Probabilistic Planning with Markov Decision Processes

Andrey Kolobov and Mausam

Computer Science and Engineering

University of Washington, Seattle

Goal

an extensive introduction to theory and algorithms in probabilistic planning

Outline of the Tutorial

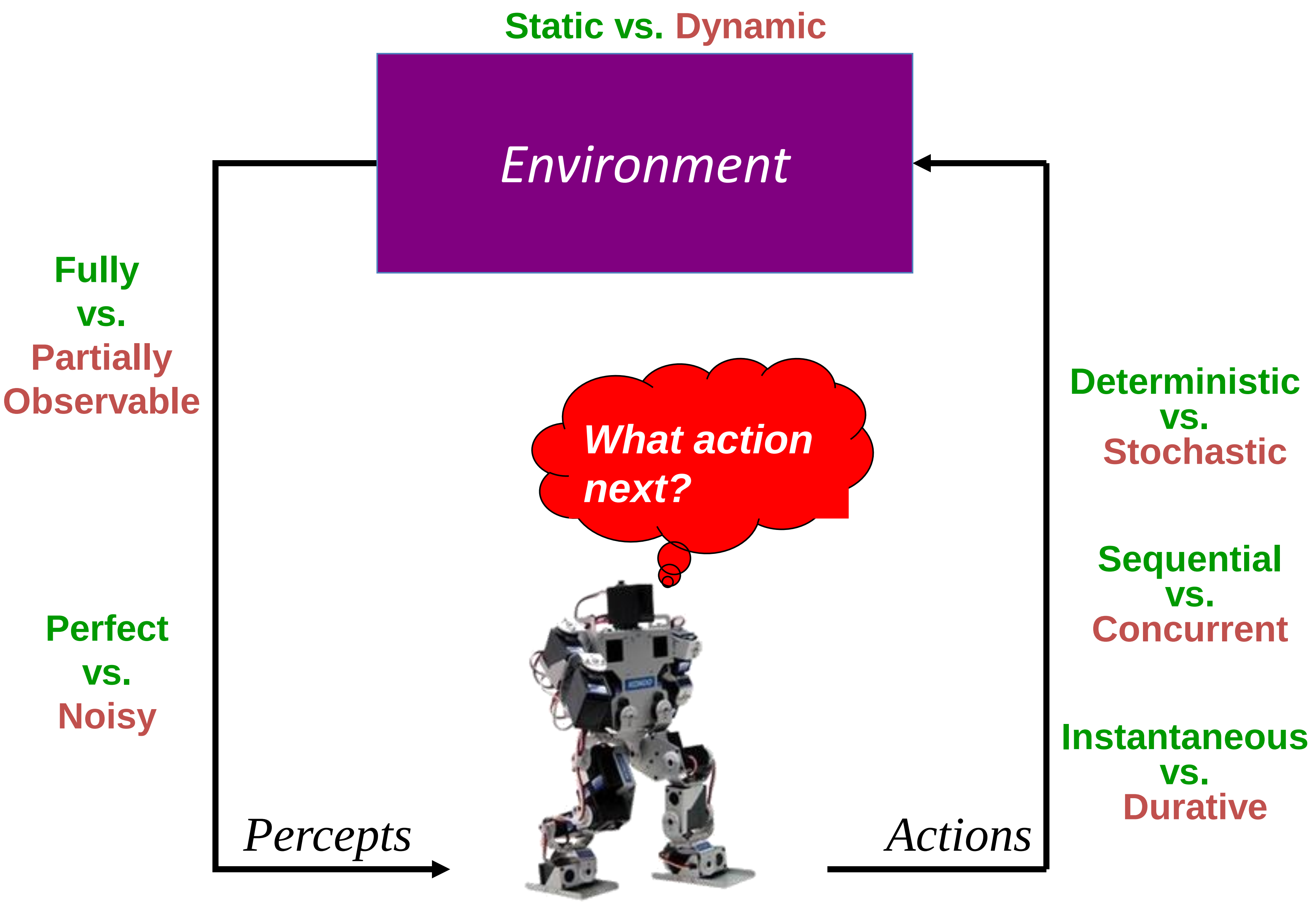
- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

Outline of the Tutorial

- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

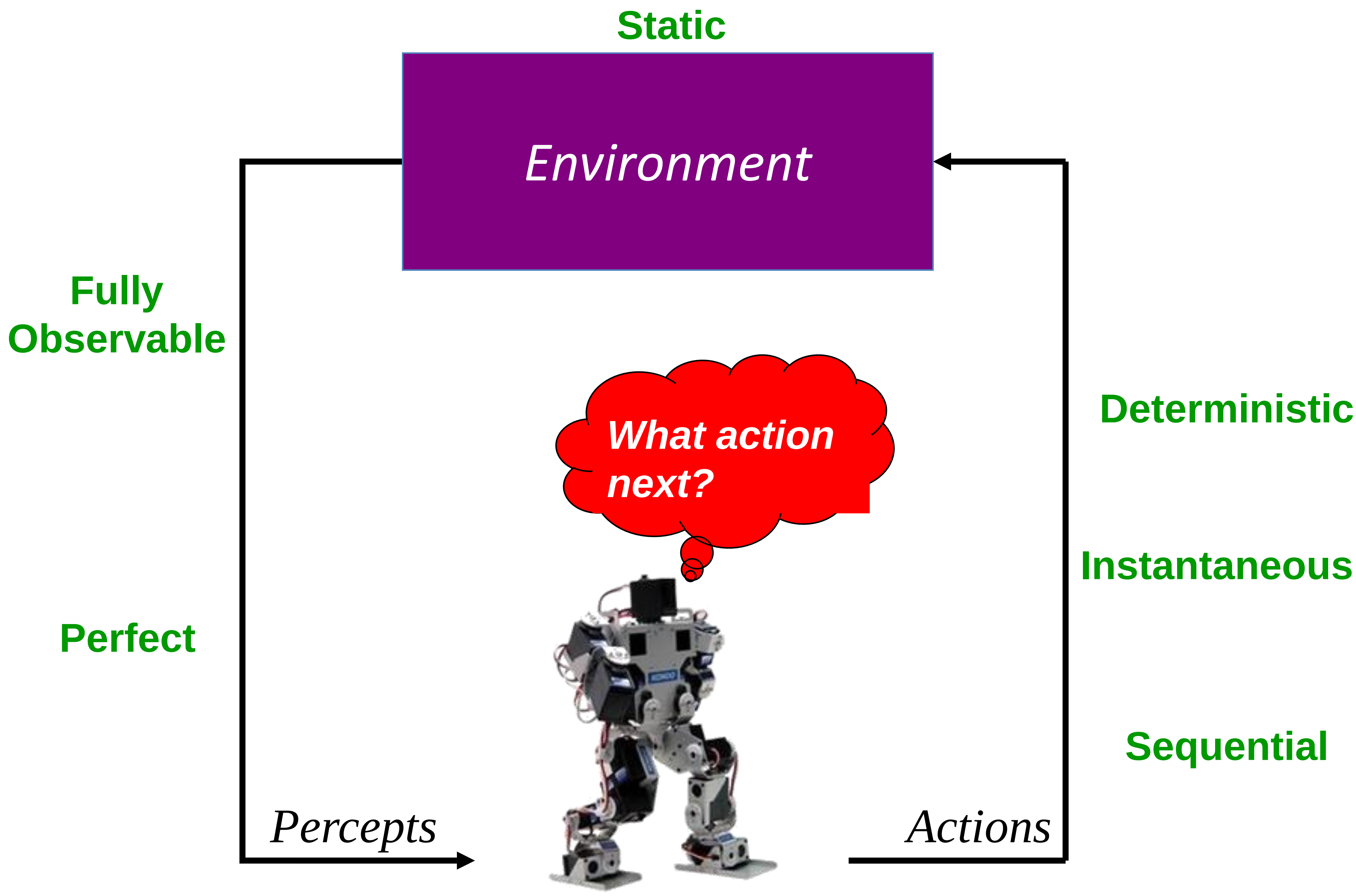
INTRODUCTION

Planning

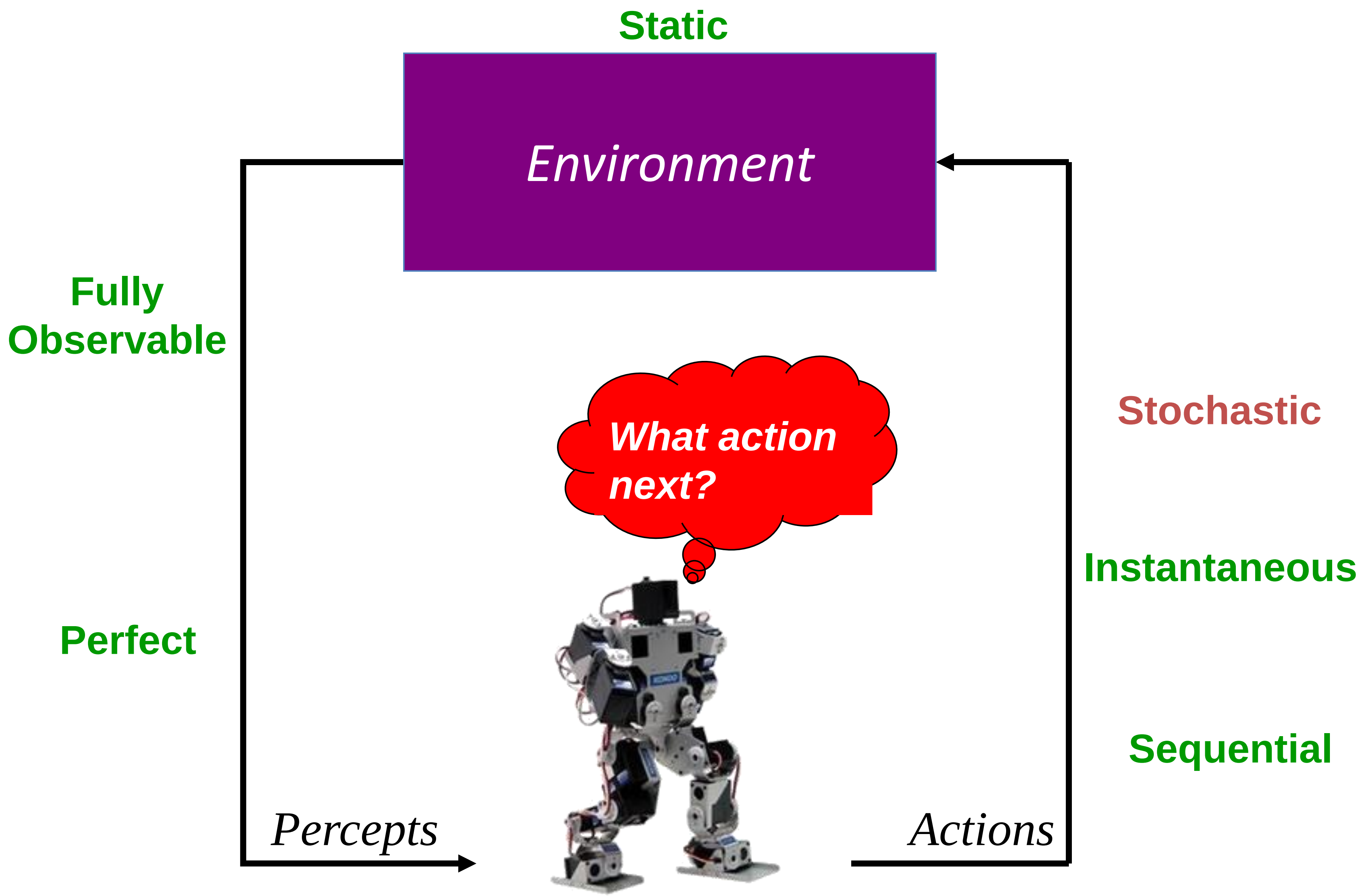


Static vs. Dynamic
Environment
Fully vs. Partially Observable
Perfect vs. Noisy
What action next?
Deterministic vs. Stochastic
Sequential vs. Concurrent
Instantaneous vs. Durative
Percepts
Actions

