Formulação e Solução do Problema de Alocação de Veículos Estocástico por meio de Programação Dinâmica Aproximada

Vitória I. T. Mendonça
Dep. de Estatística e Matemática Aplicada
Universidade Federal do Ceará
vitoria.mendonca@alu.ufc.br

Rodrigo F. Meneses
Dep. de Estatística e Matemática Aplicada
Universidade Federal do Ceará
rodrigo.meneses@alu.ufc.br

Anselmo R. Pitombeira-Neto
Dep. de Engenharia de Produção
Universidade Federal do Ceará
anselmo.pitombeira@ufc.br

***Resumo*—Neste artigo, formulamos o problema de alocação de veículos estocástico como um processo de decisão semimarkoviano. Devido ao tamanho do espaço de estados, métodos exatos para a solução do problema são computacionalmente intratáveis. Aplicamos então um algoritmo *rollout*, o qual é um método de programação dinâmica aproximada baseado em iteração de política que tem mostrado resultados promissores em diversos contextos de aplicação. Faremos uso do algoritmo *rollout* a fim de produzir uma política de decisão *online* para o problema de alocação de veículos a chamados que surgem estocasticamente em tempo contínuo. Desenvolvemos um ambiente de decisão baseado em simulação de eventos discretos e realizamos experimentos computacionais com uso de três heurísticas-bases. O resultados indicaram que o algoritmo *rollout* foi capaz de produzir políticas com desempenho pelo menos tão bom quanto e em muitos casos consideravelmente melhor que as heurísticas-base.**

***Palavras- chave* —Alocação de veículos; Programação dinâmica aproximada; Processos de decisão semimarkoviano; Algoritmo *rollout*; Otimização baseada em simulação.**

I. Introdução

O problema de alocação de veículos é um problema da classe de otimização inteira que ocorre em muitas aplicações reais, dentre as quais se destacam: a decisão de alocação de veículos a chamados em aplicativos de viagens; a alocação de entregadores a pedidos em aplicativos de entregas de alimentos; e a alocação de ambulâncias a ocorrências emergenciais. De modo geral, o problema de alocação de veículos pode ser definido por: dado um conjunto de veículos e um conjunto de tarefas (ou pedidos) distribuídos espacialmente, deseja-se alocar os veículos disponíveis às tarefas ao longo do tempo de modo a minimizar o custo médio associado às decisões de alocação.

Neste artigo, trabalhamos em um cenário em que temos um conjunto de veículos idênticos, com localização conhecida e disponibilidade variando ao longo do tempo. Estes veículos precisam ser alocados de modo a atender a demanda de clientes, que solicitam corridas saindo de um ponto de origem até algum destino. Estas corridas tornam-se conhecidas em instantes aleatórios de tempo. A decisão quanto a qual cliente alocar um determinado veículo fica sob a responsabilidade de um agente decisor, que é acionado sempre que um veículo, antes ocupado, torna-se disponível ou quando, havendo veículos disponíveis e nenhuma chamada em espera, uma nova chamada chega ao sistema (veja Fig. 1). O objetivo do agente decisor é alocar veículos de forma a minimizar o tempo médio de espera dos clientes, medido desde o momento em que o cliente inicia o chamado até o momento em que o veículo chega ao ponto de origem da corrida.

Formulamos este problema como um problema de decisão sequencial, estruturado sob a perspectiva da programação dinâmica estocástica. Dada a característica de que as decisões ocorrem em intervalos aleatórios de tempo, o problema é modelado como um processo de decisão semimarkoviano, garantindo assim todas as propriedades de um processo de decisão markoviano, diferenciando-se no fato de que os intervalos entre as épocas de decisão são variáveis aleatórias contínuas com uma distribuição de probabilidades arbitrária, de forma que o tempo também deve ser tratado como contínuo, e não discreto como frequentemente se assume nos processos de decisão markovianos.

Uma solução exata para a formulação do problema, na forma de uma política ótima de decisão, pode ser obtida por meio de algoritmos clássicos de programação dinâmica, como a iteração de valor e a iteração de política. No entanto, esses algoritmos raramente são aplicados diretamente em problemas reais, pois requerem a enumeração do espaço de estados, e portanto sofrem da chamada *maldição da dimensionalidade*, a qual se refere à explosão combinatória do número de estados com o tamanho do problema.

Para contornar as limitações dos algoritmos exatos, propomos então uma solução heurística por meio de um algoritmo *rollout* para a obtenção de políticas de decisão. Algoritmos de *rollout* fazem parte da classe de algoritmos de iteração de política aproximada, os quais fazem uso de heurísticas-base para construir políticas de decisão *online*. Em cada época de decisão, o agente decisor simula, para cada decisão possível, uma trajetória de estados e decisões futuros durante um horizonte de tempo usando uma política heurística, também chamada de heurística-base. O agente decisor escolhe então aquela decisão que tem o menor custo médio estimado.

