Capítulo 3

Algoritmos Eficientes e Exatos

Os algoritmos estudados no capítulo anterior conseguem encontrar uma política ótima, isto é, em finitas iterações uma política ótima é obtida[1].

No entanto a cada iteração do algoritmo o valor de todos os estados são processados. Se a quantidade de estados é muito grande, a complexidade do algoritmo aumenta no melhor caso, proporcionalmente e pode acontecer de não ser factível processar todos os estados. No entanto, ao considerar um estado inicial s_0, pode acontecer que a quantidade de estados alcançáveis sejam factíveis de serem processados.

Nesse capítulo serão considerados algoritmos que obtém uma política parcial para atingir uma meta a partir de um estado inicial.

[1]Na verdade o algoritmo de Iteração de Valor requer a escolha adequada do parâmetro ϵ.

3.1 Arcabouço Teórico

3.1.1 Shortest Stochastic Path

Como foi discutido no capítulo 1, a área de planejamento usualmente considera custos, metas e estado inicial. Nesse capítulo será considerado o formalismo de Caminho Estocástico mais Curto (SSP–*Shortest Stochastic Path*).

Definição 8 (SSP – Shortest Stochastic Path)**.** *Um SSP é definido por uma tupla* $\langle \mathcal{S}, \mathcal{A}, T, C, s_0, \mathcal{G} \rangle$ *onde:*

- $s \in \mathcal{S}$ *são estados possíveis;*
- $a \in \mathcal{A}$ *são ações possíveis;*
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *é a função de transição;*
- $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ *é a função custo;*
- $s_0 \in \mathcal{S}$ *é o estado inicial; e*
- $\mathcal{G}$ *é o conjunto de estados metas.*

O objetivo do agente é, partindo do estado inicial, chegar no estado meta com o menor custo acumulado esperado, isto é, o valor do estado inicial é dado por:

$$V^{\pi}(s_0) = \lim_{T \to \infty} \mathrm{E}\left[\sum_{t=0}^{t=T-1} c_t \middle| \pi, s_0 \right].$$

Os algoritmos estudados nesse capítulo fazem algumas considerações com relação ao SSP, para garantir que: exista uma solução bem definida e os algoritmos possam encontrar tal solução.

Definição 9. *Um SSP deve respeitar as seguintes condições para garantir que exista uma solução bem definida:*

1. *existe uma política própria a partir do estado inicial* s_0[2]*; e*
2. *qualquer política não-própria obtém um custo acumulado esperado infinito.*

A primeira condição garante que existe uma política sob a qual pode-se calcular um custo acumulado esperado finito, o que permite comparar políticas. A segunda condição garante que ao aplicar o operador de Bellman, políticas não-próprias sempre aumentam seu valor.

3.1.2 Operador Monotônico

Para garantir que os algoritmos encontrem uma solução ótima, uma propriedade importante do operador de Bellman, que será explorada, refere-se a monotonicidade.

Definição 10 (Operador Monotônico)**.** *Um operador* $\mathcal{F}$ *é monotônico se e somente se:*

$$V_1 \geq V_2 \Rightarrow (\mathcal{F}V_1) \geq (\mathcal{F}V_2),$$

onde $V_1 \geq V_2$ *por definição implica* $V_1(s) \geq V_2(s)$ *para todo* $s \in \mathcal{S}$*.*

Teorema 8. *O operador de Bellman*

$$(\mathcal{T}V)(s) = \min_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s \in \mathcal{S}} T(s,a,s')V(s') \right\}$$

é monotônico.

[2]Lembre-se que uma política própria alcança um estado meta com probabilidade 1.

Demonstração. Considere que $V_1(s) \geq V_2(s)$ para todo $s \in \mathcal{S}$, mostrar que o operador $\mathcal{T}$ é monotônico é mostrar que $(\mathcal{T}V_1)(s) - (\mathcal{T}V_2)(s) \geq 0$ para todo $s \in \mathcal{S}$. Então:

$$\begin{aligned}
(\mathcal{T}V_1)(s) - (\mathcal{T}V_2)(s) &= \min_{a\in\mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s\in\mathcal{S}} T(s,a,s')V_1(s') \right\} \\
&\quad - \min_{a\in\mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s\in\mathcal{S}} T(s,a,s')V_2(s') \right\} \\
&\geq \min_{a\in\mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s\in\mathcal{S}} T(s,a,s')V_1(s') \right. \\
&\quad \left. - C(s,a) - \gamma \sum_{s\in\mathcal{S}} T(s,a,s')V_2(s') \right\} \\
&= \min_{a\in\mathcal{A}} \left\{ \gamma \sum_{s\in\mathcal{S}} T(s,a,s')(V_1(s') - V_2(s')) \right\} \\
&\geq 0.
\end{aligned}$$