Classical Planning



Probabilistic Planning



Markov Decision Processes

- A fundamental framework for prob. planning

- History
  - 1950s: early works of Bellman and Howard
  - 50s-80s: theory, basic set of algorithms, applications
  - 90s: MDPs in AI literature

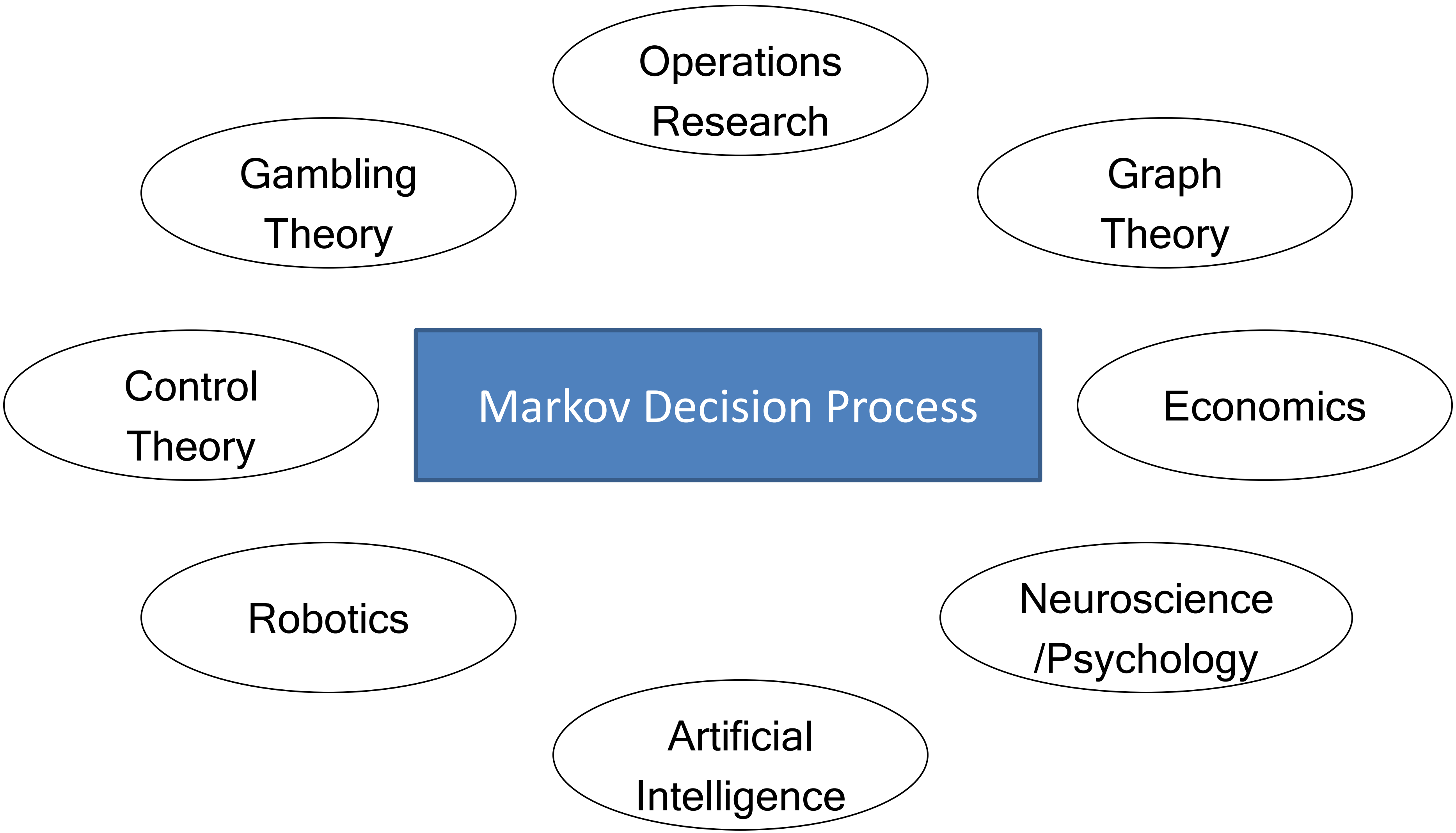
- MDPs in AI
  - reinforcement learning
  - probabilistic planning ← we focus on this

What are MDPs good for?

- Uncertain Domain Dynamics
- Sequential Decision Making
- Cyclic Domain Structures
- Full Observability and Perfect Sensors
- Fair Nature

- Rational Decision Making

An MDP-Centric View



Shameless Plug

Mausam and Andrey Kolobov

"Planning with Markov Decision Processes: An AI Perspective"

Morgan and Claypool Publishers (Synthesis Lectures Series on Artificial Intelligence)

Outline of the Tutorial

- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

FUNDAMENTALS OF
MARKOV DECISION PROCESSES

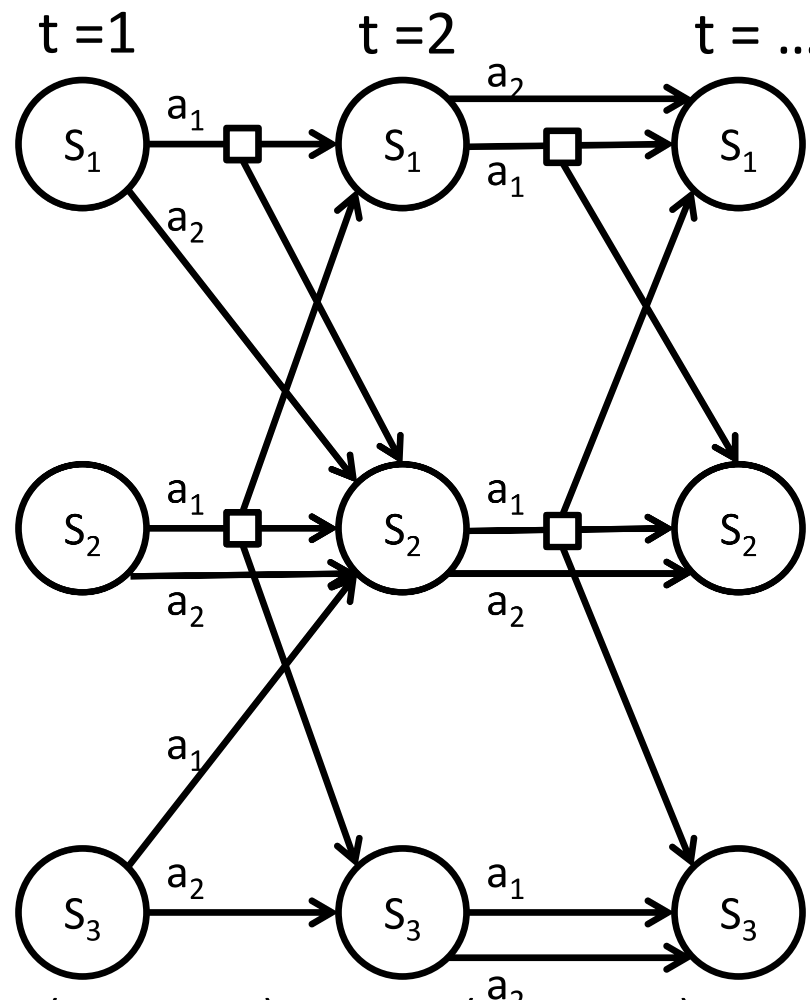
3 Questions

- What is an MDP?
- What is *an* MDP solution?
- What it an *optimal* MDP solution?

MDP: A Broad Definition

MDP is a tuple <*S*, *D*, *A*, *T*, *R*>, where:

- *S* is a finite state space
- *D* is a sequence of discrete time steps/decision epochs (1,2,3, … , *L*), *L* may be ∞
- *A* is a finite action set
- *T*: *S* x *A* x *S* x *D* → [0, 1] is a transition function
- *R*: *S* x *A* x *S* x *D* → $\mathbb{R}$ is a reward function



$T(s_3, a_1, s_2, 1) = 1.0$ $T(s_3, a_1, s_3, 2) = 1.0$

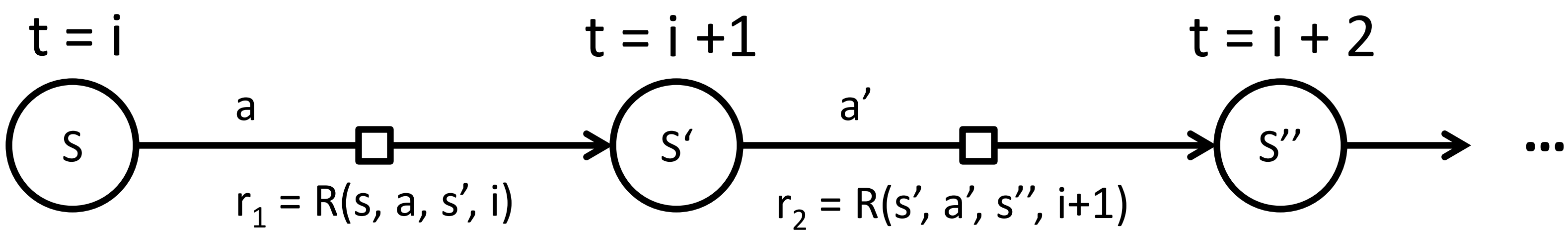
$R(s_3, a_1, s_2, 1) = -7.2$ $R(s_3, a_1, s_3, 2) = 3.5$

MDP Solution: A Broad Definition

- Want a way to choose an action in a state, i.e., a *policy* π
- What does a policy look like?
  - Can pick action based on states visited + actions used so far, i.e., *execution history* $h = s(1)\ a(1)\ s(2)\ a(2) \ldots s$
  - Can pick actions randomly
- **Thus, in general an MDP solution is a *probabilistic history-dependent* π: $H \times A \rightarrow [0,1]$**

Evaluating MDP Solutions

- Executing a policy yields a sequence of rewards



- Let R_1, R_2, … be a sequence of random vars for rewards due to executing a policy

Evaluating MDP Solutions

- Define *utility function* $u(R_1, R_2, \ldots)$ to be some "quality measure" of a reward sequence
  - Need to be careful with definition, more on this later

- Define *value function* as $V: H \rightarrow [-\infty, \infty]$

- Define *value function of a policy after history h* to be some utility function of subsequent rewards:

$$V^{\pi}(h) = u_h^{\pi}(R_1, R_2, \ldots)$$

Optimal MDP Solution: A Broad Definition

- **Want**: a behavior that is "best" in every situation.

