Advanced MDP Algorithms

Mausam

VI → Asynchronous VI

- Is backing up *all* states in an iteration essential?
  - No!

- States may be backed up
  - as many times
  - in any order

- If no state gets starved
  - convergence properties still hold!!

Residual wrt Value Function V (Res^V)

- Residual at s with respect to V
  - magnitude($\Delta V(s)$) after one Bellman backup at s

$$Res^V(s) = \left| V(s) - \min_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')[\mathcal{C}(s,a,s') + V(s')] \right|$$

- Residual wrt respect to V
  - max residual
  - $Res^V = \max_s (Res^V(s))$ ← $Res^V < \epsilon$ (ϵ-consistency)

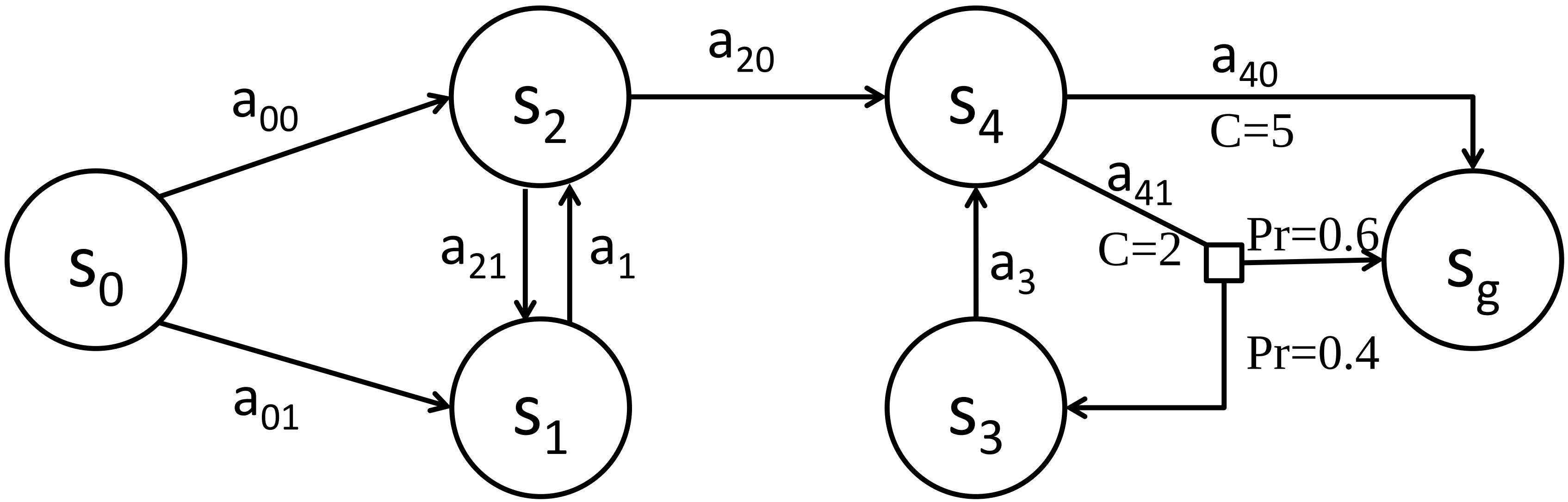
(General) Asynchronous VI

initialize V arbitrarily for each state
while $Res^V > \epsilon$ **do**
    select a state s
    compute $V(s)$ using a Bellman backup at s
    REVISE
    update $Res^V(s)$
end
return greedy policy π^V

Prioritization of Bellman Backups

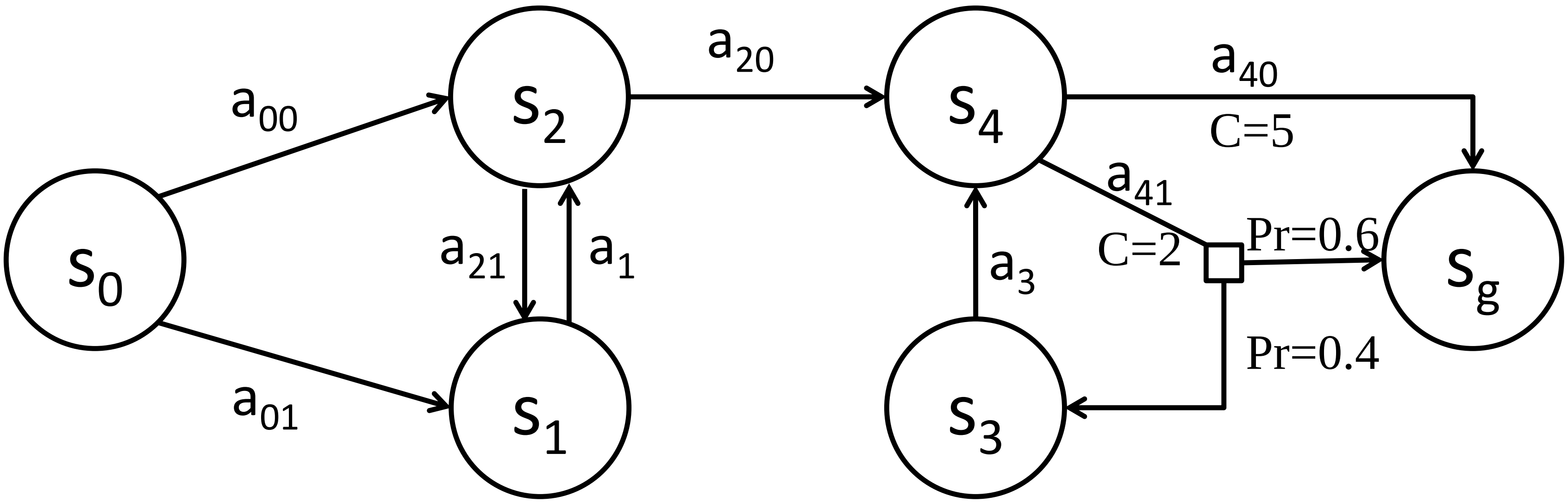
- Are all backups equally important?
- Can we avoid some backups?
- Can we schedule the backups more appropriately?

Useless Backups?



n	$V_n(s_0)$	$V_n(s_1)$	$V_n(s_2)$	$V_n(s_3)$	$V_n(s_4)$
0	3	3	2	2	1
1	3	3	2	2	2.8
2	3	3	3.8	3.8	2.8
3	4	4.8	3.8	3.8	3.52
4	4.8	4.8	4.52	4.52	3.52
5	5.52	5.52	4.52	4.52	3.808
20	5.99921	5.99921	4.99969	4.99969	3.99969

Useless Backups?



n	$V_n(s_0)$	$V_n(s_1)$	$V_n(s_2)$	$V_n(s_3)$	$V_n(s_4)$
0	3	3	2	2	1
1	3	3	2	2	2.8
2	3	3	3.8	3.8	2.8
3	4	4.8	3.8	3.8	3.52
4	4.8	4.8	4.52	4.52	3.52
5	5.52	5.52	4.52	4.52	3.808
20	5.99921	5.99921	4.99969	4.99969	3.99969

Asynch VI → Prioritized VI

initialize V

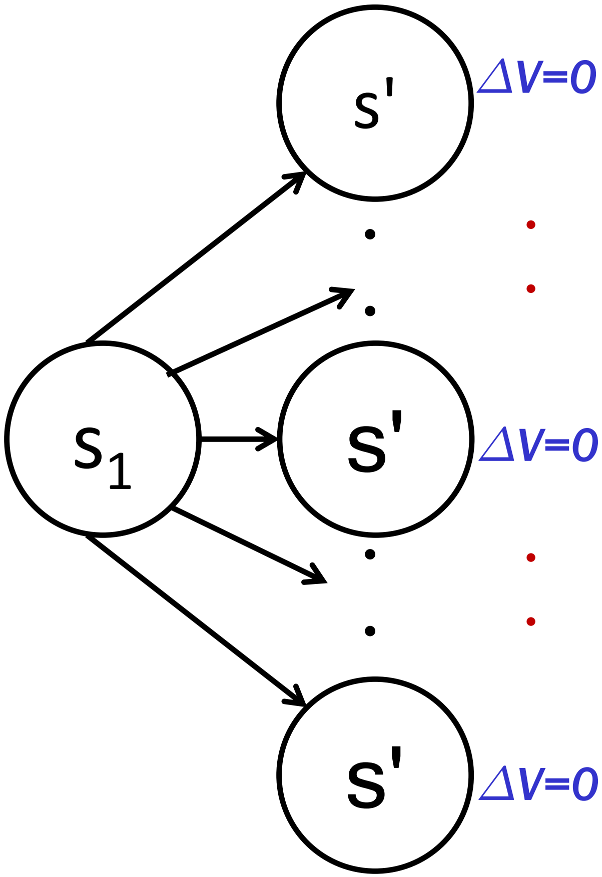
repeat

    select state s'
    compute $V(s')$ using a Bellman backup at s'
    foreach *predecessor s of* s', *i.e.*, $\{s | \exists a [\mathcal{T}(s, a, s') > 0]\}$ **do**
        compute priority(s)
        $q.push(s, \text{priority}(s))$
    end

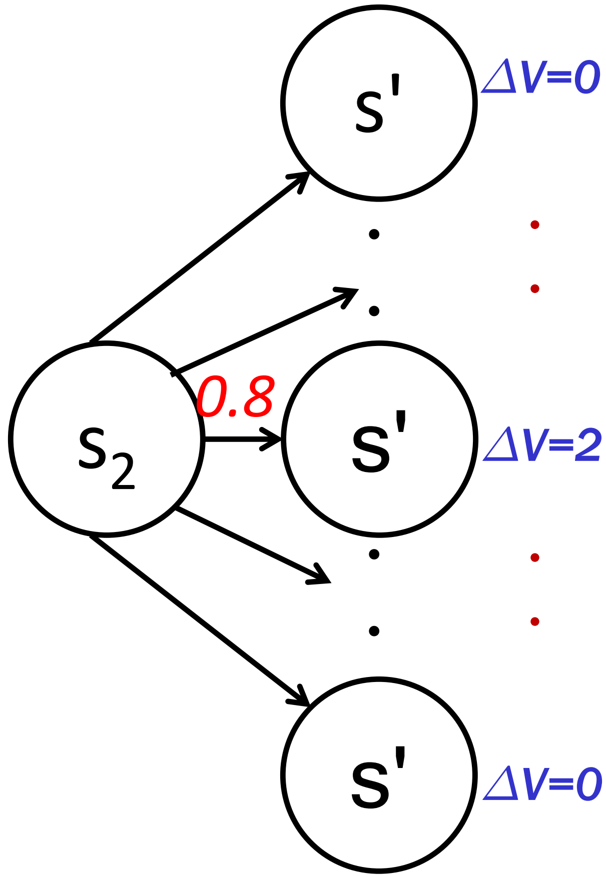
until *termination*;

return greedy policy π^V

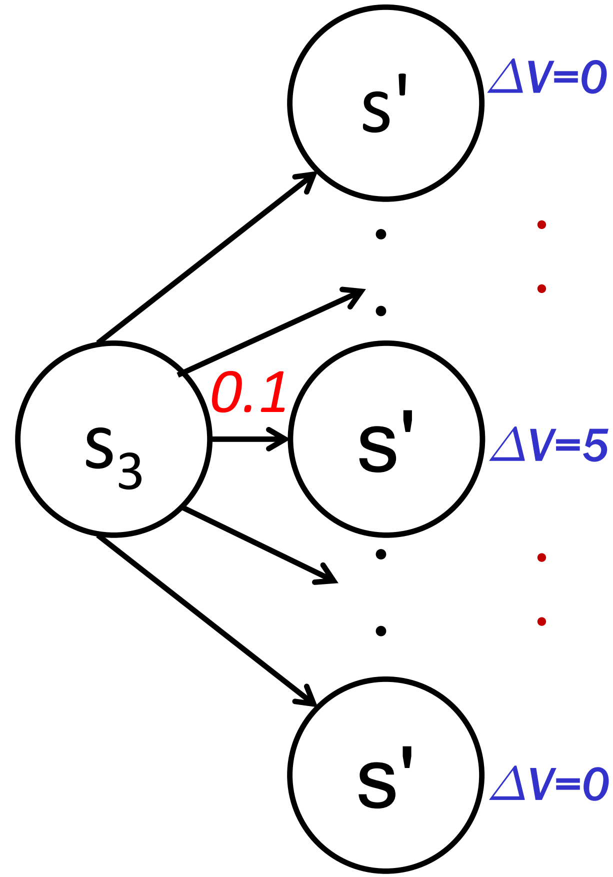
Which state to prioritize?



