, conhecida como *Erro de Bellman* (Bellman, 1957), em que V^t representa a função valor para o estado s no estágio t e $\mathcal{E}$ é um valor constante que representa o erro máximo total. Desta forma, se a desigualdade for verdadeira, todos os estados já terão um valor com erro aceitável e portanto, terão convergido para um valor $\mathcal{E}$-ótimo.

$$\max_{s \in S} |V^t(s) - V^{t-1}(s)| < \mathcal{E} \tag{2.8}$$

A seguir, é apresentado o VI (Algoritmo 2.2.2).

Algoritmo 2.2.2: IteraçãoDeValor(M, $\mathcal{E}$) (Bellman, 1957)

Entrada: M: um MDP, $\mathcal{E}$: erro máximo permitido
Saída: π^*: uma política ótima

1 **para cada** $s \in S$ **faça**
2 | *Inicialize* $V^0(s)$ *com um valor qualquer*;
3 **fim**
4 $t \leftarrow 1$;
5 **repita**
6 | **para cada** $s \in S(s)$ **faça**
7 | | **para cada** $a \in A$ **faça**
8 | | | $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{t-1}(s')$;
9 | | **fim**
10 | | $V^t(s) \leftarrow \max_{a \in A} Q(s,a)$;
11 | | $\pi(s) \leftarrow \arg\max_{a \in A} Q(s,a)$;
12 | **fim**
13 | $t \leftarrow t + 1$;
14 **até** $\max_{s \in S} |V^t(s) - V^{t-1}(s)| < \mathcal{E}$;
15 **retorna** π.

O VI também pode ser utilizado eficientemente apenas em problemas considerados pequenos por ter de atualizar todos os estados em cada iteração (Linhas 6-12). Para resolver essa limitação foram criadas várias soluções alternativas (Ghallab *et al.*, 2004), algumas são apresentadas nas próximas seções.

2.2.3 RTDP - Programação Dinâmica em Tempo Real

Para resolver problemas de planejamento probabilístico mais eficientemente, é possível modelá-los como um problema de encontrar um caminho estocástico mínimo (*Stochastic Shortest Path - SSP*) (Bertsekas e Tsitsiklis, 1991). Os SSPs são MDPs em que é acrescentado um estado inicial s_0, um conjunto G não-vazio de estados meta que são absorventes, i.e., se s_G é um estado meta, para toda ação $a \in A$, temos que $P(s_G|s_G, a) = 1$ e $R(s_G, a) = 0$. Para todo estado s que não é meta e toda ação $a \in A$, $R(s, a) < 0$. Além disso, nos SSPs é usado $\gamma = 1$.

Uma forma de resolver SSPs é utilizando o algoritmo de programação dinâmica em tempo real (*Real-Time Dynamic Programming - RTDP*)(Barto *et al.*, 1993), que atualiza o conjunto de estados de maneira assíncrona, podendo atualizar os valores de alguns estados mais vezes do que de outros. Essa técnica é classificada como *programação dinâmica assíncrona* e permite encontrar soluções de maneira mais eficiente na prática, em que o conjunto de estados S é maior do que o número de estados alcançáveis a partir de s_0.

Considere o conjunto de *estados relevantes* de um MDP, i.e., o conjunto de estados alcançáveis a partir de uma política ótima e um estado inicial s_0. O algoritmo RTDP resolve SSPs devolvendo uma política ótima parcial π^* definida sobre estados relevantes, mas para isso precisa fazer duas suposições (Barto *et al.*, 1993):

- a primeira, é que $V^0(s)$ deve ser inicializada com uma heurística admissível para todo $s \in S$, i.e., deve ser inicializada com um valor otimista com relação à função $V^*(s)$ (ou seja, $\forall s \in S, V^0(s) \geq V^*(s)$); e
- a segunda, é que existe um estado $s' \in G$ que pode ser alcançado a partir de qualquer estado $s \in S$, o que é conhecida como *hipótese de alcançabilidade*.

Com essas duas suposições, o algoritmo RTDP não entra em loops e eventualmente alcança uma política ótima parcial (Barto *et al.*, 1993). Neste trabalho, a implementação utiliza MDPs com estados iniciais, mas com as seguintes características:

- $\gamma = 0.99$ ao invés de ter G na especificação de cada problema; e
- $V^0 = \frac{\max_{s \in S, a \in A} R(s,a)}{1-\gamma}$, que é a recompensa máxima considerando horizonte infinito e o valor do fator de desconto.

O RTDP inicia o planejamento a partir de um estado inicial s_0 e visita um conjunto de estados seguindo uma política gulosa (ação que maximiza a Equação 2.7) até alcançar uma profundidade máxima ou um limite de tempo. Depois disso, o RTDP reinicia o planejamento a partir de s_0 e repete o procedimento. Cada caminho a partir de s_0 até uma profundidade máxima ou limite de tempo é chamado de *histórico* (trial no algoritmo).

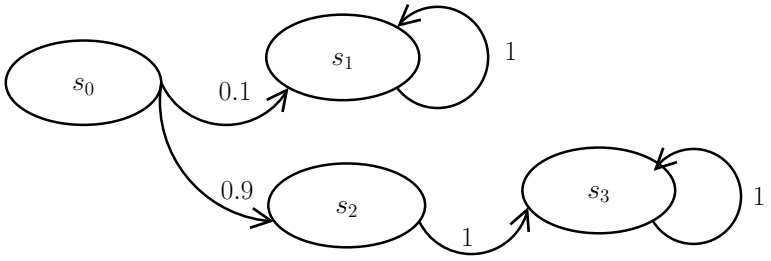


Figura 2.2: *Exemplo gráfico de um MDP que permite simulações de EscolhaProximoEstado (Algoritmo 2.2.3).*

Em cada estado s visitado no trial, o RTDP computa uma política gulosa em s; atualiza $V^t(s)$ e então sorteia um estado seguinte para o trial de acordo com a distribuição de probabilidades (Algoritmo 2.2.3). Um problema dessa abordagem é que estados com menor probabilidade são atualizados menos vezes e consequentemente, seus valores demoram mais para convergir. Apesar disso, os valores desses estados também são necessários para que o algoritmo possa garantir a convergência como um todo. Por exemplo, considere o exemplo da Figura 2.2. Se o agente estiver no estado s_0 e a ação mostrada for a política gulosa, o sorteio do RTDP (Algoritmo 2.2.3) atribui uma probabilidade de 90% de o estado seguinte ser s_2; e de 10% de o estado seguinte ser s_1.

As vantagens do RTDP estão em atualizar o conjunto de estados usando programação dinâmica assíncrona e em ter um comportamento conhecido como *anytime*, i.e., devolve boas políticas mesmo após poucos trials. A desvantagem do RTDP está em continuar atualizando os valores de estados que já convergiram. Para tratar esse problema, foram propostas algumas extensões para o RTDP. O RTDP (Algoritmo 2.2.4) é descrito a seguir.

Algoritmo 2.2.3: EscolhaProximoEstado (s, a) (Barto *et al.*, 1993)

Entrada: s: um estado em S, a: uma ação em $A(s)$.
Saída: s': um estado tal que $P(s'|s,a) > 0$.

1 *// s' é sorteado de acordo com a distribuição de probabilidades*;
2 $s' \sim P(.|s,a)$.;
3 **retorna** s'.

Algoritmo 2.2.4: RTDP (M, s_0, G) (Barto *et al.*, 1993)

Entrada: M: um MDP, s_0: um estado inicial, G: um conjunto de estados meta (G é opcional, depende se $\gamma \in]0,1[$ é ou não utilizado para computar a atualização de Bellman).

Saída: π: uma política ótima parcial.

1 $V^0 \leftarrow \frac{\max_{s \in S, a \in A} R(s,a)}{1-\gamma}$ *(uma heuristíca admissível)*;
2 **enquanto** convergência não detectada e dentro do tempo aceitável **faça**
3 $\quad profundidade \leftarrow 0$;
4 $\quad s \leftarrow s_0$;
5 $\quad$ *// Inicio do trial*;
6 $\quad$ **enquanto** $(s \notin G) \wedge (profundidade < profundidadeMaxima)$ **faça**
7 $\quad\quad profundidade \leftarrow profundidade + 1$;
8 $\quad\quad$ **para cada** $a \in A(s)$ **faça**
9 $\quad\quad\quad Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{t-1}(s')$;
10 $\quad\quad$ **fim**
11 $\quad\quad \pi(s) \leftarrow \arg\max_{a \in A} Q(s,a)$;
12 $\quad\quad V^t(s) \leftarrow \max_{a \in A} Q(s,a)$;
13 $\quad\quad s \leftarrow EscolhaProximoEstado(s, \pi(s))$;
14 $\quad$ **fim**
15 $\quad$ *// Fim do trial*;
16 **fim**
17 **retorna** π.

2.2.4 LRTDP - Programação Dinâmica em Tempo Real com Rótulos

O algoritmo de programação dinâmica em tempo real com rótulos (*Labeled Real-Time Dynamic Programming - LRTDP*) (Bonet e Geffner, 2003) é uma melhoria aplicada ao RTDP que permite que o algoritmo convirja mais rapidamente. Ao mesmo tempo que o comportamento anytime possibilita que o RTDP possa devolver políticas boas rapidamente, também faz com que o algoritmo demore a convergir por visitar poucas vezes os estados menos prováveis. O LRTDP (Bonet e Geffner, 2003) propõe a inserção de rótulos nos estados visitados pelo RTDP quando esses estados já tiverem convergido para a solução ótima, evitando que a função valor volte a ser atualizada para esses estados e terminando trials quando um estado com valor convergido é alcançado.

Seja G^s_{Greedy} o *grafo guloso de um estado* s, i.e., o grafo formado pelos estados alcançáveis a partir de s usando a política gulosa atual. O *envelope guloso de* s é dado pelo conjunto de estados de G^s_{Greedy}.

O CheckSolved (Algoritmo 2.2.5), um procedimento auxiliar do LRTDP, recebe um estado s e um erro $\mathcal{E}$ como entrada e devolve verdadeiro se os estados no envelope guloso de s convergiram. Neste caso, CheckSolved também rotula todos os estados no envelope como convergidos.

De forma mais detalhada, o CheckSolved funciona como segue. As Linhas 1-6 inicializam as estruturas de dados usadas pelo algoritmo e a variável booleana rv informa no final do algoritmo se o estado s fornecido como entrada foi resolvido ou não. As Linhas 7-25 fazem uma busca em profundidade no grafo G^s_{Greedy} e caso algum estado não convergido seja encontrado, a variável booleana rv recebe falso, ou seja, s não convergiu. Por outro lado, se rv continuar com valor verdadeiro após a execução do laço nas Linhas 7-25, então s foi resolvido assim como todos os estados no envelope guloso de s. As Linhas 27-29 fazem com que o algoritmo marque os estados em G^s_{Greedy} como resolvidos se rv for verdadeiro; mas se rv for falso, as Linhas 32-38 fazem com que o algoritmo atualize mais uma vez os estados do envelope guloso.

O LRTDP (Algoritmo 2.2.6) faz as mesmas suposições que o RTDP (Seção 2.2.3). Com isso, o LRTDP devolve uma política ótima parcial para um subconjunto de S, que é o subconjunto que

Algoritmo 2.2.5: CheckSolved(s, $\mathcal{E}$) (Bonet e Geffner, 2003)

Entrada: s (um estado do MDP), $\mathcal{E}$ (o erro máximo permitido).
Saída: rv (um valor booleano que é verdadeiro se o estado s foi resolvido; ou falso, caso contrário).

```
1  rv ← true;
2  abertos ← PILHA − VAZIA;
3  fechados ← PILHA − VAZIA;
4  se ¬s.Resolvido então
5  |  abertos.empilhe(s);
6  fim
7  enquanto ¬abertos.vazia() faça
8  |  s ← abertos.desempilhe();
9  |  fechados.empilhe(s);
10 |  para cada a ∈ A(s) faça
11 |  |  Q(s,a) ← R(s,a) + γ Σ_{s'∈S} P(s'|s,a) V^{t−1}(s');
12 |  fim
13 |  π(s) ← arg max_{a∈A} Q(s,a);
14 |  V^t(s) ← max_{a∈A} Q(s,a);
15 |  residual = |V^{t−1}(s) − V^t(s)|;
16 |  se residual > ε então
17 |  |  rv ← false;
18 |  |  continue;
19 |  fim
20 |  para cada s' tal que P(s'|s,π(s)) > 0 faça
21 |  |  se ¬s'.Resolvido ∧ s' ∉ (abertos ∪ fechados) então
22 |  |  |  abertos.empilhe(s');
23 |  |  fim
24 |  fim
25 fim
26 se rv = true então
27 |  para cada s' ∈ fechados faça
28 |  |  s'.Resolvido ← true;
29 |  fim
30 fim
31 senão
32 |  enquanto ¬fechados.vazia() faça
33 |  |  s ← fechados.desempilhe();
34 |  |  para cada a ∈ A(s) faça
35 |  |  |  Q(s,a) ← R(s,a) + γ Σ_{s'∈S} P(s'|s,a) V^{t−1}(s');
36 |  |  fim
37 |  |  V^t(s) ← max_{a∈A} Q(s,a);
38 |  fim
39 fim
40 retorna rv
```

Algoritmo 2.2.6: LRTDP (M, s_0, G) (Bonet e Geffner, 2003)

Entrada: M: um MDP, s_0: um estado inicial,
G: um conjunto de estados meta (G é opcional, depende se $\gamma \in]0,1[$ é ou não utilizado para computar a atualização de Bellman).
Saída: π: uma política ótima parcial

$V^0 \leftarrow \frac{\max_{s \in S, a \in A} R(s,a)}{1-\gamma}$ *(uma heuristíca admissível)*;
enquanto $\neg s_0.Resolvido$ **faça**
  // Inicio da rodada;
  $visitados \leftarrow PILHA - VAZIA$;
  $s \leftarrow s_0$;
  enquanto $\neg s.Resolvido$ **faça**
    $visitados.empilhe(s)$;
    se $s \in G$ **então**
      interrompa
    fim
    ;
    para cada $a \in A(s)$ **faça**
      $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{t-1}(s')$;
    fim
    $\pi(s) \leftarrow \arg\max_{a \in A} Q(s,a)$;
    $V^t(s) \leftarrow \max_{a \in A} Q(s,a)$;
    $s \leftarrow EscolhaProximoEstado(s, \pi(s))$;
  fim
  enquanto $\neg visitados.vazia()$ **faça**
    $s \leftarrow visitados.desempilhe()$;
    se $\neg CheckSolved(s, \mathcal{E})$ **então**
      interrompa
    fim
  fim
  // Fim do trial;
fim
retorna π;

contém apenas os estados relevantes de S. Por utilizar uma pilha de estados visitados, a ordem de atualização dos estados visitados acontece de maneira inversa e com isso, o último estado a convergir é o estado inicial s_0. No início da execução, apenas os estados meta convergiram (Bonet e Geffner, 2003).

Considere novamente o exemplo da Figura 2.2 e suponha que apenas a política gulosa em cada estado é exibida. Além disso, considere que s_1 e s_3 já foram resolvidos e que s_1 já convergiu (apesar de ainda não ter sido marcado como resolvido). Se o checkSolved for chamado passando s_0 como parâmetro, o algoritmo executaria as Linhas 1-16 normalmente, pularia para a Linha 20 e adicionaria os sucessores s_1 e s_2 na pilha, nessa ordem. Na iteração seguinte, s_2 estaria no topo da pilha *abertos* e seria expandido, gerando como próximo estado s_3, que por sua vez, não teria sucessores distintos. O algoritmo voltaria para a Linha 7, removendo desta vez s_1 de *abertos* e verificando nas Linhas 20 e 21 que s_1 não tem sucessores não visitados. Como o algoritmo não executa as Linhas 17-18 devido às suposições iniciais, o algoritmo vai para as Linhas 26-30 e marca os 4 estados como resolvidos. Na Linha 34, o algoritmo devolve rv que é verdadeira, o que significa que s_0 foi resolvido.

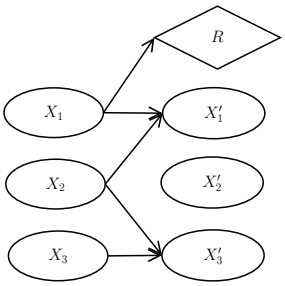
2.3 MDPs fatorados

Os algoritmos de programação dinâmica assíncrona permitem que MDPs enumerativos maiores sejam resolvidos, mas existem outras técnicas que têm sido estudadas com a finalidade de resolver problemas grandes. A dificuldade em resolver esses problemas surge devido à enumeração completa de estados (Ghallab *et al.*, 2004), que é realizada por muitos algoritmos. Uma maneira de evitar isso é a utilização de MDPs fatorados (Boutilier *et al.*, 1999).

Um MDP fatorado (Boutilier *et al.*, 1999) é uma maneira mais econômica de representar um MDP. Essa economia na representação é obtida através de uma modelagem diferente para estados, funções de recompensa, transição e valor. No MDP fatorado, a representação é feita com base em um conjunto de variáveis de estado $X = \{X_1, ..., X_n\}$. Dessa forma, cada estado s é dado por um vetor $\vec{x} = (x_1, ..., x_n)$ de atribuições em que cada x_i é um valor booleano. O conjunto completo de estados deste MDP tem um tamanho máximo de 2^n. A vantagem em utilizar representações fatoradas é que se torna possível utilizar diferentes estruturas de dados que aproveitam essa representação, removendo redundâncias do problema e explorando independências.

Para representar ações em um MDP fatorado, é necessário definir Redes Bayesianas Dinâmicas (*Dynamic Bayesian Networks - DBNs*)(Dean e Kanazawa, 1990) para cada uma dessas ações. Uma DBN para uma ação a é um grafo orientado acíclico que tem duas camadas. A primeira camada representa um conjunto X de variáveis de estado em um estado atual. A segunda camada representa um conjunto X' de variáveis de estado de um estado seguinte. Os arcos de X para X' representam as dependências entre as variáveis de estado sob uma ação a, e com base nesse relacionamento, tem-se que uma variável X_i é pai de uma variável X'_j se existe um arco de X_i para X'_j (que é o mesmo que dizer que X'_j depende de X_i). Também existem casos em que há dependência de uma variável na camada X' para outra variável que também está na camada X'.

Além das DBNs, são utilizadas tabelas de probabilidade condicional (*Conditional Probability Tables - CPTs*), que fornecem a probabilidade de uma variável X'_j ser verdadeira ou falsa; e uma função recompensa $R(\vec{x}, a)$ que tem valores de acordo com as possíveis atribuições para X. A vantagem de usar CPTs baseadas em DBNs é que não é necessário enumerar todas as possibilidades e assim, são consideradas apenas as variáveis pais de X'_j, referenciadas por $pais(X'_j)$. A Figura 2.3 apresenta um exemplo gráfico de uma DBN e suas CPTs, além da representação tabular da função recompensa.



(a) Exemplo de DBN.

$P(X'_1|\{X_1, X_2\}, a)$

X_1	X_2	$X'_1 = 1$	$X'_1 = 0$
0	0	0.65	0.35
0	1	0.7	0.3
1	0	0.8	0.2
1	1	0.8	0.2

$P(X'_2|a)$

$X'_2 = 1$	$X'_2 = 0$
0.7	0.3

$P(X'_3|\{X_2, X_3\}, a)$

X_2	X_3	$X'_3 = 1$	$X'_3 = 0$
0	0	0.5	0.5
0	1	1	0
1	0	0.7	0.3
1	1	0.7	0.3

$R(X_1, a)$

X_1	$R(X_1, a)$
0	0
1	1

(b) Exemplos de CPTs.

Figura 2.3: *Exemplo de MDP fatorado com uma ação. (a) Dependência entre variáveis de estado em uma DBN. Ovais à esquerda são variáveis do estado atual, ovais à direita são variáveis do estado seguinte, o losango é a função recompensa e os arcos são relações de dependência. (b) CPTs para cada uma das variáveis de estado seguinte X'_j; e tabela da função recomepensa.*

Utilizando representações fatoradas, as probabilidades de transição são calculadas conforme a Equação 2.9, que é a probabilidade conjunta de se alcançar $\vec{x'}$ ao executar a em $\vec{x}$:

$$P(\vec{x'}|\vec{x}, a) = \prod_{i=1}^{n} P(x'_i|pais(X'_i), a). \tag{2.9}$$

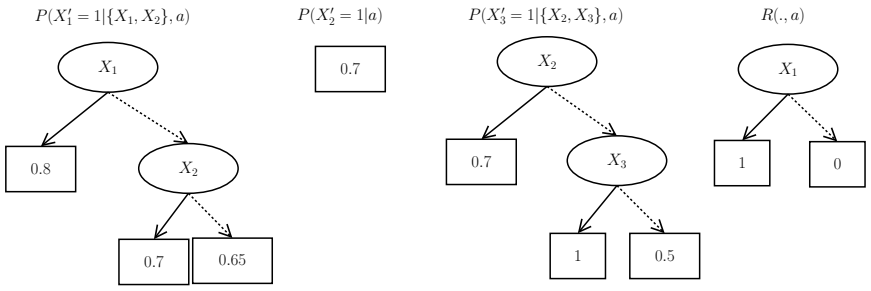


Figura 2.4: *ADDs de transição considerando apenas $X'_j = 1$; e ADD da função recompensa. Nos ADDs, ovais são variáveis de estado, folhas são o valor de recompensa ou probabilidade, linhas sólidas são atribuições verdadeiras e linhas tracejadas são atribuições falsas. (Givan et al., 2000).*

2.3.1 Diagramas de Decisão

Um modo mais eficiente para representar as CPTs é uma estrutura de dados chamada Diagrama de Decisão Algébrica (*Algebraic Decision Diagram - ADD*) (Bahar *et al.*, 1993). Os ADDs estendem os Diagramas de Decisão Binária (*Binary Decision Diagrams - BDDs*) (Bryant, 1986), que são árvores de decisão representadas de um modo mais compacto com o objetivo de definir eficientemente funções com variáveis binárias para um resultado binário, ou seja, $f : \mathbb{B}^n \mapsto \mathbb{B}$. ADDs são usados para representar funções com variáveis binárias para resultados reais, ou seja, $f : \mathbb{B}^n \mapsto \mathbb{R}$. Com isso, para resolver um MDP, pode-se representar cada função como um ADD.

Os ADDs são a melhor opção quando comparados com as CPTs e árvores de decisão por não enumerarem todos os possíveis valores para as variáveis de estado, ou seja, os ADDs simplificam a função de transição aproveitando a independência específica de contexto (*Context-Specific Independence - CSI*) (Boutilier *et al.*, 1996), que ocorre quando a probabilidade de uma variável X'_j independe de pelo menos uma variável pai X_i para determinadas valorações das variáveis em $pais(X'_j)$. A primeira CPT da Figura 2.3(b) apresenta CSI nas 3ª e 4ª linhas porque as probabilidades de X'_1 independem de X_2 nesses casos. A Figura 2.4 apresenta as mesmas funções das CPTs da Figura 2.3(b) considerando que $X'_j = 1$ (o complemento de $X'_j = 1$ é dado por $1 - P(X'_j = 1|., a)$), mas representadas por ADDs e removendo CSI. Observe que se $X_1 = 1$ no ADD da função $P(X'_1 = 1|\{X_1, X_2\}, a)$, não importa o valor de X_2, $P(X'_1 = 1|\{X_1, X_2\}, a) = 0.8$.

Formalmente, um ADD é um grafo orientado acíclico $G = (\mathcal{V} \cup \mathcal{T}, E)$ (Bahar *et al.*, 1993), em que:

- $\mathcal{V}$ é um conjunto de nós internos, com dois arcos que saem de cada nó $v \in \mathcal{V}$: $(then)$ é o arco que representa que a variável em v é verdadeira; e $(else)$ é o arco que representa que a variável em v é falsa;
- $\mathcal{T}$ é um conjunto de nós terminais (ou folhas) em que cada $t \in \mathcal{T}$ tem um valor $\mathbb{R}$; e
- E é o conjunto de arcos do grafo, em que cada arco é composto por um nó de origem e um nó de destino.

O valor dos nós internos em $\mathcal{V}$ depende dos caminhos que chegam até nós terminais a partir de v. Ou seja, dado um nó $v \in \mathcal{V}$, a função em v tem como valor:

$$f(v) = \begin{cases} f_{then}, \text{se v = T} \\ f_{else}, \text{caso contrário,} \end{cases} \tag{2.10}$$

em que $(\wedge)$ representa uma conjunção, $(\vee)$ representa uma disjunção, f_{then} representa o valor de f considerando o arco then, f_{else} o valor de f considerando o arco else e $f(v)$ é um número real se $v \in \mathcal{T}$. A seguir, as principais operações utilizadas neste trabalho são apresentadas juntamente com exemplos de cada uma:

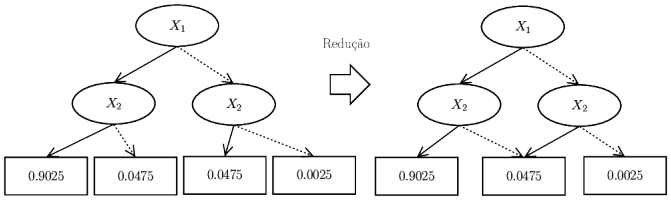
- *redução de um ADD*: consiste em agrupar folhas que possuam um mesmo valor no ADD (Figura 2.5(a));

Operação	Complexidade de pior caso
Redução	$O(\|\mathcal{G}_1\|log(\|\mathcal{G}_1\|)$
Restrição	
Intersecção	
União	$O(\|\mathcal{G}_1\|\|\mathcal{G}_2\|)$
$Max(A_1, A_2)$	
$Min(A_1, A_2)$	
Marginalização	$O(\|\mathcal{G}_1\|^2)$

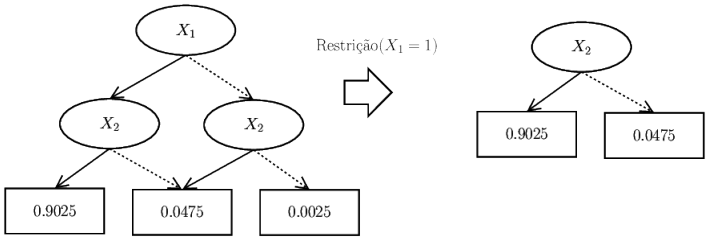
Tabela 2.3: *Complexidade de pior caso das diferentes operações envolvendo BDDs (Bryant, 1986).*

- *restrição sobre um nó interno de um ADD*: recebe um ADD A_1, um nó interno v (representado por X_i), e restringe o valor do nó interno para que esse nó seja verdadeiro ou falso (Figura 2.5(b));
- *produto (intersecção) ($\otimes$ ou $\cap$)*: recebe dois ADDs, A_1 e A_2, e devolve $A_3 = A_1 \otimes A_2$ (Figura 2.5(c));
- *soma (união) ($\oplus$ ou $\cup$)*: recebe dois ADDs, A_1 e A_2, e devolve $A_3 = A_1 \oplus A_2$; no caso de BDDs, a união é computada com $A_3 = (A_1 \oplus A_2) \ominus (A_1 \otimes A_2)$ (Figura 2.5(d));
- *marginalização (sum-out)*: recebe um ADD A_1 e um nó interno v (representado por X_i), realiza duas restrições sobre o nó interno, primeiro com valor verdadeiro e depois com valor falso (gerando dois ADDs menores), e por fim, soma os dois ADDs resultantes das operações de restrição, i.e., $\sum_{X_i \in X} A_1$ (Figura 2.5(e));
- *máximo de dois ADDs*: recebe dois ADDs, A_1 e A_2, e devolve o máximo de A_1 e A_2 considerando cada atribuição nos dois ADDs, que resulta em um terceiro ADD (Figura 2.5(f)); e
- *mínimo de dois ADDs*: recebe dois ADDs, A_1 e A_2, e devolve o mínimo de A_1 e A_2 considerando cada atribuição nos dois ADDs, que resulta em um terceiro ADD (Figura 2.5(g)).

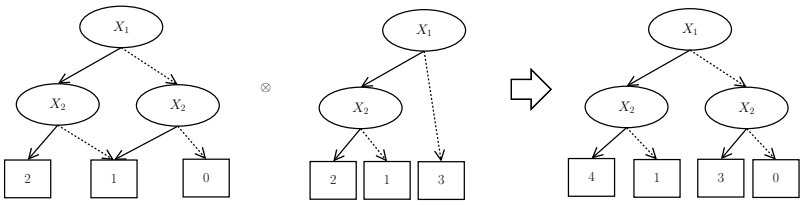
Sejam $\mathcal{G}_1 = \mathcal{V}_1 \cup \mathcal{T}_1$ e $\mathcal{G}_2 = \mathcal{V}_2 \cup \mathcal{T}_2$ os nós de dois ADDs. A complexidade de pior caso (Apêndice A) para cada uma das operações é apresentada na Tabela 2.3(Bryant, 1986).



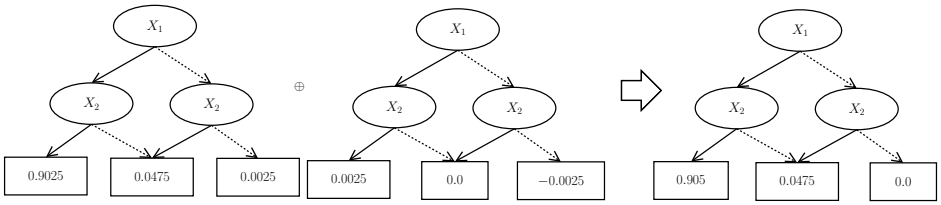
(a) Redução em um ADD.



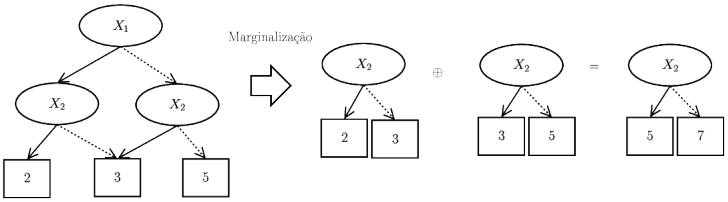
(b) Restrição sobre a variável X_1 em um ADD.



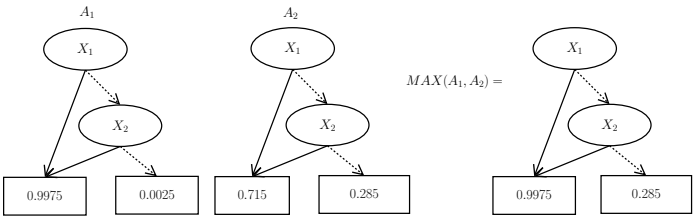
(c) Produto de dois ADDs.



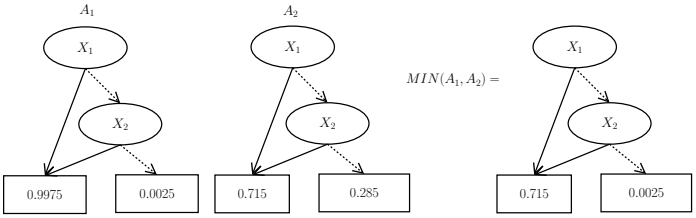
(d) Soma de dois ADDs.



(e) Marginalização sobre variável X_1 em um ADD.



(f) Máximo de dois ADDs.



(g) Mínimo de dois ADDs.

Figura 2.5: *Exemplos de operações envolvendo ADDs.*

2.3.2 Soluções para MDPs fatorados

Esta seção apresenta resumidamente algumas das técnicas baseadas em programação dinâmica que podem ser utilizadas para se resolver MDPs fatorados.

SPUDD

O SPUDD (*Stochastic Planning using Decision Diagrams*) (Hoey *et al.*, 1999) é uma extensão do algoritmo de iteração de valor que encontra a solução de MDPs fatorados. Para isso ser possível, o SPUDD utiliza ADDs para representar as funções de recompensa, transição probabilística e valor. A representação dessas funções através de ADDs é mais eficiente do que as representações baseadas em árvores de decisão como é feito em um algoritmo conhecido como SPI (*Structured Policy Iteration*) (Boutilier *et al.*, 1995), uma versão fatorada de iteração de política.

APRICODD

O APRICODD (*Approximate Policy Construction using Decision Diagrams*) (St-Aubin *et al.*, 2000) é uma versão aproximada do SPUDD. Por obter soluções aproximadas, o APRICODD utiliza menos recursos computacionais e possibilita que problemas com um número ainda maior de estados sejam resolvidos (St-Aubin *et al.*, 2000). A vantagem do APRICODD está em reduzir o número de nós folha dos ADDs agrupando as que tiverem valores similares em intervalos fechados da forma $[l, u]$ que limitam inferiormente e superiormente a função valor. Contudo, o APRICODD relaxa a restrição de encontrar políticas ótimas para encontrar políticas aproximadas.

sRTDP

O algoritmo sRTDP (*Symbolic Real-Time Dynamic Programming*) (Feng *et al.*, 2003) é uma extensão do RTDP para MDPs fatorados que utiliza ADDs para representar as seguintes funções: recompensa, transição probabilística e valor. Além disso, o sRTDP generaliza cada atualização de um estado s para um estado abstrato B_i em que $s \in B_i$. Os estados abstratos são gerados dinamicamente no momento em que cada estado é visitado pelo sRTDP.

2.4 Resumo do Capítulo

Nesse capítulo foi apresentado o seguinte conteúdo:

- o MDP, representado de forma enumerativa ou fatorada, que serve para modelar problemas de planejamento probabilistico;
- o embasamento matemático necessário para calcular uma solução de um MDP, i.e., uma política ótima;
- algoritmos usados para encontrar uma política ótima, e.g., Iteração de Valor e (L)RTDP; e
- o ADD, estrutura de dados fundamental no desenvolvimento de soluções eficientes para MDPs.

