CSE 573

Markov Decision Processes: Heuristic Search & Real-Time Dynamic Programming

Slides adapted from Andrey Kolobov and Mausam

Stochastic Shortest-Path MDPs: Definition

Bertsekas, 1995

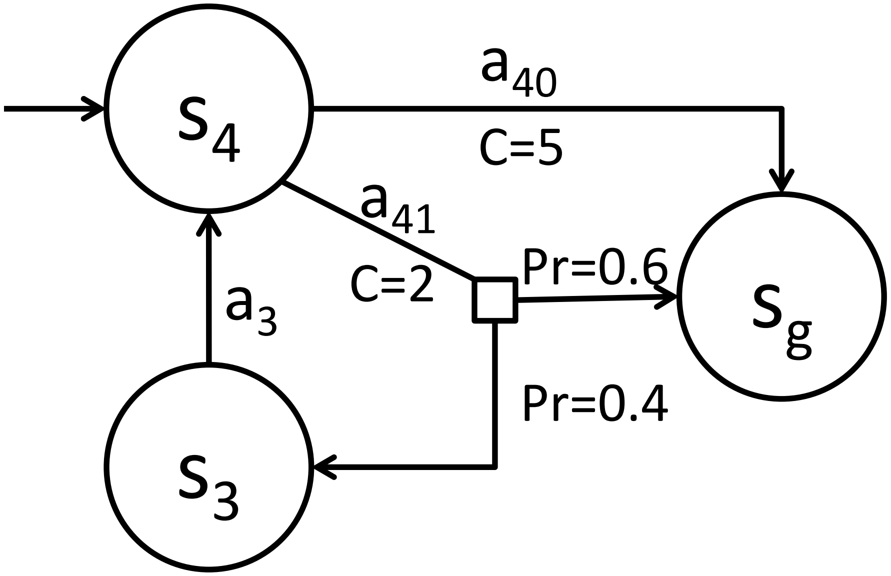
SSP MDP is a tuple <*S*, *A*, *T*, *C*, *G*>, where:

- *S* is a finite state space
- *A* is a finite action set
- *T*: *S* x *A* x *S* → [0, 1] is a stationary transition function
- *C*: *S* x *A* x *S* → $\mathbb{R}$ is a stationary *cost function* (low cost is good!)
- *G* is a set of absorbing cost-free goal states

Under two conditions:

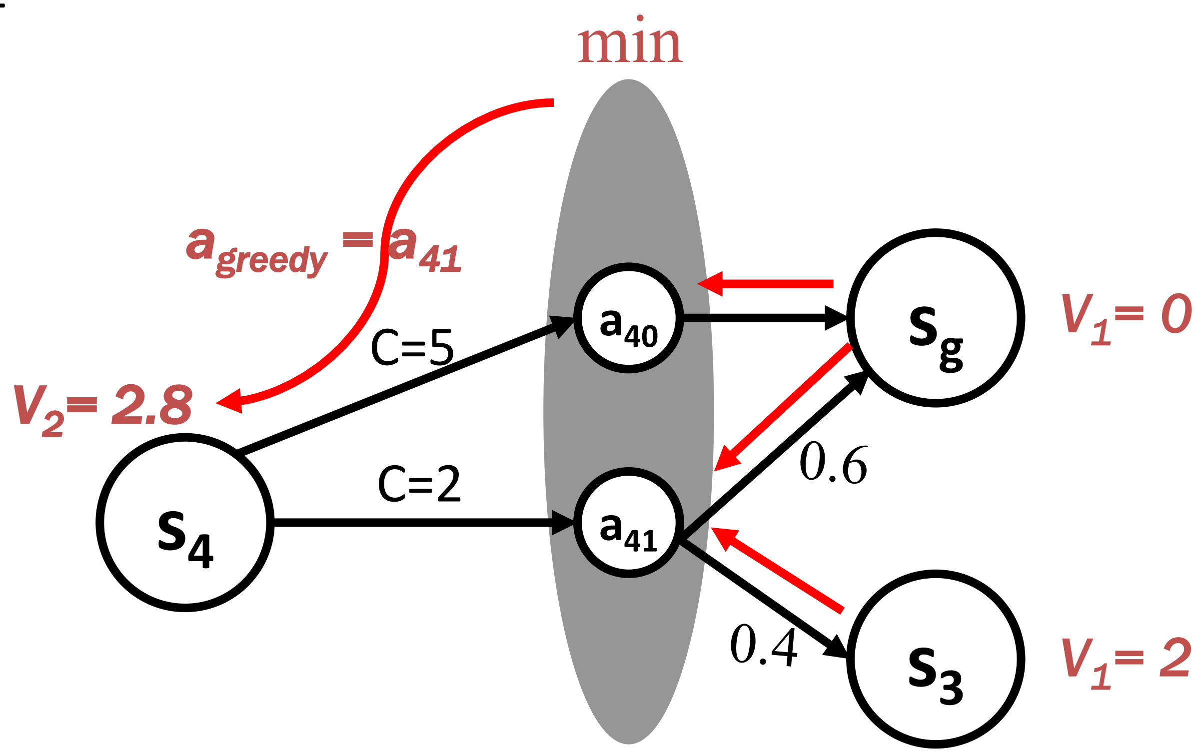
- There is a *proper policy* (reaches a goal with P= 1 from all states)
- Every *improper policy* incurs a cost of ∞ from every state from which it does not reach the goal with P=1

Bellman Backup



$Q_2(s_4,a_{40}) = 5 + 0$

$Q_2(s_4,a_{41}) = 2 + 0.6\times0$
$+ 0.4\times2$
$= 2.8$

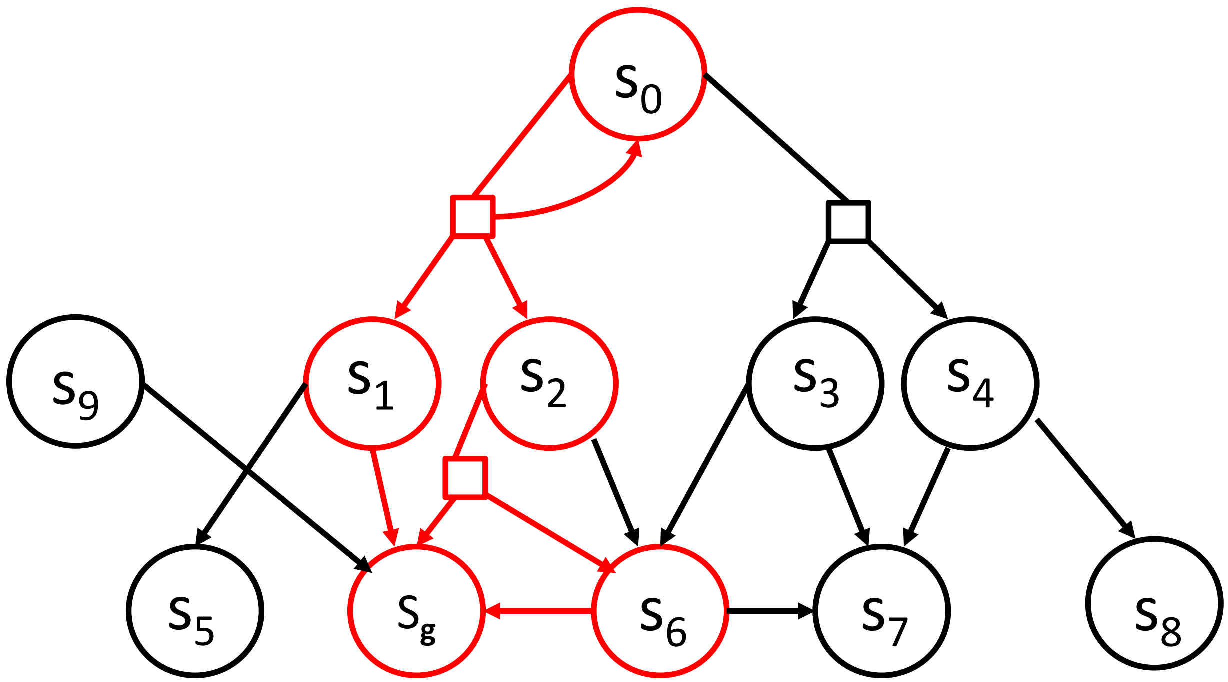


Heuristic Search

- Insight 1
  - knowledge of a start state, s_0, 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*)

Partial policy closed wrt s_0



a_1 is left action

a_2 is on right

Is this policy closed wrt s_0?

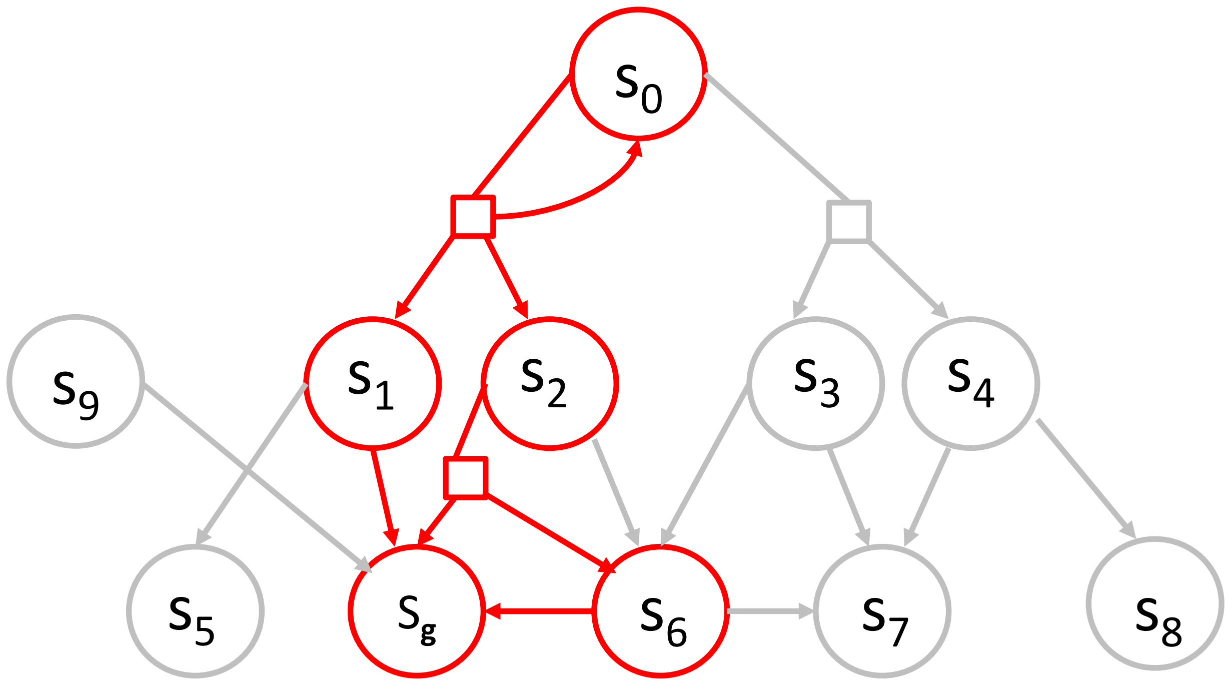
$\pi_{s0}(s_0) = a_1$

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

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

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

Policy Graph of π_{s0}



a_1 is left action
a_2 is on right

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

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 $\pi^V_{s_0}$
- Define *greedy policy graph*
  - Policy graph of $\pi^V_{s_0}$: denoted by $G^V_{s_0}$

