Danilo Issy Hirabara
Jordana C. B. A. Carnicelli
Laura Takako Kohatsu
Rodrigo Rossi dos Santos

# Exercício Programa 2

Tópicos em Planejamento em Inteligência Artificial - ACH2137
Profª Drª Karina Valdivia Delgado

Universidade de São Paulo - USP
Escola de Artes, Ciências e Humanidades
Bacharelado em Sistemas de Informação

São Paulo
2020

# ÍNDICE

# INTRODUÇÃO

## Fundamentos teóricos

O trabalho realizado aborda algoritmos simples (sem informação) de planejamento probabilístico, que é uma extensão do planejamento não-determinístico. Nessa categoria de planejamento trabalhamos com ambientes estáticos e completamente observáveis, percepções perfeitas e ações instantâneas e estocásticas.

Enquanto no planejamento clássico o objetivo final era identificar um plano (sequência de ações), no planejamento probabilístico deseja-se definir qual a melhor política, ou seja, o melhor mapeamento de estados em ações, a partir da informação das probabilidades dos eventos não-determinísticos.

As probabilidades são importantes para quantificar os custos e as probabilidades de sucesso dos planos quando as ações são não-determinísticas. Entretanto, não basta apenas ter um plano. É importante ter um plano eficiente, tal que o custo das ações não exceda as recompensas obtidas ao atingir as metas. Em alguns casos, não há a garantia de uma plano que atinja as metas. Nesses casos, é importante maximizar a probabilidade de atingir as metas. Por isso, é importante usar informações sobre as probabilidades de diferentes efeitos de operadores.

Para trabalhar com os problemas de planejamento probabilístico utilizamos um modelo matemático chamado processo de decisão markoviano, MDP. Existem 3 tipos de MDPs:

- De horizonte finito;
- De horizonte infinito de recompensa com desconto;
- **De caminho mais curto estocástico**.

O último tipo é o que utilizamos neste trabalho. Trata-se do MDP cuja característica é a ausência de informação sobre a quantidade de ações a serem tomadas, porém o conhecimento de que essa quantidade é finita, uma vez que quando um estado meta é atingido, nenhuma outra ação é realizada.

Um MDP de caminho estocástico mais curto, ou SSP MDP, pode ser definido com uma tupla <S, A, T, C, G>, onde:

- S é o conjunto finito de estados;
- A é o conjunto de ações;
- T é a função de probabilidade de transição;
- C é a função de custo;
- G é o conjunto de estados meta (um estado meta apresenta custo 0).

A solução para um SSP MDP é uma política: π: S → A . Existem duas condições básicas para que exista uma política para um MDP:

1. A política deve ser própria, ou seja, será possível chegar na meta a partir de qualquer estado;
2. Se existir alguma política imprópria, o custo para a sua execução a partir de qualquer estado será infinito.

Para compreender o funcionamento dos algoritmos a serem citados é necessário entender primeiro alguns conceitos importantes:

- **Valor de uma política π:** representado pela notação $V^{\pi}$ é o custo de alcançar a meta utilizando a política π.
- **Valor da política π no estado s:** representado pela notação $V^{\pi}(s)$ é o custo de alcançar a meta a partir do estado s utilizando a política π.
- **Avaliação de política:** há duas maneiras de se calcular o valor de uma política π para um estado s, por sistemas lineares ou iterativamente. Em ambos casos precisa-se utilizar a seguinte equação:

$$V^{\pi}(s) = 0 \text{ , } se\ s \in G\ \ ou\ \ V^{\pi}(s) = \square\ T\ (\ C\ +\ V^{\pi}(s')\ )$$

s’ representa o próximo estado, ou seja, o estado resultante da aplicação da ação. A resolução por sistemas lineares implica montar o sistema a partir da aplicação da equação acima para todos os estados. Já no modo iterativo, os valores recebem inicialmente números aleatórios e a partir deles, calcula-se a equação. O processo é repetido diversas vezes, até que os valores encontrados comecem a convergir.

