# Value Interation, LAO* e ILAO* como busca em um espaço de transição de estados

**Author: Marcelo Sabaris Carballo Pinto**

**Abstract**

Neste artigo mostraremos como problema, a escolha do melhor caminho para travessia de um rio, aonde seu caminho como ser modelado naturalmente como um problema de caminho mais curto. Apresentaremos algoritmos de heurística para encontrar os caminhos mais curtos, testando suas implementações em cenários distintos.

## Introdução

O termo busca está relacionado em encontrar soluções em um espaço de possíveis soluções , por fornecerem um arcabouço geral para solução de problemas , os métodos de buscas são bastantes empregados na construção de agentes em Inteligência artificial .Um agente inteligente que pretende resolver um problema, pode aplicar um processo de busca , em um espaço de possíveis soluções e encontrar a mais desejada .

O caso mais geral de busca, assume que o espaço de possíveis soluções é dado por um modelo baseado em transições e estados, na qual se deseja encontrar um caminho mais curto entre um estado inicial e final .

As classe de problemas de caminho mais curto (ssp) é um subconjunto de processos de decisão de Markov (mdps) que tem uma importância central para a IA são: generalização natural do modelo de pesquisa clássico para o caso de transições estocásticas e funções gerais de custo Ssps, na qual havia sido usado recentemente para modelar uma ampla gama de problemas decorrentes da navegação, controle de robôs , sistemas não determinísticos para jogo estocástico e planejamento sob incerteza e informações parciais (Bertsekas & Tsitsiklis 1996; Sutton & Barto 1998;Bonet e Geffner 2000).

A teoria dos mdps recebeu grande atenção da comunidade da IA por três razões importantes. Primeiro, fornece uma estrutura fácil para modelar problemas complexos da vida real que têm grande espaço de estado (mesmo infinito), dinâmica complexa e funções de custo. Segundo os mdps fornecem fundamentos matemáticos para o aprendizado desenvolvido independentemente dos algoritmos no aprendizado por reforço. E terceiro, algoritmos gerais e eficientes para resolver mdps foram desenvolvidos, sendo o mais importante a Iteração de Valor e Iteração de Política.

Como o próprio nome sugeri, Markov Decision Process (MPD) é um modelo sistêmico estocástico com custos ou recompensas associados a estados e escolha de ações. Uma solução para um MDP é uma estratégia que leva a meta com custo mínimo ou recompensa máxima esperado , muitas vezes, estamos interessados apenas em como chegar à meta de um estado inicial fixo em vez de conhecer a solução geral; a razão é que o espaço de estados geralmente contém muitos estados que são irrelevante. Programação dinâmica em tempo real (rtdp) (Barto, Bradtke e Singh 1995) é um algoritmo para encontrar tais soluções parciais. No entanto, o rtdp é um algoritmo probabilístico que apenas converge assintoticamente portanto, embora tenha havido resultados experimentais mostrando que o rtdp converge mais rapidamente que outros algoritmos, ele não pode ser usado como um algoritmo off-line.

A contribuição deste trabalho, se divide em três etapas. Primeiramente apresentaremos o processo markovianao de decisão , que descreve métodos gerais para solução de problemas , em particular explicaremos o value interation, seus principais parâmetros de entrada, seu algoritmo propriamente dito e testes que foram realizados para resolução do problema da travessia do rio , na qual foi dividido em 3 instancias 5X25, 20X100 , 50X250, apresentado no início do artigo. Segundo, falaremos sobre algoritmos de busca, o LAO* mostrando sua implementação e testes realizados no problema do rio e por último, apresentaremos a implementação do ILAO* que seria uma variação do LAO* , com seus devidos testes .

### Processo Markoviano de decisão

Os processos de decisão Markovianos são modelos usados para planejamento probabilístico em que as ações selecionadas por um agente causam mudanças no

