Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

Programa de Engenharia de Sistemas e Computação

CPS863 - Aprendizado de Máquina

Prof. Dr. Edmundo de Souza e Silva
(PESC/COPPE/UFRJ)

Lista de Exercícios 6

Luiz Henrique Souza Caldas
email: lhscaldas@cos.ufrj.br

16 de dezembro de 2024

Questão 1

Value Iteration, *Policy Iteration* e *Q-Learning* são algoritmos utilizados para encontrar a política ótima em problemas de decisão sequencial, como um Processo de Decisão de Markov (MDP). A diferença entre eles é a forma como a política ótima é encontrada. A descrição de cada um deles abaixo e as equações seguem a notação do livro *Reinforcement Learning: An Introduction* de Sutton e Barto [1].

Value Iteration

Calcula iterativamente a função de valor $V(s)$ para cada estado s até convergir para a função de valor ótima $V^*(s)$. A função de valor é calculada tornando a equação de Bellman de otimalidade em uma regra de atualização iterativa:

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')] \tag{1}$$

onde $V_k(s)$ é a função de valor no passo k, $p(s', r|s, a)$ é a probabilidade de transição para o estado s' e recompensa r dado o estado s e ação a e γ é o fator de desconto, que regula a importância dada as recompensas futuras.

A convergência da função de valor é dada pela condição de parada:

$$\Delta = \max_s |V_{k+1}(s) - V_k(s)| < \theta \tag{2}$$

onde θ é um pequeno limiar (*threshold*), que determina a acurácia da convergência. Ao ser atingida esta condição, podemos considerar que a função de valor ótima $V^*(s)$ foi encontrada:

$$V^*(s) \approx V_k(s) \tag{3}$$

Após a convergência da função de valor, a política ótima $\pi^*(s)$ é obtida a partir da função de valor ótima:

$$\pi^*(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')] \tag{4}$$

onde argmax é o operador que retorna o argumento que maximiza a função.

Policy Iteration

Calcula iterativamente a política ótima $\pi^*(s)$ em duas etapas: **avaliação** e **melhoria da política**.

- Na etapa de avaliação, a função de valor $V(s)$ é calculada para a política atual $\pi(s)$ a partir da equação de Bellman:

$$V_{k+1}(s) = \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V_k(s')] \tag{5}$$

onde $V(s)$ é a função de valor para o estado s, $p(s', r|s, \pi(s))$ é a probabilidade de transição para o estado s' e recompensa r dado o estado s e ação $\pi(s)$ e γ é o fator de desconto. Este processo é repetido até que se atinja o critério de convergência $\Delta = \max_s |V_{k+1}(s) - V_k(s)| < \theta$.

- Na etapa de melhoria da política, feita após a avaliação da função de valor, a política é atualizada para a ação que maximiza a função de valor:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')] \tag{6}$$

O algoritmo inicializa com uma política $\pi(s)$ arbitrária e continua iterando entre a avaliação e melhoria da política até que a política não mude mais. Neste ponto, a política ótima $\pi^*(s)$ foi encontrada.

Q-Learning

Calcula a política ótima $\pi^*(s)$ aprendendo a função de ação-valor $Q(S, A)$, que estima o retorno esperado ao tomar a ação a no estado s, sem depender de conhecimento prévio de um modelo do ambiente (transições e recompensas). O algoritmo atualiza a função de ação-valor iterativamente a partir da equação de Bellman para a função de ação-valor:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{7}$$

onde a ação A é escolhida de acordo com uma política de exploração, α é a taxa de aprendizado, R é a recompensa imediata, γ é o fator de desconto e S_{t+1} é o estado resultante da ação A_t no estado S_t.

A politica de exploração é diferente da politica ótima, e é usada para explorar o ambiente e evitar a convergência prematura para uma política subótima. Isso faz com que o algoritmo *Q-Learning* seja considerado um algoritmo de aprendizado por reforço *off-policy*. Uma política de exploração comum é a política ϵ-gulosa, que escolhe a ação que maximiza a função de ação-valor com probabilidade $1 - \epsilon$ e uma ação aleatória com probabilidade ϵ.