□

3.1.3 Políticas Parciais

Lembre-se que políticas realizam o mapeamento de situações para ações. No caso mais simples, e também o caso que interessa para SSPs, uma política determinista, markoviana e estacionária é dada por $\pi : \mathcal{S} \to \mathcal{A}$. Note que, a política π especifica para qualquer estado possível uma ação a ser executada. Por outro lado, quando um estado inicial s_0 é considerado, nem todos os estados são alcançáveis, de forma que a parte útil da política é apenas as ações determinadas por esses estados alcançáveis.

Definição 11 (Estados Alcançáveis)**.** *Dada uma política π e um*

estado inicial s_0, um estado $s \in \mathcal{S}$ é alcançável se e somente se:

$$\lim_{T \to \infty} \sum_{t=0}^{T-1} \Pr(s_t = s | s_0, \pi) > 0.$$

Definição 12 (Política Parcial)**.** *Uma política parcial $\pi : \mathcal{S}_\pi \subseteq \mathcal{S} \to \mathcal{A}$ é uma política definida apenas em um subconjunto $\mathcal{S}_\pi$ de $\mathcal{S}$.*

Definição 13. *Uma política parcial $\pi : \mathcal{S}_\pi \subseteq \mathcal{S} \to \mathcal{A}$ é fechada com relação a um estado s se todo estado alcançável s' a partir de s executando a política π está contido em $\mathcal{S}_\pi$.*

Nas próximas seções serão definidos algoritmos que encontram políticas fechadas com relação ao estado inicial s_0.

3.2 Algoritmo A*

Os algoritmos vistos nesse capítulo farão uso do problema de busca em um grafo. Nesse caso, considera-se um grafo e deseja-se encontrar o caminho de menor custo entre um nó inicial e um nó meta. Esse caminho é encontrado por meio da busca em uma árvore gerada a partir desse grafo.

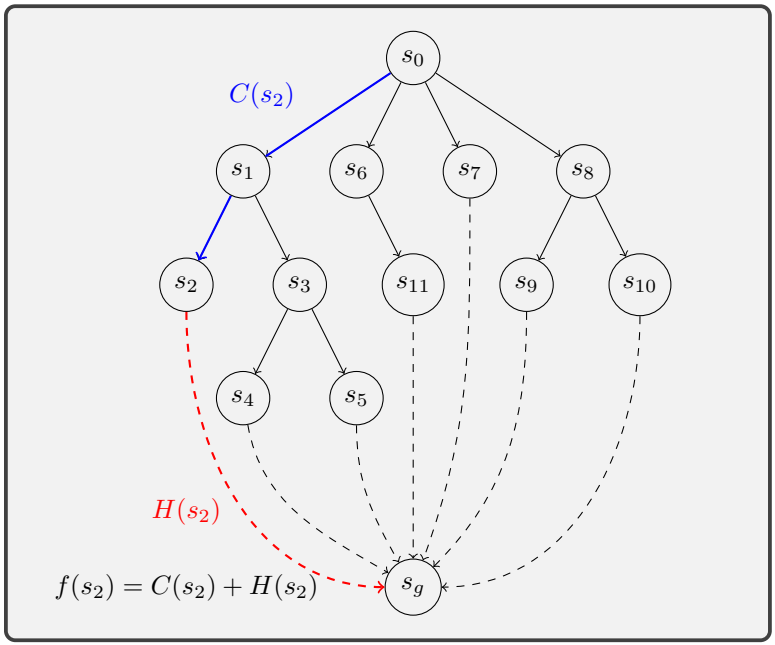
Diferentemente dos SSPs, os algoritmos de busca tradicionais consideram apenas o caso determinista, isto é, ao escolher uma ação escolhe-se indiretamente um próximo estado.