Heuristic Function

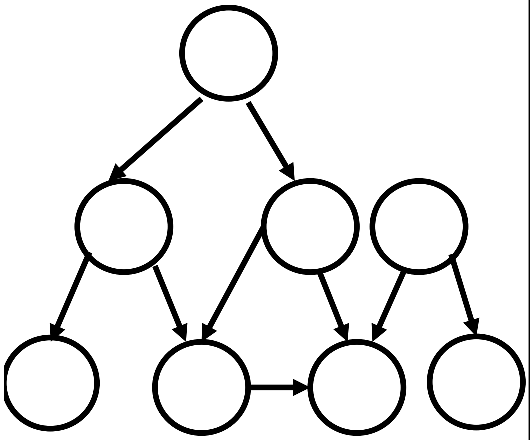
- h(s): S→R
  - estimates V*(s)
  - gives an indication about "goodness" of a state
  - usually used in initialization $V_0(s) = h(s)$
  - helps us avoid seemingly bad states

- Define *admissible* heuristic
  - Optimistic (underestimates cost)
  - $h(s) \le V^*(s)$

Heuristic Search Algorithms

- Definitions
- Find & Revise Scheme.
- LAO* and Extensions
- RTDP and Extensions
- Other uses of Heuristics/Bounds
- Heuristic Design

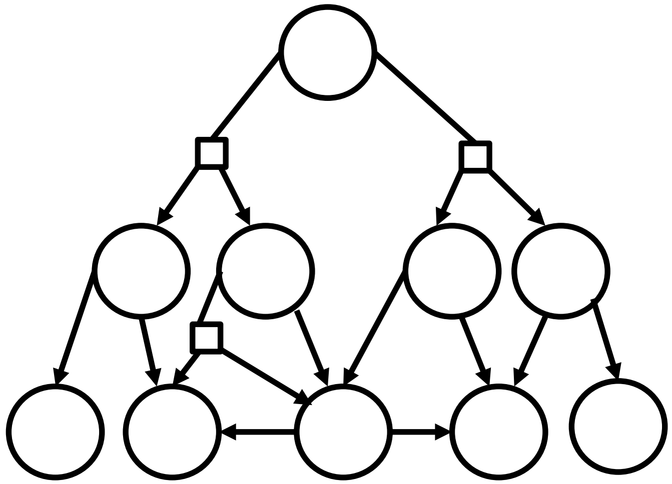
A* → LAO*



regular graph

soln:(shortest) path

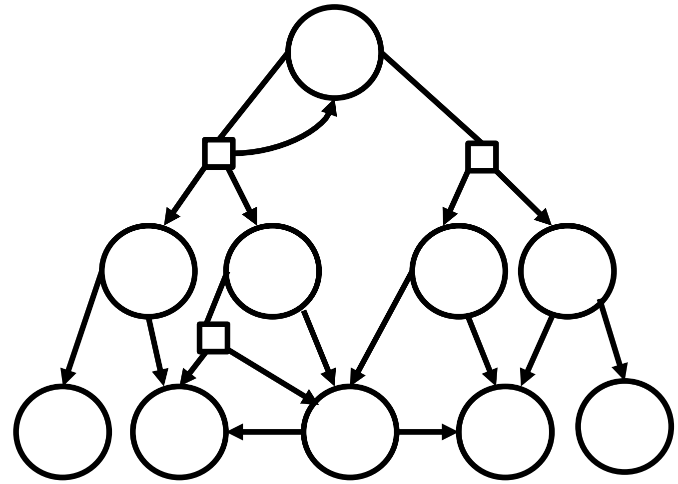
A*



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]

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
- perform some REVISE computations on this subset
- recompute the greedy graph

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

output the greedy graph as the final policy

LAO* [Hansen&Zilberstein 98]

add s_0 to the fringe and to greedy policy graph

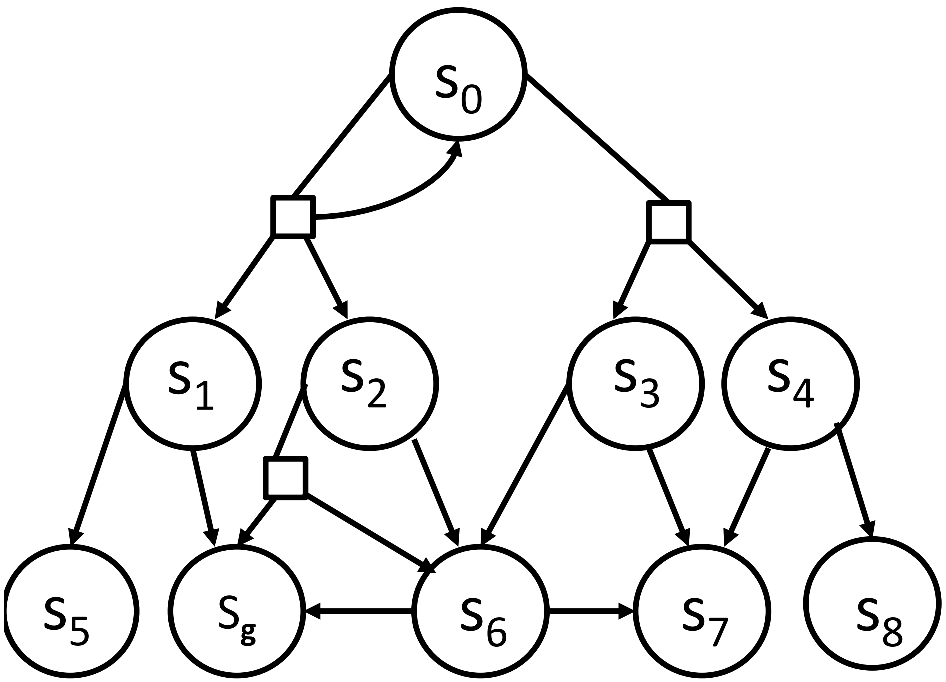
repeat

- 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
- perform PI on this subset
- recompute the greedy graph

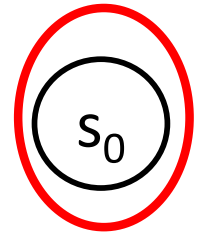
until greedy graph has no fringe ~~& residuals in greedy graph are small~~

output the greedy graph as the final policy

LAO*

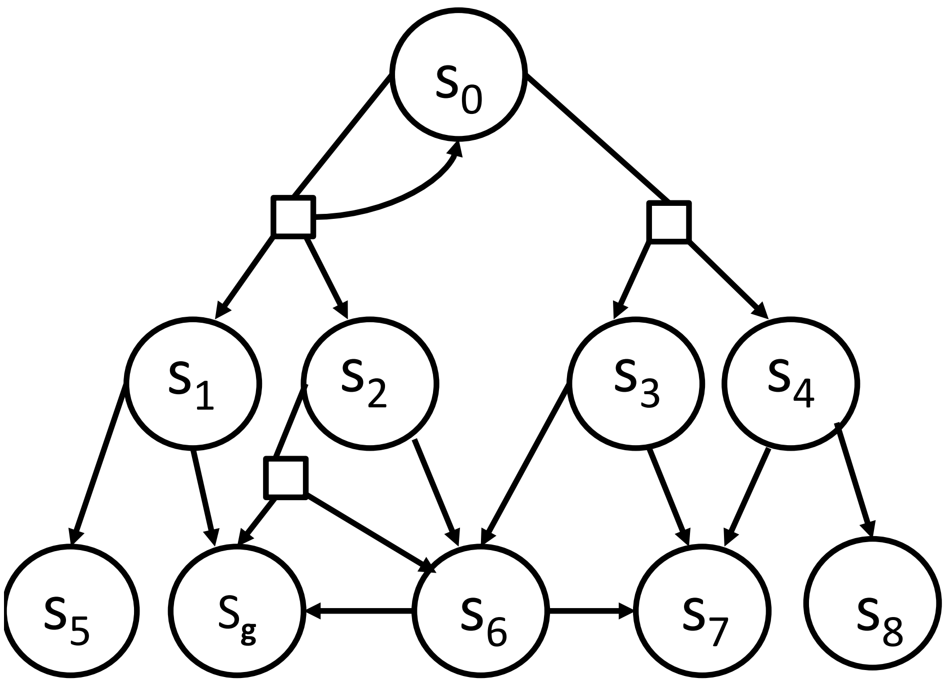


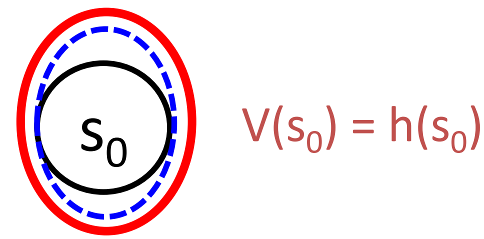
add s_0 in the fringe and in greedy graph