- **Valor ótimo de uma política no estado s:** representado pela notação V*(s) é o custo ótimo de alcançar a meta a partir do estado s, ou seja, o menor custo. Para calcular V*(s) deve-se selecionar a ação de menor custo:

$$V^*(s) = 0 \text{ , } se\ s \in G \ \ ou \ \ V^*(s) = min \square\ T\ (\ C\ +\ V^{\pi}(s')\ )$$

- **Qualidade ótima de aplicar a ação a no estado s:** representada pela notação Q*(s, a) é o custo esperado ótimo se a primeira ação a ser tomada é a no estado s, e depois mantém-se de maneira ótima, ou seja, usando apenas os valores ótimos dos próximos estados. V*(s) é obtido a partir de Q*(s, a), conforme pode-se reescrever a equação acima:

$$V^*(s) = 0 \text{ , } se\ s \in G \ \ ou \ \ V^*(s) = min\ Q^*(s,a)$$

- **Bellman backup:** a partir dos conceitos acima, pode-se pensar em calcular o valor ótimo de uma política para todos os estados de um SSP MDP. Para realizar essa tarefa são geradas várias iterações representadas pelo intervalo [0,n], sendo que para cada iteração n calcula-se o $Qn\ (s,a)$ e depois o menor valor é selecionado. Isso significa que na iteração 1, utilizam-se os valores obtidos na iteração 0, e assim por diante. Esse é o chamado algoritmo de Bellman backup:

$$Vn(s) = min \square\ T\ (\ C\ +\ Vn-1(s')\ )$$

Uma vez compreendidos os conceitos acima, é possível introduzir os dois algoritmos simples, ou seja, sem informação, que podem ser utilizados para definir uma política para um SSP MDP:

1. **Iteração de valor:** Realiza busca no espaço - infinito - de valores, e no final, identifica a política. Esse algoritmo pode ser traduzido da seguinte forma: primeiro ocorre a inicialização arbitrária do valor de cada estado para a iteração 0 (V0). Depois, em um laço, calcula-se o valor de cada estado para a iteração atual utilizando o algoritmo de Bellman backup e computa-se o valor residual - o módulo da diferença entre o valor do estado s na iteração atual e na iteração imediatamente anterior. Esse processo é realizado até que o valor residual máximo seja menor do que um valor ε definido previamente. Por fim, é necessário identificar a política responsável por aqueles valores, e para tal, utiliza-se uma função similar ao bellman backup, com a diferença

que ela retorna o *argmin*, ou seja, a ação que gera o valor mínimo encontrado, para cada um dos estados.

2. **Iteração de política:** realiza busca no espaço - finito - de políticas e no final encontra o valor da política. Em cada iteração, há duas partes, a primeira é a Avaliação de Política, nela a partir das ações da política própria, calculamos o custo de cada estado até a meta, esses valores são armazenados em uma tabela para que seja usada no próximo passo. A segunda parte é a Melhoria da Política, a qual para cada estado, calculamos o custo até a meta usando todas ações, e depois pegamos a de menor custo e armazenamos para que assim forme a nova política própria. Iremos rodar isso até que a política própria da iteração anterior seja igual a atual.

# IMPLEMENTAÇÃO

## Contextualização e Desenvolvimento

O domínio do problema a ser resolvido com o uso de MDP é o gridworld onde um agente deve percorrer o mapa até chegar ao seu objetivo definido no arquivo de configuração do problema, as ações disponíveis são movimentos nas quatro direções: norte, sul, leste e oeste.

Desenvolvemos um script em Python que pode ser executado conforme o arquivo README. Organizamos o código conforme a sequência:

1. Transformação do arquivo de domínio para o formato especificado abaixo;
2. Execução do algoritmo de iteração de valor;
3. Execução do algoritmo de iteração de política modificada.