O termo $R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$ é o erro TD (*Temporal Difference*), que é usado para atualizar a função de ação-valor. Isso faz com que o algoritmo *Q-Learning* seja incluído na categoria de métodos de aprendizado por reforço baseados em diferenças temporais (*Temporal Difference Learning*).

$Q(S, A)$ é inicializado arbitrariamente e atualizado a cada passo da simulação, até que a função de ação-valor convirja para a função de ação-valor ótima $Q^*(S, A)$. A política ótima $\pi^*(s)$ é obtida a partir da função de ação-valor ótima:

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{8}$$

Comparação

A principal diferença entre *Value Iteration* e *Policy Iteration* é que o primeiro calcula a função de valor diretamente, iterando sobre ela até encontrar o valor ótimo, para então derivar a política ótima. Já o segundo calcula a função de valor para a política atual e, em seguida, atualiza a política para a ação que maximiza a função de valor. Este processo é repetido até que a política não mude mais. Ambos métodos assumem o conhecimento prévio de um modelo do ambiente (transições e recompensas).

Por outro lado, o *Q-Learning* não precisa de um modelo do ambiente para calcular a política ótima. Ele aprende interagindo com o ambiente (ou uma simulação), atualizando a função de ação-valor iterativamente. Além disso, o *Q-Learning* é um algoritmo *off-policy*, utilizando uma política de exploração (como ϵ-gulosa) para explorar o ambiente enquanto converge para a política ótima, que é obtida escolhendo a ação que maximiza o valor estimado.

Questão 2

Definição da Cadeia de Markov

A cadeia de Markov para este problema é definida pelos seguintes elementos:

1. **Estados**: Representados pela tupla (c, s), onde:
   - c é o número de clientes no sistema (limitado entre 0 e 8).
   - s é o número de servidores disponíveis (limitado entre 1 e 3).
2. **Ações**: As ações disponíveis em cada estado são:
   - -1: Remover um servidor.
   - 0: Manter o número atual de servidores.
   - $+1$: Adicionar um servidor (respeitando os limites).
3. **Transições**: Determinadas pelas probabilidades de chegada de novos clientes no final de cada intervalo de tempo:
   - $p_0 = 0.4$: Probabilidade de 0 clientes chegarem.
   - $p_2 = 0.2$: Probabilidade de 2 clientes chegarem.
   - $p_4 = 0.4$: Probabilidade de 4 clientes chegarem.

   A transição entre estados considera:
   - Atendimento: $\min(c, s)$ clientes são atendidos no início de cada intervalo.
   - Clientes restantes: Permanecem no sistema para o próximo intervalo.
   - Restrições: O número total de clientes no sistema é limitado a 8.
4. **Ordem dos eventos**: Para cada intervalo de tempo, a ordem dos eventos considerada é:
   i. Atendimento de clientes.
   ii. Adição/remoção de servidores.
   iii. Chegada de clientes.
5. **Recompensas**: A recompensa para cada transição é composta por:
   - Ganho por cliente atendido: $T \cdot \min(c, s)$, com $T = 10$.
   - Custo por servidor: $-R_s \cdot s'$, com $R_s = 5$.
   - Penalidade por fila: $-R_q$ se $c' > 4$, caso contrário 0, com $R_q = 10$.
   - Penalidade por ociosidade: $-R_0$ por servidor não utilizado, $\max(s' - c', 0)$, com $R_0 = 2$.

   Onde c e s são os valores de clientes e servidores no estado atual e c' e s' são os valores no próximo estado.