$V(s_0) = h(s_0)$

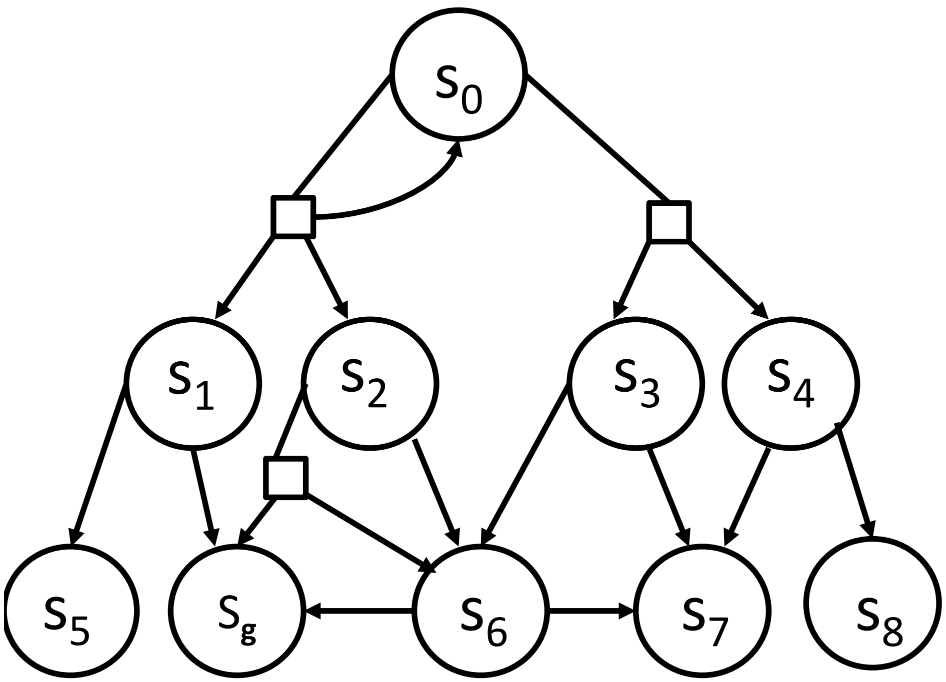
LAO*

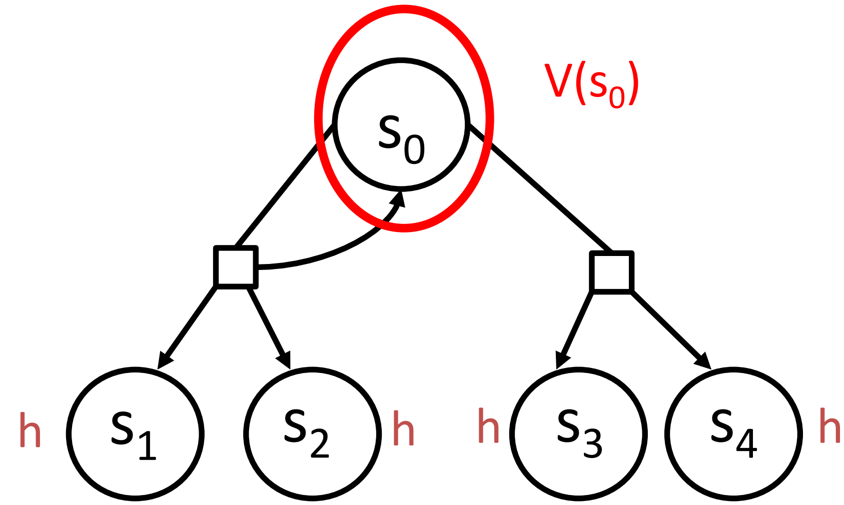




FIND: expand the best state, s_0, on the fringe (in greedy graph)

LAO*





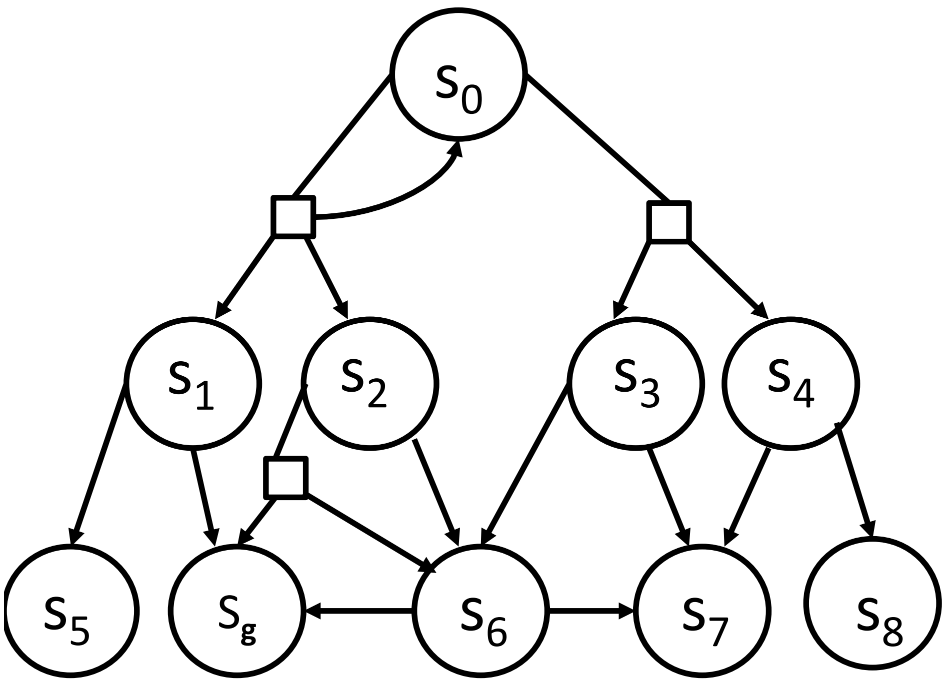
FIND: expand the best state on the fringe (in greedy graph)

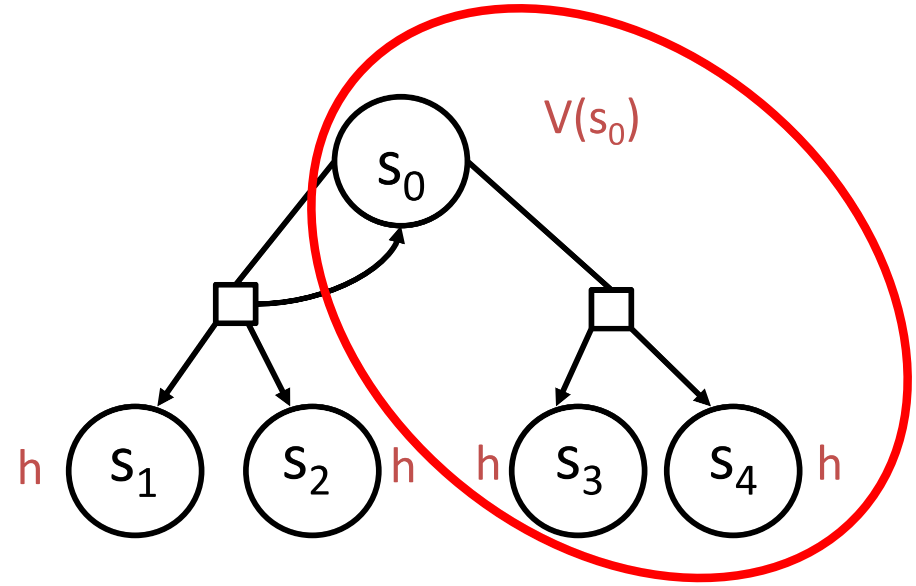
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach $s = s_0$

perform PI on this subset

LAO*





FIND: expand the best state on the fringe (in greedy graph)

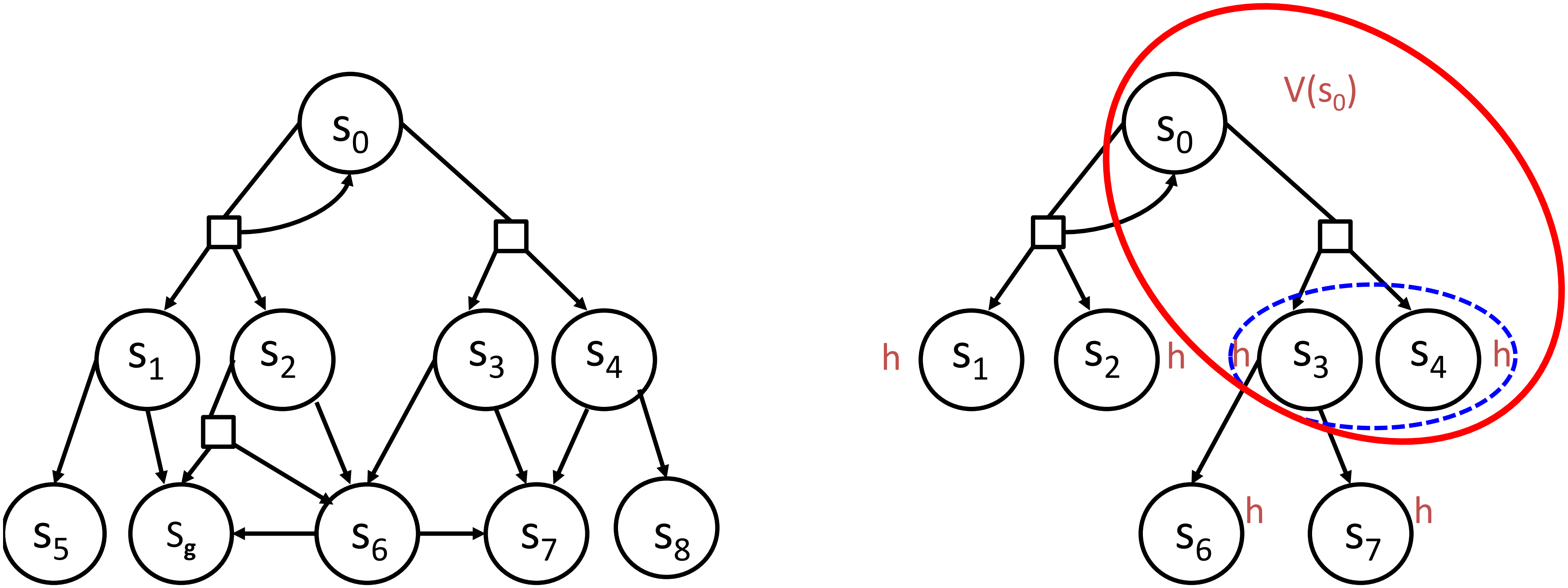
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state, s_3, on the fringe (in greedy graph)

