UM NOVO ALGORITMO AGREGAÇÃO TEMPORAL PARA RESOLVER PROCESSOS DE DECISÃO MARKOVIANOS COM CUSTO MÉDIO

Rodrigo e Alvim Alexandre

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Produção, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Produção.

Orientador: Virgílio José Martins Ferreira Filho

Rio de Janeiro
Abril de 2024

UM NOVO ALGORITMO AGREGAÇÃO TEMPORAL PARA RESOLVER PROCESSOS DE DECISÃO MARKOVIANOS COM CUSTO MÉDIO

Rodrigo e Alvim Alexandre

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE PRODUÇÃO.

Orientador: Virgílio José Martins Ferreira Filho

Aprovada por: Prof. Edilson Fernandes de Arruda
Prof. José Leandro Félix Salles
Prof. Marcelo Dutra Fragoso
Prof. Marcos Pereira Estellita Lins
Prof. Virgílio José Martins Ferreira Filho

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2024

e Alvim Alexandre, Rodrigo

Um novo algoritmo Agregação Temporal para resolver processos de decisão Markovianos com custo médio/Rodrigo e Alvim Alexandre. – Rio de Janeiro: UFRJ/COPPE, 2024.

XIV, 84 p.: il.; 29, 7cm.

Orientador: Virgílio José Martins Ferreira Filho

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Produção, 2024.

Referências Bibliográficas: p. 74 – 82.

1. processos de decisão Markovianos. 2. agregação temporal. 3. programação dinâmica. 4. aprendizado por reforço. I. José Martins Ferreira Filho, Virgílio. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Produção. III. Título.

À minha mãe.
À memória da minha avó
materna.

Agradecimentos

Agradeço à Deus.
Agradeço à minha mãe pelo carinho, incentivo e paciência.
Agradeço aos meus familiares.
Agradeço aos meus amigos.
Agradeço ao meu orientador.
Agradeço aos membros da banca.
Agradeço ao Programa de Engenharia de Produção da COPPE/UFRJ.
Agradeço à CAPES pelo apoio financeiro.
Agradeço a todos os professores que passaram pela minha vida acadêmica

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UM NOVO ALGORITMO AGREGAÇÃO TEMPORAL PARA RESOLVER PROCESSOS DE DECISÃO MARKOVIANOS COM CUSTO MÉDIO

Rodrigo e Alvim Alexandre

Abril/2024

Orientador: Virgílio José Martins Ferreira Filho

Programa: Engenharia de Produção

Esta Tese apresenta e prova a convergência de uma nova classe de algoritmos baseada em agregação temporal para resolver problemas de decisão Markovianos com custo médio. Os novos algoritmos utilizam um novo esquema de particionamento que divide o espaço de estados em múltiplos subconjuntos formados por estados de fronteira e estados interiores. A união dos estados de fronteiras dos diversos subconjuntos da partição gera um novo subconjunto, que compõe o espaço de estados da cadeia de Markov embutida. Em uma primeira etapa, o algoritmo fixa uma política *ad-hoc* para estados interiores e itera apenas nos estados de fronteira. Em contrapartida, na segunda etapa, o algoritmo itera apenas nos estados interiores. Uma das grandes vantagens da abordagem proposta é que o novo esquema de particionamento gera subconjuntos com propriedades de comunicação específicas que permitem uma melhoria de política distribuída nos estados interiores; limitando, assim, o esforço computacional. Foram implementadas e avaliadas diferentes estratégias de melhoria de política utilizando a abordagem proposta para resolver dois problemas: um problema de produção e estoque e outro de gestão de filas. Os resultados são muito encorajadores, pois os algoritmos propostos convergiram até 60(20) vezes mais rápido do que a iteração de política (valor) nos experimentos. Além dos novos algoritmos, esta Tese apresenta um conjunto de mapas conceituais por meio do qual será possível obter um panorama do estado da arte em processos de decisão Markovianos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A NEW TIME AGGREGATION ALGORITHM TO SOLVER AVERAGE COST MARKOV DECISION PROCESSES

Rodrigo e Alvim Alexandre

April/2024

Advisor: Virgílio José Martins Ferreira Filho

Department: Production Engineering

This work introduces and proves the convergence of a new class of algorithms based on time aggregation to solve Markovian decision problems with average cost. The new algorithms use a new partitioning scheme that divides the state space into multiple subsets comprised of frontier states and interior states. The frontier states give rise to a new subset that comprises the state space of the embedded Markov chain. In the first step, the algorithm sets an *ad-hoc* policy for interior states and iterates only on frontier states. In contrast, in the second step, the algorithm iterates only on the interior states. One of the great advantages of the proposed approach is that the new partitioning scheme generates subsets with specific communication properties that allow a distributed policy improvement in the interior states, thus limiting the computational effort. Different policy improvement strategies were implemented and evaluated using the proposed approach to solve two problems: one is a production and inventory problem and the other is a queue management problem. The results are very encouraging, as the proposed algorithms converged up to 60(20) times faster than the policy(value) iteration in the experiments. In addition to the new algorithms, this work presents a set of conceptual maps that characterise the state-of-the-art in Markovian decision processes.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

A	espaço de ações viáveis, p. 7
$A(s)$	espaço de ações viáveis no estado $s \in S$, p. 7
$B(s)$	Erro de Bellman, p. 17
$Q^*(s, a)$	função Q ótima, p. 11
$Q^{\mathcal{L}}(s, a)$	função Q, p. 11
S	espaço de estados do processo de decisão Markoviano, p. 7
η^*	custo médio ótimo de longo prazo, p. 13
$\eta^{\mathcal{L}}$	custo médio de longo prazo, p. 13
λ	taxa de desconto, p. 9
$\mathbb{L}$	conjunto de políticas estacionárias Markovianas, p. 8
$\mathbb{N}$	conjunto dos números naturais, p. 7
$\mathbb{R}_+$	conjunto de números reais não negativos, p. 9
$\mathbb{R}$	conjunto dos números reais, p. 9
$\mathcal{L}$	política estacionária Markoviana, p. 8
$\mathcal{L}^*$	política ótima, p. 8
$\mathcal{V}$	conjunto de funções valor, p. 9
$v_\lambda^*(s)$	função valor de estado ótima, p. 9
$v_\lambda^{\mathcal{L}}(s)$	função valor de estado, p. 9
$f(X_t, \mathcal{L}(X_t))$	função de custo instantâneo, p. 13
$f(s, a)$	custo incorrido ao selecionar a ação a no estado s, p. 9

s estado atual do sistema, p. 7

s' estado do sistema no instante seguinte, p. 7

t instante de tempo, p. 7

Lista de Abreviaturas

PDM	Processo de Decisão Markoviano, p. 1
PDSM	Processo de Decisão Semi-Markov, p. 29

Capítulo 1

Introdução

As tomadas de decisões sequenciais sob incerteza estão presentes em uma grande variedade de campos, tais como agendamento de cirurgias (SILVA e DE SOUZA, 2020), gestão de demanda (NUNES *et al.*, 2009), controle de estoques (KARA e DOGAN, 2018, SOARES *et al.*, 2020), energia (HU *et al.*, 2020, XUE *et al.*, 2022), logística (VOELKEL *et al.*, 2020), alocação de recursos (YU *et al.*, 2021, FOROOTANI *et al.*, 2020), entre outros.

A programação dinâmica é uma ferramenta poderosa para resolver uma ampla quantidade de problemas de tomada de decisões sequencias sob incerteza (PUTERMAN, 2005, RUST, 2019) utilizando processos de decisão de Markov (PDM). Alguns exemplos de aplicações de programação dinâmica em tomadas de decisões sequenciais podem ser encontrados em (SOARES *et al.*, 2020, POWELL, 2007, BOUCHERIE e VAN DIJK, 2017, ARRUDA *et al.*, 2019a). Entretanto, apesar da sua ampla aplicabilidade, a programação dinâmica sofre com o fenômeno denominado maldição da dimensionalidade. Trata-se da explosão no número de estados à medida que o número de dimensões na variável de estado aumenta (POWELL, 2016).

Com o objetivo de mitigar este problema da maldição da dimensionalidade, diversos algoritmos têm sido desenvolvidos, tais como os algoritmos de programação dinâmica aproximada, os algoritmos de aprendizado por reforço, os algoritmos de agregação de estados e os algoritmos de agregação temporal. Os algoritmos de programação dinâmica aproximada utilizam aproximações da função valor (e.g., HU *et al.*, 2020, JIANG e POWELL, 2015) ou modelos aproximados (e.g., ARRUDA *et al.*, 2013) para resolver um problema de decisão sequencial sob incerteza. A desvantagem destes algoritmos é que, de maneira geral, não encontram uma solução exata, mas sim, aproximada. Em contrapartida, os algoritmos de aprendizado por reforço (e.g., WATKINS, 1989, KAMANCHI *et al.*, 2019, JOHN *et al.*, 2020) simulam a evolução do processo de Markov decorrente das ações tomadas no percurso a fim de estimar a função valor e encontrar uma regra de decisão ótima. Contudo, para estes algoritmos alcançarem uma boa solução, é preciso executar uma grande

quantidade de simulações, o que torna o algoritmo menos eficiente computacionalmente em relação aos demais. Já a agregação de estados (e.g., XUE *et al.*, 2022), por outro lado, divide, arbitrariamente, o espaço de estados em subconjuntos. Os estados do mesmo subconjunto são combinados, formando estados agregados. Estes estados agregados compõem o espaço de estados de um novo modelo com menor dimensionalidade. Contudo, este novo modelo não preserva a propriedade de Markov (CAO *et al.*, 2002) e, portanto, é uma aproximação do modelo original. Deste modo, ao ser resolvido, a solução do modelo de menor dimensão pode ser utilizada como uma aproximação para a solução do modelo original de maior dimensionalidade.

Introduzida em (CAO *et al.*, 2002), a classe de algoritmos baseados em agregação temporal particiona o espaço de estado em dois subconjuntos: um pequeno subconjunto de estados controláveis ou que são mais interessantes sob o ponto vista do controle e outro subconjunto muito maior de estados não controláveis ou de pouco interesse no problema (SUN *et al.*, 2007, ARRUDA *et al.*, 2017). As trajetórias que saem e retornam ao subconjunto de interesse geram uma cadeia de Markov embutida (*embedded Markov chain*), veja por exemplo CAO *et al.* (2002). Desse modo, diferentemente dos algoritmos de agregação de estados, os algoritmos baseados em agregação temporal reduzem o espaço de estados mantendo a propriedade markoviana (CAO *et al.*, 2002). A agregação temporal atinge a solução ótima quando o subconjunto de maior cardinalidade é composto por estados não controláveis. Caso contrário, a abordagem alcançará uma solução sub-ótima, pois otimiza dentro de uma região arbitrária do espaço de estados enquanto aplica uma política de controle *ad hoc* fixa nos estados restantes.

Para garantir que uma política ótima sempre possa ser alcançada por meio da agregação temporal, ARRUDA e FRAGOSO (2015) propuseram uma abordagem em duas fases. A primeira fase busca uma política sub-ótima otimizando dentro de um pequeno subconjunto F do espaço de estados S, enquanto seleciona uma política *ad hoc* para todos os estados restantes em F^c. A segunda fase usa as propriedades da cadeia de Markov embutida para derivar a função valor em todos os estados e encontrar uma política de controle aprimorada em F^c, que se tornará a política *ad hoc* fixa na próxima etapa. As duas fases são alternadas até a convergência. A agregação temporal em duas fases, no entanto, requer a avaliação de trajetórias saindo e voltando para o domínio embutido. Isso se torna computacionalmente intensivo à medida que a cardinalidade do espaço de estados aumenta. Para aliviar os cálculos, ARRUDA *et al.* (2019b) introduziram um esquema de particionamento que divide o espaço de estados em um número finito, mas arbitrário de subconjuntos. Contudo, a abordagem deles não leva em consideração o controle e, portanto, não é capaz de encontrar uma política ótima, podendo ser utilizada apenas para encontrar a distribuição estacionária de uma cadeia de Markov ou a função valor de uma

política qualquer, i.e., para avaliar politicas de controle arbitrárias e pré-definidas.

Em face do exposto, o objetivo principal desta Tese de Doutorado é propor um novo algoritmo de agregação temporal que faz uso de um novo esquema de particionamento que divide o espaço de estados em um número finito, mas arbitrário de subconjuntos. Diferentemente do esquema de particionamento proposto por ARRUDA *et al.* (2019b), o esquema de particionamento proposto nesta Tese permite embutir um processo de decisão de Markov original em um processo de decisão semi-Markov equivalente por meio de uma simples análise de absorção, aplicada, individualmente, dentro de cada subconjunto da partição. Isso equivale a avaliar trajetórias *de saída* dentro de cada subconjunto, o que implica um esforço computacional que depende da cardinalidade do subconjunto em análise. Desse modo, é possível controlar e limitar o esforço computacional geral, fornecendo maior flexibilidade e uma clara melhoria em relação à agregação temporal em duas fases (ARRUDA e FRAGOSO, 2015), visto que este último algoritmo requer a avaliação de trajetórias *interiores* que evoluem dentro de um subconjunto muito grande F^c, cuja cardinalidade é fixa e análoga à cardinalidade de todo o espaço do estados.

Utilizando o novo esquema de particionamento, esta Tese propõe um conjunto de novos algoritmos de agregação temporal, cuja convergência para a solução ótima é provada e validada. Esta Tese também demonstra como os novos algoritmos podem ser combinados às abordagens de programação dinâmica aproximada ou aprendizado por reforço, gerando um novo algoritmo aproximado para resolver problemas de alta dimensionalidade com um menor tempo computacional. Os novos algoritmos desenvolvidos foram utilizados para resolver dois problemas: um de produção e gestão de estoque e outro de gestão de filas. Os resultados mostram que os algoritmos propostos tendem a alcançar a vizinhança da solução ótima significativamente mais rápido do que os algoritmos clássicos de programação dinâmica tais como Iteração de Política, o Iteração de Valor e a abordagem original da Agregação Temporal Duas Fases de (ARRUDA e FRAGOSO, 2015). Os resultados ilustram o potencial dos novos algoritmos para lidar com problemas PDM de custo médio com maior dimensionalidade. Estes utilizam um esquema de otimização distribuída pelos subconjuntos que tira proveito de um método de particionamento cuidadosamente projetado, que dá origem a propriedades de comunicação favoráveis. Tanto os algoritmos como o método de particionamento de PDMs são inovações e contribuições do presente trabalho, conforme demonstrado na revisão sistemática da literatura exposta na Seção 2.10.

Além do objetivo principal, é objetivo secundário desta Tese de Doutorado construir um Mapa Conceitual da área de processos de decisão de Markov, de modo que o leitor tenha clareza do estado da arte, bem como da inovação introduzida por este trabalho.

Por fim, esta Tese está organizada da seguinte maneira: O Capítulo 2 expõe uma revisão da literatura acerca de processos de decisão Markovianos e os principais algoritmos utilizados para resolvê-los, organizando-os em mapas conceituais ao final de cada seção. Este capítulo também explicita as motivações desta Tese de Doutorado, seu ineditismo e contribuições. Vale destacar que o Capítulo 2 contém duas revisões sistemáticas da literatura. Na sua introdução, é apresentada uma revisão sistemática referente à utilização de mapas conceituais. Já na seção 2.10, é apresentada uma revisão sistemática a cerca de algoritmos de agregação temporal. Cada revisão da literatura comprova o ineditismo dos objetivos principal e secundário desta Tese: a primeira comprova o pioneirismo do objetivo secundário, enquanto a segunda comprova a originalidade do objetivo principal. Em contrapartida, o Capítulo 3 explica o novo esquema de particionamento e um novo conjunto de algoritmos exatos desenvolvidos, ao passo que os capítulos 4 e 5 descrevem os experimentos numéricos e os resultados obtidos. O Capítulo 6 propõe um novo algoritmo aproximado para resolver problemas de alta dimensionalidade. Por último, o Capítulo 7 conclui a Tese e indica possíveis direções a serem seguidas em trabalhos futuros.

Capítulo 2

Revisão da Literatura com breve descrição dos PDMs e algoritmos

O presente capítulo apresenta uma revisão do estado da arte em processos de decisão markovianos. Inicialmente, a seção 2.1 trata do arcabouço teórico com base nos trabalhos de PUTERMAN (2005) e SUTTON e BARTO (2018). Já nas seções seguintes, será apresentada uma revisão dos principais algoritmos existentes para resolver os problemas de decisão de Markov. Importa destacar que esta revisão da literatura acerca dos algoritmos foi realizada a partir de buscas nas bases *Scopus* e *Web of Science*. Inicialmente, estas buscas foram conduzidas a partir da análise das referências em SUTTON e BARTO (2018) e nos artigos de programação dinâmica, especialmente, os artigos sobre agregação temporal. Nesta etapa, foram identificados os artigos mais clássicos. Em seguida, novas buscas foram realizadas a fim de identificar artigos mais recentes que tenham citado os artigos clássicos identificados na etapa anterior. Vale destacar que estas duas etapas (análise das referências e análise das citações) ocorreram de maneira iterativa.

Ao final de cada seção, os algoritmos mencionados serão dispostos em mapas conceituais, elaborados utilizando o programa CMap Tools (CAÑAS *et al.*, 2004). Um mapa conceitual (NOVAK e CAÑAS, 2006) consiste, essencialmente, em um gráfico com diversos conceitos envolvidos por círculos dispostos hierarquicamente e conectados por arcos que contém palavras que indicam a relação entre os conceitos (EPPLER, 2006, NOVAK e CAÑAS, 2008, BALIGA *et al.*, 2021). Deste modo, os mapas conceituais são ferramentas importantes para organizar e apresentar ideias e conhecimento sobre um assunto específico (NOVAK e CAÑAS, 2008, EACHEMPATI *et al.*, 2020, ISLAM *et al.*, 2020), podendo ser utilizados por pesquisadores para compreender uma determinada área de estudo (CONCEIÇÃO *et al.*, 2017).

Neste trabalho, os mapas conceituais proporcionaram uma visão sistêmica dos campos de conhecimento aqui estudados. Mais especificamente, os mapas conceituais permitiram uma organização estruturada da diversidade de algoritmos atual-

mente existentes para resolver problemas de decisão de Markov. Destarte, um dos diferenciais desta Tese de Doutorado é que, a partir do conjunto de todos os mapas conceituais apresentados nesta seção, será possível obter um panorama do estado da arte em processos de decisão Markovianos. Adicionalmente, será possível obter uma perspectiva do contexto em que a Tese de Doutorado está inserida, bem como as suas motivações, seus objetivos, sua relevância e suas contribuições.

Destaca-se que no dia 28/10/2023 foram realizadas buscas nas bases *Scopus* e *Web of Science* (WoS) a fim de verificar se algum mapa conceitual sobre processos de decisão Markovianos já havia sido elaborado. As palavras-chave relacionadas foram definidas após a leitura dos livros de PUTERMAN (2005), SUTTON e BARTO (2018), POWELL (2007) e BERTSEKAS *et al.* (2000), bem como de artigos clássicos e recentes. A estrutura desta revisão da literatura, bem como a quantidade de artigos encontrados, podem ser visualizados na Tabela 2.1.

Tabela 2.1: Revisão de Literatura sobre Mapa Conceitual

Base	Consulta	Filtros	Total de Artigos
Scopus	TITLE-ABS-KEY(("markov decision process*"OR "reinforcement learning"OR "neuro-dynamic programming"OR "dynamic programming") AND ("concept map"OR "mind map"OR "cognitive map"OR "thinking map"OR "knowledge map"))	Nenhum	101
WoS	TS=(("markov decision process*"OR "reinforcement learning"OR "neuro-dynamic programming"OR "dynamic programming") AND ("concept map"OR "mind map"OR "cognitive map"OR "thinking map"OR "knowledge map"))	Nenhum	66

Os resultados encontrados foram exportados para arquivos em formato BibTex que foram, em seguida, importados para o R (R CORE TEAM, 2020). Utilizando a função *mergeDbSources* do pacote Bibliometrix (ARIA e CUCCURULLO, 2017) os resultados das bases foram agrupados e os artigos repetidos foram eliminados. Desse modo, desconsiderando artigos repetidos, foram encontrados ao todo cento e vinte e sete artigos. Dentre estes artigos, o que mais se aproxima do objetivo secundário desta Tese é o trabalho de POMBO *et al.* (2017). A partir de uma revisão sistemática da literatura, que incluiu artigos publicados entre 2005 e 2015, POMBO *et al.* (2017) elaboraram um mapa mental em que foram elencados uma série de métodos de aprendizado de máquina que já foram utilizados para identificar a síndrome da apneia do sono. Embora menos relacionado a esta Tese, vale mencionar também o trabalho de NI e TANG (2023). Os autores realizaram uma análise bibliométrica da literatura acerca do problema de roteamento de veículos a partir de artigos encontrados na base *Web Of Science* com data de publicação entre 1959 e 2022.

Portanto, após uma análise dos resumos, dos títulos e das palavras-chave de todos os cento e vinte e sete artigos artigos, bem como de uma leitura dinâmica de alguns,

não foi identificado nenhum trabalho que tenha elaborado um mapa conceitual com os algoritmos das áreas de conhecimento aqui estudadas. Vale destacar que na seção 2.10 é apresentada uma nova revisão sistemática acerca de algoritmos de agregação temporal que comprova o ineditismo do objetivo principal, isto é, a originalidade do método de particionamento e dos algoritmos propostos neste trabalho.

Por último, importa mencionar que as subseções seguintes estão estruturadas da seguinte maneira:

- Primeiro, foi definido o Problema de Decisão Markoviano e detalhado 2 dos principais algoritmos no contexto da programação dinâmica clássica, baseados em modelos, propostos para solução desse problema, com forte conexão com essa tese (Algoritmo de Iteração de Política e Algoritmo de Iteração de Valor).

- Em seguida foi estruturado, através de um mapa conceitual, as principais classes de algoritmos propostos para solução de problemas de decisão Markoviano, incluindo variações dos algoritmos de Iteração de Política e de Iteração de Valor (classificados na classe dos algorítimos de programação dinâmica clássica).

- Nas subseções subsequentes foram detalhadas cada classe dos algoritmos.

2.1 Processos de Decisão Markovianos

Os processos de decisão markovianos (PDM) são caracterizados por tomadas de decisões sequenciais ao longo de um horizonte de tempo pré-determinado, que pode ser infinito. Seja S o conjunto de estados possíveis de serem ocupados por um sistema. Seja $A(s)$ o conjunto de ações possíveis de serem tomadas no estado $s \in S$ e considere $A := \{A(s), i \in S\}$. Assim, em um PDM, um tomador de decisão (ou agente), em cada instante de tempo $t \geqslant 0$, $t \in \mathbb{N}$, observa o estado ocupado pelo sistema ($X_t = s$, $s \in S$). De posse dessa informação, o tomador de decisão escolhe uma ação $a \in A(s)$. A escolha desta ação implicará na incidência imediata de uma recompensa ou de um custo e o sistema evoluirá no instante de tempo seguinte para um novo estado ($X_{t+1} = s'$, $s' \in S$), de acordo com uma distribuição de probabilidade $p(s'|s, a)$, que é funcção do estado atual do sistema e da ação escolhida no instante da tomada de decisão (PUTERMAN, 2005, SUTTON e BARTO, 2018). A Figura 2.1 ilustra um processo de decisão de Markov em que os arcos e setas representam todas as transições que podem ocorrer com probabilidade positiva, de acordo com alguma ação tomada.

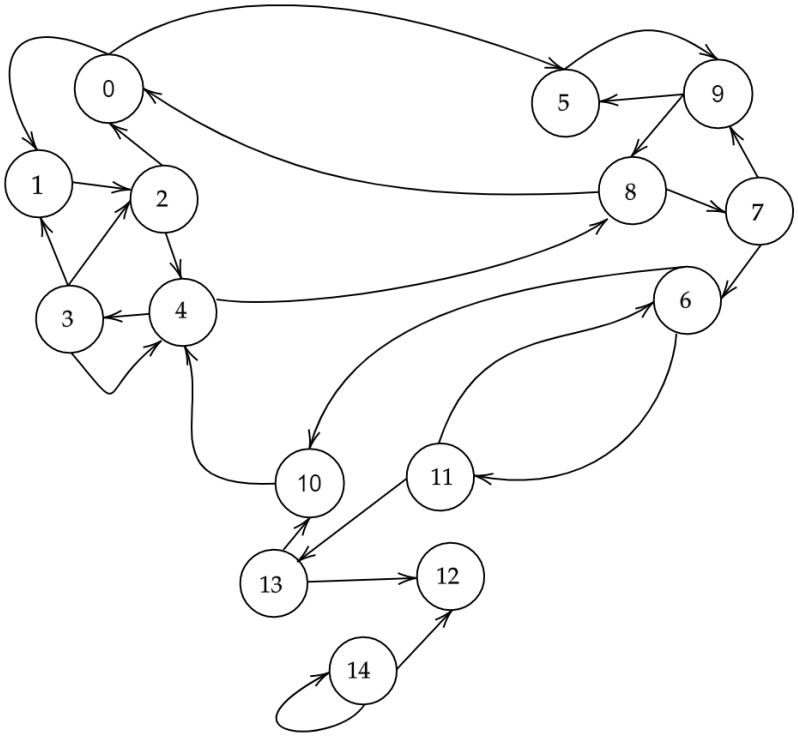

Figura 2.1: **Um exemplo de processo de decisão Markoviano.**

Seja $\mathcal{L} : S \to A$ uma política estacionária markoviana válida para todo o espaço de estado S. Portanto, $\mathcal{L}$ é uma função que especifica uma mesma ação $a \in A(s)$ a ser tomada sempre que o sistema visite o estado $s \in S$, independente do trajeto percorrido pelo sistema até chegar ao estado s. Sendo $\mathbb{L}$ o conjunto de todas as políticas estacionárias markovianas, um problema de decisão de Markov consiste em escolher, antes do primeiro instante de tomada de decisão, uma política $\mathcal{L}^* \in \mathbb{L}$ que maximiza uma determinada função da sequência de recompensas aleatórias a serem aferidas ou minimiza uma determinada função da sequência de custos aleatórios a serem incorridos (PUTERMAN, 2005).

Seja $v_\lambda^{\mathcal{L}}(s) \in \mathcal{V}$, tal que $\mathcal{V} : S \to \mathbb{R}$ seja o espaço de mapeamanetos de S para o conjunto de números reais, e o parâmetro $0 < \lambda < 1$ seja um fator de desconto que será especificao no decorrer desta sessão. Seja também $f : S \times A \to \mathbb{R}_+$ a função de custo instantâneo, com $f(s, a)$ denotando o custo incorrido ao selecionar a ação $a \in A(s)$ no estado $s \in S$, sendo $\mathbb{R}_+$ o conjunto de números reais não negativos. O custo descontado esperado, ao começar no estado $s \in S$ e seguir uma política $\mathcal{L} \in \mathbb{L}$,

é definido como:

$$v_\lambda^{\mathcal{L}}(s) = \mathbf{E}_{\mathcal{L}} \left[\sum_{t=0}^{\infty} \lambda^t f(X_t, \mathcal{L}(X_t)) \middle| S_t = s \right], \ \forall s \in S, \tag{2.1}$$

em que $\mathbf{E}_{\mathcal{L}}[\cdot]$ denota o valor esperado de uma variável aleatória dado que o tomador de decisão segue a política $\mathcal{L} \in \mathbb{L}$ e o parâmetro λ é uma taxa de desconto, tal que $0 \leq \lambda \leq 1$. A equação (2.1) pode ser reescrita de forma recursiva da seguinte maneira:

$$v_\lambda^{\mathcal{L}}(s) = f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^{\mathcal{L}}(s'), \ \forall s \in S. \tag{2.2}$$

A equação (2.2) é denominada Equação de Bellman e relaciona o valor do estado atual com os valores dos estados sucessores. Resolver o problema de decisão markoviano consiste em encontrar uma política ótima, isto é, encontrar uma política $\mathcal{L}^* \in \mathbb{L}$ tal que:

$$v_\lambda^{\mathcal{L}^*}(s) \geq v_\lambda^{\mathcal{L}}(s), \ \forall s \in S, \ \mathcal{L} \in \mathbb{L}. \tag{2.3}$$

Em modelos descontados, o valor do processo de decisão de Markov é definido como:

$$v_\lambda^*(s) \equiv \max_{\mathcal{L} \in \mathbb{L}} v_\lambda^{\mathcal{L}}, \tag{2.4}$$

em que $v_\lambda^*(s)$ é denominada função-valor ótima. É possível reescrever a equação (2.4) como:

$$v_\lambda^*(s) = \max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^{\mathcal{L}}(s') \right\}. \tag{2.5}$$

Desse modo, para uma política ótima $\mathcal{L}^* \in \mathbb{L}$:

$$v_\lambda^{\mathcal{L}^*}(s) = v_\lambda^*(s), \ \forall s \in S. \tag{2.6}$$

Cada equação no sistema de equações (2.5) é uma equação de otimalidade de Bellman. A solução do sistema de equações (2.5) poder ser encontrada por um conjunto de métodos que compreendem uma classe de algoritmos aqui denominada Programação Dinâmica Clássica a ser mais detalhada na Seção 2.3. Estes métodos são considerados como baseados em modelo (*model-based*) visto que exigem que a dinâmica do processo de decisão markoviano, estabelecida por $p(s'|s,a)$, seja conhecida (SUTTON e BARTO, 2018).

Dentre os algoritmos de programação dinâmica clássica, destaca-se a Iteração de Política (PUTERMAN, 2005, SUTTON e BARTO, 2018), detalhada no Algoritmo 1.

