Goal-Directed MDPs

Models and Algorithms

Mausam

Indian Institute of Technology, Delhi

Joint work with Andrey Kolobov and Dan Weld

Planning **à la** Sutton

- control
- full sequential
- model-based
- value-based
- tabular/function-approximation
- TD/Monte-Carlo

Typical Planning Setting

- vs. RL: model of the world is known
- vs. flat: model of the world in a declarative representation
  - symbolic
  - large problems
- vs. reward: goal directed
- vs. complete state space: knowledge of the start state
- domain independent: no additional human input

3 Key Messages

- M#0: No need for exploration-exploitation tradeoff
  - planning is purely a computational problem (V.I. ~~vs. Q~~)
- M#1: Search in planning
  - states can be ignored or reordered for efficient computation
- M#2: Representation in planning
  - develop interesting representations for Factored MDPs
    - → Exploit structure to design domain-independent algorithms
- M#3: Goal-directed MDPs
  - design algorithms/models that use explicit knowledge of goals

Agenda

- Background: Stochastic Shortest Paths MDPs
- Background: Heuristic Search for SSP MDPs
- Algorithms: Automatic Basis Function Discovery
- Models: SSPs $\rightarrow$ Generalized SSPs

Infinite Horizon Discounted Reward MDP

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- R(s,a,s'): reward
- γ: discount factor

Where Does γ Come From?

- **γ can affect optimal policy significantly**
  - $\gamma = 0 + \varepsilon$: yields myopic policies for “impatient” agents
  - $\gamma = 1 - \varepsilon$: yields far-sighted policies, inefficient to compute

- How to set it?
  - Sometimes suggested by data
    - (e.g., inflation or interest rate)
  - Often set to whatever gives a reasonable policy

Infinite Horizon Discounted Reward MDP

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- R(s,a,s'): reward
- γ: discount factor

Stochastic Shortest Path MDP

- S: A set of states
- A: A set of actions
- T(s,a,s’): transition model
- R(s,a,s’): reward
- γ: discount factor

Stochastic Shortest Path MDP

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): cost
- γ: discount factor

Stochastic Shortest Path MDP

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): cost
-

Stochastic Shortest Path MDP

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): cost
- G: set of goals

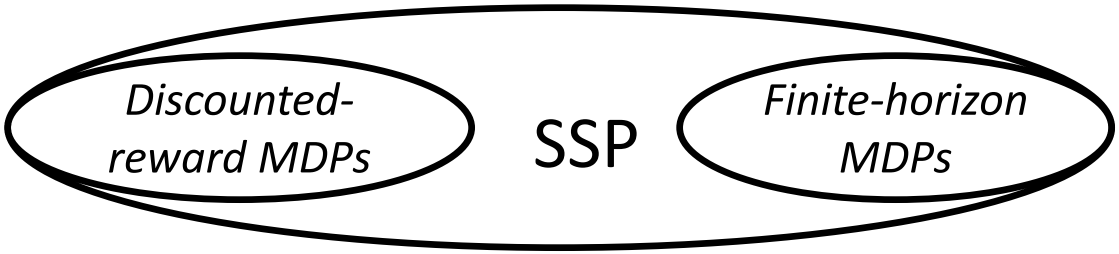
Minimize
- expected cost to reach a goal
- under full observability
- indefinite horizon

Bellman Equations for SSP

$$\boxed{\begin{aligned} V^*(s) &= 0 \qquad \text{if } s \in \mathcal{G} \\ &= \min_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{C}(s,a,s') + V^*(s')\right] \end{aligned}}$$

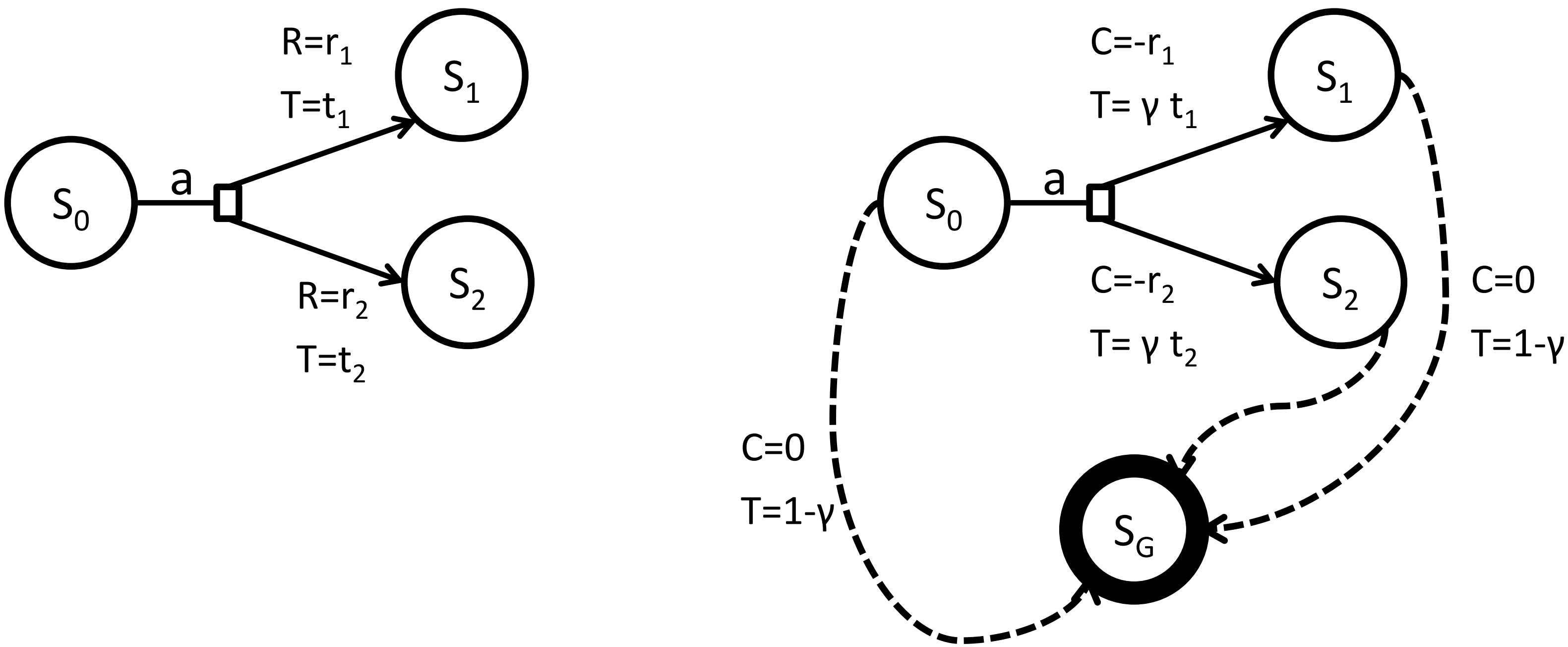
add base case; no discount factor

SSP vs. IHDR?



Discounted Reward MDP → SSP

[Bertsekas&Tsitsiklis 95]



When is SSP well formed/defined

[Bertsekas, 1995]

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): cost
- G: set of goals

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

Agenda

- Background: Stochastic Shortest Paths MDPs
- Background: Heuristic Search for SSP MDPs
- Algorithms: Automatic Basis Function Discovery
- Models: SSPs → Generalized SSPs

Heuristic Search

- Limitations of VI
  - enumeration of state space
  - curse of dimensionality

- Heuristic search: insights
  - knowledge of a start state to save on computation
    ~ (all sources shortest path → single source shortest path)
  - additional knowledge in the form of heuristic fn
    ~ (dfs/bfs → A*)

SSP_{s0}

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): cost
- G: set of goals
- s_0: start state

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

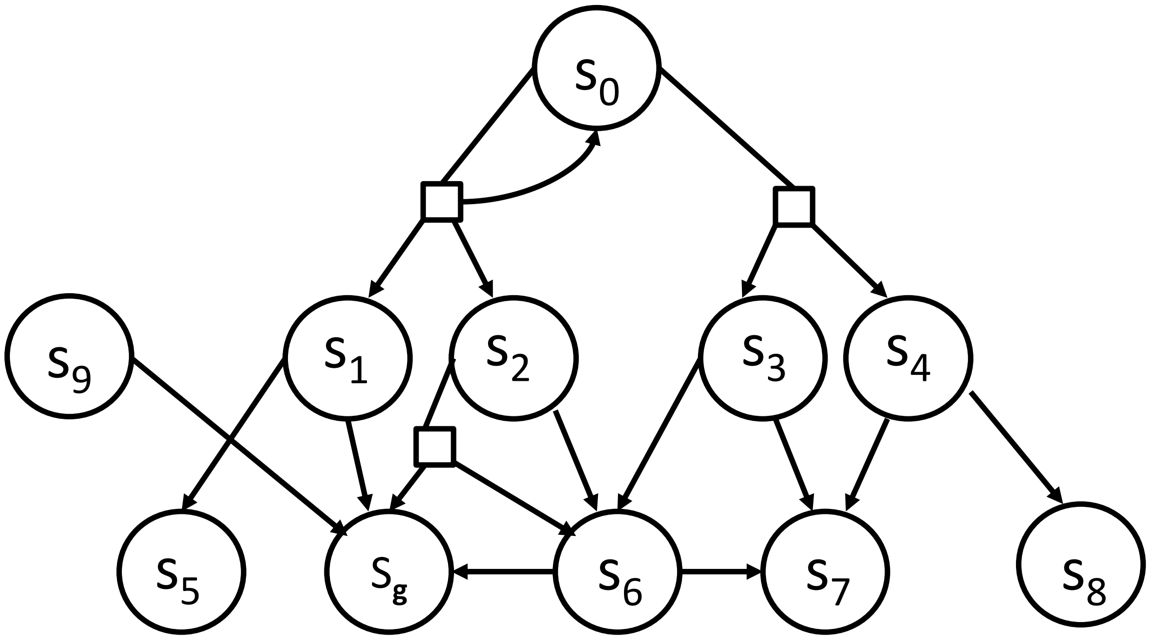
SSP_{s0}

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

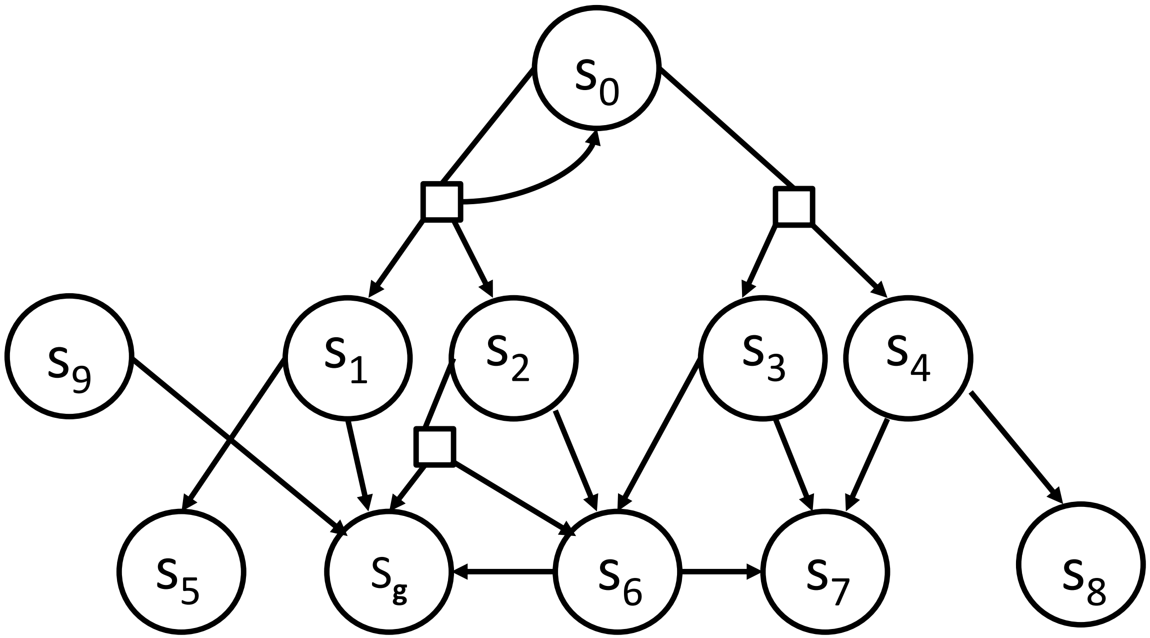
Partial Policy

- 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



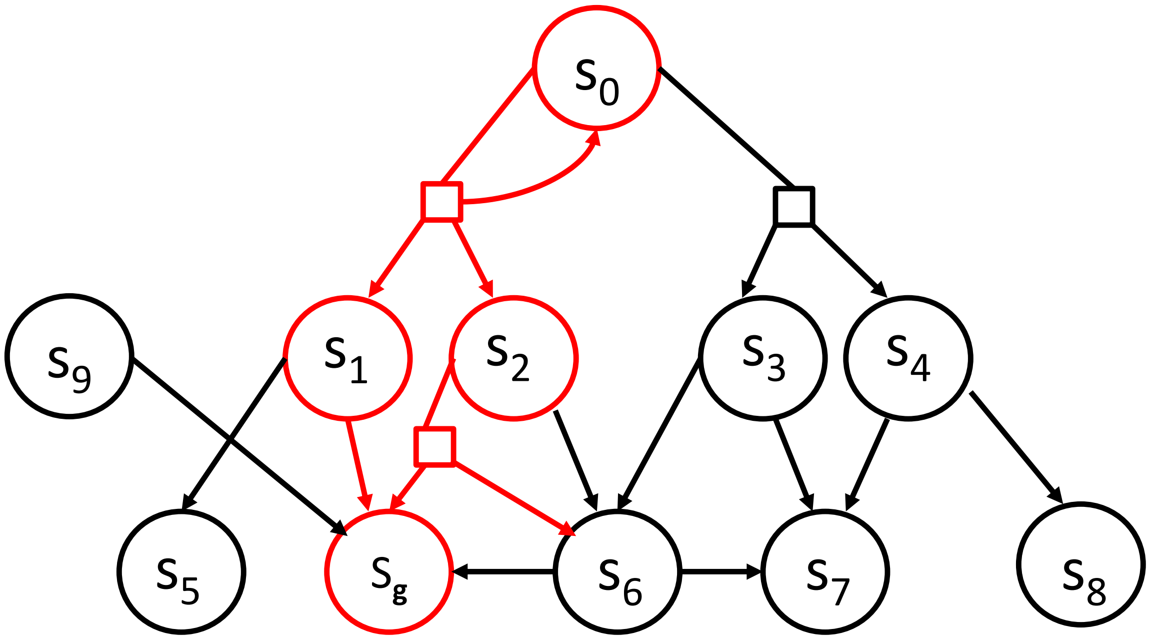
Partial policy closed wrt s_0



Is this policy closed wrt s_0?

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

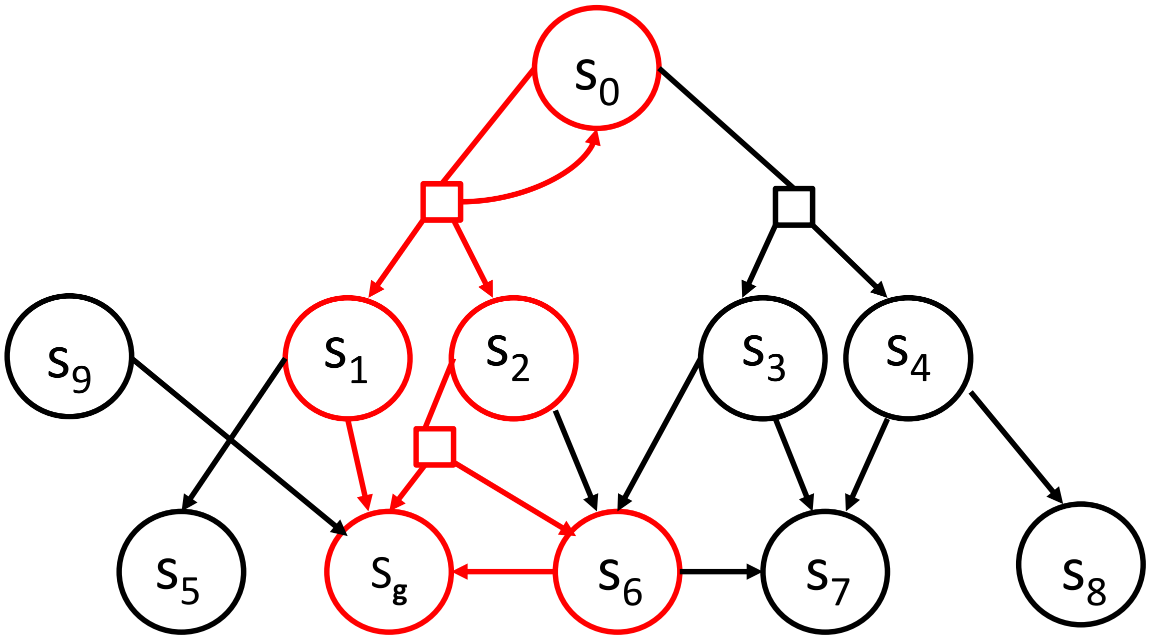
Partial policy closed wrt s_0



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?

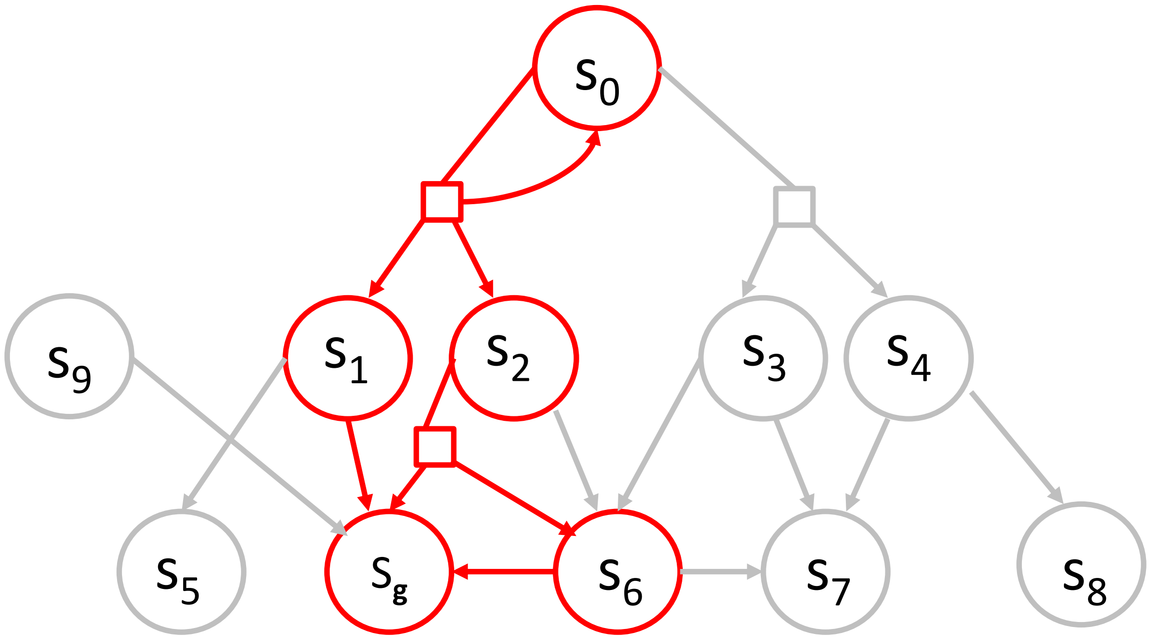
$\pi_{s0}(s_0) = a_1$

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

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

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

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*: π^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
    - perform a Bellman backup
  - until no such state remains

FIND & REVISE [Bonet&Geffner 03a]

Start with a heuristic value function $V \leftarrow h$

while V's *greedy graph* $G^V_{s_0}$ *contains a state* s *with* $Res^V(s) > \epsilon$ **do**

  FIND a state s in $G^V_{s_0}$ with $Res^V(s) > \epsilon$

  REVISE $V(s)$

(perform Bellman backups)

end

return a π^V

- Convergence to V* is guaranteed
  - if heuristic function is admissible
  - ~no state gets starved in ∞ FIND steps

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 small

output the greedy graph as the final policy

LAO* [Hansen&Zilberstein 98]

add s_0 to the fringe and to greedy policy graph

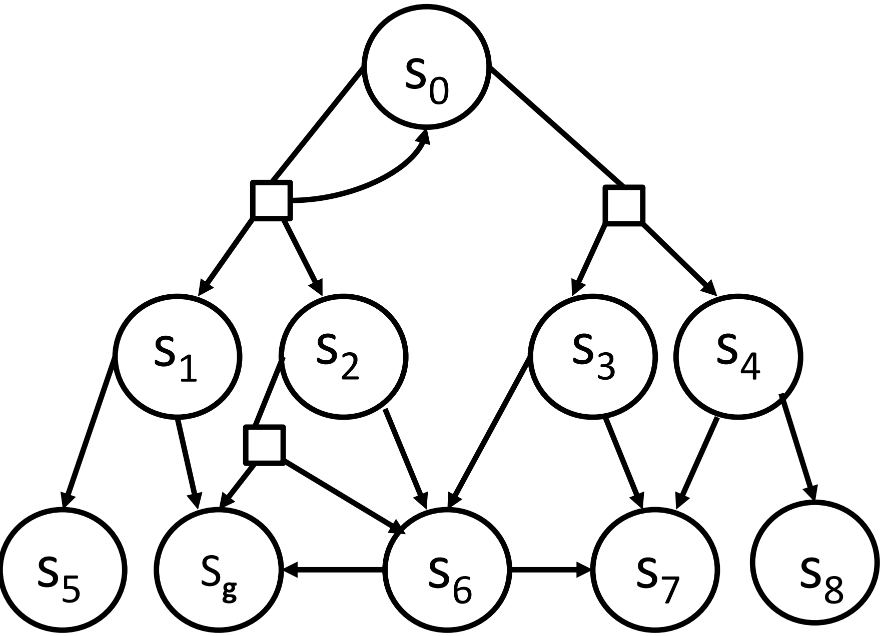
repeat

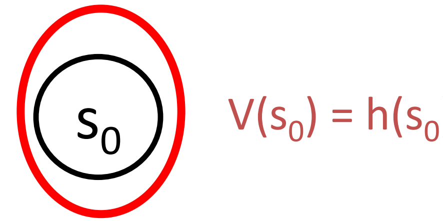
- FIND: expand best state *s* on the fringe (in greedy graph)
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach s
- perform VI on this subset
- recompute the greedy graph

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

output the greedy graph as the final policy

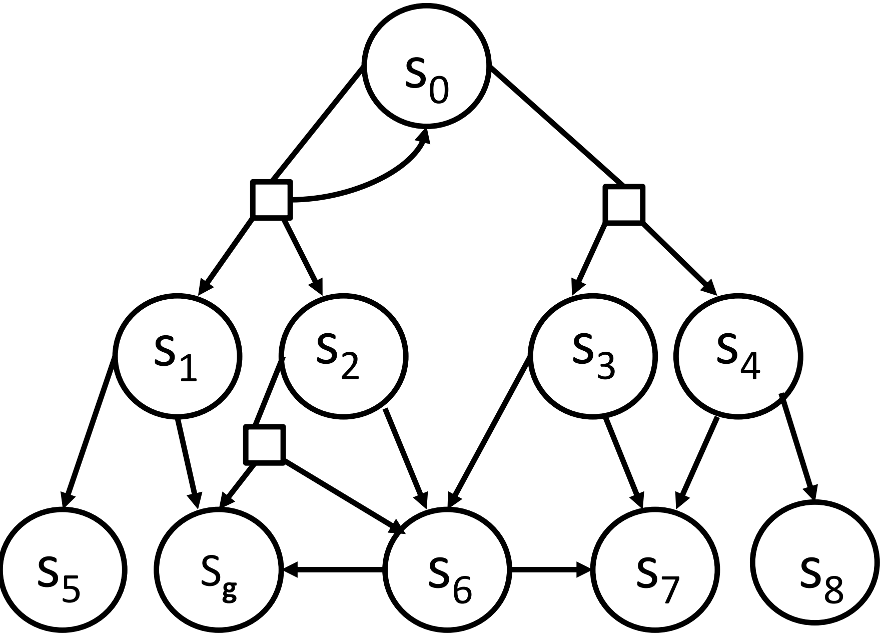
LAO*

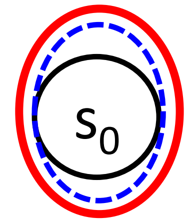




add s_0 in the fringe and in greedy graph

LAO*

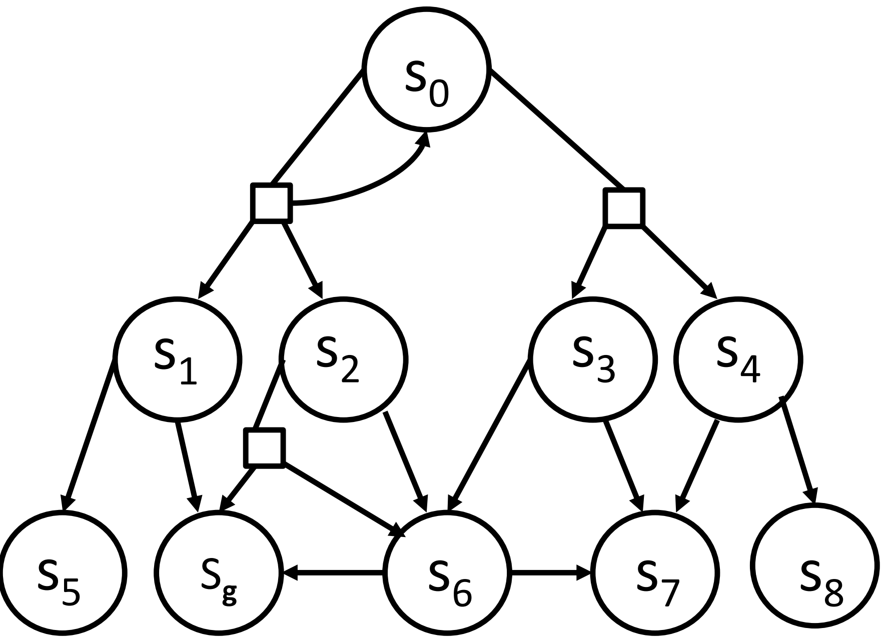


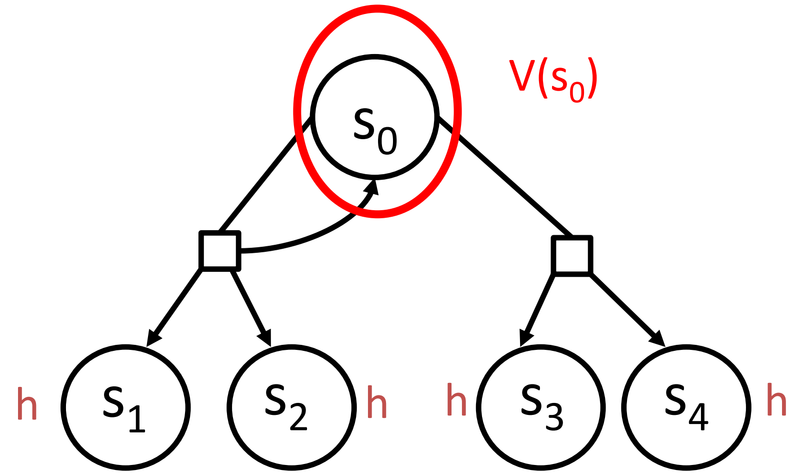


$V(s_0) = h(s_0)$

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

LAO*





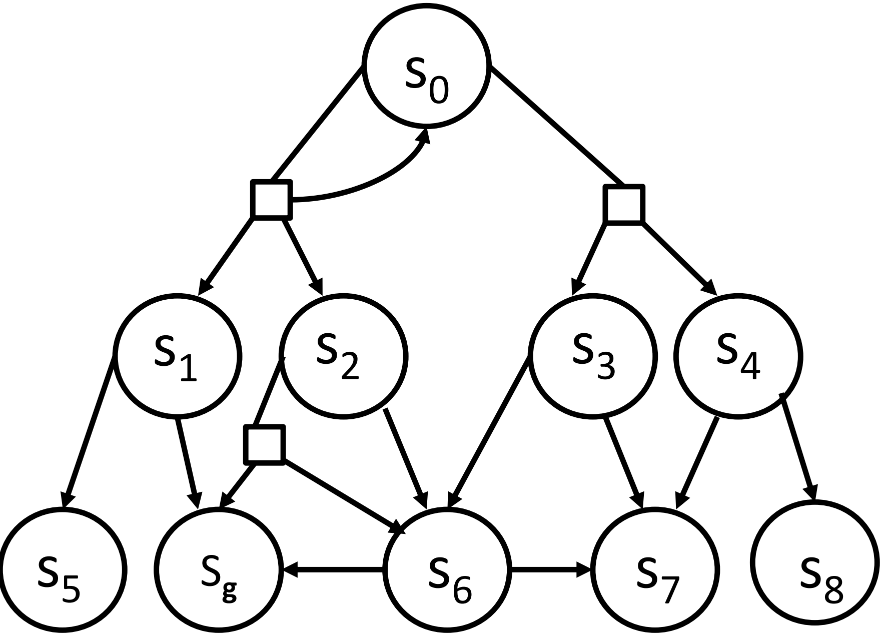
FIND: expand some states on the fringe (in greedy graph)