Criamos uma única estrutura (um dicionário) contendo tudo o que for relativo a um determinado estado. Iterando o dicionário têm-se todas as informações relativas aos estados, conforme o exemplo:

```
'robot-at-x12y18': [
    ['move-south', '1.000000', 'robot-at-x12y17', '0.500000', 'robot-at-x12y18', '0.500000'],
    ['move-north', '1.000000', 'robot-at-x12y19', '0.500000', 'robot-at-x12y18', '0.500000'],
    ['move-west', '1.000000', 'robot-at-x11y18', '0.500000', 'robot-at-x12y18', '0.500000'],
    ['move-east', '1.000000', 'robot-at-x13y18', '0.500000', 'robot-at-x12y18', '0.500000']
]
```

Para esse estado podemos aplicar as ações move-south, move-north, move-west e move-east. Os valores ao lado de cada ação representam os custos delas, e logo em seguida os estados resultantes e probabilidades. De maneira simplificada, a representação gerada é a seguinte:

*[ação, custoDessaAção, próximoEstadoX, probabilidadeX, próximoEstadoY, probabilidadeY, …]*

A implementação do algoritmo de iteração de valor foi realizada adaptando-se o texto descrito anteriormente. Já na iteração de política modificada realizamos uma pequena alteração para facilitar a implementação, porém sem prejuízo dos resultados:

a. No laço, ao invés de iniciarmos com a avaliação de política e depois realizarmos a melhoria da política, invertemos os papeis, pois na iteração n=1, que é realizada antes do laço, nós já fizemos o passo de avaliação da política.

Os principais métodos do código são:

1. **Bellman_backup:** utilizado para calcular o Qn(s,a). É criada uma lista que é preenchida com o valor Q durante o laço no qual são verificadas todas as ações possíveis para um dado estado. No final, o método retorna o menor valor ou a melhor ação encontrada, a depender dos parâmetros informados.
2. **Value_iteration:** utilizado para coordenar todas as etapas do algoritmo de iteração de valor. No início todos os valores são inicializados com números aleatórios, depois entra-se no laço principal no qual para cada estado é chamado o método bellman_backup com retorno do menor valor, calcula-se o residual e atualizam-se os valores para a iteração n. Esse laço é encerrado quando o valor máximo residual é menor do que o epsilon (0,1) e então para cada estado é chamado novamente o bellman_backup, porém para retornar a melhor ação a partir dos valores da última iteração n.
3. **Iterative_policy_evaluation:** utilizado para coordenar todas as etapas do algoritmo de iteração de política. No início todos os valores são inicializados com números aleatórios e na primeira iteração realiza-se a avaliação da política e atualizam-se os valores para n = 1. Depois, no laço principal, para cada estado é chamado o método bellman_backup com retorno da melhor ação, - passo da melhoria de política - atualiza-se a política atual com a ação recebida e calcula-se $Vn^{\pi}(s)$ - passo da avaliação de política. Esse laço é encerrado quando a política atual for igual à anterior.
4. **Policy_evaluation:** utilizado para calcular a avaliação de uma política, a partir de uma aproximação do valor de $\pi n$ .

# Testes

Para checar se o código realizava as operações corretamente, modelamos o problema do slide 68 da seguinte forma:

**Ideia:**

| | x0 | x1 | x2 |
|---|---|---|---|
| y2 | SG | S4 | S3 |
| y1 | S2 | | * |
| y0 | S0 | S1 | * |

**Grid:**

| | x0 | x1 | x2 |
|---|---|---|---|
| y2 | robot-at-x0y2 | robot-at-x1y2 | robot-at-x2y2 |
| y1 | robot-at-x0y1 | * | * |
| y0 | robot-at-x0y0 | robot-at-x1y0 | * |

| Estados: | Ações: |
|---|---|
| S0 | a00 = move-north |
| | a01 = move-east |
| S1 | a1 = move-north |
| S2 | a20 = move-north |
| | a21 = move-south |
| S3 | a3 = move-west |
| S4 | a40 = move-west |
| | a41 = move-east |

Segue comparação entre os resultados esperados e obtidos:

**Valores esperados:**

| | x0 | x1 | x2 |
|---|---|---|---|
| y2 | 0 | ~4 | ~5 |
| y1 | ~5 | | * |
| y0 | ~6 | ~6 | * |

**Valores obtidos:**

| | x0 | x1 | x2 |
|---|---|---|---|
| y2 | 0 | 4,0256 | 5,0102 |
| y1 | 5,01024 | | * |
| y0 | 6,0640 | 6,0640 | * |

