# 

# 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

Rio de Janeiro, XX de junho de 2023

1. **Resumo**

Calamidades sempre ocorreram no mundo e, muitas vezes, a improbabilidade de ocorrerem, as torna de difícil previsão e precaução. Sendo assim, busca-se o melhor comportamento e planejamento diante de um desastre, de forma a maximizar a quantidade de sobreviventes e minimizar o tempo de exposição ao perigo ao qual as vítimas estão sendo submetidas. O objetivo deste trabalho é otimizar o planejamento do resgate de vítimas, concluindo a busca com a menor quantidade de passos, dado um cenário em que conhecemos as prioridades de cada subespaço em nosso espaço de busca. 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* usa o framework matemático de *Markov Decision Process (MDP)*, em que estados futuros dependem apenas do estado presente. Pode ser representado com cinco elementos importantes: (): Estados que o agente pode entrar, (): conjunto de ações que agente pode tomar, (): probabilidade de agente se mover do estado para estado por performar ação e ): 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 o trabalho aqui considerado 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). Como o método de Monte Carlo, não requer dinâmica de modelo e, como DP, não requer esperar 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*(), a cada passo do agente no ambiente, e ao final, encontrar os pares de estado ação que levarão a maximização da recompensa final. [1]

Além do principal objetivo e da aplicabilidade direta na área de calamidades, o objetivo é que a solução proposta possa ser aplicada a qualquer outro tipo de problema que envolva o deslocamento de um agente em um espaço condicionado por pontos previamente estabelecidos como prioritários, com menor quantidade de passos dados.

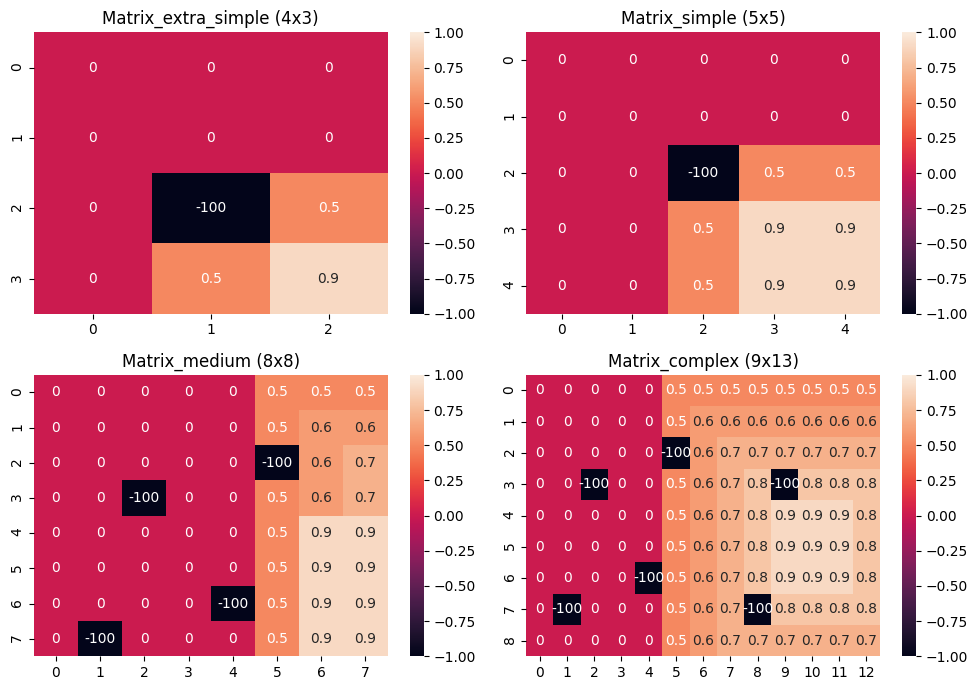
1. **Problema e Premissas**

Cada subespaço do nosso espaço de busca será subdividido em quadrantes, onde cada quadrante terá uma prioridade pré-definida. 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 considerado uma matriz de prioridades onde temos um epicentro da catástrofe com maior priorização e, quanto mais afastado deste epicentro, menores serão as prioridades.