Um algoritmo de busca segue a seguinte estrutura:

Algoritmos de Busca

1. inicializa uma árvore com o nó raiz s_0
2. repete enquanto não atende *critério de parada*
   a) escolha um nó folha para expandir
   b) expanda o nó folha escolhido
   c) faça anotações necessárias

O algoritmo mais famoso de busca é o algoritmo A*. Esse algoritmo considera: uma função heurística $H(n)$ que estima o custo para chegar na meta a partir de qualquer nó folha n, e o custo acumulado $C(n)$ do nó raiz até o nó folha em questão. O próximo nó a ser expandido é o nó com menor valor somando heurística (custo estimado dali para frente) e custo acumulado (custo realizado dali para trás), isto é, considerando uma função de avaliação $f(n) = H(n) + C(n)$.



O algoritmo A* para quando o próximo nó a ser expandido é o nó meta. O algoritmo A* é famoso por dois motivo, primeiro, ele permite adicionar informação *a priori* para acelerar a busca, segundo, se a heurística for admissível ele garante uma solução ótima.

Definição 14 (Heurística admissível)**.** *Considere a função* $V : \mathcal{S} \to \mathbb{R}^*$ *que indica o custo ótimo de um estado* s *até o estado meta* s_g*. Uma função heurística* H *é admissível se ela não superestima o custo ótimo para a meta, isto é,* $H(s) \leq V(s)$ *para todo* $s \in \mathcal{S}$*.*

Teorema 9. *O algoritmo* A^* *com uma função heurística admissível encontra o caminho ótimo.*

3.3 Hipergrafos e Grafos AND-OR

Um hipergrafo é um grafo dirigido no qual as arestas são hiperarestas, isto é, possuem uma fonte, mas vários destinos. Um hipergrafo pode ser representado por um grafo AND-OR que considera dois tipos de vértices: AND e OR. Um vértice OR é um vértice comum, no qual pode-se escolher entre qualquer das arestas de saída, portanto é chamado também de vértice de decisão (será realizada nele uma operação de minimização). Um vértice AND é um vértice no qual todas as arestas ocorrem segunda uma distribuição de probabilidade, portanto é chamado também de vértice de chance (será realizada nele uma operação de esperança).

Um SSP pode ser representado em grafos AND-OR com as seguintes associações:

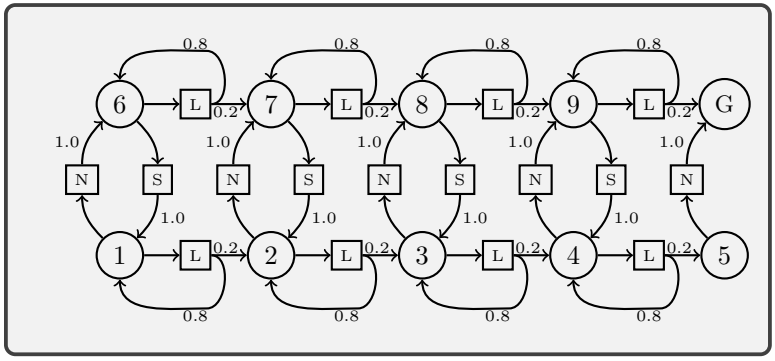
1. estados são vértices OR;
2. ações são vértices AND;
3. arestas saem de estados e vão para ações;
4. arestas saem de ações e vão para estados;
5. as probabilidades das arestas são dadas pela função de transição; e
6. o custos das arestas são dados pela função custo.

Considere o seguinte exemplo em um mundo de grades:

P	6	7	8	9	G
D	1	2	3	4	5

Na linha P pode-se escolher as ações S(ul) e L(este), enquanto em D pode-se escolher as ações N(orte) e L(este). A ação L(este) executada na linha P têm resultado probabilístico: transita para o estado vizinho escolhido com 0.2 de chance ou ficam no mesmo estado caso contrário. Todas as outras ações são deterministas.

No grafo abaixo, estados são representados por círculos e ações são representadas por quadrados. As probabilidades de transição são indicadas diretamente no grafo, enquanto os custos não são indicados.



3.4 Algoritmo LAO*

Nesta seção será apresentado o algoritmo LAO*. Esse algoritmo é baseado no algoritmo A* para lidar com SSPs. Essencialmente um SSP apresenta duas diferenças com relação ao problema de busca em um grafo: (*i*) SSPs são representados por um grafo AND-OR; e (*ii*) SSPs apresentam *loops*, o que dificulta a estratégia de custo acumulado. O algoritmo LAO* encontra soluções ótimas em um grafo AND-OR com *loops* utilizando a mesma estratégia do algoritmo A*.

