Pontifícia Universidade Católica RJ

Path Planning and Path Covering using Q Learning Algorithm

Curso de Pós-graduação: BI Master - Intelligent Business Decision Support Systems

Aluno: Daniel Werneck Rodrigues

Matrícula: 211.101.278

Rio de Janeiro, 06 de Julho de 2023

I. Resumo

As calamidades sempre ocorreram no mundo e, muitas vezes, a dificuldade na identificação dos sinais que anunciam as suas ocorrências dificultam também, a adoção de medidas de precaução. Sendo assim, é desejável que se busque o melhor comportamento e planejamento diante de um desastre, de forma a maximizar a quantidade de sobreviventes eventualmente existentes e minimizar o tempo de exposição ao perigo ao qual as vítimas poderão estar submetidas.

O objetivo deste trabalho é otimizar o planejamento do resgate de vítimas, concluindo a busca com o menor tempo (menor quantidade de passos), dado um cenário em que conhecemos as prioridades de cada subespaço em nosso espaço de busca. Além do principal objetivo e da aplicabilidade direta na área de calamidades, tem-se como um dos objetivos desta pesquisa a capacidade de generalização, ou seja, que o trabalho aqui apresentado possa ser aplicado a outros problemas com características semelhantes. Para resolução deste problema, será utilizada o método *Q Learning* na área de *Reinforcement Learning* para a otimização da rota a ser percorrida pelo agente.

Em *Reinforcement Learning*, o aprendizado ocorre através da interação de um agente com o ambiente, recebendo recompensas ou punições. Após n jogos jogados, temos um agente especializado no problema em questão que aprende através da maximização total das recompensas recebidas ao longo de cada jogo. Quase todos os problemas de *Reinforcement Learning* usam o framework matemático de *Markov Decision Process (MDP)*, no qual estados futuros dependem apenas do estado presente. Este framework pode ser representado com cinco elementos importantes: (S): Estados que o agente pode entrar, (A): conjunto de ações que agente pode tomar, ($P_{ss'}^{a}$): probabilidade de agente se mover do estado s para estado s' por executar ação a e (γ): Fator de desconto que controla a importância de recompensas imediatas em detrimento de recompensas a longo prazo. [1]

A programação dinâmica é outro conceito importante para problemas de *Reinforcement Learning*, que se consiste em uma técnica para resolução de problemas complexos através da separação desse problema em subproblemas e do aproveitamento de soluções previamente estabelecidas para poupar esforço computacional, o que minimiza o tempo de processamento. [1]

Ainda, para este trabalho será utilizada a abordagem de Diferença Temporal, que não requer conhecimento prévio a respeito da dinâmica do modelo. O algoritmo de aprendizado TD foi introduzido por Sutton em 1988. O algoritmo traz os benefícios do método de Monte Carlo e da programação dinâmica (DP). Assim como o método de Monte Carlo, não requer dinâmica de modelo e, assim como DP não requer o tempo de espera até o final do episódio para fazer uma estimativa da função de valor. Em vez disso, é realizada uma aproximação da estimativa atual com base na estimativa aprendida anteriormente. [1]

Será utilizado o algorítimo de *Q Learning*, que se trata de algoritmo simples e muito popular em *Temporal Difference Learning*. Esse algoritmo tem como objetivo a atualização do *Q Value*($Q(s, a)$) a cada passo do agente no ambiente, e ao final, encontrar os pares de estado ação que levarão a maximização da recompensa final. [1]

II. Problema e Premissas

Cada subespaço do nosso espaço de busca será subdividido em quadrantes, onde cada quadrante terá uma prioridade pré-definida, sendo posteriormente utilizada como recompensa para o treinamento do agente. Essa prioridade pode ser estabelecida de acordo com: maior chance de sobrevivência das vítimas contidas no quadrante, maior quantidade de vítimas no quadrante, ou qualquer outra forma de priorização que se julgue adequada para o problema em questão. Para este trabalho, será considerada uma matriz de prioridades onde temos um epicentro da catástrofe com maior priorização e, quanto mais afastado deste epicentro, menores serão as prioridades (exemplo: queda de avião num ponto definido, em que suas áreas adjacentes, ou seja, quanto mais longe da aeronave, menores chances de localizarem-se vítimas).

Além disso, também serão pré-definidos obstáculos no espaço de busca que estarão contidos nas matrizes de prioridades e marcadas com valor -100.

