Programação dinâmica simbólica aproximada e assíncrona para processos de decisão markovianos com variáveis contínuas

Luis Gustavo Rocha Vianna

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do Título
de
Mestre em Ciências

Programa: Mestrado em Ciência da Computação
Orientadora: Profa. Dra. Leliane Nunes de Barros

Durante o desenvolvimento deste trabalho o autor
recebeu auxílio financeiro da FAPESP, processo 2011/16962-0

São Paulo, Fevereiro de 2015

Programação dinâmica simbólica aproximada e assíncrona para processos de decisão markovianos com variáveis contínuas

Esta é a versão original da dissertação elaborada pelo candidato (Luis Gustavo Rocha Vianna), tal como submetida à Comissão Julgadora.

Resumo

VIANNA, L. G. R. **Programação dinâmica simbólica aproximada e assíncrona para processos de decisão markovianos com variáveis contínuas**. 2015. 75 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2015.

Este trabalho trata o problema de planejamento em inteligência artificial, mais especificamente, planejamento probabilístico com variáveis contínuas. Aplicações de planejamento em inteligência artificial, em geral, envolvem recursos contínuos, portanto é necessário que os agentes raciocinem com modelos que representem variáveis contínuas. Uma solução exata, recentemente proposta, para uma classe de problemas de planejamento probabilístico é a *programação dinâmica simbólica - PDS*, que é capaz de resolver de maneira eficiente problemas com variáveis discretas e contínuas, utilizando manipulação simbólica. Essa técnica resolve problemas com variáveis contínuas manipulando expressões definidas por casos que envolvem essas variáveis para obter a expressão da solução exata. No entanto, a manipulação envolve um aumento no número de casos usados na expressão, de forma que a representação exata das soluções pode se tornar intratavelmente custosa. Neste trabalho, pretendemos adaptar a PDS com uma técnica de aproximação que permite controlar o crescimento da complexidade das expressões em troca de um pequeno erro em seus valores. A maneira como pretendemos simplificar as expressões é baseada em reduzir o número de casos numa expressão simbólica, o que é feito unindo regiões de casos diferentes que apresentam valores próximos. Além disso, a eficiência da PDS pode ser melhorada modificando qual o cálculo usado para obter a expressão da solução. Uma forma de evitar cálculos desnecessários é utilizar a informação do estado inicial e fazer uma busca heurística a partir dele, restringindo a região de valores para os quais precisamos da solução ótima. Assim, pretendemos criar dois novos algoritmos que usam a manipulação simbólica das expressões com variáveis contínuas, adicionando componentes de técnicas recentes para planejamento probabilístico discreto.

Palavras-chave: Planejamento Probabilístico com Variáveis Contínuas, Programação Dinâmica Simbólica, Processo de Decisão Markoviano Híbrido.

Abstract

VIANNA, L. G. R. **Approximate and asynchronous symbolic dynamic programming for Markov decision processes in continuous spaces**. 2015. 75 f. (Master's Thesis) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2015.

This work is a study on the planning problem in artificial intelligence, specifically probabilistic planning in continuous spaces. The efficient solution of planning problems is a major goal in artificial intelligence and can be applied extensively in autonomous agents. In many applications, the modelled problem contains continuous resources, so that an optimal planner must reason over continuos quantities to obtain appropriate actions. A recent and exact solution is Symbolic Dynamic Programming, which extends discrete probabilistic planning solutions to continuous problems by using a symbolic representation of state variables. This solution is interesting because it can find optimal solutions, however it is limited in efficiency because it relies on standard dynamic programming and doesn't use initial state information or heuristic search. On this work, I will extend Symbolic Dynamic Programming to use more efficient dynamic programming approaches, based on recent solutions for discrete probabilistic planning. A novel planner using symbolic representation and heuristic search is proposed and compared to previous works on relevant continuos scenarios.

Keywords: Probabilistic Planning in Continuos Spaces, Symbolic Dynamic Programming, Hybrid Markov Decision Processes.

Sumário

Lista de Abreviaturas

Estruturas de Dados:

BDD	Diagrama de Decisão Booleano
ADD	Diagrama de Decisão Algébrico
XADD	Diagrama de Decisão Algébrico Estendido

Modelos:

MDP	Processo de Decisão Markoviano
MDPF	Processo de Decisão Markoviano com Horizonte Finito
SSP	Processo de Decisão Markoviano com Horizonte Indefinido
MDPD	Processo de Decisão Markoviano Descontado
DBN	Rede Bayesiana Dinâmica
HMDP	Processo de Decisão Markoviano Híbrido
HMDPF	Processo de Decisão Markoviano Híbrido com Horizonte Finito
HMDPD	Processo de Decisão Markoviano Híbrido Descontado
DCMDP	Processo de Decisão Markoviano com Estados Discretos e Contínuos

Algoritmos:

IV	Iteração de Valor
IP	Iteração de Política
RTDP	*Real-Time Dynamic Programming*
LRTDP	*Labeled Real-Time Dynamic Programming*
BRTDP	*Bounded Real-Time Dynamic Programming*
SPUDD	*Stochastic Planning Using Decision Diagrams*
APRICODD	*Approximate Policy Construction using Decision Diagrams*
SDP	*Symbolic Dynammic Programming*
BASDP	*Bounded Approximate Symbolic Dynamic Programming*
RTSDP	*Real-Time Symbolic Dynammic Programming*
HALP	*Hybrid Approximate Linear Programming*
p-Sulu	*Probabilistic Sulu Planner*

Lista de Símbolos

γ	Fator de desconto
$\mathcal{S}$	Espaço de estados
$\mathcal{A}$	Espaço das ações
$\mathcal{T}$	Função de transição
$\mathcal{R}$	Função de recompensa
ϕ, ψ	Fórmulas lógicas
θ	Politopo
ϵ	Valor de precisão em aproximações

Lista de Figuras

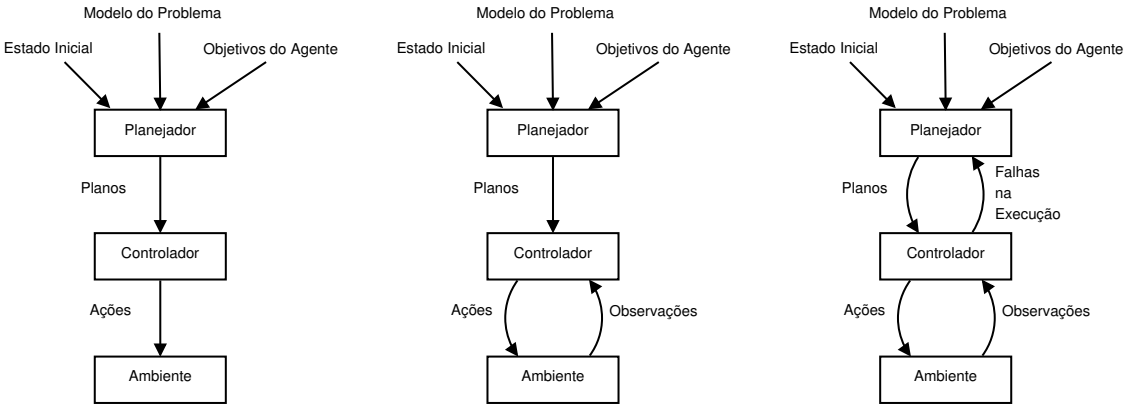
Lista de Tabelas

Lista de Algoritmos

Introdução

O problema abordado neste mestrado é o de tomada de decisão sequencial, denominado planejamento automatizado em inteligência artificial. Planejar é organizar ações de forma que sua execução resulte na satisfação de um conjunto de metas [Nau *et al.* (2004)]. Um planejador é um sistema capaz de planejar, ou seja, simular os efeitos das ações de um agente num modelo do ambiente e prever o resultado de combinações sequenciais de ações. Pode-se pensar que o planejador compõe uma camada intermediária de decisão de um agente autônomo. O planejador precisa de um modelo do ambiente, das ações que o agente pode tomar e das metas de forma a gerar um plano (sequência de ações) ou uma política (regra de decisão que depende do estado observado) que atinja os objetivos. A execução adequada do plano ou da política no ambiente real é uma tarefa de nível mais baixo, que é realizada por um controlador.

A Figura 1 ilustra as camadas de um sistema autônomo e o papel do planejador na atuação de um agente autônomo. A camada de nível mais alto é o planejador, que usa um modelo do ambiente e dos objetivos para formar planos. Esses planos são executados por um controlador que interage diretamente com o ambiente. No modelo tradicional de planejamento clássico, Figura 1a, é considerado que o ambiente é conhecido e os efeitos das ações são determinísticos, por isso o plano pode ser executado sem qualquer modificação. No caso do planejamento sob incerteza, Figura 1b, ocorrem efeitos não-determinísticos, probabilísticos ou não, e é necessário que o controle observe o resultado de suas ações no ambiente durante a execução e o plano precisa considerar cada resultado possível. Uma estrutura ainda mais complexa é a usada em planejamento com reparo de plano, Figura 1c, onde o agente pode detectar falhas na execução de seu plano e precisa replanejar.



(a) *Planejamento Clássico* **(b)** *Planejamento sob Incerteza* **(c)** *Planejamento e Reparo*

Figura 1: *Diagrama conceitual do sistema de decisão de um agente autônomo.*

Geralmente, o ambiente é modelado por um conjunto de estados possíveis, a atuação do agente é modelada por ações que provocam transições entre estados e o objetivo do agente é modelado por uma recompensa, que é recebida a cada transição e que o agente deve acumular, ou um conjunto de estados meta. Além disso, iremos considerar que o problema é dividido em passos de decisão nas quais apenas uma decisão é tomada. Em cada passo de decisão, chamado de estágio, o agente percebe seu estado atual, escolhe uma ação, efetua a transição para um novo estado e recebe uma recompensa imediata. Esse modelo de processo é ilustrado na Figura 2.

Mais formalmente, o modelo que iremos utilizar para um problema de planejamento é uma tupla $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, onde:

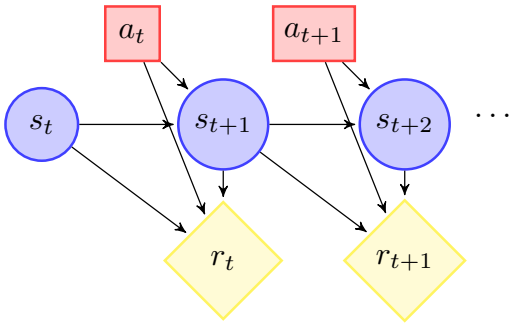


Figura 2: *Dinâmica de um processo de decisão sequêncial. Nós marcados por* $\boldsymbol{s}$, $\boldsymbol{a}$ *e* $\boldsymbol{r}$ *representam, respectivamente, o estado, a ação escolhida e a recompensa recebida. Os sub-índices* $\boldsymbol{t}$, $\boldsymbol{t+1}$ *e* $\boldsymbol{t+2}$ *denotam de que passo de decisão é um elemento.*

- $\mathcal{S}$ é um conjunto de estados que representa todas as configurações possíveis do agente e ambiente. Geralmente um estado é descrito por variáveis representando propriedades do agente e do ambiente.
- $\mathcal{A}$ é um conjunto de ações, que representa as opções de ações dentre as quais o agente deve escolher para alterar seu ambiente.
- $\mathcal{T}$ é uma função de transição, que representa a dinâmica do ambiente, que associa cada escolha de uma ação num estado a um estado sucessor, uma distribuição probabilística sobre os estados, ou um conjunto de estados sucessores possíveis.
- $\mathcal{R}$ é uma função de recompensa que representa o objetivo do agente e associa cada transição a um valor que mede a utilidade que o agente obtem executando essa transição . Note que podemos representar problemas em que o objetivo do agente é apenas alcançar metas, usando uma função de recompensa que é nula exceto nas transições que terminam em estados meta.

Exemplo 1 O problema do Robô em Marte [Bresina *et al.* (2002)] *Considere um robô em Marte, que precisa realizar tarefas de forma autônoma. Suas tarefas poderiam ser, por exemplo, fotografar fenômenos celestes e coletar amostras de substrato marciano. Vamos descrever este cenário como um problema de planejamento. O conjunto de estados* $\mathcal{S}$ *representa as situações em que ele pode se encontrar, e é descrito pela condição dos fatores que influenciam as tomadas de decisão, por exemplo: a posição atual do robô e da nave, a carga da sua bateria, data e horário atuais, quais fotos já foram tiradas e quais amostras já coletadas. O conjunto de ações* $\mathcal{A}$ *disponíveis ao robô inclui navegação, uso e ajuste dos equipamentos fotográficos e de coleta de solo. Uma função de transição* $\mathcal{T}$ *para este problema é um modelo que descreve o resultado de cada ação do robô, por exemplo: prevê qual será sua nova posição após ele executar uma ação de movimentação; como o banco de dados de imagens muda ao tirar uma fotografia; e como varia a carga de sua bateria quando ele realiza uma coleta de material marciano. A função recompensa* $\mathcal{R}$ *orienta o robô a ser produtivo. Ela incentiva com valores positivos fotos bem tiradas de fenômenos de interesse ou coletas de material adequado e reprime com valores negativos ações custosas e infrutíferas ou que danifiquem seu equipamento, como andar em círculos ou não desviar de buracos. Uma solução para este problema, é uma política que determina as ações que o robô deve executar em cada estado para alcançar seus objetivos, codificados como recompensa positiva. Assim, a política ótima é aquela que maximiza a recompensa acumulada, por exemplo, coletando amostras relevantes, tirando fotografias precisas, bem como movendo-se sempre por caminhos seguros e curtos.*

Como ilustrado pelo Exemplo 1, problemas muito complexos podem ser modelados como problemas de planejamento, de forma que a solução eficiente e automática destes é uma etapa importante para o comportamento autônomo de agentes artificiais. Essa formulação para problemas de planejamento é muito abrangente e incluí diferentes categiras de sub-problemas. Por exemplo, quando é conhecido o estado inicial $s_0 \in \mathcal{S}$ e as funções de transição e de recompensa são determinísticas, temos um problema de *planejamento clássico* [Russell e Norvig (2003)], Figura 1a. Se consideramos que as transições são probabilísticas e que o estado atual é perfeitamente observável após cada ação temos a área de *planejamento probabilístico* [Mausam e Kolobov (2012)], Figura 1b. A área de planejamento probabilístico também pode ser dividida de acordo com a estrutura do espaço de estados $\mathcal{S}$. Se $\mathcal{S}$ é finito, ou seja, pode ser descrito por variáveis com valores inteiros, temos um problema de *planejamento probabilístico com variáveis discretas*, caso contrário, $\mathcal{S}$ é descrito com variáveis discretas e contínuas, e temos o problema é de *planejamento probabilístico com variáveis contínuas.*

Objetivo

O objetivo deste trabalho de mestrado é propor novas soluções para planejamento probabilístico com variáveis contínuas. A principal solução exata esse problema é a técnica de *Programação Dinâmica Simbólica - SDP* (do inglês, *Symbolic Dynamic Programming*) [Sanner *et al.* (2011); Zamani *et al.* (2012)] que é inovadora por representar o modelo e a solução do problema como funções simbólicas das variáveis de estado. A principal contribuição dessa técnica é utilizar um cálculo baseado em casos, que permite a manipulação de funções definidas por expressões de forma a encontrar a expressão exata da solução ótima.

No entanto, a eficiência dessa técnica é limitada pela complexidade da solução ótima e, para problemas de interesse prático, essa frequentemente exige muitas separações do espaço de estados de forma que a solução simbólica completa e exata se torna intratável. Neste mestrado, minha principais contribuições são duas técnicas propostas para aumentar a eficiência de soluções baseadas em programação dinâmica simbólica.

- Como há uma grande complexidade inerente a solução exata de problemas grandes, a programação dinâmica simbólica frequentemente não tem memória suficiente para completar a solução. Para resolver essa limitação propomos incrementar a estrutura de dados usada com funções que permitam compactar a função representada com um erro limitado, identificando regiões que podem ser unificadas e gerando uma versão aproximada da solução com muito menos regiões. Essa técnica deu origem ao novo algoritmo *Bounded Approximate Symbolic Dynamic Programming - BASDP* que é consegue resolver instâncias maiores e oferece a possibilidade de trocar precisão por um ganho em eficiência.
- Uma outra abordagem para reduzir a complexidade da solução encontrada pelo SDP é evitar o cálculo da solução para todo o espaço de estados, restringindo o planejamento às regiões relevantes de acordo com a informação do estado inicial. Para isso, propomos o algoritmo de programação dinâmica simbólica assíncrona *Real-time Symbolic Dynamic Programming - RTSDP*. Neste algoritmo, usamos uma nova versão da atualização da programação dinâmica para atualizar apenas uma região de cada vez e com isso evitamos o crescimento da solução.

Organização

Essa dissertação está organizada nos seguintes capítulos:

Capítulo 1) Fundamentos de planejamento probabilístico com variáveis discretas. É apresentado o principal modelo para planejamento probabilístico, o processo de decisão markoviano, e sua solução tradicional, iteração de valor, que é base para a técnica de SDP. Além disso, apresentamos a versão fatorada do modelo e soluções de programação dinâmica simbólica com variáveis discretas.

Capítulo 2) Fundamentos de planejamento probabilístico com variáveis contínuas. Descrevemos o modelo para planejamento probabilístico com variáveis contínua, processo de decisão markoviano híbrido, e soluções existentes para esse modelo, bem como a solução que iremos estender neste mestrado: programação dinâmica simbólica com variáveis contínuas.

Capítulo 3) Apresentamos a versão aproximada da SDP para variáveis contínuas. Explicamos a técnica de aproximação para diagramas de decisão desenvolvida e resultados experimentais obtidos com o algoritmo aproximado.

Capítulo 4) Apresentamos a versão assíncrona da SDP, baseada no algoritmo RTDP. Explicamos como modificar a atualização de para ser restrita a uma região e como combinar as atualização de regiões para obter um algoritmo assíncrono. Mostramos resultados experimentais confirmando a maior eficiência do novo algoritmo.

Capítulo 5) Considerações finais sobre este trabalho e etapas futuras.

Capítulo 1

Planejamento Probabilístico com Variáveis Discretas

Um problema de planejamento probabilístico com variáveis discretas é um problema de planejamento que possui as seguintes características: (1) as transições de estado são descritas por conjuntos de estados sucessores com probabilidades, que usualmente são consideradas conhecidas e estacionárias, ou seja, que não variam com o estágio do processo; (2) o conjunto de estados é finito ou, no caso fatorado, as variáveis de estado assumem apenas um conjunto finito de valores.

Neste capítulo iremos introduzir o principal modelo usado para problemas de planejamento probabilístico discreto, o **processo de decisão markoviano**. Em seguida, descreveremos a solução tradicional deste problema utilizando programação dinâmica, o algoritmo **iteração de valor**, e uma solução mais recente que usa o conhecimento do estado inicial e funções heurísticas para restringir o espaço de busca, o algoritmo **RTDP**. Então, apresentaremos uma extensão do modelo, chamada **processo de decisão markoviano fatorado**, que utiliza a estrutura de dependência das variáveis de estado para representar o problema de forma mais eficiente. Por fim, apresentaremos generalizações das soluções para o modelo fatorado.

1.1 Processo de Decisão Markoviano

Um *Processo de Decisão Markoviano* (*Markov Decision Process - MDP*) [Puterman (1994)] fornece um arcabouço matemático elegante para representar e resolver problemas de decisão sequencial sob incerteza em ambientes completamente observáveis. O processo é composto por estágios: em cada estágio, o agente escolhe uma ação que produz de forma estocástica um estado futuro e uma recompensa, conforme a Figura 1.1. Iremos denotar com superíndice ^{i} os valores das variáveis do estágio i, assim s^i, a^i e r^i são, respectivamente, o estado visitado, a ação escolhida e a recompensa recebida no estágio i. Neste modelo, o objetivo é encontrar políticas, funções que definem qual ação tomar em cada estado, que maximizem a recompensa acumulada esperada do agente, que chamaremos de *políticas ótimas*.

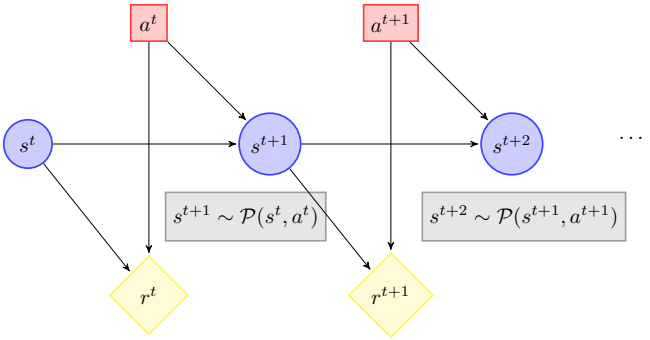


Figura 1.1: *Dinâmica de um Processo de Decisão Markoviano.*

Definição 1.1 *Um* **Processo de Decisão Markoviano (MDP)** *é definido pela tupla* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ *em que:*

- $\mathcal{S}$ *é um conjunto discreto e finito de estados completamente observáveis que modelam o mundo;*

- $\mathcal{A}$ *é um conjunto finito de ações;*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *é a função recompensa associada a cada par* $\langle$*estado, ação*$\rangle$ *e que representa as preferências do agente.*
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *é a função de probabilidade de transição sendo que* $\mathcal{P}(s, a, s')$ *é a probabilidade condicional do processo ir para o estado* s' *quando está no estado* s *e executa a ação* a*. Denotaremos por* $\mathcal{P}(s, a)$ *a distribuição de probabilidade do próximo estado.*

Estes processos são ditos markovianos, ou de Markov, pois respeitam a propriedade de Markov para processos estocásticos, isto é, o próximo estado depende apenas do estado atual e da ação escolhida, e independe de estados ou ações anteriores.

A seguir iremos apresentar as definições de funções e termos utilizados na definição de um MDP.

Definição 1.2 *Uma* **execução** *de um MDP* $\mathcal{M}$ *é descrita por três sequências:* $(s^0, s^1, \dots,)$*, onde* $s^t \in \mathcal{S}$ *é a variável aleatória que representa o estado inicial no estágio* i*;* $(a^0, a^1, \dots)$*, onde* $a^i \in \mathcal{A}$ *é a ação escolhida no estágio* i*;* $(r^0, r^1, \dots)$*, onde* $r^i \in \mathbb{R}$ *é a recompensa recebida no estágio* i*. Para as sequências satisfazerem o modelo* $\mathcal{M}$*, elas devem satisfazer as seguintes condições:*

$$\begin{aligned} Pr[s^{i+1} = s' \mid s^i = s,\ a^i = a] = &\quad \mathcal{P}(s, a, s')\forall s', s \in \mathcal{S}, a \in \mathcal{A} \\ r^i = &\quad \mathcal{R}(s^i, a^i) \end{aligned} \tag{1.1}$$

Definição 1.3 *Num MDP* $\mathcal{M}$*, uma* **política não-estacionária** *é uma função* $\pi : \mathcal{S} \times \mathbb{Z}^* \to \mathcal{A}$ *que define a ação* $\pi(s, h)$ *escolhida quando o processo está no estado* s *e no estágio* h*.*

Definição 1.4 *Definimos uma* **política estacionária** *por uma função* $\pi : \mathcal{S} \to \mathcal{A}$ *que define a ação* $\pi(s)$ *escolhida quando estamos no estado* s*. Ela é dita estacionária pois depende apenas do estado atual e não depende do estágio no qual o processo se encontra.*

Definição 1.5 *A* **execução de uma política** π *em* $\mathcal{M}$ *é uma execução de* $\mathcal{M}$ *onde para todo estágio* i*,* $a^i = \pi(s^i, i)$ *ou, se* π *é estacionária,* $a^i = \pi(s^i)$*. Denotaremos o valor esperado de uma variável* X *de* $\mathcal{M}$ *numa execução de* π *por* $E_\pi[X]$*.*

Definição 1.6 *Dizemos que um estado* s' *é* **alcançável em** k **passos** *a partir de um estado* s *por uma política* π *se na execução de* π *a partir de* s*, tivermos que* $Pr[s^k = s' | s^0 = s] > 0$*. Se existe um* $k \in \mathbb{Z}^*$ *tal que* s' *estado é alcançavel em* k *passos a partir de* s *por* π*, então dizemos que* s' *é* **alcançável** *a partir de* s *por* π*.*

Definição 1.7 *Uma* **política (não-estacionária) parcial** *é uma função* $\pi : X \subset \mathcal{S}(\times\mathbb{Z}) \to \mathcal{A}$ *que define a ação escolhida em alguns estados (e estágios), mas não necessariamente para todo o espaço de estados. Dizemos que uma política* π *é* **fechada** *em relação a um estado* s *(e estágio* h*) se* $\pi(s'(, h'))$ *está definida para todo estado* s' *alcançável (em* $h' - h$ *passos) a partir de* s *por* π*.*

Definição 1.8 *Dada uma política* π*, definimos a* **função valor não estacionária** *de* π*,* $V^\pi : \mathcal{S} \times \mathbb{Z}^* \to \mathbb{R}$ *que associa cada estado* $s \in \mathcal{S}$ *e número de passos* t *à recompensa acumulada esperada executando* π *a partir de* s *por* t *passos, ou seja:*

$$V^\pi(s, t) = E_\pi \left[\sum_{i=1}^{t} r^i \;\middle|\; s^0 = s \right] \tag{1.2}$$

Definição 1.9 *Dada uma política* π*, definimos a* **função valor estacionária** *de* π*,* $V^\pi : \mathcal{S} \to \mathbb{R}$ *que associa cada estado* $s \in \mathcal{S}$ *à recompensa acumulada esperada executando* π *a partir de* s *indefinidamente, ou seja:*

$$V^\pi(s) = E_\pi \left[\sum_{i=1}^{\infty} r^i \;\middle|\; s^0 = s \right] \tag{1.3}$$

Num MDP geral, conforme a Definição 1.1, não há garantia de que a recompensa obtida em (1.3) seja limitada, uma vez que o processo não tem condição de término. Para garantir que a recompensa máxima seja limitada e exista uma solução ótima é necessário impor restrições adicionais. Existem três modelos comuns baseados no MDP geral: *processo de decisão markoviano com horizonte finito*, *processo de decisão markoviano com horizonte infinito* e *processo de decisão markoviano com horizonte indefinido*. A seguir definiremos as condições adicionais de cada um desses modelos, bem como a expressão da recompensa acumulada e política ótima.

MDP com Horizonte Finito. Quando há um limite no número de ações que o agente deve executar, é interessante considerar o processo restrito a um numero finito de estágios.