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

1. **‘matriz muito simples’ (matriz 4x3);**
2. **‘matriz simples’ (5x5);**
3. **‘matriz média’ (8x8);**
4. **‘matriz complexa’ (9x13).**



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

1. **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, com menor quantidade de passos, percorrer todos os quadrantes com valores maiores que zero, privilegiando os subespaços com maiores valores de prioridade.

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

. ()

Onde:

Podemos definir as variáveis acima como:

Valor atualizado da Tabela Q para o estado e a ação

*learning rate*

Valor da Tabela Q no instante t para o estado e a ação

Recompensa no instante t para o estado , ação e deflator temporal

Estado no instante t

Ação no instante t

Deflator temporal no instante t

*Discount factor*

Estimativa do valor da tabela Q para o próximo estado

Quantidade de estados visitados Estados visitados

A variável de deflação temporal (d\_f), desenvolvida especificamente neste trabalho e que não faz parte da modelagem clássica, é responsável por reduzir as recompensas futuras de acordo com a quantidade de passos dados.

Além disso, podemos destacar que a ação é escolhida conforme épsilon atual, que segue a seguinte função para cada episódio.

Onde:

*Exploration rate*

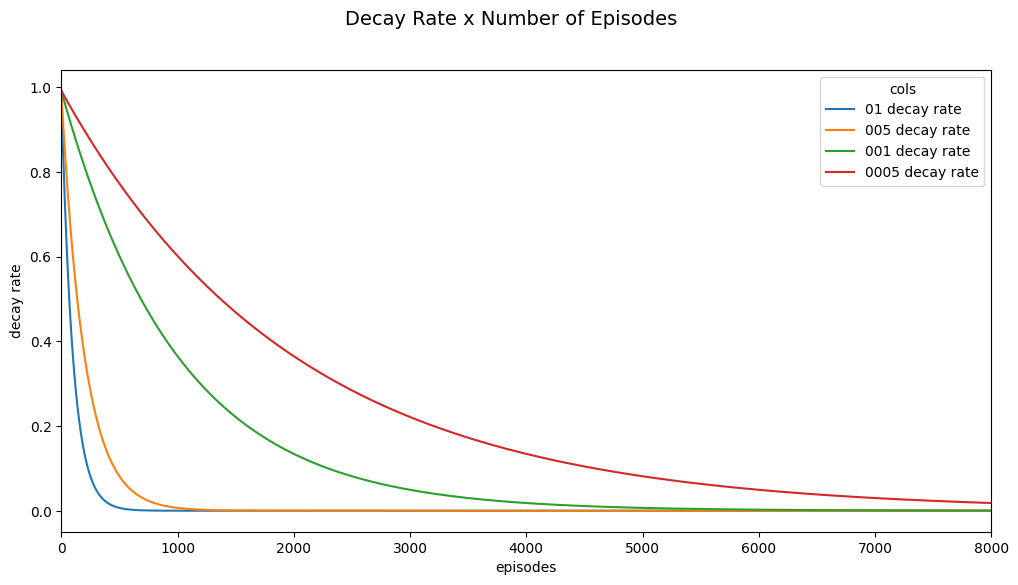
*Exploration rate(mínimo)*

*Exploration rate(máximo)*

*Fator de queda do exploration rate a cada episódio*

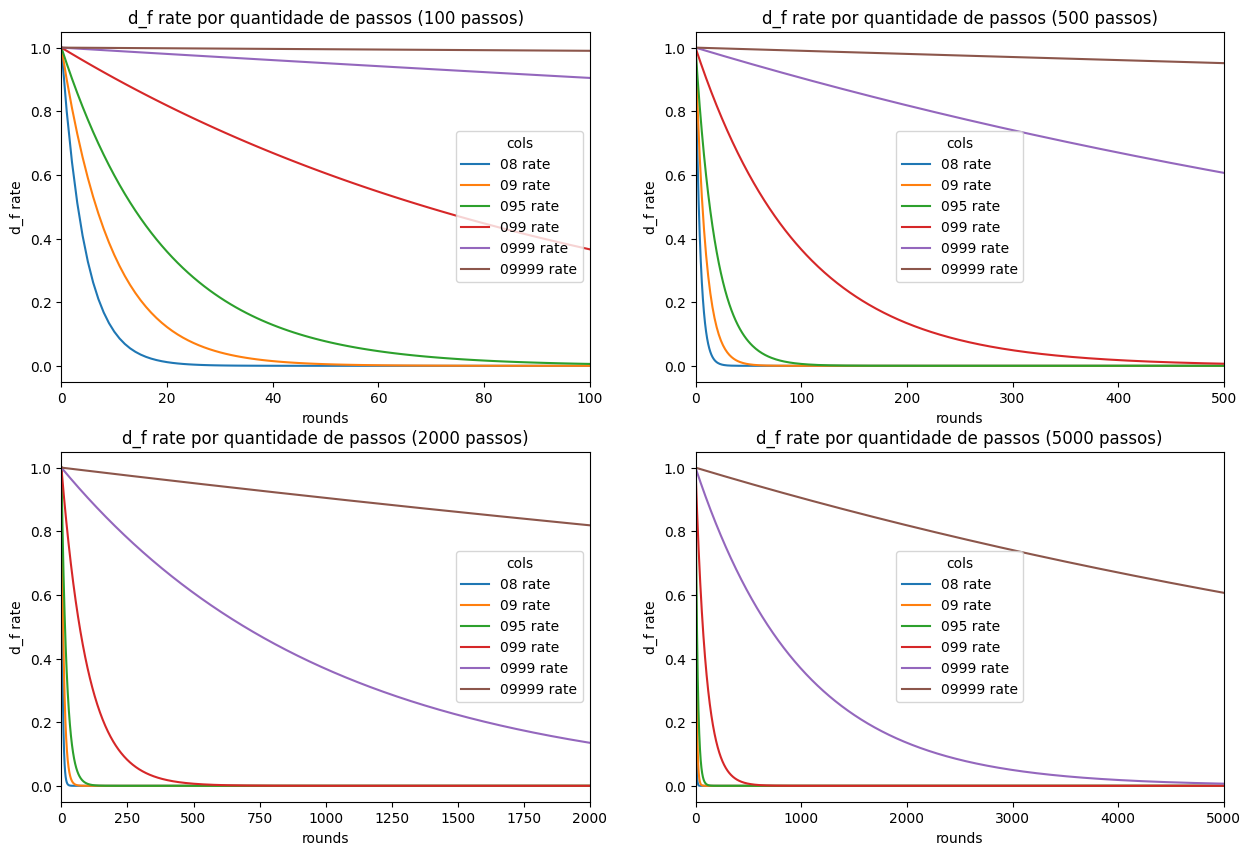
Quantidade de episódios jogados em cada configuração

Quanto menor o épsilon, maior a chance de agente escolher ação que o traga melhores recompensas e quanto maior o épsilon, maior a chance de agente escolher ação aleatória.



**Figura III.** Diferentes valores de *Épsilon* por episódio.

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. Inicialmente foi identificando, para cada cenário, o d\_f ideal dentro do seguinte espaço de busca.



**Figura II.** Referente adiferentes valores de d\_f por quantidades de rounds jogados

Cada configuração a seguir teve o modelo rodado 10 vezes para obtenção de resultados:

* + 1. ‘matriz muito simples’ (matriz 4x3) -> [0.8, 0.9, 0.95, 0.99, 0.999, 0.9999];

1. ‘matriz simples’ (5x5) -> [0.9, 0.95, 0.99, 0.999, 0.9999];
2. ‘matriz média’ (8x8) -> [0.95, 0.99, 0.999, 0.9999];
3. ‘matriz complexa’ (9x13) -> [0.95, 0.99, 0.999, 0.9999];

Posteriormente foi realizada uma busca de melhores hiperparâmetros para cada cenário, em que tivemos o modelo rodado 10 vezes para cada configuração, sendo avaliados:

* + 1. **‘matriz muito simples’ (matriz 4x3):**
* Learning rate = [0.01, 0.001, 0.0001]
* Gamma = [0.95, 0.99]
* d\_f = 0.95
* max exp\_rate (epsilon) = 1.0
* min exp\_rate decay (epsilon) = 0.01
* exp\_rate decay (epsilon) = [0.001, 0.005]
* quantidade maxima de passos = 5.000
* quantidade de episódios = 8000
  + 1. **‘matriz simples’ (5x5):**
* Learning rate = [0.01, 0.001, 0.0001]
* Gamma = [0.95, 0.99]
* d\_f = [0.8, 0.9, 0.95, 0.999]
* max exp\_rate (epsilon) = 1.0
* min exp\_rate decay (epsilon) = 0.01
* exp\_rate decay (epsilon) = [0.001, 0.005]
* quantidade maxima de passos = 10.000
* quantidade de episódios = 8000
  + 1. **‘matriz média’ (8x8):**
* Learning rate = [0.01, 0.001, 0.0001]
* Gamma = [0.95, 0.99]
* d\_f = [0.8, 0.9, 0.95, 0.999]
* max exp\_rate (epsilon) = 1.0
* min exp\_rate decay (epsilon) = 0.01
* exp\_rate decay (epsilon) = [0.001, 0.005]
* quantidade maxima de passos = 10.000
* quantidade de episódios = 8000
  + 1. **‘matriz simples’ (5x5):**
* Learning rate = [0.01, 0.001, 0.0001]
* Gamma = [0.95, 0.99]
* d\_f = [0.8, 0.9, 0.95, 0.999]
* max exp\_rate (epsilon) = 1.0
* min exp\_rate decay (epsilon) = 0.01
* exp\_rate decay (epsilon) = [0.001, 0.005]
* quantidade maxima de passos = 10.000
* quantidade de episódios = 8000
  + 1. **‘matriz complexa’ (9x13)**
* Learning rate = [0.01, 0.001, 0.0001]
* Gamma = [0.95, 0.99]
* d\_f = [0.8, 0.9, 0.95, 0.999]
* max exp\_rate (epsilon) = 1.0
* min exp\_rate decay (epsilon) = 0.01
* exp\_rate decay (epsilon) = [0.001, 0.005]
* quantidade maxima de passos = 10.000
* quantidade de episódios = 8000

1. **Resultados**

Os resultados podem ser divididos para cada problema estudado. Para os cenários i e ii , foi avaliado melhores recompensas obtidas no final do treinamento para cada cenário estudado conforme busca realizada conforme descrito acima.

1. **Resultado Matriz Muito Simples (4x3)**

(Ordenação com melhores configurações)

(Gráfico comparativo com melhores configurações e intervalo de confiança)

(Giff do jogo jogado conforme matriz v table obtida)

(solução e recompensa obtida para resultado)

1. **Resultado Matriz Simples (5x5)**

(Ordenação com melhores configurações)

(Gráfico comparativo com melhores configurações e intervalo de confiança)

(Giff do jogo jogado conforme matriz v table obtida)

(solução e recompensa obtida para resultado)

1. **Resultado Matriz Média Complexidade (8x8)**

(Ordenação com melhores configurações)

(Gráfico comparativo com melhores configurações e intervalo de confiança)

(Giff do jogo jogado conforme matriz v table obtida)

(solução e recompensa obtida para resultado)

1. **Resultado Matriz Complexa (13x9)**

(Ordenação com melhores configurações)

(Gráfico comparativo com melhores configurações e intervalo de confiança)

(Giff do jogo jogado conforme matriz v table obtida)

(solução e recompensa obtida para resultado)

1. **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. Também foi identificado que para cenários mais complexos temos aumento substancial do tempo de processamento. 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.

Além disso, de forma complementar ao trabalho aqui estudado, para obtenção da matriz de prioridades, além da possibilidade de a obtermos de acordo com a análise de um especialista, é possível a 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. Essa solução seria complementar ao trabalho aqui apresentado, que tem a matriz de prioridades como premissa.

**Referência**

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