Na aplicação apresentada neste trabalho, utilizamos três

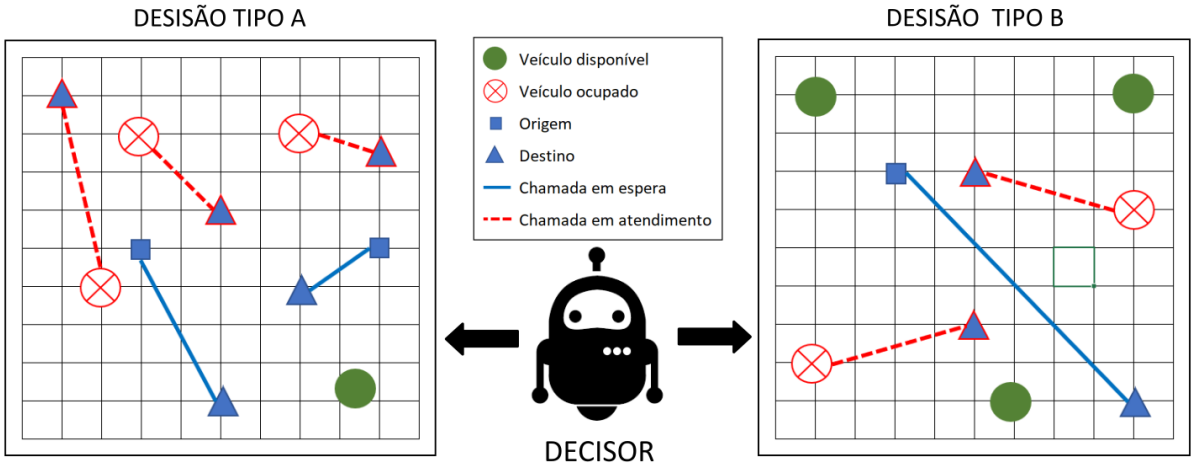

Fig. 1. A imagem acima representa os dois eventos nos quais o agente decisor é acionado para alocar um veículo a um chamado. A decisão tipo A representa o evento em que um veículo antes ocupado torna-se disponível, e o decisor escolhe uma das chamadas em espera para a alocação. Em contrapartida, a decisão tipo B representa o caso onde há mais de um veículo livre e uma nova chamada chega ao sistema.

heurísticas bases:FIFO, RANDOM e NN. Para a simulação, usamos um simulador de eventos discretos implementado em linguagem Python. Os resultados obtidos via experimentação computacional indicam que o algoritmo de *rollout* pode reduzir consideravelmente o atraso médio no atendimento dos chamados dos clientes quando comparado ao uso das heurísticas-base sem *rollout*.

A. Trabalhos Relacionados

O problema de alocação em sua versão clássica foi originalmente proposto na literatura contemplando o cenário em que desejava-se mapear dois conjuntos, denominados de recursos e tarefas. O primeiro método apresentado foi denominado de método húngaro [1], [2]. Este método formulava o problema como um modelo de programação linear inteira, cujo objetivo era minimizar o custo de cada atribuição sujeito às restrições de que cada recurso servisse a uma única tarefa e cada tarefa fosse atendida por um único recurso.

Muitas variantes deste problema foram estudadas ao longo dos anos, de tal forma que o problema de alocação é um problema clássico conhecido por toda a comunidade de otimização e pesquisa operacional. Uma extensão do problema de alocação é o problema de alocação dinâmico estocástico [3]. Sob a perspectiva dinâmica e estocástica, os recursos devem ser alocados ao longo de um horizonte de tempo sequencialmente sofrendo a influência de algum elemento aleatório interno ou externo ao ambiente de decisão.

Temos que o problema de alocação dinâmico estocástico pode ser compreendido como um problema de controle ótimo. O termo controle ótimo surgiu pra denominar o problema de designar um controlador ou decisor para otimizar decisões sobre um sistema dinâmico. Em meados na década de 50, Richard Bellman desenvolveu uma abordagem, intitulada de programação dinâmica, para lidar com problemas de controle ótimo, usando conceitos como estado do sistema, decisão, recompensa e função de valor [4]. Logo em seguida, foi introduzido o conceito de processo de decisão markoviano para lidar com problemas de controle ótimo estocástico em tempo discreto [5].

Desde então, a programação dinâmica é vista como uma das estratégias mais gerais para lidar com o problema de controle ótimo. Algumas aplicações da programação dinâmica no problema de alocação de veículos são: problema de alocação de caminhões que transportam motores [6], [7], realocação de ambulâncias para atender chamado de emergência [8], [9], e despacho de veículos em viagens sob demanda [10], [11].

Dado que a dimensionalidade do espaço de estados e espaço de decisões são limitantes para as principais técnicas de computação exata da função de valor em programação dinâmica, temos como uma via alternativa a programação dinâmica aproximada. O algoritmo *rollout* é uma possível abordagem da classe de aproximação de política [12], e tem sido utilizado em estudos com aplicações no problema de agendamento de tarefas [13], problema da mochila sequencial [14], e problema de roteamento de veículos [15].

II. Fundamentação Teórica

A. Processo de Decisão Semimarkoviano

Processos de decisão semimarkovianos (PDSM) são generalizações dos processos de decisão markovianos (PDM), em que os tempos entre épocas de decisão não são iguais, podendo ocorrer deterministicamente ou estocasticamente seguindo uma distribuição de probabilidades qualquer [16].