**Política esperada:**

| | x0 | x1 | x2 |
|---|---|---|---|
| y2 | G | move-east | move-west |
| y1 | move-north | | * |
| y0 | move-north | move-north | * |

**Política obtida:**

| | x0 | x1 | x2 |
|---|---|---|---|
| y2 | G | move-east | move-west |
| y1 | move-north | | * |
| y0 | move-north | move-north | * |

Desse modo, pode-se notar que a modelagem foi bem-sucedida, bem como a tradução dos algoritmos para código. Outra questão que observamos, foi a eficiência. Conforme era esperado, o algoritmo de iteração de valor levou mais iterações do que o de iteração de política para chegar ao resultado final. A partir disso, começamos então a registrar as demais execuções necessárias e a analisar os resultados.

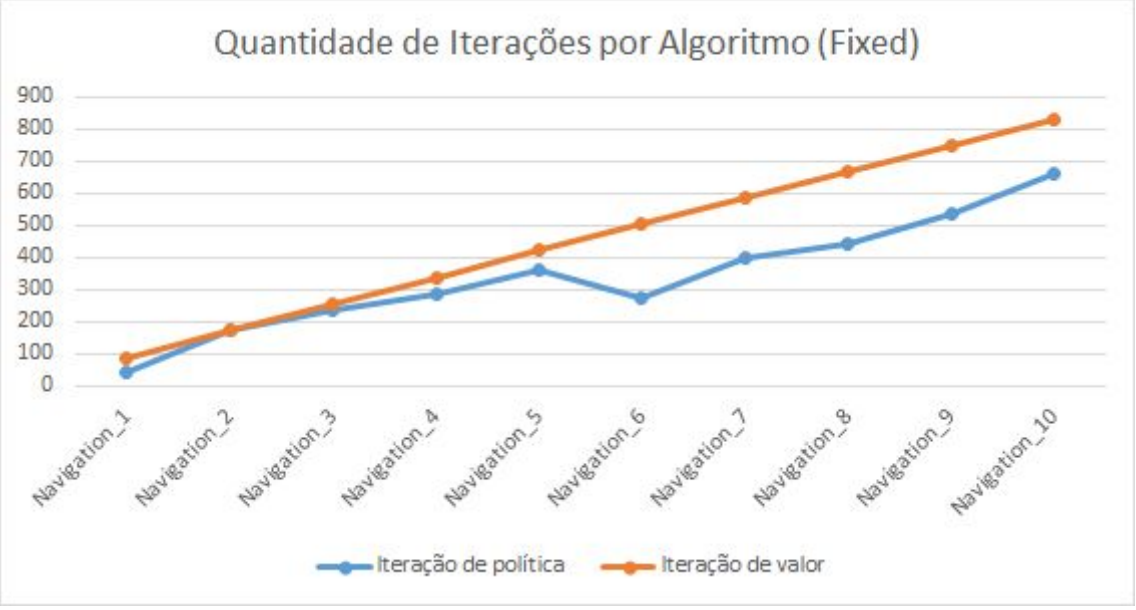
## EXPERIMENTOS E RESULTADOS

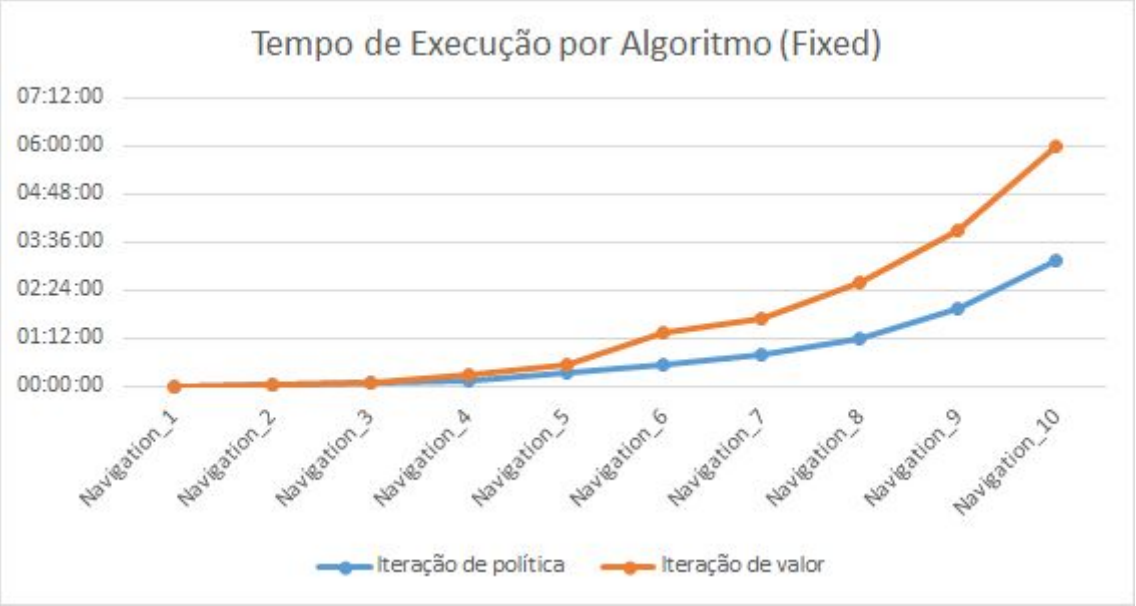
# Experimentos

Executamos todos os problemas disponibilizados, compilamos e analisamos os resultados. Para os problemas com estados iniciais e meta definidos, os resultados obtidos foram:

| Fixed Goal Initial State | Algoritmo | Iterações | Tempo de execução (s) | Tempo de execução |
|---|---|---|---|---|
| Navigation_1 | Iteração de política | 44 | 5,89 | 00:00:06 |
| Navigation_2 | Iteração de política | 173 | 90,42 | 00:01:30 |