Definição 1.10 *Um* **Processo de Decisão Markoviano com Horizonte Finito (MDPF)** *é uma tupla* $\mathcal{M}^{\mathcal{H}} = \{\mathcal{M}, \mathcal{H}\}$*, onde* $\mathcal{M}$ *é um MDP e* $\mathcal{H} \in \mathbb{N}^*$ *é o número total de estágios em que o agente deve atuar.*

Como num MDPF a quantidade de decisões é finita, o comportamento do agente passa a depender do estágio, e não só do estado. Assim, procuramos políticas não-estacionárias que maximizem funções valor não estacionárias. Como a função valor em (1.2) é uma soma finita de valores reais, sempre há pelo menos uma política que gera um valor máximo, chamamos qualquer dessas políticas de política ótima π^* e esse máximo de função valor ótima V^*, isto é:

$$V^*(s,h) = V^{\pi^*}(s,h) = \max_{\pi} V^{\pi}(s,h) \ \forall s \in \mathcal{S}, \forall h \in \{1,2,\ldots,\mathcal{H}\}. \tag{1.4}$$

MDP com horizonte indefinido. Também conhecido como caminho estocástico mínimo, modela problemas de planejamento, onde existe um conjunto de estados meta, que representam configurações terminais desejadas para o processo. O processo não tem horizonte finito, pois não sabemos a priori quantos passos levará para alcançar uma meta, mas toda execução termina em algum estado meta.

Definição 1.11 *Um problema de* **Processo de Decisão Markoviano com horizonte indefinido** *ou* **caminho estocástico mínimo (*Shortest Stochastic Path -SSP*)** *é uma tupla* $\mathcal{M}^{\mathcal{G}} = \{\mathcal{M}, \mathcal{G}\}$*, onde* $\mathcal{M}$ *é um MDP e* $\mathcal{G} \subset \mathcal{S}$ *é um conjunto de estados meta.*

Estados meta podem ser interpretados como estados absorventes em que a recompensa para qualquer ação é nula, ou seja, $\mathcal{P}(g,a,g) = 1$ e $\mathcal{R}(g,a) = 0 \ \forall g \in \mathcal{G}, a \in \mathcal{A}$. O processo termina ao alcançar um estado meta, portanto sua recompensa é finita. Além disso, para garantir que recompensa máxima é limitada, são feitas mais duas suposições:

- Para todo estado s existe uma **política própria**, isto é, uma política que quando executada a partir de s alcança algum estado meta $g \in \mathcal{G}$ com probabilidade 1.
- Toda **política imprópria**, ou seja, que não é própria, deve ter recompensa acumulada $-\infty$.

Desta forma, temos que o valor $V^{\pi_1}(s)$ de um política imprópria π_1 é sempre inferior ao valor $V^{\pi_2}(s)$de uma políticas própria π_2 para qualquer estado s. Portanto, a política que maximiza a recompensa é própria e tem função valor, conforme Equação 1.3, limitada. Assim, definimos função valor ótima e política ótima de forma semelhante ao MDPF, com a diferença que em MDPs com horizonte indefinido a política e função valor são estacionárias:

Definição 1.12 *A* **política ótima** *em um SSP* M^G *é uma política* π^* *tal que*

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}. \tag{1.5}$$

MDP com horizonte infinito. A terceira possibilidade é permitir que o processo nunca termine, mas nesse caso a somatória em (1.3) poderia ser indefinida. Para encontrar tornar essa soma finita é comum descontar as recompensas futuras, multiplicando-as por um fator de desconto $0 < \gamma < 1$, de modo que elas tenham cada vez menos influência para o valor de um estado.

Definição 1.13 *Um* **Processo de Decisão Markoviano Descontado - MDPD** *é uma tupla* $\mathcal{M}^{\gamma} = \{\mathcal{M}, \gamma\}$*, onde* $\mathcal{M}$ *é um MDP e* $0 < \gamma < 1$*.*

Definição 1.14 *A* **função valor descontada** V^{π}_{γ} *de uma política* π *é o valor esperado da recompensa acumulada descontada de sua execução, ou seja:*

$$V^{\pi}_{\gamma}(s) = E_{\pi}\left[\sum_{i=1}^{\infty} \gamma^i \cdot r^i \;\middle|\; s^0 = s\right] \tag{1.6}$$

A função recompensas é limitada, uma vez que seu domínio é finito. Assim, a somatória em (1.6) é limitada pela série trocando r^i pela recompensa máxima r^{max}:

$$E_\pi \left[\sum_{i=1}^{\infty} \gamma^i \cdot r^i \;\middle|\; s^0 = s \right] \leq \sum_{i=1}^{\infty} \gamma^i \cdot r^{max} \tag{1.7}$$

Usando o valor da série harmônica com razão γ:

$$V_\gamma^\pi(s) \leq \sum_{i=1}^{\infty} \gamma^i \cdot r^{max} = \frac{r^{max}}{(1-\gamma)} \tag{1.8}$$

Portanto, a função valor é limitada. Como o número de políticas é finito existe uma com valor máximo e podemos novamente definir função valor ótima e política ótima como a política que máxima V_γ^π e o máximo valor atingido.

Definição 1.15 *A* **política ótima** *em um MDPD $\mathcal{M}^\gamma$ é uma política π^* tal que*

$$V_\gamma^{\pi^*}(s) = \max_\pi V_\gamma^\pi(s) \;\forall s \in \mathcal{S}. \tag{1.9}$$

1.1.1 Solução de um MDP: A Equação de Bellman

Como vimos em nossos três modelos, resolver o MDP, ou encontrar uma política ótima, está associado ao cálculo da função valor, respectivamente nas Equações 1.2, 1.3 e 1.6. Vamos mostrar no caso de MDPFs, como desenvolvemos esse cálculo para obter uma forma mais simples na maximização. Pela definição de $V^\pi(s,h)$, Equação 1.2, e a linearidade do valor esperado de uma soma, temos que para qualquer política a recompensa total esperada em h passos é igual a soma da recompensa imediata mais a recompensa esperada nos próximos $h-1$ passos, ou seja:

$$\begin{aligned} V^\pi(s,h) &= E_\pi \left[\sum_{i=1}^{h} r^i \;\middle|\; s^0 = s \right] \\ V^\pi(s,h) &= E_\pi[r^1 \mid s^0 = s] + E_\pi \left[\sum_{i=2}^{h} r^i \;\middle|\; s^1 \sim \mathcal{P}(s, \pi(s,h)) \right] \\ V^\pi(s,h) &= \mathcal{R}(s,\pi(s,h)) + \sum_{s' \in S} \mathcal{P}(s,\pi(s,h),s') \cdot E_\pi \left[\sum_{i=2}^{h} r^i \;\middle|\; s^1 = s' \right] \end{aligned} \tag{1.10}$$

Como $\mathcal{P}$ e $\mathcal{R}$ são estacionárias, a somatória das recompensas para i de 2 a h com $s^1 = s'$ é equivalente a soma para i de 1 a $h-1$ com $s^0 = s'$, que tem a mesma forma da soma em (1.2), isto é:

$$E_\pi \left[\sum_{i=2}^{h} r^i \;\middle|\; s^1 = s' \right] = E_\pi \left[\sum_{i=1}^{h-1} r^i \;\middle|\; s^0 = s' \right] = V^\pi(s', h-1) \tag{1.11}$$

Substituindo (1.11) em (1.10) temos:

$$V^\pi(s,h) = \mathcal{R}(s,\pi(s,h)) + \sum_{s' \in S} \mathcal{P}(s,\pi(s,h),s') \cdot V^\pi(s', h-1) \tag{1.12}$$

O lado direito da Equação 1.12 depende diretamente da ação no estado e estágio atuais, $\pi(s,h)$, dependendo apenas indiretamente de outras ações de π. Por isso, é interessante representar separadamente essa ação do restante da política, e podemos avaliar o efeito de se mudar apenas esta ação.

Definição 1.16 *A* **qualidade** *$Q^V(s,h,a)$ de se executar uma ação a, num estado s, restando h estágios, em relação a uma determinada função valor V é a soma da recompensa imediata ao escolher a ação a com o valor esperado de V nos estados sucessores e tendo $h-1$ estágios para agir, isto é:*

$$Q^V(s,h,a) = R(s,a) + \sum_{s' \in S} P(s,a,s') \cdot V(s', h-1) \tag{1.13}$$

Definição 1.17 *A* **qualidade ótima** $Q^*(s,h,a)$ *de se executar uma ação* a*, em um estado* s*, com* h *estágios para atuar, é a qualidade dessa ação em relação a função valor ótima:*

$$Q^*(s,h,a) = Q^{V^*}(s,h,a) \tag{1.14}$$

Esta qualidade mede quão grande é a recompensa total acumulada por escolher uma ação a num estado s, considerando que as recompensas a partir dali são dadas por uma função valor V. Quando avaliada em relação a função valor ótima V^*, ela mede a qualidade ótima, que é a maior recompensa acumulada esperada que pode ser recebida se escolhemos esta ação para este estado, neste estágio. A definição de qualidade nos permite introduzir o conceito da melhor política em relação a uma função valor, conforme segue:

Definição 1.18 *Dada uma função valor* V*, a* **política gulosa** π^V *em relação a* V *é aquela que escolhe a ação de maior qualidade em cada estado, i.e. :*

$$\pi^V(s,h) \leftarrow \arg\max_{a \in A} Q^V(s,h,a) \ \forall s \in \mathcal{S} \tag{1.15}$$

Vamos mostrar como usando (1.12) na definição de valor ótimo para horizonte finito, Equação 1.4, obtemos uma expressão que calcula o valor ótimo a partir da qualidade ótima:

$$V^{\pi^*}(s,h) = \max_{\pi} \left[\mathcal{R}(s,\pi(s,h)) + \sum_{s' \in S} \mathcal{P}(s,\pi(s,h),s') \cdot V^{\pi}(s',h-1) \right]$$

Separando π em $\pi(s,h)$ e π^-, definido por $\pi^-(s'',h'') = \pi(s'',h'') \quad \forall (s'',h'') \neq (s,h)$:

$$V^{\pi^*}(s,h) = \max_{\pi(s,h),\pi^-} \left[\mathcal{R}(s,\pi(s,h)) + \sum_{s' \in S} \mathcal{P}(s,\pi(s,h),s') \cdot V^{\pi^-}(s',h-1) \right]$$

Usando que $V^{\pi^-}(s',h')$ independe de $\pi(s,h)$ e maximizando em π^-:

$$V^{\pi^*}(s,h) = \max_{\pi(s,h)} \left[\mathcal{R}(s,\pi(s,h)) + \sum_{s' \in S} \mathcal{P}(s,\pi(s,h),s') \cdot V^*(s',h-1) \right]$$

Pela definição de qualidade, (1.13):

$$V^{\pi^*}(s,h) = \max_{\pi(s,h)} [Q^*(s,h,\pi(s,h))]$$

Finalmente, substituindo $\pi(s,h)$ por uma ação qualquer a:

$$V^*(s,h) = \max_{a \in A} Q^*(s,h,a) \tag{1.16}$$

$$\pi^*(s,h) \in \arg\max_{a \in A} Q^*(s,h,a) \tag{1.17}$$

1.1.2 Programação Dinâmica para MDP: o Algoritmo Iteração de Valor

As Equações 1.16 e 1.17 indicam um caminho para resolver o MDP. A seguir, explicaremos a primeira técnica utilizada para resolver MDPs, que usa programação dinâmica e é conhecida como **Iteração de Valor - IV** [Bellman (1957)]. A técnica de programação dinâmica consiste em reduzir um problema em subproblemas mais simples, resolver os subproblemas e armazenar sua solução e então a partir das solução armazenadas para os subproblemas construímos rapidamente a solução do problema original. Essa técnica é mais eficiente que uma divisão simples pois armazenando as soluções dos subproblemas evita-se que eles tenham que ser resolvidos mais de uma vez [Cormen *et al.* (2001)].

A estratégia de divisão em subproblemas neste caso funciona assim: para calcular o valor ótimo de todos os estados restando h estágios, iremos primeiro obter os valores ótimos de todos os estados onde restam $h-1$ escolhas. Para fazer isso, notamos que a Equação 1.13, permite calcular a qualidade ótima $Q^*(\cdot,h,\cdot)$[1] restando h passos usando apenas a função valor ótima $V^*(\cdot,h-1)$, i. e. para $h-1$ passos. Além disso, podemos obter $V^*(\cdot,h)$ e $\pi^*(\cdot,h)$ a partir de $Q^*(\cdot,h)$, pelas Equações 1.16 e 1.17. Assim, combinamos as Equações 1.13 e 1.16 para obter a *Equação de Bellman para Horizonte Finito*:

$$V^*(s,h) = \max_{a \in A} \left[R(s,a) + \sum_{s' \in S} P(s,a,s') \cdot V^*(s',h-1) \right] \tag{1.18}$$

[1] Para funções de um ou mais parâmetros, iremos denotar "$F(x,y)\ \forall x$" por "$F(\cdot,y)$".

A Equação 1.18 permite estender de $V^*(\cdot, h-1)$ para $V^*(\cdot, h)$. Além disso, a função valor ótima para $h = 0$ é trivial, $V(\cdot, 0) = 0$, já que nenhuma recompensa é recebida. Portanto, é possível se obter $V(\cdot, 1)$, $V(\cdot, 2)$, ..., $V(\cdot, \mathcal{H})$ utilizando a Equação 1.18 repetidamente. Como estamos encontrando os valores da função valor não estacionária V por horizonte, podemos tratá-la como um conjunto de $\mathcal{H}$ funções valor estacionárias $V^h = V(\cdot, h)$, para $h \in \{1, \ldots, \mathcal{H}\}$. Assim, definimos uma operação que usa uma função valor estacionária V no lado direito de (1.18) e atribuir esse resultado a outra função estacionária V' no lado esquerdo, que é chamada de *Atualização de Bellman*.

$$V'(s) \leftarrow \max_{a \in A} \left[R(s,a) + \sum_{s' \in S} P(s,a,s') \cdot V(s') \right] \tag{1.19}$$

Como vimos, no caso de usarmos a função valor ótima $V^*(\cdot, h-1)$, logo no lado direito obtemos a função valor ótima $V^*(\cdot, h)$. Simultaneamente, obtemos a política ótima observando quais foram as ações escolhidas nas sucessivas maximizações.

O Algoritmo IV encontra o valor ótimo e política ótima para MDPs de horizonte finito fazendo uma iteração para cada estágio e seu pseudocódigo é mostrado no Algoritmo 1.1. O código consiste de inicializar a função valor V^0, linhas 1-2, e realizar $\mathcal{H}$ atualizações de todos estados, linhas 3-5. A função `Atualização_de_BELLMAN` faz uma atribuição de Equação 1.19 para um estado s e também retorna a ação escolhida na maximização.

Algoritmo 1.1: Iteração_de_Valor_MDPF ($\mathcal{M}$, $\mathcal{H}$)

Entrada: $\mathcal{M}$: um MDP, $\mathcal{H}$: o horizonte de interesse
Saída: $V^{\mathcal{H}}$: a função valor ótima, π: uma política não-estacionária ótima completa

1 **para cada** $s \in \mathcal{S}$ **faça**
2 $\quad V^0(s) \leftarrow 0$
3 **para** $h = 1$ *até* $\mathcal{H}$ **faça**
4 $\quad$ **para cada** $s \in \mathcal{S}$ **faça**
5 $\quad\quad V^h(s), \pi(s,h) \leftarrow$ `Atualização_de_BELLMAN`$(\mathcal{M}, s, V^{h-1})$
6 **retorna** $V^{\mathcal{H}}, \pi$.

Função `Atualização_de_Bellman` ($\mathcal{M}$, s, V)

Entrada: $\mathcal{M}$: um MDP, s: estado a ser atualizado, V função valor conhecida
Saída: V': a função valor atualizada, a': ação que maximiza a recompensa esperada por V

1 **para cada** $a \in \mathcal{A}$ **faça**
2 $\quad Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$ // $\gamma = 1$ para MDPFs
3 $V' \leftarrow \max_{a \in A} Q(s,a)$
4 $a' \leftarrow \arg\max_{a \in A} Q(s,a)$
5 **retorna** V', a'.

A solução de um MDP com horizonte infinito pode ser obtida como uma generalização da solução para horizonte finito. Nesse caso, devido ao fator de desconto γ, as recompensas futuras são cada vez menos relevantes, e a solução para um horizonte grande irá ser muito semelhante a do horizonte infinito. Como num horizonte infinito o número de estágios restantes é sempre o mesmo, uma quantidade infinita, não há por que usar funções valor ou políticas não estacionárias. De fato sempre existe uma política ótima estacionária [Howard (1960)]. Removendo a dependência dos passos restantes (h) da Equação 1.18, obtemos a *Equação de Bellman para Horizonte Infinito*:

$$V^*(s) = max_{a \in A}(Q^*(s,a)) = max_{a \in A} \left(R(s,a) + \sum_{s' \in S} \gamma \cdot P(s,a,s') \cdot V^*(s') \right) \forall s \in \mathcal{S} \tag{1.20}$$

Como temos $|\mathcal{S}|$ variáveis (o valor de cada estado) e $|\mathcal{S}|$ equações ((1.20) para cada estado), podemos encontrar a solução desse sistema com programação matemática. No entanto, por se tratarem de equações

não-lineares, devido ao operador max, o método que usaremos para resolvê-las será iterativo.

Utilizaremos agora as atualizações de Bellman de Equação 1.19 para atualizar uma só função valor, e com o desconto γ, já que para MDPDs ele é parte do cálculo da recompensa acumulada.

$$V(s) \leftarrow \max_{a \in A} \left[R(s,a) + \gamma \cdot \sum_{s' \in S} P(s,a,s') \cdot V(s') \right] \forall s \in \mathcal{S} \tag{1.21}$$

Pode se imaginar que a função V é a solução para um horizonte cada vez maior para cada vez que ela é atualizada, mas queremos a solução para um horizonte infinito, que neste caso seria o limite dessas aplicações. Foi demonstrado por Bellman (1957) que o único ponto fixo desta aplicação é a função valor ótima V^*. E portanto partindo de uma função V arbitrária e aplicando a Equação 1.21 repetidamente a função valor V converge para V^*. Como um critério de convergência, é comum executar as atualizações enquanto a maior diferença ocorrida em V numa atualização for significativa (maior que um fator de precisão ϵ). O Algoritmo 1.3 mostra o código da iteração de valor para MDPDs.

A variável Erro é usada para detectar a convergência, a função V' é usada como memória auxiliar para calcular as diferenças numa atualização e atualizar todos estados de forma síncrona.

Algoritmo 1.3: Iteração_de_Valor ($\mathcal{M}$, ϵ)

Entrada: $\mathcal{M}$: um MDP, ϵ: a precisão da solução gerada
Saída: V^*: a função valor ótima, π: uma política estacionária ótima completa

```
1  para cada s ∈ S faça
2      V(s) =0
3  Erro ← ∞
4  enquanto Erro > ε faça
5      Erro ← 0
6      para cada s ∈ S faça
7          V'(s), π(s) ← Atualização_de_BELLMAN(M, s, V)
8          Erro ← MAX(Erro, |V'(s) − V(s)|)
9      V ← V'
10 retorna V, π.
```

Uma outra solução tradicional para MDPs, que também se baseia em programação dinâmica, é o Algoritmo *Iteração de Política* [Howard (1960)]. Essa solução também utiliza atualizações de Bellman, porém ela mantém e atualiza políticas no lugar de funções valor e cada iteração é separadas em duas fases. A primeira etapa de cada iteração é a **avaliação da política**, que funciona como (1.21) mas sem a maximização pois a política está fixa:

$$V(s) \leftarrow \left[R(s,\pi(s)) + \gamma \cdot \sum_{s' \in S} P(s,\pi(s),s') \cdot V(s') \right] \forall s \in \mathcal{S} \tag{1.22}$$

A avaliação de π consiste em obter um ponto fixo da atualização de Bellman simplificada (1.22), ou resolver diretamente o sistema de $|\mathcal{S}|$ variáveis e equações relacionado. Após avaliar a política, ela é atualizada para a política que maximiza as recompensas segundo V, a política gulosa, conforme a Definição 1.18. Assim, inicia-se uma nova iteracão com a política π atualizada. Como a única política que é gulosa em relação a função valor induzida por ela é uma política ótima, essa sequência de iterações converge para a política ótima [Howard (1960)].

1.1.3 Programação Dinâmica Assíncrona: o Algoritmo RTDP

As soluções tradicionais para MDPs, como Iteração de Valor (Algoritmo 1.3), geram a política ótima completa. Porém, para problemas com muitos estados e com transições relativamente esparsas, i.e. transições que levam a poucos estados com probabilidade positiva, encontrar a política ótima em todo o espaço é muito mais custoso do que apenas a política ótima parcial fechada em relação a um dado estado inicial s_0. Isso justifica a criação de algoritmos mais eficientes que se baseiam em busca heurística e amostragem para encontrar soluções considerando apenas estados alcançáveis a partir do estado inicial conhecido, s_0.

O algoritmo *RTDP* (*Real Time Dynamic Programming - RTDP*) de Barto *et al.* (1995) é baseado em programação dinâmica, busca heurística e amostragem.

Uma *heurística* é uma função $h : \mathcal{S} \to \mathbb{R}$ que estima a recompensa acumulada esperada de um estado, ela é dita uma *heurística admissível* se não subestima a recompensa ótima, ou seja $h(s) \geq V^*(s)\ \forall s \in \mathcal{S}$.

O algoritmo RTDP funciona iniciando com uma função valor baseada em uma heurística admissível ($V(s) = h(s)$) e prossegue melhorando a estimativa repetidamente até que a política gulosa em relação a V, π^V, seja ótima e fechada em relação a s_0. Uma política π é ótima e fechada em relação a um estado s_0 se a função valor dessa política satisfaz a equação de Bellman, (1.20), em todos estados alcançáveis a partir de s_0 segundo ela mesma, π. Para melhorar a sua estimativa V, RTDP realiza simulações (*trials*) de sequências de execução de ações partindo do estado inicial e sorteando os próximos estados de acordo com a ação gulosa segundo a função valor atual. Os estados percorridos pelos *trials* têm seus valores atualizados de acordo com a atualização de Bellman, Equação 1.21.

O pseudocódigo do algoritmo RTDP para SSPs é apresentado no Algoritmo 1.4. Ele é facilmente ajustado para os outros modelos com pequenas mudanças na condição de parada dos *trials*.

Algoritmo 1.4: RTDP ($\mathcal{M}$, s_0, G)

Entrada: $\mathcal{M}$: um MDP, s_0: um estado inicial,
G: um conjunto de estados meta
Saída: π: uma política ótima fechada em relação a s_0

```
1  para cada s ∈ S faça
2      V(s) = h(s) //Inicialize V com heurística admissível
3  enquanto tempoUsado < LimiteDeTempo faça
4      //Inicia o trial no estado inicial
5      s ← s_0
6      enquanto (s ∉ G) //Enquanto não atingimos a meta
7      faça
8          // Atualiza o estado s e a política π(s)
9          V(s), π(s) ← Atualização_de_BELLMAN(M, s, V)
10         // Progresso do trial
11         s ← SorteiaProximoEstado(s, π(s))
12 retorna π.
```

Atualizar apenas os estados nos caminhos mais promissores traz ao RTDP vantagens sobre o algoritmo IV, a saber: (a) quando interrompido antes da convergência geralmente produz uma solução boa para o estado inicial; (b) possui garantia de otimalidade sem exploração exaustiva do espaço de estados, ao concentrar sua pesquisa sobre estados alcançáveis a partir do estado inicial.

Barto *et al.* (1995) mostrou que o algoritmo RTDP resolve SSPs (Definição 1.11) encontrando uma política parcial fechada ótima se satisfizermos duas suposições :

i. $V(s)$ deve ser inicializado com uma heurística admissível para todo $s \in \mathcal{S}$; e

ii. a partir de todo estado s existe um estado meta ($s' \in G$) alcançável a partir de s segundo qualquer política, o que é conhecida como *hipótese de alcançabilidade*.

Essas suposições garantem que *trials* do RTDP não continuarão indefinidamente e que eventualmente alcançam um estado meta. No caso de horizonte finito é claro que os *trials* acabam, quando chegam na profundidade igual ao horizonte máximo $\mathcal{H}$. Para MDPs de horizonte infinito, o fator de desconto γ é usado como a probabilidade do processo não ser interrompido: todo *trial* é interrompido com probabilidade $1 - \gamma$.

Uma dificuldade para o RTDP está em verificar e sua convergência, e evitar atualizar inutilmente valores de estados que já convergiram. Para tratar esses problemas, foram propostas algumas extensões para o RTDP: Labeled RTDP (LRTDP) em Bonet e Geffner (2003); Bounded RTDP (BRTDP) em McMahan *et al.* (2005); e Focused RTDP (FRTDP) em Smith e Simmons (2006), que orientam os *trials* para estados ainda não convergidos e cuja estimativa está menos precisa de modo a acelerar a convergência.

1.2 Processo de Decisão Markoviano Fatorado

Até agora consideramos processos de decisão markovianos que não consideram a estrutura interna de seus estados, apenas que eles são finitos e podem ser enumerados, por isso chamamos esse modelo também de *MDP enumerativo*. Nesta seção, nos concentramos numa representação mais prática para um MDP, que representa estados por variáveis de estado, chamada de **processo de decisão markoviano fatorado**. Versões fatoradas de processos de decisão markovianos têm sido bastante estudadas [Boutilier *et al.* (1999, 1995); Guestrin *et al.* (2011)], particularmente nas soluções de problemas grandes de planejamento, tratados em Inteligência Artificial.

Exemplo 2 O problema do administrador de sistemas [Guestrin *et al.* (2001)]. *Dada uma rede de computadores, o objetivo do administrador é manter os computadores funcionando. No entanto, a cada passo existe uma probabilidade de que um computador pare de funcionar, e esta probabilidade depende do estado dos computadores vizinhos. Quanto mais vizinhos estiverem funcionando, maior a chance de um computador continuar funcionando. O administrador deve escolher reiniciar computadores para fazer com que eles voltem a funcionar, mas pode reiniciar no máximo um por passo, de modo que é importante que o administrador priorize reiniciar computadores de maior impacto na rede.*

Ainda que o problema do Exemplo 2 possa ser transformado num MDP com a estrutura enumerativa vista nas seções anteriores, o modelo fatorado será muito mais adequado para representá-lo.

Num *MDP fatorado*, o conjunto de estados $\mathcal{S}$ é descrito por um conjunto $\vec{X} = \{X_1, X_2, ..., X_n\}$ de *variáveis de estado*. Um estado $s \in \mathcal{S}$ é representado como um vetor $\vec{x} = (x_1, x_2, ..., x_n)$, em que x_i é o valor da variável de estado X_i. Nessa representação, usaremos a notação $\vec{x} \in \mathcal{S}$. Como existem 2^n valorações distintas para n variáveis binárias, uma representação enumerativa de $\mathcal{S}$ teria um número de estados que cresce de forma exponencial com o número de variáveis. Como a representação fatorada permite uma descrição muito mais compacta de problemas de planejamento, *algoritmos polinomiais no número de estados são exponenciais na nova representação da entrada.*

Modelamos o espaço de estados do Exemplo 2 da seguinte forma: temos n computadores e cada variável binária X_i representa o estado do computador i, se $X_i = 1$; o computador i está funcionando, caso contrário ($X_i = 0$) ele não está funcionando. Quanto às ações, temos $n + 1$ ações, sendo n ações de reiniciar (a ação a_i reinicia o computador i, para i de 1 a n) mais a ação de não reiniciar nenhum computador, $a_\emptyset$, também chamada de *no-op*;

Além disso, em modelos fatorados, é comum supor que existe uma relação de dependência estrutural entre as variáveis e as funções de probabilidade de transição $\mathcal{P}$ e de recompensa $\mathcal{R}$. Essa estrutura de depêndencias pode ser utilizada para representar as funções e computar soluções de uma forma muito mais eficiente. Nas seções seguintes iremos descrever os modelos de recompensa e transição fatorados.

