UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE ESTATÍSTICA E MATEMÁTICA APLICADA

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM E MÉTODOS QUANTITATIVOS

VITÓRIA INGRID TELES MENDONÇA

FORMULAÇÃO E SOLUÇÃO DO PROBLEMA DE ALOCAÇÃO DE VEÍCULOS ESTOCÁSTICO POR MEIO DE PROGRAMAÇÃO DINÂMICA APROXIMADA

FORTALEZA

2022

VITÓRIA INGRID TELES MENDONÇA

FORMULAÇÃO E SOLUÇÃO DO PROBLEMA DE ALOCAÇÃO DE VEÍCULOS ESTOCÁSTICO POR MEIO DE PROGRAMAÇÃO DINÂMICA APROXIMADA

Dissertação apresentada do Programa de Pós-Graduação em Modelagem e Métodos quantitativos do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestra em Modelagem e Métodos Quantitativos. Área de Concentração: Inteligência Computacional e Otimização.

Orientador: Prof. Dr. Anselmo Ramalho Pitombeira Neto

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M497f Mendonça, Vitória Ingrid Teles.
Formulação e solução do problema de alocação de veículos estocástico por meio de programação dinâmica aproximada / Vitória Ingrid Teles Mendonça. – 2022.
83 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, Fortaleza, 2022.
Orientação: Prof. Dr. Anselmo Ramalho Pitombeira Neto.

1. Otimização combinatória. 2. Programação dinâmica. 3. Processo de decisão semimarkoviano. 4. Simulação de eventos discretos. I. Título.

CDD 510

VITÓRIA INGRID TELES MENDONÇA

FORMULAÇÃO E SOLUÇÃO DO PROBLEMA DE ALOCAÇÃO DE VEÍCULOS ESTOCÁSTICO POR MEIO DE PROGRAMAÇÃO DINÂMICA APROXIMADA

Dissertação apresentada do Programa de Pós-Graduação em Modelagem e Métodos quantitativos do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestra em Modelagem e Métodos Quantitativos. Área de Concentração: Inteligência Computacional e Otimização.

Aprovada em: 26/07/2022

BANCA EXAMINADORA

Prof. Dr. Anselmo Ramalho Pitombeira Neto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Ricardo Coelho Silva
Universidade Federal do Ceará (UFC)

Prof. Dr. Tibérius de Oliveira e Bonates
Universidade Federal do Ceará (UFC)

Dedico esta dissertação às mulheres fortes Maria Albertina (*in memoriam*) e Socorro Mendonça, por todo esforço e dedicação em me ajudar a ser quem eu sou.

AGRADECIMENTOS

À minha mãe, Socorro Mendonça, e à minha avó Maria Albertina (*in memoriam*), por terem me ensinado o valor de uma vida digna, sempre semeando a esperança de que os estudos podem mudar o futuro de uma vida.

À Rahysa Oliveira, minha companheira de tantas horas, que esteve comigo desde o início da jornada do mestrado, sendo minha grande incentivadora.

Ao Prof. Dr. Anselmo Ramalho Pitombeira Neto, por ser um excelente professor, de coração tão humilde, sempre ensinando seus alunos com tanta maestria e humanidade. Sua orientação foi muito importante para que essa caminhada fosse leve e prazerosa.

Aos caros colegas do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, por todos os momentos de trocas de conhecimentos e parceria, e por todas as alegrias vivenciadas no saudoso laboratório.

Ao Prof. Dr. Juvêncio Santos Nobre, por ser um exemplo de excelência, inspiração e superação.

Aos professores e professoras do Departamento de Estatística e Matemática Aplicada que, por todos os anos, desde a graduação, em tantas disciplinas, fizeram parte desta minha jornada pessoal.

O presente trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001, da Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (FUNCAP) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, por meio do projeto 422464/2016-3, além do NVIDIA GPU Grant Program.

“Esta é a vida feliz: alegra-se em Ti, de Ti, por Ti ” (Agostinho de Hipona).

RESUMO

O problema de alocação de veículos dinâmico e estocástico é um problema com vasto campo de aplicações, que ganhou notória evidência desde o surgimento das plataformas de serviço de mobilidade urbana sob demanda. Neste trabalho, o problema de alocação de veículos consiste em alocar veículos aos chamados que chegam em uma central de atendimento em instantes de tempo aleatório objetivando minimizar o tempo de espera. Este problema é formulado como um problema de decisão semimarkoviano. Dado que o tamanho do espaço de estados, bem como do estados de decisão são pontos sensíveis para obtenção de uma política ótima, métodos de programação dinâmica aproximada se apresentam como um caminho alternativo para construir políticas de decisão aproximadas. Por isso, apresentamos uma solução para o problema de alocação de veículos baseada no algoritmo *rollout*, um algoritmo de iteração de política aproximada, ou seja, dado uma política inicial, constrói uma política melhorada baseado em duas etapas: avaliação da política e melhoria da política. As políticas produzidas pelo algoritmo *rollout* são do tipo *lookahead*, que em comparação as políticas míopes, apresentam a vantagem de conduzir as decisões que equilibram o retorno atual e o retorno futuro. O algoritmo *rollout* foi experimentado em um ambiente simulado baseado em simulação de eventos discretos, de modo que o processo natural do processo de decisão semimarkoviano foi modelado como um sistema de filas, controlando o processo de chegada de novos chamados, tempo de atendimento de cada veículo, e a disciplina da fila. Assim, o algoritmo *rollout* parte de três políticas base: FIFO, o chamado é escolhido priorizando o tempo de espera na fila; NV, o chamado é escolhido priorizando a distância entre os veículos; RANDOM, o chamado é escolhido aleatoriamente. E tem por objetivo melhorar a performance destas políticas. No geral, os resultados mostram que para todas estas políticas, o algoritmo *rollout* foi capaz de produzir uma redução média de até 69% do atraso acumulado médio.

Palavras-chave: problema de alocação de veículos; processo de decisão semimarkoviano; programação dinâmica aproximada; simulação de eventos discretos; otimização combinatória; otimização baseada em simulação; iteração de política aproximada; políticas heurísticas.

ABSTRACT

The dynamic and stochastic vehicle allocation problem is a problem with a vast field of applications, which has gained considerable evidence since the emergence of on-demand urban mobility service platforms. In this work, the vehicle allocation problem consists of allocating vehicles to calls that arrive at a call center at random times in order to minimize the waiting time. This problem is formulated as a semi-Markov decision problem. Given that the size of the state space as well as the decision states are sensitive points for obtaining an optimal policy, approximate dynamic programming methods are presented as an alternative way to build approximate decision policies. Therefore, we present a solution to the vehicle allocation problem based on the *rollout* algorithm, an approximate policy iteration algorithm, that is, given an initial policy, it builds an improved policy based on two steps: policy evaluation and policy improvement. The policies produced by the *rollout* algorithm are of the *lookahead* type, which, compared to myopic policies, have the advantage of leading to decisions that balance current and future returns. The *rollout* algorithm was tested in a simulated environment based on discrete-event simulation, so that the natural process of the semi-Markovian decision process was modeled as a queuing system, controlling the arrival process of new tickets, service of each vehicle, and the discipline of the queue. Thus, the *rollout* algorithm is based on three basic policies: FIFO, the ticket is chosen prioritizing the waiting time in the queue; NV, the call is chosen prioritizing the distance between vehicles; RANDOM, the call is chosen at random. And it aims to improve the performance of these policies. Overall, the results show that for all these policies, the *rollout* algorithm was able to produce an average reduction of up to 69% of the average accumulated delay.

Keywords: stochastic vehicle allocation problem ; semi-Markov decision processes; approximate dynamic programming; discret event simulation; combinatorial optimization; simulation based optimization; approximate policy iteration; heuristics policies.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SÍMBOLOS

S	Espaço de estados
A	Espaço de decisões
T	Conjunto de épocas de decisão
V	Espaço das funções de valor
t_k	Época de decisão k
s, s^j	Estados
a, a^j	Decisões
$p(s^j \mid s, a)$	Probabilidade de estar no estado s^j dado decisão a e estado s
C_k	Retorno imediato
$c(s, a)$	Retorno imediato
G_k	Soma dos retornos obtidos a partir de k
π, π^j	Política
$X^{\pi}(\cdot)$	Regra de decisão
γ	Fator de desconto
$v_t(\cdot)$	Função de valor estado em t
$q_t(\cdot)$	Função de valor estado-decisão em t
$v^{\Lambda}(\cdot)$	Função de valor estado ótima
$q^{\Lambda}(\cdot)$	Função de valor estado-decisão ótima
$v_t(\cdot)$	Aproximação da função de valor estado em t
ϵ	Taxa de convergência para algoritmos de programação dinâmica
$Q(j, \tau \mid i, a)$	Probabilidade do próximo estado j ocorrer dentro do intervalo τ dado que no estado i foi aplicada a decisão a
$P(j \mid i, a)$	Probabilidade de transição para o estado j em qualquer intervalo de tempo dado que no estado i foi aplicada a decisão a
$p(\tau \mid i, a, j$	Probabilidade que a próxima época de decisão ocorra no intervalo τ dado que a decisão a aplicada sobre o estado i evolui o sistema para o estado j
$C(s, a)$	Taxa de retorno fixa global do PDSM

$c(i, a, j)$	Taxa de retorno adicional do PDSM
$r(i, a)$	Retorno total quando no estado i é aplicada a decisão a
$g_{\pi(i)}$	Retorno médio da política π em um PDSM.
$S_{s_k}^{a_K}$	Espaço dos possíveis estados subsequentes ao estado s_k dado a decisão a_k
H	Heurística base
h	Trajetória de estados futuros
π_H	Política base
$R(t)$	Conjunto de chamados em espera no instante t
r_j	Tupla com os atributos do chamado j
o_j	Origem do chamado j
d_j	Destino do chamado j
w_j	Instante de chegada do chamado j
$V(t)$	Conjunto de veículos disponível no instante t
$v_j(t)$	Tupla com os atributos do veículo j
$l_j(t)$	Localização do veículo j
$b_j(t)$	Disponibilidade do veículo j

SUMÁRIO

1 INTRODUÇÃO

1.1 PROBLEMA DE ALOCAÇÃO CLÁSSICO

O problema de alocação (*assignment problem*) é um problema bastante conhecido da classe de problemas de otimização combinatória. O primeiro método apresentado para resolver o problema de alocação determinístico e estático, conhecido por método húngaro, foi proposto por Kuhn (1955). O problema de alocação "envolve obter a correspondência ótima dos elementos de dois ou mais conjuntos" (Pentico, 2007, p.1). A versão clássica do problema compreende dois conjuntos, denominados genericamente de recursos e tarefas, de modo que é desejável alocar cada recurso a uma tarefa diferente, e cada tarefa deve ser atendida por um único recurso.

Considere n tarefas e n recursos. Deseja-se alocar recursos às tarefas de modo a maximizar o retorno total recebido associado as alocações realizadas. Um modelo matemático para o problema de alocação clássico é dado por:

$$max \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}; \tag{1.1}$$

$$s.a \quad \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n; \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n; \tag{1.3}$$

sendo as variáveis do modelo:

$$x_{ij} = \begin{cases} 1, & \text{se o recurso } i \text{ é alocado à tarefa } j; \\ 0, & \text{caso contrário;} \end{cases}$$

e os parâmetros da função-objetivo c_{ij} correspondem ao retorno recebido por alocar o recurso i à tarefa j. O conjunto de restrições (1.2) garante que para cada tarefa j haverá um único recurso alocado a ela. O segundo conjunto de restrições (1.3) garante que para cada recurso haverá uma única tarefa que receberá sua alocação.

1.2 PROBLEMA DE ALOCAÇÃO DE VEÍCULOS

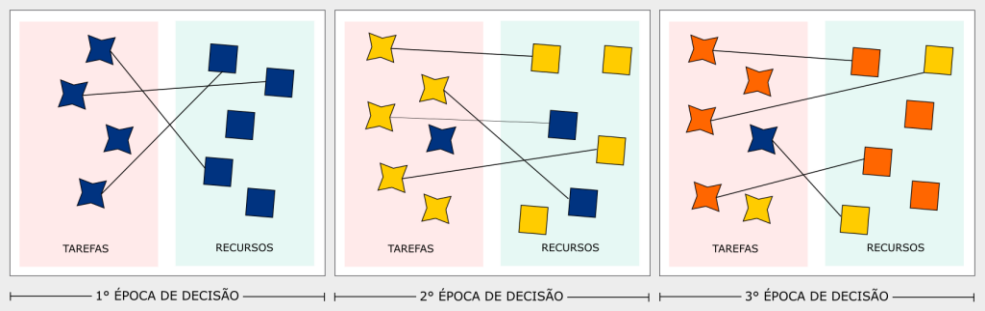
O problema de alocação clássico pode ser modelado de forma que os recursos sejam veículos e as tarefas sejam requisições de uso destes veículos. Neste caso, o problema passa a ser

denominado de problema de alocação de veículos. Em uma versão mais elementar do problema, tal como apresentada na seção anterior, as decisões ocorrem estática e deterministicamente. Porém, existem outras configurações em que o problema pode apresentar comportamentos estocásticos. Adicionalmente, também pode ocorrer que as decisões de alocação precisem ser tomadas sequencialmente. Os problemas com estas duas características recebem o nome de problema de alocação de veículos dinâmico e estocástico.

Ao longo do texto, sempre que for referenciado o termo “problema de alocação de veículos” ele será utilizando de modo equivalente ao termo “problema de alocação de veículos dinâmico e estocástico”. O problema de alocação de veículos dinâmico e estocástico será o objeto de estudo neste trabalho.

Como dito anteriormente, o problema de alocação de veículos é uma aplicação particular do problema de alocação. Por isso, primeiramente será apresentado ao leitor uma configuração genérica do problema de alocação dinâmico e estocástico.

Figura 1 – Representação da evolução do processo de decisão em um problema de alocação dinâmico estocástico.

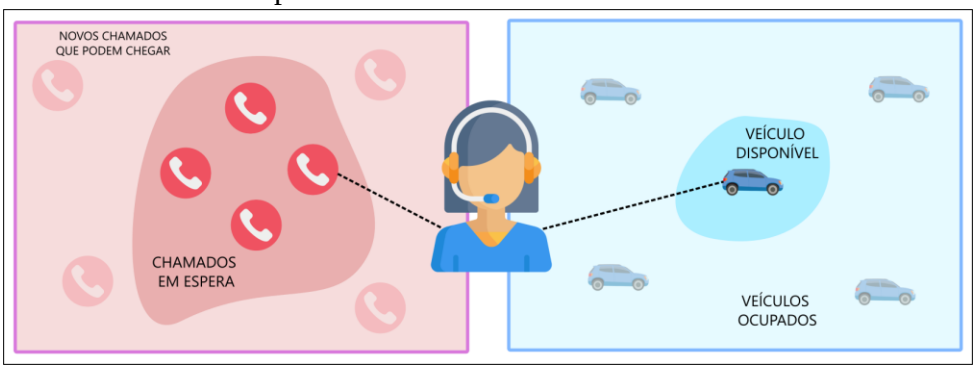


Fonte: Elaborado pela autora (2022)

Inicialmente considere o cenário quando em um dado instante de tempo existe um conjunto de recursos e um conjunto de tarefas, tal que estes recursos precisam ser atribuídos a estas tarefas a fim de atender um determinado critério de otimalidade. Quando a decisão de atribuição sobre estes dois conjuntos é realizada, os recursos e tarefas que fizeram parte da alocação deixam de pertencer aos seus respectivos conjuntos e novos recursos e tarefas tornam-se acessíveis no ambiente. Este processo de chegada de novos recursos e tarefas ao ambiente é um processo aleatório. Assim, após a alocação pode-se dizer que o sistema evoluiu para um novo cenário. Estas etapas de decisão e evolução ocorrem repetidas vezes ao longo de um horizonte temporal. Este fluxo é exemplificado na Figura 1 e ele representa o que é denominado de problema de alocação dinâmico (*dynamic assignment problem*).

Da mesma forma, o problema de alocação de veículos pode ser apresentado. Assumindo que os recursos do problema são veículos que compõem uma frota, esses veículos devem ser alocados para atender chamadas (tarefas) que chegam a uma central de atendimento. Estes chamados são solicitações por um serviço de transporte. As tarefas tornam-se conhecidas aleatoriamente ao longo do tempo. Assim, em diferentes instantes de decisão, a central de atendimento, que assume este papel de decisor, deve direcionar qual veículo realizará qual atendimento (veja a Figura 2).

Figura 2 – Ilustração genérica do mapeamento entre chamados e veículos realizado por uma central de atendimento.



Fonte: Elaborado pela autora (2022)

O problema de alocação de veículos oferece oportunas possibilidades de modelagens e soluções, principalmente considerando a favorável visibilidade promovida por plataformas de mobilidade sob demanda, tais como Uber, Lyft e Didi. Com a ampla adesão ao uso das plataformas de mobilidade sob demanda, os provedores de serviço foram impelidos a otimizar o processo de alocação dos veículos aos chamados, garantindo que o processo pudesse ser o mais automatizado possível. No início, os provedores de serviço faziam uso de regras de decisão míopes bastante elementares, como por exemplo, realizar a alocação ao veículo cuja previsão do tempo de chegada ao local da coleta fosse a menor possível. Esta abordagem é conhecida como protocolo de primeira expedição (Yan *et al.*, 2020). Özkan e Ward (2020) partem do modelo de programação linear para o problema de alocação, tal como o apresentado na seção 1.1 e desenvolvem políticas de decisão baseadas na formulação de programação linear do problema de alocação estocástico, em que a demanda e a oferta chegam de acordo com um processo de Poisson não homogêneo e as decisões de alocação são feitas imediatamente após cada solicitação. Entretanto, é importante destacar que estas abordagens levam consigo algumas desvantagens, pois não há visibilidade sobre veículos que no momento da decisão eventualmente estejam ocupados, mas que num pequeno intervalo de tempo futuro, quando disponíveis, se tornariam

aqueles que estariam a menor distância do chamado.

Alternativamente, alguns servidores têm implementado uma abordagem de correspondência, na qual a alocação dos veículos aos chamados ocorre em intervalos de tempo fixos a fim de otimizar uma função objetivo, como tempo total em rota ou receita total (Qin *et al.*, 2020). Embora essa abordagem de correspondência possa render melhor desempenho quando comparado com o protocolo de primeiro despacho, ainda é uma política míope, ou seja, não leva em consideração o impacto das decisões atuais em momentos futuros do sistema. Aplicar decisões de forma míope pode prejudicar o desempenho e até provocar fenômenos indesejáveis, como por exemplo rapidamente despachar todos os veículos disponíveis em um dado momento a fim de atender um pico de demanda, e em seguida ter os únicos veículos ociosos muito distante dos próximos chamados (Yan *et al.*, 2020).

Para melhorar o desempenho das decisões, novas políticas de decisão foram propostas, tal como política *lookahead*. Estas políticas aplicam decisões no momento presente otimizando explicitamente algum horizonte, combinando alguma aproximação de informações futuras com alguma aproximação de ações futuras (Powell, 2011). Dentro da classe de políticas *lookahead* existem algumas abordagens tais como árvore de busca (*tree search*), que realiza uma estratégia que enumera todas as ações possíveis e todos os resultados possíveis em algum horizonte; árvore de busca de amostragem esparsa (*sparse sampling tree search*) que contempla casos onde o espaço das decisões não é consideravelmente grande, e substitui o mapeamento de todas as possíveis decisões por uma amostragem; *rollout*, direcionado para problemas com espaço de decisões grande, substitui a enumeração explícita de toda a árvore por algum tipo de decisão heurística para avaliar o que pode acontecer depois que um certo estado do ambiente é visitado.

Neste trabalho, as características do problema analisado são diferentes da aplicação prática das plataformas de mobilidade, por isso, prefere-se dizer que o problema de alocação de veículos exposto se assemelha mais ao problema de despacho de veículos. Partindo disto e de todas as vantagens que existem no uso de políticas *lookahed*, a solução proposta se fundamenta nessa classe de políticas, mais especificamente em políticas do tipo *rollout*.

Então, a fim de explorar as vantagens de uma política *lookahead*, o problema de alocação de veículos será modelado como um problema de decisão sequencial, onde a Programação Dinâmica oferece uma estrutura robusta o suficiente para mapear o processo de evolução do ambiente de decisão e, portanto, possibilitar que a política *lookahead* equilibre o

impacto das decisões no presente e no futuro. Assim, ao longo deste trabalho será explorado conceitos de decisões sequenciais, programação dinâmica e políticas aproximadas, para por fim apresentar ao leitor a formulação e solução para o problema de alocação de veículos.

1.3 OBJETIVOS E MOTIVAÇÕES

Este trabalho tem por objetivo principal apresentar uma modelagem para o problema de alocação de veículos formulando sua dinâmica e seu processo de decisão como um processo de decisão semimarkoviano. Adicionalmente, pretende-se avaliar uma política de decisão aproximada obtida a partir de um algoritmo *rollout* baseado em simulação de eventos discretos, e realizar experimentos computacionais demonstrando que as decisões seguindo esta política de fato são tão boas quanto, ou melhores que as decisões seguindo somente a política base.

A motivação para a escolha da metodologia abordada neste trabalho parte de algumas vantagens encontradas em utilizar políticas *rollout*, tais como:

1. Tais políticas são conceitualmente simples e relativamente fáceis de implementar, sem a exigência do uso de parâmetros pré-definidos, tais como em métodos de aproximação paramétricos;
2. Elas gozam de garantias teóricas de desempenho, que derivam do teorema de melhoria de política (Bellman, 1957);
3. Muitos problemas têm regras de decisão heurísticas com boas propriedades de desempenho e estabilidade. Uma política *rollout* pode utilizá-las como base para produzir soluções melhores.

1.4 ESTRUTURA DO DOCUMENTO

Este texto segue apresentando um capítulo de fundamentação teórica (capítulo 2), definindo primeiramente o problema de decisão sequencial (seção 2.1). A seção 2.2.1 apresenta uma clássica estrutura para modelar problemas de decisão sequencial. Na mesma seção são expostos os principais elementos de um processo de decisão markoviano, bem como os principais algoritmos para resolver problemas de decisão sequencial. Em seguida, é discutido como um processo de decisão markoviano pode não ser a melhor estrutura para certos problemas estocásticos, e então processos de decisão semimarkovianos são apresentados como um caminho eficiente para lidar com problemas que evoluem continuamente ao longo do tempo (seção 2.3).

O capítulo de fundação teórica é concluído abordando o algoritmo *rollout*, mostrando suas condições de convergência e as vantagens associadas ao seu uso.