Capítulo 3

BMDP - MDP com intervalos

No Capítulo 2 foram vistos os MDPs, que permitem a modelagem de problemas de planejamento probabilístico com probabilidades precisas. Neste capítulo, são apresentados os BMDPs, que estendem os MDPs em termos das funções de transição e recompensa. Além deles, é importante mencionar os seguintes modelos:

- MDPs com probabilidades imprecisas (*Markov Decision Processes with Imprecise Probabilities - MDPIPs*) (Delgado, 2010; Satia e Jr., 1973), que são MDPs em que as probabilidades de transição são dadas por restrições sobre um conjunto de parâmetros; e
- MDPs com transições para conjuntos de estados (*Markov Decision Process with Set-Valued Transitions - MDPSTs*) (Trevizan *et al.*, 2006, 2007, 2008), que são MDPs em que as transições ocorrem probabilisticamente para conjuntos de estados, mas nos conjuntos os estados são escolhidos de forma não-determinística.

Os BMDPs e MDPSTs são casos especiais de MDPIPs (Trevizan *et al.*, 2007). Com isso, o MDPIP é o modelo mais geral para planejamento com probabilidades imprecisas. Neste capítulo, descrevemos em detalhes o BMDP uma vez que ele é o modelo reduzido resultante da bissimulação estocástica aproximada que será apresentada no Capítulo 5.

3.1 BMDPs - Definição

Um MDP com intervalos (*Bounded-parameter Markov Decision Process - BMDP*) (Dean *et al.*, 1997; Givan *et al.*, 2000) é uma generalização do modelo MDP. Um BMDP pode ser visto como um conjunto de MDPs no qual as probabilidades de transição e as recompensas são representadas por intervalos fechados sobre os números reais. A Figura 3.1 mostra uma transição num BMDP em que a probabilidade do agente ir para o estado s_1 após executar a ação a no estado s_0 é dada pelo intervalo $[0.2, 0.4]$. Analogamente, se em um MDP a função recompensa é $R(s_1, a) = 0.8$; em um BMDP pode-se ter $R(s_1, a) \in [0.7, 0.9]$.

Além dos BMDPs poderem ser utilizados para representar incerteza nas funções de recompensa e transição probabilística, dado um MDP grande, pode-se obter, através de algoritmos eficientes, um BMDP contendo um número menor de estados cuja solução é aproximadamente igual à solução do MDP original. Esses algoritmos são conhecidos como *técnicas de agregação de estados* (Givan *et al.*, 2000; Li *et al.*, 2006; Ravindran e Barto, 2004) e é com essas técnicas que os BMDPs são obtidos neste trabalho.

Definição 2. *Um BMDP é uma tupla* $M_{\updownarrow} = (S, A, P_{\updownarrow}, R_{\updownarrow}, \gamma)$ *(Givan et al., 2000), em que:*

- S *é um conjunto finito de estados que podem ser observados no decorrer do tempo.*
- A *é um conjunto finito de ações que o agente pode executar. Além disso,* $A(s)$ *é um subconjunto de* A *que especifica quais ações podem ser executadas no estado* $s \in S$.

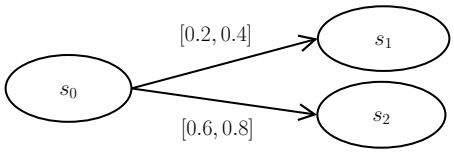


Figura 3.1: *Probabilidades de transição em um BMDP: a execução da ação a leva o agente de s_0 para s_1 com uma probabilidade dada pelo intervalo $[0.2, 0.4]$ e para s_2 com probabilidade entre $[0.6, 0.8]$.*

- *$P_{\updownarrow}(s'|s,a)$ é a medida de probabilidade de transição dada por um intervalo real $[l,u]$, ou seja, as probabilidades mínima e máxima de se alcançar o estado s' ao executar a ação a em um estado s. Para se referir ao limite inferior, utiliza-se $P_{\downarrow}$; e para se referir ao limite superior, utiliza-se $P_{\uparrow}$. O intervalo $[l,u]$ é válido se $0 \leq l \leq u \leq 1$; e a distribuição dada por intervalos a partir de um par (s,a) é válida se: $\sum_{s' \in S} P_{\downarrow}(s'|s,a) \leq 1$ e $\sum_{s' \in S} P_{\uparrow}(s'|s,a) \geq 1$. Ou seja, selecionando uma ação a, a soma dos limites inferiores para as transições deve ser menor ou igual a 1; enquanto a soma dos limites superiores para as transições deve ser maior ou igual a 1 (Givan et al., 2000).*
- *$R_{\updownarrow}(s,a)$ é dada por um intervalo real $[l,u]$ e representa os possíveis valores de recompensa obtidos com a execução da ação a no estado s. Para se referir ao limite inferior, utiliza-se $R_{\downarrow}$ e para se referir ao limite superior, utiliza-se $R_{\uparrow}$.*
- *γ é o fator desconto, sendo que $0 < \gamma < 1$.*

Um BMDP $M_{\updownarrow}$ (Definição 2) define um conjunto de MDPs sendo que um MDP $M \in M_{\updownarrow}$ é bem formado se as seguintes restrições são satisfeitas (Givan *et al.*, 2000):

1. $R(s,a) \in R_{\updownarrow}(s,a)$ para todo par (s,a);
2. $P(s'|s,a) \in P_{\updownarrow}(s'|s,a)$ para toda transição que alcança um estado s' dado o par (s,a); e
3. $\sum_{s' \in S} P(s'|s,a) = 1$ para todo par (s,a).

> **BMDP - Modelo Conceitual:** *Um BMDP modela um sistema dinâmico em que um agente interage com um ambiente e deve lidar com efeitos probabilísticos incertos causados pela Natureza. A Natureza, por sua vez, pode ser vista como um outro agente que interfere positiva ou negativamente nas interações do agente com o sistema. Com isso, o agente observa o estado atual, escolhe uma ação que possui efeitos probabilísticos imprecisos devido à presença da Natureza e recebe uma recompensa pela ação escolhida no estado atual. Após executar essa ação, o sistema fornece um novo estado. O objetivo do agente é maximizar a recompensa ao longo de uma sequência de ações escolhidas tendo em vista que a Natureza pode auxiliá-lo ou atrapalhá-lo no objetivo de maximizar sua recompensa.*

Assim, resolver um BMDP é um pouco diferente de resolver um MDP porque é necessário observar que esse modelo é visto como um conjunto de MDPs bem formados obtidos a partir da atribuição de probabilidades exatas contidas nos intervalos do BMDP. Mais do que isso, geralmente são obtidos MDPs que maximizam ou minimizam o valor de qualquer política π para qualquer MDP do conjunto (Givan *et al.*, 2000). Em outras palavras, para resolver um BMDP não basta considerar apenas as melhores ações que o agente deve tomar, mas também o comportamento da Natureza. A Natureza pode ser vista como um outro agente que atua no mesmo ambiente e que apresenta os seguintes comportamentos:

- *Natureza aliada*: neste caso, o agente pode considerar que as escolhas da Natureza maximizam sua recompensa esperada descontada; ou
- *Natureza adversária*: na qual o agente pode considerar que as escolhas da Natureza minimizam sua recompensa esperada descontada.

Devido à utilização de funções intervalares na definição de BMDPs e das possíveis escolhas da Natureza, a solução de um BMDP também é baseada em intervalos. Uma *função valor intervalar* $V_{\updownarrow}$ mapeia estados para intervalos sobre os números reais e é uma generalização da função valor dos MDPs (Seção 2.1).

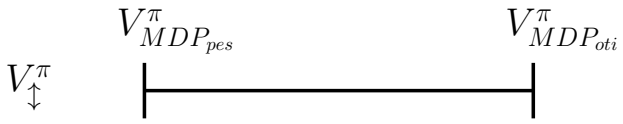


Figura 3.2: *Visualização gráfica da formação dos intervalos da função valor em um BMDP.*

A Figura 3.2 mostra como são formadas as estimativas da função valor intervalar. Dado um BMDP $M_{\updownarrow}$, a função valor $V_{\updownarrow}$ é dada inferiormente pela estimativa da função valor ótima de um MDP pessimista e superiormente pela estimativa da função valor ótima em um MDP otimista. Esses MDPs são obtidos de diferentes maneiras através da simulação das escolhas da Natureza.

Para calcular uma política ótima para os BMDPs, é necessário definir critérios para ordenar a função valor intervalar e encontrar aquela com o melhor valor. Os critérios de ordenação de intervalos podem ser otimistas ($\leq_{oti}$) ou pessimistas ($\leq_{pes}$) (Givan *et al.*, 2000), respectivamente apresentados a seguir:

$$[l_1, u_1] \leq_{oti} [l_2, u_2] \iff (u_1 < u_2) \vee (u_1 = u_2 \wedge l_1 \leq l_2) \tag{3.1}$$

e

$$[l_1, u_1] \leq_{pes} [l_2, u_2] \iff (l_1 < l_2) \vee (l_1 = l_2 \wedge u_1 \leq u_2). \tag{3.2}$$

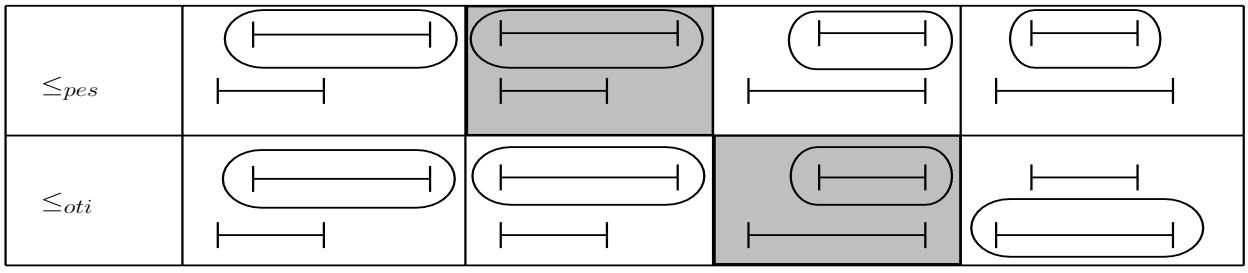


Figura 3.3: *Possíveis situações com relação aos critérios de ordenação para dois intervalos. Para cada situação, o intervalo circulado é considerado maior com relação aos operadores de intervalos $\leq_{pes}$ e $\leq_{oti}$ respectivamente (Franco, 2012).*

No critério de ordenação otimista (Equação 3.1), primeiramente ordena-se os intervalos pelos limites superiores e no caso de empate, ordena-se pelos limites inferiores. No critério pessimista (Equação 3.2), a ordem em que se verifica os valores dos intervalos é invertida, verifica-se os limites inferiores, e no caso de empate, os limites superiores. A Figura 3.3 ilustra as possíveis situações de 2 intervalos ordenados pelos critérios $\leq_{oti}$ e $\leq_{pes}$. Os casos de empate estão ilustrados pelas regiões com fundo cinza. Em ambos os critérios, é difícil ocorrer empates porque a função valor é dada por intervalos reais. Por esse motivo, existem trabalhos em que é realizada apenas a comparação de um dos extremos para ordenar os intervalos da função valor com os critérios puramente otimistas e puramente pessimistas (Buffet e Aberdeen, 2005). Consequentemente, a função valor passa a ser um número real, ao invés de uma função intervalar.

Neste trabalho, vamos considerar a Natureza adversária (puramente pessimista), que permite a obtenção de políticas ótimas robustas, i.e., políticas ótimas considerando o pior caso (Buffet e Aberdeen, 2005). Para isso, é necessário encontrar $\min_{M \in M_{\updownarrow}}$, i.e., um MDP $M \in M_{\updownarrow}$ que minimize o valor de qualquer política π. Ou seja, dada uma política π qualquer, é possível encontrar um MDP $M \in M_{\updownarrow}$ que minimiza o valor de π em todo estado $s \in S$ (Givan *et al.*, 2000), ou seja:

$$\forall s \in S, V_M^{\pi}(s) \leq V_{M' \in M_{\updownarrow}}^{\pi}(s), \tag{3.3}$$

sendo $V_M^{\pi}(s) = V_{MDP_{pes}}^{\pi}$.

3.2 BMDPs - Soluções

Considerando o critério de ordenação dos intervalos $\leq_{pes}$, é possível obter uma política ótima pessimista que é denotada por π_{pes}^*. A *equação de Bellman para BMDPs* considerando políticas ótimas pessimistas é dada por (Givan *et al.*, 2000):

$$V_{pes}^*(s) = \max_{a \in A}\{R_{\downarrow}(s,a) + \gamma \min_{P \in P_{\updownarrow}} \sum_{s' \in S} P(s'|s,a) V_{pes}^*(s')\}, \tag{3.4}$$

em que $\min_{P \in P_{\updownarrow}}$ é a distribuição computada pelo Algoritmo 3.2.1 considerando o par (s,a).

Com base nos critérios de ordenação apresentados para a função valor intervalar, foi criado o algoritmo Iteração de Valor Intervalar (*Interval Value Iteration - IVI*) (Givan *et al.*, 2000), que encontra π_{pes}^* para um dado BMDP. Nas seções seguintes, são apresentados os algoritmos para encontrar soluções para BMDPs, dentre eles, o IVI. Além disso, é usado o critério de ordenação $\leq_{pes}$ considerando apenas o limitante inferior dos intervalos.

3.2.1 Iteração de Valor Intervalar

Assim como Iteração de Valor, foi proposta uma solução básica para BMDPs que é conhecida como Iteração de Valor Intervalar (*Interval Value Iteration - IVI*) (Dean *et al.*, 1997; Givan *et al.*, 2000). O IVI recebe um BMDP e um erro $\mathcal{E}$ como entrada e devolve uma política ótima pessimista como solução.

Para computar π_{pes}^*, IVI utiliza a *Equação de Bellman para BMDPs* (Equação 3.4) da seguinte maneira: antes de escolher a melhor ação em cada iteração, deve encontrar uma função de transição pessimista com base em cada par (s,a) através do Algoritmo 3.2.1. Depois de encontrar a função de transição pessimista, basta atualizar os valores dos estados como seria feito com o algoritmo VI. Como IVI se baseia em VI, apresenta o mesmo problema, que é ter de atualizar os valores de todo o conjunto de estados, sendo ineficiente em problemas grandes.

A atualização de Bellman para BMDPs, considerando apenas $V_{\downarrow}$, que tem por objetivo encontrar π_{pes}^*, é dada pela equação a seguir que combina os conceitos de ordenação de função valor intervalar e de MDPs que minimizam o valor de qualquer política (Givan *et al.*, 2000):

$$V_{\downarrow}^{t+1}(s) = \max_{a \in A, \leq_{pes}} \{R_{\downarrow}(s,a) + \gamma \min_{P \in P_{\updownarrow}} P(s'|s,a) V_{\downarrow}^t(s')\}. \tag{3.5}$$

O Algoritmo 3.2.1 implementa o comportamento da Natureza adversária em cada estado da seguinte maneira: recebe um par (s,a), obtém um subconjunto de S com k estados alcançáveis considerando $P_{\updownarrow}(.|s,a)$ e ordena esses estados de acordo com o critério pessimista $\leq_{pes}$ (Critério 3.2), ou seja, $V_{\updownarrow}(s'_1) \leq_{pes} ... \leq_{pes} V_{\updownarrow}(s'_k)$. Depois disso, encontra o maior índice r tal que (Givan *et al.*, 2000):

$$\sum_{i=1}^{r-1} P_{\uparrow}(s'_i|s,a) + \sum_{i=r}^{k} P_{\downarrow}(s'_i|s,a) \leq 1. \tag{3.6}$$

Algoritmo 3.2.1: ObtemPiorModelo(s, a, V) (Buffet e Aberdeen, 2005)

Entrada: (s, a): um par estado e ação; e $V_{\updownarrow}$: a função valor estimada para todo $s \in S$.
Saída: P: uma função de transição que minimiza o valor de qualquer política através da associação de maiores probabilidades para os estados com menor valor.

$L \leftarrow (s'_1, ..., s'_k)$: *ordenação dos estados* $s' \in S$ *t.q.* $P_{\downarrow}(s'|s, a) > 0$. *A lista* L *é colocada em ordem crescente de acordo com a função valor intervalar estimada e o critério* $\leq_{pes}$ (a ordenação pode ser realizada pelo MergeSort ou qualquer outro algoritmo de ordenação (Cormen *et al.*, 2009))
$limiteMinimo \leftarrow 0$
para cada $s' \in L$ **faça**
  $limiteMinimo \leftarrow limiteMinimo + P_{\downarrow}(s'|s, a)$
fim
$limite \leftarrow limiteMinimo$
$i \leftarrow 1$
enquanto $limite - P_{\downarrow}(L[i]|s, a) + P_{\uparrow}(L[i]|s, a) < 1$ **faça**
  $limite \leftarrow limite - P_{\downarrow}(L[i]|s, a) + P_{\uparrow}(L[i]|s, a)$
  $P(L[i]|s, a) \leftarrow P_{\uparrow}(L[i]|s, a)$
  $i \leftarrow i + 1$
fim
$r \leftarrow i$
$P(L[r]) \leftarrow 1 - (limite - P_{\downarrow}(L[r]|s, a))$
para cada $i \in \{r + 1, ..., k\}$ **faça**
  $P(L[i]|s, a) \leftarrow P_{\downarrow}(L[i]|s, a)$
fim
retorna P

Ao determinar o índice r da Equação 3.6, é possível atribuir as maiores probabilidades para os estados com menores valores de acordo com o critério $\leq_{pes}$ (Givan *et al.*, 2000). O índice r indica o estado s'_r, que recebe o restante da probabilidade de forma a gerar um MDP bem formado. As probabilidades de transição nesse MDP são valoradas da seguinte maneira:

$$P(s'_i|s,a) = \begin{cases} P_{\uparrow}(s'_i|s,a) & i < r \\ P_{\downarrow}(s'_i|s,a) & i > r \\ 1 - \sum_{i=1, i\neq r}^{k} P(s'_i|s,a) & i = r. \end{cases} \tag{3.7}$$

Note que com uma simples modificação do Algoritmo 3.2.1, é possível obter uma distribuição otimista que se aplicada para todo par (s,a) gera um MDP M que maximiza o valor de qualquer política π, denotado por $\max_{M\in M_{\updownarrow}}$, isto é, para soluções considerando a Natureza aliada.

As Figuras 3.4(a) e 3.4(b) apresentam uma transição de um estado s_0 ao aplicar uma ação a em um BMDP e os sucessores ordenados de acordo com uma estimativa da função valor $V^t_{\updownarrow}$ para o critério $\leq_{pes}$. A Figura 3.4(c) apresenta como fica essa transição em um MDP obtido ao aplicar o Algoritmo 3.2.1 sobre o par (s,a). A ordenação realizada pelo Algoritmo 3.2.1 para esse exemplo, ocorre da seguinte maneira. Primeiramente, os estados sucessores s'_0, s'_1, s'_2, s'_3 são ordenados de forma crescente de acordo com a estimativa atual da função valor seguindo o critério $\leq_{pes}$. Depois disso, nas Linhas 2-4 é obtida a somatória dos limites inferiores das probabilidades (que seguindo a Definição 2 deve ser menor ou igual a 1). Nas Linhas 7-11, os estados são visitados de acordo com a ordenação que foi obtida anteriormente e é realizada uma tentativa de subtrair algum valor de probabilidade inferior se for possível somar um valor de probabilidade superior considerando os intervalos no BMDP. Em outras palavras, o Algoritmo 3.2.1 atribui sempre que possível probabilidades maiores para os estados com menores valores, o que caracteriza o comportamento da Natureza adversária. Na linha 13, o Algoritmo 3.2.1 distribui o restante da probabilidade para um estado de índice r. Nas linhas 14-15, são atribuidos os valores de probabilidade inferiores para o restante dos estados de forma que a distribuição no MDP tenha soma igual a 1.

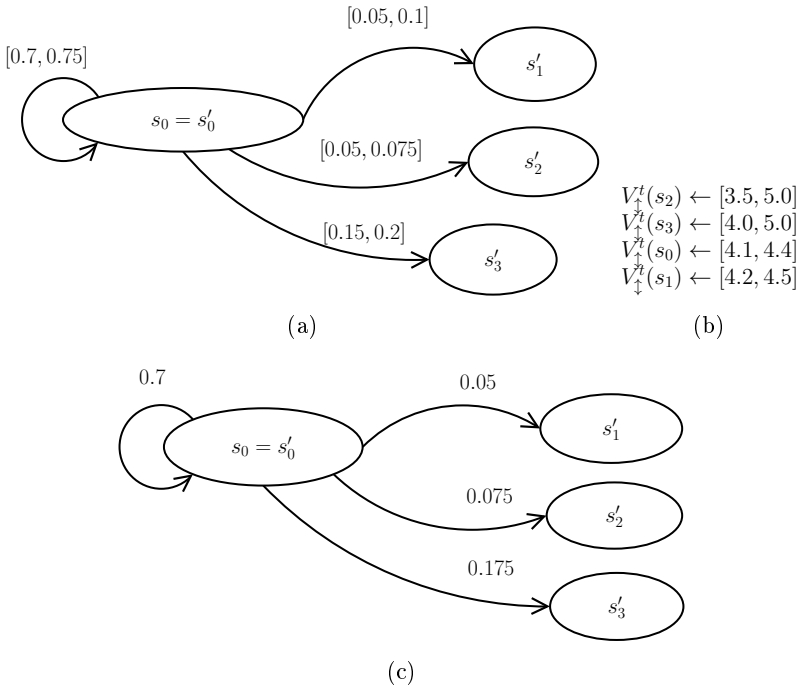


Figura 3.4: *(a) Probabilidades de transição de um BMDP considerando uma ação a executada em um estado s_0. (b) Função valor intervalar dos estados sucessores de (s,a) e ordenação dos estados segundo o critério $\leq_{pes}$. (c) Visualização gráfica do MDP pessimista obtido pelo Algoritmo 3.2.1.*

Algoritmo 3.2.2: IVI($M_{\updownarrow}$, $\mathcal{E}$) (Dean *et al.*, 1997)

Entrada: $M_{\updownarrow}$: um BMDP, $\mathcal{E}$: erro máximo permitido
Saída: π: uma política ótima pessimista

1 **para cada** $s \in S$ **faça**
2 | *Inicialize* $V_{\downarrow}^{0}(s)$ *com um valor qualquer*;
3 **fim**
4 $t \leftarrow 1$;
5 **repita**
6 | **para cada** $s \in S$ **faça**
7 | | **para cada** $a \in A(s)$ **faça**
8 | | | $P(.|s,a) \leftarrow ObtemPiorModelo(s,a)$;
9 | | | $Q(s,a) \leftarrow R_{\downarrow}(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{\downarrow}^{t-1}(s')$;
10 | | **fim**
11 | | $V_{\downarrow}^{t}(s) \leftarrow \max_{a \in A} Q(s,a)$;
12 | | $\pi(s) \leftarrow \arg\max_{a \in A} Q(s,a)$;
13 | **fim**
14 | $t \leftarrow t + 1$;
15 **até** $\max_{s \in S} |V_{\downarrow}^{t}(s) - V_{\downarrow}^{t-1}(s)| < \mathcal{E}$;
16 **retorna** π.

3.2.2 RTDP e LRTDP Robustos

Os algoritmos *RTDP* e *LRTDP Robustos* (Buffet e Aberdeen, 2005) são algoritmos de programação dinâmica assíncrona para BMDPs. Assim como nas soluções para MDPs, ao se atualizar apenas os valores de estados relevantes, esses algoritmos se tornam mais eficientes que IVI.

O LRTDP Robusto, assim como IVI, chama o Algoritmo 3.2.1 com o intuito de se obter uma distribuição pessimista do BMDP. Depois disso, o MDP resultante dessa distribuição é resolvido normalmente com atualizações como aquelas realizadas pelo LRTDP. Os algoritmos LRTDP Robusto (Algoritmo 3.2.4) e a versão robusta do CheckSolved (Algoritmo 3.2.3) são apresentados a seguir. O RTDP Robusto não é apresentado porque pode ser obtido com uma simples modificação no LRTDP Robusto (Algortimo 3.2.4).

A diferença do Algoritmo 3.2.3 com relação ao Algoritmo 2.2.5 é a inclusão da chamada ao Algoritmo 3.2.1 para obter uma distribuição pessimista quando as Linhas 11 e 37 são executadas. Além disso, para computar a função valor ótima considerando o modelo pessimista, usamos apenas os limites inferiores ao atualizar a função valor. De forma similar, a diferença do Algoritmo 3.2.4 com relação ao Algoritmo 2.2.6 é a inclusão da Linha 12, também utilizada para obter uma distribuição pessimista.

Algoritmo 3.2.3: CheckSolved-Robusto(s, $\mathcal{E}$) (Buffet e Aberdeen, 2005)

Entrada: s (um estado do BMDP), $\mathcal{E}$ (o erro máximo permitido).
Saída: rv (um valor booleano que é verdadeiro se o estado s foi resolvido; ou falso, caso contrário).

$rv \leftarrow true$;
$abertos \leftarrow PILHA - VAZIA$;
$fechados \leftarrow PILHA - VAZIA$;
se $\neg s.Resolvido$ **então**
  $abertos.empilhe(s)$;
fim
enquanto $\neg abertos.vazia()$ **faça**
  $s \leftarrow abertos.desempilhe()$;
  $fechados.empilhe(s)$;
  para cada $a \in A(s)$ **faça**
    $P(.|s,a) \leftarrow ObtemPiorModelo(s,a)$;
    $Q(s,a) \leftarrow R_{\downarrow}(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{\downarrow}^{t-1}(s')$;
  fim
  $V_{\downarrow}^{t}(s) \leftarrow \max_{a \in A} Q(s,a)$;
  $\pi(s) \leftarrow \arg\max_{a \in A} Q(s,a)$;
  $residual = |V_{\downarrow}^{t-1}(s) - V_{\downarrow}^{t}(s)|$;
  se $residual > \mathcal{E}$ **então**
    $rv \leftarrow false$;
    $continue$;
  fim
  para cada s' *tal que* $P(s'|s,\pi(s)) > 0$ **faça**
    se $\neg s'.Resolvido \wedge s' \notin (abertos \cup fechados)$ **então**
      $abertos.empilhe(s')$;
    fim
  fim
fim
se $rv = true$ **então**
  para cada $s' \in fechados$ **faça**
    $s'.Resolvido \leftarrow true$;
  fim
fim
senão
  enquanto $\neg fechados.vazia()$ **faça**
    $s \leftarrow fechados.desempilhe()$;
    para cada $a \in A(s)$ **faça**
      $P(.|s,a) \leftarrow ObtemPiorModelo(s,a)$;
      $Q(s,a) \leftarrow R_{\downarrow}(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{\downarrow}^{t-1}(s')$;
    fim
    $V_{\downarrow}^{t}(s) \leftarrow \max_{a \in A} Q(s,a)$;
  fim
fim
retorna rv

Algoritmo 3.2.4: LRTDP-Robusto ($M_{\updownarrow}$, s_0, G) (Buffet e Aberdeen, 2005)

Entrada: $M_{\updownarrow}$: um BMDP, s_0: um estado inicial,
G: um conjunto de estados meta (G é opcional, dependendo se γ é ou não utilizado para computar a atualização de Bellman para BMDPs).
Saída: π: uma política parcial ótima pessimista

$V_{\downarrow}^{0} \leftarrow \frac{\max_{s\in S, a\in A} R_{\downarrow}(s,a)}{1-\gamma}$ *(uma heurística admissível)*;
enquanto $\neg s_0.Resolvido$ **faça**
  // Inicio da rodada;
  $visitados \leftarrow PILHA - VAZIA$;
  $s \leftarrow s_0$;
  enquanto $\neg s.Resolvido$ **faça**
    $visitados.empilhe(s)$;
    se $s \in G$ **então**
      interrompa
    fim
    ;
    para cada $a \in A(s)$ **faça**
      $P(.|s,a) \leftarrow ObtemPiorModelo(s,a)$;
      $Q(s,a) \leftarrow R_{\downarrow}(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) V_{\downarrow}^{t-1}(s')$;
    fim
    $V_{\downarrow}^{t}(s) \leftarrow \max_{a\in A} Q(s,a)$;
    $\pi(s) \leftarrow \arg\max_{a\in A} Q(s,a)$;
    $s \leftarrow EscolhaProximoEstado(s, \pi(s))$;
  fim
  enquanto $\neg visitados.vazia()$ **faça**
    $s \leftarrow visitados.desempilhe()$;
    se $\neg CheckSolved(s, \mathcal{E})$ **então**
      interrompa
    fim
  fim
  // Fim do trial;
fim
retorna π;

3.3 Resumo do Capítulo

Nesse capítulo foi apresentado o seguinte conteúdo:

- o BMDP, modelo para planejamento probabilístico em que as funções de recompensa e transição são dadas por intervalos;
- o embasamento matemático necessário para resolver BMDPs de forma robusta, i.e., encontrar políticas ótimas no pior caso; e
- os algoritmos IVI e (L)RTDP Robusto, usados para encontrar soluções para BMDPs.

Capítulo 4

Bissimulação estocástica exata

Dado um MDP M, uma política ótima para M pode ser encontrada através de algoritmos como os mencionados no Capítulo 2. Uma outra forma de encontrar uma política ótima (ou aproximada) é baseada em dois passos: (1) obter um modelo equivalente (ou aproximado) de tamanho reduzido a partir do MDP original; e (2) resolver o modelo obtido de forma que a solução do modelo reduzido possa ser aplicada ao MDP original. O passo 1 descreve a *agregação de estados para MDPs*, que se baseia em mapear a representação original de um problema para uma representação abstrata que é mais compacta e fácil de resolver (Li *et al.*, 2006). O passo 2 usa o mapeamento do passo 1 para aplicar a solução do modelo de tamanho reduzido ao MDP original.

De forma geral, os algoritmos de agregação de estados partem de um MDP $M = (S, A, P, R, \gamma)$ e geram um novo MDP $M' = (S', A, P', R', \gamma)$ (ou um BMDP $M'_{\updownarrow} = (S', A, P'_{\updownarrow}, R'_{\updownarrow}, \gamma)$) que também representa um problema de planejamento probabilístico. É desejável que o conjunto de estados de M' (ou $M'_{\updownarrow}$) seja bem menor do que o conjunto de estados de M, ou seja, que $|S'| \ll |S|$, e que a solução de M' (ou $M'_{\updownarrow}$) possa ser utilizada em M.

Os algoritmos de agregação de estados apresentam ainda algumas propriedades como precisão e adaptatividade (Boutilier *et al.*, 1999). Com relação à precisão, a agregação de estados pode ser exata ou aproximada. O algoritmo é *exato* se os estados em M' são obtidos com base em igualdades nas funções do MDP, e neste caso, tudo o que é eliminado da descrição original do MDP é informação redundante. Por outro lado, o algoritmo é *aproximado* se gera estados em $M'_{\updownarrow}$ tomando como base funções aproximadamente iguais, e desta forma algumas informações da descrição original do MDP são perdidas (Boutilier *et al.*, 1999). Com relação à adaptatividade, o algoritmo é *fixo* quando apenas uma pré-computação é realizada para se obter M'; e é chamado de *dinâmico* quando o modelo de tamanho reduzido, M', se adapta a cada passo do algoritmo enquanto o problema é resolvido (Boutilier *et al.*, 1999).

Neste capítulo, apresentamos dois algoritmos de agregação de estados: redução (exato e fixo) e minimização (exato e fixo) de modelos, que geram modelos reduzidos exatos, ou seja, o resultado da agregação de estados é um novo MDP M'.

4.1 Partições do conjunto de estados

Os algoritmos de agregação de estados em geral se baseiam no conceito matemático de partições de conjuntos, no caso de um MDP, em partições do conjunto de estados S. Nesta seção, são apresentados os conceitos e operações que envolvem partições.

Dado um MDP enumerativo e o conjunto de estados S, uma partição de S é dada por $\mathcal{P} = \{B_1, ..., B_k\}$, em que $B_i \subseteq S$ e $\bigcup_{i=1}^{k} B_i = S$.

Considerada a importância de partições para os algoritmos de agregação de estados, duas formas de manipular partições são importantes: (1) *refinar uma partição (ou obter um refinamento)*, i.e., obter uma partição em que alguns blocos são divididos e; (2) *engrossar uma partição* (*coarsening*), i.e., obter uma partição através da junção de alguns blocos. Uma partição $\mathcal{P}'$ é um refinamento de uma partição $\mathcal{P}$ se e somente se cada bloco de $\mathcal{P}'$ é um subconjunto de algum bloco em $\mathcal{P}$. Assim,

se $\mathcal{P}' = \mathcal{P}$, $\mathcal{P}'$ ainda é considerada um refinamento de $\mathcal{P}$. A operação de engrossar uma partição é inversa à operação de refinar: se $\mathcal{P}'$ é um refinamento de $\mathcal{P}$, $\mathcal{P}$ é *mais grossa* do que $\mathcal{P}'$ (Givan *et al.*, 2003).

Definição 3. *Dado que $\mathcal{P}_1$ e $\mathcal{P}_2$ são partições de S descritas por conjuntos de estados enumerativos, podemos gerar um refinamento delas com a intersecção de $\mathcal{P}_1$ e $\mathcal{P}_2$, i.e., $\mathcal{P}_3 = \mathcal{P}_1 \cap \mathcal{P}_2$ de forma que cada bloco $B_k \in \mathcal{P}_3$ é calculado pela intersecção de dois blocos $B_i \in \mathcal{P}_1$ e $B_j \in \mathcal{P}_2$ tal que $B_i \cap B_j \neq \emptyset$.*

Além da possibilidade de particionar o conjunto S que é dado de forma enumerativa, é possível gerar partições do conjunto de estados de um MDP fatorado. Seja $X = \{X_1, ..., X_n\}$ um conjunto de variáveis de estado de um dado MDP e $S \subseteq 2^n$ um conjunto de estados válidos desse MDP. Uma partição de S é um conjunto de conjuntos disjuntos cuja união desses conjuntos disjuntos é igual a S. Uma partição rotulada de S é um conjunto $\mathcal{P} = \{(B_1, v_1), ..., (B_k, v_k)\}$ tal que $(\bigcup_{i=1}^{k} B_i) = S$. Além disso, quaisquer $B_i, B_j \in \mathcal{P}$ são disjuntos se $B_i \neq B_j$ e $v_i \neq v_j$, ou seja, cada bloco possui um identificador único. Cada $(B_i, v_i) \in \mathcal{P}$ é uma tupla com um bloco B_i e um rótulo único v_i que é comum para todo estado $s \in B_i$. Os blocos de estados podem ser caracterizados por expressões booleanas sobre X na forma normal disjuntiva (*Disjunctive Normal Form - DNF*, i.e., uma disjunção de conjunções). Por exemplo, $\mathcal{P} = \{(X_1, 2), (\neg X_1, 3)\}$ é uma partição rotulada com dois blocos: um bloco de estados, rotulado com 2, em que X_1 é satisfeita e um bloco de estados, rotulado com 3, em que $\neg X_1$ é satisfeita.