1.2.1 Modelo de Recompensa Fatorado

Ainda que a recompensa possa depender de todas as variáveis, ela, em geral, é composta por termos locais que dependem apenas de algumas variáveis. Assim, o modelo de recompensa fatorado permite evidenciar essa característica por funções de recompensa local.

Definição 1.19 *Uma* **função de recompensa fatorada** $\mathcal{R}(\vec{x}, a)$ *de um MDP fatorado é a soma de* N_R *funções de recompensa local* $R_j(\vec{x}, a)$*:*

$$\mathcal{R}(\vec{x}, a) = \sum_{j=1}^{N_R} R^j(\vec{x}, a). \tag{1.23}$$

Definição 1.20 *Uma* **função de recompensa local** $R^j(\vec{x}, a)$ *é definida por um subconjunto de variáveis de estado* $\vec{X}^j \subset \vec{X}$ *e uma função* $r^j : \vec{X}^j \times \mathcal{A} \to \mathbb{R}$ *tal que*

$$R^j(\vec{x}, a) = r^j(\vec{x}|_{\vec{X}^j}, a), \tag{1.24}$$

onde $\vec{x}|_{\vec{X}^j}$ *é a restrição do vetor* $\vec{x}$ *às variáveis* $\vec{X}^j$*.*

Essa subdivisão em funções locais muitas vezes permite uma representação muito mais compacta. Por exemplo, para representar duas funções de k variáveis binárias são necessários $2 \cdot 2^k (= 2^{k+1})$ valores, enquanto para representar uma função de $2k$ variáveis são necessários 2^{2k} valores.

No exemplo do administrador, Exemplo 2, a recompensa é definida como a quantidade de computadores que estão ligados menos o custo de reiniciar um computador. Temos n funções de recompensa local $R^j(\vec{x}, a)$,

que avaliam se o computador j está funcionando, mais uma função local, $R^{n+1}(\vec{x}, a)$, que depende só da ação, e que representa o custo de reiniciar um computador, ou seja,

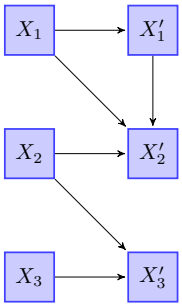
$$R^j(\vec{x}, a) = (\{X_j\}, x_j), \text{ para } 1 \leq j \leq n \text{ e}$$

$$R^{n+1}(\vec{x}, a) = \begin{cases} a \neq a_\emptyset : & -0.5 \\ a = a_\emptyset : & 0 \end{cases}, \text{onde } a_\emptyset \text{é a ação } \textit{no-op}$$

1.2.2 Modelo de Transição Fatorado: Redes Bayesianas Dinâmicas

Num MDP fatorado as probabilidades de transição de estado são representadas usando **Redes Bayesianas Dinâmicas** (**DBNs**)(Dean e Kanazawa (1990)).

Definição 1.21 *Uma* **Rede Bayesiana Dinâmica** *é um grafo dirigido acíclico com duas camadas: uma camada representa as variáveis de estado atual e a outra camada representa as variáveis do próximo estado, arcos entre os nós representam que a variável onde o arco termina é dependente da variável de onde o arco começa.*



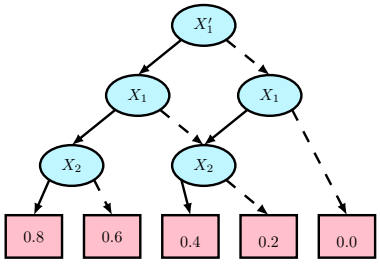
(a)

$P(X_1'\|X_1)$	$X_1 = 0$	$X_1 = 1$
$X_1' = 0$	0.9	0.2
$X_1' = 1$	0.1	0.8

(b)

$P(X_2' = 1\|X_1, X_2, X_1')$	$X_1' = 0$	$X_1' = 1$
$X_1 = 0, X_2 = 0$	0.0	0.2
$X_1 = 0, X_2 = 1$	0.0	0.4
$X_1 = 1, X_2 = 0$	0.2	0.6
$X_1 = 1, X_2 = 1$	0.4	0.8

(c)



(d)

Figura 1.2: *a) Uma Rede Bayesiana Dinâmica (DBN) para uma ação $a \in A$. b) Tabela de probabilidade condicional para X_1'. c) Tabela de probabilidade condicional para $X_2' = 1$. d) Diagrama de Decisão Algébrico (ADD) representando a probabilidade condicional de $X_2' = 1$ de (c); a linha contínua indica o ramo verdadeiro da variável de teste e a linha tracejada indica o ramo falso.*

Os nós da primeira e da segunda camada são denotados por X_i e X_i', respectivamente, representando a variável i no estado atual e no próximo estado. Arcos são permitidos de nós da primeira camada para a segunda camada, e também entre nós da segunda camada, desde que não se formem ciclos. Denotamos por $Pais(X_i')$ os pais de X_i' no grafo, isto é, nós de onde partem arcos que chegam em X_i'. Por exemplo, na Figura 1.2.a, $Pais(X_1') = \{X_1\}$, $Pais(X_2') = \{X_1, X_2, X_1'\}$ e $Pais(X_3') = \{X_2, X_3\}$. Numa Rede Bayesiana, cada vértice é uma variável aleatória, e vale a seguinte condição: uma variável X é condicionalmente independente de seus não-descendentes não-pais dados seus pais. Isso permite a seguinte fatoração das probabilidades de transição:

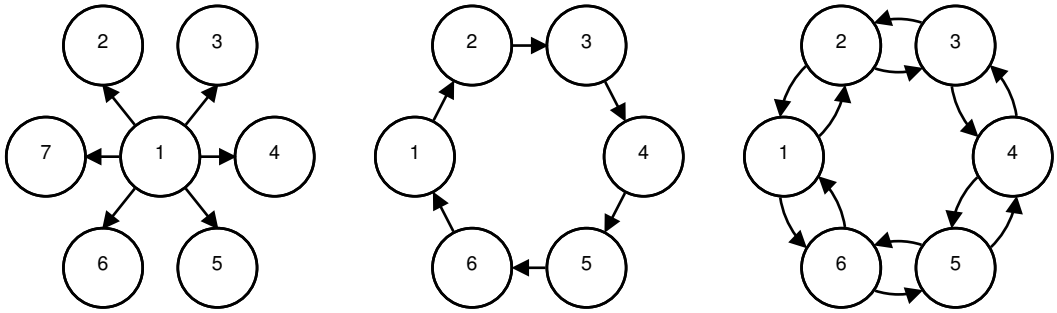
$$P(\vec{x}'|\vec{x}, a) = \prod_{i=1}^{n} P(x_i'|Pais(X_i'), a), \tag{1.25}$$

isto é, a probabilidade de ir para $\vec{x}' \in \mathcal{S}$, dado que o agente está no estado $\vec{x} \in \mathcal{S}$ e executa a ação $a \in \mathcal{A}$, é o produto das probabilidades condicionais para cada variável $(X_i' = x_i')$ dados os valores de seus pais. Como em geral o conjunto de pais de uma variável é bem menor que $\vec{X}$, a representação fatorada é eficiente por armazenar funções de menos parâmetros.

No exemplo do administrador de sistemas, é comum considerar diversos tipos de topologia na rede de computadores, como visto na Figura 1.3. Para cada organização da rede, a rede bayesiana dinâmica das transições é diferente, e cada conexão entre os computadores é causa um arco entre as variáveis correspondentes na DBN.

1.2.3 Representação fatorada eficiente: Diagramas de Decisão Algébricos

Para representar de maneira eficiente as DBNs, no lugar de armazenar tabelas (Figura 1.2.b e c) para cada probabilidade condicional $P(x_i'|PaisX_i', a)$ é interessante utilizar uma estrutura mais compacta: os *diagramas de decisão algébricos* [Bahar *et al.* (1993)].



(a) *Estrela* **(b)** *Anel* **(c)** *Anel Bidirecional*

Figura 1.3: *Exemplos de topologias da rede de computadores para o problema do Administrador de Sistemas Guestrin et al. (2011). Os arcos indicam que o computador de entrada depende do computador de saída. a) Topologia de estrela; b) Topologia em Anel; c)Anel Bidirecional.*

Definição 1.22 *Um* **Diagramas de Decisão Algébrico (Algebraic Decision Diagram - ADD)** *é um grafo acíclico dirigido com dois tipos de nós: nós internos ou de decisão, que contém uma variável X_i; e nós terminais ou folhas, que contém um valor real. Um ADD F, com k variáveis booleanas em seus nós decisão, representa uma função $f : [0,1]^k \to \mathbb{R}$ e a maneira de avaliar a função $f(x_1, \ldots, x_k)$ é percorrendo o grafo de F da seguinte forma:*

- *O caminho sempre inicia na raiz de F, usualmente representada no topo;*
- *Se estamos num nó de decisão X_i, verificamos a atribuição da variável x_i, caso seja 1, seguimos pelo arco "verdadeiro", representado por uma seta contínua, caso seja 0 seguimos pelo arco "falso", representado por uma seta tracejada. Note que as variáveis booleanas são ordenadas, de forma que nenhum caminho passa duas vezes por uma mesma variável e se uma variável vem antes de outra em um caminho, essa ordem nunca será invertida em outro caminho do mesmo ADD.*
- *Quando chegamos num nó terminal r, este é o valor de $f(x_1, \ldots, x_k)$.*

Note que não é necessário testar todas as variáveis e que caminhos com decisões diferentes podem chegar ao mesmo nó, o que permite uma representação mais compacta do que uma árvore de decisão. Veja Figura 1.2.d.

Além de representar funções de forma compacta, os ADDs que usam a mesma ordem de variáveis são especialmente úteis por permitirem que operações algébricas entre eles sejam feitas de forma eficiente, aproveitando resultados de operações já realizadas e evitando cálculos repetidos que ocorreriam numa representação enumerativa ou de tabela. Veja um exemplo de operação entre ADDs na Figura 1.4.

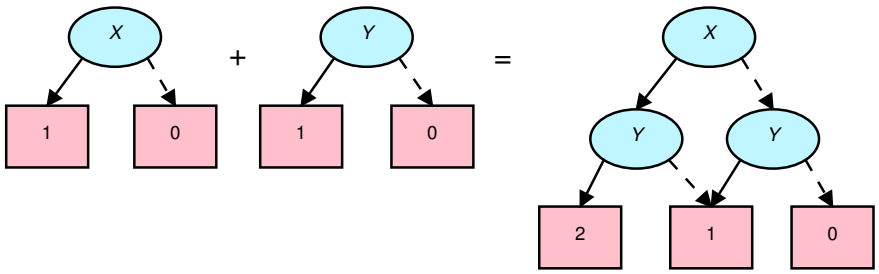


Figura 1.4: *Exemplo de uma operação de soma entre ADDs representando funções de variáveis booleanas. As elipses azuis são nós de decisão, onde X e Y são variáveis booleanas, os retângulos rosas são nós terminais com valores reais. Setas com linha contínua indicam o ramo seguido se o valor da variável booleana é "verdadeiro", e setas tracejadas indicam o ramo seguido se o valor da variável booleana é "falso".*

Outras duas operações importantes que são realizadas em ADDs são a *restrição* e a *maximização*. Uma ADD F restrita por $Z = 1$, denotado $F|_{Z=1}$ é o diagrama resultante trocando os nós de decisão que dependem de Z pelo nó apontado pelo seu ramo verdadeiro. Veja Figura 1.5.

O ADD $F_M = max(F_1, F_2)$ resultado da maximização de dois ADDs, F_1 e F_2, é o ADD cujas folhas tem o valor máximo entre as folhas de F_1 e F_2 para cada caminho. Veja Figura 1.6.

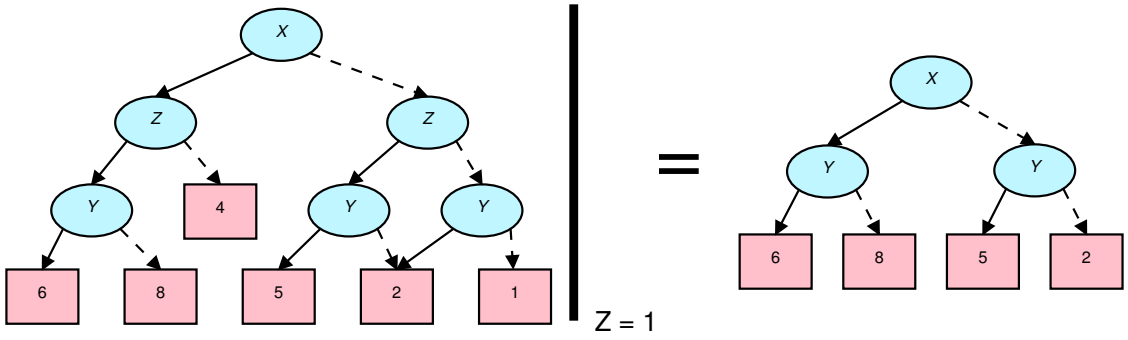


Figura 1.5: *Exemplo de uma restrição: um ADD com três variáveis, X, Y e Z, restrito por $Z = 1$ a um ADD com apenas X e Y.*

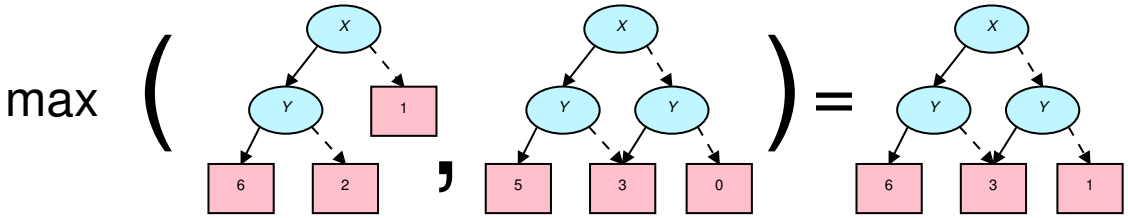


Figura 1.6: *Exemplo de uma maximização de ADDs. A folha num caminho do ADD máximo sempre vale o mesmo que o máximo entre as folhas correspondentes nos outros diagramas.*

1.3 Programação Dinâmica Simbólica com Variáveis Discretas

O algoritmo de programação dinâmica para planejamento probabilístico, Iteração de Valor, foi apresentado desconsiderando as dependências entre os estados, e supondo representação completa de todos estados. Dessa forma, mesmo que os valores forem representados de forma fatorada eficiente, como cada estado é atualizado independentemente, é necessário uma quantidade exponencial de atualizações. Nesta seção vamos apresentar a técnica de **Programação Dinâmica Simbólica** que realiza uma atualização simbólica, atualizando de uma só vez os valores de todos os estados. Na seção seguinte, descrevemos a versão original desta técnica para planejamento probabilístico com variáveis discretas; em seguida, apresentamos a versão aproximada com variáveis discretas; finalmente, as soluções para planejamento com variáveis contínuas são apresentadas no Capítulo 2.

1.3.1 Programação Dinâmica Simbólica Exata com Variáveis Discretas

A eficiência da representação fatorada para problemas estruturados pode ser utilizada para realizar o planejamento de forma mais eficiente. O algoritmo *Stochatic Planning Using Decision Diagrams - SPUDD* de Hoey *et al.* (1999) utiliza a mesma idéia de programação dinâmica que Iteração de Valor, porém, representando as funções valor, recompensa e probabilidade através de ADDs. A equação de Bellman, Equação 1.20, é reescrita em termos de ADDs:

$$V^*_{DD}(\vec{x}) = max_{a \in A} \left(\sum_{j=1}^{\phi} R^j_{DD}(\vec{x}, a) + \sum_{\vec{x'} \in S} \gamma \cdot \prod_{i=1}^{n} P_{DD}(x'_i | Pais(X'_i), a) \cdot V^*_{DD}(\vec{x'}) \right) \quad (1.26)$$

O aproveitamento da dependência local, apenas subconjuntos de $\vec{X}$, permite que os cálculos sejam eficientes em tempo e espaço mesmo quando representar o valor de todos estados enumeráveis seria inviável. Note que como as operações usadas em (1.26) são definidas para ADDs, a atualização de Bellman é aplicada simbolicamente para todos estados. Assim, o algoritmo SPUDD é uma generalização direta do IV, apenas trocando a atualização de cada estado pela atualização do ADD de toda função valor.

1.3.2 Programação Dinâmica Símbolica Aproximada com Variáveis Discretas

A representação fatorada com diagramas de decisão usada pelo SPUDD, permite a resolução de problemas muito maiores que os enumerativos, mas ainda apresenta dificuldade no caso de a função valor ter muitos valores distintos, exigindo testar muitas variáveis para ser percorrida. Nessas condições, para tornar eficiente a resolução de problemas com muitas variáveis, é interessante considerar uma solução aproximada com erro limitado. Podemos reduzir, ou até mesmo limitar, o número de nós terminais do XADD se unirmos os nós com valores próximos.

A versão aproximada do SPUDD foi chamada APRICODD [St-Aubin *et al.* (2000)]. Assim, entre as iterações de retropropagação é executada uma etapa de compactação com erro limitado por uma precisão ϵ.

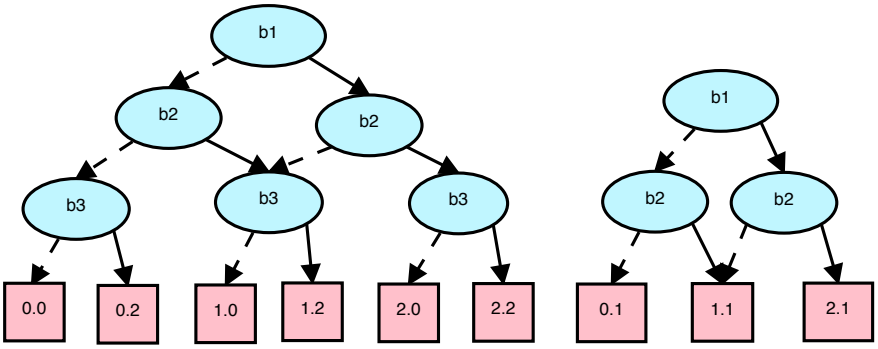


Figura 1.7: *Exemplo da técnica de aproximação de ADDs pela fusão de nós terminais próximos substituindo-os pelo valor médio, neste caso, gerando um erro de 0.1.*

Capítulo 2

Planejamento Probabilístico com Variáveis Contínuas

Domínios de planejamento muitas vezes envolvem recursos que são descritos naturalmente por variáveis contínuas, como a localização de um veículo, a carga de uma bateria ou o volume de água num reservatório. Para esse tipo de problema, o modelo MDP com variáveis discretas que vimos no Capítulo 1 não é apropriado. Neste capítulo iremos introduzir a modelagem utilizada para problemas de planejamento com variáveis discretas e contínuas, tal modelo é chamado de **Processo de Decisão Markoviano Híbrido**. Apresentaremo o algoritmo de programação dinâmica simbólica exata para HMDPs- Symbolic Dynamic Programming -SDP que resolve este problema de maneira exata usando técnicas de representação simbólica para tratar as variáveis contínuas. Como esse algoritmo é a base para os novos algoritmos propostos nesse mestrado, explicaremos em detalhes seu funcionamento, bem como a estrutura de dados utilizada, o *Diagrama de Decisão Algébrico Estendido (eXtended Algebraic Decision Diagram - XADD)*, que traz uma contribuição importante para a eficiência da programação dinâmica simbólica. No final do capítulo, discutimos sobre os problemas da solução SDPoutras soluções para HMDPs e quais trabalhos correlatos e outras soluõescomo a nossa proposta estende a solução de programação dinâmica simbólica com variáveis contínuas.

2.1 Processo Markoviano de Decisão Híbrido

O processo de decisão markoviano é o modelo matemático para tomada de decisão sequencial em situações estocásticas. Adicionar variáveis contínuas permite que o planejador tome decisões dependentes dos valores quantitativos de recursos, sem necessitar discretizá-los. Além disso, ele permite que a ação executada tenha parâmetros contínuos, o que é essencial para o controle preciso das variáveis contínuas.

Um exemplo clássico de variável contínua é o tempo. Para tratar o tempo em problemas de decisão sequencial, é necessário definir como os estágios onde as decisões são tomadas correspondem a instantes no tempo. Isso pode ser feito de maneira *a priori*, pela discretização do tempo, isto é, tornando o tempo uma sequência de passos uniformemente separados, mas neste caso perdemos a possibilidade de atuar por intervalos menores que o passo de tempo, bem como de iniciar ações entre os passos. Uma outra maneira é utilizar uma variável contínua para representar o tempo, neste caso podemos incluir como um parâmetro de cada decisão o intervalo de atuação, e a nova decisão será tomada imediatamente após esse intervalo, o que permite ações arbitrariamente curtas e que se iniciam em qualquer valor contínuo do tempo.

Tendo em vista que problemas com variáveis contínuas requerem uma representação diferente, introduzimos uma extensão do MDP, o *Processo de Decisão Markoviano Híbrido*, também chamado de *MDP com variáveis híbridas de estado e ação*:

Definição 2.1 *Um* **Processo Markoviano de Decisão Híbrido (Hybrid Markov Decision Process - HMDP)** *é definido por uma tupla* $\mathcal{M} = \langle \vec{B}, \vec{X}, \mathcal{A}_{par}, \mathcal{R}, \mathcal{P} \rangle$ *em que:*

- $\vec{B}$ *é um vetor de* m *variáveis de estado booleanas,* $\vec{B} = \{B_1, B_2, \ldots, B_m\}$*;*
  Uma atribuição de $\vec{B}$ *é um vetor* $\vec{b} = (b_1, \ldots, b_m)$*, onde* $b_i \in \{0, 1\}$ $1 \leq i \leq m$*.*
- $\vec{X}$ *é um vetor de* n *variáveis de estado contínuas,* $\vec{X} = \{X_1, X_2, \ldots, X_n\}$*;*
  Uma atribuição de $\vec{X}$ *é um vetor* $\vec{x} = (x_1, \ldots, x_n)$*, onde* $x_j \in [x_j^i, x_j^s] \subset \mathbb{R}$ $1 \leq j \leq n$*.*
  Denotamos por $\mathcal{S} = \vec{B} \times \vec{X}$ *o conjunto de estados, onde um estado é uma atribuição das variáveis* $s = (\vec{b}, \vec{x}) = (b_1, \ldots, b_m, x_1, \ldots, x_n) \in \vec{B} \times \vec{X}$*;*

- *$\mathcal{A}_{par}$ é um conjunto de k ações paramétricas $\{a_1, \ldots, a_k\}$. Uma ação paramétrica a_i é uma função que atribui um vetor de $d(a_i)$ parâmetros reais $\vec{y} \in \mathbb{R}^{d(a_i)}$ a uma ação instanciada $a_i(\vec{y})$.*

  Denotaremos o conjunto de ações paramétricas instanciadas, ou apenas ações, por $\mathcal{A}$.

- *$\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ é a função recompensa que atribui a um estado e uma ação, ou uma valoração das variáveis de estado e dos parâmetros de uma ação paramétrica, um número real.*

  Note que não há necessidade da recompensa depender do estado sucessor, já que consideramos apenas o valor esperado dessa recompensa.

- *$\mathcal{P} : \mathcal{S} \times \mathcal{A} \to D(\mathcal{S})$ é a função de transição probabilística, que associa um estado e uma ação a uma distribuição de probabilidade sobre o conjunto de estados. Denotaremos por $\mathcal{P}(s, a)$ a distribuição de probabilidade sobre $\mathcal{S}$ do próximo estado, e por $\mathcal{P}(s, a, s')$ a densidade de probabilidade associada ao resultado s' na distribuição $\mathcal{P}(s, a)$.*

A Definição 2.1 do HMDP é uma generalização direta do MDP descrito na Definição 1.1, no entanto, neste caso há algumas questões adicionais:

- Uma função de variáveis discretas (com domínio finito) assume um conjunto finito de valores, por isso pode ser representada por uma tabela de valores. Ao contrário, uma função de variáveis contínuas arbitrária pode não admitir qualquer representação exata com um número finito de valores.
- A política ótima para um MDP é um mapeamento simples de um número finito de estados num número finito de ações, porém no caso de variáveis de estado contínuas e ações paramétricas, a política é uma função das variáveis contínuas do estado e além de determinar qual ação paramétrica, também determina quais os valores de seus parâmetros.

Por causa dessas questões, algoritmos para a solução exata de HMDPs precisam fazer suposições restritivas sobre as funções $\mathcal{R}$ e $\mathcal{P}$. Conforme apresentarmos os algoritmos nas próximas seções, iremos clarificar quais as restrições assumidas em cada versão do modelo. Antes disso, vamos estender para o HMDP os conceitos fundamentais desenvolvidos no caso de variáveis discretas.

A Definição 1.2 de execução do MDP como sequência de triplas (estado, ação recompensa) se estende sem qualquer modificação para o HMDP. Uma política, estacionária ou não, num HMDP, é uma função que associa o estado, e o horizonte, a uma ação paramétrica e a valores para os parâmetros dessa ação. O conceito de alcançabilidade e política parcial também são os mesmos do MDP. Quanto a avaliação das execuções, a análise do horizonte de execução e os submodelos MDPF, SSP e MDPD, dadas no Capítulo 1, são naturalmente estendidas para o HMDP. Denotaremos por HMDPF e HMDPD, respectivamente as versões com horizonte finito e infinito descontado do HMDP. Para resolver o HMDP, primeiro vamos estender a definição de qualidade das ações e encontrar uma versão da Equação de Bellman para horizonte finito (1.18) para HMDPs.