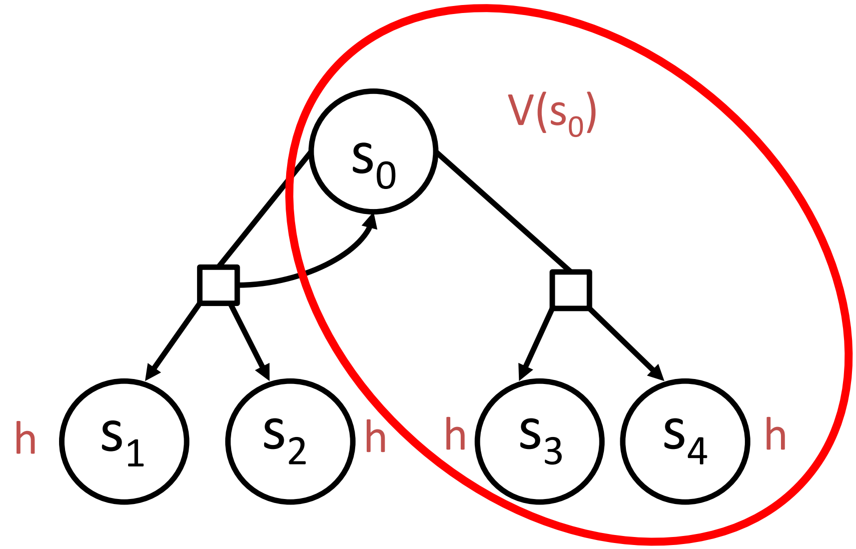
initialize all new states by their heuristic value

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)

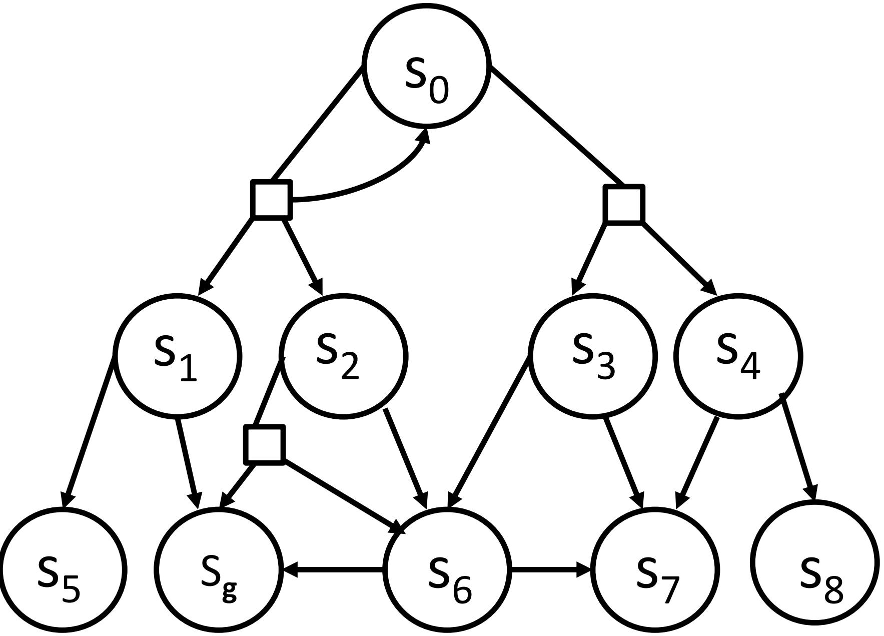
initialize all new states by their heuristic value

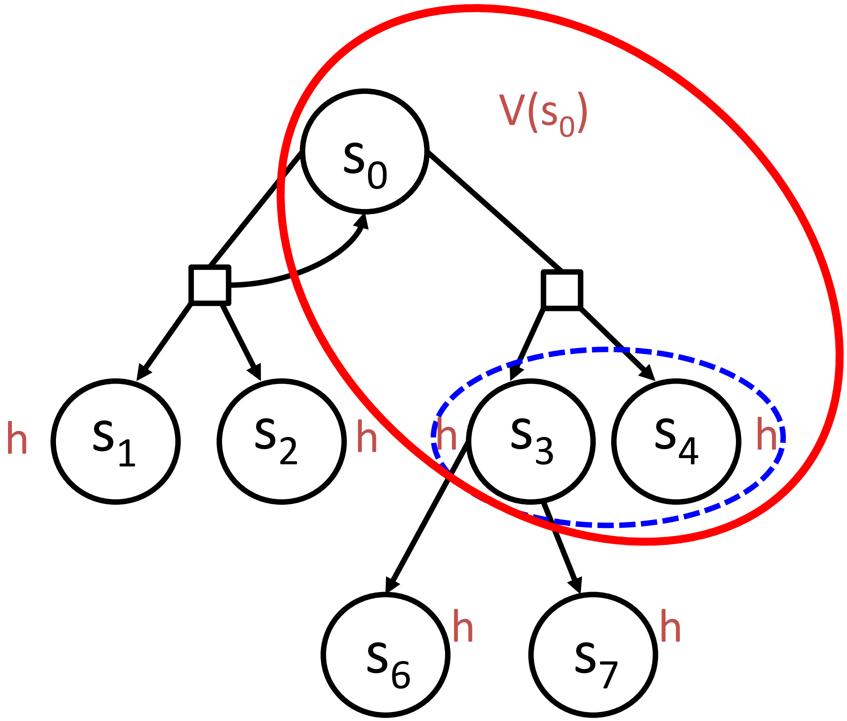
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





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

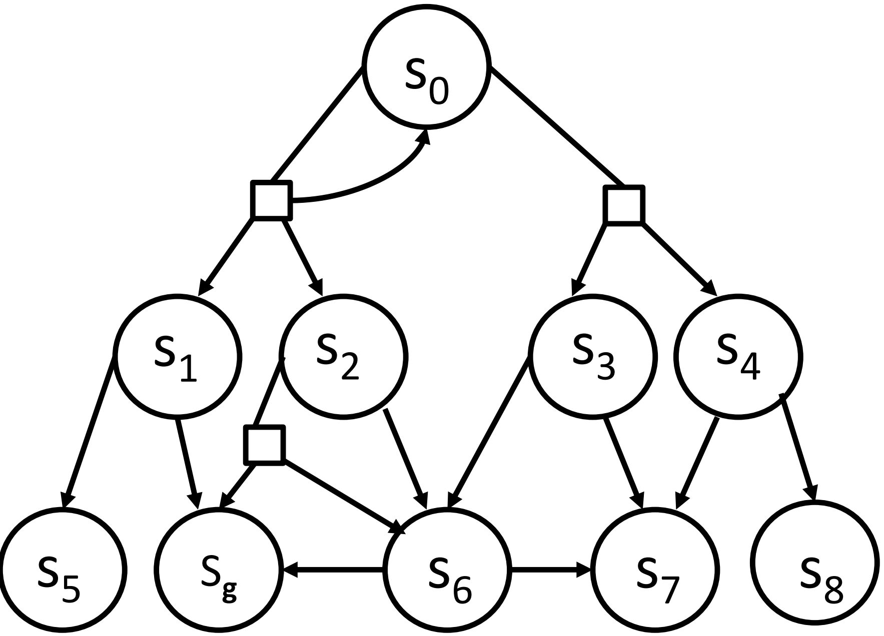
initialize all new states by their heuristic value

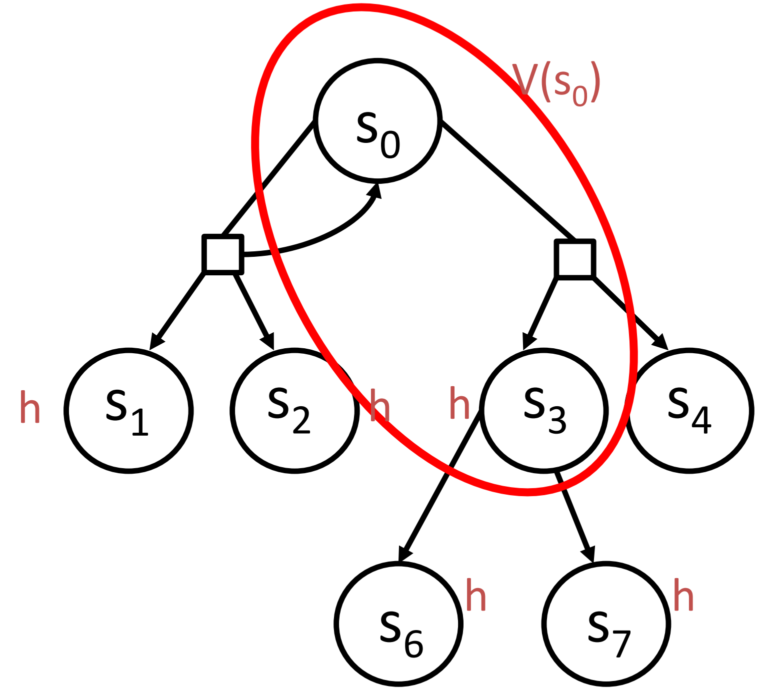
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





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

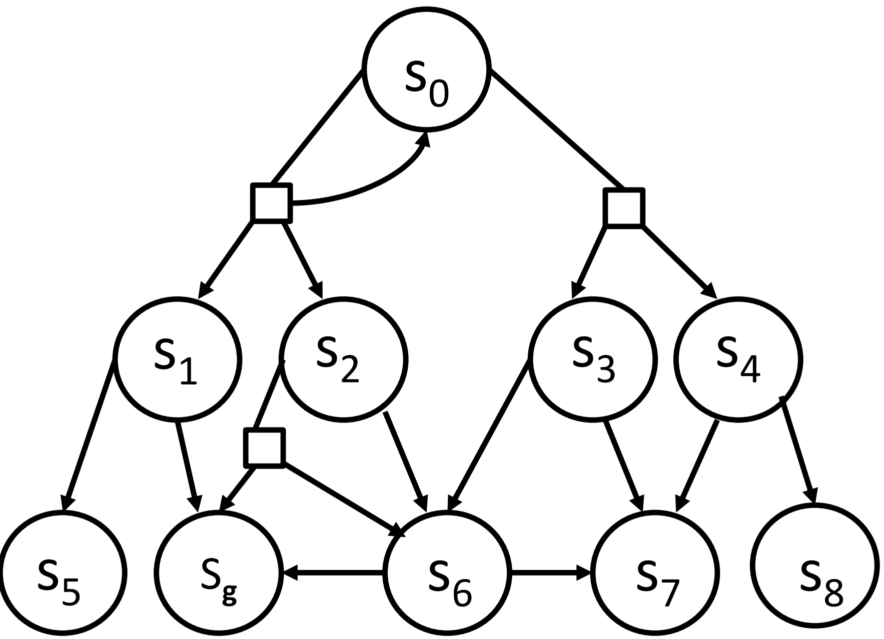
initialize all new states by their heuristic value

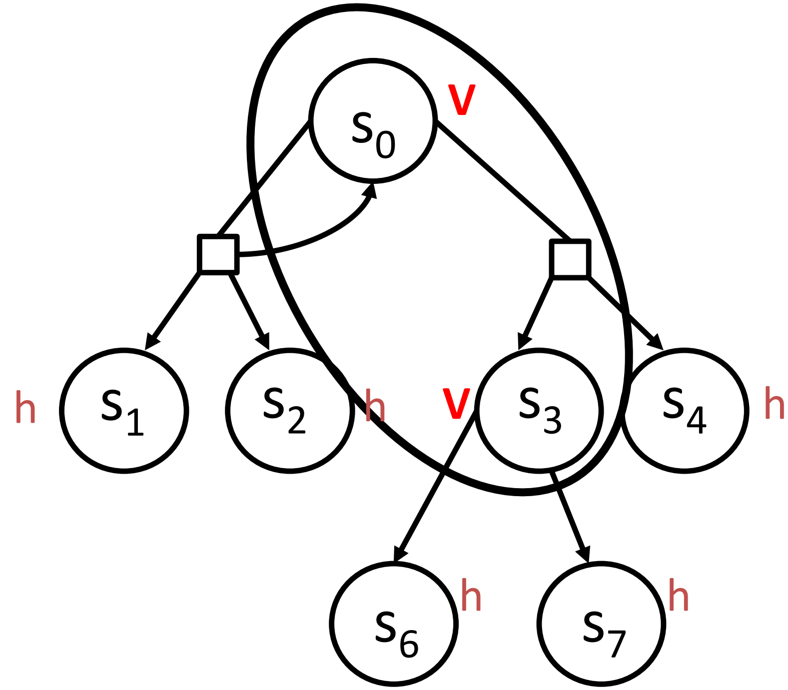
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





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

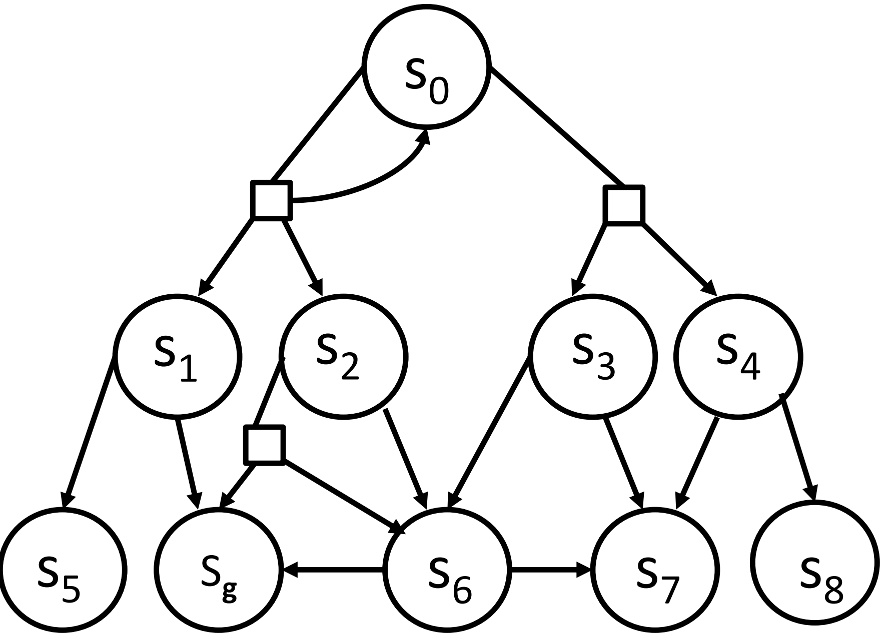
initialize all new states by their heuristic value

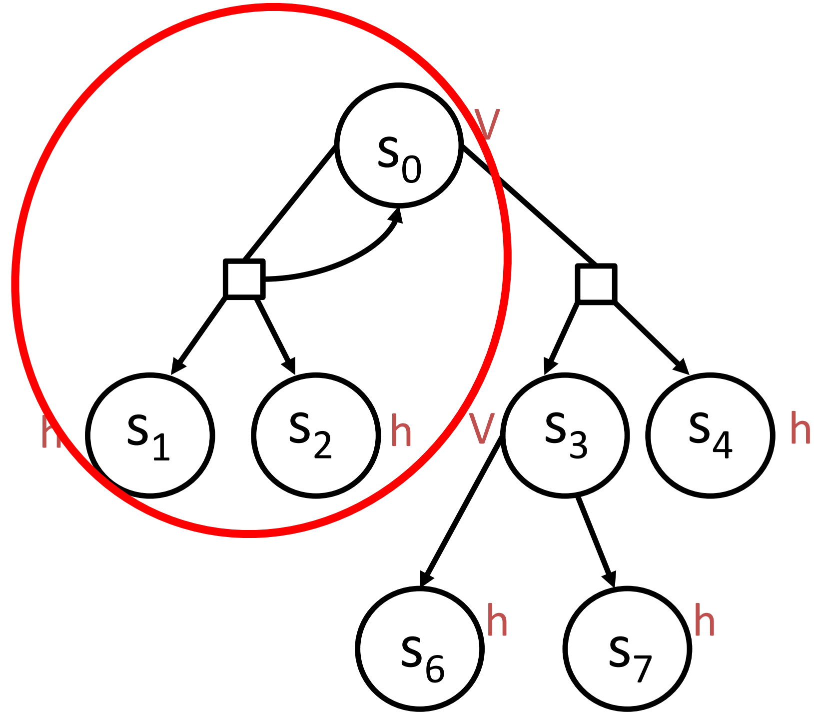
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





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

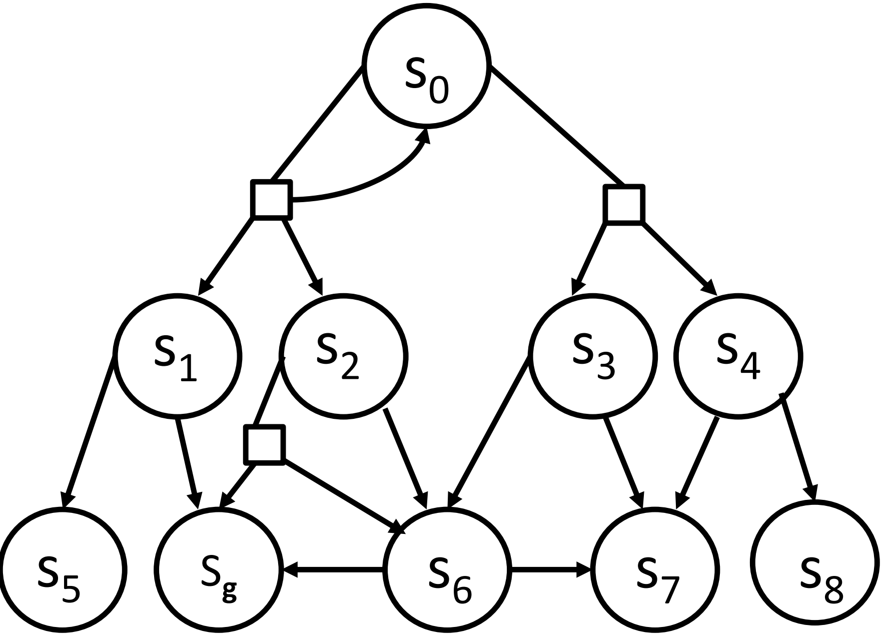
initialize all new states by their heuristic value

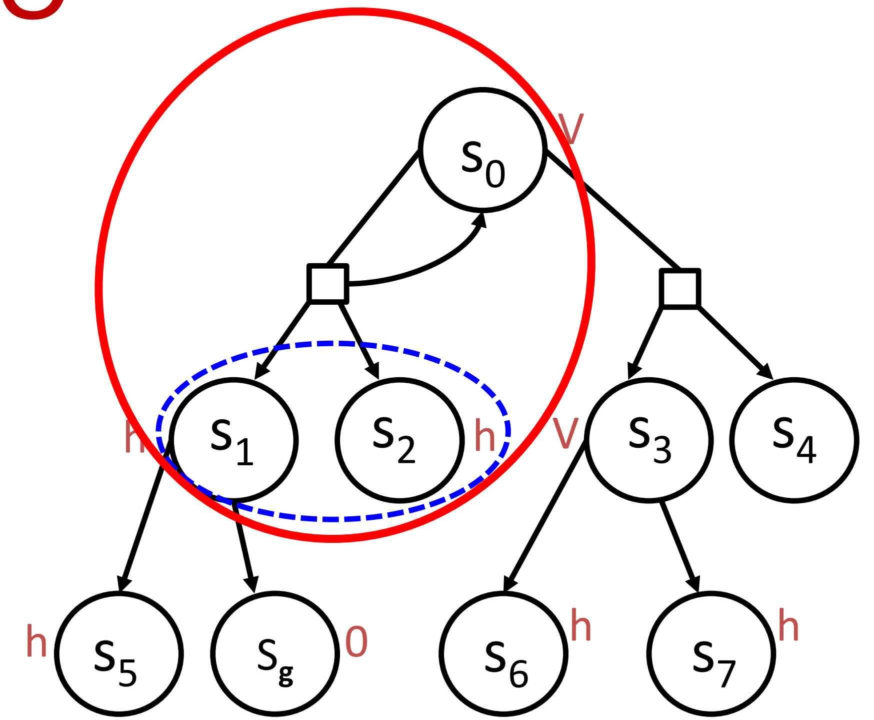
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





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

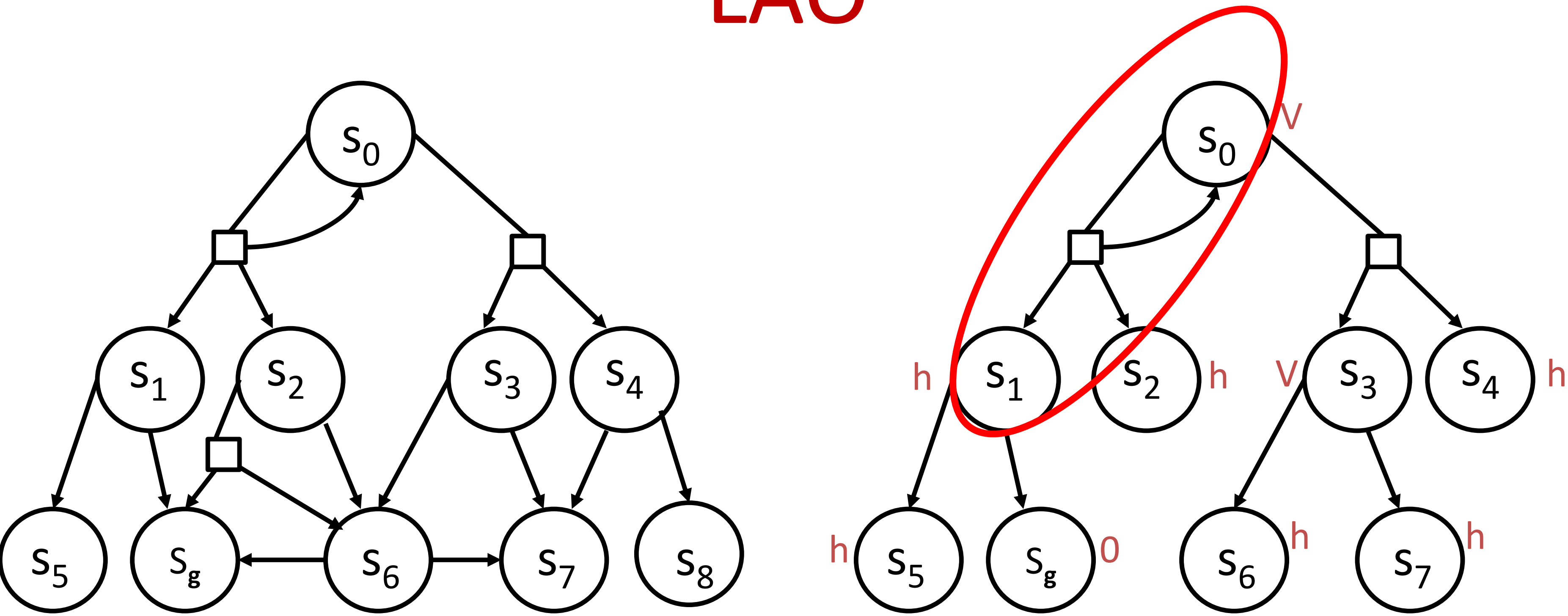
initialize all new states by their heuristic value

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)

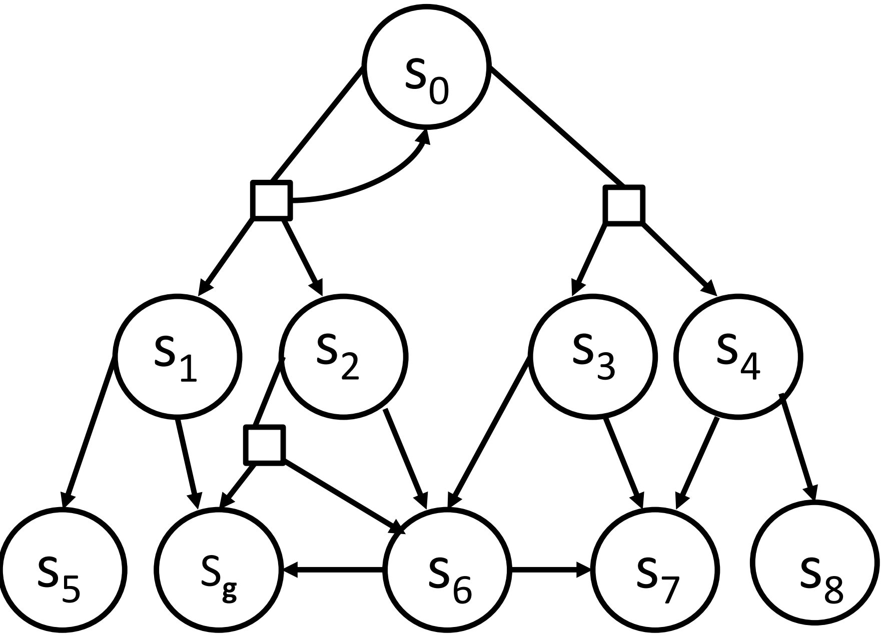
initialize all new states by their heuristic value

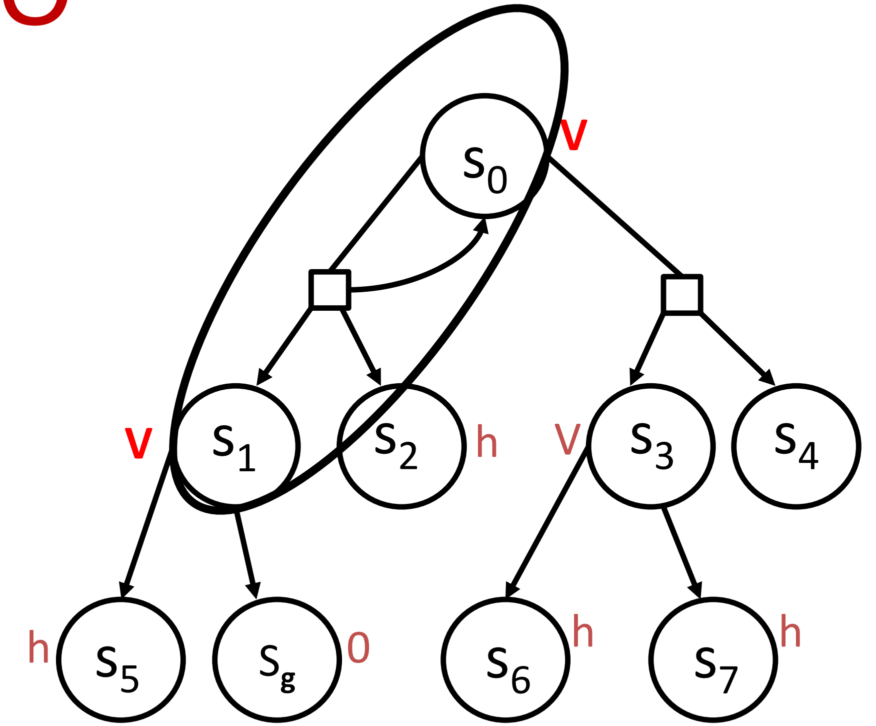
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