No capítulo 3 é apresentado um breve resumo de trabalho relacionados. Por fim, no capítulo 4 o problema de alocação de veículos é formulado sob os moldes de um PDSM. Os experimentos computacionais e seus resultados estão expostos no capítulo 5 e as conclusões do trabalho estão expostas no capítulo 6.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 PROBLEMA DE DECISÃO SEQUENCIAL

Um problema de otimização, em sua forma mais fundamental, envolve realizar decisões que respeitem alguns critérios de viabilidade a fim de maximizar ou minimizar o resultado de uma certa função. Os problemas mais elementares envolvem otimizar decisões aplicadas em um único estágio. Problemas deste tipo são modelados a partir da seguinte forma:

$$\begin{aligned} \max_{x} \quad & f(x) \\ s.a \quad & x \in \Omega, \end{aligned}$$

sendo que x corresponde a variável de decisão, Ω é o espaço de decisões viáveis, $f(\cdot)$ é a função a ser otimizada, conhecida por função objetivo, e

$$x^* \in \arg\max_{x \in \Omega} f(x)$$

é uma solução ótima.

Alguns problemas exigem uma modelagem diferente, em que decisões são aplicadas sequencialmente ao longo de um determinado intervalo de otimização. Neste sentindo, a solução para o problema de otimização será uma tupla $(x_1, x_2, \cdots, x_n) \in \Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$, onde x_i e Ω_i são respectivamente, a variável de decisão e o espaço de decisões viável no estágio de decisão i, $i \in \{1, 2, \cdots, n\}$. A forma do problema de otimização sequencial é representada por:

$$\begin{aligned} \max_{(x_1, x_2, \cdots, x_n)} \quad & f(x_1, x_2, \cdots, x_n) \\ s.a. \quad & x_i \in \Omega, \quad \forall i. \end{aligned}$$

Uma das formas de resolver este problema é tratá-lo como um problema de otimização combinatória, enumerando todas as possíveis soluções e avaliando-as independentemente. Em grande parte dos problemas combinatórios, o número de soluções cresce exponencialmente à medida em que o número de decisões aumenta. Quando se trata de problemas das classes NP ou NP-HARD, resolvê-los seguindo uma abordagem exaustiva é computacionalmente impraticável, dado que o espaço de soluções viáveis pode facilmente se tornar potencialmente grande, até mesmo em aplicações com objetivos didáticos (Lew; Mauch, 2006).

Problemas de decisão sequencial também podem ser resolvidos recursivamente. A formulação para uma abordagem recursiva poderia ser:

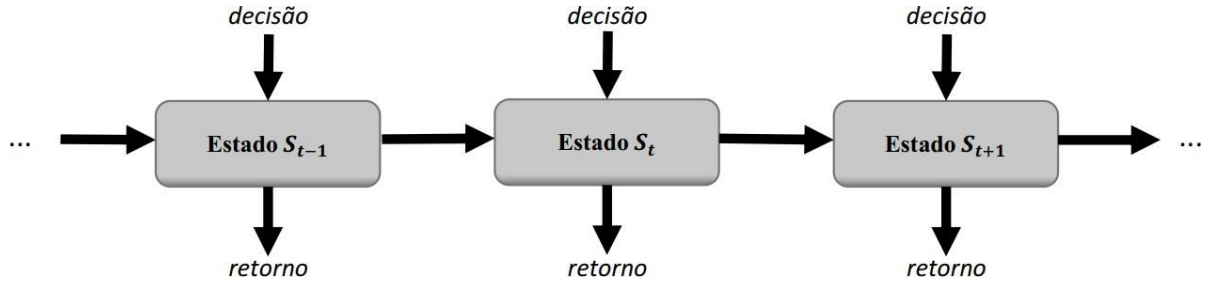
$$
\begin{aligned}
f_n^* &= \max_{x_n} f_n(x_n) \\
&: \\
f_2^* &= \max_{x_2} f_2(x_2, f_3^*) \\
f_1^* &= \max_{x_1} f_1(x_1, f_2^*) \\
s.a \quad & x_i \in \Omega_i \quad \forall i \in \{1, 2, \cdots, n\}.
\end{aligned}
$$

O problema de otimização, tal como apresentado acima, a partir de um modelo genérico, pode representar o que é conhecido por problema de decisão sequencial. Este problema surge em contextos nos quais decisões são aplicadas sequencialmente ao longo de um horizonte de decisão, e cada decisão incide em um retorno que pode incorrer imediatamente ou após um intervalo arbitrariamente longo. Além disso, ao realizar uma decisão em um dado momento do processo sequencial, a decisão que será escolhida é fortemente influenciada por decisões anteriores, e invariavelmente afetará as decisões posteriores (Sniedovich, 2010). Portanto, o objetivo é otimizar a sequência de decisões, e não tomar decisões consecutivas de forma independente. Logo, deseja-se obter o melhor ajuste entre o benefício de curto e longo prazo. A relevância desta classe de problemas de otimização ao longo do tempo surge em várias configurações, que podem envolver gestão de recursos físicos, financeiros, permeando as áreas da computação, pesquisa operacional e engenharia (Powell, 2011).

Um problema de decisão sequencial pode ser modelado sob diferentes níveis de abstração, com algumas características que variam de acordo com o problema modelado. A Figura 3 representa o fluxo de um processo de decisão sequencial determinístico. Nesta configuração determinística, em um estado específico do horizonte temporal de decisão, um agente, decisor ou controlador observa o estado do sistema, e então aplica uma decisão. A decisão produz dois resultados: é gerado um retorno, que pode ser uma recompensa ou punição, e o sistema evolui para um estado subsequente (Puterman, 2005). Sendo este processo determinístico, uma mesma decisão aplicada sobre um mesmo estado, invariavelmente produzirá o mesmo retorno e o processo transicionará sempre para o mesmo estado seguinte.

É possível ampliar o cenário e pensar em um contexto cuja evolução dos estados não ocorra de forma determinística como ilustrado na Figura 3, mas no caso em que há uma

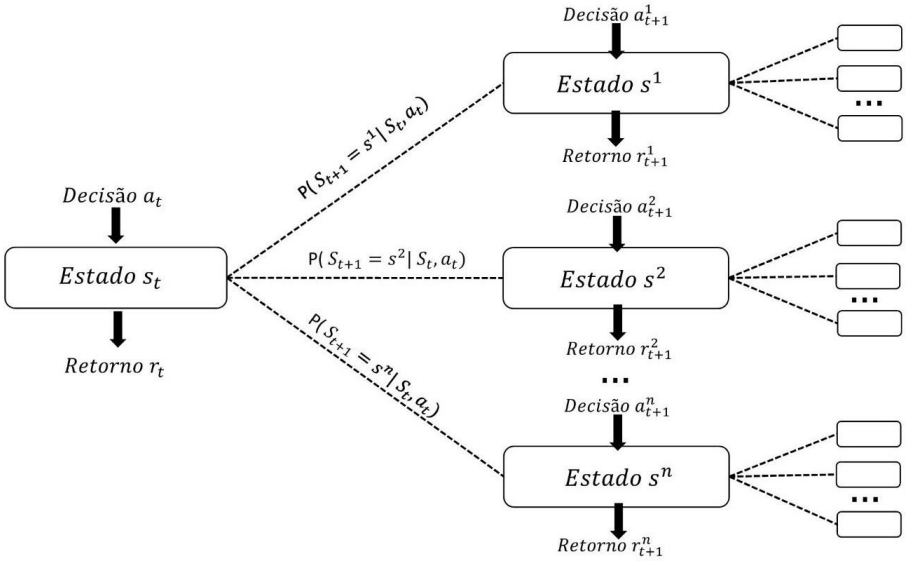
Figura 3 – Representação do fluxo de um processo de decisão sequencial determinístico.



Fonte: PUTERMAN (2005), adaptado pela autora (2021)

distribuição de probabilidade, que pode ser conhecida ou não, que rege esta evolução, conferindo ao sistema uma caracterização estocástica/aleatória (Figura 4). Isto significa que, se repetidas vezes uma mesma decisão for aplicada sobre o mesmo estado, o estado resultante da evolução não será necessariamente o mesmo, havendo uma probabilidade condicionada ao estado e decisão atual, possibilitando diferentes evoluções de estado no sistema.

Figura 4 – Representação do fluxo de um processo de decisão sequencial estocástico.



Fonte: Elaborado pela autora (2022)

A abordagem em (Barreto *et al.*, 2010) para modelar um problema de decisão sequencial pode ser classificada como:

1. **Markoviano ou não markoviano:** Em problemas caracterizados como markovianos, as transições e recompensas entre estados dependem somente do atual estado e da decisão aplicada sobre ele. Isto implica dizer que neste contexto, um estado markoviano detém em si mesmo todas as informações relevantes suficientes para descrever a condição do

ambiente e prover ao decisor o necessário para aplicar uma ação. Assim, uma vez que o estado atual seja conhecido, o histórico de estados antecessores é desprezível para o propósito da tomada de decisão. O caso não markoviano é o oposto do apresentado anteriormente, o decisor precisa do histórico de estados antecessores ao atual para ser capaz de absorver as informações minimamente necessários para aplicar uma decisão.

2. **Determinístico ou estocástico:** Em um problema de decisão sequencial determinístico, ao aplicar uma decisão sobre um dado estado do sistema, o estado imediatamente posterior será invariavelmente o mesmo. Em contraste, em um ambiente estocástico há uma distribuição de probabilidade associada à transição entres os estados. Logo, o sistema pode ocupar diferentes estados a partir da execução de uma mesma ação em um mesmo estado em diferentes instantes de decisão. Note que o caso determinístico pode ser interpretado como um subproblema estocástico, em que ao aplicar uma decisão sobre um estado, para todos os possíveis estados subsequentes, apenas um possui probabilidade de ser ocupado igual a um e todos os outros possuem probabilidade nula.
3. **Estacionário ou não estacionário:** Em problemas de decisão estacionários, a regra de decisão aplicada sobre qualquer que seja a época de decisão segue a mesma lógica ou função de escolha. Ou seja, as regras que ditam as transições entre estados não variam com o tempo. No caso em que o ambiente de decisão é não estacionário, tem-se que a regra de decisão aplicada sobre os estados depende diretamente do instante no qual o ambiente está, de modo que existem diferentes funções de decisão para diferentes épocas de decisão.

No que diz respeito aos níveis de classificação de um problema de decisão sequencial, se for necessário, pode-se avançar em outros níveis de classificação mais específicos, tais como os apresentados por Barreto *at al.* (2010, p.195). No entanto, para o que se pretende analisar neste trabalho, estes três grupos são suficientes para amparar as discussões sobre o problema estudado.

Em resumo, as estruturas de decisão apresentadas anteriormente (Figura 3, Figura 4) são construídas a partir destes elementos (Puterman, 2005):

1. Um conjunto de épocas de decisão;
2. Um conjunto de estados do ambiente;
3. Um conjunto de decisões viáveis;
4. Um conjunto de recompensas ou custos imediatos dependentes do estado e decisão;
5. Um conjunto de probabilidades de transição dependentes do estado e decisão.

Este modelo é desenvolvido a partir do que se conhece por programação dinâmica (*dynamic programming*). A Programação Dinâmica (PD) é uma técnica matemática para resolver problemas de decisão sequencial baseado em computação indutiva"(Puterman, 2005). A PD também pode ser entendida como "uma classe de algoritmos que resolve problemas complexos combinando soluções de seus subproblemas"(Geramifard *et al.*, 2013).

A técnica de Programação Dinâmica surgiu da necessidade de se trabalhar com problemas de decisão sequencial de forma mais eficiente, uma vez que as técnicas disponíveis até então se tornavam custosas e ineficientes quando se tratava de um problema caracterizado como estocástico. Então, em 1952 Richard Bellman apresentou o primeiro estudo intitulado de *On the theory of dynamic programming* (Bellman, 1952). O termo Programação Dinâmica foi empregado pela primeira vez pelo próprio Bellman, que ao elaborar o que ele denominou de "o princípio da otimalidade", se viu na necessidade de nomear o que ele estava desenvolvendo.

O estudo dos processos de decisão sequencial, a partir da construção da Programação Dinâmica começou a ser popularizado por Richard Bellman (Bellman, 1957) e Ronald Howard (Howard, 1960). Desde então, a Programação Dinâmica segue em desenvolvimento, impulsionada principalmente pelo avanço obtido em capacidade computacional que favoreceu uma pluralidade de estudos em diversos campos de aplicação.

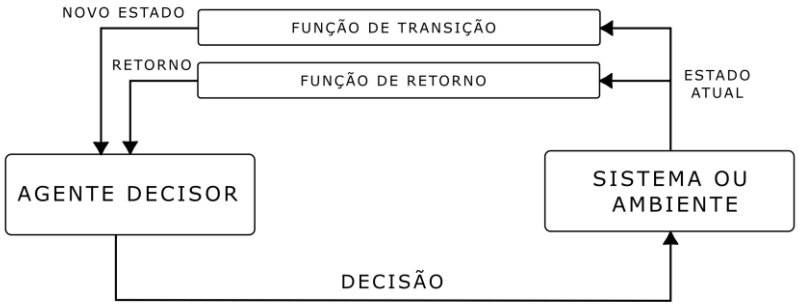
2.2 PROCESSO DE DECISÃO MARKOVIANO

2.2.1 A estrutura de um processo de decisão markoviano

Um processo de decisão markoviano (PDM) é uma formalização clássica do processo de tomada de decisão sequencial, em que as ações influenciam não somente na geração de recompensas imediatas, mas também situações subsequentes, ou estados, através de recompensas futuras (Sutton; Barto, 2018). Um PDM é apresentado como um método para lidar diretamente com o ambiente de decisão tal como ilustrado na Figura 5. Em resumo, esta dinâmica ocorre da seguinte forma: em um dado momento inicial, um agente decisor, orientado por uma regra de decisão, aplica uma decisão sobre o ambiente baseado em informações internas e/ou externas a este ambiente. Esta ação promove a evolução do ambiente, e a partir do estado e da decisão aplicada, uma função de transição produz o novo estado para o ambiente, e adicionalmente, uma função de retorno gera uma recompensa ou punição que indica a qualidade da decisão aplicada. Concluído este ciclo, o agente decisor se encontra em um novo estado e o processo é repetido,

agora munido de novas informações adquiridas do ciclo anterior que servirão de apoio para a nova decisão.

Figura 5 – Dinâmica de iteração entre decisor e o ambiente em um problema de decisão sequencial.



Fonte: Elaborado pela autora (2022)

A seguir, cada um destes elementos que compõe a dinâmica de iteração entre o decisor e o ambiente no processo de tomada de decisão de um problema sequencial serão apresentados.

2.2.1.1 Variável de estado

Seja S o conjunto de todas as variáveis de estados viáveis. Define-se uma variável de estado $s \in S$ como uma variável cujo valor especifica completamente o cenário instantâneo do processo de decisão. As variáveis de estado devem oferecer ao decisor tudo o que ele precisa saber sobre o ambiente a fim de aplicar sua decisão (Howard, 2007). Assim, uma variável de estado é uma função minimamente dimensionada do histórico que oferece as informações necessárias para capturar o que é substancial para modelar dinâmicas futuras (Powell, 2003).

No caso de uma variável de estado de um processo markoviano, o valor por ela assumido carrega em si informações sobre a sequência de decisões passadas, e cada estado traduz uma situação particular no qual se encontra o processo. A quantidade de atributos que serão utilizados na composição da variável de estado é totalmente arbitrária. Entretanto, é fundamental modelar está variável considerando apenas as informações necessárias e suficientes para o decisor, pois à medida em que são adicionados novos atributos, maior será a dimensão do espaço desta variável e consequentemente maior o nível de complexidade no seu tratamento.

2.2.1.2 Variáveis de decisão

A partir da observação do ambiente por meio das informações extraídas das variáveis de estado, o decisor tem a oportunidade de atuar sobre o ambiente modificando o seu estado. Então, esteja o decisor no estado $s \in S$, ele pode escolher uma decisão $a \in A_s$, onde A_s é o conjunto com todos as decisões viáveis para o estado s. O conjunto de todas as decisões viável é dado por $A = \bigcup_{s \in S} A_s$.

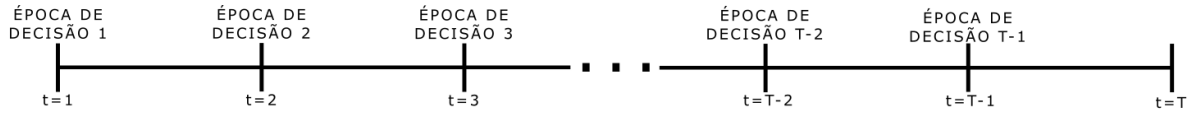
O decisor é requisitado a tomar decisões em pontos específicos ao longo da dinâmica do processo. Estes pontos de decisão são conhecidos como épocas de decisão, e a união de todas as épocas de decisão geram o horizonte de decisão. Um horizonte de decisão pode ser classificado como finito ou infinito.

- **Horizonte finito:** Seja T o conjunto ordenado de todas as épocas de decisão. Quando T é finito, temos um horizonte de decisão finito (Fig. 6). Em um PDM que possui esta característica, a época de decisão juntamente à variável de estado, é um fator de dependência que influência o comportamento do decisor.
- **Horizonte infinito:** Se T for um conjunto infinito, tem-se que o horizonte de decisão do modelo é um horizonte infinito. Neste caso, a informação temporal sobre qual época de decisão se encontra o sistema é irrelevante para o decisor.

Um processo de decisão também pode ser classificado no que diz respeito aos intervalos entre as épocas de decisão. Em resumo, um processo pode ser caracterizado como um processo de tempo contínuo ou tempo discreto.

- **Processos de tempo contínuo:** É um processo no qual o intervalo entre as épocas de decisão é dado por uma variável aleatória. Assim, para quaisquer duas épocas de decisão subsequentes, o intervalo entre elas não será necessariamente o mesmo que o intervalo em outras duas épocas de decisão subsequentes diferentes.
- **Processos de tempo discreto:** Em processos deste tipo, o intervalo entre as épocas de decisão é igual para qualquer que seja o par de épocas de decisão subsequentes.

Figura 6 – Representação de um horizonte de decisão discreto e finito.



Fonte: Elaborado pela autora (2022)

2.2.1.3 Função de transição

Considere um sistema de tempo discreto, sendo S o espaço de estados em um conjunto finito. Após o decisor observar o ambiente na época de decisão $t \in T$, suponha que no estado s o decisor escolhe a decisão $a \in A_s$. Independentemente do histórico de estados visitados anteriormente, o sistema transicionará para um estado qualquer s'. Esta transição é regida por uma função de transição que orienta a dinâmica do sistema. Uma função de transição depende do atual estado, da decisão aplicada sobre este estado e da informação exógena ao sistema para definir o próximo estado do ambiente.

Relembrando os conceitos apresentados anteriormente, a dinâmica do sistema pode ser determinística ou estocástica. A fim de reforçar o conceito, tem-se que um processo pode ser classificado como:

- **Determinístico:** Em um processo determinístico, se o decisor aplicar uma decisão a sobre o estado s em diferentes épocas de decisão, o estado resultante desta intervenção será invariavelmente s'. Assim, a função de transição é uma função definida como $f : S \times A \rightarrow S$.
- **Estocástico:** Em contrapartida, no caso estocástico há um fator aleatório operando sobre o ambiente, de modo que os estados resultantes da atuação do decisor sobre o ambiente a partir da aplicação de uma decisão é uma variável aleatória. Portando, temos que a função de transição será uma função tal que $p : S \times A \rightarrow [0, 1]$. Note que, a transição entre os estados neste caso é regida por uma distribuição probabilística. Seja S uma variável aleatória que representa o estado atual, e S' a variável aleatória representando o estado seguinte. Se no estado $S = s$ o decisor escolhe a decisão $A = a$, sendo A uma variável aleatória, a probabilidade do próximo estado ser s' é dada por:

$$p(s'|s, a) = P(S' = s'|S = s, A = a). \tag{2.1}$$

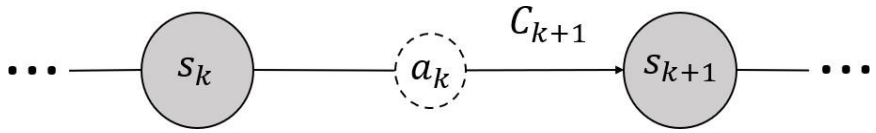
2.2.1.4 Função objetivo

Em um processo de decisão sequencial, o objetivo do decisor é gerar uma sequência de decisões cuja recompensa total acumulada seja maximizada, ou minimizado se o retorno for interpretado como custo. Portanto, é importante e indispensável que o modelo possua uma forma de mensurar a qualidade das decisões. Assim, a função objetivo em um processo de decisão markoviano é o elemento do modelo que exerce esse papel avaliador das decisões.

Assuma que na k-ésima época de decisão, ao aplicar uma decisão a_k sobre um estado s_k é obtido um retorno $C_{k+1} \in \mathrm{R}$, cujo fluxo está representado na Figura 7. O retorno $C_{k+1} = c(s_k, a_k)$ é recebido imediatamente após a aplicação da decisão sobre o sistema. Desta forma, quando o sistema transicionar sequencialmente entre os estados a partir de um instante k, uma sequência de retornos $C_{k+1}, C_{k+2}, C_{k+3}, \cdots, C_N$ é gerada, em que N é considerado o instante final. Considerando que o objetivo seja otimizar o retorno acumulado, isto implica dizer que se deseja otimizar a soma dos retornos imediatos

$$G_k = C_{k+1} + C_{k+2} + C_{k+3} + \cdots + C_N. \tag{2.2}$$

Figura 7 – Fluxo da etapa de transição entre estados que produz um retorno imediato para o sistema.



Fonte: Elaborado pela autora (2022)

Quando $N \to \infty$, que é o caso onde o horizonte de decisão é infinito, não há garantia de convergência na soma que resulta em G_k. Para contornar esta limitação, é aplicada uma técnica que computa no presente o valor de retornos futuros. Portanto, o retorno acumulado passa a ser

$$G_k = C_{k+1} + \gamma C_{k+2} + \gamma^2 C_{k+3} + \cdots = \sum_{\alpha=0}^{\infty} \gamma^{\alpha} C_{k+\alpha+1}, \tag{2.3}$$

onde γ é um parâmetro, $0 \leq \gamma \leq 1$, denominado de fator de desconto.