A Iteração de Política (*Policy Iteration*) é um algoritmo iterativo composto por duas etapas que se alternam até a convergência. Uma das etapas é denominada Avaliação de Política (*Policy Evaluation*) e corresponde ao passo 4 do Algoritmo 1. Esta etapa consiste em um processo iterativo para resolver a equação (2.2) e encontrar $v_\lambda^{\mathcal{L}}(s)$ para todo $s \in S$. A Avaliação de Política busca, portanto, calcular o retorno descontado esperado de uma política $\mathcal{L} \in \mathbb{L}$ para todo estado $s \in S$. A segunda etapa é denominada Melhoria de Política (*Policy Improvement*) e corresponde ao passo 9 do Algoritmo 1. Esta etapa consiste em utilizar a equação (2.5) para encontrar uma política $\mathcal{L}' \in \mathbb{L}$ que seja melhor do que a política atual $\mathcal{L}$. Conforme explicitado no passo 11 do Algoritmo 1, a Iteração de Política converge quando não é mais possível encontrar uma nova política $\mathcal{L}'$ que seja melhor do que a política atual $\mathcal{L}$.

Algoritmo 1 Iteração de Política

1: Escolha uma política inicial arbitrária $\mathcal{L} = d_0,\ d_0 \in \mathbb{L}$, um valor inicial arbitrário $v_\lambda^0 \in \mathcal{V}$ e uma tolerância ϵ
2: $n \leftarrow 0$
3: $k \leftarrow 0$
4: Para todo $s \in S$, calcule:

$$v_\lambda^{k+1}(s) = f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^k(s') \tag{2.7}$$

5: $er \leftarrow \max_s \{v_\lambda^{k+1}(s) - v_\lambda^k(s)\}$
6: **if** $|er| \geq \epsilon$ **then**
7: $\quad k \leftarrow k+1$ e retornar para o passo 4
8: **else**
9: $\quad$ Escolha d_{n+1} que satisfaça:

$$d_{n+1} \in \arg\max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^k(s') \right\} \tag{2.8}$$

10: $\quad$ Se possível, escolha $d_{n+1} = d_n$
11: $\quad$ **if** $d_{n+1} = d_n$ **then**
12: $\quad\quad$ Pare e faça $\mathcal{L}^* = d_n$
13: $\quad$ **else**
14: $\quad\quad$ $n \leftarrow n+1$ e retorne ao passo 3
15: $\quad$ **end if**
16: **end if**

Outro algoritmo de destaque em Programação Dinâmica Clássica é o algoritmo de Iteração de Valor (PUTERMAN, 2005, SUTTON e BARTO, 2018), especificado no Algoritmo 2. Primeiramente, escolhe-se um valor inicial arbitrário $v^0 \in \mathcal{V}$ e uma tolerância ϵ. Em seguida, no passo 3 do Algoritmo 2, a equação de otimalidade de Bellman é utilizada para encontrar uma nova função valor v_λ^{n+1} melhor do que a função valor atual v_λ^n. Este passo é repetido até a convergência, que ocorre quando a diferença do custo descontado esperado estimado entre duas iterações consecutivas é inferior a um limite pré-determinado ϵ, conforme explicitado no passo 5 do Algoritmo 2. Neste caso, é possível encontrar a política ótima que soluciona a equação do passo 8 do Algoritmo 2, dentro da tolerância especificada.

Comparativamente, cada iteração do algoritmo Iteração de Valor demanda menos esforço computacional que uma iteração do algoritmo Iteração de Política, visto que a primeira contém apenas uma única etapa. Contudo, a Iteração de Política utiliza

Algoritmo 2 Iteração de Valor

1: Escolha um valor inicial arbitrário $v^0 \in \mathcal{V}$ e uma tolerância ϵ.
2: $n \leftarrow 0$.
3: Para todo $s \in S$, calcule:

$$v_\lambda^{n+1}(s) = \max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^n(s') \right\} \tag{2.9}$$

4: $er \leftarrow \max_s \{v_\lambda^n(s) - v_\lambda^{n-1}(s)\}$
5: **if** $|er| \geq \epsilon$ **then**
6: $\quad n \leftarrow n+1$ e retornar para o passo 3
7: **else**
8: $\quad$ Para todo $s \in S$, escolha:

$$\mathcal{L}^*(s) = \arg\max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^n(s') \right\} \tag{2.10}$$

9: **end if**

menos iterações que a Iteração de Valor, uma vez que sua convergência é quadrática e a convergência do algoritmo de iteração de valor é linear (PUTERMAN, 2005). Adicionalmente, a Iteração de Política gera um valor exato para uma política subótima em qualquer iteração enquanto a Iteração de Valor produz apenas um valor aproximado para as políticas, gerando um valor exato apenas no final, isto é, quando o algoritmo converge e encontra a política ótima.

Embora clássicos, os algoritmos Iteração de Política e Iteração de Valor não são aplicáveis em situações em que $p(s'|s,a)$, seja desconhecido. Nestes casos, é possível utilizar algoritmos considerados livres de modelo (*model-free*) que utilizam Monte Carlo e Diferenças Temporais (SUTTON e BARTO, 2018). Estes serão mais explorados nas seções 2.5 e 2.6, respectivamente.

Dentre os algoritmos de Diferença Temporal, destaca-se o Aprendizagem Q (*Q-learning*), que utiliza uma função Q definida como (WATKINS, 1989):

$$Q^{\mathcal{L}}(s,a) = \mathbf{E}_{\mathcal{L}} \left[\sum_{t=0}^{\infty} \lambda^t f(X_t, \mathcal{L}(X_t)) \middle| S_t = s, A_t = a \right], \ \forall s \in S, \ a \in A(s). \tag{2.11}$$

Assim, a função $Q^{\mathcal{L}}(s,a)$ corresponde ao valor de tomar uma ação $a \in A(s)$ no estado $s \in S$ e seguir uma política $\mathcal{L} \in \mathbb{L}$ em diante. A equação (2.11) pode ser ainda reescrita da seguinte maneira:

$$Q^{\mathcal{L}}(s,a) = f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) v_\lambda^{\mathcal{L}}(s'). \tag{2.12}$$

Também é definida uma função Q-ótima como:

$$Q^*(s,a) \equiv \max_{\mathcal{L} \in \mathbb{L}} Q^{\mathcal{L}}(s,a). \tag{2.13}$$

Portanto, a função Q-ótima corresponde ao retorno descontado esperado ao tomar uma ação $a \in A(s)$ no estado $s \in S$ e seguir uma política ótima daí em diante. Assim, a função valor e a função Q-ótima estão relacionadas da seguinte maneira:

$$v_\lambda^{\mathcal{L}^*}(s) = \max_{a \in A(s)} Q^{\mathcal{L}^*}(s,a). \tag{2.14}$$

Em contrapartida, a equação de otimalidade de Bellman para a função Q pode ser escrita como:

$$Q^*(s,a) = f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) \max_{a'} Q(s',a'). \tag{2.15}$$

O Algoritmo 3 detalha os passos utilizados pela Aprendizagem Q para resolver um processo de decisão Markoviano. Conforme, pode ser observado no Algoritmo 3, a Aprendizagem Q utiliza os conceitos de episódio e estado terminal. Um episódio é uma execução do processo de decisão de Markov e o estado terminal é um estado que, quando é alcançado pelo sistema, o processo de decisão de Markov se reinicia. Desse modo, em cada passo de um episódio, o tomador de decisão observa o estado atual do sistema e escolhe uma ação de acordo com uma política estocástica, por exemplo, uma política ε-gulosa. Em uma política ε-gulosa, na maior parte do tempo, a ação que possui o maior valor estimado é escolhida, mas com probabilidade ε uma ação não ótima é escolhida aleatoriamente. A partir da execução da ação escolhida, o agente observa o novo estado do sistema e recebe, imediatamente, uma recompensa. Em seguida, o valor de Q é atualizado a partir de uma taxa de aprendizagem α, de acordo com a equação (2.16) do Algoritmo 3.

Algoritmo 3 Aprendizagem Q

1: Escolha um valor inicial arbitrário $Q(s,a)$ para todo $s \in S$ e $a \in A(s)$ com exceção do estado terminal, em que $Q(terminal, a) = 0$ para todo $a \in A(terminal)$
2: Escolha $0 < \alpha \leq 1$ e um valor $\varepsilon > 0$
3: **for each** Episódio **do**
4: Escolha um estado $s \in S$
5: **while** s não é um estado terminal **do**
6: Escolha uma ação $a \in A(s)$ utilizando uma política ε-gulosa
7: Execute a ação a escolhida na etapa anterior e observe r e s' resultante
8: Calcule:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha \left[r_n + \lambda \max_{a'} Q(s',a') \right] \tag{2.16}$$

9: $s \leftarrow s'$
10: **end while**
11: **end for**

Até agora, para resolvermos um problema de decisão de Markov e encontrar uma política ótima, utilizamos o custo descontado esperado. Contudo, conforme destacado em (PUTERMAN, 2005), em problemas com taxa de desconto muito

próximas de 1, como é o caso de situações em que a tomada de decisão é frequente, ao invés de utilizar o custo descontado, o decisor pode optar por comparar as políticas utilizando o custo médio de longo prazo dado por:

$$\eta^{\mathcal{L}} = \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N} f(X_t, \mathcal{L}(X_t)), \tag{2.17}$$

em que $f(X_t, \mathcal{L}(X_t))$ é a função de custo instantâneo, denotando o custo incorrido ao seguir a política $\mathcal{L}$ quando o processo se encontra em um estado X_t no instante de tempo t. Neste caso, o tomador de decisão busca uma política ótima estacionária $\mathcal{L}^* \in \mathbb{L}$ tal que:

$$\eta^* = \eta^{\mathcal{L}^*} \leq \eta^{\mathcal{L}}, \ \forall \mathcal{L} \in \mathbb{L}. \tag{2.18}$$

Vale destacar que, a exemplo de CAO *et al.* (2002), nesta Tese, assumiremos que a cadeia de Markov é ergódica sob qualquer política viável $\mathcal{L} \in \mathbb{L}$. Em uma cadeia ergódica, é possível ir de qualquer estado para qualquer estado em tempo finito e o custo médio de longo prazo independe do estado inicial (BRÉMAUD, 1999).

A Eq. (2.18) pode ser resolvida por meio do algoritmo Iteração de Valor Relativo (IVR) clássico (e.g., PUTERMAN, 2005). O algoritmo de Iteração de Valor Relativo começa com uma solução inicial arbitrária $v^0 \in \mathcal{V}$. A cada iteração, o sistema de equações (2.19) e (2.20) do Algoritmo 4 é resolvido e a função valor v é atualizada. O algoritmo converge quando a diferença entre os valores máximo e mínimo de v é inferior a um limite pré-determinado ϵ, conforme explicitado no passo 6 do Algoritmo 4.

Algoritmo 4 Iteração de Valor Relativo

1: Escolha um valor inicial arbitrário $v^0 \in \mathcal{V}$, um estado qualquer de referência $s^* \in S$ e uma tolerância ϵ.
2: $n \leftarrow 0$.
3: Para todo $s \in S$, calcule:

$$Tv^n(s) = \max_{a\in A(s)} \left\{ f(s,a) + \sum_{s'\in S} p(s'|s,a) v^n(s') \right\} \tag{2.19}$$

4: Para todo $s \in S$, calcule:

$$v^{n+1}(s) = Tv^n(s) - Tv^n(s^*) \tag{2.20}$$

5: $er \leftarrow \max_s v^n(s) - \min_s v^n(s)$
6: **if** $|er| \geq \epsilon$ **then**
7: $\quad n \leftarrow n+1$ e retorne ao passo 3
8: **else**
9: $\quad v^* \leftarrow v^n$
10: $\quad \eta^* \leftarrow Tv^*(s^*)$
11: $\quad$ Para todo $s \in S$, escolha:

$$\mathcal{L}^*(s) = \arg\max_{a\in A(s)} \{Tv^*(s)\} \tag{2.21}$$

12: **end if**

Além dos algoritmos Iteração de Política, Iteração de Valor, Aprendizagem Q e Iteração de Valor Relativo, outros algoritmos podem ser utilizados para resolver PDMs e serão explicitados nas próximas seções.

2.2 Algoritmos para resolver PDM

Neste trabalho, os principais algoritmos que, atualmente, existem para resolver os problemas de decisão markovianos foram organizados em classes, que estão expostas na figura 2.2.

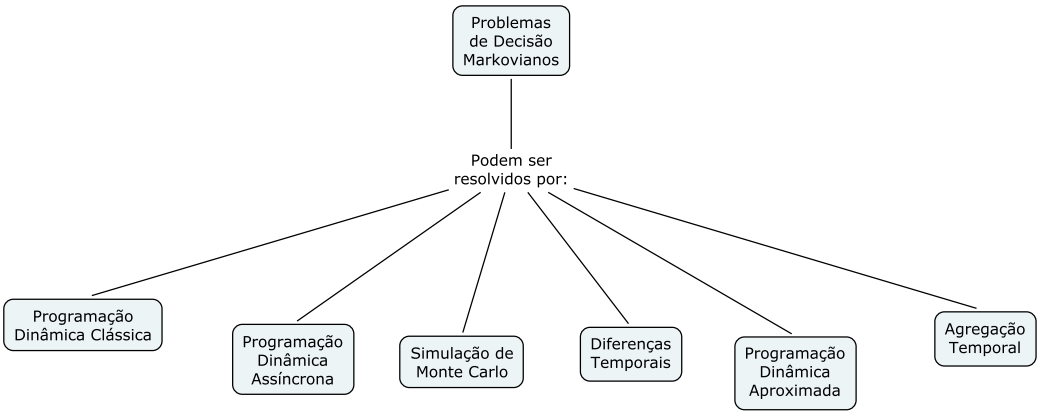

Figura 2.2: **Classe de algoritmos que resolvem PDMs**

Nas seções seguintes serão explicitadas as motivações, vantagens e desvantagens de cada uma dessas classes. Serão também elencados e explicados os principais algoritmos que as compõem. Inicialmente, na seção 2.3, será apresentada a classe de algoritmos aqui denominada Programação Dinâmica Clássica, que compreende os algoritmos Iteracão de Politica e Iteracão de Valor e suas respectivas extensões.

2.3 Programação Dinâmica Clássica: Algumas variações

Além dos algoritmos Iteração de Política e Iteração de Valor detalhados na seção anterior, a Programação Dinâmica Clássica compreende ainda outros algoritmos que consistem em meras variações. Alguns exemplos dessas variações são: Iteração de Política Modificada (PUTERMAN, 2005, PUTERMAN e SHIN, 1978), Iteração de Valor Jacobi (PUTERMAN, 2005), Iteração de Valor Gauss-Seidel (PUTERMAN, 2005) e Excesso de Relaxamento Sucessivo (PUTERMAN, 2005, REETZ, 1973). Na realidade, todas estas variações buscam ser mais eficientes computacionalmente comparadas aos algoritmos Iteração de Política e Iteração de Valor.

O algoritmo Iteração de Política Modificada (*Modified Policy Iteration*), por exemplo, não executa a etapa de Avaliação de Política até a convergência como no passo 4 do Algoritmo 1. Mais especificamente, ao invés de definir uma tolerância ϵ e executar a Avaliação de Política até que $|er|$ seja menor do que a tolerância, na Iteração de Política Modificada, a etapa de Avaliação de Política é executada uma

quantidade pré-determinada de vezes, independente da tolerância ϵ, sendo, portanto, obtido um valor aproximado para a política atual.

O algoritmo Iteração de Valor Jacobi (*Jacobi Value Iteration*), por outro lado, substitui a Eq. (2.9) no Algoritmo 2 pela Eq. (2.22):

$$\upsilon_{\lambda}^{n+1}(s) = (1 - \lambda p(s|s,a))^{-1} \cdot \max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{\substack{s' \in S \\ s' \neq s}} p(s'|s,a) \upsilon_{\lambda}^{n}(s') \right\}. \tag{2.22}$$

Em contrapartida, o Iteração de Valor Gauss-Seidel (*Gauss-Seidel Value Iteration*) também utiliza outra equação para atualização da função valor no passo 3 do Algoritmo 2. Seja um processo de decisão markoviano com espaço de estado S finito de cardinalidade $|S|$. Seja $i, j \in \{1, 2, 3, ..., |S|\}$. A equação (2.9) do Iteração de Valor pode ser reescrita da seguinte maneira:

$$\upsilon_{\lambda}^{n+1}(s_j) = \max_{a \in A(s_j)} \left\{ f(s_j,a) + \lambda \sum_{i} p(s_i|s_j,a) \upsilon_{\lambda}^{n}(s_i) \right\}. \tag{2.23}$$

A Iteração de Valor Gauss-Seidel substitui a Eq. (2.9) ou Eq.(2.23), pela Eq. (2.24):

$$\upsilon_{\lambda}^{n+1}(s_j) = \max_{a \in A(s_j)} \left\{ f(s_j,a) + \lambda \left[\sum_{i<j} p(s_i|s_j,a) \upsilon_{\lambda}^{n+1}(s_i) + \sum_{i \geq j} p(s_i|s_j,a) \upsilon_{\lambda}^{n}(s_i) \right] \right\}. \tag{2.24}$$

Diferente dos demais algoritmos supracitados, o algoritmo Excesso de Relaxamento Sucessivo (*Successive Over-relaxation*) calcula a função valor da iteração atual como uma combinação linear da função valor da iteração atual calculada pelo algoritmo Iteração de Valor Gauss-Seidel e a função valor da iteração anterior.

Finalmente, importa destacar os algoritmos Iteração de Conjunto de Políticas (CHANG, 2013) (*Policy Set Iteration*) e Iteração de Conjunto de Valores (*Value Set Iteration*) (CHANG, 2014). Seja $\Delta_n \in \mathbb{L}$ um conjunto de políticas estacionárias markovianas, sorteadas na iteração n de acordo com uma distribuição de probabilidade qualquer. Na etapa de Avaliação de Política da Iteração de Conjunto de Políticas, é calculado $\upsilon_{\lambda}^{\mathcal{L}}$ para todo $\mathcal{L} \in \Delta_n$. Em contrapartida, na etapa de Melhoria de Política, a Eq. (2.8) do Algoritmo 1 é substituída pela Eq. (2.25):

$$d_{n+1} \in \arg\max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) \max_{\mathcal{L} \in \Delta_n} \upsilon_{\lambda}^{\mathcal{L}}(s') \right\}. \tag{2.25}$$

Inspirado pela ideia do Iteração do Conjunto de Políticas, o algoritmo Iteração

de Conjunto de Valores considera um conjunto de funções valor a cada iteração, utilizando a Eq. (2.26) ao invés da Eq. (2.9) do Algoritmo 2:

$$v_{\lambda}^{n+1}(s) = \max_{a \in A(s)} \left\{ f(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) \max \left\{ v_{\lambda}^{n}(s'), \max_{\mathcal{L} \in \Delta_n} v_{\lambda}^{\mathcal{L}}(s') \right\} \right\}. \quad (2.26)$$

LEITE *et al.* (2020) estenderam o algoritmo Iteração do Conjunto de Políticas, propondo uma modificação que mantém uma única função valor, mas que maximiza a expressão do lado direito de (2.26) considerando apenas um subconjunto arbitrário das ações. Note que ao considerar um conjunto limitado de ações em cada estado, o algoritmo explora todo o conjunto de políticas que podem ser geradas a partir das combinações das ações consideradas em cada estado.

O mapa conceitual exposto na Figura 2.3 sumariza os algoritmos que compõem a classe, aqui denominada de Programação Dinâmica Clássica.

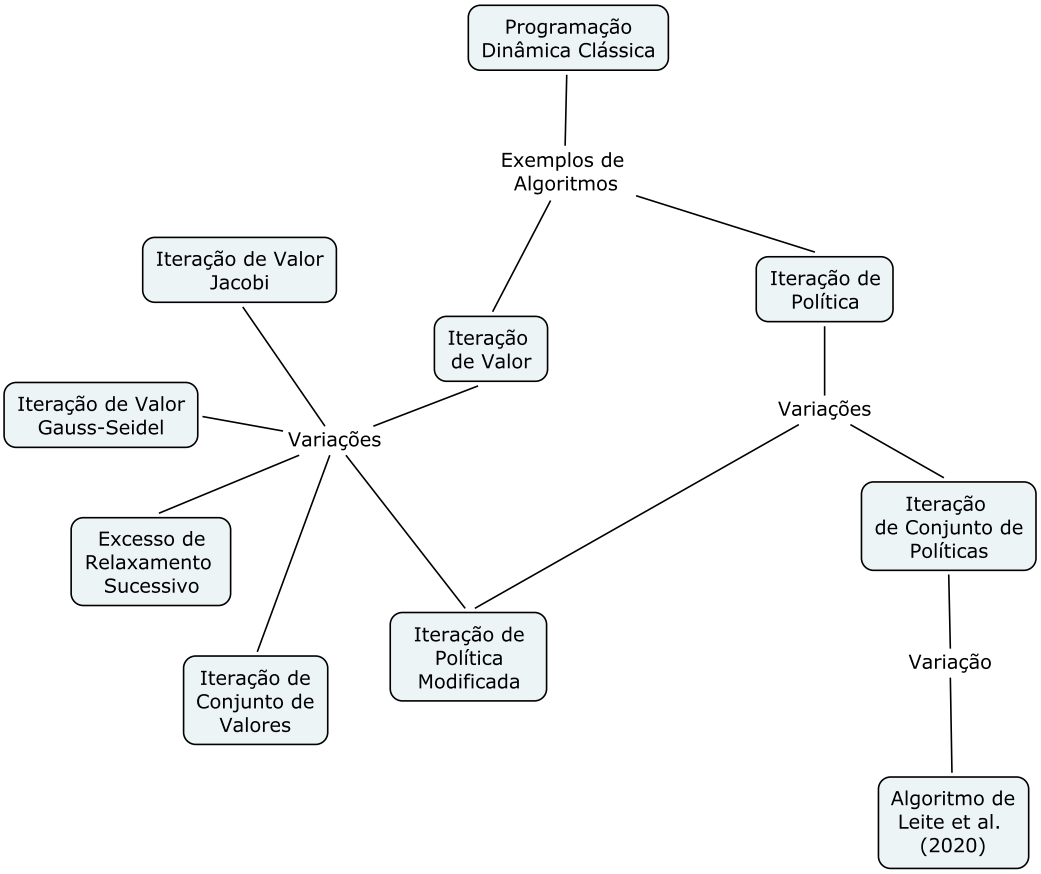

Figura 2.3: **Algoritmos de Programação Dinâmica Clássica**

2.4 Programação Dinâmica Assíncrona

Outra forma de resolver problemas de decisão de Markov é utilizando a classe de algoritmos denominada Programação Dinâmica Assíncrona. Os algoritmos da Programação Dinâmica Assíncrona utilizam a mesma Eq. (2.24) de Gauss-Seidel para atualização da função valor dos estados. Contudo estes algoritmos utilizam um método próprio para definir a ordem em que os estados tem sua função valor atualizada. Para estes algoritmos, a ordem em que a função valor do estados é atualizada influencia no tempo de convergência. Conforme o método utilizado, os algoritmos da Programação Dinâmica Assíncrona podem ser divididos em 4 grupos. Cada um desses 4 grupos tem seu algoritmo precursor que são: Varredura Priorizada (MOORE e ATKESON, 1993), Programação Dinâmica em Tempo Real (BARTO *et al.*, 1995), LAO* (HANSEN e ZILBERSTEIN, 2001) e Iteração de Valor Topológica (DAI e GOLDSMITH, 2007a, DAI *et al.*, 2011).

O Varredura Priorizada (*Prioritized Sweeping* - PS) (MOORE e ATKESON, 1993) e suas variações utilizam uma função matemática para criar uma fila de priorização de estados de modo que os estados no topo da fila tem sua função valor atualizada primeiro. Algumas extensões do Varredura Priorizada são: Programação Dinâmica Focada (FERGUSON e STENTZ, 2004), Iteração de Valor Priorizada (WINGATE *et al.*, 2005), Melhoria da Iteração de Valor Priorizada (DE GUADALUPE GARCIA-HERNANDEZ *et al.*, 2012), Iteração de Valor com base no Tempo Médio de Primeira Passagem (DEBNATH *et al.*, 2018) e os algoritmos desenvolvidos por MOHAGHEGHI *et al.* (2020). Estes algoritmos se diferenciam essencialmente com relação à função matemática utilizada para ordenar a fila de prioridades. O PS, por exemplo, utiliza o erro de Bellman para ordenar os estados na fila, sendo o erro de Bellman dado por:

$$B(s) = v_\lambda^{n+1}(s) - v_\lambda^n(s). \tag{2.27}$$

Outro conjunto de algoritmos tem como precursor o algoritmo Programação Dinâmica em Tempo Real (*Real-Time Dynamic Programming* - RTDP) (BARTO *et al.*, 1995). No RTDP e suas variações, a ordem em que os estados possuem seus valores atualizados é a ordem em que eles são visitados em uma trajetória simulada ou real (SUTTON e BARTO, 2018). No RTDP, por exemplo, o valor do estado corrente é atualizado. Em seguida, o algoritmo escolhe a ação gulosa e simula a transição para um estado seguinte por meio das probabilidades de transição. Assim, o valor do novo estado é atualizado e a simulação prossegue até que um estado terminal seja visitado. Se a cadeia de Markov for ergódica ou se um estado inicial diferente puder ser escolhido a cada nova simulação, a convergência é garantida (BARTO *et al.*, 1995). Contudo, realizar simulações do processo de decisão Markoviano im-

põe um esforço computacional adicional em relação aos algoritmos da Programação Dinâmica Clássica.

Algumas variações do algoritmo RTDP são: Programação Dinâmica em Tempo Real Rotulada (*Labeled Real-Time Dynamic Programming - Labeled RTDP* ou LRTDP) (BONET e GEFFNER, 2003a), Programação Dinâmica em Tempo Real Limitada (*Bounded Real-Time Dynamic Programming - Bounded RTDP* ou BRTDP) (MCMAHAN *et al.*, 2005), Programação Dinâmica em Tempo Real Focalizada (*Focused Real-Time Dynamic Programming - FRTDP*) (SMITH e SIMMONS, 2006) e Programação Dinâmica em Tempo Real Incremental Limitada (*Bounded Incremental Real-Time Dynamic Programming - BIRTDP*). Estes algoritmos se diferenciam com relação ao critério de parada e reinício da simulação ou com relação ao modo como a ação gulosa é escolhida. No LRTDP, por exemplo, a simulação é interrompida e reiniciada quando o estado visitado possui um valor que já tenha convergido satisfatoriamente, independentemente se o estado é terminal ou não. Já no BRTDP a ação gulosa é escolhida com base em um limite inferior calculado para a função valor ótima. Em contrapartida, no FRTDP, a ação gulosa é escolhida com base em um limite superior calculado para a função valor ótima.

Além dos algoritmos supracitados, vale destacar nesta seção os algoritmos de buscas heurísticas, como o LAO* (HANSEN e ZILBERSTEIN, 2001) e HDP (BONET e GEFFNER, 2003b). O algoritmo HDP realiza uma busca em profundidade no espaço de estados do MDP à medida que atualiza a função valor dos estados que visita e os rotula como resolvidos ou não resolvidos. Um estado é considerado resolvido pelo algoritmo quando seu resíduo de Bellman e de todos os seus estados sucessores sob uma política gulosa são menores do que um determinado limite. Um estado rotulado como resolvido não tem mais seu valor atualizado. Assim, o algoritmo termina quando todos os estados são considerados como resolvidos, isto é, quando o erro de Bellman de todos os estados for menor do que um determinado limite.

O algoritmo LAO* é derivado dos algoritmos A* (HART *et al.*, 1968, EDELKAMP e SCHRODL, 2011) e AO* (EDELKAMP e SCHRODL, 2011, NILSSON, 1982). O algoritmo A* resolve problemas determinísticos e o algoritmo AO* resolve problemas estocásticos em que um processo não pode retornar para um estado já visitado. O LAO* resolve problemas de caminho mínimo estocástico mais gerais em que é permitido regressar a qualquer estado anteriormente visitado. Este algoritmo é caracterizado pela construção de um grafo cujos nós correspondem aos estados do processo de decisão markoviano. A construção do grafo se inicia pela estado inicial do MDP. A cada iteração, o algoritmo expande um nó não terminal, adicionando os seus respectivos estados sucessores. O nó folha que será expandido é escolhido utilizando uma heurística. Em seguida, o algoritmo aplica o iteração de política ou

iteração de valor para atualizar o valor do estado expandido, bem como o valor de todos os estados antecessores por meio do qual é possível alcançar o estado expandido utilizando a política gulosa. Uma desvantagem destes algoritmos é que expandir o grafo referente ao processo de decisão markoviano impõe um esforço computacional adicional em relação aos algoritmos da Programação Dinâmica Clássica.

Algumas extensões do algoritmo LAO* são: ILAO* (HANSEN e ZILBERSTEIN, 2001), LAO* Biderecional (BLAO$\star$) (BHUMA, 2004), LAO* Reverso (RLAO*) (DAI e GOLDSMITH, 2006) e o Multi-thread BLAO* (MBLAO*) (DAI e GOLDSMITH, 2007b). Estes algoritmos se diferenciam, basicamente, pela maneira como o grafo é expandido. No ILAO*, por exemplo, a cada iteração, todos os nós folha do grafo são expandidos ao invés de expandir apenas um único nó como no LAO*. Já no BLAO*, a expansão do grafo é realizada tanto a partir do estado inicial quanto a partir do estado terminal e de maneira paralela. Em contrapartida, no RLAO*, o grafo é expandido apenas a partir do estado terminal de modo que todos os estados que transicionam para o estado folha são adicionados ao grafo. Por último, no MBLAO*, a expansão do grafo pode ser realizada não só a partir do estado inicial e/ou estado terminal, mas também a partir de outros estados intermediários.