LAO*





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

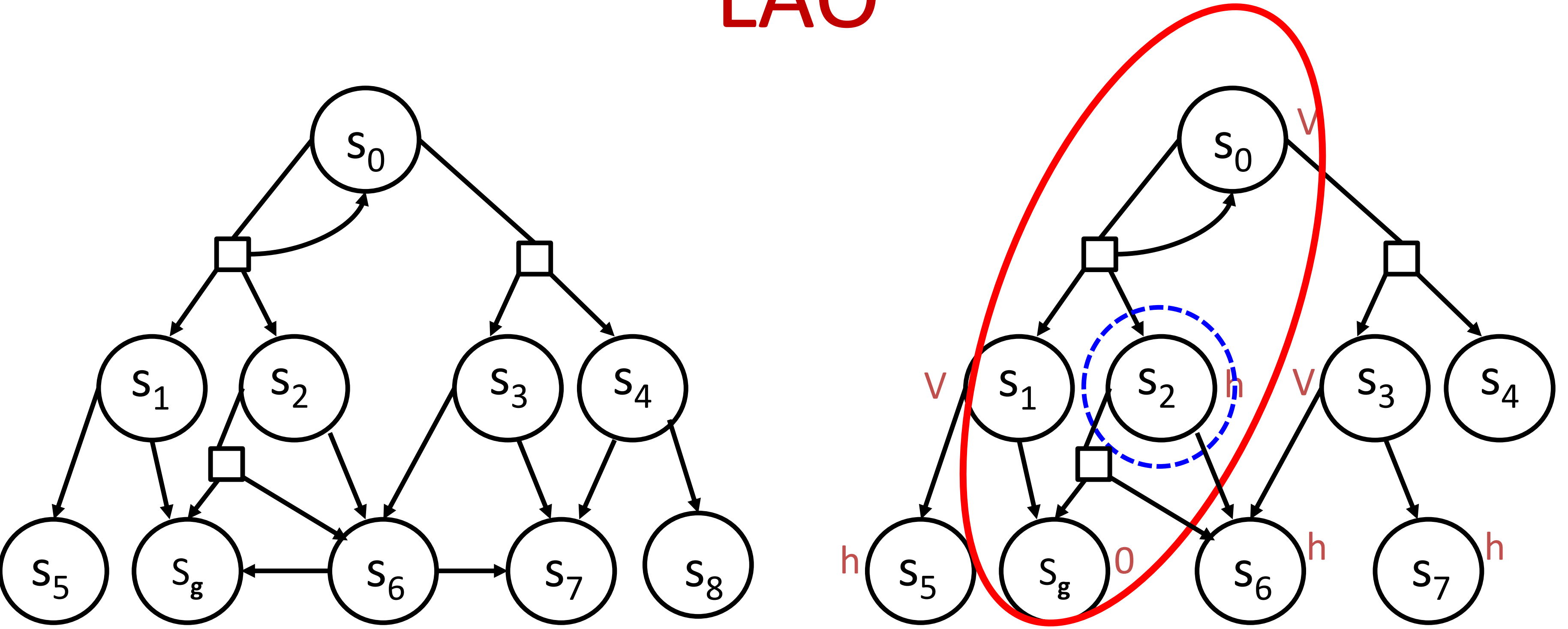
initialize all new states by their heuristic value

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)

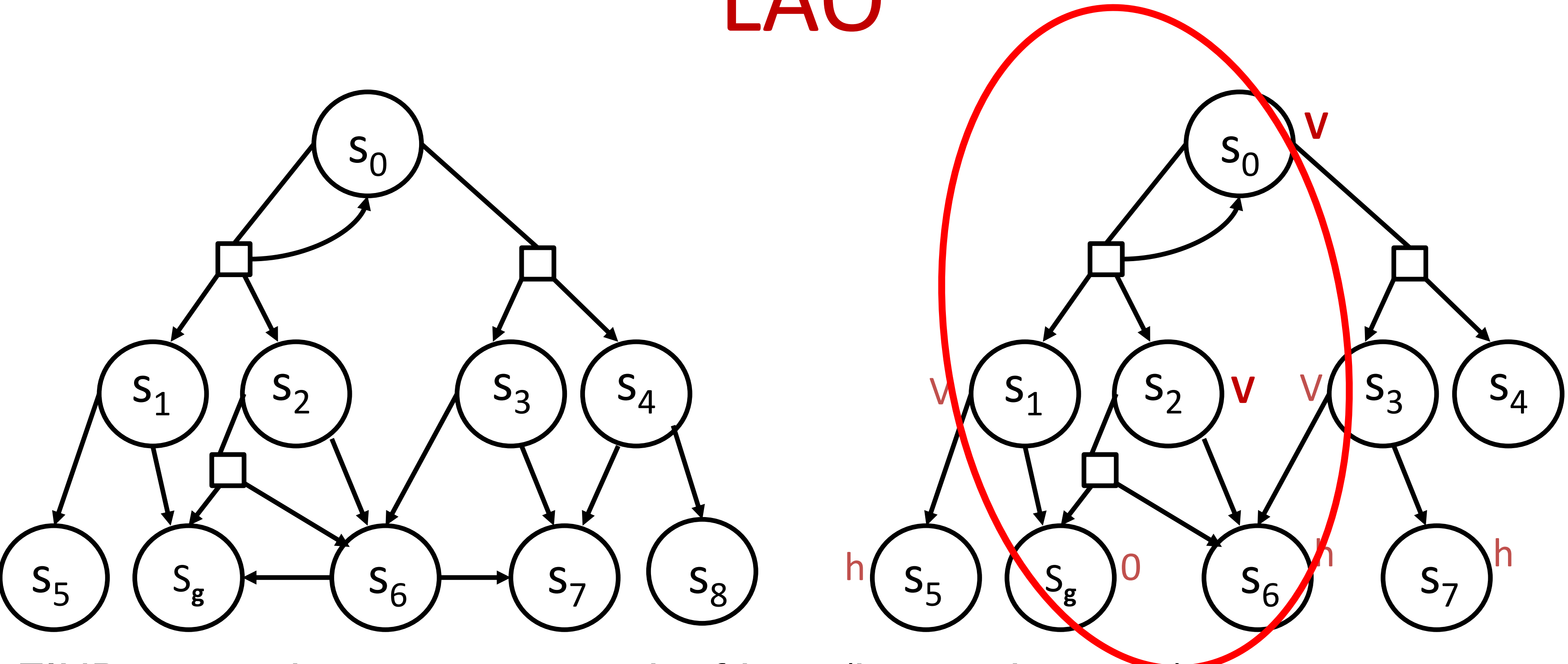
initialize all new states by their heuristic value

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)

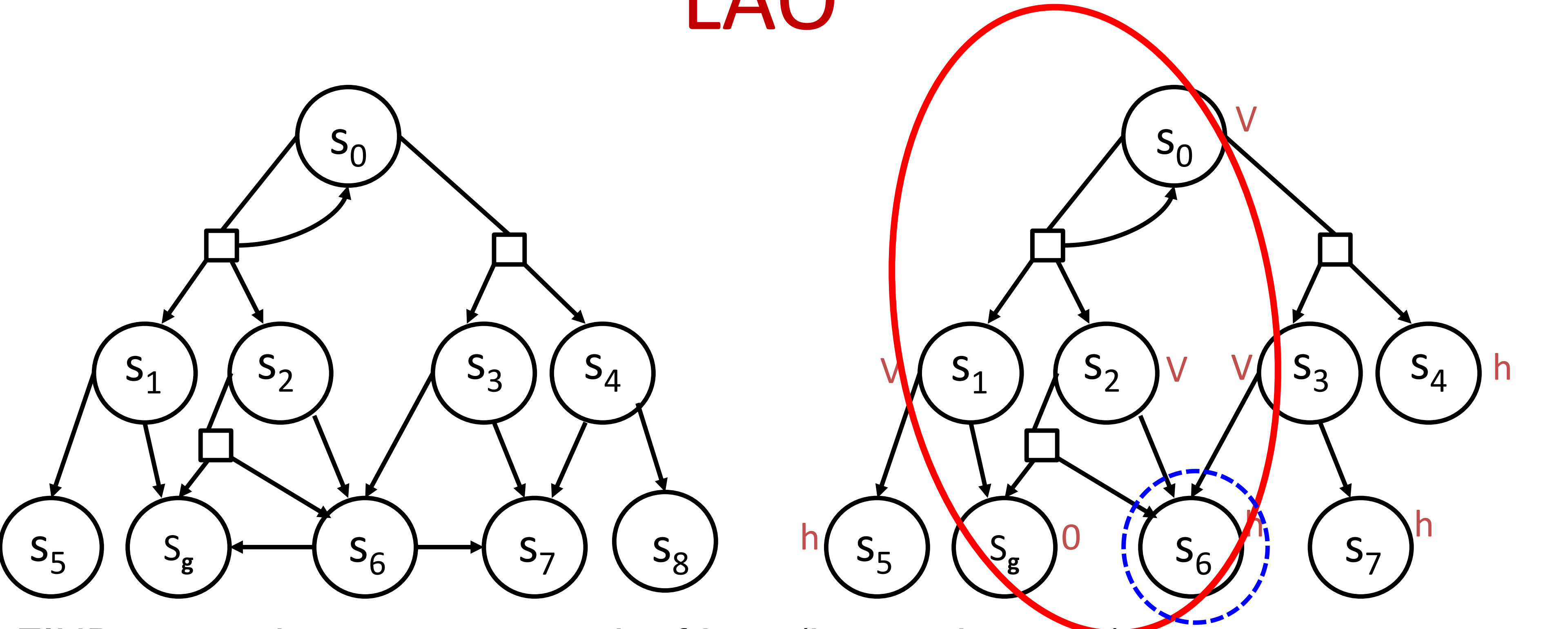
initialize all new states by their heuristic value

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)

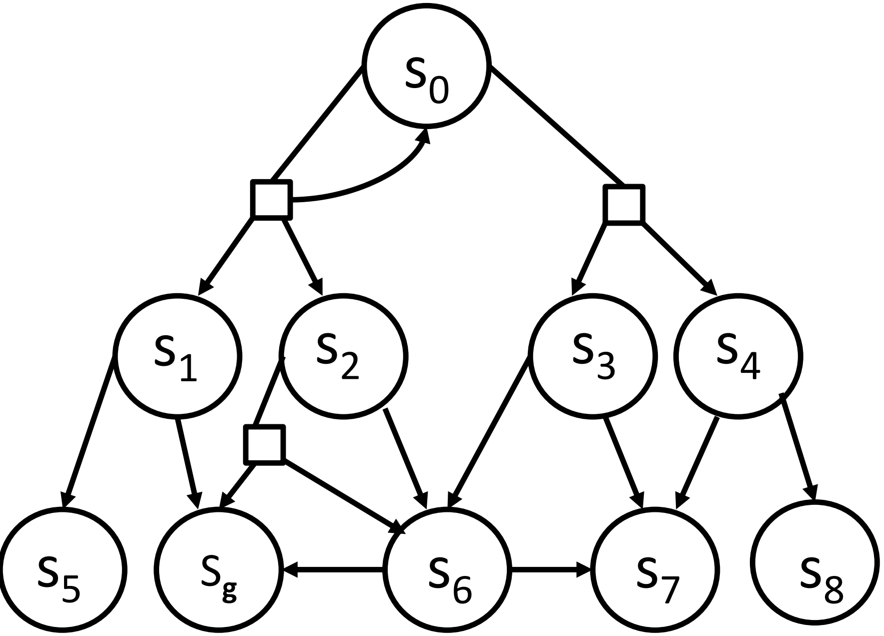
initialize all new states by their heuristic value

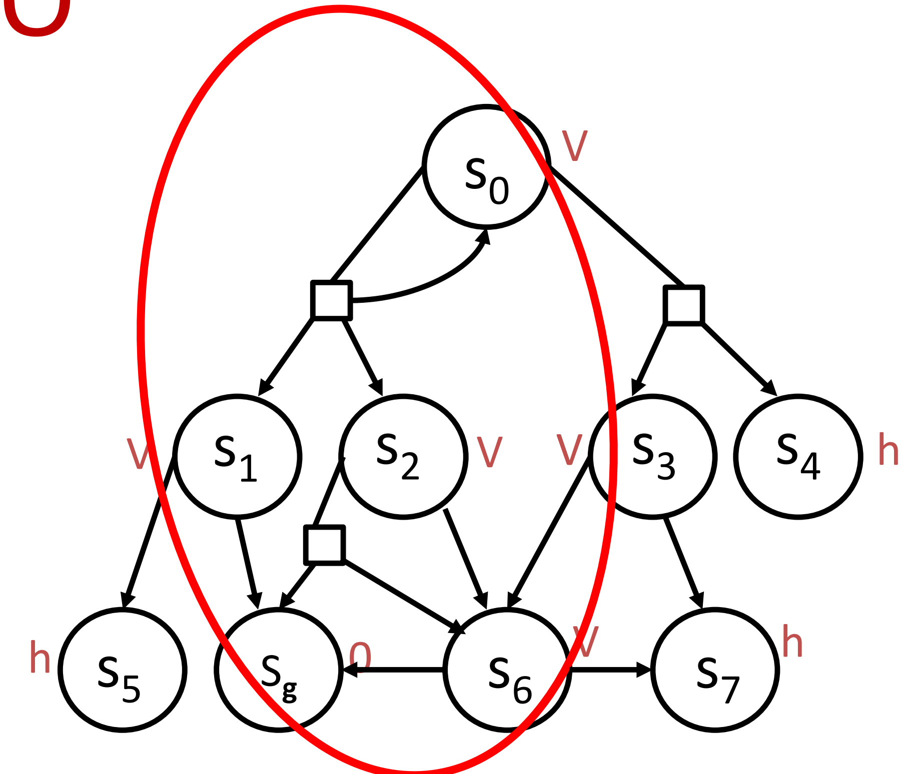
subset = all states in expanded graph that can reach s

perform VI on this subset

recompute the greedy graph

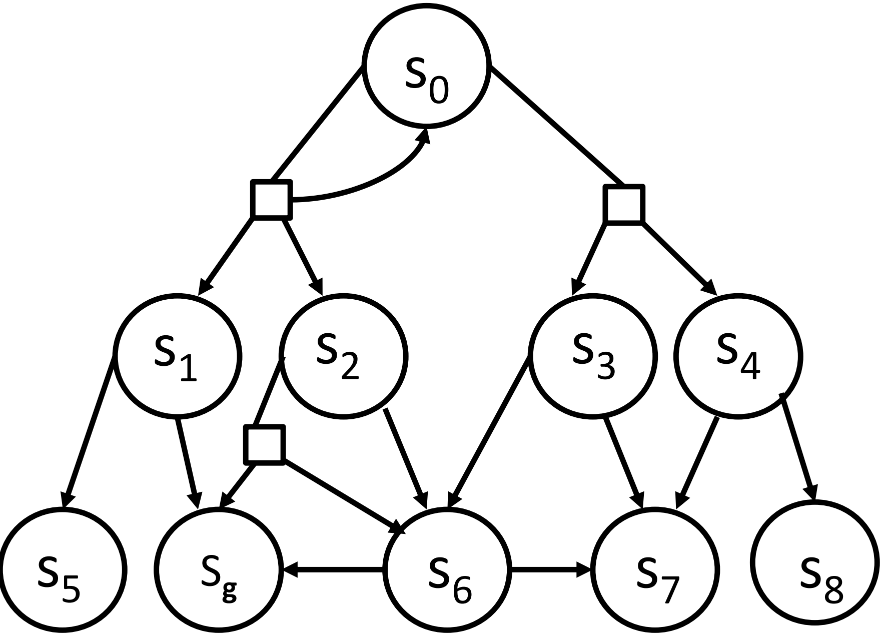
LAO*

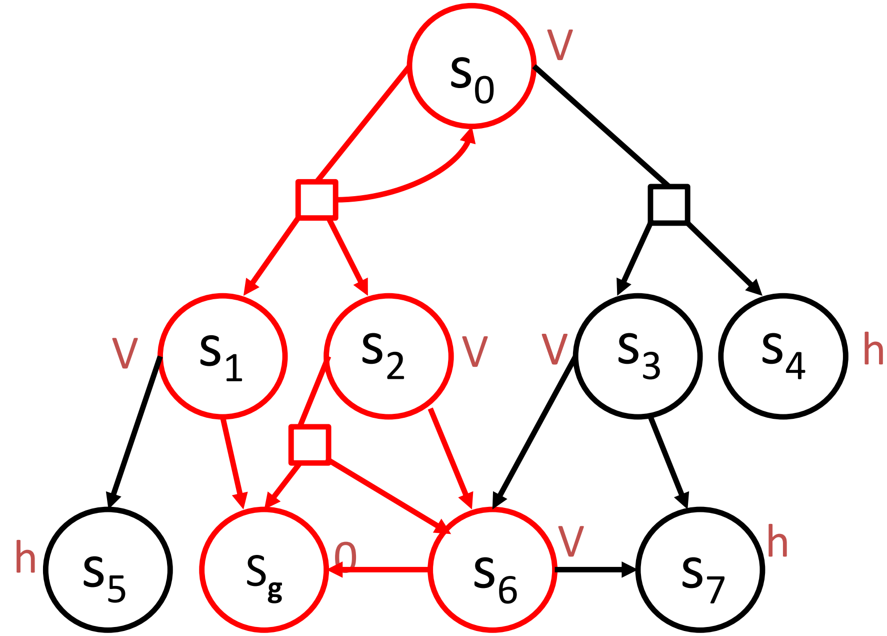




output the greedy graph as the final policy

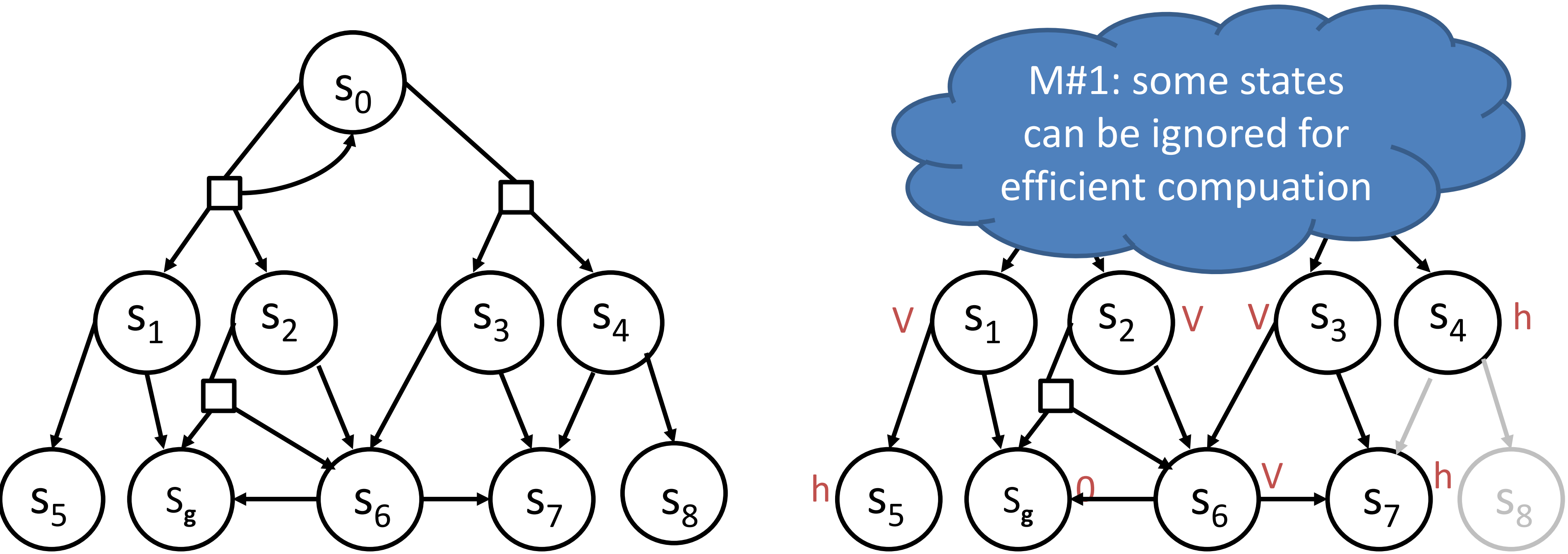
LAO*





output the greedy graph as the final policy

LAO*



s_4 was never expanded

s_8 was never touched

LAO* [Hansen&Zilberstein 98]

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
- 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* [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

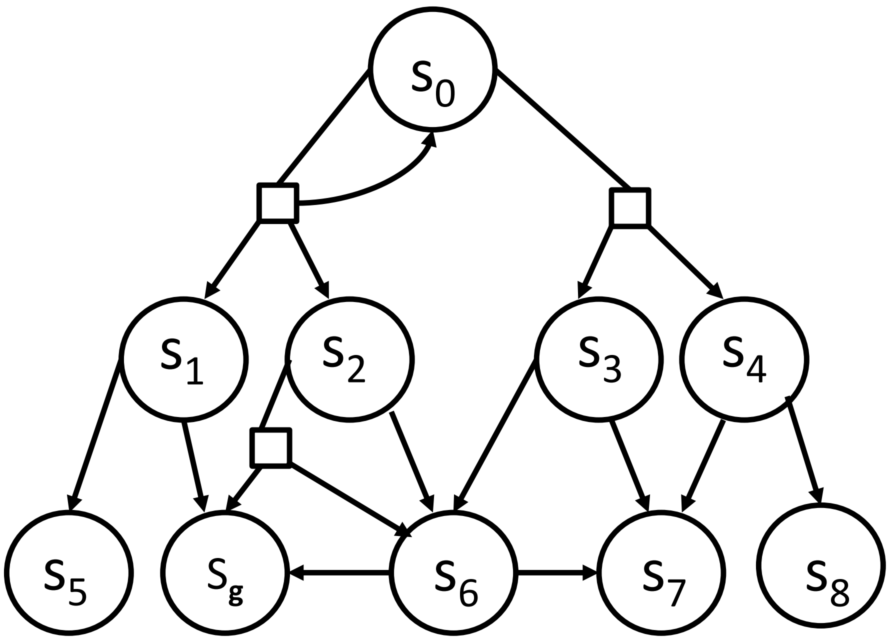
Real Time Dynamic Programming

[Barto et al 95]

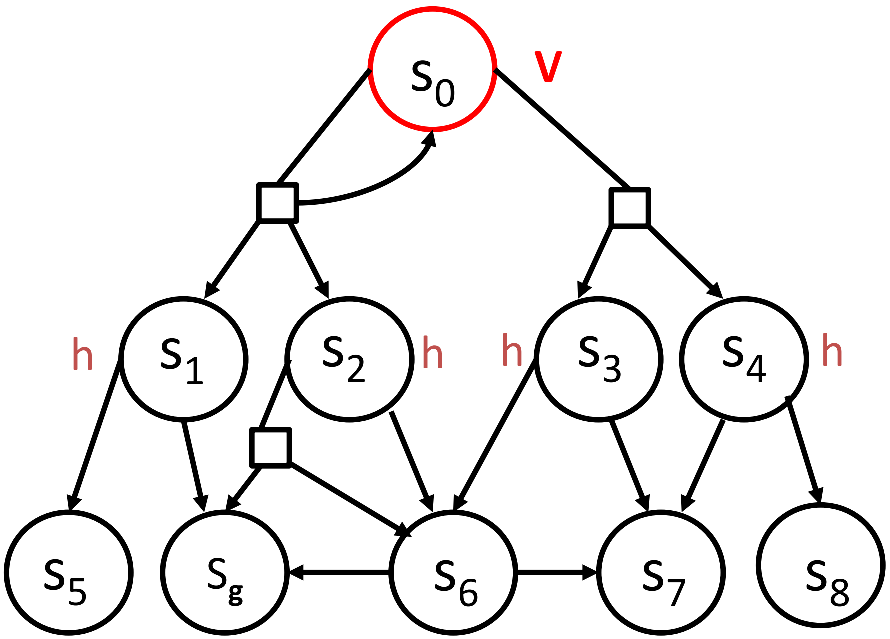
- Original Motivation
  - agent acting in the real world
- Trial
  - simulate greedy policy starting from start state;
  - perform Bellman backup on visited states
  - stop when you hit the goal
- RTDP: repeat trials forever
  - Converges in the limit #trials $\rightarrow \infty$

No termination condition!

Trial



Trial



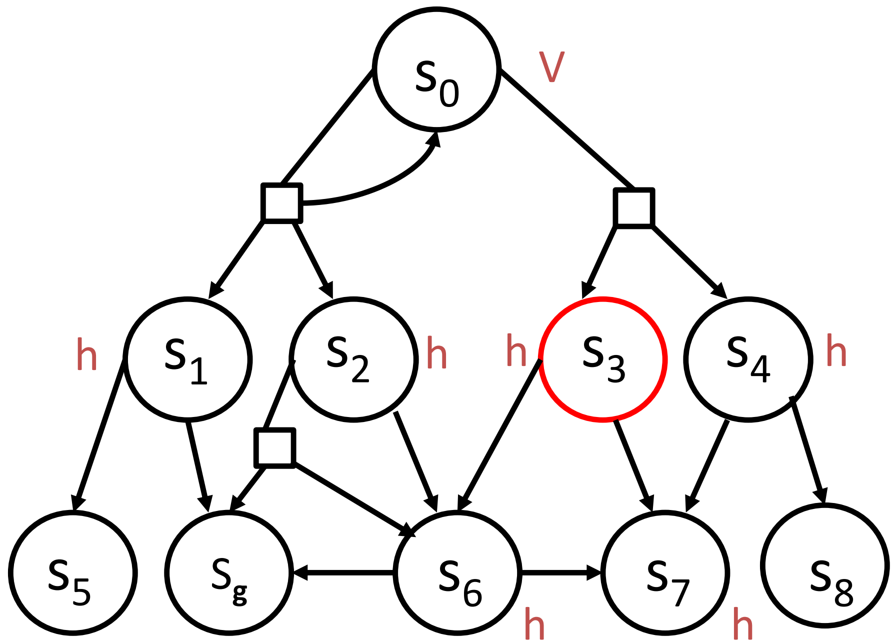
start at start state

repeat

    perform a Bellman backup

    simulate greedy action

Trial



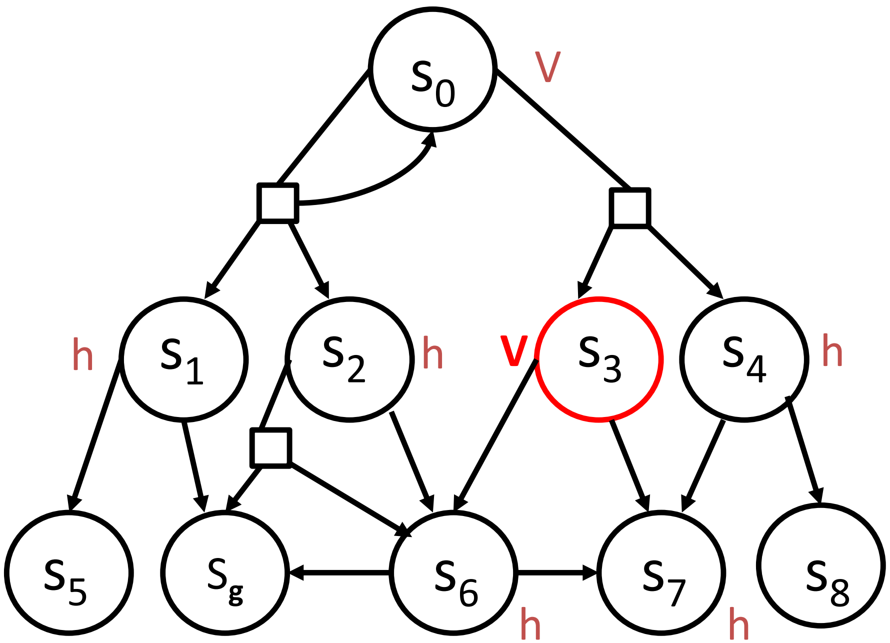
start at start state

repeat

    perform a Bellman backup

    simulate greedy action

Trial



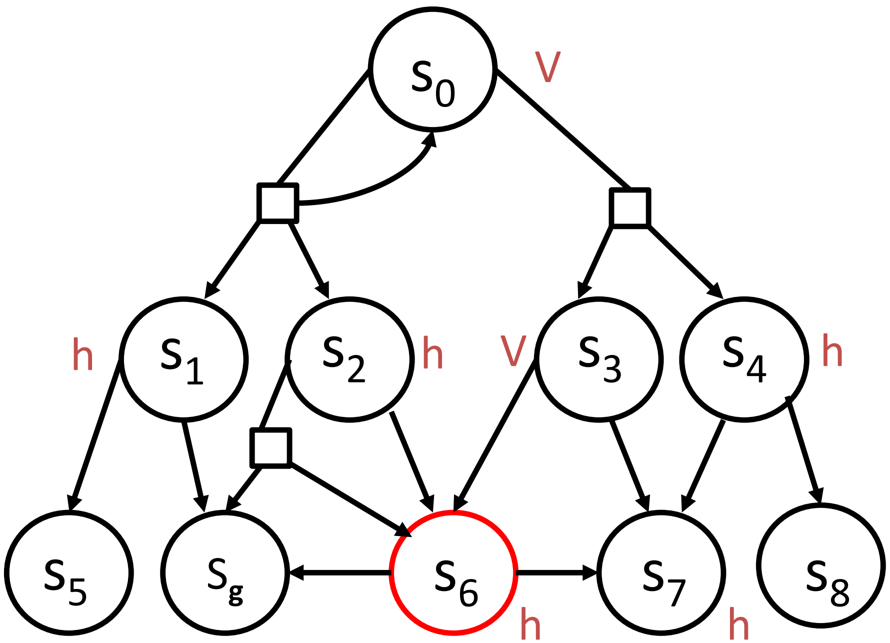
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