Para lidar com *loops* e grafos AND-OR, o algoritmo LAO* inicializa o valor de qualquer estado com uma heurística admissível e atualiza o valor dos estados expandido utilizando o operador de Bellman. Lembre-se que o operador de Bellman é monotônico; logo, se o valor inicial dos estados for inicializado como uma heurística admissível, após aplicar o operador de Bellman, a propriedade de admissibilidade continua sendo verdade.

3.4.1 Preliminares

Como já comentado, um SSP pode ser representado por um hipergrafo (grafo AND-OR). Considere que $G_{\mathcal{S}}$ é o hipergrafo de Conectividade de um SSP, isto é, representa as transições de um SSP para todos os estados.

Definição 15 (Alcançabilidade)**.** *Um estado s_n é alcançável a partir de s_0 em um hipergrafo G, se existe um caminho entre s_0 e s_n em G.*

Definição 16. *O Hipergrafo de Conectividade G_{s_0} com estado inicial s_0 é o hipergrafo G_{s_0} que contém o vértice s_0 e todos os estados s' alcançáveis de s_0, mais suas respectivas hiperarestas.*

Definição 17. *O Hipergrafo Guloso de Conectividade $G^V_{s_0}$ de uma função valor V com estado inicial s_0 é o hipergrafo $G^V_{s_0}$ que contém o vértice s_0 e todos os estados s' alcançáveis de s_0 ao executar qualquer política gulosa segundo V, mais suas respectivas hiperarestas apontadas por tais políticas.*

Note que se pode limitar qualquer algoritmo de planejamento a processar apenas estados em G_{s_0} e, no melhor caso, deseja-se processar apenas os estados em $G^{V^*}_{s_0}$. O algoritmo LAO*, limitado por G_{s_0} no pior caso, busca processar a menor quantidade de estados possível. O algoritmo LAO* mantém:

- uma função valor $V(s) \leq V^*(s)$ para todo $s \in \mathcal{S}$; e
- hipergrafos $\hat{G}^V_{s_0} \subseteq \hat{G}_{s_0} \subseteq G_{s_0}$.

Em cada iteração o algoritmo LAO*:

1. aumenta a fronteira de $\hat{G}_{s_0}$,
2. atualiza V, e
3. reconstrói $\hat{G}^V_{s_0}$.

3.4.2 Descrição

O hipergrafo $\hat{G}_{s_0}$ contém dois tipos de estados: expandidos (I) e na fronteira (F); o mesmo vale para o hipergrafo $\hat{G}^V_{s_0}$. Em cada iteração é escolhido (6.a) um nó fronteira s de $\hat{G}^V_{s_0}$ para ser expandido, isto é, adicionar seus sucessores (ações e próximos estados) ao grafo $\hat{G}_{s_0}$ (6.c).

A atualização de V (6.g) é feita apenas para os estados em Z, os que são afetados pela a expansão do estado s, isto é, estados antecessores à s na política gulosa (6.f).

O algoritmo para quando não existe mais nenhum nó fronteira de $\hat{G}^V_{s_0}$ para ser expandido.

O algoritmo LAO* é apresentado no quadro abaixo:

Algoritmo LAO*

1. inicializa $V(s) = H(s) \leq V^*(s)$
2. $F \leftarrow \{s_0\}$ *// nós na fronteira*
3. $I \leftarrow \emptyset$ *// nós expandidos*
4. $\hat{G}_{s_0} \leftarrow I \cup F$
5. $\hat{G}^V_{s_0} \leftarrow \{s_0\}$
6. enquanto existe $s \in F \cap \hat{G}^V_{s_0}$ e $s \notin \mathcal{G}$ faça
   a) $s \leftarrow$ algum estado não meta em $F \cap \hat{G}^V_{s_0}$
   b) $F \leftarrow F \setminus \{s\}$
   c) $F \leftarrow F \cup \{x \notin I : \exists a \in \mathcal{A}\ T(s, a, x) > 0\}$
   d) $I \leftarrow I \cup \{s\}$
   e) $\hat{G}_{s_0} \leftarrow I \cup F$
   f) $Z \leftarrow \{s$ e todos estados que, executando política gulosa, podem alcançar $s\}$
   g) Atualize V para cada estado em Z, considerando estados em $s' \in F$ como estados terminais com valor $H(s')$
   h) Reconstrua $\hat{G}^V_{s_0}$ sobre estados de $\hat{G}_{s_0}$
7. retorne a política parcial $\pi^V_{s_0}$ que começa em s_0 e determinada pela função V