Tanto os algoritmos relacionados ao PS quanto os algoritmos relacionados ao RTDP e LAO* não levam em conta a estrutura da cadeia de Markov. Entretanto, é claro que o valor de um estado $s \in S$ depende do valor de seus estados sucessores $s' \in S$ (DAI *et al.*, 2011). Desse modo, esta dependência causal, sugere que, normalmente, seja mais eficiente atualizar o valor dos estados sucessores s' antes de atualizar o valor do estado s (DAI *et al.*, 2011). Esta é justamente a ideia do algoritmo Iteração de Valor Topológica (*Topological Value Iteration* - TVI) (DAI e GOLDSMITH, 2007a, DAI *et al.*, 2011).

Considere o processo de decisão de Markov da Figura 2.4. No Iteração de Valor Topológica, o PDM é dividido em componentes fortemente conectados. No exemplo da Figura 2.4, o PDM é dividido em 3 componentes fortemente conectados: $C1$, $C2$ e $C3$. Em seguida, na ordem topológica encontrada, ele resolve cada componente individualmente utilizando o Iteração de Valor (DAI *et al.*, 2009). Assim, os estados em $C3$ teriam sua função valor atualizada primeiro do que os estados em $C2$ e $C1$. Por outro lado, os estados em $C1$ seriam os últimos a terem sua função valor atualizada.

Algumas variações do TVI são: Iteração de Valor Topológica Focalizada (DAI *et al.*, 2011, 2009) e Iteração de Valor Hierárquica Acelerada (LARACH *et al.*, 2017). A diferença entre o Iteração de Valor Topológica Focalizada (*Focused Topological Value Iteration* - FTVI) e o TVI é que o primeiro elimina ações sub-ótimas antes de encontrar os componentes fortemente conectados do PDM. O FTVI leva em consideração que, embora algumas ações não façam parte da política ótima, elas geram

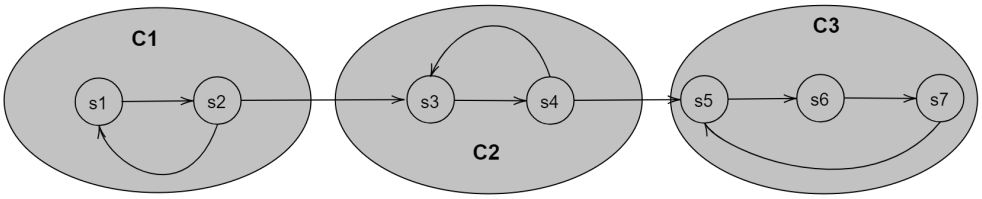

Figura 2.4: **Componentes Fortemente Conectados de um PDM**

conexões entre vários estados e, consequentemente, provocam a formação de grandes componentes conectados (DAI *et al.*, 2011). Neste sentido, ao eliminar as políticas sub-ótimas, o FTVI visa reduzir a dimensionalidade dos componentes conectados do PDM, sendo, portanto, executado mais rápido que o TVI (DAI *et al.*, 2009). Em contrapartida, o algoritmo Iteração de Valor Hierárquica Acelerada (*Accelerated Hierarchical Value Iteration* - AHVI), desenvolvido por LARACH *et al.* (2017), utiliza um algoritmo diferente dos artigos anteriores para encontrar os componentes fortemente conectados do PDM.

Conforme destacado por DIBANGOYE *et al.* (2008), identificar os componentes fortemente conectados para estabelecer a ordem em que os estados devem ser atualizados diminui a quantidade de iterações necessárias para encontrar uma política ótima. Por outro lado, esta etapa de identificação dos componentes fortemente conectadas impõe um esforço computacional adicional em relação aos algoritmos da Programação Dinâmica Clássica. Além disso, quando todos os estados do problema formam juntos um único componente fortemente conectado, a Iteração de Valor Topológica apresentará uma performance esperada equivalente à Iteração de Valor tradicional.

Por último, DIBANGOYE *et al.* (2008) destaca que o TVI e suas extensões identificam a ordem em que os componentes fortemente conectados devem ser atualizados, mas não diferencia os estados dentro do mesmo componente. Como explicado por DIBANGOYE *et al.* (2008), seja um estado s'' sucessor do estado s', que é, por sua vez, sucessor do estado s. Portanto, $v(s)$ é mais dependente de $v(s')$ do que $v(s'')$. Isto significa que o nível de dependência entre s e s' é maior do que entre s e s''. De fato, o efeito de uma atualização de $v(s')$ em $v(s)$ será sentido após uma única iteração enquanto o efeito de $v(s'')$ necessitará de 2 iterações. Por este motivo, desejamos atualizar o valor de s'' primeiro, depois o valor de s' e, por último, o valor de s. Inspirado por este fato, DIBANGOYE *et al.* (2008) desenvolveu o algoritmo Iteração de Valor Topológica Incrementada (*Improved Topological Value Iteration* - iTVI) (DIBANGOYE *et al.*, 2008). O iTVI divide o espaço de estados em grupos de acordo com nível de iterações mínimo necessário para que uma atualização da

função valor de um estado tenha efeito sobre a função valor do estado inicial. Além disso, a ordem de atualização da função valor dos estados é estabelecida por este nível de iteração mínimo necessário.

Finalizando esta seção, vale destacar alguns pontos. Primeiro, embora estes algoritmos buscam atualizar a função valor dos estados em uma ordem tal que a quantidade de iterações necessárias para encontrar uma política ótima seja menor do que nos algoritmos da Programação Dinâmica Clássica, eles impõem um esforço computacional adicional ao utilizar simulações e heurísticas. Além disso, como cada grupo de algoritmos utiliza um método diferente que envolve simulações ou heurísticas para definir uma ordem de atualização da função valor dos estados, não existe uma superioridade absoluta de um algoritmo específico, isto é, não existe um algoritmo que tenha desempenho superior em todos os problemas, conforme relatado nos resultados dos experimentos descritos nos artigos citados nesta seção.

O mapa conceitual da Figura 2.5 evidencia as relações entre os algoritmos da Programação Dinâmica Assíncrona.

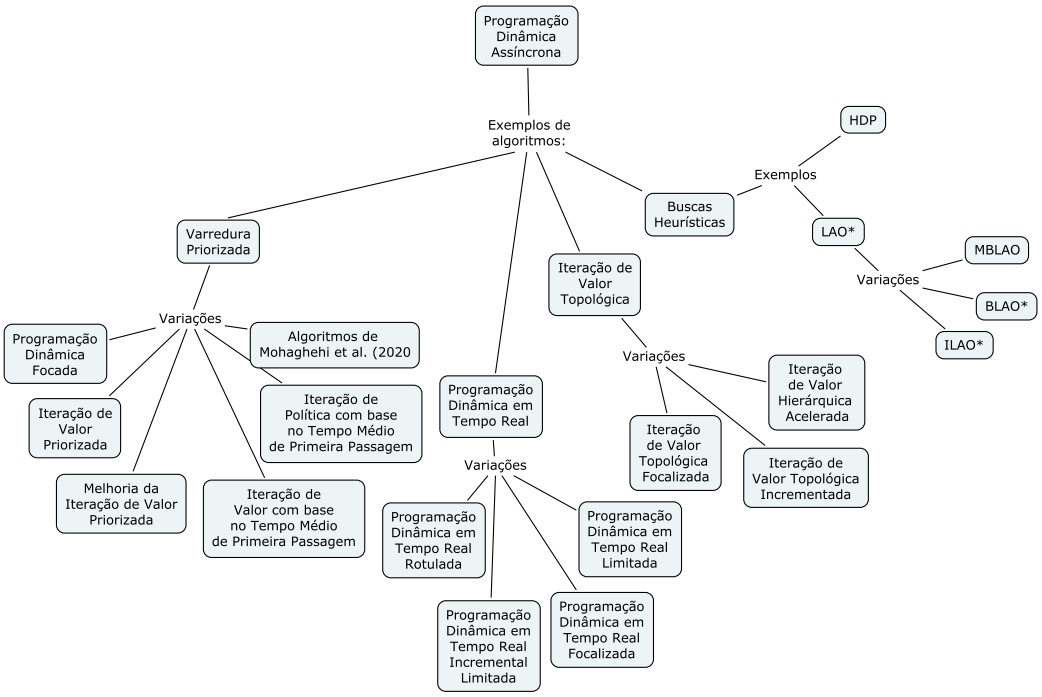

Figura 2.5: **Algoritmos de Programação Dinâmica Assíncrona**

2.5 Simulação de Monte Carlo

Além dos algoritmos da Programação Dinâmica Clássica e os algoritmos da Programação Dinâmica Assíncrona, existem algoritmos baseados em simulação de Monte Carlo. Estes últimos, diferente dos primeiros, não exigem conhecer a dis-

tribuição de probabilidade de todas as possíveis transições de estados SUTTON e BARTO (2018).

Em essência, os algoritmos que utilizam simulação de Monte Carlo buscam estimar a função Q da Eq. (2.11) simulando inúmeras vezes a evolução de um processo de decisão Markoviano até um estado terminal (SUTTON e BARTO, 2018). Desse modo, estes algoritmos estimam a função Q da Eq. (2.11) calculando a média dos retornos observados nas simulações depois das visitas ao estado $s \in S$, tendo escolhido uma ação $a \in A(s)$ e seguindo uma política $\mathcal{L} \in \mathbb{L}$ em diante. Pela Lei dos Grandes Números, à medida que mais retornos sejam observados, a média deverá convergir para o valor esperado (SUTTON e BARTO, 2018).

Evidentemente, para encontrar uma política ótima, estes algoritmos precisam estimar o valor de todas as ações $a \in A(s)$ para todos os estados $s \in S$ e, portanto, é preciso garantir que todos os pares estados-ação sejam visitados um número infinito de vezes em um número infinito de simulações. Uma maneira de garantir uma maior cobertura dos pares estado-ação disponíveis é utilizando políticas estocásticas com uma probabilidade não-nula de selecionar todas as ações $a \in A(s)$ em todos os estados $s \in S$ (SUTTON e BARTO, 2018). Um exemplo de política estocástica é a política ε-gulosa. Em uma política ε-gulosa, na maior parte do tempo, a ação que possui o maior valor estimado é escolhida, mas com probabilidade ε uma ação não ótima é escolhida aleatoriamente.

A desvantagem dos algoritmos que utilizam simulação de Monte Carlo está na necessidade de simular o processo de Markov até alcançar um estado terminal, que em alguns casos pode demorar bastante. Em outros casos, pode nem existir um estado terminal e a simulação de Monte Carlo terá quer ser terminada arbitrariamente. Além disso, se o espaço de estados ou espaço de ações for muito grande, será necessário realizar muitas simulações para que o algoritmo convirja satisfatoriamente, tornando a convergência destes algoritmos mais lenta que os algoritmos da Programação Dinâmica Clássica, Programação Dinâmica Assíncrona e Aprendizagem por Diferença Temporal.

2.6 Aprendizagem por Diferença Temporal

Assim como os algoritmos baseados em Monte Carlo, os algoritmos de Aprendizagem por Diferença Temporal utilizam simulações do processo de decisão Markoviano para estimar a função valor de um estado e por isso não exigem conhecimento da distribuição de probabilidade de todas as possíveis transições de estados (SUTTON e BARTO, 2018). Entretanto, assim como os algoritmos da Programação Dinâmica Clássica, eles estimam a função valor de um estado utilizando as estimativas da função valor de seus estados sucessores (SUTTON e BARTO, 2018). A Tabela 2.2

lista os principais algoritmos de Diferença Temporal e suas respectivas referências.

Algoritmo	**Referência**
Aprendizagem Q	(WATKINS, 1989)
SARSA	(SUTTON e BARTO, 2018, RUMMERY e NIRANJAN, 1994)
SARSA Esperado	(SUTTON e BARTO, 2018)
Aprendizagem Q(λ)	(PENG e WILLIAMS, 1994)
Aprendizagem Atrasada Q	(STREHL *et al.*, 2006)
Aprendizagem Duplo Q	(HASSELT, 2010)
Aprendizagem Q Veloz	(AZAR *et al.*, 2011)
Aprendizagem Q Duplo Ponderado	(ZHANG *et al.*, 2017)
Aprendizagem Atrasada Duplo Q	(ABED-ALGUNI e OTTOM, 2018)
Aprendizagem Q Zap	(DEVRAJ e MEYN, 2017a,b, DEVRAJ *et al.*, 2019)
Aprendizagem Q Zap e Deslize	(HE *et al.*, 2020)
Aprendizagem Q SOR	(KAMANCHI *et al.*, 2019)
Aprendizagem Generalizada Q Veloz	(JOHN *et al.*, 2020)

Tabela 2.2: Conjunto de algoritmos de Diferença Temporal

De certo modo, todos os algoritmos de Diferenças Temporais se assemelham ao algoritmo Aprendizagem Q, diferenciando-se apenas em alguns aspectos a fim de obter um melhor desempenho computacional. No SARSA, por exemplo, em vez de se utilizar o valor da ação ótima no estado sucessor por meio do operador max na Eq. (2.16), utiliza-se o valor de uma ação sorteada a partir de uma distribuição de probabilidade definida sobre o espaço de ações. Mais especificamente, no SARSA, escolhe-se uma ação a ser tomada no estado sucessor de acordo com uma política estocástica, por exemplo, uma política ε-gulosa em que, na maior parte do tempo, a ação que possui o maior valor estimado é escolhida, mas com probabilidade ε uma ação não ótima é escolhida aleatoriamente. Assim, é o valor desta ação no estado sucessor que é utilizado para atualizar o valor da ação no estado atual. Em contrapartida, o SARSA Esperado (*Expected SARSA*) utiliza o valor esperado do estado sucessor sob a política atual para atualizar o valor do estado corrente.

Sabe-se que em alguns modelos estocásticos, o algoritmo Aprendizagem Q não apresenta bom desempenho, pois superestima o valor de ações, isto é, gera um viés positivo no valor das ações uma vez que utiliza o operador máximo para determinar o valor do próximo estado (HASSELT, 2010). Em outras palavras, o algoritmo Aprendizagem Q apresenta um viés de maximização (SUTTON e BARTO, 2018).

Para superar esta questão, HASSELT (2010) propôs o algoritmo Aprendizagem Duplo Q (*Double Q-learning*) em que são utilizadas duas funções Q e cada uma delas é atualizada utilizando o valor da outra para o estado sucessor.

Entretanto, o algoritmo Aprendizagem Duplo Q subestima o valor das ações (HASSELT, 2010). Desse modo, na tentativa de encontrar um equilíbrio entre a superestimação do Aprendizagem Q e a subestimação do Aprendizagem Duplo Q, ZHANG *et al.* (2017) propuseram o algoritmo Aprendizagem Q Duplo Ponderado (*Weighted Double Q-learning*) que usa um estimador duplo ponderado. Em outras palavras, este algoritmo utiliza duas funções Q e calcula uma média ponderada entre elas.

Outra variação do algoritmo Aprendizagem Q é o algoritmo Aprendizagem Atrasada Q (*Delayed Q-learning*). A principal diferença entre os algoritmos é que, no Aprendizagem Q, o valor do par estado-ação é atualizado sempre que o par é visitado, ao passo que no Aprendizagem Atrasada Q, o valor do par estado-ação é atualizado após uma quantidade pré-determinada de visitas ao par. Entretanto, assim como o Aprendizagem Q, o Aprendizagem Atrasada Q superestima o valor das ações (ABED-ALGUNI e OTTOM, 2018). Assim, a fim de superar esta questão, ABED-ALGUNI e OTTOM (2018) propuseram o algoritmo Aprendizagem Atrasada Duplo Q (*Double Delayed Q-learning*) que consiste, essencialmente, na combinação do algoritmo Aprendizagem Duplo Q com o Aprendizagem Atrasada Q.

Igualmente, a fim de obter uma convergência mais rápida que o Aprendizagem Q, o algoritmo Aprendizagem Q Veloz usa duas estimativas sucessivas do valor da função Q em cada passo do episódio. Em contrapartida, o algoritmo Aprendizagem Q(λ) incorpora o conceito de rastreios de elegibilidade (*eligibility traces*) no algoritmo Aprendizagem Q.

O Aprendizagem Q Zap (*Zap Q-learning*) introduz uma matriz de ganho no algoritmo Aprendizagem Q de maneira semelhante ao algoritmo de Newton-Raphson estocástico, com o objetivo de melhorar a taxa de convergência do Aprendizagem Q. No entanto, baixas taxas de aprendizado podem anular o benefício proporcionado pela matriz de ganhos (HE *et al.*, 2020). Neste sentido o algoritmo Aprendizagem Q Zap e Deslize (*Glide and Zap Q-learning*) introduz outro parâmetro a fim de minimizar o efeito originado pelas baixas taxas de aprendizado. Já o Aprendizagem Q SOR (*SOR Q-learning*) foi desenvolvido, essencialmente, a partir da combinação do algoritmo Aprendizagem Q com a abordagem do algoritmo Excesso de Relaxamento Sucessivo. De maneira semelhante, o algoritmo Aprendizagem Generalizada Q Veloz (*Generalized Speedy Q-learning*) originou-se a partir da combinação do algoritmo Aprendizagem Q Veloz com a abordagem do algoritmo Excesso de Relaxamento Sucessivo.

Por fim, os algoritmos de Diferença Temporal podem ser relacionados conforme

o mapa conceitual da Figura 2.6

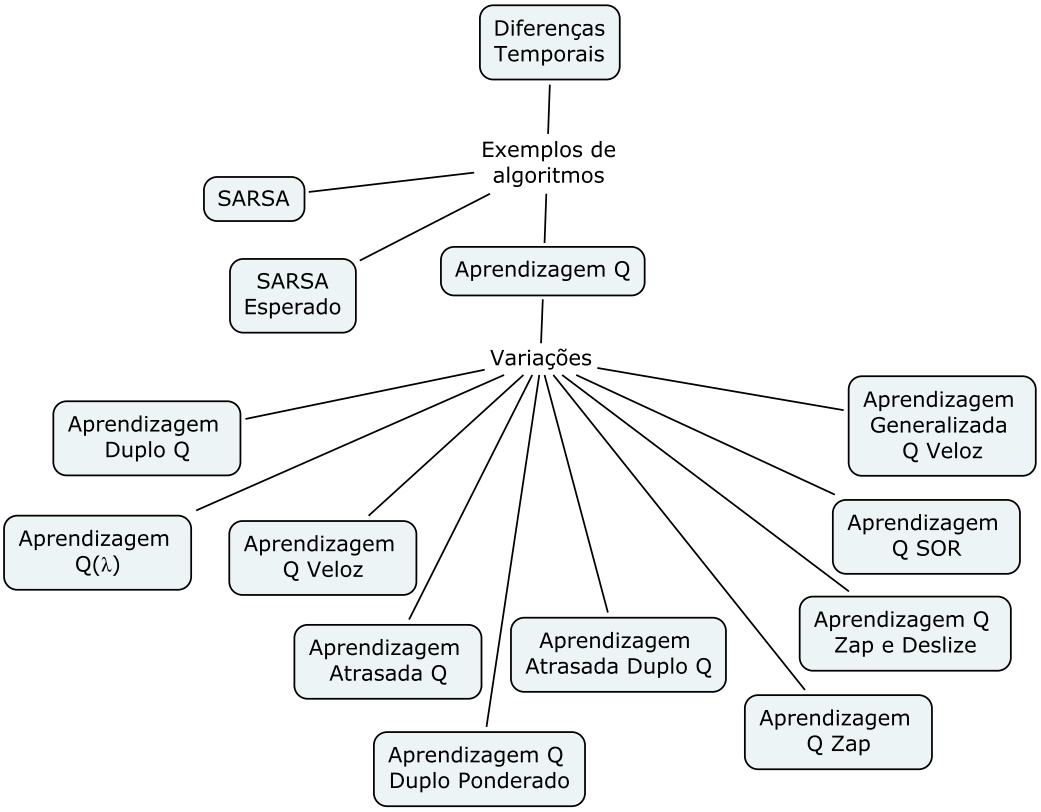

Figura 2.6: **Algoritmos de Diferença Temporal**

2.7 Programação Dinâmica Aproximada

Além de exigirem um modelo para o sistema, os algoritmos da Programação Dinâmica Clássica exigem um crescimento exponencial no esforço computacional à medida que cresce a dimensão do problema (BERTSEKAS *et al.*, 2000). Esse crescimento foi batizado como *maldição da dimensionalidade*. Neste contexto, POWELL (2007) salienta que os modelos de Programação Dinâmica Aproximada podem ser utilizados para resolver problemas que sofrem da maldição da dimensionalidade, especialmente problemas em que o espaço de estados é demasiado grande a ponto de tornar inviável sua enumeração.

Os algoritmos baseados em aproximação de modelo fazem parte da Programação Dinâmica Aproximada. Estes algoritmos utilizam um PDM aproximado para encontrar a política ótima. O algoritmo Iteração de Valor com Informação Parcial (*Partial Information Value Iteration*) (ARRUDA *et al.*, 2013), por exemplo, utiliza um modelo aproximado gerado pelo truncamento das distribuições de probabilidades de transição considerando uma tolerância predeterminada (ARRUDA *et al.*, 2013). A cada iteração, a tolerância diminui e o modelo é refinado até convergir para o

modelo verdadeiro a uma taxa linear ótima que coincide com a taxa de convergência do algoritmo Iteração de Valor original (ARRUDA *et al.*, 2013).

Adicionalmente, fazem parte da Programação Dinâmica Aproximada os algoritmos baseados em Aproximação da Função Valor (*Value-function Approximation*). Estes algoritmos utilizam modelos paramétricos e não paramétricos para estimar a função valor. Neste sentido, conforme destacado por SUTTON e BARTO (2018), a atualização do valor de um estado provoca uma atualização dos valores de outros estados no mesmo instante. Alguns exemplos de algoritmos que utilizam Aproximação da Função Valor podem ser encontrados em CHEN *et al.* (2020) e DIDDIGI *et al.* (2019).

O mapa conceitual da Figura 2.7 apresenta, resumidamente, os algoritmos relacionados à Programação Dinâmica Aproximada.

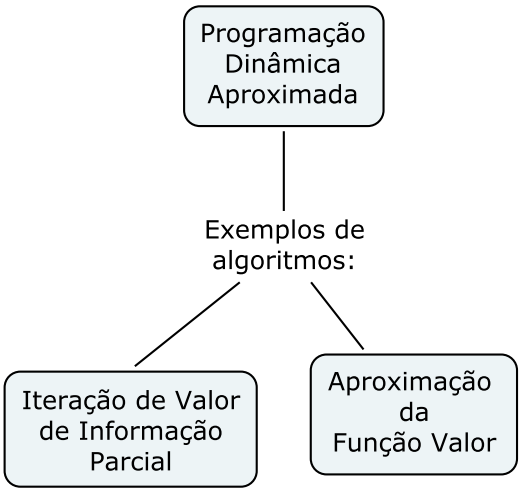

Figura 2.7: **Algoritmos de Programação Dinâmica Aproximada**

2.8 Agregação de Estados

Outra classe de algoritmos comumente utilizada para resolver PDM com alta dimensionalidade é a Agregação de Estados (*State Aggregation*). Estes algoritmos visam reduzir o espaço de estados do problema ao combinar vários estados para formar estados agregados. Estes estados agregados formam outro problema com um espaço de estados menor, que pode ser resolvido pelos algoritmos da Programação Dinâmica Clássica (BERTSEKAS *et al.*, 2000). Assim, a função valor ótima dos estados agregados pode ser utilizada como uma aproximação da função valor ótima dos estados do problema de Markov original (CHEN *et al.*, 2021, BERTSEKAS, 2022). Contudo, um ponto que merece destaque é o fato da agregação de estados não preservar a propriedade markoviana. A propriedade de Markov diz respeito ao

estado futuro de um sistema depender apenas do estado atual dele e não do seu estado nos instantes anteriores.

Considere um problema de decisão de Markov com espaço de estados $S = \{1, 2, \cdots, n\}$. Seja $p_{ij}(a)$ a probabilidade de transição do estado $i \in S$ para o estado $j \in S$ quando o tomador de decisão escolhe a ação $a \in A$. Seja um conjunto finito C de estados agregados cuja cardinalidade seja m, tal que $m < n$. Considere que os estados agregados sejam representados pelas letras x e y, e os estados originais, pelas letras i e j. Seja $\phi_{jy} \geq 0$ a probabilidade do estado original j pertencer ao estado agregado y, tal que $\sum_{y \in C} \phi_{jy} = 1, \forall j \in S$. Seja também $d_{xi} \geq 0$ a probabilidade do estado agregado x estar associado ao estado original i, tal que $\sum_{i=1}^{n} d_{xi} = 1, \forall x \in C$. Assim, a probabilidade de transição do estado agregado x para o estado agregado y quando o agente escolhe a ação $a \in A$ é dada por (BERTSEKAS, 2022):

$$\hat{p}_{xy}(a) = \sum_{i=1}^{n} d_{xi} \cdot \sum_{j=1}^{n} p_{ij}(a) \cdot \phi_{jy}. \tag{2.28}$$

Repare que no cálculo de $\hat{p}_{xy}(a)$, somente são levadas em consideração as transições em 1 passo. Contudo, um processo partindo de um estado qualquer $i \in x$, pode passar antes por outro estado $k \in x$ antes de alcançar um estado qualquer $j \in y$, e a probabilidade disso acontecer não está sendo levada em conta na equação (2.28). Desse modo, de maneira geral, $\hat{p}_{xy}(a)$ não possui a propriedade de Markov, e a função valor, $\hat{v}(x), x \in C$, encontrada para os estados da cadeia agregada por meio dos algoritmos das seções anteriores é aproximada.

Contudo, KEMENY e SNELL (1976) apresentam uma condição necessária e suficiente para que a propriedade de Markov seja preservada na agregação de estados: uma cadeia de Markov agregada possui a propriedade de Markov se e somente se, para cada par de subconjuntos (x, y), a probabilidade de transicionar de um estado $i \in S$ para o subconjunto y é a mesma para todo $i \in x$. Neste caso, não fará diferença em qual estado i do subconjunto x o processo irá começar, a probabilidade de alcançar o subconjunto y será a mesma, e a trajetória do processo dentro do subconjunto x até alcançar o subconjunto y não irá importar. Nesta circunstância, a probabilidade $\hat{p}_{xy}(a)$ possuirá a propriedade de Markov e a solução encontrada para os estados da cadeia agregada será exata. Entretanto, ainda assim, a solução para os estados de fora da cadeia agregada será aproximada visto que a função valor destes estados é calculada utilizando as probabilidades ϕ_{jy} da seguinte maneira (BERTSEKAS, 2022):

$$\tilde{v}(j) = \sum_{y \in C} \phi_{jy} \cdot \hat{V}(y). \tag{2.29}$$

Os algoritmos de agregação de estados se diferenciam pela forma como o espaço

de estado da cadeia de Markov original é particionado. Por exemplo, os algoritmos de agregação rígida de estados (*Hard State Aggregation*) (BERTSEKAS *et al.*, 2000) agregam os estados em subconjuntos disjuntos. Por outro lado, nos algoritmos de agregação flexível de estados (*Soft State Aggregation*) (BERTSEKAS *et al.*, 2000, SINGH *et al.*, 1994, DUAN *et al.*, 2019), os estados da cadeia original podem pertencer a mais de um subconjunto da cadeia de Markov agregada (BERTSEKAS *et al.*, 2000). Em contrapartida, a agregação de estados por meio de redes grosseiras (*Coarse Grid State Aggregation*), também denominada agregação por estados representativos (*aggregation by representative states*) combina ideias da agregação rígida de estados com a agregação flexível de estados (BERTSEKAS *et al.*, 2000).

Adicionalmente, importa destacar os algoritmos de agregação hierárquica de estados (*Hierarchical State Aggregation*) (VOELKEL *et al.*, 2020, POWELL, 2007, 2022) e os algoritmos de agregação de estados baseados em características (*Feature-based State Aggregation*) (BERTSEKAS, 2022, TSITSIKLIS e VAN ROY, 1996, BERTSEKAS, 2018). Esses algoritmos são considerados extensões dos algoritmos anteriores. Os algoritmos de agregação hierárquica de estados, ao invés de utilizarem apenas um nível de agregação, utilizam diversos níveis de agregação estruturados de maneira hierárquica (POWELL, 2022). Em contrapartida, os algoritmos de agregação de estados baseados em características (*Feature-based State Aggregation*) (BERTSEKAS, 2022, TSITSIKLIS e VAN ROY, 1996, BERTSEKAS, 2018) agrupam os estados com características semelhantes em um mesmo subconjunto. Por último, vale mencionar os algoritmos de agregação adaptativa de estados (CHEN *et al.*, 2021, BERTSEKAS *et al.*, 1989) em que os estados da cadeia de Markov original que pertencem aos subconjuntos da cadeia de Markov agregada mudam a cada iteração.

O mapa conceitual da Figura 2.8 apresenta, resumidamente, os algoritmos relacionados à Agregação de Estados.

2.9 Agregação Temporal

Por último, importa destacar uma classe de algoritmos denominada Agregação Temporal (*Time Agregation*), desenvolvida a partir do trabalho de CAO *et al.* (2002). Diferente da Agregação de Estados, a Agregação Temporal visa reduzir o espaço de estados do problema mantendo a propriedade markoviana (CAO *et al.*, 2002). Em síntese, na Agregação Temporal o espaço de estados é dividido em dois subconjuntos disjuntos: um pequeno subconjunto $F \in S$ de estados controláveis ou que são mais interessantes sob o ponto vista do controle e outro subconjunto $F^c \in S$ muito maior de estados não controláveis (SUN *et al.*, 2007, ARRUDA *et al.*, 2017). As trajetórias que saem e retornam ao subconjunto de interesse F geram uma cadeia de Markov embutida (CAO *et al.*, 2002).