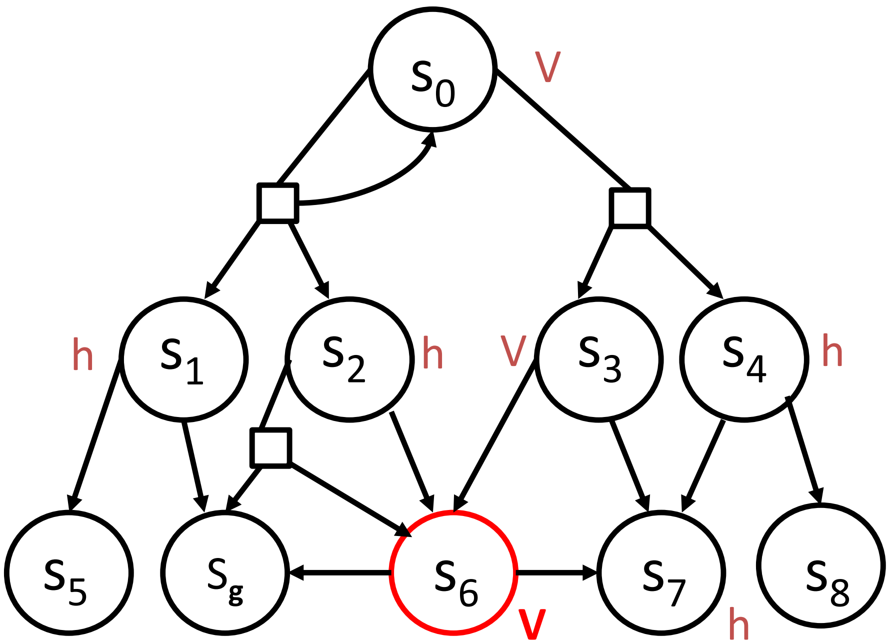
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



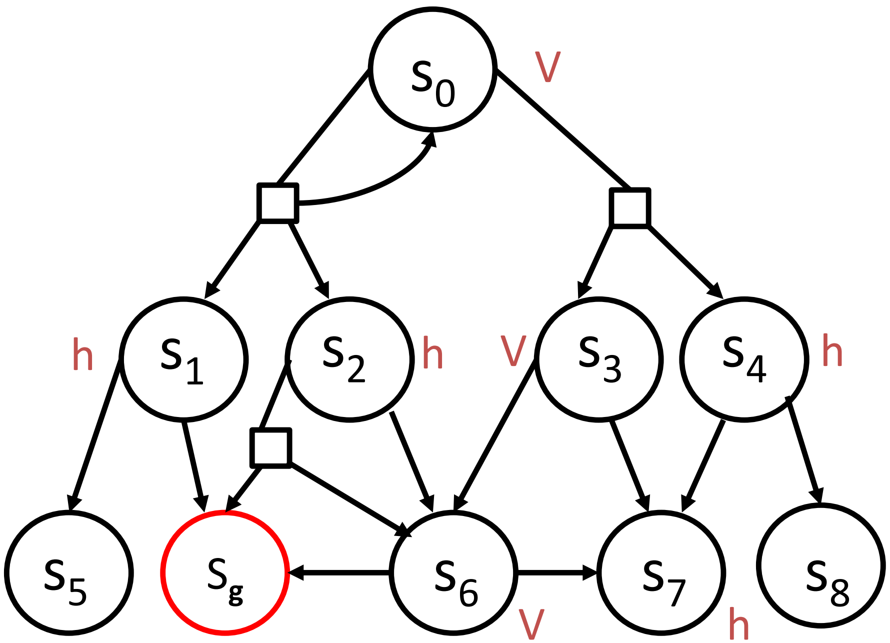
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



start at start state

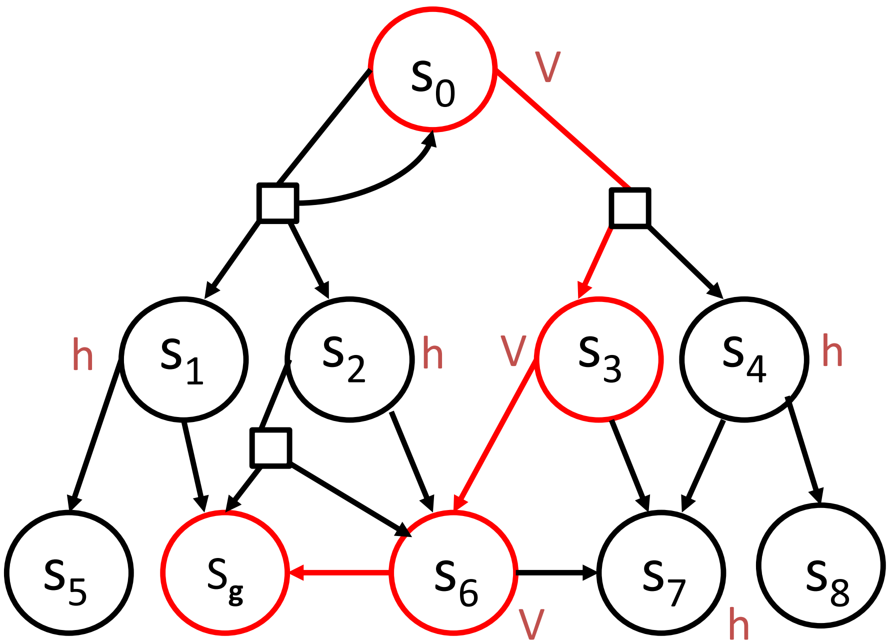
repeat

perform a Bellman backup

simulate greedy action

until hit the goal

Trial



RTDP

repeat forever

start at start state

repeat

perform a Bellman backup

simulate greedy action

until hit the goal

RTDP Family of Algorithms

repeat

$s \leftarrow s_0$

repeat *//trials*

REVISE s; identify a_{greedy}

FIND: pick s' s.t. $T(s, a_{greedy}, s') > 0$

$s \leftarrow s'$

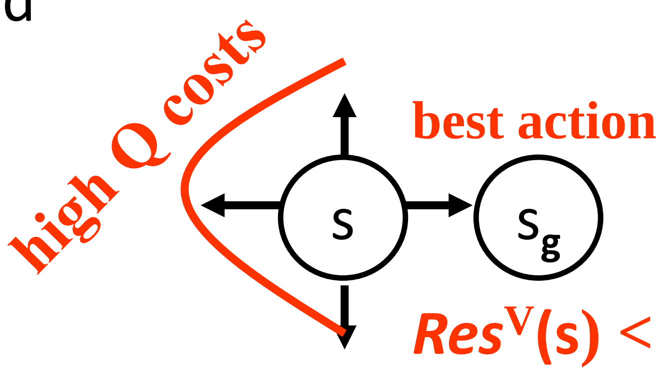
until $s \in G$

until termination test

Termination Test: Labeling

- Admissible heuristic
  - $\Rightarrow V(s) \leq V^*(s)$
  - $\Rightarrow Q(s,a) \leq Q^*(s,a)$

- Label a state s as solved
  - if V(s) has converged

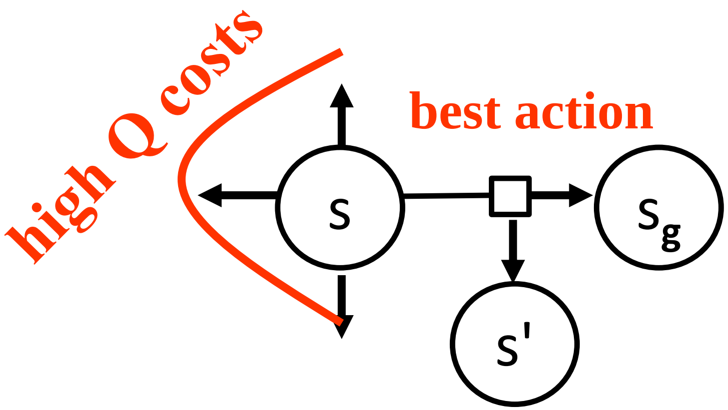


$Res^V(s) < \epsilon$

$\Rightarrow$ V(s) won't change!

label s as solved

Labeling (contd)



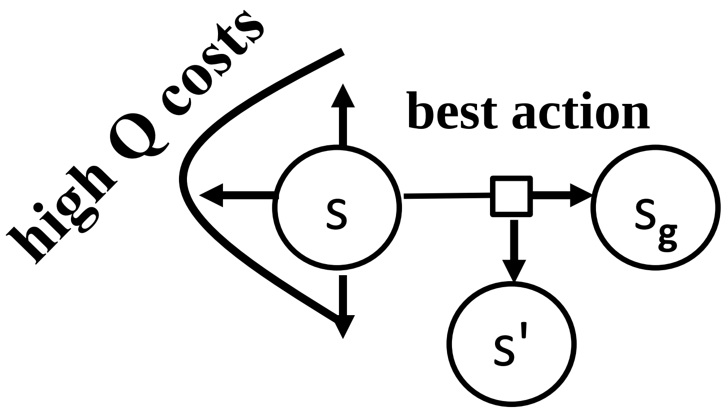
Res^V(s) $< \epsilon$

s' already solved

$\Rightarrow$ V(s) won't change!

label s as solved

Labeling (contd)

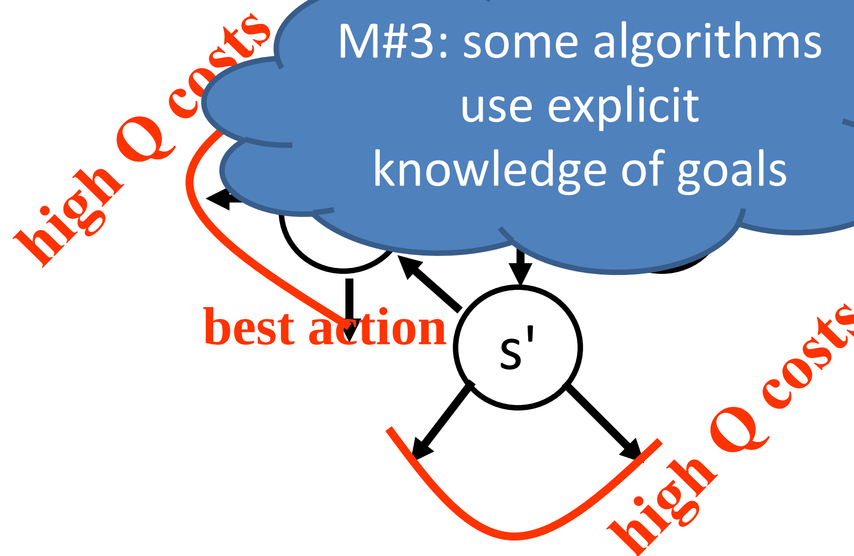


Res^V(s) < ϵ

s' already solved

$\Rightarrow$ V(s) won't change!

label s as solved



M#3: some algorithms use explicit knowledge of goals

M#1: some states can be ignored for efficient computation

label s, s' as solved

Labeled RTDP [Bonet&Geffner 03b]

repeat

  $s \leftarrow s_0$

  label all goal states as solved

  repeat *//trials*

    REVISE s; identify a_{greedy}

    FIND: sample s' from $T(s, a_{greedy}, s')$

    $s \leftarrow s'$

  until s is solved

  for all states s in the trial

    try to label s as solved

until s_0 is solved

LRTDP

- terminates in finite time
  - due to labeling procedure
- anytime
  - focuses attention on more probable states
- fast convergence
  - focuses attention on unconverged states

LRTDP Extensions

- Different ways to pick next state
- Different termination conditions

- Bounded RTDP [McMahan et al 05]
- Focused RTDP [Smith&Simmons 06]
- Value of Perfect Information RTDP [Sanner et al 09]

Where do Heuristics come from?

- Domain-dependent heuristics

- Domain-independent heuristics
  - dependent on specific domain representation

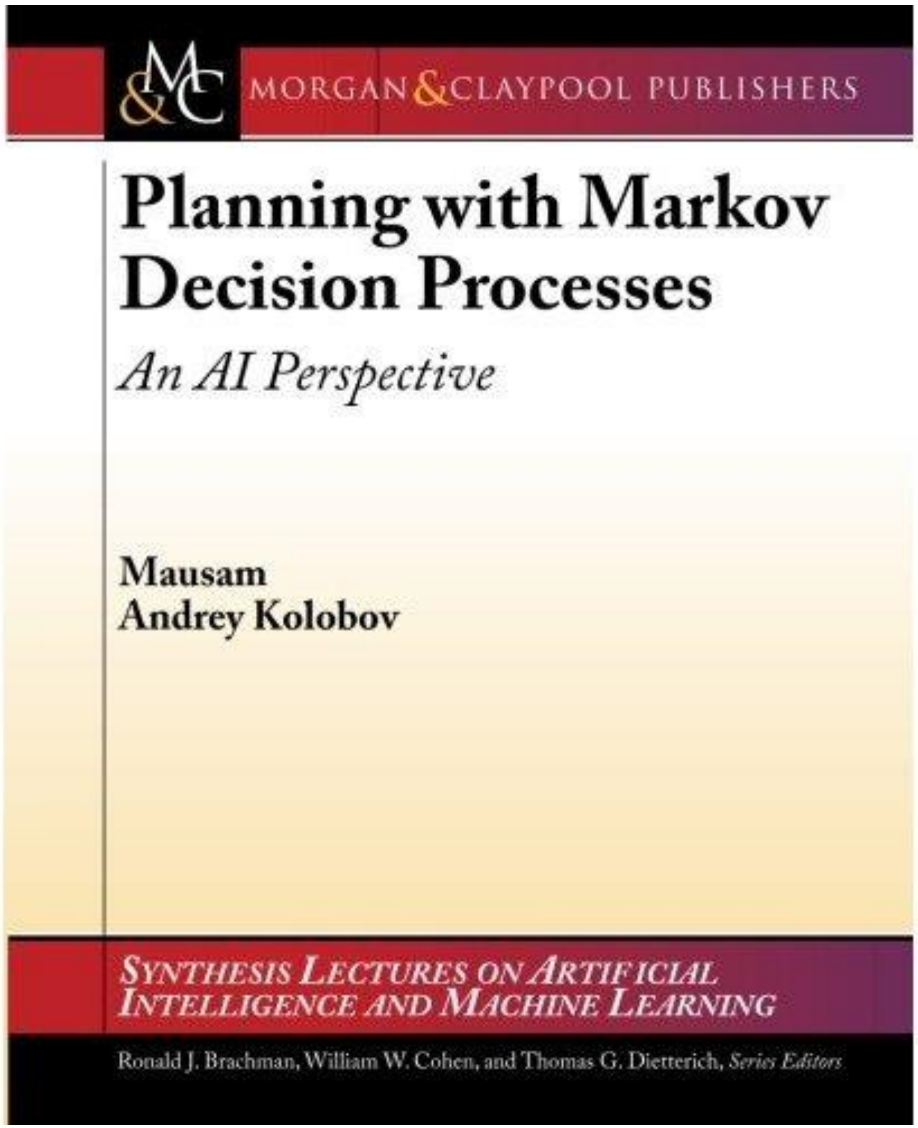
M#2: factored representations expose useful problem structure

Take-Homes

- efficient computation given start state s_0
  – heuristic search

- automatic computation of heuristics
  – domain independent manner

Shameless Plug



Agenda

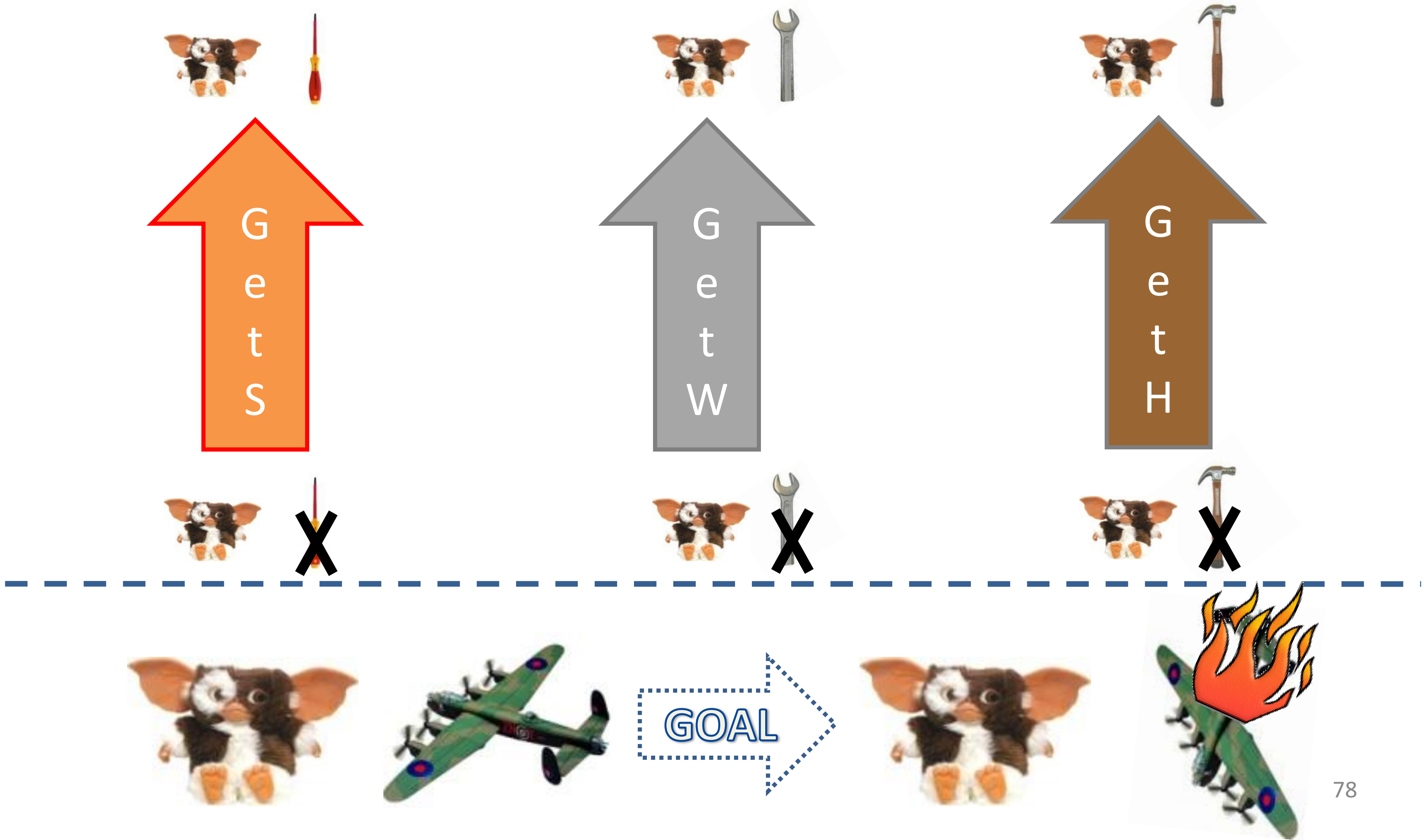
- Background: Stochastic Shortest Paths MDPs
- Background: Heuristic Search for SSP MDPs
- Algorithms: Automatic Basis Function Discovery
- Models: SSPs → Generalized SSPs

Previous Work
Our Work

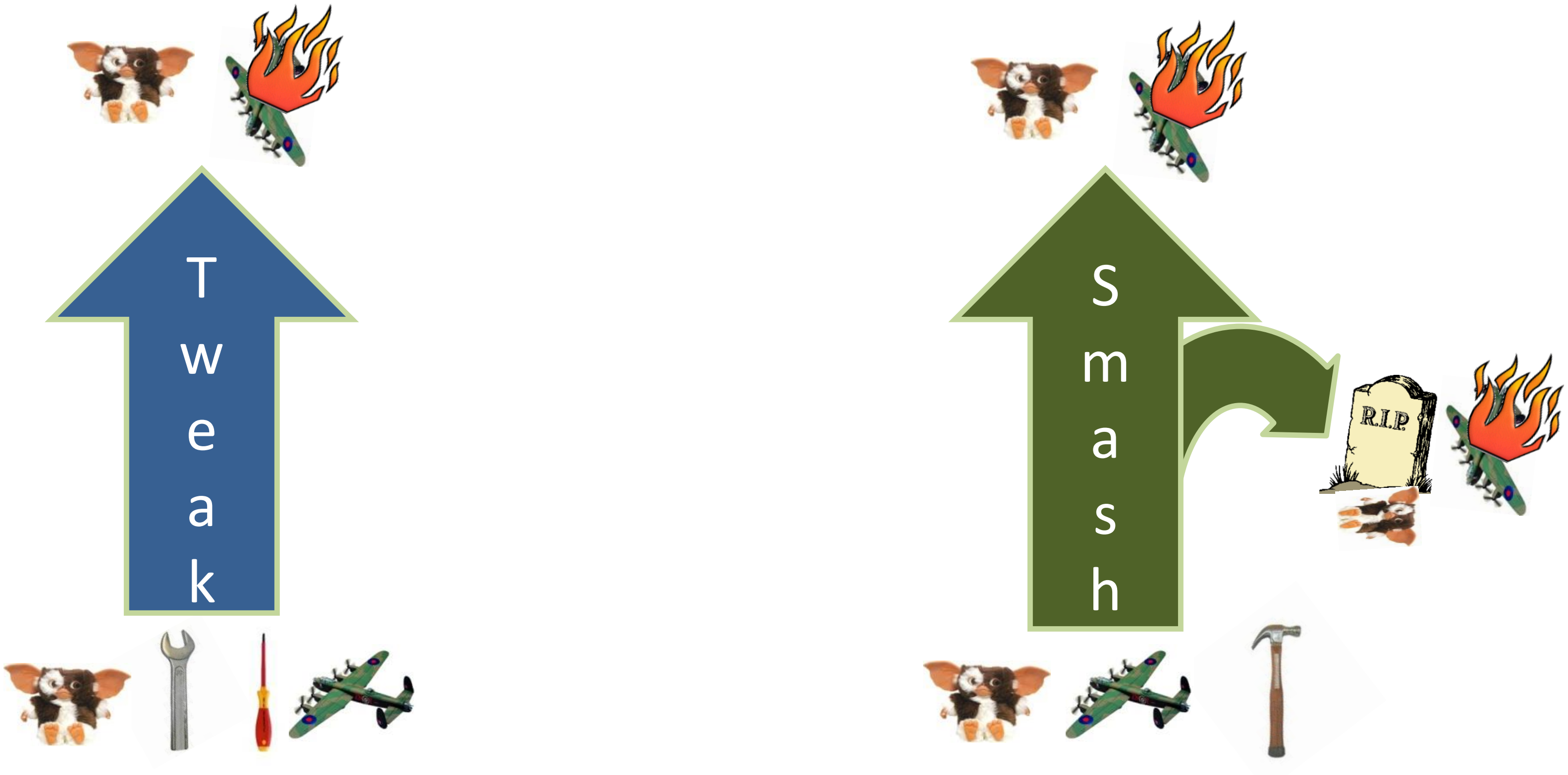
- Determinization
  - Determinize the MDP
  - Classical planners ***fast***
  - E.g., FF-Replan
  - Cons: may be troubled by
    - Complex contingencies
    - Probabilities
- Function Approximation
  - Dimensionality reduction
  - Represent state values with basis functions
    - E.g., $V^*(s) \approx \sum_i w_i b_i(s)$
  - Cons:
    - Need a human to get b_i

Marry these paradigms to extract problem-specific structure in a fast, problem-independent way.

Example Domain



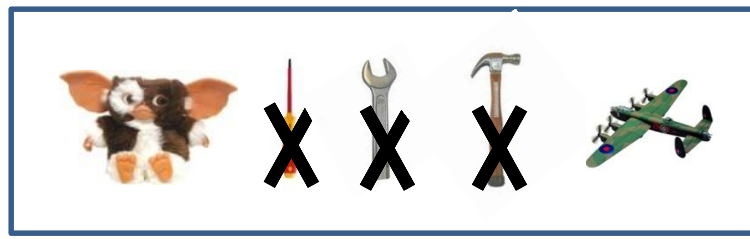
Example Domain (cont'd)



SSP_{s0} MDP

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): action cost
- s_0: start state
- G: set of goals

GetW, GetH, GetS, Tweak, Smash



Contributions

ReTrASE — a *scalable* approximate MDP solver

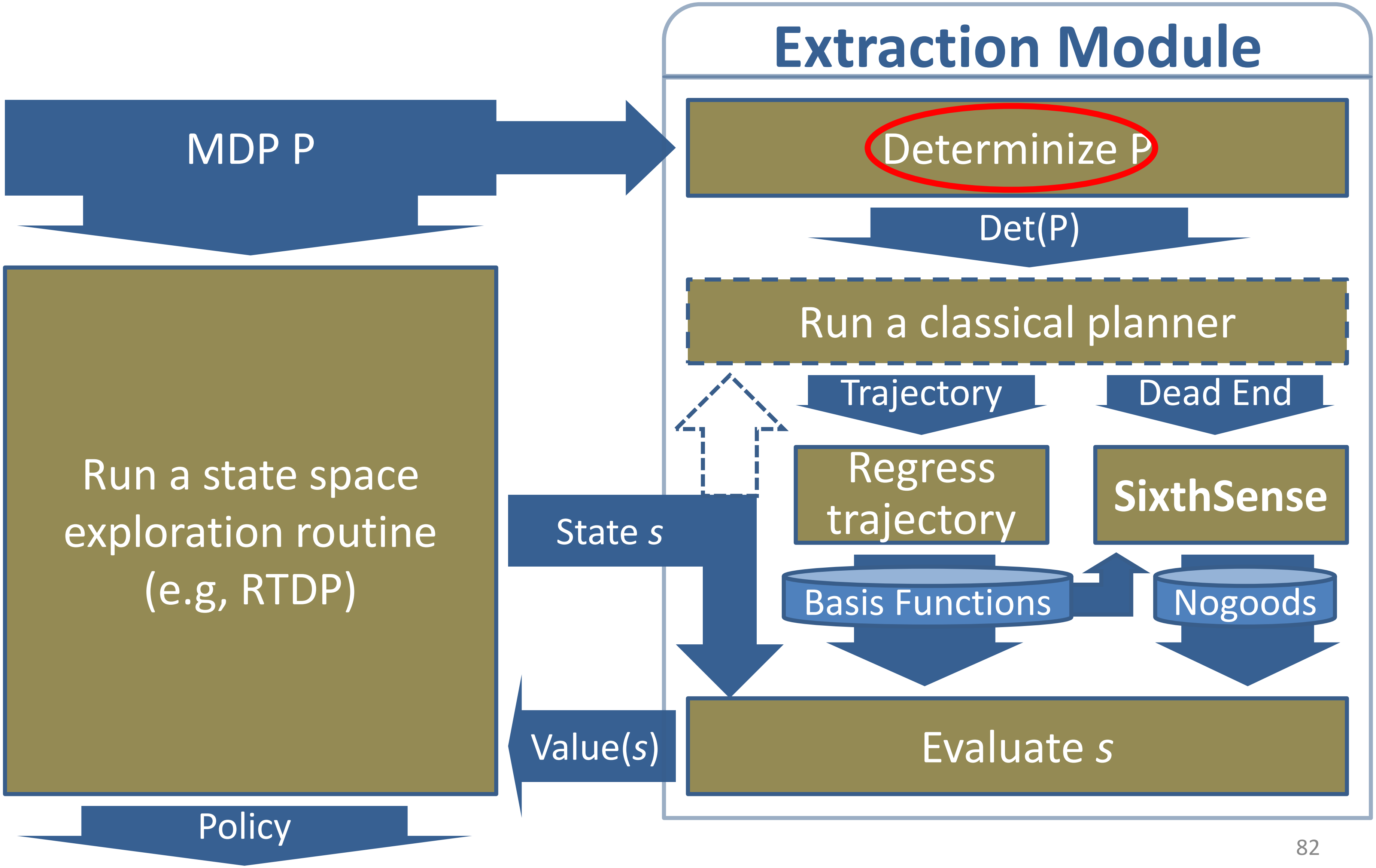
– Combines function approximation with classical planning