- π^* is an *optimal policy* if $V^*(h) \geq V^\pi(h)$ for all π, for all h

- Intuitively, a policy is optimal if its utility vector dominates
  - π^* not necessarily unique!

3 Questions Revisited

- What is an MDP?
  - **M = <*S, D, A, T, R*>**
- What is *an* MDP solution?
  - **Policy π: $H \times A \rightarrow [0,1]$, a mapping from histories to distributions over actions**

DONE!

- What it an *optimal* MDP solution?
  - **π^* s.t. $V^*(h) \geq V^{\pi}(h)$ for all π and h, where $V^{\pi}(h)$ is some utility of rewards obtained after executing history h**

Anything Wrong w/ These Definitions?

Optimality criterion is underspecified, optimal policy may not exist!

- What is an MDP?
  - **M = <*S, D, A, T, R*>**
- What is *an* MDP solution?
  - **Policy π: *H x A* → [0,1], a mapping from histories to distributions over actions.**
- What it an *optimal* MDP solution?
  - **π^* s.t. $V^*(h) \geq V^{\pi}(h)$ for all π and *h*, where $V^{\pi}(h)$ is some utility of rewards obtained after executing history h**

Fundamentals of MDPs

✓ General MDP Definition
- **Expected Linear Additive Utility**
- The Optimality Principle
- Finite-Horizon MDPs
- Infinite-Horizon Discounted-Reward MDPs
- Stochastic Shortest-Path MDPs
- A Hierarchy of MDP Classes
- Factored MDPs
- Computational Complexity

Dealing with Optimal Solution Existence

- **Need to be careful when defining utility $u(R_1, R_2, \ldots)$,**
  - E.g., $u(R_1, R_2, \ldots) = R_1 + R_2 + \ldots$ for the same h can be different across policy executions (i.e., not a well-defined function)

- **Even for a well-defined $u(R_1, R_2, \ldots)$, a policy π^* s.t. $V^*(h) \geq V^\pi(h)$ for all π and h may not exist!**

Expected Linear Additive Utility

- Let's use *expected linear additive utility (ELAU)*

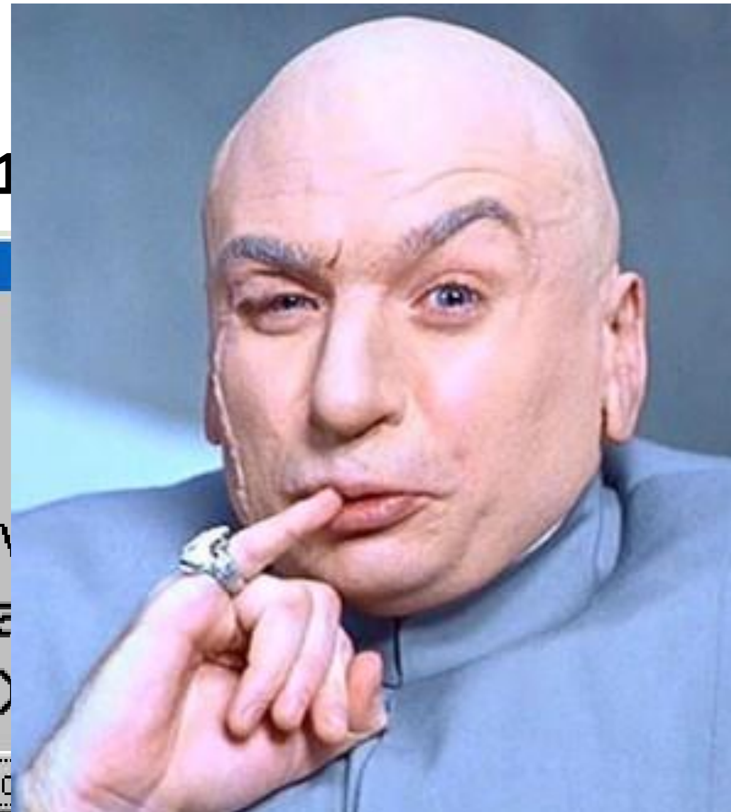
$$u(R_1, R_2, \ldots) = \mathbb{E}[R_1 + \gamma R_2 + \gamma^2 R_3 \ldots]$$

where γ is the *discount factor*

- **Assume γ = 1 unless stated otherwise**
  - $0 \leq \gamma < 1$: agent prefers more immediate rewards
  - $\gamma > 1$: agent prefers more distant rewards
  - $\gamma = 1$: rewards are equally valuable, independently of time

Is ELAU What We Want?

Policy 1 **Policy 2**



- If it lands heads, you get **$2M**
- If it lands tails, you get nothin'

Is ELAU What We Want?

- **ELAU: *"the utility of a policy is as good as the amount of reward the policy is expected to bring"***
  - Agents using ELAU are "rational"

- Assumes the agent is *risk-neutral*
  - Indifferent to policies with equal reward expectation
  - E.g., disregards policies' *variance* (in the previous example, policy 1 has lower variance)

- Not always the exact criterion we want, but...
  - "Good enough"
  - Convenient to work with
  - Guarantees the *Optimality Principle*

Fundamentals of MDPs

- ✓ General MDP Definition
- ✓ Expected Linear Additive Utility
- **The Optimality Principle**
- Finite-Horizon MDPs
- Infinite-Horizon Discounted-Reward MDPs
- Stochastic Shortest-Path MDPs
- A Hierarchy of MDP Classes
- Factored MDPs
- Computational Complexity

The Optimality Principle

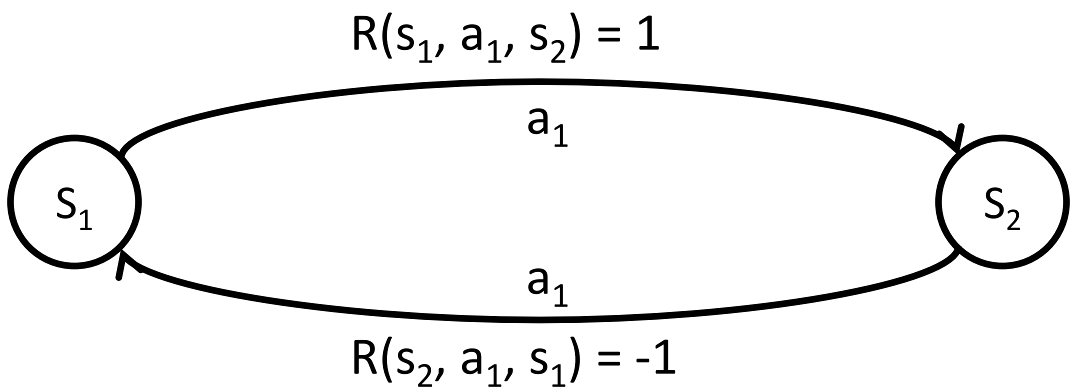
Guarantees that an optimal policy exists **when** ELAU is well-defined!

If the quality of every policy can be measured by its expected linear additive utility, there is a policy that is optimal at every time step.

(Stated in various forms by Bellman, Denardo, and others)

The Optimality Principle: Caveat #1

- When can policy quality *not* be measured by ELAU?

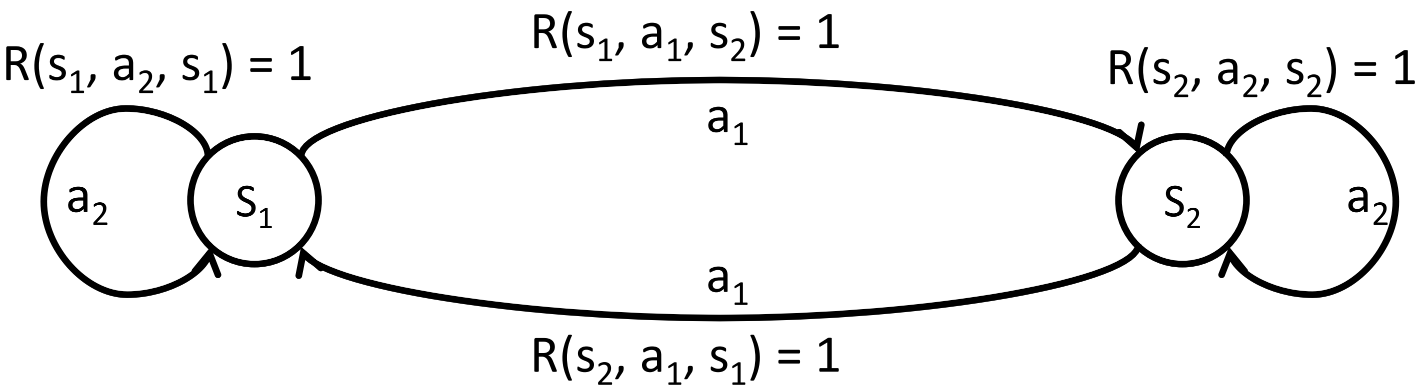


  – Utility of above policy at s_1 oscillates between 1 and 0

- **ELAU isn't well-defined unless the limit of the series $\mathbb{E}[R_1 + R_2 + \ldots]$ exists**

The Optimality Principle: Caveat #2

- The utility of many policies may be infinite



  - Every policy allows for ∞ reward from every state above

- **ELAU may not be a meaningful criterion unless $u(R_1, R_2, \ldots) = \mathbb{E}[R_1 + R_2 + \ldots]$ is bounded above.**

Recap So Far

- What is an MDP?
  - **M = <*S, D, A, T, R*>**
- What is *an* MDP solution?
  - **Policy π: $H \times A \rightarrow [0,1]$, a mapping from histories to distributions over actions.**
- What it an *optimal* MDP solution?
  - **π^* s.t. $V^*(h) \geq V^{\pi}(h)$ for all π and h, where $V^{\pi}(h)$ is the expected linear additive utility of rewards obtained after executing h**

Coming Up Next

Make sure ELAU is well-defined

- What is an MDP?
  - **Stationary M = <*S, D, A, T, R*>**
- What is *an* MDP solution?
  - **Policy π: *H x A* → [0,1], a mapping from histories to distributions over actions.**
- What it an *optimal* MDP solution?
  - **π^* s.t. $V^*(h) \geq V^\pi(h)$ for all π and *h*, where $V^\pi(h)$ is the expected linear additive utility of rewards obtained after executing h**

3 Models with Well-Defined Policy ELAU

1) Finite-horizon MDPs

2) Infinite-horizon discounted-reward MDPs

3) Stochastic Shortest-Path MDPs

Fundamentals of MDPs

- ✓ General MDP Definition
- ✓ Expected Linear Additive Utility
- ✓ The Optimality Principle
- **Finite-Horizon MDPs**
- Infinite-Horizon Discounted-Reward MDPs
- Stochastic Shortest-Path MDPs
- A Hierarchy of MDP Classes
- Factored MDPs
- Computational Complexity

Finite-Horizon MDPs: Motivation

- **Assume the agent acts for a finite # time steps, *L***
- Example applications:
  - Inventory management

    "How much X to order from the supplier every day 'til the end of the season?"