O problema será resolvido em etapas, da mais simples para a mais complexa. Sendo assim serão considerados os cenários:

i. **'matriz muito simples' (matriz 4x3);**
ii. **'matriz simples' (5x5);**
iii. **'matriz média' (8x8);**
iv. **'matriz complexa' (9x13).**

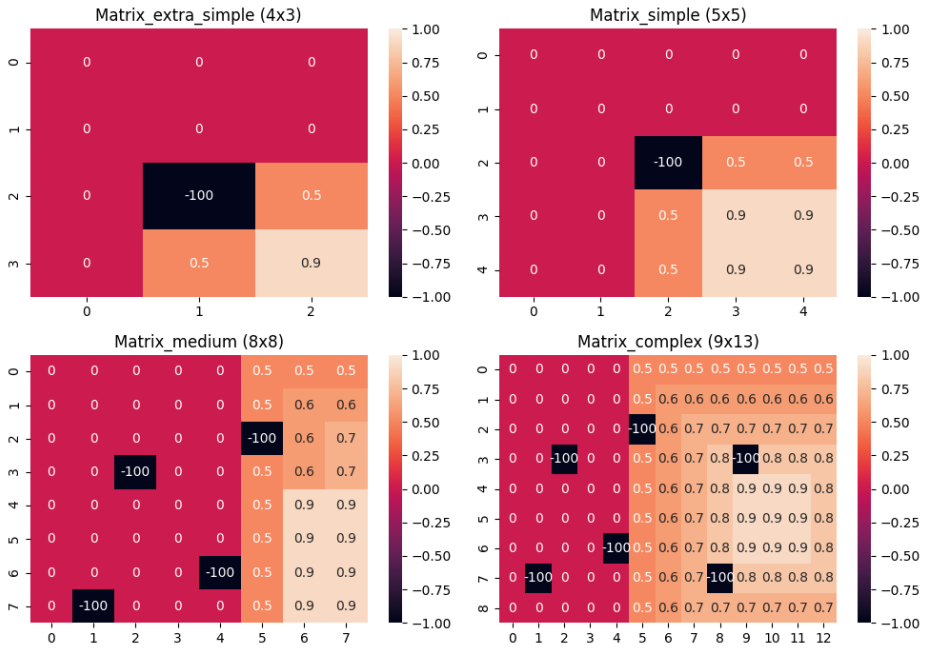

Figura I. As matrizes de prioridades abordadas como premissa para desenvolvimento

III. Modelagem

Para resolução do problema foi desenvolvido um ambiente com a definição de duas classes. A primeira de "estado", que define os atributos de estado percorrido pelo agente, e "agente", com os atributos e os métodos pertinentes, como o de execução de cada jogo e a atualização da tabela Q por cada passo (*round*). O objetivo de cada jogo é, sempre iniciando do estado (0,0), com a menor quantidade de passos, percorrer todos os quadrantes com valores maiores que zero, privilegiando os subespaços com maiores valores de prioridade e por fim, maximizando a recompensa total.

A atualização da tabela Q é realizada a cada passo de acordo com a modelagem clássica:

$$Q^{new}(s_t, a_t) = (1-\alpha).Q(s_t, a_t) + \alpha.(r_{tf}(s_t, a_t, d\_f_t) + \gamma.\max_a \mathrm{Q}(s_{t+1}, a))$$

Onde:

$$r_{tf}(s_t, a_t, d\_f_t) = \begin{cases} r_t(s_t, a_t).\left(d\_f_t^{\,n_s-1}\right), & s_t \notin \mathrm{s} \\ 0, & s_t \subset \mathrm{s} \end{cases}$$

Podemos definir as variáveis acima como:

$Q^{new}(s_t, a_t)$ → Valor atualizado da Tabela Q para o estado s_t e a ação a_t

α → *learning rate*

$Q(s_t, a_t)$ → Valor da Tabela Q no instante t para o estado s_t e a ação a_t

$r_{tf}(s_t, a_t, d\_f_t)$ → Recompensa no instante t para o estado s_t, ação a_t e deflator temporal d_f_t

s_t → Estado no instante t

a_t → Ação no instante t

d_f_t → Deflator temporal no instante t

γ → *Discount factor*

$\mathrm{Q}(s_{t+1}, a)$ → Estimativa do valor da tabela Q para o próximo estado

n_s → Quantidade de estados visitados
s → Estados visitados

A ação é escolhida conforme épsilon atual, que segue a seguinte função para cada episódio:

$$\varepsilon_e = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}).e^{-\varepsilon_{decay}.n_e}$$

Onde:

ε_e → *Exploration rate*

ε_{min} → *Exploration rate(mínimo)*

ε_{max} → *Exploration rate(máximo)*

ε_{decay} → Fator de queda do *exploration rate* a cada episódio

n_e → Quantidade de episódios jogados em cada configuração

A ação é escolhida conforme épsilon atual, que segue função que retorna a ação com melhores recompensas para casos de épsilon mais próximos a zero (*exploitation*) e ações aleatórias para casos de épsilon mais próximos a um (*exploration*). Nos primeiros episódios, jogados têm maiores épsilons e nos últimos, menores, de forma a, inicialmente, explorar as possíveis soluções e mais tarde se especializar nas melhores soluções obtidas. Sendo assim, o fator de decaimento da taxa de exploração *(ε_{decay})* é um dos hiperparâmetros importantes para o problema.