– Uses classical planner to automatically generate basis functions

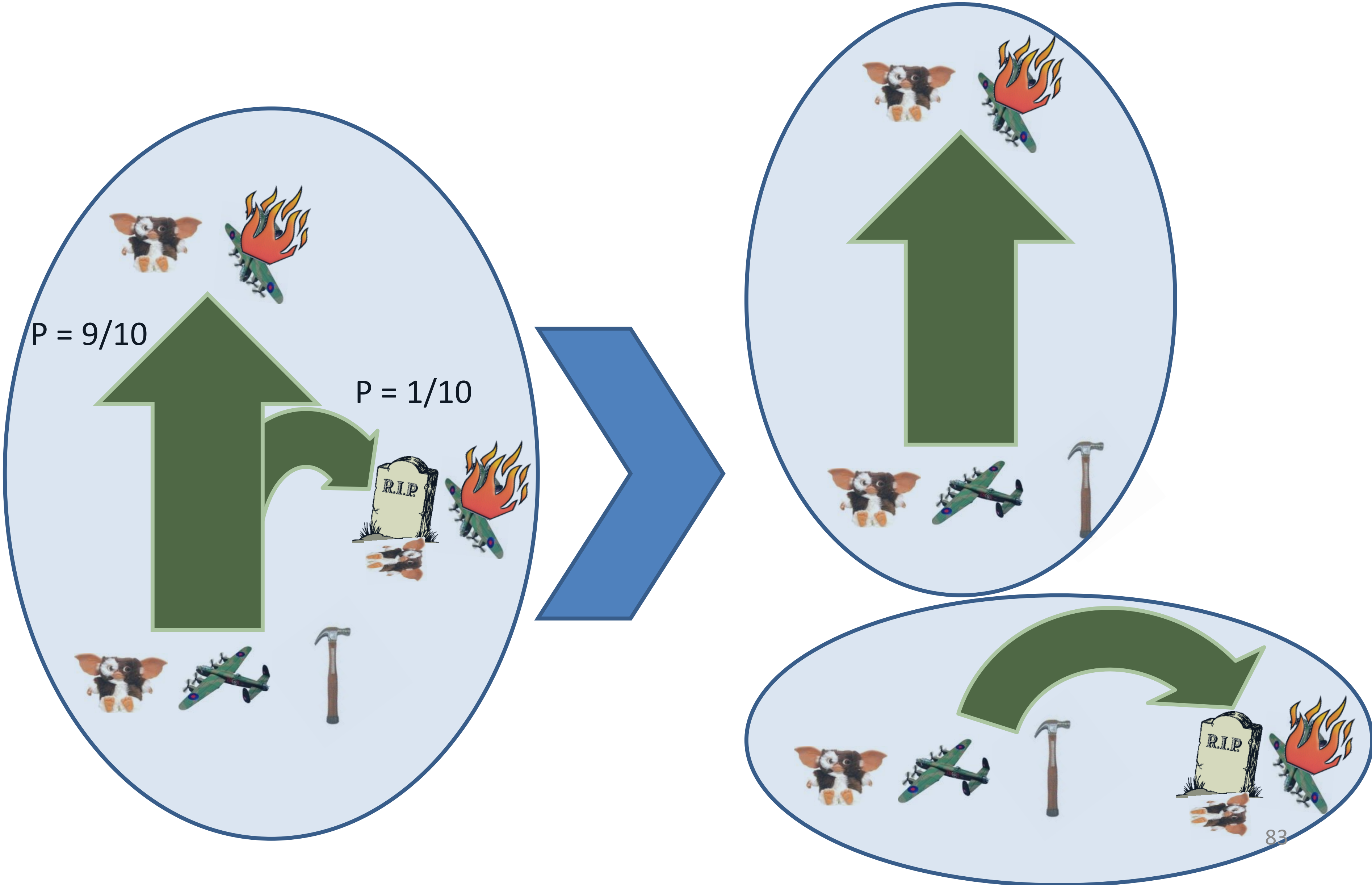
– Fast, memory-efficient, high-quality policies

The Big Picture: **ReTrASE**

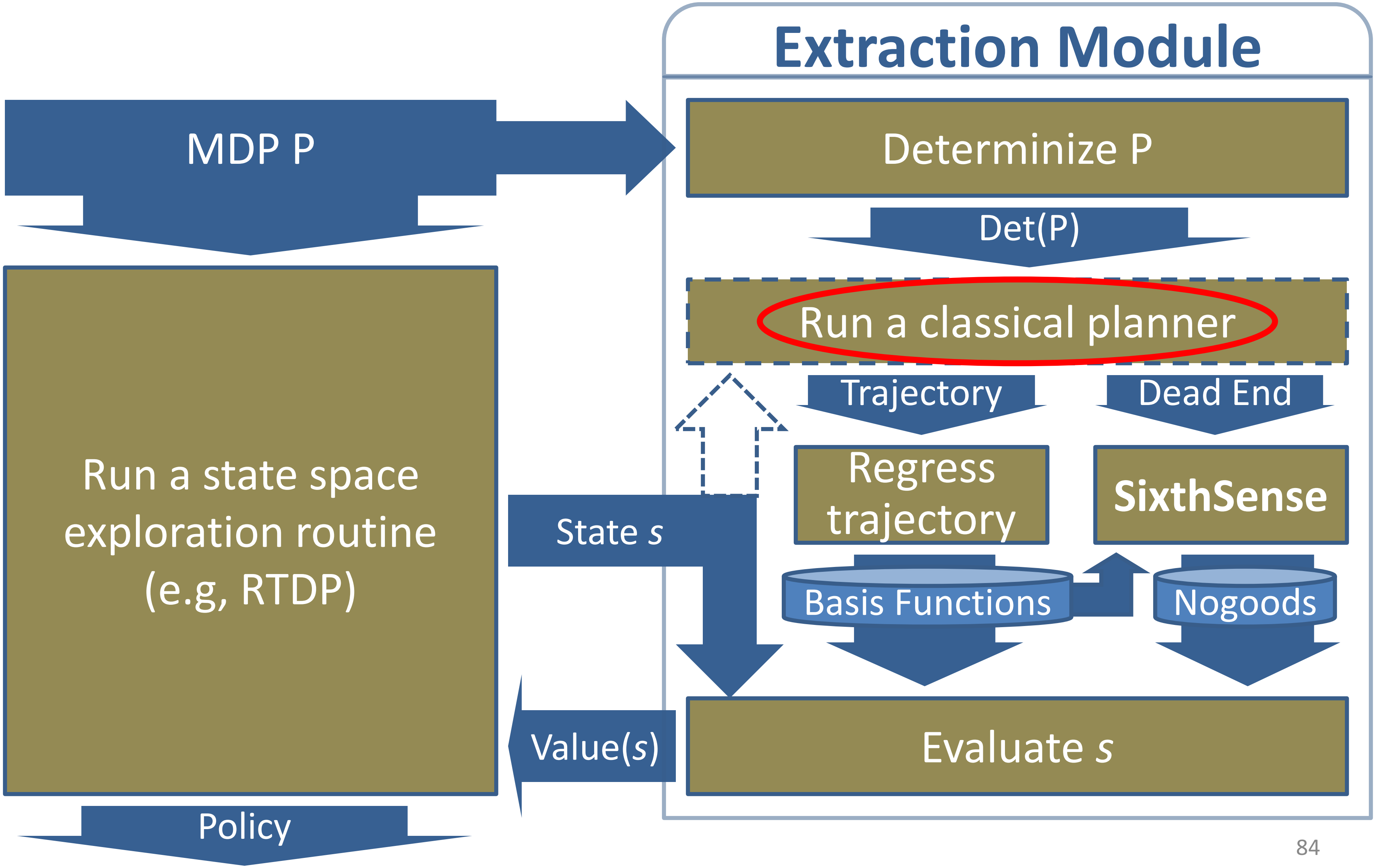
[Kolobov, Mausam, Weld, AIJ'12]



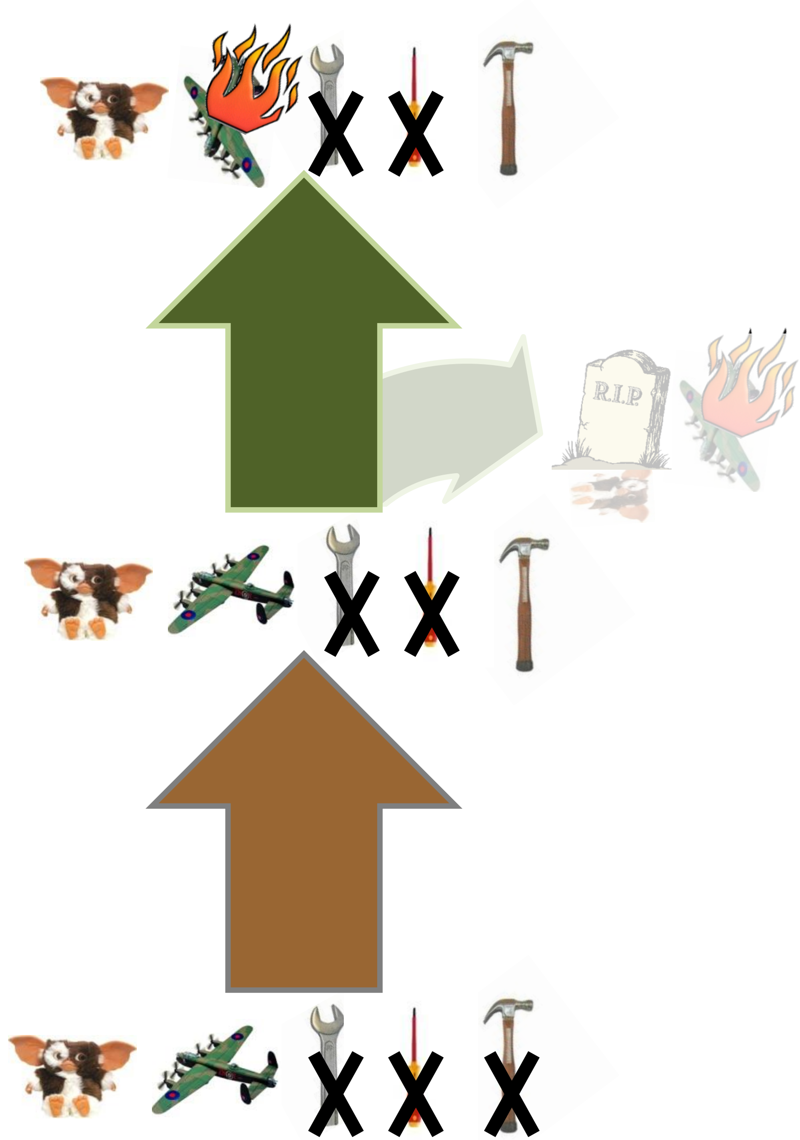
Determinizing the Domain



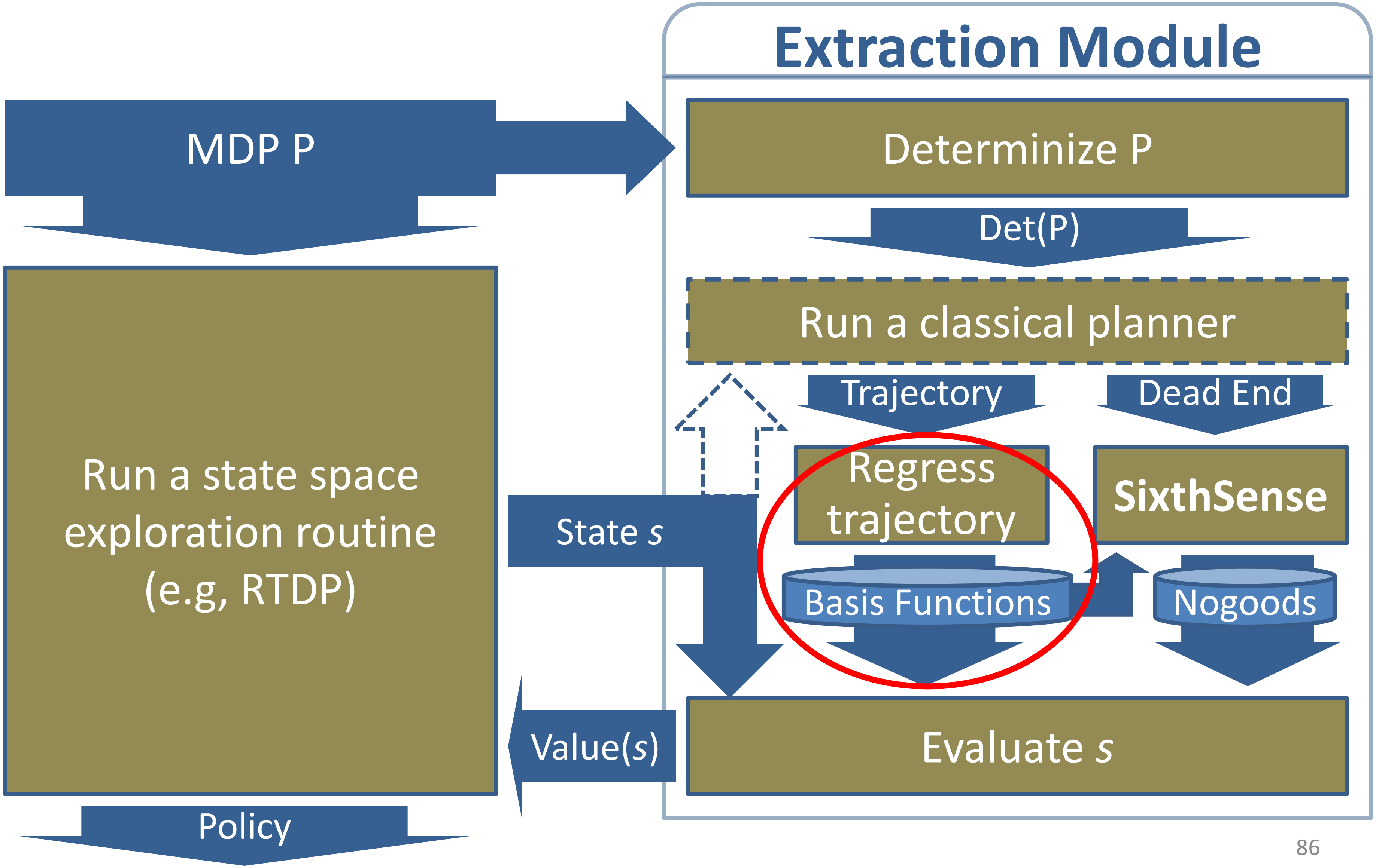
Generating Trajectories



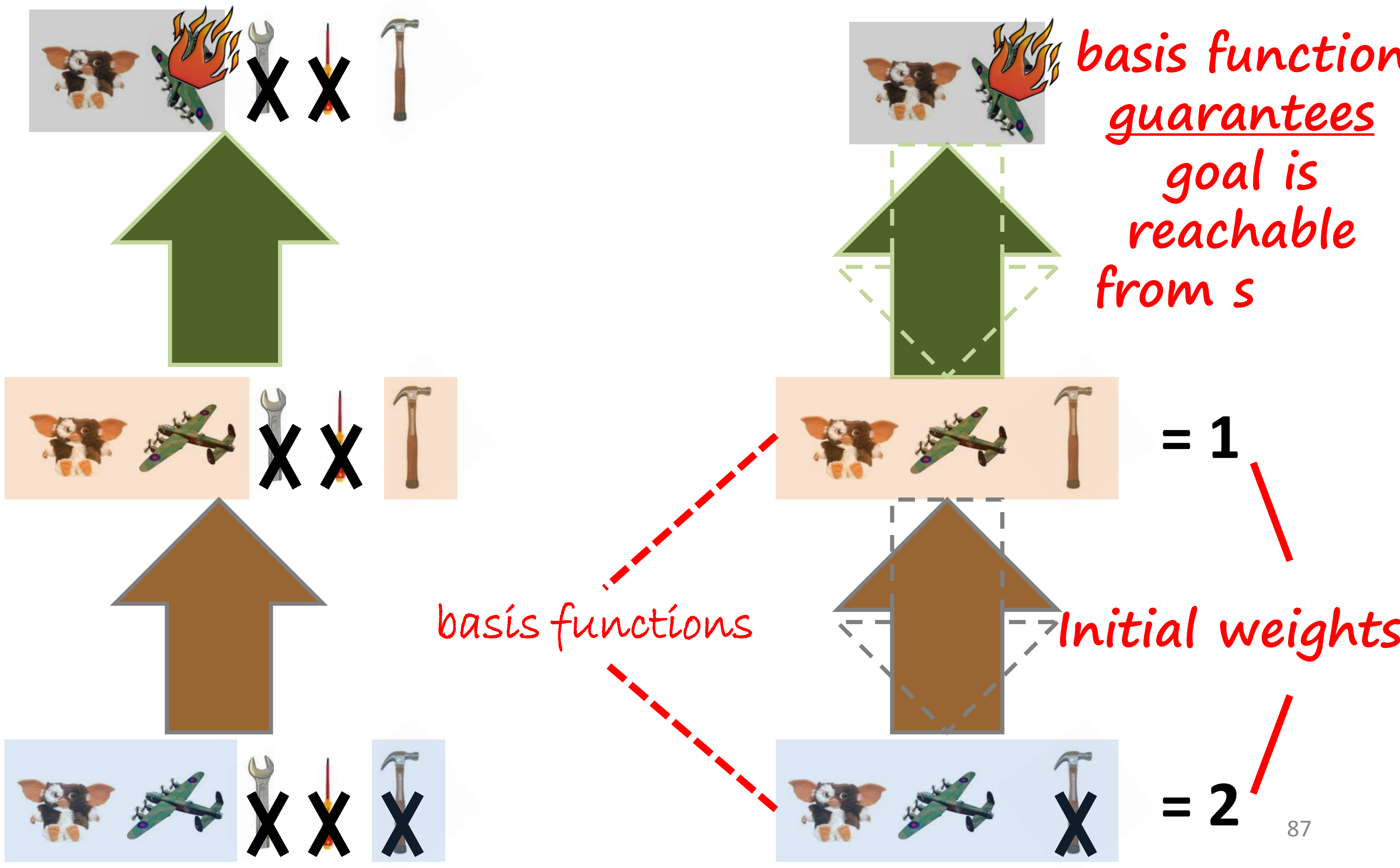
Generating Trajectories

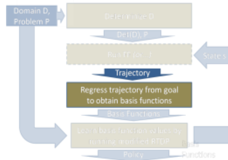


Computing Basis Functions

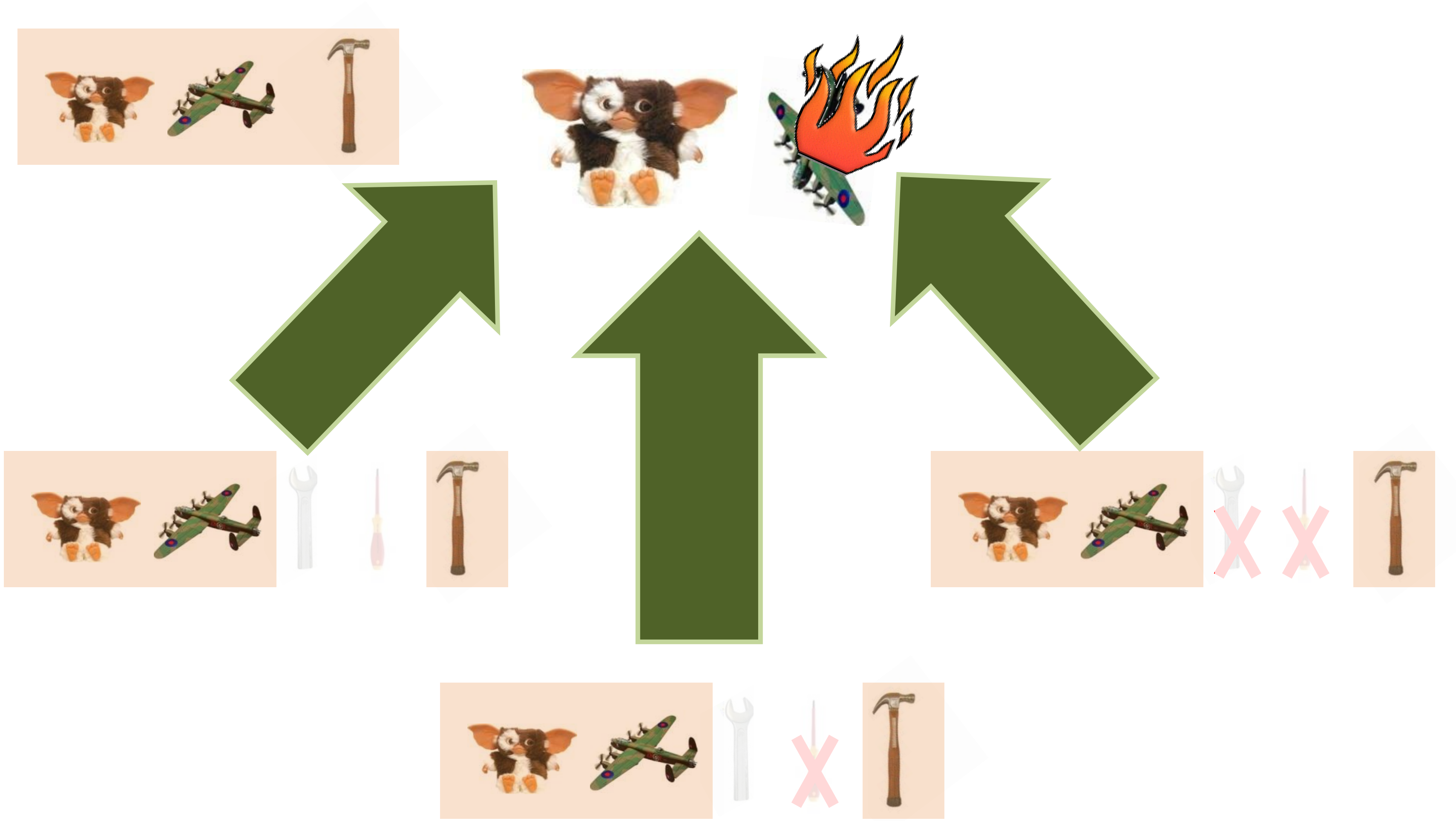


Regressing Trajectories

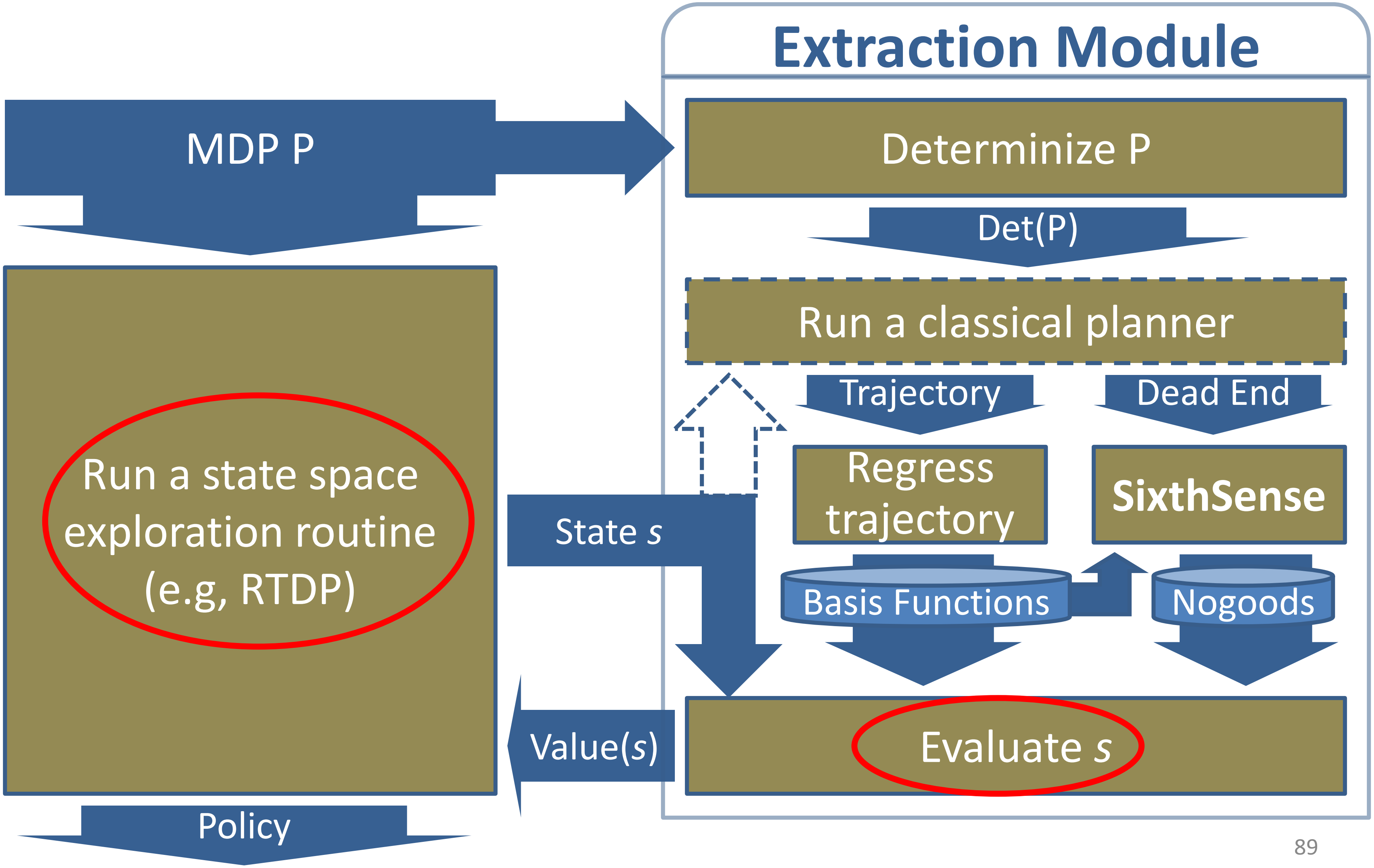




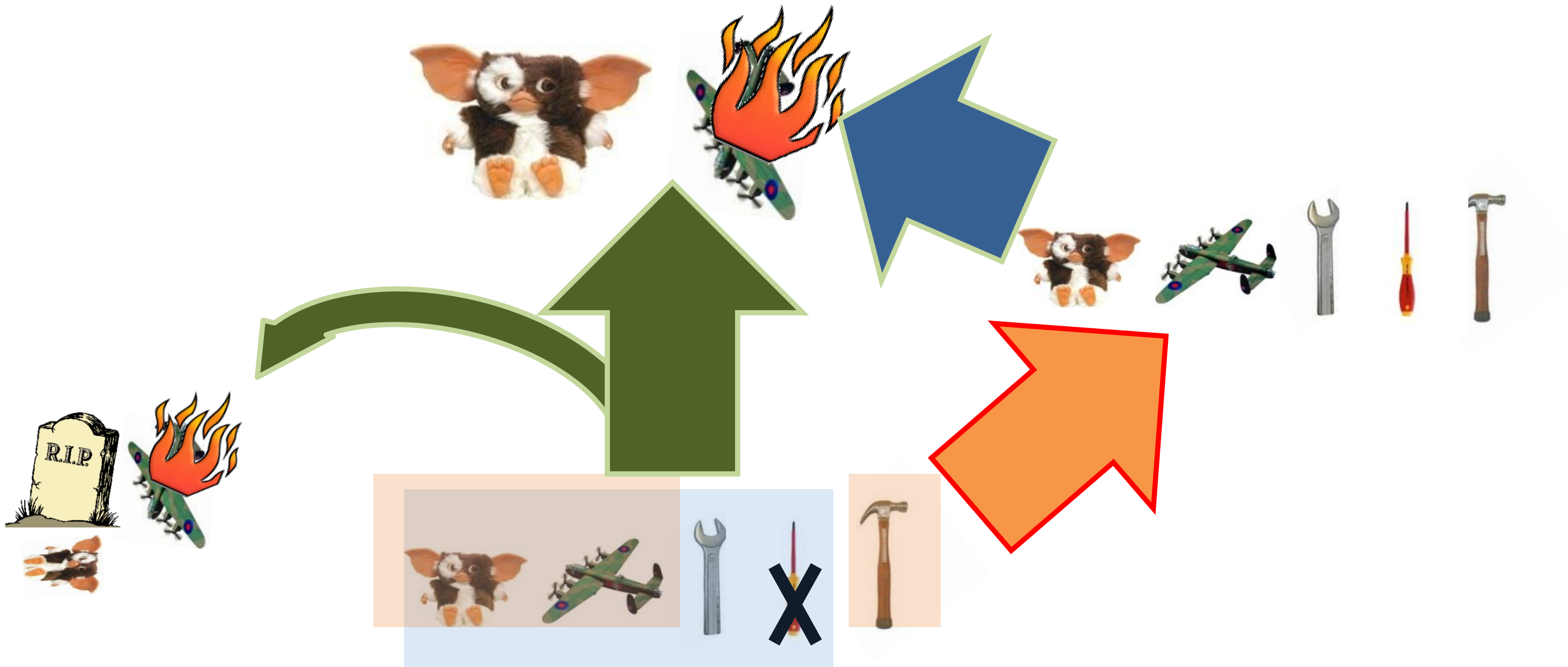
Basis Functions



Computing Values



Meaning of Basis Function Weights



Want to compute basis function weights so that the blue basis function looks "better" than the pink one!