  - Maintenance scheduling

    "When to schedule disruptive maintenance jobs by their deadline?"

Finite-Horizon MDPs: Definition

Puterman, 1994

FH MDP is a tuple <*S, A, D, T, R*>, where:

- *S* is a finite state space
- *D* is a sequence of time steps *(1,2,3, …, L)* up to a finite *horizon L*
- *A* is a finite action set
- *T*: *S* x *A* x *S* x *D* → [0, 1] is a transition function
- *R*: *S* x *A* x *S* x *D* → $\mathbb{R}$ is a reward function

Policy value = ELAU over the remaining time steps

Aside: Deterministic Markovian Policies

- **For FH MDPs, we can consider only *deterministic Markovian* solutions**
  - Will shortly see why

- A policy is *deterministic* if for every history, it assigns all probability mass to one action:

$$\pi: H \rightarrow A$$

- A policy is *deterministic Markovian* if its decision in each state is independent of execution history:

$$\pi: S \times D \rightarrow A$$

Aside: Markovian Value Functions

- Markovian policies can be evaluated with *Markovian value functions*

- Let $h_{s,t}$ denote history ending in state s at time t

- $V^\pi(h_{s,t}) = V^\pi(h'_{s,t})$ for all $h_{s,t}$, $h'_{s,t}$ if π is Markovian

- Call V *Markovian* if for all $h_{s,t}$, $h'_{s,t}$, $V(h_{s,t}) = V(h'_{s,t})$
  - For each s, t denote Markovian V as $V(s,t)$

Finite-Horizon MDPs: Optimality Principle

For an FH MDP with *horizon* $|D| = L < \infty$, let:

Exp. Lin. Add. Utility

For every history, the value of every policy is well-defined!

- $V^{\pi}(h_{s,t}) = \mathbb{E}^{\pi}_{h,s,t}[R_1 + \ldots + R_{L-t}]$ for all $1 \le t \le L$
- $V^{\pi}(h_{s,L+1}) = 0$

Each E[R_i] is finite

terms in the series is finite

Then:

- V^* exists and is Markovian, π^* exists and is det. Markovian
- For all s and $1 \le t \le L$:

If you act optimally now

Immediate utility of the next action

$$V^*(s,t) = \max_{a \text{ in } A} \left[\sum_{s' \text{ in } S} T(s, a, s', t) \left[R(s, a, s', t) + V^*(s', t+1) \right] \right]$$

$$\pi^*(s,t) = \operatorname{argmax}_{a \text{ in } A} \left[\sum_{s' \text{ in } S} T(s, a, s', t) \left[R(s, a, s', t) + V^*(s', t+1) \right] \right]$$

Highest utility derivable from s at time t

In expectation

Highest utility derivable from the next state

Perks of the FH MDP Optimality Principle

- **V*, π* Markovian ⇨ consider only Markovian V, π!**

	Probabilistic history-dep. π	**Deterministic Markovian π**
Number	∞	$\boldsymbol{|A|^{|S||D|}}$
Size of each	Ginormous!	**O(|S||D|)**

- **Can easily compute π*!**
  - For all s, compute $V^*(s, t)$ and $\pi^*(s, t)$ for $t = L, \ldots, 1$

Moving to In(de)finite Horizon

- Finite known horizon sometimes not good enough
  - Doesn’t cover autonomous agents with long lifespans

- **Two other options:**
  - *Infinite horizon* (horizon known to be infinite)
  - *Indefinite horizon* (horizon known to be unbounded)

Moving to Infinite Horizon

- Finite known horizon sometimes not good enough
  - Doesn't cover autonomous agents with long lifespans
- **Two other options:**
  - *Infinite horizon* (horizon known to be infinite)
  - *Indefinite horizon* (horizon known to be unbounded)

Analyzing MDPs with In(de)finite Horizon

- **Hard to specify time-dependent *T*, *R*, etc. for a large (infinite) # steps**

- Need *stationary* (time-independent) *functions*:

  - *Stationary transition function* of the form

    T: $S \times A \times S \rightarrow [0, 1]$

  - *Stationary reward function* of the form

    R: $S \times A \times S \rightarrow \mathbb{R}$

  - *Stationary deterministic Markovian policy* of the form

    π: $S \rightarrow A$

  - *Stationary Markovian value function* of the form

    V: $S \rightarrow [-\infty, \infty]$

Fundamentals of MDPs

✓ General MDP Definition
✓ Expected Linear Additive Utility
✓ The Optimality Principle
✓ Finite-Horizon MDPs
- **Infinite-Horizon Discounted-Reward MDPs**
- Stochastic Shortest-Path MDPs
- A Hierarchy of MDP Classes
- Factored MDPs
- Computational Complexity

Infinite-Horizon Discounted-Reward MDPs: Motivation

- **Assume the agent acts for an infinitely long time**
- Example applications:
  - Portfolio management

    "How to invest money under a given rate of inflation?"

  - Unstable system control

    "How to help fly a B-2 bomber?"

Infinite-Horizon Discounted MDPs: Definition

Puterman, 1994

IHDR MDP is a tuple $<S, A, T, R, \gamma>$, where:

- S is a finite state space
- D is an infinite sequence $(1,2, \ldots)$
- A is a finite action set
- T: $S \times A \times S \rightarrow [0, 1]$ is a stationary transition function
- R: $S \times A \times S \rightarrow \mathbb{R}$ is a stationary reward function
- γ is a discount factor satisfying $0 \leq \gamma < 1$

Policy value = discounted ELAU over infinite time steps

Infinite-Horizon Discounted-Reward MDPs: Optimality Principle

For an IHDR MDP, let:

Exp. Lin. Add. Utility

For every history, the value of a policy is well-defined thanks to $0 \leq \gamma < 1$!

– $V^{\pi}(h) = \mathbb{E}^{\pi}_{h}[R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots]$ for all h

All $\gamma^i E[R_i]$ are bounded by some finite K and converge geometrically

Then:

– V^* exists and is stationary Markovian, π^* exists and is stationary deterministic Markovian

– For all s:

Optimal utility is time-independent!

Future utility is discounted

$$\boldsymbol{V^*(s) = \max_{a \text{ in } A} [\, \textstyle\sum_{s' \text{ in } S} T(s, a, s') [\, R(s, a, s') + \gamma V^*(s') \,]\,]}$$

$$\boldsymbol{\pi^*(s) = \operatorname{argmax}_{a \text{ in } A} [\, \textstyle\sum_{s' \text{ in } S} T(s, a, s') [\, R(s, a, s') + \gamma V^*(s') \,]\,]}$$

Perks of the IFHD MDP Optimality Principle

- ***V^*, π^** stationary Markovian ⇨ consider only stationary Markovian *V, π*!**

	Deterministic Markovian π	**Stationary deterministic Markovian π**
Number	∞	$\boldsymbol{\lvert A \rvert^{\lvert S \rvert}}$
Size of each	∞	**$O(\lvert S \rvert)$**

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

Moving to Indefinite Horizon

- Finite known horizon sometimes not good enough
  - Doesn't cover autonomous agents with long lifespans.

- **Two other options:**
  - *Infinite horizon* (horizon known to be infinite)
  - *Indefinite horizon* (horizon known to be unbounded)

Fundamentals of MDPs

- ✓ General MDP Definition
- ✓ Expected Linear Additive Utility
- ✓ The Optimality Principle
- ✓ Finite-Horizon MDPs
- ✓ Infinite-Horizon Discounted-Reward MDPs
- **Stochastic Shortest-Path MDPs**
- A Hierarchy of MDP Classes
- Factored MDPs
- Computational Complexity

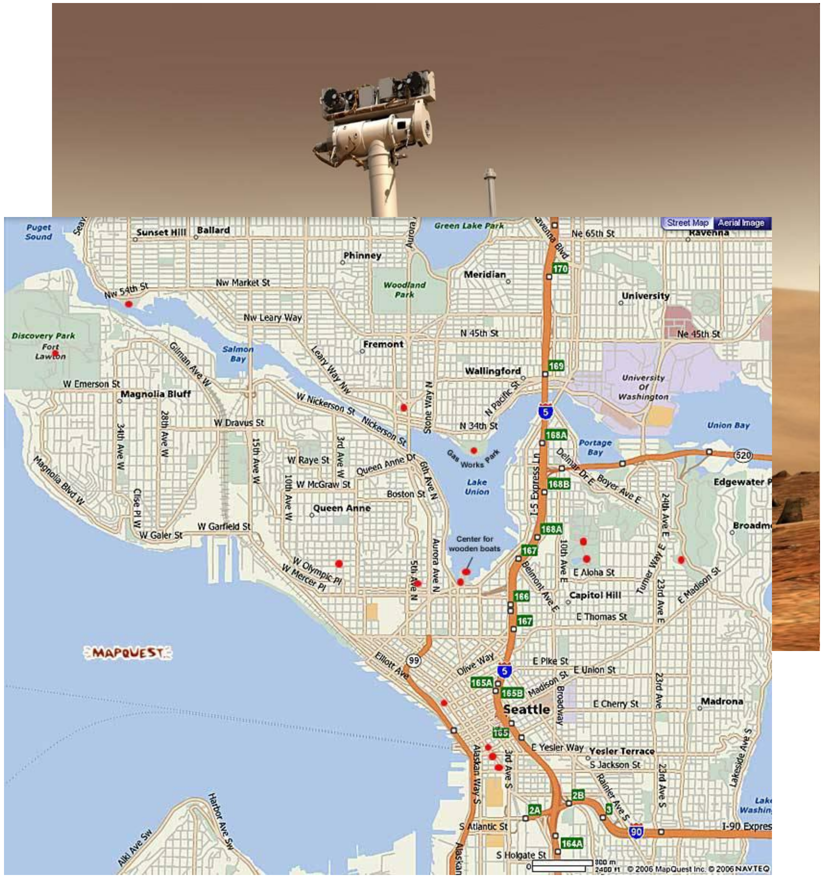
Stochastic Shortest-Path MDPs: Motivation

- **Assume the agent pays cost to achieve a goal**
- Example applications:
  - Controlling a Mars rover

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

  - Navigation

    "What's the fastest way to get to a destination, taking into account the traffic jams?"