Definição 4. *Dado que $\mathcal{P}_1$ e $\mathcal{P}_2$ são partições rotuladas de S descritas por expressões DNF, podemos gerar um refinamento delas com a intersecção de $\mathcal{P}_1$ e $\mathcal{P}_2$, dada por $\mathcal{P}_3 = \mathcal{P}_1 \cap \mathcal{P}_2$ de forma que cada bloco $B_k \in \mathcal{P}_3$ é calculado pela conjunção ($\wedge$) de dois blocos $B_i \in \mathcal{P}_1$ e $B_j \in \mathcal{P}_2$ tal que B_i e B_j não sejam expressões DNF mutuamente exclusivas, por exemplo, $X_1 \wedge \neg X_1$. O rótulo v_k deve ser gerado de forma que seja distinto dos rótulos v_i e v_j. Por definição, uma partição de S com um único bloco é representada pela expressão verdadeiro.*

Por exemplo, dadas as partições:

$$\mathcal{P}_1 = \{(X_1, 2), (\neg X_1, 3)\} \text{ e}$$
$$\mathcal{P}_2 = \{(X_2 \wedge X_3, 5), (X_2 \wedge \neg X_3, 7), (\neg X_2, 11)\},$$

a intersecção dessas partições resulta em uma partição $\mathcal{P}_3$ (Figura 4.1):

$$\mathcal{P}_3 = \{(X_1 \wedge X_2 \wedge X_3, 10), (X_1 \wedge X_2 \wedge \neg X_3, 14), (X_1 \wedge \neg X_2, 22),$$
$$(\neg X_1 \wedge X_2 \wedge X_3, 15), (\neg X_1 \wedge X_2 \wedge \neg X_3, 21), (\neg X_1 \wedge \neg X_2, 33)\},$$

em que $\mathcal{P}_3$ é um refinamento de $\mathcal{P}_1$ e $\mathcal{P}_2$.

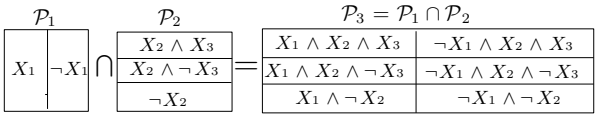


Figura 4.1: *Refinamento de partições $\mathcal{P}_1$ com 2 blocos e $\mathcal{P}_2$, com 3 blocos. O resultado do refinamento é dado por $\mathcal{P}_1 \cap \mathcal{P}_2$, uma partição com 6 blocos.*

Dadas m partições $\mathcal{P}_1$, ..., $\mathcal{P}_m$, é possível obter um refinamento para $q \leq m$ partições computando a intersecção delas, denotada por $\mathcal{P}_{q+1} = \bigcap_{i=1}^{q} \mathcal{P}_i$ (Givan *et al.*, 2003).

A Figura 4.2 apresenta uma sequência de refinamentos em que no início é dada uma partição com um único bloco. Após um primeiro refinamento, a partição de origem é dividida em 2 blocos. Depois disso, nos dois refinamentos seguintes são obtidas respectivamente partições com 4 e 5 blocos. Enfim, após uma nova etapa de refinamento, a partição não se altera. Consequentemente, não é mais necessário dividir os blocos. Desse exemplo, seria extraído um modelo M' em que $|S'| = 5$.

Dado um MDP fatorado, é possível usar suas funções de recompensa e transição para identificar blocos de estados com a mesma recompensa ou a mesma probabilidade de transição gerando

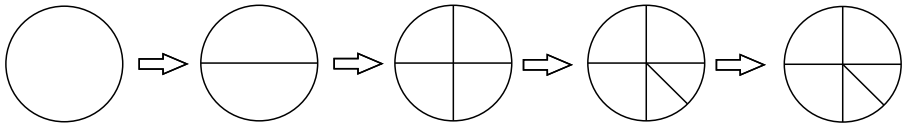
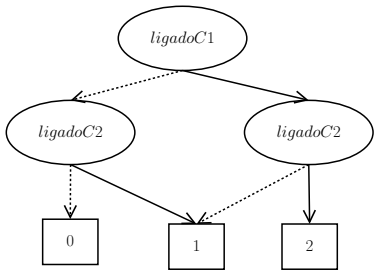


Figura 4.2: *Exemplo de sequência de refinamentos que se encerra com um partição $\mathcal{P}$ que contém 5 blocos. Note que sucessivos refinamentos implicam em partições com um número maior ou igual de blocos.*

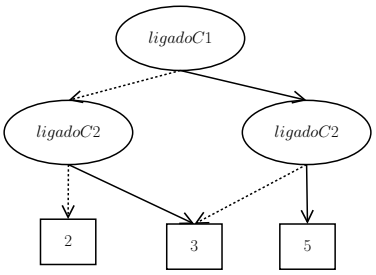
diferentes partições. Os algoritmos de agregação de estados de um MDP geralmente começam com a partição mais grossa que inclui todos os estados em um único bloco e em seguida, refinam essa partição com base em outras partições do MDP. Por exemplo, considere o domínio *SysAdmin*, no qual um agente deve gerenciar uma rede com n computadores de forma que o máximo de computadores fiquem ligados. Considere uma instância com 2 computadores: $C1$ e $C2$. Essa instância tem 2 variáveis de estado: ligadoC1 e ligadoC2; e 3 ações: noop (não reiniciar os computadores), reiniciarC1 e reiniciarC2. A função recompensa para a ação noop pode ser visualizada na Figura 4.3(a). Por exemplo, quando a ação noop é executada no estado em que ligadoC1 é verdadeiro e ligadoC2 é falso, a recompensa é 1. Para se referir às partições de um MDP, é utilizada uma notação especial: para cada ação $a \in A$, temos uma partição $\mathcal{P}_R^a$ com relação à função recompensa; e para cada par de ação $a \in A$ e variável de estado $X_i \in X$ que pode ser alterada por a, há uma partição $\mathcal{P}_{X_i}^a$ com relação à função de transição probabilística em sua forma fatorada (Givan *et al.*, 2003).

ligadoC1	ligadoC2	recompensa
0	0	0
0	1	1
1	0	1
1	1	2

(a) Tabela da função recompensa para noop



(b) ADD da função recompensa



(c) ADD de partição baseada na função recompensa

Figura 4.3: *a) Função recompensa para a ação noop em uma instância do domínio SysAdmin com 2 computadores. b) A mesma função recompensa representada por um ADD. c) Partição obtida com base na função recompensa representada por um ADD em que as folhas são números primos.*

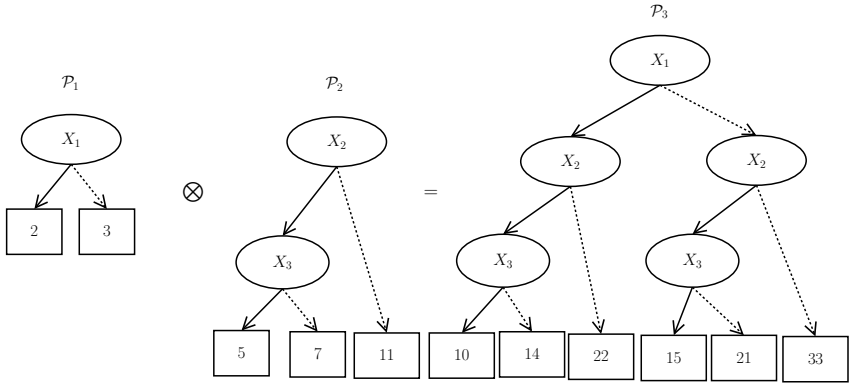


Figura 4.4: *Refinamento das partições $\mathcal{P}_1$ e $\mathcal{P}_2$ da Figura 4.1, computado como o produto entre ADDs.*

A partição obtida da função recompensa na Figura 4.3(a) é implicitamente representada como $\mathcal{P}_R^{noop} = \{(B_1, v_1), (B_2, v_2), (B_3, v_3)\}$, em que: $B_1 = \{ligadoC1 \wedge ligadoC2\}$, com recompensa 2;
$B_2 = \{(\neg ligadoC1 \wedge ligadoC2) \vee (ligadoC1 \wedge \neg ligadoC2)\}$, com recompensa 1; e
$B_3 = \{\neg ligadoC1 \wedge \neg ligadoC2\}$, com recompensa 0.

Podemos observar que, dado um MDP conforme a Definição 1, o número máximo de diferentes partições é $|A| + (|A| \times |X|)$, sendo que $|A|$ partições são obtidas das diferentes funções de recompensa considerando cada ação $a \in A$ e $|A| \times |X|$ são obtidas das partições baseadas na transição probabilística, ou seja, uma para cada variável $X_i \in X$ que pode ser alterada por cada ação $a \in A$. Com isso, as partições obtidas para o exemplo *SysAdmin*, considerando as diferentes funções são: $\mathcal{P}_R^{noop}$, $\mathcal{P}_R^{ligadoC1}$ e $\mathcal{P}_R^{ligadoC2}$ baseadas nas funções de recompensa e; $\mathcal{P}_{ligadoC1}^{noop}$, $\mathcal{P}_{ligadoC2}^{noop}$, $\mathcal{P}_{ligadoC1}^{reiniciarC1}$, $\mathcal{P}_{ligadoC2}^{reiniciarC1}$, $\mathcal{P}_{ligadoC1}^{reiniciarC2}$ e $\mathcal{P}_{ligadoC2}^{reiniciarC2}$ baseadas nas funções de transição probabilística.

4.2 Partições representadas por ADDs

Uma partição pode ser representada usando um ADD em que cada folha representa um rótulo único v_i. A expressão DNF que caracteriza um bloco de estados B_i é dada pela disjunção de conjunções obtidas de diferentes caminhos que vão da raiz do ADD até a folha rotulada com v_i. Por exemplo, a Figura 4.5 mostra um ADD que representa a seguinte partição $\mathcal{P} = \{(X_1 \wedge X_2, 2), ((X_1 \wedge \neg X_2) \vee (\neg X_1 \wedge \neg X_2), 3), (\neg X_1 \wedge X_2, 5)\}$.

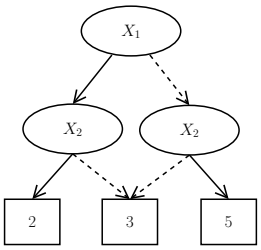


Figura 4.5: *Uma partição $\mathcal{P}$ do conjunto S em que os estados e blocos são representados pelo conjunto de variáveis de estado $X = \{X_1, X_2\}$ sendo $\mathcal{P} = \{(X_1 \wedge X_2, 2), ((X_1 \wedge \neg X_2) \vee (\neg X_1 \wedge \neg X_2), 3), (\neg X_1 \wedge X_2, 5)\}$.*

Para computar o refinamento de q partições representadas como ADDs, é necessário calcular o produto dos ADDs que representam essas partições (que é o mesmo que computar a intersecção das partições) (Givan *et al.*, 2003). Nesse caso, é necessário definir rótulos únicos para os blocos que ainda sejam únicos depois que o produto é calculado. Com base nisso, os blocos podem ser rotulados com números primos. A Figura 4.3(b) ilustra o ADD que representa a função recompensa para a ação noop no domínio *SysAdmin*, que corresponde à representação tabular na Figura 4.3(a). É possível gerar uma partição baseada neste ADD criando uma cópia do ADD e alterando os valores das folhas para números primos distintos (Figura 4.3(c)).

A Figura 4.4 mostra o refinamento das partições $\mathcal{P}_1$ e $\mathcal{P}_2$ mostrado na Figura 4.1.

Para se referir a partições representadas como ADDs, adotaremos a seguinte notação: $\mathcal{P}_R^a$ se torna $\mathcal{P}_{DD}^{a,R}$ e $\mathcal{P}_{X_i}^a$ se torna $\mathcal{P}_{DD}^{a,X_i}$.

4.3 Bissimulação estocástica exata - Definições

Os algoritmos que são apresentados neste capítulo recebem como entrada um MDP (enumerativo ou fatorado) e computam um MDP enumerativo de tamanho reduzido que pode ser resolvido em menos tempo do que o MDP original (Dean e Givan, 1997). Esses algoritmos geram partições do conjunto de estados de um MDP com o intuito de encontrar uma *bissimulação estocástica exata* (Givan *et al.*, 2003), uma partição na qual estados em um mesmo bloco tem o mesmo comportamento sob qualquer ação do MDP, i.e., as mesmas recompensas e as mesmas probabilidades de transição entre blocos da partição. Com isso, esses estados possuem o mesmo comportamento independente da ação executada e a partição pode ser vista como uma relação de equivalência no conjunto S (Dean e Givan, 1997).

Definição 5. *Seja $\mathcal{P}$ uma partição de S. Dizemos que $\mathcal{P}$ é uma* ***partição uniforme com relação à função recompensa*** *(Dean e Givan, 1997) se, para cada $B_i \in \mathcal{P}$, para cada par $s, s' \in B_i$ e para cada ação $a \in A$:*

$$R(s,a) = R(s',a). \tag{4.1}$$

Definição 6. *Considere cada par de estados s, s' que pertencem a um mesmo bloco $B_j \in \mathcal{P}$. Se os estados nesse bloco têm a mesma probabilidade de alcançar um bloco $B_w \in \mathcal{P}$, sob cada ação $a \in A$ que possa ser aplicada em s e s', então dizemos que o* ***bloco*** *$B_j \in \mathcal{P}$* ***é estável com relação ao bloco*** *$B_w \in \mathcal{P}$ (Dean e Givan, 1997). Formalmente, isso é descrito como:*

$$\forall a \in A, \sum_{s'' \in B_w} P(s''|s,a) = \sum_{s'' \in B_w} P(s''|s',a). \tag{4.2}$$

Definição 7. *Um* ***bloco*** *B_j* ***é estável*** *(Dean e Givan, 1997) se é estável com relação a todo bloco $B_w \in \mathcal{P}$.*

Definição 8. ***Uma partição*** *$\mathcal{P}$* ***é homogênea*** *(Dean e Givan, 1997) se $\mathcal{P}$ é uniforme com relação à função recompensa e se todos os blocos em $\mathcal{P}$ são estáveis.*

Definição 9. *Uma partição $\mathcal{P}$ é uma bissimulação estocástica exata se for homogênea (Dean e Givan, 1997).*

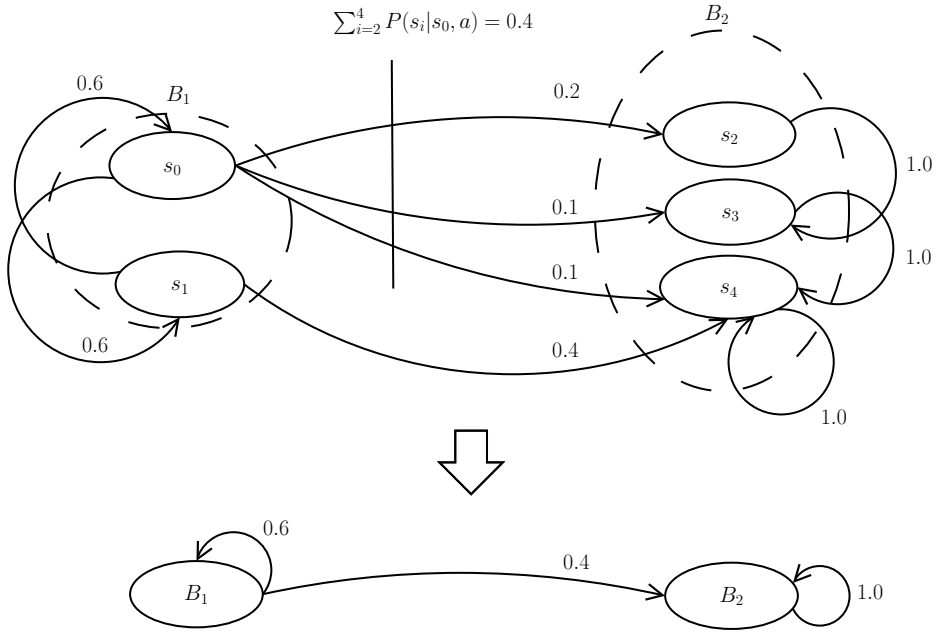


Figura 4.6: *Exemplo de uma partição $\mathcal{P} = \{B_1, B_2\}$ que é homogênea, i.e., uma bissimulação estocástica exata. Dessa partição, é possível reduzir um MDP M com 5 estados para um MDP M' com 2 estados.*

A Figura 4.6 mostra um exemplo de uma partição homogênea de um MDP, i.e., uma bissimulação estocástica. Neste exemplo, o MDP original possui 5 estados, enquanto o MDP reduzido possui apenas 2 estados.

Dada uma partição $\mathcal{P}$ que é uma bissimulação estocástica exata, é possível construir um **MDP enumerativo reduzido** selecionando cada $B_i \in \mathcal{P}$ como um **estado abstrato**. A função recompensa para cada par de estado abstrato B_i e ação a, é obtida escolhendo qualquer estado $s \in B_i$ como segue (Dean e Givan, 1997):

$$R(B_i,a) = R(s,a). \tag{4.3}$$

A função de transição entre um estado abstrato obtido de um bloco B_j e qualquer estado abstrato obtido do bloco B_w para uma ação a é computada considerando um estado $s \in B_j$ e todo estado $s' \in B_w$ da seguinte maneira (Dean e Givan, 1997):

$$P(B_w|B_j, a) = \sum_{s' \in B_w} P(s'|s, a). \tag{4.4}$$

Depois de obter o MDP enumerativo reduzido a partir de uma partição homogênea $\mathcal{P}$, é possível renomeá-lo para $M' = (S', A, P', R', \gamma)$. A vantagem de resolver M' é a possibilidade de fazer menos atualizações de Bellman enquanto se resolve o MDP. Por exemplo, se $s_1, s_5 \in S$ e $B_1 = \{s_1, s_5\} \in S'$, é possível atualizar apenas $V^t(B_1)$ ao invés de $V^t(s_1)$ e $V^t(s_5)$. Além disso, como é constatado a seguir pelo Teorema 1, $\pi^*(s_1) = \pi^*(s_5) = \pi^*(B_1)$.

Teorema 1. *Dado que $\mathcal{P}$ é uma bissimulação estocástica exata de um MDP M e o MDP reduzido M' obtido através de $\mathcal{P}$, uma política ótima para M' também é ótima se for utilizada em M (Givan et al., 2003).*

4.4 Redução e Minimização de Modelos (enumerativos)

Para obter uma bissimulação estocástica exata, é necessário definir uma partição inicial que pode ser a partição baseada na função recompensa. Depois disso, é necessário refinar os blocos dessa partição com o objetivo de torná-los estáveis (Definição 7), i.e., dividir blocos que contém estados que não deveriam estar juntos. O processo termina quando todos os blocos são estáveis. Consequentemente, a partição resultante é uma bissimulação estocástica exata (Dean e Givan, 1997).

Algoritmo 4.4.1: Redução de modelos (M) (Givan *et al.*, 2003)

Entrada: M: um MDP
Saída: $\mathcal{P}$: uma partição homogênea, i.e., uma bissimulação estocástica exata

```
1 P ← ⋂_{a∈A} P_R^a;
2 enquanto existir B_j, B_w ∈ P tal que B_j não é estável com relação a B_w faça
3 |   P_{B_j} ← ⋂_{a∈A} DividirBloco(B_j, B_w, a) ;
4 |   P ← {P − B_j} ∪ P_{B_j};
5 fim
6 retorna P;
```

Algoritmo 4.4.2: Minimização de modelos (M) (Givan *et al.*, 2003)

Entrada: M: um MDP
Saída: $\mathcal{P}$: uma partição homogênea minimal, i.e., uma bissimulação estocástica exata com o menor número de blocos

```
1 P ← ⋂_{a∈A} P_R^a;
2 enquanto existir B_j, B_w ∈ P tal que B_j não é estável com relação a B_w faça
3 |   P_{B_j} ← ⋂_{a∈A} DividirBloco(B_j, B_w, a) ;
4 |   P_{B_j} ← ⋂_{a∈A} JuntarBlocos(P_{B_j}, B_w, a);
5 |   P ← {P − B_j} ∪ P_{B_j};
6 fim
7 retorna P;
```

De forma genérica, a bissimulação estocástica exata pode ser computada com o Algoritmo 4.4.1, chamado de *redução de modelos*. Na Linha 1, o algoritmo computa uma partição uniforme com relação à função recompensa considerando cada ação $a \in A$. Na Linha 3, a operação $DividirBloco(B_j, B_w, a)$ divide um bloco B_j em sub-blocos $\mathcal{P}_{B_j} = \{B'_{j_1}, ..., B'_{j_l}\}$ estáveis com relação ao bloco B_w considerando cada ação e depois faz o refinamento das partições resultantes através da intersecção. Na Linha 4, o bloco B_j é substituído por sub-blocos em $\mathcal{P}_{B_j}$ que são estáveis

com relação ao bloco B_w. O tipo de refinamento realizado pelo Algoritmo 4.4.1 nas Linhas 3 e 4 é conhecido como *refinamento não-ótimo* (ou *adequado*).

Uma outra forma de computar uma bissimulação estocástica exata é com o Algoritmo 4.4.2, chamado de *minimização de modelos*. A única diferença entre os Algoritmos 4.4.1 e 4.4.2 é que o Algoritmo 4.4.2 utiliza a operação $JuntarBlocos(\mathcal{P}_{B_j}, B_w, a)$, que permite que cada sub-bloco de $\mathcal{P}_{B_j}$ gerado na Linha 3 seja maximal, i.e., agrupe o máximo de estados gerando o menor número de blocos. A Linha 4 da minimização de modelos possibilita o *refinamento ótimo*, i.e., permite gerar um MDP minimal equivalente ao MDP original (Dean *et al.*, 1997) (existe um compromisso entre armazenamento e tempo, porque a partição pode se tornar menor, mas a computação de $\mathcal{P}$ se torna mais lenta). No pior caso, os Algoritmos 4.4.1 e 4.4.2 devolvem um MDP do mesmo tamanho que o MDP original.

Os detalhes de como implementar refinamentos não-ótimos e ótimos de forma fatorada são respectivamente descritos nas Seções 4.5 e 4.6.

4.5 Redução de modelos em MDPs fatorados: refinamento não-ótimo fatorado

O algoritmo de redução de modelos com refinamentos fatorados permite encontrar bissimulações estocásticas exatas usando MDPs fatorados e conceitos de partições de MDPs. Isso é feito pela operação SSPLIT (*Structure-based Split*) (Givan *et al.*, 2003), que refina um bloco $B_j \in \mathcal{P}$ com relação a um bloco $B_w \in \mathcal{P}$ com o intuito de gerar sub-blocos não-maximais estáveis com relação a B_w, i.e., SSPLIT é um refinamento não-ótimo. A operação SSPLIT recebe um bloco B_j, um bloco B_w e devolve uma partição $\mathcal{P}'$ que é um refinamento de $\mathcal{P}$ na qual B_j é substituído por $\mathcal{P}_{B_j}$ cujos sub-blocos são estáveis com relação ao bloco B_w. SSPLIT é computada como segue:

$$SSPLIT(B_j, B_w, \mathcal{P}) = (\mathcal{P} - \{B_j\}) \cup (\bigcap_{a \in A} DividirBloco(B_j, B_w, a)), \tag{4.5}$$

em que a versão fatorada da operação $DividirBloco(B_j, B_w, a)$ é dada por $B_j \cap (\bigcap_{X_i \in vars(B_w)} \mathcal{P}^a_{X_i})$, que representa uma divisão de bloco considerando uma única ação e $vars(B_w)$ são as variáveis de estado usadas para representar o bloco B_w (Givan *et al.*, 2003). A operação $DividirBloco(B_j, B_w, a)$ constrói partições $\mathcal{P}^a_{X_i}$, uma para cada variável que $X_i \in vars(B_w)$. Cada partição $\mathcal{P}^a_{X_i}$ agrupa os estados que têm a mesma probabilidade de alterar X_i usando a ação a. Contudo, para garantir que os blocos em $\mathcal{P}_{B_j}$ são estáveis com relação ao bloco B_w, é necessário considerar cada X_i mencionado em B_w. Isso é feito com a intersecção computada na operação $DividirBloco(B_j, B_w, a)$.

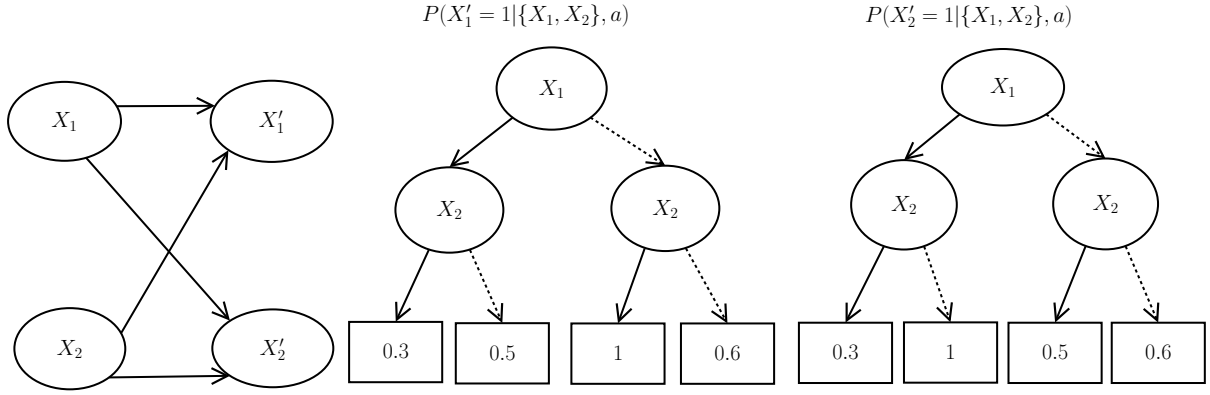
Com SSPLIT, o algoritmo de redução de modelos com refinamentos fatorados (*Model Reduction with Factored Splits - MRFS*) (Givan *et al.*, 2003) não enumera todos os estados explicitamente enquanto refina blocos porque os refinamentos são feitos usando apenas as variáveis de estado em $vars(B_w)$ para refinar um bloco B_j com relação a um bloco B_w (Givan *et al.*, 2003). Isso é possível porque, se observarmos a DBN (Seção 2.3), algumas variáveis independem de outras. O algoritmo MRFS corresponde ao Algoritmo 4.4.1 substituindo as linhas 3 e 4 pela Equação 4.5.

MRFS computa uma bissimulação estocástica exata considerando apenas as partições, mas ignorando probabilidades conjuntas entre blocos. Se as probabilidades conjuntas entre blocos são consideradas, alguns blocos podem ser juntados *(coarsening)* quando houver igualdades, é o que faz o algoritmo de minimização de modelos (Algoritmo 4.4.2 (Givan *et al.*, 2003)).

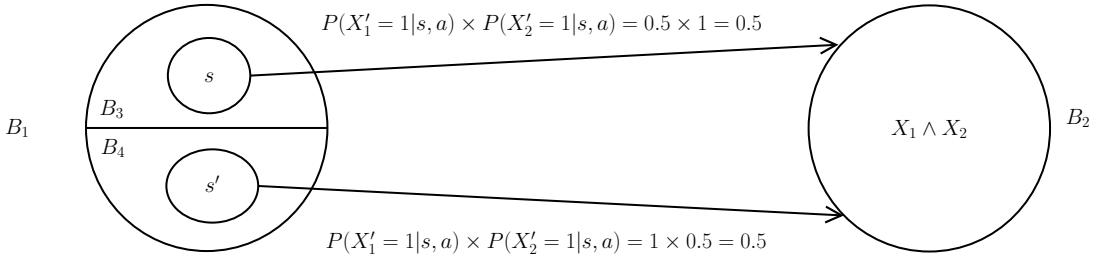
4.6 Minimização de modelos em MDPs fatorados: refinamento ótimo fatorado

Os refinamentos realizados pela operação SSPLIT descritos na seção anterior são não-ótimos, i.e., não garantem a geração de partições com o menor número de blocos. Para obter partições com o menor número de blocos possível, mas que ainda possam ser consideradas homogêneas, é

necessário realizar a *junção dos blocos (coarsening)* que foram divididos de forma precipitada por SSPLIT. O algoritmo 4.6.1 é chamado de minimização de modelos com refinamentos fatorados (*Model Minimization with Factored Splits - MMFS*) (Givan *et al.*, 2003). Considere o exemplo da Figura 4.7(a) e suponha que os estados s e s' da Figura 4.7(b) pertenciam a um mesmo bloco B_1. Ao realizar o refinamento de B_1 com relação ao bloco B_2 utilizando a operação SSPLIT, obteríamos os blocos B_3 e B_4 no lugar de B_1. Observe que os estados s e s' foram separados devido às partições obtidas através das funções $P(X_1' = 1|\{X_1, X_2\}, a)$ e $P(X_2' = 1|\{X_1, X_2\}, a)$. Contudo, as probabilidades conjuntas $P(B_2|s, a)$ e $P(B_2|s', a)$ são iguais. Com isso, o bloco B_1 é estável com relação ao bloco B_2 e a ação a. Portanto, s e s' podem ser agrupados novamente.



(a) DBN de uma ação a e seus ADDs da função de transição.



(b) Exemplo de divisão e junção de blocos. Considere que $s = X_1 \wedge \neg X_2$ e $s' = \neg X_1 \wedge X_2$.

Figura 4.7: *(a) DBN de uma ação a e seus ADDs da função de transição probabilística. (b) Exemplo em que dois estados s e s′ estavam agrupados em um bloco B_1, mas foram separados através da operação SSPLIT que gerou os blocos B_3 e B_4. Contudo, s e s′ podem ser reagrupados, gerando novamente B_1 porque ambos tem a mesma probabilidade conjunta de alcançar o bloco B_2.*

Os refinamentos obtidos pela combinação da operação SSPLIT com uma função para juntar blocos são chamados de **refinamentos ótimos**. Quando os estados são representados com expressões DNF, a junção dos blocos se dá por meio da união (disjunção) das expressões de cada bloco utilizado na junção. No exemplo dado, $B_1 = \{(X_1 \wedge \neg X_2) \vee (\neg X_1 \wedge X_2)\}$.

Os experimentos realizados neste trabalho mostram que na maior parte dos casos não é vantajoso juntar blocos que foram divididos com os refinamentos fatorados porque os modelos minimais geralmente já são encontrados pela redução de modelos.

4.7 Computação eficiente da minimização de modelos

Uma variação do Algoritmo 4.4.2 foi proposta com o objetivo de tornar o refinamento dos blocos mais eficiente (Guo e Leong, 2010). Para isso, é necessário gerar partições de cada ação localmente, o que implica em um número menor de refinamentos.

Na estrutura dos MDPs fatorados, podem existir variáveis de estado que não interferem nas probabilidades de transição. Uma forma de identificar as *variáveis de estado essenciais* X_e, i.e., que interferem probabilisticamente nas transições do MDP, é verificar recursivamente que variáveis de

Algoritmo 4.6.1: $JuntarBlocos(\mathcal{P}_{B_j}, B_w, a)$ (Givan *et al.*, 2003)

Entrada: $\mathcal{P}_{B'_j}$: uma partição de B_j em que cada bloco $B'_j \in \mathcal{P}_{B_j}$ é estável com relação ao bloco B_w; B_w: um bloco de $\mathcal{P}$; e a: a ação atual.
Saída: $\mathcal{P}_{B_j}$: uma partição de B_j em que cada sub-bloco é estável com relação ao bloco B_w considerando a ação a. Além disso, $\mathcal{P}_{B_j}$ é minimal.

enquanto *houver algum par de blocos* $B'_{j_1}, B'_{j_2} \in \mathcal{P}_{B_j}$ *t.q.* $P(B_w|B'_{j_1}, a) = P(B_w|B'_{j_2}, a)$ **faça**
  $\mathcal{P}_{B_j} \leftarrow \mathcal{P}_{B_j} - B'_{j_1} - B'_{j_2}$;
  $\mathcal{P}_{B_j} \leftarrow \mathcal{P}_{B_j} \cup (B'_{j_1} \cup B'_{j_2})$;
fim
retorna $\mathcal{P}_{B_j}$;

estado aparecem em $\mathcal{P}_R^A = \bigcap_{a \in A} \mathcal{P}_R^a$, e determinar o conjunto de variáveis que são pais dessas variáveis na partição por recompensa através das DBNs (Guo e Leong, 2010). Considerando o MDP das Figuras 2.3(a) e 2.4 (Dean e Givan, 1997), sem conhecer os blocos intermediários gerados durante a minimização de modelos, é possível perceber que X_1 é a única variável da qual a função recompensa depende e portanto $X_e = \{X_1\}$. O próximo passo, é determinar recursivamente que variáveis de estado podem ser acrescentadas ao conjunto $X_e = \{X_1\}$, verificamos que variáveis são pais de X'_1, e neste caso $pais(X'_1) = \{X_1, X_2\}$. Após adicionar X_2 ao conjunto X_e, é possível perceber pela DBN que X'_2 não tem pais e portanto, $X_e = \{X_1, X_2\}$.

O Algoritmo 4.7.1 é uma versão melhorada do Algoritmo 4.4.1 que usa os conceitos de computação de partições locais para cada ação e em seguida, encontra uma partição que é um refinamento de cada uma das partições locais. Com os refinamentos locais, ainda é possível realizar a junção de blocos considerando cada ação apenas uma vez no Algoritmo 4.7.2. Depois de gerar a partição localmente, ela é propagada para a partição principal com a operação de refinamento. O resultado final ainda é uma bissimulação estocástica, mas com menos computações redundantes (Guo e Leong, 2010).