Value of a Basis Function

- Basis function enables at least one trajectory
  - applicable from all relevant states
- Trajectories combine to form policies
- *Value* of a basis function ~ "quality" of its policies

- Algorithm based on RTDP
  - Learn basis function values
  - Use them to compute values of states

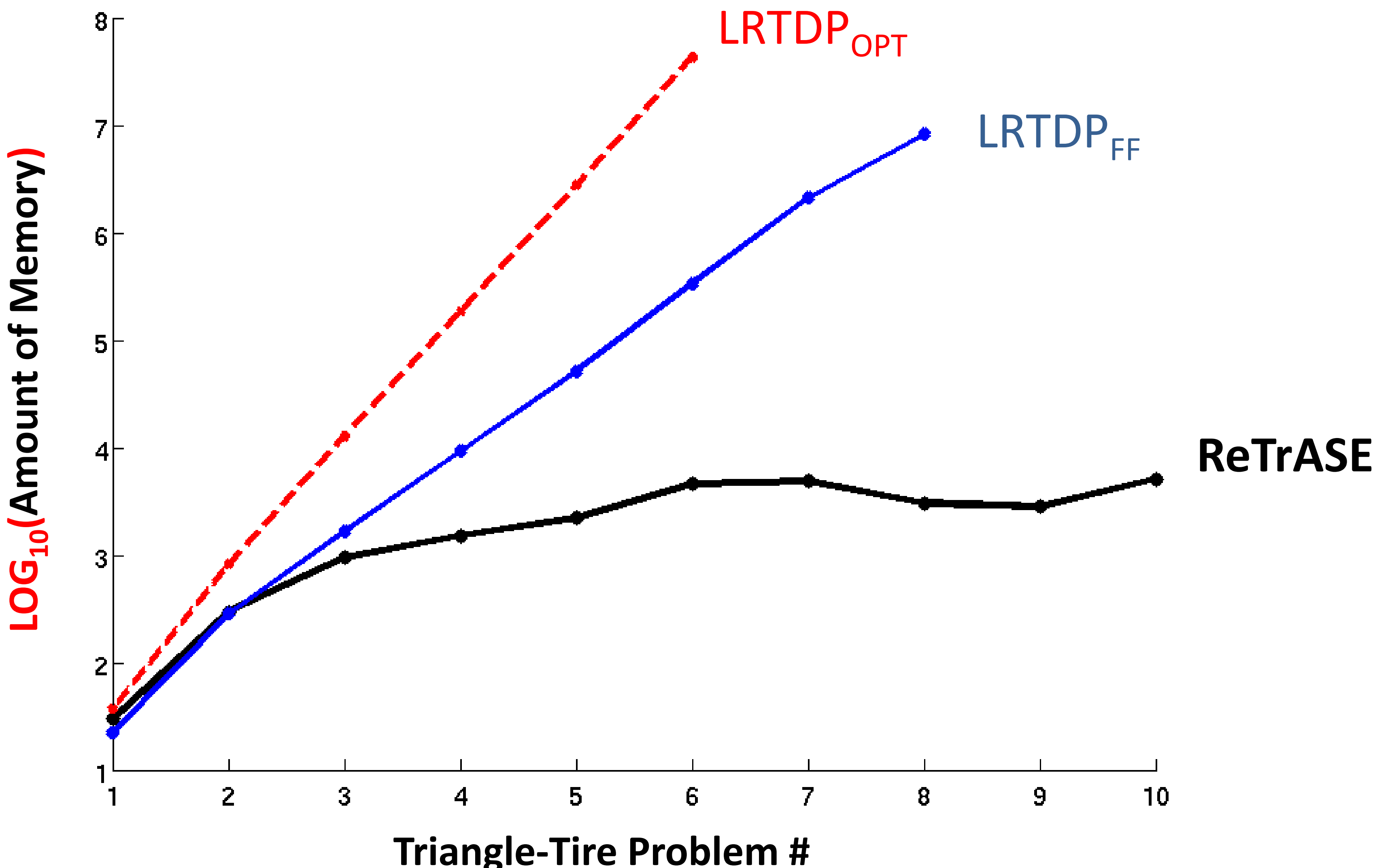
Experimental Results

- Criteria:
  - Scalability (vs. VI/RTDP-based planners)
  - Solution quality (vs. IPPC winners)
- Domains: 6 from IPPC-06 and IPPC-08
- Competitors:
  - Best performer on the particular domain
  - Best performer in the particular IPPC
  - LRTDP

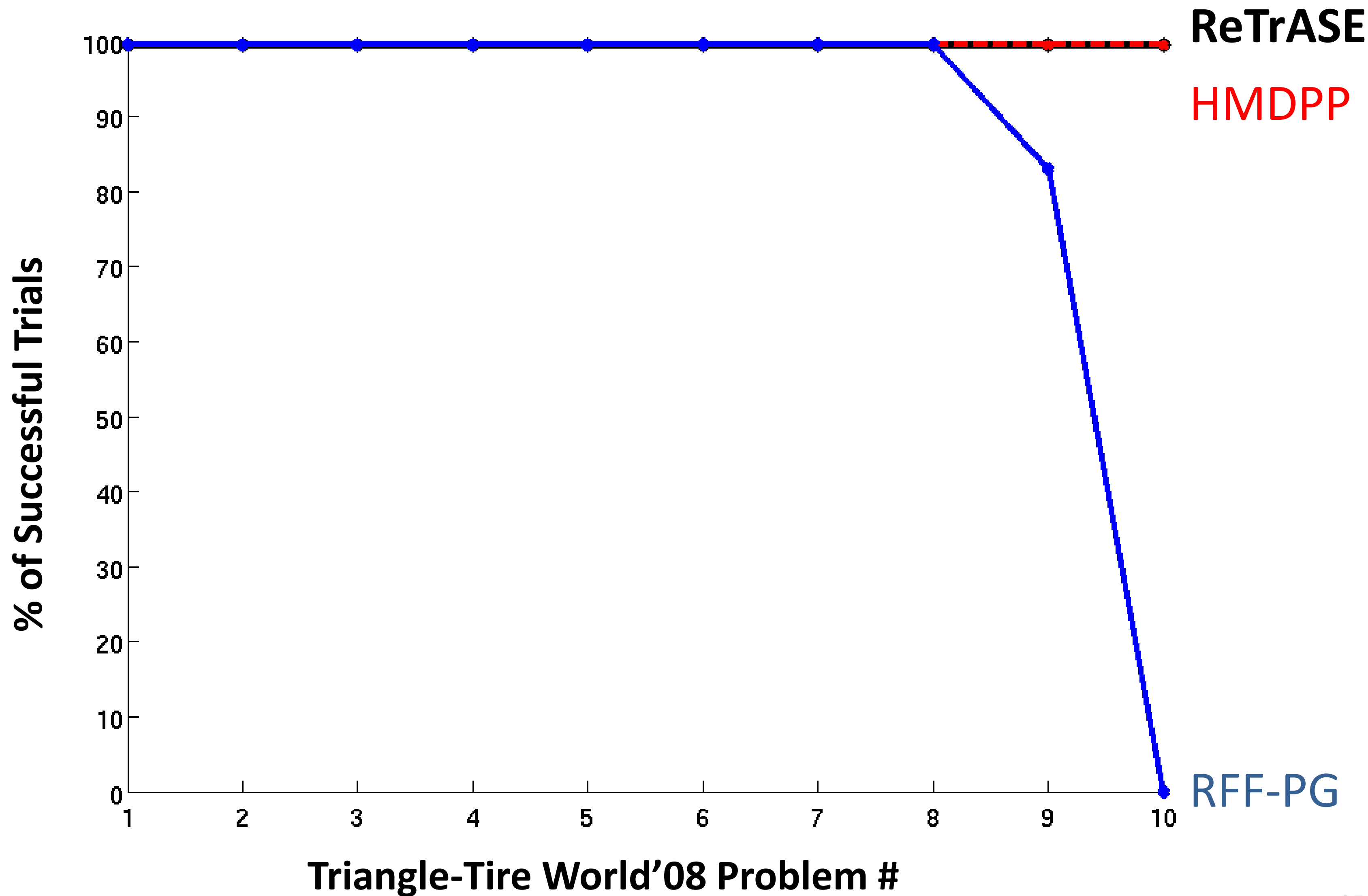
The Big Picture

- ReTrASE is vastly more scalable than VI/RTDP-based planners
- ReTrASE typically rivals or outperforms the best-performing planners on IPPC goal-oriented domains

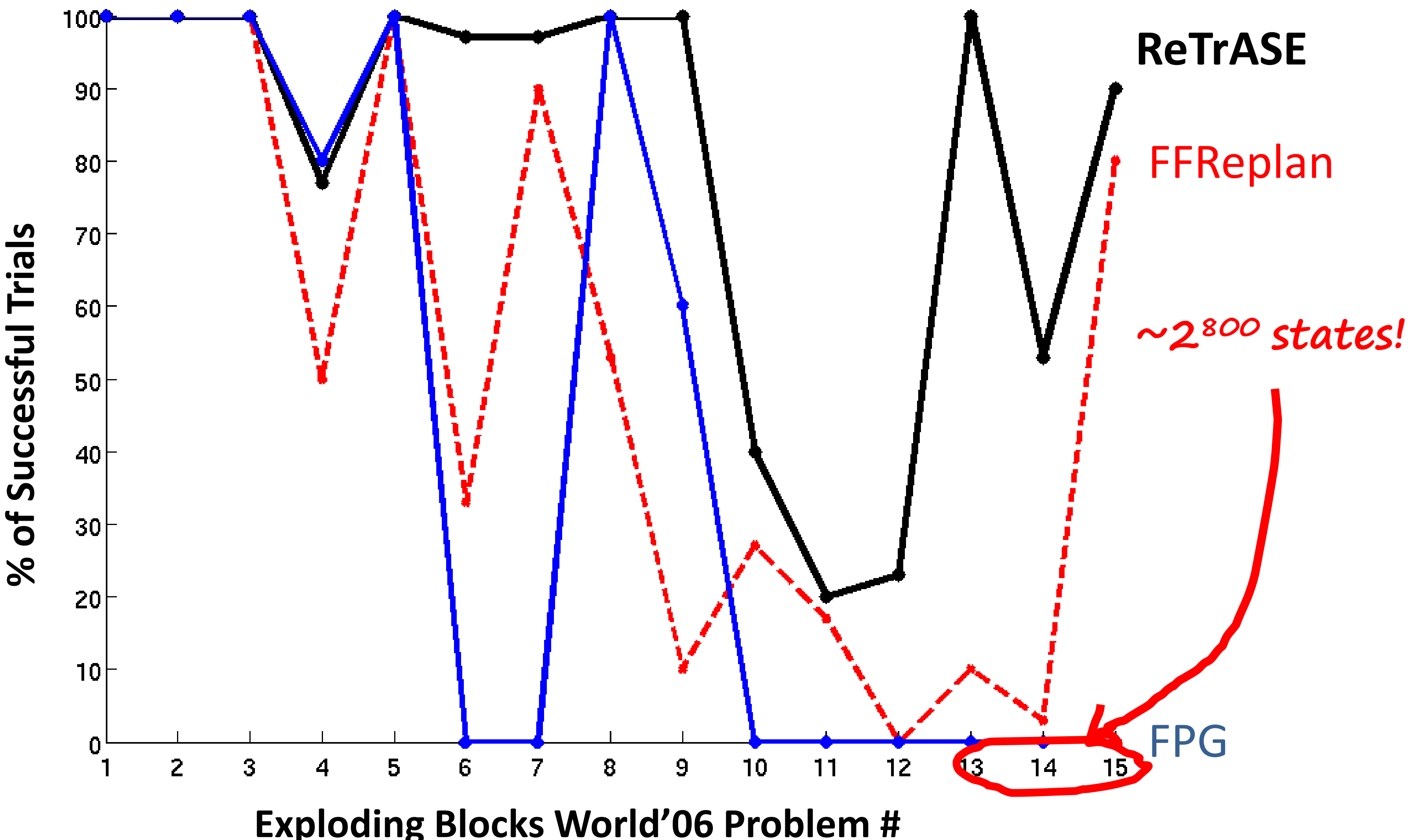
Triangle-Tire: Memory Consumption



Triangle-Tire: Success Rate



Exploding Blocks World: Success Rate



~~SSP_{s0}~~

- S: A set of states
- A: A set of actions
- T(s,a,s'): transition model
- C(s,a,s'): cost
- G: set of goals
- s_0: start state

?

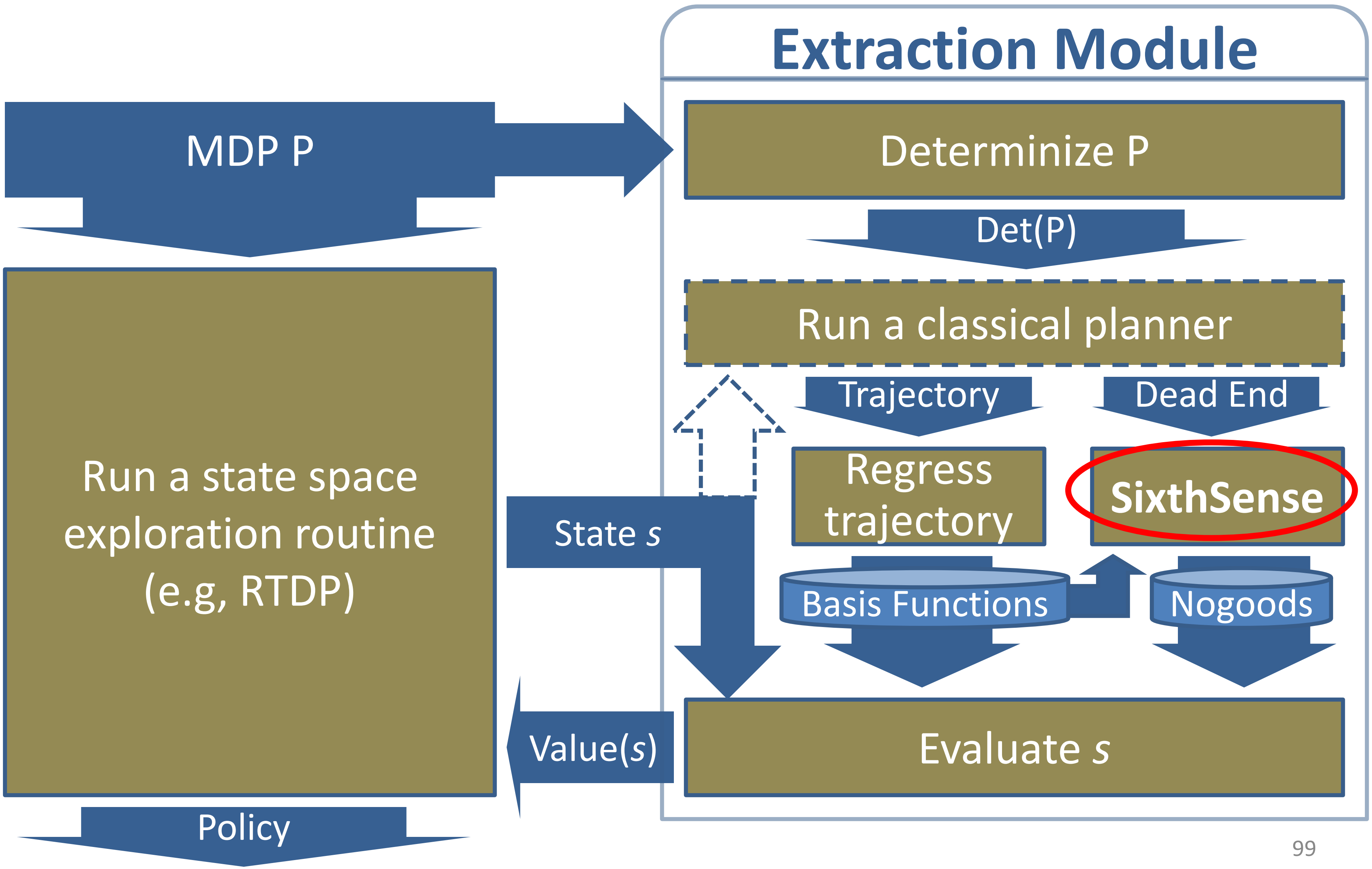
Under two conditions:

- 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

Key Drawback of ReTrASE…

- Dead-end handling expensive
  - expensive to identify: drain on time
  - too many to store: drain on space

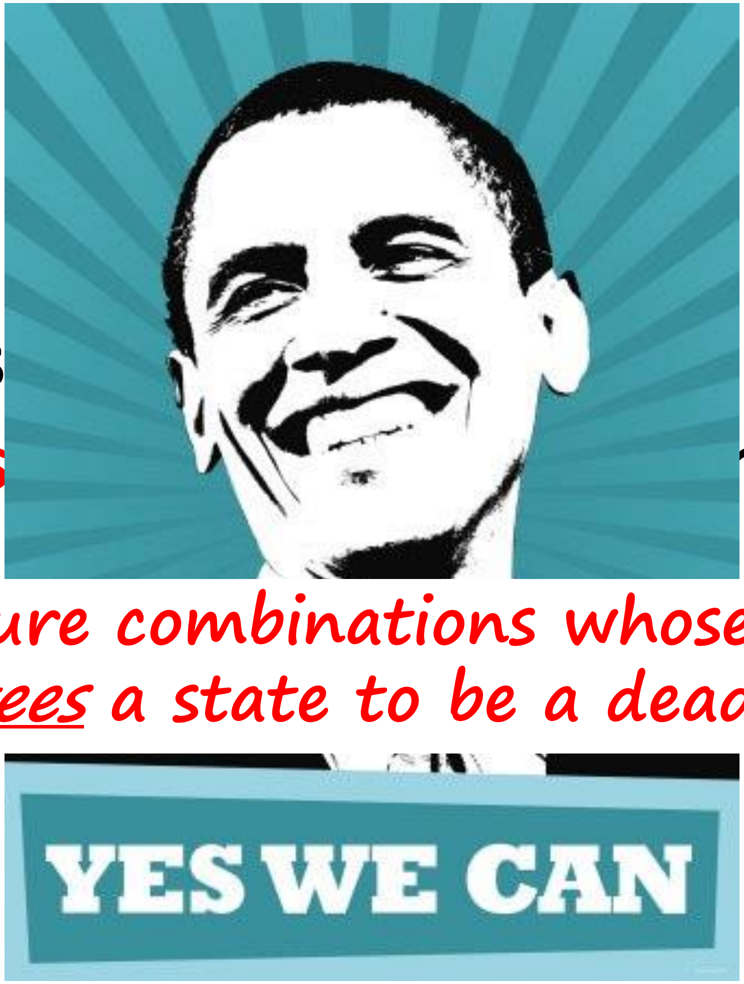
Computing Values



Research Question

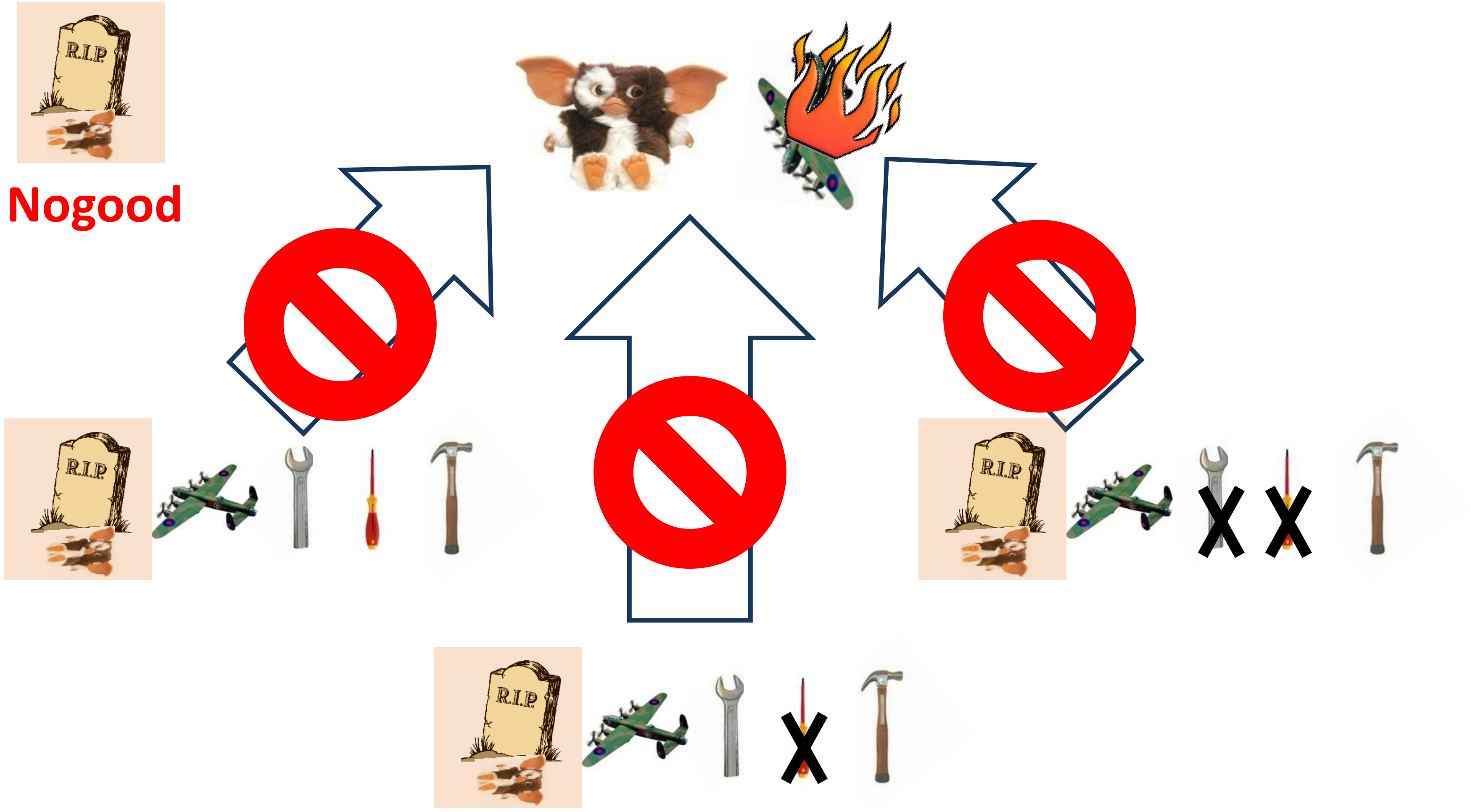
Can we devis dentification

procedure fas memoization?



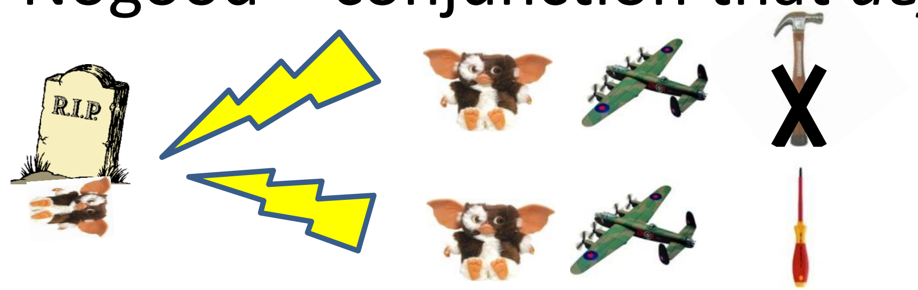
Learns feature combinations whose presence
<u>guarantees</u> a state to be a dead end

Nogoods



Generate-and-Test Procedure

- **Generate a nogood candidate**
  - Key insight: Nogood = conjunction that *defeats* all b.f.s

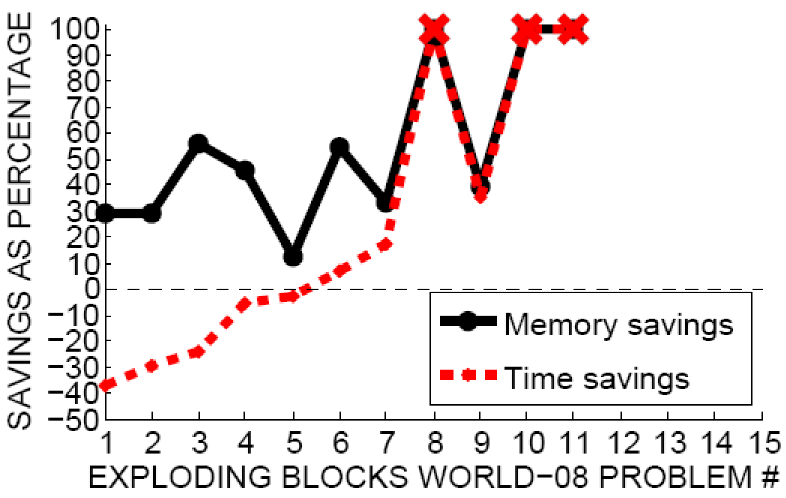


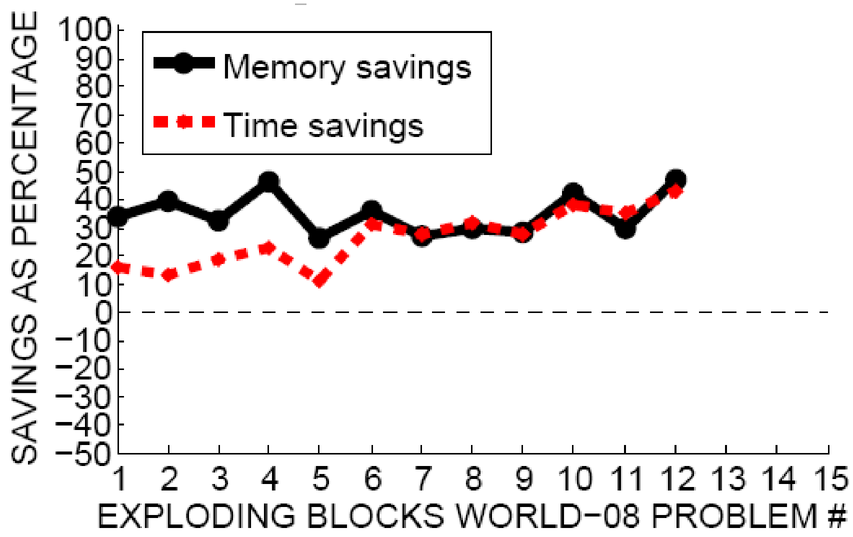
  - For each b.f., pick a literal that defeats it

- **Test the candidate**
  - Needed for soundness, since we don't know all b.f.s
  - Use the non-relaxed Planning Graph algorithm

Benefits of SixthSense

- **Can act as submodule of many planners and ID dead ends**
  - By checking discovered nogoods against every state





–

Take Homes

- Novel ideas to learn structure in the domain
- Basis functions
  - Learn by regressing trajectories
  - Represent good structure
  - Generalize across states
- Nogoods
  - Learn inductively; prove using a sound procedure
  - Represent bad structure
  - Generalize across dead-end states

Take Homes

- A novel use of classical planners for MDP algos
  - retains the decision-theoretic nature of MDPs
  - exploits the scalability of clas

M#2: factored representations expose useful problem structure

- Automatic ways to generate basis functions
  - no longer an onus on human designer
  - exploits factored domain model

Agenda

- Background: Stochastic Shortest Paths MDPs
- Background: Heuristic Search for SSP MDPs
- Algorithms: Automatic Basis Function Discovery
- Models: SSPs → Generalized SSPs

Theme of the Workshop

- Value Functions → Generalized Value Functions
- Gradient →Extra-gradient
- KL divergence → Bergman divergence
- Contextual bandits → Linear bandits
- SSPs → ?