Stochastic Shortest-Path MDPs: Definition

Bertsekas, 1995

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

- *S* is a finite state space
- (*D* is an infinite sequence *(1,2, …)*)
- *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* (= -R: *S* x *A* x *S* → $\mathbb{R}$)
- *G* is a set of absorbing cost-free goal states

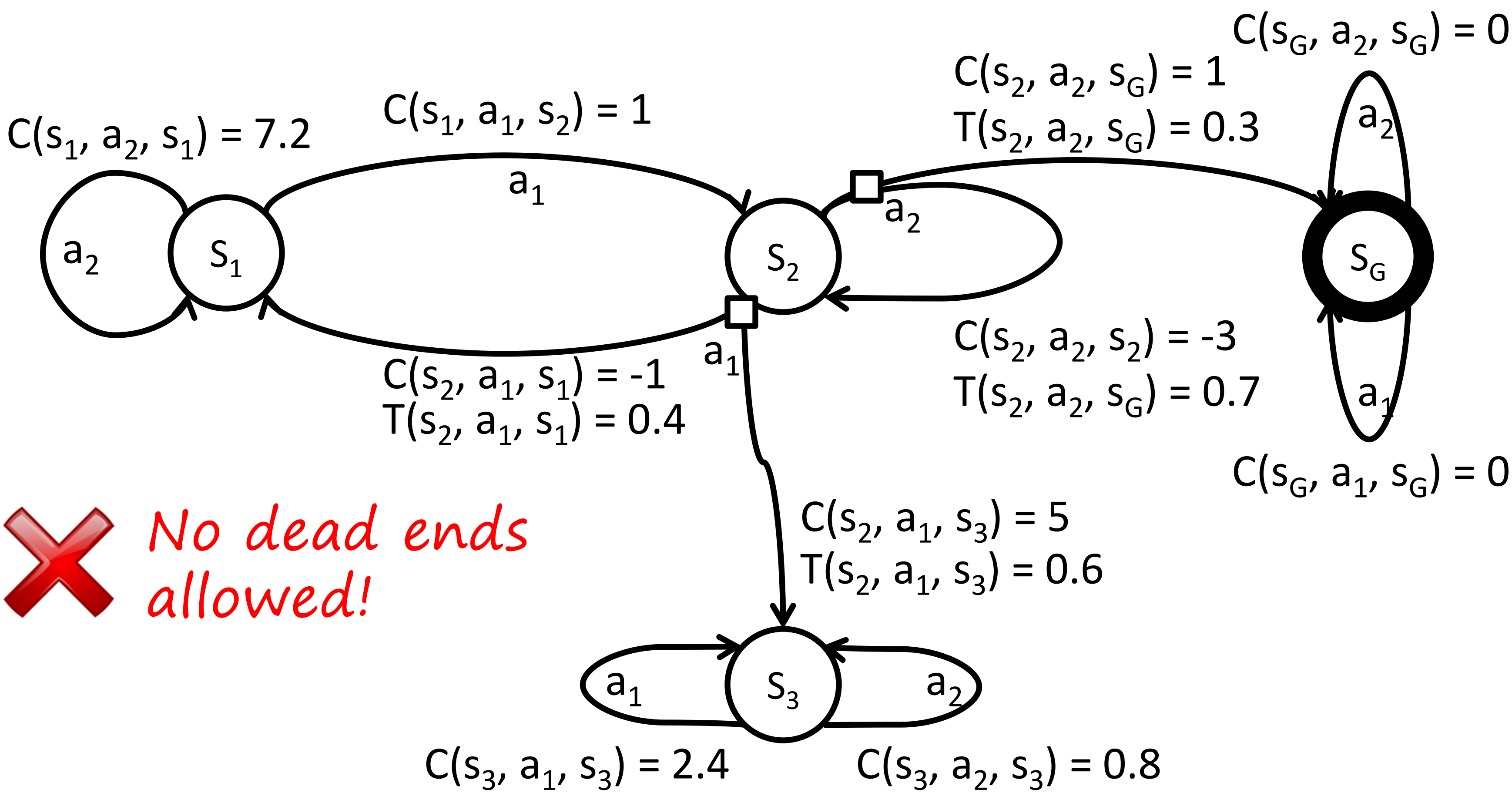
Under two conditions:

- 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

SSP MDP Details

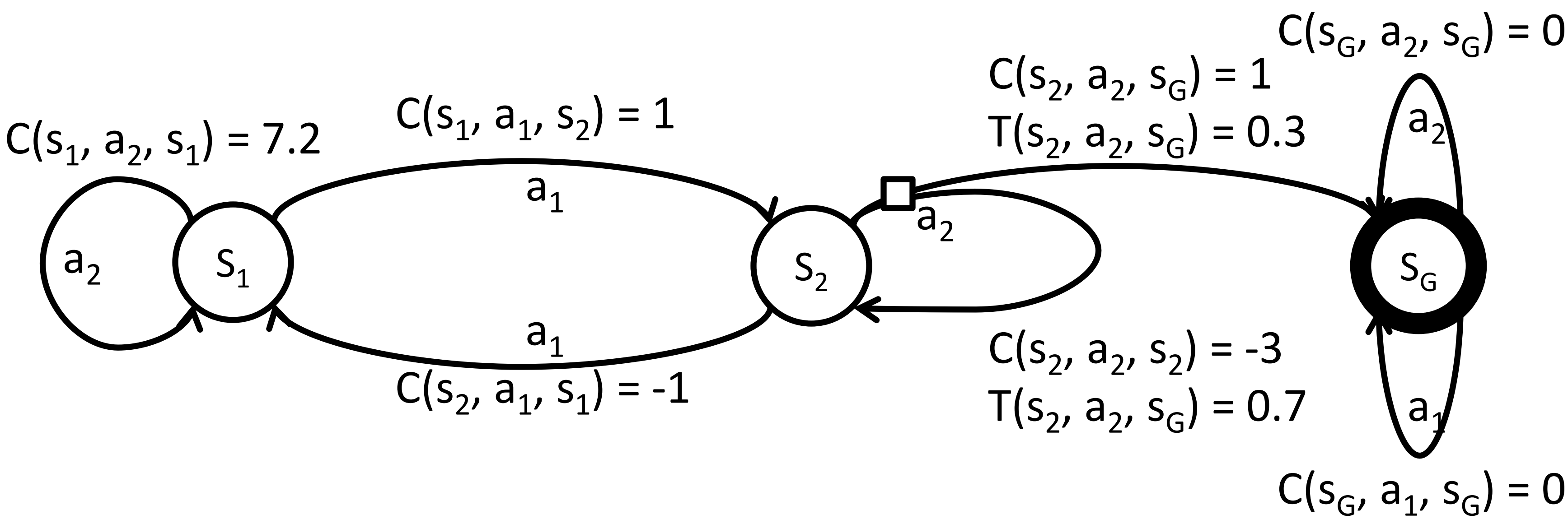
- In SSP, maximizing ELAU = *minimizing* exp. cost
- Every cost-minimizing policy is proper!
- **Thus, an optimal policy = cheapest way to a goal**
- Why are SSP MDPs called “indefinite-horizon”?
  - If a policy is optimal, it will take a finite, but apriori unknown, time to reach goal

SSP MDP Example, not!



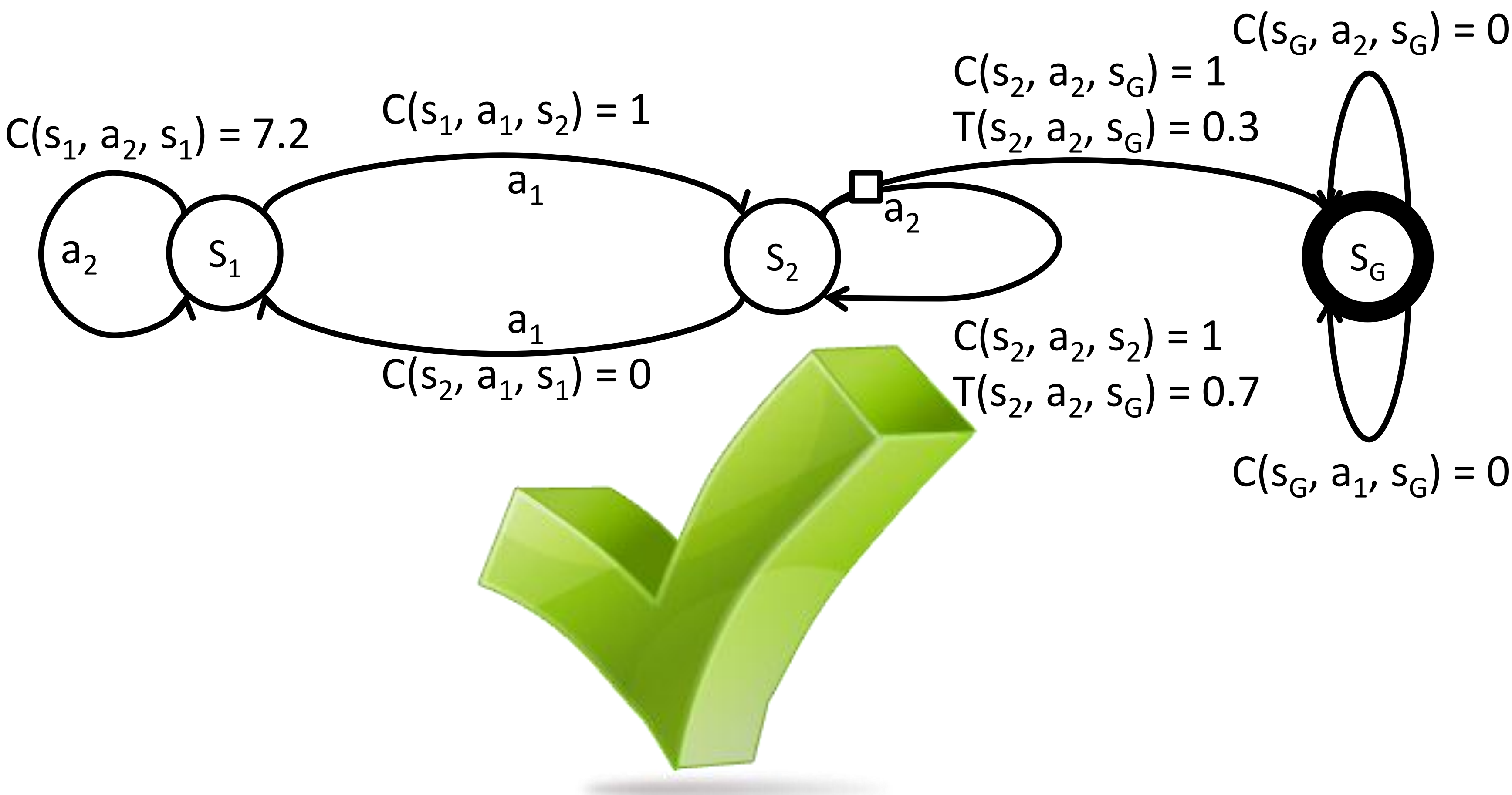
No dead ends allowed!