O fator de desconto determina o valor presente de retornos futuros então um retorno recebido no instante k, no futuro vale apenas $\gamma^{\alpha-1}$ vezes o que valeria se fosse recebido imediatamente. Se $\gamma < 1$, então G_k terá um valor finito desde que a sequência de retornos imediatos seja limitada. Se $\gamma = 0$, o decisor será míope, o que implica dizer que as decisões são tomadas considerando apenas os retornos imediatos, o retorno futuro não é contemplado pelo decisor (Sutton; Barto, 2018).

Assumindo um ambiente de decisão que evolui estocasticamente, a soma dos retornos acumulados será avaliada a partir do valor esperado dos retornos acumulados, tal como apresentado abaixo:

$$\mathrm{E}\left[\sum_{\alpha=0}^{\infty} \gamma^{\alpha} C_{k+\alpha+1}\right]. \tag{2.4}$$

Em resumo, a função objetivo de um processo de decisão markoviano opera a fim de otimizar os retornos acumulados obtidos para qualquer que seja o estado inicial.

2.2.1.5 Políticas

Uma política em um PDM pode ser compreendida como um mapeamento que associa uma decisão viável a cada estado do sistema (Barto *et al.*, 1989). Uma política especifica qual regra de decisão será usada em cada época de decisão, onde uma regra de decisão é uma função que determina uma decisão para um dado estado (Puterman, 2005).

Seja a função $x : S \rightarrow A$ uma regra de decisão, definimos uma política π como uma sequência de regras de decisões, ou seja, $\pi = (x_1, x_2, \cdots, x_k, \cdots, x_{N-1})$, onde x_k é a regra de decisão da k-ésima época de decisão. Note que em um ambiente de decisão determinístico a otimização é feita sobre uma única sequência de decisões. Vale destacar que se encontra uma grande diferença entre o contexto determinístico e estocástico, pois no ambiente estocástico a otimização é feita sobre uma política, ou seja, desejamos obter um família de funções que otimize o retorno obtido. Assim, políticas são objetos mais gerais do que uma sequência de decisões, e na presença de incertezas estocásticas, podem resultar retornos melhorados, visto que permitem escolhas de decisões que incorporam o conhecimento do estado (Bertsekas, 2019).

Uma política pode ser classificada em duas categorias: não-estacionárias e estacionárias. Uma política π é dita ser não-estacionária se cada regra de decisão da sequência gerada por esta política é particular para cada época de decisão, ou seja , $\pi = (x^{\pi}_1, x^{\pi}_2, \cdots, x^{\pi}_{N-1})$. Em contrapartida, uma política estacionária produz uma sequência de decisões $\pi = (x^{\pi}, x^{\pi}, \cdots, x^{\pi})$, em que a regra x^{π} será a mesma para qualquer que seja a época de decisão.

Dado que resolver PDM envolve encontrar uma política que otimize o retorno esperado acumulado, pode se dizer que uma política π será melhor que uma outra política π' se o seu retorno acumulado esperado obtido for igual ou melhor que o retorno acumulado esperado obtido por π' para todos os estados. Consequentemente, no caso de maximização, se o retorno acumulado esperado de uma política for maior ou igual ao retorno acumulado esperado de todas as outras políticas viáveis para todos os estados, pode se afirmar que esta política é uma política ótima.

2.2.2 Equação de Bellman e o princípio da otimalidade

Sejam $s \in \mathsf{S}$ um dado estado do sistema e uma política π, tem-se que o retorno esperado quando o sistema ocupa o estado s na época de decisão k e o decisor segue uma política fixa π é dado por:

$$v^{\pi}(s) = \mathrm{E}^{\pi}\left[G_k \mid S_k = s\right] = \mathrm{E}^{\pi}\left\{\sum_{\alpha=0}^{\infty} \gamma^{\alpha} C_{k+\alpha+1} \mid S_k = s\right\}, \tag{2.5}$$

sendo $v : \mathsf{S} \rightarrow \mathrm{R}$ uma função que mapeia cada estado viável e associa um valor real denominada por função de valor. De maneira semelhante, pode se definir uma função $q : \mathsf{S} \times \mathsf{A} \rightarrow \mathrm{R}$ dada por:

$$q^{\pi}(s, a) = \mathrm{E}^{\pi}\left[G_k \mid S_k = s, A_k = a\right] = \mathrm{E}^{\pi}\left\{\sum_{\alpha=0}^{\infty} \gamma^{\alpha} C_{k+\alpha+1} \mid S_k = s, A_k = a\right\}, \tag{2.6}$$

sendo $q^{\pi}(s,\ a)$ denominada de função de valor estado e decisão. Esta função computa o valor de tomar uma decisão a no estado s quando o sistema está na época de decisão k sob o exercício de uma política fixa π.

Note que a partir destas funções é possível avaliar a qualidade de uma uma política. Entretanto, calcular diretamente o valor esperado das equações 2.5 e 2.6 pode ser computacionalmente caro. Então, motivado por contornar esta limitação, estudos desenvolvidos em (Bellman, 1957) iniciaram um novo paradigma para lidar com esta barreira computacional. A estratégia é reescrever as funções de valor de modo a satisfazer uma relação de recursividade. Assim, sabendo que a função de valor parte da soma dos retornos obtidos, G_k pode ser rescrito como:

$$G_k = C_{k+1} + \gamma\, C_{k+2} + \gamma^2\, C_{k+3} + \gamma^3\, C_{k+4} + \cdots \tag{2.7}$$

$$G_k = C_{k+1} + \gamma \quad C_{k+2} + \gamma\, C_{k+3} + \gamma^2\, C_{k+4} + \cdots \tag{2.8}$$

$$G_k = C_{k+1} + \gamma\, G_{k+1} \tag{2.9}$$

$$G_k = c(s_k, a_k) + \gamma\, G_{k+1}. \tag{2.10}$$

Assim, para qualquer política π, a função de valor para um estado inicial s preserva uma condição de consistência entre o valor do estado inicial e do estado sucessor, tal como

apresentado a seguir:

$$v^{\pi}(s) = \mathrm{E}^{\pi}\,[G_k \mid S_k = s] \tag{2.11}$$

$$v^{\pi}(s) = \mathrm{E}^{\pi}\,[C_{k+1} + \gamma\, G_{k+1} \mid S_k = s] \tag{2.12}$$

$$v^{\pi}(s) = \mathrm{E}^{\pi}\,[c(s_k, x^{\pi}(s_k)) + \gamma\, G_{k+1} \mid S_k = s] \tag{2.13}$$

$$v^{\pi}(s) = c(s_k, x^{\pi}(s_k)) + \gamma\, \mathrm{E}^{\pi}\,[G_{k+1} \mid S_{k+1} = s^{\prime}] \tag{2.14}$$

$$v^{\pi}(s) = c(s_k, x^{\pi}(s_k)) + \gamma\, v^{\pi}(s^{\prime}). \tag{2.15}$$

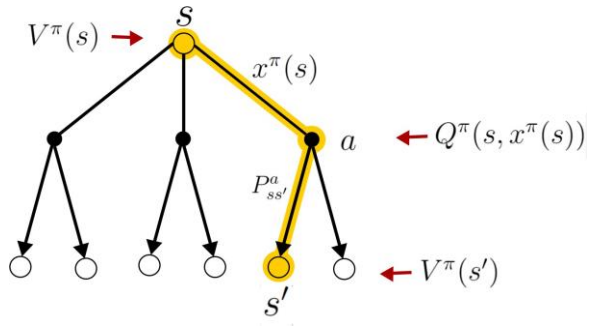
Esta relação de recursividade pode ser observada a partir do diagrama na Fig 8 (a). Tal como mostra a imagem, considere s o ponto de partida de uma trajetória. Obedecendo a dinâmica de um PDM, o decisor escolhe uma ação, de modo que esta escolha será determinada por uma regra de decisão x^{π}. Ao escolher a decisão a, o decisor obtém o retorno imediato, computado a partir da função $c(s,\ x^{\pi}(s))$. A partir deste ponto, o decisor pode observar o sistema em um estágio pré-transicional, imediatamente após a decisão e antes da evolução entre os estados, e consequentemente avaliar o valor de uma decisão a após aplicada em um estado s por meio da função $q^{\pi}(s,\ x^{\pi}(s))$. O estado seguinte é uma variável aleatória, logo há uma lei de probabilidade associada a escolha do estado subsequente. Esta lei é a função de transição $p(s^{\prime}|s,\ a)$. Quando o sistema ocupa um novo estado, o ciclo novamente se repete, pois estando em $s^{\prime}$, a função $v^{\pi}(s^{\prime})$ computará o valor de estar neste estando, o decisor escolherá uma nova decisão, obterá o retorno e transicionará para outro estado.

Em resumo, a função de valor $v^{\pi}(s)$ equivale ao retorno imediato recebido logo após uma decisão, somado ao valor de estar em qualquer próximo estado viável, ponderado pela probabilidade deste estado viável ser o novo estado após a transição, dado o estado anterior e a decisão aplicada, tal como apresentado na equação abaixo:

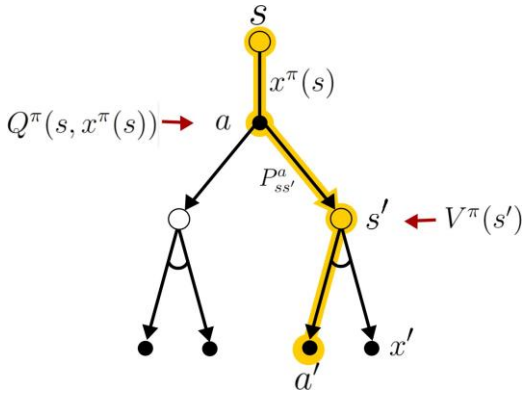
$$v^{\pi}(s) = c(s_k, x^{\pi}(s_k)) + \gamma \sum_{s^{\prime} \in S} p(s^{\prime}|s, a)\, v^{\pi}(s^{\prime}). \tag{2.16}$$

Em um PDM, quando um decisor pode observar o retorno da função de valor para todos os estados viáveis, ele se torna capaz de avaliar de forma direta e exata a qualidade das decisões que foram orientadas pela política vigente naquele momento. Portanto, a função de valor pode definir uma ordenação parcial entre políticas, classificando uma dada política como melhor do que outra. Então, sendo π uma política, pode-se dizer que ela é melhor ou tão boa quanto uma política $\pi^{\prime}$ se o seu retorno esperado for maior (considerando que seja um problema em que se deseja maximizar os ganhos) ou igual ao retorno esperado obtido por $\pi^{\prime}$. Em outras

Figura 8 – Diagramas do processo de decisão iniciando no estado s seguindo uma política π



(a) Diagrama da função de valor de estado.



(b) Diagrama da função de valor de estado e decisão.

Fonte: (Sutton; Barto, 2018), adaptado pela autora (2022)

palavras, $\pi \geq \pi'$ se e somente se $v^{\pi} \geq v^{\pi'}$ para todo $s \in \mathsf{S}$. Quando uma política é melhor ou igual a todas as outras políticas viáveis, então está será uma política ótima.

Para um mesmo problema é possível existir mais de uma política ótima. Apesar disto, um ponto em comum entre todas as políticas ótimas é o valor resultante da função de valor, que será o mesmo para todas as políticas ótimas. Assim, esta função de valor será a função de valor ótima, denotada por $v^{\wedge}$ e definida como

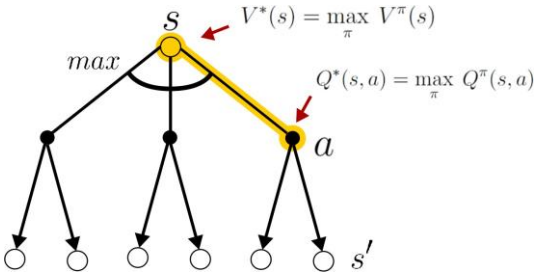
$$v^{\wedge}(s) = \max_{\pi} v^{\pi}(s) \quad \forall s \in \mathsf{S}. \tag{2.17}$$

Semelhantemente, a função ótima de valor de estado e decisão é dada por

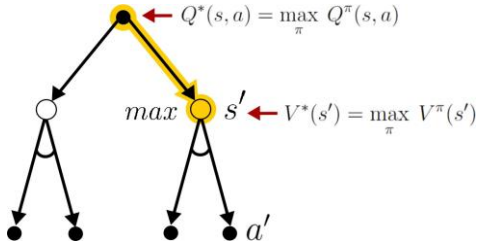
$$q^{\wedge}(s, a) = \max_{\pi} q^{\pi}(s, a) \quad \forall s \in \mathsf{S}, \forall a \in \mathsf{A}, \tag{2.18}$$

sendo $q^{\wedge}(s, a)$ o retorno esperado para a realização da decisão a no estado s seguindo uma política ótima.

Figura 9 – Diagramas representando (a) $v^{\wedge}(s)$ e (b) $q^{\wedge}(s, a)$



(a) Diagrama da otimalidade da função de valor de estado.



(b) Diagrama da otimalidade da função de valor de estado e decisão.

Fonte: (Sutton; Barto, 2018), adaptado pela autora (2022)

Uma característica muito importante das funções $v^{\wedge}$ e $q^{\wedge}$ é que existe uma relação entre elas. Observe a primeira imagem na Figura 9(a), e note que a função de valor de estado ótima para o estado s equivale a escolher a decisão viável a que maximiza a função de valor de estado e decisão. Assim, pode-se rescrever a $v^{\wedge}$ como:

$$v^{\wedge}(s) = q^{\wedge}(s, x^{\pi}(s)) \tag{2.19}$$

$$v^{\wedge}(s) = \max_{\pi} q^{\pi}(s, x^{\pi}(s)) \tag{2.20}$$

$$v^{\wedge}(s) = \max_{a \in A} q(s, a). \tag{2.21}$$

Por outro lado, para garantir que o valor de estar em um estado s após a decisão a seja ótimo, tem-se que o valor de estar nos estados seguintes também deve ser ótimo para qualquer estado viável. Assim, tal como se pode observar na Figura 9(b), o estado posterior à tomada de uma decisão seja aquele no qual a função de valor de estado é ótima. Baseado nisto, $q^{\wedge}(s, a)$ pode ser reescrito como:

$$q^{\wedge}(s, a) = c(s, a) + \gamma \, \mathrm{E}\left[v^{\wedge}(s^{\prime}) \mid s, a\right]. \tag{2.22}$$

Agora, veja que substituindo $q^{\wedge}(s, a)$ na equação (2.21) pela equação (2.22) a equação resultante será:

$$v^{\wedge}(s) = \max_{a \in A} [c(s, a) + \gamma \, E [v^{\wedge}(s^{J}) \mid s, a]] . \quad (2.23)$$

Esta equação é popularmente conhecida por equação de Bellman ou equação de otimalidade, ou simplesmente equação de valor ótima.

Então, a equação de Bellman garante um caminho para obter uma política ótima. Primeiramente, esta função é otimizada e em seguida uma política associada a ela é capturada. Assumindo que a política seja uma política gulosa com relação a escolha das decisões, está política será dada por:

$$x^{\pi}(s) \in \operatorname{argmax}_{a \in A} \{c(s, a) + \gamma \, E [v^{\wedge}(s^{J}) \mid s, a]\}, \quad (2.24)$$

sendo que argmax significa que se deseja escolher uma decisão a que maximiza a expressão que está entre chaves.

Este caminho de formulação da equação de otimalidade garante um princípio que é fundamental e seminal em PDM. Nomeado por princípio da otimalidade, este princípio diz que (Bellman, 1957):

PRINCÍPIO DA OTIMALIDADE: *Uma política ótima tem a propriedade de que qualquer que seja o estado inicial e a decisão inicial, as decisões restantes devem constituir uma política ótima em relação ao estado resultante da primeira decisão.*

Portanto, a equação de Bellman amparada pelo princípio da otimalidade fornece a base necessária para a construção dos algoritmos que são utilizados para resolver um PDM. Estes algoritmos serão apresentados na seção a seguir.

2.2.3 Algoritmos de programação dinâmica

Fundamentalmente, um dos paradigmas que viabiliza resolver um problema de programação dinâmica é a possibilidade de reduzi-lo em subproblemas menos complexos, resolver estes subproblemas e armazenar sua solução e então, a partir destas soluções construir a solução do problema original. Baseado nisto é que se fundamenta os algoritmos clássicos de programação dinâmica. Nesta seção serão apresentados três algoritmos, que serão apresentados de acordo com a configuração do horizonte de decisão do problema.

- **Horizonte finito**
  - *Algoritmo de programação dinâmica reverso:* Para este algoritmo é necessário assumir que a função de valor é conhecida para a última época de decisão. O processo iterativo inicia na última época de decisão, seguindo uma trajetória reversa no horizonte de decisão, computando a função de valor para cada estado viável.
- **Horizonte infinito**
  - *Iteração de política:* Para uma dada política inicial, este algoritmo realiza um ciclo iterativo alternando entre avaliação e melhoria de política.
  - *Iteração de valor:* Faz uma varredura no espaço das funções de valor a fim de encontrar uma função ótima, para então utilizá-la para computar uma política ótima.

2.2.3.1 Algoritmos para problemas de horizonte finito

Um modelo de programação dinâmica pode ser formulando como sendo de horizonte finito quando há um horizonte de decisão bem específico e limitado. Ocasionalmente, alguns problemas de horizonte infinito podem ser reduzidos em sua modelagem a problemas de horizonte finito, principalmente quando o decisor objetiva decidir sobre uma janela temporal específica (Powell, 2011).

No contexto de horizonte finito, o algoritmo de programação dinâmica reverso é um técnica que pode ser utilizada para gerar políticas ótimas. Este algoritmo assume que a função de valor $v_T(s_T)$ é data, ou seja, é conhecido o valor de se estar em qualquer estado viável para a última época de decisão. Além disto, é necessário que o processo de decisão esteja bem definido e seja conhecido. Por convenção, se define $v_T(s_T) = 0$ para todo $s_T \in S$.

Algoritmo 1: Algoritmo de programação dinâmica reverso

Entrada: γ, PDM

início
  Inicialize $v_T(s_T) = 0$ para todo $s \in S$
  $t = T - 1$
  Enquanto $t > 0$ **faça:**
    Para todo $s_t \in S$, **calcule:**
    $$v_t(s_t) = \max_{a_t \in A_t} \{ c_t(s_t, a_t) + \gamma \sum_{s' \in S} p(s'|s_t, a_t) v_{t+1}(s') \}$$
    $t = t - 1$
  Fim
fim

O algoritmo de programação dinâmica reverso faz varreduras sobre todo o espaço de estados em cada época de decisão e computa explicitamente a função de valor, seguindo uma trajetória reversa ao longo de todo o horizonte de decisão. O que torna o algoritmo limitado computacionalmente é a necessidade de avaliar a função de valor para todo o espaço de estados. Por isso, a dimensão do conjunto S deve ser um ponto muito importante a ser considerado quando se deseja utilizar este algoritmo na resolução de um problema de horizonte finito.

2.2.3.2 Algoritmos para problemas de horizonte infinito

Problemas de decisão sequencial de horizonte infinito apresentam um contexto em que a política é predominantemente estacionária. Em casos como este, a regra de decisão não muda em instantes diferentes. É importante destacar que mesmo que em muitos problemas práticos possam ser reduzidos ao modelo de horizonte finito, a suposição de um número infinito de estágios de decisão é muito importante em termos de análise de comportamento assintótico (Androulakis, 2009).

Seja $s \in S$ um estado inicial de um problema de horizonte infinito, deseja-se obter uma política π que otimize a função de valor em que

$$v(s) = \lim_{t\to\infty} V_t(s_t) \tag{2.25}$$

$$v(s) = \lim_{t\to\infty} \max_{a_t \in A} \left(c(s_t, a_t) + \gamma \sum_{s' \in S} p(s'|s, a)\, v^{\pi}(s') \right) . \tag{2.26}$$

Note que o problema de horizonte infinito estacionário pode ser compreendido como um problema sem a dimensão temporal. Então, assumindo que o limite existe para a equação (2.25), a equação de otimalidade para o problema estacionário será

$$v(s) = \max_{a \in A} \left(c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)\, v(s') \right) . \tag{2.27}$$

2.2.3.3 Algoritmo de iteração de política

O algoritmo de iteração de política (*policy iteration*) produz uma sequência de política estacionárias associadas à função de valor através de duas etapas: avaliação da política (*policy evaluation*) e melhoria da política (*policy improvement*) (Szepesvári, 2010). Dada uma política estacionária inicial π_0, esta política é avaliada a partir da função de valor e em seguida, esta mesma função é utilizada para construir uma política melhorada. Este ciclo iterativo é

apresentado no fluxo abaixo:

$$\pi_0 \xrightarrow{\text{avalia}} v^{\pi_0} \xrightarrow{\text{melhora}} \pi_1 \xrightarrow{\text{avalia}} v^{\pi_1} \xrightarrow{\text{melhora}} \cdots v^{\pi_*} \xrightarrow{\text{melhora}} \pi_*.$$

Para cada uma destas políticas é garantido que a nova política obtida não será pior que a política anterior. Esta propriedade é assegurada pelo fato de que, sendo o espaço de políticas finito, o PDM converge para uma política ótima em um número finito de operações (Sutton; Barto, 2018). No geral, ao utilizar o algoritmo de iteração de política o problema de um PDM se resume a resolver um conjunto de equações lineares e comparar soluções subsequentes, onde cada política obtida a partir do processo iterativo é objetivamente tão boa ou melhor que a anterior (Howard, 1960).

2.2.3.3.1 Avaliação da política

Considere um estado $s \in S$ e uma política π, e seja $v^0, v^1, v^2, \cdots$ uma sequência de funções de valor que aproxima o valor de estar no estado s seguindo a política π. Esta sequência parte de uma aproximação inicial que é escolhida arbitrariamente, e cada sucessiva aproximação é obtida a partir da equação de otimalidade de Bellman com a seguinte regra de atualização:

$$v^{n+1}(s) = c(s, x^{\pi}(s)) + \gamma \sum_{s' \in S} p(s'|s, x^{\pi}(s))\, v^n(s'). \qquad (2.28)$$

Note que $v^n = v^{\pi}$ é um ponto fixo nesta regra de atualização. A sequência $v^0, v^1, v^2, \ldots$, de modo geral, convergirá para v^{π} se a sequência de retornos for limitada e $0 \leq \gamma < 1$.

Para produzir sucessivas aproximações, o algoritmo aplica a seguinte operação para todos os estados: computa o valor de estar em todos os estados subsequentes viáveis a partir última função de valor armazenada e soma ao retorno imediato obtido. Estas aproximações são sucessivamente atualizadas, tal como apresentado abaixo, até convergir para a função de valor exata.

$$v^0 \rightarrow v^1 \rightarrow v^2 \rightarrow v^3 \rightarrow \cdots \sim v^{\pi}$$

Em resumo, a etapa de avaliação de política tem por objetivo mensurar a qualidade de uma política dada, onde cada função de valor formulada nesta avaliação produz uma sequência de aproximações que convergem para a função de valor da política ótima.