Algoritmo 4.7.1: Redução de modelos melhorada (M) (Guo e Leong, 2010)

Entrada: M: um MDP
Saída: $\mathcal{P}$: uma partição homogênea, i.e., uma bissimulação estocástica exata

$\mathcal{P} \leftarrow \bigcap_{a \in A} \mathcal{P}_R^a$;
repita
  para cada $a \in A$ **faça**
    $\mathcal{P}_{X_e}^a \leftarrow \bigcap_{X_i \in X_e} \mathcal{P}_{X_i}^a$;
    $\mathcal{P} \leftarrow \mathcal{P} \cap \mathcal{P}_{X_e}^a$;
  fim
até *até* $\mathcal{P}$ *não mudar*;
retorna $\mathcal{P}$;

A computação de MRFS pode ser realizada de forma ainda mais eficiente usando ADDs (Kim e Dean, 2002). Desta forma, é necessário gerar uma partição de recompensa para cada ação e transição probabilística com base nas variáveis de estado essenciais do MDP. Depois disso, basta computar o refinamento delas, ou seja, produto entre os ADDs ($\otimes$) dessas partições (i.e., uma bissimulação estocástica exata (Guo e Leong, 2010)) como é apresentado pela equação a seguir:

Algoritmo 4.7.2: Minimização de modelos melhorada (M) (Guo e Leong, 2010)

Entrada: M: um MDP
Saída: $\mathcal{P}$: uma partição homogênea minimal, i.e., uma bissimulação estocástica exata com o menor número de blocos possível.

$\mathcal{P} \leftarrow \bigcap_{a \in A} \mathcal{P}_R^a$;
repita
  para cada $a \in A$ **faça**
    $\mathcal{P}_{X_e}^{a_{temp}} \leftarrow \bigcap_{X_i \in X_e} \mathcal{P}_{X_i}^a$;
    $\mathcal{P}_{X_e}^a \leftarrow \bigcap_{B_w \in \mathcal{P}_{X_e}^{a_{temp}}} JuntarBlocos(\mathcal{P}_{X_e}^{a_{temp}}, B_w, a)$;
    $\mathcal{P} \leftarrow \mathcal{P} \cap \mathcal{P}_{X_e}^a$;
  fim
até *até* $\mathcal{P}$ *não mudar*;
retorna $\mathcal{P}$;

$$\mathcal{P}_{DD} = \mathcal{P}_{DD}^{A,R} \otimes \mathcal{P}_{DD}^{A,X_e}, \tag{4.6}$$

$$\text{em que } \mathcal{P}_{DD}^{A,R} = \bigotimes_{a \in A} \mathcal{P}_{DD}^{a,R} \text{ e}$$
$$\mathcal{P}_{DD}^{A,X_e} = \bigotimes_{a \in A} (\bigotimes_{X_i \in X_e} \mathcal{P}_{DD}^{a,X_i}).$$

O Algoritmo 4.7.3 é uma versão do Algoritmo 4.7.1 que usa ADDs para representar as partições. De forma similar, acrescentando uma chamada à função para juntar blocos, podemos obter uma versão do MMFS (Algoritmo 4.7.2) com ADDs.

Algoritmo 4.7.3: Reduçao de modelos melhorada com ADDs (M, X_e) (Guo e Leong, 2010)

Entrada: M: um MDP, X_e: conjunto de variáveis de estado essenciais
Saída: $\mathcal{P}$: uma partição homogênea, i.e., uma bissimulação estocástica exata

$\mathcal{P}_{DD} \leftarrow \bigotimes_{a \in A} \mathcal{P}_{DD}^{a,R}$;
repita
  para cada $a \in A$ **faça**
    $\mathcal{P}_{DD}^{a,X_e} \leftarrow \bigotimes_{X_i \in X_e} \mathcal{P}_{DD}^{a,X_i}$;
    $\mathcal{P}_{DD} \leftarrow \mathcal{P}_{DD} \otimes \mathcal{P}_{DD}^{a,X_e}$;
  fim
até *até* $\mathcal{P}_{DD}$ *não mudar*;
retorna $\mathcal{P}_{DD}$;

4.8 Exemplo de redução e minimização de modelos com refinamentos fatorados

Considere o exemplo das Figuras 2.3(a) e 2.4, em que é dada a DBN de uma ação a, o ADD da função recompensa e os ADDs da função de transição probabilística para cada variável de estado. Observe que os ADDs da função de transição probabilística consideram apenas $X'_j = 1, \forall j \in \{1, 2, 3\}$. Isso acontece porque no diagrama dual, o ADD de $X'_j = 0$ é igual ao ADD de $X'_j = 0$, exceto pelos valores nas folhas. Por esse motivo, os dois lados geram partições iguais.

Seja $\mathcal{P}^a_R = \{X_1, \neg X_1\}$ a partição baseada na função recompensa (Figura 4.8(a)), que neste caso não é uma bissimulação estocástica exata. Apesar disso, dada uma partição inicial de S, é possível obter uma bissimulação estocástica exata aplicando o algoritmo de minimização (ou redução) de modelos que utiliza a operação SSPLIT (Dean e Givan, 1997).

Supondo que temos a DBN (Figura 2.3(a)) e a partição $\mathcal{P}^a_R$ (Figura 4.8(a)), a minimização de modelos (Algoritmo 4.4.2) seleciona um bloco em $\mathcal{P}^a_R$ para verificar a estabilidade com relação a todos os blocos de $\mathcal{P}^a_R$. Suponha que foi escolhido o bloco $B_1 = \{X_1\}$, para obter sub-blocos estáveis com relação ao próprio bloco B_1. Para isso, o algoritmo verifica quais variáveis de estado descrevem o bloco com o qual é necessário obter estabilidade. Nesse caso, o algoritmo obtém que o conjunto de variáveis que descreve B_1 contém apenas a variável X_1. Com base nisso, é necessário calcular a intersecção de B_1 com $\mathcal{P}^a_{X_1}$ para que B_1 seja estável com relação ao próprio B_1, i.e., para que qualquer par de estados $s, s' \in B_1$, a probabilidade de alcançar o próprio B_1 seja a mesma. A computação desses sub-blocos de B_1, que chamaremos de $\mathcal{P}_{B_1}$, é realizada da seguinte maneira:
$\mathcal{P}_{B_1} = B_1 \cap \mathcal{P}^a_{X_1} = \{X_1\} \cap \{X_1, \neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\} = \{X_1\}$.

Como pode ser visto, o bloco $\mathcal{P}_{B_1}$ é igual ao bloco B_1 (o que elimina a necessidade de junção). Em seguida, o algoritmo precisa verificar a estabilidade de B_1 com relação ao bloco B_2, o resultado é o mesmo porque B_2 também é descrito apenas pela negação da variável X_1. Em seguida, dividindo B_2 com relação ao próprio bloco B_2, a partição é refinada porque a intersecção passa a depender da utilização de outras duas variáveis de estado que não eram utilizadas (ADDs da Figura 2.4). Logo, temos:

$\mathcal{P}_{B_2} = B_2 \cap \mathcal{P}^a_{X_1} = \{\neg X_1\} \cap \{X_1, \neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\} = \{\neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\}$.

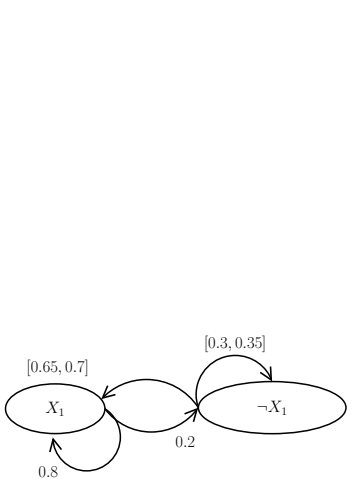
Após a obtenção de $\mathcal{P}_{B_2}$, a minimização de modelos tenta juntar os blocos descritos por $\neg X_1 \wedge X_2$ e $\neg X_1 \wedge \neg X_2$. Mas isso não é possível por que as probabilidades desses blocos alcançarem os blocos antes da divisão diferem.

Depois disso, é realizado o teste de estabilidade de $\mathcal{P}_{B_2}$ com relação ao bloco B_1, mas a partição $\mathcal{P}_{B_2}$ mantém-se a mesma. Como $\mathcal{P}_{B_2}$ inclui em sua descrição a variável de estado X_2, o SSPLIT deve considerar essa variável e a partição $P^a_{X_2}$ na iteração seguinte do algoritmo. Contudo, $P^a_{X_2}$ não inclui nenhuma variável de estado diferente na descrição da partição atual porque X'_2 não depende de nenhuma variável conforme pode ser visto na DBN (Figura 2.3(a)). Observe também que a variável de estado X_3 não aparece na descrição de nenhum bloco e quando for verificada estabilidade entre blocos, X_3 não é mencionada. Por esse motivo, não é necessário considerar a variável de estado seguinte X'_3 porque ela não influencia as transições de nenhum par de blocos e consequentemente, podemos ignorar a partição $\mathcal{P}^a_{X_3}$.

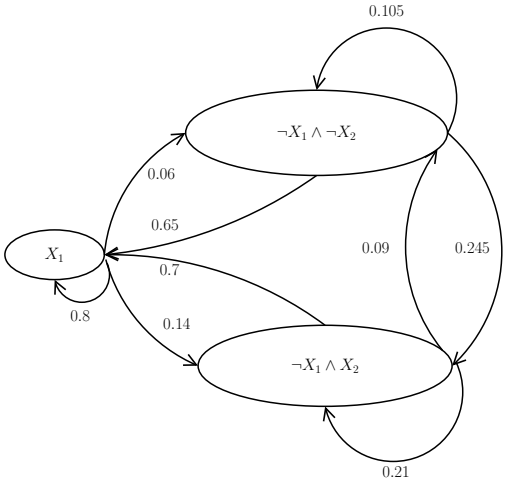
Após cada divisão de blocos com SSPLIT, o algoritmo de minimização realiza a junção de blocos B_{j_1} e B_{j_2} que têm a mesma probabilidade conjunta de alcançar um bloco B_w. Contudo, neste exemplo isso não acontece porque não aparecem coincidências como ocorre no exemplo da Figura 4.7. Com isso, a partição resultante $\mathcal{P}$, que é uma bissimulação estocástica exata, é dada por (Figura 4.8(b)):

$\mathcal{P} = \{X_1, \neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\}$.

Uma outra forma de obter a mesma partição é usando o Algoritmo 4.7.2. A diferença é que nesse outro algoritmo, X_e deve ser calculado antes e as partições são calculadas localmente para cada ação. Depois disso, é computada a intersecção de todas as partições usadas (que também ignoram $\mathcal{P}^a_{X_3}$).



(a) Partição uniforme com relação à função recompensa, i.e., $\mathcal{P}_R^A$.



(b) Partição homogênea.

Figura 4.8: *Partições baseadas no MDP definido com as Figuras 2.3(a) e 2.4 (Givan et al., 2000)*

4.9 Resumo do Capítulo

Nesse capítulo foi apresentado o seguinte conteúdo:

- as partições e como elas podem ser usadas para encontrar bissimulações estocásticas exatas, que são relações de equivalência sobre o conjunto de estados do MDP;
- como representar partições usando ADDs; e
- diferentes algoritmos, baseados em partições enumerativas e fatoradas, para encontrar bissimulações estocásticas exatas.

Capítulo 5

Bissimulação estocástica aproximada

Em uma *bissimulação estocástica aproximada*, é possível agrupar estados mesmo que eles não sejam equivalentes. Desta forma, os estados que têm transições aproximadamente iguais ao executar cada uma das ações também podem ser agrupados se forem considerados intervalos de valores para as funções de recompensa e transição probabilística.

As vantagens de se computar uma bissimulação estocástica aproximada são: (1) o modelo reduzido pode ser menor do que os MDPs obtidos a partir da bissimulação estocástica exata; e (2) é mais fácil de computar do que partições homogêneas que geram MDPs minimais. Contudo, ao encontrar uma bissimulação estocástica aproximada, a política ótima do modelo reduzido pode não ser ótima no modelo original, uma vez o modelo reduzido é um BMDP (Givan *et al.*, 2000) (Capítulo 3). Além dessa desvantagem, BMDPs são mais difíceis de resolver do que MDPs.

5.1 Bissimulação estocástica aproximada - Definições

As definições apresentadas para bissimulação estocástica exata (Seção 4.3) são estendidas na bissimulação estocástica aproximada permitindo aproximações com base em uma constante $\epsilon \in [0, 1]$.

Definição 10. *Uma partição $\mathcal{P}$ é* ***ϵ-uniforme com relação à função recompensa*** *(Dean et al., 1997) se para cada $B_i \in \mathcal{P}$, para cada par $s, s' \in B_i$ e para cada ação $a \in A$, temos:*

$$|R(s, a) - R(s', a)| \leq \epsilon. \tag{5.1}$$

Caso $R(s, a) \notin [0, 1]$, pode-se normalizar a função recompensa com (Ferns *et al.*, 2004):

$$R'(s, a) = \frac{R(s, a) - \min_{s \in S, a \in A} R(s, a)}{\max_{s \in S, a \in A} R(s, a) - \min_{s \in S, a \in A} R(s, a)}. \tag{5.2}$$

Definição 11. ***Um bloco $B_j \in \mathcal{P}$ é ϵ-estável com relação a um bloco $B_w \in \mathcal{P}$*** *(Dean et al., 1997), se para todo par de estados $s, s' \in B_j$ e para toda ação $a \in A$, tivermos:*

$$|\sum_{s'' \in B_w} P(s''|s, a) - \sum_{s'' \in B_w} P(s''|s', a)| \leq \epsilon. \tag{5.3}$$

Definição 12. ***Um bloco $B_j \in \mathcal{P}$ é ϵ-estável*** *(Dean et al., 1997) se for ϵ-estável com relação a todos os blocos de $\mathcal{P}$.*

Definição 13. ***Uma partição é ϵ-homogênea*** *(Dean et al., 1997) se houver todos os blocos de $\mathcal{P}$ forem ϵ-estáveis. Para $\epsilon = 0$, tem-se a bissimulação estocástica exata como um caso particular da bissimulação estocástica aproximada.*

Cada bloco de uma partição $\mathcal{P}$ que é ϵ-homogênea pode ser visto como um estado abstrato do BMDP $M'_{\updownarrow}$, que é o modelo reduzido obtido da bissimulação estocástica aproximada. Sejam

B_i, B_j, B_w estados abstratos de $M_{\updownarrow}$. As funções de recompensa e transição probabilística para cada par de estado abstrato e ação no BMDP são definidas da seguinte maneira:

$$R'_{\updownarrow}(B_i, a) = [min_{s \in B_i} R(s, a), max_{s \in B_i} R(s, a)] \tag{5.4}$$

e

$$P'_{\updownarrow}(B_w | B_j, a) = [min_{s \in B_j} \sum_{s'' \in B_w} P(s'' | s, a), max_{s' \in B_j} \sum_{s'' \in B_w} P(s'' | s', a)]. \tag{5.5}$$

A Figura 5.1 apresenta um exemplo em que um bloco B_1 é 0.2-estável com relação ao próprio bloco B_1 e também com relação ao bloco B_2. Supondo que uma partição 0.2-homogênea contém os blocos B_1 e B_2, com a Equação 5.5 é possível obter a função de transição da Figura 5.2 no BMDP.

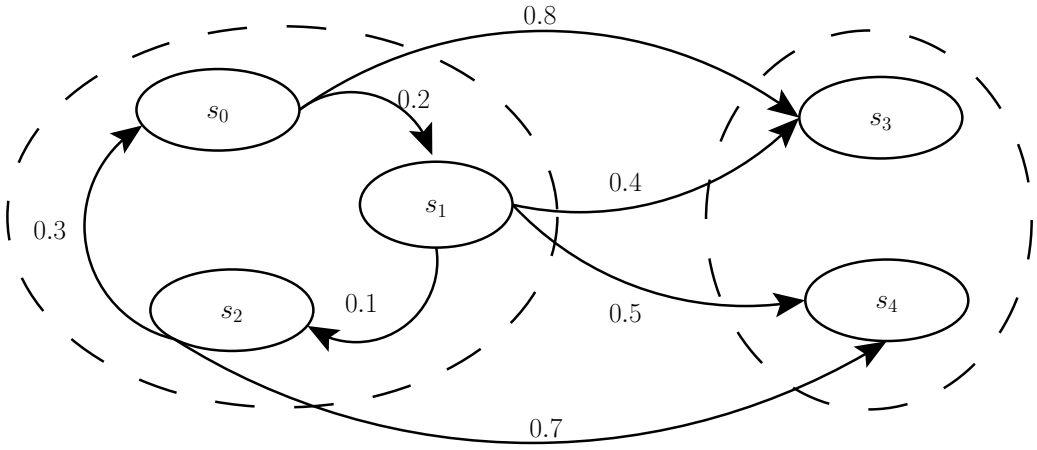


Figura 5.1: *Transições de B_1 para B_2, dois blocos de uma partição ϵ-homogênea ($\epsilon = 0.2$), considerando que há uma única ação $a \in A$ e que a função recompensa é igual para todo $s \in S$.*

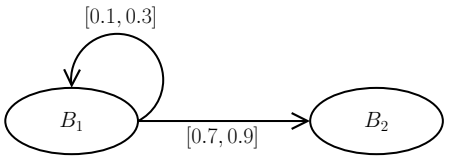


Figura 5.2: *Transições do BMDP extraído a partir do par de blocos da Figura 5.1.*

Os algoritmos para encontrar bissimulações estocásticas aproximadas particionam o conjunto de estados com base na função recompensa e depois refinam a partição até que seja gerada uma partição ϵ-homogênea. Depois de obtido, o BMDP pode ser resolvido com os algoritmos apresentados no Capítulo 3.

5.2 Junção de Blocos Aproximada

Como foi visto no Capitulo 4, a bissimulação estocástica exata pode ser computada através dos algoritmos MRFS e MMFS. A junção de blocos realizada por MMFS considera juntar os blocos que têm probabilidades conjuntas iguais de alcançar um dado bloco B_w. Uma outra forma de realizar essa junção é considerando probabilidades conjuntas aproximadamente iguais de se alcançar B_w, i.e., uma *junção de blocos aproximada*.

Para computar uma junção de blocos aproximada, suponha que são dados sub-blocos em $\mathcal{P}_{B_j}$ estáveis com relação a um dado bloco B_w, e que esses blocos podem ser colocados em ordem crescente de acordo com $P(B_w | B'_j, a)$ dado que $B'_j \in \mathcal{P}_{B_j}$. Depois disso, é possível agrupar de forma gulosa os blocos $B'_j \in \mathcal{P}_{B_j}$ que satisfaçam à Equação 5.3. As Figuras 5.3(a), 5.3(b) e 5.3(c) exemplificam esse processo de junção de blocos de forma aproximada após a utilização de SSPLIT.

Algoritmo 5.1.1: *JuntarBlocosAproximado*$(\mathcal{P}_{B_j}, B_w, a, \epsilon)$ (Dean *et al.*, 1997)

Entrada: $\mathcal{P}_{B_j}$: uma partição de B_j em que cada bloco $B'_j \in \mathcal{P}_{B_j}$ é estável com relação ao bloco B_w; B_w: um bloco de uma partição $\mathcal{P}$; a: a ação atual; e ϵ: uma constante em $[0, 1]$ usada na aproximação.

Saída: $\mathcal{P}_{B_j}$: uma partição de B_j em que cada sub-bloco $B'_j \in \mathcal{P}_{B_j}$ é ϵ-estável com relação ao bloco B_w considerando a ação a.

1 **enquanto** *houver algum par de blocos* $B'_{j1}, B'_{j2} \in \mathcal{P}_{B_j}$ *t.q.* $|P(B_w|B'_{j1}, a) - P(B_w|B'_{j2}, a)| \leq \epsilon$ **faça**
2 $\quad \mathcal{P}_{B_j} \leftarrow \mathcal{P}_{B_j} - B'_{j1} - B'_{j2}$;
3 $\quad \mathcal{P}_{B_j} \leftarrow \mathcal{P}_{B_j} \cup (B'_{j1} \cup B'_{j2})$;
4 **fim**
5 **retorna** $\mathcal{P}_{B_j}$;

Para implementar a ϵ-redução de modelos (ϵ *model reduction with factored splits* - ϵ*MRFS*), é necessário usar o Algoritmo 4.4.2 e substituir a chamada ao algoritmo *JuntarBlocos* (Algoritmo 4.6.1) por uma chamada ao algoritmo *JuntarBlocosAproximado* (Algoritmo 5.1.1).

Quando a ϵ-redução de modelos é computada, não pode ser garantido que a partição obtida será a menor partição ϵ-homogênea. Isso acontece porque as escolhas de quais blocos juntar considerando o valor ϵ são realizadas de forma gulosa (Dean *et al.*, 1997). Além disso, dados dois valores ϵ_1 e ϵ_2 tal que $\epsilon_1 < \epsilon_2$ e ambos estão no intervalo $[0, 1]$, não se pode garantir que as partições obtidas pelo algoritmo usando ϵ_2 terão menos blocos do que as partições obtidas quando for usado ϵ_1. Apesar disso, se $\epsilon_1 = 0$ e $\epsilon_2 \in]0, 1]$, a partição obtida usando ϵ_2 é do mesmo tamanho ou menor do que a partição obtida com $\epsilon_1 = 0$. Obter a menor partição com um dado ϵ é um problema impraticável (Givan *et al.*, 2000; Goldsmith e Sloan, 2000).

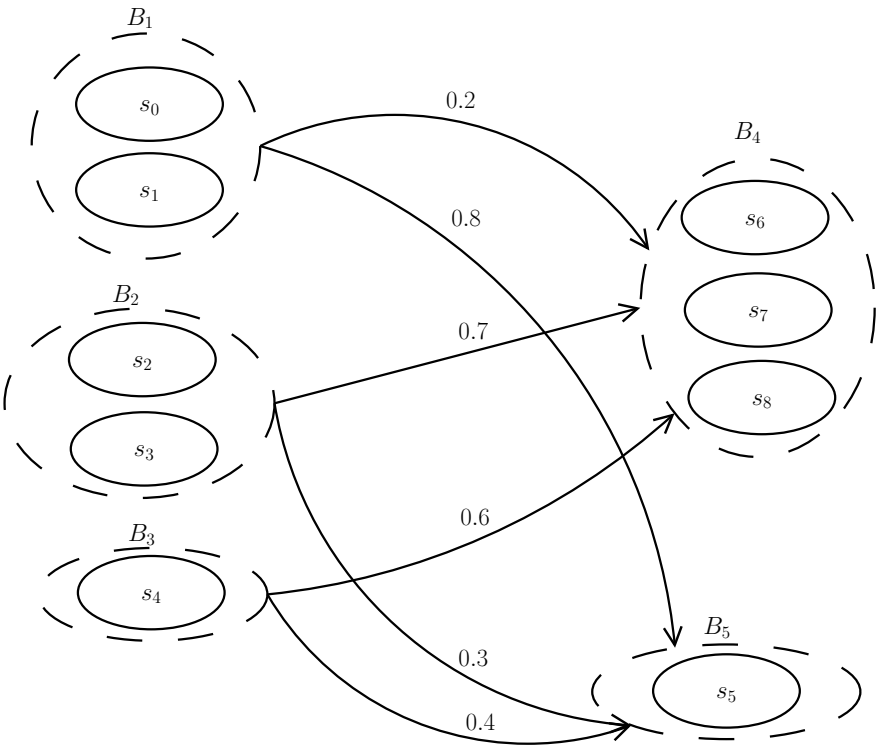
A Figura 5.4 resume o funcionamento dos algoritmos que permitem encontrar bissimulações estocásticas (exatas e aproximadas). Dentre eles: a redução de modelos (Figura 5.4(a)), que apenas refina os blocos; a minimização de modelos (Figura 5.4(b)), que refina os blocos e tenta obter uma partição minimal exata a partir de junções; e a ϵ-redução de modelos (Figura 5.4(c)), que obtém partições aproximadas que na maioria das vezes são menores que a minimal.

Assim como na bissimulação estocástica exata, os algoritmos descritos nesse capítulo também podem ser implementados com operações entre ADDs, o que os torna mais eficientes. Essa foi a versão implementada e avaliada neste trabalho.

5.3 Exemplo com ϵ-Redução de Modelos

Nesta seção, é realizada uma revisão do exemplo da Seção 4.8, mas desta vez com maior foco na utilização de um valor $\epsilon > 0$, neste caso, $\epsilon = 0.05$. Com este exemplo, é apresentado como é possível obter partições ainda menores que a partição homogênea, obtida por MMFS que gera um MDP minimal (Givan *et al.*, 2000).

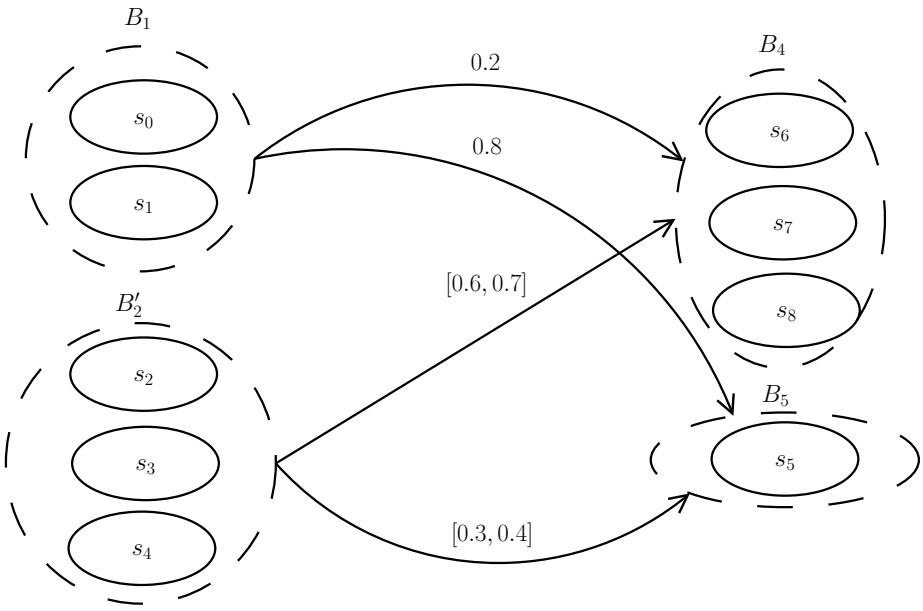
Supondo que temos a partição $\mathcal{P}^a_R$ (Figura 4.8(a)), ϵMRFS seleciona um bloco em $\mathcal{P}^a_R$ para torná-lo ϵ-estável com relação a todos os blocos de $\mathcal{P}^a_R$. Suponha que foi escolhido o bloco $B_1 = \{X_1\}$. O primeiro passo, é a execução da linha 3 do Algoritmo 4.4.2, que permite a obtenção de sub-blocos estáveis com relação ao próprio bloco B_1. Para isso, o algoritmo verifica quais variáveis de estado descrevem o bloco com o qual é necessário obter estabilidade. Nesse caso, o algoritmo obtém que o conjunto de variáveis que descreve B_1 contém apenas a variável X_1. Com base nisso, é necessário calcular a intersecção de B_1 com $\mathcal{P}^a_{X_1}$ para que $\mathcal{P}_{B_1}$ seja estável com relação ao bloco B_1, i.e., para que qualquer par de estados $s, s' \in B'_i \in \mathcal{P}_{B_1}$, a probabilidade de alcançar B_1 seja a mesma com relação à ação a (neste exemplo, $|A| = 1$). A computação desses sub-blocos de B_1, que chamaremos de $\mathcal{P}_{B_1}$, é realizada da seguinte maneira:
$\mathcal{P}_{B_1} = B_1 \cap \mathcal{P}^a_{X_1} = \{X_1\} \cap \{X_1, \neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\} = \{X_1\}$.



(a) Blocos obtidos com SSPLIT em um MDP arbitrário.

Ordenacao de transicoes fixando B_4		Ordenacao de transicoes fixando B_5	
$P(B_4\|B_1,a)=0.2$ $P(B_4\|B_3,a)=0.6$ $P(B_4\|B_2,a)=0.7$	$)\epsilon=0.1$	$P(B_5\|B_2,a)=0.3$ $P(B_5\|B_3,a)=0.4$ $P(B_5\|B_1,a)=0.8$	$)\epsilon=0.1$

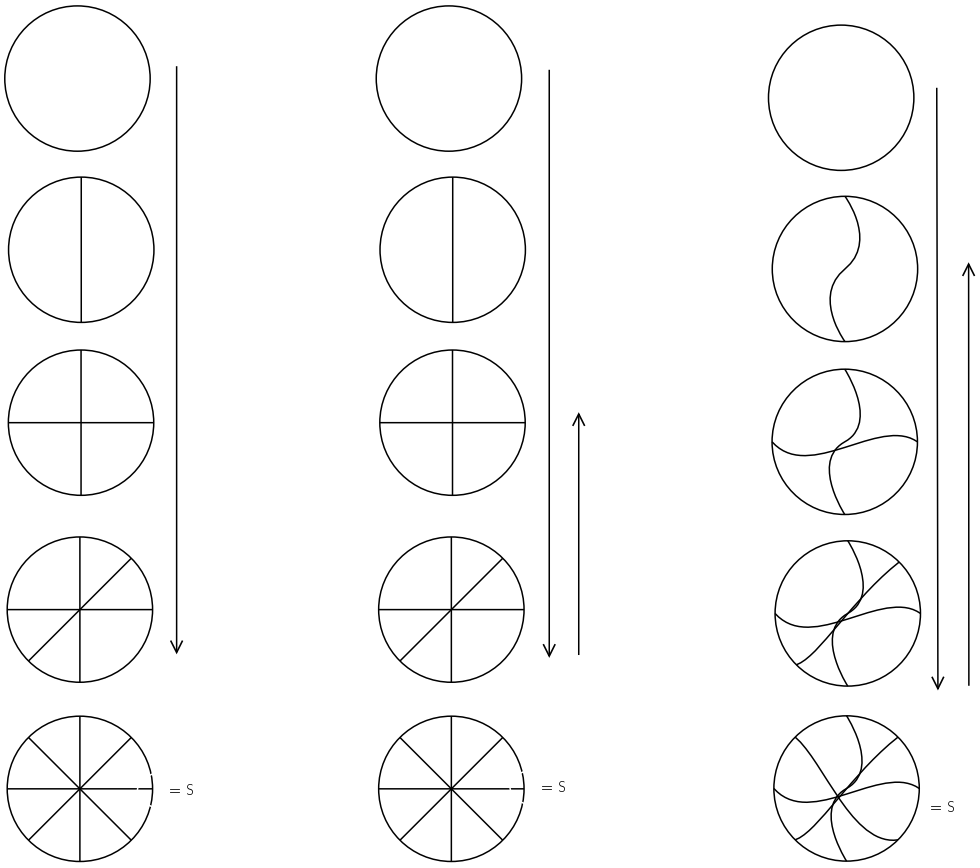
(b) Ordenações de blocos de acordo com as probabilidades de transição.



(c) Blocos juntados de forma gulosa de acordo com as ordenações realizadas.

Figura 5.3: *Exemplo de junção de blocos aproximada em um MDP considerando que* $\epsilon = 0.1$ *e que a recompensa é igual para* B_1, B_2 *e* B_3.

Como pode ser visto, $\mathcal{P}_{B_1}$ é igual ao bloco B_1 e portanto, não é necessário usar a junção de blocos aproximada porque o bloco não foi dividido e não há blocos para juntar. Em seguida, o algoritmo precisa verificar a ϵ-estabilidade de B_1 com relação ao bloco B_2, o resultado é o mesmo



(a) Redução (b) Minimização (c) ϵ-Redução

Figura 5.4: *Exemplo de como refinamentos e junções são realizados na computação de bissimulações estocásticas exatas ou aproximadas através dos algoritmos de redução, minimização e ϵ-redução de modelos. As setas para baixo indicam refinamentos; enquanto as setas para cima indicam junções.*

.

porque B_2 também é descrito apenas pela negação da variável X_1. No passo seguinte, ao dividir B_2 com relação ao próprio bloco B_2, a partição é refinada porque a intersecção passa a depender da utilização de outras duas variáveis de estado que não eram utilizadas (ADDs da Figura 2.4). Logo, temos:

$\mathcal{P}_{B_2} = B_2 \cap \mathcal{P}^a_{X_1} = \{\neg X_1\} \cap \{X_1, \neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\} = \{\neg X_1 \wedge X_2, \neg X_1 \wedge \neg X_2\}$.

Como resultado do último SSPLIT, $\mathcal{P}_{B_2}$ tem dois sub-blocos que chamaremos de $B'_{2_1} = \{\neg X_1 \wedge X_2\}$ e $B'_{2_2} = \{\neg X_1 \wedge \neg X_2\}$. A partir desses dois blocos, é possível selecionar cada um deles e obter ordenações de acordo com a probabilidade de alcançar cada bloco de $\mathcal{P}$, a partição antes do SSPLIT (Figura 4.8(b)). Desta forma, fixando B'_{2_1}, as probabilidades de alcançar cada bloco de $\mathcal{P}$ seriam $P(B_1|B'_{2_1}, a) = 0.7$ e $P(B_2|B'_{2_1}, a) = 0.3$. Por outro lado, as probabilidades de alcançar cada bloco de $\mathcal{P}$ fixando B'_{2_2} seriam $P(B_1|B'_{2_2}, a) = 0.65$ e $P(B_2|B'_{2_2}, a) = 0.35$. Observe que selecionando cada bloco de $\mathcal{P}$, a diferença entre as probabilidades de se alcançar B_1 ou B_2 tomando cada bloco de $\mathcal{P}_{B_2}$ é menor do que ou igual a $\epsilon = 0.05$, i.e., os blocos que foram divididos de forma exata podem ser juntados se o valor $\epsilon = 0.05$ for considerado. Como isso ocorre quando os dois blocos que foram divididos são selecionados e para todos os blocos alcançáveis de $\mathcal{P}$, é possível juntar os blocos que foram divididos obtendo novamente a partição da Figura 4.8(a), que é uma partição ϵ-homogênea (observe que a diferença de $P_\uparrow$ e $P_\downarrow$ é menor ou igual a 0.05 em todas as transições).