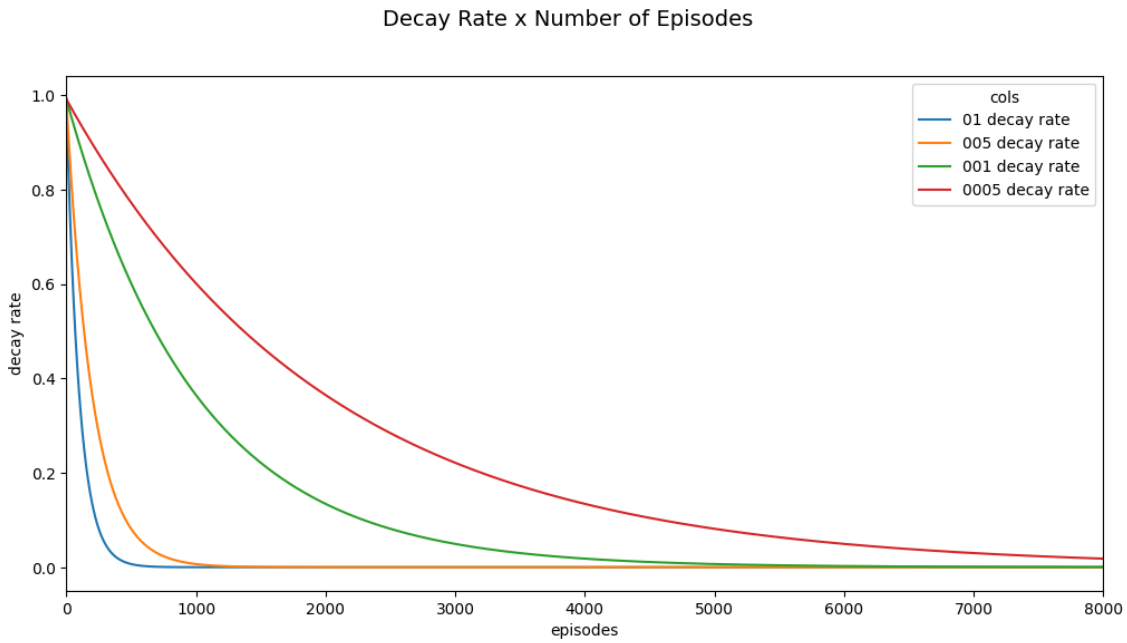

Figura III. Diferentes valores de decaimento de *Épsilon (ε_{decay})* por episódio.

Além disso, foi desenvolvida para este trabalho, a variável de deflação temporal (d_f), que não faz parte da modelagem clássica. Ela é responsável por reduzir as recompensas futuras de acordo com a quantidade de passos dados, atuando como "fator de urgência", que deteriora a prioridade (recompensa) a cada passo do agente no sistema, simulando assim, a passagem do tempo de busca.

Como para cada cenário avaliado temos diferente quantidade de passos médios para conclusão do problema, se faz necessário identificar o número ideal da variável deflatora (d_f) para cada cenário.