2.1.1 Equação de Bellman e Programação Dinâmica para HMDPs

A função valor não estacionária $V^\pi(\vec{b}, \vec{x}, h)$ é a recompensa acumulada esperada na execução de uma política π por h estágios:

$$V^\pi(\vec{b}, \vec{x}, h) = E_\pi \left[\sum_{i=1}^{h} r^i \;\middle|\; s^0 = (\vec{b}, \vec{x}) \right] \tag{2.1}$$

Novamente, podemos expandir um passo dessa execução para obter uma relação entre a função valor de um horizonte h e a seguinte, de horizonte $h - 1$.

$$\begin{aligned}
V^\pi(\vec{b}, \vec{x}, h) &= E_\pi \left[\sum_{i=1}^{h} r^i \;\middle|\; s^0 = (\vec{b}, \vec{x}) \right] \\
V^\pi(\vec{b}, \vec{x}, h) &= E_\pi[r^1 \mid s^0 = (\vec{b}, \vec{x})] + E_\pi \left[\sum_{i=2}^{h} r^i \;\middle|\; s^1 \sim \mathcal{P}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) \right] \\
V^\pi(\vec{b}, \vec{x}, h) &= \mathcal{R}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) + E_\pi \left[\sum_{i=2}^{h} r^i \;\middle|\; s^1 = (\vec{b'}, \vec{x'}) \sim \mathcal{P}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) \right]
\end{aligned} \tag{2.2}$$

$\mathcal{P}$ e $\mathcal{R}$ são estacionárias, então a somatória das recompensas para i de 2 a h com $s^1 = (\vec{b'}, \vec{x'})$ é equivalente a soma para i de 1 a $h-1$ com $s^0 = (\vec{b'}, \vec{x'})$, que tem a mesma forma da definição de função valor, Equação 2.1, isto é:

$$E_\pi \left[\sum_{i=2}^{h} r^i \,\middle|\, s^1 = (\vec{b'}, \vec{x'}) \right] = E_\pi \left[\sum_{i=1}^{h-1} r^i \,\middle|\, s^0 = (\vec{b'}, \vec{x'}) \right] = V^\pi(\vec{b'}, \vec{x'}, h-1) \tag{2.3}$$

Substituindo (2.3) em (2.2) temos:

$$V^\pi(\vec{b}, \vec{x}, h) = \mathcal{R}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) + E\left[V^\pi(\vec{b'}, \vec{x'}, h-1) \,\middle|\, (\vec{b'}, \vec{x'}) \sim \mathcal{P}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) \right] \tag{2.4}$$

Como o próximo estado, $(\vec{b'}, \vec{x'})$, é uma combinação de variáveis aleatórias discretas e contínuas, para obter o valor esperado em (2.4) é necessário somar sobre os resultados possíveis das variáveis discretas e integrar sobre a densidade de probabilidade das variáveis contínuas:

$$E\left[V^\pi(\vec{b'}, \vec{x'}, h-1) \,\middle|\, (\vec{b'}, \vec{x'}) \sim \mathcal{P}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) \right] = \sum_{\vec{b'} \in \vec{B}} \int_{\vec{x'} \in \vec{X}} V^\pi(\vec{b'}, \vec{x'}, h-1) \cdot \mathcal{P}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h), \vec{b'}, \vec{x'}) \tag{2.5}$$

Usando (2.5) em (2.4) obtemos finalmente a expressão de $V^\pi(\cdot, h)$ em função de $V^\pi(\cdot, h-1)$.

$$V^\pi(\vec{b}, \vec{x}, h) = \mathcal{R}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h)) + \sum_{\vec{b'} \in \vec{B}} \int_{\vec{x'} \in \vec{X}} V^\pi(\vec{b'}, \vec{x'}, h-1) \cdot \mathcal{P}(\vec{b}, \vec{x}, \pi(\vec{b}, \vec{x}, h), \vec{b'}, \vec{x'}) \tag{2.6}$$

A partir dessa equação, podemos definir a qualidade $Q^V(\vec{b}, \vec{x}, a, \vec{y}, h)$ da ação $a(\vec{y})$ no estado $(\vec{b}, \vec{x})$ e estágio h, usando a função valor V, como segue:

$$Q^V(\vec{b}, \vec{x}, a, \vec{y}, h) = \mathcal{R}(\vec{b}, \vec{x}, a(\vec{y})) + \sum_{\vec{b'} \in \vec{B}} \int_{\vec{x'} \in \vec{X}} V(\vec{b'}, \vec{x'}, h-1) \cdot \mathcal{P}(\vec{b}, \vec{x}, a(\vec{y}), \vec{b'}, \vec{x'}) \tag{2.7}$$

Como fizemos na Seção 1.1.1, a qualidade ótima é aquela definida usando a função valor ótima como recompensa esperada dos estados sucessores:

$$Q^*(\vec{b}, \vec{x}, a, \vec{y}, h) = \mathcal{R}(\vec{b}, \vec{x}, a(\vec{y})) + \sum_{\vec{b'} \in \vec{B}} \int_{\vec{x'} \in \vec{X}} V^*(\vec{b'}, \vec{x'}, h-1) \cdot \mathcal{P}(\vec{b}, \vec{x}, a(\vec{y}), \vec{b'}, \vec{x'}) \tag{2.8}$$

Do mesmo modo que no caso discreto, o princípio de otimalidade nos diz que a escolha de uma ação só é ótima se ela maximiza a soma da recompensa imediata com a recompensa esperada ótima, ou seja:

$$\pi^*(\vec{b}, \vec{x}, h) = \arg\max_{a, \vec{y}} [Q^*(\vec{b}, \vec{x}, a, \vec{y}, h)] \tag{2.9}$$

Finalmente, como o valor ótimo V^* é aquele obtido ao se executar a política ótima, podemos escrever a **Equação de Bellman para HMDPFs**.

$$V^*(\vec{b}, \vec{x}, h) = \max_{a, \vec{y}} \left[\mathcal{R}(\vec{b}, \vec{x}, a(\vec{y})) + \sum_{\vec{b'} \in \vec{B}} \int_{\vec{x'} \in \vec{X}} V^*(\vec{b'}, \vec{x'}, h-1) \cdot \mathcal{P}(\vec{b}, \vec{x}, a(\vec{y}), \vec{b'}, \vec{x'}) \right] \tag{2.10}$$

Como fizemos na Seção 1.1.2, a Equação 2.10 pode ser usada numa solução de programação dinâmica para HMDPFs, o que é de fato feito pela programação dinâmica simbólica com variáveis contínuas, na qual nosso trabalho se baseia e que será apresentada em detalhes na seção seguinte.

2.2 Programação Dinâmica Simbólica Exata com Variáveis Contínuas

A técnica de programação dinâmica simbólica foi inicialmente aplicada de forma exata para HMDPs considerando variáveis de estado discretas e contínuas e ações não parametrizadas, em Sanner *et al.* (2011), e posteriormente foi estendida permitindo ações com parâmetros contínuos, em Zamani *et al.* (2012). Para uma apresentação clara e incremental das propriedades dessa técnica iremos descrever os dois trabalhos. No

entanto, para uma apresentação detalhada de suas técnicas é necessário entender a representação simbólica que é utilizada nestes trabalhos e a estrutura de dados que implementa essa representação de forma eficiente, que são partes essenciais da técnica SDP.

2.2.1 Representação de Funções baseadas em Casos

A representação de funções baseadas em casos é a representação simbólica usada por Sanner *et al.* (2011) e que também usaremos em nossa extensão dessas soluções. Iremos considerar um conjunto $\vec{X} = (X_1, \ldots, X_n)$ de variáveis contínuas ($x_j \in [x_j^i, x_j^s]$) e um conjunto $\vec{B} = (B_1, \ldots, B_m)$ de variáveis booleanas ($b_i \in \{0, 1\}$). Uma atribuição dessas variáveis é um vetor $(\vec{b}, \vec{x})$. Note que os parâmetros contínuos de ações paramétricas também são variáveis contínuas e serão representados deste mesmo modo. Para representar as funções de variáveis contínuas de HMDPs de forma exata, vamos supor que estas são funções lineares ou polinomiais baseadas em casos, ou definidas por partes. Os casos ou partes são definidos a partir de expressões lineares ou polinomiais, como segue:

Definição 2.2 *Uma* **expressão linear**[1] *f sobre $\vec{X}$ é uma expressão do tipo:*

$$f(\vec{x}) = c_0 + c_1 x_1 + c_2 x_2 + \cdots + c_n x_n = \sum_{i=0}^{n} c_i x_i. \tag{2.11}$$

Iremos denotar uma expressão linear como um produto escalar $f_l(\vec{x}) = \vec{c} \cdot \vec{x} := \sum_{i=0}^{n} c_i x_i$, onde $x_0 = 1$ é usado para simplificar a notação.

Definição 2.3 *Um* **monômio m** *sobre $\vec{X}$ é um termo:*

$$m(\vec{x}) = c x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n} = c \cdot \prod_{i=0}^{n} x_i^{\alpha_i}, \text{ onde cada } \alpha_i \in \mathbb{N}. \tag{2.12}$$

Definição 2.4 *Um* **polinômio**, *ou uma* **expressão polinomial**, *f sobre $\vec{X}$ é uma soma finita de n_p monômios:*

$$f(\vec{x}) = \sum_{j=1}^{n_p} m_j(\vec{x}), \tag{2.13}$$

onde cada m_j é um monômio sobre $\vec{X}$ e $n_p \in \mathbb{N}$ é o número de termos deste polinômio.

Usando essas expressões, vamos definir os casos, que serão os componentes de uma função definida por partes.

Definição 2.5 *Um* **caso** *ϕ é uma fórmula lógica sobre as variáveis $\vec{B}$ e $\vec{X}$ que define uma região $C(\phi)$ do espaço onde ela é válida. Os casos serão definidos por combinações lógicas ($\vee, \wedge, \neg$) de* **decisões**, *que podem ser de dois tipos: uma variável booleana, $B_j \in \vec{B}$ ou uma inequação entre expressões (lineares ou polinomiais) sobre $\vec{X}$. Casos são lineares ou polinomiais, respectivamente, se são definidos por inequações com expressões lineares e inequações com expressões polinomiais.*

Exemplos de casos: $\phi_1 = B_1$; $\phi_2 = \neg(X_1 > 2)$; $\phi_3 = B_1 \wedge ((X_1 + X_2 < 0) \vee \neg B_2)$.

Casos são fórmulas lógica proposicionais (suas proposições são decisões, i.e. variáveis booleanas ou inequações) portanto podem ser colocado em *forma normal disjuntiva (Disjunctive Normal Form- DNF)*, conforme segue.

Definição 2.6 *Uma fórmula em* **forma normal disjuntiva (DNF)** *é uma disjunção de formulas puramente conjuntivas, ou seja, de conjunções de literais. Denotaremos as fórmulas puramente conjuntivas por θ, e assim um caso ϕ em DNF será da forma:*

$$\phi = \bigvee_{j=1}^{n_\phi} \theta_j \tag{2.14}$$

[1] O termo matematicamente correto para essas expressões é "expressões afins", uma vez que permitem a adição de uma constante. No entanto, esse uso de "expressão linear" é comum na literatura e não deve provocar confusões.

Por exemplo, o caso $\phi_3 = B_1 \wedge ((X_1 + X_2 < 0) \vee \neg B_2)$ em DNF seria $(B_1 \wedge (X_1 + X_2 < 0)) \vee (B_1 \wedge \neg B_2)$. A motivação de representar casos em DNF é que essa organização irá permitir o compartilhamento eficiente de decisões entre casos.

Finalmente, podemos definir as funções baseadas em casos:

Definição 2.7 *Uma* **função baseada em casos** *$f(\vec{b}, \vec{x})$ de variáveis discretas e contínuas é da seguinte forma:*

$$f(\vec{b}, \vec{x}) = \begin{cases} \phi_1(\vec{b}, \vec{x}) : & f_1(\vec{x}) \\ \vdots & \vdots \\ \phi_k(\vec{b}, \vec{x}) : & f_k(\vec{x}) \end{cases} . \tag{2.15}$$

Onde cada $\phi_i(\vec{b}, \vec{x})$ é um caso e cada $f_i(\vec{x})$ é uma expressão sobre as variáveis contínuas, chamada definição de f para o caso ϕ_i. Quando todos ϕ_i e f_i são lineares ou polinomiais, a função é, respectivamente, **linear por partes** *ou* **polinomial por partes**.

Para se avaliar uma função definida por casos $f(\vec{b}, \vec{x})$, primeiro é preciso identificar qual o caso ϕ_i que é satisfeito por $(\vec{b}, \vec{x})$, então o valor de f será dado por $f_i(\vec{x})$. Para a função definida em (2.15) ser bem formada é necessário que os casos ϕ_i sejam uma partição do espaço representado pelas variáveis, isto é:

- A união dos casos deve ser o espaço completo: $\bigcup_{i=1}^{k} C(\phi_i) = \vec{B} \times \vec{X}$;
- Os casos devem ser disjuntos: $C(\phi_i) \cap C(\phi_j) = \emptyset \ \forall i \neq j$;

Um exemplo de uma função baseada em casos é dado na Figura 2.1c. A função é descrita por casos lineares e expressões lineares.

A representação de funções baseadas em caso permite também que sejam realizadas operações entre funções. Por exemplo, a adição e multiplicação de funções caso são denotadas por $\oplus$ e $\otimes$. Para deixar mais claro como são feitas as operações aritméticas entre funções baseadas em casos, vamos ilustrar com uma operação $f \bigodot g$ (em que $\bigodot$ representa uma operação aritmética, $\oplus$ denota adição e $\otimes$, multiplicação). Numa operação aritmética entre funções baseadas em casos, são definidos novos casos para todas as conjunções válidas dos casos das funções f e g e então a operação é realizada diretamente entre as expressões definidas para cada combinação de casos, conforme ilustrado a seguir.

$$\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} \bigodot \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 \cdot g_1 \\ \phi_1 \wedge \psi_2 : & f_1 \cdot g_2 \\ \phi_2 \wedge \psi_1 : & f_2 \cdot g_1 \\ \phi_2 \wedge \psi_2 : & f_2 \cdot g_2 \end{cases}$$

Além de operações aritméticas, podemos realizar **substituições** em funções simbólicas. Uma substituição numa função baseada em casos f é uma operação do tipo $f\{v = g\}$ onde toda ocorrência da variável v, tanto nas fórmulas dos casos, como nas expressões, em f é substituída por g, que pode ser uma variável, ou expressão simbólica. Como um exemplo de substituição, temos $f\{x_1 = x_2 + 1\}$:

$$f\{x_1 = x_2 + 1\} = \left(\begin{cases} x_1 > 0 : & 0 \\ x_1 < 0 : & x_1 - 1 \end{cases} \right) \{x_1 = x_2 + 1\} = \begin{cases} x_2 + 1 > 0 : & 0 \\ x_2 + 1 < 0 : & x_2 \end{cases}$$

Uma outra operação importante entre funções baseadas em casos é a **maximização**. Em particular, o máximo entre duas funções lineares ou polinomiais por partes também é, respectivamente, linear ou polinomial por partes. Para efetuar a maximização, além de combinar os casos das funções originais, são adicionadas inequações que comparam as expressões correspondentes em cada combinação, por exemplo, $\max(f, g)$ é dado por:

$$\max \left(\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases}, \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge f_1 > g_1 : & f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \leq g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : & f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \leq g_2 : & g_2 \\ \phi_2 \wedge \psi_1 \wedge f_2 > g_1 : & f_2 \\ \phi_2 \wedge \psi_1 \wedge f_2 \leq g_1 : & g_1 \\ \phi_2 \wedge \psi_2 \wedge f_2 > g_2 : & f_2 \\ \phi_2 \wedge \psi_2 \wedge f_2 \leq g_2 : & g_2 \end{cases}$$

Como pode ser observado nesse exemplo, a maximização entre funções definidas por partes pode aumentar muito o número de casos na função resultante, o que justifica a necessidade de uma representação eficiente.

A representação por casos é interessante por representar exatamente funções complexas como combinações de expressões simples. No entanto, da maneira como a definimos, a representação em casos é ineficiente, já que cada fórmula ϕ é definida de maneira independente das outras e por isso há um grande número de ocorrências repetidas de inequações e variáveis booleanas. Uma maneira mais eficiente de representar a partição do espaço é fazendo decisões sequenciais até se determinar a qual região os parâmetros correspondem. Isso é feito na representação como *diagrama de decisão algébrico estendido*, que apresentamos a seguir.

2.2.2 Diagramas de Decisão Algébricos Estendidos (XADD)

O **Diagrama de decisão Algébrico Estendido (eXtended Algebraic Decision Diagram - XADD)** é uma extensão do ADD apresentado na Seção 1.2.3, que permite representar não só funções de variáveis booleanas, mas também funções de variáveis contínuas. As folhas de um XADD são expressões de variáveis contínuas, e os nós internos podem ser variáveis binárias ou inequações envolvendo as variáveis contínuas.

Definição 2.8 *Um* **Diagrama de Decisão Algébrico Estendido (XADD)** *é uma representação de uma função definida por partes através de grafo acíclico dirigido com dois tipos de nós: nós internos ou de decisão e nós terminais ou folhas. Os nós internos contém uma decisão descrita por uma variável booleana ou uma inequação. As decisões são ordenadas, e todo caminho dirigido no grafo mantém a relação de ordem entre duas decisões. De cada nó interno saem dois arcos, um deles é seguido se a variável ou inequação for verdadeira para os valores dos parâmetros, e o outro é seguido se a decisão for falsa. Os nós terminais contém expressões (Definição 2.2 ou 2.4) sobre $\vec{X}$. O valor da função representada pelo XADD é o valor obtido substituindo os valores das variáveis contínuas na expressão terminal correspondente.*

Vamos exemplificar um XADD com a função linear da Figura 2.1b. Os nós internos são representados em azul, o arco verdadeiro é representado por linha contínua e o arco falso é representado por linha tracejada.

Um caminho num XADD corresponde a uma conjunção de decisões, mais especificamente, à fórmula obtida pela conjunção de cada uma das decisão dos nós percorridos, negadas (condição falsa) ou não (condição verdadeira), em função do arco usado no caminho. Essa é uma fórmula puramente conjuntiva, e corresponde a um θ de uma fórmula DNF, conforme a Definição 2.6.

Podemos representar uma função baseada em casos num XADD, cada nó terminal num XADD corresponde a um caso e uma expressão da representação usada na Definição 2.7. O caso correspondente a um nó terminal é a fórmula obtida fazendo a disjunção das fórmulas associadas a cada caminho que termina nesse nó. A expressão para esse caso é aquela que está armazenada no nó terminal. A figura 2.1 representa uma função linear por partes (2.1c). Em 2.1b temos a representação dessa função como um XADD: o nó terminal na direita, marcado com $0.5 * x$, representa o caso ϕ_1 e a expressão f_1 e o nó terminal da esquerda representa o caso ϕ_2 e a expressão f_2. Por exemplo, a fórmula puramente conjuntiva θ_{11} é representada pelo caminho direto da raíz até nó $0.5 * x$ e θ_{12}, pelo caminho que passa pela decisão $y > x$ e toma o arco tracejado (falso). Assim, o caso ϕ_1 é a disjunção dessas fórmulas, que são os caminhos até o nó terminal que o representa.

A vantagem da representação como XADD vêm de compartilhar as inequações e variáveis booleanas usadas em diferentes casos, o que permite uma avaliação mais eficiente. Além disso, operações entre dois XADDs que usam a mesma ordem testam cada inequação ou variável apenas umas vez em ambos diagramas, percorrendo-os simultaneamente e muitas vezes reutilizando cálculos de operações anteriores.

Para resolver um HMDP, são utilizados quatro tipos de operações entre XADDs Sanner *et al.* (2011); Zamani *et al.* (2012):

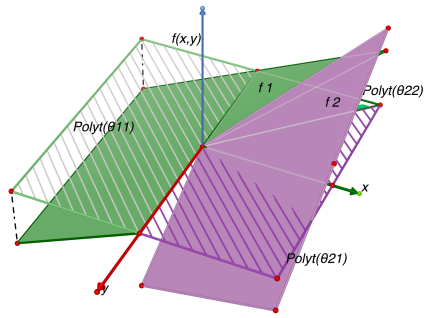
(i) *Operações Algébricas* (soma $\oplus$, produto $\otimes$), são definidas naturalmente dentro de cada região, então a partição resultante é o produto cartesiano das partições originais, como no caso das ADDs;

(ii) *Operação de Substituição*, onde uma variável é substituída por uma expressão nova em todas suas ocorrências no diagrama. (Veja a Figura 2.2a);

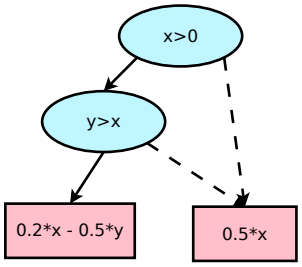
(iii) *Maximização*, que envolve comparar as decisões e além disso adiciona comparações entre os nós terminais (Veja Figura 2.2b);

(iv) *Maximização de Parâmetro*, ($pmax$) remove um parâmetro do diagrama substituindo pelo valor que torna esse diagrama máximo em cada caminho. (Veja Figura 2.2c).

Enfim, temos as definições necessárias para apresentar o algoritmo de programação dinâmica simbólica com variáveis contínuas, conforme descreveremos a seguir.



(a) *Gráfico de f*



(b) *XADD de f*

$$f(x,y) = \begin{cases} \phi_1: & f_1 \\ \phi_2: & f_2 \end{cases} \qquad \begin{aligned} \phi_1 &= \theta_{11} \vee \theta_{12} \\ \theta_{11} &= x < 0 \\ \theta_{12} &= x > 0 \wedge x < -y \\ f_1 &= \frac{x}{2} \end{aligned} \qquad \begin{aligned} \phi_2 &= \theta_{21} \\ \theta_{21} &= x > 0 \wedge y > x \\ f_2 &= \frac{x}{5} - \frac{y}{2} \end{aligned}$$

(c) *Definição de f*

Figura 2.1: *Representações de uma função linear por partes: (a) Gráfico da função $f(x,y)$; (b) Representação com o diagrama de decisão algébrico estendido (XADD); (c) Definição de $f(x,y)$ baseada em casos.*

2.2.3 Programação Dinâmica Simbólica com Variáveis de Estado Contínuas

O primeiro trabalho usando a técnica SDP para planejamento probabilístico com variáveis contínuas, foi *Symbolic Dynamic Programming for Discrete and Continuous State MDPs* [Sanner *et al.* (2011)]. O modelo de HMDP usado, chamado de *MDP com Estados Discretos e Contínuos (Discrete and Continuous State MDP - DCMDPs)*, se baseia na Definição 2.1, com as seguintes restrições:

- **Ações Discretas:** Não é permitido o uso de parâmetros contínuos para as ações $a \in \mathcal{A}$, sendo $\mathcal{A}$ é um conjunto finito de ações;
- **Transições Estocásticas Discretas:** As transições de variáveis contínuas são determinísticas, porém, elas dependem dos resultados dos eventos probabilísticos das variáveis discretas, de forma que após a transição existe um conjunto finito de resultados possíveis para cada variável contínuas.
- **Funções Baseadas em Casos:** As funções de variáveis contínuas, de transição e de recompensa, são polinomiais por partes.

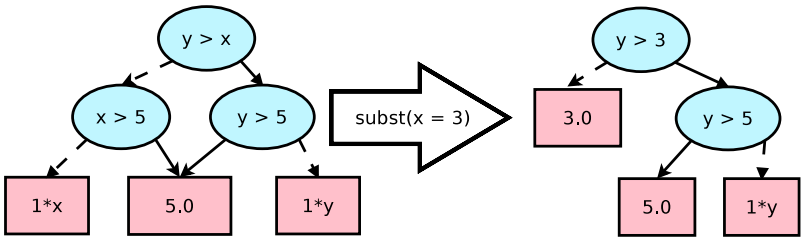
Mais precisamente, o modelo de transição é da seguinte forma:

$$\mathcal{P}(\vec{b}, \vec{x}, a, \vec{b'}, \vec{x'}) = \prod_{i=1}^{m} \mathcal{P}(b'_i | \vec{b}, \vec{x}, a) \prod_{j=1}^{n} \mathcal{P}(x'_j | \vec{b}, \vec{x}, a, \vec{b'}) \tag{2.16}$$

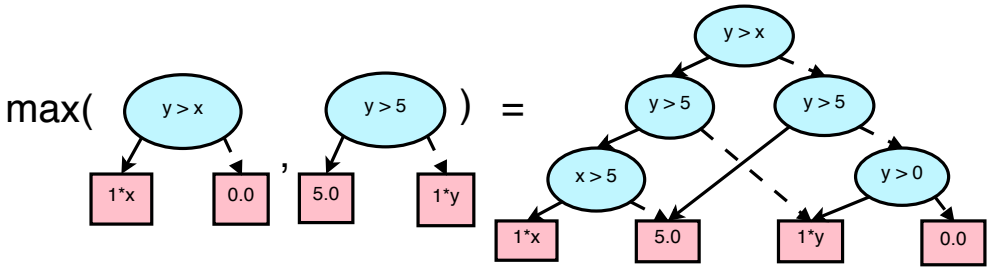
onde $\mathcal{P}(b'_i|\vec{b}, \vec{x}, a)$ são probabilidades condicionais de transição discretas, i.e. valores reais positivos que somam 1, e $\mathcal{P}(x'_j|\vec{b}, \vec{x}, a, \vec{b'})$ são densidades de probabilidade determinísticas.

Definição 2.9 *Uma* **densidade de probabilidade determinística** *é uma distribuição $\delta[t]$ de Dirac que satisfaz a seguinte propriedade:*

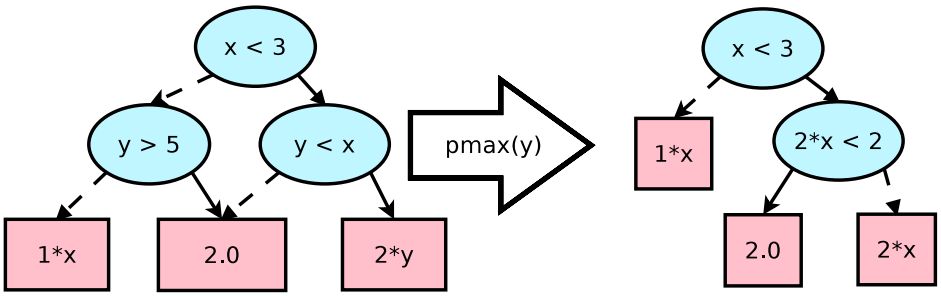
$$\int_Z \delta[t](z) f(z) dz = f(t) \tag{2.17}$$



(a) *Substituição (subst).*



(b) *Maximização (*max*)*



(c) *Maximização de Parâmetro (*pmax*).*

Figura 2.2: *Exemplos de operações definidas para XADDs: Substituição de variável por uma expressão; Maximização entre dois XADDs e Maximização de um parâmetro de um XADD.*

Ou seja, ela é uma distribuição probabilística que representa uma transição determinística para um valor específico. Calcular o valor esperado de f(z) quando a distribuição de z é $\delta[t]$ é substituir a variável aleatória z pelo valor t.

Dessa forma $\mathcal{P}(x'_j|\vec{b},\vec{x},a,\vec{b'})$ é uma distribuição que determina o valor de x'_j, ou seja:

$$\mathcal{P}(x'_j|\vec{b},\vec{x},a,\vec{b'}) = \delta[f(\vec{b},\vec{x},a,\vec{b'})] \tag{2.18}$$

o que significa que o valor de x'_j será deterministicamente $f(\vec{b},\vec{x},a,\vec{b'})$, onde f é uma função baseada em casos como na Definição 2.7. Iremos denotar por $\mathcal{P}_{x'_j}(\vec{b},\vec{x},a,\vec{b'})$ a função f que determina a variável x_j no próximo estágio.