SSP MDP Example, also not!



No cost-free "loops" allowed!

SSP MDP Example



SSP MDPs: Optimality Principle

For an SSP MDP, let:

Exp. Lin. Add. Utility

For every history, the value of a policy is well-defined!

– $V^\pi(h) = \underbrace{\mathbb{E}^\pi_h[C_1 + C_2 + \ldots]}_{\text{Every policy either takes a finite exp. \# of steps to reach a goal, or has an infinite cost.}}$ for all h

Then:

– V^* exists and is stationary Markovian, π^* exists and is stationary deterministic Markovian

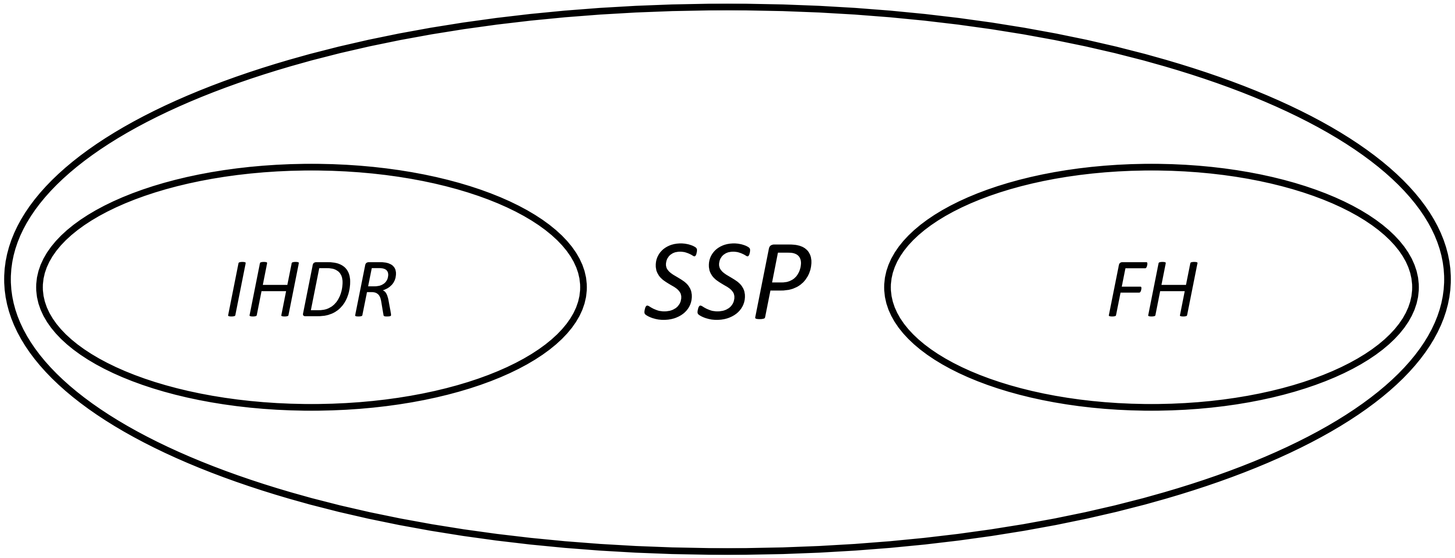
– For all s:

$$\boldsymbol{V^*(s) = \min_{a \text{ in } A} [\ \textstyle\sum_{s' \text{ in } S} T(s, a, s')\ [\ C(s, a, s') + V^*(s')\ ]\ ]}$$
$$\boldsymbol{\pi^*(s) = \operatorname{argmin}_{a \text{ in } A}\ [\ \textstyle\sum_{s' \text{ in } S} T(s, a, s')\ [\ C(s, a, s') + V^*(s')\ ]\ ]}$$

Fundamentals of MDPs

- ✓ General MDP Definition
- ✓ Expected Linear Additive Utility
- ✓ The Optimality Principle
- ✓ Finite-Horizon MDPs
- ✓ Infinite-Horizon Discounted-Reward MDPs
- ✓ Stochastic Shortest-Path MDPs
- **A Hierarchy of MDP Classes**
- Factored MDPs
- Computational Complexity

SSP and Other MDP Classes



- ***FH* => *SSP***: turn all states (s, L) into goals
- ***IHDR* => *SSP***: add (1-γ)-probability transitions to goal
- **Will concentrate on *SSP* in the rest of the tutorial**

Fundamentals of MDPs

- ✓ General MDP Definition
- ✓ Expected Linear Additive Utility
- ✓ The Optimality Principle
- ✓ Finite-Horizon MDPs
- ✓ Infinite-Horizon Discounted-Reward MDPs
- ✓ Stochastic Shortest-Path MDPs
- ✓ A Hierarchy of MDP Classes
- **Factored MDPs**
- Computational Complexity

Factored SSP MDPs: Motivation

- How to describe an MDP instance?
  - $S = \{s_1, \dots, s_n\}$ – *flat* representation
  - $T(s_i, a_j, s_k) = p_{i,j,k}$ for every state, action, state triplet
  - …
- **Flat representation too cumbersome!**
  - Real MDPs have billions of billions of states
  - Can't enumerate transition function explicitly
- **Flat representation too uninformative!**
  - State space has no meaningful distance measure
  - Tabulated transition/reward function has no structure

Factored SSP MDPs: Definition

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

- *X* is a finite set of *state variables* (*domain variables, features*)
- (*D* is an infinite sequence *(1,2, …)*)
- *A* is a finite action set
- *T*: $(\mathrm{dom}(X_1) \times \ldots \times \mathrm{dom}(X_n)) \times A \times (\mathrm{dom}(X_1) \times \ldots \times \mathrm{dom}(X_n)) \rightarrow [0, 1]$ is a stationary transition function
- *C*: $(\mathrm{dom}(X_1) \times \ldots \times \mathrm{dom}(X_n)) \times A \times (\mathrm{dom}(X_1) \times \ldots \times \mathrm{dom}(X_n)) \rightarrow \mathbb{R}$ is a stationary cost function
- *G,* given by a conjunction over a subset of *X*, is a set of goal states

The conditions of the flat SSP MDP definition still apply

Factored Representation Languages

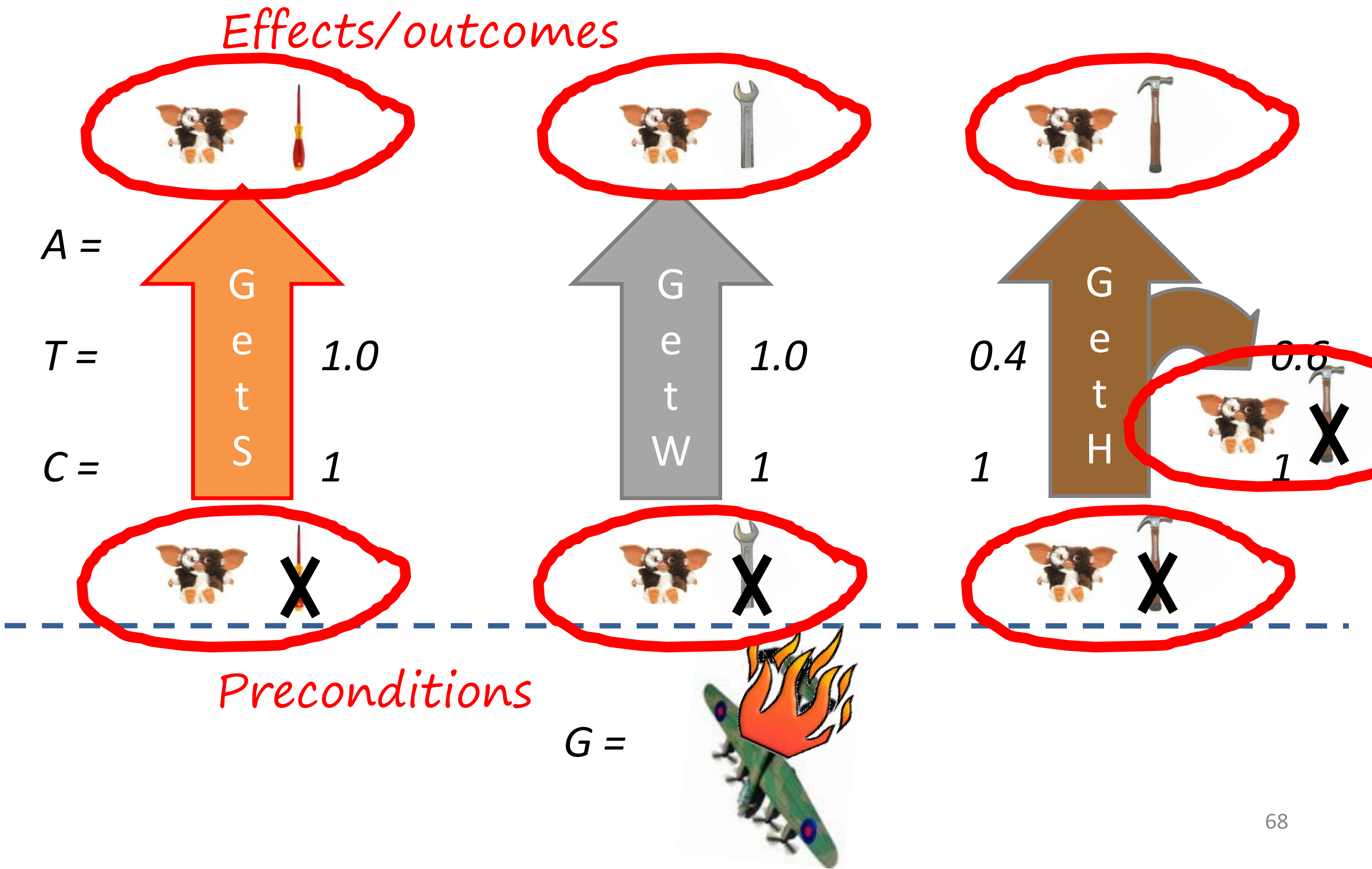
- *PPDDL* – Prob. Planning Domain Definition Language [Younes and Littman, 2004]