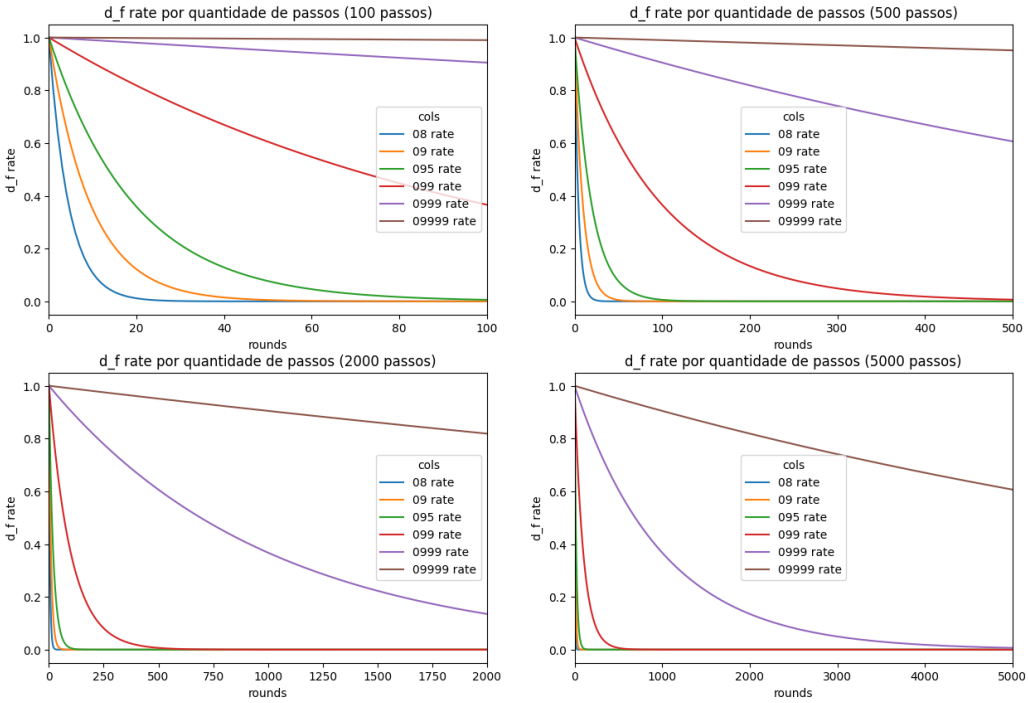

Figura II. Referente a diferentes valores de d_f por quantidades de rounds jogados

Foi avaliado o d_f em cada cenário, mantendo-se constantes os demais hiperparâmetros do sistema. Cada configuração a seguir teve o modelo rodado 10 vezes para obtenção de resultados:

i. ‘matriz muito simples’ (matriz 4x3) -> [0.8, 0.9, 0.95, 0.99, 0.999, 0.9999];
ii. ‘matriz simples’ (5x5) -> [0.8, 0.9, 0.95, 0.99, 0.999, 0.9999];
iii. ‘matriz média’ (8x8) -> [0.99, 0.999, 0.9999];

Como a recompensa final é maior quanto maior for o d_f utilizado, essa não foi utilizada como métrica principal para escolha do d_f. Ao invés disso, foi selecionado o d_f que produziu menor quantidade de passos, solução mais estável, com convergência mais rápida para a melhor solução encontrada e com tempo de execução aceitável. Foi notado que menores d_fs aumentam consideravelmente o tempo de execução para cenários de maiores complexidades, sendo assim, este ponto também foi levado em consideração para avaliação e seleção de melhor d_f.

Posteriormente foi realizada uma busca de melhores hiperparâmetros para cada cenário, sendo importante destacar que, para o cenário 9x13 não foi realizada busca de melhor d_f e de hiperparâmetros e, ao invés disso, utilizou-se os valores ótimos avaliados no cenário 8x8. Dessa forma, é avaliado como seria realizado o planejamento em um cenário mais realista e complexo com melhores parâmetros escolhidos a partir de cenário de menor complexidade.

A busca por hiperparâmetros se deu da seguinte forma, em 8.000 episódios e com a mesma configuração executada 10 vezes:

i. **'matriz muito simples' (matriz 4x3):**
   - Learning rate = [0.1, 0.01, 0.001]
   - Gamma = [0.95, 0.99]
   - d_f = [0.95]
   - Decaimento da taxa de exploração *(ε_{decay})* = [0.001, 0.0005]
   - Quantidade máxima de passos = 5000

ii. **'matriz simples' (5x5):**
   - Learning rate = [0.01]
   - Gamma = [0.95, 0.99]
   - d_f = [0.8]
   - Decaimento da taxa de exploração *(ε_{decay})* = [0.001, 0.0005]
   - Quantidade máxima de passos = 10000

iii. **'matriz média' (8x8):**
   - Learning rate = [0.01]
   - Gamma = [0.97, 0.99]
   - d_f = [0.99]
   - Decaimento da taxa de exploração *(ε_{decay})* = [0.001, 0.0005]
   - Quantidade máxima de passos = 10000

IV. Resultados

i. Resultado Matriz Muito Simples (4x3)

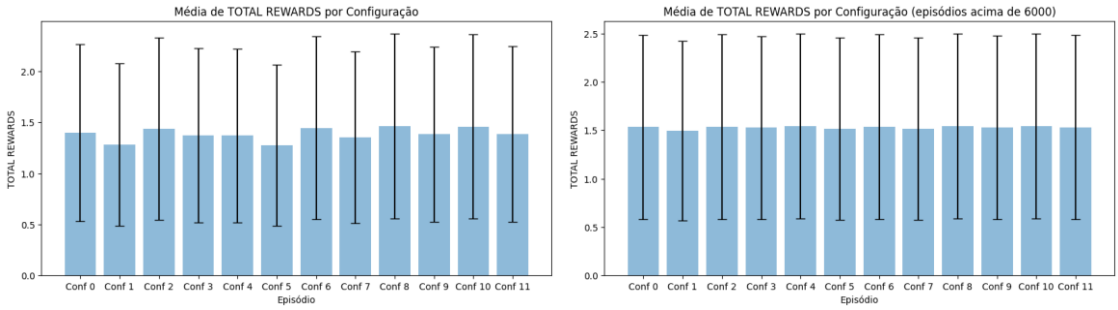

Figura III. Matriz 4x3 - Média de *Total Reward* por configuração geral e acima de 6000 episódios.