2.2.3.3.2 Melhoria da política

Dada uma política π e sua função de valor $v^{\pi}(s)$ para um estado $s \in \mathsf{S}$, deseja-se saber se existe uma outra política π', tal que sua função de valor $v^{\pi'}(s)$ seja objetivamente melhor que $v^{\pi}(s)$. Isto equivale dizer que interessa ao decisor saber se aplicar uma decisão $a' = x^{\pi}(s)$ é melhor do que seguir a decisão $a = x^{\pi}(s)$ orientada pela política atual. Então, é intuitivo que a escolha entre as decisões a' e a seja feita a partir da comparação entre elas. Sabendo que o valor de quaisquer duas decisões a' e a sobre um estado s é dado respectivamente por $q^{\pi}(s, a')$ e $q^{\pi}(s, a)$, então se

$$q^{\pi}(s, a') \geq q^{\pi}(s, a)$$

tal que $x^{\pi'}(s) = a' \neq a = x^{\pi}(s)$, será mais favorável para o decisor seguir a decisão gerada pela função de decisão $x^{\pi'}(s)$. Esta observação resulta do teorema de melhoria da política (Howard, 1960).

TEOREMA DE MELHORIA DA POLÍTICA: *Sejam π e π' duas políticas quaisquer tais que, para todo $s \in \mathsf{S}$*

$$q^{\pi}(s, x^{\pi'}(s)) \geq v^{\pi}(s), \tag{2.29}$$

então, a política π' deve ser tão boa, ou melhor que π. Isto é, deve gerar um retorno esperado melhor ou igual para todos os estados $s \in \mathsf{S}$, ou seja

$$v^{\pi'}(s) \geq v^{\pi}(s). \tag{2.30}$$

Note que quando este processo comparativo é aplicado sobre todas as decisões viáveis em todo o espaço de estados, a decisão escolhida será aquela cuja função de valor possui a melhor performance para cada estado. Então, a política melhorada no algoritmo de iteração de valor será a política que otimiza o valor de aplicar uma decisão sobre um dado estado, para qualquer que seja a decisão e para qualquer que seja o estado. Esta política melhorada é a política gulosa (*greedy policy*) com relação a todos os estados, e é dada por:

$$x^{\pi'} \in \operatorname{argmax} q^{\pi}(s, a). \tag{2.31}$$

A política gulosa atende às condições do teorema de melhoria da política (Equação 2.29), portando, ela será tão boa quanto, ou melhor que a política atual (Sutton; Barto, 2018).

Algoritmo 2: Algoritmo de iteração de política

Entrada: θ, PDM

início
  $x^{\pi}(s) \leftarrow$ inicialize para todo $s \in S$
  $v^{\pi}(s) \leftarrow$ inicialize para todo $s \in S$
  mudar $\leftarrow$ True
  Enquanto *mudar = True* **faça:**
    $\Delta \leftarrow 0$
    Enquanto $\Delta < \theta$ **faça:**
      Para $s \in S$ **faça:**
        $v \leftarrow v^{\pi}(s)$
        $v^{\pi}(s) \leftarrow c(s, x^{\pi}(s)) + \gamma \sum_{s' \in S} p(s'|s, x^{\pi}(s)) v^{\pi}(s')$
        $\Delta \leftarrow \max(\Delta, |v - v^{\pi}(s)|)$
      Fim
    Fim
    Para $s \in S$ **faça:**
      $x^{\pi}(s) \leftarrow \underset{a}{\operatorname{argmax}}\, c(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v(s')$
    Fim
    Se $x^{\pi^+} = x^{\pi}$ **então**
      mudar $\leftarrow$ Falso
    Fim
    $x^{\pi}(s) \leftarrow x^{\pi^+}$
  Fim
  return x^{π}
fim

Assim, este teorema garante que, havendo uma política melhor que a atual é impossível que não seja encontrada em algum momento da rotina de melhoria da política (Howard, 1960).

A composição das rotinas de avaliação da política e melhoria da política formam o algoritmo de iteração de política (Algoritmo 2).

2.2.3.4 Algoritmo de iteração de valor

O algoritmo de iteração de valor opera atualizando a função de valor iterativamente para todos os estados até que um critério de convergência seja atingido. A cada iteração do algoritmo, a equação de Bellman é atualizada de acordo com a regra abaixo:

$$v^{k+1} = \max_{a \in A} c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)\, v^{k}(s'). \tag{2.32}$$

.

Cada uma destas etapas de atualização é denominadas de *backup* de Bellman. Quando a sequência de aproximações de valor converge para a função de valor ótima, o algoritmo captura

a política associada, e,portanto, resolver o PDM.

Uma vantagem da iteração de valor frente ao algoritmo apresentado anteriormente está em que no algoritmo de iteração de política cada iteração envolve o processo de avaliação da política separado do processo de melhoria da política, de modo que em cada teste para atualização da política, são feitas duas varreduras sobre o espaço de estados. Em contrapartida, o algoritmo de iteração de valor combina em uma única varredura do espaço de estados as etapas de avaliação e melhoria da política (Sutton; Barto, 2018). Além disto, aa iteração de valor aprimora a política com mais frequência e reduz a complexidade do laço de repetição principal. Em termos de memória, não há uma diferencia considerável entre eles (Dai; Goldsmith, 2007),(Geramifard *et al.*, 2013).

Esta técnica se assemelha ao algoritmo para problema de horizonte finito apresentado anteriormente (Algoritmo 1). No caso do algoritmo de programação dinâmica reverso, a iteração parte de um uma época de decisão T , e segue decrementando o contador do ciclo iterativo, construindo a função de valor para cada estado. Semelhantemente, porém em um sentido oposto, a iteração de valor parte de um ponto inicial, e avança incrementando o contador do ciclo iterativo até satisfazer um critério de convergência (Powell, 2022). O algoritmo para quando

$$\| v^{n+1} - v^{n} \| < \epsilon\,(1-\gamma)/2\,\gamma, \tag{2.33}$$

em que γ, $0 \leq \gamma < 1$, é o fator de desconto, $\epsilon > 0$ é uma tolerância de erro, e $\|v^{n+1} - v^{n}\| = \max_s |v^{n+1}(s) - v^{n}(s)|$.

Veja 2 o pseudocódigo algoritmo de iteração de valor. Na próxima subseção é apresentada a garantia de convergência do algoritmo.

2.2.3.4.1 Convergência do algoritmo de iteração de valor

Esta seção é dedicada a apresentar a prova da convergência do algoritmo de iteração de valor. As demonstrações aqui apresentadas também podem ser consultadas em (Bellman, 1957). Antes de desenvolver a demonstração, é fundamental apresentar algumas suposições e definições necessárias.

Suposição 1. ***Funções de retorno e transição estacionárias****: As funções $c(s, a)$ e $p(s'|s, a)$ não variam ao longo das épocas de decisão.*

Suposição 2. ***Retornos limitados****: $|\, c(s, a)\,| \leq M < \infty$ para todo $a \in \mathsf{A}$, $s \in \mathsf{S}$.*

Algoritmo 3: Algoritmo de iteração de valor

Entrada: ϵ, PDM

início
  $v(s) \leftarrow$ inicialize para todo $s \in S$
  Enquanto $\Delta > \epsilon$ **faça:**
    Para $s \in S$ **faça:**
      $v \leftarrow v(s)$
      $v(s) \leftarrow \max_a c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)\, v(s')$
      $x^{\pi}(s) \leftarrow \operatorname{argmax}_{a \in A} c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)\, v(s')$
      $\Delta \leftarrow \max(\Delta, \|v - v(s)\|)$
    Fim
  Fim
fim

Suposição 3. ***Fator de desconto****: Retornos futuros são descontados de acordo com o fator de desconto* γ*, em que* $0 \leq \gamma < 1$.

Suposição 4. ***Espaço de estados de discretos****: O conjunto* S *é finito ou enumerável.*

Definição 2.1. *Seja* V *um conjunto de funções, limitado com valores reais, define-se a norma de* $V \in V$ *por:*

$$\|V\| = \max_{s \in S} v(s).$$

Dado que V *é fechado com relação a adição e multiplicação por escalar e possui uma norma, pode-se dizer que* V *é um espaço linear normado.*

Definição 2.2. *Dizemos que um operador* $T : V \rightarrow V$ *é uma mapeamento da contração se exite um* γ, $0 \leq \gamma < 1$, *tal que:*

$$\|T(V) - T(U)\| \leq \gamma \|V - U\|.$$

Definição 2.3. *Uma sequência* $\{v^n\} \in V$, $n \geq 1$, *é dita ser uma sequência de Cauchy se para todo* $\epsilon > 0$ *existe* N *tal que para todo* $m, n \leq N$*:*

$$\|v^n - v^m\| < \epsilon.$$

Ou seja, uma sequência de Cauchy é aquela cujo os elementos se tornam arbitrariamente próximos uns dos outros à medida em que a sequência avança.

Definição 2.4. *Um espaço linear normado é completo se cada sequência de Cauchy contém um ponto limite neste espaço. Este espaço é chamado de espaço de Banach.*

A seguir é apresentado o teorema que fundamenta o argumento imediato para provar a convergência do algoritmo de iteração de valor.

Teorema 2.1. *(Teorema do ponto fixo de Banach) Seja* $\mathbb{V}$ *um espaço de Banach, e seja* $T : \mathbb{V} \to \mathbb{V}$ *um mapeamento de contração. Então:*

(a) Existe um único elemento $v^\wedge \in \mathbb{V}$ *tal que* $T(v^\wedge) = v^\wedge$.

(b) Para um $v^0 \in \mathbb{V}$ *arbitrário, a sequência* $\{v^n\}$ *definida por*

$$v^{n+1} = T(v^n) = T^{n+1}(v^0)$$

converge para $v^\wedge$.

Demostração: Seja a sequência $\{v^n\}$, $n \geq 1$. Inicialmente, será mostrado que a distância entre v^n e v^{n+m} tende a zero para um n suficientemente grande e um certo $m \geq 1$. Então, a diferença $v^{n+m} - v^n$ é escrita como sendo

$$\begin{aligned} v^{n+m} - v^n &= v^{n+m} - v^{n+m+1} + v^{n+m+1} - \cdots - v^{n+1} + v^{n+1} - v^n \\ &= \sum_{k=0}^{m-1} (v^{n+k+1} - v^{n+k}). \end{aligned}$$

Aplicando a norma sobre os dois lados da igualdade, e em seguida utilizando-se da definição de desigualdade triangular, tem-se:

$$\begin{aligned} \|v^{n+m} - v^n\| &= \left\| \sum_{k=0}^{m-1} (v^{n+k+1} - v^{n+k}) \right\| \\ &\leq \sum_{k=0}^{m-1} \|(v^{n+k+1} - v^{n+k})\| \\ &= \sum_{k=0}^{m-1} \|(T^{n+k}(v^1) - T^{n+k}(v^0))\| \\ &\leq \sum_{k=0}^{m-1} \gamma^{n+k} \|v^1 - v^0\| \\ &= \frac{\gamma^n (1 - \gamma^m)}{1 - \gamma} \|v^1 - v^0\|. \end{aligned}$$

Sabendo que γ, $0 \leq \gamma < 1$, para n suficientemente grande, a igualdade acima pode se tornar arbitrariamente pequena. Note que a partir disto é certo afirmar que a sequência $\{v^n\}$ é uma sequência de Cauchy. Dado que $\mathbb{V}$ é um espaço completo, é necessário que v^n tenha um ponto limite dentro deste espaço. A partir disso, conclui-se que:

$$\lim_{n \to \infty} v^n = v^\wedge.$$

Agora, deseja-se mostrar que $v^{\wedge}$ é um ponto fixo do mapeamento T. Para isto, observe que

$$\begin{aligned} 0 \leq & \|T(v^{\wedge}) - v^{\wedge}\| \\ = & \|T(v^{\wedge}) - v^{n} + v^{n} - v^{\wedge}\| \\ \leq & \|T(v^{\wedge}) - v^{n}\| + \|v^{n} - v^{\wedge}\| \\ = & \|T(v^{\wedge}) - T(v^{n-1})\| + \|v^{n} - v^{\wedge}\| \\ \leq & \gamma \|v^{\wedge} - v^{n-1}\| + \|v^{n} - v^{\wedge}\|. \end{aligned}$$

Sabendo que $\lim_{n\to\infty} \|v^{n} - v^{\wedge}\| = \lim_{n\to\infty} \|v^{\wedge} - v^{n-1}\| = 0$, tem-se que

$$0 \leq \|T(v^{\wedge}) - v^{\wedge}\| \leq 0,$$

portanto,

$$\|T(v^{\wedge}) - v^{\wedge}\| = 0,$$

que significa que $T(v^{\wedge}) = v^{\wedge}$. Agora, a unicidade de $v^{\wedge}$ será provada por contradição. Assuma que existem dois pontos de limite, representados por $v^{\wedge}$ e U^{*}. A suposição de que T é um mapeamento de contração requer que

$$\|T(v^{\wedge}) - T(U^{*})\| \leq \gamma \|v^{\wedge} - U^{*}\|.$$

Note, que se $v^{\wedge}$ e U^{*} são pontos de limite, então $T(v^{\wedge}) = v^{\wedge}$ e $T(U^{*}) = U^{*}$, isto significa que

$$\|v^{\wedge} - U^{*}\| \leq \gamma \|v^{\wedge} - U^{*}\|.$$

Porém, sabendo que $\gamma < 1$, isto é uma contradição, o que significa que $v^{\wedge} = U^{*}$, e portanto, $v^{\wedge}$ é único. □

Agora, de posse do Teorema 2.1 é possível mostrar que o algoritmo de iteração de valor converge para a solução ótima. Para isto, basta mostrar que o operador M, sendo $\mathsf{M}(V) = \max V$, é um mapeamento de contração. A proposição abaixo oferece essa garantia.

Proposição 2.1. *Se* $0 \leq \gamma < 1$, *então* M *é um mapeamento de contração de* V.

Demonstração. Sejam $V, U \in \mathsf{V}$, assuma que $\mathsf{M}(V) \geq \mathsf{M}(U)$. Considere certo estado $s \in (S)$, admita que existe

$$a_{s}^{*}(V) \in \underset{a\in A}{\operatorname{argmax}} \left(c(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)\, v(s') \right).$$

Então,

$$
\begin{aligned}
0 \leq\ & \mathsf{M}(v(s)) - \mathsf{M}(U(s)) \\
=\ & c(s, a_s^*(V)) + \gamma \sum_{s' \in S} P_{ss'}^{a_s^*(V)} v(s') \Big) - \Big(c(s, a_s^*(U)) + \gamma \sum_{s' \in S} P_{ss'}^{a_s^*(U)} U(s') \Big) \\
\leq\ & c(s, a_s^*(V)) + \gamma \sum_{s' \in S} P_{ss'}^{a_s^*(V)} v(s') \Big) - \Big(c(s, a_s^*(V)) + \gamma \sum_{s' \in S} P_{ss'}^{a_s^*(V)} U(s') \Big) \\
=\ & \gamma \sum_{s' \in S} P^{a_s^*(V)}_{ss'} [v(s') - U(s')] \\
\leq\ & \gamma \sum_{s' \in S} P^{a_s^*(V)}_{ss'} \| V - U \| \\
=\ & \gamma \| V - U \| \sum_{s' \in S} P^{a_s^*(V)}_{ss'} \\
=\ & \gamma \| V - U \| .
\end{aligned}
$$

Este resultado afirma que $\mathsf{M}(V) \geq \mathsf{M}(U)$, então $\mathsf{M}(v(s)) - \mathsf{M}(U(s)) \leq \gamma |v(s) - U(s)|$. Por outro lado, se a suposição inicial fosse que $\mathsf{M}(V) \leq \mathsf{M}(U)$, semelhantemente se chegaria a conclusão que $\mathsf{M}(v(s)) - \mathsf{M}(U(s)) \geq -\gamma |v(s) - U(s)|$. Isto significa que,

$$|\mathsf{M}(v(s)) - \mathsf{M}(U(s))| \leq \|v(s) - U(s)\|,$$

é valido para todos $s \in S$. A partir da definição da norma (def. 2.1), tem-se que

$$
\begin{aligned}
\max_{s \in S} |\mathsf{M}(v(s)) - \mathsf{M}(U(s))| &= \|\mathsf{M}(v(s)) - \mathsf{M}(U(s))\| \\
&\leq \gamma \|V - U\| .
\end{aligned}
$$

Isto significa que M é um mapeamento de contração, e, portanto que a sequência v^n gerada por $v^{n+1} = \mathsf{M}(v^n)$ converge para um único ponto de limite v^Λ que satisfaz a equação de otimalidade. □

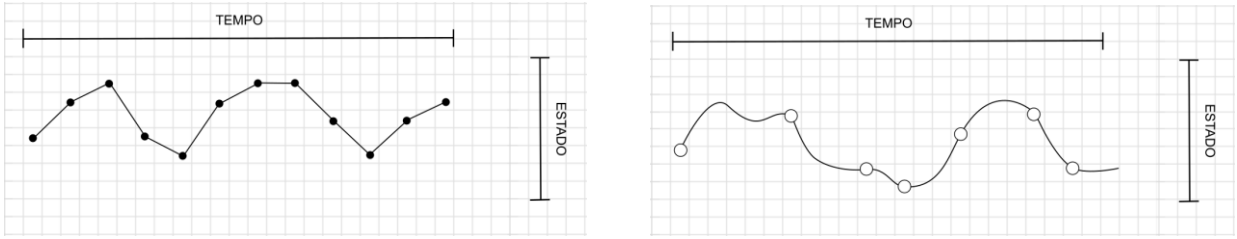
2.3 PROCESSO DE DECISÃO SEMIMARKOVIANO

Dado o problema de tomar decisões diante de um cenário de incerteza, há uma vasta classe destes problemas que podem ser modelados como um processo de decisão markoviano (PDM). Entretanto, existem alguns problemas que carregam em si determinadas particularidades que exigem um certo nível de generalização do processo estocástico, objetivando melhor representá-lo. Tome como um exemplo um processo de filas, cujo decisão envolve determinar

qual cliente da fila será atendido em um dado momento por um servidor. Perceba que os intervalos entre as decisões nem sempre serão iguais, e eventualmente ocorrem alterações de estado independente de uma decisão diretamente aplicada sobre ele. Diante desta característica do problema, um caminho viável é tratar esse processo de decisão como um processo de decisão semimarkoviano.

Processos de decisão semimarkovianos (PDSM) são generalizações de um processo de decisão markoviano (PDM), em que os tempos entre as épocas de decisão da cadeia de Markov não são iguais, podendo ocorrer deterministicamente ou estocasticamente, embora o mais comum seja que este intervalo seja uma variável aleatória (Gallager, 2013), (Baykal-Gursoy; Gursoy, 2007), (Gosavi, 2015). De modo geral, pode-se dizer que um PDM é um caso especial de um PDSM em que o tempo entre as transições é sempre unitário. PDSM são apropriados para modelar sistemas de eventos discretos de tempo contínuo. Neste sentindo, em um PDSM as decisões aplicadas em intervalos variáveis destinam-se a modelar cursos de ações estendidos temporalmente (Sutton *et al.*, 1999).

Figura 10 – Trajetórias da evolução dos estados em um processo de decisão markoviano e semimarkoviano.



(a) Trajetória da evolução dos estados em um PDM. (b) Trajetória da evolução dos estados em um PDSM.

Fonte: (Sutton *et al.*, 1999), adaptado pela autora (2022)

Quando o tempo entre as épocas de decisão é uma variável aleatória, o PDSM pode ser classificado em duas categorias:

- **Processo de decisão semimarkoviano generalizado:** Quando a dstribuição de probabilidade associada a variável aleatória é uma distribuição qualquer;
- **Processo de decisão markoviano de tempo contínuo:** As variáveis aleatórias que definem o intervalo entre as épocas de decisão são do tipo exponencial.

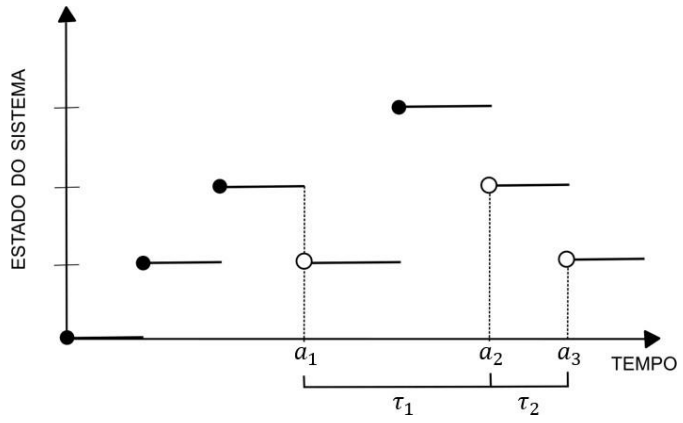
Para qualquer que seja a classificação acima, pode-se pensar no PDSM como um processo onde os sucessivos estados ocupados são governados por uma probabilidade de transição de um processo markoviano, mas o tempo de permanência em qualquer estado é descrito por uma

variável aleatória que depende do estado ocupado atualmente e do estado para o qual a próxima transição será realizada. Então, nos instantes de transição o PDSM se comporta como um PDM. Entretanto, o tempo de ocorrência entre as transições são governados por um mecanismo probabilístico diferente (Howard, 2007).

2.3.1 Processo natural e processo semimarkoviano

Voltando ao exemplo do processo da fila apresentado no início desta seção, considere que o sistema pode ser observado continuamente ao longo de sua evolução, e o observador tem acesso ao tamanho da fila. Assumindo que uma vez que o cliente entra na fila e só sai após ser atendido, tem-se que o estado da fila varia sob duas condições: quando chega um novo cliente ou quando o servidor atende um cliente que estava em espera. Perceba que a chegada de novos clientes na fila independe de qualquer decisão, enquanto a redução da fila depende da decisão de escolher um cliente para ser atendido. Entre o atendimento de dois clientes consecutivos, o tamanho da fila pode variar, mas veja que para a escolha do próximo cliente a ser atendido, importa ao decisor saber o estado da fila somente no momento em que a decisão será tomada.

Figura 11 – Exemplo do fluxo de um processo natural, em que os círculos abertos representam os estados de decisão do PDSM.