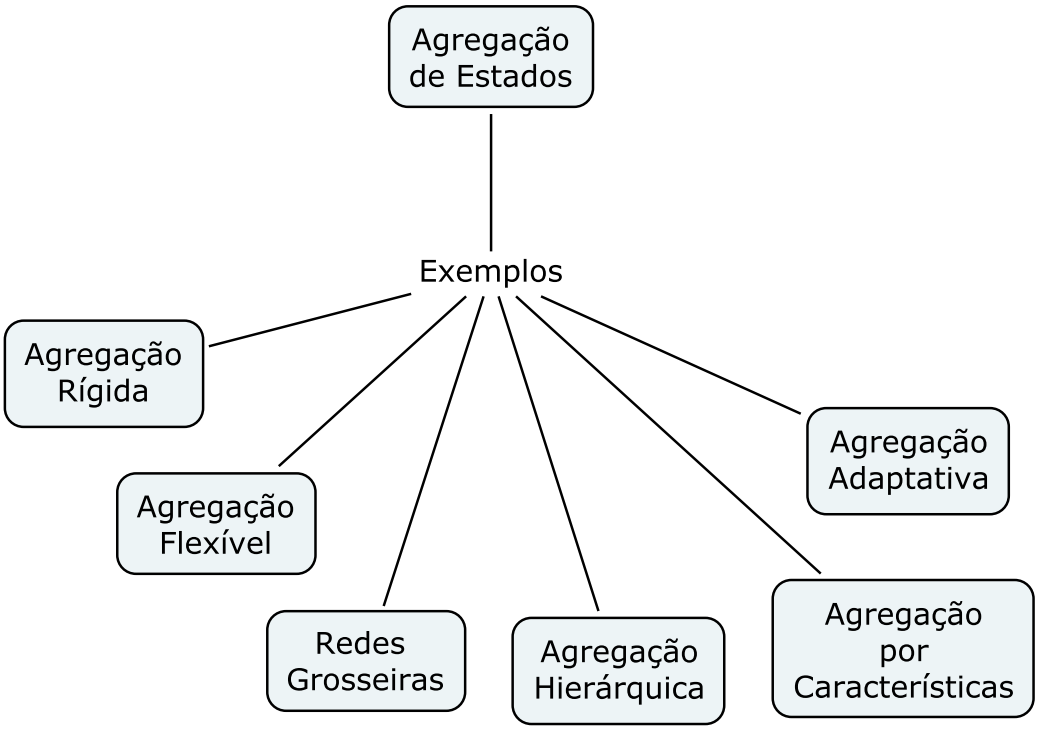

Figura 2.8: **Agregação de Estados**

Para resolver problemas de decisão Markovianos de horizonte de tempo infinito e com custo médio a partir de um processo de decisão semi-Markov (PDSM) com espaço de estado reduzido gerado pela Agregação Temporal, CAO *et al.* (2002) propõem o algoritmo Iteração de Política Agregada por Tempo (*Time-Aggregated Policy Iteration*) (CAO *et al.*, 2002) enquanto SUN *et al.* (2007) propõem o algoritmo Iteração de Valor Incremental (*Incremental Value Iteration*) (SUN *et al.*, 2007).

A título de exemplo, considere que um tomador de decisão tenha optado por dividir o espaço de estado do processo de decisão markoviano da Figura 2.1 conforme explicitado na Figura 2.9.

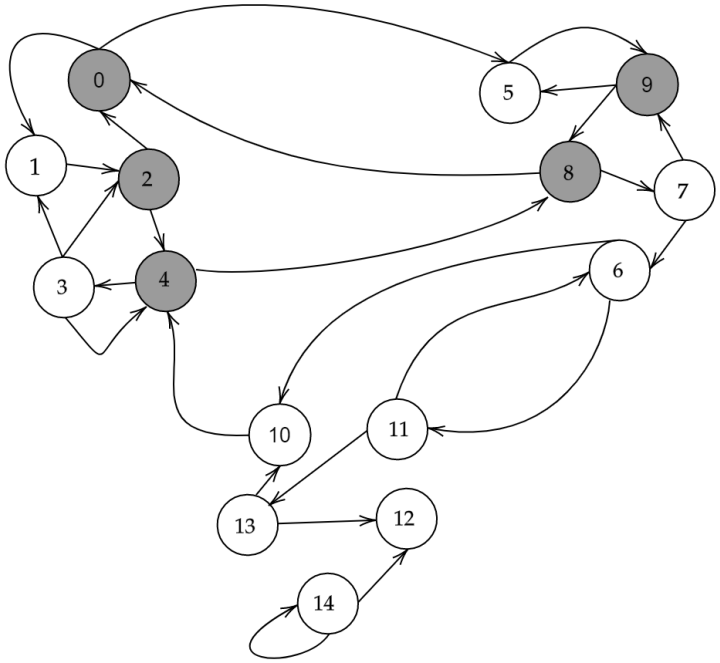

Figura 2.9: **Exemplo de particionamento do espaço de estado de um PDM.**

Assim, de forma bem resumida, o algoritmo de CAO *et al.* (2002) aplica o Iteração de Política Agregado por Tempo nos estados em cinza enquanto fixa uma política *ad-hoc* para os estados em branco. Por outro lado, o algoritmo de SUN *et al.* (2007) aplica o Iteração de Valor Incremental nos estados em cinza após fixar uma política *ad-hoc* para os estados em branco. Neste caso, os algoritmos de CAO *et al.* (2002) e SUN *et al.* (2007) atingem a solução ótima apenas quando o subconjunto de maior cardinalidade (conjunto de estados em branco) é composto por estados não controláveis. Caso contrário, eles alcançarão uma solução sub-ótima, pois otimiza dentro de uma região arbitrária F do espaço de estados (estados em cinza) enquanto aplica uma política de controle *ad hoc* fixa nos estados restantes (estados em branco).

Utilizando uma abordagem diferente, ARRUDA e FRAGOSO (2009, 2011) propõem utilizar a Agregação Temporal como um passo intermediário para resolver problemas de decisão de Markov. Especificamente, em uma primeira etapa, eles sugerem aplicar a técnica de Agregação Temporal proposta por CAO *et al.* (2002) para encontrar um PDSM equivalente ao PDM original. Em seguida, eles propõem aplicar a abordagem de PUTERMAN (2005) para transformar o PDSM equivalente obtido na etapa anterior em um novo PDM com espaço de estado reduzido. Desse

modo, é possível aplicar o algoritmo Iteração de Valor tradicional para resolver o novo PDM cuja solução é a mesma do PDM original. Pela mesma razão que os algoritmos de CAO *et al.* (2002) e SUN *et al.* (2007), a otimalidade do algoritmo de ARRUDA e FRAGOSO (2009, 2011) somente é garantida quando o subconjunto de maior cardinalidade for composto por estados não controláveis.

Para garantir que uma política ótima sempre possa ser encontrada em problemas de horizonte de tempo infinito e custo médio utilizando a agregação temporal, ARRUDA e FRAGOSO (2015) propuseram o algoritmo Agregação Temporal em Duas Etapas (*Two Phase Time Aggregation*). Na primeira etapa, é fixada uma política *ad-hoc* para os estados em F^c e aplica-se a Iteração de Política ou Iteração de Valor nos estados em F. Na segunda etapa, a função valor de todos os estados é calculada utilizando as propriedades da cadeia de Markov embutida. Assim, nesta etapa também é possível encontrar uma política melhor para os estados em F^c, que se tornará a política *ad-hoc* fixa na próxima iteração. Estas etapas são aplicadas sucessiva e alternadamente até a convergência.

Para exemplificar, considere, novamente, o processo de decisão Markoviano da Figura 2.1 e o particionamento proposto na Figura 2.9. O algoritmo de ARRUDA e FRAGOSO (2015), propõe, inicialmente, fixar uma política *ad-hoc* para os estados em branco enquanto aplica Iteração de Política ou Iteração de Valor nos estados em cinza, que formam o processo de decisão semi-Markov. Assim, o algoritmo deles obtém uma nova política e uma nova função valor para estes estados. Após esta primeira etapa, o algoritmo de ARRUDA e FRAGOSO (2015) aplica uma etapa de melhoria de política nos estados em branco a partir da função valor dos estados em cinza obtida na etapa anterior. Estas duas etapas se repetem alternadamente até a convergência. Como o algoritmo de ARRUDA e FRAGOSO (2015) otimiza nos dois subconjuntos da partição, é garantido que a política ótima seja encontrada. Posteriormente, ARRUDA e FRAGOSO (2016) estenderam o algoritmo Agregação Temporal em Duas Etapas para resolver PDM de horizonte de tempo infinito e custo descontado.

Um ponto desfavorável do algoritmo de ARRUDA e FRAGOSO (2015) é o esforço computacional exigido visto que o algoritmo calcula as trajetórias de fora do subconjunto de interesse, isto é, no subconjunto F^c, cuja cardinalidade é análoga a de todo o espaço de estados. Considere, novamente, o processo de decisão Markoviano da Figura 2.1 e o particionamento proposto na Figura 2.9. O algoritmo de ARRUDA e FRAGOSO (2015) avalia as trajetórias fora da cadeia embutida, isto é, as trajetórias em F^c. Em outras palavras, o algoritmo de ARRUDA e FRAGOSO (2015) avalia as trajetórias que passam pelos estados em F^C (estados em branco) antes de atingir algum estado em F (estados em cinza). Como a cardinalidade de F^c é muito maior do que a cardinalidade de F, o algoritmo se torna computacionalmente

intensivo.

Por último, importa destacar o algoritmo de ARRUDA *et al.* (2017, 2019b), que propõe generalizar a abordagem de Agregação Temporal proposta por CAO *et al.* (2002) ao permitir que o espaço de estados seja particionado em um número finito, mas arbitrário de subconjuntos. Desse modo, diferentemente dos algoritmos elaborados por CAO *et al.* (2002), SUN *et al.* (2007) e ARRUDA e FRAGOSO (2015), que calculam as trajetórias de fora do subconjunto de interesse, o algoritmo proposto por ARRUDA *et al.* (2017, 2019b) calcula as trajetórias de saída dos subconjuntos da partição, sendo, por isso, computacionalmente mais eficiente. Ao calcular as trajetória de saída dos subconjuntos, o algoritmo de ARRUDA *et al.* (2017, 2019b) evita a inversão de grandes matrizes (ARRUDA *et al.*, 2017) visto que as cardinalidades dos subconjuntos tendem a ser menores do que o espaço de estado. Inicialmente, os autores desenvolveram a nova abordagem, denominada Agregação Temporal Multi-Agrupamentos (*Multi-cluster time aggregation*), para encontrar a distribuição estacionária da cadeia de Markov. Em seguida, LEITE *et al.* (2020) estenderam a abordagem para resolver problemas de decisão de Markov em que os estados de cada subconjunto são independentes e suas probabilidades são estacionárias.

O mapa conceitual da Figura 2.10 elenca os principais algoritmos de Agregação Temporal atualmente existentes.

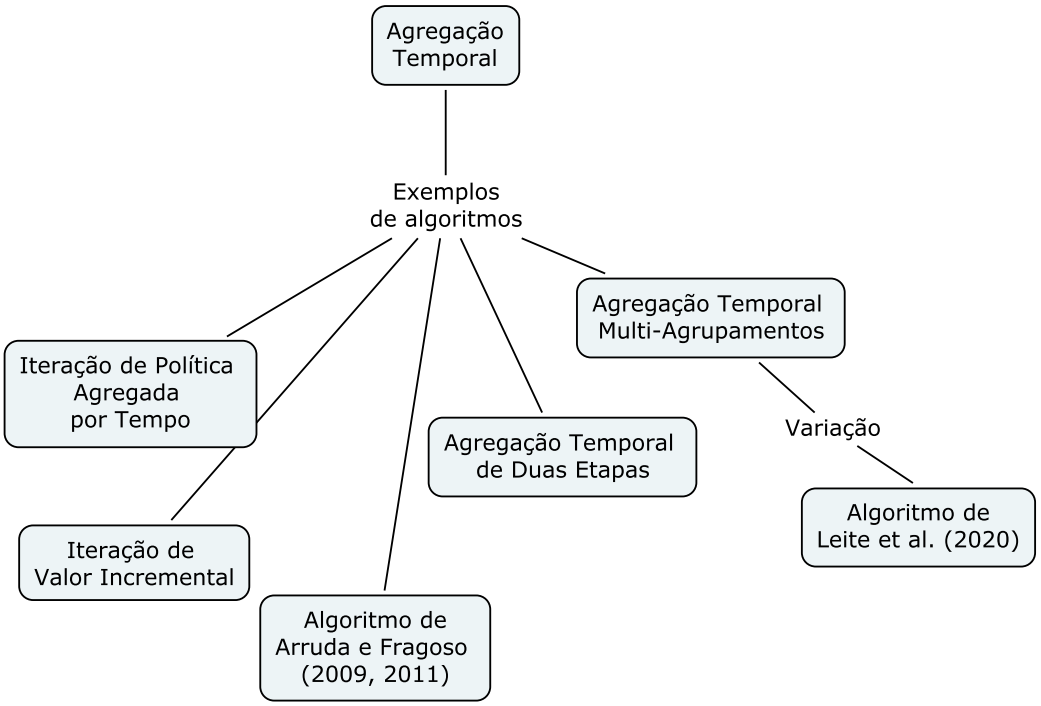

Figura 2.10: **Algoritmos de Agregação Temporal**

2.10 Contribuições da Tese

Conforme explicado na seção anterior, os algoritmos de Agregação Temporal propostos por CAO *et al.* (2002), SUN *et al.* (2007) e ARRUDA e FRAGOSO (2009, 2011) atingem a solução ótima apenas quando o subconjunto de maior cardinalidade é composto por estados não controláveis. Caso contrário, eles alcançarão uma solução sub-ótima, pois otimizam dentro de uma região arbitrária do espaço de estados enquanto aplicam uma política de controle *ad hoc* fixa nos estados restantes. Para garantir que uma política ótima sempre possa ser alcançada por meio da Agregação Temporal, é possível utilizar o algoritmo de ARRUDA e FRAGOSO (2015). Contudo, conforme já explicado na Seção 2.9, o algoritmo de ARRUDA e FRAGOSO (2015) exige um maior esforço computacional visto que calcula as trajetórias de fora do subconjunto de interesse cuja cardinalidade é análoga a de todo o espaço de estados.

Neste contexto, esta Tese de Doutorado tem como objetivo apresentar algoritmos inéditos que garantam que uma política ótima sempre possa ser encontrada como no algoritmo de ARRUDA e FRAGOSO (2015), porém que sejam mais eficientes computacionalmente. A convergência dos novos algoritmos para a solução ótima é provada e validada. Adicionalmente, esta Tese demonstra como os novos algoritmos podem ser combinados às abordagens de programação dinâmica aproximada ou aprendizado por reforço, gerando um novo algoritmo aproximado para resolver problemas de alta dimensionalidade com um menor tempo computacional. Os resultados provenientes da aplicação dos algoritmos novos propostos a um problema de produção e gestão de estoques e a outro problema de gestão de filas ilustram o potencial do método para lidar com problemas PDM de custo médio com alta dimensionalidade. Mais especificadamente, os resultados mostram um comportamento interessante, ou seja, que os algoritmos inéditos propostos tendem a alcançar a vizinhança da solução ótima significativamente mais rápido do que os algoritmos clássicos de programação dinâmica tais como o Iteração de Política, o Iteração de Valor e a abordagem original de Agregação Temporal de Duas Fases de ARRUDA e FRAGOSO (2015).

Os algoritmos propostos nesta Tese utilizam um novo esquema de particionamento que divide o espaço de estado em um número finito, mas arbitrário de subconjuntos. Diferente do esquema de particionamento proposto em ARRUDA *et al.* (2019b), o esquema de particionamento proposto nesta Tese permite embutir um PDM em um PDSM equivalente por meio de uma simples análise de absorção, aplicada individualmente dentro de cada subconjunto da partição, o que implica um esforço computacional que depende da cardinalidade do subconjunto em análise. Desse modo, é possível controlar e limitar o esforço computacional geral, fornecendo

maior flexibilidade e uma clara melhoria em relação à Agregação Temporal em Duas Fases (ARRUDA e FRAGOSO, 2015).

A título de exemplo, considere novamente o processo de decisão markoviano da Figura 2.1 e o particionamento proposto na Figura 2.11. Conforme explicado na seção anterior 2.9, o algoritmo de ARRUDA e FRAGOSO (2015) avalia as trajetórias que passam pelos estados em branco antes de atingir algum estado em cinza. Como a quantidade de estados em branco é muito maior do que a quantidade de estados em cinza, o algoritmo se torna computacionalmente intensivo. Utilizando o esquema de particionamento proposto nesta Tese, os novos algoritmos avaliam as trajetórias de fora da cadeia embutida, mas dentro de cada subconjunto por vez. Em outras palavras, os novos algoritmos avaliam as trajetórias que passam pelos estados em branco de cada subconjunto separadamente antes de atingir algum estado em cinza. Por isso, o esforço computacional depende da cardinalidade do subconjunto em análise, sendo portanto menor se comparado ao algoritmo de ARRUDA e FRAGOSO (2015).

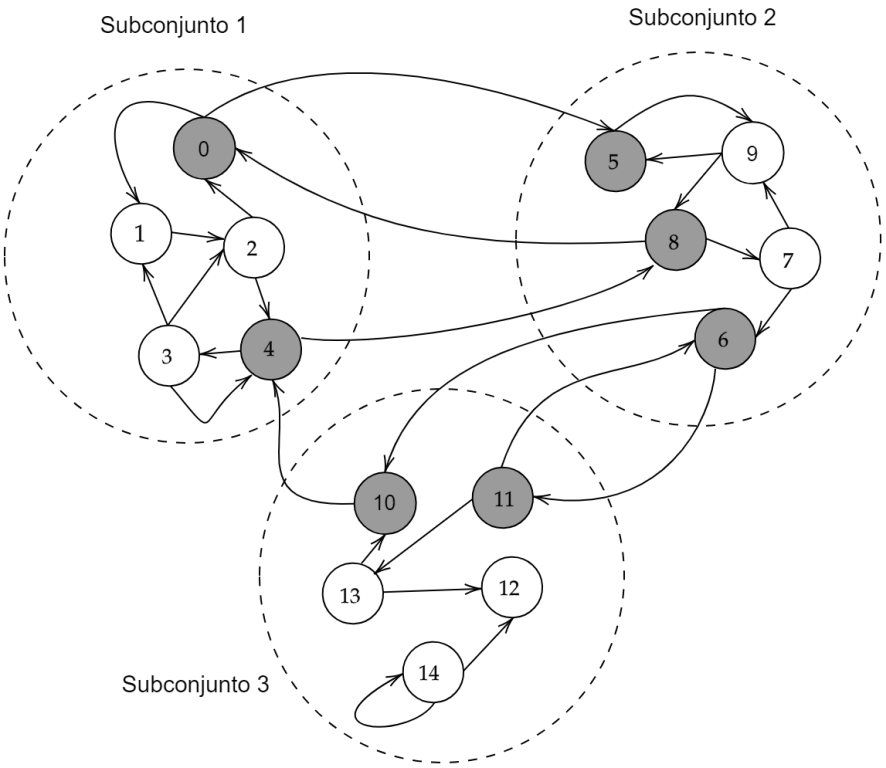

Figura 2.11: **Exemplo de multi-particionamento do espaço de estado de um PDM.**

Pode-se argumentar que o esquema de particionamento proposto é semelhante ao Iteração de Valor Topológico (DAI e GOLDSMITH, 2007a), pois explora as pro-

priedades de conectividade do MDP. No entanto, como o TVI decompõe o espaço de estados em componentes fortemente conectados, seu desempenho está vinculado às propriedades topológicas do MDP. De fato, foi relatado que o desempenho do TVI diminui para MDPs com grandes componentes fortemente conectados (DAI e GOLDSMITH, 2007a). O esquema de particionamento proposto aqui evita esse problema, pois as propriedades de comunicação dentro dos subconjuntos são definidas de maneira a permitir um cálculo distribuído da função valor que usa um único subconjunto da partição por vez.

Baseando-se na avaliação de trajetórias semi-regenerativas, nossa abordagem tem alguma semelhança conceitual com algoritmos de *rollout* (POWELL, 2007, BERTSEKAS, 2021, CAKIR *et al.*, 2023), já que estes avaliam a função valor com base em trajetórias simuladas de comprimento fixo a partir do estado atual. Considere, por exemplo, o processo de decisão markoviano da Figura 2.1 e o particionamento proposto na Figura 2.9. Os algoritmos de *rollout* simulam a evolução do PDM, partindo de cada estado em cinza, por uma quantidade fixa de passos (e.g., 5). A desvantagem destes algoritmos de *rollout* é que a complexidade das trajetórias cresce com o comprimento do horizonte de simulação, assim como o número de destinos possíveis, limitando assim o desempenho do algoritmo. Em contraste, nossa abordagem depende da agregação de tempo para dar origem a trajetórias semi-regenerativas de duração aleatória, mas com um conjunto fixo de estados de destino estabelecidos pelo particionamento proposto. Além disso, propomos um esquema de particionamento com múltiplos subconjuntos e propriedades de comunicação projetadas para limitar o comprimento das trajetórias semi-regenerativas. O esquema é definido de modo que cada trajetória semi-renegerativa seja restrita a um único subconjunto da partição.

Por fim, é mister destacar que, no dia 18/11/2023, foi realizada uma busca nas bases *Scopus* e *Web of Science* (WoS) por artigos que tenham citado os trabalhos de CAO *et al.* (2002), ARRUDA *et al.* (2017) e ARRUDA *et al.* (2019b). A estrutura desta revisão da literatura, bem como, a quantidade de artigos encontrados podem ser visualizados na Tabela 2.3.

Tabela 2.3: Revisão de Literatura sobre Agregação Temporal

Base	Resultados
Scopus	66 artigos citaram: A time aggregation approach to Markov decision processes (CAO *et al.*, 2002)
Scopus	2 artigos citaram: Multi-partition time aggregation for Markov Chains (ARRUDA *et al.*, 2017)
Scopus	2 artigos citaram: A multi-cluster time aggregation approach for Markov Chains (ARRUDA *et al.*, 2019b)
WoS	57 artigos citaram: A time aggregation approach to Markov decision processes (CAO *et al.*, 2002)
WoS	Nenhum artigo citou: Multi-partition time aggregation for Markov Chains (ARRUDA *et al.*, 2017)
WoS	1 artigo citou: A multi-cluster time aggregation approach for Markov Chains (ARRUDA *et al.*, 2019b)

Os resultados encontrados foram exportados para arquivos em formato BibTex que foram, em seguida, importados para o R (R CORE TEAM, 2020). Utilizando a função *mergeDbSources* do pacote Bibliometrix (ARIA e CUCCURULLO, 2017), os resultados das bases foram agrupados e os artigos repetidos foram eliminados. Desconsiderando artigos repetidos, foram encontrados ao todo sessenta e oito artigos e, após uma análise do resumo de todos eles e uma leitura dinâmica de alguns, não foi identificado nenhum trabalho que tenha implementado alguma das propostas acima elencadas. Enquanto alguns artigos mencionam os trabalhos explicitados na Tabela 2.3, mas não aplicam, de fato, a Agregação Temporal (e.g. WU e JIA, 2020)), outros não utilizam uma abordagem multi-agrupamentos (e.g. LI e WU, 2016). Na realidade, o trabalho mais próximo de um dos objetivos acima elencados é o artigo de LEITE *et al.* (2020), que estenderam a abordagem Agregação Temporal Multi-Agrupamentos para resolver problemas de decisão de Markov em que os estados de cada subconjunto são independentes e suas probabilidades são estacionárias. Contudo, não foi identificado nenhum trabalho que tenha estendido a abordagem de ARRUDA *et al.* (2017, 2019b), para resolver problemas de decisão markovianos mais gerais, como proposto por esta Tese. Desse modo, esta Tese de Doutorado será o primeiro estudo que terá desenvolvido qualquer uma das propostas acima.

O mapa conceitual da Figura 2.12 contextualiza e expõe as motivações e objetivo principal desta Tese de Doutorado. Conforme pode ser visualizado na Figura 2.12 e descrito ao longo das seções desta Tese, os algoritmos atualmente existentes para resolver problemas de decisão markovianos possuem determinados pontos negativos que serviram de motivação para o objetivo principal desta Tese de Doutorado, que

é a construção de um conjunto de novos algoritmos com potencial para resolver problemas de decisão markovianos maiores, de maneira mais eficiente computacionalmente e com garantia de convergir sempre para a política ótima. Cabe ressaltar que a agregação temporal é a base para os novos algoritmos propostos neste trabalho.

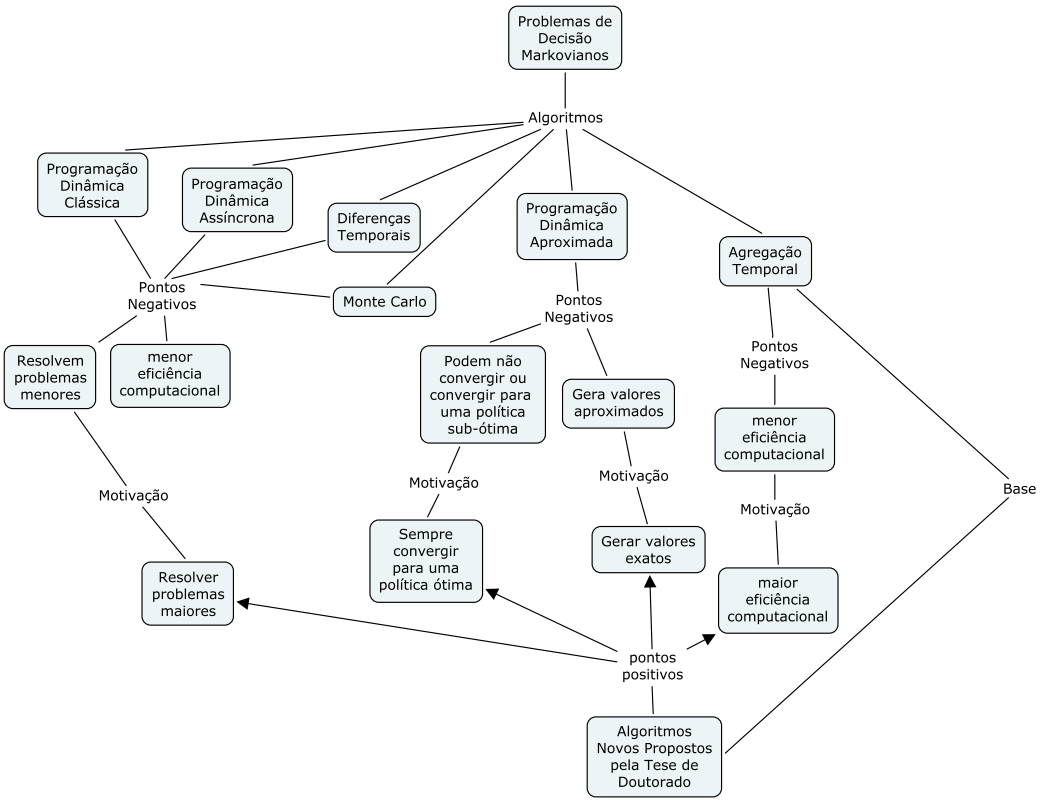

Figura 2.12: **Contextualização e Motivações da Tese de Doutorado**

Capítulo 3

As principais contribuições: Novo particionamento e algoritmos

Neste Capítulo, inicialmente, será apresentado o problema que será tratado. Em seguida, será proposto um novo esquema de particionamento que dá origem a propriedades de comunicação que viabilizam um esquema de computação distribuída pelos subconjuntos partição. Por último, são apresentados os novos algoritmos desenvolvidos que utilizam o novo esquema de particionamento proposto, bem como as provas de convergência para a política ótima.

3.1 Definição do problema

Seja um processo de decisão Markoviano, cuja dinâmica controlada é representada pelo processo estocástico $\{X_t\}, t \geqslant 0$ - com espaço de estados S finito de cardinalidade $|S|$. Defina-se também $A(i)$ como o conjunto finito de ações possíveis de serem tomadas quando o processo se encontra no estado $i \in S$. Assim, $A = \bigcup_{i \in S} A(i)$ é o conjunto de todas as ações possíveis.

A cada instante de tempo $t \geqslant 0$ em que o processo $\{X_t\}$ visita o estado $i \in S$, um agente toma uma ação $a \in A$ seguindo uma política estacionária Markoviana $\mathcal{L} : S \to A$, dentre um conjunto $\mathbb{L}$ de políticas estacionárias markovianas viáveis para o sistema. Cabe aqui ressaltar que uma política estacionária markoviana é uma função que especifica uma mesma ação $a \in A(s)$ a ser tomada sempre que o sistema visita o estado $s \in S$, independente do trajeto percorrido pelo sistema até chegar ao estado s (PUTERMAN, 2005). Após o agente tomar a ação, o processo se move no instante de tempo seguinte para um estado $X_{t+1} = j \in S$ com uma probabilidade, p_{ij}^a, que é função do estado atual do sistema e da ação tomada. Conforme já disposto, a Figura 2.1 da seção 2.1 ilustra um processo de decisão markoviano.