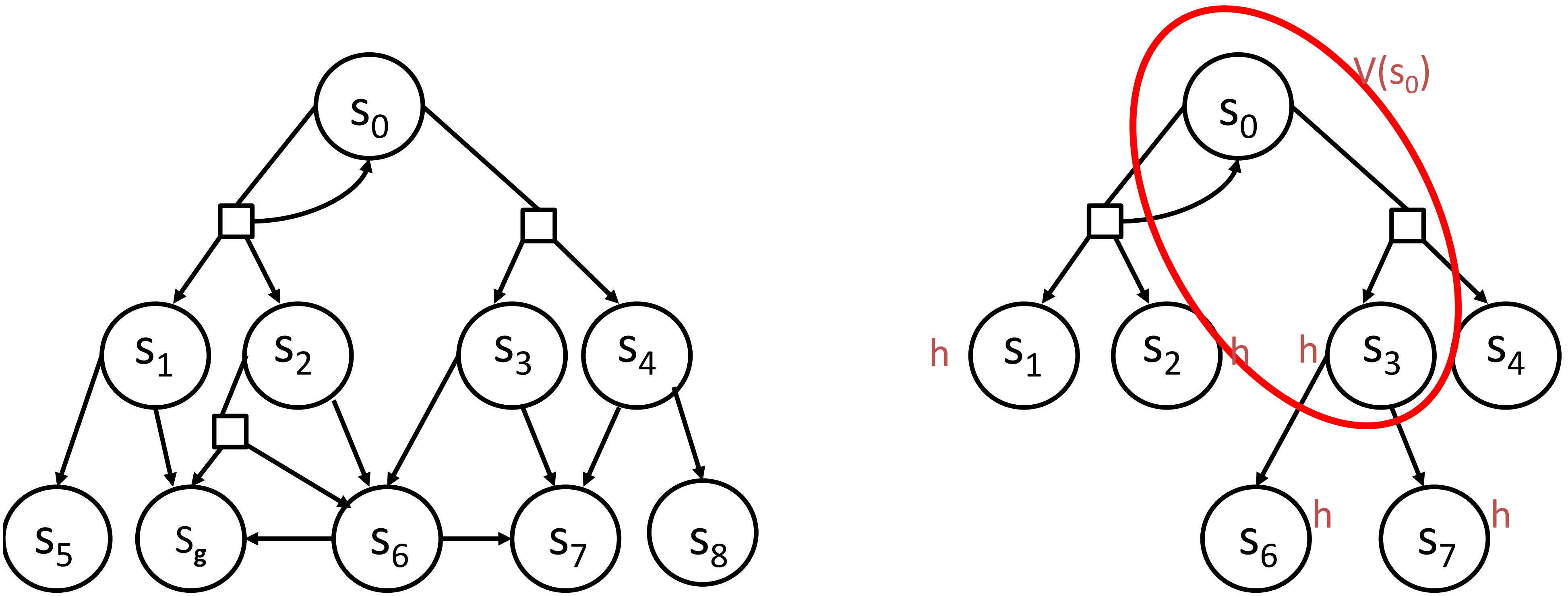
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state on the fringe (in greedy graph)

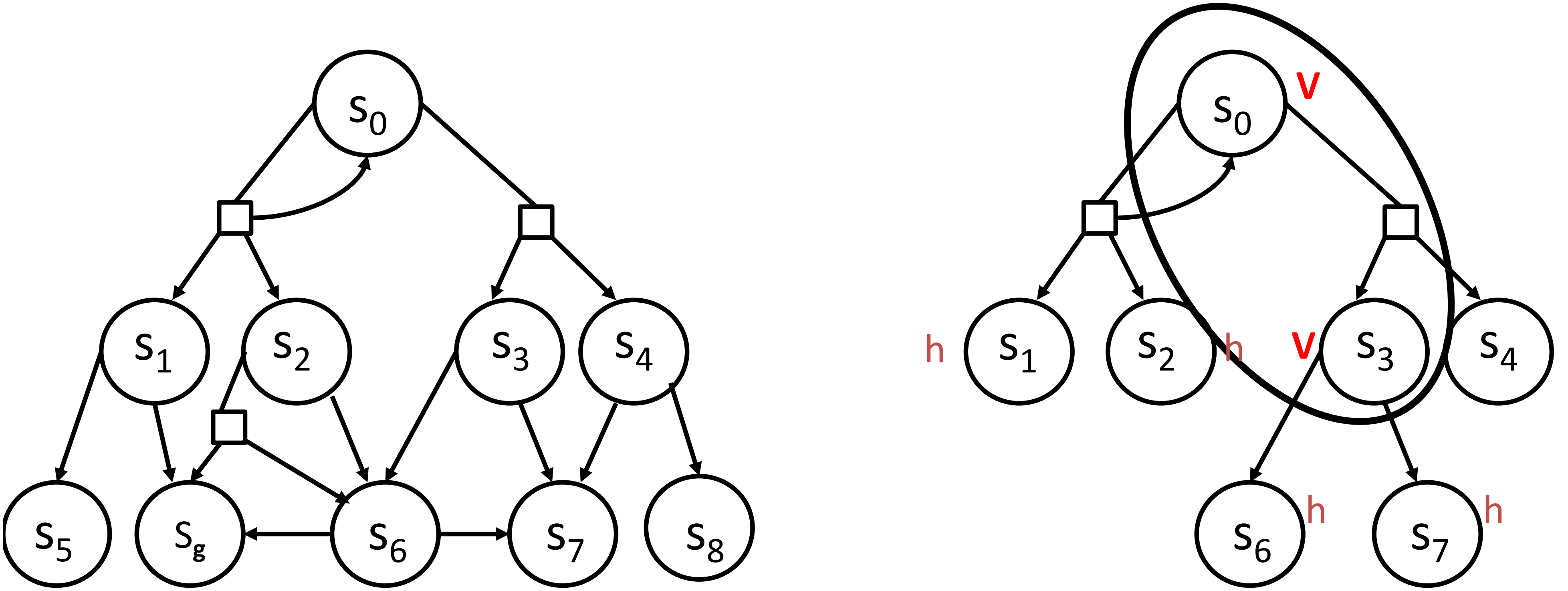
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach $s = s_3$

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state on the fringe (in greedy graph)

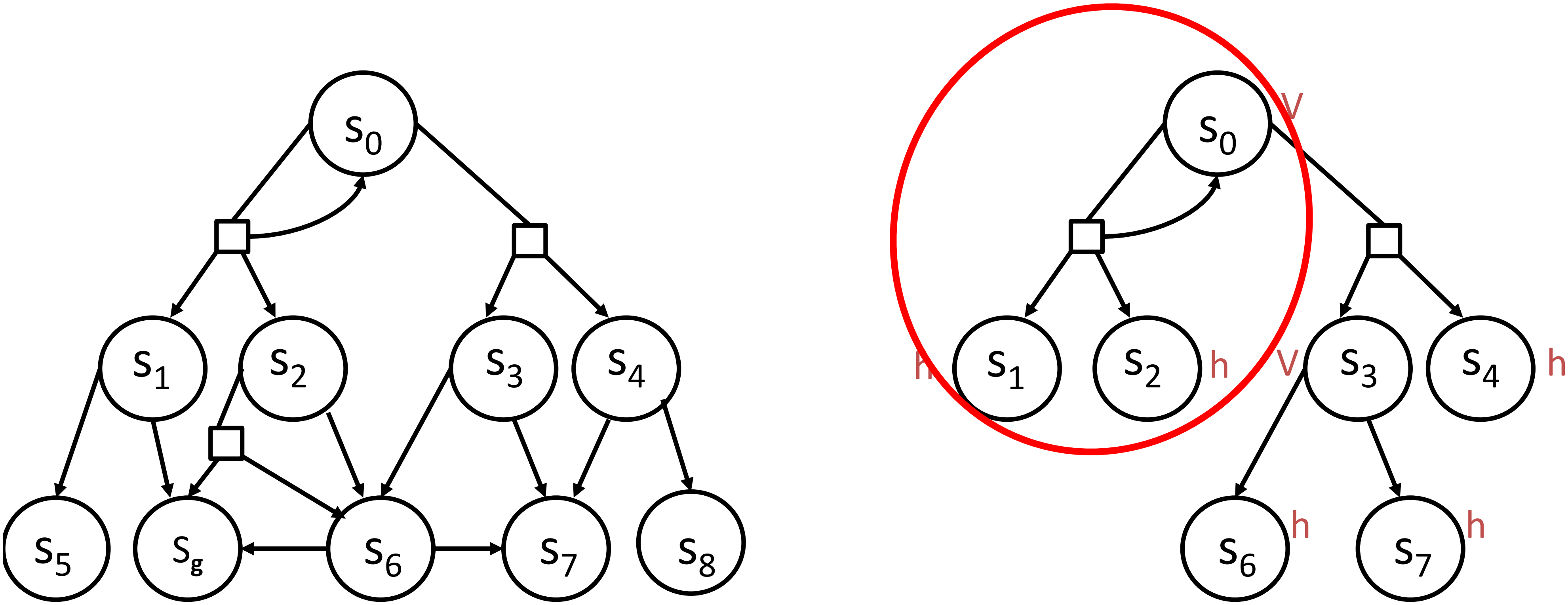
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state on the fringe (in greedy graph)

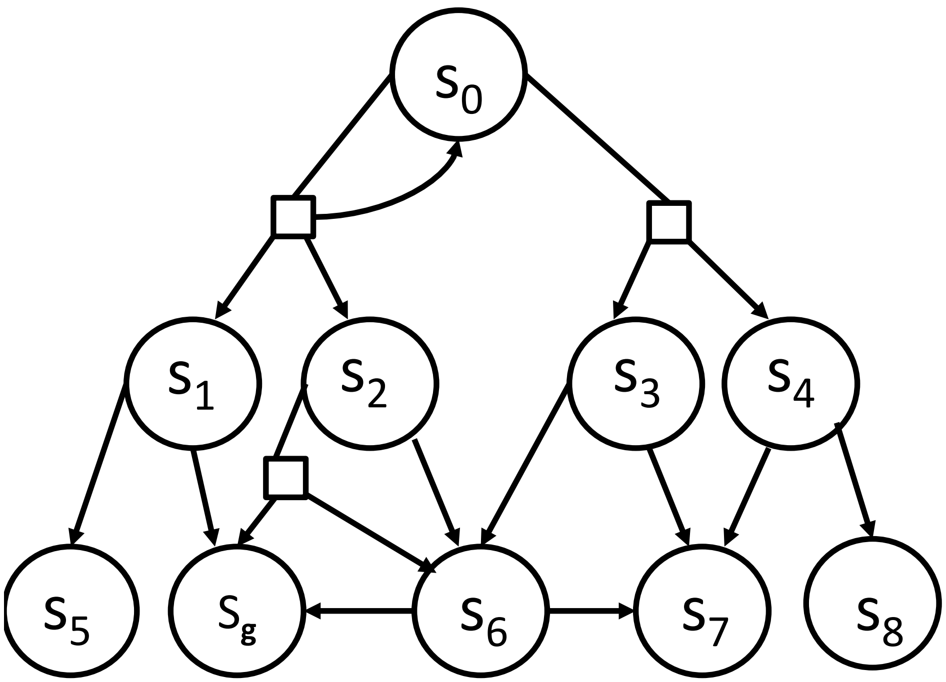
initialize all new states by their heuristic value