Fonte: (Puterman, 2005), adaptado pela autora (2022)

Este exemplo do processo de filas, traz a evidência dois conceitos importantes em um PDSM. Note que, embora o sistema evolua continuamente, apenas os estados nos quais existe a aplicação direta de uma decisão oferecem informações relevantes ao decisor. Portanto, é conveniente distinguir o processo estocástico que produz a evolução contínua do sistema do processo que representa a evolução dos estados somente em épocas de decisão. A cada um

destes processos, dá-se respectivamente o nome de processo natural e processo semimarkoviano (Puterman, 2005).

Seja S o espaço de estados finito ou enumerável, defina $\{S(t)\}_{t\in[0,T]}$ como sendo um processo estocástico, tal que $S(t)$ é uma variável aleatória que assume valores em S, e $0 < T \leq \infty$ é o horizonte temporal. Considerando que as épocas de decisão ocorrem em pontos discretos e aleatórios do tempo, estabeleça t_k, $k \in \{0, 1, 2, \cdots\}$, como a k-ésima época de decisão. Na época de decisão t_k, o decisor observa o estado $s_k = S(t_k)$ e aplica a decisão $a_k \in \mathsf{A}_{s_k}$, em que $\mathsf{A}_{s_k} \subseteq \mathsf{A}$ denota o conjunto de decisões viáveis no estado s_k e A é uma conjunto finito ou enumerável que contém todas as decisões viáveis.

As transições entre estados de decisão e o intervalo temporal entre duas épocas de decisão são governados por uma função de probabilidade estacionária dada por:

$$Q(j, \tau | i, a) = \mathrm{P}\left[s_{k+1} = j, t_{k+1} - t_k \leq \tau | s_k = i, a_k = a\right], \tag{2.34}$$

onde $Q(j, \tau | i, a)$ descreve a probabilidade do sistema ocupar o estado j no tempo t_{k+1} num intervalo de tempo de no máximo τ dado que no estado i foi aplicada a decisão a. Adicionalmente, tem-se que a probabilidade do sistema ocupar um estado j na próxima época de decisão é :

$$P(j|i, a) = \mathrm{P}\left[s_{k+1} = j | s_k = i, a_k = a\right], \tag{2.35}$$

a qual pode ser obtida também por:

$$P(j|i, a) = \lim_{\tau\to\infty} Q(j, \tau | i, a). \tag{2.36}$$

Agora, seja

$$p(\tau | i, a, j) = \mathrm{P}[t_{k+1} - t_k \leq \tau | s_{k+1} = j, a_k = a, s_k = i], \tag{2.37}$$

a probabilidade de que a próxima época de decisão ocorra num intervalo de no máximo τ , dado que o estado atual seja i, o próximo estado seja j e a decisão aplicada seja a. Note que $Q(j, \tau | i, a)$ pode ser obtida a partir de $P(j|i, a)$ e $p(\tau | i, a, j)$ (Bertsekas, 2007), tal como apresentado a abaixo:

$$Q(j, \tau | i, a) = p(\tau | i, a, j) P(j | i, a). \tag{2.38}$$

Esta dinâmica de transição entre as épocas de decisão é denominada de cadeia de Markov embutida ou processo de decisão markoviano embutido (Puterman, 2005), (Gosavi, 2015).

2.3.2 Equação de Bellman em PDSM

Dado que em um estado $i \in S$ seja escolhido aplicar a decisão $a \in A_s$, tem-se que o retorno fixo global $C(i, a)$ e um taxa de retorno adicional $c(i, a, j)$ que ocorre enquanto o processo natural permanece em um estado de decisão j. Então, o retorno total entre épocas de decisão subsequentes t_k e t_{k+1} é dado por:

$$r(i, a) = C(i, a) + \mathrm{E}\int_{t_k}^{t_{k+1}} c(i, a, S(t))dt \tag{2.39}$$

em que t_k e t_{k+1} são o tempo de duas épocas de decisão consecutivas e o estado $S(t)$ pode mudar diversas vezes entre estas duas épocas de decisão.

Seja π uma política estacionária e determinística definida como uma função $\pi : S \rightarrow A$. A performance de π pode ser avaliada a partir de diferentes funções objetivo. Neste trabalho será utilizado o retorno médio da política, que dado um estado inicial $s_0 = i$, é definida como:

$$g_\pi(i) = \lim_{n\to\infty} \mathrm{E}\left[\frac{1}{\sum_{k=1}^{n} \tau_k | i, \pi}\, \mathrm{E}\left\{\sum_{k=0}^{n-1} C(s_k, \pi(s_k)) + \int_{t_k}^{t_{k+1}} c(s_k, \pi(s_k), S(t))dt \;\middle|\; i, \pi\right\}\right], \forall i \in S, \tag{2.40}$$

em que τ_k denota o tempo de permanência entre as épocas de decisão $k-1$ e k. Resolver um PDSM corresponde a encontrar uma política ótima π^* definida como:

$$\pi^\wedge(i) \in \arg\min_{\pi\in\Pi} g_\pi(i) \quad \forall i \in S, \tag{2.41}$$

em que Π é uma classe de políticas. Adicionalmente, assuma que a cadeia de Markov é do tipo unicadeia (uma cadeia com apenas uma classe recorrente e possivelmente alguns estados transientes), de modo que o custo médio esperado ótimo é dado por:

$$g^\wedge = g^{\pi^s} = \min_{\pi\in\Pi} g_\pi(i) \quad \forall i \in S, \tag{2.42}$$

é independente do estado inicial i (Gosavi, 2015).

Teoricamente, uma política ótima pode ser obtida resolvendo a equação de Bellman, que no caso de um PDSM é formulada sobre a cadeia de Markov embutida com probabilidade de transição apresentada em (2.36). Assim, a equação de Bellman de um PDSM é dada pela função de valor na seguinte forma (Bertsekas, 2007; Bellman, 1952)

$$h^\wedge(i) = \min_{a\in A_i} \left(r(i, a) - g^\wedge \bar{\tau}(i, a) + \sum_{j\in S} P(j|i, a) h^\wedge(j) \right) \quad \forall i \in S, \tag{2.43}$$

sendo $r(i, a)$ o retorno esperado por transição, $\bar{\tau}(i, a)$ o tempo médio de permanência entre duas épocas de decisão e $g^\wedge$ é o custo médio mínimo.

Em particular, neste trabalho será mostrado que modelar o problema tratado como um PDSM, ao invés de um PDM, oferece uma vantagem computacional. Além disto, em princípio pode-se utilizar métodos baseados na equação de Bellman para encontrar a política, tal como os algoritmos de iteração de valor e iteração de política, apresentado respectivamente nas secções (2.2.3.4) e (2.2.3.3). Entretanto, em problemas com espaço de estados suficientemente grandes, estes métodos são computacionalmente inviáveis, um vez que necessitam varrer todo o espaço de estados. Na prática, um caminho alternativo é recorrer a métodos de aproximação. Neste trabalho é proposto uma abordagem baseada em uma política heurística *rollout* para obter uma solução para o problema de alocação de veículos.

2.4 ALGORITMO ROLLOUT

Em seções anteriores foram apresentando estruturas para modelar problemas de decisão sequencial, cujos os algoritmos apresentados (seção 2.2.3.4 e 2.2.3.3) constituem técnicas para computar exatamente a função de valor para cada estado viável. Entretanto, algoritmos como iteração de valor e iteração de política exigem um processo iterativo que percorre múltiplas vezes o espaço de estados e o espaço de decisões, tornando o desempenho computacional extremamente sensível à dimensionalidade destes espaços. Adicionalmente, a abordagem clássica e exata do cálculo da função de valor necessita que o processo estocástico seja plenamente conhecido, de modo que a distribuição de probabilidade de transição entre os estados e de geração de informações exógenas deve estar acessível ao modelo.

Diante do problema denominado de maldição da dimensionalidade, surge um dos principais impasses da programação dinâmica tradicional, que é o fato da equação de Bellman não ser aplicável para qualquer problema de decisão sequencial. Entretanto, há um caminho que reúne as propriedades vantajosas da programação dinâmica e as combina a técnicas de aproximação de funções e simulação, equilibrando a utilização de recursos computacionais na busca de políticas sub-ótimas eficientes (Wang *et al.*, 2009).

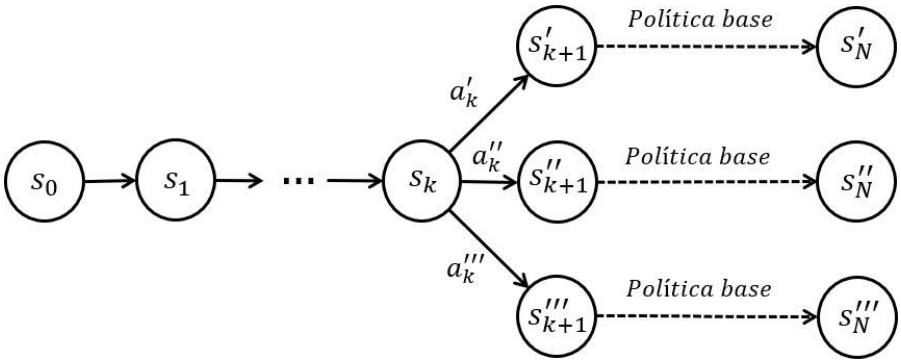
Neste trabalho, dadas as condições da alta dimensionalidade do espaço de estados e decisão do problema de alocação de veículos, a solução apresentada tem como objetivo encontrar uma política que produza uma solução viável e aceitável baseada do algoritmo *rollout*. *Rollout* é uma ferramenta heurística de otimização sequencial útil para resolver problemas de programação dinâmica, cujo procedimento envolve escolher decisões em tempo real somente para os estados visitados, via uma política de decisão *lookahead* (Goodson *et al.*, 2017). A técnica *rollout* é um

método de aproximação categorizado na classe de algoritmo de iteração de política aproximada, em que a função de valor é aproximada por uma função computacionalmente viável e que não requer armazenamento.

O algoritmo *rollout* funciona simulando futuras trajetórias do sistema para cada estado visitado pelo decisor seguindo uma política inicialmente dada para cada estado possível. As trajetórias simuladas podem ser utilizadas para aproximar a função de valor destas políticas iniciais. O método então aplica uma etapa de otimização que produz uma nova política pelo menos tão boa quanto a anterior se algumas condições, que serão apresentadas logo a seguir, forem satisfeitas. Este processo iterativo do *rollout* equivale ao método de iteração de política aproximada com uma única iteração (Bertsekas, 2013).

É importante compreender que o propósito do *rollout* não é obter uma estimativa completa da função de valor. O principal objetivo do algoritmo *rollout* é melhorar uma dada política, ou seja, partindo de uma política sub-ótima, definida *a priori* para o horizonte do problema, chamada de política base, deseja-se produzir uma política melhorada, chamada de política *rollout* (Bertsekas, 2007). É garantido que a política *rollout* melhora a política base se algumas condições sobre esta política forem respeitadas. Tais condições serão apresentadas mais à frente.

Figura 12 – Ilustração da estrutura lógica do algoritmo *rollout*.



Fonte: (Bertsekas, 2019), adaptado pela autora (2022)

Em princípio, considere um PDM cujo espaço de estados S é um conjunto finito e enumerável, onde $s_k \in \mathsf{S}$ é o estado do sistema na época de decisão k. Define-se

$$S_{s_k}^{a_k} = \{s_{k+1} | \mathrm{P}(s_{k+1} | s_k, a_k) > 0\}$$

como o conjunto de estados imediatamente alcançáveis quando o processo ocupa o estado s_k e a decisão $a_k \in \mathsf{A}_k$ é aplicada. Dado o estado s_k o algoritmo *rollout* considera todos os

subproblemas que evoluem a partir de cada estado $s_{k+1} \in S_{s_k}^{a_k}$ ao longo das épocas de decisão em um horizonte de tempo limitado, e os resolve por meio da heurística base H (veja Figura 12). A partir da heurística base H é computada uma sequência de estados e decisões que formam uma trajetória $h = \{s_{k+1}, a_{k+1}, \cdots, a_{N-1}, s_N\}$, em que todo estado subsequente desta trajetória é gerado por simulação. O algoritmo *rollout* escolhe a decisão $\hat{a}_k = \hat{\pi}_k(s_k)$ no estado s_k que otimiza a soma do retorno esperado imediato e a estimativa da função de valor futuro, tal que:

$$\hat{a}_k \in \arg \min_{a_k \in A(s_k)} \{C(s_k, a_k) + \hat{g}_{\pi_H}(s_{k+1})\}, \tag{2.44}$$

sendo $\hat{g}_{\pi_0}(s_{k+1})$ uma estimativa do retorno médio associado à trajetória h começando no estado $s_{k+1} \sim P(s_{k+1}|s_k, a_k)$ seguindo a política base π_H. Então, este processo define uma política sub-ótima $\hat{\pi}_k(s_k)$ denominada de política *rollout*.

2.4.1 Condições de melhoria da política

Para o algoritmo *rollout* a escolha da política base fica aberta, não há nenhuma restrição limitadora para a execução do algoritmo. Diversos métodos de otimização aproximada podem ser utilizados, tais como algoritmos gulosos, busca local, algoritmos genéticos, entre outros (Bertsekas, 2019). Entretanto, a escolha de uma boa política base é de extrema relevância para uma melhor performance do algoritmo. Além disso, é desejável que a política base possa oferecer soluções de boa qualidade e sua execução seja computacionalmente tratável. Assim, para garantir que o desempenho da política *rollout* seja tão boa ou melhor que o da política base, algumas propriedades devem ser asseguradas. Estas propriedades sobre a política base são denominadas de sequencialmente consistente e sequencialmente melhorável.

De modo geral, dizemos que uma política base é sequencialmente consistente se sempre que esta política produzir uma trajetória parcial

$$\{s_k, a_k, s_{k+1}, a_{k+1}, \cdots, s_N\}$$

começando do estado s_k, então também produzirá a trajetória

$$\{s_{k+1}, a_{k+1}, s_{k+2}, a_{k+2}, \cdots, s_N\}$$

começando do estado s_{k+1}. Em outras palavras, ser sequencialmente consistente significa que a trajetória gerada pela política base não é alterada à medida em que estados subsequentes passam a ser o estado inicial da sequência. Políticas bases sequencialmente consistentes são frequentemente utilizadas. Por exemplo, políticas gulosas tendem a ser sequencialmente consistentes (Bertsekas, 2020).

Adicionalmente, para que a política *rollout* tenha um desempenho melhor do que a política base, espera-se que a política base seja uma política sequencialmente melhorável. Então, sabendo que a decisão que será aplicada no estado s_k é determinada por

$$\hat{a}_k \in_{a_k \in A(s_k)} \{C(s_k, a_k) + \hat{g}_{\pi_H}(s_{k+1})\},$$

diz-se que a política base é sequencialmente melhorável se para todo estado s_k, tem-se que

$$\min_{a_k \in A_k} \{C(s_k, a_k) + \hat{g}_{\pi}(s_{k+1})\}_{H} \leq \hat{g}_{\pi}(s_k). \quad (2.45)$$

Ou seja, a propriedade afirma que a função de valor aproximada seguindo a política base gera um retorno esperado tão bom ou melhor do que aplicar localmente a política base sobre cada estado.

O conceito do algoritmo *rollout* formulado até este momento pode ser facilmente modelável para a versão generalizada de um PDM, que é um PDSM. A ideia geral permanece a mesma, mas a função de valor precisa ser readequada a fim de contemplar o fato de que a evolução do processo ocorre em um horizonte de tempo contínuo. Então a equação (2.44) passa a ser

$$\hat{a}_k \in_{a_k \in A_k} \{C(s_k, a_k) + \hat{g}_{\pi_H}(S(t_k + dt))\} \quad (2.46)$$

sendo $S(t_k + dt)$ o estado imediatamente posterior a época decisão k, que não necessariamente será o estado imediatamente posterior ao estado atual, mas sim o próximo estado de decisão que está diretamente condicionado à decisão aplicada na época k.

2.4.2 Vantagens do algoritmo rollout

Por fim, embora algoritmos *rollout* exijam mais recursos computacionais do que executar somente o algoritmo heurístico, ainda sim há algumas vantagens em sua utilização. Estas são:

1. Podem ser utilizados tanto no contexto *offline* quanto *online*, pois calculam aproximações para avaliar funções quando necessário, e sua aproximações não são armazenadas.
2. Muitos problemas possuem políticas de decisão heurísticas com bons desempenhos validados na literatura e propriedades bem estabelecidas, que podem ser otimizados a partir da utilização do algoritmo *rollout*.
3. Existe a possibilidade de ser combinado com outros métodos *offline* de programação dinâmica aproximada, como por exemplo uma rede neural produzindo políticas base para aproximar a função de valor.

3 TRABALHOS RELACIONADOS

O problema de alocação de veículos também pode ser encontrado na literatura como problema de gestão de frota dinâmico (*dynamic fleet management*). Um dos primeiros trabalhos a abordar o problema de alocação dinâmico aplicado a área de transporte foi apresentado por Powell (1996) , que desenvolveu modelos de programação matemática determinísticos e estocásticos que tratam de previsões de demanda e reposicionamento de caminhões. Desde então diversos modelos foram apresentados, e neste contexto houveram diversas contribuições de métodos aproximados. Como exemplo, pode-se destacar métodos de programação dinâmica aproximada para lidar com o problema de gerenciamento de frota em grande escala tais como os desenvolvidos por Godfrey e Powell (2002) e Simão *et al.* (2009). A ideia desenvolvida por estes autores se resumia a estender o problema de alocação clássico para um ambiente multiperíodo dado por um processo de decisão markoviano de tempo discreto. Em cada época de decisão, há muitos veículos disponíveis para alocação a muitas tarefas, resultado em um largo espaço de decisões. Para lidar com essa adversidade derivada do problema combinatório, uma das possíveis técnicas era realizar aproximações lineares da função de valor. Tal abordagem permitia resolver de forma eficiente o problema de decisão como um problema de programação linear.

Nos primeiros anos de uso das plataformas digitais de mobilidade, as empresas adotaram política simples para o despacho de veículos às solicitações. Liao (2001) e Lee *et al.* (2004) adotam em seus trabalhos política míopes simples para alocação de veículos com base no menor tempo de trajeto para chegar ao local do cliente, considerando as condições do tráfego no momento da decisão. A principal desvantagem dessas políticas é que elas agem localmente e de forma míope, não levando em conta o impacto global das decisões sobre estados futuros e, portanto, não otimizam uma sequência de decisões, mas uma decisão por vez.

Seow *et al.* (2009) formula o problema de alocação de veículos em cada época de decisão como um problema de alocação linear cujo objetivo é minimizar a soma dos tempos estimados de viagem de veículos disponíveis para solicitações atualmente pendentes. Em vez de resolver o problema de forma centralizada, os autores propõem um algoritmo colaborativo no qual grupos de agentes representado por motoristas negociam entre si até chegar em uma decisão satisfatória. Os autores mostram através de experimentos computacionais que essa abordagem supera regras simples aplicadas por sistemas centralizados. Zhang *et al.* (2017) formularam um problema de alocação cuja função objetivo era a soma das probabilidades dos condutores aceitarem a alocação. Estas probabilidades foram estimadas a partir de um modelo de regressão

logística. Como a função objetivo do problema de alocação é não linear, os autores fizeram uso de uma heurística para resolvê-la. Bertsimas *et al.* (2019) trata o problema de alocação de veículos a partir de uma técnica de re-otimização na qual um modelo estático de programação inteira mista é resolvido em intervalos de tempo fixos. O modelo contempla todas as solicitações pendentes, todos os chamados ociosos e os tempos de viagens estimados no momento da otimização e implementa apenas as ações que podem ser aplicadas antes da próxima época de decisão. Mais recentemente, trabalhos utilizando técnicas de aprendizado por reforço foram apresentados à comunidade. Xu *et al.* (2018) faz uso de um modelo de decisão markoviano em que o provedor do serviço deve combinar as solicitações e os veículos disponíveis em intervalos de tempo constantes. Para lidar com o problema do espaço de decisões se tornar suficientemente grande, os autores assumem que a função de valor global é separável em funções de valor locais associadas aos veículos, e o modelo conduz ao aprendizado de uma função de valor para cada veículo, como resultado, o espaço de decisões é reduzido. Os autores deste trabalho fazem uso de dados históricos de viagens de táxi para aprender a função de valor de uma política base usada durante a amostragem utilizando uma representação tabular de programação dinâmica. As decisões são tomadas em tempo real pelo servidor resolvendo um problema de alocação linear cuja função de custo é a soma das funções aprendidas por cada veículo.

Para superar as limitações da representação tabular, Wang *et al.* (2018) desenvolveram uma técnica baseada em aprendizado por reforço profundo, em que uma *deep Q-network* recebe pares de estado e decisão como entrada e retorna à função de valor estado e decisão associada a este par. A rede neural é treinada usando gradiente descendente com dados de histórico de viagens. Então, a função de valor de estado e decisão aprendida é utilizada como a função objetivo do problema de alocação. Uma abordagem semelhante é desenvolvida no trabalho elaborador por Liang *et al.* (2021). Tang *et al.* (2019) estende a formulação baseada em um processo de decisão markoviano para um modelo baseado em um processo de decisão semi-markoviano, e desenvolve um nova arquitetura de rede neural. Muitos destes desenvolvimentos são sumarizados por Qin *et al.* (2020).

Outros trabalhos consideram a tarefa de reposicionamento, seja de forma separada ou integrada ao despacho dos veículos. Miao *et al.* (2016) propõe uma abordagem de controle de horizonte de retrocesso para o reposicionamento de veículos em antecipação à demanda, de modo a equilibrar a oferta e a demanda e minimizar a distância total. Holler *et al.* (2019) consideram que as épocas de decisão ocorrem sempre que um veículo fica livre, e então decide

entre escolher uma solicitação pendente a ser atendida dentro de um raio de transmissão do veículo ou reposicionar o veículo se não houver nenhum chamado em esperar neste raio. Neste trabalho, é utilizado uma *deep Q-neural* e aproximações de políticas para aprender a política de decisão. Seus resultados mostram que as políticas obtidas a partir do processo de aprendizado é comparativamente melhor em relação às políticas míopes simples.