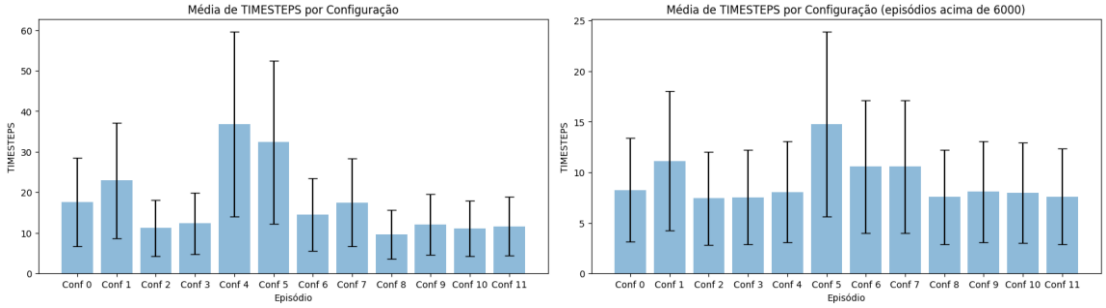

Figura IV. Matriz 4x3 - Média de *Time Steps* por configuração geral e acima de 6000 episódios.

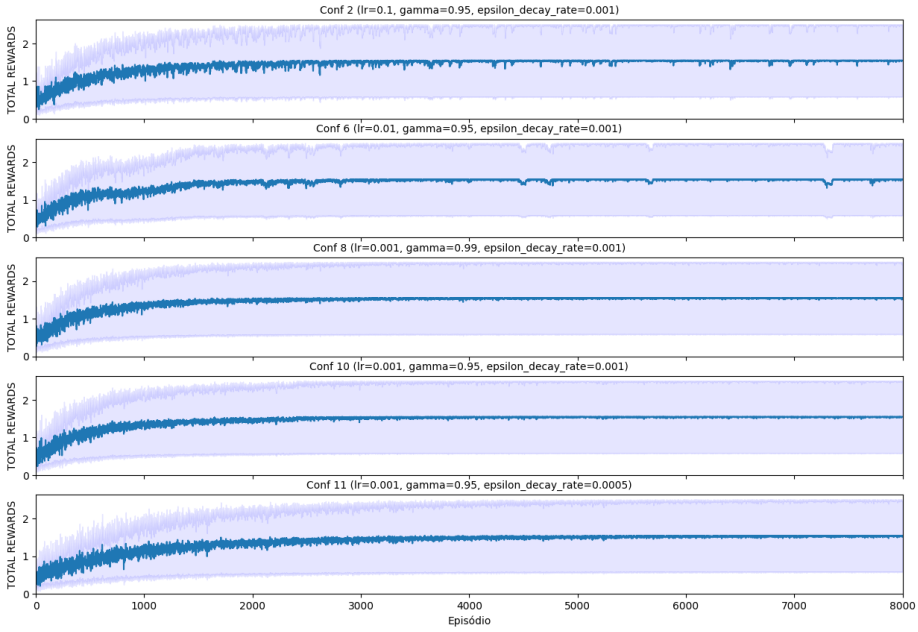

Figura V. Matriz 4x3 - *Total Reward* por episódio para cada configuração (melhores configurações).

Configuração 8 foi selecionada pois possui maior média geral de *Total Rewards* e menor média geral de quantidade de passos. Acima de 6000 episódios, possui solução estável e, aparentemente, possui convergência mais rápida que em outras configurações.

ii. Resultado Matriz Simples (5x5)

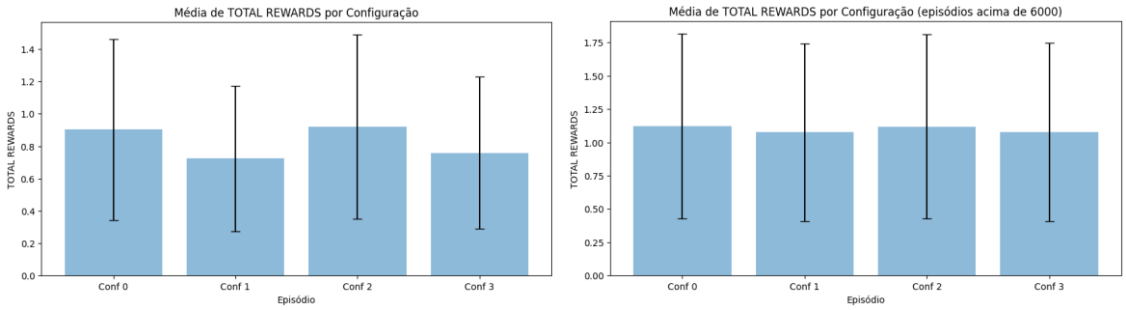

Figura VI. Matriz 5x5 - Média de *Total Reward* por configuração geral e acima de 6000 episódios.