s_1 is zero priority



s_2 is higher priority



s_3 is low priority

Prioritized Sweeping

$$\boxed{\text{priority}_{PS}(s) = \max\left\{\text{priority}_{PS}(s), \max_{a \in \mathcal{A}}\{\mathcal{T}(s,a,s')Res^{V}(s')\}\right\}}$$

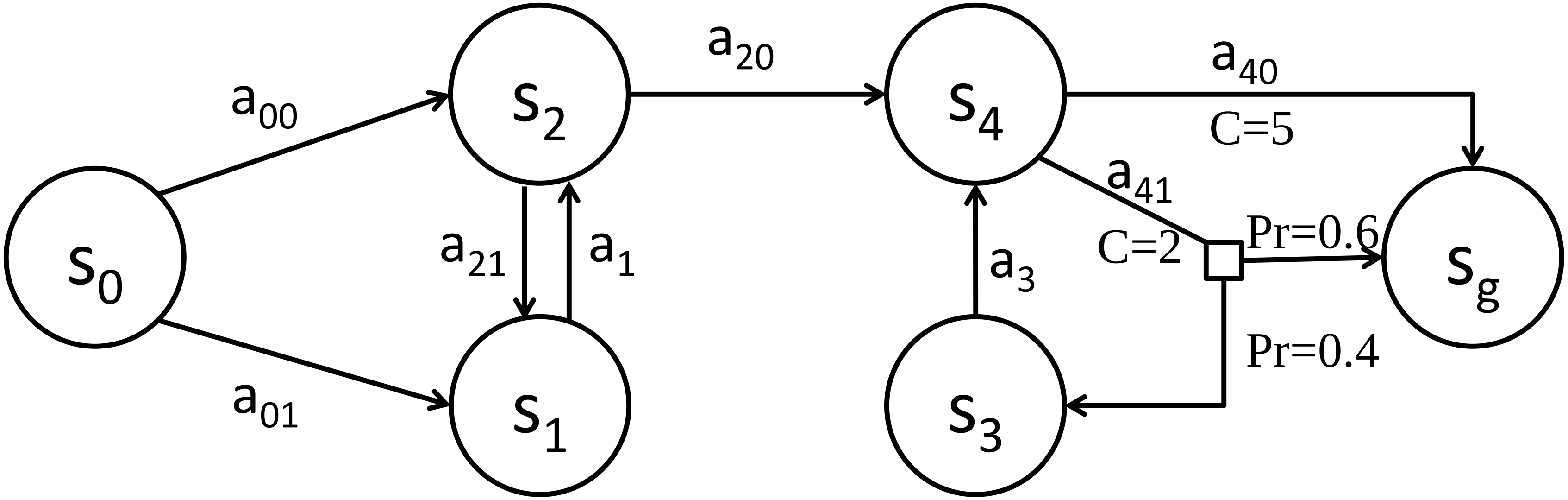
- Convergence [Li&Littman 08]

  Prioritized Sweeping converges to optimal in the limit,

  if all initial priorities are non-zero.

  (does not need synchronous VI iterations)

Prioritized Sweeping



	$V(s_0)$	$V(s_1)$	$V(s_2)$	$V(s_3)$	$V(s_4)$
Initial V	3	3	2	2	1
	3	3	2	2	2.8
Priority	0	0	1.8	1.8	0
Update	3	3	3.8	3.8	2.8
Priority	2	2	0	0	1.2
Update	3	4.8	3.8	3.8	2.8

Limitations of VI/Extensions

- Scalability
  - Memory linear in size of state space
  - Time at least polynomial or more
- Polynomial is good, no?
  - state spaces are usually huge.
  - if *n* state vars then 2^n states!
- Curse of Dimensionality!

Heuristic Search

- Insight 1
  - knowledge of a start state to save on computation

    ~ (all sources shortest path → single source shortest path)

- Insight 2
  - additional knowledge in the form of heuristic function

    ~ (dfs/bfs → A*)

Model

- MDP with an additional start state s_0
  - denoted by MDP_{s0}

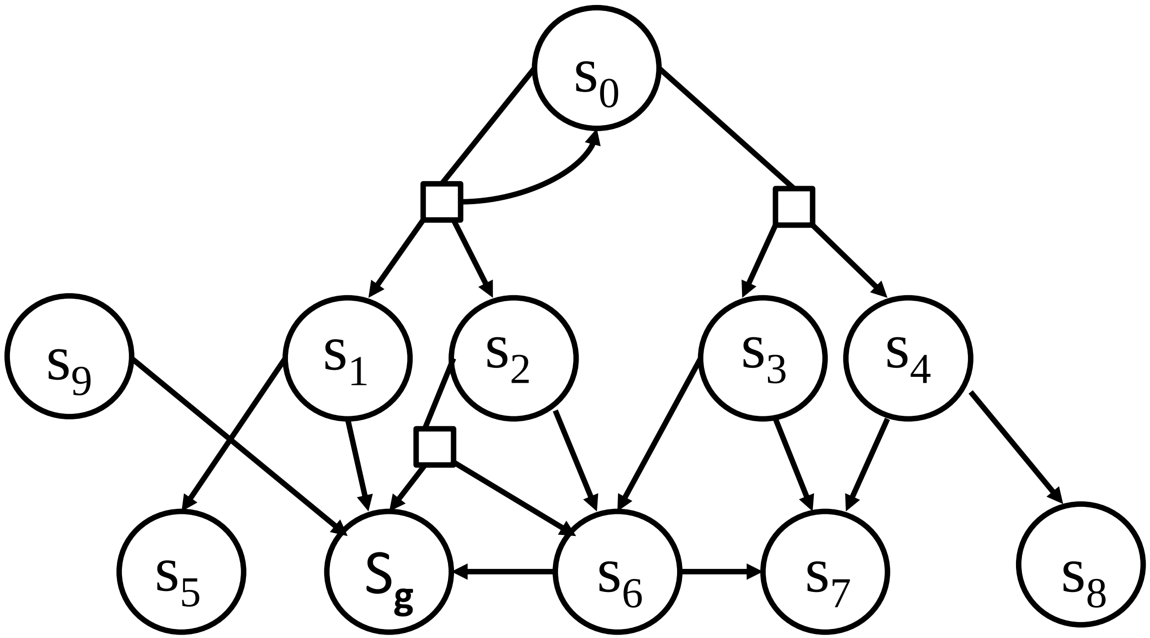
- What is the solution to an MDP_{s0}
- Policy ($S \rightarrow A$)?
  - are states that are not reachable from s_0 relevant?
  - states that are never visited (even though reachable)?

Partial Policy

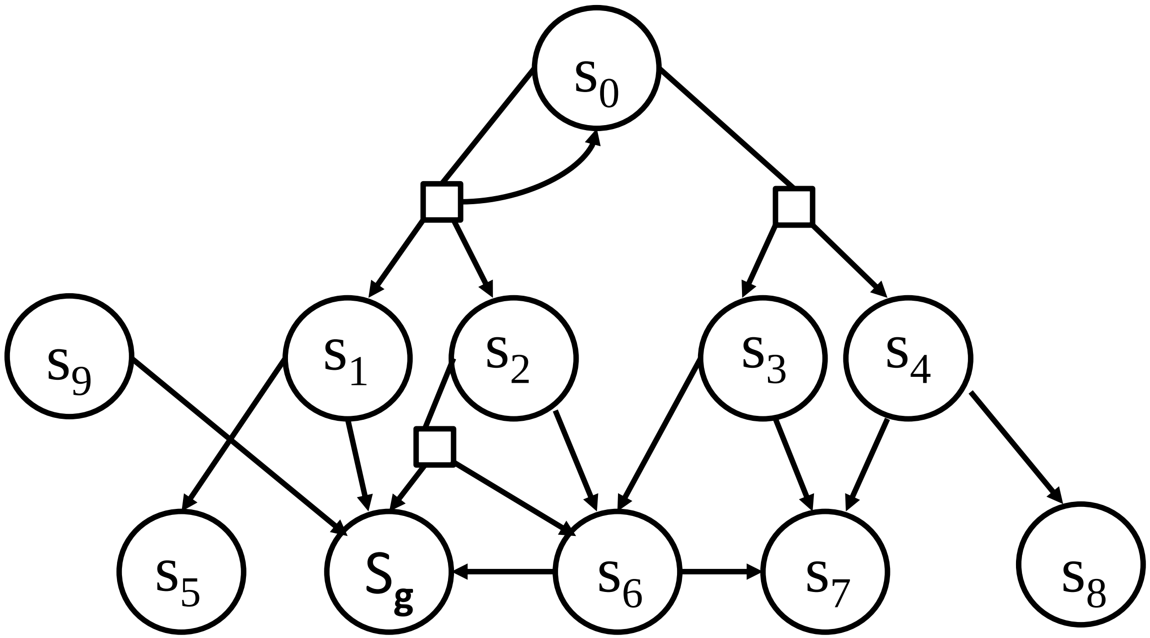
- Define *Partial policy*
  - π: S' $\rightarrow$ *A*, where S'$\subseteq$ *S*

- Define *Partial policy closed w.r.t. a state s.*
  - is a partial policy π_s
  - defined for all states *s'* reachable by π_s starting from *s*

Partial policy closed wrt s_0



Partial policy closed wrt s_0



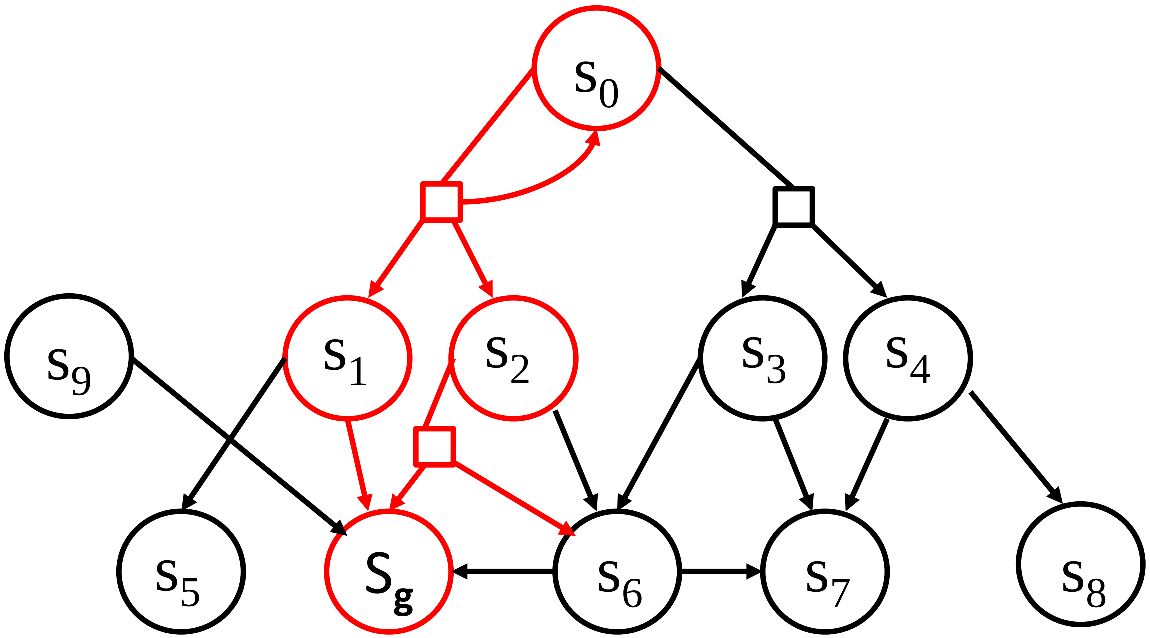
Is this policy closed wrt s_0?