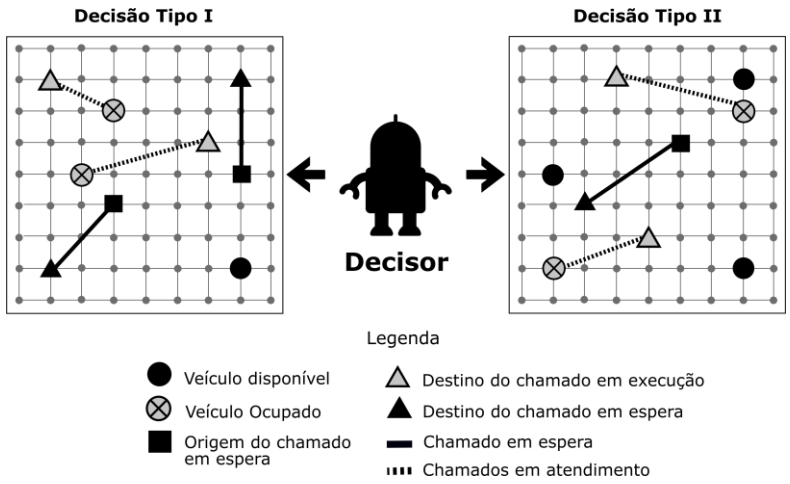
Kullman *et al.* (2021) desenvolveram uma técnica de aprendizado por reforço profundo (*deep reinforcement learning*) para o problema de alocação com veículos elétricos. Em cada época de decisão, um servidor central deve decidir quais tarefas atribuir a cada veículo elétrico, como por exemplo atender uma solicitação pendente ou reposicionar o veículo para uma estação de carregamento. Os autores também utilizaram redes neurais profundas (*deep Q-neural*) para aprender a função de valor e comparar as políticas aprendias com uma abordagem de re-otimização míope. Liu *et al.* (2020) desenvolveram uma abordagem baseada em *deep Q-neural* sensível ao contexto de reposicionamento de veículos. No estudo proposto pelos autores, a decisão envolvia reposicionar táxis ociosos para outros pontos de espera, dada a demanda prevista, a fim de equilibrar oferta e demanda. A alocação desses táxis aos chamados é decidida considerando a menor distância. Tang *et al.* (2021) apresentam uma técnica integrada para despacho e reposicionamento. Eles utilizaram uma rede neural para representar a função de valor. A obtenção da política era dada a partir da resolução de um problema de alocação linear cuja função objetivo é dada pela soma dos erros de diferença temporal associadas a par recurso-tarefa.

4 FORMULAÇÃO DO PROBLEMA DE ALOCAÇÃO DE VEÍCULOS COMO UM PDSM

Considere um ambiente em que há um conjunto com n veículos que recebem solicitações de atendimento que chegam continuamente ao longo do tempo. Neste ambiente existe um agente munido da capacidade de decidir qual veículo atribuir a uma dada solicitação de serviço. Estas solicitações são concentradas em uma central de atendimento, e chegam ao decisor como um pedido para realizar o translado de um certo ponto de origem até um outro ponto de destino. Quando o decisor realiza a alocação do veículo ao chamado, o veículo parte da sua localização atual em direção ao ponto de origem, e então se move até o destino, concluindo a execução do atendimento. Após isto, o veículo assume um estado de espera, permanecendo no local até que o decisor atribua a ele um outro chamado. Uma boa política para este problema será aquela que minimiza o tempo médio de espera para o atendimento de cada chamado, medido desde o momento em que o chamado é recebido na central de atendimento até o momento em que o veículo chega ao ponto de origem da corrida.

Cada chamado que chega ao sistema surge em um ponto aleatório do tempo. Tal informação é recebida pela central de atendimento como uma demanda que precisa ser suprida. Então, o fluxo de decisão sobre qual tarefa alocar a qual recurso aciona o agente decisor em dois contextos: sempre que um veículo, antes ocupado, torna-se disponível ou quando, havendo pelo menos um veículo disponível e nenhuma chamada em espera, uma nova solicitação é recebida. Neste contexto, o decisor pode facilmente ser um software que recebe as tarefas e distribui aos veículos disponíveis.

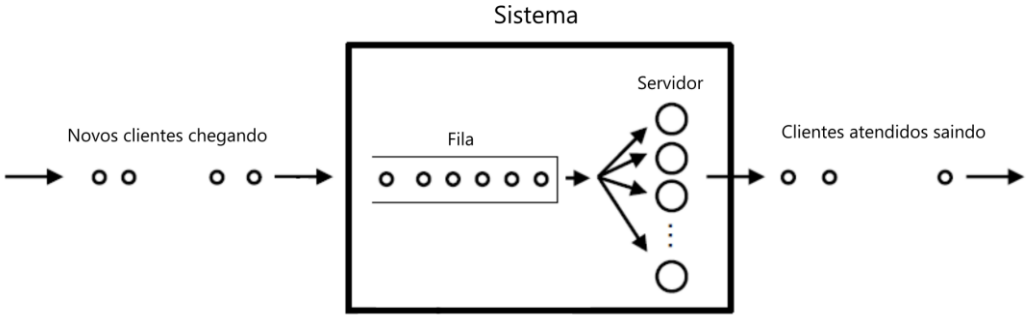
Figura 13 – Representação dos eventos que acionam o decisor para realizar alocação do chamado ao veículo.



Fonte: Elaborado pela autora (2022).

Note que, enquanto não houver pelo menos um veículo disponível para realizar a tarefa, novos chamados podem continuar chegando ao sistema, o que implica em uma formação de fila de chamados. Semelhantemente, enquanto não existirem chamados em espera, novos veículos podem ter seu estado alterado para ocioso, formando assim uma fila de veículos. Esta característica do problema permite que seja feito uso dos fundamentos de um sistema de filas para auxiliar na modelagem da dinâmica do sistema.

Figura 14 – Representação de um sistema de filas clássico.



Fonte: (Shortle *et al.*, 2018), editado pela autora

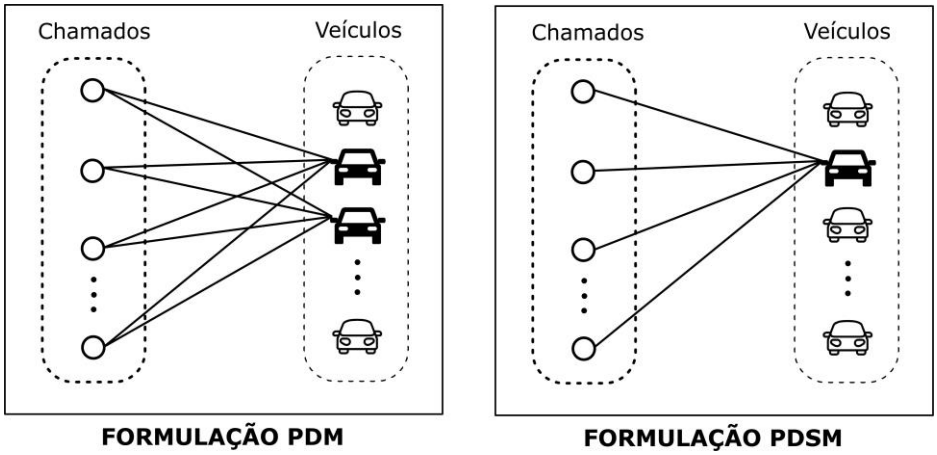
A Figura 14 mostra uma representação do sistema de fila. Este sistema é modelado a partir de alguns componentes elementares, três destes estão apresentados abaixo (Shortle *et al.*, 2018):

- **Padrão de chegada dos clientes:** O cliente em um sistema de filas é o elemento que demanda o atendimento do servidor do sistema. Geralmente, em um sistema de filas o processo de chegada é estocástico. Assim, para que o processo de chegada seja modelado é necessário conhecer a distribuição de probabilidade que rege o tempo de chegada dos chamados no sistema;
- **Padrão de atendimento do servidor:** Cada servidor em um sistema de filas assume o papel de atender a demanda dos clientes que estão no sistema. Semelhante ao padrão de chegada dos clientes, o padrão do atendimento no geral é estocástico. Portanto, também é necessário conhecer a distribuição de probabilidade que produz a variável aleatória que corresponde ao tempo de atendimento de cada servidor;
- **Disciplina da fila:** Refere-se à maneira pela qual os clientes são selecionados para o atendimento quando uma fila se forma.

Para o problema de alocação de veículos as filas seriam formadas em dois contextos, no caso da decisão tipo I (veja Figura 13), os chamados são os clientes e os servidores são os

veículos, e o processo de chegada dos chamados e o tempo de atendimento dos servidores são conhecidos. Para a decisão tipo II, os clientes são os veículos e os servidores são os chamados. Nos dois casos, o decisor gerencia a fila, executando a política de atribuição. Entretanto, este problema não se enquadra em um sistema de filas clássicos, pois os servidores do sistema mudam ao longo do tempo e existe um atributo espacial, conferindo aos elementos do processo uma variável adicional, atualizada dinamicamente ao longo do evolução do sistema. Portanto, é indispensável que a política de decisão contemple esta característica espacial, dado que à medida que mensurará a qualidade das decisões dependerá diretamente da localização dos cliente e servidores.

Figura 15 – Representação do espaço de decisões para um PDM e para um PDSM.



Fonte: Elaborado pela autora (2022)

Embora problemas de decisão sequencial sejam geralmente modelados como um PDM, este problema de alocação de veículos em especial, possui certas características que fundamentalmente precisam ser contempladas pelo modelo utilizado para representá-lo, e um PDM não as contempla. Esta particularidade se dá com relação ao tempo no qual as épocas de decisão ocorrem. Em um PDM, as épocas de decisão ocorrem em intervalos de tempo constantes. Isto implica dizer que entre as épocas de decisão, muitos veículos podem ficar ociosos ao mesmo tempo que a fila aumenta com a chegada de novos chamados, resultando em um espaço de estados arbitrariamente grande. Em contrapartida, em um PDSM as épocas de decisão ocorrem em intervalos de tempo discretos, cuja vantagem está no fato de que cada decisão é demanda mediante a ocorrência de um determinado evento. Por exemplo, no caso do evento que aciona a decisão do tipo I, um único veículo fica ocioso por vez. Portanto, uma das principais motivações para formular o problema de alocação de veículos como um PDSM é a possibilidade de redução

da dimensionalidade do espaço de decisões, dado que estaremos sempre decidindo a alocação de um veículo por vez ou de um chamado por vez.

Como exemplo, considere uma época de decisão em um PDM em que 10 chamados estão aguardando para serem atendidos, e existem 3 veículos ociosos no momento. O número de decisões viáveis neste estado corresponde a todas as correspondências possíveis entre o conjunto de chamados e o conjunto de veículos, que é $10 \times 9 \times 8 = 720$. Por outro lado, em um PDSM as decisões seriam consideradas em um veículo por vez, totalizando $10 + 9 + 7 = 26$ decisões viáveis. É notório a redução do esforço computacional à medida em que o número de recursos e tarefas aumenta.

4.1 FORMULAÇÃO MATEMÁTICA

Considere $n \in \mathrm{N}$ veículos idênticos $v_1, v_2, \ldots, v_n \in \mathrm{V}$, em que V é o conjunto de todos os veículos disponíveis para alocação. Estes veículos são conhecidos por um agente decisor, que gerencia a alocação de solicitações aos veículos. Para cada tempo t existe um conjunto de chamadas em espera $\mathrm{R}(t)$, sendo cada chamado representado por uma tupla $r_j(t) = (o_j, d_j, w_j)$, em que o_j e a localização da origem do chamado, d_j a localização do destino do chamado, w_j o instante no qual o chamado chegou ao sistema e $j \in \mathrm{N}$. Note que para qualquer t, existe a possibilidade de $\mathrm{R}(t) = \emptyset$. Tanto a origem o_j quanto o destino d_j assumem valores que pertencem a um conjunto finito de localizações L, e o tempo de chegada do chamado w_j assumem valores em $[0, \infty)$. Vale destacar que a suposição de um conjunto finito de localizações não é restritiva do ponto de vista da aplicação, pois comumente o espaço de localizações é discretizado.

Defina o conjunto de veículos $\mathrm{V}(t)$ que muda dinamicamente ao longo do tempo. Um veículo é definido como uma tupla $v_j(t) = (l_j(t), b_j(t))$, com $j \in \{1, 2, \ldots, n\}$. Define-se $l_j(t) \in \mathrm{L}$ como sendo a localização do veículo no tempo t e $b_j(t) \in \{0, 1\}$ a variável de disponibilidade do veículo, em que $b_j(t) = 1$ se o veículo estiver ocupado e $b_j(t) = 0$, caso contrário.

A variável de estado do sistema $S(t)$ em um dado instante t será uma tupla dada pelo conjunto de veículos e o conjunto de chamadas solicitadas, isto é, $S(t) = (\mathrm{V}(t), \mathrm{L}(t))$, e $t \in [0, T)$, sendo T o limite do horizonte de decisão. Portanto, S será o conjunto de todos os possíveis estados do sistema.

A transição de estados neste modelo é conduzida a partir da chegada de novos

chamados ao sistema e da atualização da disponibilidade dos veículos. Assim, o conjunto de veículos no instante $t + 1$ será

$$V(t + 1) = V(t) - \{v^{\wedge}(t)\} + V^{+}(t + 1),$$

ou seja, será o conjunto de veículos disponíveis no instante t subtraído o conjunto que contém o veículo que foi alocado somado ao conjunto com os veículos que se tornaram ociosos no instante $t + 1$. Semelhantemente, tem-se que o conjunto de chamados no instante $t + 1$ será o conjunto de chamados em espera no instante t subtraído o conjunto que contém o chamado atendido mais o conjunto dos chamados que chegaram no instante $t + 1$, tal como apresentado abaixo:

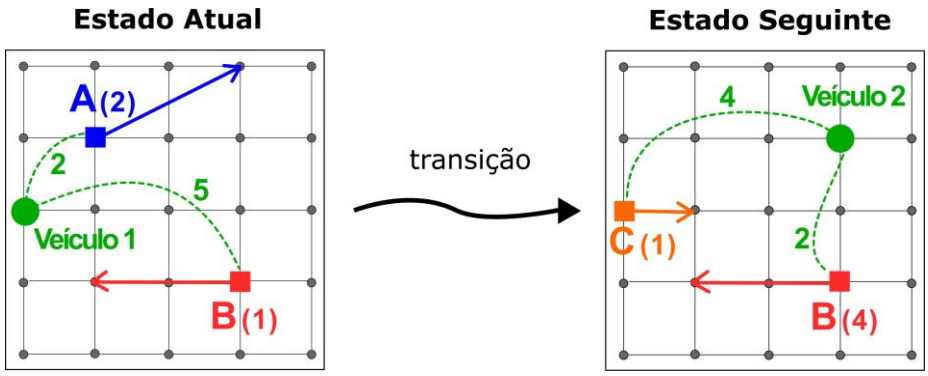
$$R(t + 1) = R(t) - \{r^{\wedge}(t)\} + R^{+}(t + 1).$$

Sempre que um veículo ficar livre, se $R(t) \neq \emptyset$, o decisor irá atribuir este veículo a um dos chamados que estão em espera. Em seguida, o veículo se deslocará em direção ao ponto de origem do chamando, partindo de sua localização atual, e posteriormente seguirá em direção ao destino do chamado, após o qual terminará a execução da tarefa. O sistema evolui continuamente no tempo. Entretanto, as épocas de decisão ocorrem em instantes discretos de tempo representados por t_k, $k \in \{0, 1, 2, 3, \ldots\}$. Estas épocas de decisão ocorrem em dois contextos:

1. Todos os veículos estão ocupados e um veículo termina a corrida, tornando-se livre, e exitem chamados em espera. Neste caso, o decisor escolhe qual chamada será atribuída ao veículo (Veja Figura 13 do lado esquerdo).
2. Existe um ou mais de um veículo livre e uma nova chamada chega ao sistema quando não há nenhum outro chamado em espera. Neste caso, o decisor escolhe qual dos veículos será atribuído a este chamado (Veja Figura 13 do lado direito).

A Figura 16 ilustra a transição entre dois estados de decisão no contexto de decisão tipo I. No estado atual, existem dois chamados em espera, o chamado A e o chamado B, cujo tempo de espera na fila é respectivamente, 2 unidades de tempo e 1 unidade de tempo. Existe também um veículo que se tornou ocioso neste estado. Nesse momento, o decisor pode escolher alocar ao veículo 1 o chamado A, onde o tempo de descolamento do veículo até o ponto de origem será de 2 unidade de tempo, ou pode escolher alocar ao chamado B, com tempo de deslocamento de 5 unidade de tempo. Então o decisor escolhe atribuir o veículo ao chamado A, e o chamado B permanece na fila em espera. Após 3 unidades de tempo, um outro veículo torna-se livre e disponível para alocação. Agora, um novo estado de decisão, subsequente ao anterior, se

Figura 16 – Representação da transição entre estados do PDSM quando ocorre um evento de decisão do tipo I.



Fonte: Elaborado pela autora (2022)

torna conhecido ao decisor. Neste estado, o tempo de espera do chamado B foi incrementado, sendo 4 unidades de tempo e um novo chamado C chegou ao sistema e está em espera há 1 unidade de tempo.

Em um dado estado $s \in \mathsf{S}$, o decisor executa uma decisão $a \in \mathsf{A}_s$, que corresponde ao chamado que será alocado a um determinado veículo no caso da decisão tipo I, ou o veículo que será alocado a um determinado chamado no caso da decisão tipo II. Dado o estado s e a decisão a, o sistema apresenta ao decisor o retorno esperado dado por

$$c^{-}(s, a) = C(s, a) + \mathrm{E} \int_{t_k}^{t_{k+1}} \delta(s, a, S(t))\, dt \quad , \tag{4.1}$$

em que $C(s, a)$ é o custo imediato correspondente ao tempo de espera na fila do chamado escolhido, $\delta(s, a, S(t))$ é a taxa de custo adicional, sendo $\delta(s, a, S(t)) = 1$ enquanto o processo natural evolui no intervalo $[t_k, t_{k+1} + \Delta(s, a)]$ e $\delta(s, a, S(t)) = 0$ enquanto o processo natural evolui no intervalo $[t_k + \Delta(s, a), t_{k+1} + \Delta(s, a)]$, onde $\Delta(\cdot, \cdot)$ é uma variável aleatória que retorna o tempo de deslocamento do veículo de sua localização atual até a localização de origem do chamado.

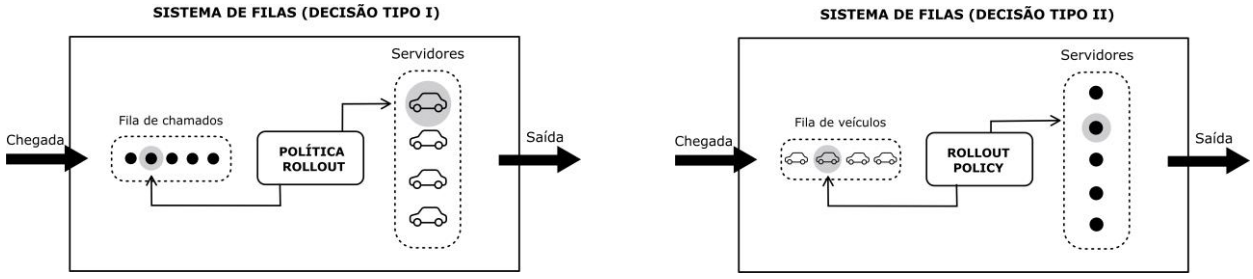
4.2 UM ALGORITMO *ROLLOUT* BASEADO EM SIMULAÇÃO DE EVENTOS DISCRETOS

Partindo da premissa adotada, que o problema de alocação de veículos é modelado como um PDSM, o fluxo do processo natural é um componente indispensável. Na seção anterior foi definido como ocorrem as épocas de decisão, ou seja, como é dado o processo semimarkoviano. No caso do processo natural, que modela a evolução do sistema em estados que

não são estados de decisão, haverá um simulador controlando a dinâmica enquanto o algoritmo *rollout* é utilizado para estimar a função de valor dos estados futuros.

A chegada dos chamados ao sistema, bem como a dinâmica de atribuição permitem uma interpretação complementar do problema como sendo um sistema de filas controlado. A Figura 17 mostra uma representação do funcionamento do sistema para os dois contextos de decisão: quando os servidores são os veículos (Figura 18(a)) e quando os servidores são os chamados (Figura 18(b)).

Figura 17 – Representação do sistema de fila para o sistema de alocação de veículos.



(a) Sistema de filas para a decisão do tipo I. (b) Sistema de filas para a decisão do tipo II.

Fonte: Elaborada pela autora (2022)

Para além dos componentes elementares de um sistema de fila, tais como os clientes, servidores, processo de chegada e política da fila, há um outro componente que precisa ser contemplado na modelagem do problema de alocação de veículos. Os servidores e os clientes estão posicionados sobre uma malha de localizações. Esta característica adicional influencia potencialmente a qualidade do atendimento, uma vez que a decisão ideal objetiva minimizar o tempo de espera do cliente.

O sistema de filas, bem como o ambiente do processo natural do problema de alocação de veículos será reproduzido via simulação. A simulação de eventos discretos (SED) é a principal técnica utilizada para simular sistema de filas. SED utiliza um mecanismo muito inteligente para simular o tempo contínuo. Primeiro, baseia-se no conceito do evento discreto, que é qualquer ocorrência que altera o estado do sistema em intervalos discretos de tempo. O sistema muda de estado somente quando ocorre um evento e permanece no mesmo estado durante os intervalos de tempo entre os eventos. Desta forma, ao invés de simular a passagem do tempo em pequenos incrementos, a SED funciona visitando os instantes nos quais os eventos ocorrem e atualizando o estado do sistema de acordo com a necessidade. Esta característica atribui uma vantagem computacional, pois torna a simulação muito mais eficiente, desprezando

o instante de tempo no horizonte nos quais não há alteração do estado do ambiente. No geral, um sistema de simulação de eventos discreto pode ser modelado a partir dos seguintes componentes (Law, 2014):

- **Estado do sistema:** São as variáveis que descrevem o sistema em um determinado momento;
- **Relógio de simulação:** É a Variável que fornece o valor atual do tempo simulado;
- **Lista de eventos:** Lista que contém o instante no qual um certo tipo de evento irá ocorrer;
- **Rotina de inicialização:** Um subprograma para inicializar o modelo de simulação no instante inicial;
- **Rotina de temporização:** Um subprograma que determina o próximo evento da lista de eventos e, em seguida, avança o relógio da simulação para o instante em que ele ocorre;
- **Rotina de eventos:** Um subprograma que atualiza o estado do sistema quando um determinado tipo de evento ocorre.

4.2.1 Simulação do ambiente de decisão

O ambiente de decisão para o problema de alocação de veículos foi implementado em Python utilizando a biblioteca de simulação de eventos discretos SimPy 4.0. O simulador tem por principal objetivo reproduzir o processo natural associado ao PDSM. Então, inicialmente foram implementados os elementos principais do problema de alocação de veículos: os recursos, que são os veículos; as tarefas, que são os chamados e o agente decisor.