- *RDDL* – Relational Domain Definition Language [Sanner, 2011]

Example Factored SSP MDP in PPDDL

- Gremlin wants to sabotage an airplane
- Can use tools to fulfill its objective
- Needs to pick up the tools
- X = {  }

Example Factored SSP MDP in PPDDL

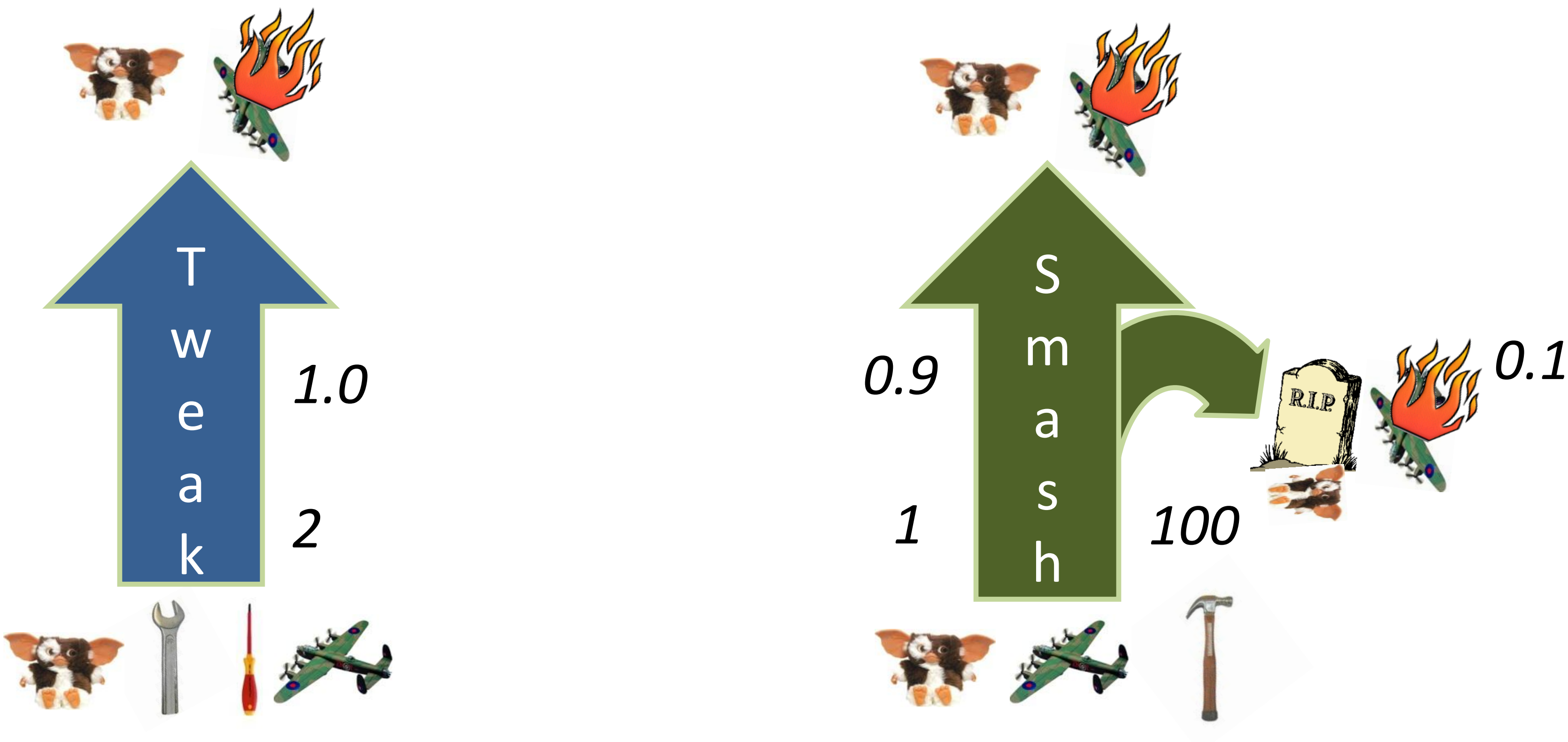


Example Factored SSP MDP in PPDDL

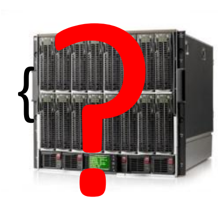
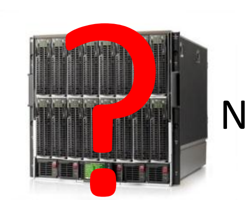
A =

T =

C =

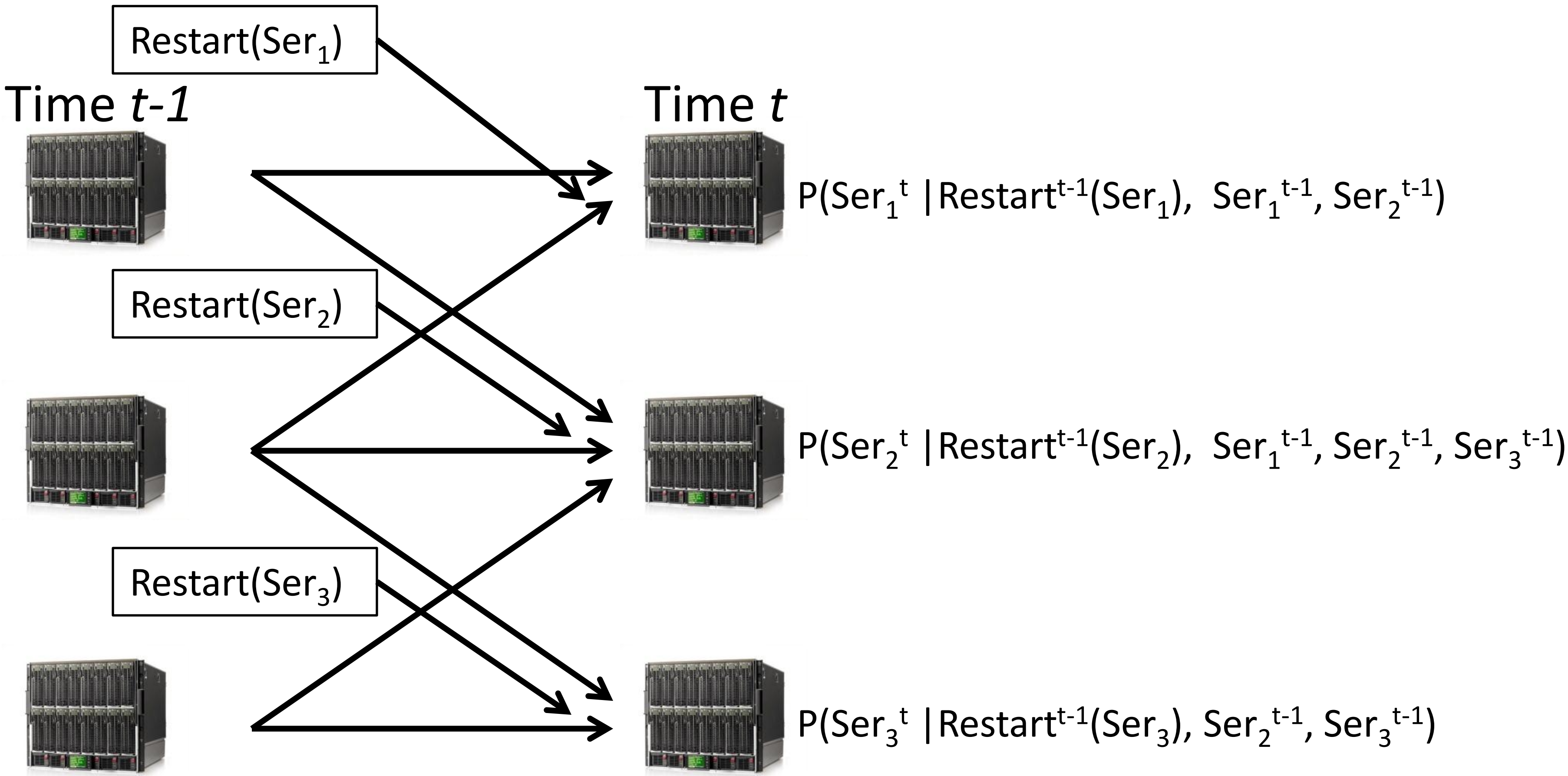


Example Factored SSP MDP in RDDL

- Sysadmin needs to maintain a network of servers until time *L*
  - Gets paid proportionately to the # of servers running at each time step
- Each server can go up or down with some probability
  - And drag its neighbors down – probability of going down increases with the number of down neighbors
- Sysadmin can restart just one server per time step
- **Enormous number of uncorrelated effects for each action**
  - 2^N for a problem with N servers
- X = {  $_1$, … ,  $_N$ }, G = any state at time *L*

Example Factored SSP MDP in RDDL

A: C: $-\sum_i [Ser_i = \uparrow]$ *T*:



Factored Representation Languages Summary

- *PPDDL* – Prob. Planning Domain Definition Language
  - Represents MDP actions as templates
  - Good for MDPs with strongly *correlated effects*
  - Inconvenient for MDPs with uncorrelated effects

- *RDDL* – Relational Domain Definition Language
  - Represents MDP as a Dynamic Bayes Net
  - Shows how each variable evolves under every action
  - Good for MDPs with uncorrelated effects
  - Inconvenient for MDPs with uncorrelated strongly *correlated effects*

Benefits of Factored Representations

- Can meaningfully group states
  - E.g., by similarity
  - And assign the same policy to each group

- Can meaningfully express *V* as a function of state variables
  - Using mathematical operations, e.g. $V(s) = X_1(s) + \ldots + X_n(s)$
  - Basis of *dimensionality reduction* techniques

- Can manipulate values of sets of states
  - Symbolic and approximate algorithms, more on this later

Fundamentals of MDPs

- ✓ General MDP Definition
- ✓ Expected Linear Additive Utility
- ✓ The Optimality Principle
- ✓ Finite-Horizon MDPs
- ✓ Infinite-Horizon Discounted-Reward MDPs
- ✓ Stochastic Shortest-Path MDPs
- ✓ A Hierarchy of MDP Classes
- ✓ Factored MDPs
- **Computational Complexity**

Computational Complexity of MDPs

- **Good news**:
  - Solving *IHDR, SSP* in flat representation is *P*-complete
  - Solving *FH* in flat representation is *P*-hard
  - That is, they don't benefit from parallelization, but are solvable in polynomial time!