Em um PDSM, a decisão escolhida determina a distribuição de probabilidades conjunta do estado subsequente e o tempo entre as épocas de decisão. Uma importante característica do PDSM está no fato de que o sistema pode evoluir, mudando de estado diversas vezes entre épocas de decisão consecutivas. Desta forma, apenas os estados em que ocorre uma época de decisão oferecem informações relevantes para o decisor.

Assim, é conveniente distinguir o processo estocástico de transição dos estados que ocorre continuamente no tempo e o processo que ocorre somente a cada época de decisão. Nomeamos estes dois processos, respectivamente, de processo natural e processo semimarkoviano. O processo natural modela a evolução do estado do sistema como se estivéssemos observando continuamente ao longo do tempo, enquanto o processo semimarkoviano representa a evolução do estado do sistema apenas em épocas de decisão [5].

Seja $\{S(t)\}_{t\in[0,T]}$ um processo estocástico, em que $S(t)$ é uma variável aleatória que assume valores em $\mathcal{S}$, o qual é um conjunto finito ou enumerável que representa o espaço de estados, e $0 < T \leq \infty$ é o horizonte de tempo. Considerando que as épocas de decisão ocorrem em pontos discretos e aleatórios do tempo, definimos t_k, $k \in \{0, 1, 2, \cdots\}$, como a *k*-ésima época de decisão. Se em alguma época de decisão t_k o sistema ocupar um dado estado $s_k = S(t_k)$, o agente decisor deverá escolher uma ação $a_k \in \mathcal{A}_{s_k}$, em que $\mathcal{A}_{s_k}$ é o conjunto de decisões viáveis para o estado s_k. Como consequência da escolha da ação a_k, o sistema ocupará um próximo estado s_{k+1} na próxima época de decisão no tempo t_{k+1} com probabilidade definida pela função:

$$Q(j, \tau|i, a) = \mathbb{P}[s_{k+1} = j, t_{k+1} - t_k \leq \tau | s_k = i, a_k = a].$$

Note que a probabilidade do sistema ocupar um estado j na próxima época de decisão é dada por

$$P(j|i, a) = \mathbb{P}[s_{k+1} = j | s_k = i, a_k = a],$$

a qual pode ser obtida também por:

$$P(j|i, a) = \lim_{\tau\to\infty} Q(j, \tau|i, a). \tag{1}$$

Adicionalmente, se definirmos

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

como a probabilidade que a próxima época de decisão ocorra em até τ, dado o estado atual igual a i, o próximo estado igual a j, e dado que o decisor escolheu a ação a, podemos obter $Q(j, \tau|i, a)$ via [21]

$$Q(j, \tau|i, a) = p(\tau|i, a, j)P(j|i, a).$$

Esta dinâmica de transição entre as épocas de decisão é denominada de cadeia de Markov embutida ou processo de decisão markoviano embutido [5], [17].

Associado a cada decisão aplicada sobre um estado existe um custo que é retornado ao decisor para mensurar a qualidade da decisão. Dado que em um estado $s \in \mathcal{S}$ seja escolhido aplicar a decisão $a \in \mathcal{A}_s$, temos um custo fixo global $C(s, a)$ e uma taxa de custo adicional $c(s, a, j)$ que ocorre enquanto o processo natural permanece em um estado de decisão j. Assim, o custo total entre épocas de decisão subsequentes t_k e t_{k+1}, no qual o estado do sistema em t_k é s e a decisão aplicada é a, é dado por:

$$r(s, a) = C(s, a) + \mathbb{E}\left[\int_{t_k}^{t_{k+1}} c(s, a, S(t))dt\right].$$

Seja π um política de decisão estacionária e determinística definida como uma função $\pi : \mathcal{S} \to \mathcal{A}$. Denotamos por

$$g_\pi(s) = \lim_{N\to\infty} \frac{1}{\mathbb{E}\left[\sum_{k=1}^{N} \tau_k | s, \pi\right]} \mathbb{E}\left[\sum_{k=0}^{N-1} \Big(C(s_k, \pi(s_k)) + \int_{t_k}^{t_{k+1}} c(s_k, \pi(s_k), S(t))dt\Big) \;\Big|\; s, \pi\right], \tag{2}$$

o custo médio esperado gerado pela política π dado que o sistema ocupa o estado s em $t = 0$, onde $\tau_k = t_k - t_{k+1}$.