comportamento do sistema, estado atual do sistema e ação escolhida para ser executada gerando uma recompensa ou custo e determinam uma distribuição de probabilidades para o sistema .
A dinâmica do sistema é governada por funções probabilísticas de transição que mapeiam estados e controles para estados, na qual o agente incorre em um custo ou recompensa. Assim, a tarefa é encontrar uma estratégia de controle (política) que minimiza o custo total esperado ao longo do tempo. Formalmente, um mdp é definido por:
(M1) O espaço de estado S = {1 ,. . . , n}
(M2) Estado Inicial: S0
M(3) Possibilidades de ações A = {a0,a1....,an}
M(4) Modelo de Transição T (s,a,s')
M(5) Função de recompensa R(s)
O modelo de transição retorna à probabilidade de atingir o estado s' se a ação $a$ uma é feita no estado s' , dado que s e a são condicionalmente independentes de todos os estados e ações anteriores (propriedade de Markov). A função de R recompensa , sempre retorna um valor real toda vez que o agente se move de um estado para outro, como temos uma função de recompensa, podemos dizer que alguns estados são mais desejáveis que outros porque, quando o agente se move nesses estados, recebe uma recompensa maior ou um custo menor, dependendo do contexto do problema modelado . Pelo contrário, existem estados que não são desejáveis, porque quando o agente se move para lá recebe uma recompensa negativa ou um custo maior.
O problema que o agente tem para maximizar a recompensa, evitando estados que retornam valores negativos e escolhendo aquele que retorna valores positivos, para resolução de tal problema damos o nome de política $\pi*$ , que deverá encontrar a melhor estratégia.
Uma estratégia ou política π é uma sequência infinita (μ0, μ1,..) de funções em que μk mapeia estados para controles para que o agente aplique o controle μk (i) em estado xk = i no tempo k; a única restrição é que μk (i) ∈ U (i) para todos os i ∈ S. Se π = (μ, μ,...), a política é chamado estacionário (ou seja, o controle não depende pontualmente) e é simplesmente indicado por μ. O custo associado à política π quando o sistema inicia no estado x0 é definido como:

$$J_\pi(x_0) = \lim_{N\to\infty} E\left\{\sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_k(x_k))\right\} \qquad (1)$$

Para todas as outras políticas π. (embora não possa haver mais do que uma e uma política ideal), o vetor de custo ideal J ∗ é sempre único. A existência de π∗ e como calculá-las são problemas matemáticos não triviais.
No entanto, quando α <1, a política ideal sempre existe e, mais importante, existe uma política estacionária isso é ótima. Nesse caso, J∗ é uma solução única às equações de Bellman Optimality:

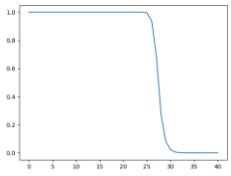
$$J^*(i) = \min_{u\in U(i)} g(i,u) + \alpha \sum_{j=1}^{n} p(i,u,j) J^*(j). \qquad (3)$$

Depois de resolver o J, o algoritmo value interation computa J* interações usando o cálculo descrito no item 3, aonde terá um critério de parada de resido máximo pré-determinado (maxi∈S |Jk+1(i) − Jk(i)| .

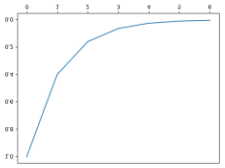
**Algoritmo**
**Input**:
M: MDP.
$\epsilon$: erro.
**Output**: Devolve uma política $\epsilon$-ótima do MDP

```
foreach s∈S do
    V_0(s) = C(s,a)
end
n = 0;
repeat
    n = n + 1;
    foreach s ∈ S do
        foreach a ∈ A do
            Q_n(s,a) = C(s,a) + α Σ_{s'∈S} P(s'|s,a) V(s')
        end
        V_n(s) = min_{a∈A} Q_n(s,a);
        π_n(s) = arg min_{a∈A} Q_n(s,a);
    end
until |V_n(s) - V_{n-1}(s)| < ε(1-γ)/γ; ∀s∈S ;
return π_n;
```

Na execução do algoritmo acima, no problema apresentando neste artigo, consegui encontrar a solução ótima, mas me deparei com os problemas descritos a seguir . Demora e uso excessivo de memória na execução dos cenários com grandes dimensões (800 MB no Visual Studio Code 2020) , problemas na configuração do α, pois ao superestima-lo houve uma demorava na conversão, conforme demonstrado nos gráficos abaixo e dificuldade em achar o E (erro) ideal, pois em cenários com subestimação do erro , o mesmo convergia rápido, mas não achava a solução ideal, já com valores elevados o mesmo demorava para convergir , mas achava a solução ideal.



α maior

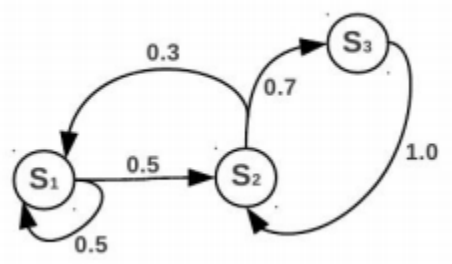


α menor

**Busca heurísticas, LAO***

Nos tópicos acima , vimos que usando o value interation, conseguimos uma política que nos indica uma ação para tomarmos em qualquer estado S. As políticas encontradas pelos métodos de programação dinâmica têm uma limitação prática, a quantidade de memória que precisamos está diretamente ligada ao tamanho dos estados que precisamos (isto acontece pois ele encontra uma política ótima