O algoritmo que será usado para obter a solução do HMDP será baseado no algoritmo Iteração de Valor (Seção 1.1.2). São construídas funções valor e políticas não-estacionárias ótimas para valores cada vez maiores de horizonte. Iniciando com $V^0 = 0$, usamos atualizações de Bellman, i.e. aplicações da equação de Bellman (2.10), para obter V^1 e π^1, V^2 e π^2, e assim sucessivamente, até o horizonte desejado.

Para realizar essas operações com variáveis contínuas, vamos transformar a atualização de Bellman em uma sequência de operações simbólicas. A equação da atualização de Bellman ((2.10)) é a seguinte:

$$V^*(\vec{b},\vec{x},h) \leftarrow \max_{a,\vec{y}} \left[\mathcal{R}(\vec{b},\vec{x},a(\vec{y})) + \sum_{\vec{b'}\in\vec{B}} \int_{\vec{x'}\in\vec{X}} V^*(\vec{b'},\vec{x'},h-1)\cdot\mathcal{P}(\vec{b},\vec{x},a(\vec{y}),\vec{b'},\vec{x'}) \right] \tag{2.19}$$

A atualização, que calcula $V(\cdot,h)$ a partir de $V(\cdot,h-1)$, será feita simbolicamente em cinco etapas, descritas a seguir:

1. *Marcação das variáveis do próximo estado.* As variáveis simbólicas do estado atual, $\vec{B}$ e $\vec{X}$, são subs-

tituídas simbolicamente pelas variáveis do próximo estado $\vec{B'}$ e $\vec{X'}$ na representação da função valor $V(\cdot, h)$. Ou seja, a função valor atual se torna a função valor de próximo estado, para iniciar a regressão. Na Equação 2.10, essa etapa corresponde a fazer o $V(\cdot, h)$ de uma iteração ser o $V(\cdot, h-1)$ da iteração seguinte. Note que durante as operações teremos funções simbólicas com as variáveis do estado atual $(\vec{B}, \vec{X})$ e do estado seguinte $(\vec{B'}, \vec{X'})$ que são variáveis diferentes.

2. *Adição Simbólica.* Essa etapa é o cálculo função de qualidade de uma ação, construída através de manipulações simbólicas. Numa representação comum (não-simbólica) a operação para se obter a qualidade de uma ação para todos estados é a seguinte:

$$Q(s,a) \leftarrow \mathcal{R}(s,a) + \sum_{s' \in S} [\mathcal{P}(s,a,s')V(s')] \forall s \in S. \tag{2.20}$$

Mas na representação simbólica isso é substituído por operações entre funções baseadas em casos:

$$\tilde{Q}_a \leftarrow \mathcal{R}_a \oplus V', \tag{2.21}$$

onde $\mathcal{R}_a$, V' e $\tilde{Q}_a$ são funções baseadas em casos, respectivamente, das variáveis $(\vec{B}, \vec{X})$, $(\vec{B'}, \vec{X'})$ e $(\vec{B}, \vec{X}, \vec{B'}, \vec{X'})$ e $\oplus$ é o símbolo usado para representar a soma de funções baseadas em casos. A operação simbólica em (2.21) corresponde a executar (2.20) em todo o espaço de estados.

A função simbólica para $\tilde{Q}_a$ que obtermos em (2.21), não é a função correta da qualidade Q_a, pois ainda depende de variáveis do próximo estado $(\vec{B'}, \vec{X'})$. Essas variáveis precisam ser removidas através de sua substituição de acordo com a função de transição $\mathcal{P}$ para obtermos Q_a. Isso é feito nas próximas duas etapas, a integração sobre as variáveis contínuas e a marginalização sobre as variáveis discretas.

3. *Integração sobre as variáveis contínuas.* Nessa etapa, realizamos a integração contínua, ou seja, iremos remover as variáveis $\vec{X'}$ de $\tilde{Q}_a$ obtendo a função $\dot{Q}_a$ correspondente ao valor esperado em relação a essas variáveis. Isso é definido pela seguinte operação:

$$\dot{Q}_a = E_{\vec{X'}}\left[\tilde{Q}_a\right] = \int_{\vec{x'} \in \vec{X}} \tilde{Q}_a(\vec{x'}) \mathcal{P}(\vec{x'} | \vec{b}, \vec{x}, a, \vec{b'}) \tag{2.22}$$

Como discutimos no inicio da seção, esta integral é na verdade feita sobre uma função δ de Dirac, e portanto se torna uma substituição simbólica. Para cada variável contínua x'_j, temos uma função do tipo:

$$\mathcal{P}(x'_j | \vec{b}, \vec{x}, a, \vec{b'}) = \delta[\mathcal{P}_{x'_j}(\vec{b}, \vec{x}, a, \vec{b'})],$$

que representa uma transição do tipo $x'_j = \mathcal{P}_{x'_j}(\vec{b}, \vec{x}, a, \vec{b'})$,conforme Definição 2.9. Portanto a integral de (2.22) será um conjunto de substituições da forma:

$$\dot{Q}_a = \tilde{Q}_a\{x'_1 = \mathcal{P}_{x'_1}(\vec{b}, \vec{x}, a, \vec{b'})\}\{x'_2 = \dots\} \dots \{x'_n = \mathcal{P}_{x'_1}(\vec{b}, \vec{x}, a, \vec{b'})\} \tag{2.23}$$

onde $f\{x' = g(x)\}$ é a função obtida substituindo toda ocorrência de x' em f por $g(x)$.

No caso da integração contínua, as substituições são feitas pelas funções $\mathcal{P}_{x'_j}(\vec{b}, \vec{x}, a, \vec{b'})$, que são funções simbólicas de $\vec{B}$, $\vec{X}$, a e $\vec{B'}$. Assim, o resultados dessa etapa é a função baseada em casos $\dot{Q}_a$ que não possui ocorrências de $\vec{X'}$.

4. *Marginalização Discreta.* A operação de marginalizarão é o cálculo do valor esperado de uma função de uma variável aleatória discreta. Por exemplo para uma variável booleana B:

$$E_B[f(B)] = f(true)Pr[B = true] + f(false)Pr[B = false]. \tag{2.24}$$

No cálculo da função qualidade, a função que é marginalizada é a qualidade $\dot{Q}_a$ e estamos fazendo o valor esperado em relação a $\vec{B'}$:

$$Q_a = E_{\vec{B'}}\left[\dot{Q}_a\right] = E_{b'_1}\left[E_{b'_2}\left[\dots E_{b'_m}\left[\dot{Q}_a\right]\right]\right], \tag{2.25}$$

onde cada operação de marginalização é feita como em (2.24)

$$E_{b'_j}\left[f(b'_j)\right] = \left[f(b'_j) \otimes \mathcal{P}(b'_j | \vec{b}, \vec{x}, a)\right]\Big|_{b'_j = true} \oplus \left[f(b'_j) \otimes \mathcal{P}(b'_j | \vec{b}, \vec{x}, a)\right]\Big|_{b'_i = false} \tag{2.26}$$

A marginalização remove as variáveis $\vec{B}'$ de $\dot{Q}_a$ e obtemos a função baseada em casos para a qualidade Q_a envolvendo apenas as variáveis do estado atual.

5. *Maximização.* Finalmente, como há um número finito de ações, e temos a função simbólica Q_a para cada ação, podemos finalizar o cálculo da atualização de Bellman, (2.19), fazendo a maximização entre as ações. Para isso fazemos uma sequência de maximizações entre pares de funções baseadas em casos, a cada passo obtendo uma função V ótima dentre as ações que já foram maximizadas.

$$V = \max(Q_{a_1}, \max(\ldots, \max(Q(a_{|\mathcal{A}|-1}, Q(a_{|\mathcal{A}|})))...)) \tag{2.27}$$

Após essa maximização, a função encontrada para V é de fato função valor no lado esquerdo da atualização de bellman, (2.19).

Algoritmo 2.1: `SDP-DCMDP` (DCMDP $\mathcal{M}$, $\mathcal{H}$) $\longrightarrow$ ($V^*(\mathcal{H})$)

Entrada: $\mathcal{M}$: Descrição do problema; $\mathcal{H}$: Horizonte máximo para a solução
Saída: $V^*(\mathcal{H})$: Função valor ótima restando $\mathcal{H}$ passos;

$V^*(0) = 0$// *Inicialização da Programação Dinâmica*
para h *de* 1 *até* $\mathcal{H}$ **faça**
  // *Efetua atualização de Bellman, Equação 2.19*
  $V^*(h) \leftarrow -\infty$ //*Valor inicial para a maximização*
  para cada *ação* a *em* $\mathcal{A}$ **faça**
    $Q_a \leftarrow$ `Retropropagação`$(V^*(h-1), a)$// *Algoritmo 2.2*
    $V^*(h) \leftarrow max(V^*(h), Q_a)$ //*Maximização simbólica (Etapa 5)*
retorna $V^*(\mathcal{H})$

Algoritmo 2.2: `Retropropagação`(V,a) $\longrightarrow$ Q_a

Entrada: V: função valor da iteração anterior, a: ação a ser avaliada
Saída: Q_a: Qualidade da ação a em relação a V

$V' \leftarrow$ `Marcação`(V)// *Transforma todos* $b_i \rightarrow b'_i$ *e* $x_i \rightarrow x'_i$ *(Etapa 1)*
$Q \leftarrow \mathcal{R}(\vec{b}, \vec{x}, a) \oplus (\gamma \cdot V')$ // *Qualidade como função do estado atual e futuro (Etapa 2)*
para cada x'_j *em* $\vec{X}$ **faça**
  // *Integral contínua por substituição (Etapa 3)*
  $Q \leftarrow Q\{x'_j \rightarrow \mathcal{P}(\vec{b}, \vec{x}, a, \vec{b'})\}$
para cada b'_j *em* $\vec{B}$ **faça**
  //*Marginalização das variáveis discretas (Etapa 4)*
  $Q := \Big[Q \otimes \mathcal{P}(b'_i|\vec{b}, \vec{x}, a)\Big]\Big|_{b'_i=1} \oplus \Big[Q \otimes \mathcal{P}(b'_i|\vec{b}, \vec{x}, a)\Big]\Big|_{b'_i=0}$
retorna Q

Tendo apresentado como é feita a atualização de Bellman de forma simbólica, nas cinco etapas descritas anteriormente, podemos definir a técnica SDP como uma versão simbólica do algoritmo Iteração de Valor, (Algoritmos 2.1 e 2.2). Note que as cinco etapas estão demarcadas nas linhas correspondentes dos algoritmos. O Algoritmo 2.1 realiza a programação dinâmica em seu laço externo sobre o horizonte h, e a maximização entre as qualidades no laço interno sobre as ações $\mathcal{A}$. O Algoritmo 2.2 realiza as operações simbólicas para obter a qualidade Q_a de uma ação.

2.2.4 Programação Dinâmica Simbólica com Variáveis de Estado e Ação Contínuas

O trabalho de Zamani *et al.* (2012) estende a técnica SDP apresentada na seção anterior para permitir que as ações possuam parâmetros contínuos. Além de manter a suposição do modelo de transição estocástico

discreto, é feita a restrição que as funções baseadas em casos $\mathcal{R}$ e $\mathcal{P}$ devem ser lineares por partes. Essa restrição é necessária para que a maximização dos parâmetros contínuos possa ser realizada de forma exata. A única modificação que precisa ser feita no algoritmo descrito na seção anterior para suportar ações contínuas é na etapa da maximização, pois para as outras etapas, os parâmetros $\vec{y}$ são apenas variáveis simbólicas que são substituídas e manipuladas como as outras. Assim, a *marcação*, *adição*, *integração* e *marginalização* são realizadas da mesma forma que em Sanner *et al.* (2011), e produzem uma função simbólica $Q_a(\vec{y})$ que contém variáveis do estado atual $(\vec{B}, \vec{X})$ e parâmetros $(\vec{y})$. Para obter as qualidades Q_a ótimas para cada ação paramétrica a é necessário fazer uma maximização contínua sobre os parâmetros $\vec{y}$:

$$Q_a \leftarrow \max_{\vec{y}} Q_{a(\vec{y})}$$

Para realizar essa maximização, a primeira simplificação é notar que cada parâmetro y_j pode ser maximizado independentemente, gerando funções simbólicas apenas do estado e dos outros parâmetros $(y_j \neq y_i)$, então a maximização é em uma variável de cada vez:

$$Q_a \leftarrow \max_{y_1}(\max_{y_2}(\dots \max_{y_{d(a)}} Q_{a(\vec{y})}) \dots))$$

Agora, iremos definir como realizar a maximização simbólica de uma função linear por partes $f(\vec{y})$ em relação a uma de suas variáveis, y_1, gerando uma função das outras variáveis, $\vec{y}^{-1} = (y_2, \dots, y_n)$. Para isso, lembramos que o máximo de uma função por partes é o maior dos máximos de suas partes, ou seja:

$$\max_{y_1}\left(\begin{cases}\phi_1: & f_1(\vec{y}) \\ \phi_2: & f_2(\vec{y}) \\ \dots: & \dots \\ \phi_n: & f_n(\vec{y})\end{cases}\right) = \max\left(\max_{y_1}(\phi_1 : f_1(\vec{y})), \max_{y_1}(\phi_2 : f_2(\vec{y})), \dots, \max_{y_1}(\phi_n : f_n(\vec{y}))\right)$$

Por isso, basta mostrarmos como calcular o máximo simbólico em uma função definida em uma única região. Note que as funções do tipo $\max_{y_1}(\phi_i : f_i(\vec{y}))$ são parciais, mas o máximo entre elas se torna uma função completa já que para qualquer atribuição das variáveis $(\vec{y}^{-1})$ pelo menos uma delas será satisfeita e tomamos o máximo entre essas pois nesse caso a escolha depende apenas de y_1, que estamos maximizando.

Para obtermos o máximo de uma função de uma única região, vamos primeiro classificar as decisões em $\phi_i(\vec{y})$ em três categorias:

- Independentes: São as decisões de ϕ_i em que a variável y_1 que está maximizando não ocorre. Essas decisões não restringem o máximo atingido nesta região, e dependem apenas de outras variáveis, caso elas não sejam satisfeitas o máximo será de outra região. Será denotado por $Ind(\phi_i)$ o conjunto das decisões de ϕ_i independentes de y_1.
- Limitantes Inferiores: São as decisões da forma $y_1 > l_i(\vec{y}^{-1})$ que limitam inferiormente os valores que a variável y_1 pode assumir. As funções l_i juntas definem o limite inferior, que também é uma função das outras variáveis $\vec{y}^{-1}$, $LI(\vec{y}^{-1}) = \max(l_1(\vec{y}^{-1}), \dots, l_n(\vec{y}^{-1}))$
- Limitantes Superiores: São as decisões da forma $y_1 < u(\vec{y}^{-1})$ que limitam superiormente os valores que a variável y_1 pode assumir. As funções u_i juntas definem o limite superior, que também é uma função das outras variáveis $\vec{y}^{-1}$, $LS(\vec{y}^{-1}) = \max(u_1(\vec{y}^{-1}), \dots, u_n(\vec{y}^{-1}))$

Como o máximo de uma função linear é sempre encontrado em um de seus extremos, temos que neste caso será o máximo entre $f_i(y_1 = LI)$ e $f_i(y_1 = LS)$. Com essa substituição estamos removendo as decisões que dependiam de y_1, mas para garantir que os limites inferiores e superiores ainda são respeitados precisamos garantir que $LI < LS$. Desta forma, o máximo da função de uma região é o máximo entre seu valor nos extremos, mantendo as restrições independentes e com a restrição adicional da ordem dos extremos:

$$\max_{y_1}(\phi_i : f_i(\vec{y})) = Ind(\phi_i) \wedge (LI < LS) : \max(f_i(y_1 = LI), f_i(y_1 = LS))$$

Esse máximo é uma função parcial de $\vec{y}^{-1}$ e podemos juntá-la com a das outras regiões para obter a função completa $g(\vec{y}^{-1}) = \max_{y_1}(f(\vec{y}))$.

Uma vez que temos as funções máximas para cada ação paramétrica, a maximização discreta entre as ações paramétricas é igual a maximização entre diagramas vista na seção anterior, de forma que a única modificação no algoritmo de programação dinâmica simbólica para tratar das ações contínuas é a adição da etapa de maximização contínua (linha 8) no Algoritmo 2.3.

Algoritmo 2.3: `SDP-HMDP` (HMDP $\mathcal{M}$, $\mathcal{H}$) $\longrightarrow$ ($V^*(\mathcal{H})$)

Entrada: $\mathcal{M}$: Descrição do problema; $\mathcal{H}$: Horizonte máximo para a solução
Saída: $V^*(\mathcal{H})$: Função valor ótima restando $\mathcal{H}$ passos;

```
1  V*(0) = 0// Inicialização da Programação Dinâmica
2  para h de 1 até H faça
3      // Efetua atualização de Bellman, Equação 2.19
4      V*(h) ← −∞ //Valor inicial para a maximização
5      para cada ação a em A faça
6          Q_a ← Avaliação(V*(h − 1), a)// Algoritmo 2.2
7          para cada parâmetro y_i em Q_a faça
8              Q_a = MaxContinuo(Q_a, y_i) //Maximização simbólica contínua
9          V*(h) ← max(V*(h), Q_a) //Maximização discreta
10 retorna V*(H)
```

2.3 Trabalhos Correlacionados

Planejamento probabilístico em espaços híbridos é uma tarefa extremamente complexa em seu caso geral, devido a complexidade na estrutura do espaço de estados e ações e nas funções de transição e recompensa. Trabalhos nesta área fazem suposições restritivas para obter um problema tratável. Além disso, os trabalhos se diferenciam pelo formato da solução gerada, por exemplo, podem obter solução exata para todo o espaço, aproximada para um estado inicial, aproximada com garantias de risco, entre outros. A seguir iremos descrever algumas das soluções anteriores relevantes.

2.3.1 Soluções baseadas em Controle Ótimo

O problema de planejamento em espaços contínuos em inteligência artificial se assemelha com o problema de controle ótimo na teoria de controle. Em ambos casos buscamos utilizar atuadores, representados por ações em planejamento e por controles na teoria de controle, de forma a otimizar o desempenho de um agente, considerando a dinâmica do ambiente onde ele está. A principal diferença entre planejamento e controle reside na forma como os atuadores são controlados. Na teoria de controle é comum considerar que controles são aplicados e modificados contínuamente no tempo, fomando trajetórias contínuas e utilizando equações diferenciais para representar a dinâmica do sistema. Em planejamento, é comum considerar a evolução do sistema como uma sequência de estágios, as ações são tomadas apenas em intervalos e a transição é vista de forma descontínua. Mesmo assim, na solução de muitos problemas de controle onde não há solução analítica, é comum se discretizar o tempo e nessa aproximação se obtém um modelo equivalente aos processos de decisão markovianos usados em planejamento.

Ono *et al.* (2013) apresenta um planejador para ambientes contínuos que utiliza um modelo semelhante ao de controle ótimo. Em seu modelo, é suposto uma discretização no tempo e a dinâmica é dada por transformações lineares no estado e no controle e o efeito probabilístico é tratado apenas como um ruído gaussiano. Uma contribuição desse trabalho é uma nova representação dos objetivos do agente. Além de uma função de custo a minimizar, são utilizados eventos e episódios para codificar metas ou restrições na trajetória do sistema, especialmente são fornecidas garantias de encontrar trajetórias ótimas dentro de limites de risco. Por exemplo, garantir que atinje todas metas com 99% de probabilidade e que só viola restrições com 0.001% de probabilidade. A relevância desse trabalho para o deste mestrado se dá para ilustrar como as escolhas de modelagem alteram a resolução do problema e mesmo como uma solução alternativa para nossos problemas.

2.3.2 Programação Dinâmica Estruturada

Aqui discutiremos trabalhos que resolvem HMDPs de horizonte finito através de representação estruturada de sua função valor e utilização de programação dinâmica (iteração de valor) nessas estruturas. Nesses trabalhos são feitas suposições que garantem que a retropropagação de Bellman não aumente a complexidade função valor, permitindo um numero arbitrário de iterações.

O trabalho de Feng *et al.* (2004) supõe um conjunto finito de ações, e que as funções de transição e recompensa são lineares por partes rectilineares, isto é, lineares em regiões definidas por inequações uni-

dimensionais ou retângulos. Os efeitos probabilísticos são discretos, ou seja, existe um conjunto finito de resultados possíveis para cada par estado-ação. Com essas suposições, a função valor encontrada em cada iteração se mantém linear por partes rectilineares e a retropropagação depende dessa estrutura. Além disso, para o caso híbrido, com variáveis discretas e contínuas, a estrutura utliizada, KD-Tree, não permite compartilhamento e é necessário estruturas independentes para cada valoração dos estados discretos, o que pode exigir um grande gasto de memória mesmo com poucas variáveis discretas. A solução é teoricamente exata, porém como as suposições são muito restritivas é considerado que há uma etapa prévia de aproximação para se obter um modelo linear com restrições unidimensionais e a solução final seria aproximada por ser a solução exata do modelo aproximado.

Li e Littman (2005) propuseram uma abordagem que permite transições cuja densidade de probabilidade é uma função constante por partes. A convolução, uma das etapas da retropropagação, de uma função polinomial por partes com uma função constante por partes gera uma função polinomial por partes de grau maior, por isso esse modelo não permite uma solução exata com estrutura fechada pela retropropagação. Para manter a estrutura da função valor, este trabalho incluí uma etapa de aproximação a cada iteração. A função valor é mantida na forma constante por partes, e após a convolução, se torna linear por partes, que é então aproximada por uma função constante por partes , recuperando a estrutura para a próxima iteração. Assim, este trabalho, denominado aproximação adiada (do inglês Lazy Approximation) permite um modelo mais expressivo em troca de obter uma solução aproximada.

Capítulo 3

Programação Dinâmica Simbólica Aproximada para Processos de Decisão Markovianos Híbridos

Nesse capítulo apresentamos um dos resultados deste mestrado, que foi parcialmente realizado durante o estágio de pesquisa realizado no NICTA, Australia, sob a orientação do pesquisador Scott Sanner [Sanner (2013)], e que gerou o artigo apresentado em julho de 2013 na conferência internacional *Uncertainty on Artificial Intelligence* [Vianna *et al.* (2013)].

Os algoritmos de programação dinâmica simbólica com variáveis contínuas, apresentados no Capítulo 2, são soluções exatas para HMDPs, porém sua eficiência é limitada. Enquanto a representação com XADDs é uma forma eficiente de manipular funções definidas por partes, já que compartilha partições de acordo com independência de contexto, ela ainda pode ficar muito complexa para funções com muitas regiões. Para muitos problemas de planejamento, a solução do HMDP representada pela função valor ótima pode ser definida por muitas partes e não permitir uma representação exata compacta, ainda que exista uma função compacta que represente aproximadamente a mesma função. Por exemplo, a função na Figura 3.1(a), representada pelo XADD da Figura 3.1(c), pode ser aproximada por uma função mais simples se permitirmos um erro pequeno, com a função na Figura 3.1(b) e seu diagrama em Figura 3.1(d).

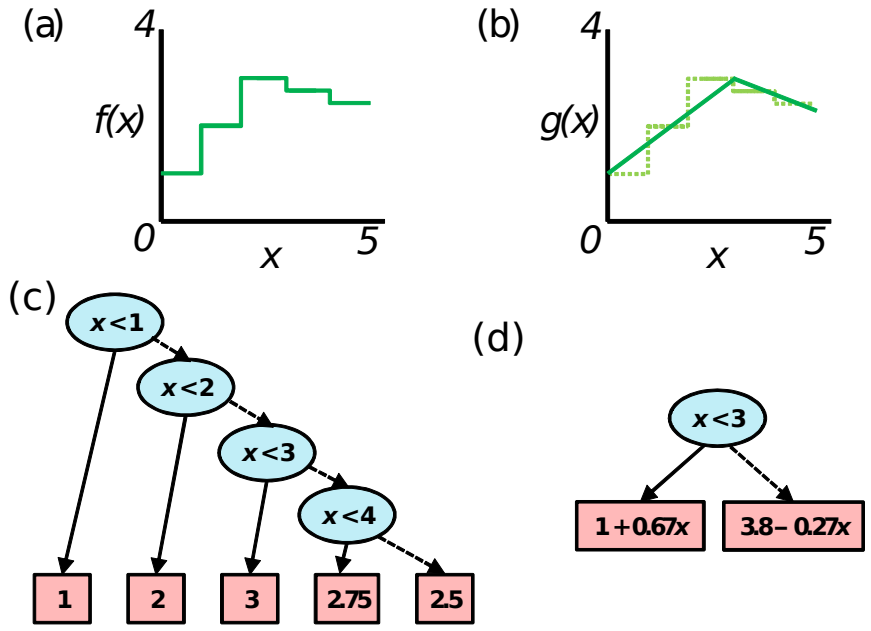


Figura 3.1: *(a) Uma função constante por partes $f(x)$ para $x \in [0, 5]$; (c) O XADD que representa f(x); (b) A versão aproximada $g(x)$ de $f(x)$; (d) O XADD da aproximação g(x).*

Assim, em busca de uma versão de programação dinâmica simbólica aproximada, neste capítulo apresentamos uma operação de compactação de XADDs com erro limitado e mostramos como pode ser utilizada

na solução de um HMDP.

3.1 Técnica de Aproximação para Diagramas de Decisão Algébricos Estendidos

Vimos na Seção 1.3.2 que, para o caso de variáveis discretas, é possível aproximar um ADD através de uma técnica que une os nós terminais com valores próximos, criando um novo nó cujo valor é a média dos valores originais. Como a estrutura de diagrama de decisão é semelhante para ADDs e XADDs, a estratégia de usar a união de nós terminais para gerar a função aproximada é conveniente e também pode ser utilizada na compactação de XADDs. No entanto, num XADD, os nós terminais contém expressões lineares e, nesse caso, usar a média aritmética para gerar a expressão do novo nó não é uma solução ótima. Como exemplo disso, pode-se mostrar o caso de aproximar uma função escada como a da Figura 3.1(a). A função que minimiza o erro tem expressões dependentes de x, apesar de que os nós da função original contém apenas expressões constantes. Na verdade, será necessário resolver um problema de otimização para encontrar os coeficientes da expressão que melhor aproxima as funções anteriores. Assim, o problema de obter uma função aproximada é subdividido em duas etapas: (1) *Aproximação por Fusões Sucessivas*; (2)*Fusão de Nós Terminais*;

3.1.1 Aproximação por Fusões Sucessivas

Para aproximar uma função definida por partes por uma nova, com menos regiões, é razoável considerar que as divisões de uma boa aproximação estão presentes na função original, como é feito na aproximação do APRICODD [St-Aubin *et al.* (2000)] para ADDs. Assim, iremos realizar a aproximação através de simplificações da função original. A operação de simplificação que iremos utilizar será a união de dois nós terminais, que, aplicada repetidamente, permite a redução significativa de todo o diagrama. A opção por fazer fusões de dois nós terminais evita o problema combinatório de juntar conjuntos de nós de tamanho arbitrário, mas não limita a estrutura gerada pois conjuntos de nós com mais elementos ainda podem se unir por etapas sucessivas. Um aspecto importante do procedimento é que a cada fusão, o novo nó criado armazena também o erro máximo acumulado nas sucessivas fusões, e assim podemos controlar o erro numa sequência de fusões. A Figura 3.2 demonstra, num exemplo simples, o procedimento de fusões sucessivas.

que é, descrito no Algoritmo 3.1. No pseudocódigo, *Aberto* é um conjunto com os nós terminais que ainda podem ser unidos. Enquanto há nós em *Aberto* escolhemos um nó, L_1 no algoritmo, e tentamos uni-lo com todos os outros, um de cada vez. A cada união que é efetivada L_1 muda e o outro nó é removido do conjunto, já que agora ele faz parte de L_1. O algoritmo acaba quando todas uniões de pares de nós formariam um erro maior que ϵ, logo não é possível mais fusões. Para completar a técnica de aproximação falta apenas definir como são encontrados a expressão para o novo nó e o erro pela substituição, o que será apresentado na seção seguinte.