Resolver um PDSM corresponde a encontrar um política ótima π^* definida como

$$\pi^* \in \arg\min_{\pi\in\Pi} g_\pi(s), \quad \forall s \in \mathcal{S},$$

em que Π é uma classe de políticas. Se assumirmos que a cadeia de markov é do tipo *unicadeia* (uma cadeia com apenas uma classe recorrente e possivelmente alguns estados transientes), o custo médio esperado ótimo é definido como

$$g^* = g^{\pi^*} = \min_{\pi\in\Pi} g_\pi(s), \quad \forall s \in \mathcal{S},$$

e independe do estado inicial [5]. Semelhantemente ao que ocorre em um PDM, em um PDSM podemos obter uma política ótima resolvendo a equação de Bellman, que é formulada sobre a cadeia de Markov embutida de um PDSM com probabilidade de transição dada por (1). Assim, a equação de Bellman de um PDSM é dada pela função de valor na seguinte forma

$$h^*(s) = \max_{a\in\mathcal{A}_s} \left\{ r(s, a) - g^*\bar{\tau}(s, a) + \sum_{j\in\mathcal{S}} P(j|s, a)h^*(j) \right\} \quad \forall s \in \mathcal{S} \tag{3}$$

em que $r(s, a)$ é o custo esperado por transição, $\bar{\tau}(s, a)$ é o tempo médio entre épocas de decisão, e $g*$ é o custo médio mínimo.

Obter um política ótima via otimização da equação de Bellman é computacionalmente intratável em boa parte dos casos práticos. Em particular, na nossa aplicação lidamos com um extenso espaço de estados e espaço de decisões. Adicionalmente a isto, desconhecemos a lei de probabilidade que rege o intervalo entre as épocas de decisão. Então, utilizar os algoritmos clássicos de iteração de política e iteração de valor não é viável no contexto deste trabalho. Uma alternativa para contornar esta limitação é utilizar métodos de programação dinâmica aproximada. Propomos a utilização de um algoritmo de iteração de política aproximada, denominado de algoritmo *rollout*.

B. Algoritmo Rollout

Rollout é uma solução heurística de otimização sequencial *online* para resolver problemas de programação dinâmica, cujo procedimento é escolher decisões em tempo real apenas sobre os estados visitados, via uma política de decisão *lookahead* [18].

A técnica *rollout* pode ser categorizada como sendo da classe de algoritmos de iteração de política aproximada, nos quais a função de valor é aproximada por uma função que é facilmente computada e que não requer armazenamento. Assim, o algoritmo *rollout* é interpretado como uma única iteração do método de iteração de política. O processo iterativo do método inicia a partir de uma política dada previamente, em seguida esta política é avaliada, e consequentemente, baseado nesta avaliação é obtida um política melhorada [19].

É importante compreender que o objetivo do *rollout* não é obter uma política ótima ou obter uma estimativa completa da função de valor [20]. O principal objetivo dos algoritmos *rollout* é obter uma política melhorada, ou seja, partindo uma política subótima definida *a priori* para o horizonte do problema, chamada de política-base, produzir uma política melhorada, chamada de política *rollout* [21]. Seja o espaço

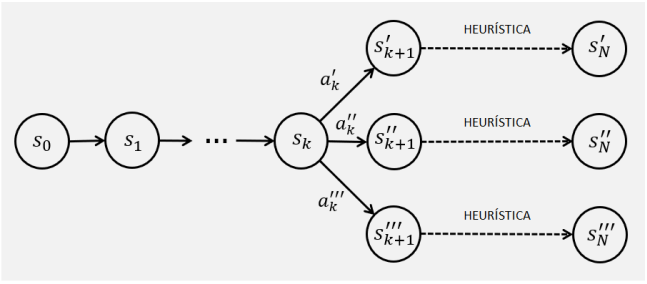

Fig. 2. Ilustração da estrutura lógica de um algoritmo *rollout*.

de estados $\mathcal{S}$ um conjunto finito ou enumerável e $s_k \in \mathcal{S}$ o estado do sistema na época de decisão k. Considere $\mathcal{S}_{s_k}^{a_k} = \{s_{k+1} | P(s_{k+1}|s_k, a_k) > 0\}$ como sendo o conjunto de estados alcançáveis quando o processo ocupa o estado s_k e a decisão $a \in \mathcal{A}_{s_k}$ 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 de uma heurística-base $\mathcal{H}$ (c.f. Fig. 2). A partir da heurística base $\mathcal{H}$ é computada uma sequência de estados e decisões que formam a trajetória $h = \{s_{k+1}, a_{k+1}, \cdots, a_{N-1}, s_N\}$, em que $s_{k+1} \sim P(s_{k+1}|s_k, a_k)$ é gerado por simulação durante a execução da heurística. O algoritmo *rollout* então escolhe a decisão $\hat{\pi}_k(s_k)$ no estado s_k que otimiza a soma do custo esperado imediato e a estimativa da função de valor futuro:

$$\hat{\pi}_k(s_k) \in \arg \min_{a_k \in \mathcal{A}_{s_k}} \{C(s_k, a_k) + H_{k+1}(s_{k+1})\}, \quad (4)$$

em que $H_{k+1}(s_{k+1})$ é uma estimativa da função de valor futuro associada às épocas de decisão $k+1$ até N e obtida por meio da heurística-base. Este processo define um política subótima $\hat{\pi}_k(s_k)$ denominada de política *rollout* [21].