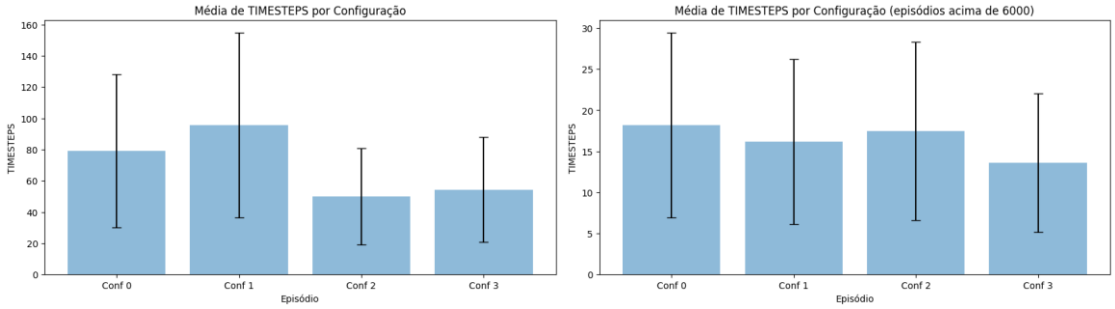

Figura VII. Matriz 5x5 - Média de *Time Steps* por configuração geral e acima de 6000 episódios.

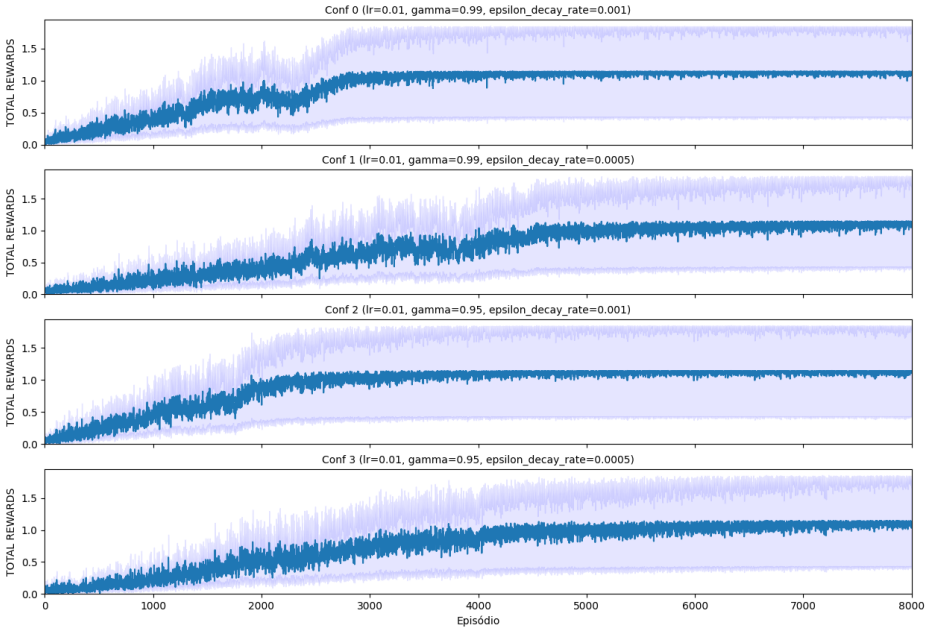

Figura VIII. Matriz 5x5 - *Total Reward* por episódio para cada configuração.

Configuração 2 foi selecionada pois possui maior média geral de *Total Rewards* e menor média geral de quantidade de passos. Acima de 6000 episódios, possui solução estável e, aparentemente, possui convergência mais rápida que em outras configurações.

iii. Resultado Matriz Média Complexidade (8x8)

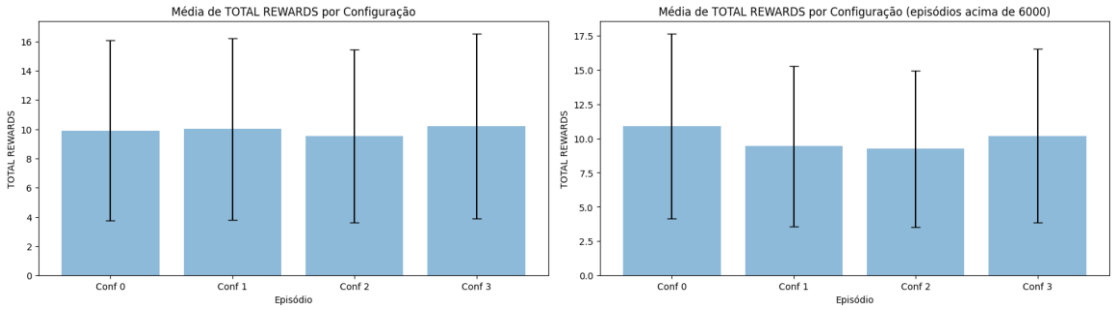

Figura VI. Matriz 8x8 - Média de *Total Reward* por configuração geral e acima de 6000 episódios.