| Navigation_3 | Iteração de política | 235 | 278,42 | 00:04:38 |
| Navigation_4 | Iteração de política | 290 | 600,65 | 00:10:01 |
| Navigation_5 | Iteração de política | 361 | 1218,21 | 00:20:18 |
| Navigation_6 | Iteração de política | 275 | 1883,64 | 00:31:24 |
| Navigation_7 | Iteração de política | 397 | 2905,90 | 00:48:26 |
| Navigation_8 | Iteração de política | 446 | 4348,67 | 01:12:29 |
| Navigation_9 | Iteração de política | 537 | 6987,51 | 01:56:28 |
| Navigation_10 | Iteração de política | 659 | 11410,99 | 03:10:11 |
| Navigation_1 | Iteração de valor | 88 | 14,80 | 00:00:15 |
| Navigation_2 | Iteração de valor | 173 | 142,91 | 00:02:23 |
| Navigation_3 | Iteração de valor | 257 | 406,41 | 00:06:46 |
| Navigation_4 | Iteração de valor | 340 | 997,79 | 00:16:38 |
| Navigation_5 | Iteração de valor | 423 | 1988,99 | 00:33:09 |
| Navigation_6 | Iteração de valor | 505 | 4818,79 | 01:20:19 |
| Navigation_7 | Iteração de valor | 587 | 6115,41 | 01:41:55 |
| Navigation_8 | Iteração de valor | 670 | 9293,51 | 02:34:54 |
| Navigation_9 | Iteração de valor | 751 | 13999,94 | 03:53:20 |
| Navigation_10 | Iteração de valor | 833 | 21578,28 | 05:59:38 |

Comparando-se as quantidades de iterações e os tempos de execução, temos as seguintes representações gráficas:

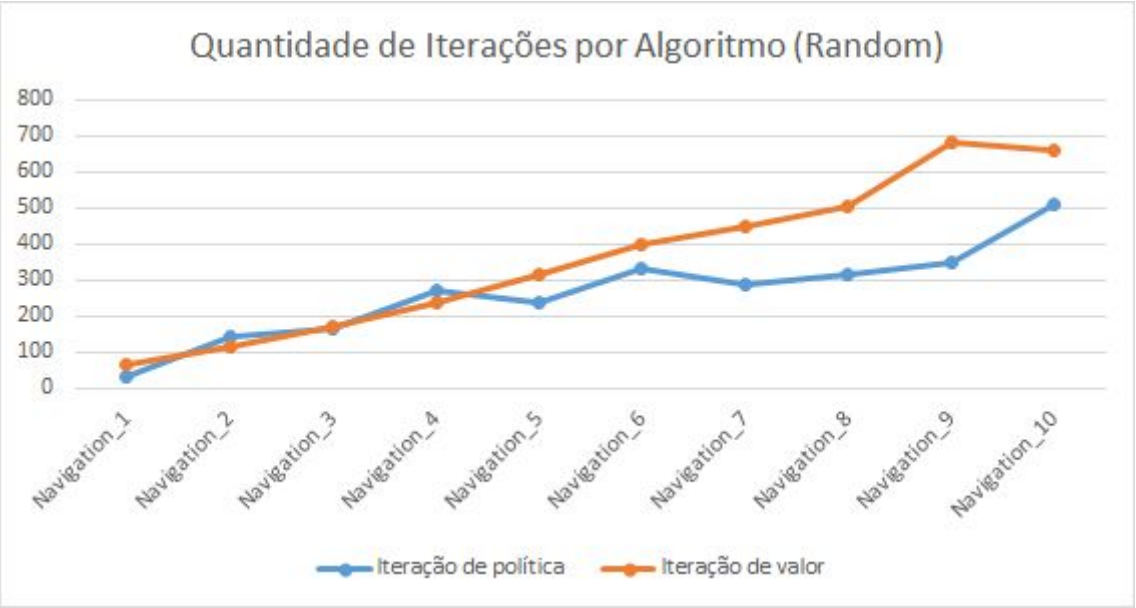


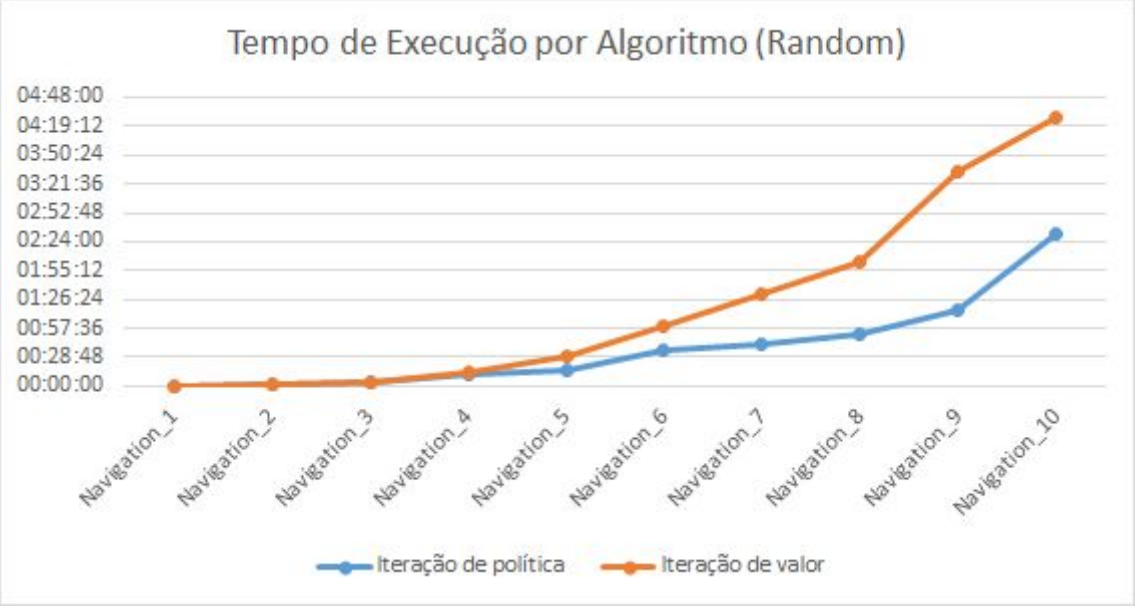



Para os problemas com estados iniciais e meta aleatórios, os resultados obtidos foram:

| Random Goal Initial State | Algoritmo | Iterações | Tempo de execução (s) | Tempo de execução |
|---|---|---|---|---|
| Navigation_1 | Iteração de política | 33 | 5,16 | 00:00:05 |
| Navigation_2 | Iteração de política | 143 | 89,84 | 00:01:30 |
| Navigation_3 | Iteração de política | 167 | 189,63 | 00:03:10 |
| Navigation_4 | Iteração de política | 272 | 731,65 | 00:12:12 |
| Navigation_5 | Iteração de política | 239 | 994,73 | 00:16:35 |
| Navigation_6 | Iteração de política | 333 | 2098,49 | 00:34:58 |
| Navigation_7 | Iteração de política | 287 | 2505,96 | 00:41:46 |
| Navigation_8 | Iteração de política | 318 | 3102,44 | 00:51:42 |
| Navigation_9 | Iteração de política | 351 | 4527,81 | 01:15:28 |
| Navigation_10 | Iteração de política | 509 | 9091,00 | 02:31:31 |
| Navigation_1 | Iteração de valor | 66 | 14,07 | 00:00:14 |
| Navigation_2 | Iteração de valor | 116 | 102,39 | 00:01:42 |
| Navigation_3 | Iteração de valor | 173 | 278,80 | 00:04:39 |
| Navigation_4 | Iteração de valor | 240 | 864,25 | 00:14:24 |
| Navigation_5 | Iteração de valor | 315 | 1800,20 | 00:30:00 |
| Navigation_6 | Iteração de valor | 402 | 3536,61 | 00:58:57 |
| Navigation_7 | Iteração de valor | 451 | 5475,35 | 01:31:15 |
| Navigation_8 | Iteração de valor | 505 | 7422,28 | 02:03:42 |
| Navigation_9 | Iteração de valor | 682 | 12828,94 | 03:33:49 |
| Navigation_10 | Iteração de valor | 659 | 16054,94 | 04:27:35 |

Comparando-se as quantidades de iterações e os tempos de execução, temos as seguintes representações gráficas:




## Conclusões

Como era esperado, o algoritmo de iteração de política se mostrou, no geral mais eficiente que o de iteração de valor, tanto em quantidade de iterações, quanto em tempo de execução. Esse fator era esperado por conta do espaço de busca de ambos. Enquanto o iteração de valor trabalha com números reais, e, portanto, em um intervalo infinito, o iteração de política trabalha com políticas, representadas por um conjunto finito. Desse modo, era esperada uma conversão mais rápida e com menos passos por parte da iteração de política.

O algoritmo de iteração de valor apresenta gráficos mais comportados, demonstrando crescimento linear na quantidade de iterações de acordo com o crescimento do tamanho dos problemas e crescimento exponencial de tempo. O algoritmo de iteração de

política apresenta padrões menos constantes, porém fica evidente que também cresce linearmente em quantidade de iterações e exponencialmente em tempo. Além disso, para os problemas menores ambos algoritmos apresentam comportamentos similares, porém na medida em que trabalhamos com grids maiores, eles tendem a se separar em intervalos cada vez maiores, sendo que o algoritmo de iteração de política tem crescimento, tanto para iterações, quanto de tempo, menores do que o de iteração de valor.

Existem dois fatores principais que podem ter afetado o desempenho dos algoritmos:

1. A estrutura auxiliar criada para a representação do contexto (do grid);
2. A utilização de dataframes do Pandas, pois essa estrutura de dados é ótima para a realização de operações em tabelas (linhas e colunas), entretanto é ineficiente para atualização individual de valores por posição. Essa escolha foi realizada baseada nos conhecimentos prévios da equipe, possibilitando maior facilidade na codificação dos algoritmos e mais tempo para realizar ajustes e correções.

Em suma, conclui-se que os algoritmos utilizados foram eficazes, porém, para os problemas maiores não foram tão eficientes. A primeira opção seria refatoração do código para otimizá-lo, eliminando, por exemplo, o dataframe do pandas e o substituindo por outras estruturas de dados mais eficientes para as operações realizadas. Todavia, um algoritmo como o assíncrono VI ou algum outro do grupo de algoritmos priorizados teria desempenho melhor para esses problemas, exatamente por não atualizar os valores dos estados em todas as iterações.