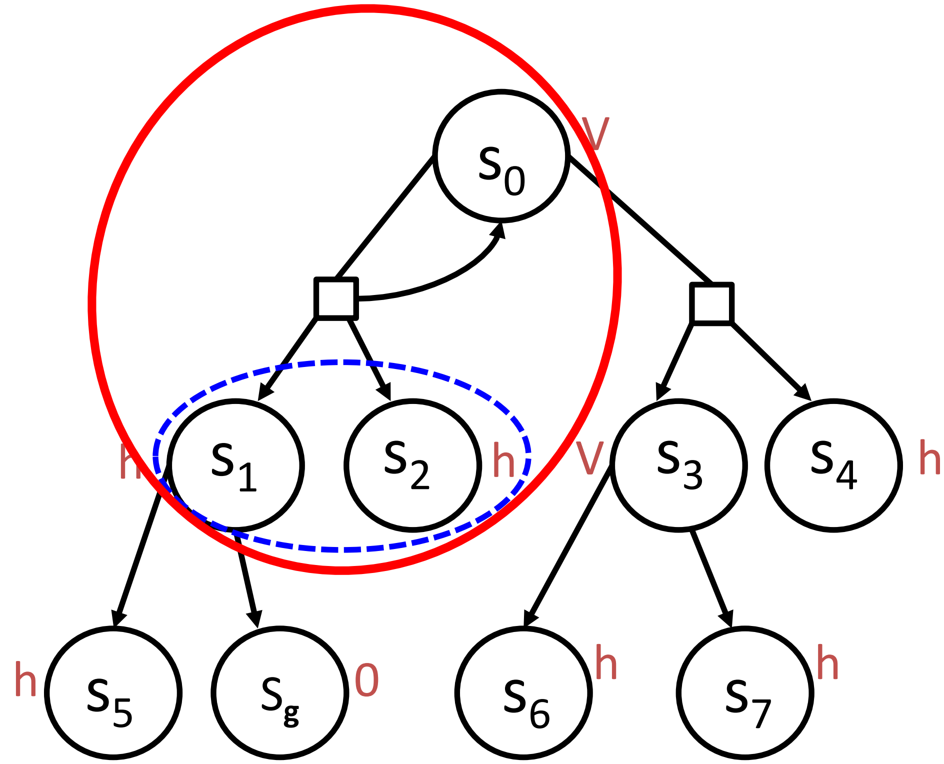
subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*





FIND: expand the best state on the fringe (in greedy graph)

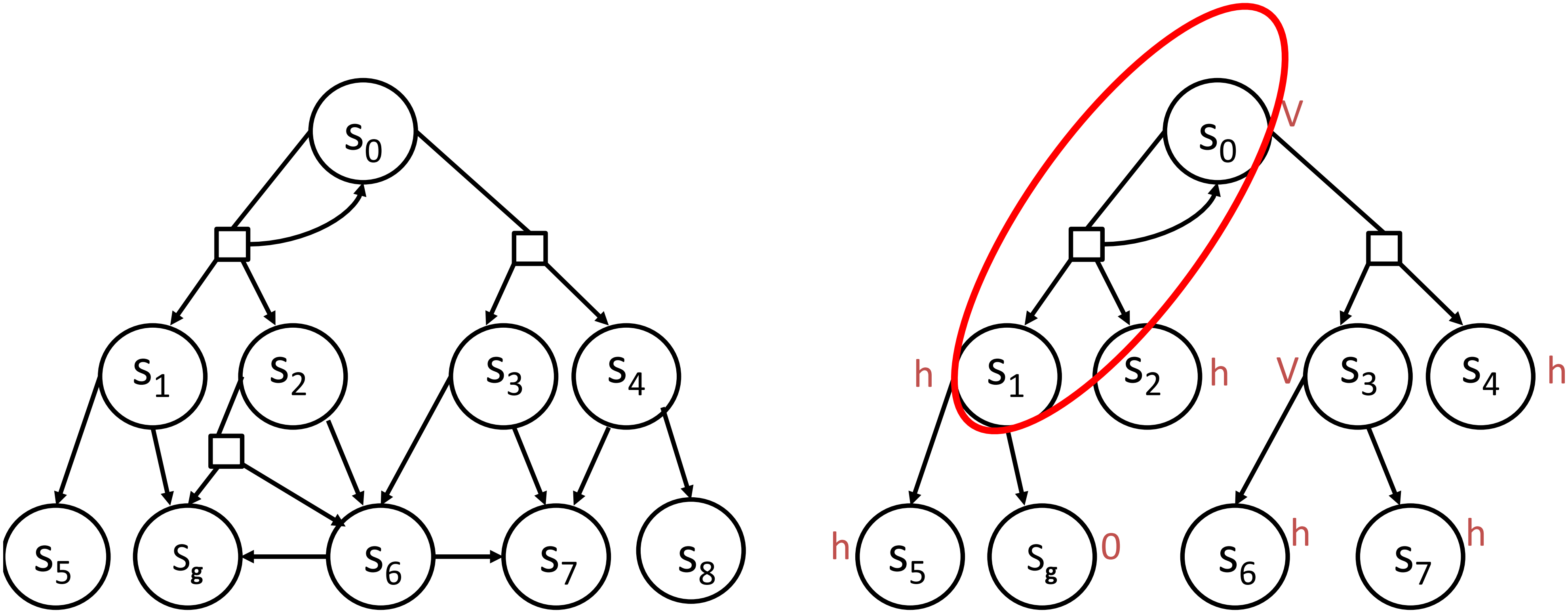
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

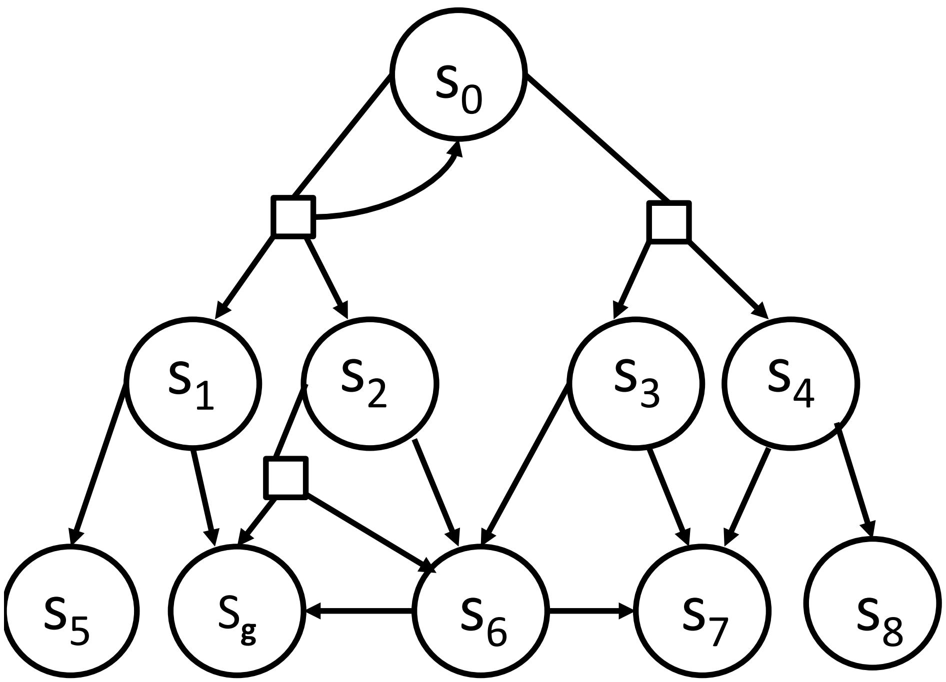
initialize all new states by their heuristic value

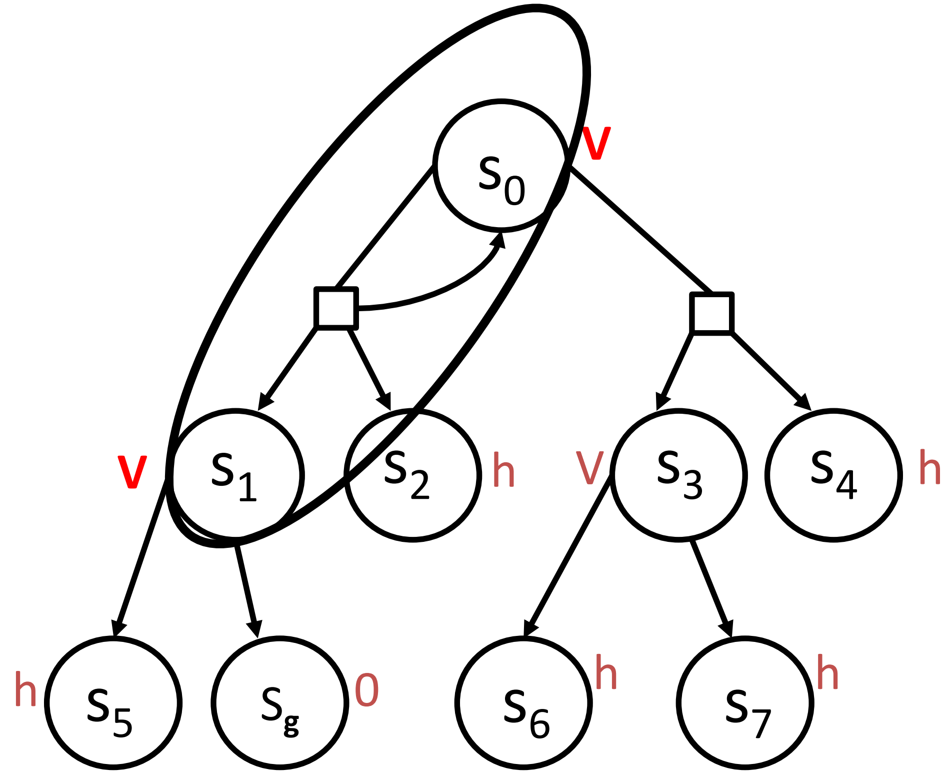
subset = all states in expanded graph that can reach s = s_1

perform PI on this subset

recompute the greedy graph

LAO*





FIND: expand the best state on the fringe (in greedy graph)