3.1.2 Fusão de Nós Terminais

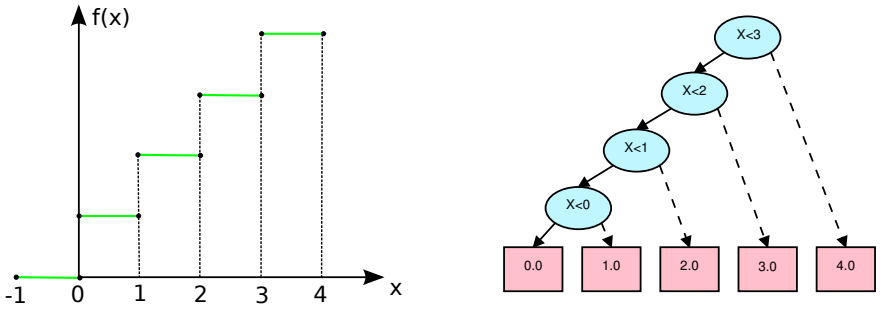
Para realizar a fusão de nós terminais devemos resolver o seguinte problema: Dadas duas funções do tipo caso lineares $L_1 = \langle f_1, \phi_1 \rangle$ e $L_2 = \langle f_2, \phi_2 \rangle$, qual é a melhor aproximação linear para L_1 e L_2? Como a expressão do nó obtido pela fusão é válida na união das duas regiões, $\phi_1 e \phi_2$, estamos procurando uma função tipo caso da forma $L^* = \langle f^*, \phi_1 \vee \phi_2 \rangle$. Como estamos considerando apenas expressões lineares em $\vec{X}$ podemos representá-las vetorialmente: $f_1 = \vec{c_1} \cdot \vec{x}$, $f_2 = \vec{c_2} \cdot \vec{x}$ e $f^* = \vec{c^*} \cdot \vec{x}$. Logo, o problema se reduz a encontrar o vetor de coeficientes $\vec{c^*}$ ótimo, i.e. o vetor $\vec{c^*}$ que minimiza o erro quando a função f^* substitui f_1 e f_2. O erro que queremos minimizar é o erro máximo na região, (ϕ_1 ou ϕ_2 respectivamente para f_1 ou f_2). Formalmente, o problema para determinar c^* é o seguinte:

$$\min_{\vec{c^*}} \max_{i \in \{1,2\}} \max_{\vec{x} \in C(\phi_i)} |\underbrace{\vec{c_i} \cdot \vec{x}}_{f_i(\vec{x})} - \underbrace{\vec{c^*} \cdot \vec{x}}_{f^*(\vec{x})}| \tag{3.1}$$

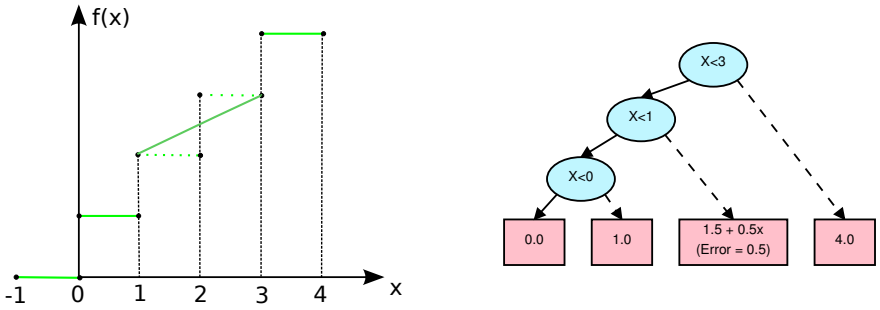
Note que essa forma é bilinear, já que envolve o produto de $\vec{c^*}$ e $\vec{x}$.

Para deixar mais clara a estrutura deste problema de otimização, nos referimos a Figura 3.3: Na esquerda, temos duas funções f_1 e f_2 em suas respectivas regiões $C(\phi_1)$ e $C(\phi_2)$; Na direita, mostramos uma possível expressão f para a fusão das funções f_1 e f_2.

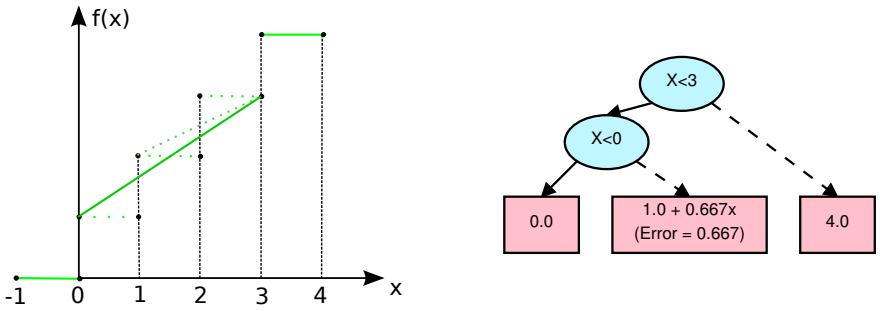
Como as regiões definidas por ϕ_1 e ϕ_2 podem ser uniões de politopos, mas queremos uma obter formulação de problema linear, iremos decompô-los em seus politopos constituintes. $C(\phi_i) := \bigcup_j Politopo(\theta_{ij})$. Além



(a) *Original*



(b) *Etapa 1*



(c) *Etapa 2*

Figura 3.2: *Exemplo unidimensional da aproximação de XADD através de fusões sucessivas de nós terminais. A cada etapa dois nós são selecionados, um novo nó com a expressão que minimiza o erro é criado e, se o erro for menor que o permitido para a aproximação, os nós são unidos e a estrutura do XADD é simplificada, removendo decisões que se tornaram desnecessárias.*

disso, introduzimos uma variável extra ϵ para representar o erro, i.e. o erro máximo entre todos politopos. Assim, nós reescrevemos a formulação (4.8) como um problema de otimização linear em dois níveis:

$$\min_{\vec{c^*},\epsilon} \epsilon \qquad (3.2)$$
$$s.t.\ \epsilon \geq \begin{pmatrix} \max\limits_{\vec{x}} & \left|\vec{c_i}\cdot\vec{x} - \vec{c^*}\cdot\vec{x}\right| \\ s.t. & \vec{x} \in Politopo(\theta_{ij}) \end{pmatrix}; \forall i \in \{1,2\}, \forall \theta_{ij}$$

As restrições em (3.2) não são lineares por causa do operador valor absoluto $|\cdot|$, porém podem ser transformadas em lineares facilmente, substituindo o valor absoluto pelo máximo entre seu argumento e seu oposto, da seguinte maneira:

$$\epsilon \geq \left|\vec{c_i}\cdot\vec{x} - \vec{c^*}\cdot\vec{x}\right|$$

Algoritmo 3.1: `AproximaXADD`(XADD X, erro ϵ) $\longrightarrow$ (XADD $\hat{X}$, erro $\hat{\epsilon}$)

Entrada: X: XADD que queremos reduzir, ϵ: erro permitido na aproximação
Saída: $\hat{X}$: Aproximação de X, $\hat{\epsilon}$: erro máximo efetuado

```
1  ε̂ ← 0
2  X̂ ← X
3  Aberto ← {L_i ∈ X̂} // onde L_i = ⟨φ_i, f_i⟩ são nós terminais em X̂
4  enquanto Aberto ≠ ∅ faça
5      L_atual ← Aberto.pop()
6      para cada L ∈ Aberto faça
7          //Calcula expressão e erro para a união destes nós
8          (f*, ε̃) ←Fusão(L_atual, L)//Algoritmo 3.2 na Seção 3.1.2
9          f*.erro ← ε̃ + max(f_1.erro, f_2.erro)
10         // Une nós se este erro ainda é permitido
11         se f*.erro < ε então
12             ε̂ ← max(ε̂, f*.erro)
13             Aberto.remove(L)
14             X̂.substitui(L_atual, L, f*) // Substitui nós terminais em X̂
15             L_atual ← ⟨φ_1 ∨ φ_2, f*⟩// Continua fazendo fusões com L_atual
16 retorna (X̂, ε̂)
```

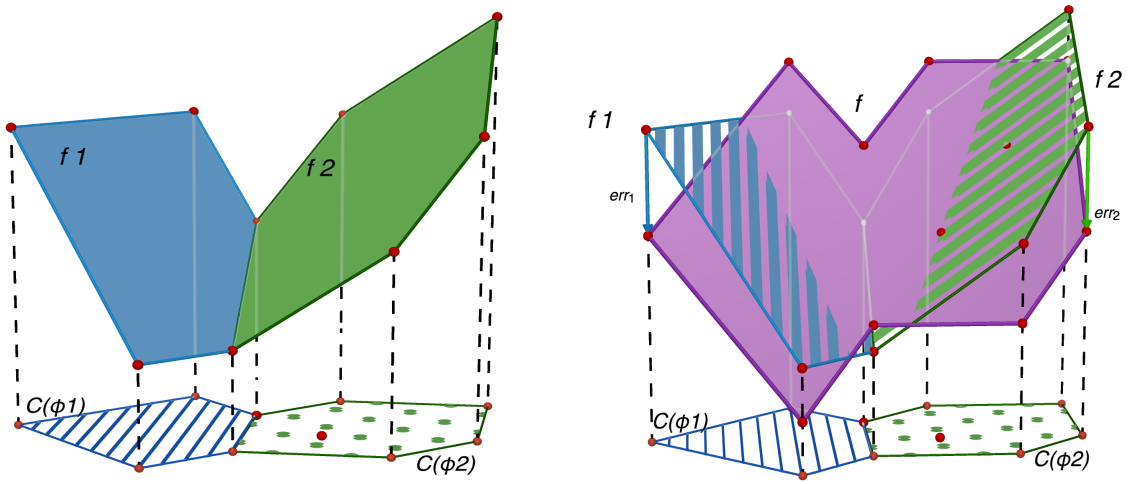


Figura 3.3: *Ilustração do problema de fusão de nós terminais.* (Esquerda) *funções originais* f_1 *e* f_2 *em suas respectivas regiões* $C(\phi_1)$ *e* $C(\phi_2)$. (Direita) *uma aproximação linear* f *sobreposta a* f_1 *e a* f_2*, em suas respectivas regiões, mostrando erro nos vértices dos politopos* (err_1, err_2).

equivale a

$$\epsilon \geq \left(\vec{c_i} \cdot \vec{x} - \vec{c^*} \cdot \vec{x}\right) \,\&$$
$$\epsilon \geq \left(-\vec{c_i} \cdot \vec{x} + \vec{c^*} \cdot \vec{x}\right)$$

Ainda que o problema descrito por (3.2), continue um problema não linear, vamos mostrar que o problema de maximização interno pode ser removido totalmente. Implicitamente, o estágio de maximização corresponde a um numero *infinito* de restrições, basicamente $\epsilon \geq Erro(\vec{x}) \forall \vec{x} \in Politopo(\theta_{ij})$, observando a Figura 3.3 podemos ver como substitui-las por um conjunto *finito* de restrições. Como f, f_1, e f_2 são todas expressões lineares e $C(\phi_1), C(\phi_2)$ representam (uniões de) politopos lineares convexos, sabemos que o máximo das expressões lineares $f - f_i$ ou $f_i - f$ será atingido em um dos vértices dos politopos convexos.

Assim, denotando por $\vec{x}_{ij}^k$ ($k \in \{1 \ldots N_{ij}\}$) cada vértice do politopo definido pelas restrições lineares θ_{ij}, obtemos um programa linear para (4.8) com um número *finito* de restrições:

$$\min_{\vec{c}^*,\epsilon} \epsilon \tag{3.3}$$
$$s.t.\ \epsilon \geq \left|\vec{c_i} \cdot \vec{x}_{ij} - \vec{c^*} \cdot \vec{x}^k_{ij}\right|;\ \begin{matrix}\forall i \in \{1,2\}, \forall \theta_{ij}, \\ \forall k \in \{1 \dots N_{ij}\}\end{matrix}$$

No entanto, essa formulação ainda é ineficiente, já que para n_{ij} restrições lineares no $Politopo(\theta_{ij})$, o número de vértices pode ser *exponencial*, ou seja, $N_{ij} = O(\exp n_{ij})$. Porém essa limitação, pode ser resolvida com a seguinte consideração: ainda que o número de restrições em (3.3) possa ser exponencial no pior caso, se fixarmos os coeficientes $\vec{c^*}$ existe um modo eficiente de encontrar o ponto $\vec{x^k_{ij}}$ de cada $Politopo(\theta_{ij})$ que gera erro máximo — Isso é exatamente o que obtemos no segundo estágio do programa linear (3.2). Portanto, é possível usar a técnica de *geração de restrições* para resolver (3.3) sem representar um numero exponencial de restrições. O Algoritmo 3.2 descreve nossa abordagem.

Iniciando com um conjunto de restrições C vazio e um vetor arbitrário $\vec{c}$, nós iterativamente calculamos os pontos de cada politopo convexo de erro máximo para $\vec{c}$ e adicionamos as restrições correspondentes a esses pontos em C. Então resolvemos o problema de minização do erro sujeito a C para obter um novo $\vec{c}$. Enquanto o erro na minimização mudar entre iterações significa que estamos ajustando a novos pontos do politopo, mas quando houverem vértices o suficiente para seja impossível melhorar o erro, teremos atingido os valores ótimos, de coeficientes e erro.

Finalmente, observamos que o Algoritmo 3.2 sempre termina, já que a cada etapa são adicionadas pelo menos uma nova restrição e o número total de restrições é finito. Na Seção 3.2.2 mostramos que o algoritmo funciona eficientemente sugerindo que de fato é gerado apenas um subconjunto pequeno do possivelmente exponencial conjunto de restrições.

Algoritmo 3.2: $\texttt{Fusão}(L_1, L_2) \longrightarrow (\vec{c}^*, \epsilon)$

Entrada: L_1, L_2: funções caso que pretendemos unir
Saída: $\vec{c}^*$: coeficientes da expressão ótima, ϵ: erro mínimo para a fusão

1 $\vec{c^*} := \vec{0}$ *// Inicialização arbitrária*
2 $\epsilon^* := \infty$ *// Erro mínimo encontrado*
3 $C := \emptyset$ *// Inicia conjunto de restrições vazio*
4 *// Crie as restrições dos vértices com maior erro para* $\vec{c^*}$
5 **para** $i \in \{1,2\}, \theta_{ij} \in C(\phi_i)$ **faça**
6
$$\vec{x}^k_{ij+} := \arg\max_{\vec{x}} \left(\vec{c_i} \cdot \vec{x} - \vec{c^*} \cdot \vec{x}\right)$$
$$\text{s.t. } \vec{x} \in Politopo(\theta_{ij})$$
$$\vec{x}^k_{ij-} := \arg\max_{\vec{x}} \left(\vec{c^*} \cdot \vec{x} - \vec{c_i} \cdot \vec{x}\right)$$
$$\text{s.t. } \vec{x} \in Politopo(\theta_{ij})$$
$$C := C \cup \{\epsilon > (\vec{c_i} - \vec{c^*}) \cdot \vec{x}^k_{ij+}\}$$
$$C := C \cup \{\epsilon > (\vec{c^*} - \vec{c_i}) \cdot \vec{x}^k_{ij-}\}$$
7 *// Re-resolva o LP com as novas restrições*
8 $(\vec{c^*}, \epsilon^*_{new}) := \arg\min_{\vec{c}^*,\epsilon} \epsilon$ sujeito a C
9 **se** $\epsilon^*_{new} \neq \epsilon^*$ **então**
10 $\epsilon^* := \epsilon^*_{new}$, **volte** para a linha 5
11 **retorna** $(\vec{c^*}, \epsilon^*)$*// Coeficientes ótimos e erro mínimo*

Juntando o resultado do algoritmo de fusão dessa seção com o de aproximação por fusões sucessivas da Seção 3.1.1, descrevemos um procedimento para aproximar XADDs com controle do erro usado. A seguir, apresentaremos o algoritmo baseado em SDP que desenvolvemos usando esse procedimento de aproximação durante a resolução de HMDPs.

3.2 Programação Dinâmica Simbólica com XADDs Aproximados

Nesta seção, descrevemos como a técnica de aproximação definida na Seção 3.1 é utilizada num algoritmo de SDP com variáveis contínuas para resolver HMDPs de forma eficiente.

3.2.1 Algoritmo BASDP

O algoritmo *BASDP (Bounded Approximate Symbolic Dynamic Programming)*, Algoritmo 3.3, junto com os Algoritmos 3.1 e 3.2 é nossa versão aproximada com erro limitado de SDP. Os algoritmos de SDP com variáveis contínuas se baseiam essencialmente na manipulação simbólica da função valor até obter a função ótima para o horizonte desejado. Todas operações são realizadas entre diagramas XADD, portanto uma redução no tamanho da estrutura XADD da função valor aumenta a eficiência de todas operações de planejamento. Como a cada iteração da SDP as funções representadas são funções valor para horizontes maiores, é esperado que a complexidade da função cresça e o número de regiões na XADD aumente. Nossa versão aproximada busca evitar esse crescimento adicionando uma etapa de aproximação entre as iterações, com a técnica descrita na Seção 3.1. A descrição do BASDP está no Algoritmo 3.3. Baseando-se no Algoritmo 2.3 da SDP exata, adicionamos a etapa de aproximação com a chamada da função `AproximaXADD`(Algoritmo 3.1) na linha 9. O erro em cada aproximação é ϵ, como é feita exatamente uma aproximação por iteração, em $\mathcal{H}$ iterações o erro total é limitado pelo produto $\mathcal{H}\epsilon$.

Na seção seguinte apresentamos os experimentos realizados com BASDP em diferentes níveis de aproximação.

Algoritmo 3.3: `BASDP`(HMDP $\mathcal{M}$, $\mathcal{H}$, ϵ) $\longrightarrow V^{\mathcal{H}}$

Entrada: $\mathcal{M}$, HMDP para resolver, $\mathcal{H}$ numero de estágios, ϵ erro tolerado
Saída: $V^{\mathcal{H}}$: XADD aproximado da função valor ótima

1 $V^0 \leftarrow 0$
2 **para** h *de* 1 *até* $\mathcal{H}$ **faça**
3 $V^h = -\infty$
4 **para cada** $a \in A$ **faça**
5 $Q_a \leftarrow$ `Retropropagação`(V^{h-1}, a)
6 **para cada** *parâmetro* y_i *em* Q_a **faça**
7 $Q_a \leftarrow$ `MaxContinuo`(Q_a, y_i) // *Maximização dos parâmetros* $\vec{y}$
8 $V^h \leftarrow \max(V^h, Q_a)$ // *Maximização das ações paramétricas* a
9 $V^h =$ `AproximaXADD`(V^h, ϵ)
10 **retorna** $V^{\mathcal{H}}$

3.2.2 Resultados Experimentais

Nesta seção comparamos a eficiência da solução exata de SDP, representada por $\epsilon = 0$, com diferentes níveis de erro de aproximação, $\epsilon > 0$, para avaliar os benefícios obtidos em tempo e espaço em função do erro permitido na aproximação. Avaliamos o algoritmo BASDP em três domínios — Robô em Marte 1D, Robô em Marte 2D e Controle de Estoque— detalhados a seguir.

O problema do Robô em Marte 1D [Bresina *et al.* (2002)]: A posição do robô é representada por uma única variável contínua x e a meta é tirar fotografias em regiões específicas. As fotografias são tiradas automaticamente, sendo que o robô deve se posicionar corretamente para obter maior recompensa. A única ação é $mover(a_x)$, onde o parâmetro a_x é a distância. Na descrição do problema abaixo, existem duas localizações onde devem ser tiradas as fotos e duas variáveis booleanas (tf_1 and tf_2) que armazenam

quais fotos já foram tiradas. A dinâmica de transição para a ação $mover(a_x)$ é como segue:

$$tf'_1 = \begin{cases} tf_1 \vee (x > 40 \wedge x < 60) & : 1 \\ \text{c.c.} & : 0 \end{cases}$$
$$tf'_2 = \begin{cases} tf_2 \vee (x > -60 \wedge x < -40) & : 1 \\ \text{c.c.} & : 0 \end{cases}$$
$$x' = x + a_x.$$

Ou seja, se a posição do robô estiver entre 40 e 60, a foto 1 é tirada; se estiver entre -60 e -40, a foto 2 é tirada; a nova posição x' do robô é simplesmente adicionada de a_x. A função recompensa para o problema Robô em Marte 1Dé a composta por três termos: R_1 é a recompensa pela foto 1; R_2 é a recompensa pela foto 2; R_3 é a recompensa pelo deslocamento realizado, isto é:

$$R = R_1 + R_2 + R_3$$
$$R_1 = \begin{cases} (tf'_1) \wedge (\neg tf_1) \wedge (x > 50) & : 40 - 0.2 * (x - 50) \\ (tf'_1) \wedge (\neg tf_1) \wedge (x < 50) & : 40 - 0.2 * (50 - x) \\ (tf'_1) \wedge (tf_1) & : 1.1 \\ \text{c.c.} & : -2 \end{cases}$$
$$R_2 = \begin{cases} (tf'_2) \wedge (\neg tf_2) \wedge (x > -50) & : 60 - 0.2 * (-x + 50) \\ (tf'_2) \wedge (\neg tf_2) \wedge (x < -50) & : 60 - 0.2 * (x + 50) \\ (tf'_2) \wedge (tf_2) & : 1.2 \\ \text{c.c.} & : -1 \end{cases}$$
$$R_3 = \begin{cases} a_x > 0 & : -0.1 * a_x \\ a_x < 0 & : 0.1 * a_x \end{cases}$$

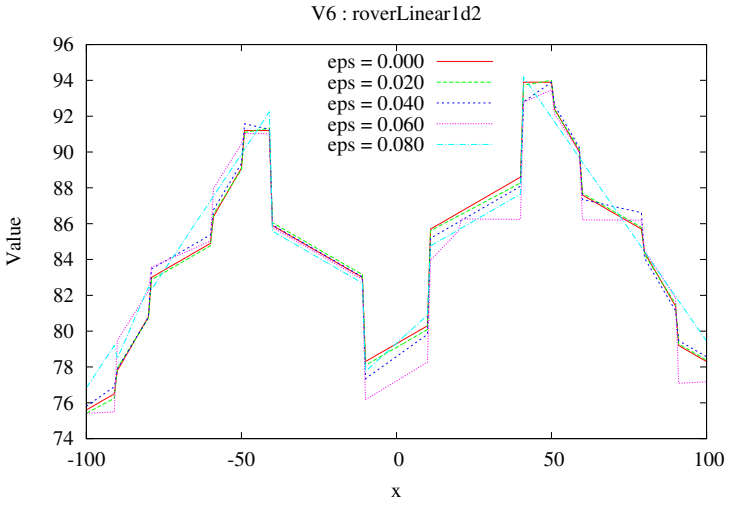


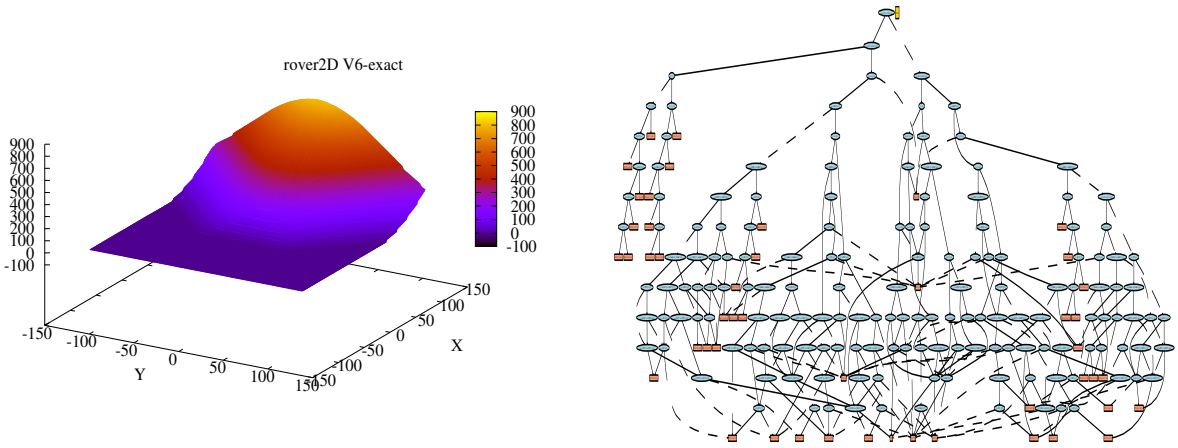
Figura 3.4: *Função valor da iteração 6 para o problema* Robô em Marte 1D*, mostrando como o erro considerado (eps) leva a diferentes aproximações.*

O domínio do robô unidimensional é usado para visualizar a variação na função valor e o efeito do erro de aproximação. Na Figura 3.4, apresentamos a função valor para horizonte 6 para o problema Robô em Marte 1Dobtida com diferentes níveis de aproximação(eps). Note que os picos nas regiões -50 e 50 indicam as localizações onde devem ser retiradas as fotos. Comparando as diferentes curvas, vemos que em geral valores de erro permitido ϵ maiores promovem um ajuste da função valor menos próximo da solução exata e com menos regiões. No entanto, a escolha gulosa na seleção de nós para unir permite que ocorram

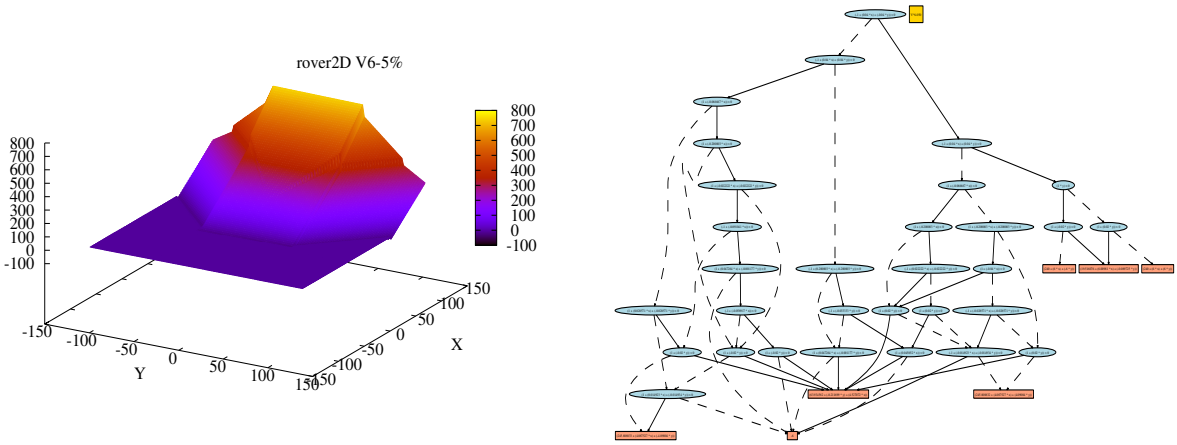
exceções, por exemplo na região próxima a $x = 0$, a aproximação com $eps = 0.80$ está mais próxima da solução exata que a aproximação com $eps = 0.6$.