para cada estado). No entanto dado um estado inicial S0 e um final, podemos orientar o nosso MDP, para encontrar uma política ótima a partir deste ponto . Podemos modelar problemas para serem executados no nosso MDP através de um hipergrafo, por exemplo para um MDP com conjuntos s={s1,s2,s3}, com ações = A , probabilidade de transição P(s1|s1|a1)= 0,5 P(s2|S1,a1) = 0,5 e P(s1|s2,a1) = 0.3 ,P(s3|s2,a1) = 0.7 e P(s2|s3,a1) = 1, a figura abaixo ilustra a representação do mesmo .




Em geral políticas MDP resolvem problemas cíclicos, já o AO* , ele busca soluções ótimas para problemas não cíclicos, não precisando avaliar todos os estados do grafo.

Para correção do problema do AO*, foi se criado o LAO* que é uma extensão do AO* que permite buscar as soluções ótimas , sem ter que expandir todos os nós. . No algoritmo LAO* mantemos as três fases descritas para o algoritmo AO*, isto é, a construção do hipergrafo solução parcial (HGSP), expansão do hipergrafo solução parcial (HGSP) e a atualização de custos. As primeiras duas fases consistem em construir o hipergrafo solução parcial HGSP com as ações gulosas e selecionar um nó folha do HGSP e o expandimos para gerar os nós sucessores. Porém, a terceira fase apresenta uma variação importante. Enquanto o algoritmo AO* atualiza os nós do conjunto Z (composto pelo nó expandido e seus ancestrais) numa ordem topológica bottom-up (somando todos os custos ), o LAO* atualiza os valores dos nós do conjunto Z usando Iteração de Valor ( ou Iteração de Política . Dessa maneira, o algoritmo LAO* generaliza a fase de atualização de custos do AO* usando um algoritmo de programação dinâmica de horizonte infinito para MDPs, para lidar com a presença de ciclos. LAO* constrói um hipergrafo explícito G' que inicialmente só contém o nó inicial. O hipergrafo G' é expandido usando políticas gulosas, até completar o hipergrafo solução (HGS). Quando um hipergrafo está direcionado para um nó já visitado, esse nó sucessor não é adicionado a G'. Durante a construção do HGSP, a ação que minimiza o custo esperado é escolhida e etiquetada como a melhor ação do nó visitado (política gulosa). Cada caminho a partir do nó inicial até os nós folha é um caminho de menor custo esperado (estimado). Segue abaixo algoritmo detalhado do processo .

**Input**:
$G$: Hipergrafo implícito dado pela função de transição probabilística e pelo estado inicial.
$s_0$: Estado inicial.
$C$: Função custo.
$\hat{h}$: Estimativa heurística do custo.
$\epsilon$: erro.
**Output**: Devolve o hipergrafo solução ótimo ou $\epsilon$-ótimo

1 O hipergrafo explícito inicial $G'$ consiste inicialmente do nó inicial $n_0$=$s_0$;
2 **while** O hipergrafo solução parcial tiver algum nó não-terminal **do**
3 | */* Expandir a melhor solução parcial */*;
4 | Selecionar algum nó não-terminal $n_i$ do hipergrafo solução parcial;
5 | Expandir o nó $n_i$ e adicionar os novos nós sucessores a $G'$. Para cada novo nó $n_j$ adicionado a $G'$ pela expansão de $n_i$, se $n_j$ for nó meta então $V(n_j) = 0$, caso contrário $V(n_j) = \hat{h}(n_j)$;
6 | */* Atualização do custo dos nós e etiquetação das melhores hiperarcos (ações) */*;
7 | Criar um conjunto $Z$ que contém o nó expandido e todos seus ancestrais em $G'$ seguindo os hiperarcos etiquetados, isto é, $Z$ contém aqueles nós ancestrais que podem alcançar o nó expandido seguindo a melhor solução atual;
8 | Executar *Iteração de Política* ou *Iteração de Valor* sobre os nós do conjunto $Z$ para atualizar os custos e determinar o melhor hiperarco para cada nó;
9 | Reconstruir $G'$;
10 **end**
11 */* Teste de convergência */*;
12 **if** *Iteração de Política* foi usada **then**
13 | Ir para a linha 21;
14 **end**
15 **else**
16 | Executar *Iteração de Valor* sobre os nós do hipergrafo solução parcial;
17 | Continuar até que alguma das seguintes condições seja alcançada;
18 | | **i**) Se o erro for menor do que $\epsilon$, para todos os nós $n_i$ de $G'$, então ir à linha 21;
19 | | **ii**) Se o hipergrafo solução parcial tiver algum nó não-terminal não expandido, ir à linha 2;
20 **end**
21 **return** Devolver o hipergrafo solução ótimo ou $\epsilon$-ótimo;

Fonte(Haunsen e Zibertein,2001)

Na execução do LAO para o problema proposto neste artigo, do cenários com poucas dimensões a diferença foi bem sucinta , já para grafos com dimensões maiores , o tempo de processamento foi 20% menor e uso de memória foi 15% menor, não tive um ganho tão considerável como relatado em outros artigos analisados e citados nas referências.