Solução por *Value Iteration*

Foi implemmentado em Python uma função que calcula a política ótima para a cadeia de Markov descrita. O código fonte está disponível no repositório indicado no final deste relatório, na pasta `lista_6`, arquivo `value_itaration.py`. O código principal, onde são definidas as probabilidades de transição e as recompensas, está disponível no arquivo `main.py`. O código segue o seguinte fluxo:

1. Inicializa $V[s] = 0$ para todos os estados s.
2. Iterativamente calcula os valores $V[s]$ para cada estado, atualizando-os com base na equação de Bellman:
$$V(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \cdot (r + \gamma \cdot V(s'))\,.$$
3. Em cada iteração, verifica a convergência comparando a maior mudança (Δ) entre os valores antigos e novos. O loop termina quando $\Delta < \theta$, o limiar definido.
4. Após convergir, calcula a política ótima $\pi^*(s)$ para cada estado, escolhendo a ação a que maximiza o valor esperado $V(s)$:
$$\pi^*(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \cdot (r + \gamma \cdot V(s'))\,.$$
5. Retorna a função de valor ótima $V(s)$ e a política ótima $\pi^*(s)$.

Foi utilizado um fator de desconto $\gamma = 0.9$ e um limiar de convergência $\theta = 1e - 6$. Para esses valores, a convergência ocorreu em 2457 iterações. A função de valor ótima calculada e a política ótima derivada dela são apresentadas na figura 1. A variação de Δ ao longo das iterações é apresentada na figura 2.

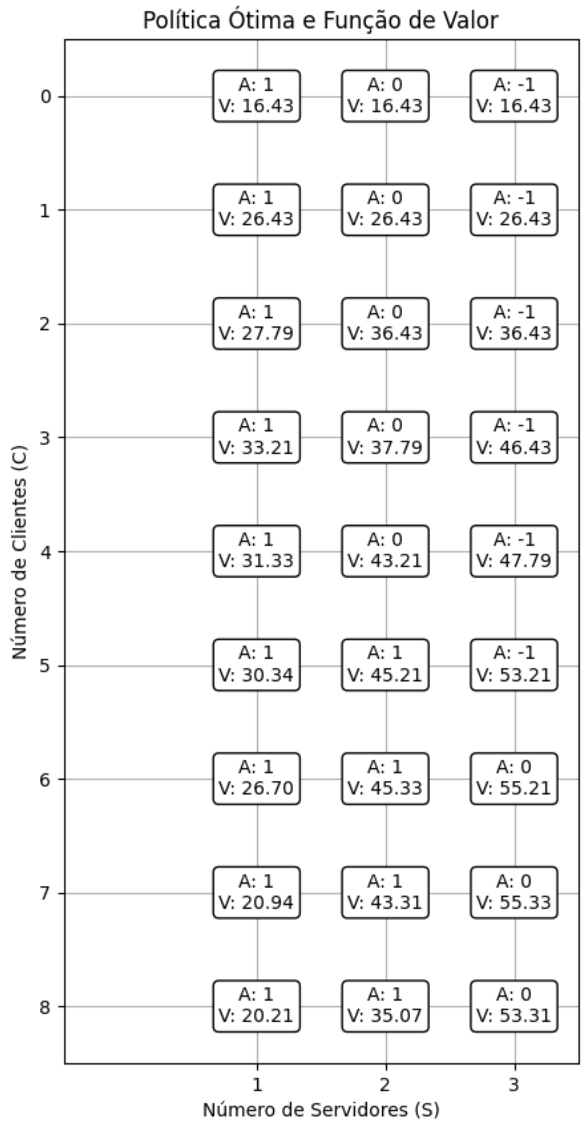


Figura 1: Função de valor ótima e política ótima calculadas por *Value Iteration*.

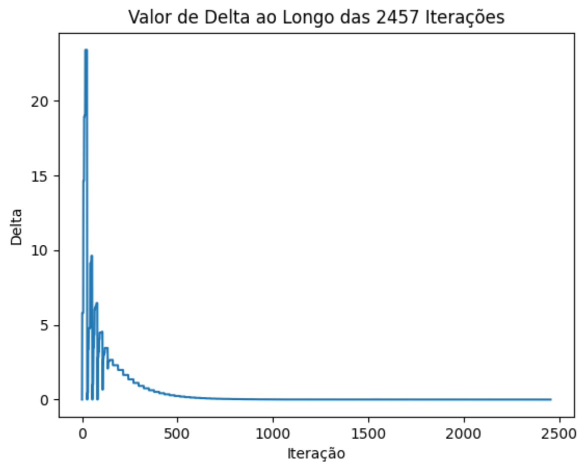


Figura 2: Variação de Δ ao longo das iterações de *Value Iteration*.