SSP/SSP$_{s0}$

SSP MDP is a tuple <$S, A, T, C, G, (s_0)$>, where:

- S is a finite state space
- A is a finite action set
- T is a stationary transition function
- C is a stationary cost function
- G is a set of absorbing cost-free goal states
- (s_0 is an initial state)

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 $P_G = 1$

Disallows dead ends

Prevents algos from halting if we allowed dead ends, make cost a meaningless criterion

Stochastic Shortest-Path MDPs

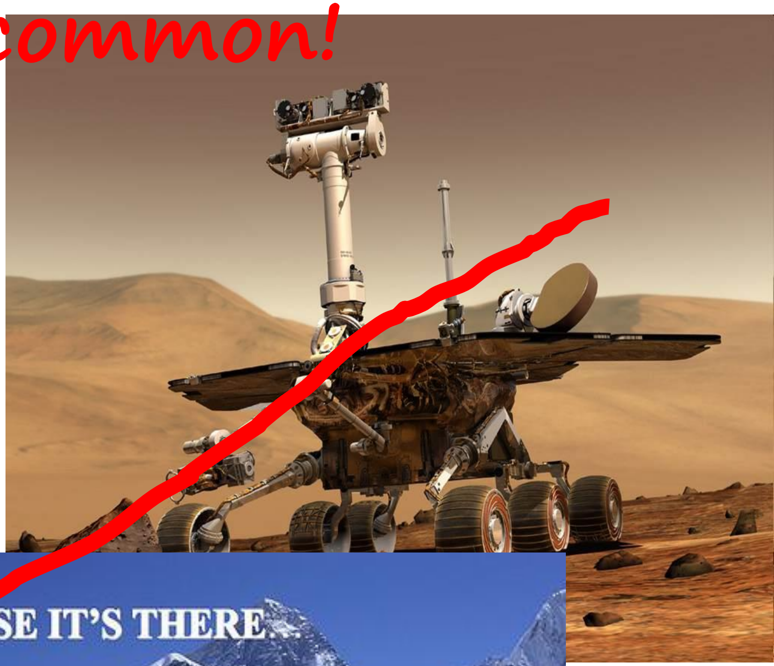
Dead ends are common!

- Example applications:
  - Controlling a Mars rover

    "How to collect scientific data without damaging the rover?"

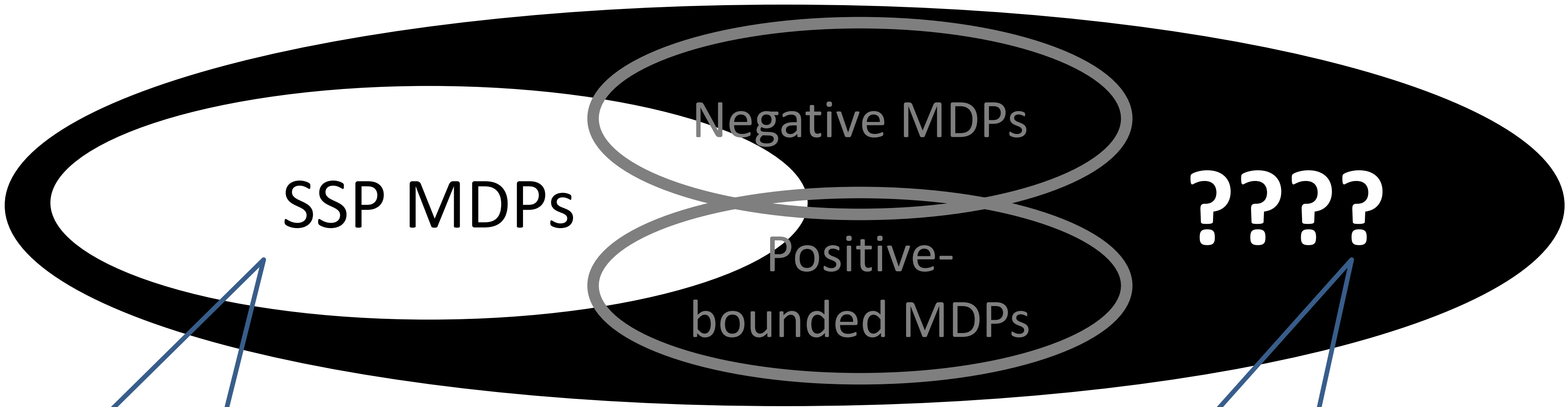
  - Route planning

    "How to climb mount Everest in the cheapest way?"



Discrete MDP Research So Far

Goal-oriented MDPs



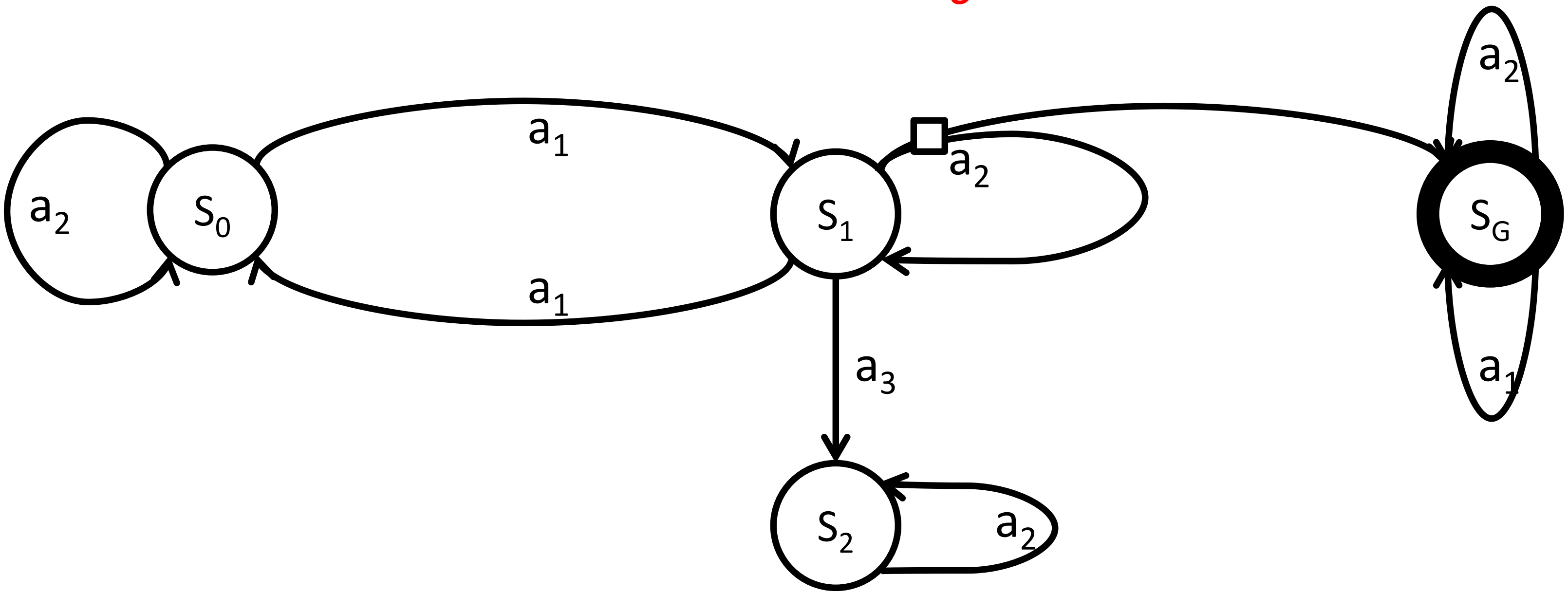
-Model many interesting scenarios
-Efficiently* solvable by heuristic search

-What interesting problems are here?
-How do we solve them efficiently?

SSPADE: Dead Ends are Avoidable from s_0

[Kolobov, Mausam, Weld, UAI'12]

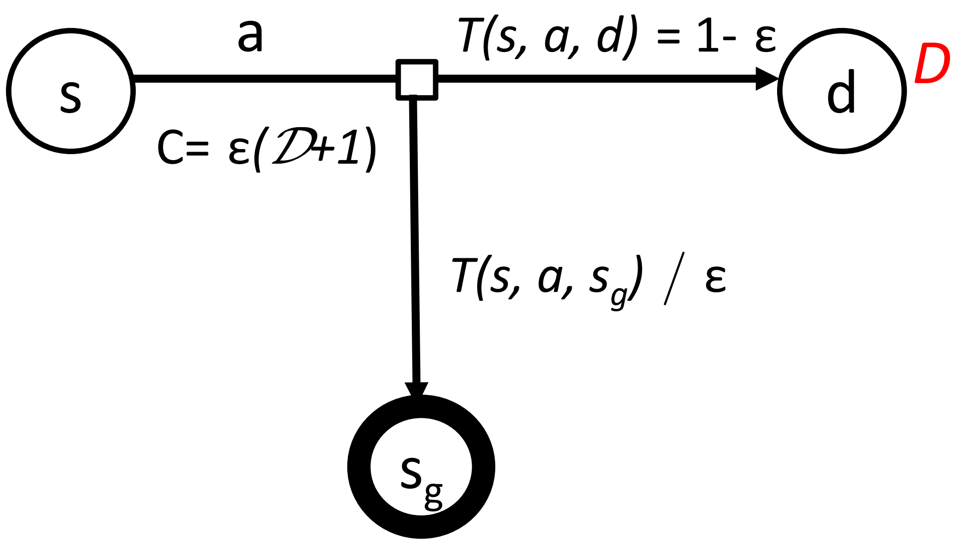
- D.e.s may be avoidable *from* s_0 via an optimal policy



- Can't compute *V*(s)* for every state
- But need only "relevant" states to get the "right" value
- Can be solved with optimal heuristic search from s_0
  - FIND shouldn't starve states; REVISE should halt

fSSPUDE: SSP with Unavoidable Dead Ends (and a Finite Penalty on Them)

- First attempt: if the agent reaches a d.e., it pays *D*



$$V^*(s) = \varepsilon(\mathcal{D}+1) + \varepsilon \cdot 0 + (1-\varepsilon) \cdot \mathcal{D}$$

- Makes non-d.e.s more "expensive" than d.e.s!
  - *Oops…*

fSSPUDE: SSP with Unavoidable Dead Ends (and a Finite Penalty on Them)

[Kolobov, Mausam, Weld, UAI'12]

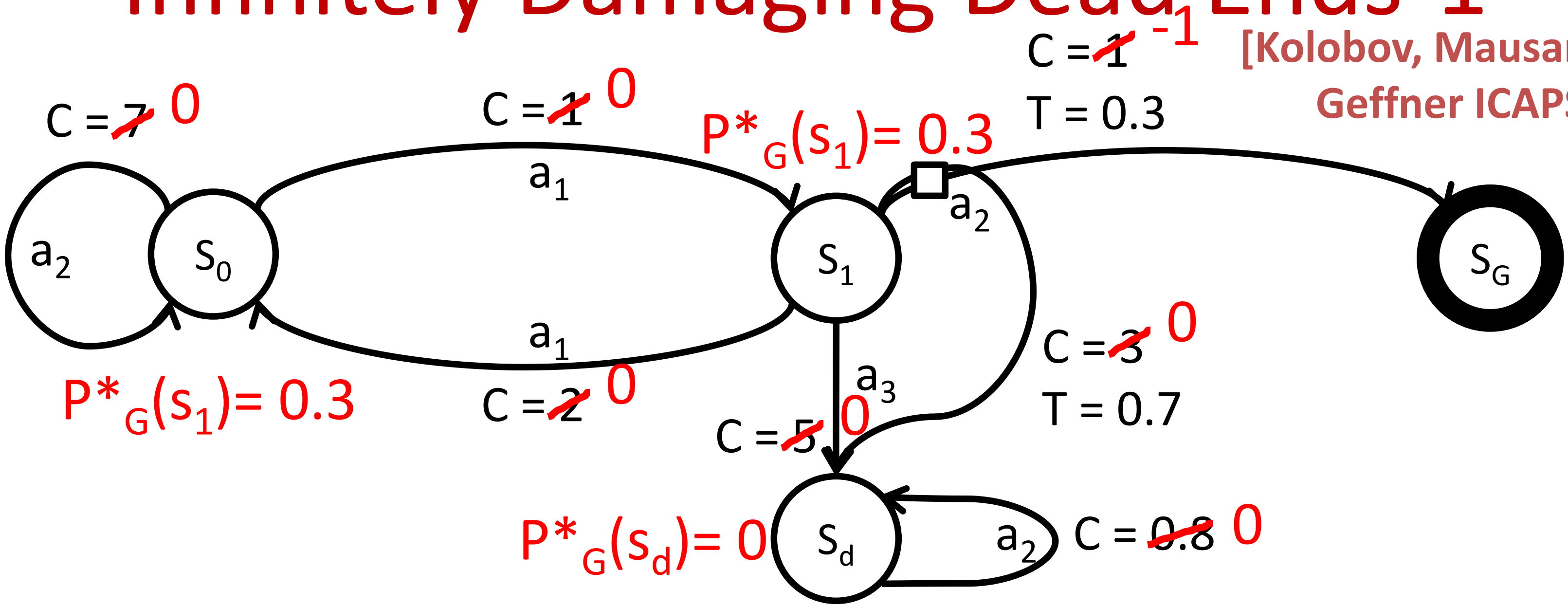
- Second attempt: agent allowed to stop at ***any*** state
  - by paying a price = penalty *D*
  - Intuition: achieving a goal is worth –*D* to the agent

- Equivalent to SSP MDP with a special a_{stop} action
  - applicable in each state
  - leads directly to a goal by paying cost *D*

- Thus, algorithms for SSP apply to fSSPUDE!

MAXPROB: Dealing with Unavoidable Infinitely Damaging Dead Ends-1

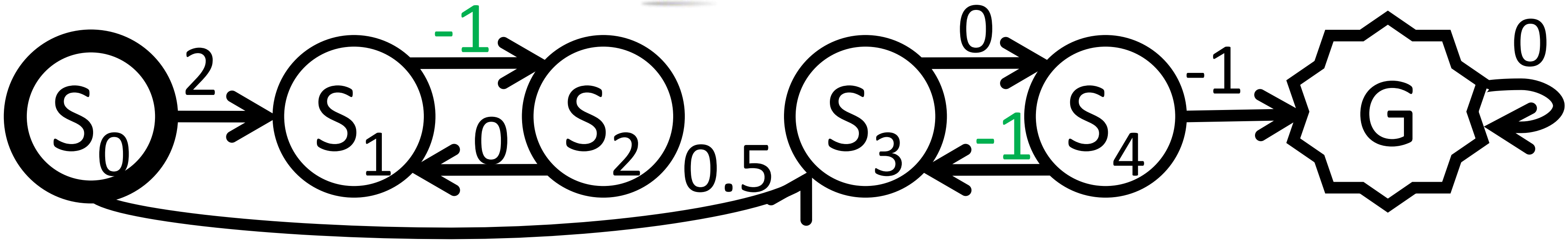
[Kolobov, Mausam, Weld, Geffner ICAPS'11]



- Comparing policies in terms of cost meaningless
- MAXPROB/GSSP MDPs: evaluate policies by probability of reaching goal
  - Set all action costs to 0 (they don't matter), reward 1 for reaching goal
  - Fixed-point methods such as VI or LRTDP don't converge because of traps

MDP Examples

SSP



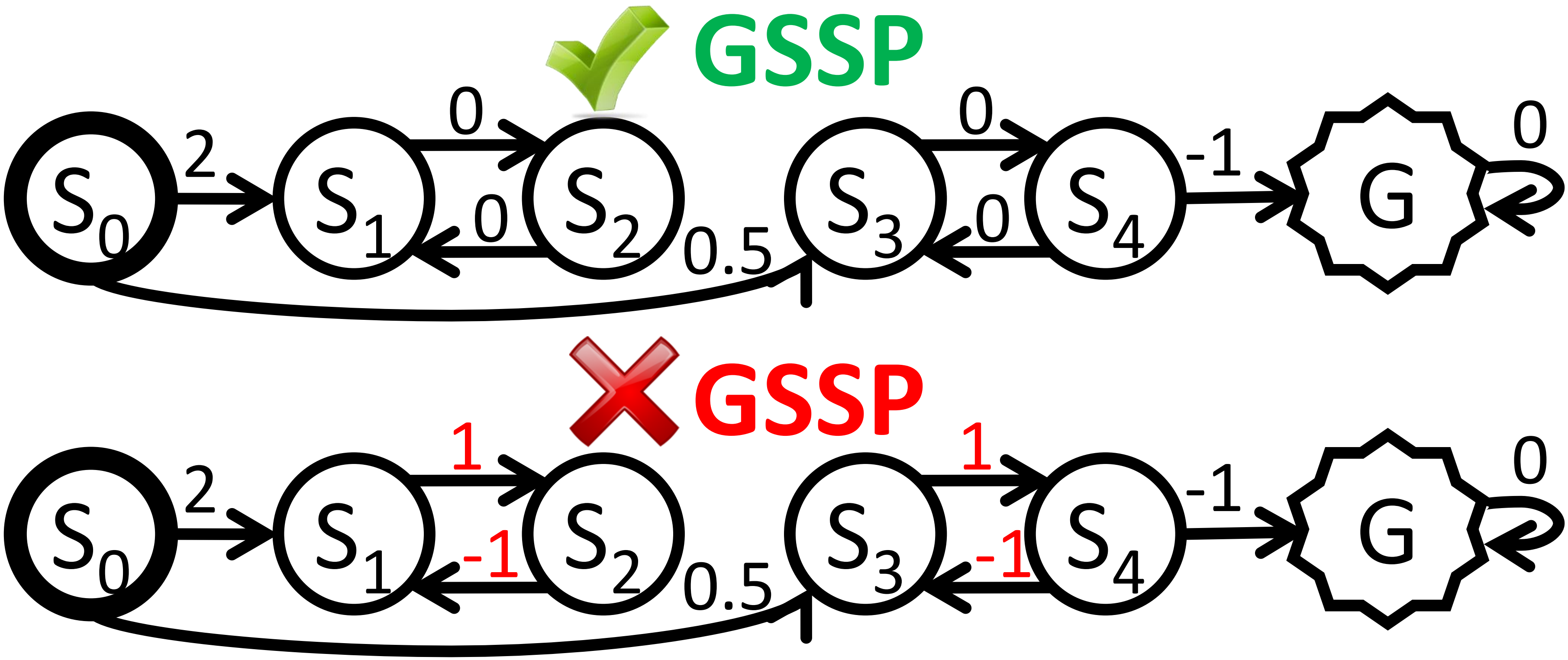
SSP

SSP

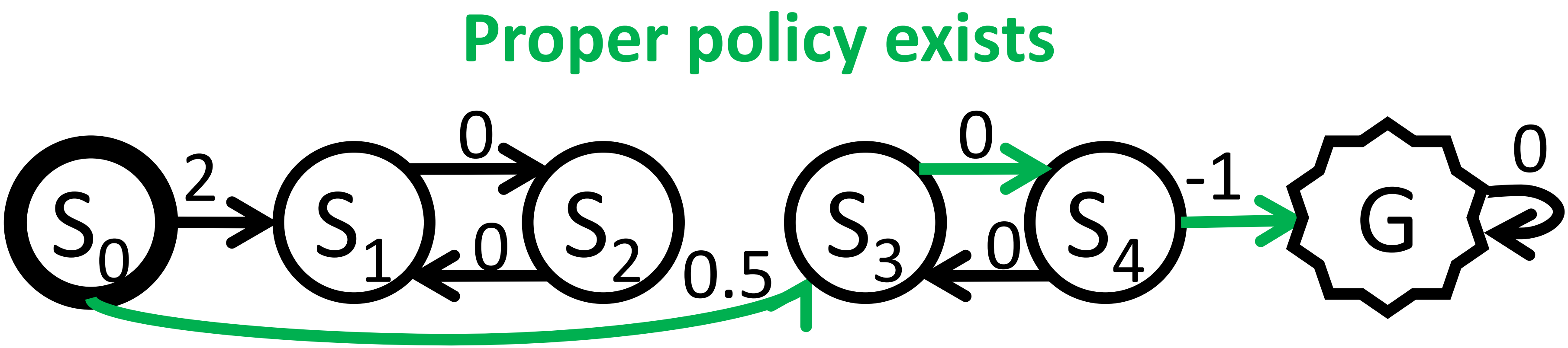
Generalized SSPs: Definition

- An MDP M = <S, A, T, R, G, s_0> for which
  - There is a proper policy (reaches the goal with P=1)
  - Sum of *non-negative* rewards accumulated by any policy starting at s_0 is bounded from above
- Solving a GSSP = finding a reward-maximizing Markovian policy *that reaches the goal*

Generalized SSPs: Example

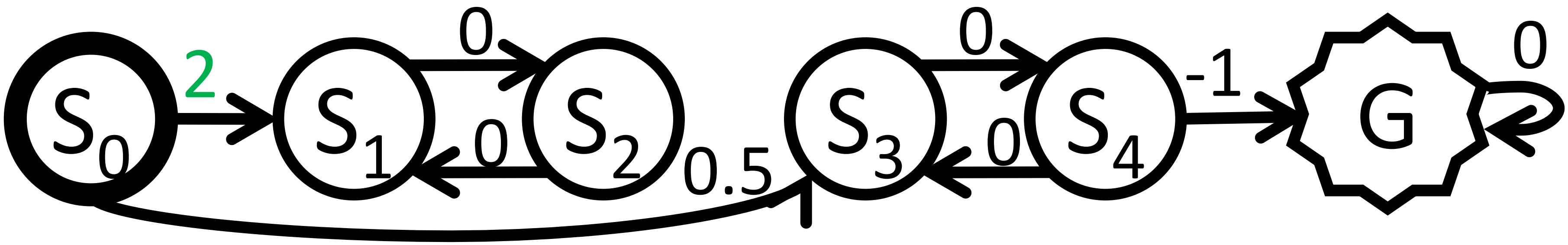


Generalized SSPs: Example



Generalized SSPs: Example

For any Π, sum of non-negative rewards ≤ 2



Generalized SSPs: Example

Solution

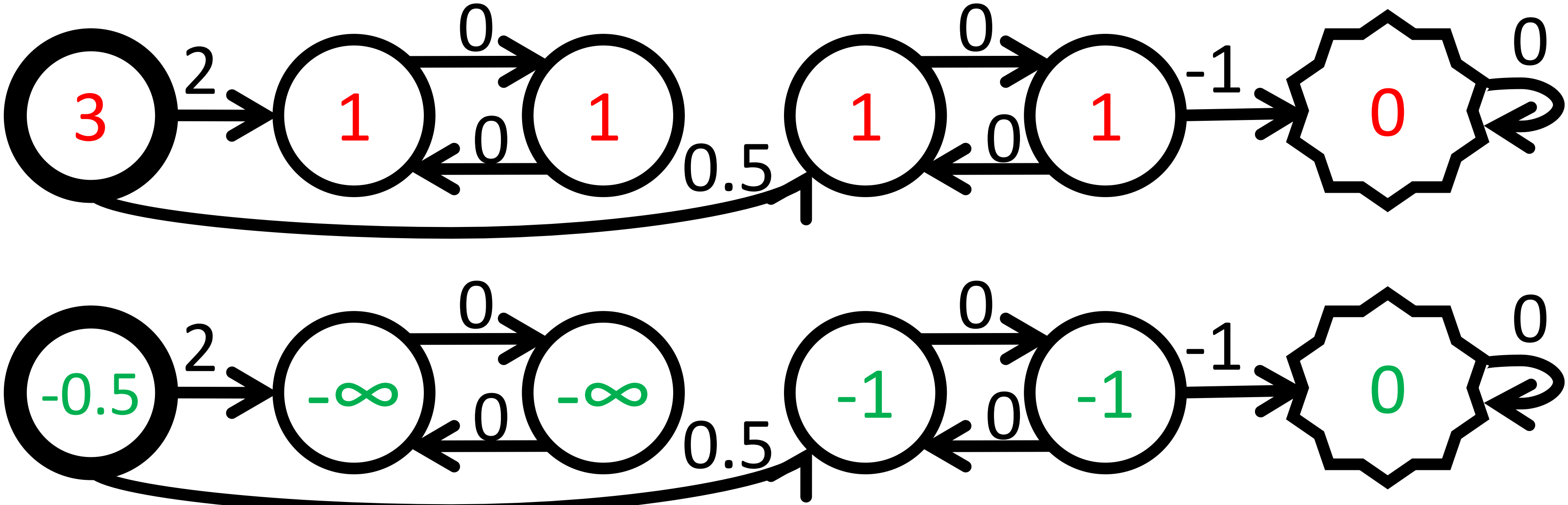
Not a solution

GSSPs: Is V* A Fixed Point of *B*?

- Reminder: in SSPs, V* = *B* V*, where
  - *B* is the *Bellman backup operator*
  - $B\,V(s) = \max_a \{R(s, a) + \sum_{s' \text{ in } succ(s,a)} T(s, a, s')V(s')\}$
- In SSPs, V* is a fixed point of *B*
  - **Still true in GSSPs:**

GSSPs: Is V* *The Unique* Fixed Point of *B*?

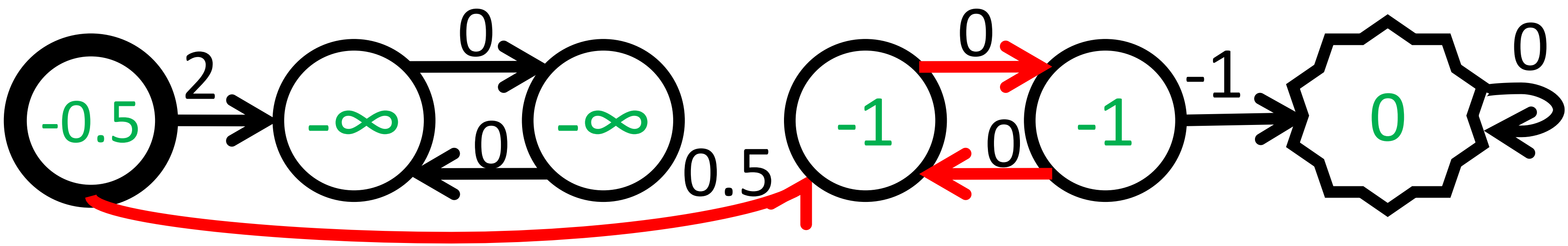
- In SSPs, V* is the unique fixed point of *B*
  - I.e., $V^* = B \circ B \circ \ldots B\, V_0$, V_0 is a *heuristic value function*
  - **Not true in GSSPs:**



  - **Moreover, all suboptimal fixed points are admissible!**

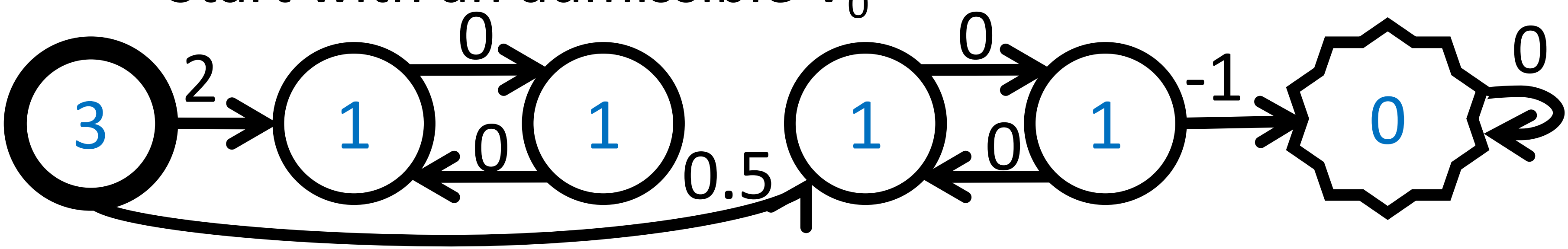
GSSPs: Is Every V*-greedy ∏ A Solution?