Uma outra forma de obter o mesmo resultado é usando o Algoritmo 4.7.2 combinado com o Algoritmo 5.1.1. A diferença é que nesse outro algoritmo, X_e deve ser calculado antes e as partições aproximadas são calculadas localmente para cada ação e considerando a constante ϵ nas junções. Depois disso, é computada a intersecção das partições locais obtidas.

5.4 Resumo do Capítulo

Nesse capítulo foi apresentado o seguinte conteúdo:

- as bissimulações estocásticas aproximadas e como elas podem gerar abstrações ainda menores do MDP original; e
- como obter BMDPs a partir de bissimulações estocásticas aproximadas.

Capítulo 6

Bissimulação estocástica sobre estados alcançáveis

As otimizações propostas neste capítulo são: (1) computação da bissimulação estocástica apenas sobre os estados alcançáveis a partir de um estado inicial s_0 (ao invés de considerar todo o conjunto de estados do problema); e (2) remoção de partições repetidas no algoritmo MRFS (Algoritmo 4.7.1) e em sua variante que é computada sobre os estados alcançáveis a partir de s_0. Em trabalhos anteriores que usavam bissimulações estocásticas e apresentavam resultados empíricos, os maiores MDPs reduzidos tinham 20 variáveis (Kim e Dean, 2002). Com as melhorias deste trabalho, é possível reduzir problemas com um número 7 vezes maior. Além disso, é possível reduzir esses problemas num tempo menor.

6.1 Bissimulações estocásticas sobre estados alcançáveis

Os algoritmos MRFS, MMFS e ϵMRFS podem ser melhorados se usarmos a informação sobre o estado inicial (geralmente conhecido nos problemas de planejamento). A Figura 6.1 apresenta, de forma resumida, três diferentes partições de um conjunto S:

(a) uma partição em que os estados em S são tratados individualmente, i.e., cada bloco contém um único estado (Figura 6.1(a));

(b) uma partição sobre S de forma que os blocos não são necessariamente unitários (Figura 6.1(b)); e

(c) uma partição sobre S em que 1 bloco contém os estados não alcançáveis a partir de s_0 (Figura 6.1(c)).

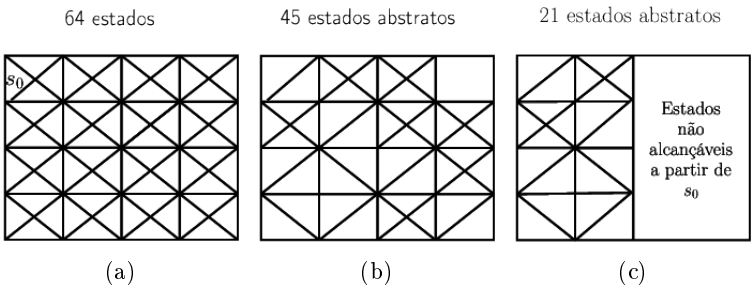


Figura 6.1: *Exemplo de como as partições podem ser realizadas sobre S. O primeiro quadrado, contém 64 triângulos de tamanho unitário que representam cada estado $s \in S$ (com o estado inicial s_0 identificado). O segundo quadrado mostra uma possível partição de S em que alguns estados foram agrupados. O terceiro quadrado mostra uma partição baseada nos estados alcançáveis, ignorando os estados inalcançáveis.*

A ideia de se considerar apenas partições do tipo (c) para cálculo da bissimulação estocástica implica nas seguintes vantagens:

- o modelo reduzido resultante passa a ser menor do que se fossem consideradas partições do tipo (b); e
- a computação da bissimulação estocástica passa a utilizar partições menores, o que torna essa computação mais eficiente.

6.1.1 Análise de Alcançabilidade com BDDs

Se os estados alcançáveis são computados antes de se procurar por uma bissimulação estocástica, é possível obter a ***partição por alcançabilidade*** $\mathcal{P}_{DD}^{S|s_0} = \{(B_{alcancaveis|s_0}, 1), (B_{\neg alcancaveis|s_0}, 0)\}$, em que, dado um estado inicial s_0, $B_{alcancaveis|s_0}$ representa o bloco de estados alcançáveis a partir de s_0 e $B_{\neg alcancaveis|s_0}$ representa o bloco de estados inalcançáveis a partir de s_0.

Com o objetivo de encontrar $\mathcal{P}_{DD}^{S|s_0}$, é necessário visitar todos os estados alcançáveis a partir de s_0. Este procedimento é feito camada por camada, simulando cada ação do MDP em cada estado alcançável (como em uma busca em largura (Russel e Norvig, 2003)). Isso pode ser feito eficientemente se representarmos os estados em cada camada como BDDs (Fourman, 2000) e efetuarmos as operações de união, intersecção e marginalização entre BDDs.

Por exemplo, a Figura 6.2 ilustra uma instância do domínio *Navigation*, no qual o objetivo do agente é navegar em uma grade 2×3 (Figura 6.2(a)) e chegar na posição (X_2, Y_3) partindo da posição (X_2, Y_1), usando ações que o levam para cada uma das seguintes direções: norte, sul, leste e oeste. A partição $P_{DD}^{S|s_0}$ para esse problema é mostrada na Figura 6.2(b). Note que os estados que correspondem a caminhos que terminam na folha de valor 1 são os estados alcançáveis a partir do estado inicial $s_0 = \neg X_1, Y_1 \wedge \neg X_1, Y_2 \wedge \neg X_1, Y_3 \wedge X_2, Y_1 \wedge \neg X_2, Y_2 \wedge \neg X_2, Y_3$.

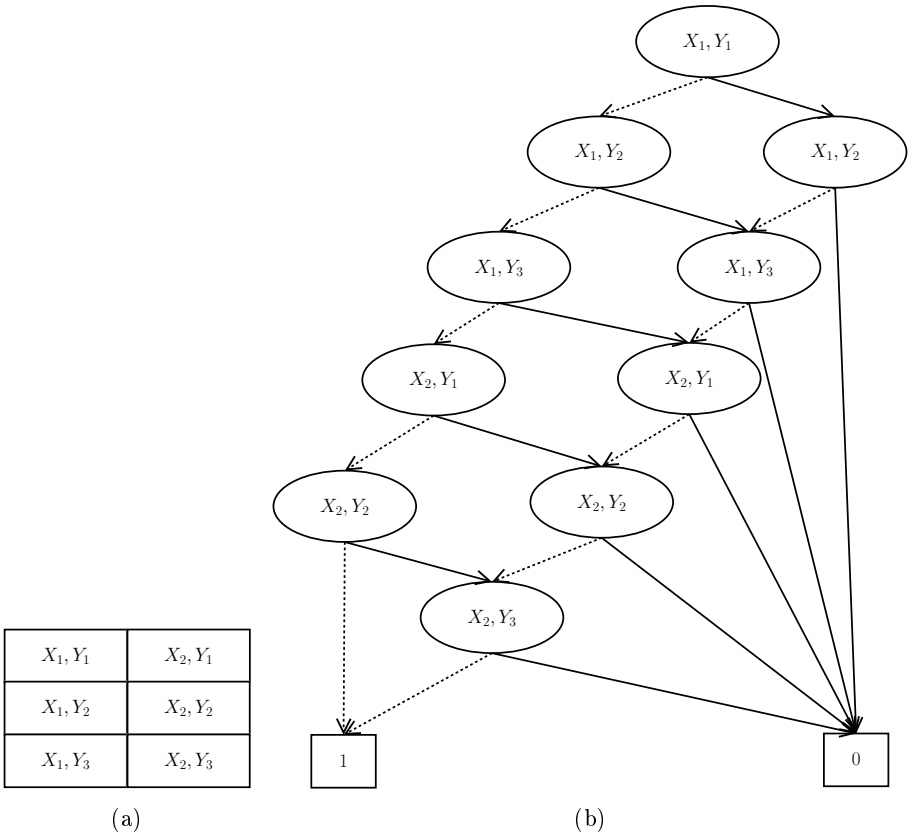


Figura 6.2: *Instância do domínio Navigation com 6 variáveis de estado (2^6 estados). (a) Grade 2 por 3. (b) Partição por alcançabilidade a partir de $s_0 = \neg X_1 Y_1 \wedge \neg X_1 Y_2 \wedge \neg X_1 Y_3 \wedge X_2 Y_1 \wedge \neg X_2 Y_2 \wedge \neg X_2 Y_3$.*

Algoritmo 6.1.1: ObtemSucessoresPorAcao ($CamadaAtual_{DD}$, a)

Entrada: $CamadaAtual_{DD}$: um BDD representando um conjunto de estados alcançados após n ações; a: uma ação do MDP fatorado.

Saída: $Sucessores_{DD}$: um BDD representando um conjunto de estados alcançáveis a partir de $CamadaAtual_{DD}$ ao executar a ação a.

1 $Sucessores_{DD} \leftarrow CamadaAtual_{DD}$;
2 **para cada** $X_i \in X$ **faça**
3 | $SucessoresParciais_{DD} \leftarrow$ *BDD em que cada folha da CPT de* (a, X_i') *com probabilidade positiva é igual a 1*;
4 | $Sucessores_{DD} \leftarrow Sucessores_{DD} \cap SucessoresParciais_{DD}$;
5 **fim**
6 **para cada** $X_i \in X$ **faça**
7 | $Sucessores_{DD} \leftarrow$ *Marginalização em* $Sucessores_{DD}$ *sobre variável* X_i;
8 **fim**
9 $Sucessores_{DD} \leftarrow$ *Mapeie todas as variáveis de* $Sucessores_{DD}$ *para* X_i *ao invés de* X_i';
10 **retorna** $Sucessores_{DD}$;

Um conjunto de estados pode ser representado por um BDD da seguinte forma: os estados que pertencem ao conjunto levam a folhas com valor 1; e os estados que não pertencem ao conjunto levam a folhas com valor 0. O Algoritmo 6.1.1 recebe um BDD, que representa um conjunto contendo os estados da camada atual, e uma ação a. Como resultado, o algoritmo devolve um BDD que informa quais são os estados sucessores da execução de a na camada atual.

O funcionamento do Algoritmo 6.1.1 é detalhado a seguir. Na Linha 1, a camada de sucessores é inicializada com a camada de estados atuais, isso é feito de forma temporária. Nas Linhas 2-4, o algoritmo considera para cada variável $X_i \in X$ quais são os estados que podem ser alcançados verificando o ADD $P(X_i'|CamadaAtual_{DD}, a)$. Para tornar essa computação mais eficiente, os ADDs são substituidos por BDDs de forma que se a probabilidade for positiva, substituímos seu valor por 1. Na Linha 5, o algoritmo calcula a intersecção dos estados que podem ser alcançados considerando cada variável de estado X_i' de forma que apenas aqueles com probabilidade conjunta diferente de 0 sejam incluídos na camada de sucessores da ação a. Após a execução da Linha 5, temos um BDD em $Sucessores_{DD}$ que é composto por estados na camada atual e estados na camada seguinte. Nas Linhas 6-8 é computada uma marginalização sobre cada variável $X_i \in X$ cujo resultado é $Sucessores_{DD}$ contendo apenas os estados da camada seguinte. Na Linha 9, o algoritmo mapeia as variáveis usadas para descrever estados da camada seguinte de forma que essas variáveis sejam da forma X_i ao invés de X_i'. Com isso, a camada seguinte pode ser usada adequadamente para refinamentos se a partição já tiver sido encontrada e para a computação das camadas de estados mais adiante no MDP se o processo ainda não tiver terminado.

O Algoritmo 6.1.2 recebe um MDP M, um estado inicial s_0 e uma profundidade máxima p. Esse algoritmo explora os estados alcançáveis a partir de s_0, camada por camada, chamando o Algoritmo 6.1.1 para cada camada e ação. O algoritmo termina quando a profundidade p é alcançada ou se a camada seguinte é igual à camada atual, i.e., todos estados alcançáveis foram visitados.

O Algoritmo 6.1.2 é explicado em detalhes a seguir. Na Linha 1, a partição por alcançabilidade é inicializada com uma única folha de valor 0, que representa todos estados como não alcançáveis. Nas Linhas 2-3, é criado um BDD para representar a camada atual, que inicialmente possui apenas um caminho que leva a uma folha de valor 1 e esse caminho é dado pelas atribuições do estado inicial. Nas Linhas 2-20, o laço principal visita os estados alcançáveis, camada por camada. Internamente ao laço principal, as Linhas 5-14 permitem descobrir os estados da camada seguinte fazendo chamadas ao Algoritmo 6.1.1. Antes de visitar a próxima camada, o algoritmo adiciona a camada seguinte à partição por alcançabilidade e verifica nas Linhas 16-18 se a camada seguinte é diferente ou se foi encontrado um ponto fixo (momento no qual o algoritmo pode ser interrompido). Na Linha 19, o algoritmo faz com que a camada atual receba a camada seguinte para iniciar a próxima iteração.

Algoritmo 6.1.2: ObtemEstadosAlcançáveis (M, s_0, p)

Entrada: M: um MDP fatorado, s_0: um estado inicial representado de forma fatorada e p: uma profundidade máxima.
Saída: $\mathcal{P}_{DD}^{S|s_0}$: uma partição por alcançabilidade.

$\mathcal{P}_{DD}^{S|s_0} \leftarrow \{0\}$;
$CamadaAtual_{DD} \leftarrow \{0\}$;
$CamadaAtual_{DD} \leftarrow CamadaAtual_{DD} \cup \{s_0 \mapsto 1\}$;
para $i \leftarrow 0$ *até* p **faça**
    $CamadaSeguinte_{DD} \leftarrow \emptyset$;
    para cada $a \in A$ **faça**
        se $CamadaSeguinte_{DD} = \emptyset$ **então**
            $CamadaSeguinte_{DD} \leftarrow ObtemSucessoresPorAcao(CamadaAtual_{DD}, a)$;
        fim
        senão
            $CamadaSeguinteAcao_{DD} \leftarrow ObtemSucessoresPorAcao(CamadaAtual_{DD}, a)$;
            $CamadaSeguinte_{DD} \leftarrow CamadaSeguinte_{DD} \cup CamadaSeguinteAcao_{DD}$;
        fim
    fim
    $\mathcal{P}_{DD}^{S|s_0} \leftarrow \mathcal{P}_{DD}^{S|s_0} \cup CamadaSeguinte_{DD}$;
    se $CamadaSeguinte_{DD} = CamadaAtual_{DD}$ **então**
        interrompa;
    fim
    $CamadaAtual_{DD} \leftarrow CamadaSeguinte_{DD}$;
fim
retorna $\mathcal{P}_{DD}^{S|s_0}$;

Na Linha 21, após o laço principal, o algoritmo devolve a partição por alcançabilidade.

Teorema 2. *Os Algoritmos 6.1.1 e 6.1.2 encontram $\mathcal{P}_{DD}^{S|s_0}$ no pior caso em tempo polinomial no número de estados.*

Demonstração. Considere que $|S| = 2^n$ é o tamanho do conjunto de estados. O número máximo de nós em um BDD que considera cada variável $X_i \in X$ e $X_i' \in X'$ é limitado superiormente por $2 \times |S|^2 - 1$, i.e., $O(|S|^2)$. Considere as complexidades apresentadas para cada operação envolvendo BDDs na Tabela 2.3. O Algoritmo 6.1.1 tem sua complexidade dada pela seguinte soma, em que ci é a complexidade da linha i:

$$c1 + |X| \times (c3 + c4) + |X| \times c7 + c9 + c10. \tag{6.1}$$

Substituindo cada ci pelas devidas complexidades é possível limitar superiormente a Equação 6.1 por:
$O(|S|^2) + |X| \times [O(|S|^2) + O(|S|^4)] + |X| \times O(|S|^4) + O(|S|^2) + O(|X|) =$
$2 \times O(|S|^2) + |X| \times [O(|S|^2) + O(|S|^4)] + |X| \times O(|S|^4) + O(|X|) \leq$ // já que $|S| \geq |X|$ para $|X| \geq 1$
$3 \times O(|S|^2) + |X| \times [O(|S|^2) + O(|S|^4)] + |X| \times O(|S|^4) =$
$(|X| + 3) \times O(|S|^2) + 2 \times |X| \times O(|S|^4)] \leq$
$2 \times |X| \times O(|S|^4) + 2 \times |X| \times O(|S|^4)] =$
$6 \times |X| \times O(|S|^4) = O(|S|^4).$
Portanto, o Algoritmo 6.1.1 é polinomial no número de estados para uma constante $c = 6 \times |X|$ e $|S| \geq 1$. [1]

De forma similar, a complexidade de cada linha do Algoritmo 6.1.2 é dada por di e pode ser calculada da seguinte maneira:

$$d1 + d2 + d3 + p \times [d5 + |A| \times (d11 + d12) + d15 + d16 + d17 + d19] + d21. \tag{6.2}$$

Com isso, podemos limitar a Equação 6.2 superiormente com:
$2 \times O(|A|) + O(|S|^4) + p \times [O(|A|) + 2 \times O(|S|^4) + O(|S|^4) + 2 \times O(|A|) + O(|S|^2)] + O(|A|) =$
$3 \times O(|A|) + O(|S|^4) + p \times [3 \times O(|S|^4) + O(|S|^2) + 3 \times O(|A|)] \leq$ [2]
$3 \times O(|S|^4) + p \times [3 \times O(|S|^4) + 4 \times O(|S|^2)] \leq$
$3 \times O(|S|^4) + p \times [7 \times O(|S|^4)] =$
$(7p + 3) \times O(|S|^4).$
Como S é um conjunto finito de estados e p é uma constante que delimita uma profundidade máxima, o Algoritmo 6.1.2 é polinomial em S para uma constante $c = 7p + 3$ e $|S| \geq 1$.

Como os Algoritmos 6.1.1 e 6.1.2 são polinomiais no número de estados do MDP, o mesmo pode ser afirmado com relação à complexidade de pior caso para encontrar a partição $\mathcal{P}_{DD}^{S|s_0}$. □

Depois de computar a partição por alcançabilidade $\mathcal{P}_{DD}^{S|s_0}$ em tempo polinomial, podemos utilizá-la como uma máscara que permite economizar tempo e espaço evitando que refinamentos sejam realizados sobre blocos de estados inalcançáveis dado s_0.

6.1.2 Redução de modelos sobre estados alcançáveis

O algoritmo MRFS pode ser melhorado se utilizarmos a partição por alcançabilidade $\mathcal{P}_{DD}^{S|s_0}$, especialmente em problemas com matrizes de transição esparsas em que o número de estados alcançáveis pode ser bem menor do que o conjunto completo de estados.

O algoritmo *Reachability-based MRFS (ReachMRFS)* (Algoritmo 6.1.3) funciona como segue. Primeiro, suponha que $\mathcal{P}_{DD}^{S|s_0}$ é dada como entrada. Nas Linhas 1-5, cada partição $P_{DD}^{A,R}$ é multiplicada pela partição $\mathcal{P}_{DD}^{S|s_0}$, o que permite a obtenção de partições simplificadas $\mathcal{Q}_{DD}^{a,R}$, em que os

[1] Os cálculos foram realizados supondo que os BDDs não estão em sua forma reduzida. Contudo, na prática, sempre são usados na forma reduzida.

[2] Os conjuntos A e S de um MDP são finitos, então $|A|$ e $|S|$ são constantes. Dessa forma, temos que $|A|$ é limitado superiormente por $O(|S|)$ e $|S|$ é limitado superiormente por $O(|A|)$.

estados inalcançáveis são rotulados com 0. Como a partição por alcançabilidade tem apenas dois blocos, identificados com os valores 0 e 1, ao realizar o produto dessa partição com as partições do MDP, algumas folhas passam a ter valor 0 e outras continuam como estavam. Dessa forma, muitas folhas podem passar a ter valor 0, o que reduz o número de folhas e permite um armazenamento mais compacto em memória. Além disso, ao realizar refinamentos entre partições que já foram simplificadas, o número de blocos na partição refinada tende a crescer menos do que em partições sobre S. As partições $\mathcal{Q}_{DD}^{a,R}$ são usadas para computar uma partição uniforme com relação à função recompensa sobre os estados alcançáveis. Depois de obter as partições por recompensa sobre S_{alc}, nas Linhas 6-13 é computada uma partição com blocos estáveis sobre os estados alcançáveis do mesmo modo, obtendo simplificações a partir de $P_{DD}^{A,X}$. Finalmente, na Linha 14 é computada a bissimulação estocástica sobre os estados alcançáveis.

Algoritmo 6.1.3: ReachMRFS-V1(M, $P_{DD}^{S|s_0}$)

Entrada: M: um MDP fatorado; e $P_{DD}^{S|s_0}$: uma partição por alcançabilidade.
Saída: $\mathcal{P}$: uma partição em que existe uma bissimulação estocástica sobre os estados alcançáveis e 1 bloco contém todos os estados inalcançáveis dado s_0.
// MRFS sobre estados alcançáveis

1 $\mathcal{P}_{DD}^{A,R} \leftarrow 1$;
2 **para** $a \in A$ **faça**
3 $\quad \mathcal{Q}_{DD}^{a,R} \leftarrow \mathcal{P}_{DD}^{a,R} \otimes \mathcal{P}_{DD}^{S|s_0}$;
4 $\quad \mathcal{P}_{DD}^{A,R} \leftarrow \mathcal{P}_{DD}^{A,R} \otimes \mathcal{Q}_{DD}^{a,R}$;
5 **fim**
6 $\mathcal{P}_{DD}^{A,X} \leftarrow 1$;
7 **para** $a \in A$ **faça**
8 $\quad \mathcal{Q}_{DD}^{a,X} \leftarrow 1$;
9 $\quad$ **para** $X_i \in X_e$ **faça**
10 $\quad\quad \mathcal{Q}_{DD}^{a,X} \leftarrow \mathcal{Q}_{DD}^{a,X} \otimes \mathcal{P}_{DD}^{a,X_i} \otimes \mathcal{P}_{DD}^{S|s_0}$;
11 $\quad$ **fim**
12 $\quad \mathcal{P}_{DD}^{A,X} \leftarrow \mathcal{P}_{DD}^{A,X} \otimes \mathcal{Q}_{DD}^{a,X}$;
13 **fim**
14 $\mathcal{P}_{DD} \leftarrow \mathcal{P}_{DD}^{A,R} \otimes \mathcal{P}_{DD}^{A,X}$;
15 **retorna** $\mathcal{P}_{DD}$;

6.1.3 Minimização e ϵ-Redução sobre estados alcançáveis

Os algoritmos *Reachability-based MMFS (ReachMMFS)* e *Reachability-based ϵMRFS (Reach-ϵMRFS)* são obtidos de forma similar ao ReachMRFS, ou seja, com o uso de $\mathcal{P}_{DD}^{S|s_0}$ para evitar computações com estados inalcançáveis dado o estado inicial s_0. Além disso, ReachMMFS usa Juntar-Blocos (Algoritmo 4.6.1); enquanto Reach-ϵMRFS usa JuntarBlocosAproximado (Algoritmo 5.1.1). Nessas versões com alcançabilidade, ao invés de computar junções de todos para todos entre os blocos da partição, utiliza-se apenas os blocos com rótulos diferentes de zero, i.e., blocos alcançáveis dado s_0.

Para resolver o MDP minimal obtido com ReachMMFS, basta usar o LRTDP. Nesse caso, é utilizado o mesmo procedimento que obtém as probabilidades de transição entre blocos com os MDPs reduzidos obtidos por ReachMRFS. Por outro lado, para resolver um BMDP reduzido obtido com Reach-ϵMRFS, o procedimento se baseia no LRTDP Robusto. Para identificar as probabilidades de transição entre blocos que agora são dadas por intervalos, é necessário obter as distribuições de probabilidades de cada estado no bloco porque os estados em um mesmo bloco podem ter

diferentes distribuições. Isso é um pouco mais difícil do que escolher um representante como é feito nas partições obtidas com base em ReachMRFS e ReachMMFS. Após o término de Reach-ϵMRFS com uma combinação dos Algoritmos 6.1.3 e 5.1.1, é necessário resolver o BMDP com o LRTDP Robusto (Algoritmo 3.2.4).

Na implementação de Reach-ϵMRFS deste trabalho, as funções $R_{\updownarrow}$ e $P_{\updownarrow}$ são obtidas conforme é apresentado no Algoritmo 6.1.4, no qual são encontradas as probabilidades de cada estado num mesmo bloco alcançarem outros blocos. Depois disso, para cada bloco alcançável B', são definidos quais estados minimizam e maximizam a probabilidade de alcançar o bloco B' selecionado. Após avaliar cada bloco B' com esse algoritmo, o Algoritmo 6.1.4 devolve as funções encontradas, que são usadas na Equação 3.5 para resolver o BMDP. Contudo, ainda é necessário obter probabilidades exatas entre os blocos para encontrar uma solução robusta. Isso é feito com o Algoritmo 3.2.1, que foi visto no Capítulo 3.

Algoritmo 6.1.4: ObtemFuncoesBloco($\mathcal{P}$, B, a)

Entrada: $\mathcal{P}$: uma partição que é ϵ-homogênea; B: um bloco alcançável em $\mathcal{P}$; e a: uma ação do MDP.

Saída: $(R_{\updownarrow}, P_{\updownarrow})$: função recompensa intervalar para (B, a) e uma distribuição de probabilidades dadas por intervalos $P_{\updownarrow}$, que apresenta os intervalos de transição, i.e., as probabilidades mínima e máxima de alcançar cada bloco seguinte considerando a ação a.

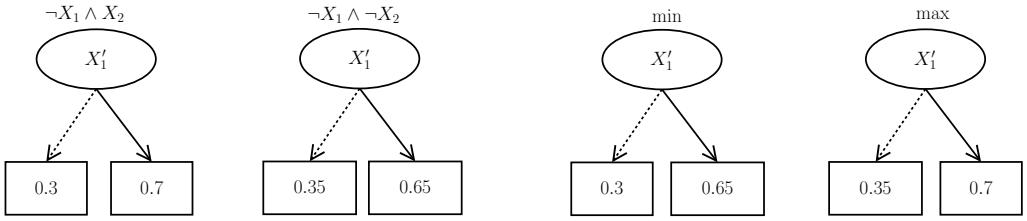
$R_{\uparrow}(B, a) \leftarrow \max_{s \in B} R(s, a)$;
$R_{\downarrow}(B, a) \leftarrow \min_{s \in B} R(s, a)$;
para $s \in B$ **faça**
  para $B' \in \mathcal{P}$ **faça**
    $P(B'|s, a) \leftarrow P(B'|s, a) + P(s'|s, a)$;
  fim
fim
para $B' \in \mathcal{P}$ **faça**
  $P_{\downarrow}(B'|B, a) \leftarrow \min_{s \in B} P(B'|s, a)$;
  $P_{\uparrow}(B'|B, a) \leftarrow \max_{s \in B} P(B'|s, a)$;
fim
retorna $(R_{\downarrow}, P_{\updownarrow})$

Considerando que a implementação é fatorada, para cada s que satisfaz a fórmula DNF que descreve B, é possível obter o valor mínimo $R_{\downarrow}$ e máximo $R_{\uparrow}$ com base em cada $R(s, a)$.

Em um segundo momento, deve-se encontrar $P_{\updownarrow}(B'|B, a)$, que é uma tarefa mais complicada. Para cada s que satisfaz a fórmula que descreve B, é necessário encontrar os sucessores de s calculando a probabilidade conjunta para MDPs fatorados (Equação 2.9). Considerando um BDD para cada $B' \in \mathcal{P}$ alcançável, é possível multiplicar o BDD atual pela probabilidade conjunta, o que permite simplificar o ADD da função de transição restando apenas as probabilidades de alcançar o bloco B' fixado. Ao somar essas probabilidades, é obtido $P(B'|s, a)$. Depois disso, basta realizar a mesma operação para outros estados em B e outros blocos B' (todas as combinações considerando estados em B e blocos alcançáveis B'). Por fim, com vários ADDs que possuem as probabilidades $P(B'|s, a)$, é calculado o mínimo desses ADDs que é referenciado por $P_{\downarrow}(B'|B, a)$; e o máximo, que é referenciado por $P_{\uparrow}(B'|B, a)$ (vide operações com ADDs no Capítulo 2). Dessa forma, o processo termina e as probabilidades $P_{\updownarrow}(B'|B, a)$ para cada B' podem ser devolvidas para serem usadas pelo Algoritmo 3.2.4. O mesmo processo se repete para cada bloco do BMDP resultante obtido por Reach-ϵMRFS.

Considere o exemplo da Figura 4.8(a). Supondo que o estado inicial é dado pela atribuição $\neg X_1 \wedge \neg X_2 \wedge \neg X_3$, através da partição 0.05-homogênea é possível perceber que esse estado é representado pelo estado abstrato descrito por $\neg X_1$. Considerando as variáveis de estado essenciais

$X_e = \{X_1, X_2\}$, os estados que satisfazem $\neg X_1$ são aqueles que têm em suas descrições $\neg X_1 \wedge X_2$ e $\neg X_1 \wedge \neg X_2$. Com isso, as probabilidades conjuntas considerando esses dois conjuntos de estados como origem são exibidas na Figura 6.3(a). Observe que apenas X_1 aparece nos ADDs de probabilidade conjunta (isso é simplificado dessa forma porque apenas X_1 é usada na partição 0.05-homogênea). Ao computar o mínimo e o máximo desses ADDs, os ADDs da Figura 6.3(b) são obtidos. A partir desses ADDs, é possível obter que $P_{\updownarrow}(X_1' = 0|X_1 = 0, a) = [0.3, 0.35]$ e $P_{\updownarrow}(X_1' = 1|X_1 = 0, a) = [0.65, 0.7]$.



(a) Probabilidade de alcançar cada bloco a partir de dois estados diferentes.

(b) Probabilidades mínimas e máximas de alcançar cada bloco.

Figura 6.3: *Exemplo de computação do Algoritmo 6.1.4 (Linhas 3-11) com o uso de ADDs.*

6.2 Eliminação de partições repetidas

Uma outra melhoria proposta nesse trabalho para a computação eficiente da bissimulação estocástica é a eliminação de partições repetidas nos algoritmos MRFS e ReachMRFS, que envolvem partições obtidas com base em: (1) função recompensa considerando todas as ações do MDP; e (2) função de transição considerando as variáveis de estado essenciais X_e. De forma geral, o número de partições distintas utilizadas por esses algoritmos pode ser no máximo $|A|+(|A|\times|X|)$ (Capitulo 4). Contudo, em situações práticas, essas partições não são todas distintas e utilizar partições repetidas torna a computação da bissimulação estocástica mais lenta. A seguir, definimos uma outra forma de se computar uma bissimulação estocástica exata fazendo a eliminação de partições repetidas antes do processo de refinamentos.

Sejam $\mathcal{P}$ e $\mathcal{P}'$ partições de um MDP. Dizemos que $\mathcal{P} = \mathcal{P}'$ se $|\mathcal{P}| = |\mathcal{P}'|$ e se para cada $B_i \in \mathcal{P}$ existe um bloco $B_j \in \mathcal{P}'$ com a mesma expressão DNF caracterizando ambos. Com base nisso, uma partição $\mathcal{P}'$ obtida de uma função f (recompensa ou transição probabilística) é repetida se em MRFS foi encontrada uma partição $\mathcal{P}$ (antes de encontrar $\mathcal{P}'$), obtida de uma função g (recompensa ou transição probabilística), e temos que $\mathcal{P} = \mathcal{P}'$. Dada uma lista de partições L, dizemos que as partições são distintas entre si se para cada partição $\mathcal{P}_i \in L$ obtida a partir de uma função do MDP, não houver $\mathcal{P}_j \in L$ obtida a partir de uma outra função do MDP tal que $\mathcal{P}_i = \mathcal{P}_j$.

Com base na relação de igualdade entre partições, é possível ignorar durante a etapa de refinamento de MRFS qualquer partição repetida. Ao evitar partições que não refinam a partição atual e considerar apenas uma lista L de partições distintas entre si, a computação de MRFS se torna mais eficiente. O Algoritmo 6.2.1 apresenta uma maneira de criar uma lista que contém partições mutuamente distintas de um MDP.