Para implementar o ambiente de simulação foram utilizadas algumas bibliotecas auxiliares, como: Numpy 1.22.4, SciPy 1.8.1, OS 3.10.5, Time 3.9.13 (veja código-fonte 1).

Código-fonte 1 – Bibliotecas Python importadas na implementação do modelo de simulação.

```
1 import os
2 import simpy
3 import numpy as np
4 from numpy.random import default_rng
5 from scipy.spatial.distance import cdist
6 import time
```

Para modelar os recursos do problema de alocação dentro da implementação do ambiente de simulação foi criada a classe `Car` (veja código-fonte 2). Esta classe recebe como

parâmetro o agente, o número de veículos no ambiente, a matriz de deslocamento e as localizações. Cada veículo tem um conjunto de atributos tais como sua localização atual, seu *status* de ocupação, e um número de série para identificação. A classe Car tem dois principais métodos:

1. Método life: Define o processo principal de um veículo, constituindo sua existência no modelo;
2. Método ride: Define o processo de atendimento do chamado. Atualizando a localização do veículo e seu *status*. Este método retorna o atraso e o tempo de duração da viagem.

Código-fonte 2 – Classe utilizada para definir o elemento veículo no modelo de simulação.

```
class Car:
    def __init__(self, env, agente, car_number, time_matrix, location):
        self.env = env
        self.agente = agente
        self.number = car_number
        self.time_matrix = time_matrix
        self.location = rd.choice(location)
        self.busy = False
        self.life_proc = env.process(self.life())
        self.start_driving = env.event()

    def life(self):
        while True:
            received_call = yield self.start_driving
            self.busy = True
            atraso, duracao_origem = yield self.env.process(self.ride(
                received_call))
            self.busy = False
            self.agente.free_car.succeed((self.number, atraso,
                duracao_origem))
            self.start_driving = self.env.event()

    def ride(self,call)
        duracao_viagem_origem = self.time_matrix[self.location,call.
            origem]
        yield self.env.timeout(duracao_viagem_origem)
        self.location = call.origem
        atraso = self.env.now - call.birth_time
```

```
26          duracao_viagem_destino = self.time_matrix[self.location, call.
                dest]
27          yield self.env.timeout(duracao_viagem_destino)
28          self.location = call.dest
29
30          return atraso, duracao_viagem_origem
```

Um novo chamado passa a existir dentro do ambiente de decisão sempre que a classe Call é acionada. Então, para cada chamado são definidos os atributos que especificam a origem e o destino do chamado, o momento em que este chamado chegou ao sistema e seu número identificador (veja código-fonte 3). Os chamados chegam ao sistema seguindo um processo de Poisson, ou seja, os intervalos entre chamados consecutivos é uma variável aleatória que segue uma distribuição exponencial com parâmetro λ, sendo λ a taxa média de chegada

Código-fonte 3 – Classe utilizada para definir um novo chamado no modelo de simulação.

```
1 class  Call:
2     def __init__(self, serial_number, birth_time, origem, dest):s
3         self.serial_number=serial_number
4         self.birth_time = birth_time
5         self.origem = origem
6         self.dest = dest
```

A classe Call modela um novo chamado, e o nascimento de cada novo chamado é orientado pelo padrão de chegada dos clientes. Este padrão é modelado pela função arrival_process. Lembre-se que durante a execução do algoritmo *rollout* são produzidas trajetórias para mensurar o impacto das decisões no futuro. Neste modelo, foi adotado a premissa que durante a execução da simulação para produzir estas trajetórias futuras não ocorre a chegada de novos chamados. Por isso, foi necessário implementar um controle para identificar qual era o ambiente de simulação do processo natural e qual o ambiente de simulação que produz as trajetórias. Para este controle foi utilizado um artifício do sistema operacional para clonar processos. Basicamente, sempre que o agente estivesse diante de um estado de decisão, um clone do ambiente era gerado. Neste processo filho, cópia do processo do ambiente natural, o agente seguia alocando os chamados em espera aos veículos disponíveis orientados pela política base. Quando a fila esvaziasse ou um tempo limite fosse atingido, o processo filho era encerrado e

retornava ao agente o atraso médio acumulado da trajetória. No código-fonte 4, as linhas 6 e 7 dizem que sempre um processo filho estiver em execução novos chamados não devem chegar ao sistema. Por fim, para cada novo chamado é indicado seu ponto de origem, seu ponto de destino, o momento em que chegou ao sistema e em seguida é adicionado à lista de eventos.

Código-fonte 4 – Função que modela o padrão de chegada dos chamados no sistema.

```
1  def arrival_process(env, tec, locations, agente):
2      global rd
3      global pid
4      serial_numbers = 0
5      while True:
6          if pid == 0:
7              break
8          possible_locations = list(locations)
9          origem = rd.choice(possible_locations)
10         possible_locations.remove(origem)
11         destino = rd.choice(possible_locations)
12         call = Call(serial_numbers, env.now, origem, destino)
13
14         if len(agente.lista_calls) == 0:
15             agente.new_call.succeed()
16         agente.lista_calls.append(call)
17         interarrival_time = rd.exponential(tec)
18         serial_numbers += 1
19         yield env.timeout(interarrival_time)
```

Apresentado como foram implementados os veículos e os chamados no sistema de simulação de eventos discretos, é oportuno explorar com maiores detalhes a implementação do agente decisor, que rege a disciplina da fila. Ele é acionado em dois contextos: quando ocorre o evento do tipo I (veículo livre) ou evento do tipo II (novo chamado). Nestes casos, o método `control` da classe `Agente` é acionado (veja código-fonte 5). O método `control` carrega em si o núcleo do algoritmo *rollout*. Na linha 19 do código-fonte 5 é feito a validação dos eventos do tipo I e tipo II. Quando ocorre o evento do tipo II (linha 20 até linha 34), o agente decisor simplesmente aloca o chamado ao veículo mais próximo, otimizando o tempo de deslocamento até seu ponto de origem. A política de decisão neste caso é dada pela função `choose_car`.

Código-fonte 5 – Função que modela o padrão de chegada dos chamados no sistema.

```
class Agent:
    def __init__(self, env, rollout, politica):
        self.env = env
        self.rollout = rollout
        self.start_time = 0
        self.cars = []
        self.lista_calls = []
        self.atraso_total = 0
        self.duracao_chegada_total = 0
        self.free_car = env.event()
        self.new_call = env.event()
        self.controller_proc = env.process(self.control())
        self.politica = politica


    def control(self):
        global rd
        while True:
            yield self.free_car | self.new_call
            if self.new_call.triggered:
                self.new_call = self.env.event()

                veiculo_livre = False
                free_cars = []
                for car in self.cars:
                    if car.busy == False:
                        free_cars.append(car.number)
                        veiculo_livre = True

                if veiculo_livre == True:
                    selected_call = self.lista_calls.pop()
                    car_index = self.choose_car(free_cars,selected_call)
                    selected_car_number = free_cars[car_index]
                    selected_car = self.cars[selected_car_number]
                        selected_car.start_driving.succeed(selected_call)

            if self.free_car.triggered:
```

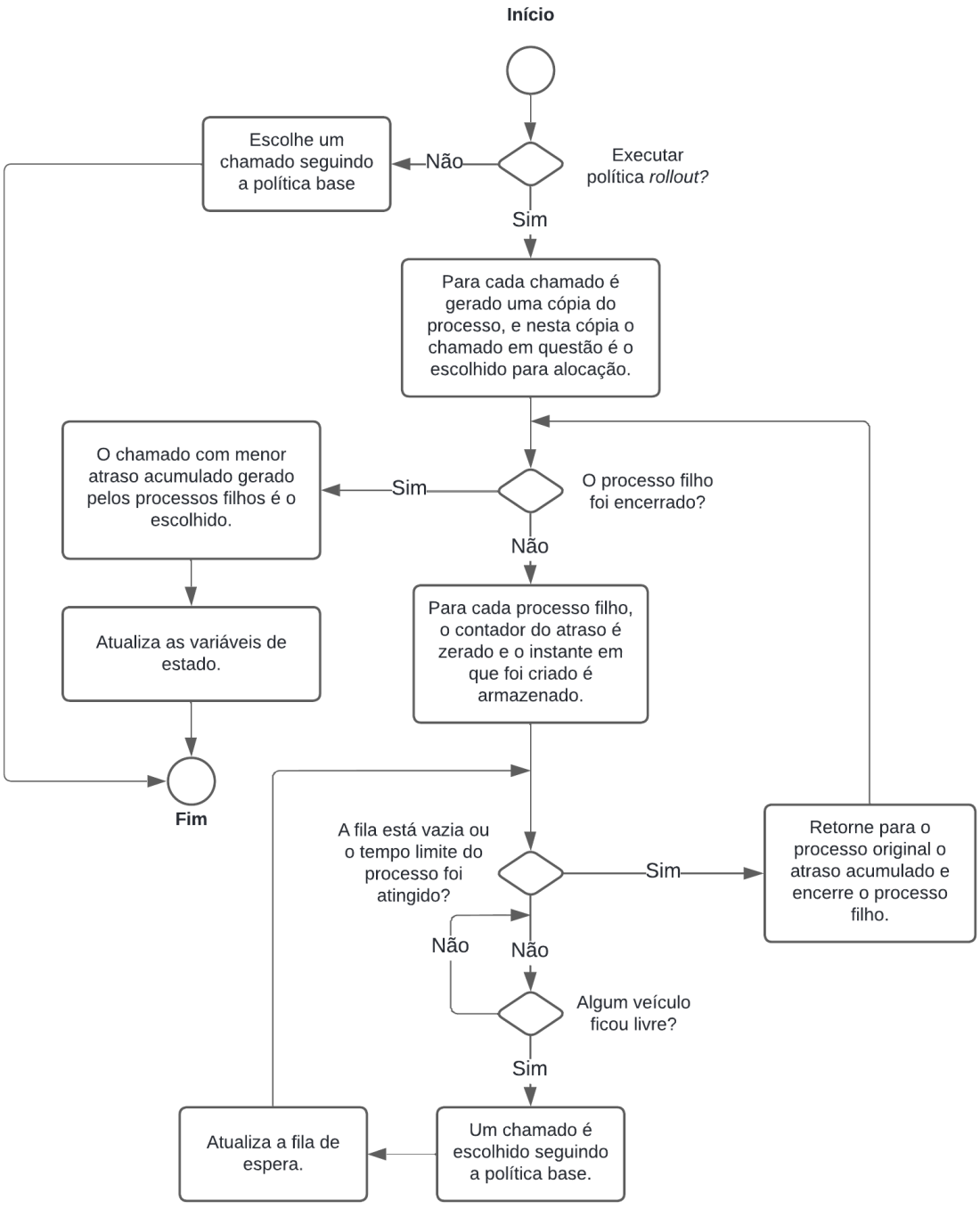
```
37                  car_number , atraso , duracao_origem = self.free_car.value
38                  self.atraso_total += atraso
39                  self.duracao_chegada_total += duracao_origem
40
41                  if len(self.lista_calls) > 0:
42                      selected_call = self.choose_call()
43                       self.cars[car_number].start_driving.succeed(
                             selected_call)
44
45                  self.free_car = self.env.event()
```

Agora, quando ocorre o evento do tipo II uma dinâmica de decisão diferente é executada pelo agente decisor (linha 36 até linha 45). Quando o veículo muda seu estado de ocupado para ocioso é analisado o tamanho da fila de chamados. Havendo chamados em espera, a função choose_call é acionada. É neste ponto que o algoritmo *rollout* é executado.

A figura 18 apresenta o fluxograma da execução do algoritmo *rollout* em uma única época de decisão. Quando o agente decisor segue uma política míope, ele simplesmente escolhe o chamado seguindo a política base. Agora, quando ele segue a política *rollout*, o fluxo inicia com o agente observando todos os possíveis estados subsequentes, ou seja, quais chamados podem ser alocados ao veículo ocioso. Para cada um destes chamados é gerada uma cópia do processo, em que cada processo filho parte do pressuposto que um dos chamados viáveis da fila de espera foi o escolhido para alocação. Nesta cópia do processo, enquanto houverem chamados na fila ou enquanto o tempo de simulação para este processo filho não for atingido sempre que um novo veículo estiver disponível, um dos chamados será escolhido a partir da política base. Durante a evolução do processo filho o atraso de alocação vai sendo armazenado. No momento em que a fila de chamados em espera esvaziar ou o tempo limite for atingido, a soma dos atrasos armazenados é retornada ao decisor no processo pai, que é o processo original e o processo filho é encerrado. Neste ponto, o decisor tem consigo o retorno de todas as trajetórias construídas a partir dos processos filhos, e pode escolher o chamado com menor atraso dentre todos os que foram avaliados.

No geral, esta é a estrutura do algoritmo *rollout* baseado em simulação de eventos discretos para o problema de alocação de veículos. No capítulo a seguir, será apresentado os resultados de alguns experimentos aplicando este algoritmo ao problema de alocação de veículos.

Figura 18 – Fluxograma representando uma etapa de decisão do algoritmo *rollout*.



Fonte: Elaborado pela autora (2022)

5 EXPERIMENTOS COMPUTACIONAIS E RESULTADOS

Neste capítulo serão apresentados detalhes dos experimentos que foram realizados utilizando o algoritmo *rollout* baseado em simulação de eventos discretos sobre instâncias sintéticas para o problema de alocação de veículos. O experimento objetiva comparar a performance das decisões orientadas pela política *rollout* com as que seguem a política base. A medida de desempenho utilizada é o atraso médio associado aos chamados. O atraso de um pedido corresponde ao intervalo de tempo entre seu nascimento até o momento em que o veículo chega ao local de origem onde está aguardando (Ver discussão na seção 4.1, em particular a equação 4.1).

Nos experimentos, o espaço de localizações é representado por uma malha quadricular (Figura 13). Os veículos podem estar localizados em qualquer vértice do grafo reticulado. Este grafo reticulado possui dimensão $n \times n$. Os chamados são caracterizados por um vértice de origem e um vértice de destino. O tempo de deslocamento entre quaisquer dois vértices é uma função da distância entre eles. A distância entre dois vértices é dada pelo comprimento do caminho mais curto entre os vértices, obtido a partida da norma L1, ou como é mais conhecida, distância Manhattan, dada pela soma das diferenças absolutas de suas coordenadas.

Nesta etapa de experimentação, foram testadas três políticas base:

- **FIFO (*first-in-first-out*):** Esta política orienta o decisor a escolher o chamado que está a mais tempo na fila de espera. Sua implementação é bem simples, removendo o primeiro chamado adicionado na lista de chamados (código-fonte 6);
- **NV (*nearest-vehicle*):** Esta política direciona o decisor a escolher o chamado que se encontra mais próximo do veículo disponível. A função que executa esta política captura a localização do veículo disponível e a localização de todos os chamados em espera para então capturar o chamado mais próximo (código-fonte 7);
- **RANDOM:** Quando o decisor segue esta política, o chamado é escolhido aleatoriamente. A função desta política mapeia todos os chamados em espera e realiza uma amostra uniforme de um destes chamados (código-fonte 8);

Código-fonte 6 – Função que modela a política base FIFO

```
1 def fifo_policy(self):
2         selected_call = self.lista_calls.pop(0)
3         return selected_call
```

Código-fonte 7 – Função que modela a política base NV.

```
def nv_policy(self):
        free_car = self.free_car.value[0]
        position_car = self.cars[free_car].location
        index_call = np.argmin([self.cars[free_car].time_matrix[
            position_car, call.origem] for call in self.lista_calls])
        selected_call=self.lista_calls[index_call]
        self.lista_calls.pop(index_call)

        return selected_call
```

Código-fonte 8 – Função que modela a política base RANDOM.

```
def random_policy(self):
        index_call = np.random.choice(range(len(self.lista_calls)))s
        elected_call=self.lista_calls[index_call]
        return selected_call
```

As instâncias utilizadas no experimento foram geradas artificialmente. Para gerar as instâncias foram adotadas os seguintes parâmetros:

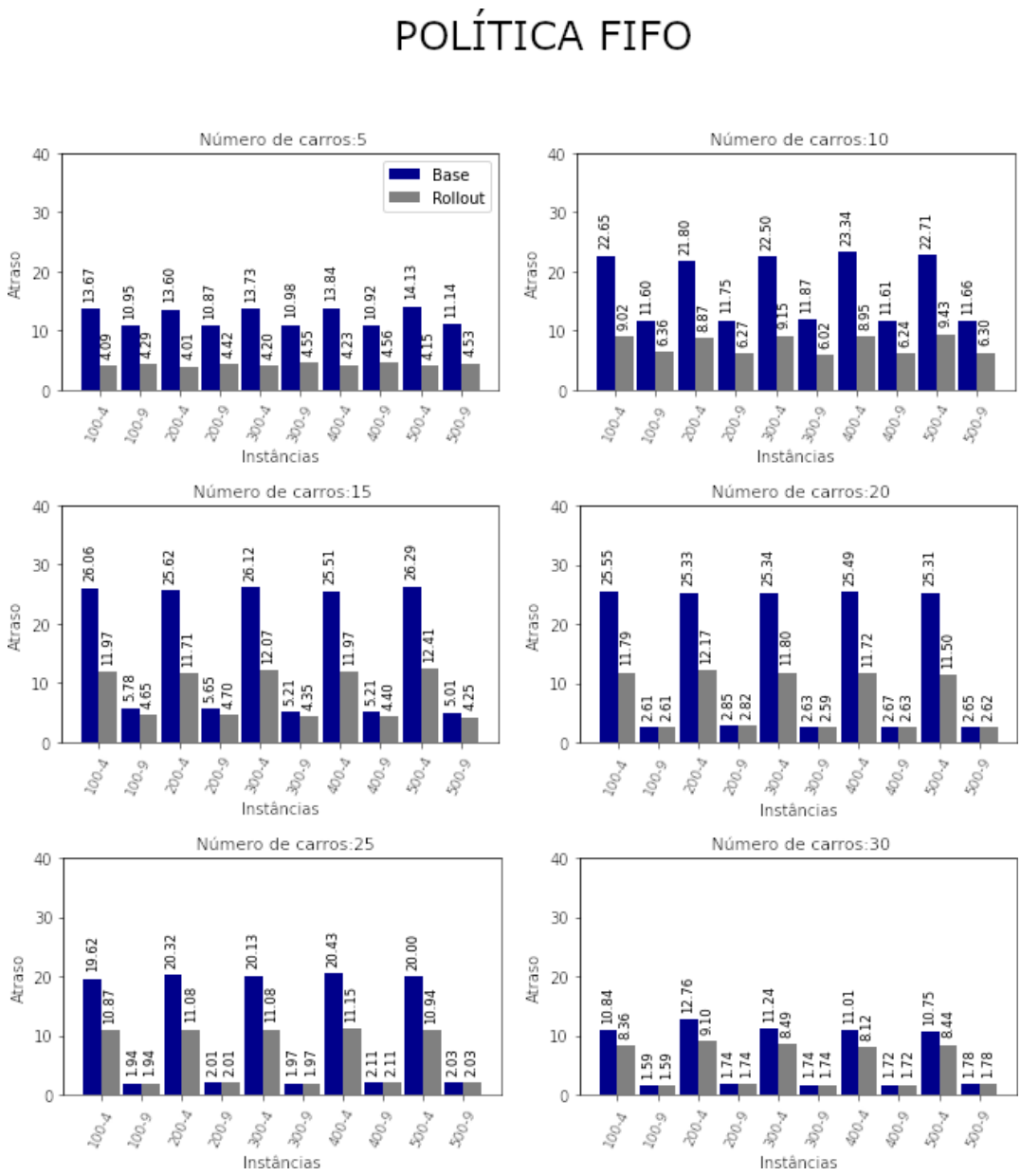
1. Número de veículos: 5, 10, 15, 20, 25 e 30.
2. Número de localizações: 100, 200, 300, 400 e 500.
3. Taxa média de chegada dos chamados: 4 e 9 por unidade de tempo.

Combinando estes parâmetros, foram obtidas 60 instâncias. Para cada instância, foram executadas 30 rodadas de simulação com um horizonte de 1000 unidades de tempo. Cada rodada de simulação consiste em executar duas etapas independentes: a primeira, executando as 3 políticas bases, a segunda, aplicando o algoritmo *rollout* para as mesmas políticas bases. Durante a execução do algoritmo *rollout*, as simulações de trajetórias futuras foram executadas até que a fila ficasse vazia ou até atingir um horizonte de 400 unidades de tempo.

Ao final da execução do algoritmo, foi obtido o atraso acumulado médio para cada rodada de simulação. A partir dessa métrica, as duas abordagens, política *rollout* e política base, foram comparadas, e os resultados a seguir têm por objetivo validar a propriedade de melhoria da política promovida pelo algoritmo *rollout*.