III. Formulação do problema como um PDSM

Considere n veículos $v_1, v_2, \cdots, v_n \in \mathcal{V}$, em que $\mathcal{V}$ é o conjunto de todos os veículos disponíveis para alocação. Cada veículo é representado por uma tupla $v_i = (l_i, b_i)$, com $i \in \{1, \cdots, n\}$, em que $l_i \in \mathcal{L}$ é a localização do veículo e $b_i \in \{0, 1\}$ é estado do veículo, que indica se o veículo está ocupado ($b_i = 1$) ou livre ($b_i = 0$).

Para cada instante de tempo t, definimos o conjuntos de chamadas solicitadas em espera $\mathcal{R}(t)$, onde cada chamada solicitada por um cliente é representado pela tupla $r_j = (o_j, d_j, w_j)$, $j \in \{1, 2, \cdots\}$ em que o_j é a localização da origem do chamado, d_j é a localização do destino e w_j é o instante no qual o chamada chegou ao sistema. Note que para qualquer t, podemos ter $\mathcal{R}(t) = \varnothing$. Ambos os valores assumidos por o_j e d_j pertencem ao conjunto finito de localizações $\mathcal{L}$, e o tempo de chegada da chamada no sistema w_j assume valores em $[0, +\infty)$.

Definimos o estado do sistema $S(t)$ em um dado instante t como a tupla dada pelo conjunto de veículos e o conjunto de chamadas solicitadas, isto é, $S(t) = (\mathcal{V}(t), \mathcal{R}(t))$, onde $t \in [0, T]$, sendo T o limite do horizonte de decisão e $\mathcal{S}$ o conjunto de todos os possíveis estados do sistema.

A transição de estados neste modelo ocorre a partir da chegada de novas chamadas ao sistema e da mudança na disponibilidade dos veículos. Portanto, logo que um veículo fica livre, se $\mathcal{R}(t) \neq \varnothing$, o decisor atribui uma chamada ao veículo. Em seguida, o veículo se desloca, partindo de sua atual localização até o local de origem do chamado e, logo depois, para o destino do chamado, após o qual termina a execução da tarefa.

As épocas de decisões ocorrem em instantes discretos de tempo t_k, $k \in \{0, 1, 2, \cdots\}$, embora o sistema evolua continuamente no tempo. O decisor é acionado para escolher uma ação sempre que um destes dois eventos ocorre:

1) Todos os veículos estão ocupados e um veículo termina a corrida, tornando-se livre, e existem chamadas em espera em $\mathcal{R}(t)$. Neste caso, o decisor, escolhe qual das chamadas atribuirá ao veículo (c.f. Fig. 1 A);
2) existem mais de um veículo livre e uma nova chamada chega ao sistema quando não há nenhuma outra chamada à espera. (c.f. Fig. 1 B)

Seja s_k o estado em uma dada época de decisão no tempo t_k e $a_k \in \mathcal{A}_{s_k}$ a escolha do decisor, que corresponde a qual chamado será alocado ao veículo no caso do evento 1 descrito acima, ou a qual veículo será alocado a chamada no caso do evento 2. Aplicada a decisão sobre o estado, o sistema retorna ao decisor o custo deste par estado-decisão, que corresponde ao tempo de espera da chamada na fila até que seja alocada a um veículo somado ao tempo de deslocamento do veículo até a origem da chamada. Assim, o custo da decisão a_k no estado s_k é dado por:

$$r(s_k, a_k) = C(s_k, a_k) + \mathbb{E}\left[\int_{t_k}^{t_{k+1}} \delta(s_k, a_k, S(t))\, dt\right], \quad (5)$$

em que $C(s_k, a_k)$ é o tempo de espera do chamado na fila, $\delta(s_k, a_k, S(t))$ é taxa de custo adicional, definida por $\delta(s_k, a_k, S(t)) = 1$ enquanto o processo evolui no intervalo $[t_k, t_k + \Delta(s_k, a_k)]$, e $\delta(s_k, a_k, S(t)) = 0$ enquanto o processo evolui no intervalo $[t_k + \Delta(s_k, a_k), t_{k+1}]$, em que $\Delta(\cdot, \cdot)$ é uma variável aleatória que retorna o tempo de deslocamento

do veículo de sua localização atual até a localização de origem da chamada. Note que o segundo termo do lado direito em (5) representa o tempo esperado da viagem do veículo de sua localização atual até a localização de origem da chamada.

IV. Implementação Computacional via Simulação de Eventos Discretos

O modelo proposto para o problema de alocação de veículos detém uma característica importante que nos possibilita simular o ambiente de decisão utilizando técnicas de simulação de eventos discretos. Note que as chamadas que chegam ao sistema permanecem em espera até que o decisor escolha uma destas para atribuir ao veículo. O mesmo ocorre com os veículos, que permanecem em espera até que um chamado lhes seja atribuído. Assim, temos que no ambiente de decisão deste problema ocorre a formação de um sistema de filas. A Fig. 3 exibe uma representação de como funciona o sistema de filas no contexto em que a época de decisão é dada pelo evento tipo 1 apresentado na seção anterior.