Solução por *Policy Iteration*

Foi implementado em Python uma função que calcula a política ótima para a cadeia de Markov descrita. O código fonte está disponível no repositório indicado no final deste relatório, na pasta `lista_6`, arquivo `policy_itaration.py`. O código principal, onde são definidas as probabilidades de transição e as recompensas, está disponível no arquivo `main.py`. O código segue o seguinte fluxo:

1. Inicializa uma política arbitrária $\pi(s) = -1$ e uma função de valor $V(s) = 0$ para todos os estados s.
2. **Etapa 1 - Avaliação da Política:**
   (a) Para cada estado s, atualiza $V(s)$ usando a equação:
   $$V(s) = \sum_{s',r} P(s', r \mid s, \pi(s)) \cdot (r + \gamma \cdot V(s'))\,.$$
   (b) Repete as atualizações até que a maior mudança em $V(s)$ entre duas iterações seja menor que um limiar θ.
3. **Etapa 2 - Melhoria da Política:**
   (a) Para cada estado s, identifica a melhor ação a que maximiza o valor esperado:
   $$\pi(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \cdot (r + \gamma \cdot V(s'))\,.$$
   (b) Se a nova política $\pi(s)$ for igual à política anterior para todos os estados, o algoritmo termina. Caso contrário, retorna à etapa de avaliação.
4. Retorna a política ótima $\pi^*(s)$ e a função de valor ótima $V^*(s)$.

Foi utilizado um fator de desconto $\gamma = 0.9$ e um limiar de convergência $\theta = 1e - 6$. Para esses valores, a convergência ocorreu em 4 iterações do laço externo *while principal*, porém foram contabiizadas um total de 5292 iterações internas da etapa de avaliação da política. A função de valor ótima calculada e a política ótima derivada dela são apresentadas na figura 3. A variação de Δ ao longo das iterações é apresentada na figura 4.

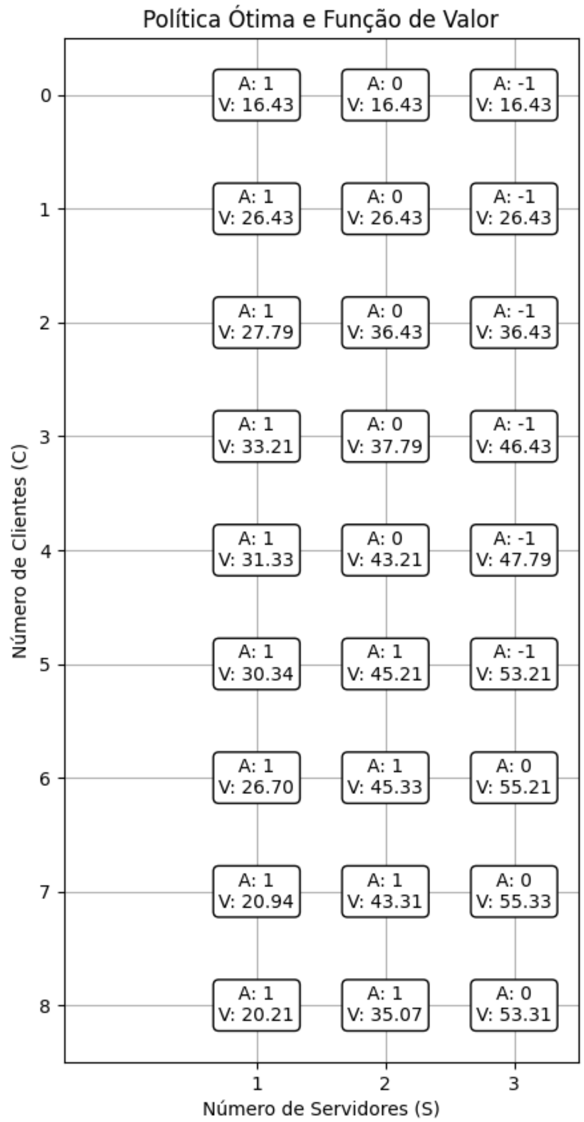


Figura 3: Função de valor ótima e política ótima calculadas por *Policy Iteration.*