Seja $f : S \times A \to \mathbb{R}_+$ a função de custo imediato, sendo $f(i, a)$ o custo imediato

de escolher a ação $a \in A(i)$ no estado $i \in S$, em que $\mathbb{R}_+$ denota o conjunto de números reais não negativos. Assim, toda política $\mathcal{L} \in \mathbb{L}$ possui um custo médio de longo prazo associado dado por:

$$\eta^{\mathcal{L}} = \lim_{N \to \infty} \frac{1}{N} \sum_{t=0}^{N} f(X_t, \mathcal{L}(X_t)). \tag{3.1}$$

A exemplo de CAO *et al.* (2002), assumiremos que a cadeia de Markov é ergódica sob qualquer política viável $\mathcal{L} \in \mathbb{L}$, o que faz com que o custo médio de longo prazo independa do estado inicial. Cabe lembrar que em uma cadeia ergódica, é possível ir de qualquer estado para qualquer estado em tempo finito e o custo médio de longo prazo independe do estado inicial (BRÉMAUD, 1999) De acordo com a teoria clássica sobre PDM, para qualquer política $\mathcal{L} \in \mathbb{L}$, a solução da Equação (3.1) pode ser obtida encontrando um par $(V^{\mathcal{L}}, \eta^{\mathcal{L}})$ que resolva a equação de Poisson (PUTERMAN, 2005, CAO *et al.*, 2002):

$$V^{\mathcal{L}}(i) = f(i, \mathcal{L}(i)) - \eta^{\mathcal{L}} + \sum_{j \in S} p_{ij}^{\mathcal{L}(i)} V^{\mathcal{L}}(j),\ i \in S, \tag{3.2}$$

em que $V^{\mathcal{L}} : S \to \mathbb{R}$ é uma função valor real no espaço $\mathcal{V}$ de funções valores reais em S. Adicionalmente, Corolário 2 de ARRUDA e FRAGOSO (2015) afirma que, para qualquer ponto de parada $0 \leq \tau < \infty$:

$$V^{\mathcal{L}}(i) = E\left\{ \sum_{t=0}^{\tau-1} \left(f(X_t, \mathcal{L}(X_t)) - \eta^{\mathcal{L}} \right) + V^{\mathcal{L}}(X_\tau) \right\},\ i \in S. \tag{3.3}$$

A fim de minimizar o custo médio de longo prazo, o tomador de decisões busca uma política estacionária ótima $\mathcal{L}^* \in \mathbb{L}$ tal que:

$$\eta^* = \eta^{\mathcal{L}^*} \leq \eta^{\mathcal{L}}, \forall \mathcal{L} \in \mathbb{L}. \tag{3.4}$$

A Eq. (3.4) pode ser resolvida por meio do algoritmo clássico de Iteração de Valor Relativo (e.g., PUTERMAN, 2005).

3.1.1 O problema agregado

Quando o espaço de estado do MDP é muito grande, é possível definir um subconjunto bem menor $F \subset S$, fixar uma política d para o restante do espaço de estado, isto é, $d : F^c \to A$, $d(i) \in A(i)$, $\forall i \in F^c$, em que $F^c = S \setminus F$ e usar a agregação temporal para encontrar uma política sub-ótima para o problema original,

resolvendo o seguinte problema de otimização (ARRUDA e FRAGOSO, 2015):

$$\begin{aligned} \min \quad & \eta \\ \text{subject to} \quad & \eta \geq \eta^{\mathcal{L}} \ \forall \mathcal{L} \in \mathbb{L} \\ & \mathcal{L}_{\text{out}} = d, \end{aligned} \tag{3.5}$$

em que $\mathcal{L}_{\text{out}} : S \to F^c$ denota uma política estacionário restrita ao subconjunto $F^c \subset S$. O Teorema seguinte é uma reformulação do ARRUDA e FRAGOSO (2015, Teorema 1) e estabelece uma relação entre a agregação temporal e o problema original. Esta conexão será explorada pelos novos algoritmos a serem propostos neste trabalho.

Teorema 1. *Seja $\mathcal{L}^*$ a solução da Eq.* (3.4) *e seja $\mathcal{L}_{out}(i) = d(i) = \mathcal{L}^*(i)$, $\forall i \in F^c$ no Problema* (3.5)*. Então, as soluções para os problemas* (3.4) *e* (3.5) *são idênticas.*

Demonstração. Seja η_d^* a solução do problema 3.5. Assim, perceba que o espaço de soluções da Equação (3.5) é um subconjunto de $\mathbb{L}$, que é o espaço de soluções do problema de Markov original (3.4). Portanto, temos que $\eta_d^* \geq \eta^*$. Além disso, perceba que $\mathcal{L}^*$ também é uma solução viável para a Equação (3.5). Isto implica que $\eta_d^* \leq \eta^*$. Juntando as duas inequações, concluímos que $\eta_d^* = \eta^*$ □

Para resolver (3.5), é necessário definir um conjunto de tempos de parada τ_k, $k \geq 0$ tal que $\tau_0 = 0$ e $\tau_{k+1} = \min\{t > \tau_k : X_t \in F\}$. Para qualquer política $\mathcal{L} : \mathcal{L}_{\text{out}} = d$, o custo médio $\eta^{\mathcal{L}}$ coincide com o custo médio do processo semi-Markov (PUTERMAN, 2005, Capítulo 11) $Y_k = X_{\tau_k}$, $k \geq 0$, que evolui no subconjunto F e é composto pelos seguintes elementos:

- um espaço de ações $A_y = \bigcup_{i \in F} A_y(i)$, em que $A_y(i) = A(i) \, \forall i \in F$, i.e., as ações no subconjunto F coincidem com as ações do MDP original;
- uma função custo $h_f : F \times A \to \mathbb{R}_+$ que satisfaz:

$$h_f(i,a) = f(i,a) + E\left\{ \sum_{t=1}^{\tau_1 - 1} f(X_t, \mathcal{L}(X_t)) \, | X_0 = i \right\}, \ i \in F, \ a \in A(i); \tag{3.6}$$

- tempos de transição definidos como:

$$h_1(i,a) = h_f(i,a), \text{ for } f(j,a) \equiv 1 \ \forall j \in S, \ a \in A(j); \tag{3.7}$$

- e probabilidades de transição:

$$\tilde{p}_{ij}^a = p_{ij}^a + \sum_{k \in F^c} p_{ik}^a p_{kj}^d(\tau_1), \ p_{kj}^d(\tau_1) = P^{\mathcal{L}}(X_{\tau_1} = j | X_0 = k). \tag{3.8}$$

Se η_d^* resolve (3.5) e $\mathcal{L}_d^*$ é uma política ótima para o problema (3.5), então existe um par (V_d^*, η_d^*), em que $V_d^* : F \to \mathbb{R}$ é uma função valor real em F, que satisfaz:

$$V_d^*(i) = h_f(i,a) - \eta_d^* h_1(i,a) + \sum_{j \in F} \tilde{p}_{ij}^a V_d^*(j), i \in F, \tag{3.9}$$

em que η_d^* é também uma solução para (3.1) sob a política $\mathcal{L}_d^*$, com:

$$\mathcal{L}_d^*(i) = \begin{cases} \arg\min\limits_{a \in A(i)} \left\{ (h_f(i,a) - \eta_d^* h_1(i,a)) + \sum\limits_{j \in F} \tilde{p}_{ij}^a V_d^*(j) \right\}, & \text{if } i \in F, \\ d(i), & \text{if } i \in F^c. \end{cases} \tag{3.10}$$

O Teorema 2 a seguir, que é uma reformulação do ARRUDA e FRAGOSO (2015, Teorema 2), estabelece uma relação entre a função valor do problema de decisão semi-Markov (PDSM) proveniente da agregação temporal e a função valor do PDM original.

Teorema 2. *Seja o par (V_d^*, η_d^*) a solução da Eq. (3.9). Assumiremos que $\mathcal{L}_d^*$ satisfaz a Eq. (3.10). Seja $V^{\mathcal{L}_d^*}$ a função valor da política $\mathcal{L}_d^*$ no PDM original. Então:*

$$V^{\mathcal{L}_d^*}(i) = V_d^*(i), \forall i \in F, \tag{3.11}$$

e $\eta^{\mathcal{L}_d^*} = \eta_d^*$, em que o par $(V^{\mathcal{L}_d^*}, \eta^{\mathcal{L}_d^*})$ satisfaz a Eq.(3.2).

Demonstração. Utilizando o Teorema 1, temos que $\eta^{\mathcal{L}_d^*} = \eta_d^*$. Agora, seja $\tau = \min\{t > \tau_{i-1} : X_t \in F\}$. Seja também $\tilde{p}_{ij}^a = P(X_\tau = j | X_0 = i, \mathcal{L}(i) = a)$, $i, j \in F$. Utilizando a Eq. (3.3), é possível escrever:

$$\begin{aligned} V^{\mathcal{L}_d^*}(i) &= E\left\{ \sum_{t=0}^{\tau-1} \left(f(X_t, \mathcal{L}_d^*(X_t)) - \eta^{\mathcal{L}_d^*} \right) + V^{\mathcal{L}_d^*}(X_\tau) \right\}, i \in F, a = \mathcal{L}_d^*(i) \\ V^{\mathcal{L}_d^*}(i) &= h_f(i,a) - \eta_d^* h_1(i,a) + \sum_{j \in F} \tilde{p}_{ij}^a V^{\mathcal{L}_d^*}(j), i \in F, a = \mathcal{L}_d^*(i), \end{aligned} \tag{3.12}$$

em que a última igualdade é obtida utilizando as definições (3.6) e (3.7), e substituindo $\eta^{\mathcal{L}_d^*}$ por η_d^*. Por fim, perceba que a última equação é equivalente à Eq. (3.9). □

O próximo lema desempenhará um papel fundamental na prova de um resultado-chave que ilustra o poder do método de particionamento aqui proposto e nos permitirá implementar uma melhoria de política distribuída.

Lema 1. *Seja η_d^* a solução de* (3.5) *e assuma que $\mathcal{L}_d^*$ é uma política ótima para o problema* (3.5)*, então a função valor da política $\mathcal{L}_d^*$ no MDP original é dada por:*

$$V^{\mathcal{L}_d^*}(i) = \begin{cases} V_d^*(i), & \forall\, i \in F, \\ E\left\{\sum_{t=0}^{\tau_1 - 1} \left(f(X_t, \mathcal{L}_d^*(X_t)) - \eta_d^*\right) + V_d^*(X_{\tau_1})\right\}, & \forall\, i \in F^c. \end{cases} \tag{3.13}$$

Demonstração. A parte superior da Eq. (3.13), correspondente aos estados $i \in F$, é uma aplicação direta do Teorema 2. Em contrapartida, a parte referente aos estados $i \in F^c$ decorre da aplicação direta da Eq. (3.3) à política $\mathcal{L}_d^*$ com tempo de parada τ_1, substituindo $V^{\mathcal{L}_d^*}(X_{\tau_1})$ por $V_d^*(X_{\tau_1})$. A substituição é possível porque $\tau_1 = \min\{t > 0 : X_t \in F\}$, e isto implica que $X_{\tau_1} \in F$. □

Trajetórias semi-regenerativas

Embora a solução por agregação temporal via Eq. (3.9) atue somente no subconjunto F, é preciso resolver também as Eq. (3.6)-(3.8). Estas equações calculam os custos totais e as probabilidades de transição das trajetórias semi-regenerativas desde o começo no subconjunto F^c até o primeiro retorno de X_t, $t \geq 0$, ao subconjunto F. Para avaliar estas trajetórias, é necessário definir $\mathcal{O}_{F^c}^d := [p^d(i,j)]$, $i, j \in F^c$, sendo que p_{ij}^d é a probabilidade de transição entre estados $i \in F^c$ e $j \in F^c$ sob a política $\mathcal{L}_{\text{out}} = d$ no subconjunto F^c, e

$$E_{F^c} = \left(I - \mathcal{O}_{F^c}^d\right)^{-1}.$$

A teoria clássica de cadeias de Markov (e.g., BRÉMAUD, 1999) diz que o elemento $e_{F^c}[i,j]$ da matriz E_{F^c} é o número esperado de vezes que uma trajetória, partindo do estado $i \in F^c$, passará pelo estado $j \in F^c$ antes do processo X_t, $t \geq 0$, atingir o subconjunto F. Adicionalmente, a teoria clássica de análise de absorção indica que (e.g., CAO *et al.*, 2002, ARRUDA e FRAGOSO, 2015):

$$\begin{aligned} h_f(i,a) &= f(i,a) + \sum_{k \in F^c} p_{ik}^a \sum_{j \in F^c} e_{F^c}[k,j] f(j, d(j)),\ i \in F, \\ \tilde{p}_{ij}^a &= p_{ij}^a + \sum_{k \in F^c} p_{ik}^a \sum_{m \in F^c} e_{F^c}[k,m] p_{mj}^d,\ i\,, j \in F. \end{aligned} \tag{3.14}$$

É válido notar que a Eq. (3.14) possui somatórios tanto em F quanto em F^c. De fato, ARRUDA e FRAGOSO (2015, Remark 1) reportam uma complexidade da ordem $O(|F| \cdot |A| + |F|^3 + |F^c|^3)$, que é dominada pela maior cardinalidade entre os subconjuntos F e F^c. Portanto, visto que a agregação temporal é proposta para o caso em que $|F| << |F^c|$, limitar o esforço computacional nos somatórios que envolvem o subconjunto F^c é uma forma clara de melhorar a eficiência geral da abordagem

de agregação temporal. Em face do exposto, a próxima seção indtroduzirá um novo método de particionar o espaço de estados pensado para fazer exatamente isso ao induzir uma estrutura de comunicação específica entre os subconjuntos da partição.

3.2 Uma nova abordagem para particionar o espaço de estados em múltiplos conjuntos

Nesta seção, é apresentado um novo método que permite particionar o espaço de estados em múltiplos conjuntos de diferentes formas e que, quando combinado com a agregação temporal (CAO *et al.*, 2002), pode originar algoritmos eficientes para solucionar processos de decisão de Markov, como será explicitado mais adiante.

Definição 1 (O esquema de particionamento em múltiplos conjuntos $\mathcal{P}(S)$)**.** *Particione o espaço de estados S em n subconjuntos disjuntos $\{B_1, B_2, \ldots, B_n\}$, de modo que $\bigcup_{l=1}^{n} B_l = S$, $B_l \cap B_m = \emptyset$, $\forall\, l, m \in \{1, \ldots, n\} : l \neq m$:*

$$F_l \triangleq \{j \in B_l : \exists\, i \notin B_l \ e\ \exists\, a \in A(i) \ \ tal\ que\ p_{ij}^{a} > 0\}, \tag{3.15}$$

$$F = \bigcup_{l=1}^{n} F_l, \quad I_l = B_l \setminus F_l,\ 1 \leq l \leq n, \quad F^c = S \setminus F = \bigcup_{l=1}^{n} I_l. \tag{3.16}$$

A novidade do esquema de particionamento em múltiplos conjuntos como na Definição 1 é que ele dá origem a uma topologia que concentra a informação dentro de um pequeno subconjunto de estados (o conjunto F na Eq. (3.16)) que podem ser acessados, em uma única transição, a partir de diferentes subconjuntos dentro da partição. Isso, por sua vez, permite simplificar, distribuir e reduzir a complexidade na avaliação das trajetórias entre visitas sucessivas ao subconjunto F, conforme detalhado a seguir.

O esquema proposto é ilustrado na Figura 3.1 para o processo de decisão markoviano da Figura 2.1. Para cada subconjunto B_l na partição, os estados em cinza pertencem ao conjunto $F_l \subset B_l$ de estados de *fronteira* da Eq. (3.15). Perceba que F_l é composto pelos estados que podem ser diretamente acessados por estados pertencentes a outros subconjuntos da partição. Por exemplo, o estado $0 \in B_1$ na Figura 3.1 é estado de fronteira e pertence a F_1 porque este pode ser acessado a partir B_2. Similarmente, $4 \in B_1$ também faz parte do conjunto de estados fronteira $F_1 = \{0, 4\}$, uma vez que pode ser acessado diretamente a partir de B_3. Por outro lado, os demais estados em B_1 somente podem ser acessados diretamente por outros estados no mesmo subconjunto e, portanto, não são estados de fronteira.

Em contrapartida, os estados em branco na Figura 3.1 pertencem ao conjunto

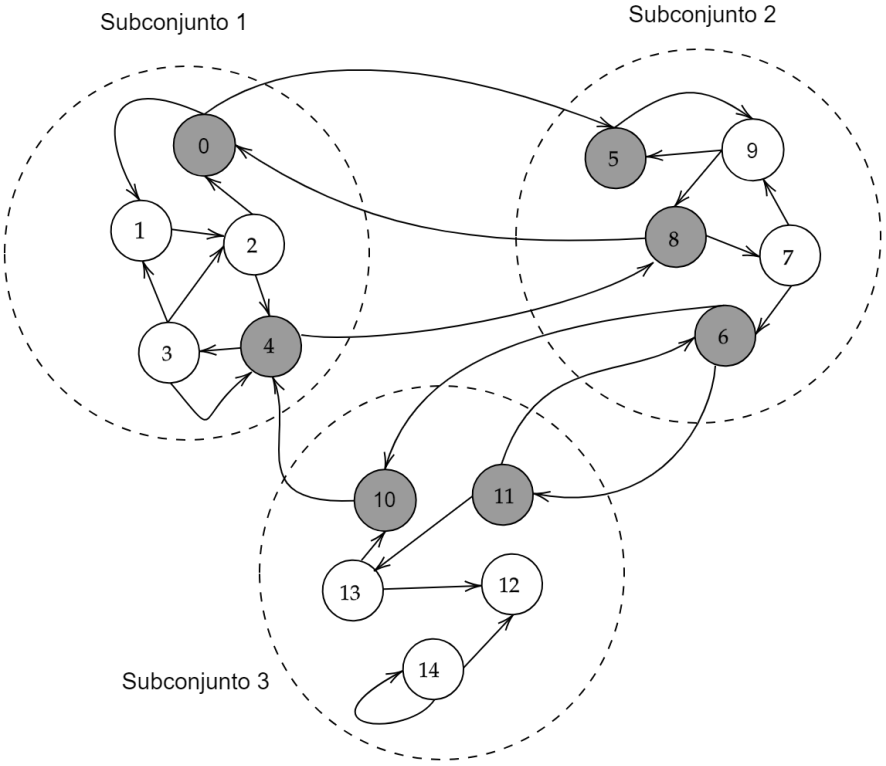

Figura 3.1: **Exemplo de multi-particionamento do espaço de estado de um PDM.**

$I_l = B_l \setminus F_l$ de estados *interiores* da Eq. (3.16). Estes somente podem ser acessados diretamente por estados que pertencem ao mesmo subconjunto B_l. Note, por exemplo, que $I_2 = \{7, 9\}$ só pode ser diretamente acessado a partir dos estados $\{5, 6, 7, 8, 9\}$, todos pertencentes a B_2. Observe que isto garante que uma trajetória semi-regenerativa que começa em um dado subconjunto I_l jamais alcançará outro estado interior de um subconjunto I_m, $m \neq l$, sem antes passar pelo conjunto F. Assim, segue-se que uma trajetória começada no subconjunto F pode

i) continuar em F, ou

ii) visitar apenas um único subconjunto $I_l \in B_l$ antes de retornar ao conjunto F.

Assim, a cardinalidade das trajetórias semi-regenerativas que saem de um estado de fronteira $i \in F : i \in B_l$ (até o retorno a F) é limitada e função da cardinalidade de subconjuntos I_l, e não mais da cardinalidade de F^c, como é o caso das abordagens anteriores de agregação temporal (e.g., CAO *et al.*, 2002, ARRUDA e FRAGOSO, 2015).

Para finalizar a configuração da abordagem, basta aplicar o esquema de particionamento proposto na Definição 1 à abordagem da agregação temporal caracterizada

pela Eq. (3.5). Perceba que o subconjunto F^c em (3.5), conforme definido em (3.16), compreende todos os estados *interiores* do esquema de particionamento. Isto significa que a política $\mathcal{L}_{\text{out}} = d$ na formulação da agregação temporal em (3.5) fixa uma ação $d(i) \in A(i)$ para cada estado *interior* $i \in F^c$, enquanto otimiza no conjunto de estados de *fronteira* F definido em (3.15).

3.2.1 Propriedades do método de particionamento proposto

Esta subseção estabelece importantes propriedades de comunicação dos estados de *fronteira* pertencentes ao subconjunto F e dos estados *interiores* em F^c.

Inicialmente, será mostrado que, utilizando o método de partição proposto, um estado interior de um subconjunto I_l qualquer se conecta diretamente apenas com os estados em $F \cup I_l$. O lema abaixo formaliza uma propriedade importante mencionada na seção anterior com respeito às propriedades do sistema de particionamento proposto:

- que trajetórias semi-regenerativas em um conjunto I_l obrigatoriamente passam por algum estado fronteira em $F \subset S$ antes de visitar um conjunto diferente de estados interiores I_m, $m \neq l$.

Lema 2. *Suponha uma partição $\mathcal{P}(S)$ de acordo com a Definição 1. Desse modo, segue que:*

$$p_{ij}^a = 0,\ \forall i, j \in F^c,\ i \in I_l \text{ e } j \in I_m,\ m \neq l,\ l, m \in \{1, \ldots, , n\}.$$

Demonstração. Seja $i \in I_l$, e seja outro estado $j \in S \setminus B_l$ tal que $p_{ij}^a > 0$ para algum $a \in A(i)$. Então, segue de (3.15) que j pertence a F. Assim, $j \in I_m$ implica que $p_{ij}^a = 0$ se $i \in I_l$, $l \neq m$, e isto conclui a prova. □

O próximo Teorema estabelece uma propriedade geral de comunicação presente no esquema de particionamento proposto, que é fundamental para desenvolver algoritmos novos computacionalmente eficientes.

Teorema 3. *Seja uma partição $\mathcal{P}(S)$ conforme a Definição 1. Além disso, seja um processo controlado X_t, $t \geq 0$, irredutível sob qualquer política viável $\mathcal{L} \in \mathbb{L}$. Então, para todo subconjunto $I_l \subset B_l$, $1 \leq l \leq n$ e para toda política $\mathcal{L} \in \mathbb{L}$ é válido que:*

$$\sum_{j \in F} p_{ij}^{\mathcal{L}(i)} > 0,\ \textit{para algum } i \in I_l,$$

e

$$\sum_{j \in F^c} p_{ij}^{\mathcal{L}(i)} = \sum_{j \in I_l} p_{ij}^{\mathcal{L}(i)},\ \forall i \in I_l.$$

Demonstração. A última igualdade decorre diretamente do Lema 2. Para provar a primeira igualdade por contradição, considere que $\sum_{j \in F} p_{ij}^{\mathcal{L}(i)} = 0$ para todos os estados $i \in I_l$. Então, em vista da última equação, obtém-se:

$$\sum_{j \in F^c} p_{ij}^{\mathcal{L}(i)} = \sum_{j \in I_l} p_{ij}^{\mathcal{L}(i)} = 1, \, \forall i \in I_l.$$

Isto, por sua vez, significa que os estados em I_l formam uma classe de comunicação fechada (BRÉMAUD, 1999) e, portanto, o processo X_t, $t \geq 0$, não é irredutível. Como esse resultado contradiz a hipótese inicial, a primeira equação deve ser verdadeira. □

Cabe ressaltar que o Teorema 3 estabelece duas propriedades importantes do esquema de particionamento proposto:

i) O conjunto de estados de fronteira F é alcançável a partir de qualquer conjunto de estados interiores I_l sob qualquer política $\mathcal{L} \in \mathbb{L}$;

ii) Uma trajetória semi-regenerativa que sai do subconjunto F por meio de um estado $j \in I_l$ tem que obrigatoriamente retornar ao subconjunto F antes de visitar qualquer outro subconjunto I_m, $m \neq l$.

Na sequência, será demonstrado que qualquer estado $i \in F_l$ pode acessar o subconjunto F^c somente por meio de estados $j \in I_l$. Isto significa que um estado de fronteira de um dado subconjunto B_l nunca irá acessar um conjunto de estados interiores I_m de outro subconjunto B_m da partição. Esta propriedade permite que a Eq. (3.14) seja reescrita somente em função do subconjunto de estados interiores I_l para todos os estados de fronteira em F_l, como será explicitado na próxima subseção.

Teorema 4. *Considere uma partição $\mathcal{P}(S)$ tal qual a Definição 1. Então, para cada subconjunto $F_l \subset F$, $1 \leq l \leq n$, segue que:*

$$p_{ij}^{a} = 0, \; \forall j \in F^c \setminus I_l, \;\; a \in A(i), \quad i \in F_l. \tag{3.17}$$

Demonstração. Para provar o resultado por contradição, suponha que exista um $j \in I_m$, $m \neq l$ tal que $p_{ij}^{a} > 0$ para algum $a \in A(i)$. Então, de acordo com (3.15), segue que $j \in F_m$ visto que $j \in B_m$ e j é acessível por um estado $i \in F_l$, $l \neq m$. Desta forma, por definição, $j \notin I_m$ visto que $j \in B_m$ e $I_m := B_m \setminus F_m$ conforme a Eq. (3.16). Isso contradiz a hipótese inicial de que $j \in I_m$ e assim se conclui a prova. □

3.2.2 Trajetórias semi-regenerativas sob o método de particionamento proposto

Nesta subseção, o Lema 1 será revisto à luz do esquema de particionamento $\mathcal{P}(S)$ proposto pela Definição 1, a fim de que a função valor possa ser calculada de maneira distribuída nos diferentes subconjuntos da partição. Este resultado é fundamental para o presente trabalho.

Seja $\mathcal{L}_{\text{out}} = d$ a política aplicada no subconjunto F^c, como na Eq. (3.5), e seja $p_{ij}^d = p_{ij}^{d(i)}$, $i \in F^c$, a probabilidade de transição do estado $i \in F^c$ para o estado $j \in S$. Partindo de qualquer estado $i \in I_l$, segundo a teoria clássica de cadeias de Markov (e.g., BRÉMAUD, 1999), o valor esperado do número de visitas ao estado $j \in I_l$ antes de deixar o subconjunto I_l é o elemento $e_l^d[i,j]$ da matriz E_l^d, dada por:

$$E_l^d = (I - \mathcal{O}_l^d)^{-1}, \quad l = 1, \cdots, n, \tag{3.18}$$

em que $\mathcal{O}_l^d := [p_{ij}^d]$, $i, j \in I_l, l = 1, ..., n$. Em contrapartida, o elemento $a_l^d[i,j]$ da matriz A_l^d definida abaixo denota a probabilidade de uma trajetória de saída partindo de $i \in I_l$ alcançar o estado de *fronteira* $j \in F$ ao deixar I_l. Seja $R_l^d := [p_{ij}^d]$, $i \in I_l$, $j \in F$. A teoria clássica de análise de absorção em cadeias de Markov diz que (BRÉMAUD, 1999):

$$A_l^d = E_l^d \cdot R_l^d, \quad l = 1, \cdots, n. \tag{3.19}$$

Lembrando que X_t, $t \geq 0$, é a trajetória controlada e definindo $\tau = \{\min t > 0 : X_t \in F\}$, obtém-se:

$$a_l^d[i,j] = P(X_\tau = j | X_0 = i, \mathcal{L}_{\text{out}} = d).$$

Definido na Eq. (3.6), o custo acumulado de uma trajetória que parte do estado de *fronteira* $i \in F_l$ sob uma ação $a \in A(i)$ e chega ao subconjunto F pode ser reescrito como:

$$h_f(i,a) = f(i,a) + \sum_{k \in I_l} p_{ik}^a \sum_{j \in I_l} e_l^d[k,j] f(j, d(j)). \tag{3.20}$$

O somatório da Eq. (3.20) corresponde ao custo total incorrido dentro do subconjunto I_l antes de chegar ao próximo estado de *fronteira*. O somatório no conjunto I_l é suficiente devido ao Teorema 4, que demonstra que uma trajetória semi-regenerativa partindo de F_l somente pode acessar estados interiores em I_l. Analogamente, o comprimento de uma trajetória que parte de um estado de *fronteira* $i \in F_l$ e chega ao

próximo estado de *fronteira* é dado por:

$$h_1(i,a) = 1 + \sum_{k \in I_l} p_{ik}^a \sum_{j \in I_l} e_l^d[k,j]. \tag{3.21}$$

Perceba que a Eq. (3.21) é uma mera aplicação de (3.20) com $f(i,a) = 1, \forall (i,a) \in S \times A$. Finalmente, a probabilidade de transição do processo semi-Markov embutido no subconjunto de estados de *fronteira* F é dada por:

$$\tilde{p}_{ij}^a = p_{ij}^a + \sum_{k \in I_l} p_{ik}^a a_l^d[k,j],\ i \in F_l,\ j \in F. \tag{3.22}$$

Considerando o esquema de particionamento proposto, é possível reescrever os resultados do Lema 1 da seguinte maneira:

Lema 3. *Seja uma partição $\mathcal{P}(S)$ conforme a Definição 1. Considere também que o par (V_d^*, η_d^*) satisfaz a Eq.* (3.9) *para o problema gerado pela agregação temporal* (3.5)*, com $\mathcal{L}_{out} = d$, utilizando o esquema de particionamento proposto. Além disso, seja a política $\mathcal{L}_d^*$ uma política ótima que satisfaz* (3.10)*. Então, a função valor do PDM original sob a política $\mathcal{L}_d^*$ satisfaz:*

$$V^{\mathcal{L}_d^*}(i) = \begin{cases} V_d^*(i), & \text{se } i \in F, \\ \displaystyle\sum_{k \in I_l} e_l^d[i,k]\,(\,f(k,d(k)) - \eta_d^*\,) + \sum_{j \in F} a_l^d[i,j] V_d^*(j), & \text{se } i \in F^c. \end{cases} \tag{3.23}$$

Demonstração. A primeira expressão à direita da Eq. (3.23) é uma aplicação direta do Lema 1. Seja $i \in I_l$, então de acordo com o Lema 1

$$V^{\mathcal{L}_d^*}(i) = E\left\{ \sum_{t=0}^{\tau_1 - 1} (f(X_t, \mathcal{L}_d^*(X_t)) - \eta_d^*) + V_d^*(X_{\tau_1}) \right\},\ \tau_1 = \min\{t > 0 : X_t \in F\}, \tag{3.24}$$

visto que $i \in I_l$ implica que $i \in F^c$. Agora, se $p_{ij}^{d(i)} > 0$ para algum $j \notin I_l$, então, por definição $j \in F_m$ para algum $m \in \{1, \ldots, n\}$ e, portanto, $j \in F$. Consequentemente, decorre também que $\tau_1 = \min\{t > 0 : X_t \notin I_l\}$ visto que qualquer estado em I_l somente pode acessar imediatamente estados em I_l ou estados em F. Portanto, a teoria clássica sobre cadeias de Markov diz que:

$$E\left[\sum_{t=0}^{\tau_1 - 1} (f(X_t, \mathcal{L}_d^*(X_t)) - \eta_d^*) \right] = \sum_{j \in I_l} e_l^d[i,j]\,(\,f(j,d(j)) - \eta_d^*\,),$$

em que $E_l^d = e_l^d[i,j]$, $k,j \in I_l$ é definida em (3.18). A equação anterior corresponde ao custo acumulado de uma trajetória desde sua partida do estado $i \in I_l$ até deixar o subconjunto I_l, utilizando a função custo $(\,f(i,d(i)) - \eta_d^*\,)$. Ainda segundo a teoria

clássica sobre análise de absorção em cadeias de Markov:

$$E\left(V(X_{\tau_1})\right) = \sum_{j \in F} a_l^d[i,j] V_d^*(j),$$

em que $A_l^d = a_d^d[i,j]$, $i \in I_l$, $j \in F$ é definida em (3.19). A equação anterior denota a probabilidade que uma trajetória partindo de $i \in I_l$ alcance o estado $j \in F$ imediatamente ao deixar o subconjunto de estados interiores I_l.