O problema do Robô em Marte 2D: Uma variação do Robô em Marte 1D onde não há pontos para fotografias, mas o robô precisa seguir um caminho. A posição é representada por um par de variáveis contínuas (x, y) e existe apenas uma ação $move(a_x, a_y)$, com parâmetros a_x e a_y, limitados por $|a_x| < 10$ e $|a_y| < 10$. A nova posição é dada por $(x', y') = (x + a_x, y + a_y)$. Neste exemplo a recompensa é codificada por:

$$R = \begin{cases} (x > y + 25) \wedge (x > -y + 25) \wedge (y > 0) : & -10 + x - y \\ (x > y + 25) \wedge (x > -y + 25) \wedge (y < 0) : & -10 + x + y \\ \text{c.c.} : & -1 \end{cases}$$



(a) *Função Valor para a sexta iteração com SDP exata.*



(b) *Função Valor para a sexta iteração com SDP com 5% de aproximação.*

Figura 3.5: *Gráficos e diagrama XADD para a função valor na iteração 6 para o domínio* Robô em Marte 2D*; (a) Função valor exata; (b) Função valor com aproximação de 5%;*

Na Figura 3.5, podemos ver nitidamente o efeito da aproximação. Nos gráficos 3D a superfície com 5% de erro é muito mais simples e, correspondentemente, nos diagramas, o número de nós é drasticamente reduzido, permitindo computações muito mais velozes nas operações em XADDs.

Problema do Controle de Estoque [Scarf (2002)]: Neste problema, consideramos um estoque com n recursos contínuos que são comprados e vendidos. Existem n ações, *comprar-i* para cada recurso $1 \leq i \leq n$. A quantidade máxima de cada recurso que é vendida por iteração depende de uma variável aleatória booleana d que mede se a demanda está alta ou baixa, sendo que $d = 1$ (alta) com 60% de probabilidade. Quando uma ação *comprar-i* é executada, a quantidade do recurso x_i é aumentada em 200, e do todos recursos é reduzida pela quantidade que foi vendida, no máximo 150 para a demanda alta ou 50 para a demanda baixa. A representação em casos dessa transição pela ação *comprar-i* para x_i é:

$$x'_i = \begin{cases} (d') \wedge (x_i > 150) & : x_i + 200 - 150 \\ (d') \wedge (x_i < 150) & : 200 \\ (\neg d') \wedge (x_i > 50) & : x_i + 200 - 50 \\ (\neg d') \wedge (x_i < 50) & : 200 \end{cases}$$

e o estoque dos demais recursos x'_j, $1 \leq j \leq n$, $j \neq i$ é dado por:

$$x'_j = \begin{cases} (d') \wedge (x_j > 150) & : x_j - 150 \\ (d') \wedge (x_j < 150) & : 0 \\ (\neg d') \wedge (x_j > 50) & : x_j - 50 \\ (\neg d') \wedge (x_j < 50) & : 0 \end{cases}$$

$$R = \sum_k (x'_k - x_k)$$

,

A Figura 3.6 mostra a análise do tempo, espaço e erro de aproximação para as soluções obtidas com o algoritmo BASDP em diferentes níveis de aproximação, incluindo a solução exata (eps =0),para os domínios Robô em Marte 2D(gráficos inferiores) e Controle de Estoque(gráficos superiores). Nos gráficos de espaço (esquerda) vemos que a aproximação reduz o número de nós do diagrama drasticamente para horizontes maiores, mesmo com valores pequenos de ϵ(eps). Podemos observar que o número de nós no XADD da solução aproximada cresce linearmente com o horizonte, enquanto na solução exata o XADD tem crescimento não-linear.

Com relação ao tempo de solução (gráficos centrais), as soluções aproximadas chegam a ser quase 10× mais rápidas que a solução exata para os maiores horizontes. Além disso, examinamos o erro relativo máximo ocorrido na aproximação (direita) e contatamos em geral é significantemente menor que o erro permitido.

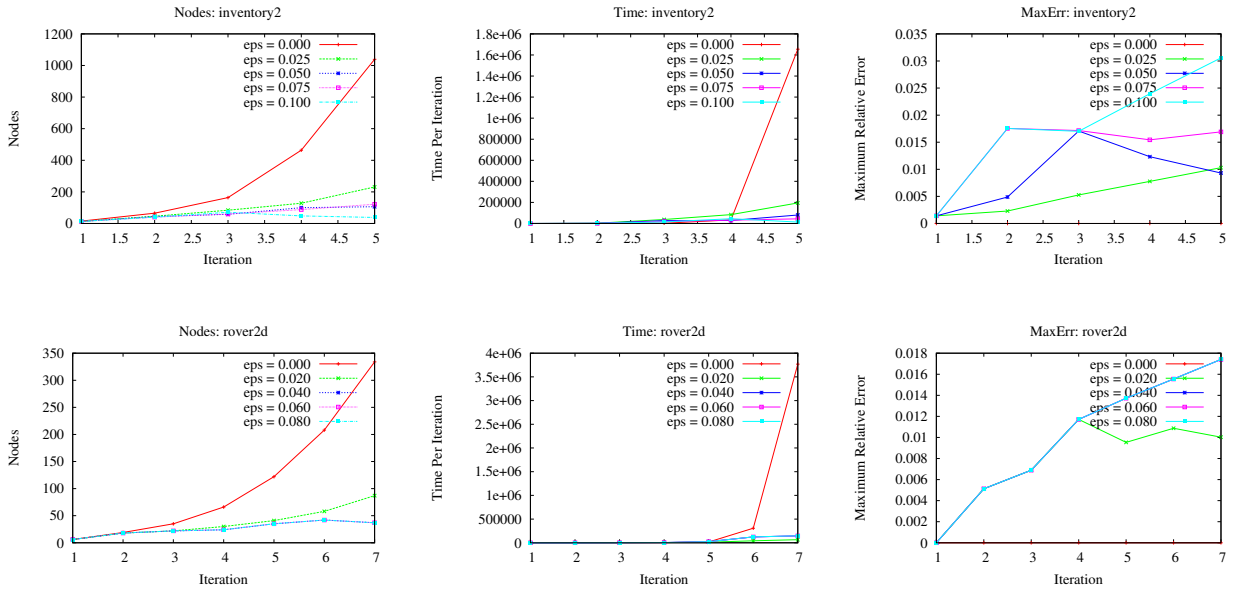


Figura 3.6: *Gráficos comparando a eficiência do algoritmo BASDP com 5 graus diferentes de aproximação para a solução dos problemas* Robô em Marte 2D *(baixo) e* Controle de Estoque *(cima). Espaço (número de nós)(esquerda); Tempo(milisegundos)(meio); Erro como fração do valor máximo(direita).*

Resumo

Neste capítulo apresentamos uma técnica para tornar a programação dinâmica simbólica mais eficiente. A quantidade de regiões que surgem na solução exata de problemas com variáveis contínuas é muito grande, e muitas vezes mesmo que existem grupos de regiões com valores muito próximos, ainda que expressões distintas. Para aproveitar essa semelhança entre as regiões e reduzir o tamanho dos diagramas propomos uma algoritmo aproximado com erro limitado, o *Bounded Approximate Symbolic Dynamic Programming - BASDP).*

A aproximação é realizada através de fusões de folhas com expressões compatíveis. Para calcular a melhor expressão num nó que substituí dois outros temos um problema bilinear, que pode ser transformado em um problema linear de dois níveis e resolvido com a técnica de geração de restrições. Após unirmos as folhas o diagrama como um todo fica mais compacto porque decisões internas ficam irrelevantes e são removidas. Para utilizar a aproximação na solução de HMDPs, é natural aplicar a função de aproximação no final de cada iteração do algoritmo de programação dinâmica simbólica.

Os experimentos preliminares usando a técnica BASDP mostraram que a aproximação da função valor pode ter um efeito significativo na redução do consume de espaço e tempo utilizados para a resolução de HMDPs, especialmente para horizontes maiores. Além disso, a capacidade de escolher o valor de erro a ser usado permite que o algoritmo seja usado em diferentes contextos, inclusive podem ser feitas soluções sucessivamente mais precisas modificando apenas o valor do erro.

Capítulo 4

Programação Dinâmica Simbólica Assíncrona para Processos de Decisão Markovianos Híbridos

Neste capítulo iremos apresentar nossa segunda estratégia original para aprimorar as soluções de programação dinâmica simbólica para HMDPs. Nossa ideia é generalizar a vantagem das soluções assíncronas para MDPs, especialmente o algoritmo RTDP (Seção 1.1.3), para problemas com variáveis contínuas, propondo o algoritmo *Real-time Symbolic Dynamic Programming - RTSDP*. A técnica de programação dinâmica simbólica vista no capítulo 2 é uma solução exata e completa para HMDPs, porém ela tem duas desvantagens que pretendemos tratar: (1) ela encontra a função valor ótima para todos estados do problema, o que pode ser muito maior do que a solução necessária quando se tem um estado inicial conhecido; (2) ela não possui uma forma de verificar se as decisões não lineares são incompatíveis e pode incorporar no diagrama uma quantidade enorme de nós e caminhos que não alcançados por nenhuma atribuição das variáveis. O algoritmo RTSDP proposto consegue resolver ambos problemas, pois usamos uma nova operação: a atualização local de regiões. Atualizar apenas uma região do diagrama permite resolver o problema (1) pois escolhemos atualizar apenas regiões relevantes para o estado inicial de interesse, minimizando a quantidade de informação desnecessário nos cálculos. A questão (2) também é tratada pelas atualizações locais, já que as regiões que são atualizadas serão sempre regiões possíveis, uma vez que temos um estado relevante encontrado dentro dessas regiões. Demonstramos a eficiência do algoritmo RTSDP comparando a sua eficiência com a do SDP original em três domínios de planejamento probabilístico com variáveis contínuas.

4.1 *Real-Time Symbolic Dynamic Programming - RTSDP*

Para facilitar a explicação do algoritmo RTSDP, vamos brevemente revisar o funcionamento e as propriedades do algoritmo RTDP no qual nos baseamos. Em seguida, descreveremos a nova operação de atualização de regiões que permite a solução assíncrona de HMDPs.

4.1.1 Revisão do RTDP original

Na Seção 1.1.3, descrevemos o algoritmo RTDP, proposto por Barto *et al* Barto *et al.* (1995) como uma das soluções assíncronas eficientes para MDPs. Vamos resgatar aqui a estrutura desse algoritmo para adaptá-la ao caso de variáveis contínuas.

O pseudocódigo do algoritmo RTDP para MDPs de horizonte finito está apresentado no Algoritmo 4.1.

O algoritmo funciona melhorando iterativamente uma função heurística inicial. A melhora da função ocorre em dentro de simulações de execuções, denominadas *trials*. Todos *trials* começam no estado inicial, e seguem da seguinte forma: primeiro atualizam o valor do estado corrente, depois escolhem uma ação gulosa e sorteiam um estado sucessor dessa ação. Desse modo todo estado visitado é relevante ao estado inicial, pois está num caminho onde todas escolhas foram gulosas.

A operação de atualização corresponde a calcular o termo direito da equação de Bellman (Equação 1.20) e atribuir este novo valor para o estado em questão. Esse algoritmo funciona da mesma maneira que as outras soluções de programação dinâmica, com a principal mudança que apenas um estado é atualizado em cada etapa, no entanto como o procedimento dos trials garante que todos estados relevantes serão atualizados até sua convergência, o procedimento tem a garantia de encontrar o valor ótimo para todos estados relevantes.

Algoritmo 4.1: RTDP ($\mathcal{M}$, s_0, H)

Entrada: $\mathcal{M}$: MDP, s_0: estado inicial, H: horizonte máximo do MDP
Saída: $V = (V^0, V^1, \ldots, V^H)$: função valor não-estacionária ótima para s_0

para cada h **em** *0,1,2,..., H* **faça**
  $V^h := g(h)$ % $g(h)$ *é uma heurística admissível para h passos*
enquanto nTrials < LimiteDeTrials **ou** Erro(V,s_0) $> \epsilon$ **faça**
  $V := \texttt{RealizaTrial}(\mathcal{M}, V, s_0, H)$
retorna V.

Função `RealizaTrial` ($\mathcal{M}$, s, V, h) $\longrightarrow V'$

Entrada: $\mathcal{M}$: MDP, s: estado atual, V: função valor atual, h: horizonte do trial.
Saída: V': função valor após o Trial.
se $h = 0$ **então**
  retorna V % Final do Trial
$V^h(s), a_g := \texttt{Atualiza}(M, s, V^{h-1})$ % Faz a atualização de Bellman em s
$s' := \texttt{Sorteia}(s, a_g, \mathcal{M})$ % `Sorteia` um novo estado segundo $\mathcal{T}$
return $\texttt{RealizaTrial}(\mathcal{M}, s', V, h-1)$

Função `Atualiza` ($\mathcal{M}$, s, V) $\longrightarrow V(s), a_g$

Input: $\mathcal{M}$: MDP, s: estado atual, V^{h-1}: função valor do próximo passo
Output: $V^h(s)$: valor atualizado de s, a_g: ação gulosa para s.
para cada $a \in \mathcal{A}$ **faça**
  $Q_a(s) := \sum_{s' \in S} P(s, a, s') \cdot [R(s, a, s') + \gamma V(s')]$
$a_g := \arg\max_a Q_a(s)$
$V^h(s) := Q_{a_g}(s)$
retorna $V(s), a_g$

4.1.2 Estendendo RTDP para Variáveis Contínuas

Agora vamos generalizar o algoritmo RTDP para tratar com variáveis contínuas e resolver HMDPs. O pseudocódigo do RTDP (Algoritmos 4.1) foram apresentados em funções para deixar claro quais são as partes que precisam ser modificadas para que o algoritmo seja usado em HMDPs. A função principal do algoritmo e a função `RealizaTrial` são definidas diretamente como operações na função valor, e por isso, podem ser diretamente aplicadas mesmo se a função valor tiver variáveis contínuas se uma representação simbólica eficiente for usada.

O procedimento que requer uma extensão mais interessante é a função `Atualiza` já que não queremos mais fazer a atualização de apenas um estado, mas definir uma nova operação para uma atualização local numa região contínua.

Para fazer isso, vamos modificar a atualização da programação dinâmica simbólica (Seção 2.2.3) que é síncrona para todos estados numa atualização de regiões. Ao invés de modificar as expressões de todos os caminhos do XADD da função valor, $V^{DD(\vec{b},\vec{x})}$, a atualização de uma região modifica apenas um dos caminhos do XADD, aquele que correspond a região que contem o estado que está sendo visitado $(\vec{b}_t, \vec{x}_t)$, denotada por Ω.

Essa operação é realizada através do uso de uma "máscara" que limita a operação de atualização. A "mascara" é um XADD indicador, ou seja, tem apenas duas folhas, uma representando regiões válidas e outra as inválidas. Na atualização de Ω a máscara é um indicador de se o valor de $(\vec{b}, \vec{x})$ corresponde ou não a região Ω, i.e.:

$$I[(\vec{b}, \vec{x}) \in \Omega] = \begin{cases} 0, & \text{se } (\vec{b}, \vec{x}) \in \Omega \\ NaN, & \text{caso contrario,} \end{cases} \tag{4.1}$$

onde 0 indica regiões válidas e NaN inválidas. O valor 0 é utilizado como válido pois quando aplicamos a

máscara com uma operação de soma ele não altera o valor da região de interesse, por outro lado, o valor NaN colapsa todas outras regiões num único nó, reduzindo muito o tamanho do XADD.

A máscara é aplicada nas funções de probabilidade, transição e recompensa:

$$T_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})} = T_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})} \oplus I[(b,x) \in \Omega] \tag{4.2}$$

$$P_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} = P_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \oplus I[(b,x) \in \Omega] \tag{4.3}$$

$$R_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} = R_a^{DD(\vec{b},\vec{x},\vec{y})} \oplus I[(b,x) \in \Omega] \tag{4.4}$$

Finalmente, a *atualização de uma região* é definida de forma análoga a atualização original, mas usando as versões restritas das funções simbólicas:

- Regressão de uma ação a na região Ω:

$$Q_{h,a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} = R_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \oplus \sum_{\vec{b'}} P_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \otimes \left[subst_{(x'=T_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})})} V_{h-1}^{\prime DD(b',x')} \right]. \tag{4.5}$$

- Maximização na região Ω:

$$V_{h,\Omega}^{DD(\vec{b},\vec{x})} = \max_{a \in \mathcal{A}} \left(\max_{\vec{y}} \; Q_{h,a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \right). \tag{4.6}$$

- Atualização da função valor V_h na região Ω:

$$V_{h,novo}^{DD(b,x)} \leftarrow \begin{cases} V_{h,\Omega}^{DD(b,x)}, & \text{se } (\vec{b},\vec{x}) \in \Omega \\ V_{h,antigo}^{DD(b,x)}, & \text{caso contrário,} \end{cases} \tag{4.7}$$

esta atualização é feita como uma minimização de XADDs: $V_{h,novo}^{DD(b,x)} \leftarrow \min(V_{h,\Omega}^{DD(b,x)}, V_{h,antigo}^{DD(b,x)})$.. Assim o novo valor de V_h está correto já que é $V_{h,\Omega}$ é considerado $+\infty$ para as regiões inválidas e $V_{h,\Omega} \leq V_{h,antigo}$ nas regiões válidas, assim preservando V_h como um limitante superior do valor ótimo V_h^*.

O algoritmo 4.4 descreve a nova atualização. Note que o estado atual é denotado por $(\vec{b_a}, \vec{x_a})$ para não ser confundido com as variáveis simbólicas $(\vec{b},\vec{x})$.

Algoritmo 4.4: `AtualizaRegião`(HMDP,$(\vec{b_a},\vec{x_a})$,V, h) $\longrightarrow$ V_h, a_g, $\vec{y}_g$

1 //`CriaMascara` *monta uma máscara com o caminho de* $(\vec{b_c},\vec{x_c})$)
2 $I[(\vec{b},\vec{x}) \in \Omega] \leftarrow$ `CriaMascara`$(V_h, (\vec{b_c},\vec{x_c}))$
3 $V'^{DD(b',x')} \leftarrow$`Marca` $(V_{h-1}^{DD(b,x)})$ //*Marca variáveis do próximo estado com '*
4 **para cada** $a(\vec{y}) \in \mathcal{A}$ **faça**
5 $\quad P_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \leftarrow P_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \oplus I[(b,x) \in \Omega]$
6 $\quad T_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})} \leftarrow T_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})} \oplus I[(b,x) \in \Omega]$
7 $\quad R_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \leftarrow R_a^{DD(\vec{b},\vec{x},\vec{y})} \oplus I[(b,x) \in \Omega]$
8 $\quad Q_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \leftarrow R_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \oplus \sum_{\vec{b'}} P_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \otimes \left[subst_{(x'=T_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})})} V'^{DD(b',x')} \right]$
9 $\quad \vec{y}_g{}^a \leftarrow \arg\max_{\vec{y}} Q_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})}$ //*Encontra o parâmetro que maximiza* Q
10 $V_{h,\Omega}^{DD(\vec{b},\vec{x})} \leftarrow \max_a \left(\text{pmax}_{\vec{y}} \; Q_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \right)$
11 $a_g \leftarrow \arg\max_a \left(Q_{a,\Omega}^{DD(\vec{b},\vec{x})}(\vec{y}_g{}^a) \right)$
12 //A atualização final é feita com uma minimização.
13 $V_h^{DD(b,x)} \leftarrow \min(V_h^{DD(b,x)}, V_{h,\Omega}^{DD(b,x)})$
14 **retorna** $V_h(b_c,x_c), a_g, \vec{y}_g{}^{a_g}$

Juntando nossa nova atualização (Algoritmo 4.4) com a estrutura do RTDP (Algoritmo 4.1), temos o novo algoritmo RTSDP:

Algoritmo 4.5: RTSDP ($\mathcal{M}$, s_0, H)

Entrada: $\mathcal{M}$: HMDP, s_0: estado inicial, H: horizonte máximo do HMDP
Saída: $V = (V^0, V^1, \ldots, V^H)$: função valor não-estacionária ótima para s_0

para cada h **em** *0,1,2,..., H* **faça**
  $V^h := g(h)$ % $g(h)$ *é uma heurística admissível para h passos*
enquanto nTrials < LimiteDeTrials **ou** Erro(V,s_0) > ϵ **faça**
  $V :=$ `RealizaTrial`$(\mathcal{M}, V, s_0, H)$
retorna V.

Função `RealizaTrialRegião`$(\mathcal{M}, s, V, h) \longrightarrow V'$

Entrada: $\mathcal{M}$: HMDP, s: estado atual, V: função valor atual, h: horizonte do trial.
Saída: V': função valor após o Trial.
se $h = 0$ **então**
  retorna V % Final do Trial
$V^h, a_g :=$ `AtualizaRegião`(M, s, V^{h-1}) % Atualização a região de s
$s' :=$ `Sorteia`$(s, a_g, \mathcal{M})$ % `Sorteia` um novo estado segundo $\mathcal{T}$
return `RealizaTrialRegião`$(\mathcal{M}, s', V, h-1)$

4.2 Avaliação Empírica

Nessa seção, avaliamos o algoritmo RTSDP em três domínios: Administração de Represas (Problema 1), Controle de Estoque (Problema 2) e Controle do Trânsito (Problema 3). Esses domínios apresentam características desafiadoras distintas para planejamento: Controle de Estoque contém ações com parâmetros contínuos, Administração de Represas tem uma função de recompensa não linear e Controle do Trânsito tem dinâmica de transição não linear.

Em todos experimentos, o algoritmo RTSDP foi inicializado com uma função valor heurística admissível constante, a recompensa máxima acumulada, ou seja, $V_h(s) = h \cdot \max_{s,a} \mathcal{R}(s,a) \forall s$. Essa função foi escolhida por ser simples de calcular e fornecer pouca informação, mostrando que RTSDP pode ser melhor que as técnicas síncronas mesmo sem ter boas heurísticas.

Problema 1 - Administração de Represas Mahootchi (2009); Yeh (1985). *Neste problema, é administrar uma rede de represas de água que consiste de k represas, cada uma com seu nível de água contínuo L_i. As ações $drenar_i$ e $no\text{-}drenar_i$ determinam se a água será drenada da represa i para a próxima ou não. A quantidade de água drenada é linearmente proporcional ao nível, além disso a quantidade de energia produzida é proporcional ao produto da quantidade drenada e o nível, sendo portanto uma função quadrática do nível de água. Em cada iteração, a represa no topo da cadeia recebe uma quantidade constante de água de fontes e uma quantidade aleatória de chuva, por outro lado a represa mais baixa perde uma parte de sua água para os usos urbanos. Por exemplo, o nível de água na represa mais alta é modificado da seguinte forma:*

$$L_1' = \begin{cases} \text{se } chuva \wedge drenar_1 & : L_1 + 1000 - 0.5L_1 \\ \text{se } \neg chuva \wedge drenar_1 & : L_1 + 500 - 0.5L_1 \\ \text{se } chuva \wedge \neg drenar_1 & : L_1 + 1000 \\ \text{se } \neg chuva \wedge \neg drenar_1 & : L_1 + 500 \end{cases}$$

Uma recompensa **não linear** *é obtida por gerar energia e há uma forte penalização caso algum dos níveis de água ultrapasse valores mínimos ou máximos nominais.*

$$R_{drenar_i}(L_i) = \begin{cases} \text{se } 50 \leq L_i \leq 4500 & : 0.008 \cdot L_i^2 \\ \text{senão} & : -300 \end{cases}$$

Problema 2 - Controle de Estoque Scarf (2002). *Um problema de controle de estoque consiste em determinar quais itens do estoque devem ser repostos no final do mês e quanto deve ser pedido de cada. Neste problema, há uma variável contínua x_i para cada item i e uma única ação repor com parameters contínuos dx_i que correspondem a quantidade a ser reposta para cada item. Existe um limite L para o máximo de itens que cabem no estoque e um limite l_i para quanto de um item pode ser pedido num estágio. Os items são vendidos de acordo com uma demanda aleatória, modelada por variáveis booleanas d_i que representam se a demanda está alta (Q unidades) ou baixa (q unidades) para o item i. A recompensa é obtida por cada item vendido, mas é um custo linear para items repostos e armazenados no estoque. Por exemplo, para um item i, a recompensa é como segue:*

$$R_i(x_i, d_i, dx_i) = \begin{cases} \text{se } d_i \wedge x_i + dx_i > Q & : Q - 0.1x_i - 0.3dx_i \\ \text{se } d_i \wedge x_i + dx_i < Q & : 0.7x_i - 0.3dx_i \\ \text{se } \neg d_i \wedge x_i + dx_i > q & : q - 0.1x_i - 0.3dx_i \\ \text{se } \neg d_i \wedge x_i + dx_i < q & : 0.7x_i - 0.3dx_i \end{cases}$$

Problema 3 - Controle do Trânsito Daganzo (1994). *Esse domínio descreve o controle de uma junção de trânsito aceleração usando o modelo de célula de transmissão (*Cell Transmission Model*). Cada segmento de estrada i antes da junção é representado por uma densidade de carros k_i e uma variável booleana hi_i que indica a se a densidade de carros que está para chegar é grande ou pequena. O segmento c após a junção é representado por uma densidade de carros k_c e uma velocidade média v_c. As ações correspondem a modificar os tempos de verde dos semáforos controlando o acesso, aumentando ou diminuindo a fração do ciclo que é dada para cada segmento. É esperado que o controle de maior tempo para os segmentos que estiverem com uma densidade maior. A quantidade de carros que passam pela junção é dada por:*

$$q_{i,a_j}(k_i, v_c) = \alpha_{i,j} \cdot k_i \cdot v_c,$$

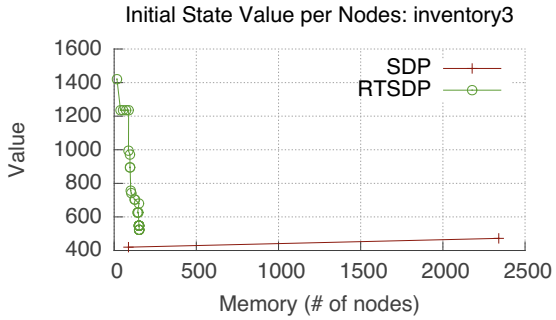
onde $\alpha_{i,j}$ é a fração do tempo do semáforo que é dada para o segmento i. pela ação a_j. A velocidade v_c no segmento após a junção é obtida como uma função não linear da densidade k_c:

$$v_c' = \begin{cases} \text{se } k_c \leq 0.25 & : 0.5 \\ \text{se } 0.25 < k_c \leq 0.5 & : 0.75 - k_c \\ \text{se } 0.5 < k_c & : k_c^2 - 2k_c + 1 \end{cases}$$

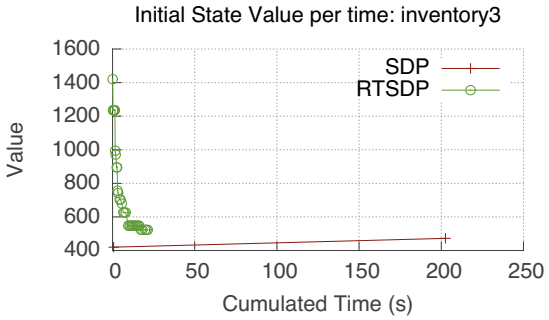
A recompensa obtida é proporcional a quantidade de carros que conseguem atravessar a junção.