**Busca heurísticas, ILAO***

O algoritmo chamado de Improved LAO* (ILAO*) (Hansen e Zilberstein, 2001), tem como principal diferença entre LAO* é que não usa os algoritmos de programação dinâmica síncrona para atualizar custos. Na fase de construção do hipergrafo solução parcial (HGSP), ILAO* o constrói seguindo os hipergrafos marcados, isto é, hipegrafos que correspondem com escolhas gulosas. Na fase de expansão do hipergrafo solução parcial, o algoritmo ILAO* realiza uma busca em profundidade do melhor hipergrafo solução parcial para encontrar os nós não-terminais e fazer a expansão. O algoritmo ILAO* expande todos esses nós do melhor hipergrafo solução parcial. Além disso, ao invés de executar Iteração de Política ou Iteração de Valor na fase de atualização de custos, ILAO* atualiza todos os nós do melhor grafo solução somente uma vez. Porém, essa atualização é realizada numa ordem inversa à busca em

profundidade executada no melhor hipergrafo solução parcial atual.

**Input**:
$G$: Hipergrafo implícito dado pela função de transição probabilística e pelo estado inicial.
$s_0$: Estado inicial.
$C$: Função custo.
$\hat{h}$: Estimativa heurística do custo.
**Output**: Devolve o hipergrafo solução ótimo ou $\epsilon$-ótimo

1 O hipergrafo explícito inicial $G'$ consiste inicialmente do nó inicial $n_0$=$s_0$;
2 **while** O hipergrafo solução parcial tiver algum nó não-terminal **do**
3 /*Busca em profundidade e expansão*/;
4 Executar busca em profundidade no melhor hipergrafo solução parcial atual;
5 **foreach** nó $n_i$ visitado em pós-ordem **do**
6 Se o nó $n_i$ não estiver expandido, expanda-o e para cada nó sucessor $n_j$ inicializar $V(n_j)$=$\hat{h}(n_j)$;
7 /*Atualização*/;
8 $V(n_i) = \min_{a \in A(n_i)} [C(n_i, a) + \sum_{n_j \in S} P(n_j|n_i, a)V(n_j)]$ e
9 etiquetar o melhor hiperarco (ação) para $n_i$;
10 **end**
11 Reconstruir $G'$;
12 **end**
13 /*Teste de convergência*/;
14 Executar *Iteração de Valor* sobre os nós do melhor hipergrafo solução;
15 Continuar até que alguma das seguintes condições seja alcançada;
16 **i)** Se o erro for menor do que $\epsilon$ então ir à linha 18;
17 **ii)** Se o melhor hipergrafo solução atual tiver algum nó não-terminal não expandido, ir à linha 2;
18 **return** Devolver o hipergrafo solução ótimo ou $\epsilon$-ótimo;

Na execução do ILAO para o problema proposto neste artigo, do cenários com poucas dimensões a diferença foi bem sucinta , já para grafos com dimensões maiores , o tempo de processamento e uso de memória foi menor, obtive ganhos similares aos obtidos em artigos correlatos, mas não consegui chegar a solução ótima na minha implementação .

## References


Bellman, R. (1957). A Markovian decision process (No. P-1066). RAND CORP SANTA MONICA CA.

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (2003). Artificial intelligence: a modern approach (Vol. 2).

Upper Saddle River: Prentice hallHasling, D. W.; Clancey, W. J.; and Rennels, G. R. 1983. Strategic Explanations in Consultation. *The International Journal of Man-Machine Studies* 20(1): 3–19.

Bertsekas, D., and Tsitsiklis, J. 1996. Neuro-Dynamic Programming. Athena Scientific

Blai Bonet & H´ector Geffner , Solving Stochastic Shortest-Path Problems with RTDP ,

Stefam Eldkmark, Heurist Serach

Hansen e Zilberstein(2001) E. Hansen e S. Zilberstein. LAO*: A heuristic search algorithm that nds solutions with loops