Considere o exemplo da Figura 6.4 em que $X = \{X_1, X_2\}$ e $A = \{a_1, a_2\}$. Observe que os ADDs $R(X, a_1)$, $P(X_1' = 1|X, a_1)$, $P(X_2' = 1|X, a_1)$, $R(X, a_2)$, $P(X_1' = 1|X, a_2)$ e $P(X_2' = 1|X, a_2)$, dados na definição do MDP, são mutuamente distintos. Apesar disso, as partições que são obtidas com cada ADD não são distintas. A partir desse MDP, são geradas as seguintes partições: $\mathcal{P}_R^{a_1} = \{X_1, \neg X_1\}$, $\mathcal{P}_{X_1}^{a_1} = \{X_1 \wedge \neg X_2, X_1 \wedge X_2, \neg X_1\}$, $\mathcal{P}_{X_2}^{a_1} = \{X_1, \neg X_1\}$, $\mathcal{P}_R^{a_2} = \{X_1 \wedge \neg X_2, X_1 \wedge X_2, \neg X_1\}$, $\mathcal{P}_{X_1}^{a_2} = \{X_1, \neg X_1\}$ e $\mathcal{P}_{X_2}^{a_2} = \{verdadeiro\}$ ($\mathcal{P}_{X_2}^{a_2}$ tem um único bloco com todos os estados). Uma implementação do Algoritmo 6.2.1 que recebesse o MDP fatorado deste exemplo, devolveria as seguintes partições (considerando a ordem que as partições foram mencionadas): $\mathcal{P}_R^{a_1} = \{X_1, \neg X_1\}$, $\mathcal{P}_{X_1}^{a_1} = \{X_1 \wedge \neg X_2, X_1 \wedge X_2, \neg X_1\}$ e $\mathcal{P}_{X_2}^{a_2} = \{verdadeiro\}$. As outras partições são repetidas, por

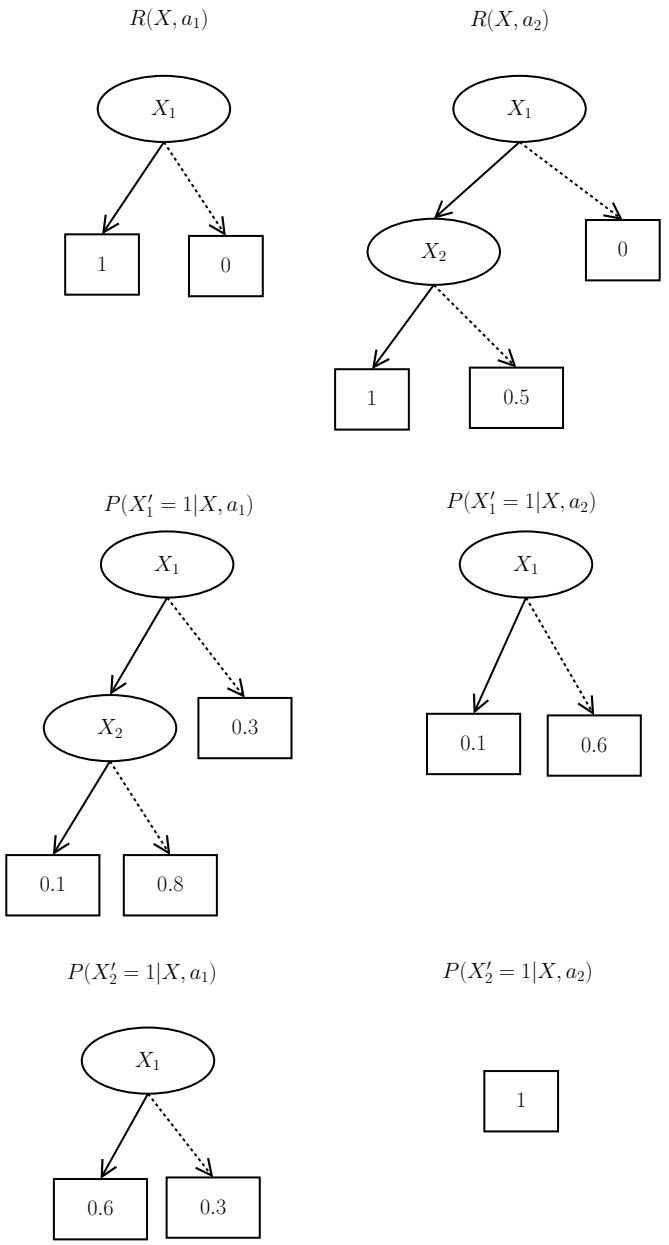


Figura 6.4: *Um MDP fatorado que possui funções a partir das quais é possível gerar partições repetidas.*

exemplo: $\mathcal{P}_R^{a_1} = \mathcal{P}_{X_1}^{a_2} = \mathcal{P}_{X_2}^{a_1} = \{X_1, \neg X_1\}$. Dessa forma, computar a intersecção das 6 partições resulta em $\mathcal{P} = \{X_1 \wedge \neg X_2, X_1 \wedge X_2, \neg X_1\}$, que é o mesmo resultado obtido se forem computadas as intersecções considerando as 3 partições (sem repetições). A Figura 6.5 ilustra as partições obtidas a partir das funções do MDP que foram mostradas na Figura 6.4.

O Algoritmo 6.2.1 recebe um MDP M e um conjunto de variáveis de estado essenciais X_e para M. Nas Linhas 1-7, o algoritmo inicializa o mapeamento $\mathcal{P}_{distintas}$ de forma que cada partição adicionada seja considerada distinta (verdadeiro). Depois disso, nas Linhas 8-16, as partições são verificadas na mesma ordem da inserção e o algoritmo alterna o mapeamento das partições que se repetem para que seus valores sejam recebam falso, i.e., não são consideradas distintas. Enfim, nas Linhas 17-22, o algoritmo preenche a lista L com as partições distintas.

O Algoritmo 6.2.2 recebe um MDP M e um conjunto de variáveis de estado essenciais X_e. Antes de computar uma bissimulação estocástica, o algoritmo obtém uma lista L de partições distintas computada pelo Algoritmo 6.2.1. Depois de obter as $|L|$ partições distintas, o algoritmo computa um refinamento dessas partições usando a operação de intersecção de partições apresentada no Capítulo 4.

Algoritmo 6.2.1: ObtemPartiçõesDistintas (M, X_e)

Entrada: M: um MDP fatorado, X_e: um conjunto de variáveis de estado essenciais para o MDP.

Saída: L: uma lista que contém as partições distintas do MDP de entrada.

```
1  P_distintas ← ∅ /* inicializa mapa que informa se cada partição é repetida ou não.*/;
2  para cada a ∈ A faça
3  |  P_distintas.adicione(P_R^a ↦ true);
4  |  para cada X_i ∈ X_e faça
5  |  |  P_distintas.adicione(P_{X_i}^a ↦ true)
6  |  fim
7  fim
8  para i = 1 até |P_distintas| faça
9  |  se P_distintas[i] = true então
10 |  |  para j = i + 1 até |P_distintas| faça
11 |  |  |  se P_distintas[i] = P_distintas[j] então
12 |  |  |  |  P_distintas[j] ← (P_distintas[j] ↦ false);
13 |  |  |  fim
14 |  |  fim
15 |  fim
16 fim
17 L ← ∅;
18 para i = 1 até |P_distintas| faça
19 |  se P_distintas[i] = true então
20 |  |  L.adicione(P_distintas[i]);
21 |  fim
22 fim
23 retorna L;
```

Algoritmo 6.2.2: ReducaoDeModelosSemParticoesRepetidas (M, X_e)

Entrada: M: um MDP fatorado, X_e: um conjunto de variáveis de estado essenciais para o MDP.

Saída: $\mathcal{P}$: uma partição homogênea que representa um MDP enumerativo reduzido.

```
1  P ← 1;
2  L ← ObtemParticoesDistintas(M, X_e);
3  para P_i ∈ L faça
4  |  P ← P ∩ P_i;
5  fim
6  retorna P;
```

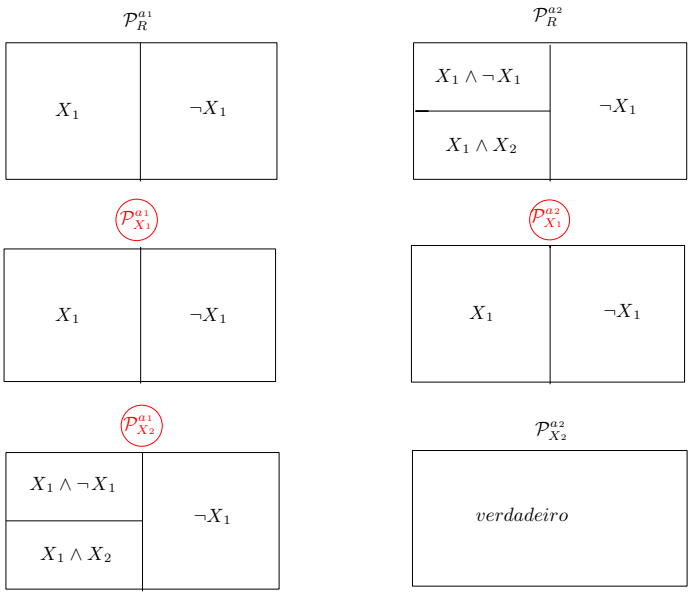


Figura 6.5: *Exemplo de partições obtidas do MDP fatorado da Figura 6.4. Os nomes das partições em vermelho e circulados identificam partições consideradas repetidas pelo Algoritmo 6.2.1.*

O Algoritmo 6.1.3, apresentado anteriormente neste capítulo, reduz um MDP considerando os estados alcançáveis, as partições por recompensa e partições obtidas de X_e para cada ação. Uma forma de evitar as partições repetidas e ainda assim computar bissimulações estocásticas sobre os estados alcançáveis é com a combinação de ReachMRFS e eliminação de partições repetidas, conforme é apresentado no Algoritmo 6.2.3.

Algoritmo 6.2.3: ReachMRFS-V2(M, $P_{DD}^{S|s_0}$)

Entrada: M: um MDP fatorado; e $P_{DD}^{S|s_0}$: uma partição por alcançabilidade.
Saída: $\mathcal{P}_{DD}$: uma partição em que existe uma bissimulação estocástica sobre os estados alcançáveis e 1 bloco contém todos os estados inalcançáveis dado s_0.
// MRFS sobre estados alcançáveis sem partições repetidas do MDP

1 $\mathcal{P}_{DD} \leftarrow 1$;
2 $L \leftarrow ObtemParticoesDistintas(M, X_e)$;
3 **para** $\mathcal{P}_{DD}^i \in L$ **faça**
4 $\quad \mathcal{P}_{DD}^i \leftarrow \mathcal{P}_{DD}^i \cap \mathcal{P}_{DD}^{S|s_0}$;
5 $\quad \mathcal{P}_{DD} \leftarrow \mathcal{P}_{DD} \cap \mathcal{P}_{DD}^i$;
6 **fim**
7 **retorna** $\mathcal{P}_{DD}$;

O Algoritmo 6.2.3 computa uma lista L de partições distintas chamando o Algoritmo 6.2.1 e depois as simplifica utilizando a partição $\mathcal{P}_{DD}^{S|s_0}$. Como resultado do algoritmo, a partição $\mathcal{P}_{DD}$ contém uma bissimulação estocástica sobre os estados alcançáveis. Por outro lado, os estados não alcançáveis a partir de s_0 estão todos em um único bloco. Com base na partição encontrada pelos Algoritmos 6.1.3 e 6.2.3, se é dado um estado inicial s_0, é possível descobrir em qual bloco da partição o estado s_0 está. Da mesma forma, é possível saber em que bloco cada estado s está. Como a partição devolvida por esses algoritmos é uma bissimulação estocástica exata sobre os estados alcançáveis, qualquer estado alcançável em um mesmo bloco têm a mesma probabilidade de alcançar outros blocos. Desta forma, se considerarmos o LRTDP, cada estado visitado no trial passa a representar um bloco de estados e assim, sempre que algum estado num bloco já representado é visitado, basta atualizar o representante, que é o primeiro estado visitado no bloco atual. De maneira similar, a probabilidade de $s \in B$ alcançar algum bloco B_w é dada pela Equação 4.4.

Neste trabalho, foi possível observar que em alguns problemas é obtida uma aceleração na computação de MRFS e ReachMRFS quando partições repetidas são eliminadas durante a redução. Além disso, o número de blocos na partição homogênea resultante é o mesmo, se consideramos ou não as partições repetidas.

6.3 Resumo do Capítulo

Nesse capítulo foram apresentadas as seguintes contribuições:

- a computação da bissimulação estocástica sobre os estados alcançáveis de um MDP com estado inicial;
- a demonstração de que a análise de alcançabilidade para determinar os estados alcançáveis de um MDP pode ser realizada com ADDs em tempo polinomial no número de estados desse MDP;
- a eliminação de partições repetidas antes da computação da bissimulação estocástica exata quando essa última é calculada usando o algoritmo de redução de modelos; e
- a combinação de bissimulação estocástica exata sobre os estados alcançáveis e eliminação de partições repetidas (ReachMRFS-V2).

Capítulo 7

Análise Experimental

Neste capítulo, apresentamos uma breve descrição dos domínios utilizados para a análise comparativa dos algoritmos propostos. Os experimentos realizados tiveram como objetivo:

1. análise comparativa entre os algoritmos para a computação da bissimulação estocástica: sobre o conjunto completo de estados (algoritmo MRFS) e sobre os estados alcançáveis (algoritmo ReachMRFS);
2. análise do algoritmo de computação da bissimulação estocástica com alcançabilidade eliminando partições repetidas;
3. análise do valor de ϵ no cálculo da bissimulação estocástica aproximada com alcançabilidade (algoritmo Reach-ϵMRFS);
4. análise comparativa entre os 3 algoritmos propostos: ReachMRFS, ReachMMFS e Reach-ϵMRFS; e
5. análise de desempenho das soluçõoes do MDP original e do MDP reduzido.

Os experimentos realizados para este trabalho rodaram em um computador com 4GB de RAM e processador Intel Core i5. As implementações foram desenvolvidas em Java usando o sistema RDDLSim (Sanner, 2010), o simulador oficial da IPPC-2011 (*International Probabilistic Planning Competition-2011*) (Sanner, 2010). Dentre essas implementações foram desenvolvidas variações de agregação de estados que permitem encontrar bissimulações estocásticas sobre S ou $S_{alcancaveis|s_0}$ (o conjunto de estados alcançáveis dado um estado inicial s_0). Depois de executar as diferentes formas de agregação de estados para computar bissimulações estocásticas, o MDP (ou BMDP, se $\epsilon > 0$) reduzido foi resolvido usando o algoritmo LRTDP (Seção 2.2.4) (ou LRTDP Robusto (Seção 3.2.2)) com $\gamma = 0.99$ e $\mathcal{E} = 10^{-3}$.

7.1 Domínios para análise comparativa

Para analisar as implementações feitas neste trabalho, foram realizados experimentos com os problemas da IPPC-2011. Nessa competição, os problemas foram especificados em RDDL (*Relational Dynamic Influence Diagram Language*) (Sanner, 2010), uma linguagem para representar MDPs fatorados.

Na IPPC-2011 haviam 8 domínios com 10 instâncias cada. Para avaliar as implementações deste trabalho, foram escolhidos 6 dos 8 domínios: *Crossing Traffic*, *Elevators*, *Game of Life*, *Navigation*, *Skill Teaching* e *SysAdmin* (Apêndice B). Além disso, foram incluídas algumas instâncias com tamanhos diferentes dos usados na competição para permitir uma melhor comparação entre os algoritmos propostos.

Os domínios também podem ser classificados de acordo com a densidade da função de transição, como foi definido no Capítulo 2. Se a matriz de transição for densa para pelo menos uma ação $a \in A$

do MDP, então consideramos que *o domínio é denso*; caso contrário, consideramos que *o domínio é esparso*.

Nas próximas subseções, são apresentados os domínios da IPPC-2011.

7.1.1 Crossing Traffic

O *Crossing Traffic* pode ser visto como uma versão de planejamento para o jogo eletrônico Frogger (Konami, 1981). Este é um domínio esparso em que o agente está em uma grade de m linhas por n colunas ($m \times n$). O agente está inicialmente na posição de coordenada $(1, n)$, e tem como meta atravessar a rua (que é representada pela sub-grade que vai da posição $(2, 1)$ até $(m - 1, n)$), e chegar ao destino, que é dado pela coordenada (m, n). Contudo, o agente deve evitar atravessar a rua diretamente e buscar um percurso mais seguro porque obstáculos que representam veículos vêm da coluna mais à direita para a coluna mais à esquerda e podem atropelar esse agente. O atropelamento representa o pior estado do MDP, em que todas as ações passam a ter recompensas negativas.

Figura 7.1: *Visualização no RDDLSim de um estado em uma instância do domínio Crossing Traffic.*

A Figura 7.1 apresenta o visualizador do RDDLSim para o domínio *Crossing Traffic* em uma instância com 32 variáveis de estado, que neste caso são 4×4 para as possíveis posições do agente e 4×4 para as possíveis posições de obstáculos. Neste exemplo, o agente é representado pelo quadrado azul, os obstáculos são representados pelos quadrados verdes e o estado meta é representado pela letra 'G'.

7.1.2 Elevators

No domínio *Elevators*, que também é considerado esparso, o objetivo do agente é minimizar o tempo que as pessoas esperam por elevadores em um prédio. Com isso, o agente deve decidir entre ações como fazer os elevadores subirem ou descerem de um andar para o outro, além de abrir e fechar as portas do elevador.

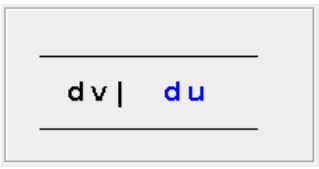


Figura 7.2: *Visualização no RDDLSim de um estado em uma instância do domínio Elevators.*

A Figura 7.2 apresenta o visualizador do RDDLSim para o domínio *Elevators* em uma instância com 13 variáveis de estado. A instância possui 1 elevador e 3 andares, sendo que as variáveis de estado do MDP são definidas por propriedades relacionadas ao elevador e aos andares como: se a porta do elevador está aberta ou fechada, e se existem pessoas esperando em um dado andar. O estado visualizado inclui:

- um elevador no primeiro andar e com a porta fechada '|';
- esse mesmo elevador está descendo 'v';
- duas pessoas querendo descer para o térreo 'd' (o 'd' à esquerda da porta representa que a pessoa está dentro do elevador; enquanto o 'd' à direita representa que a pessoa está fora do elevador); e

- uma pessoa querendo subir para o segundo andar representada pela letra 'u' à direita, i.e., fora do elevador.

7.1.3 Game of Life

O domínio *Game of Life*, baseado no jogo de mesmo nome proposto por John H. Conway em 1970 (Gardner, 1970), é representado em RDDL por uma grade $x \times y$ contendo células biológicas que podem estar vivas ou mortas de acordo com a vida existente nas células vizinhas. Em planejamento, diferentemente do que ocorre no *Game of Life* original, o agente pode interferir a qualquer momento dando vida a células específicas com o intuito de maximizar o número de células vivas na grade ao longo das gerações, i.e., diferentes instantes de tempo. Devido à possibilidade de interferência e dependência das células com suas vizinhas, *Game of Life* é um domínio denso.

```
TIME = 5:

XX.
X.X
.XX
```

Figura 7.3: *Visualização no RDDLSim de um estado em uma instância do domínio Game Of Life.*

A Figura 7.3 apresenta o visualizador do RDDLSim para o domínio *Game of Life* em uma instância com 9 variáveis de estado, i.e., uma grade 3×3. A igualdade 'TIME = 5' corresponde ao horizonte atual. Além disso, pontos (.) correspondem a células mortas; enquanto as letras xis (X) correspondem a células vivas. Desta forma, as posições $(1, 3), (2, 2)$ e $(3, 1)$ da grade têm células mortas; enquanto as outras estão vivas.

7.1.4 Navigation

O domínio *Navigation* é esparso e similar ao domínio *Crossing Traffic*. No estado inicial, o agente também está na posição $(1, n)$ e deve ir até a posição (m, n). Além disso, o agente não pode seguir diretamente pela coluna n porque desta forma, existe uma grande probabilidade de o agente desaparecer. Neste domínio, quanto mais distante o agente permanecer da coluna n (exceto nas linhas 1 e m), mais seguro o agente estará.

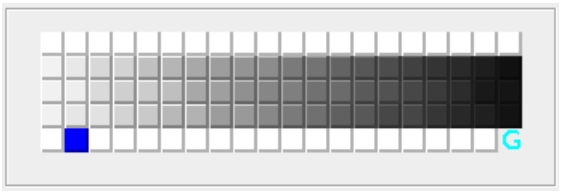


Figura 7.4: *Visualização no RDDLSim de um estado em uma instância do domínio Navigation.*

A Figura 7.4 apresenta o visualizador do RDDLSim para o domínio *Navigation* em uma instância com 100 variáveis de estado. Assim como em *Crossing Traffic*, o agente é representado pelo quadrado azul. Os quadrados de cor branca representam posições seguras da grade. Por outro lado, os quadrados em tons de cinza são as posições nas quais o agente tem maior probabilidade de desaparecer (quanto mais escuro, maior é essa probabilidade). A letra 'G' representa o estado meta.

7.1.5 Skill Teaching

No domínio *Skill Teaching*, que é considerado esparso, o objetivo do agente é ensinar disciplinas para um estudante. O ensino ocorre por meio de dicas e questões de múltipla escolha. Além disso, algumas disciplinas possuem pré-requisitos. O agente deve maximizar a pontuação do estudante planejando quais disciplinas ensinar e quando ensinar. Para otimizar o ensino, o agente deve considerar

quando dar dicas ou usar questões de múltipla escolha. Esse domínio não possui um visualizador específico no RDDLSim.

7.1.6 SysAdmin

No domínio *SysAdmin*, que é considerado denso, o agente é um administrador de uma rede com n computadores. Nessa rede, cada computador é representado por uma variável de estado e o agente deve manter o maior número de computadores ligados usando ações que reiniciam cada um dos n computadores. O agente deve considerar se outros computadores estão ligados antes de reiniciar um computador da rede e recebe maiores recompensas se houver muitos computadores ligados.

```
TIME = 4: ...X.
```

Figura 7.5: *Visualização no RDDLSim de um estado em uma instância do domínio SysAdmin.*

A Figura 7.5 apresenta o visualizador do RDDLSim para o domínio *SysAdmin* em uma instância com 5 variáveis de estado. A igualdade 'TIME = 4' corresponde ao horizonte atual. Além disso, nesse visualizador, os pontos (.) representam computadores ligados; enquanto as letras xis (X), representam computadores desligados. Portanto, de acordo com o exemplo, o computador 4 é o único desligado.

7.1.7 Instâncias dos domínios analisados

As instâncias usadas neste trabalho estão brevemente descritas na Tabela 7.1. Os campos dessa tabela são: Dom-Inst - nome do domínio e número da instância; $|X|$ - número de variáveis de estado; $|S|$ - número máximo de estados; $|S_{alc}|$ - número de estados alcançáveis a partir de s_0; e $|A|$ - número de ações.

Dom - Inst	$\|X\|$	$\|S\|$	$\|S_{alc}\|$	$\|A\|$
Crossing Traffic - 1	18	2^{18}	80	5
Crossing Traffic - 2	18	2^{18}	80	5
Crossing Traffic - 3	32	2^{32}	4312	5
Crossing Traffic - 4	32	2^{32}	4312	5
Elevators - 1	13	2^{13}	114	5
Elevators - 2	20	2^{20}	5184	25
Elevators - 3	20	2^{20}	5184	25
Elevators - 4	16	2^{16}	832	5
Elevators - 5	24	2^{24}	43264	25
Elevators - 6	24	2^{24}	43264	25
Elevators - 7	19	2^{19}	4352	5
Elevators - 10	22	2^{22}	21504	5
Game of Life - 1	9	2^{9}	2^{9}	10
Game of Life - 2	9	2^{9}	2^{9}	10
Game of Life - 3	9	2^{9}	2^{9}	10
Game of Life - 4	16	2^{16}	2^{16}	17
Game of Life - 5	16	2^{16}	2^{16}	17
Game of Life - 6	16	2^{16}	2^{16}	17
Navigation - 1	12	2^{12}	13	5
Navigation - 2	15	2^{15}	16	5
Navigation - 3	20	2^{20}	21	5
Navigation - 4	30	2^{30}	31	5
Navigation - 5	30	2^{30}	31	5
Navigation - 6	40	2^{40}	41	5
Navigation - 7	50	2^{50}	51	5
Navigation - 8	60	2^{60}	61	5
Navigation - 9	80	2^{80}	81	5
Navigation - 10	100	2^{100}	101	5
Skill Teaching - 1	12	2^{12}	63	5
Skill Teaching - 2	12	2^{12}	63	5
Skill Teaching - 3	24	2^{24}	1053	9
Skill Teaching - 4	24	2^{24}	1053	9
Skill Teaching - 5	36	2^{36}	13851	13
Skill Teaching - 6	36	2^{36}	13851	13
SysAdmin - 1	10	2^{10}	2^{10}	11
SysAdmin - 2	10	2^{10}	2^{10}	11

Tabela 7.1: *Instâncias dos domínios da IPPC-2011 analisadas neste trabalho.*

7.2 Análise Experimental

7.2.1 Comparação entre ReachMRFS e MRFS

Esta seção apresenta os resultados empíricos da comparação entre os algoritmos para computação da bissimulação estocástica sobre o conjuntos de estados completo (MRFS) e sobre os estados alcançáveis a partir de um estado inicial s_0 (ReachMRFS).

A Figura 7.6 mostra o tempo médio (considerando 30 execuções de cada algoritmo) para reduzir cada instância (MDP M) dos domínios Crossing Traffic, Elevators e Skill Teaching, obtendo um modelo reduzido M' e resolvendo-o com o LRTDP. Note que foi possível reduzir e resolver mais instâncias com o algoritmo ReachMRFS. Além disso, para os problemas resolvidos por ambos, ReachMRFS foi mais rápido porque: (1) esse algoritmo gasta menos tempo para reduzir os

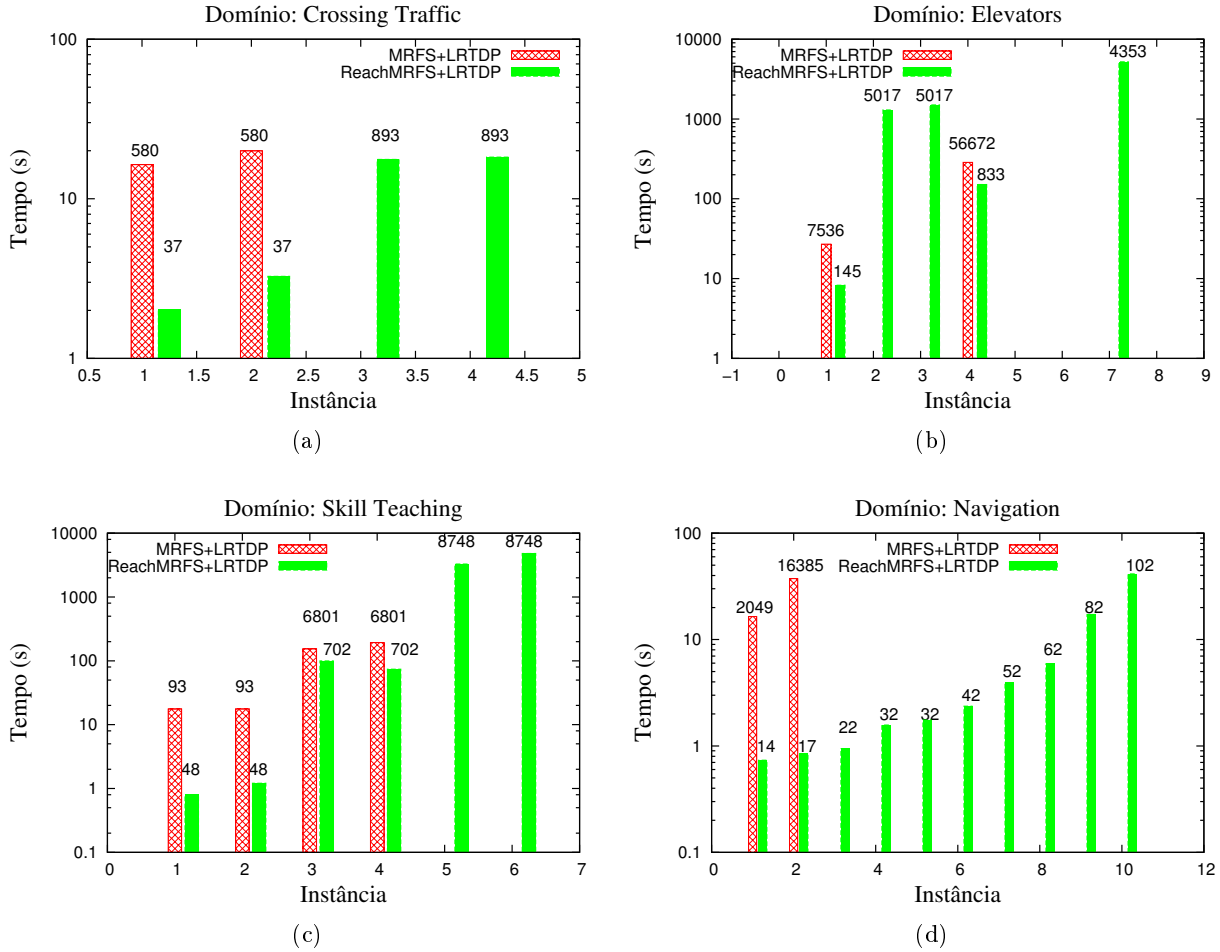


Figura 7.6: *Comparação entre ReachMRFS e MRFS nos domínios Crossing Traffic, Elevators, Skill Teaching e Navigation. Cada barra indica o tempo para reduzir e resolver uma instância reduzida com o LRTDP. Os números sobre as barras indicam $|S'|$ obtido pelo algoritmo de redução.*

problemas e; (2) os MDPs reduzidos obtidos são menores. No domínio *Elevators*, as instâncias 5-6 e 10 também puderam ser resolvidas, mas não são mostradas no gráfico devido aos longos tempos de processamento. Para essas instâncias somente o algoritmo ReachMRFS foi capaz de obter um modelo reduzido que pudesse ser resolvido (Tabela 7.2).

Domínio-Instância	**|S|**	**|S'|**	**Tempo para reduzir e resolver M'**
elevators-5	2^{24}	41793	$\approx$ 10 horas
elevators-6	2^{24}	41793	$\approx$ 8 horas e 30 minutos
elevators-10	2^{22}	21505	$\approx$ 4 dias e 12 horas

Tabela 7.2: *Desempenho de ReachMRFS em instâncias difíceis do domínio Elevators.*

Para o domínio *Navigation* (Figura 7.6(d)), os resultados foram ainda melhores: o algoritmo LRTDP não foi capaz de resolver as instâncias 3-10 quando os problemas foram reduzidos pelo MRFS; apenas quando reduzidos com ReachMRFS. O motivo que fez ReachMRFS ter melhor desempenho nos problemas apresentados foi porque esses problemas têm matrizes de transição esparsas.

O pior caso de MRFS e ReachMRFS foi nos domínios com matrizes de transição densas. Na IPPC-2011, os domínios com esse tipo de matriz são *Game of Life* e *SysAdmin*. Nesses domínios não há vantagem em usar análise de alcançabilidade porque $S = S_{alcancaveis}$ e por essa razão, as reduções obtidas por MRFS e ReachMRFS foram as mesmas. O pior dos cenários apareceu no

domínio *SysAdmin*, em que não houve redução alguma.

7.2.2 Redução de modelos: eliminação de partições repetidas no ReachMRFS

Como foi visto no Capítulo 6, um MDP pode ter muitas partições repetidas. Note que, eliminar partições repetidas é possível apenas para os algoritmos MRFS e ReachMRFS. Isso porque os algoritmos que juntam blocos (de forma exata ou aproximada), algoritmos MMFS (Seção 4.4) e ϵMRFS (Seção 5.2), precisam de todas as partições para computar a probabilidade conjunta.

Nessa seção mostramos que a eliminação de partições repetidas pelo algoritmo proposto ReachMRFS pode tornar a redução de modelos muito mais rápida.

A Figura 7.7 apresenta a análise de tempo e do número de partições usadas na redução de modelos sobre estados alcançáveis considerando partições distintas (algoritmo ReachMRFS-V2) para os domínios *Elevators*, *Skill Teaching* e *Navigation*. Os resultados mostram que considerar apenas partições distintas (ReachMRFS-V2) durante a redução de modelos implica em um aumento significativo de desempenho.

Note que, para o domínio *Skill Teaching*, as instâncias 5 e 6 provocam estouro de memória quando são usadas as partições repetidas durante a computação da bissimulação estocástica sobre $S_{alcancaveis}$ (algoritmo ReachMRFS-V1).

No domínio *Navigation*, a diferença entre as quantidades de partições utilizadas nas diferentes versões de ReachMRFS não é tão grande conforme pode ser visto pelos números da Figura 7.7. E também não é possível observar um grande impacto em desempenho como nos outros domínios. Apesar disso, ReachMRFS sem partições repetidas ainda é mais eficiente, o que mostra que evitar refinamentos excessivos é uma boa estratégia mesmo em domínios com poucas partições repetidas.

7.3 Comparação entre os três algoritmos propostos de redução com alcançabilidade: ReachMRFS, ReachMMFS e Reach-ϵMRFS

7.3.1 Tempo de redução e tamanho dos modelos reduzidos

A Tabela 7.3 apresenta os tamanhos dos MDPs e BMDPs obtidos pelos algoritmos ReachMRFS-V2, ReachMMFS e Reach-ϵMRFS para os problemas da IPPC-2011. Os campos da tabela são: DomInst - nome do domínio e número da instância; $|S|$ - o número máximo de estados; $|S|_{alc}$ - número de estados alcançáveis dado s_0; $|S'|$ - tamanho do conjunto de estados do MDP reduzido/minimal; $\%|S'|$ - percentual de redução/minimização do MDP; T_s - tempo médio em segundos para encontrar $\mathcal{P}_{DD}^{S|s_0}$ e executar MRFS/MMFS sobre $B_{alcancaveis|s_0}$; e $\epsilon \in \{0.1, 0.2, 0.3\}$ - tamanho do conjunto de estados após ϵ-redução sobre estados alcançáveis. A sigla NT significa 'Não Terminou' e isso pode ser causado por estouros de memória ou tempos maiores que 12 horas para reduzir o problema. É possível observar que em todos os domínios escolhidos, exceto em *Crossing Traffic*, não há vantagens em utilizar ReachMMFS. E ainda assim, a vantagem de reduzir mais os problemas de *Crossing Traffic* não é tão grande. Além disso, ao observar o algoritmo Reach-ϵMRFS temos uma maior redução no tamanho de $|S'|$ quando ϵ aumenta, mas o algoritmo é tão ineficiente quanto ReachMMFS.

Com base nessas observações, a conclusão é que ao reduzir os MDPs de forma exata, a utilização da redução de modelos é melhor porque: (1) a minimização de modelos nem sempre gera MDPs com menos estados do que a redução de modelos; e (2) a minimização de modelos gasta muito tempo de processamento durante a etapa de junção de blocos (especialmente quando $|\mathcal{P}|$ é grande e o problema tem matriz de transição densa).

Além de não reduzir mais os problemas na maioria dos experimentos, ReachMMFS consome mais tempo que ReachMRFS porque é necessário calcular a probabilidade conjunta entre todos os blocos para verificar quais blocos podem ser reagrupados. Desta forma, em domínios em que as matrizes de transição são densas, verificar a estabilidade entre blocos pode ser difícil mesmo em problemas considerados pequenos, por exemplo, $|S| = 2^{10}$.