$\pi_{s0}(s_0)= a_1$

$\pi_{s0}(s_1)= a_2$

$\pi_{s0}(s_2)= a_1$

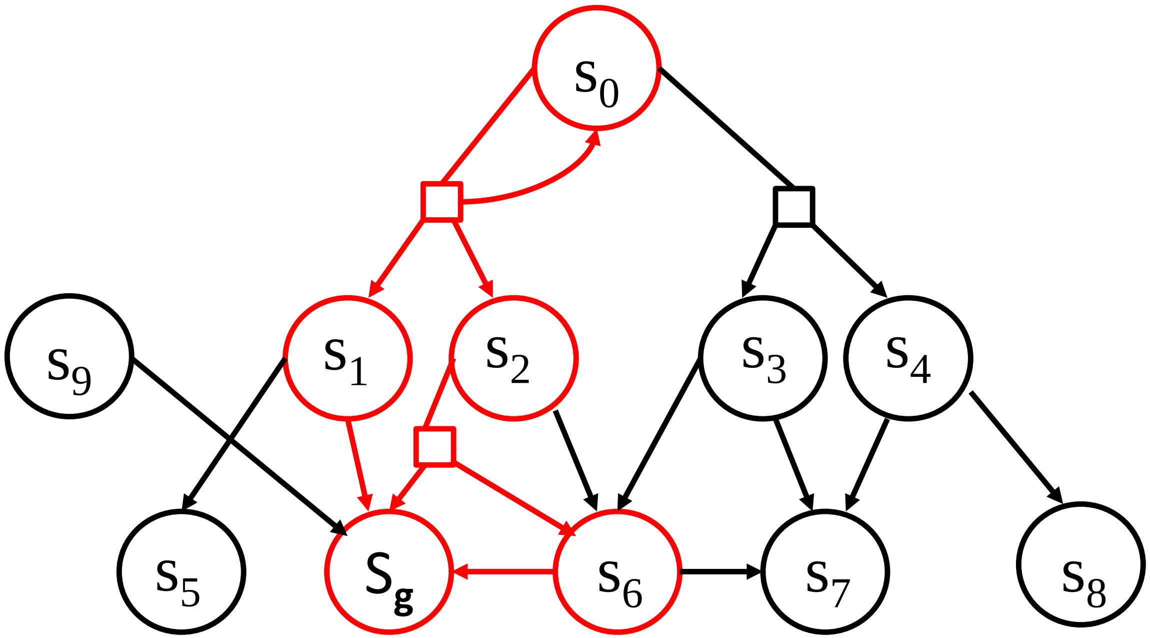
Partial policy closed wrt s_0



Is this policy closed wrt s_0?

$\pi_{s0}(s_0) = a_1$
$\pi_{s0}(s_1) = a_2$
$\pi_{s0}(s_2) = a_1$

Partial policy closed wrt s_0



Is this policy closed wrt s_0?

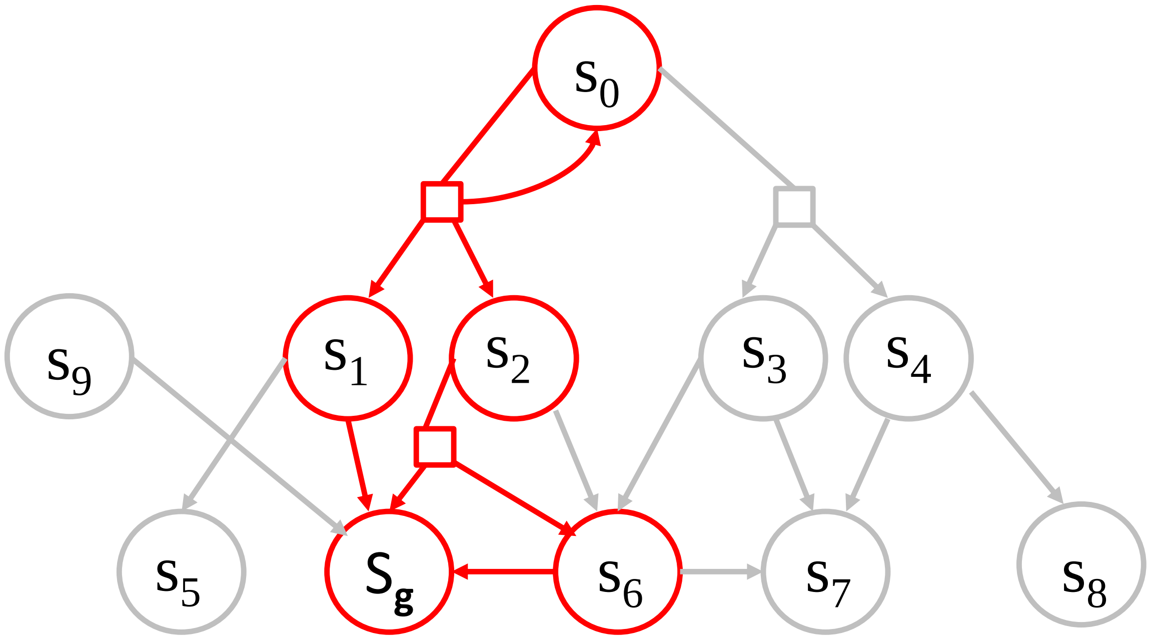
$\pi_{s0}(s_0) = a_1$

$\pi_{s0}(s_1) = a_2$

$\pi_{s0}(s_2) = a_1$

$\pi_{s0}(s_6) = a_1$

Policy Graph of π_{s0}



$\pi_{s0}(s_0) = a_1$
$\pi_{s0}(s_1) = a_2$
$\pi_{s0}(s_2) = a_1$
$\pi_{s0}(s_6) = a_1$

Greedy Policy Graph

- Define *greedy policy*: $\pi^V = \text{argmin}_a\, Q^V(s,a)$

- Define *greedy partial policy rooted at s_0*
  - Partial policy rooted at s_0
  - Greedy policy
  - denoted by π^V_{s0}

- Define *greedy policy graph*
  - Policy graph of π^V_{s0} : denoted by G^V_{s0}

Heuristic Function

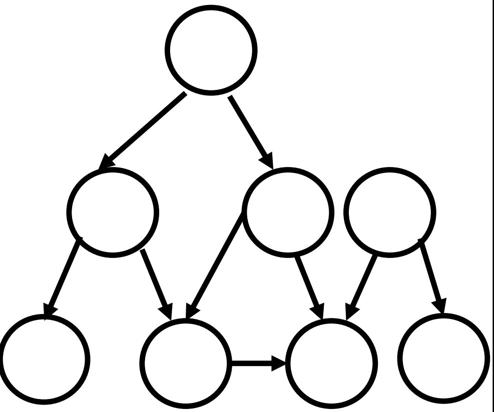
- $h(s): S \rightarrow \mathbb{R}$
  - estimates $V^*(s)$
  - gives an indication about “goodness” of a state
  - usually used in initialization $V_0(s) = h(s)$
  - helps us avoid seemingly bad states

- Define *admissible* heuristic
  - optimistic
  - $h(s) \leq V^*(s)$

A General Scheme for Heuristic Search in MDPs

- Two (over)simplified intuitions
  - Focus on states in greedy policy wrt V rooted at s_0
  - Focus on states with residual > ϵ
- Find & Revise:
  - repeat
    - find a state that satisfies the two properties above
    - revise: perform a Bellman backup
  - until no such state remains

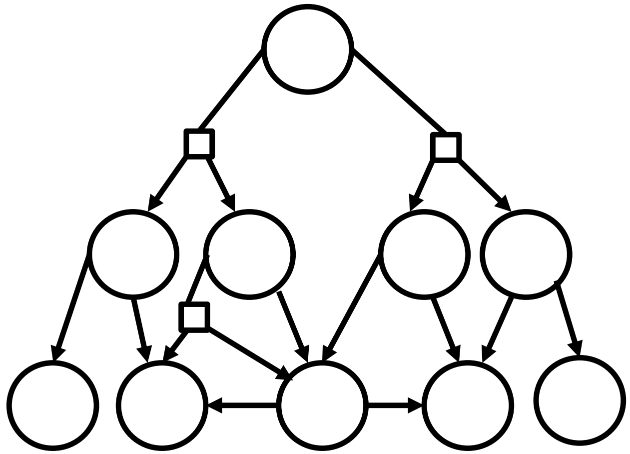
A* → LAO*



regular graph

soln:(shortest) path

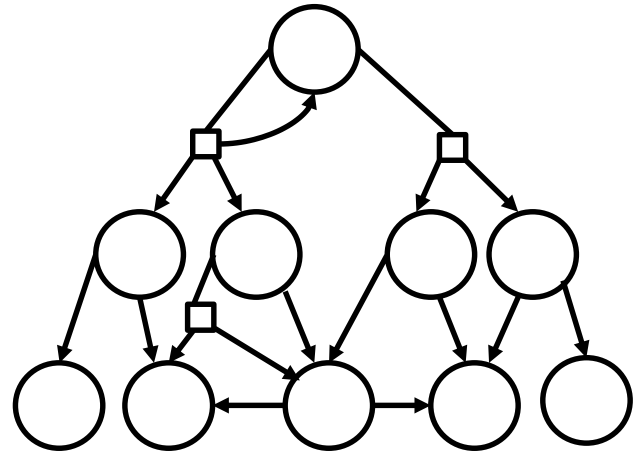
A*



acyclic AND/OR graph

soln:(expected shortest) acyclic graph

AO* [Nilsson'71]



cyclic AND/OR graph

soln:(expected shortest) cyclic graph

LAO* [Hansen&Zil.'98]

All algorithms able to make effective use of reachability information!