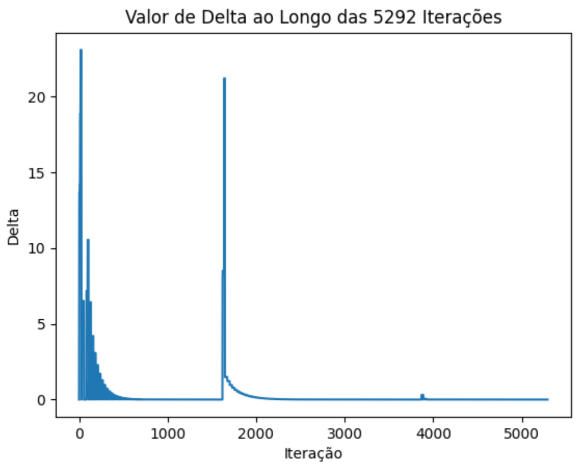


Figura 4: Variação de Δ ao longo das iterações de *Policy Iteration*.

Q-Learning

Foi implementado em Python um algoritmo de *Q-Learning* para resolver o problema da cadeia de Markov descrita. O código fonte está disponível no repositório indicado no final deste relatório, na pasta `lista_6`, arquivo `q_learning.py`. O código principal, onde são definidas as probabilidades de transição e as recompensas, está disponível no arquivo `main.py`. Como o problema em questão não possui estado terminal, o algoritmo foi adaptado para, a cada epsódio, limitar o número de passos. Além disso, foi implementada uma classe `EnvironmentSimulator` para simular o ambiente, recebendo como entrada o estado atual e a ação tomada e retornando o próximo estado e a recompensa obtida. O código segue o seguinte fluxo:

1. Inicializa $Q(s, a)$ com zeros para todos os estados e ações.
2. Para cada episódio:
   (a) Escolhe um estado inicial aleatório.
   (b) Para cada passo no episódio:
      i. Escolhe uma ação (usando política ϵ-gulosa):
         - Com probabilidade ϵ, escolhe uma ação aleatória (*exploration*).
         - Caso contrário, escolhe a ação que maximiza $Q(s, a)$ (*exploitation*).
      ii. Simula o ambiente com a ação escolhida para obter:
         - O próximo estado s'.
         - A recompensa r.
      iii. Atualiza $Q(s, a)$ usando a equação de aprendizado por diferença temporal (TD):

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

      iv. Atualiza o estado atual para s'.

v. Se atingir o número máximo de passos, termina o episódio.

(c) Após o episódio, calcula:

- A nova função de valor $V(s) = \max_a Q(s, a)$.
- A mudança absoluta máxima $\Delta V = \max_s |V_{\text{novo}}(s) - V(s)|$.

(d) Atualiza $V(s)$ e armazena ΔV para análise de convergência.

3. Deriva a política ótima $\pi^*(s)$ escolhendo, para cada estado, a ação que maximiza $Q(s, a)$.

4. Retorna a política ótima $\pi^*(s)$, a função de valor $V(s)$, e a lista de ΔV ao longo dos episódios.

Foram utilizados os seguintes hiperparâmetros: fator de desconto $\gamma = 0.9$, taxa de aprendizado $\alpha = 0.1$, probabilidade de exploração $\epsilon = 0.1$, número de episódios 1000 e limite de passos por episódio 100. Como não há um critério de convergência, o algoritmo executa 100 mil iterações. A função de valor ótima calculada e a política ótima derivada dela são apresentadas na figura 5. A variação de ΔV ao longo dos episódios é apresentada na figura 6.