Dom-Inst	$\|S\|$	$\|S\|_{alc}$	Alcançabilidade e Redução (ReachMRFS)			Alcançabilidade e Minimização (ReachMMFS)			Alcançabilidade e ϵ-Redução (Reach-ϵMRFS - apenas $\|S'\|$)		
			$\|S'\|$	$\%\|S'\|$	T_s	$\|S'\|$	$\%\|S'\|$	T_s	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$
Crossing Traffic-1	2^{18}	80	37	$\approx 99.9859\%$	0.52	26	$\approx 99.9901\%$	0.89	26	26	26
Crossing Traffic-2	2^{18}	80	37	$\approx 99.9859\%$	0.55	26	$\approx 99.9901\%$	0.68	26	26	26
Crossing Traffic-3	2^{32}	4312	893	$\approx 100\%$	5.06	365	$\approx 100\%$	58.36	365	365	365
Crossing Traffic-4	2^{32}	4312	893	$\approx 100\%$	5.11	365	$\approx 100\%$	57.47	365	365	365
Elevators-1	2^{13}	144	145	$\approx 98.23\%$	0.49	145	$\approx 98.23\%$	2.68	121	117	117
Elevators-2	2^{20}	5184	5017	$\approx 99.5215\%$	13.15	NT	NT	NT	NT	NT	NT
Elevators-3	2^{20}	5184	5017	$\approx 99.5215\%$	9.27	NT	NT	NT	NT	NT	NT
Elevators-4	2^{16}	832	833	$\approx 98.7289\%$	1.43	833	$\approx 98.7289\%$	17.46	701	685	661
Elevators-5	2^{24}	43264	41793	$\approx 99.7509\%$	755.83	NT	NT	NT	NT	NT	NT
Elevators-6	2^{24}	43264	41793	$\approx 99.7509\%$	811.27	NT	NT	NT	NT	NT	NT
Elevators-7	2^{19}	4352	4353	$\approx 99.1697\%$	9.35	NT	NT	NT	NT	NT	NT
Elevators-10	2^{22}	21504	21505	$\approx 99.4873\%$	195.02	NT	NT	NT	NT	NT	NT
Game Of Life-1	2^{9}	2^{9}	253	$\approx 50.5859\%$	0.18	253	$\approx 50.5859\%$	124.29	252	236	215
Game Of Life-2	2^{9}	2^{9}	253	$\approx 50.5859\%$	0.20	253	$\approx 50.5859\%$	110.62	252	236	220
Game Of Life-3	2^{9}	2^{9}	253	$\approx 50.5859\%$	0.18	253	$\approx 50.5859\%$	105.43	252	236	201
Game Of Life-4	2^{16}	2^{16}	20929	$\approx 68.0649\%$	104.37	NT	NT	NT	NT	NT	NT
Game Of Life-5	2^{16}	2^{16}	20929	$\approx 68.0649\%$	107.66	NT	NT	NT	NT	NT	NT
Game Of Life-6	2^{16}	2^{16}	20929	$\approx 68.0649\%$	96.06	NT	NT	NT	NT	NT	NT
Navigation-1	2^{12}	13	14	$\approx 99.6582\%$	0.29	14	$\approx 99.6582\%$	0.41	14	14	14
Navigation-2	2^{15}	16	17	$\approx 99.9481\%$	0.25	17	$\approx 99.9481\%$	0.39	17	17	17
Navigation-3	2^{20}	21	22	$\approx 99.9995\%$	0.47	22	$\approx 99.9995\%$	0.97	22	22	22
Navigation-4	2^{30}	31	32	$\approx 100\%$	0.93	32	$\approx 100\%$	1.43	32	32	32
Navigation-5	2^{30}	31	32	$\approx 100\%$	0.96	32	$\approx 100\%$	1.36	32	32	32
Navigation-6	2^{40}	41	42	$\approx 100\%$	1.42	42	$\approx 100\%$	2.77	42	42	42
Navigation-7	2^{50}	51	52	$\approx 100\%$	2.56	52	$\approx 100\%$	6.27	52	52	52
Navigation-8	2^{60}	61	62	$\approx 100\%$	4.08	62	$\approx 100\%$	5.07	62	62	62
Navigation-9	2^{80}	81	82	$\approx 100\%$	13.35	82	$\approx 100\%$	17.09	82	82	82
Navigation-10	2^{100}	101	102	$\approx 100\%$	33.24	102	$\approx 100\%$	39.92	102	102	102
Skill Teaching-1	2^{12}	63	48	$\approx 98.8281\%$	0.52	48	$\approx 98.8281\%$	1.27	35	35	27
Skill Teaching-2	2^{12}	63	48	$\approx 98.8281\%$	0.42	48	$\approx 98.8281\%$	1.36	35	35	27
Skill Teaching-3	2^{24}	1053	702	$\approx 99.9958\%$	3.15	702	$\approx 99.9958\%$	48.75	367	299	252
Skill Teaching-4	2^{24}	1053	702	$\approx 99.9958\%$	3.50	702	$\approx 99.9958\%$	54.18	416	299	254
Skill Teaching-5	2^{36}	13851	8748	$\approx 100\%$	110.08	NT	NT	NT	NT	NT	NT
Skill Teaching-6	2^{36}	13851	8748	$\approx 100\%$	140.99	NT	NT	NT	NT	NT	NT
SysAdmin-1	2^{10}	2^{10}	2^{10}	0%	0.17	NT	NT	NT	NT	NT	NT
SysAdmin-2	2^{10}	2^{10}	2^{10}	0%	0.15	NT	NT	NT	NT	NT	NT

Tabela 7.3: *Tamanhos dos MDPs obtidos por ReachMRFS e ReachMMFS.*

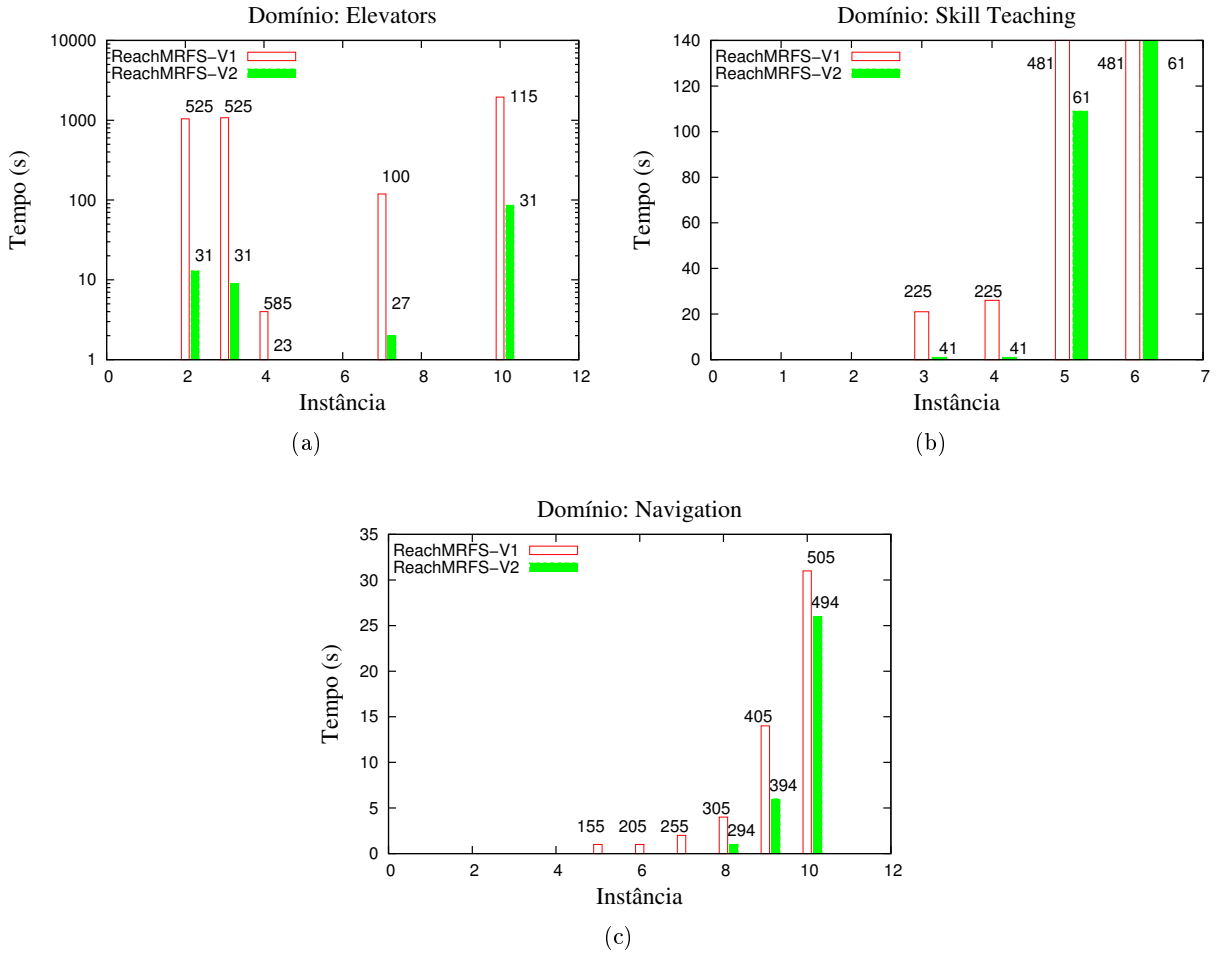


Figura 7.7: *Comparação entre tempo gasto para reduzir MDPs com os algoritmos ReachMRFS-V1 e ReachMRFS-V2 nos domínios Elevators, Skill Teaching e Navigation. Os números acima das barras indicam o número de partições usadas pelos algoritmos.*

7.3.2 Tempo total (redução e solução) e qualidade das soluções aproximadas

Existe um compromisso entre tempo e tamanho das partições obtidas por ReachMRFS, ReachMMFS e Reach-ϵMRFS. Apesar de ReachMRFS não garantir que computa a menor partição homogênea possível sobre $S_{alcancaveis}$, a partição é computada rapidamente. Por outro lado, ReachMMFS garante que devolve a menor partição homogênea possível sobre $S_{alcancaveis}$, mas não é no menor tempo. Além disso, são frequentes os casos em que as partições obtidas por ReachMMFS são iguais àquelas calculadas por ReachMRFS (Tabela 7.3). Enfim, Reach-ϵMRFS pode obter partições aproximadas ainda menores do que aquelas obtidas por ReachMMFS, mas a partição resultante tem pior qualidade devido às aproximações realizadas e o modelo resultante é um BMDP, que é mais difícil de resolver.

Análise de tempo total

A comparação apresentada nesta seção é baseada nos tempos e porcentagens de redução do conjunto de estados para os domínios *Game of Life* e *Skill Teaching*, que são mais beneficiados pelo valor de ϵ. A implementação de Reach-ϵMRFS se baseia em dividir os blocos com SSPLIT e depois juntar blocos que foram divididos de forma exata usando $\epsilon \in \{0.1, 0.2, 0.3\}$, que são valores para os quais o BMDP não é tão diferente do MDP original (Tabela 7.3). O tempo total, por sua vez, é dado pela soma de tempo para reduzir e resolver cada instância.

Com base nos experimentos para os dois domínios (Figura 7.8), é possível perceber que Rea-

chMRFS e ReachMMFS geram MDPs iguais, mas ReachMRFS permite a computação da solução ótima em menos tempo porque gera uma bissimulação estocástica exata sobre os estados alcançáveis sem usar partições repetidas e sem juntar blocos. Por outro lado, ReachMMFS tende a gastar mais tempo porque não elimina partições repetidas e precisa realizar a junção de blocos. Ao comparar Reach-ϵMRFS com ReachMMFS, Reach-ϵMRFS tende a ser mais eficiente por utilizar partições menores durante o processo de redução e junção, especialmente com valores maiores para ϵ. Finalmente, ao comparar ReachMRFS e Reach-ϵMRFS, é possível concluir que com $\epsilon > 0.3$, Reach-ϵMRFS tende a se aproximar de ReachMRFS. Contudo, Reach-ϵMRFS ainda é pior que ReachMRFS porque consome mais memória na etapa de junção de blocos, que não é realizada em ReachMRFS e além disso, precisa de um ϵ grande para gerar um BMDP reduzido, mas que pode ser bastante impreciso. Dessa forma, concluímos que ReachMRFS é a forma mais eficiente de se agregar estados estudada neste trabalho.

Apesar de ReachMRFS não garantir que gera o menor MDP equivalente ao MDP original, esse algoritmo já reduz muito os problemas, obtendo reduções maiores do que 98% na maioria dos casos (Tabela 7.3). Além disso, é uma forma exata de se reduzir os problemas, diferentemente de Reach-ϵMRFS que é aproximada e começa a ser melhor apenas quando ϵ já é grande ($\epsilon \geq 0.3$). Contudo, quando ϵ é grande, Reach-ϵMRFS gera partições que muitas vezes não são aproximadamente equivalentes ao MDP original e políticas bastante ruins são obtidas a partir do BMDP resultante.

Análise de qualidade das soluções do modelo aproximado

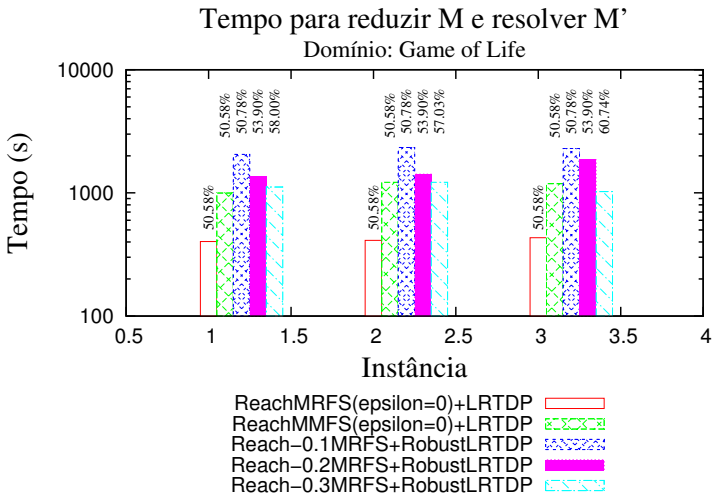
Após aplicar o algoritmo Reach-ϵMRFS sobre o MDP original, o BMDP pode ser resolvido utilizando um critério pessimista. Desta forma, considerando a utilização do LRTDP Robusto, a solução é uma política ótima pessimista π^*_{pes}, cujo valor é dado por V^*_{pes}. Esse valor normalmente é diferente do valor ótimo V^* computado diretamente com base no MDP original. Por esse motivo, é necessário algum critério para definir uma porcentagem de erro de V^*_{pes} se comparado com V^*.

Neste trabalho, a porcentagem de erro de V^*_{pes} se comparado com V^* é computada com base nos valores ótimos e robustos após terem convergido no estado inicial s_0. Nos problemas analisados, s_0 é único e contém informação de todos os outros estados convergidos. Com isso, a porcentagem de erro é calculada da seguinte maneira:

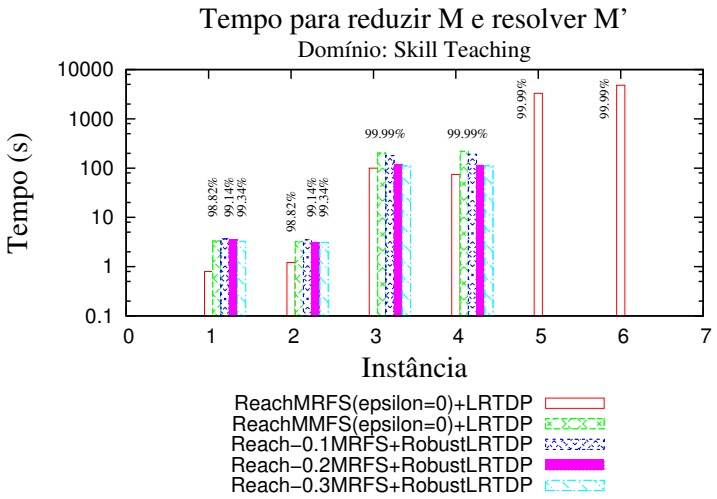
$$\frac{|V^*(s_0) - V^*_{pes}(s_0)|}{|V^*(s_0)|} \times 100. \tag{7.1}$$

A Figura 7.9 apresenta a porcentagem de erro computada em s_0 nas três primeiras instâncias do domínio *Game of Life*. Com base nos experimentos, quanto maior o valor de ϵ (e porcentagem de redução) no Reach-ϵMRFS, maior o erro no valor da política π^*_{pes}, podendo chegar a quase 15% de erro na terceira instância usando $\epsilon = 0.3$.

Enquanto em *Game of Life* o erro pode crescer bastante, a situação no domínio *Skill Teaching* (Figura 7.10) é diferente. Ao utilizar $\epsilon = 0.3$ em *Skill Teaching*, o maior erro obtido a partir da solução aproximada foi menor do que 3%, o que é considerado muito pequeno se observarmos que a redução foi de aproximadamente 99.99%. Como foi mencionado anteriormente, Reach-ϵMRFS consome mais memória do que ReachMRFS, e isso inviabilizou experimentos com instâncias maiores usando a implementação aproximada nos dois domínios.



(a)



(b)

Figura 7.8: *Tempo gasto pelas implementações de redução, minimização, ϵ-redução de modelos sobre os estados alcançáveis somado com o tempo para resolver os problemas com LRTDP ou LRTDP Robusto nos domínios Game of Life e Skill Teaching. No caso da ϵ-redução de modelos, foram utilizados $\epsilon \in \{0.1, 0.2, 0.3\}$*

7.4 MDP original versus MDP reduzido: análise de tempo

Com base nos resultados anteriores podemos concluir que o algoritmo ReachMRFS é a melhor implementação para a computação de bissimulações estocásticas proposta neste trabalho. Nessa seção, queremos comparar o tempo gasto na solução original do MDP, com a solução do modelo reduzido sobre os estados alcançáveis. Para obter uma comparação justa, usaremos o algoritmo LRTDP para resolver ambos os modelos: o original e o reduzido por ReachMRFS.

No domínio *Crossing Traffic* (Figura 7.11(a)), ReachMRFS+LRTDP é mais lento em todas as instâncias se comparado com o próprio LRTDP sobre um MDP original M. Apesar disso, nas instâncias 3 e 4 desse domínio, ReachMRFS+LRTDP quase empata com o LRTDP que resolve M de forma direta. No domínio *Skill Teaching* (Figura 7.11(d)), os resultados são parecidos, mas ReachMRFS+LRTDP ainda consegue superar o LRTDP na última instância.

Por outro lado, o LRTDP é mais lento ao resolver o MDP M do que ao resolver o MDP M' obtido por ReachMRFS+LRTDP no domínio *Game of Life* (Figura 7.11(b)). Infelizmente, por ser um domínio denso, *Game of Life* é mais complexo e apenas 3 instâncias foram resolvidas com ambas as implementações.

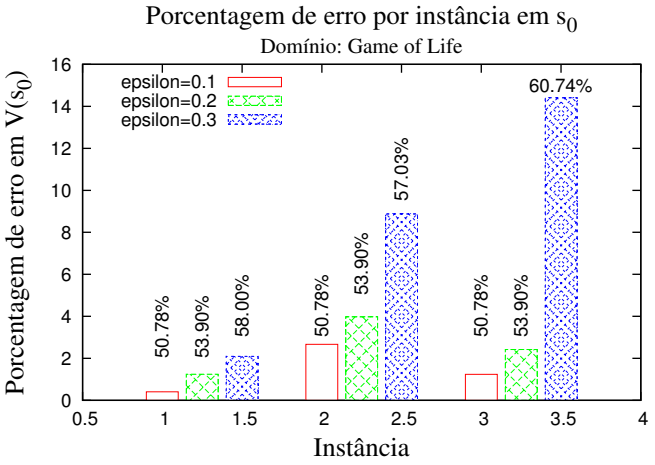


Figura 7.9: *Perda de qualidade com relação a três diferentes valores de ϵ nas primeiras instâncias de Game of Life. Os valores sobre as barras indicam a porcentagem de redução.*

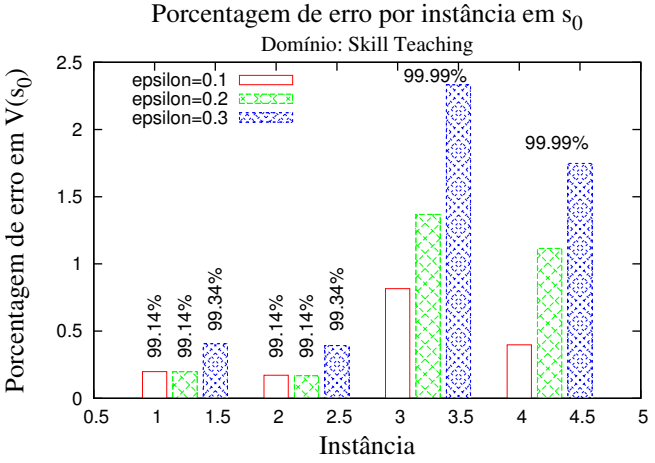


Figura 7.10: *Perda de qualidade com relação a três diferentes valores de ϵ nas primeiras instâncias de Skill Teaching. Os valores sobre as barras indicam a porcentagem de redução.*

No domínio *Navigation* (Figura 7.11(c)), ReachMRFS+LRTDP começa perdendo por pouco do LRTDP que resolve o MDP original M, mas a partir da instância 7 a vantagem do LRTDP começa a aumentar. Essa mudança ocorre porque à medida que $|X|$ aumenta, se torna mais difícil encontrar uma bissimulação estocástica sobre $S_{alcancaveis}$. Além disso, a redução nesse domínio é principalmente devido à análise de alcançabilidade, que é realizada de forma implícita no LRTDP através dos trials.

Com base nos experimentos dessa seção, podemos concluir que ReachMRFS pode tornar alguns problemas bem mais fáceis de se resolver. Contudo, existe um custo de se realizar essa redução. Ao somar o tempo de redução e o tempo para resolver o MDP reduzido, foi observado que ReachMRFS obtém vantagem quando o problema é complexo e a partição homogênea sobre os estados alcançáveis é menor que o conjunto de estados alcançáveis.

7.5 Resumo do Capítulo

Nesse capítulo foram apresentados os seguintes conteúdos:

- domínios da IPPC-2011 usados para análise empírica;
- resultados da análise empírica que comparou técnicas tradicionais de bissimulação estocástica com as técnicas propostas neste trabalho; e

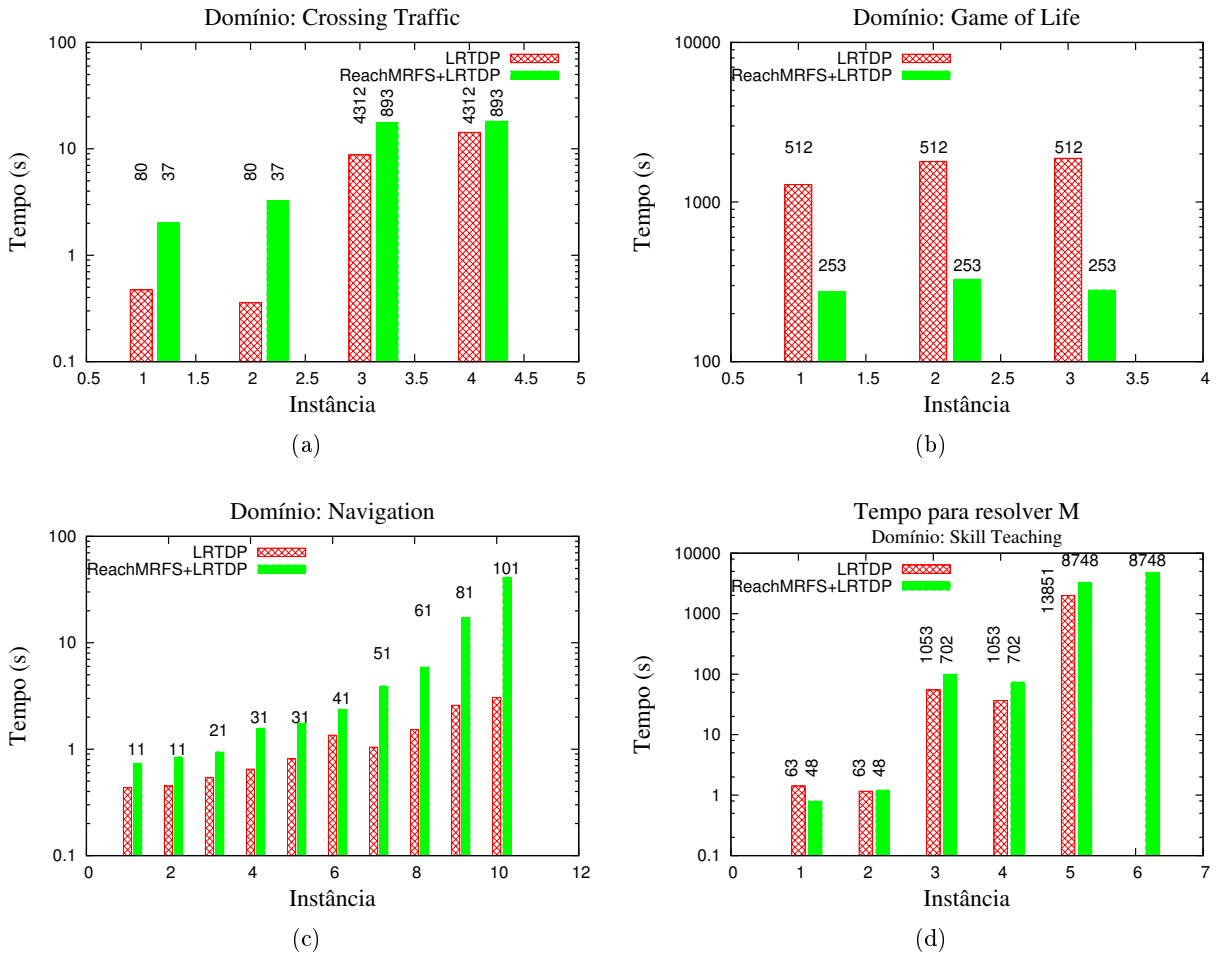


Figura 7.11: *Comparação entre ReachMRFS+LRTDP e LRTDP nos domínios Crossing Traffic, Game of Life, Navigation e Skill Teaching. Os números sobre as barras indicam o número de estados encontrados em cada solução.*

- resultado da comparação entre a melhor técnica de bissimulação estocástica proposta neste trabalho (ReachMRFS-V2+LRTDP) com o LRTDP aplicado diretamente sobre o MDP original.

Capítulo 8

Conclusões e Trabalhos Futuros

Encontrar políticas ótimas para MDPs considerando os estados relevantes a partir de um estado inicial s_0 é uma abordagem que tem sido exaustivamente explorada pela comunidade de planejamento probabilístico. Neste trabalho, é utilizada a informação de estados alcançáveis durante o processo de encontrar uma bissimulação estocástica, que pode ser exata ou aproximada. Os resultados empíricos mostram que a partição por alcançabilidade permite: (1) uma computação mais eficiente do modelo reduzido; (2) uma representação das partições de forma mais compacta durante a computação da bissimulação estocástica e consequentemente um modelo abstrato menor; e (3) a resolução de problemas com um grande número de estados.

Além dos resultados já mencionados, foi observado que o algoritmo ReachMRFS (Reachability-based Model Reduction with Factored Splits) possibilita a resolução de problemas maiores, mesmo reduzindo relativamente menos que os algoritmos ReachMMFS (Reachability-based Model Minimization with Factored Splits) e Reach-ϵMRFS (Reachability-based ϵ-Model Reduction with Factored Splits). Isso acontece porque a computação de ReachMRFS é mais simples e permite relaxar algumas etapas da computação, diferentemente dos outros dois algoritmos. Dentre as etapas que podem ser otimizadas em ReachMRFS estão: eliminação de partições repetidas (que não pode ser feito em ReachMMFS) e determinação da função de transição entre estados abstratos. Além dessas vantagens, os modelos obtidos com ReachMRFS são MDPs, que são mais fáceis de resolver do que os BMDPs obtidos pelo Reach-ϵMRFS.

Com base nas observações anteriores, chegamos a conclusão que houveram melhorias nos algoritmos de agregação de estados que permitem encontrar bissimulações estocásticas exatas ou aproximadas considerando estados alcançáveis, sendo essa a principal contribuição desse trabalho.

8.1 Trabalhos correlatos

Duas técnicas similares às ideias apresentadas neste trabalho são Symbolic RTDP (sRTDP) (Feng *et al.*, 2003) e análise de alcançabilidade estruturada para MDPs (Boutilier *et al.*, 1998). O sRTDP resolve MDPs exatos usando representações fatoradas e realizando reduções dinamicamente, i.e., enquanto encontra uma política ótima para o MDP. Contudo, as atualizações de Bellman no sRTDP são muito custosas (Feng *et al.*, 2003). A segunda técnica, apresentada em (Boutilier *et al.*, 1998), executa uma análise de alcançabilidade estruturada antes de resolver o MDP. Depois disso, utiliza essa informação para simplificar as DBNs do MDP fatorado. Finalmente, resolve o MDP fatorado de forma aproximada.

8.2 Publicações

Com base neste trabalho foi gerado um artigo que apresenta as melhorias desenvolvidas para o algoritmo de redução de modelos com refinamentos fatorados sobre estados alcançáveis (ReachMRFS-V2). O artigo foi apresentado na BRACIS 2013 (*Brazilian Conference on Intelligent Systems*).

8.3 Trabalhos Futuros

Este trabalho pode ser estendido de algumas formas:

- **Utilização de heurísticas para a remoção de ações do MDP de origem.** Durante a computação da bissimulação estocástica (exata e aproximada) são realizados vários refinamentos com base nas partições obtidas das diferentes ações do MDP. Contudo, o número de blocos da partição resultante tende a aumentar com cada refinamento que é computado. Isso pode se tornar um problema porque a partição resultante pode ter muitos blocos e consequentemente, será muito custoso representar computacionalmente essa partição e realizar operações com ela. Obter um subconjunto $A' \subseteq A$ de ações mais relevantes pode permitir que estouros de memória sejam evitados. Mas com isso, no modelo reduzido, o mesmo subconjunto A' deve ser utilizado para encontrar uma política (não necessariamente ótima). A estratégia de parar os refinamentos antes de alcançar uma partição homogênea já foi estudada de algumas formas diferentes em planejamento determinístico e probabilístico (Dean *et al.*, 1997; Katz *et al.*, 2012; Kim e Dean, 2003).

- **Aperfeiçoar análise de alcançabilidade.** Existem diferentes formas de se representar os estados alcançáveis em um MDP. Neste trabalho, é obtida uma partição por alcançabilidade dada por um BDD em que as atribuições com valor 1 representam estados alcançáveis e as atribuições com valor 0, estados inalcançáveis. Uma outra maneira é criar restrições por meio de combinações de variáveis de estado para identificar estados que não podem ser alcançados (Boutilier *et al.*, 1998). Em alguns casos, essa representação pode ser mais compacta e permitir que a análise de alcançabilidade seja realizada em problemas ainda maiores. Como resultado, pode ser possível encontrar a bissimulação estocástica sobre os estados alcançáveis nesses problemas.

  Uma outra possibilidade, é não visitar todos os estados alcançáveis. Ao invés disso, visitar os estados alcançáveis dada uma profundidade máxima d. Apesar disso, o resultado pode ser ruim se houver estados com as melhores recompensas em uma profundidade além de d.

  É importante ressaltar que a análise de alcançabilidade auxiliou num melhor desempenho na computação de bissimulações estocásticas exatas e aproximadas deste trabalho, mas que otimizar a forma como essa análise é realizada pode permitir resultados ainda melhores em domínios como: *Crossing Traffic*, *Elevators* e *Skill Teaching*.

- **Utilizar técnicas de amostragem do Glutton.** O Glutton (Kolobov *et al.*, 2012) é uma implementação enumerativa otimizada do LRTDP que foi um dos destaques da IPPC-2011. Dentre as otimizações que o Glutton incorpora, está a *amostragem da função de transição*, i.e., quando um estado tem muitos sucessores, o Glutton obtém apenas um subconjunto desses estados para computar a política gulosa. Dessa forma, os trials do Glutton são muito mais rápidos e isso afeta pouco na qualidade da política obtida (Kolobov *et al.*, 2012). Com esse recurso, seria possível obter melhores resultados em domínios com matrizes de transição densas como *Game of Life* e *SysAdmin*.

- **Comparação de ReachMRFS+LRTDP com sRTDP.** O sRTDP é uma solução fatorada para MDPs que pode gerar estados abstratos dinamicamente de duas formas considerando: (1) estados que têm valores similares; ou (2) estados que alcançam um mesmo subconjunto de estados. Devido às semelhanças entre sRTDP e ReachMRFS+LRTDP, a comparação entre ambos pode apresentar vantagens e desvantagens das diferentes formas de agregação de estados.

Apêndice A

Notação Assintótica

A notação assintótica (Cormen *et al.*, 2009) é uma das ferramentas da análise de algoritmos para medir de forma teórica a complexidade de algoritmos quando a entrada tende a assumir valores muito grandes. Para fazer isso, a notação assintótica define que algoritmos têm complexidades medidas em cada instrução por meio de funções que podem ser: constantes, logarítmicas, lineares, polinomiais, etc. Neste trabalho, a análise assintótica é usada por meio da notação O (O Grande), que define a complexidade de pior caso para um dado algoritmo com base nos parâmetros de entrada.

Seja A um algoritmo e n o tamanho de uma entrada para A. Com a notação O, é possível dizer que, $A(n) = O(f(n))$, i.e., o algoritmo A tem complexidade de pior caso dada por uma função $f(n)$. Em outras palavras, a complexidade de A é limitada superiormente por uma função de n. Para afirmar que $A(n) = O(f(n))$, é necessário encontrar constantes c e n_0 com as quais seja possível demonstrar que $A(n) \leq c \times f(n)$ para todo $n \geq n_0$.

Existem outras notações que são utilizadas para analisar a complexidade dos algoritmos, mas neste trabalho apenas a notação O é usada.

Apêndice B

Domínios especificados em RDDL

Este apêndice apresenta os 6 domínios utilizados neste trabalho. Os domínios são apresentados na linguagem RDDL (*Relational Dynamic Influence Diagram Language*) (Sanner, 2010). Além deles, é apresentado também o domínio e problema utilizado no artigo de ϵ-redução de modelos (Dean *et al.*, 1997).