LAO* Family

add s_0 to the fringe and to greedy policy graph

repeat

- FIND: expand some states on the fringe (in greedy graph)
- initialize all new states by their heuristic value
- choose a subset of affected states
- REVISE: perform some Bellman backups on this subset
- recompute the greedy graph

until greedy graph has no fringe & residuals in greedy graph small

output the greedy graph as the final policy

LAO*

add s_0 to the fringe and to greedy policy graph

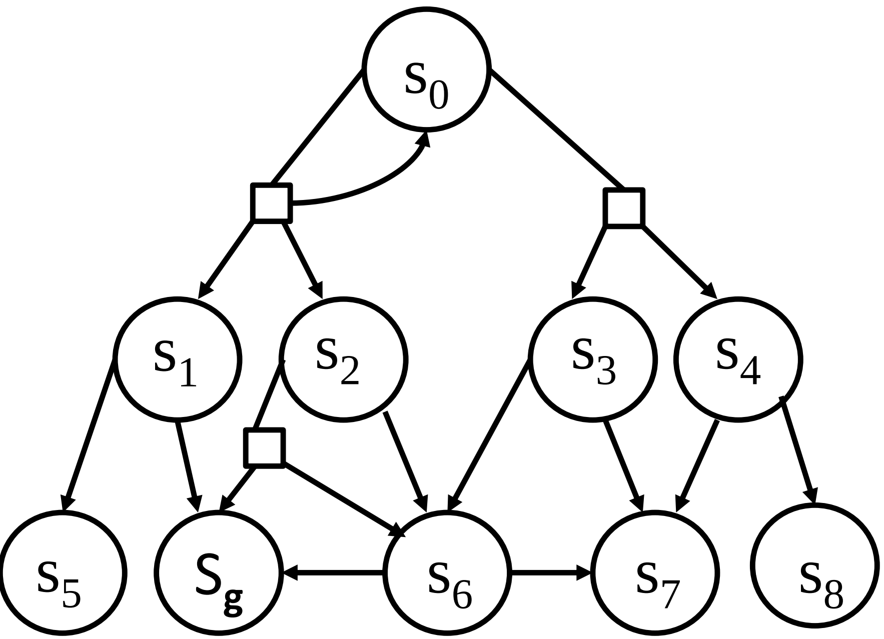
repeat

- FIND: expand best state *s* on the fringe (in greedy graph)
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- REVISE: perform VI on this subset
- recompute the greedy graph

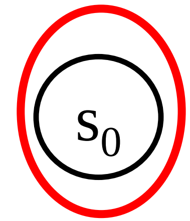
until greedy graph has no fringe ~~& residuals in greedy graph small~~

output the greedy graph as the final policy

LAO*

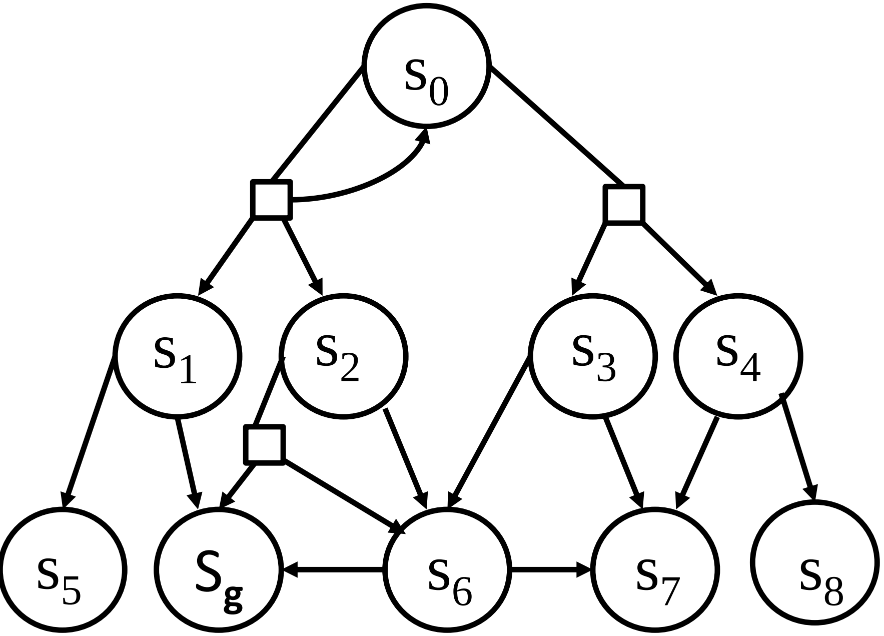


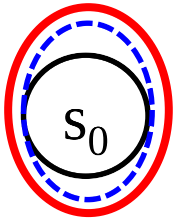
add s_0 in the fringe and in greedy graph



$V(s_0) = h(s_0)$

LAO*

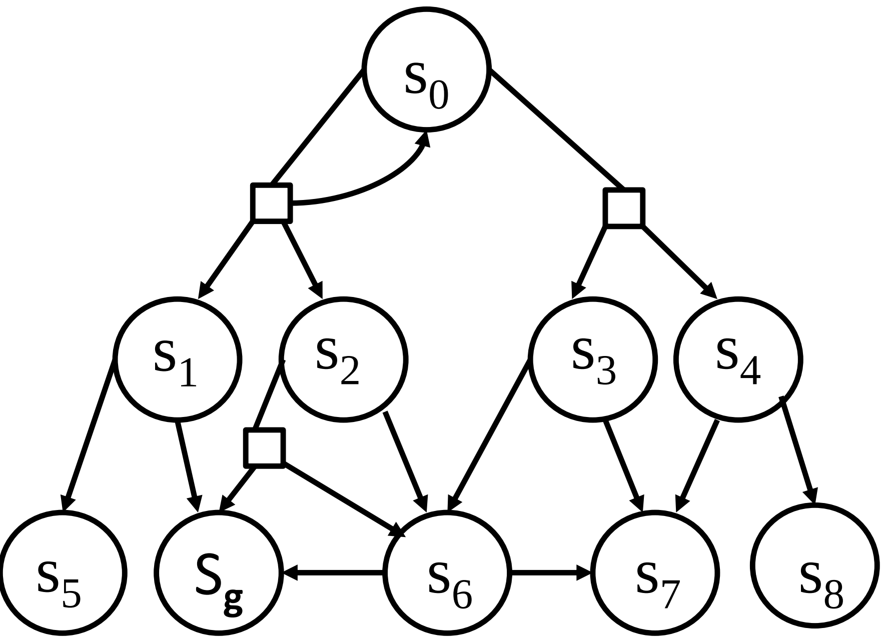


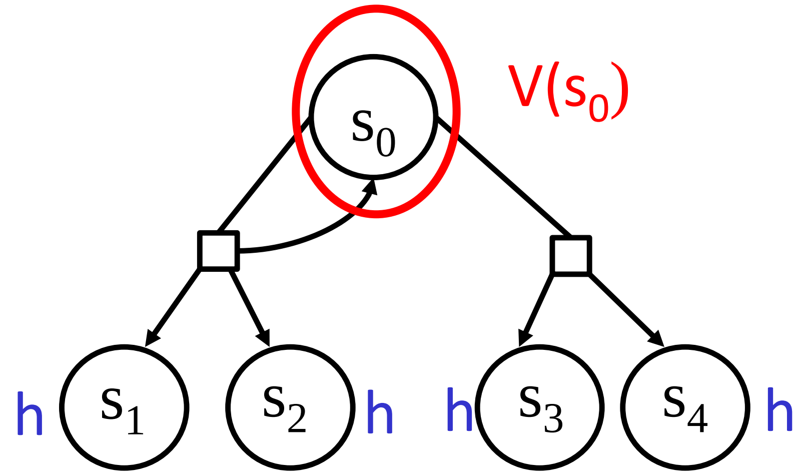


$V(s_0) = h(s_0)$

FIND: expand some states on the fringe (in greedy graph)

LAO*





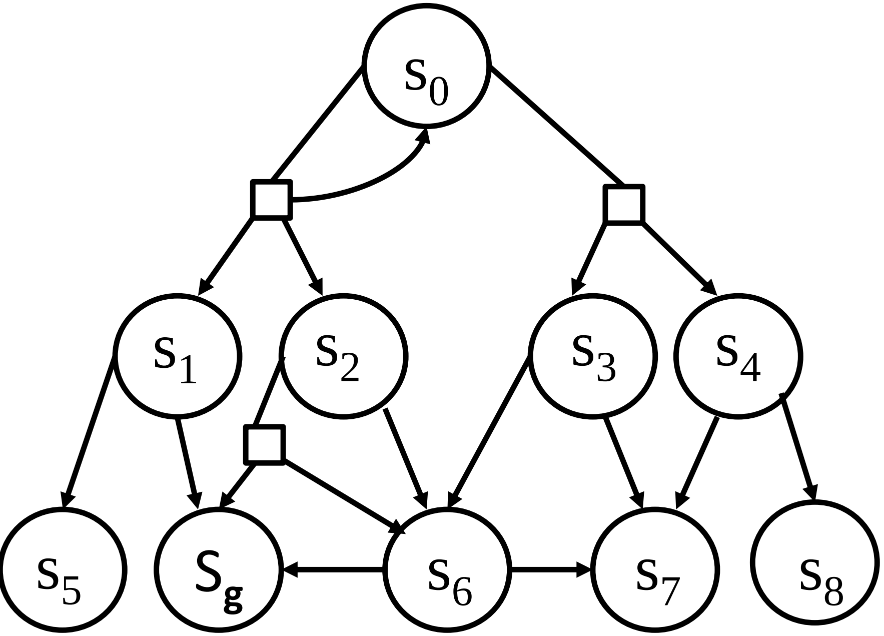
FIND: expand some states on the fringe (in greedy graph)

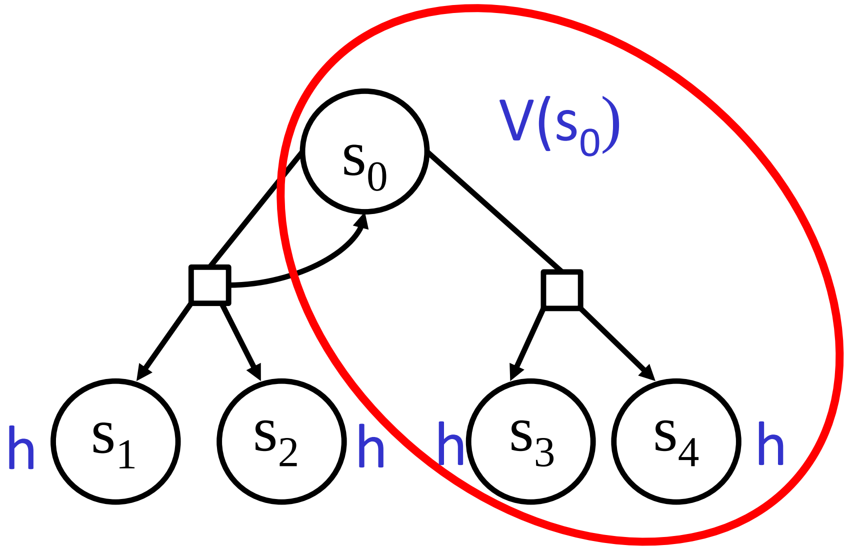
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

LAO*





FIND: expand some states on the fringe (in greedy graph)