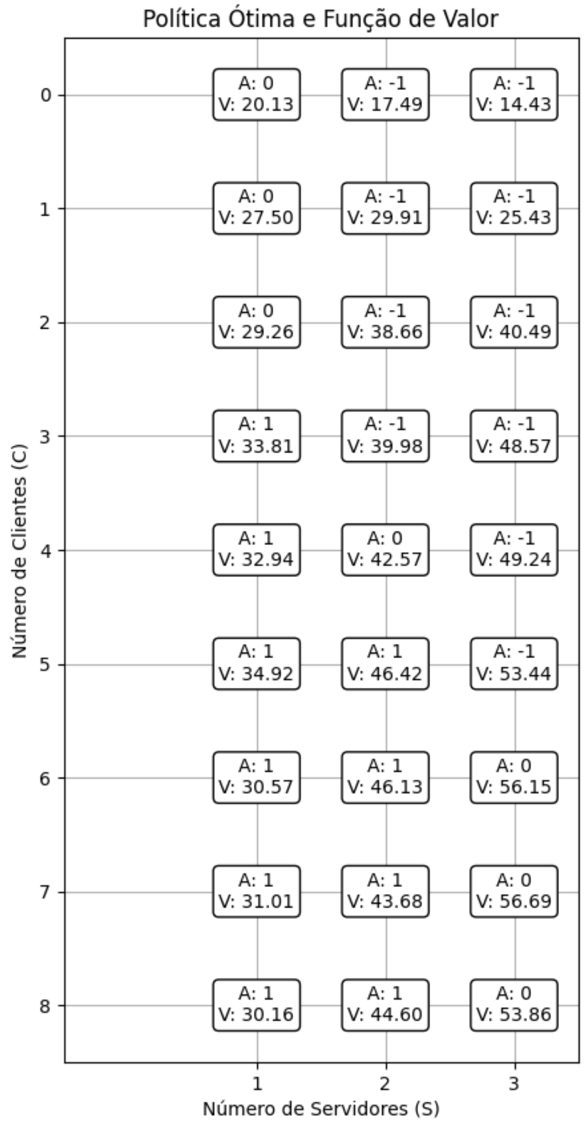


Figura 5: Função de valor ótima e política ótima calculadas por *Q-Learning*.