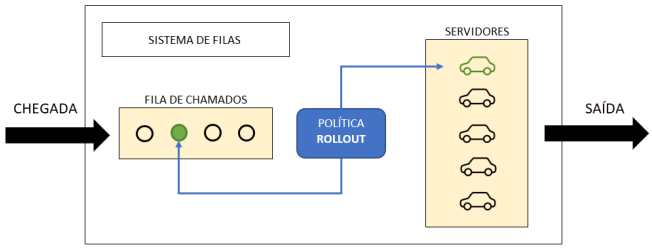

Fig. 3. Representação do sistema de fila para o sistema de alocação de veículos. A figura ilustra a ocorrência de um evento tipo 1, em que um veículo fica livre e uma das chamadas na fila de espera é escolhida para ser atendida pelo veículo.

Adicionalmente aos principais elementos que compõem um sistema de fila, tal como os clientes, servidores, processo de chegada e política da fila, temos uma propriedade que confere uma outra variável ao sistema de filas. Os servidores e clientes estão posicionados sobre uma malha quadriculada de localizações. Assim, a posição em que se encontra um determinado veículo ou cliente influencia potencialmente a qualidade do antedimento, uma vez que buscamos decisões cujo tempo de espera do cliente até que o veículo se encontre na sua localização de origem da chamada seja minimizado.

Representamos neste modelo o espaço geográfico de localizações através de um grafo reticulado quadrático de dimensão $n \times n$, em que cada vértice do grafo representa uma localização viável. As posições de cada vértice foram geradas de forma sintética, e as distâncias entres eles são calculadas utilizando a distância de Manhattan.

Neste ambiente de decisão, as chamadas chegam ao sistema seguindo um processo de Poisson. Seja λ a taxa média de chegada dos chamados, temos que os intervalos entre as chegadas são variáveis aleatórias que seguem a distribuição exponencial com parâmetro λ. Ao chegar ao sistema, o chamado é adicionado a uma fila de espera até que seja alocado a um veículo. Cada veículo assume o papel de servidor neste sistema de filas, atendendo um único chamado por vez. A política da fila é definida pela própria política *rollout*. Assim, a ordem na qual os clientes serão atendidos pelos servidores será escolhida por um agente decisor guiada pela política *rollout*.

Foi construído um modelo de simulação em linguagem Python com uso da biblioteca de simulação de eventos discretos SimPy 4.0. O modelo simula o processo natural associado ao PDSM subjacente ao problema de alocação de veículos. Durante a simulação, sempre que ocorre um dos eventos do tipo 1 ou 2, o agente decisor (implementado como uma classe em Python) aplica o algoritmo *rollout*. Para cada possível decisão de alocação, o agente decisor cria uma cópia da simulação do processo natural para cada estado viável subsequente resultante do par estado-decisão, e simula o processo durante um horizonte de tempo enquanto aplica uma política de decisão heurística $\mathcal{H}$.

Em cada cópia da simulação, o processo de chegada é interrompido de forma que o processo é simulado somente até que todas as chamadas em espera sejam completamente atendidas ou enquanto um determinado horizonte de simulação não seja atingido. Quanto à implementação computacional, as cópias das simulações são executadas em paralelo com uso de múltiplos processos do sistema operacional Linux. Após esta etapa, o decisor terá uma aproximação do custo futuro obtido via heurística para cada possível decisão. A política *rollout* escolherá a decisão que corresponda ao atraso mínimo acumulado conforme (4).

V. Experimentos e Resultados

Nesta seção, vamos analisar a performance do algoritmo *rollout* aplicado em um cenário sintético e simulado. O objetivo é mostrar que por meio deste algoritmo obtemos uma política cujo desempenho é pelo menos tão bom quanto da política-base e muitas vezes consideravelmente melhor. Para o problema de alocação de veículos proposto neste trabalho, a melhor política é aquela que apresenta o menor atraso acumulado, ou seja, a política que executa a alocação do veículos de modo que o tempo entre a chegada do chamado na fila até o momento em que o veículo chega para o atendimento seja o menor possível.

Nos experimentos, foram testadas três heurísticas-base: FIFO (*first-in-first-out*), que corresponde a escolher sempre o chamado há mais tempo na fila de espera, independente da sua localização; RANDOM, a escolha para alocação é feita aleatoriamente e uniformemente; NN (*nearest-neighbor*), o chamado escolhido para alocação será aquele cujo local é o mais próximo do veículo disponível, independente do tempo de espera.

As instâncias utilizadas para o experimentos foram geradas artificialmente. Trabalhamos com os seguintes parâmetros para produzir as instâncias:

- Número de carros: 5, 10, 15, 20, 25 e 30;
- Dimensão da matriz de localizações: 100, 200, 300, 400 e 500;
- Taxa média de chegada dos chamados: 4 e 9 chamados por unidade de tempo (min).

Combinando todos estes parâmetros, obtivemos 60 instâncias para o experimento. Para cada instância, fizemos 30 rodadas de simulação com horizonte de tempo igual a 1000, em que cada rodada consistia em executar duas etapas de experimentação independentes: primeiramente executamos a simulação utilizando apenas as 3 heurísticas bases, e em seguida fizemos uso do algoritmo *rollout* com as mesmas heurísticas-base. Durante a aplicação do algoritmo *rollout*, as simulações das heurísticas-base foram executadas com horizonte de tempo igual a 400 ou até que a fila de chamados se esvaziasse.