Para concluir a prova, basta substituir o último par de expressões na Eq. (3.24), que resulta na expressão da função valor para $i \in I_l$ em (3.23). □

O resultado do Lema 3 ilustra o poder do método de particionamento proposto, visto que demonstra que para determinar a função valor dos estados em qualquer subconjunto I_l, é necessário apenas armazenar a função valor dos estados em F, que pode ser obtida a partir da cadeia semi-Markov embutida em F - Eq. (3.5).

Observação 1. *Observe que em* (3.20)-(3.22)*, a inversão de uma matriz com dimensão* $|F^c|$ *na Eq.* (3.14) *é substituída pela inversão de uma matriz com dimensão* $|F_l|$ *para cada subconjunto* $F_l, 1 \leq l \leq n$*, podendo utilizar um esquema de computação distribuída que aplica a Eq.* (3.18)*. Portanto, o esforço computacional de* $O(|F^c|^3)$ *no subconjunto original é substituído por um esforço computacional de* $O(\sum_{l=1}^{n} |I_l|^3)$*. Visto que* $|F^c| = \sum_{l=1}^{n} |I_l|$*, o método proposto possui um potencial de reduzir consideravelmente o esforço computacional no caso de* $|I_l| << |F^c| \approx |S|$*.*

Observação 2. *A ideia de dividir o espaço de estados em estados de* fronteira *e estados* interiores *torna possível resolver o sistema a partir de uma análise de absorção (*(3.20)-(3.22)*) em função dos tempos de estadia dentro de cada subconjunto* I_l *individualmente. Isto se deve às propriedades de comunicação geradas pelo esquema de partição resumidas nos Teoremas 3 e 4.*

3.3 Algoritmos com agregação temporal distribuída para Processos de Decisão de Markov com custo médio

Nesta seção, *o esquema de particionamento* $\mathcal{P}(S)$ da Definição 1 será aplicado da seguinte maneira:

- Primeiro, será proposto um algoritmo que utilizará o esquema de computação distribuída na Eq. (3.20)-(3.22) para resolver a formulação de agregação temporal. Este algoritmo também utilizará o Lema 3 para implementar uma

etapa de melhoria de política distribuída, que itera em um único subconjunto I_l, $1 \leq l \leq n$, por vez. Cada iteração do algoritmo proposto resolve a Eq. (3.9) para uma formulação diferente da Eq. (3.5). O procedimento é descrito no Algoritmo 5 abaixo.

- Na sequência, serão exploradas algumas alternativas de melhoria do Algoritmo 5. Em decorrência disso, foi proposta uma generalização do Algoritmo 5 que pode convergir mais rapidamente ao utilizar, por exemplo, estratégias de atualização de política mais eficientes na etapa de melhoria de política. O pseudocódigo do algoritmo generalizado é apresentado no Algoritmo 6. É possível que outras melhorias sejam implementadas a partir destas ideias, porém mantendo a essência do Algoritmo 5.

Algoritmo 5 Algoritmo de Agregação Temporal de Duas Fases Distribuído (AT2FD)

1: Particione o espaço de estado em n subconjuntos disjuntos $\{B_1, \cdots, B_n\}$
2: Para cada $l \in \{1, 2, \ldots, n\}$, $F_l = \{j \in B_l : \exists\, i \notin B_l \text{ e} \exists\, a \in A(i) \text{ tal que } p_{ij}^a > 0\}$, $I_l = B_l \setminus F_l$.
3: Escolha uma política inicial arbitrária $\mathcal{L}_{int} = d_0$, $d_0 \in \mathbb{L}_{int}$ e uma tolerância ϵ
4: $k \leftarrow 0$, $er \leftarrow \infty$, $\eta_0 \leftarrow \infty$, $\bar{d}_0 = d_0$, $\eta_k^l = \infty$, $1 \leq l \leq n$
5: **while** $|er| > \epsilon$ **do**
6: $\quad d_{k+1} = \bar{d}_k$, $\eta_{k+1} \leftarrow \eta_k^n$, $k \leftarrow k+1$
7: $\quad$ **for** $l = 1$ to n **do**
8: $\quad\quad$ Utilize a Eq. (3.9) para encontrar o par (V_d^*, η_d^*) que resolve o problema agregado (3.5), com $d = \bar{d}_k$. Faça $\eta_k^l = \eta_d^*$ e $V_k^l(i) = V_d^*(i)$, $\forall i \in F$
9: $\quad\quad$ Utilize o Lema 3 para encontrar a função valor $V_k^l(i) = V^{\mathcal{L}_{\bar{d}_k}}(i)$, $i \in I_l$ do PDM original
10: $\quad\quad$ Execute a etapa de melhoria de política:

$$\bar{d}_k(i) = \arg\min_{a \in A(i)} \left\{ f(i,a) + \sum_{j \in I_l \cup F} p_{ij}^a V_k^l(i) \right\}, \forall i \in I_l \tag{3.25}$$

mantendo $\bar{d}_k(i) = d_k(i)$ sempre que possível em (3.25)
11: $\quad\quad$ Utilize a nova melhor política em I_l para gerar um novo problema agregado com $\mathcal{L}_{\text{out}} = \bar{d}_k$. Isto significa atualizar (3.20)-(3.22) para todos os pares estado-ação (i,a), $i \in F_l$, $a \in A(i)$, tal que $p_{ij}^a > 0$, para algum $j \in I_l$, mantendo $h_f(i,a)$, $h_1(i,a)$, $\tilde{p}_{ij}^a$ inalterado para todos os pares estado-ação restantes, i.e., para todos os pares $(i,a) : i \notin F_l$.
12: $\quad$ **end for**
13: $\quad$ $er \leftarrow \eta_k^n - \eta_k$
14: **end while**

Agora, será demonstrado que o Algoritmo 5 converge para a mesma solução do problema original - Eq. (3.4). Para tanto, será preciso provar alguns resultados intermediários na sequência. Primeiro, será apresentado o Teorema 5, que é uma reformulação de CAO (1998, Lemma 2).

Teorema 5. *Sejam os pares* $(V^{\mathcal{L}}, \eta^{\mathcal{L}})$ *e* $(V^{\mathcal{L}'}, \eta^{\mathcal{L}'})$ *soluções para a equação de Poisson* (3.2)*. Se o valor da equação de Poisson para a política* $\mathcal{L}$ *for menor ou igual ao valor da equação de Poisson para a política* $\mathcal{L}'$*, então* $\eta^{\mathcal{L}} \leq \eta^{\mathcal{L}'}$*.*

Lema 4. *Seja* η_k^l, $1 \leq l \leq n$ *o valor obtido no passo 8 do Algoritmo 5, então*

$$\eta_k^{l+1} \leq \eta_k^l,\ 1 \leq l < n.$$

Além disso, se $\eta_k^{l+1} < \eta_k^l$ *então* $\bar{d}_k(i) \neq d_k(i)$*, para algum* $i \in I_l$*.*

Demonstração. A prova decorre diretamente do CAO (1998, Lemma 2), que implica que se $\bar{d}_k(i) \neq d_k(i)$ para algum $i \in I_l$ na etapa de melhoria de política - Eq. (3.25), então $\eta_k^{l+1} < \eta_k^l$, as $\eta_k^{l+1} = \eta_d^*$ in (3.5) com $d = \bar{d}_k$. Por outro lado, se $\bar{d}_k(i) = d_k(i)$, $\forall i \in I_l$, então é óbvio que $\eta_k^{l+1} = \eta_k^l$ visto que em ambos os casos o mesmo problema gerado pela agregação temporal é resolvido. □

Lema 5. *Seja* η_k *o valor obtido no passo 6 do Algoritmo 5, então:*

$$\eta_{k+1} \leq \eta_k,\ k \geq 0. \tag{3.26}$$

Demonstração. Segue diretamente do Lema 4 que:

$$\eta_{k+1} = \eta_k^n = \min_{1 \leq l \leq n} \eta_k^l \leq \eta_k^1.$$

Para concluir a prova, basta perceber que o passo 6 do Algoritmo 5 faz com que $\bar{d}_k = \bar{d}_{k-1}$ na iteração k para $l = 1$, em que $\bar{d}_{k-1}$ é a política em F^c seguindo a última melhoria de política da iteração $k-1$; i.e., quando $l = n$ no *for* (passo 7 do algoritmo). Portanto, é possível aplicar novamente o (CAO, 1998, Lemma 2) para ver que:

$$\eta_k \leq \eta_{k-1}^n,$$

e a prova é concluída. □

Teorema 6. *Algoritmo 5 termina em uma política ótima para o problema* (3.4)*.*

Demonstração. Lema 5 estabelece que no Algoritmo 5 o custo médio de longo prazo decresce monotonicamente. Isto implica que o algoritmo converge em um tempo finito, dado que o espaço de estados e o espaço de ações são finitos. Deste modo, existe um número finito de política possíveis de serem escolhidas e o custo médio não pode ser melhorado para sempre.

Agora, seja d_k a política no subconjunto F^c ao convergir. Suponha que $d_k \neq \mathcal{L}_{\text{out}}^*$, em que

$$\mathcal{L}_{\text{out}}^*(i) = \mathcal{L}^*(i),\ \forall i \in F^c,$$

para alguma política $\mathcal{L}^*$ que resolva (3.4). Então, segue que $d_{k+1}(i) = \bar{d}_k(i) \neq d_k(i)$ para pelo menos um $i \in F^c$ em (3.25), o que contradiz a hipótese inicial, visto que convergir implica que $d_{k+1} = d_k$. Portanto, ao convergir $d_k = \mathcal{L}_{\text{out}}^*$. Como resultado, o Teorema 1 implica que η_k deve ser igual a solução de (3.4), e a prova está concluída. □

3.3.1 Generalização dos algoritmo com agregação temporal distribuída para Processos de Decisão de Markov com custo médio

Nesta seção, será apresentada uma generalização do Algoritmo 5. Conforme pode ser visualizado no pseudocódigo 6, o novo algoritmo proposto nesta seção possui uma estrutura mais flexível em relação à etapa de melhoria de política aplicada aos múltiplos subconjuntos da partição. Contudo, o Algoritmo 6 ainda guarda a essência do Algoritmo 5, no sentido de preservar a monotonicidade na aplicação da Eq. (3.25) em cada iteração, o que garante a convergência, conforme demonstrado no Teorema 6.

Algoritmo 6 Algoritmo de Agregação Temporal de Duas Fases Distribuído e Generalizado

1: Particione o espaço de estados em n subconjuntos disjuntos $\{B_1, \cdots, B_n\}$
2: Para cada $l \in \{1, 2, \ldots, n\}$, $F_l = \{j \in B_l : \exists\, i \notin B_l \text{ e} \exists\, a \in A(i) \text{ tal que } p_{ij}^a > 0\}$, $I_l = B_l \setminus F_l$.
3: Escolha uma política inicial arbitrária $\mathcal{L}_{int} = d_0$, $d_0 \in \mathbb{L}_{int}$ e uma tolerância ϵ
4: $k \leftarrow 0$, $er \leftarrow \infty$, $\eta_0 \leftarrow \infty$, $\bar{d}_0 = d_0$, $\eta_k^l = \infty$, $1 \leq l \leq n$, $\text{Best}_\eta \leftarrow \infty$
5: **while** $|er| > \epsilon$ **do**
6: $d_{k+1} = \bar{d}_k$, $\eta_{k+1} \leftarrow \text{Best}_\eta$, $k \leftarrow k + 1$
7: Sweep $= \emptyset$
8: **while** Sweep $\neq \{1, \ldots, n\}$ **do**
9: Selecione l do conjunto $\{1, \ldots, n\}$ e faça Sweep = Sweep $\cup\, l$
10: Utilize a Eq. (3.9) para encontrar o par (V_d^*, η_d^*) que resolve o problema agregado (3.5), com $d = \bar{d}_k$. Faça $\eta_k^l = \eta_d^*$ e $V_k^l(i) = V_d^*(i)$, $\forall i \in F$
11: $\text{Best}_\eta = \min\{\text{Best}_\eta, \eta_k^l\}$
12: Utilize o Lema 3 para encontrar a função valor $V_k^l(i) = V^{\mathcal{L}_{\bar{d}_k}}(i)$, $i \in I_l$ do PDM original.
13: Para cada $i \in I_l$ tal que

$$f(i, \bar{d}_k(i)) + \sum_{j \in I_l \cup F} p_{ij}^{\bar{d}_k(i)} V_k^l(i), \forall i \in I_l > \arg \min_{a \in A(i)} \left\{ f(i,a) + \sum_{j \in I_l \cup F} p_{ij}^a V_k^l(i) \right\}$$

selecione uma ação $\bar{d}_k(i) = a$, $a \in A(i)$ tal que

$$f(i,a) + \sum_{j \in I_l \cup F} p_{ij}^a V_k^l(i) < f(i, \bar{d}_k(i)) + \sum_{j \in I_l \cup F} p_{ij}^{\bar{d}_k(i)} V_k^l(i)$$

14: Utilize a nova melhor política em I_l para gerar o novo problema agregado com $\mathcal{L}_{\text{out}} = \bar{d}_k$. Isto significa atualizar (3.20)-(3.22) para todos os pares estado-ação (i,a), $i \in F_l$, $a \in A(i)$, mantendo $h_f(i,a)$, $h_1(i,a)$, $\tilde{p}_{ij}^a$ inalterado para todos os pares estado-ação restantes, i.e., para todos os pares $(i,a) : i \notin F_l$.
15: **end while**
16: $er \leftarrow \eta_k^n - \eta_k$
17: **end while**

Na sequência, será provada a convergência do Algoritmo 6 e serão discutidas suas propriedades e diferenças em relação ao Algoritmo 5.

Teorema 7. *Algoritmo 6 termina em uma política ótima para o problema* (3.4).

Demonstração. Os argumentos são similares àqueles do Teorema 6. Uma vez que a etapa de melhoria de política - Passo 13 do Algoritmo 6 seleciona uma política melhor sempre que possível, decorre do (CAO, 1998, Lemma 2) que $\eta_k^l \leq \text{Best}_\eta$ a

cada iteração do Passo 13. Portanto:

$$\eta_{k+1} \leq \eta_k,\ k \geq 0.$$

Isto implica que o algoritmo converge em tempo finito dado que o espaço de estados e o espaço de ações são finitos. Portanto, existe um número finito de política possíveis de serem escolhidas e o custo médio de longo prazo não pode melhorar para sempre.

Para provar que o algoritmo converge para a política ótima, basta verificar que o Passo 8 percorre, necessariamente, todo subconjunto I_l, $l = \{1, \ldots, n\}$ e, portanto, ao convergir, pelo menos uma etapa de melhoria de política terá sido aplicada para todos os estados $i \in F^c$, sem mudar a solução inicial. Agora, seja d_k a política do subconjunto F^c ao convergir. Suponha que $d_k \neq \mathcal{L}^*_{\text{out}}$, em que

$$\mathcal{L}^*_{\text{out}}(i) = \mathcal{L}^*(i),\ \forall i \in F^c,$$

para uma política $\mathcal{L}^*$ que resolva (3.4). Então $d_{k+1}(i) = \bar{d}_k(i) \neq d_k(i)$ para pelo menos um $i \in F^c$ em (3.25), o que contradiz a hipótese inicial, uma vez que convergir implica que $d_{k+1} = d_k$. Portanto, ao convergir, obtém-se $d_k = \mathcal{L}_{\text{out}}$. Consequentemente, Teorema 1 implica que η_k precisa ser igual à solução de (3.4), e a prova está concluída. □

Observação 3. *Uma diferença importante entre os Algoritmos 5 e 6 é que, enquanto o Algoritmo 5 realiza uma única etapa de busca local em cada subconjunto I_l, $1 \leq l \leq n$ por iteração, o Algoritmo 6 permite procedimentos de busca local mais gerais, que podem explorar com mais frequência regiões do espaço de estados mais promissoras. É possível, por exemplo, realizar buscas locais mais frequentes em subconjuntos I_l que sejam visitados mais frequentemente ao seguir uma política, visto que estes estados possivelmente possuem um impacto maior no custo médio de longo prazo. Outra possibilidade é aplicar estratégias heurísticas como as usadas em aprendizado por reforço, tais como varredura priorizada (MOORE e ATKESON, 1993, KIM, 2022). É válido, contudo, observar que a estratégia de busca no Passo 7 do Algoritmo 5 é um caso particular de uma classe de estratégias do Passo 8 do Algoritmo 6.*

Outra importante diferença é que a etapa de melhoria de política no Passo 13 do Algoritmo 6 é mais geral e permite uma configuração mais flexível. Isso ocorre porque ele se concentra no requisito de que cada nova política melhore a atual, independentemente de como essa nova política é selecionada. É válido destacar que a melhoria de política do Passo 8 do Algoritmo 5 é um caso particular da classe de estratégias do Passo 13 do Algoritmo 6.

Capítulo 4

Aplicação a um problema de produção e gestão de estoque

A fim de ilustrar a aplicabilidade dos algoritmos propostos, esta seção replica o problema de produção e gestão de estoques explicitado em (ARRUDA e FRAGOSO, 2015). O problema consiste em uma única máquina que pode produzir apenas um único produto por vez dentre três produtos possíveis. Contudo, a máquina pode trocar de produto sem nenhum custo. Sempre que uma nova demanda chega ou a produção de um produto termina, o responsável pode decidir continuar a fabricação do mesmo produto, fabricar um novo produto ou parar completamente a produção. Se a produção estiver interrompida, o responsável decidirá se manterá a produção interrompida ou se retomará a produção de algum produto.

Seja o nível de estoque de cada produto uma variável inteira no intervalo $\{-100, \ldots, 25\}$. Isto significa que a quantidade máxima de déficit de estoque permitida para cada produto é de 100 unidades e nenhum pedido adicional é aceito quando este nível de déficit for atingido. Por outro lado, a quantidade máxima de produto em estoque permitida é de 25 unidades. Portanto, o espaço de estados S é composto de $126^3 \approx 2 \cdot 10^6$ elementos, cada um correspondendo a uma possível combinação da quantidade de falta ou estoque de cada um dos três produtos.

Suponha que as demandas dos produtos 1, 2 e 3 seguem uma distribuição de Poisson com taxas 3, 2 e 1, respectivamente. Por outro lado, os tempos de produção seguem uma distribuição exponencial com taxa $\mu = 8$. Considere também que cada pedido é composto por um único item e a fábrica pode continuar a produção mesmo que não haja nenhum consumidor esperando. O custo de estocagem ou falta é dado por:

$$f(x) = |x_1| + 2|x_2| + 3|x_3|,$$

em que x_1, x_2 e x_3 correspondem à quantidade em estoque ou falta dos produtos 1, 2 e 3, respectivamente. O objetivo é encontrar a política $\mathcal{L}^*$ que minimiza o custo

médio e satisfaça (3.4). Para executar os experimentos, foi utilizado um notebook com processador Intel Core i3 com 2.30 GHz, memória RAM de 4 GB, sistema operacional Windows 10 e a linguagem de programação C++ (PRATA, 2001). A tolerância utilizada foi de 0.001, isto é, $\epsilon = 0.001$ no Passo 5 dos Algoritmos 5 e 6.

4.1 Método de Particionamento

Para resolver o problema, foi aplicado um método de particionamento em que o número de agrupamentos é igual ao número de produtos, sendo os subconjuntos $\{C_1, C_2, C_3\}$ definidos como:

$$\begin{aligned} C_1 &= x \in S : x_1 \leq x_2, \text{ e } x_1 \leq x_3; \\ C_2 &= x \in S : x_1 > x_2, \text{ e } x_2 \leq x_3; \\ C_3 &= S \setminus \{C_1 \cup C_2\}. \end{aligned} \tag{4.1}$$

A lógica é dividir o espaço de estados conforme o produto com o menor nível de estoque. Em caso de empate, isto é, se em um determinado estado, dois ou mais produtos tiverem o mesmo nível de estoque, o estado pertencerá ao subconjunto com o menor índice.

4.2 Resultados

Inicialmente, o problema foi resolvido utilizando os clássicos algoritmos Iteração de Política (IP) e Iteração de Valor Relativo (IVR), bem como o algoritmo de Agregação Temporal de Duas Fases (AT2F) (ARRUDA e FRAGOSO, 2015). Estes três algoritmos atuarão como referências para os algoritmos propostos neste trabalho. Para o algoritmo Agregação Temporal em Duas Fases, o subconjunto F foi composto pelos estados cuja falta ou estoque dos 3 produtos estivessem no intervalo $[-10, 9]$, isto é, $F := \{x \in S : -10 \leq x_1 \leq 9, -10 \leq x_2 \leq 9, -10 \leq x_3 \leq 9\}$. O tempo de convergência e o número de iterações de cada algoritmo estão explicitados na Tabela 4.1.

Tabela 4.1: Resultado computacional

Algoritmo	**Custo médio**	**Tempo**	**It.**
IP	3,40952	19.993 s	14
IVR	3,40952	5.972 s	1.290
AT2F	3,40955	8.199 s	5

A Tabela 4.1 mostra que o Iteração de Valor Relativo foi o algoritmo mais rápido dentre os três algoritmos mais clássicos. Portanto, este será utilizado junto com o

algoritmo de Agregação Temporal de Duas Fases como referência para efeito de comparação com os algoritmos novos propostos nesta Tese.

4.2.1 Resultados dos Algoritmos Novos Propostos

Nesta seção, serão discutidos os resultados computacionais obtidos ao aplicar os algoritmos novos propostos nesta Tese utilizando o método de particionamento descrito na Seção 4.1 para resolver o problema de produção e gestão de estoque anteriormente descrito.

A partir das Tabelas 4.1 e 4.2, é possível verificar que o algoritmo Agregação Temporal de Duas Fases Distribuído (AT2FD) - Algoritmo 5 - foi mais lento que o algoritmo Agregação Temporal de Duas Fases. De fato, o algoritmo Agregação Temporal Duas Fases Distribuído demorou $10.815s$ para convergir, enquanto que o algoritmo Agregação Temporal Duas Fases levou $8.199s$. Uma possível explicação para este resultado é que o número necessário de avaliações da cadeia embutida no Passo 8 do Algoritmo 5 é maior. Isto ocorre, pois sempre que a política dos estados de um subconjunto l for atualizada no Passo 7, será necessário resolver novamente a cadeia embutida.

Para amenizar este efeito, foi fixado em 25 o número máximo de iterações no Passo 8. Esta nova versão do algoritmo foi denominada *Agregação Temporal Duas Fases Distribuído e Modificado* (AT2FDM). Conforme pode ser visualizado na Tabela 4.2, limitar o número de iterações em 25 na avaliação da cadeia embutida teve um impacto significativo no tempo para a convergência, reduzindo-o para apenas 470 segundos.

Tabela 4.2: Resultado computacional

Algoritmo	**Custo médio**	**Tempo**	**It.**
AT2F Distribuído (AT2FD)	3,40953	10.815 s	4
AT2F Distribuído e Modificado (AT2FDM)	3,40963	470 s	5
AT2F Distribuído, Priorizado e Modificado (AT2FDPM)	3,40963	461 s	5
AT2F Distribuído, Focalizado e Modificado (AT2FDFM)	3,40977	378 s	4
AT2F Distribuído, Priorizado, Focalizado e Modificado (AT2FDPFM)	3,40977	331 s	4

Outro aspecto importante a ser avaliado é se o tempo até a convergência é influenciado pela ordem em que os subconjuntos são atualizados. Em face do exposto, foi implementada outra variação do algoritmo 6, denominada *Agregação Temporal Duas Fases Distribuído, Priorizado e Modificado* (AT2FDPM). O termo *priorizado* indica que uma regra de priorização determina qual subconjunto terá a política de seus estados atualizada primeiro. Neste experimento, a ordem de priorização foi estabelecida de acordo com o método de particionamento proposto na Seção 4.1. Desse modo, o subconjunto mais prioritário é o subconjunto C_1, pois o produto 1 é o produto com maior taxa de demanda. O segundo subconjunto mais prioritário é o

subconjunto C_3 visto que o produto 3 é o produto com maior custo de estocagem e falta. O último subconjunto é o subconjunto C_2 - ver Eq. (4.1) para a definição dos subconjuntos. Contudo, conforme exposto na Tabela 4.2, a ordem em que os subconjuntos são atualizados tem um efeito pequeno no tempo para convergência, que decresceu para apenas $461s$. Uma possível explicação é que a diferença entre as taxas de demanda e os custos de falta e estocagem dos 3 produtos não é tão significativa a ponto da priorização de subconjuntos reduzir muito o tempo computacional.

Por último, foi testado se o tempo até a convergência diminuiria com a inclusão em F dos estados que se espera que sejam os mais visitados. A partir daí, foi elaborado um novo algoritmo denominado *Agregação Temporal Duas Fases Distribuído, Focalizado e Modificado* (AT2FDFM). O termo *focalizado* significa que os estados mais próximos do nível de estocagem zero foram adicionados ao subconjunto F, visto que espera-se que esses estados sejam visitados mais frequentemente. Assim como no algoritmo de Agregação Temporal de Duas Fases tradicional, foram adicionados ao subconjunto F os estados cujos estoques de todos os produtos estivessem no intervalo $[-10, 9]$, i.e, $F := \{x \in S : -10 \leq x_1 \leq 9, -10 \leq x_2 \leq 9, -10 \leq x_3 \leq 9\}$. Conforme pode ser visualizado na Tabela 4.2, adicionar estes estados ao subconjunto F levou a uma redução no tempo de convergência, alcançando 378 segundos.

Ainda de acordo com a Tabela, combinar as abordagens *focalizada* e *priorizada* em um mesmo algoritmo denominado *Agregação Temporal Duas Fases Distribuído, Priorizado, Focalizado e Modificado* (AT2FDPFM) gerou o melhor tempo computacional, convergindo em apenas 331 segundos, aproximadamente 4,0% do tempo de convergência do algoritmo Agregação Temporal Duas Fases (5,5% do tempo de convergência do Iteração de Valor Relativo e 1,7% do tempo de convergência do Iteração de Política, conforme a Tabela 4.1).

4.2.2 Outros Métodos de Particionamento Testados

Nesta seção, serão apresentados os resultados computacionais obtidos ao aplicar os algoritmos novos propostos nesta Tese utilizando outros dois métodos de particionamento para resolver o mesmo problema de produção e gestão de estoque anteriormente descrito. Primeiramente, serão apresentados os resultados obtidos utilizando a soma absoluta dos estoques dos três produtos.

Soma Absoluta dos Estoques

Este método de particionamento consiste em dividir o espaço de estados em 3 subconjuntos utilizando a soma absoluta dos estoques dos três produtos. Conforme detalhado anteriormente, o nível de estoque de cada produto é uma variável inteira no intervalo $\{-100, \ldots, 25\}$ de modo que a soma absoluta dos estoques dos produtos

varia de 0 a 300. Sendo $Z = |x_1| + |x_2| + |x_3|$, o espaço de estados foi dividido da seguinte maneira:

$$\begin{aligned} C_1 &= x \in S : 0 \leq Z < 100; \\ C_2 &= x \in S : 100 \leq Z < 200; \\ C_3 &= S \setminus \{C_1 \cup C_2\}. \end{aligned} \tag{4.2}$$

Os resultados computacionais obtidos ao aplicar os algoritmos novos propostos utilizando o método de particionamento proposto nesta seção pode ser visualizado na Tabela 4.3.