3.4.3 Convergência

Teorema 10. *Se as condições na definição 9 são atendidas e a heurística $H(s)$ é admissível, então o algoritmo LAO* converge para a política ótima.*

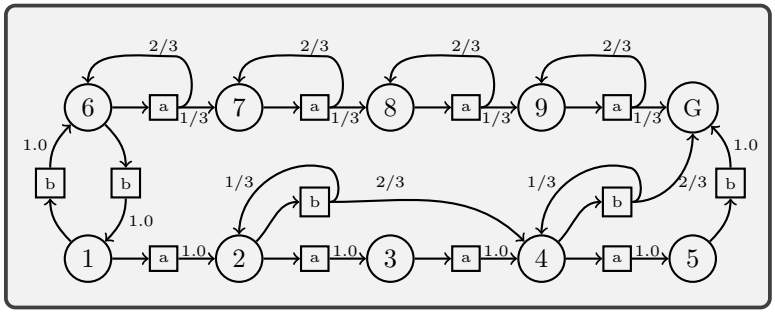
Demonstração. Podemos dividir o algoritmo em dois casos:

- todos os nós foram adicionados a $\hat{G}_{s_0}$ e portanto $\hat{G}_{s_0} = G_{s_0}$, então a atualização de V garante a política ótima; ou
- nem todos os nós foram adicionados, nesse caso o grafo $\hat{G}^V_{s_0}$ deve incluir o estado meta e a meta é alcançada com probabilidade 1, caso contrário o grafo $\hat{G}^V_{s_0}$ não é um grafo guloso, uma vez que políticas não-próprias tem custo infinito.

Finalmente, note que por conta da garantia de que V é sempre admissível, pois foi iniciado de forma admissível e o operador $\mathcal{T}$ é monotônico, nenhum outro caminho pode obter custo esperado menor do que o caminho em $\hat{G}^V_{s_0}$. □

3.4.4 Exemplo

Considere o hipergrafo abaixo no qual G é o estado meta e $s_0 = 6$ é o estado inicial. Considere ainda uma heurística admissível sem informação $H(s) = 0$ para todo $s \in \mathcal{S}$.



Inicialmente tem-se:

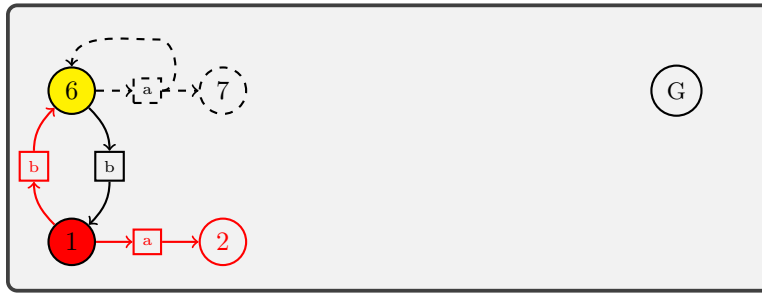
$\hat{G}_{s_0}$	6
V	0.0
$\hat{G}^V_{s_0}$	6

Então, escolhe-se o estado 6 para ser expandido.

Obtém-se:

$\hat{G}_{s_0}$	6	7	1
V	1.0	0.0	0.0
$\hat{G}^V_{s_0}$	6		1

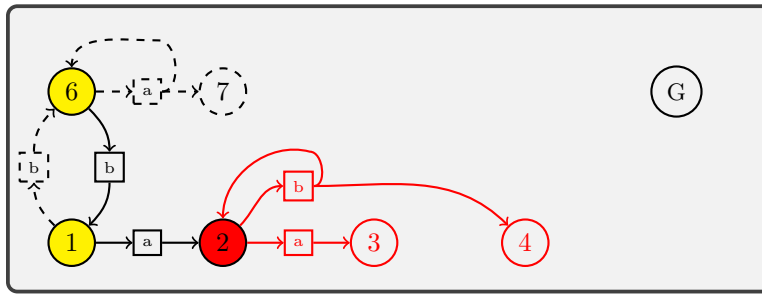
Então, escolhe-se o estado 1 para ser expandido.