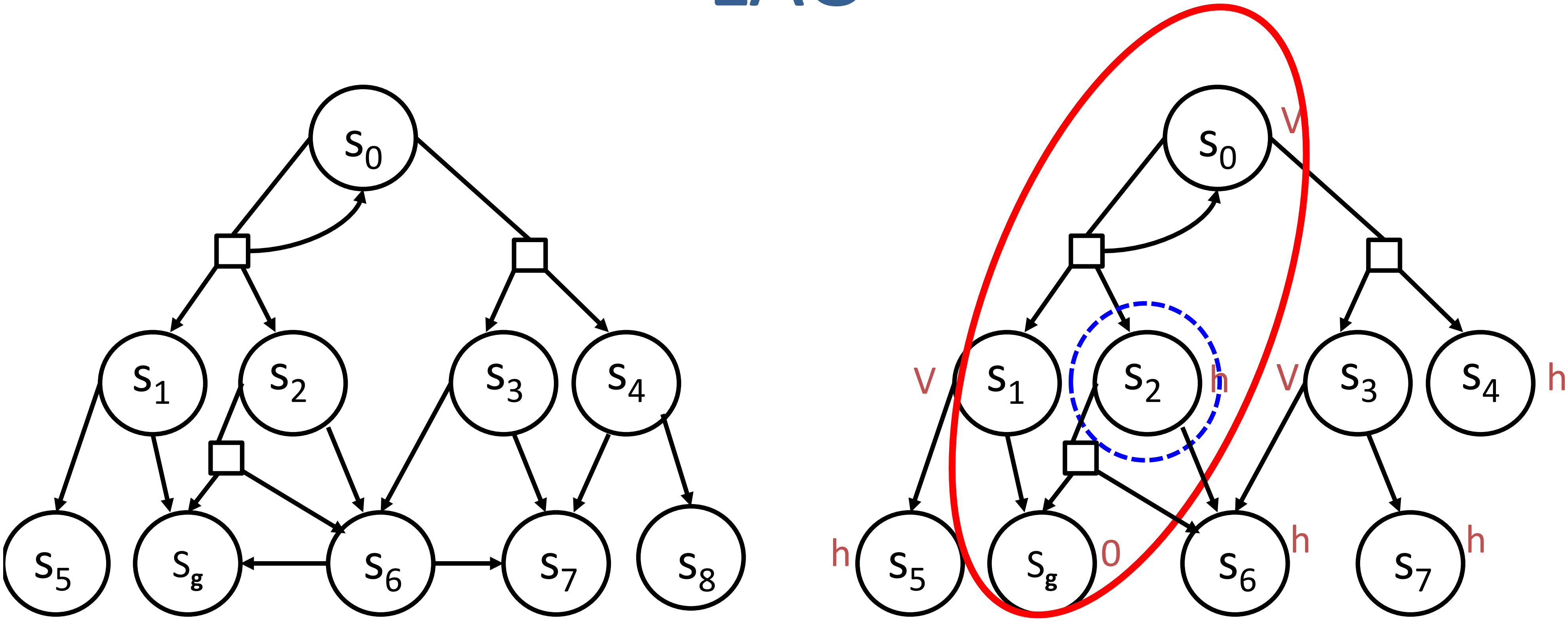
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state on the fringe (in greedy graph)

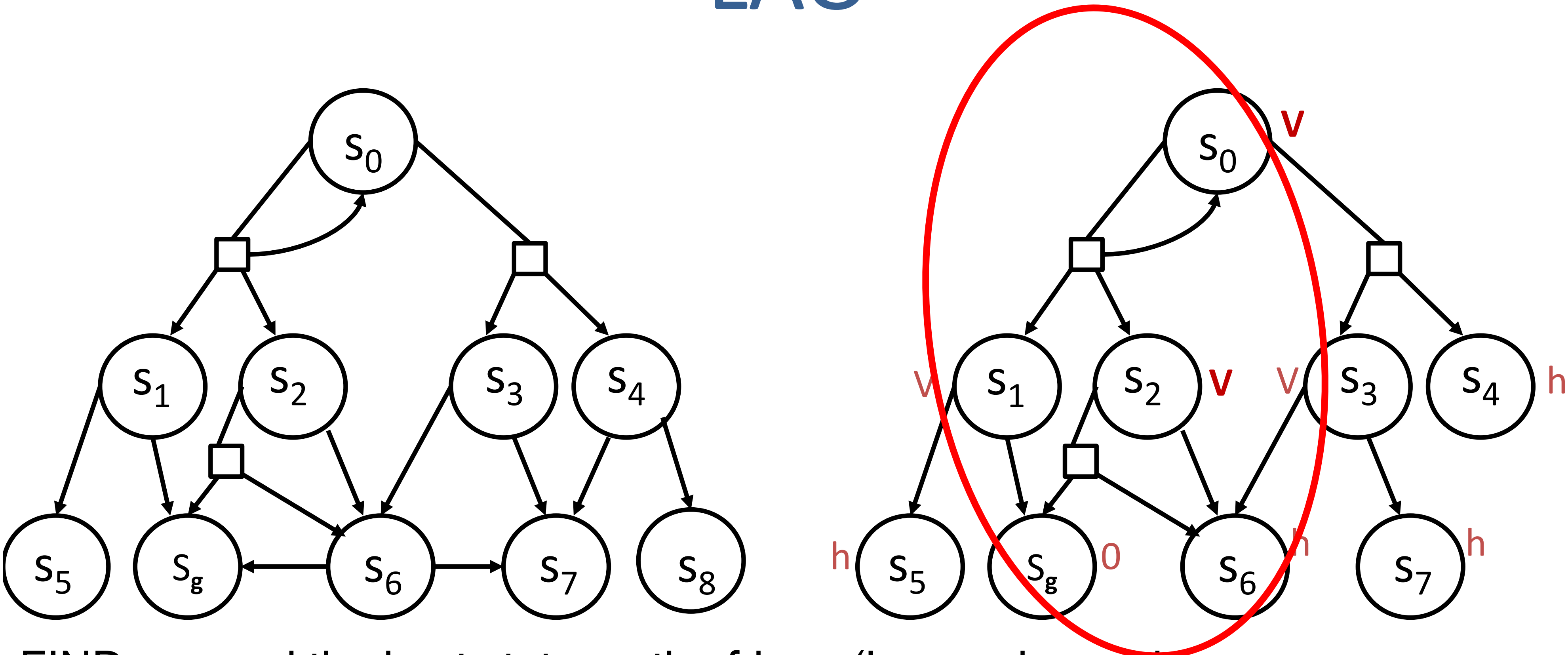
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state on the fringe (in greedy graph)

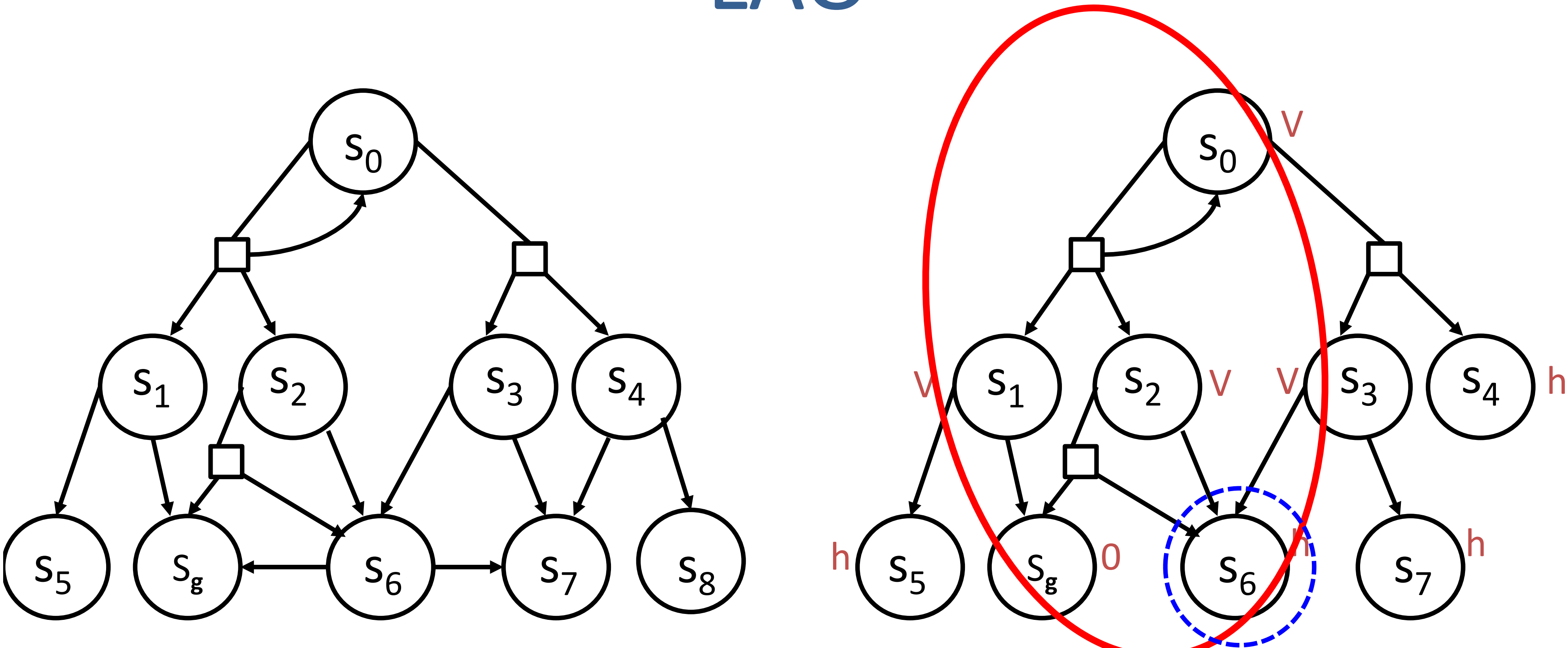
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



FIND: expand the best state on the fringe (in greedy graph)

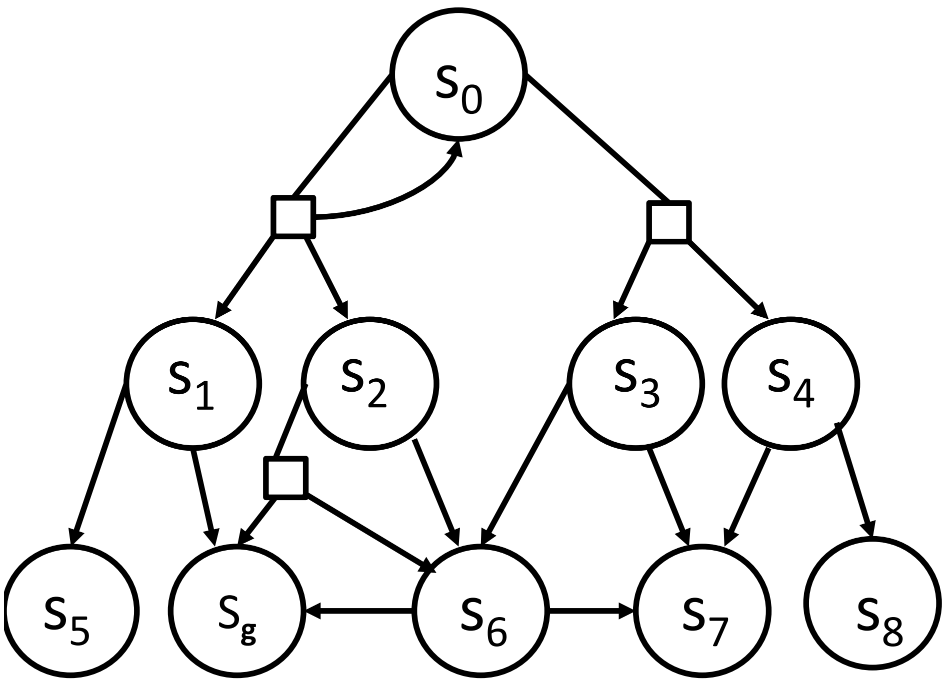
initialize all new states by their heuristic value