A eficiência dos algoritmos para resolver os problemas é comparada em termos do tempo de solução e da quantidade de nós usados na representação XADD. Na Figura 4.1 mostramos como o valor do estado inicial no RTSDP converge par o valor ótimo $V_H(s_0)$, cada ponto do gráfico corresponde a estimative do valor do estado inicial no final de um *trial*. A curva SDP mostra o comportamento da solução síncrona e cada ponto corresponde ao valor ótimo do estado inicial para um horizonte diferente, i.e. $V_h(s_0)$ é o *h-ésimo* ponto na curva. Note que para os problemas Controle de Estoque e Controle do Trânsito, RTSDP atinge o valor ótimo muito antes que SDP e a função valor tem muito menos nós. Para o problema Administração

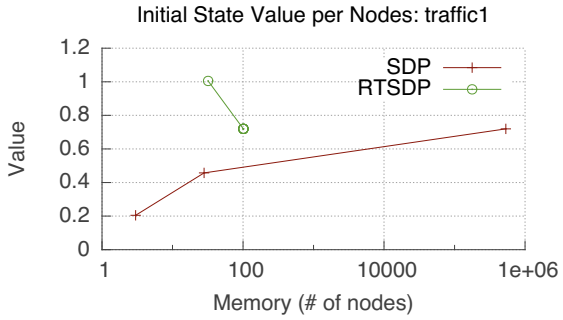
de Represas, SDP não encontrou uma solução para o horizonte desejado pois o tamanho do diagrama acabou com a memória.



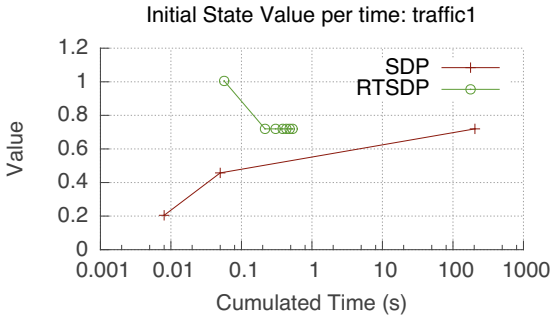
(a) Controle de Estoque*:* $V(s_0)$ *vs Espaço.*



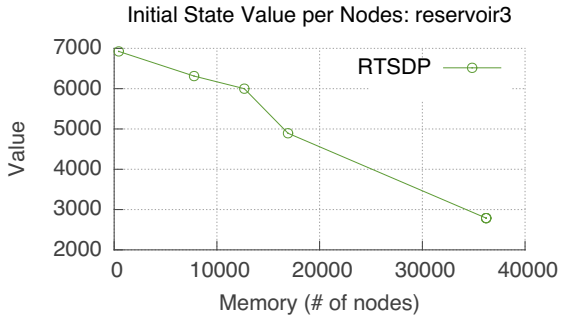
(b) Controle de Estoque*:* $V(s_0)$ *vs Tempo.*



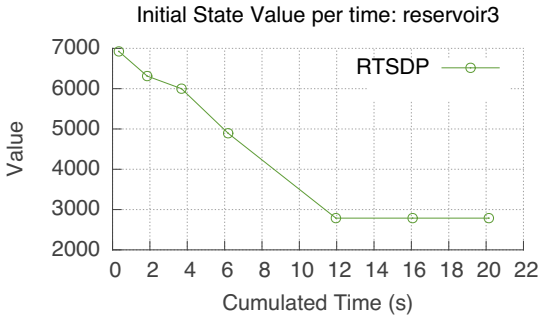
(c) Controle do Trânsito*:* $V(s_0)$ *vs Espaço(log).*



(d) Controle do Trânsito*:* $V(s_0)$ *vs Tempo(log).*



(e) Administração de Represas*:* $V(s_0)$ *vs Espaço.*

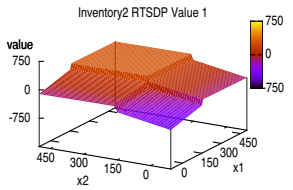


(f) Administração de Represas*:* $V(s_0)$ *vs Tempo.*

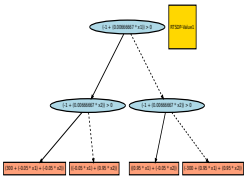
Figura 4.1: *Gráfico da convergência do RTSDP e comparação de eficiência com SDP nos três problemas:* Controle de Estoque *(cima - 3 variáveis contínuas),* Controle do Trânsito *(meio - 4 variáveis contínuas) e* Administração de Represas *(baixo - 3 variáveis contínuas). Gráficos do valor do estado inicial contra espaço (esquerda) e tempo (direita) no final de cada* trial *do RTSDP ou iteração de SDP.*

Na Figura 4.2 mostramos a função valor para diferentes horizontes geradas pelo algoritmo RTSDP *(esquerda)* e por SDP *(direita)* na resolução da segunda instância do problema Controle de Estoque, uma instância com duas variáveis contínuas e 2 parâmetros contínuos nas ações. As funções valor são apresentadas em duas formas: Gráfico 3D (Variáveis de estado vs Valor do estado) e nos diagramas XADD correspondentes. Já que todos *trials* do RTSDP se iniciam no estado inicial, a função valor com horizonte H, V_H *(baixo)* é sempre atualizada na mesma região, aquela que contém o estado inicial. As atualização adicionam novas decisões no XADD, separando essa região em partes menores, mas destas partes, apenas aquela que ainda contém o estado inicial vai continuar sendo atualizada (Veja Figura 4.2m).

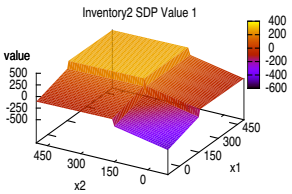
Outra observação interessante é como o tamanho do diagrama varia com o horizonte nos dois algoritmos. Para o SDP (figuras 4.2d, 4.2h, 4.2l e 4.2p), o número de nós cresce rápidamente com o horizonte (do topo para baixo). No entanto, para o RTSDP (figuras 4.2b, 4.2f, 4.2j and 4.2n) temos um padrão diferente: para os menores valores de horizonte, o número de regiões cresce devagar com o horizonte, e para o maior horizonte (horizonte da solução desejada, Figura 4.2n), a solução é bem compacta. Isso é explicado porque



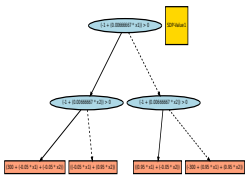
(a) *RTSDP - V^1*



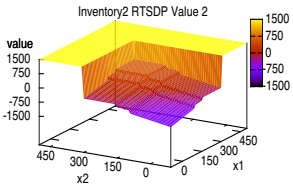
(b) *RTSDP-DD^1 - 7 Nós*



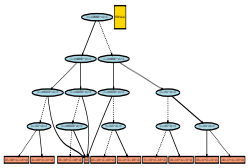
(c) *SDP - V^1*



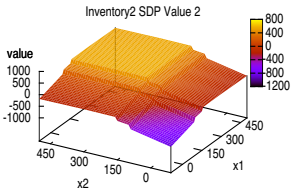
(d) *SDP-DD^1 - 7 Nós*



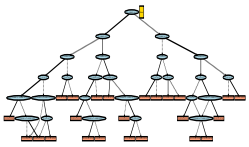
(e) *RTSDP - V^2*



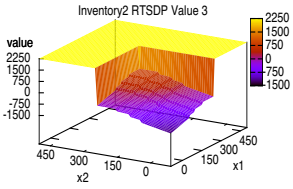
(f) *RTSDP-DD^2 - 22 Nós*



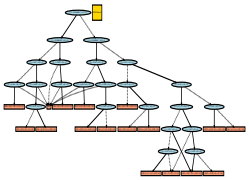
(g) *SDP - V^2*



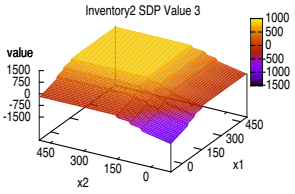
(h) *SDP-DD^2 - 50 Nós*



(i) *RTSDP - V^3*

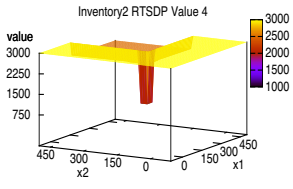


(j) *RTSDP-DD^3 - 39 Nós*

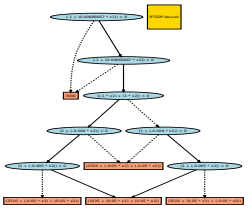


(k) *SDP - V^3*

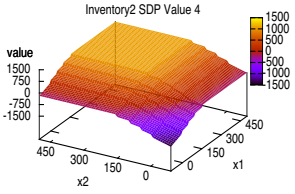
(l) *SDP-DD^3 - 142 Nós*



(m) *RTSDP - V^4*



(n) *RTSDP-DD^4 - 12 Nós*



(o) *SDP - V^4*

(p) *SDP-DD^4 - 272 Nós*

Figura 4.2: *Value functions generated by RTSDP and SDP on* Controle de Estoque, *with* $H = 4$.

a quantidade de regiões alcançáveis a partir do estado inicial em poucos passos é bem pequena assim as atualizações são concentradas e a maior parte do diagrama completo para aquele horizonte fica representado compactamente com um valor heurístico. Essa diferença é facilmente notada nos gráficos 3D, pois as regiões q não estão sendo atualizadas ficam em grandes "plateau"s com valor heurístico por exemplo(e.g. o "plateau" com valor 2250 no gráfico $RTSDP - V^3$).

Instância (#Vars. Cont.)	SDP		RTSDP	
	Tempo (s)	# Nós	Tempo (s)	# Nós
Controle de Estoque 1 (2)	0.97	43	1.03	37
Controle de Estoque 2 (4)	2.07	229	1.01	51
Controle de Estoque 3 (6)	202.5	2340	21.4	152
Administração de Represas 1 (1)	Falta de Memória		1.6	1905
Administração de Represas 2 (2)	Falta de Memória		11.7	7068
Administração de Represas 3 (3)	Falta de Memória		160.3	182000
Controle do Trânsito 1 (4)	204	533000	0.53	101
Controle do Trânsito 2 (5)	Falta de Memória		1.07	313

Tabela 4.1: *Comparação em tempo e espaço entre SDP e RTSDP nos 3 problemas.*

A Tabela 4.1 apresenta nossos resultados na comparação de desempenho entre os algoritmos RTSDP e SDP em algumas instâncias de cada domínio. Note que a diferença em eficiência entre os algoritmos cresce significantemente com o aumento do número de variáveis contínuas. Isso se deve ao fato de que conforme o espaço de estados aumenta, é mais provável que partes maiores dele não sejam relevantes para um estado inicial, assim, atualizações locais se tornam necessárias para uma solução eficiente. SDP não pode resolver a maior parte dos problemas com funções não-lineares já que não tem como evitar decisões incompatíveis que fazem com que nós impossíveis de se alcançar ocupem um grande espaço do diagrama, a atualização local nunca expande uma região inalcançável e com isso evita este problema.

4.3 Programação Dinâmica Simbólica Assíncrona e Aproximada

Nesta seção, vamos mostrar como as duas técnicas desenvolvidas neste mestrado, programação dinâmica simbólica aproximada (Algoritmo BASDP) e programação dinâmica simbólica assíncrona (Algoritmo RTSDP) podem ser combinadas para obter uma solução ainda mais eficiente por ser aproximada e assíncrona. Uma das restrições para a combinação das duas técnicas é que como o método de aproximação depende que os XADDs contenham apenas funções lineares, a técnica assíncrona e aproximada somente pode ser utilizada em domínios lineares, com ações paramétricas ou não.

A técnica de aproximação pode ser utilizada para melhorar a eficiência do algoritmo RTSDP de duas maneiras distintas: (1) usar a aproximação para criar uma heurística inicial para o RTSDP, obtendo uma solução exata mais rapidamente; ou (2) realizar aproximações durante a solução assíncrona para obter rapidamente uma solução aproximada mais compacta.

4.3.1 Usar BASDP para gerar uma heurística admissível

A técnica de aproximação com erro limitado descrita no Capítulo 3 é a transformação de um XADD linear V em um novo diagrama V' com menos nós e cujo valor difere do original por menos do que um ϵ escolhido. Dessa forma, o erro da aproximação pode ser tanto positivo ($V'(s) > V(s)$) quanto negativo ($V'(s) < V(s)$). Uma função valor heurística é admissível se para todos estados $V(s) \geq V^*(s)$, neste caso se V é admissível V' pode não ser, já que existem estados s tais que $V'(s) < V(s)$ e possivelmente $V'(s) < V^*(s)$. Por isso, para que a admissibilidade de uma função valor heurística seja mantida é necessário modificar a técnica de aproximação de modo que o erro seja sempre positivo.

Como vimos na Seção 3.1.2, a aproximação (ϕ^*, f^*) de duas folhas (ϕ_1, f_1) e (ϕ_2, f_2) é calculada com um problema bilinear nos parâmetros $\vec{c}$ e $\vec{x}$:

$$\min_{\vec{c}^*} \max_{i \in \{1,2\}} \max_{\vec{x} \in C(\phi_i)} |\underbrace{\vec{c_i} \cdot \vec{x}}_{f_i(\vec{x})} - \underbrace{\vec{c^*} \cdot \vec{x}}_{f^*(\vec{x})}| \tag{4.8}$$

Podemos garantir que a nossa aproximação $f^*(\vec{x})$ será admissível se fizermos a seguinte modificação nesse problema: removermos o módulo da diferença $\vec{c_i} \cdot \vec{x} - \vec{c^*} \cdot \vec{x}$ e adicionarmos a restrição: $\vec{c_i} \cdot \vec{x} < \vec{c^*} \cdot \vec{x}$ Dessa forma nosso problema seria:

$$\begin{array}{ll} \min_{\vec{c^*}} \max_{i \in \{1,2\}} \max_{\vec{x} \in C(\phi_i)} & \vec{c^*} \cdot \vec{x} - \vec{c_i} \cdot \vec{x} \\ s.t. & \vec{c_i} \cdot \vec{x} < \vec{c^*} \cdot \vec{x} \end{array} \tag{4.9}$$

A solução dessa formulação é feita de forma análoga à realizada com o problema original na Seção 3.1.2 assim podemos obter uma função de aproximação de erro limitado admissível.

Finalmente, podemos usar uma versão do algoritmo BASDP com essa aproximação admissível e um erro razoavelmente grande para obter uma solução grosseira de forma muito rápida e essa solução é usada como entrada para o RTSDP.

4.3.2 Aproximações na solução assíncrona

Uma maneira alternativa de unir obter um algoritmo que aproveite as vantagens das duas técnicas desenvolvidas neste trabalho é incorporar diretamente a aproximação no algoritmo assíncrono, assim no algoritmo *Bounded Approximate Real-time Symbolic Dynamic Programming - BARTSDP* é realizada a aproximação com erro limitado na função valor não estacionária V ao final de cada *trial*. A principal vantagem de uma solução é assíncrona e aproximada como essa é que ela minimiza o espaço ocupa por regiões não relevantes e pode acelerar muito a solução.

Algoritmo 4.7: BARTSDP ($\mathcal{M}$, s_0, H)

Entrada: $\mathcal{M}$: HMDP, s_0: estado inicial, H: horizonte máximo do HMDP
Saída: $V = (V^0, V^1, \ldots, V^H)$: função valor não-estacionária ótima para s_0

1 $V^h :=$ `BASDP`$(\mathcal{M}, h, 100 * \epsilon)$ % Algoritmo 3.3 com erro grande é uma heurística
2 **enquanto** nTrials < LimiteDeTrials **ou** Erro(V,s_0) > ϵ **faça**
3 $\quad V :=$ `RealizaTrialRegião`$(\mathcal{M}, V, s_0, H)$ % Algoritmo 4.6
4 $\quad V :=$ `AproximaXADD`(V, ϵ) % Algoritmo 3.1
5 **retorna** V.

4.4 Resumo

Neste capítulo propusemos nossa solução assíncrona, o algoritmo *Real-Time Symbolic Dynamic Programming - RTSDP* para a classe geral de HMDPs contendo dinamica linear e ações paramétricas como no problema Controle de Estoque, ou problemas com dinâmicas não-lneares, como o problema Controle do Trânsito. RTSDP usa atualizações por regiões para visitar apenas uma fração do espaço de estados e evitar a expansão de regiões impossíveis obtidas por combinação de decisões não-lineare incompatíveis. Dessa forma ele supera dois problemas graves no algoritmo SDP, como pode ser visto de forma muito clara nos resultados experimentais. Uma vantagem adicional do algoritmo RTSDP é poder usar uma função heurística na inicialização da função valor, o que pode torna-lo ainda melhor para problemas práticos. Além disso, vimos como a técnica de aproximação vista no Capítulo 3 é uma boa opção para gerar funções heurísticas já que pode se resolver o problema com um fator de aproximação grande e depois procurar uma solução exata com o RTSDP e também pode ser combinada com os *trials* para gerar um algoritmo assíncrono e aproximado, o BARTSDP.

Capítulo 5

Conclusão

Neste mestrado, o problema que tratado foi o de aprimorar as soluções de programação dinâmica simbólica para processos de decisão markovianos híbridos. A técnica *Symbolic Dynamic Programming - SDP* foi a base de nossas extensões, que pretenderam ampliar a quantidade e diversidade dos problemas que essa técnica pode resolver.

A primeira extensão foi baseada em algoritmos de aproximação para diagramas de decisão algébricos, que reduzem o número de nós com um erro limitado. Aplicar essa idéia para diagramas de decisão algébricas estendidos - XADDs apresenta dificuldades adicionais devido a necessidade de unir expressões lineares e não apenas números reais. A aproximação é uma ideia interessante pois a quantidade de regiões que surgem na solução exata de problemas com variáveis contínuas é muito grande, e muitas vezes existem grupos de regiões com valores muito próximos. O algoritmo proposto, o *Bounded Approximate Symbolic Dynamic Programming - BASDP)* aproveita essas semelhanças para simplificar os diagramas com um erro pequeno e assim acelerar significantemente a solução, além de permitir a solução de problemas que o SDP não conseguia por falta de espaço computacional.

Os experimentos realizados com a técnica BASDP mostraram que a aproximação da função valor pode ter um efeito significativo na redução do consume de espaço e tempo utilizados para a resolução de HMDPs, especialmente para horizontes maiores. Além disso, a capacidade de escolher o valor de erro a ser usado permite que o algoritmo seja usado em diferentes contextos, inclusive podem ser feitas soluções sucessivamente mais precisas modificando apenas o valor do erro.

Como uma segunda proposta de para melhorar a SDP, propomos nossa solução assíncrona, o algoritmo *Real-Time Symbolic Dynamic Programming - RTSDP*. Além de ser capaz de reduzir o tamanho das soluções, esse algoritmo é mais eficiente para HMDPs com funções não-lineares pois evita a expansão de nós inatingíveis. RTSDP usa atualizações por regiões para visitar apenas uma fração do espaço de estados e evitar a expansão de regiões impossíveis obtidas por combinação de decisões não-lineares incompatíveis. Dessa forma ele supera dois problemas graves no algoritmo SDP, como pode ser visto de forma muito clara nos resultados experimentais. Uma vantagem adicional do algoritmo RTSDP é poder usar uma função heurística na inicialização da função valor, o que pode torna-lo ainda melhor para problemas práticos.

Por fim, mostramos que as duas técnicas podem ser combinadas em um algoritmo assíncrono e aproximado que prevemos ser mais eficiente do que ambos. Uma tarefa futura deste trabalho será realizar experimentos comparando os algoritmos em nos mesmo e em novos domínios.

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 Michael R. Lightner e Jochen A. G. Jess, editors, *ICCAD*, páginas 188–191. IEEE Computer Society. ISBN 0-8186-4490-7. Citado na pág. 14

Barto *et al.* **(1995)** A.G. Barto, S.J. Bradtke e S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(94)00011-O. Citado na pág. 12, 43

Bellman (1957) R. E. Bellman. *Dynamic Programming*. Princeton University Press, USA. Citado na pág. 9, 11

Bonet e Geffner (2003) B. Bonet e H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. Em *ICAPS-03*, páginas 12–21. AAAI Press. Citado na pág. 12

Boutilier *et al.* **(1999)** C. Boutilier, S. Hanks e T. Dean. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94. ISSN 1076-9757. Citado na pág. 13

Boutilier *et al.* **(1995)** Craig Boutilier, Thomas Dean e Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. Em *Proceedings of the Third European Workshop on Planning*. URL planning.pdf. Citado na pág. 13

Bresina *et al.* **(2002)** John L. Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramkrishnan, David E. Smith e Richard Washington. Planning under continuous time and resource uncertainty:a challenge for ai. Em *Uncertainty in Artificial Intelligence (UAI-02)*. Citado na pág. 2, 38

Cormen *et al.* **(2001)** Thomas H. Cormen, Clifford Stein, Ronald L. Rivest e Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd ed. ISBN 0070131511. Citado na pág. 9

Daganzo (1994) Carlos F Daganzo. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B: Methodological*, 28(4): 269–287. Citado na pág. 47

Dean e Kanazawa (1990) T. Dean e K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150. ISSN 0824-7935. Citado na pág. 14

Feng *et al.* **(2004)** Zhengzhu Feng, Richard Dearden, Nicolas Meuleau e Richard Washington. Dynamic programming for structured continuous markov decision problems. Em *Proceedings of the Twentieth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, páginas 154–161, Arlington, Virginia. AUAI Press. Citado na pág. 30

Guestrin *et al.* **(2001)** Carlos Guestrin, Daphne Koller e Ronald Parr. Max-norm projections for factored MDPs. Em *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, páginas 673–680. Morgan Kaufmann. ISBN 1-55860-777-3. URL http://select.cs.cmu.edu/publications/paperdir/ijcai2001-guestrin-koller-parr.pdf. Citado na pág. 13

Guestrin *et al.* **(2011)** Carlos Guestrin, Daphne Koller, Ronald Parr e S. Venkataraman. Efficient solution algorithms for factored MDPs. *CoRR*, abs/1106.1822. Citado na pág. 13, 15

Hoey *et al.* **(1999)** Jesse Hoey, Robert St-Aubin, Alan Hu e Craig Boutilier. SPUDD: stochastic planning using decision diagrams. Em *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, UAI'99, páginas 279–288, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. ISBN 1-55860-614-9. URL http://dl.acm.org/citation.cfm?id=2073796.2073828. Citado na pág. 16

Howard (1960) R. A. Howard. *Dynamic Programming and Markov Process*. The MIT Press. ISBN 0262080095. Citado na pág. 10, 11

Li e Littman (2005) Lihong Li e Michael L. Littman. Lazy approximation for solving continuous finite-horizon MDPs. Em Manuela M. Veloso e Subbarao Kambhampati, editors, *AAAI*, páginas 1175–1180. AAAI Press / The MIT Press. ISBN 1-57735-236-X. Citado na pág. 31

Mahootchi (2009) Masoud Mahootchi. *Storage System Management Using Reinforcement Learning Techniques and Nonlinear Models*. Tese de Doutorado, University of Waterloo,Canada. Citado na pág. 46

Mausam e Kolobov (2012) Mausam e Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. Citado na pág. 2

McMahan *et al.* **(2005)** H. B. McMahan, M. Likhachev e G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. Em *22nd ICML*, páginas 569–576, New York, NY, USA. ACM. ISBN 1-59593-180-5. doi: http://doi.acm.org/10.1145/1102351.1102423. Citado na pág. 12

Nau *et al.* **(2004)** Dana Nau, Malik Ghallab e Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1558608567. Citado na pág. 1

Ono *et al.* **(2013)** Masahiro Ono, Brian C. Williams e Lars Blackmore. Probabilistic planning for continuous dynamic systems under bounded risk. *J. Artif. Intell. Res. (JAIR)*, 46:511–577. Citado na pág. 30

Puterman (1994) M. L. Puterman. *Markov decision processes*. John Wiley and Sons, New York. Citado na pág. 5

Russell e Norvig (2003) Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 ed. ISBN 0137903952. Citado na pág. 2

Sanner (2013) Scott Sanner, 2013. URL http://users.cecs.anu.edu.au/~ssanner/. [Acessado em 07/10/2013]. Citado na pág. 33

Sanner *et al.* **(2011)** Scott Sanner, Karina Valdivia Delgado e Leliane Nunes de Barros. Symbolic dynamic programming for discrete and continuous state MDPs. Em *Proceedings of the 27th Conference on Uncertainty in AI (UAI-2011)*, Barcelona. Citado na pág. 3, 21, 22, 24, 25, 29

Scarf (2002) Herbert E Scarf. Inventory theory. *Operations Research*, 50(1):186–191. Citado na pág. 40, 47

Smith e Simmons (2006) T. Smith e R. G. Simmons. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. Em *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*. Citado na pág. 12

St-Aubin *et al.* **(2000)** Robert St-Aubin, Jesse Hoey e Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. Em *NIPS-2000*, páginas 1089–1095, Denver. Citado na pág. 17, 34

Vianna *et al.* **(2013)** Luis Gustavo Rocha Vianna, Scott Sanner e Leliane Nunes de Barros. Bounded approximate symbolic dynamic programming for hybrid MDPs. Em *Proceedings of the Twenty-Ninth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-13)*, páginas 674–683, Corvallis, Oregon. AUAI Press. Citado na pág. 33

Yeh (1985) William G Yeh. Reservoir management and operations models: A state-of-the-art review. *Water Resources research*, 21,12:1797ñ1818. Citado na pág. 46

Zamani *et al.* **(2012)** Zahra Zamani, Scott Sanner e Cheng Fang. Symbolic dynamic programming for continuous state and action MDPs. Em Jörg Hoffmann e Bart Selman, editors, *AAAI*. AAAI Press. Citado na pág. 3, 21, 24, 28