Os experimentos foram executados tendo por objetivo avaliar a magnitude da melhoria sobre as heurísticas-base promovida pelo uso do algoritmo *rollout*. Como dito anteriormente, o agente decisor foi modelado para que suas decisões objetivassem minimizar o atraso acumulado. Assim, a métrica utilizada para avaliar a qualidade das políticas foi o atraso acumulado médio.

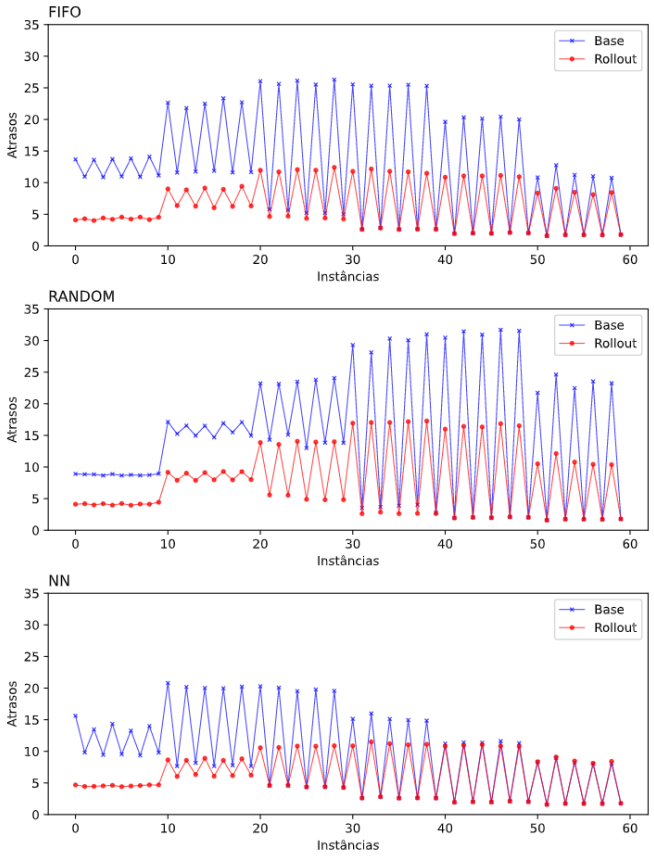

Fig. 4. Gráfico comparando o atraso acumulado médio obtido a partir da experimentação do uso das heurísticas-base e o uso da política *rollout*.

Apresentamos na Fig. 4 o atraso médio obtido para cada instância, comparando os resultados produzidos a partir da experimentação utilizando apenas as heurísticas-base com os resultados da política *rollout*. Esta primeira análise destina-se a evidenciar a propriedade de melhoria de política do algoritmo *rollout*. Observe que os resultados promovidos pela política *rollout* foram tão bons ou melhores que a heurística base. Os cenários nos quais a política *rollout* superou a política base foi no contexto em que havia maior demanda para o uso dos veículos, ou seja, quando havia uma alta taxa de chegada de novos pedidos e um baixo número de veículos disponível. Quando trabalhamos com instâncias com uma baixa taxa de chegada e uma maior disponibilidade de veículos, o resultando das duas abordagens eram semelhantes, dado que muitos dos chamados eram atribuídos logo que chegavam ao sistema.

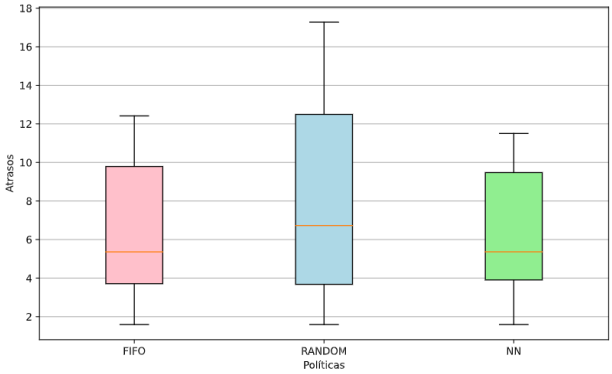

Fig. 5. Boxplot do atraso acumulado médio referente ao algoritmo *rollout* com uso das heurística-base FIFO, RANDOM e NN.

Obter boas políticas a partir do algoritmo *rollout* está diretamente relacionado à escolha de boas heurísticas-base. Portanto, além de mostrar que a política *rollout* é melhor que a política base, queremos também analisar qual das políticas escolhidas para este trabalho é a melhor para o nosso modelo. A partir dos resultados dos nossos experimentos, observamos que a política RANDOM apresentou a maior média dos atrasos, se comparado com a demais políticas-base. Veja na Fig. 5 que a melhor das três políticas experimentadas foi a política NN, com a menor média dos atrasos e menor variabilidade nos resultados. Porém, a política FIFO apresentou valores muito próximos da política NN. É fácil perceber o motivo pelo qual as políticas NN e FIFO foram melhores que a RANDOM, pois como o objetivo é minimizar o atraso, a FIFO é um política que minimiza o tempo de espera priorizando a ordem de chegada do clientes, e a NN minimiza o atraso considerando qual será o deslocamento mais breve.