Obtém-se:

$\hat{G}_{s_0}$	6	7	1	2
V	2.0	0.0	1.0	0.0
$\hat{G}^V_{s_0}$	6		1	2

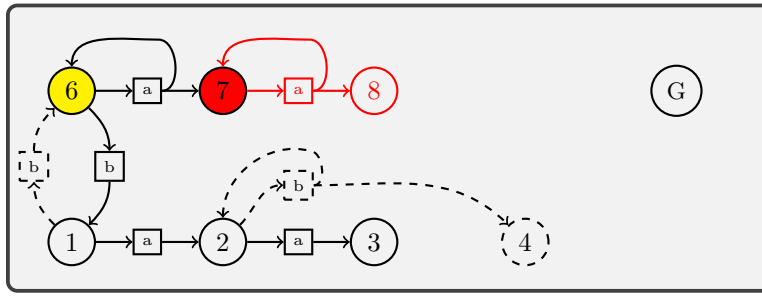
Então, escolhe-se o estado 2 para ser expandido.



Obtém-se:

$\hat{G}_{s_0}$	6	7	1	2	3	4
V	3.0	0.0	2.0	1.0	0.0	0.0
$\hat{G}^V_{s_0}$	6	7	1	2	3	

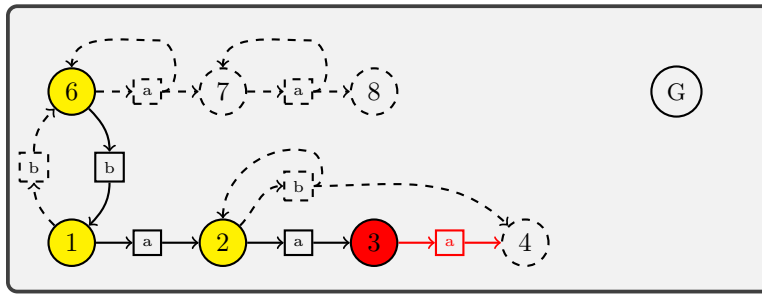
Note que agora qualquer dos dois caminhos a partir de 6 obtém o mesmo valor, por isso os dois estão fazem parte do caminho hipergrafo guloso. Então, escolhe-se o estado 7 (poderia também ser o 3) para ser expandido.



Obtém-se:

$\hat{G}_{s_0}$	6	7	1	2	3	4	8
V	3.0	3.0	2.0	1.0	0.0	0.0	0.0
$\hat{G}^V_{s_0}$	6		1	2	3		

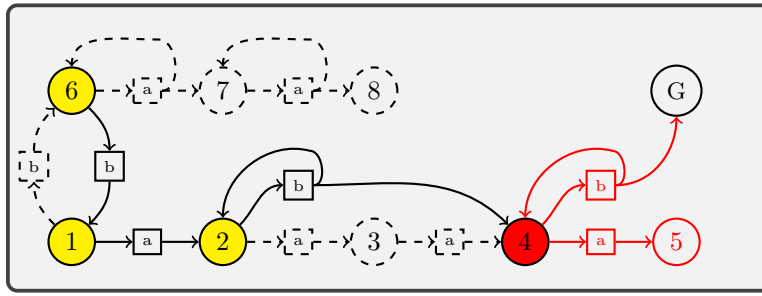
Então, escolhe-se o estado 3 para ser expandido.



Obtém-se:

$\hat{G}_{s_0}$	6	7	1	2	3	4	8
V	3.5	3.0	2.5	1.5	1.0	0.0	0.0
$\hat{G}^V_{s_0}$	6		1	2		4	

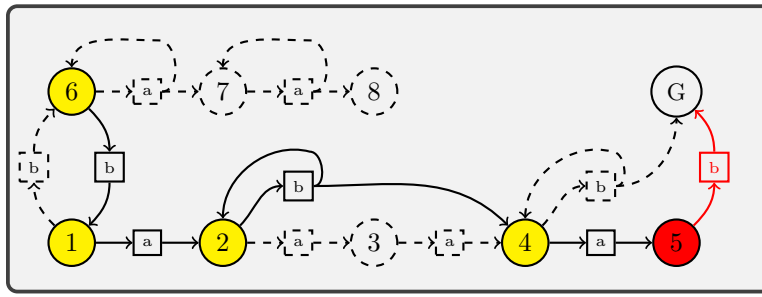
Então, escolhe-se o estado 4 para ser expandido.