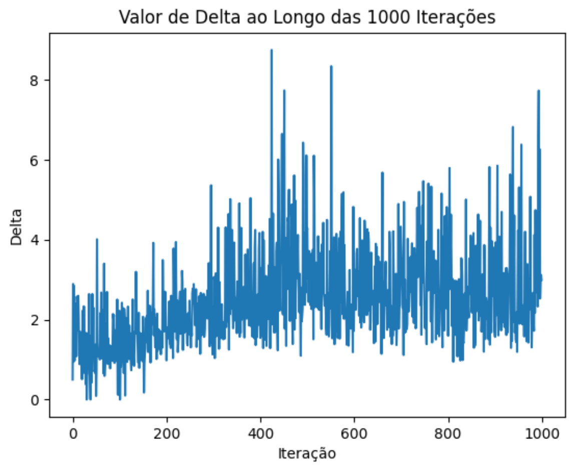
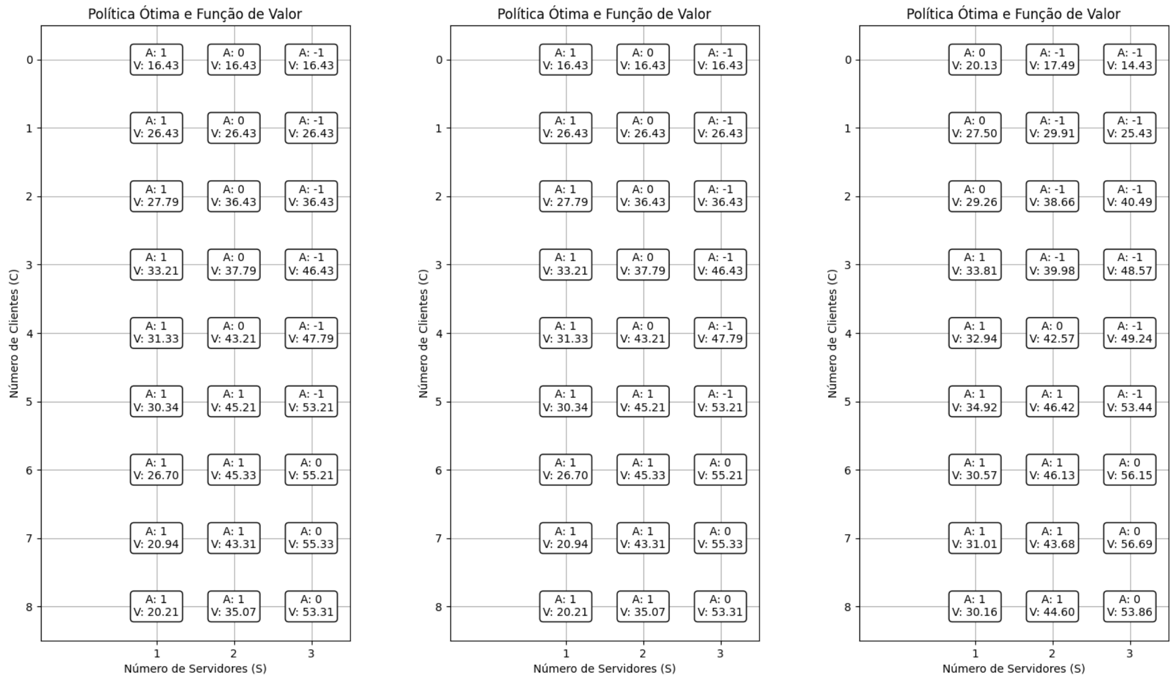


Figura 6: Variação de ΔV ao longo dos episódios de *Q-Learning*.

OBS: devido a aleatoriedade do algoritmo, e a falta de um estado terminal, não é possível observar uma convergência clara na figura 6, como nos métodos anteriores. Entretanto é poss ivel observar que os valores de $V^*(s)$ para cada estado na figura 5 são próximos dos valores obtidos pelos métodos anteriores e as ações da política ótima também são semelhantes. Foram feitas várias execuções do algoritmo alterando os hiperparâmetros, e nenuma delas fez ele convergir.

Comparação dos Métodos

Abaixo vemos as figuras 1, 3 e 5 exibidas lado a lado.



(a) *Value Iteration* **(b)** *Policy Iteration* **(c)** *Q-Learning*

Figura 7: Comparação das políticas e funções de valor ótimas calculadas pelos métodos.

O métodos *Value Iteration* e *Policy Iteration* apresentaram resultados identicos, com mesma política ótima e função de valor ótima. O método *Q-Learning* apresentou resultados semelhantes, com valores de $V^*(s)$ próximos e ações semelhantes a da política ótima. Entretanto, o método *Q-Learning* não convergiu, possivelmente devido a aleatoriedade do algoritmo e a falta de um estado terminal.

O método *Value Iteration* foi o mais rápido, convergindo em 2457 iterações. O método *Policy Iteration* foi o segundo mais rápido, convergindo em 4 iterações do laço externo *while principal*, porém foram contabiizadas um total de 5292 iterações internas da etapa de avaliação da política. O método *Q-Learning*, por não ter um critério de convergência, executando sempre o mesmo número de passos vezes o número de episódios, foi o mais lento, com 100 mil iterações.

Códigos

Os códigos utilizados para a resolução dos exercícios estão disponíveis no repositório do GitHub: `https://github.com/lhscaldas/cps863/`

Referências

[1] SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. 2nd. ed. Cambridge, MA: MIT Press, 2018. ISBN 978-0262039246.