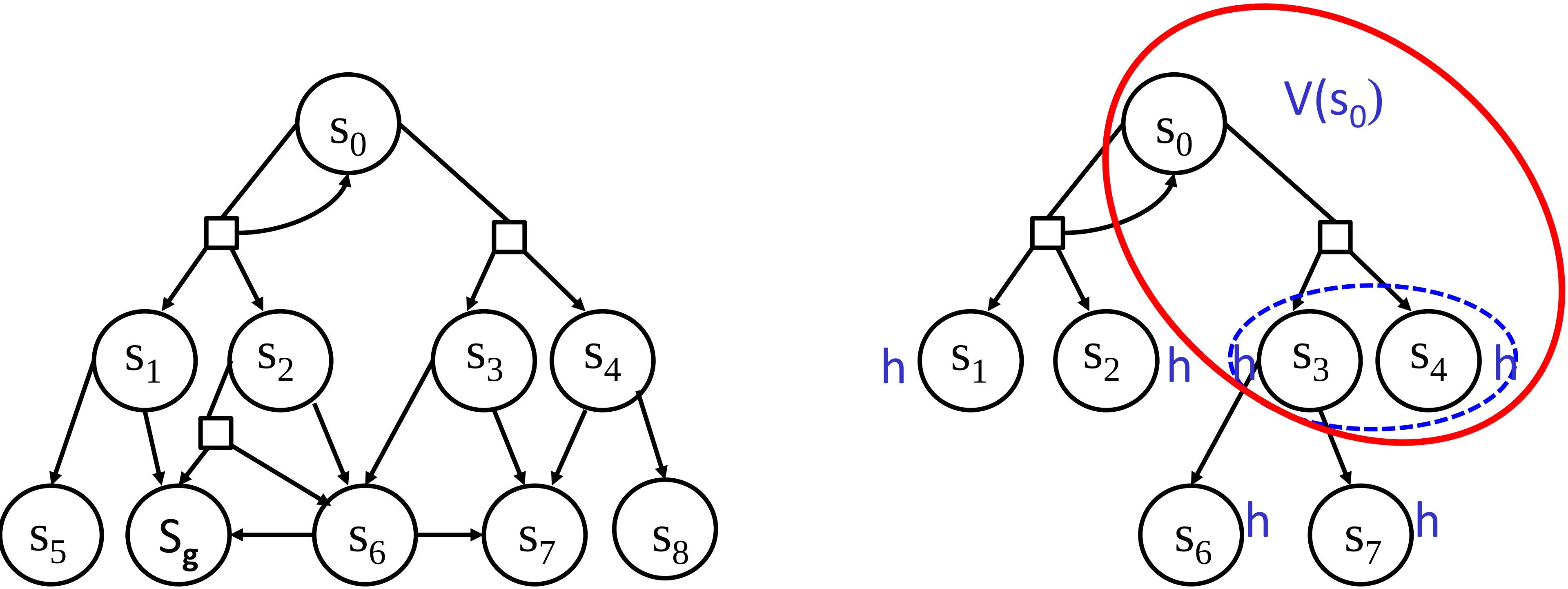
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*



FIND: expand some states on the fringe (in greedy graph)

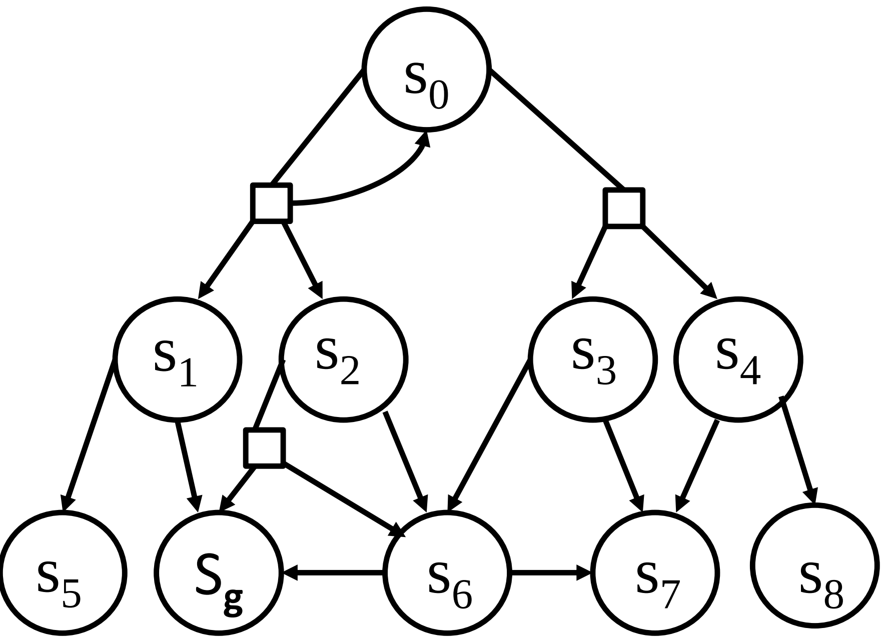
initialize all new states by their heuristic value

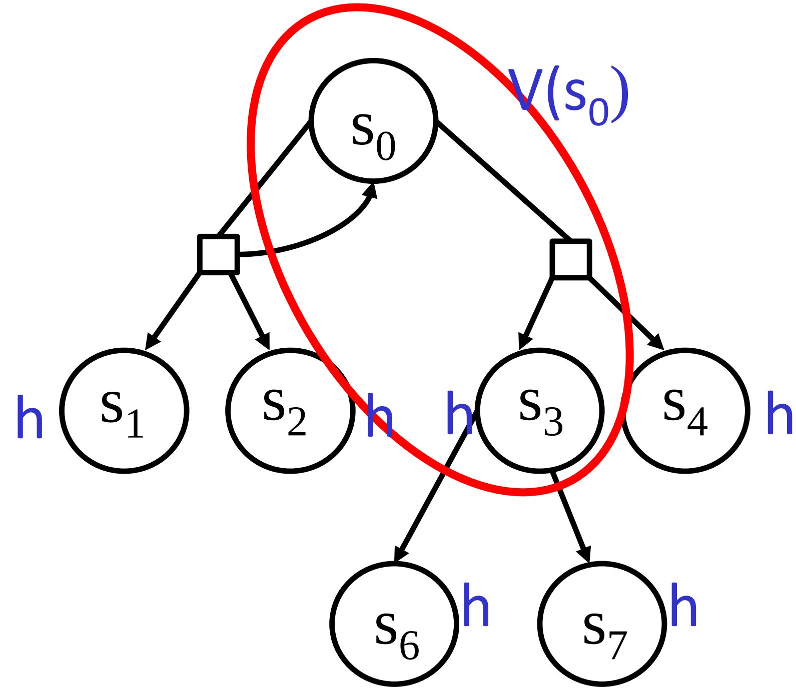
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





FIND: expand some states on the fringe (in greedy graph)

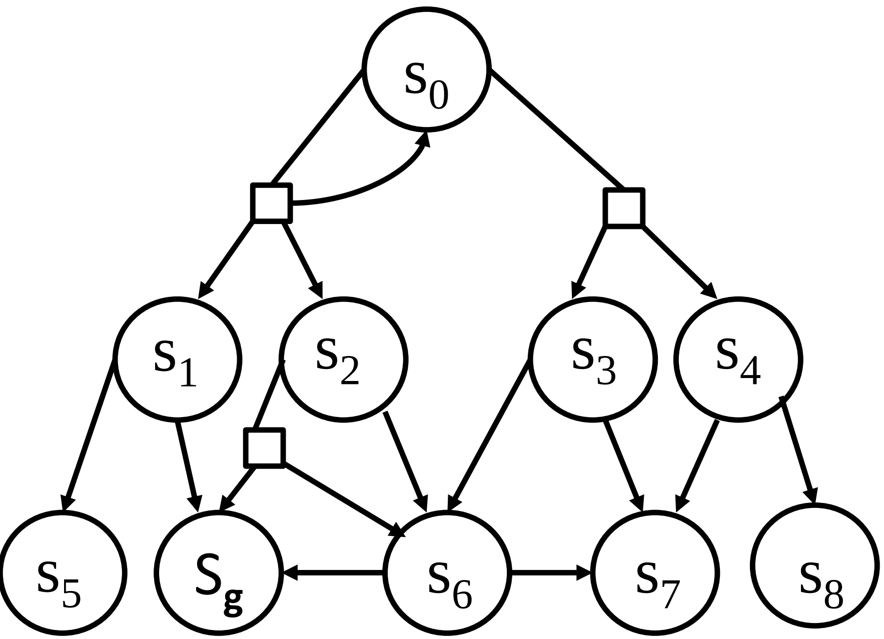
initialize all new states by their heuristic value

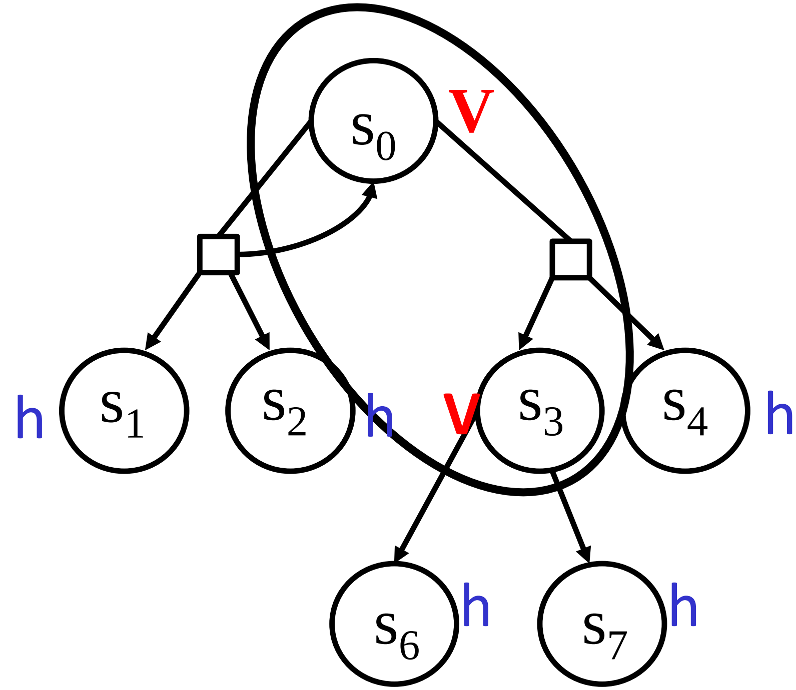
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





FIND: expand some states on the fringe (in greedy graph)

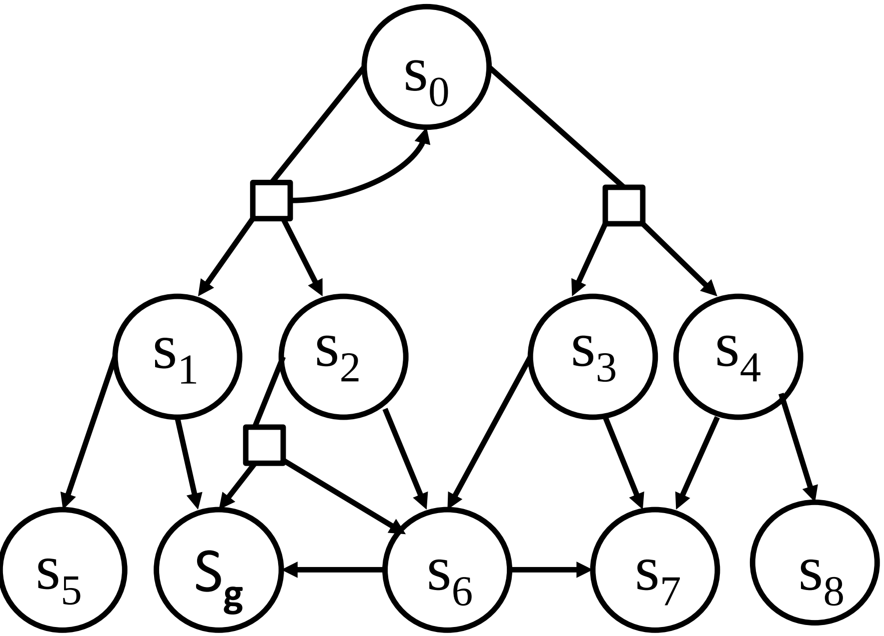
initialize all new states by their heuristic value