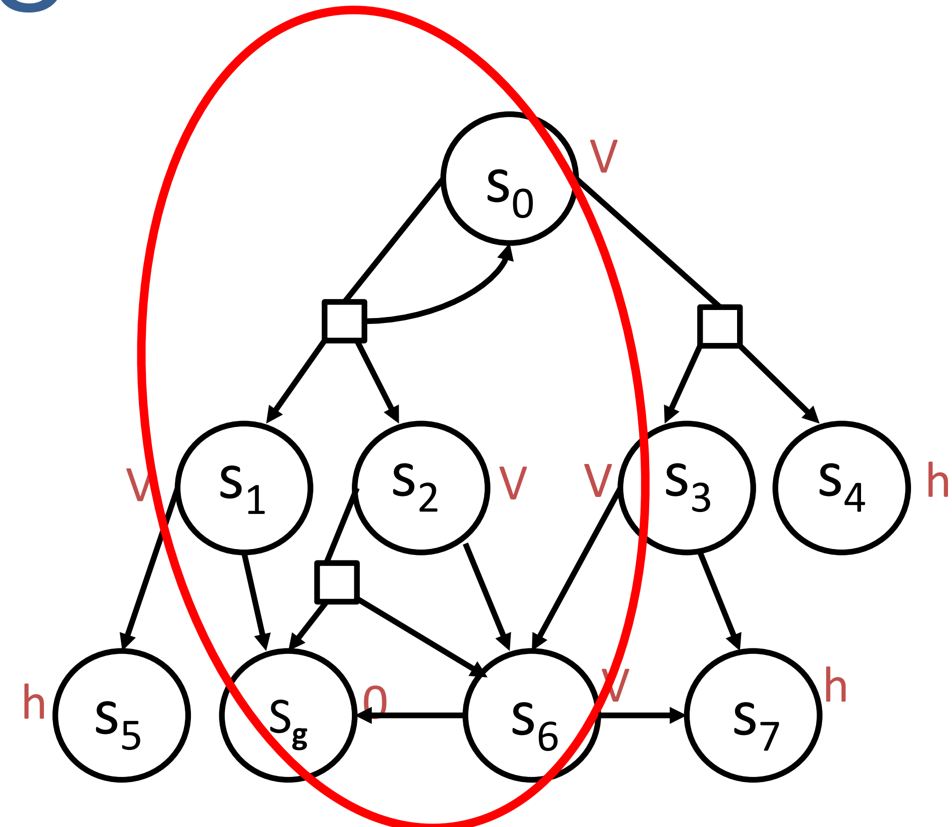
subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

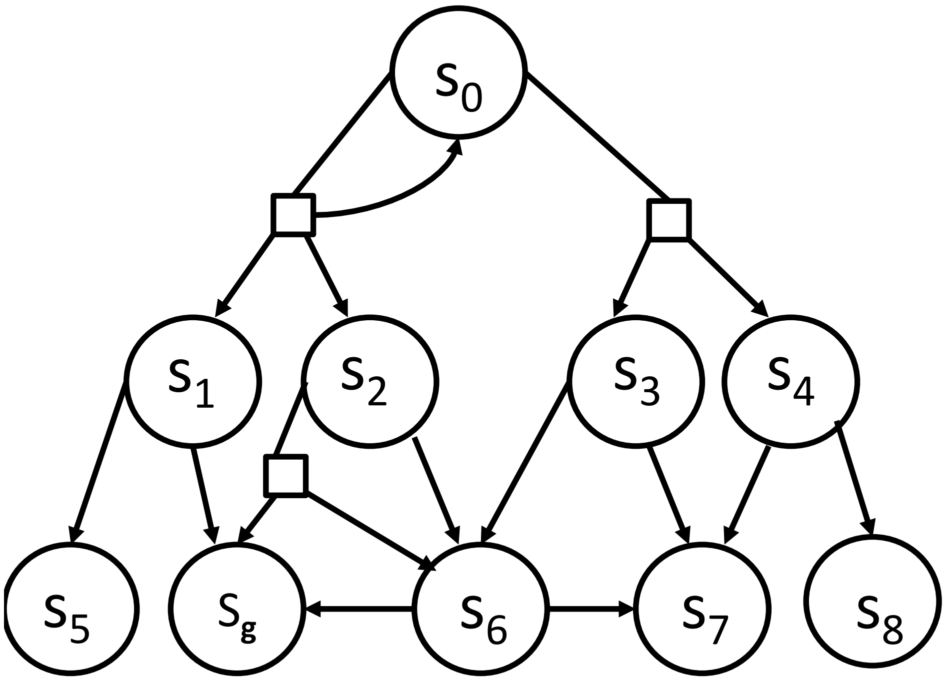
LAO*

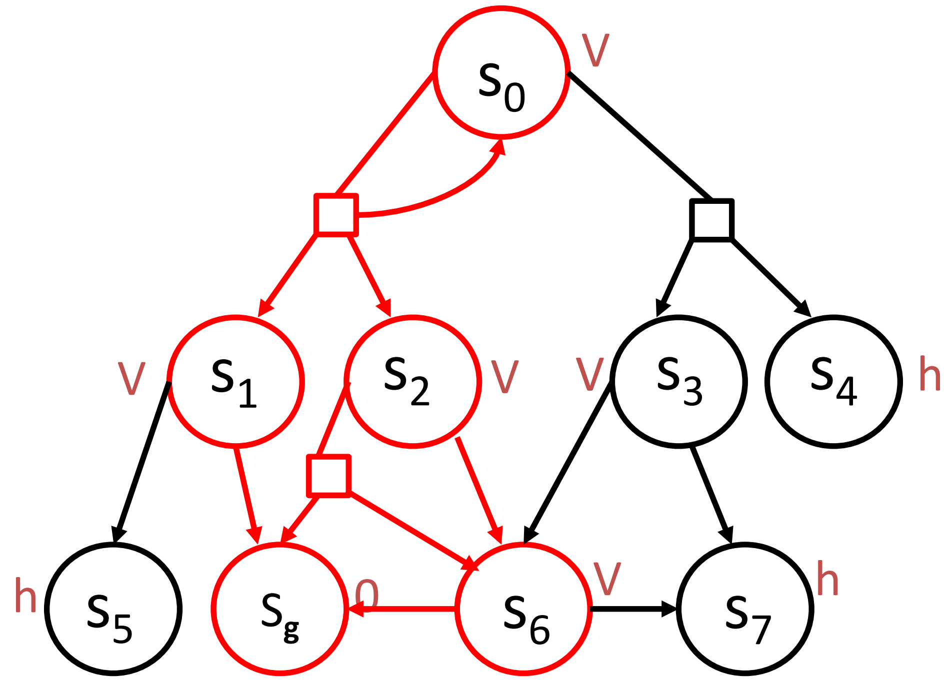




output the greedy graph as the final policy

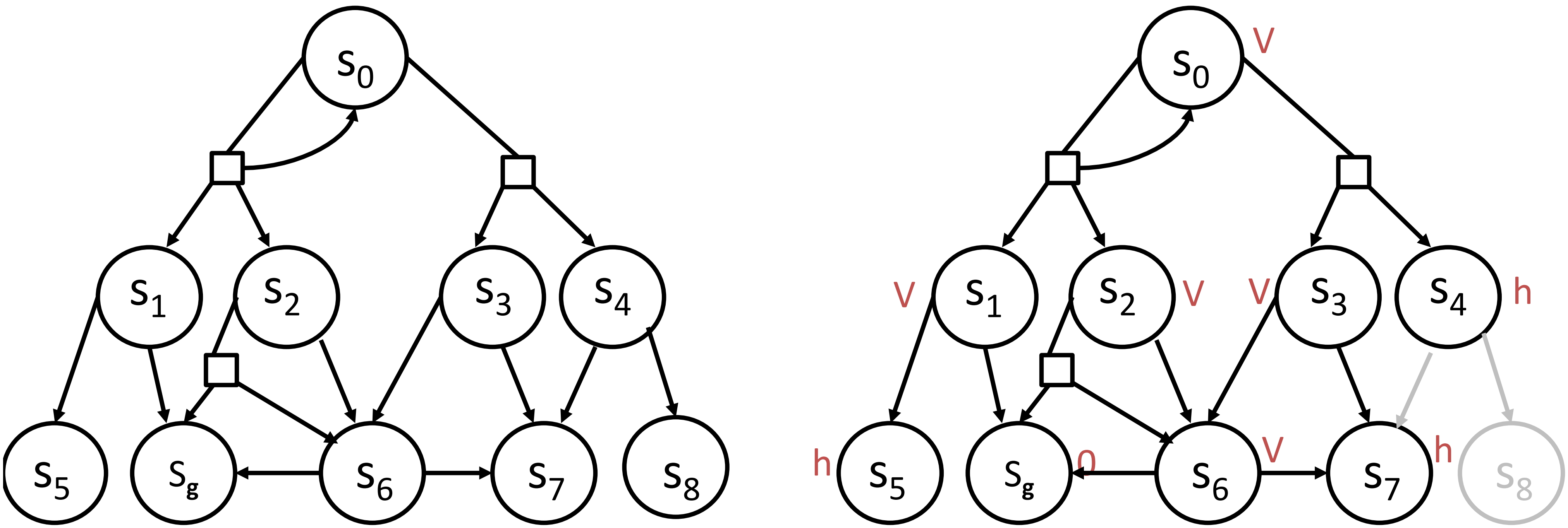
LAO*





output the greedy graph as the final policy

LAO*



s_4 was never expanded

s_8 was never touched

LAO* [Hansen&Zilberstein 98]