Tabela 4.3: Resultado computacional

Algoritmo	**Custo médio**	**Tempo**	**It.**
AT2F Distribuído (AT2FD)	3,40952	29.720 s	13
AT2F Distribuído e Modificado (AT2FDM)	3,40948	744 s	8
AT2F Distribuído, Priorizado e Modificado (AT2FDPM)	3,40979	324 s	4
AT2F Distribuído, Focalizado e Modificado (AT2FDFM)	3,40977	435 s	4
AT2F Distribuído, Priorizado, Focalizado e Modificado (AT2FDPFM)	3,40977	423 s	4

A partir das Tabelas 4.1, 4.2 e 4.3, é possível verificar que, utilizando o método da soma absoluta dos estoques, o algoritmo AT2FD ficou ainda mais lento que o algoritmo Agregação Temporal Duas Fases. De fato, o algoritmo AT2FD demorou $29.720s$ para convergir, enquanto que o algoritmo Agregação Temporal Duas Fases levou $8.199s$. Conforme já explicado na seção anterior, uma possível explicação para este resultado é que o número necessário de avaliações da cadeia embutida no Passo 8 do Algoritmo 5 é maior. Isto ocorre uma vez que sempre que a política dos estados de um subconjunto l for atualizada no Passo 7, será necessário resolver novamente a cadeia embutida.

Adicionalmente, repare que o algoritmo AT2FD desempenhou melhor com o particionamento do menor produto em estoque. Uma possível explicação para este melhor desempenho computacional é o fato de que o particionamento do menor produto em estoque gerar subconjuntos com cardinalidades mais próximas, conforme pode ser visualizado na Tabela 4.4.

Tabela 4.4: Cardinalidade dos Subconjuntos

Subconjuntos	**Menor Produto em Estoque**	**Soma Absoluta dos Estoques**
C_1	674.751	615.125
C_2	666.750	1.198.575
C_3	658.875	186.676

Em contrapartida, o algoritmo AT2FDM, mesmo utilizando o particionamento da soma absoluta dos estoques, convergiu em apenas 744 segundos. Este resultado mostra mais uma vez que limitar em 25 o número máximo de iterações no Passo

8 contribuiu significativamente para redução do tempo para a convergência. Contudo, vale lembrar que o mesmo algoritmo utilizando o particionamento do menor produto em estoque convergiu ainda mais rápido (470s). Mais uma vez, o fato deste particionamento gerar subconjuntos com cardinalidades semelhantes pode ser uma explicação para o algoritmo AT2FDM ter apresentado um melhor desempenho computacional com este particionamento.

Com relação ao algoritmo AT2FDPM, a ordem de priorização adotada foi a seguinte: as políticas dos subconjuntos foram atualizadas em ordem crescente da soma absoluta dos estoques dos três produtos. Desse modo, a ordem de priorização foi C_1, C_2 e C_3. Comparando as Tabelas 4.2 e 4.3, vemos que com o método de particionamento da soma absoluta dos estoque, a ordem em que os subconjuntos são atualizados teve um efeito considerável no tempo para convergência, que decresceu para $324s$. Além disso, o algoritmo AT2FDPM ficou ainda mais rápido utilizando o particionamento da soma absoluta dos estoques. Uma provável explicação é a composição dos subconjuntos. Perceba que no método de particionamento do menor produto em estoque, os estados com estoques próximos de zero estão dispersos entre os três subconjuntos. Por outro lado, no método de particionamento da soma absoluta dos estoques, os estados próximos de zero estão concentrados no mesmo subconjunto C_1, que é o primeiro subconjunto na ordem de priorização. Como os estados mais próximos de zero tendem a ser os mais visitados, atualizar a política destes estados primeiro contribui para acelerar o algoritmo.

Por último, vale destacar que, utilizando o método da soma absoluta dos estoques, o algoritmo AT2FDPFM apresentou um desempenho pior do que o algoritmo AT2FDPM. Este resultado foi diferente quando utilizado o método de particionamento do menor produto em estoque. Uma provável explicação é que no algoritmo AT2FDPFM, além dos estados de fronteira, são adicionados à cadeia de Markov embutida, os estados cujos estoques de todos os produtos estejam no intervalo $[-10, 9]$. Deste modo, a cadeia de Markov embutida gerada pelo algoritmo AT2FDPFM possui uma cardinalidade maior do que a cadeia de Markov embutida gerada pelo algoritmo AT2FDPM. Assim, é possível que esta quantidade adicional no número de estados da cadeia embutida tenha anulado os ganhos obtidos ao priorizar os subconjuntos, até porque, os estados acrescentados à cadeia embutida já fazem parte do subconjunto prioritário.

Soma dos Estoques

Este método de particionamento consiste em dividir o espaço de estados em 3 subconjuntos utilizando a soma dos estoques dos três produtos. Conforme detalhado anteriormente, o nível de estoque de cada produto é uma variável inteira no intervalo $\{-100, \ldots, 25\}$ de modo que a soma dos estoques dos produtos varia de -300 a 75.

Sendo $Z = x_1 + x_2 + x_3$, o espaço de estados foi dividido da seguinte maneira:

$$\begin{aligned} C_1 &= x \in S : -300 \leq Z < -175; \\ C_2 &= x \in S : -175 \leq Z < -50; \\ C_3 &= S \setminus \{C_1 \cup C_2\}. \end{aligned} \tag{4.3}$$

Os resultados computacionais obtidos ao aplicar os algoritmos novos propostos utilizando o método de particionamento proposto nesta seção pode ser visualizado na Tabela 4.5.

Tabela 4.5: Resultado computacional

Algoritmo	**Custo médio**	**Tempo**	**It.**
AT2F Distribuído (AT2FD)	3,40952	34.570 s	14
AT2F Distribuído e Modificado (AT2FDM)	3,40948	712 s	8
AT2F Distribuído, Priorizado e Modificado (AT2FDPM)	3,40979	323 s	4
AT2F Distribuído, Focalizado e Modificado (AT2FDFM)	3,40958	611 s	6
AT2F Distribuído, Priorizado, Focalizado e Modificado (AT2FDPFM)	3,40977	371 s	4

A partir das Tabelas 4.1, 4.2 e 4.5, é possível verificar que, utilizando o método da soma dos estoques, o algoritmo AT2FD ficou muito mais lento que o algoritmo Agregação Temporal de Duas Fases. De fato, o algoritmo AT2FD demorou $34.570s$ para convergir, enquanto que o algoritmo Agregação Temporal Duas Fases levou $8.199s$. Também, é possível observar que o algoritmo AT2FD continuou desempenhando melhor com o particionamento do menor produto em estoque. Mais uma vez, uma possível explicação para este melhor desempenho computacional é o fato de que o particionamento do menor produto em estoque gera subconjuntos com cardinalidades mais próximas, conforme pode ser visualizado na Tabela 4.4.

Tabela 4.6: Cardinalidade dos Subconjuntos

Subconjuntos	**Menor Produto em Estoque**	**Soma Absol. dos Estoques**	**Soma dos Estoques**
C_1	674.752	615.125	333.375
C_2	666.759	1.198.575	1.325.625
C_3	658.875	186.676	341.376

Em contrapartida, o algoritmo AT2FDM, utilizando o particionamento da soma dos estoques, convergiu em apenas 712 segundos. Este resultado mostra mais uma vez que limitar em 25 o número máximo de iterações no Passo 8 contribuiu significativamente para redução do tempo para a convergência. Contudo, vale destacar que o mesmo algoritmo utilizando o particionamento do menor produto em estoque convergiu ainda mais rápido (470s). Mais uma vez, o fato deste particionamento gerar subconjuntos com cardinalidades semelhantes pode ser uma explicação para o algoritmo AT2FDM ter apresentado um melhor desempenho computacional com este particionamento.

Com relação ao algoritmo AT2FDPM, a ordem de priorização adotada foi a seguinte: as políticas dos subconjuntos foram atualizadas em ordem crescente da soma dos estoques dos três produtos. Deste modo, a ordem de priorização foi C_3, C_2, C_1. Conforme, pode ser visualizado na Tabela 4.5, este algoritmo com o método de particionamento da soma dos estoque foi o que apresentou o menor tempo para a convergência. Comparando as Tabelas 4.2 e 4.5, vemos que com o método de particionamento da soma dos estoque, a ordem em que os subconjuntos são atualizados teve um efeito considerável no tempo para convergência, que decresceu para $323s$, praticamente o mesmo desempenho do algoritmo com o particionamento da soma absoluta. Uma provável explicação é a composição dos subconjuntos. Perceba que no método de particionamento do menor produto em estoque, os estados com estoques próximos de zero estão dispersos entre os três subconjuntos. Por outro lado, no método de particionamento da soma dos estoques, os estados próximos de zero estão concentrados no mesmo subconjunto C_3, que é o primeiro subconjunto na ordem de priorização. Como os estados mais próximos de zero tendem a ser os mais visitados, atualizar a política destes estados primeiro contribui para acelerar o algoritmo.

Por último, vale destacar que, utilizando o método da soma dos estoques, o algoritmo AT2FDPFM apresentou um desempenho pior do que o algoritmo AT2FDPM. Este resultado foi diferente quando utilizado o método de particionamento do menor produto em estoque, mas semelhante ao resultado obtido quando utilizado o particionamento da soma absoluta. Uma provável explicação é que no algoritmo AT2FDPFM, além dos estados de fronteira, são adicionados à cadeia de Markov embutida, os estados cujos estoques de todos os produtos estejam no intervalo $[-10, 9]$. Deste modo, a cadeia de Markov embutida gerada pelo algoritmo AT2FDPFM possui uma cardinalidade maior do que a cadeia de Markov embutida gerada pelo algoritmo AT2FDPM. Assim, é possível que esta quantidade adicional no número de estados da cadeia embutida tenha anulado os ganhos obtidos ao priorizar os subconjuntos, até porque, os estados acrescentados à cadeia embutida já fazem parte do subconjunto prioritário.

Pela análise dos resultados disponibilizados nas Tabelas 4.1, 4.2, 4.3 e 4.5, é possível verificar que, à exceção do algoritmo AT2FDPM, o particionamento do menor produto em estoque tender a ser mais eficiente que os demais particionamentos. Possivelmente, isto se deve ao fato do método do menor produto em estoque gerar subconjuntos com cardinalidades mais próximas. Em virtude disso, na próxima seção será analisado como os algoritmos novos propostos, utilizando o particionamento do menor produto em estoque, evoluem, iteração a iteração, até atingir a solução ótima.

4.2.3 Evolução até a proximidade da solução ótima

Nesta seção será analisado como os algoritmos novos propostos, utilizando o particionamento do menor produto em estoque, convergem para a solução ótima. A Tabela 4.7 mostra, para cada algoritmo implementado, a evolução do custo médio a cada iteração.

Tabela 4.7: Evolução de custo médio

	Tempo em segundos (Custo médio)						
It.	**IP**	**AT2P**	**AT2PD**	**AT2PDM**	**AT2PDPM**	**AT2PDFM**	**AT2PDPFM**
1	2.577 (60,8539)	2.332 (3,4887)	4.613 (3,4192)	83 (3,5615)	80 (3,5615)	89 (3,4375)	84 (3,4375)
2	3.924 (52,0270)	3.969 (3,4138)	6.858 (3,4163)	179 (3,4156)	169 (3,4156)	183 (3,4188)	167 (3,4188)
3	5.871 (41,5221)	5.465 (3,4127)	9.061 (3,4096)	280 (3,4142)	265 (3,4142)	280 (3,4107)	247 (3,4107)
4	7.055 (25,9269)	6.906 (3,4096)	10.815 (3,4096)	375 (3,4102)	358 (3,4102)	378 (3,4098)	331 (3,4098)
5	8.811 (26,5572)	8.198 (3,4096)	-	470 (3,4096)	461 (3,4096)	-	-
6	10.110 (22,7465)	-	-	-	-	-	-
7	11.664 (13,3264)	-	-	-	-	-	-
8	12.996 (11,6875)	-	-	-	-	-	-
9	14.563 (6,3609)	-	-	-	-	-	-
10	15.825 (5,8927)	-	-	-	-	-	-
11	17.302 (3,5493)	-	-	-	-	-	-
12	18.604 (3,4173)	-	-	-	-	-	-
13	19.723 (3,4095)		-	-	-	-	-
14	19.993 (3,4095)	-	-	-	-	-	-

É possível notar na Tabela 4.7 que o algoritmo de Iteração de Política é o que se aproxima mais lentamente da solução ótima. De fato, o algoritmo de Iteração de Política precisa de 14 iterações, e aproximadamente 20.000 segundos para convergir e 18.604 segundos para se aproximar da solução ótima. Em contrapartida, o algoritmo de Agregação Temporal de Duas Fases apresentou um desempenho significativamente superior ao aproximar da solução ótima em apenas 2 iterações e 3.968 segundos, visto que este algoritmo se concentra apenas nos estados do subconjunto F. Pela mesma razão, o algoritmo *Agregação Temporal Duas Fases Distribuído* se aproxima da solução ótima em, aproximadamente, $4.613s$.

Por outro lado, os demais algoritmos com o termo *modificado* em seu nome levaram aproximadamente 3 minutos para se aproximar da solução ótima. Estes resultados destacam a eficiência da abordagem proposta, que combina uma busca local no subconjunto F com passos de melhoria de política distribuída dentro dos subconjuntos em F^c.

No próximo capítulo será avaliado como os algoritmos novos performaram em outro exemplo diferente do anterior no contexto de gestão de filas, utilizando um método de particionamento análogo ao método do menor produto em estoque, visto que este método trouxe os resultados mais satisfatórios e encorajadores.

Capítulo 5

Aplicação a um problema de gerenciamento de filas

Esta seção apresenta os resultados obtidos ao aplicar os algoritmos novos propostos a um problema de gestão de filas apresentado em (VEATCH, 2001). Neste problema, os clientes chegam segundo uma distribuição de Poisson com taxa 1 e são direcionados para atendimento em um dos três servidores disponíveis, cada um com sua própria fila. Portanto, a cada chegada de um novo cliente, deve-se decidir para qual fila direcioná-lo.

Seja o tamanho da fila de cada servidor uma variável inteira no intervalo $\{0, \ldots, 150\}$. Isto significa que a quantidade máxima de clientes na fila permitida para cada servidor é 150 e nenhum cliente adicional é aceito quando este nível é atingido. Portanto, o espaço de estados S é composto de $151^3 \approx 3 \cdot 10^6$ elementos, cada um correspondendo a uma possível combinação da quantidade de clientes na fila de cada um dos três servidores. Suponha que as taxas de produção de todos os servidores sigam uma distribuição de Exponencial com taxas 0,5 para os servidores 1 e 2, e taxa 1,5 para o servidor 3. Considere também que o custo do estado do sistema é dado por:

$$f(x) = x_1 + 2x_2 + 4x_3,$$

em que x_1, x_2 e x_3 correspondem ao tamanho da fila nos servidores 1, 2 e 3, respectivamente. O objetivo é encontrar a política $\mathcal{L}^*$ que minimiza o custo médio e satisfaça (3.4). Para executar os experimentos, foi utilizado um notebook com processador Intel Core i3 com 2.30 GHz, memória RAM de 4 GB, sistema operacional Windows 10 e a linguagem de programação C++ (PRATA, 2001). A tolerância utilizada foi de 0.001, isto é, $\epsilon = 0.001$ no Passo 5 dos Algoritmos 5 e 6.

5.1 Método de Particionamento

Para resolver o problema de gestão de filas descrito nesta seção, foi aplicado um método de particionamento análogo ao método de particionamento do menor produto em estoque. Assim, o método utilizado neste capítulo divide o espaço de estados em uma quantidade de subconjuntos igual ao número de servidores, sendo os subconjuntos $\{C_1, C_2, C_3\}$ definidos como:

$$\begin{aligned} C_1 &= x \in S : x_1 \leq x_2, \text{ e } x_1 \leq x_3; \\ C_2 &= x \in S : x_1 > x_2, \text{ e } x_2 \leq x_3; \\ C_3 &= S \setminus \{C_1 \cup C_2\}. \end{aligned} \tag{5.1}$$

A lógica é dividir o espaço de estados conforme o servidor com a menor fila. Em caso de empate, isto é, se em um determinado estado, dois ou mais servidores terem o mesmo tamanho de fila mais curta, o estado pertencerá ao subconjunto com o menor índice.

5.2 Resultados

Os resultados obtidos utilizando os clássicos algoritmos Iteração de Política (IP) e Iteração de Valor Relativo (IVR), bem como o algoritmo de Agregação Temporal de Duas Fases (AT2F) (ARRUDA e FRAGOSO, 2015) estão dispostos na Tabela 5.1. Mais uma vez, estes três algoritmos atuarão como referências para os algoritmos propostos neste trabalho. Para o algoritmo Agregação Temporal em Duas Fases, o subconjunto F foi composto pelos estados cujo tamanho da filas dos 3 servidores fosse igual ou menor a 9, isto é, $F := \{x \in S : 0 \leq x_1 \leq 9, 0 \leq x_2 \leq 9, 0 \leq x_3 \leq 9\}$. O tempo de convergência e o número de iterações de cada algoritmo estão explicitados na Tabela 5.1.

Tabela 5.1: Resultado computacional

Algoritmo	**Custo médio**	**Tempo**	**It.**
IP	3,00022	6.109 s	4
IVR	3,00022	2.038 s	866
AT2F	3,00022	3.406 s	2

A Tabela 5.1 mostra que o Iteração de Valor Relativo foi o algoritmo mais rápido dentre os três algoritmos mais clássicos. Portanto, este será utilizado junto com o algoritmo de Agregação Temporal em Duas Fases como referência para efeito de comparação com os algoritmos novos propostos nesta Tese.

5.2.1 Resultados dos Algoritmos Novos Propostos

Nesta seção, serão discutidos os resultados computacionais obtidos ao aplicar os algoritmos novos propostos nesta Tese, utilizando o método de particionamento descrito na Seção 5.1 para resolver o problema de gestão de filas descrito anteriormente.

Tabela 5.2: Resultado computacional

Algoritmo	**Custo médio**	**Tempo**	**It.**
AT2F Distribuído (AT2FD)	3,00022	5.043 s	2
AT2F Distribuído e Modificado (AT2FDM)	3,00022	321 s	2
AT2F Distribuído, Priorizado e Modificado (AT2FDPM)	3,00022	308 s	2
AT2F Distribuído, Focalizado e Modificado (AT2FDFM)	3,00022	292 s	2
AT2F Distribuído, Priorizado, Focalizado e Modificado (AT2FDPFM)	3,00022	288 s	2

A partir das Tabelas 5.1 e 5.2, é possível verificar que o algoritmo de Agregação Temporal Duas Fases Distribuído (AT2FD) - Algoritmo 5 - novamente foi mais lento que o algoritmo de Agregação Temporal Duas Fases. De fato, o algoritmo Agregação Temporal Duas Fases Distribuído demorou $5.043s$ para convergir, enquanto que o algoritmo Agregação Temporal Duas Fases levou $3.406s$. Este fato reforça a explicação de que o número necessário de avaliações da cadeia embutida no Passo 8 do Algoritmo 5 torna este algoritmo mais lento, visto que sempre que a política dos estados de um subconjunto l for atualizada no Passo 7, será necessário resolver novamente a cadeia embutida.

Para amenizar este efeito, mais uma vez foi fixado em 25 o número máximo de iterações no Passo 8 do algoritmo AT2FDM. Também, conforme pode ser visualizado na Tabela 5.2, mais uma vez, limitar o número de iterações em 25 na avaliação da cadeia embutida teve um impacto significativo no tempo para a convergência, reduzindo-o para apenas 321 segundos.

Com relação ao algoritmo AT2FDPM, a ordem de priorização adotada foi a seguinte: o subconjunto mais prioritário é o subconjunto C_3, pois o servidor 3 é o servidor com a maior taxa de produção. O segundo subconjunto mais prioritário é o subconjunto C_2 visto que o servidor 2 possui maior custo do que o servidor 1. O último subconjunto é o subconjunto C_1 - ver Eq. (5.1) para a definição dos subconjuntos. Conforme exposto na Tabela 4.2, a ordem em que os subconjuntos são atualizados novamente reduziu o tempo para convergência para $308s$.

Contudo, o algoritmo AT2FDFM, mais uma vez, apresentou um tempo computacional ainda menor. No algoritmo AT2FDFM, foram adicionados ao subconjunto F os estados cujas filas de todos os servidores estivessem no intervalo $[0, 9]$, i.e, $F := \{x \in S : 0 \leq x_1 \leq 9, 0 \leq x_2 \leq 9, 0 \leq x_3 \leq 9\}$. Conforme pode ser visualizado na Tabela 5.2, adicionar estes estados ao subconjunto F levou a uma redução no tempo de convergência, alcançando 292 segundos. Ainda de acordo com a Tabela, combinar as abordagens *focalizada* e *priorizada* no algoritmo AT2FDPFM gerou

o melhor tempo computacional, convergindo em apenas 288 segundos, aproximadamente 8,5% do tempo de convergência do algoritmo Agregação Temporal Duas Fases (14,1% do tempo de convergência do Iteração de Valor Relativo e 4,7% do tempo de convergência do Iteração de Política, conforme a Tabela 4.1).

Apesar dos resultados promissores encontrados com os métodos de particionamento do menor produto em estoque e o servidor com menor fila, no próximo capítulo, será proposto mais um novo esquema de particionamento em que é possível combinar os Algoritmos de Agregação Temporal em Duas Fases Distribuído com abordagens de programação dinâmica aproximada e aprendizado por reforço.

Capítulo 6

Um novo esquema de particionamento para um algoritmo aproximado

Nesta seção, será introduzido um esquema de particionamento que segue a Definição 1 e utiliza o Teorema de Foster. Com este novo esquema particionamento, é possível combinar a classe de algoritmos Agregação Temporal em Duas Fases Distribuído com abordagens de programação dinâmica aproximada ou aprendizado por reforço. Desse modo, será possível resolver problemas de alta dimensionalidade com menor tempo computacional à custa de obter um custo aproximado e não exato. Os resultados detalhados que sustentam o esquema de particionamento proposto podem ser encontrados no Apêndice A.

Apresentado na Hipótese 1, o Teorema de Foster estabelece condições suficientes para recorrência positiva de uma cadeia de Markov, garantindo que a cadeia de Markov controlada seja ergódica sob qualquer política viável (veja (FOSTER, 1953) e (BRÉMAUD, 2020, Teorema 7.1.1, página 227)), mesmo que a cardinalidade do espaço de estados seja muito grande.

Hipótese 1 (Condição de Foster)**.** *Assuma que para qualquer política viável $\mathcal{L} \in \mathbb{L}$, existe uma função Foster-Lyapunov $g : S \to \mathbb{Z}_+$, em que $\mathbb{Z}_+$ é o conjunto de números inteiros não negativos, de modo que:*

$$\begin{aligned} &\sum_{j \in S} p_{ij}^{\mathcal{L}} g(j) < \infty, i \in S_1 \\ &\sum_{j \in S} p_{ij}^{\mathcal{L}} g(j) \leq g(i) - \epsilon, i \notin S_1, \\ &\max_{i \in S_1} g(i) < \min_{j \notin S_1} g(j), \end{aligned} \tag{6.1}$$

para algum conjunto finito $S_1 \subset S$ e escalar $\epsilon > 0$.

O Teorema 8 do Apêndice A prova que, sob a Hipótese 1, a probabilidade estacionária de que $g(X_t)$ seja maior que um dado limite $\bar{\sigma}$ decresce exponencialmente à medida que $\bar{\sigma}$ aumenta - se o processo X_t, $t \geq 0$ evolui fora de uma região S_1. Assim, será proposto um esquema de particionamento em função de $g(\cdot)$, que pode ser aplicado a problemas com espaços de estados com cardinalidade muito grande.

Definição 2 (Esquema de particionamento com subconjuntos de cardinalidades exponencialmente crescentes (EPCEC))**.** *Seja $\mathcal{P}(S)$ um esquema de particionamento que satisfaz a Definição 1, de modo que*

$$n = \left\lceil \ln\left(\max_{j \in S} g(j)\right) \right\rceil, \quad B_1 = \{j \in S : g(j) < e^1\}$$

e

$$B_l = \{j \in S \text{ tal que } j \notin \bigcup_{m=1}^{l-1} B_m \text{ e } g(j) < e^l\},\ l = 2, \ldots, n.$$

Perceba que a cardinalidade de B_l aumenta exponencialmente com l. Contudo, este aumento exponencial na cardinalidade dos subconjuntos é compensado pela diminuição exponencial na probabilidade estacionária de que $g(\cdot)$ fique fora do subconjunto $\bigcup_{m=1}^{l} B_m$ (veja o Teorema 8). Pela Eq. (6.1), é esperado que o número de estados de *fronteira* em B_l também seja limitado em relação à cardinalidade do subconjunto, em virtude da segunda expressão em (6.1). Isso manterá limitada a cardinalidade do conjunto de estados de fronteira na cadeia embutida, tornando viável a solução da formulação embutida na Etapa 6 do Algoritmo 6.

Além disso, de acordo com o Teorema 8, a importância de um subconjunto B_l para a média de longo prazo diminui exponencialmente em l. Desse modo, à medida que a cardinalidade do conjunto de estados interiores I_l aumenta, pode-se aplicar uma política aproximada com efeito limitado no desempenho de longo prazo. Portanto, a Etapa 13 do Algoritmo 6 pode ser aproximada - por exemplo, via simulação de Monte Carlo de uma política ad-hoc - para grandes l, e otimização de um problema aproximado apenas para subconjuntos com l reduzido.

A seção seguinte, descreve a aplicação do esquema de particionamento proposto no mesmo problema de gestão de filas do Capítulo 5.

6.1 Aplicação da abordagem aproximada para o problema de gestão de filas

Considere o exemplo numérico de gestão de filas do Capítulo 5. Neste problema, os clientes chegam segundo uma distribuição de Poisson com taxa 1 e são direcionados para atendimento em um dos três servidores disponíveis, cada um com sua

própria fila. Portanto, a cada chegada de um novo cliente, o decisor deve decidir para qual fila direcioná-lo.

O tamanho da fila de cada servidor é uma variável inteira no intervalo $\{0, \ldots, 150\}$. Isto significa que a quantidade máxima de clientes na fila permitida para cada servidor é 150 e nenhum cliente adicional é aceito quando este nível é atingido. Portanto, o espaço de estados S é composto de $151^3 \approx 3 \cdot 10^6$ elementos, cada um correspondendo a uma possível combinação da quantidade de clientes na fila de cada um dos três servidores.

Assim, considerando que a função de Lyapunov retorna o tamanho da maior fila de um determinado estado do espaço de estados, temos que:

$$n = \left\lceil \ln\left(\max_{j \in S} g(j)\right) \right\rceil = \lceil \ln 150 \rceil = 6$$

Portanto, o espaço de estados pode ser dividido em 6 (seis) subconjuntos, de modo que cada subconjunto é dado por:

$$\begin{aligned}
B_1 &= \{j \in S : g(j) < e^1\} = \{j \in S : g(j) < 2,72\};\\
B_2 &= \{j \in S : e^1 \leq g(j) < e^2\} = \{j \in S : 2,72 \leq g(j) < 7,39\};\\
B_3 &= \{j \in S : e^2 \leq g(j) < e^3\} = \{j \in S : 7,39 \leq g(j) < 20,09\};\\
B_4 &= \{j \in S : e^3 \leq g(j) < e^4\} = \{j \in S : 20,09 \leq g(j) < 54,60\};\\
B_5 &= \{j \in S : e^4 \leq g(j) < e^5\} = \{j \in S : 54,60 \leq g(j) < 148,41\};\\
B_6 &= \{j \in S : e^5 \leq g(j) < e^6\} = \{j \in S : 148,41 \leq g(j) < 403,43\};
\end{aligned} \tag{6.2}$$

Suponha que as taxas de produção de todos os servidores seguem uma distribuição de Exponencial com taxas 0,5 para os servidores 1 e 2, e taxa 1,5 para o servidor 3. Considere também que o custo do estado do sistema é dado por:

$$f(x) = x_1 + 2x_2 + 4x_3,$$

em que x_1, x_2 e x_3 correspondem ao tamanho da fila nos servidores 1, 2 e 3, respectivamente. O objetivo é encontrar a política $\mathcal{L}^*$ que minimiza o custo médio e satisfaça (3.4).

Para resolver este problema, foram utilizados os algoritmos IVR e AT2FD Aproximado. No algoritmo aproximado, um modelo aproximado foi resolvido e sua solução foi utilizada como uma aproximação para o modelo original. No modelo aproximado foi realizado um truncamento do espaço de estados a partir do particionamento 6.2 de modo que o Algoritmo 5 iterou apenas nos estados em que todos os servidores possuem um tamanho de fila igual ou inferior a 55 - isto é, o espaço de estados foi restrito aos subconjuntos B_1 a B_4. Em contrapartida, uma política

ad-hoc foi fixada para todos os demais estados. Portanto, no modelo aproximado, uma política *ad-hoc* foi fixada para todos os estados dos subconjuntos B_5 e B_6 e para os estados interiores do subconjunto B_4; enquanto uma otimização conforme o Algoritmo 5 foi aplicada nos demais estados. Desse modo, o algoritmo aproximado varreu um espaço de estados de tamanho $51^3 \approx 1 \cdot 10^5$, ao invés de varrer um espaço de estado de cardinalidade $151^3 \approx 3 \cdot 10^6$ como no algoritmo IVR.

Para executar os experimentos, foi utilizado um notebook com processador Intel Core i3 com 2.30 GHz, memória RAM de 4 GB, sistema operacional Windows 10 e a linguagem de programação C++ (PRATA, 2001). A tolerância utilizada foi de 0.001, isto é, $\epsilon = 0.001$ no Passo 5 dos Algoritmos 5 e 6.