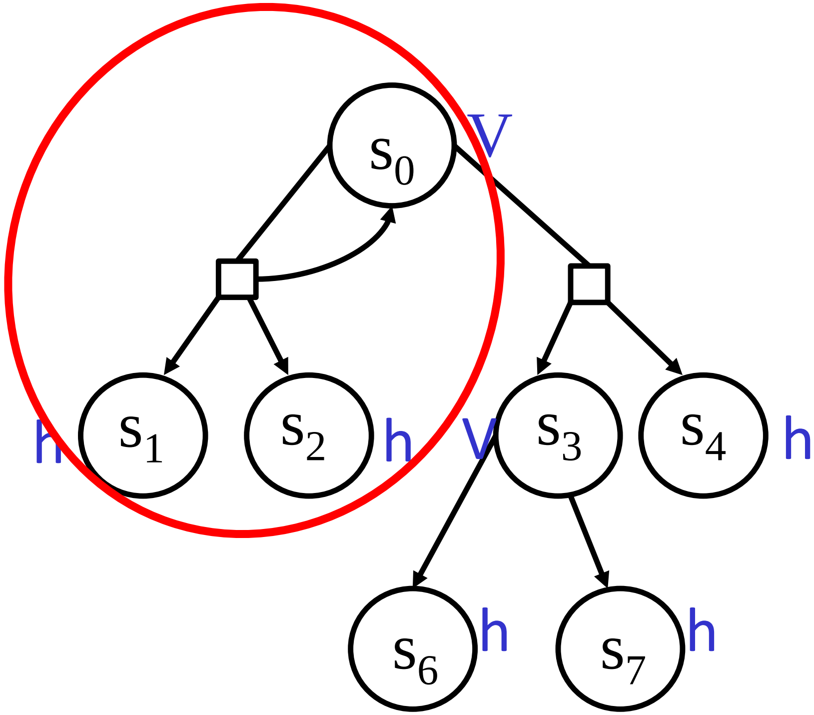
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





FIND: expand some states on the fringe (in greedy graph)

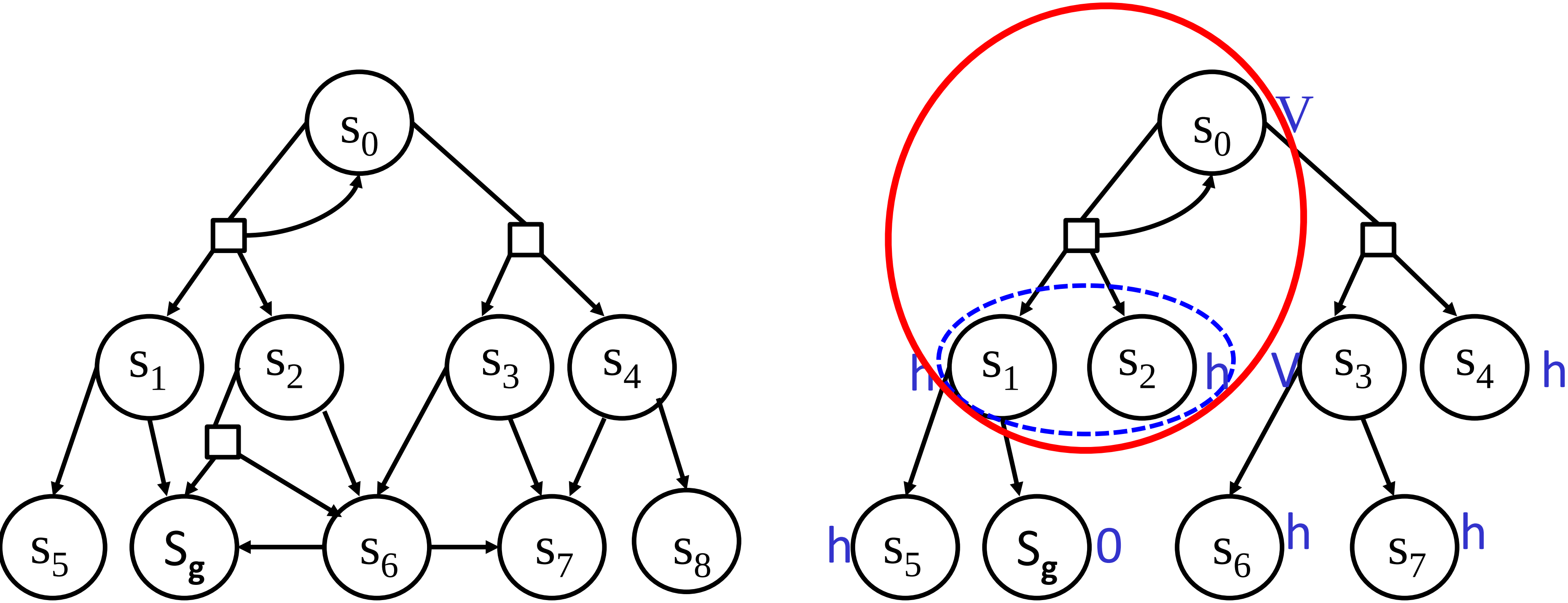
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*



FIND: expand some states on the fringe (in greedy graph)

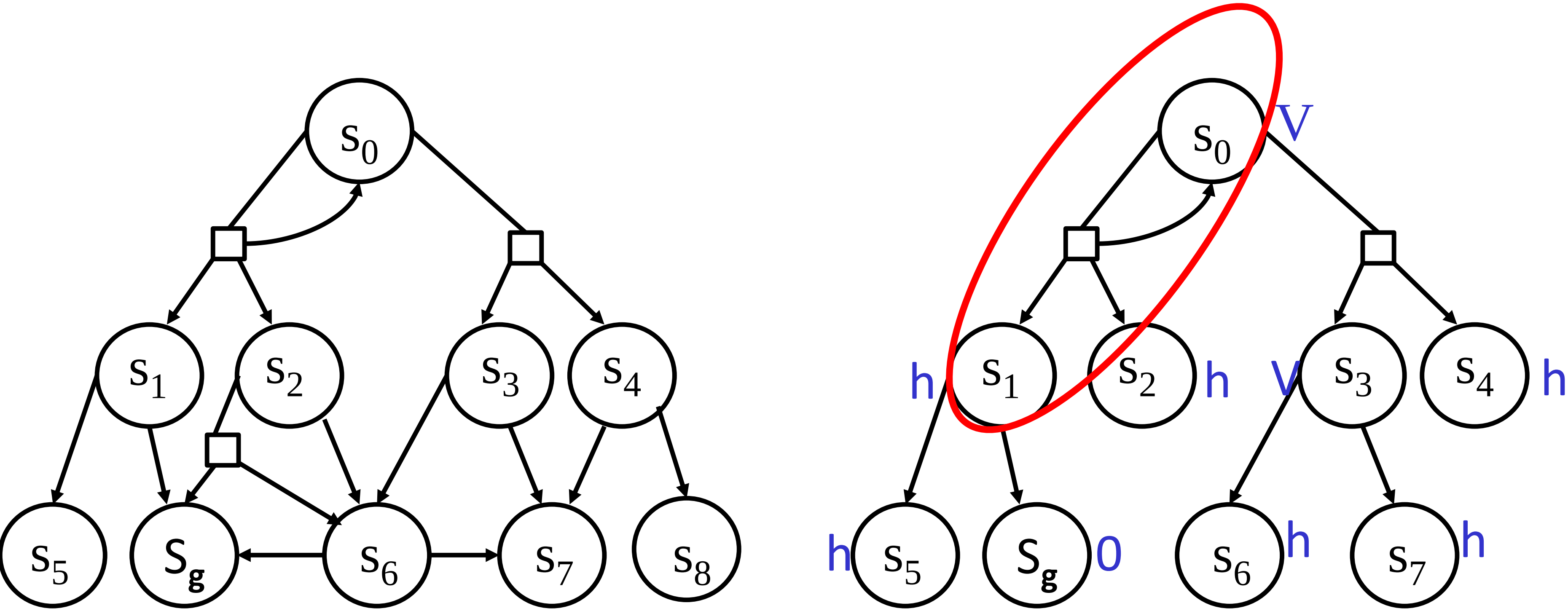
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*



FIND: expand some states on the fringe (in greedy graph)

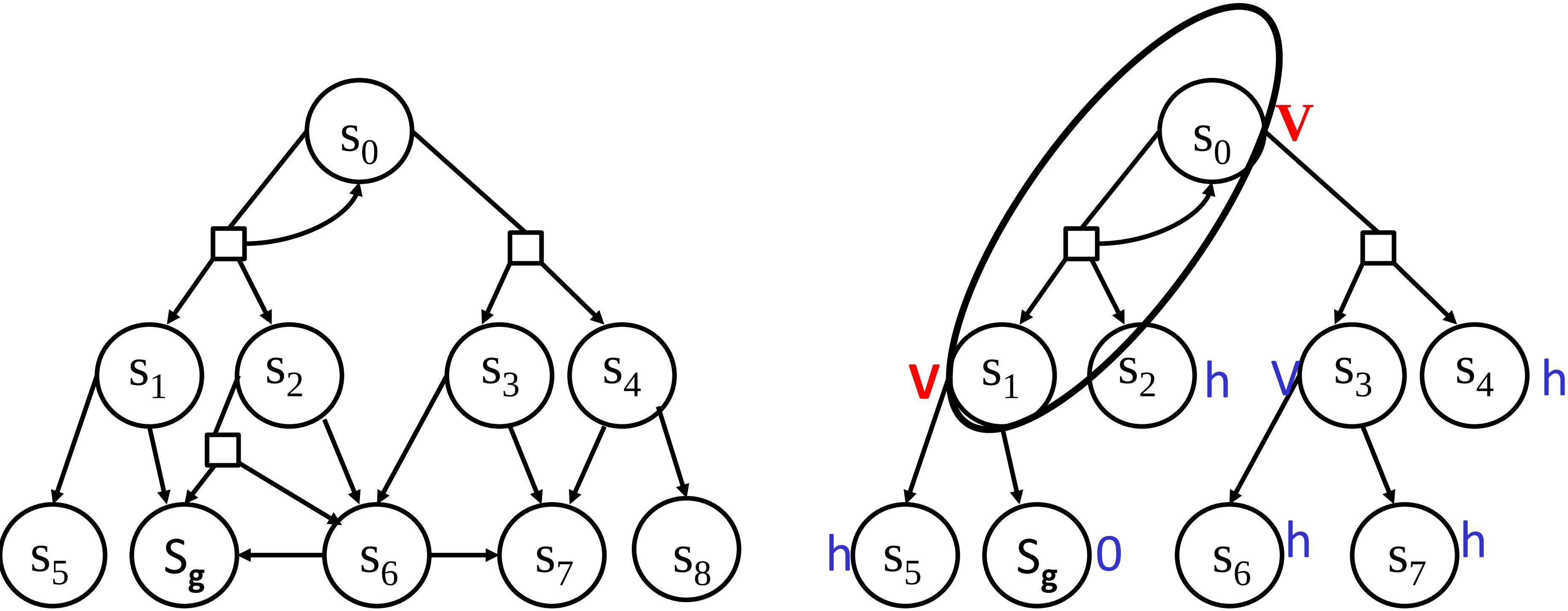
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*