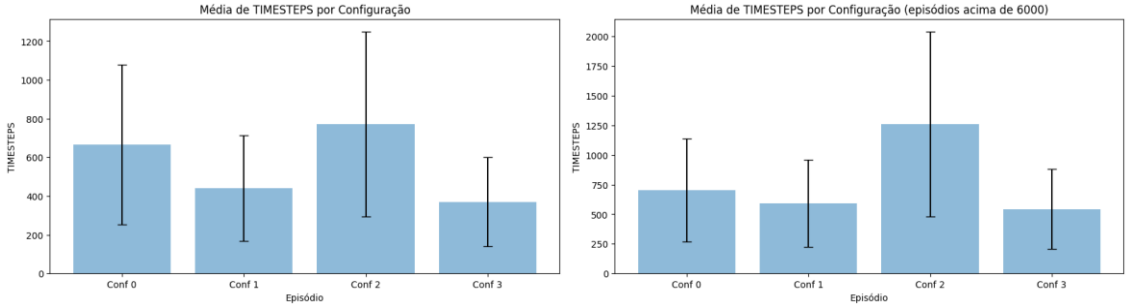

Figura VII. Matriz 8x8 - Média de *Time Steps* por configuração geral e acima de 6000 episódios.

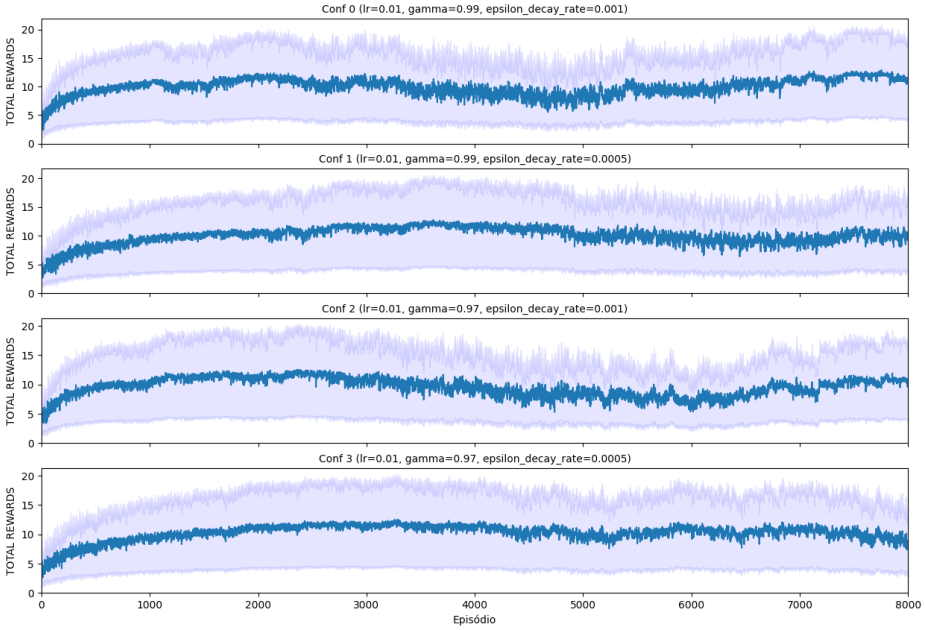

Figura VIII. Matriz 8x8 - *Total Reward* por episódio para cada configuração.

A configuração 3 foi selecionada pois possui maior média geral de *Total Rewards* e menor média geral de quantidade de passos. Aparentemente, possui convergência mais rápida que em outras configurações, apesar de que acima de 4000 episódios, resultados começarem a apresentar ruídos.

É importante notar que, para as configurações com taxa de decaimento da taxa de exploração de 0.001 e 0.0005, o resultado de *Total Rewards* começa a se deteriorar a partir de aproximadamente 2800 e 4500 episódios, respectivamente, sugerindo que modelo pode se beneficiar de maiores quantidade de episódios submetidos a maiores taxas de exploração.

No entanto, foi obtido resultado aceitável (sub-ótimo ou ótimo, pois não há garantia de máximo global) referente à configuração selecionada com quantidade de *Timesteps* de 29 e *Total Reward* de 13.37. Esse resultado foi obtido selecionando a instância com o maior *Total Reward* e menor quantidade de *Timesteps* obtida no treinamento. Dessa forma, a deterioração da convergência teve pouco impacto no resultado final.

iv. Resultado Matriz Complexa (9x13)