Obtém-se:

$\hat{G}_{s_0}$	6	7	1	2	3	4	8	5	G
V	4.5	3.0	3.5	2.5	1.0	1.0	0.0	0.0	0.0
$\hat{G}^V_{s_0}$	6		1	2		4		5	

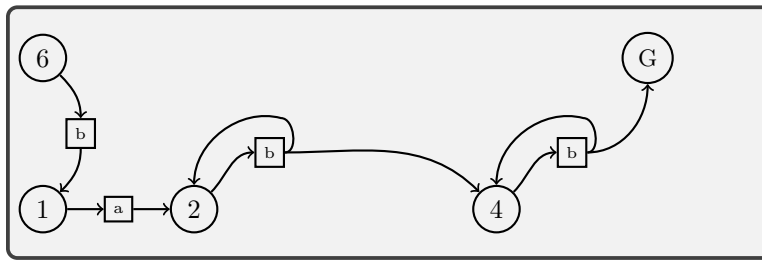
Então, escolhe-se o estado 5 para ser expandido.



Obtém-se:

$\hat{G}_{s_0}$	6	7	1	2	3	4	8	5	G
V	5.0	3.0	4.0	3.0	1.0	1.5	0.0	1.0	0.0
$\hat{G}^V_{s_0}$	6		1	2		4		5	

Então, o algoritmo para e obtém a política parcial abaixo.



Note que:

- o estado 9 não foi inicializado;
- o estado 8 não foi expandido; e
- o estado 3 não continua sendo atualizado uma vez que ele foi descartado da política ótima.

Todas essas observações permite que menos operações de Bellman sejam realizadas. Ainda, se uma heurística mais informativa (mais próxima da função valor V^*) fosse utilizada, poderia-se evitar que, por exemplo, os estados 7, 3 e 5 fossem expandidos.

3.5 LRTDP

O algoritmo LAO* evita processar todos os estados e por isso ele pode ser muito mais rápido que o iteração de valor dependendo da heurística utilizada e da dinâmica do problema. No entanto ele apresenta um problema: ele só encontra uma política própria após encontrar a meta. O algoritmo LRTDP (Labeled Real-Time Dynamic Programming) busca sanar esse problema, apresentando um algoritmo que pode ser parado em qualquer momento e extraída uma política de qualidade, se há tempo suficiente disponível, a solução ótima é encontrada.

O algoritmo LRTDP utiliza-se de três estratégias:

1. etiquetagem de estados para os quais se conhece a política ótima;

2. simulações da execução de uma política gulosa π^V até encontrar um estado etiquetado; e

3. atualização da função valor V para garantir que o agente não fique preso em alguma região e não encontre estados já etiquetados.

O algoritmo LRTDP realiza buscas em profundidade guiadas pela função valor V (2.c.iii e 2.c.v) a partir do estado inicial (2.a). A busca para quando um estado marcado como resolvido é encontrado (2.c) ou um estado meta. A cada estado s encontrado, o operador de Bellman é aplicado a tal estado (2.c.iv). Todos os estados encontrados são empilhados (2.c.i). A cada busca em profundidade realizada os estados visitados são avaliados em ordem inversa (2.d.i) enquanto os estados podem ser etiquetados como resolvido.

Algoritmo LRTDP

1. $V(s) = H(s) \leq V^*(s)$
2. enquanto s_0 não foi etiquetado *Solved*
   a) $s \leftarrow s_0$
   b) $visited \leftarrow EMPTYSTACK$
   c) enquanto s não foi etiquetado *Solved*
      i. $visited.PUSH(s)$
      ii. se $s \in \mathcal{G}$ então **break**
      iii. $a \leftarrow \pi^V(s)$
      iv. $V(s) \leftarrow (\mathcal{T}V)(s)$
      v. $s \sim T(s, a, \cdot)$
   d) enquanto $visited \neq EMPTYSTACK$
      i. $s \leftarrow visited.POP$
      ii. se não $CHECKSOLVED(s, \epsilon)$ então **break**
3. retorne a política π^V

Etiquetar um estado como resolvido é uma parte essencial do algoritmo LRTDP. Um estado s é marcado como resolvido se o resíduo da última atualização realizada pelo operador de Bellman para esse estado é menor que ϵ (5.c). Além disso, os estados alcançáveis a partir de s quando se considera a política gulosa π^V já devem ter sido marcados como resolvidos ou terem resíduos menor que ϵ (5.e).