6.1.1 Resultados

A Tabela 6.1 mostra o custo médio e o tempo para convergência (em parêntesis) obtidos ao resolver o problema de gestão de filas para diferentes tempos de chegada.

Tabela 6.1: Resultado computacional

	Custo médio (Tempo em seg.)		
Algoritmo	$\lambda = 1$	$\lambda = 1.5$	$\lambda = 2.0$
IVR	3,00022 (2.038 s)	5,81106 (2.512 s)	12,6144 (6.211 s)
AT2FD Aproximado	3.00023 (199 s)	5,81106 (410 s)	12,6143 (1.697 s)

Conforme podemos avaliar na Tabela 6.1, a abordagem aproximada resultou num custo médio aproximado bem próximo ao custo médio ótimo, calculado pelo Iteração de Valor Relativo, o que demonstra a qualidade da solução encontrada pelo algoritmo novo aqui proposto. É evidente, contudo, que a qualidade da solução aproximada irá variar com o nível de truncamento realizado. A Tabela 6.2, mostra os resultados obtidos ao simular a trajetória do processo de Markov original de acordo com a política encontrada pelo algoritmo AT2FD Aproximado. Entre parêntesis, consta o número de vezes que o processo visitou os estados fora do truncamento, isto é, os estados em que algum servidor possui um tamanho de fila acima de 55. Em contrapartida, fora dos parêntesis, consta o custo médio obtido pela simulação da mesma política.

Tabela 6.2: Resultado computacional

	Custo médio (Visitas fora do truncamento)		
Algoritmo	$\lambda = 1$	$\lambda = 1.5$	$\lambda = 2.0$
Simulação	3,00344 (0)	5,79566 (0)	12,5572 (0)

Como é possível notar na Tabela 6.2, na simulação realizada, o processo de Markov não visitou nenhuma vez os estados fora da região de truncamento. Este

fato corrobora o Teorema 8 ao demonstrar que os estados dos subconjuntos mais distantes possuem um impacto muito pequeno para o custo médio de longo prazo. Por isso, o custo médio encontrado pelo algoritmo AT2FD Aproximado é muito próximo ao custo médio ótimo.

Adicionalmente, perceba que o algoritmo AT2FD Aproximado é baseado na construção de um modelo aproximado gerado por um truncamento do espaço de estados a partir do particionamento 6.2, sendo a região de truncamento definida *à priori*. Tudo isso demonstra a capacidade do Algoritmo AT2FD Aproximado em resolver problemas de alta dimensionalidade. É claro que a qualidade da solução irá depender da região de truncamento do espaço de estados, bem como das taxas de chegada e partida, que, neste exemplo, geram as probabilidades de transição do processo de Markov.

Desse modo, para problemas com 3 servidores, truncar o espaço de estados em 55 não teve impacto na qualidade da solução além de ter resolvido o algoritmo em poucos segundos quando comparado ao IVR. Para problemas maiores, com mais variáveis (e.g. servidores), talvez, seja necessário um truncamento ainda maior para tornar a solução viável computacionalmente ou até mesmo para resolver o problema em um tempo computacional menor. O problema é que este truncamento maior pode resultar numa solução aproximada mais distante da solução exata a depender das taxas taxas de chegada e partida neste exemplo. Portanto, ao resolver um problema de decisão de Markov com o algoritmo AT2FD Aproximado é preciso ponderar a qualidade da solução desejada com o tempo e a viabilidade computacional para definir a região de truncamento que será empregada.

Capítulo 7

Conclusões

Este trabalho propôs e combinou um novo esquema de particionamento à abordagem Agregação Temporal para derivar uma classe de algoritmos eficientes para processos de decisão de Markov sob o critério de custo médio. O novo esquema de particionamento induz propriedades topológicas desejáveis dentro de cada subconjunto da partição. Isso permite avaliar as trajetórias semi-regenerativas agregadas no tempo, concentrando-se em um único subconjunto de cada vez, e assim limitando o esforço computacional. Também permite etapas de melhoria de política de forma distribuída que focam em um único subconjunto por vez, o que contribui para acelerar a convergência. O novo esquema de particionamento também permite uma Agregação Temporal em duas fases ao gerar um subconjunto F onde o processo semi-Markov embutido evolui. Esse subconjunto compreende os estados que podem ser acessados por diferentes subconjuntos na partição, permitindo assim uma pesquisa local dentro dos estados a partir dos quais as informações se espalham pelo espaço de estado. Em contrapartida, a Agregação Temporal Duas Fases clássica particiona o espaço de estados arbitrariamente, contando fortemente com o conhecimento do domínio do problema por parte do tomador de decisão.

Assim, combinado com a agregação temporal, o esquema de particionamento proposto dá origem a uma classe de algoritmos que convergem para a política ótima e que podem encontrar a solução ótima de maneira significativamente mais rápida em comparação com os algoritmos clássicos e com a Agregação Temporal Duas Fases. Adicionalmente, utilizando o esquema de particionamento proposto, esta Tese também desenvolveu um novo algoritmo aproximado para resolver problemas de alta dimensionalidade com menor tempo computacional. Os experimentos demonstraram o potencial da abordagem proposta, sugerindo caminhos potenciais para trabalhos futuros. Isso inclui extensões para MDPs com critério de custo ou retorno descontado, bem como a exploração de novas estratégias de busca dentro dos subconjuntos de partição. Pode-se também explorar diferentes alternativas para as funções de Lyapunov que auxiliam o particionamento, ou fazer uso de avanços re-

centes em aprendizado por reforço e programação dinâmica aproximada nos passos de melhoria de política em estados interiores.

Por último, esta Tese de Doutorado apresentou ainda um conjunto de mapas conceituais que introduz ao leitor um panorama do estado da arte em processos de decisão de Markov.

Referências Bibliográficas

SILVA, T. A., DE SOUZA, M. C. "Surgical scheduling under uncertainty by approximate dynamic programming", *Omega*, v. 95, pp. 102066, 2020.

NUNES, L. G. N., DE CARVALHO, S. V., RODRIGUES, R. D. C. M. "Markov decision process applied to the control of hospital elective admissions", *Artificial intelligence in medicine*, v. 47, n. 2, pp. 159–171, 2009.

KARA, A., DOGAN, I. "Reinforcement learning approaches for specifying ordering policies of perishable inventory systems", *Expert Systems with Applications*, v. 91, pp. 150–158, 2018.

SOARES, H. L., ARRUDA, E. F., BAHIENSE, L., et al. "Optimisation and control of the supply of blood bags in hemotherapic centres via Markov Decision Process with discounted arrival rate", *Artificial Intelligence in Medicine*, v. 104, pp. 101791, 2020.

HU, B., LIU, D., ZANG, C., et al. "Coordinate Dispatch of Combined Heat and Power System Based on Approximate Dynamic Programming". In: *2020 IEEE 4th Conference on Energy Internet and Energy System Integration (EI2)*, pp. 3772–3778. IEEE, 2020.

XUE, X., AI, X., FANG, J., et al. "Real-time schedule of integrated heat and power system: A multi-dimensional stochastic approximate dynamic programming approach", *International Journal of Electrical Power & Energy Systems*, v. 134, pp. 107427, 2022.

VOELKEL, M. A., SACHS, A.-L., THONEMANN, U. W. "An aggregation-based approximate dynamic programming approach for the periodic review model with random yield", *European Journal of Operational Research*, v. 281, n. 2, pp. 286–298, 2020.

YU, L., ZHANG, C., JIANG, J., et al. "Reinforcement learning approach for resource allocation in humanitarian logistics", *Expert Systems with Applications*, v. 173, pp. 114663, 2021.

FOROOTANI, A., IERVOLINO, R., TIPALDI, M., et al. "Approximate dynamic programming for stochastic resource allocation problems", *IEEE/CAA Journal of Automatica Sinica*, v. 7, n. 4, pp. 975–990, 2020.

PUTERMAN, M. L. *Markov decision processes: discrete stochastic dynamic programming*. New Jersey, John Wiley & Sons, 2005.

RUST, J. "Has dynamic programming improved decision making?" *Annual Review of Economics*, v. 11, pp. 833–858, 2019.

POWELL, W. B. *Approximate Dynamic Programming: Solving the curses of dimensionality*, v. 703. New Jersey, John Wiley & Sons, 2007.

BOUCHERIE, R. J., VAN DIJK, N. M. *Markov decision processes in practice*. Gewerbestrasse, Springer, 2017.

ARRUDA, E. F., PEREIRA, B. B., THIERS, C. A., et al. "Optimal testing policies for diagnosing patients with intermediary probability of disease", *Artificial Intelligence in Medicine*, v. 97, pp. 89–97, 2019a. ISSN: 0933-3657. doi: https://doi.org/10.1016/j.artmed.2018.11.005.

POWELL, W. B. "Perspectives of approximate dynamic programming", *Annals of Operations Research*, v. 241, n. 1, pp. 319–356, 2016.

JIANG, D. R., POWELL, W. B. "An approximate dynamic programming algorithm for monotone value functions", *Operations Research*, v. 63, n. 6, pp. 1489–1511, 2015.

ARRUDA, E. F., OURIQUE, F. O., LACOMBE, J., et al. "Accelerating the convergence of value iteration by using partial transition functions", *European Journal of Operational Research*, v. 229, n. 1, pp. 190–198, 2013.

WATKINS, C. J. C. H. *Learning from delayed rewards*. Tese de D.Sc., King's College, Cambridge, United Kingdom, 1989.

KAMANCHI, C., DIDDIGI, R. B., BHATNAGAR, S. "Successive Over-Relaxation Q-Learning", *IEEE Control Systems Letters*, v. 4, n. 1, pp. 55–60, 2019.

JOHN, I., KAMANCHI, C., BHATNAGAR, S. "Generalized speedy q-learning", *IEEE Control Systems Letters*, v. 4, n. 3, pp. 524–529, 2020.

CAO, X.-R., REN, Z., BHATNAGAR, S., et al. "A time aggregation approach to Markov decision processes", *Automatica*, v. 38, n. 6, pp. 929–943, 2002.

SUN, T., ZHAO, Q., LUH, P. B. "Incremental value iteration for time-aggregated Markov-decision processes", *IEEE Transactions on Automatic Control*, v. 52, n. 11, pp. 2177–2182, 2007.

ARRUDA, E. F., FRAGOSO, M. D., OURIQUE, F. O. "Multi-partition time aggregation for Markov Chains". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 4922–4927. IEEE, 2017.

ARRUDA, E., FRAGOSO, M. D. "Solving average cost Markov decision processes by means of a two-phase time aggregation algorithm", *European Journal of Operational Research*, v. 240, n. 3, pp. 697–705, 2015.

ARRUDA, E., FRAGOSO, M., OURIQUE, F. "A multi-cluster time aggregation approach for Markov chains", *Automatica*, v. 99, pp. 382–389, 2019b.

SUTTON, R. S., BARTO, A. G. *Reinforcement learning: An introduction*. Cambridge, MIT press, 2018.

CAÑAS, A. J., HILL, G., CARFF, R., et al. "CmapTools: a knowledge modeling and sharing environment". In: *Concept maps: theory, methodology, technology: Proceedings of the First International Conference on Concept Mapping*, pp. 125–133. Dirección de Publicaciones de la Universidad Pública de Navarra, 2004.

NOVAK, J. D., CAÑAS, A. J. "The origins of the concept mapping tool and the continuing evolution of the tool", *Information visualization*, v. 5, n. 3, pp. 175–184, 2006.

EPPLER, M. J. "A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing", *Information visualization*, v. 5, n. 3, pp. 202–210, 2006.

NOVAK, J. D., CAÑAS, A. J. "The theory underlying concept maps and how to construct them", *Florida Institute for Human and Machine Cognition*, v. 1, n. 1, pp. 1–31, 2008.

BALIGA, S. S., WALVEKAR, P. R., MAHANTSHETTI, G. J. "Concept map as a teaching and learning tool for medical students", *Journal of Education and Health Promotion*, v. 10, 2021.

EACHEMPATI, P., RAMNARAYAN, K., KS, K. K., et al. "Concept Maps for Teaching, Training, Testing and Thinking", *MedEdPublish*, v. 9, 2020.

ISLAM, I., MUNIM, K. M., OISHWEE, S. J., et al. "A critical review of concepts, benefits, and pitfalls of blockchain technology using concept map", *IEEE Access*, v. 8, pp. 68333–68341, 2020.

CONCEIÇÃO, S. C., SAMUEL, A., YELICH BINIECKI, S. M. "Using concept mapping as a tool for conducting research: An analysis of three approaches", *Cogent Social Sciences*, v. 3, n. 1, pp. 1404753, 2017.

BERTSEKAS, D. P., OTHERS. *Dynamic programming and optimal control: Vol. 1.* Nashua, Athena scientific Belmont, 2000.

R CORE TEAM. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2020. Disponível em: <`https://www.R-project.org/`>.

ARIA, M., CUCCURULLO, C. "bibliometrix: An R-tool for comprehensive science mapping analysis", *Journal of Informetrics*, v. 11, n. 4, pp. 959–975, 2017. Disponível em: <`https://doi.org/10.1016/j.joi.2017.08.007`>.

POMBO, N., GARCIA, N., BOUSSON, K. "Classification techniques on computerized systems to predict and/or to detect Apnea: A systematic review", *Computer methods and programs in biomedicine*, v. 140, pp. 265–274, 2017.

NI, Q., TANG, Y. "A Bibliometric Visualized Analysis and Classification of Vehicle Routing Problem Research", *Sustainability*, v. 15, n. 9, pp. 7394, 2023.

BRÉMAUD, P. *Gibbs fields, Monte Carlo simulation, and queues.* New York, Springer-Verlag, 1999.

PUTERMAN, M. L., SHIN, M. C. "Modified policy iteration algorithms for discounted Markov decision problems", *Management Science*, v. 24, n. 11, pp. 1127–1137, 1978.

REETZ, D. "Solution of a Markovian decision problem by successive overrelaxation", *Zeitschrift für Operations Research*, v. 17, n. 1, pp. 29–32, 1973.

CHANG, H. S. "Policy set iteration for Markov decision processes", *Automatica*, v. 49, n. 12, pp. 3687–3689, 2013.

CHANG, H. S. "Value set iteration for Markov decision processes", *Automatica*, v. 50, n. 7, pp. 1940–1943, 2014.

LEITE, J. M. L. G., ARRUDA, E. F., BAHIENSE, L., et al. "Mine-to-client planning with Markov Decision Process". In: *2020 European Control Conference (ECC)*, pp. 1123–1128. IEEE, 2020.

MOORE, A. W., ATKESON, C. G. "Prioritized sweeping: Reinforcement learning with less data and less time", *Machine learning*, v. 13, n. 1, pp. 103–130, 1993.

BARTO, A. G., BRADTKE, S. J., SINGH, S. P. "Learning to act using real-time dynamic programming", *Artificial intelligence*, v. 72, n. 1-2, pp. 81–138, 1995.

HANSEN, E. A., ZILBERSTEIN, S. "LAO*: A heuristic search algorithm that finds solutions with loops", *Artificial Intelligence*, v. 129, n. 1-2, pp. 35–62, 2001.

DAI, P., GOLDSMITH, J. "Topological Value Iteration Algorithm for Markov Decision Processes." In: *IJCAI*, pp. 1860–1865, 2007a.

DAI, P., WELD, D. S., GOLDSMITH, J., et al. "Topological value iteration algorithms", *Journal of Artificial Intelligence Research*, v. 42, pp. 181–209, 2011.

FERGUSON, D., STENTZ, A. "Focussed processing of MDPs for path planning". In: *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 310–317. IEEE, 2004.

WINGATE, D., SEPPI, K. D., MAHADEVAN, S. "Prioritization Methods for Accelerating MDP Solvers." *Journal of Machine Learning Research*, v. 6, n. 5, 2005.

DE GUADALUPE GARCIA-HERNANDEZ, M., RUIZ-PINALES, J., ONAINDIA, E., et al. "New prioritized value iteration for Markov decision processes", *Artificial Intelligence Review*, v. 37, n. 2, pp. 157–167, 2012.

DEBNATH, S., LIU, L., SUKHATME, G. "Solving Markov Decision Processes with Reachability Characterization from Mean First Passage Times". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7063–7070. IEEE, 2018.

MOHAGHEGHI, M., KARIMPOUR, J., ISAZADEH, A. "Prioritizing methods to accelerate probabilistic model checking of discrete-time Markov models", *The Computer Journal*, v. 63, n. 1, pp. 105–122, 2020.

BONET, B., GEFFNER, H. "Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming." In: *ICAPS*, v. 3, pp. 12–21, 2003a.

MCMAHAN, H. B., LIKHACHEV, M., GORDON, G. J. "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees". In: *Proceedings of the 22nd international conference on Machine learning*, pp. 569–576, 2005.

SMITH, T., SIMMONS, R. "Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic". In: *AAAI*, pp. 1227–1232, 2006.

BONET, B., GEFFNER, H. "Faster heuristic search algorithms for planning with uncertainty and full feedback". In: *IJCAI*, pp. 1233–1238. Citeseer, 2003b.

HART, P. E., NILSSON, N. J., RAPHAEL, B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Transactions on Systems Science and Cybernetics*, v. 4, n. 2, pp. 100–107, 1968. doi: 10.1109/TSSC.1968.300136.

EDELKAMP, S., SCHRODL, S. *Heuristic search: theory and applications*. Waltham, USA, Elsevier, 2011.

NILSSON, N. J. *Principles of artificial intelligence*. USA, Morgan Kaufmann Publishers, 1982.

BHUMA, V. D. K. *Bidirectional LAO* algorithm (a faster approach to solve goal-directed MDPs)*. Tese de Mestrado, University of Kentucky, Lexington, Kentucky, Estados Unidos, 2004.

DAI, P., GOLDSMITH, J. "LAO*, RLAO*, or BLAO*". In: *AAAI Workshop on Heuristic Search*, pp. 59–64, 2006.

DAI, P., GOLDSMITH, J. "Multi-Threaded BLAO* Algorithm." In: *FLAIRS Conference*, pp. 56–61, 2007b.

DAI, P., WELD, D. S., OTHERS. "Focused topological value iteration". In: *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

LARACH, A., CHAFIK, S., DAOUI, C. "Accelerated decomposition techniques for large discounted Markov decision processes", *Journal of Industrial Engineering International*, v. 13, n. 4, pp. 417–426, 2017.

DIBANGOYE, J. S., CHAIB-DRAA, B., MOUADDIB, A.-I. "A Novel Prioritization Technique for Solving Markov Decision Processes." In: *FLAIRS Conference*, pp. 537–542, 2008.

RUMMERY, G. A., NIRANJAN, M. *On-line Q-learning using connectionist systems*, v. 37. Cambridge, University of Cambridge, Department of Engineering Cambridge, UK, 1994.

PENG, J., WILLIAMS, R. J. "Incremental multi-step Q-learning". In: *Machine Learning Proceedings 1994*, Elsevier, pp. 226–232, New Brunswick, New Jersey, USA, 1994.

STREHL, A. L., LI, L., WIEWIORA, E., et al. "PAC model-free reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*, pp. 881–888, 2006.

HASSELT, H. "Double Q-learning", *Advances in neural information processing systems*, v. 23, pp. 2613–2621, 2010.

AZAR, M. G., MUNOS, R., GHAVAMZADAEH, M., et al. "Speedy Q-learning", *Advances in neural information processing systems*, v. 24, pp. 2411–2419, 2011.

ZHANG, Z., PAN, Z., KOCHENDERFER, M. J. "Weighted Double Q-learning." In: *IJCAI*, pp. 3455–3461, 2017.

ABED-ALGUNI, B. H., OTTOM, M. A. "Double delayed Q-learning", *International Journal of Artificial Intelligence*, v. 16, n. 2, pp. 41–59, 2018.

DEVRAJ, A. M., MEYN, S. P. "Fastest convergence for Q-learning", *arXiv preprint arXiv:1707.03770*, 2017a.

DEVRAJ, A. M., MEYN, S. P. "Zap Q-learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 2232–2241, 2017b.

DEVRAJ, A. M., BUŠIĆ, A., MEYN, S. "Zap Q-Learning-A User's Guide". In: *2019 Fifth Indian Control Conference (ICC)*, pp. 10–15. IEEE, 2019.

HE, X., JIN, R., DAI, H. "Glide and Zap Q-Learning". In: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1147–1152. IEEE, 2020.

CHEN, S., DEVRAJ, A. M., LU, F., et al. "Zap Q-learning with nonlinear function approximation", *Advances in Neural Information Processing Systems*, v. 33, pp. 16879–16890, 2020.

DIDDIGI, R. B., KAMANCHI, C., BHATNAGAR, S. "A convergent off-policy temporal difference algorithm", *arXiv preprint arXiv:1911.05697*, 2019.

CHEN, G., GAEBLER, J. D., PENG, M., et al. "An Adaptive State Aggregation Algorithm for Markov Decision Processes", *arXiv preprint arXiv:2107.11053*, 2021.

BERTSEKAS, D. *Abstract dynamic programming*. Nashua, Athena Scientific, 2022.

KEMENY, J. G., SNELL, J. L. *Finite Markov Chains*. New York, Springer-Verlag, 1976.

SINGH, S., JAAKKOLA, T., JORDAN, M. "Reinforcement learning with soft state aggregation", *Advances in neural information processing systems*, v. 7, 1994.

DUAN, Y., KE, T., WANG, M. "State aggregation learning from markov transition data", *Advances in Neural Information Processing Systems*, v. 32, 2019.

POWELL, W. B. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. Princeton, Wiley, 2022.

TSITSIKLIS, J. N., VAN ROY, B. "Feature-based methods for large scale dynamic programming", *Machine Learning*, v. 22, n. 1, pp. 59–94, 1996.

BERTSEKAS, D. P. "Feature-based aggregation and deep reinforcement learning: A survey and some new implementations", *IEEE/CAA Journal of Automatica Sinica*, v. 6, n. 1, pp. 1–31, 2018.

BERTSEKAS, D. P., CASTANON, D. A., OTHERS. "Adaptive aggregation methods for infinite horizon dynamic programming", *IEEE Transactions on Automatic Control*, 1989.

ARRUDA, E. F., FRAGOSO, M. D. "Standard dynamic programming applied to time aggregated Markov decision processes". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 2576–2580. IEEE, 2009.

ARRUDA, E. F., FRAGOSO, M. D. "Time aggregated Markov decision processes via standard dynamic programming", *Operations Research Letters*, v. 39, n. 3, pp. 193–197, 2011.

ARRUDA, E. F., FRAGOSO, M. D. "Discounted Markov decision processes via time aggregation". In: *2016 European Control Conference (ECC)*, pp. 2578–2583. IEEE, 2016.

BERTSEKAS, D. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021.

CAKIR, F., THOMAS, B. W., STREET, W. N. "Rollout-based routing strategies with embedded prediction: A fish trawling application", *Computers & Operations Research*, v. 150, pp. 106055, 2023.

WU, J., JIA, Q.-S. "Performance Loss Bound for State Aggregation in a Class of Supply Demand Matching Systems". In: *2020 39th Chinese Control Conference (CCC)*, pp. 4307–4312. IEEE, 2020.

LI, Y., WU, X. "A unified approach to time-aggregated Markov decision processes", *Automatica*, v. 67, pp. 77–84, 2016.

CAO, X.-R. "The relations among potentials, perturbation analysis, and Markov decision processes", *Discrete Event Dynamic Systems*, v. 8, n. 1, pp. 71–87, 1998.

KIM, K. "Multi-Agent Deep Q Network to Enhance the Reinforcement Learning for Delayed Reward System", *Applied Sciences*, v. 12, n. 7, 2022.

PRATA, S. *C++ Primer Plus*. 4th ed. USA, Sams, 2001.

VEATCH, M. H. "Fluid analysis of arrival routing", *IEEE Transactions on Automatic Control*, v. 46, n. 8, pp. 1254–1257, 2001.

FOSTER, F. G. "On the Stochastic Matrices Associated with Certain Queuing Processes", *The Annals of Mathematical Statistics*, v. 24, n. 3, pp. 355–360, 1953. ISSN: 00034851.

BRÉMAUD, P. *Gibbs fields, Monte Carlo simulation, and queues*. N. 31, Texts in Applied Mathematics. 2nd edition ed. Gewerbestrasse, Springer-Verlag, 2020.

CIUCU, F., POLOCZEK, F., RIZK, A. "Queue and Loss Distributions in Finite-Buffer Queues", *Proc. ACM Meas. Anal. Comput. Syst.*, v. 3, n. 2, 2019. doi: 10.1145/3341617.3326146.

Apêndice A

A escolha dos subconjuntos da partição

Este apêndice apresenta um resultado fundamental que embasa o método de particionamento proposto no Capítulo 6. Sumariamente, os resultados de FOSTER (1953) e CIUCU *et al.* (2019) são utilizados para provar que, pelo método de particionamento proposto no Capítulo 6, a importância de um subconjunto B_l para a média de longo prazo diminui exponencialmente em l de modo que é possível aplicar uma política aproximada para os estados dos subconjuntos mais distantes com efeito limitado no desempenho de longo prazo. O resultado mais importante deste apêndice é o Teorema 8. Contudo, inicialmente, será discutido as implicações do teorema de Foster.

Assuma que para qualquer política viável $\mathcal{L} \in \mathbb{L}$, exista uma função Foster-Lyapunov $g : S \to \mathbb{Z}_+$, em que $\mathbb{Z}_+$ é o conjunto de números inteiros não negativos, tal que:

$$\begin{aligned} &\sum_{j \in S} p_{ij}^{\mathcal{L}} g(j) < \infty, i \in S_1 \\ &\sum_{j \in S} p_{ij}^{\mathcal{L}} g(j) \leq g(i) - \epsilon, i \notin S_1, \\ &\max_{i \in S_1} g(i) < \min_{j \notin S_1} g(j), \end{aligned} \tag{A.1}$$

para um conjunto finito $S_1 \subset S$ e escalar $\epsilon > 0$.

De acordo com o teorema de Foster, esta é uma condição suficiente para garantir que a cadeia de Markov é ergódica sob todas as políticas viáveis, conforme assumido no Capítulo 3.1, mesmo com o espaço de estado de cardinalidade infinita.

Agora, vamos definir uma sequência de tempos de parada $\tau_0 = 0$ e $\tau_k = \min\{t > \tau_{k-1} : X_t \in S_1^c\}$, $k = 1, 2, \ldots$, em que $S_1^c = S \setminus S_1$ é o complemento do subconjunto S_1. A cadeia de Markov embutida $Y_k \triangleq X_{\tau_k}$, $k \geq 0$ representa as trajetórias do

processo controlado X_t, $t \geq 0$ em S_1^c. Vamos definir também:

$$\bar{\beta} \triangleq \max_{j \in S_1^c} E^{\mathcal{L}} \left[\max \left(g(X_t) - g(X_{t+1}) \right) \mid X_t = j \right],$$

Adicionalmente, seja β uma variável aleatória tal que:

$$P(\beta = c) = \begin{cases} \dfrac{\bar{\beta}}{\lceil \bar{\beta} \rceil}, & \text{if } c = \lceil \bar{\beta} \rceil, \\ 1 - \dfrac{\bar{\beta}}{\lceil \bar{\beta} \rceil}, & \text{if } c = 0. \end{cases}$$

É possível perceber que $E(\beta) = \bar{\beta}$, em que β pode assumir apenas um dos seguintes valores: c ou 0. Agora, é possível representar o aumento acumulado na função Lyapunov $g : S \to \mathbb{Z}_+$ dentro do conjunto S_1^c pelo processo:

$$Z_{t+1} = \max\{0, Z_t + a_t - \beta\},\ Z_0 = 0, \tag{A.2}$$

em que

$$a_t \triangleq r(Y_t) = \beta + \left(g(X_1) - g(X_0) \right) \mid X_0 = Y_t,$$

é uma variável aleatória inteira modelada pelo processo Y_t, $t \geq 0$. É possível perceber também que o processo Z_t, $t \geq 0$ é estável se $E(a_t) < E(\beta)$, $\forall t \geq 0$.

O seguinte resultado é uma consequência direta das definições anteriores. Segue-se de (CIUCU *et al.*, 2019, Section 6.2 and Theorem 3) que:

$$\lim_{t \to \infty} P(Z_t > \sigma) \leq M e^{-\theta \sigma}, \tag{A.3}$$

para $0 < M < \infty$ e $\theta > 0$. O valor de θ depende da intensidade do drift do processo Z_t, $t \geq 0$, que por definição é igual ao drift em S_1^c na Eq. (A.1). O resultado mais importante da seção é o Teorema 8 seguinte:

Teorema 8. *Considere que $g : S \to \mathbb{Z}_+$ satisfaz a condição de estabilidade de Foster em* (A.1) *e $D = \{j \in S_1^c : \exists i \in S_1$ tal que $p_{ij}^{\mathcal{L}} > 0\}$ representa os estados em S_1^c que podem se diretamente acessados de S_1. Assuma que o processo de Markov controlado X_t, $t \geq 0$ é operado pela política $\pi \in \Pi$. Assim:*

$$\lim_{t \to \infty} P\left(g(X_t) > \bar{g} + \sigma \right) \leq M e^{-\theta \sigma}, \qquad \bar{g} = \max_{j \in D} g(j). \tag{A.4}$$

Demonstração. Eq. (A.4) deriva diretamente de (A.3), visto que $\bar{g}$ é o maior valor possível de $g(Y_t)$ no início da trajetória em S_1^c, quando $Z_t = 0$,uma vez que Z_t, $t \geq 0$ representa o incremento cumulativo do processo Y_t, $t \geq 0$ em F_1^c, e considerando que a trajetória tenha começado em $g(Y_t) \leq \bar{g}$. □