VI. Conclusão

Neste artigo, propomos e avaliamos o uso de políticas geradas por um algoritmo *rollout*, uma estratégia de programação dinâmica aproximada, para o problema de alocação de veículos estocástico. Propomos uma abordagem que partiu da premissa de que o intervalo entre as épocas de decisão eram estocásticos, o que representa mais fielmente o cenário real do problema, o qual foi formulado como um processo de decisão semimarkoviano.

Os experimentos desenvolvidos evidenciaram que o algoritmo *rollout* é potencialmente útil para melhorar a performance de heurísticas-base, produzindo decisões com atraso médio em geral inferiores às políticas-base FIFO, RANDOM e NN testadas.

Ademais, a abordagem proposta é escalável para problemas reais e pode ser implementada *online*, uma vez que não precisa percorrer todo o espaço de estados como ocorre com métodos exatos de programação dinâmica.

Para trabalhos futuros, sugerimos estender a modelagem do problema de alocação de veículos para variantes mais ricas, adicionando características que ofereçam ao decisor outras possíveis decisões, tais como rejeitar um chamado ou decidir que o veículo seja realocado enquanto aguarda por uma nova tarefa para realizar.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico - FUNCAP; do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq por meio do projeto 422464/2016-3; e do NVIDIA GPU Grant Program.

REFERÊNCIAS

[1] H. Kuhn, "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly, vol. 2, no. 1-2, pp. 83-97, 1955. Available: 10.1002/nav.3800020109.

[2] D. Pentico, "Assignment problems: A golden anniversary survey", European Journal of Operational Research, vol. 176, no. 2, pp. 774-793, 2007. Available: 10.1016/j.ejor.2005.09.014.

[3] Spivey, M. and Powell, W., 2004. The Dynamic Assignment Problem. Transportation Science, 38(4), pp.399-419.

[4] R. Bellman, Dynamic programming. Princenton: Princenton University Press, 1957.

[5] Puterman, M. L. (1994). Markov decision processes: Discrete stochastic dynamic programming. Wiley.

[6] W. Powell, "A Stochastic Formulation of the Dynamic Assignment Problem, with an Application to Truckload Motor Carriers", Transportation Science, vol. 30, no. 3, pp. 195-219, 1996. Available: 10.1287/trsc.30.3.195.

[7] H. Simão, J. Day, A. George, T. Gifford, J. Nienow and W. Powell, "An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application", Transportation Science, vol. 43, no. 2, pp. 178-197, 2009. Available: 10.1287/trsc.1080.0238.

[8] M. Maxwell, S. Henderson and H. Topaloglu, "Tuning Approximate Dynamic Programming Policies for Ambulance Redeployment via Direct Search", Stochastic Systems, vol. 3, no. 2, pp. 322-361, 2013. Available: 10.1287/10-ssy020.

[9] V. Schmid, "Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming", European Journal of Operational Research, vol. 219, no. 3, pp. 611-621, 2012. Available: 10.1016/j.ejor.2011.10.043.

[10] Z. Xu et al., "Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms", Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018. Available: 10.1145/3219819.3219824 [Accessed 20 June 2021].

[11] Holler, J. et al. "Deep Reinforcement Learning for Multi-driver Vehicle Dispatching and Repositioning Problem." 2019 IEEE International Conference on Data Mining (ICDM) (2019): 1090-1095.

[12] Bertsekas, D., J. Tsitsiklis, and C. Wu (1997). Rollout algorithms for combinatorial optimization. Journal of Heuristics 3(3), 245–262.

[13] Bertsekas, D. and D. Castanon (1998). Rollout algorithms for stochastic scheduling problems. Journal of Heuristics 5(1), 89–108.

[14] Bertsimas, D. and R. Demir (2002). An approximate dynamic programming approach to multidimensional knapsack problems. Management Science 48(4), 550–565

[15] Secomandi, N. (2001). A rollout policy for the vehicle routing problem with stochastic demands. Operations Research 49(5), 796–802.

[16] Gallager, R. G. (2013). Stochastic Processes: Theory for Applications. Cambridge University Press.

[17] Gosavi, A. (2003). Simulation-Based Optimization. Springer US. https://doi.org/10.1007/978-1-4757-3766-0

[18] Goodson, J. C., Thomas, B. W., and Ohlmann, J. W. (2017). A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. European Journal of Operational Research, 258(1), 216–229. https://doi.org/10.1016/j.ejor.2016.09.040

[19] D. P. Bertsekas, "Rollout algorithms for discrete optimization: A survey," in Handbook of Combinatorial Optimization, P. M. Pardalos, D. Z. Du, and R. L. Graham, Eds. New York, NY, USA: Springer-Verlag, 2013, pp. 2989–3013

[20] Sutton, R. S. (Ed.). (1992). Reinforcement Learning. Springer US. https://doi.org/10.1007/978-1-4615-3618-5

[21] Bertsekas, D. P. (2007). Dynamic Programming and Optimal Control, Vol. II (3a ed.). Athena Scientific.