$CHECKSOLVED(s, \epsilon)$

1. $rv = TRUE$
2. $open \leftarrow EMPTYSTACK$
3. $closed \leftarrow EMPTYSTACK$
4. se s não foi etiquetado *Solved* então $open.PUSH(s)$
5. enquanto $open \neq EMPTYSTACK$
   a) $s \leftarrow open.POP$
   b) $closed.PUSH(s)$
   c) se $Res^V(s) > \epsilon$ então
      i. $rv = FALSE$
      ii. **continue**
   d) $a \leftarrow \pi^V(s)$
   e) para todo $s' \in \{x \in \mathcal{S} : T(s, a, x) > 0\}$ faça
      i. se s' não foi etiquetado *Solved* e s' não está em $open \cup closed$ então
         A. $open.PUSH(s')$
6. se $rv = true$ então
   a) para todo $s' \in closed$ etiquete s' como *Solved*

   caso contrário
   b) para todo $s' \in closed$ faça $V(s') \leftarrow (\mathcal{T}V)(s')$
7. Retorna rv

Teorema 11. *Se as condições na definição 9 são atendidas, a heurística $H(s)$ é admissível e a meta é alcançada a partir de qualquer estado, então o algoritmo LRTDP converge para a política ótima.*

Demonstração. A condição para um estado ser etiquetado como

resolvido é que seus sucessores ante a política gulosa, essa é justamente a condição de convergência para o algoritmo iteração de valor.

Falta garantir que todos os estados em $G_{s_0}^{V^*}$ sejam visitados até ser etiquetado. Por conta do valor inicial de V ser admissível e o operador $\mathcal{T}$ ser monotônico; se em algum momento a ação ótima não é a de menor valor, após finitas iterações a ação ótima será a de menor valor.

Um outro problema é que por conta da simulação realizada, não é possível encontrar um tempo finito T tal que esse estado será visitado em no máximo T passos. No entanto o algoritmo $CHECKEDSOLVED$ garante que o operador $\mathcal{T}$ é aplicado aos estados sucessores dos estados avaliados visitados, garantindo que mesmos os estados com pouca chance de serem visitados são atualizados.

Por fim, a condição de que a meta é alcançável de qualquer estado e a condição de que políticas não-próprias possuem valor infinito, junto com as aplicações do operador $\mathcal{T}$ aos estados visitados, garantem que a meta é sempre encontrada. □

3.6 LAO*, LRTDP e suas Variações

Nesse capítulo foram apresentadas versões básicas dos algoritmos LAO* e LRTDP, mas vários algoritmos foram derivados a partir deles.

Um algoritmo antecessor do LAO* é o algoritmo AO*. Esse algoritmo é uma versão do LAO* específica para hipergrafos acíclicos. O critério de horizonte finito é um exemplo de hipergrafo acíclico.

O algoritmo Improved LAO* (ILAO*) ao atualizar a função V para os estados em Z, aplica apenas uma vez o operador $\mathcal{T}$ para cada estado. Isso permite que menos tempo seja gasto para convergir para um valor errado para o estado.

O algoritmo Bidirectional LAO* (BLAO*) realiza a busca em duas direções: do estado inicial para a meta e da meta para o estado inicial. Essa estratégia faz com que menos estados precisem serem expandido antes que uma rota até a meta possa ser encontrada.

Um algoritmo antecessor do LRTDP é o algoritmo RTDP. Esse algoritmo não faz use da etiquetagem dos estados resolvidos e portanto não garante que a política ótima é encontrada em um tempo finito.

O principal problema para convergência do algoritmo LRTDP são estados pouco visitados por conta de uma baixa probabilidade de transição. Algumas variações do algoritmo LRTDP (BRTDP, FRTDP, VPI-RTDP) guardam limites inferiores e superiores para ranquear os próximos possíveis estados e fazer uma simulação direcionada para estados que mais necessitam atualização.