FIND: expand some states on the fringe (in greedy graph)

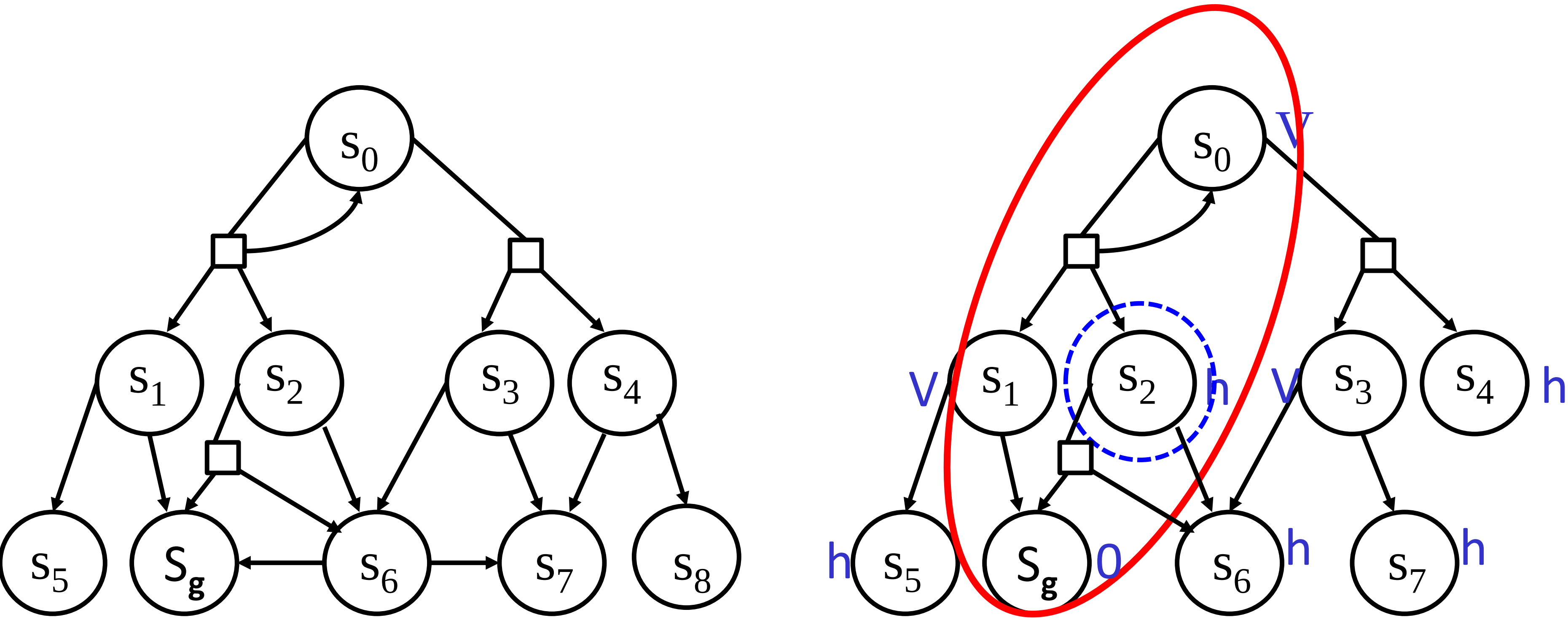
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*



FIND: expand some states on the fringe (in greedy graph)

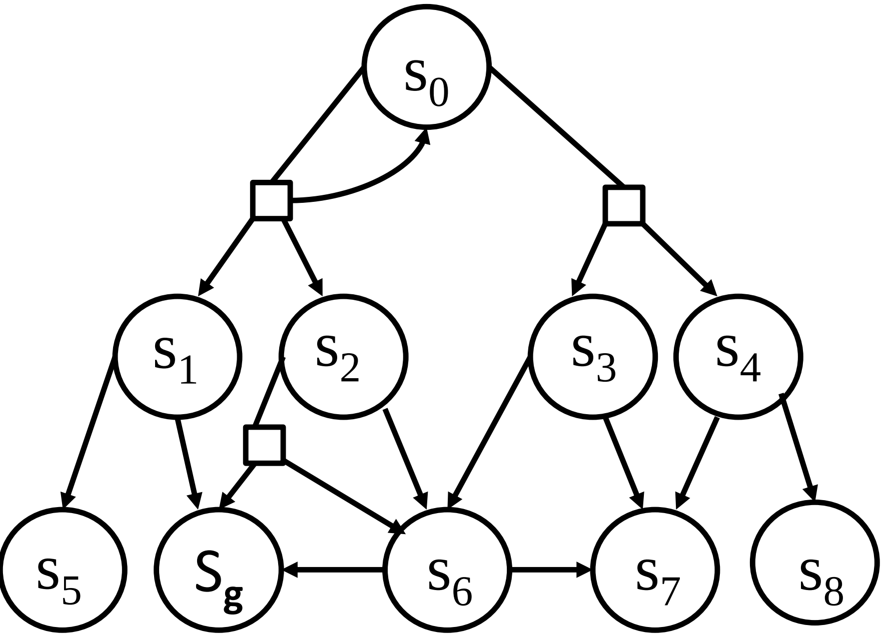
initialize all new states by their heuristic value

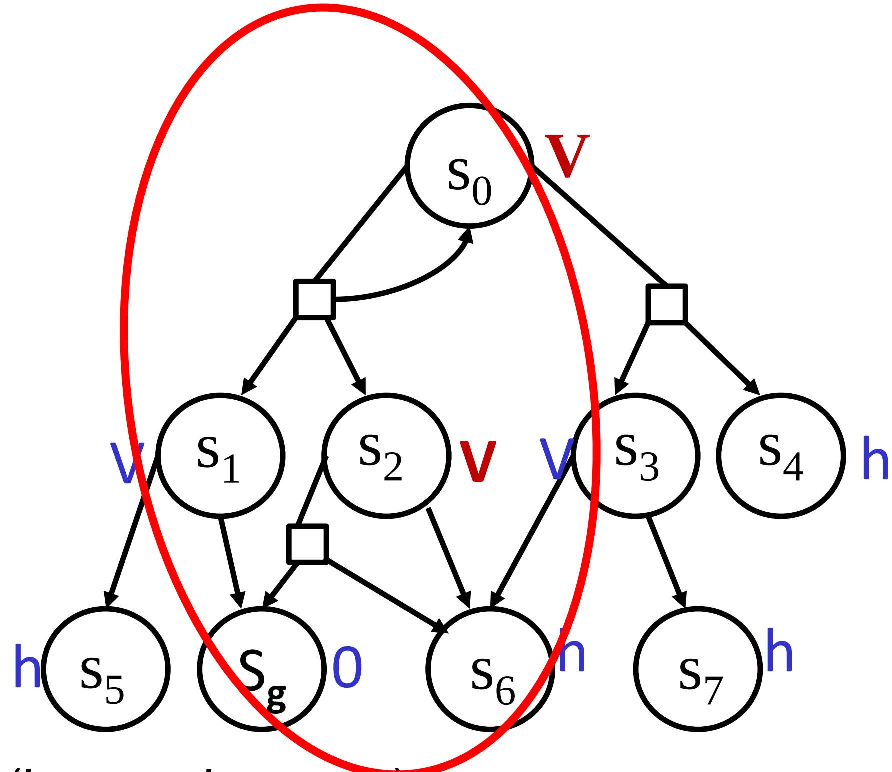
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





FIND: expand some states on the fringe (in greedy graph)

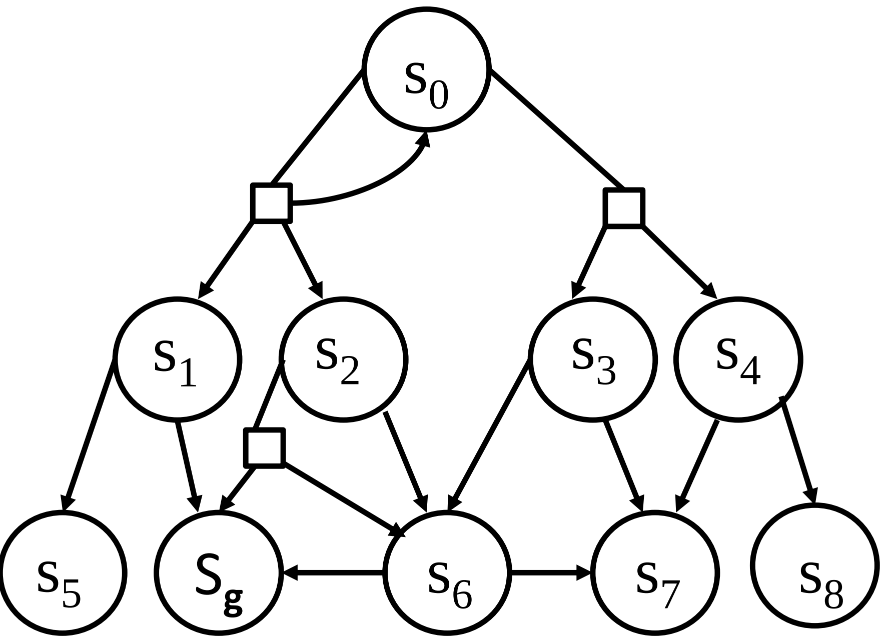
initialize all new states by their heuristic value

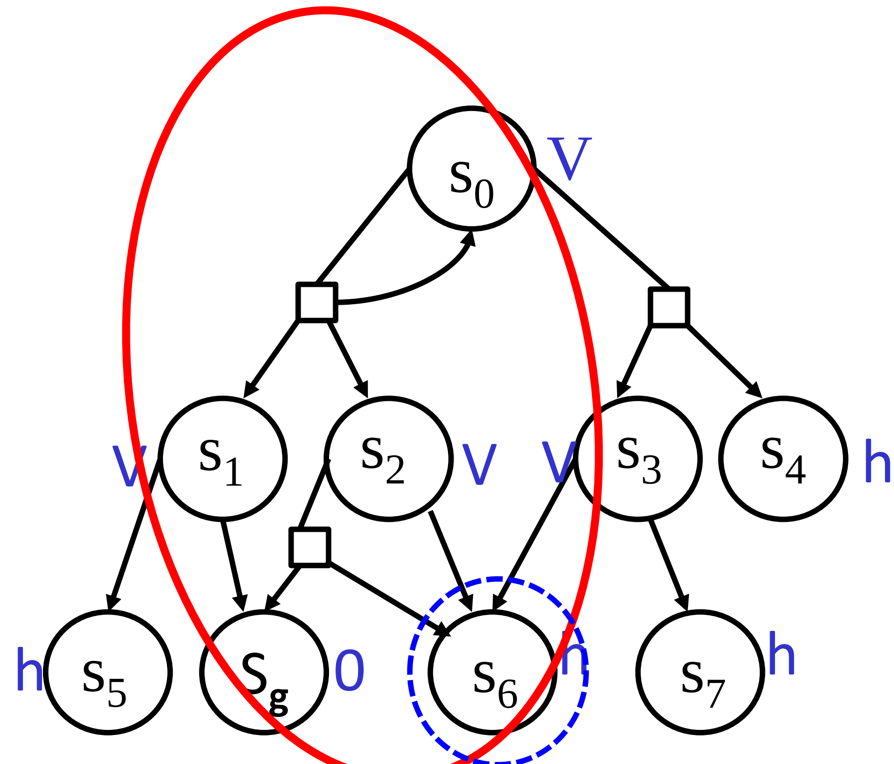
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