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
- perform PI on this subset
- recompute the greedy graph

until greedy graph has no fringe

output the greedy graph as the final policy

one expansion

lot of computation

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* [Hansen&Zilberstein 01]

add s_0 to the fringe and to greedy policy graph

repeat

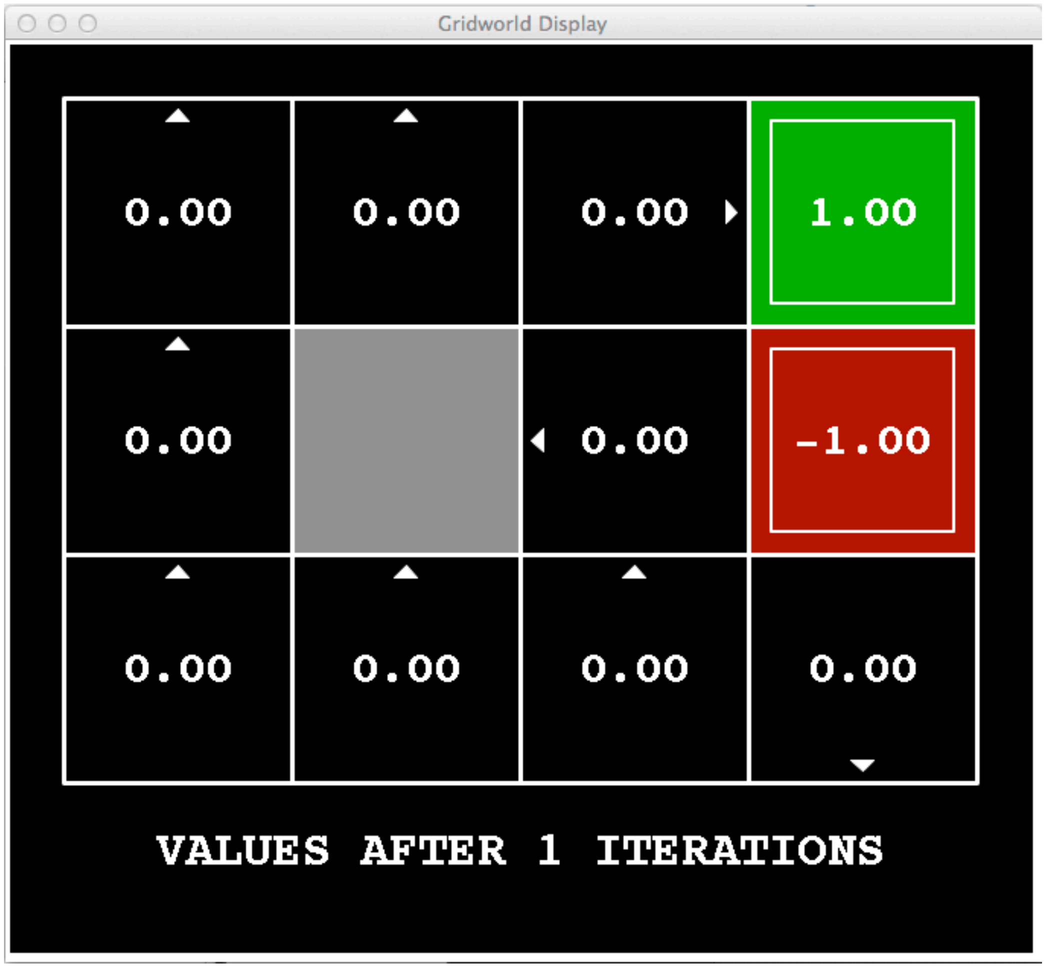
- FIND: expand all states in greedy fringe
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- only one backup per state in greedy graph
- recompute the greedy graph

until greedy graph has no fringe

in what order? (fringe → start) DFS postorder

output the greedy graph as the final policy

Backup Order Matters
VI - k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

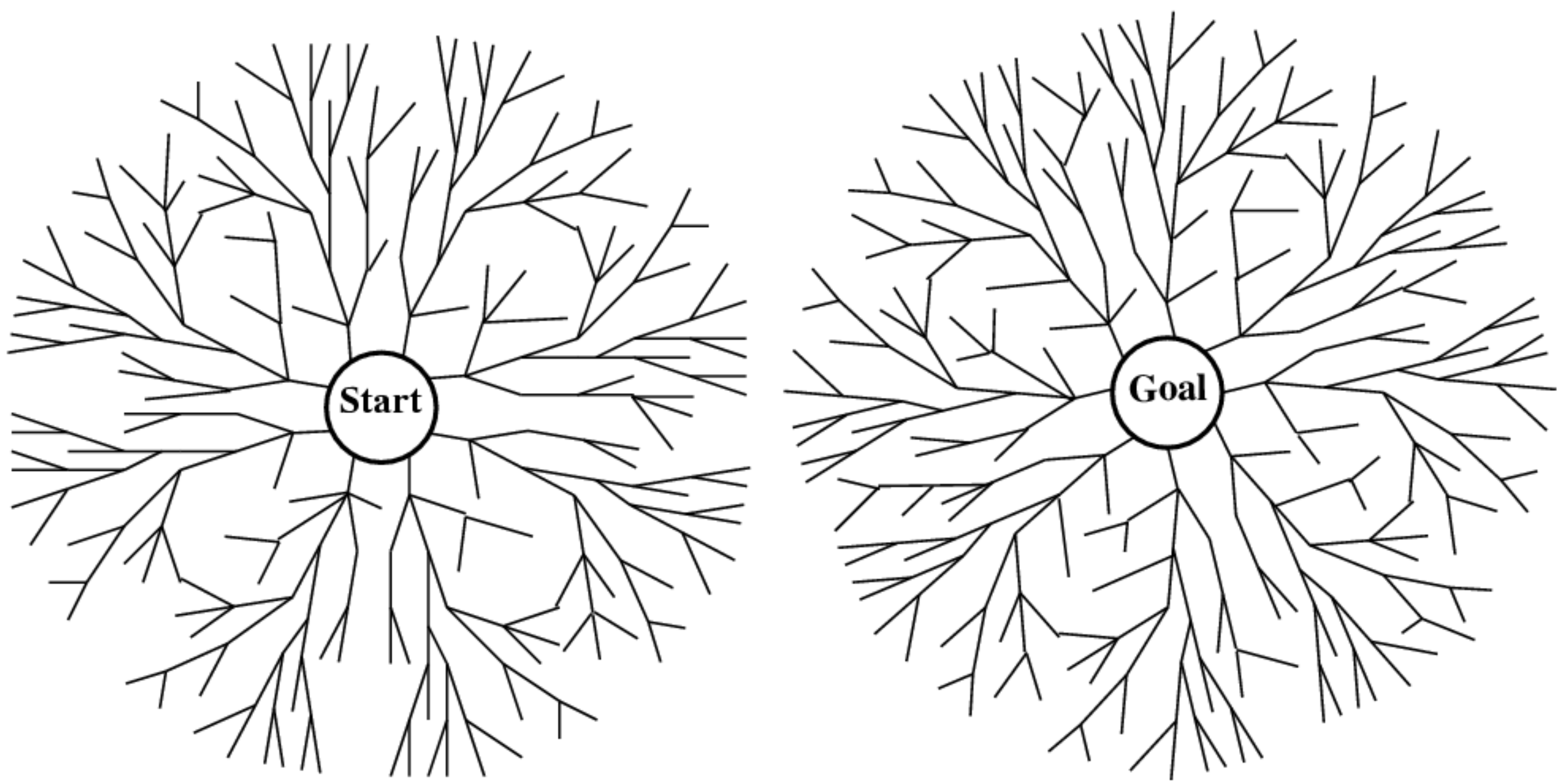
Noise = 0.2
Discount = 0.9
Living reward = 0

Reverse LAO* [Dai&Goldsmith 06]

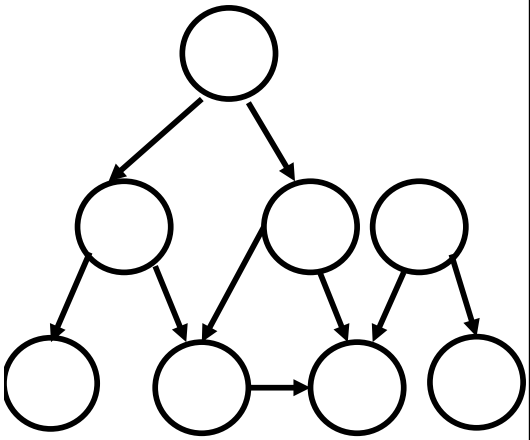
- LAO* may spend huge time until a goal is found
  - guided only by s_0 and heuristic
- LAO* in the reverse graph
  - guided only by goal and heuristic
- Properties
  - Works when 1 or handful of goal states
  - May help in domains with small fan in

Bidirectional LAO* [Dai&Goldsmith 06]

- Go in both directions from start state and goal
- Stop when a bridge is found



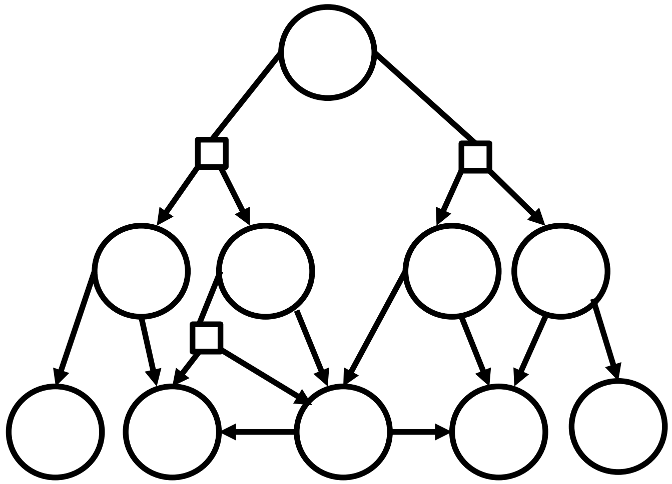
A* → LAO*



regular graph

soln:(shortest) path

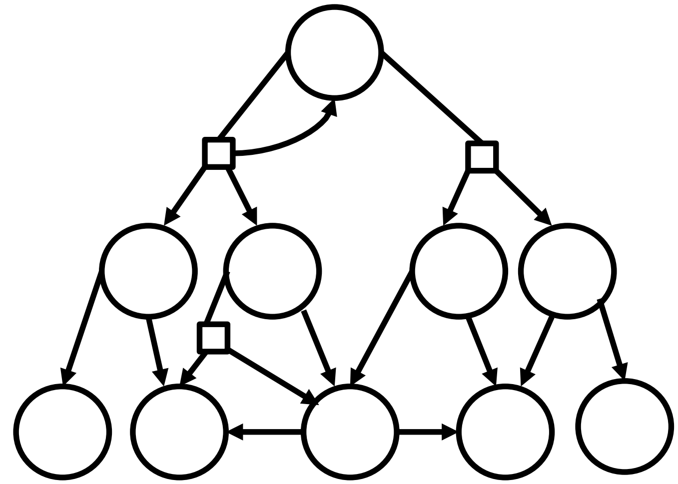
A*



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 exploit heuristic guidance & reachability!