B.1 Crossing Traffic

```
1   domain crossing_traffic_mdp {
2     requirements = {
3       reward-deterministic
4     };
5
6     types {
7       xpos : object;
8       ypos : object;
9     };
10
11    pvariables {
12
13      NORTH(ypos, ypos) : {non-fluent, bool, default = false};
14      SOUTH(ypos, ypos) : {non-fluent, bool, default = false};
15      EAST(xpos, xpos)  : {non-fluent, bool, default = false};
16      WEST(xpos, xpos)  : {non-fluent, bool, default = false};
17
18      MIN-XPOS(xpos) : {non-fluent, bool, default = false};
19      MAX-XPOS(xpos) : {non-fluent, bool, default = false};
20      MIN-YPOS(ypos) : {non-fluent, bool, default = false};
21      MAX-YPOS(ypos) : {non-fluent, bool, default = false};
22
23      INPUT-RATE : {non-fluent, real, default = 0.2};
24
25      GOAL(xpos,ypos) : {non-fluent, bool, default = false};
26
27      // Fluents
28      robot-at(xpos, ypos)    : {state-fluent, bool, default = false};
29      obstacle-at(xpos, ypos) : {state-fluent, bool, default = false};
30
31      // Actions
32      move-north : {action-fluent, bool, default = false};
33      move-south : {action-fluent, bool, default = false};
34      move-east  : {action-fluent, bool, default = false};
35      move-west  : {action-fluent, bool, default = false};
36    };
37
38    cpfs {
39
40      robot-at'(?x,?y) =
41
42        // Goal is absorbing so robot stays put
43        if ( GOAL(?x,?y) ^ robot-at(?x,?y) )
44        then
45          KronDelta(true)
```

```
46          else if ( exists_{?x2 : xpos, ?y2 : ypos} [ GOAL(?x2,?y2) ^ robot-at(?x2,?y2)
                 ] )
47          then
48            KronDelta(false) // because of fall-through we know (?x,y) != (?x2,?y2)
49
50          // Check for legal robot movement (robot disappears if at an obstacle)
51          else if ( move-north ^ exists_{?y2 : ypos} [ NORTH(?y2,?y) ^ robot-at(?x,?y2)
                ^ ~obstacle-at(?x,?y2) ] )
52          then
53            KronDelta(true) // robot moves to this location
54          else if ( move-north ^ exists_{?y2 : ypos} [ NORTH(?y,?y2) ^ robot-at(?x,?y)
                ] )
55          then
56            KronDelta(false) // robot leaves this location
57          else if ( move-south ^ exists_{?y2 : ypos} [ SOUTH(?y2,?y) ^ robot-at(?x,?y2)
                ^ ~obstacle-at(?x,?y2) ] )
58          then
59            KronDelta(true) // robot moves to this location
60          else if ( move-south ^ exists_{?y2 : ypos} [ SOUTH(?y,?y2) ^ robot-at(?x,?y)
                ] )
61          then
62            KronDelta(false) // robot leaves this location
63          else if ( move-east ^ exists_{?x2 : xpos} [ EAST(?x2,?x) ^ robot-at(?x2,?y) ^
                ~obstacle-at(?x2,?y) ] )
64          then
65            KronDelta(true) // robot moves to this location
66          else if ( move-east ^ exists_{?x2 : xpos} [ EAST(?x,?x2) ^ robot-at(?x,?y) ]
                )
67          then
68            KronDelta(false) // robot leaves this location
69          else if ( move-west ^ exists_{?x2 : xpos} [ WEST(?x2,?x) ^ robot-at(?x2,?y) ^
                ~obstacle-at(?x2,?y) ] )
70          then
71            KronDelta(true) // robot moves to this location
72          else if ( move-west ^ exists_{?x2 : xpos} [ WEST(?x,?x2) ^ robot-at(?x,?y) ]
                )
73          then
74            KronDelta(false) // robot leaves this location
75
76          // A noop or illegal movement, so state unchanged
77          else
78            KronDelta( robot-at(?x,?y) ^ ~obstacle-at(?x,?y) );
79
80        obstacle-at'(?x, ?y) =
81
82          // No obstacles in top or bottom row (these rows are safe havens)
83          if ( MIN-YPOS(?y) | MAX-YPOS(?y) )
84            then KronDelta( false )
85
86          // Check for RHS border input cell
87          else if ( MAX-XPOS(?x) )
88            then Bernoulli( INPUT-RATE )
89
90          // Not a top or bottom row and not a border input cell -- inherits obstacle
                to east
91          else
92            KronDelta( exists_{?x2 : xpos} [EAST(?x,?x2) ^ obstacle-at(?x2,?y)] );
93
94        };
95
96        // 0 reward for reaching goal, -1 in all other cases
97        reward = [sum_{?x : xpos, ?y : ypos} -(GOAL(?x,?y) ^ ~robot-at(?x,?y))];
98      }
```

B.2 Elevators

```
domain elevators_mdp {

  requirements = {
    constrained-state,
    reward-deterministic
  };

  types {
    elevator : object;
    floor    : object;
  };

  pvariables {

    // Probability someone arrives at the floor (up or down)
    ARRIVE-PARAM(floor) : { non-fluent, real, default = 0.0 };

    // Penalty for persons in the elevator going in right/wrong direction
    // Note: a constant 1.0 penalty for people waiting at a floor
    ELEVATOR-PENALTY-RIGHT-DIR : { non-fluent, real, default = 0.75 };
    ELEVATOR-PENALTY-WRONG-DIR : { non-fluent, real, default = 3.00 };

    // Useful definitions
    TOP-FLOOR(floor)           : { non-fluent, bool, default = false };
    BOTTOM-FLOOR(floor)        : { non-fluent, bool, default = false };
    ADJACENT-UP(floor, floor)  : { non-fluent, bool, default = false };

    // Person waiting state
    person-waiting-up(floor)   : { state-fluent, bool, default = false };
    person-waiting-down(floor) : { state-fluent, bool, default = false };
    person-in-elevator-going-up(elevator)   : { state-fluent, bool, default = false
        };
    person-in-elevator-going-down(elevator) : { state-fluent, bool, default = false
        };

    // Elevator state
    elevator-dir-up(elevator) : { state-fluent, bool, default = true };
    elevator-closed(elevator) : { state-fluent, bool, default = true };
    elevator-at-floor(elevator, floor) : { state-fluent, bool, default = false };

    // Actions: the elevator must move in one direction, it can only switch
    //          direction by signaling the change when the door opens
    //          (i.e., the passengers must know which direction the
    //           elevator is going before they get on... then the elevator
    //           is constrained to go in that direction when the door closes).
    move-current-dir(elevator)     : { action-fluent, bool, default = false };
    open-door-going-up(elevator)   : { action-fluent, bool, default = false };
    open-door-going-down(elevator) : { action-fluent, bool, default = false };
    close-door(elevator)           : { action-fluent, bool, default = false };
  };

  cpfs {

    // We might even allow people to get off the elevator if it switches
    // directions on them while they're in it, but we won't model this now.

    // A person is waiting unless they get on an elevator going in their
    // direction.
    person-waiting-up'(?f) =
      if (person-waiting-up(?f) ^
        ~exists_{?e: elevator} [elevator-at-floor(?e, ?f) ^ elevator-dir-up(?e) ^ ~
            elevator-closed(?e)])
      then KronDelta(true)
      else Bernoulli(ARRIVE-PARAM(?f));

    person-waiting-down'(?f) =
      if (person-waiting-down(?f) ^
        ~exists_{?e: elevator} [elevator-at-floor(?e, ?f) ^ ~elevator-dir-up(?e) ^
            ~elevator-closed(?e)])
      then KronDelta(true)
      else Bernoulli(ARRIVE-PARAM(?f));

```

```
69          // A person is in the elevator going in a direction if someone gets on
70          // in that direction or someone was already on in that direction and does
71          // not get off.
72          person-in-elevator-going-up'(?e) =
73            if (person-in-elevator-going-up(?e))
74              // If elevator not at top floor then stays true, otherwise set to false
75              then KronDelta( ~exists_{?f : floor} [elevator-at-floor(?e, ?f) ^ TOP-FLOOR
                    (?f)] )
76            else
77              // No one in elevator going up... can only be true if someone going up gets
                    in
78              KronDelta( exists_{?f : floor}
79                [ elevator-at-floor(?e, ?f) ^ elevator-dir-up(?e) ^
80                  ~elevator-closed(?e) ^ person-waiting-up(?f) ] );
81
82          person-in-elevator-going-down'(?e) =
83            if (person-in-elevator-going-down(?e))
84              // If elevator not at bottom floor then stays true, otherwise set to false
85              then KronDelta( ~exists_{?f : floor} [elevator-at-floor(?e, ?f) ^ BOTTOM-
                    FLOOR(?f)] )
86            else
87              // No one in elevator going up... can only be true if someone going up gets
                    in
88              KronDelta( exists_{?f : floor}
89                [ elevator-at-floor(?e, ?f) ^ ~elevator-dir-up(?e) ^
90                  ~elevator-closed(?e) ^ person-waiting-down(?f) ] );
91
92          // Elevator needs to be explicitly closed
93          elevator-closed'(?e) =
94            KronDelta([elevator-closed(?e) ^ ~open-door-going-up(?e) ^ ~open-door-going-
                  down(?e)]
95                | close-door(?e));
96
97          // Elevator's destination is set when door is opened (to signal
98          // to people which direction the elevator is going)
99          elevator-dir-up'(?e) =
100           if (open-door-going-up(?e))
101             then KronDelta(true)
102           else if (open-door-going-down(?e))
103             then KronDelta(false)
104           else
105             // If not explicitly set then previous direction persists
106             KronDelta( elevator-dir-up(?e) );
107
108         // Elevator movement
109         //
110         // Note: if the elevator should pause at a floor, it can simply open
111         //       do noops (all actions false).
112         elevator-at-floor'(?e, ?f) =
113
114           ////////////////////////////////////////////////////////////////
115           // Elevator does not move if door is open or elevator does not move
116           ////////////////////////////////////////////////////////////////
117           if (~elevator-closed(?e) | ~move-current-dir(?e))
118             then KronDelta( elevator-at-floor(?e, ?f) )
119
120           ////////////////////////////////////////////////////////////////
121           // These handle the floor that is moved to
122           ////////////////////////////////////////////////////////////////
123           else if (move-current-dir(?e) ^ elevator-dir-up(?e) ^ exists_{?cur : floor}
                    [elevator-at-floor(?e, ?cur) ^ ADJACENT-UP(?cur,?f)])
124             then KronDelta(true)
125           else if (move-current-dir(?e) ^ ~elevator-dir-up(?e) ^ exists_{?cur : floor}
                     [elevator-at-floor(?e, ?cur) ^ ADJACENT-UP(?f,?cur)])
126             then KronDelta(true)
127
128           ////////////////////////////////////////////////////////////////
129           // These handle failed actions -- stay at current floor
130           ////////////////////////////////////////////////////////////////
131           else if (move-current-dir(?e) ^ elevator-dir-up(?e) ^ ~exists_{?next : floor}
                    [elevator-at-floor(?e, ?f) ^ ADJACENT-UP(?f,?next)])
132             then KronDelta( elevator-at-floor(?e, ?f) )
133           else if (move-current-dir(?e) ^ ~elevator-dir-up(?e) ^ ~exists_{?next : floor
                    }
134                 [elevator-at-floor(?e, ?f) ^ ADJACENT-UP(?next,?f)])
```

```
138             then KronDelta( elevator-at-floor(?e, ?f) )
139
140           ////////////////////////////////////////////////////////////////
141           // Otherwise elevator ?e does not move to floor ?f
142           ////////////////////////////////////////////////////////////////
143           else
144             // If here, state persists
145             KronDelta( false );
146       };
147
148       // Reward is a sum of waiting penalties for those in elevators and at floor
149       reward =
150         [sum_{?e: elevator} [
151           -ELEVATOR-PENALTY-RIGHT-DIR * (person-in-elevator-going-up(?e) ^ elevator-dir
                -up(?e))
152         ]] +
153         [sum_{?e: elevator} [
154           -ELEVATOR-PENALTY-RIGHT-DIR * (person-in-elevator-going-down(?e) ^ ~elevator-
                dir-up(?e))
155         ]] +
156         [sum_{?e: elevator} [
157           -ELEVATOR-PENALTY-WRONG-DIR * (person-in-elevator-going-up(?e) ^ ~elevator-
                dir-up(?e))
158         ]] +
159         [sum_{?e: elevator} [
160           -ELEVATOR-PENALTY-WRONG-DIR * (person-in-elevator-going-down(?e) ^ elevator-
                dir-up(?e))
161         ]] +
162         [sum_{?f: floor} [
163           - person-waiting-up(?f) - person-waiting-down(?f)
164         ]];
165       };
166     }
```

B.3 Game of Life

```
1     domain game_of_life_mdp {
2
3       requirements = { reward-deterministic };
4
5       types {
6         x_pos : object;
7         y_pos : object;
8       };
9
10      pvariables {
11        NOISE-PROB(x_pos,y_pos) : { non-fluent, real, default = 0.1 };
12        NEIGHBOR(x_pos,y_pos,x_pos,y_pos) : { non-fluent, bool, default = false };
13        alive(x_pos,y_pos) : { state-fluent,  bool, default = false };
14        set(x_pos,y_pos)   : { action-fluent, bool, default = false };
15      };
16
17      cpfs {
18        // For interactivity: we allow an agent to explicitly set different cells.
19
20        alive'(?x,?y) =
21          if ([alive(?x,?y) ^ ([sum_{?x2 : x_pos, ?y2 : y_pos} NEIGHBOR(?x,?y,?x2,?y2)
                ^ alive(?x2,?y2)] >= 2)
22                ^ ([sum_{?x2 : x_pos, ?y2 : y_pos} NEIGHBOR(?x,?y,?x2,?y2) ^ alive(?
                    x2,?y2)] <= 3)]
23              | [~alive(?x,?y) ^ ([sum_{?x2 : x_pos, ?y2 : y_pos} NEIGHBOR(?x,?y,?x2
                    ,?y2) ^ alive(?x2,?y2)] == 3)]
24              | set(?x,?y))
25          then Bernoulli(1.0 - NOISE-PROB(?x,?y))
26          else Bernoulli(NOISE-PROB(?x,?y));
27      };
28
29      reward = sum_{?x : x_pos, ?y : y_pos} [alive(?x,?y) - set(?x,?y)];
30
31    state-action-constraints {
32      forall_{?x : x_pos, ?y : y_pos}
33        [(NOISE-PROB(?x,?y) >= 0.0) ^ (NOISE-PROB(?x,?y) <= 1.0)];
34    };
35    }
```

B.4 Navigation

```
domain navigation_mdp {
  requirements = {
    reward-deterministic
  };

  types {
    xpos : object;
    ypos : object;
  };

  pvariables {

    NORTH(ypos, ypos) : {non-fluent, bool, default = false};
    SOUTH(ypos, ypos) : {non-fluent, bool, default = false};
    EAST(xpos, xpos)  : {non-fluent, bool, default = false};
    WEST(xpos, xpos)  : {non-fluent, bool, default = false};

    MIN-XPOS(xpos) : {non-fluent, bool, default = false};
    MAX-XPOS(xpos) : {non-fluent, bool, default = false};
    MIN-YPOS(ypos) : {non-fluent, bool, default = false};
    MAX-YPOS(ypos) : {non-fluent, bool, default = false};

    P(xpos, ypos) : {non-fluent, real, default = 0.0};

    GOAL(xpos,ypos) : {non-fluent, bool, default = false};

    // Fluents
    robot-at(xpos, ypos) : {state-fluent, bool, default = false};

    // Actions
    move-north : {action-fluent, bool, default = false};
    move-south : {action-fluent, bool, default = false};
    move-east  : {action-fluent, bool, default = false};
    move-west  : {action-fluent, bool, default = false};
  };

  cpfs {

    robot-at'(?x,?y) =

      if ( GOAL(?x,?y) ^ robot-at(?x,?y)  )
      then
        KronDelta(true)
      else if (( exists_{?x2 : xpos, ?y2 : ypos} [ GOAL(?x2,?y2) ^ robot-at(?x2,?y2) ) ] )
          | ( move-north ^ exists_{?y2 : ypos} [ NORTH(?y,?y2) ^ robot-at(?x,?y) ] )
          | ( move-south ^ exists_{?y2 : ypos} [ SOUTH(?y,?y2) ^ robot-at(?x,?y) ] )
          | ( move-east ^ exists_{?x2 : xpos} [ EAST(?x,?x2) ^ robot-at(?x,?y) ] )
          | ( move-west ^ exists_{?x2 : xpos} [ WEST(?x,?x2) ^ robot-at(?x,?y) ] ))
      then
        KronDelta(false)
      else if (( move-north ^ exists_{?y2 : ypos} [ NORTH(?y2,?y) ^ robot-at(?x,?y2) ) ] )
          | ( move-south ^ exists_{?y2 : ypos} [ SOUTH(?y2,?y) ^ robot-at(?x,?y2) ] )
          | ( move-east ^ exists_{?x2 : xpos} [ EAST(?x2,?x) ^ robot-at(?x2,?y) ] )
          | ( move-west ^ exists_{?x2 : xpos} [ WEST(?x2,?x) ^ robot-at(?x2,?y) ] )
      then
        Bernoulli( 1.0 - P(?x, ?y) )
      else
        KronDelta( robot-at(?x,?y) );

  };

  // 0 reward for reaching goal, -1 in all other cases
  reward = [sum_{?x : xpos, ?y : ypos} -(GOAL(?x,?y) ^ ~robot-at(?x,?y))];

}
```

B.5 Skill Teaching

```
1   domain skill_teaching_mdp {
2
3     requirements = {
4       reward-deterministic
5     };
6
7     types {
8       skill : object;
9     };
10
11    pvariables {
12
13      //how valuable is this skill?
14      SKILL_WEIGHT(skill) : { non-fluent, real, default = 1.0 };
15
16      //some skills are pre-reqs for others.  Your ability to achiev a higher level
            skill is dependent on how
17      //many of the pre-reqs you have mastered
18      PRE_REQ(skill, skill) : { non-fluent, bool, default = false };
19
20      //probability of getting a question right if you have all the pre-reqs
21      PROB_ALL_PRE(skill) : { non-fluent, real, default = 0.8 };
22      //if you don't have all the pre-cons, probaility mass is summed using these
            individual pieces
23      PROB_PER_PRE(skill) : { non-fluent, real, default = 0.1 };
24
25      PROB_ALL_PRE_MED(skill) : { non-fluent, real, default = 1.0 };
26      //if you don't have all the pre-cons, probaility mass is summed using these
            individual pieces
27      PROB_PER_PRE_MED(skill) : { non-fluent, real, default = 0.3 };
28
29      PROB_HIGH(skill) : { non-fluent, real, default = 0.9 };
30
31      LOSE_PROB(skill) : { non-fluent, real, default = 0.02 };
32
33      //proficiency values, they accumulate so low and med can be on at the same time
            and only high will turn off
34      proficiencyMed(skill) : { state-fluent, bool, default = false };
35      proficiencyHigh(skill) : { state-fluent, bool, default = false };
36
37      updateTurn(skill) : {state-fluent, bool, default = false};
38
39      answeredRight(skill): {state-fluent, bool, default = false};
40      hintedRight(skill): {state-fluent, bool, default = false};
41      hintDelayVar(skill) : {state-fluent, bool, default = false};
42
43      //two actions.  Hint can get you directly to proficiencyMed, but only if all
            the pre_reqs are on
44      askProb(skill) : {action-fluent, bool, default = false};
45      giveHint(skill) :  {action-fluent, bool, default = false};
46    };
47
48    cpfs {
49
50      updateTurn'(?s) =
51        KronDelta( [forall_{?s2: skill} ~updateTurn(?s2)] ^ (askProb(?s) | giveHint(?
              s)) );
52
53      //without intermediate nodes, we need to keep ``on'' all proficiency levels
            that have been attained
54
55      answeredRight'(?s) =
56        if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^ proficiencyHigh(?s
              ))
57          then Bernoulli(PROB_HIGH(?s))
58        else if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^
              proficiencyMed(?s) ^forall_{?s3: skill}[PRE_REQ(?s3, ?s) =>
              proficiencyHigh(?s3)])
59          then Bernoulli(PROB_ALL_PRE_MED(?s))
60      else if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^proficiencyMed(?s)
            ^ askProb(?s))
61          then Bernoulli(sum_{?s2: skill}[PRE_REQ(?s2, ?s) * PROB_PER_PRE_MED(?s)])
```

```
62          else if ([forall_{?s3: skill} ~updateTurn(?s3)] ^ askProb(?s) ^forall_{?s2:
                skill}[PRE_REQ(?s2, ?s) => proficiencyHigh(?s2)])
63            then Bernoulli(PROB_ALL_PRE(?s))
64        else if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s)  ^ askProb(?s))
65          then Bernoulli(sum_{?s2: skill}[PRE_REQ(?s2, ?s) * PROB_PER_PRE(?s)])
66          else
67            KronDelta( false );
68
69          hintedRight'(?s) =
70            KronDelta( [forall_{?s3: skill} ~updateTurn(?s3)] ^ giveHint(?s) ^ forall_{?
                s2: skill}[PRE_REQ(?s2, ?s) => proficiencyHigh(?s2)] );
71
72          hintDelayVar'(?s) =
73            KronDelta( [forall_{?s2: skill} ~updateTurn(?s2)] ^ giveHint(?s) );
74
75          //proficiencyMed can be reached through a hint if all preconditions are known
                or by a problem answered correctly
76          proficiencyMed'(?s) =
77        if (~updateTurn(?s) ^ proficiencyMed(?s))
78          then KronDelta( true )
79        else if (updateTurn(?s) ^ hintedRight(?s))
80          then KronDelta( true )
81        else if (updateTurn(?s) ^ answeredRight(?s))
82          then KronDelta( true )
83        else if (proficiencyHigh(?s)) //may come down
84          then KronDelta( true )
85        else if (proficiencyMed(?s) ^ updateTurn(?s) ^ hintDelayVar(?s))
86          then KronDelta( true ) //can't lose it on a hint
87          else
88            KronDelta( false );
89
90          //high proficiency is reached by getting a question and having proficiencyMed
91          //but you can lose it too if you get questions wrong
92          proficiencyHigh'(?s) =
93        if (forall_{?s2: skill}[~updateTurn(?s2)])  //student turn
94          then KronDelta( proficiencyHigh(?s) )
95        else if (~updateTurn(?s) ^ proficiencyHigh(?s))
96          then Bernoulli(1.0 - LOSE_PROB(?s))
97        else if (proficiencyMed(?s) ^ updateTurn(?s) ^ answeredRight(?s))
98          then KronDelta( true )
99        else if (proficiencyHigh(?s) ^ updateTurn(?s) ^ (hintDelayVar(?s) | answeredRight
              (?s))) //can't lose it on a hint
100         then KronDelta( true )
101       else KronDelta( false );
102
103       };
104
105       reward = [sum_{?s : skill} [SKILL_WEIGHT(?s) * proficiencyHigh(?s)]] + [sum_{?s :
              skill} -[SKILL_WEIGHT(?s) * ~proficiencyMed(?s)]];
106
107     }
```

B.6 SysAdmin

```
1   domain sysadmin_mdp {
2
3     requirements = {
4       reward-deterministic // this domain does not use a stochastic reward
5     };
6
7     types {
8       computer : object;
9     };
10
11    pvariables {
12
13      REBOOT-PROB : { non-fluent, real, default = 0.1 };
14      REBOOT-PENALTY : { non-fluent, real, default = 0.75 };
15
16      CONNECTED(computer, computer) : { non-fluent, bool, default = false };
17
18      running(computer) : { state-fluent, bool, default = false };
19
20      reboot(computer) : { action-fluent, bool, default = false };
21    };
22
23    cpfs {
24
25      running'(?x) = if (reboot(?x))
26              then KronDelta(true)  // if computer is rebooted then must be running
27              else if (running(?x)) // otherwise outcome depends on network
                    properties
28                then Bernoulli(.45 + .5*[1 + sum_{?y : computer} (CONNECTED(?y,?x) ^
                      running(?y))]
29                             / [1 + sum_{?y : computer} CONNECTED(?y,?x)])
30                else Bernoulli(REBOOT-PROB);
31    };
32
33    reward = sum_{?c : computer} [running(?c) - (REBOOT-PENALTY * reboot(?c))];
34  }
```

B.7 Exemplo usado no artigo de BMDPs descrito com RDDL

```
1 // Domínio adaptado do artigo 'Model Reduction Techniques for
2 //  Computing Approximately Optimal Solutions for MDPs' de Givan & Dean.
3
4 domain example_aprox_givan_mdp {
5   requirements = {
6     reward-deterministic
7   };
8
9   pvariables {
10    x1 : {state-fluent, bool, default = false};
11    x2 : {state-fluent, bool, default = false};
12    x3 : {state-fluent, bool, default = false};
13  };
14
15  cpfs {
16    x1' = if (x1) then Bernoulli (0.8)
17        else if (~x1 ^ x2) then Bernoulli (0.7)
18        else Bernoulli (0.65);
19    x2' = Bernoulli (0.7);
20    x3' = if (x2) then Bernoulli (0.7)
21        else if (~x2 ^ x3) then KronDelta (true)
22        else Bernoulli (0.5);
23  };
24
25  reward = if (x1) then 1 else 0;
26 }
```

```
1 // Instância adaptada do artigo 'Model Reduction Techniques
2 // for Computing Approximately Optimal Solutions for MDPs' de Givan & Dean.
3
4 instance example_aprox_givan_inst_mdp {
5   domain = example_aprox_givan_mdp;
6
7   init-state {
8     ~x1;
9     ~x2;
10    ~x3;
11  };
12
13  max-nondef-actions = 1;
14  horizon = 40;
15  discount = 0.99;
16 }
```

Referências Bibliográficas

Bahar *et al.***(1993)** R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo e Fabio Somenzi. Algebraic decision diagrams and their applications. Em *Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided design*, páginas 188–191. Citado na pág. 20

Barto *et al.***(1993)** Andrew G. Barto, Steven J. Bradtke e Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138. Citado na pág. xix, 2, 10, 13, 14, 15

Bellman(1957) R. E. Bellman. *Dynamic Programming*. Princeton University Press. Citado na pág. xix, 2, 11, 12, 13

Bertsekas e Tsitsiklis(1991) Dimitri P. Bertsekas e John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16:580–595. Citado na pág. 13

Bonet e Geffner(2003) Blai Bonet e Héctor Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. Em *Proceedings of 13th International Conference on Automated Planning and Scheduling*, páginas 12–21. AAAI Press. Citado na pág. xix, 15, 16, 17, 18

Boutilier *et al.***(1995)** Craig Boutilier, Richard Dearden e Moisés Goldszmidt. Exploiting structure in policy construction. Em *IJCAI-95*, páginas 1104–1111. Citado na pág. 23

Boutilier *et al.***(1996)** Craig Boutilier, Nir Friedman, Moises Goldszmidt e Daphne Koller. Context-specific independence in Bayesian networks. Em *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, páginas 115–123. Citado na pág. 20

Boutilier *et al.***(1998)** Craig Boutilier, Ronen I. Brafmany e Christopher Geibz. Structured reachability analysis for Markov decision processes. Em *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, páginas 24–32. Citado na pág. 79, 80

Boutilier *et al.***(1999)** Craig Boutilier, Thomas Dean e Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence*, 11:1–94. Citado na pág. 1, 19, 35

Bryant(1986) R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691. Citado na pág. 20, 21

Buffet e Aberdeen(2005) Olivier Buffet e Douglas Aberdeen. Robust planning with (L)RTDP. Em *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, páginas 1214–1219. Citado na pág. xix, 2, 27, 28, 29, 31, 32, 33

Cormen *et al.***(2009)** Thomas H. Cormen, Charles H. Leiserson, Ronald L.Rivest e Clifford Stein. *Introduction to algorithms*. MIT Press, 3rd edition edição. Citado na pág. 29, 81

Dean e Givan(1997) Thomas Dean e Robert Givan. Model minimization in Markov decision processes. Em *Proceedings of the 14th National Conference on Artificial Intelligence*, páginas 106–111. Citado na pág. 38, 39, 40, 43, 45

Dean e Kanazawa(1990) Thomas Dean e Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150. Citado na pág. 19

Dean *et al.***(1997)** Thomas Dean, Robert Givan e Sonia Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. Em *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, páginas 124–131. Morgan Kauffman. Citado na pág. xix, 2, 25, 28, 31, 41, 47, 49, 80, 83

Delgado(2010) Karina Valdivia Delgado. *Processos de decisão Markovianos fatorados com probabilidades imprecisas*. Tese de Doutorado, Instituto de Matemática e Estatística da Universidade de São Paulo. Citado na pág. 25

Feng *et al.***(2003)** Zhengzhu Feng, Eric A. Hansen e Shlomo Zilberstein. Symbolic generalization for on-line planning. Em *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, páginas 109–116. Citado na pág. 23, 79

Ferns *et al.***(2004)** Norm Ferns, Prakash Panangaden e Doina Precup. Metrics for finite Markov decision porcesses. Em *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, páginas 162–169. Citado na pág. 47

Fourman(2000) Michael Fourman. Propositional planning. Em *Proceedings Workshop on Model Theoretic Approaches to Planning.*, páginas 10–17. Citado na pág. 54

Franco(2012) Fabio Franco. Jogos Markovianos alternados sob incerteza. Dissertação de Mestrado, Instituto de Matematica e Estatistica - Universidade de Sao Paulo. Citado na pág. xv, 27

Gardner(1970) Martin Gardner. Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123. Citado na pág. 5, 67

Ghallab *et al.***(2004)** Malik Ghallab, Dana S. Nau e Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kauffman. Citado na pág. 12, 13, 19

Givan *et al.***(2000)** Robert Givan, Sonia Leach e Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122:71–109. Citado na pág. xv, xvi, 2, 20, 25, 26, 27, 28, 30, 46, 47, 49

Givan *et al.***(2003)** Robert Givan, Matthew Greig e Thomas Dean. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147:163–223. Citado na pág. xix, 2, 36, 37, 38, 40, 41, 42, 43

Goldsmith e Sloan(2000) Judy Goldsmith e Robert H. Sloan. The complexity of model aggregation. Em *Proceedings of the 5th International Conference on Artificial Intelligence*, páginas 122–129. Citado na pág. 49

Guo e Leong(2010) Wenyuan Guo e Tze-You Leong. An analytic characterization of model minimization in factored Markov decision processes. Em *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, páginas 1077–1082. Citado na pág. xix, 42, 43, 44

Hoey *et al.***(1999)** Jesse Hoey, Robert St-Aubin, Alan Hu e Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. Em *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, páginas 279–288. Morgan Kauffman. Citado na pág. 2, 23

Howard(1960) R.A. Howard. *Dynamic Programming and Markov Process*. The MIT Press. Citado na pág. xix, 12

Katz *et al.***(2012)** M. Katz, J. Hoffman e M. Helmert. How to relax bisimulation? Em *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, páginas 102–109. Citado na pág. 80

Kim e Dean(2003) Kee-Eung Kim e Thomas Dean. Solving factored MDPs via non-homogeneous partitioning. *Artificial Intelligence - special issue on planning with uncertainty and incomplete information*, 147:225–251. Citado na pág. 80

Kim e Dean(2002) Kee-Eung Kim e Thomas Dean. Solving factored MDPs with large action space using algebraic decision diagrams. Em *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*, páginas 80–89. Citado na pág. 6, 43, 53

Kolobov *et al.***(2012)** Andrey Kolobov, Peng Dai, Mausam e Daniel S. Weld. Reverse iterative deepening for finite-horizon MDPs with large branching factors. Em *ICAPS*, páginas 146–154. Citado na pág. 80

Konami(1981) Konami. Frogger, 1981. URL http://www.konami.com/officialsites/frogger/. Citado na pág. 66

Li *et al.***(2006)** Lihong Li, Thomas J. Walsh e Michael L. Littman. Towards a unified theory of state abstraction for MDPs. Em *Proceedings of the 9th International Sysmposium on Artificial Intelligence and Mathematics*, páginas 531–539. Citado na pág. 25, 35

Puterman(1994) Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons. Citado na pág. 1, 9, 10, 11

Ravindran e Barto(2004) Balaraman Ravindran e Andrew G. Barto. Approximate homomorphisms: A framework for non-exact minimization in Markov decision processes. Em *Proceedings of the 5th International Conference on Knowledge Based Computer Systems*. Citado na pág. 25

Russel e Norvig(2003) Stuart Russel e Peter Norvig. *Inteligência Artificial: Uma abordagem moderna*. Campus, segunda edição. Citado na pág. 1, 10, 54

Sanner(2010) Scott Sanner. Relational dynamic influence diagram language (rddl): Language description, 2010. URL http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf. Citado na pág. 65, 83

Satia e Jr.(1973) Jay K. Satia e Roy E. Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740. Citado na pág. 25

St-Aubin *et al.***(2000)** Robert St-Aubin, Jesse Hoey e Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. Em *Proceedings of Conference on Neural Information Processing Systems*, páginas 1089–1095. Citado na pág. 23

Trevizan *et al.***(2006)** Felipe Werndl Trevizan, Fabio Gagliardi Cozman e Leliane Nunes de Barros. Unifying nondeterministic and probabilistic planning through imprecise Markov decision processes. Em *IBERAMIA-SBIA*, páginas 502–511. Citado na pág. 25

Trevizan *et al.***(2007)** Felipe Werndl Trevizan, Fabio Gagliardi Cozman e Leliane Nunes de Barros. Planning under risk and knightian uncertainty. Em *IJCAI*, páginas 2023–2028. Citado na pág. 25

Trevizan *et al.***(2008)** Felipe Werndl Trevizan, Fabio Gagliardi Cozman e Leliane Nunes de Barros. Mixed probabilistic and nondeterministic factored planning through markov decision processes with set-valued transitions, 2008. In: Workshop of ICAPS 2008 on A Reality Check for Planning and Scheduling Under Uncertainty, 2008, Sydney. Workshop Notes WS5 of ICAPS 2008, 2008. Citado na pág. 25