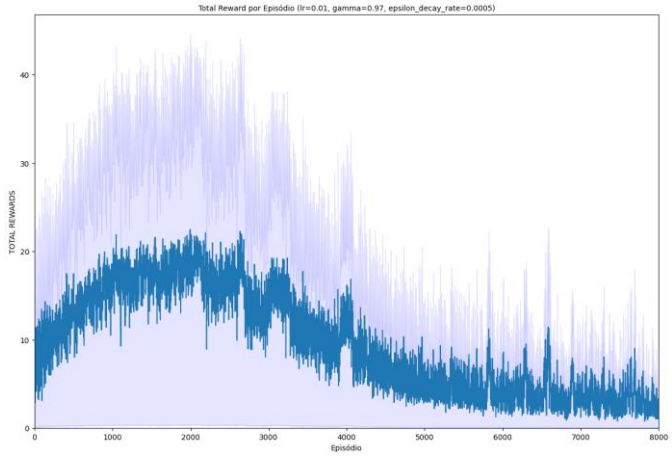

Figura IX. Matriz 9x13 - *Total Reward* por episódio para cada configuração.

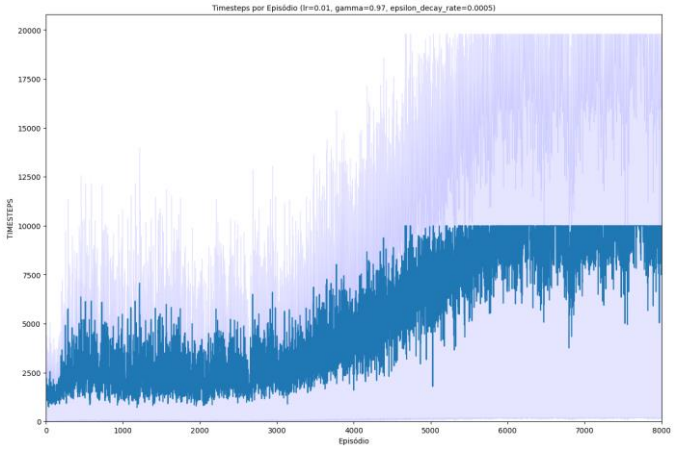

Figura X. Matriz 9x13 - *Timesteps* por episódio para cada configuração.

É percebida uma queda em *Total Rewards* e um aumento de *Timesteps* a partir de aproximadamente 2800 episódios. De maneira semelhante ao que ocorreu no cenário 8x8, que também teve piores resultados a partir de determinado limiar de episódios.

No entanto, ao contrário do resultado obtido no cenário 8x8, não foi percebido um resultado sub-ótimo aceitável, com a quantidade de *Timesteps* do agente no ambiente de 1013 e *Total Reward* máximo percebido de 28.96.

V. Conclusão

Os resultados foram obtidos com sucesso, para os cenários apresentados. Para futuros trabalhos é importante à utilização de outras técnicas de *Reinfocerment Learning* e de otimização para comparação com os resultados obtidos neste trabalho.

Foi percebido que para cada cenário há diferença nos melhores hiperparâmetros utilizados, o que demanda tempo e esforço computacional. Apesar disso, as diferenças nos resultados de recompensas totais foram sutis de configuração para configuração, sendo percebidos bons resultados em hiperparâmetros subótimos em relação aos escolhidos.

Para configuração da matriz de média complexidade (8x8) e alta complexidade (9x13) houve problemas na convergência do modelo, com a deterioração da recompensa total para episódios com menores taxas de exploração. Como sugestão para futuros trabalhos, é pertinente a utilização de outras curvas para redução do épsilon de maneira a aumentar a quantidade de episódios submetidos a uma maior taxa de exploração de forma a se obter maior convergência nesses cenários. Também pode ser interessante a definição de um valor mínimo para a taxa de exploração, de maneira que agente continue com chances de explorar mesmo em episódios mais avançados.

Também foi identificado que para cenários mais complexos, tem-se aumento substancial do tempo de processamento e maior dificuldade de convergência do modelo. Além disso, para estes cenários a redução no valor dos hiperparâmetros está, normalmente, associada com maiores tempos de processamento e esforço computacional. Essas características tornam difícil a utilização do trabalho proposto em cenários reais. Para futuros desenvolvimentos, verificar se treinamento de algoritmo de *Deep Reinforcement Learning*, pode ser aplicado de forma a mitigar esses problemas de aplicação e melhorar desempenho do modelo. Além disso, pode ser pertinente à otimização do código python visando tornar o esforço computacional menor na execução do modelo e na busca por melhores hiperparâmetros e consequentemente menor tempo.

Ademais, de forma complementar ao trabalho aqui apresentado, para a obtenção da matriz de prioridades, além da possibilidade de a obtermos de acordo com a análise de um especialista, pode ser possível, mediante a testes, à utilização de modelo de Redes Neurais Convolucionais para obtê-la a partir de imagens de satélite de forma a classificar determinado quadrante em relação a sua prioridade.

Referência

[1] Sudharsan Ravichandiran, “Hands-On Reinforcement Learning with Python”, 2018, Birmingham, United Kingdom, 2018