Os gráficos a seguir apresentam os resultados para cada uma das políticas bases

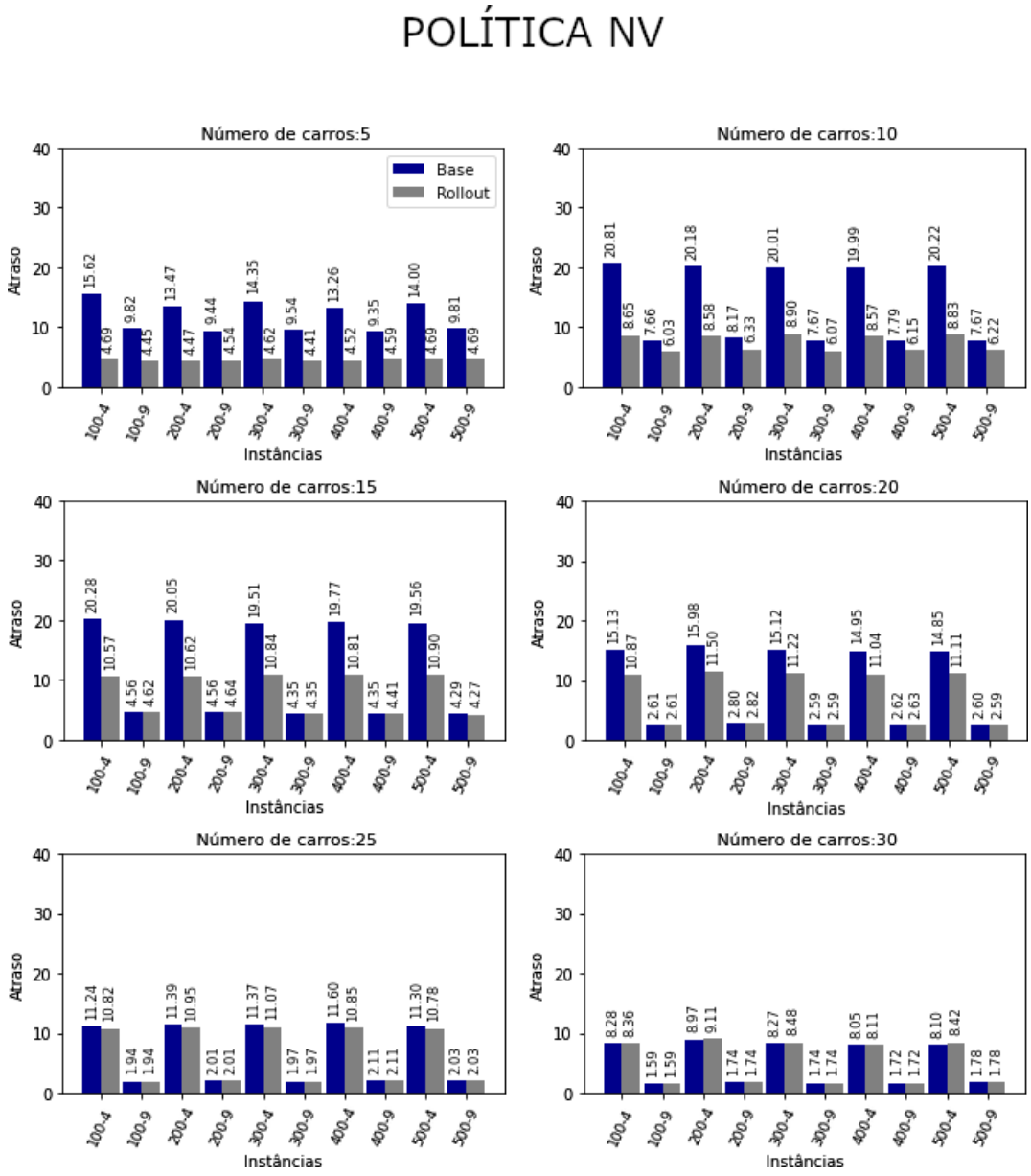
Figura 19 – Comparativo do atraso acumulado médio seguindo a política FIFO.



Fonte: Elaborado pela autora (2022).

experimentadas: FIFO (Figura 19), NV (Figura 20) e RANDOM (Figura 21). Perceba que para cada uma delas, no algoritmo *rollout* o decisor realizou a alocação dos veículos com atraso acumulado médio equivalente ou inferior ao atraso do decisor guiado somente pela política base. Uma característica importante da performance realizada pelo algoritmo *rollout* é que ele melhora potencialmente decisões no contexto de maior demanda, ou seja, nos cenários nos quais haviam menos veículos com uma taxa maior de chegada de chamados. No caso das instâncias em que o atraso acumulado médio das duas soluções foi equivalente, pode se observar que o ambiente

Figura 20 – Comparativo do atraso acumulado médio seguindo a política NV.

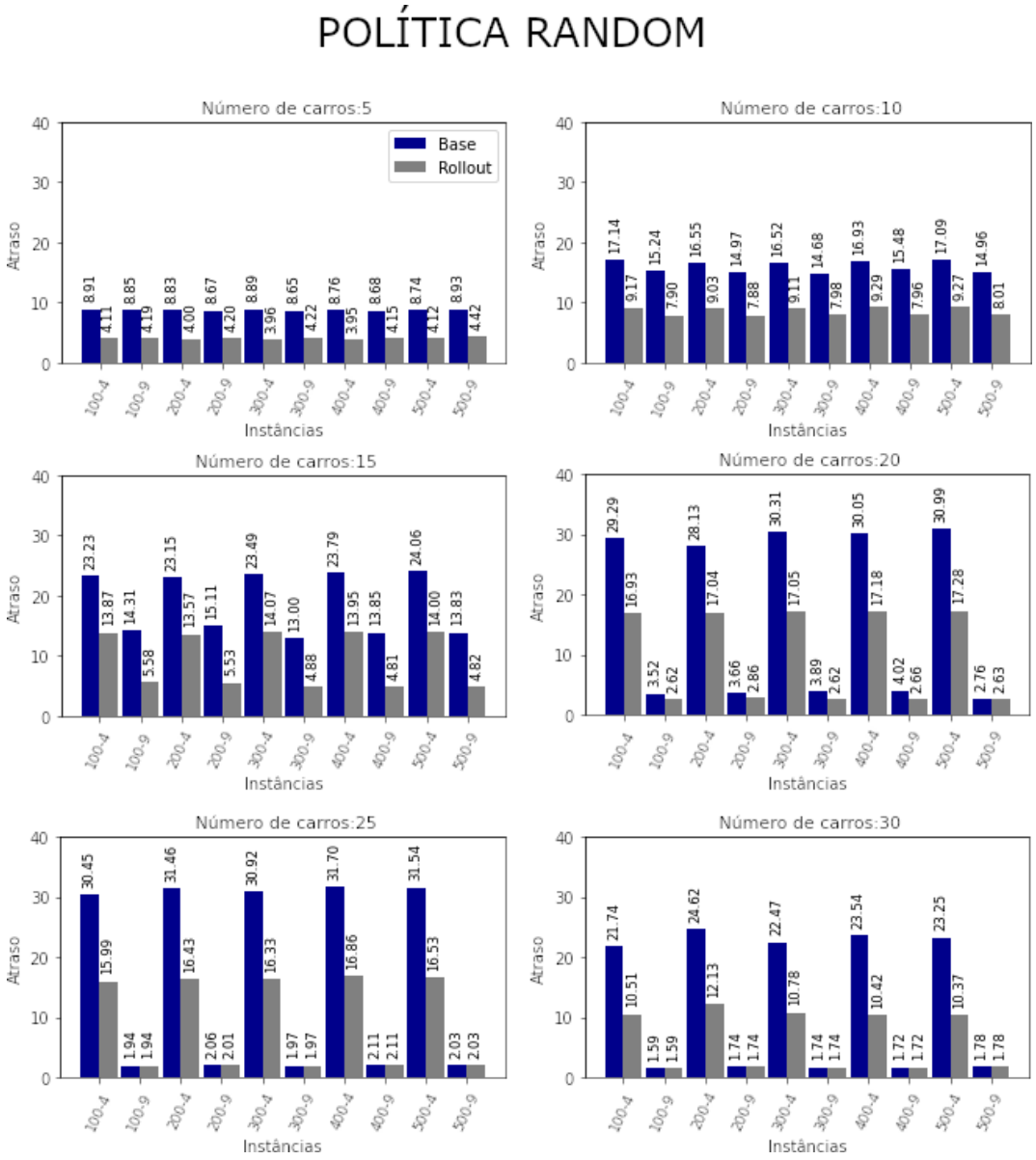


Fonte: Elaborado pela autora (2022).

oferecia uma oferta suficiente para atender imediatamente qualquer novo chamado que chegasse, sem que houvesse a necessidade que ele fosse direcionado à fila para aguardar. Assim, o atraso corresponde somente ao tempo em que o chamado aguarda o veículo se deslocar até o ponto de origem.

O ganho na performance da política de decisão implica em uma redução no atraso acumulado médio que chega até 55,55% para a política FIFO 69,99% para a política NV e 65,28% para a política RANDOM. No geral, a taxa de melhoria média é apresentada na Tabela 1.

Figura 21 – Comparativo do atraso acumulado médio seguindo a política RANDOM.



Fonte: Elaborado pela autora (2022)

Tabela 1 – Taxa de melhoria promovida pelo algoritmo *rollout.*

Política FIFO	Política NV	Política RANDOM
35,87%	22,65%	39,20%

A Tabela 2 exibe algumas informações descritivas a respeito da distribuição do atraso acumulado médio extraídas das amostras obtidas para cada instâncias em todas as rodadas de simulação. Entre as políticas FIFO, NV e RANDOM utilizadas no algoritmo *rollout* a

política RANDOM foi a que apresentou a maior média, bem como a maior amplitude e maior desvio padrão. Embora o objetivo do experimento não seja obter a melhor política aproximada, é importante reforçar que produzir boas políticas a partir do algoritmo *rollout* passa em um primeiro momento por escolher uma boa política base. Neste caso, as melhores políticas seriam a FIFO e a NV, sendo a FIFO uma política que minimiza o tempo de espera priorizando a ordem de chegada dos chamados, e a NV minimiza priorizando o menor deslocamento.

Tabela 2 – Atraso acumulado médio das políticas com e sem utilização do algoritmo *rollout.*

	FIFO		NV		RANDOM	
	Política Base	Política Rollout	Política Base	Política Rollout	Política Base	Política Rollout
Média	12,7984	6,5102	9,7765	6,3427	14.8708	7,9301
Desv. Padrão	8,6653	3,7032	6,3382	3,4112	9,9982	5,3723
Mín.	1,5889	1,5889	1,5889	1,5889	1,5889	1,5889
25%	4,4699	3,7101	3,9173	3,9043	7,4943	3,6759
50%	11,6083	5,3596	9,1610	5,3627	14,8181	6,7281
75%	20,7710	9,7886	14,8787	9,4740	23,3067	12,4908
Máx.	26,2889	12,4140	20,8098	11,5030	31,6982	17,2772

Fonte: Elaborado pela autora (2022)

O ganho em performance ao utilizar o algoritmo *rollout* atribui um custo computacional no tempo de construção da solução. Com os experimentos executados em um computador com Intel(R) Core(TM) i7-8565U de 1.80GHz e 16GB de memória RAM, o tempo médio de execução aplicando somente a política base foi de 7,8 segundos com desvio padrão de 7,67 segundo. Em contrapartida, para o algoritmo *rollout*, as execuções levaram em média 917,1 segundos com 1085,7 segundos de desvio padrão. Apesar desta diferença no tempo de execução das duas abordagens, este impacto pode ser reduzido produzindo política *offline*.

6 CONCLUSÕES

Nesta dissertação foi proposta uma modelagem para o problema de alocação de veículos como sendo um processo de decisão semimarkoviano e realizada a avaliação de políticas de decisão geradas por um algoritmo *rollout* baseado em simulação de eventos discretos. Tal abordagem permitiu representar o cenário de decisão do problema de alocação de veículos mais aproximado do contexto real, uma vantagem frente aos modelos que retratam o problema de decisão como um processo de decisão markoviano, em que os intervalos entre as decisões são discretos e homogêneos.

Os experimentos evidenciaram que o algoritmo *rollout* é potencialmente útil para melhorar a performance de uma dada política base inicial previamente conhecida. Assim, além da redução do atraso médio acumulado, o algoritmo *rollout* pode ser apresentado como um algoritmo de fácil implementação, e por ser baseado em simulação de eventos discretos permitiu o aprendizado de políticas sem a exigência do conhecimento explícito da distribuição de probabilidade associadas à transição entre estados, geração de retornos e instantes das épocas de decisão.

As principais contribuições deste trabalho foram:

1. Apresentação de um novo modelo para o problema de alocação de veículos como um PDSM. Ao contrário de um PDM, os PDSMs modelam a dinâmica de transição entre os estados em tempo contínuo. Modelar o problema de alocação de veículos como um PDSM reduziu a complexidade do espaço de decisões, uma vez que as decisões são tomadas individualmente para cada veículo ou para cada chamado, enquanto em um PDM as decisões quanto ao mapeamento de vários veículos a vários chamados devem ser feitas ao mesmo tempo.
2. Desenvolvimento de um novo simulador baseado em simulação de eventos discretos. A formulação de um PDSM levou naturalmente à interpretação do problema de alocação de veículos como um problema de controle de filas com dimensão espacial
3. Até onde sabemos, este é o primeiro trabalho a propor uma solução para o problema de alocação de veículos utilizando o algoritmo *rollout* com realização de experimentos comparando políticas *rollout* e políticas base.

Durante o período de realização da pesquisa para elaboração deste trabalho, foram apresentados dois artigos com contribuição na área de programação dinâmica aproximada, cujos títulos são apresentados a seguir:

1. Aplicação de Aprendizado por Reforço ao Problema de Corte de Estoque Estocástico, apresentado no LII Simpósio Brasileiro de Pesquisa Operacional, em setembro de 2020;
2. Formulação e Solução do Problema de Alocação de Veículos Estocástico por meio de Programação Dinâmica Aproximada, apresentado XV *Brazilian Congress on Computational Intelligence*, em agosto de 2021.

REFERÊNCIAS

ANDROULAKIS, I. P. Dynamic programming: infinite horizon problems, overview. *In*: PARDALOS, P. M. **Encyclopedia of Optimization**. Boston, MA: Springer US, 2009. p. 850–853.

BARRETO, A. M. S.; AUGUSTO, D. A.; BARBOSA, H. J. C. On the characteristics of sequential decision problems and their impact on evolutionary computation and reinforcement learning. *In*: ARTIFICAL Evolution. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 194–205.

BARTO, A. G.; SUTTON, R. S.; WATKINS, C. J. C. H. Learning and sequential decision making. **Learning and computational neuroscience**, Amherst, MA, p. 539–602,1989.

BAYKAL-GURSOY, M.; GURSOY, K. Semi-markov decision processes: nonstandard criteria. **Probability in the engineering and informational sciences**, New York, NY, v. 21, n. 4, p. 635–657, 2007.

BELLMAN, R. On the theory of dynamic programming. **Proceedings of the national academy of sciences**, Washington, DC, v. 38, n. 8, p. 716–719, ago. 1952.

BELLMAN, R. **Dynamic programming**. Garden City, NY: Dover Publications, 1957.

BERTSEKAS, D. **Reinforcement learning and optimal control**. Raleigh, NC: Athena Scientific, 2019.

BERTSEKAS, D. P. **Dynamic Programming and Optimal Control, Vol. II**. 3rd. ed. Raleigh, NC: Athena Scientific, 2007.

BERTSEKAS, D. P. Rollout algorithms for discrete optimization: a survey. **Handbook of Combinatorial Optimization**, New York, NY, p. 2989–3013, 2013.

BERTSEKAS, D. P. Constrained multiagent rollout and multidimensional assignment with the auction algorithm. Cambridge, MA, abs/2002.07407, 2020.

BERTSIMAS, D.; JAILLET, P.; MARTIN, S. Online vehicle routing: the edge of optimization in large-scale applications. **Operations Research**, Catonsville, MD, v. 67, n. 1, p. 143–162, 2019.

DAI, P.; GOLDSMITH, J. Topological value iteration algorithm for markov decision processes. **IJCAI International Joint Conference on Artificial Intelligence**, Lexington, KY, p. 1860–1865, 01 2007.

GALLAGER, R. G. **Stochastic processes**. New York: Cambridge University Press, 2013.

GERAMIFARD, A.; WALSH, T. J.; STEFANIE, T.; CHOWDHARY, G.; ROY, N.; HOW, J. P. **A tutorial on linear function approximators for dynamic programming and reinforcement learning**. Hanover, MA: Now Publishers, 2013.

GODFREY, G. A.; POWELL, W. B. An adaptive dynamic programming algorithm for dynamic fleet management, i: single period travel times. **Transportation Science**, v. 36, n. 1, p. 21–39, 2002.

GOODSON, J. C.; THOMAS, B. W.; OHLMANN, J. W. A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. **European Journal of Operational Research**, v. 258, n. 1, p. 216–229, 2017. ISSN 0377-2217.

GOSAVI, A. **Simulation-based optimization**. New York, NY: Springer US, 2015.

HOLLER, J.; VUORIO, R.; QIN, Z.; TANG, X.; JIAO, Y.; JIN, T.; SINGH, S.; WANG, C.; YE, J. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. *In*: IEEE. **2019 IEEE International Conference on Data Mining (ICDM)**. 2019. p. 1090–1095.

HOWARD, R. **Dynamic probabilistic systems**. New York, NY: Dover Publications, Inc, 2007. ISBN 978-0486458724.

HOWARD, R. A. **Dynamic programming and markov processes**. Cambridge, MA: MIT Press, 1960.

KUHN, H. W. The hungarian method for the assignment problem. **Naval Research Logistics Quarterly**, v. 2, n. 1–2, p. 83–97, 1955.

KULLMAN, N. D.; COUSINEAU, M.; GOODSON, J. C.; MENDOZA, J. E. Dynamic ride-hailing with electric vehicles. **Transportation Science,** 2021.

LAW, A. M. **Simulation modeling and analysis**. 5. ed. New York, NY: McGraw-Hill Education, 2014. (Mcgraw-hill Series in Industrial Engineering and Management). ISBN 0073401323; 9780073401324.

LEE, D.-H.; WANG, H.; CHEU, R. L.; TEO, S. H. Taxi dispatch system based on current demands and real-time traffic conditions. **Transportation Research Record**, SAGE Publications Sage CA: Los Angeles, CA, v. 1882, n. 1, p. 193–200, 2004.

LEW, A.; MAUCH, H. **Dynamic programmingl**: a computational tool. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 3540370137.

LIANG, E.; WEN, K.; LAM, W. H.; SUMALEE, A.; ZHONG, R. An integrated reinforcement learning and centralized programming approach for online taxi dispatching. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, 2021.

LIAO, Z. Taxi dispatching via global positioning systems. **IEEE Transactions on Engineering Management**, United States, v. 48, n. 3, p. 342–347, 2001.

LIU, Z.; LI, J.; WU, K. Context-aware taxi dispatching at city-scale using deep reinforcement learning. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, 2020.

MIAO, F.; HAN, S.; LIN, S.; STANKOVIC, J. A.; ZHANG, D.; MUNIR, S.; HUANG, H.; HE, T.; PAPPAS, G. J. Taxi dispatch with real-time sensing data in metropolitan areas: a receding horizon control approach. **IEEE Transactions on Automation Science and Engineering**, IEEE, v. 13, n. 2, p. 463–478, 2016.

ÖZKAN, E.; WARD, A. R. Dynamic matching for real-time ride sharing. **Stochastic Systems**, Institute for Operations Research and the Management Sciences (INFORMS), v. 10, n. 1, p. 29–70, mar. 2020.

PENTICO, D. W. Assignment problems: A golden anniversary survey. **European Journal of Operational Research**, v. 176, n. 2, p. 774–793, jan. 2007.

POWELL, W. B. A stochastic formulation of the dynamic assignment problem with an application to truckload motor carriers. **Transportation Science**, Institute for Operations Research and the Management Sciences (INFORMS), v. 30, n. 3, p. 195–219, ago. 1996.

POWELL, W. B. **Approximate dynamic programming for asset management**: solving the curses of dimensionality. 1. ed. Princeton, NJ, 2003.

POWELL, W. B. **Approximate dynamic programming**: solving the curses of dimensionality. 2nd. ed. Hoboken, NJ: Wiley, 2011. (Wiley Series in Probability and Statistics).

POWELL, W. B. **Reinforcement learning and stochastic optimization**: a unified framework for sequential decisions. Hoboken, NJ: Wiley-Blackwell, 2022.

PUTERMAN, M. L. **Markov decision processes**: discrete stochastic dynamic programming. New York, NY: John Wiley Sons Inc., 2005.

QIN, Z.; TANG, X.; JIAO, Y.; ZHANG, F.; XU, Z.; ZHU, H.; YE, J. Ride-hailing order dispatching at DiDi via reinforcement learning. **INFORMS Journal on Applied Analytics**, INFORMS, v. 50, n. 5, p. 272–286, 2020.

SEOW, K. T.; DANG, N. H.; LEE, D.-H. A collaborative multiagent taxi-dispatch system. **IEEE Transactions on Automation science and engineering**, IEEE, v. 7, n. 3, p. 607–616, 2009.

SHORTLE, J. F.; THOMPSON, J. M.; GROSS, D.; HARRIS, C. M. **Fundamentals of queueing theory**. 5. ed. Nashville, TN: John Wiley & Sons, 2018. (Wiley Series in Probability and Statistics).

SIMãO, H. P.; DAY, J.; GEORGE, A. P.; GIFFORD, T.; NIENOW, J.; POWELL, W. B. An approximate dynamic programming algorithm for large-scale fleet management: a case application. **Transportation Science**, v. 43, n. 2, p. 178–197, 2009.

SNIEDOVICH, M. **Dynamic programming**. 2. ed. Boca Ranton, FL: CRC Press, 2010. ISBN 0824740998.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning**: an introduction. Second. Cambridge, MA: The MIT Press, 2018.

SUTTON, R. S.; PRECUP, D.; SINGH, S. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. **Artificial Intelligence**, El Segundo, CA, v. 112, n. 1, p. 181–211, 1999. ISSN 0004-3702.

SZEPESVÁRI, C. Algorithms for reinforcement learning. **Synthesis Lectures on Artificial Intelligence and Machine Learning**, Morgan & Claypool Publishers LLC, San Rafael, CA, v. 4, n. 1, p. 1–103, jan. 2010.

TANG, X.; QIN, Z.; ZHANG, F.; WANG, Z.; XU, Z.; MA, Y.; ZHU, H.; YE, J. A deep value-network based approach for multi-driver order dispatching. *In*: PROCEEDINGS of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019. p. 1780–1790.

TANG, X.; ZHANG, F.; QIN, Z.; WANG, Y.; SHI, D.; SONG, B.; TONG, Y.; ZHU, H.; YE, J. Value function is all you need: A unified learning framework for ride hailing platforms. *In:* PROCEEDINGS of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; DataMining. : ACM, 2021. (KDD ’21).

WANG, L.; PENG, H.; ZHU, H.-y.; SHEN, L.-c. A survey of approximate dynamic programming. **2009 International Conference on Intelligent Human-Machine Systems and Cybernetics**, Hangzhou, China, v. 241, p. 396–399, 08 2009.

WANG, Z.; QIN, Z.; TANG, X.; YE, J.; ZHU, H. Deep reinforcement learning with knowledge transfer for online rides order dispatching. *In*: IEEE. **2018 IEEE International Conference on Data Mining (ICDM)**. 2018. p. 617–626.

XU, Z.; LI, Z.; GUAN, Q.; ZHANG, D.; LI, Q.; NAN, J.; LIU, C.; BIAN, W.; YE, J. Large-scale order dispatch in on-demand ride-hailing platforms: a learning and planning approach. *In*: PROCEEDINGS of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018. p. 905–913.

YAN, C.; ZHU, H.; KOROLKO, N.; WOODARD, D. Dynamic pricing and matching in ride-hailing platforms. **Naval Research Logistics (NRL)**, v. 67, n. 8, p.705–724, 2020.

ZHANG, L.; HU, T.; MIN, Y.; WU, G.; ZHANG, J.; FENG, P.; GONG, P.; YE, J. A taxi order dispatch model based on combinatorial optimization. *In*: PROCEEDINGS of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. 2017.p. 2151–2159.