- In SSPs, every ∏ greedy w.r.t V* reaches the goal
  - **Not true in GSSPs:**

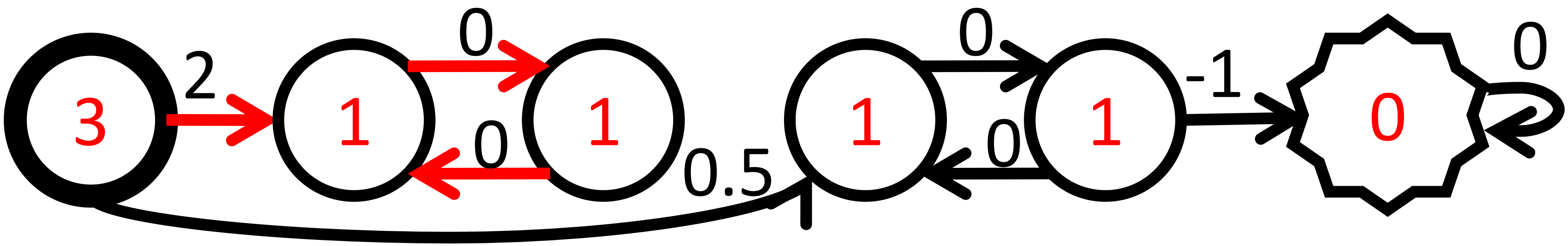


Efficiently Solving GSSPs: Attempt #1

- **Just Run F&R!**
  - Start with an admissible V_0



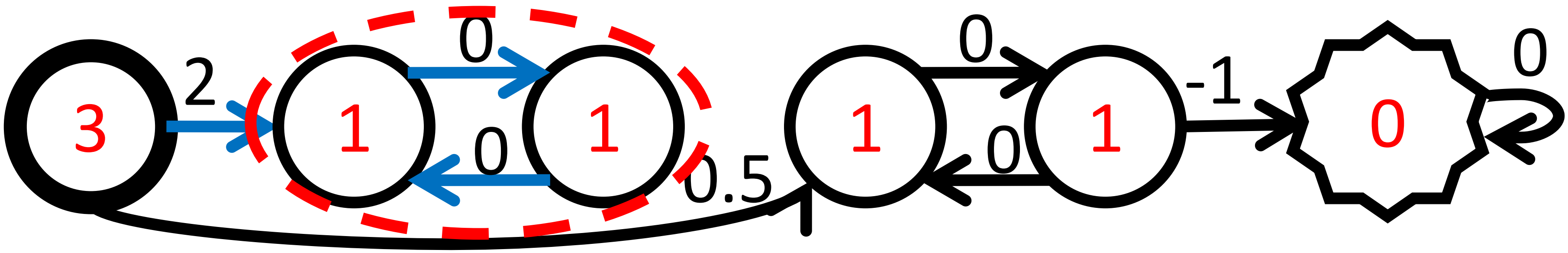
  - Done!



Attempt #1: What Went Wrong?

- In GSSPs, suboptimal fixed points are admissible!
  – When starting with $V_0 \geq V^*$, F&R hit one of them.

  – *B* can't change V over ***traps*** – strongly connected components in V's greedy graph



- Can yield an arbitrarily poor solution

Efficiently Solving GSSPs: **FRET**

- **F**ind, **R**evise, **E**liminate **T**raps
  - First heuristic search algorithm for MDPs beyond SSP
  - Provably optimal if the heuristic is admissible

- Main idea
  - Run F&R until convergence
  - Eliminate traps in the policy envelope
  - Repeat until no more traps

FRET Example: Finding V*

R
e
p
e
a
t

Start with an admissible V_0

Run **F&R** until convergence

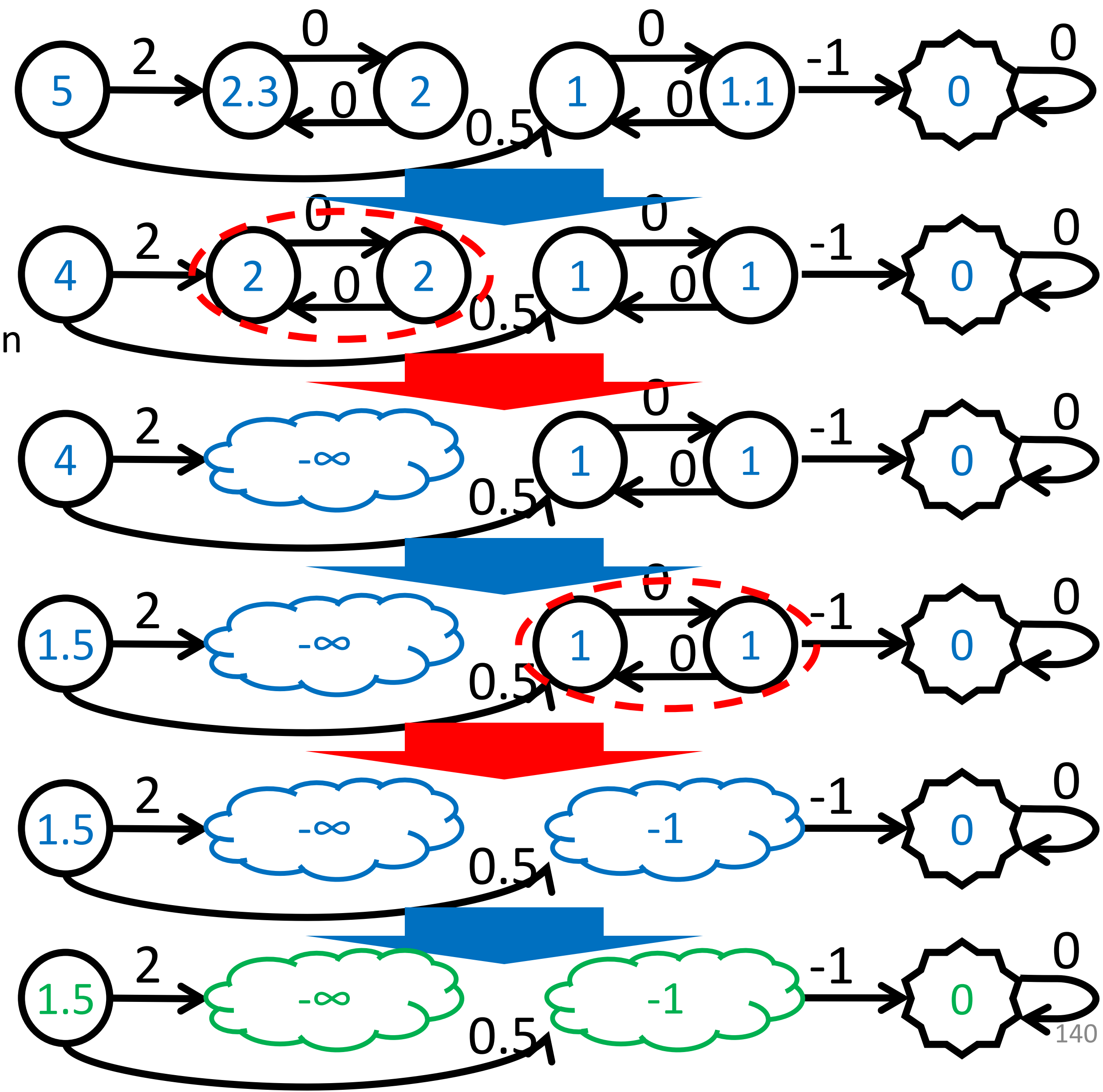
Eliminate Traps in the resulting V_i

Find-and-Revise

Eliminate Traps

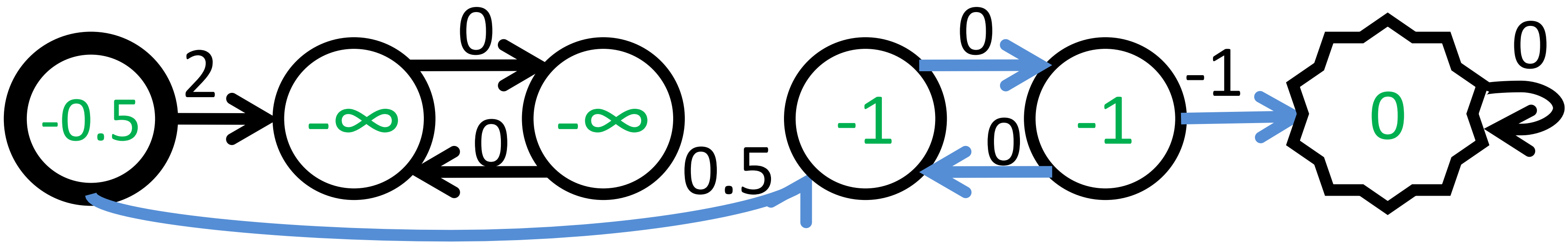
Find-and-Revise

No traps left – done!



FRET Example: Extracting ∏*

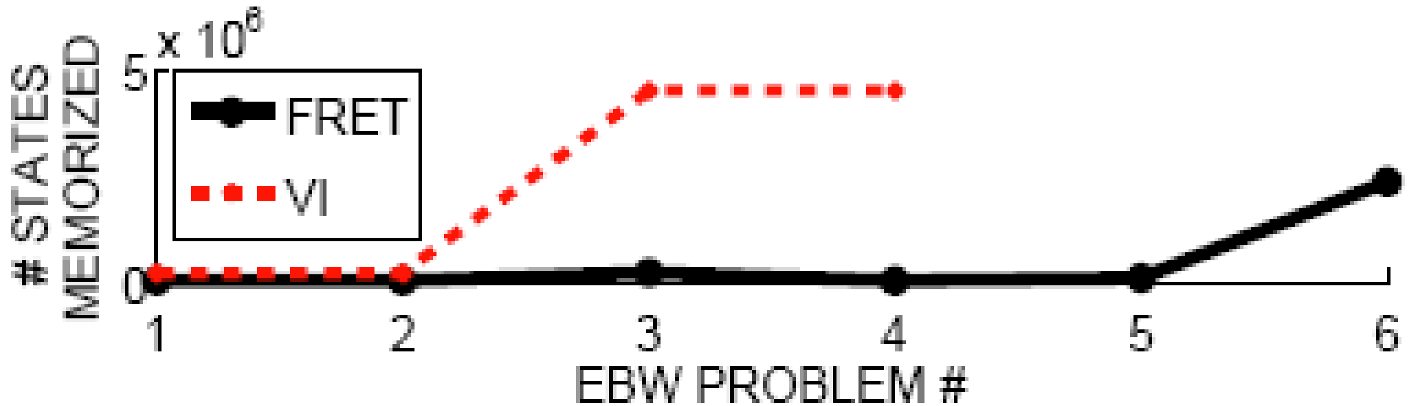
- Iteratively "connect" states to the goals
  - Using optimal actions
  - Until s_0 is connected

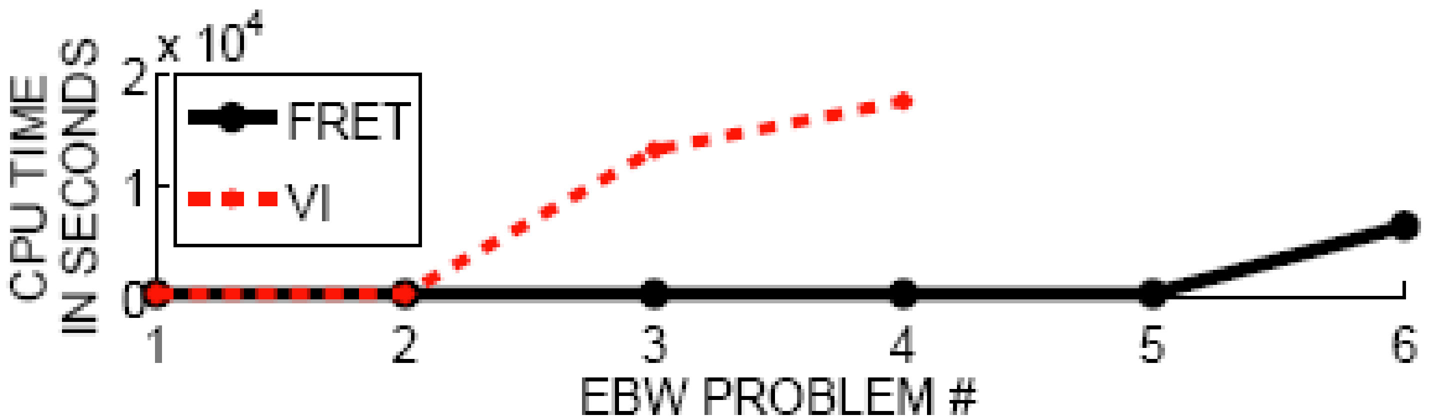


Experimental Setup

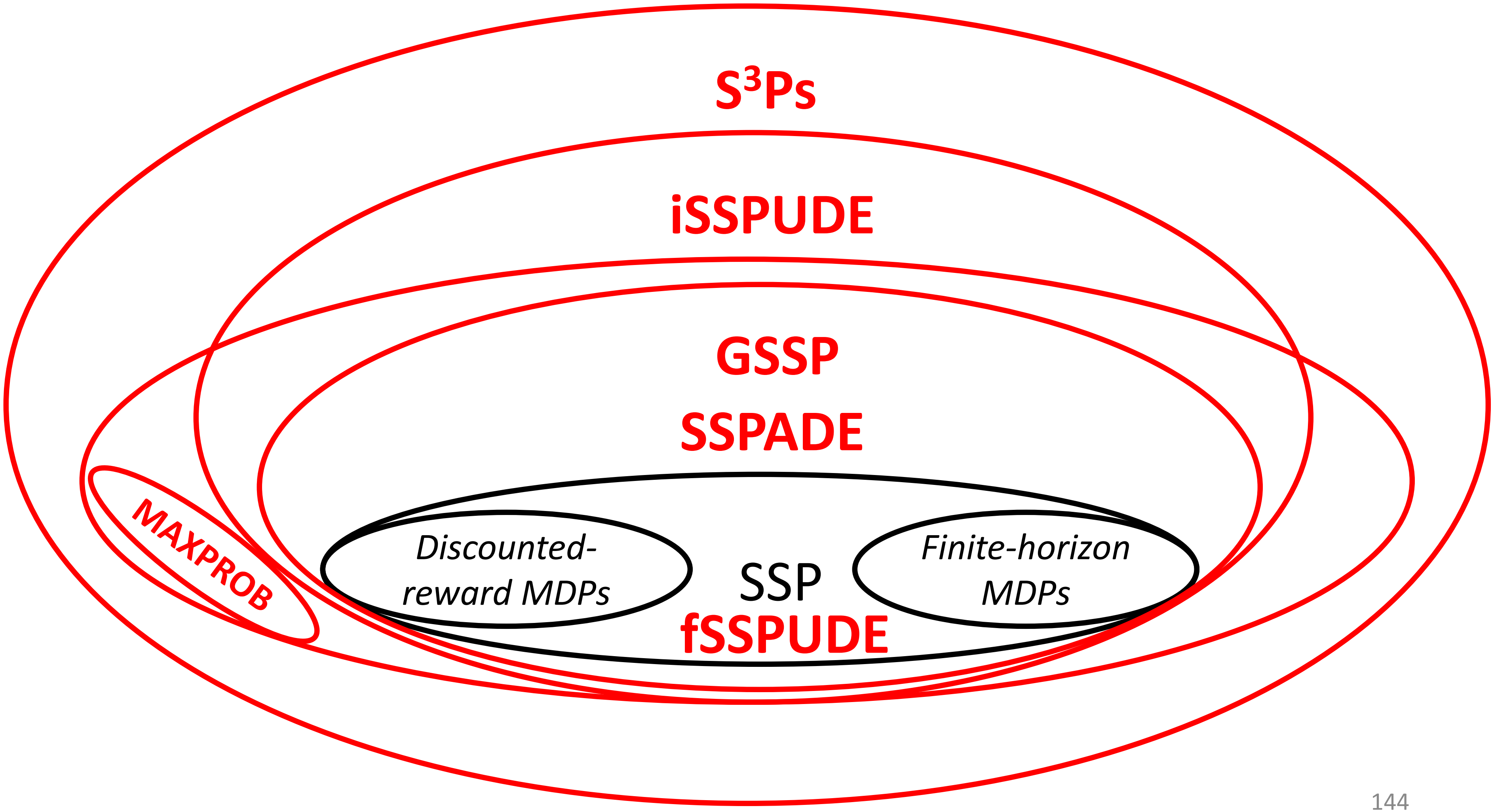
- Problems: MAXPROB versions of EBW
- Planners: VI vs FRET
- Heuristics: Zero for VI, One+SixthSense for FRET
  - SixthSense soundly identifies some of the “dead ends”; their values are set to 0

Experimental Setup



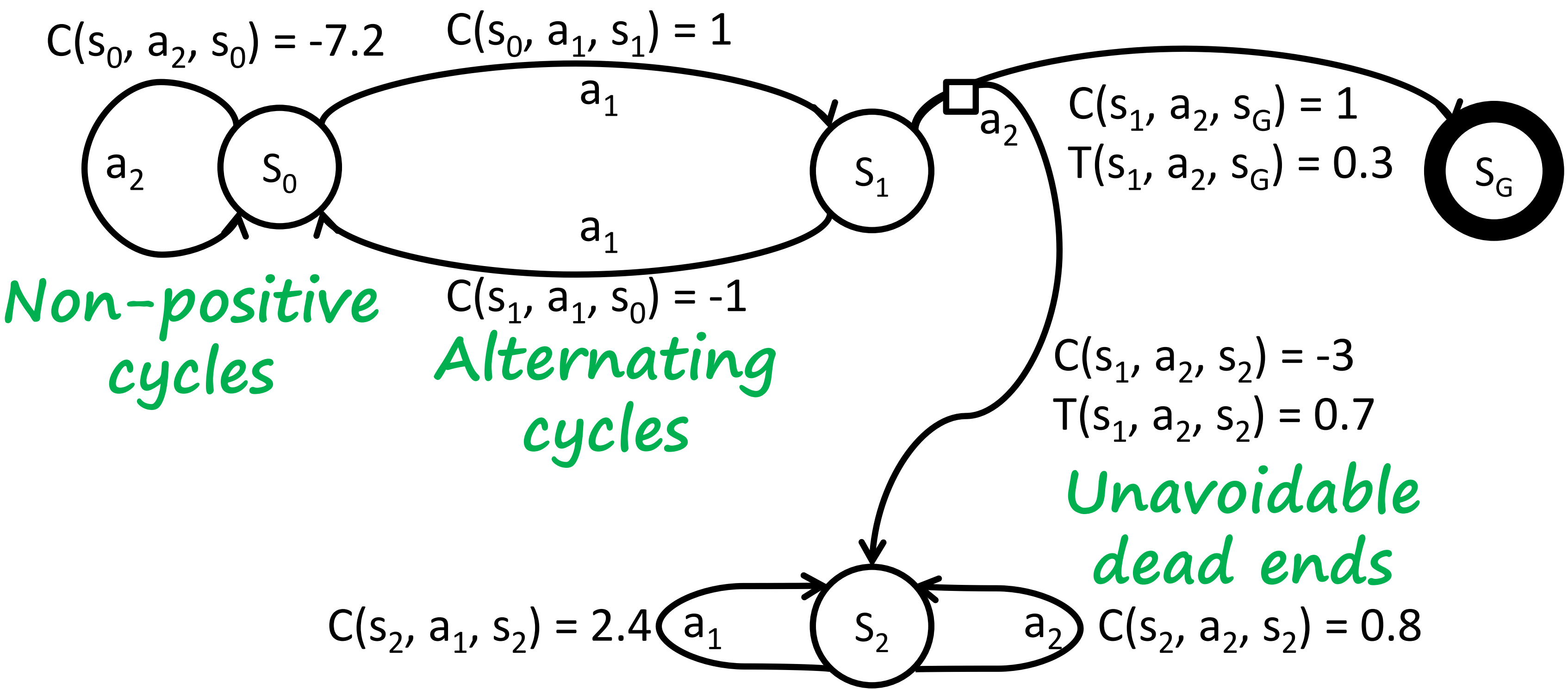


Goal-Oriented MDP Hierarchy



Future Work: Solving S³P

- Stochastic Safest and Shortest Path (S³P) MDPs
  - Teichteil-Koenigsbuch, AAAI'12
  - Goal-oriented MDPs with no restriction on costs



Take Homes

- SSP MDPs exclude interesting planning scenarios

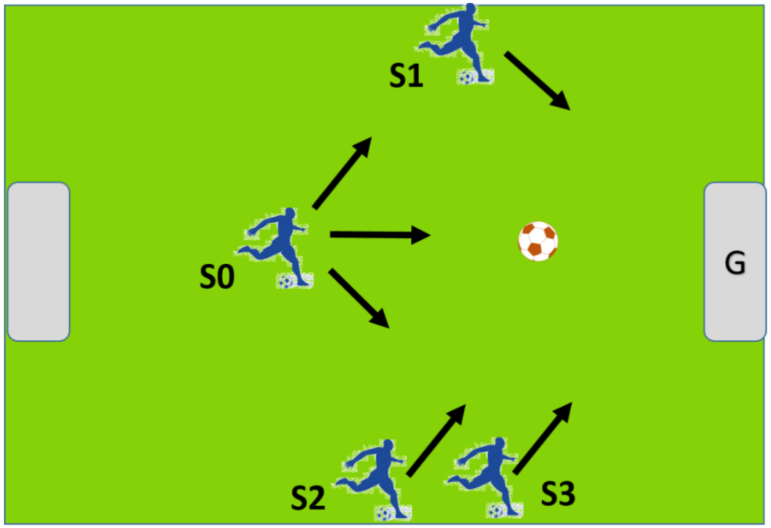
- Generalized SSPs
  - handle zero-cost cycles
  - GSSP contains SSP and several ot
  - heuristic search algorithm (FRET)

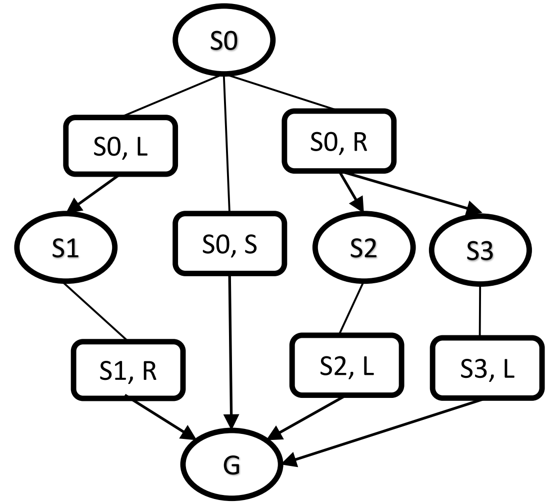
M#3: some models use explicit knowledge of goals

- Dead-ends tricky in undiscounted goal MDPs
- Well-formed extensions of SSP MDPs
  - can have unintuitive DP properties
  - what is beyond GSSPs?
  - loads of open questions: theoretical & algorithmic

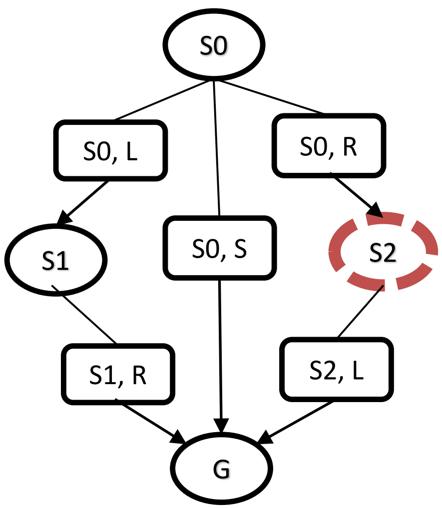
Agenda

- Background: Stochastic Shortest Paths MDPs
- Background: Heuristic Search for SSP MDPs
- Algorithms: Automatic Basis Function Discovery
- Models: SSPs $\rightarrow$ Generalized SSPs

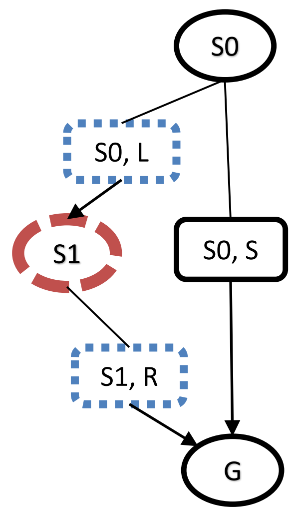




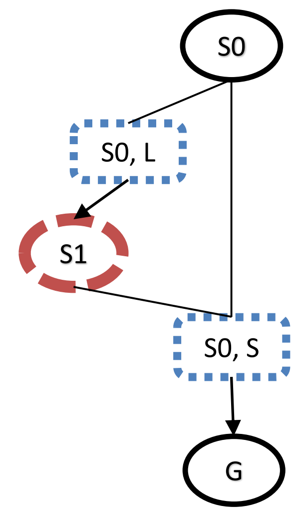
AND-OR Graph in Flat Space



AS Graph[1]



ASAM Graph[2]



ASAP Graph

[1]: Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. Artificial Intelligence, 2003
[2]: Balaraman Ravindran and A Barto. Approximate homomorphisms: A framework for nonexact minimization in Markov decision processes. In ICKBCS, 2004.

Key Properties

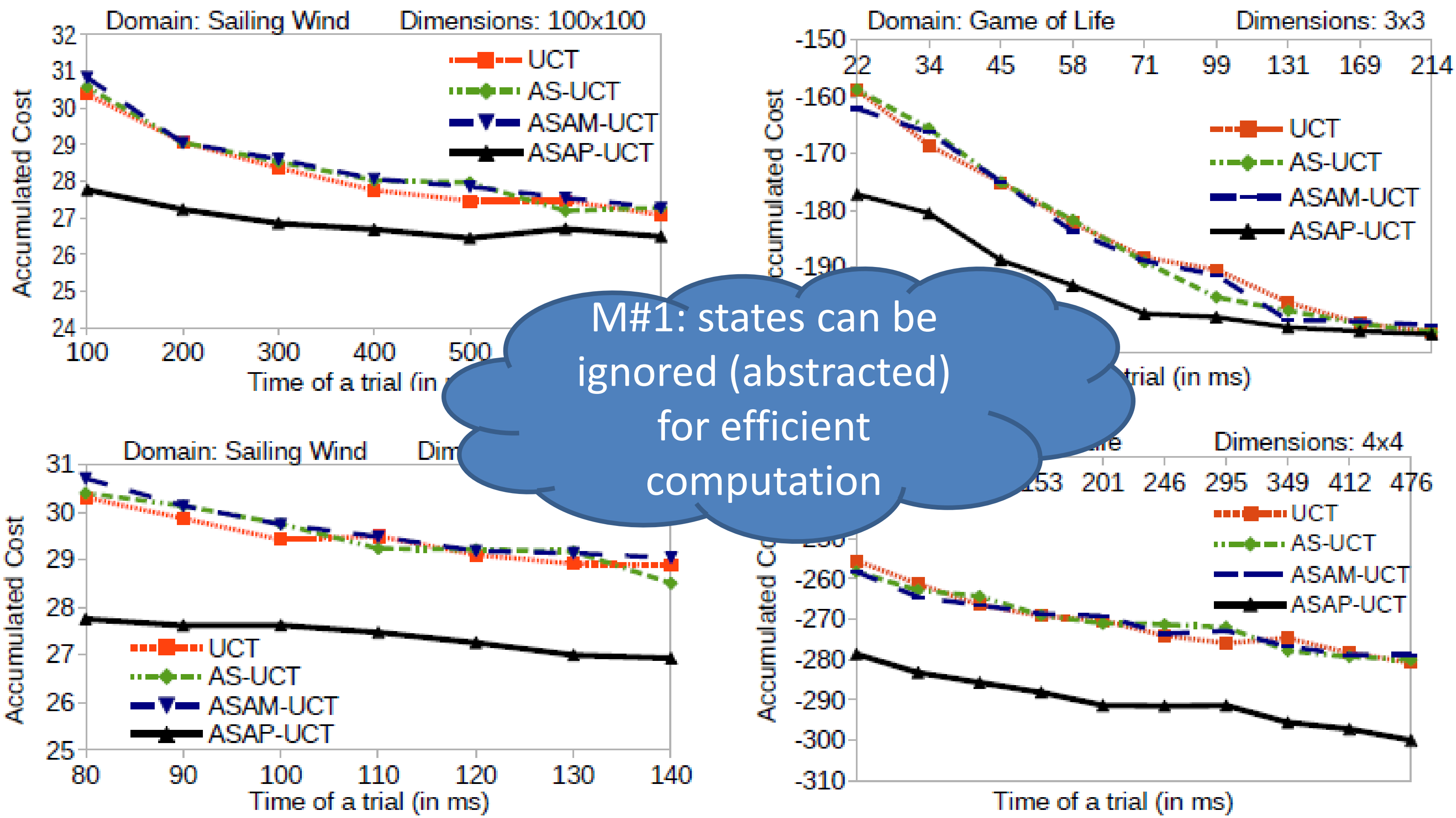
PROPERTY 1: The original MDP does not reduce to an abstract MDP

PROPERTY 2: ASAP subsumes abstractions computed by AS and ASAM

PROPERTY 3: Value Iteration on abstract AND-OR graph returns optimal value functions for the original MDP

Experiments

[Anand, Grover, Mausam, Singla – submitted]



M#1: states can be ignored (abstracted) for efficient computation

3 Key Messages

- M#0: No need for exploration-exploitation tradeoff
  - planning is purely a computational problem (V.I. ~~vs. Q~~)
- M#1: Search in planning
  - states can be ignored or reordered for efficient computation
- M#2: Representation in planning
  - develop interesting representations for Factored MDPs
    → Exploit structure to design domain-independent algorithms
- M#3: Goal-directed MDPs
  - design algorithms/models that use explicit knowledge of goals