FIND: expand some states on the fringe (in greedy graph)

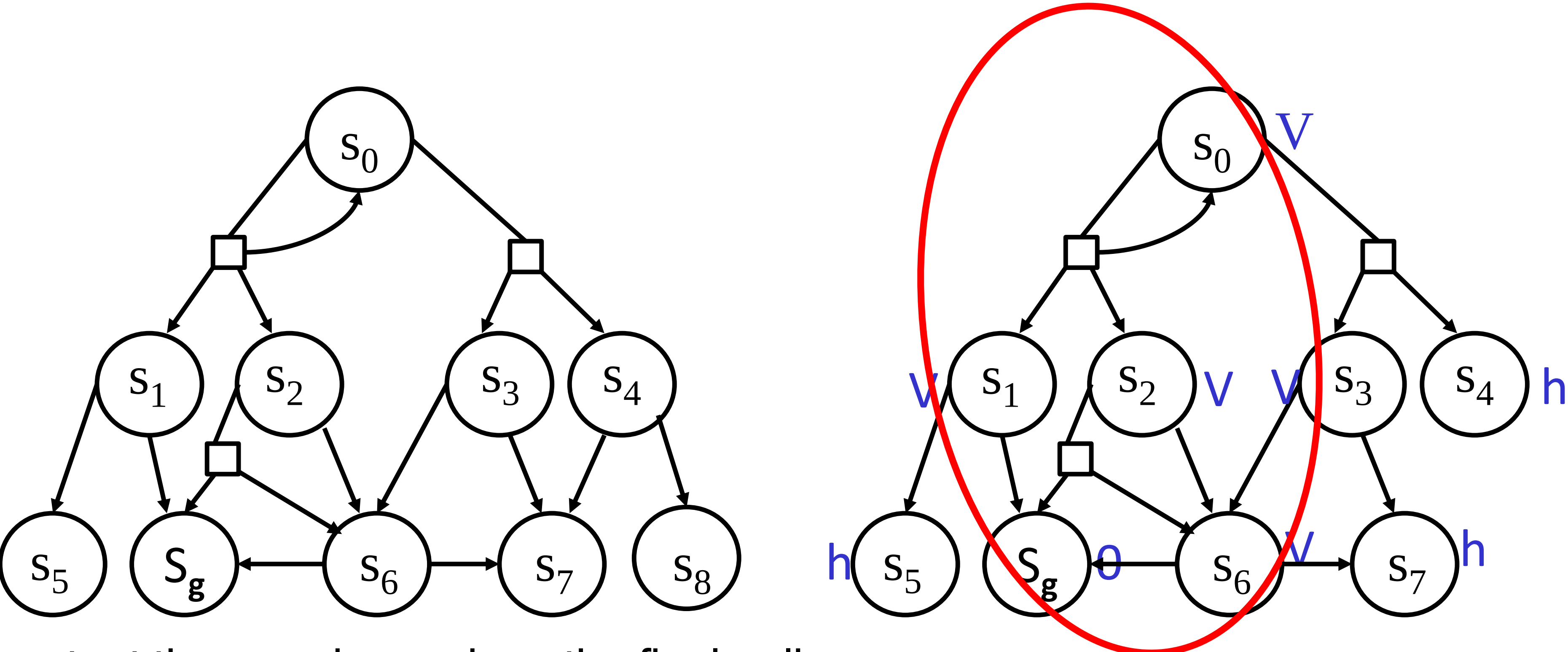
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform VI on this subset

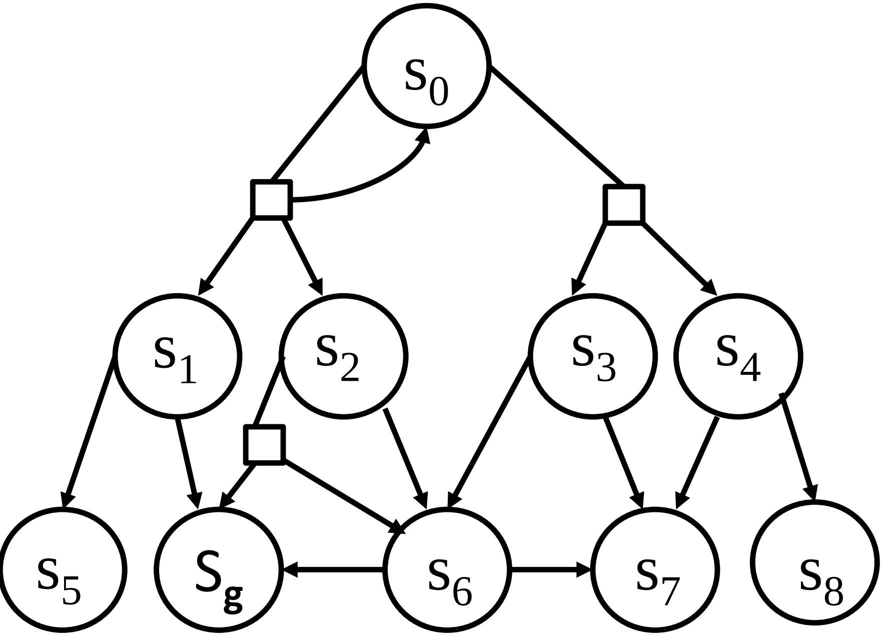
recompute the greedy graph

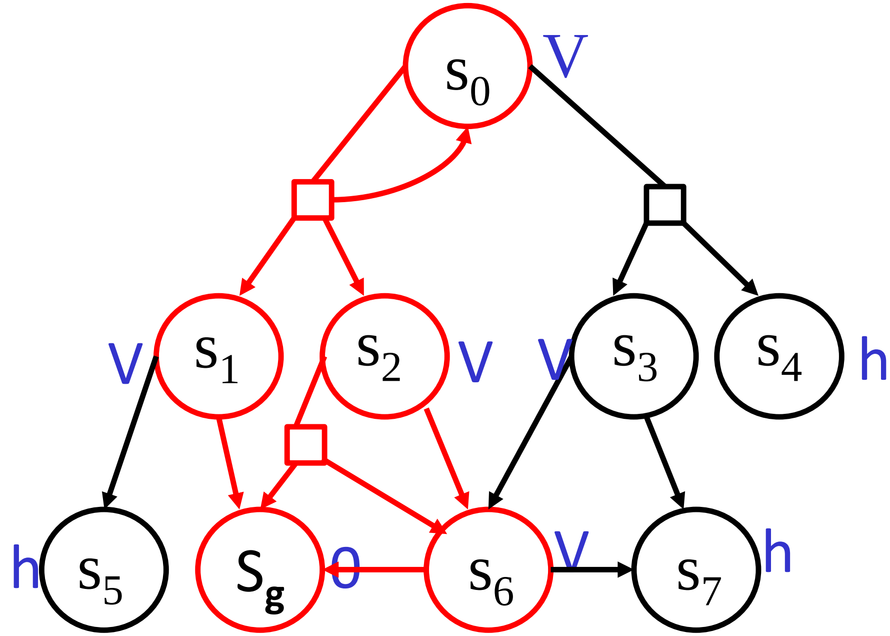
LAO*



output the greedy graph as the final policy

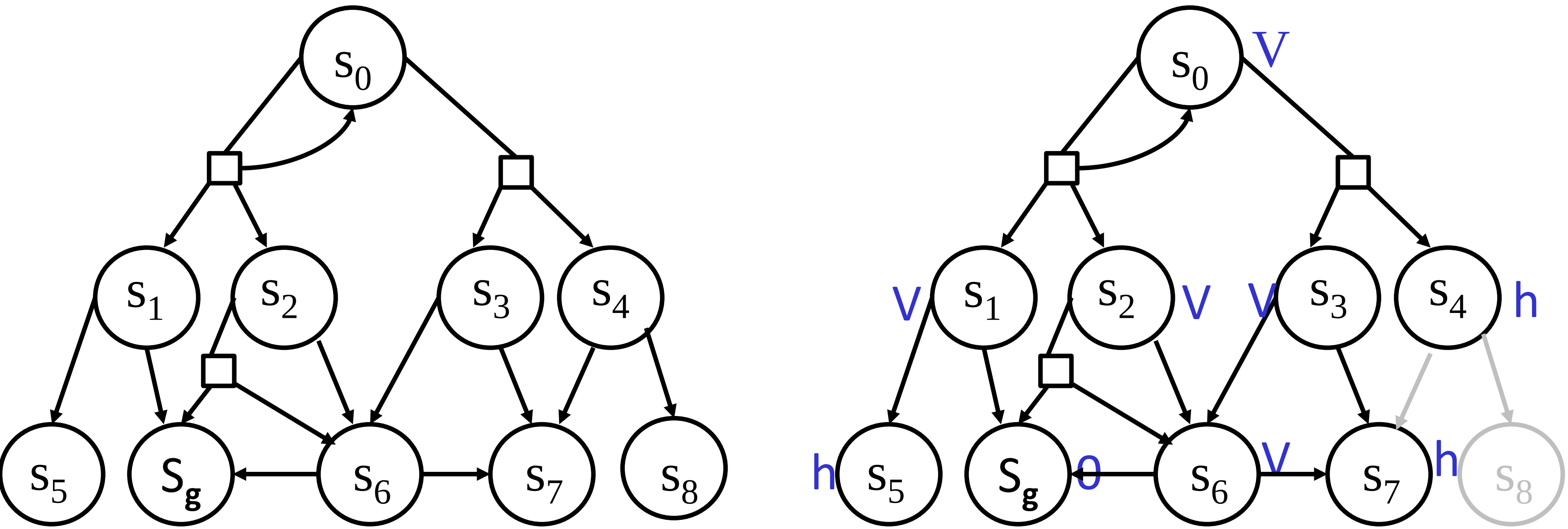
LAO*





output the greedy graph as the final policy

LAO*



s_4 was never expanded

s_8 was never touched

LAO*

add s_0 to the fringe and to greedy policy graph

one expansion

repeat

- FIND: expand best state *s* on the fringe (in greedy graph)
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- REVISE: perform VI on this subset
- recompute the greedy graph

until greedy graph has no fringe

lot of computation

output the greedy graph as the final policy

Optimizations in LAO*

add s_0 to the fringe and to greedy policy graph

repeat

- FIND: expand best state *s* on the fringe (in greedy graph)
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- VI iterations until greedy graph changes (or low residuals)
- recompute the greedy graph

until greedy graph has no fringe

output the greedy graph as the final policy

Optimizations in LAO*

add s_0 to the fringe and to greedy policy graph

repeat

- FIND: expand all states in greedy fringe
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- VI iterations until greedy graph changes (or low residuals)
- recompute the greedy graph

until greedy graph has no fringe

output the greedy graph as the final policy

iLAO*

add s_0 to the fringe and to greedy policy graph

repeat

- FIND: expand all states in greedy fringe
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- only one backup per state in greedy graph
- recompute the greedy graph

until greedy graph has no fringe

in what order?
(fringe → start)
DFS postorder

output the greedy graph as the final policy

Extensions

- Heuristic Search + Dynamic Programming
  - AO*, LAO*, RTDP, …
- Factored MDPs
  - add planning graph style heuristics
  - use goal regression to generalize better
- Hierarchical MDPs
  - hierarchy of sub-tasks, actions to scale better
- Reinforcement Learning
  - learning the probability and rewards
  - acting while learning – connections to psychology
- Partially Observable Markov Decision Processes
  - noisy sensors; partially observable environment
  - popular in robotics