Computational Complexity of MDPs

- **Bad news**:
  - Solving *FH, IHDR, SSP* in factored representation is *EXPTIME*-complete!
  - Flat representation doesn't make MDPs harder to solve, it makes big ones easier to describe.

Computational Complexity of MDPs

- **Consolation**:
  - Introduce *factored SSP_{s0} (FH_{s0}, $IFHD_{s0}$)*– factored MDP with a designated *initial state* s_0
  - Assume an optimal policy starting at s_0 visits at most O(poly|X|) states
  - *FH, IHDR* SSP_{s0} with O(poly|X|) optimal policy size are *PSPACE*-complete!

Summary So Far

- **Introduced a broad MDP definition**
  - It had an ill-defined optimal solution concept
- **Imposed restrictions on the general definition to make optimal solution well-defined**
  - Based on expected linear additive utility
  - Gave rise to *FH, IHDR,* and *SSP*
- **Introduced factored representations**
  - Convenient to use, but make MDPs look hard to solve
  - In fact, they *are* hard to solve…

Outline of the Tutorial

- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

UNINFORMED ALGORITHMS

Uninformed Algorithms

- Definitions
- Fundamental Algorithms
- Prioritized Algorithms
- Partitioned Algorithms
- Other models

Stochastic Shortest-Path MDPs: Definition

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*
- *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

- Solution of an SSP: policy (π: S→A)

Uninformed Algorithms

- Definitions
- Fundamental Algorithms
- Prioritized Algorithms
- Partitioned Algorithms
- Other models

Brute force Algorithm

- Go over all policies π
  - How many? $|A|^{|S|}$ finite
- Evaluate each policy how to evaluate?
  - $V^{\pi}(s) \leftarrow$ expected cost of reaching goal from s
- Choose the best
  - We know that best exists (SSP optimality principle)
  - $V^{\pi*}(s) \leq V^{\pi}(s)$

Policy Evaluation

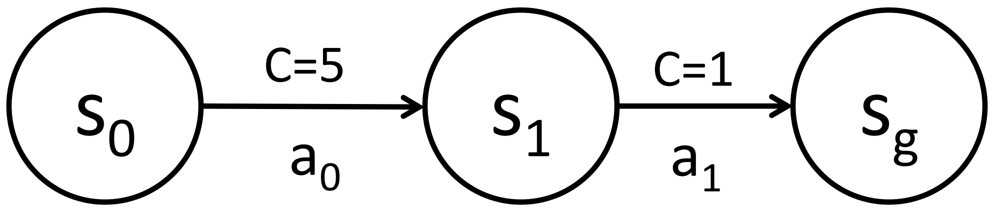
- Given a policy π: compute V^{π}

- TEMPORARY ASSUMPTION: π is proper
  - execution of π reaches a goal from any state

Deterministic SSPs

- Policy Graph for π

  $\pi(s_0) = a_0;\ \pi(s_1) = a_1$

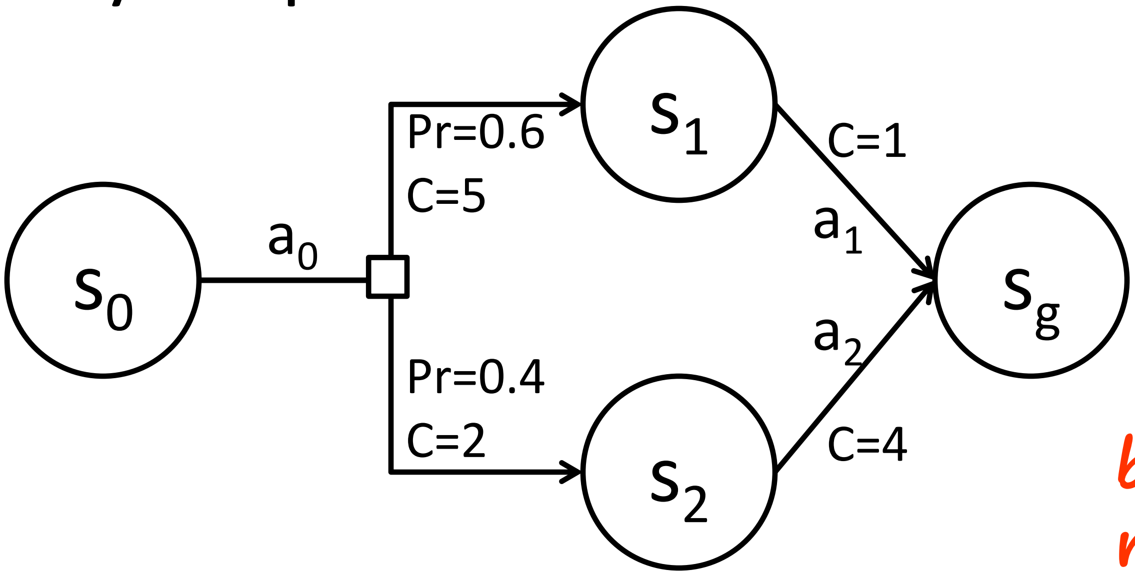


- $V^\pi(s_1) = 1$
- $V^\pi(s_0) = 6$

add costs on ***path*** to goal

Acyclic SSPs

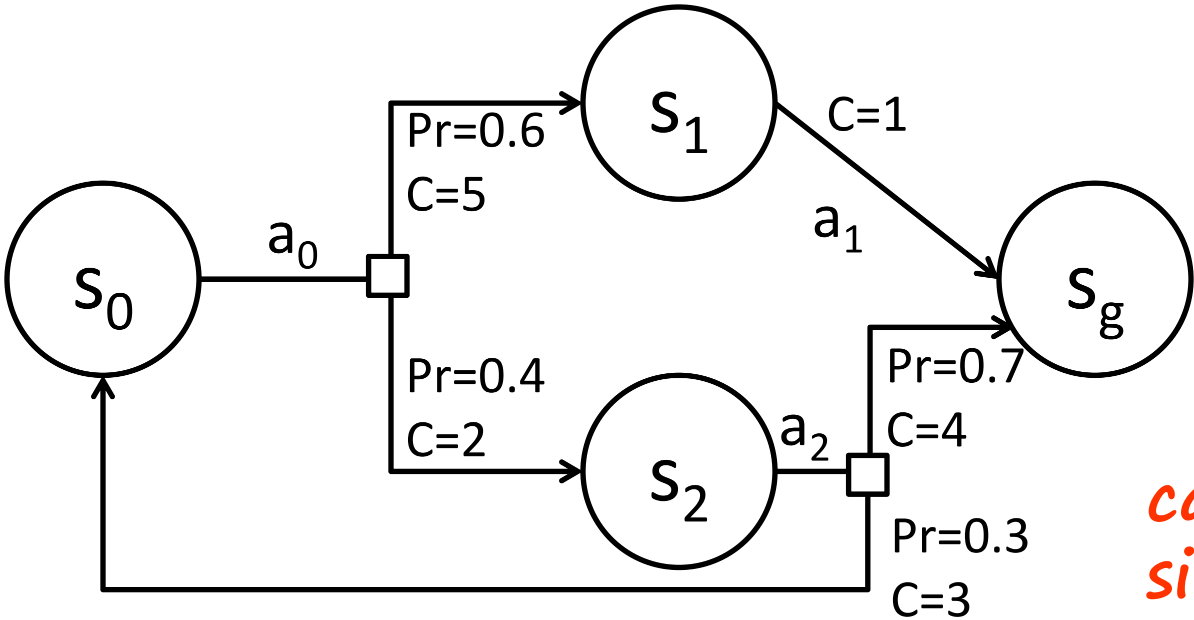
- Policy Graph for π



backward pass in reverse topological order

- $V^{\pi}(s_1) = 1$
- $V^{\pi}(s_2) = 4$
- $V^{\pi}(s_0) = 0.6(5+1) + 0.4(2+4) = 6$

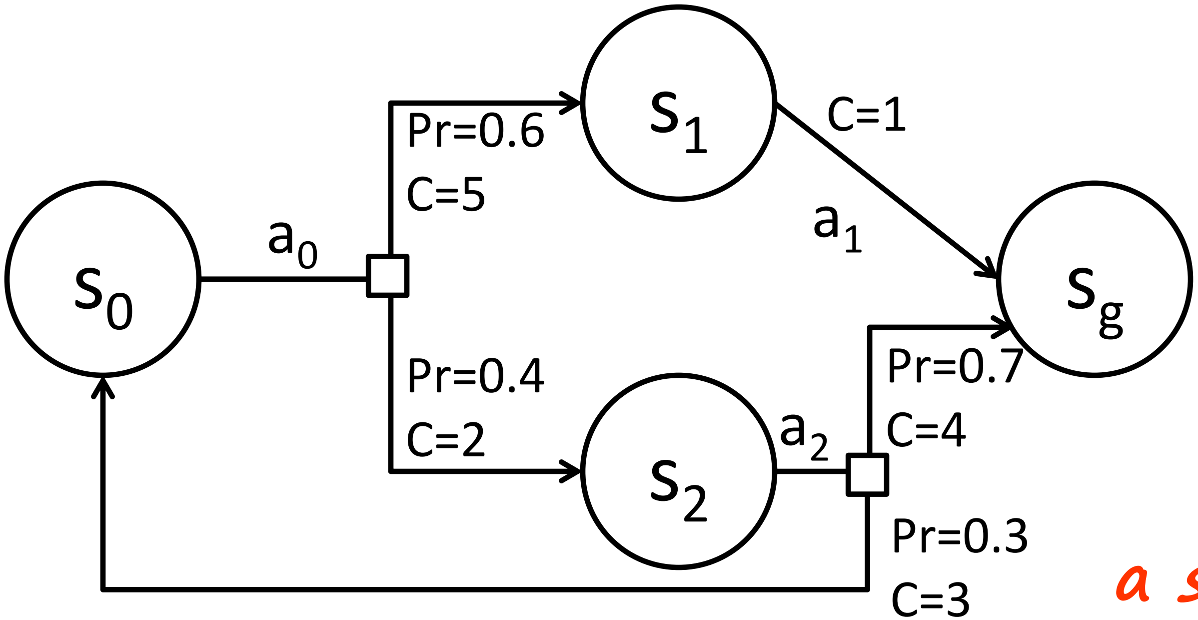
General SSPs can be cyclic!



cannot do a simple single pass

- $V^{\pi}(s_1) = 1$
- $V^{\pi}(s_2)$ = ?? (depends on $V^{\pi}(s_0)$)
- $V^{\pi}(s_0)$ = ?? (depends on $V^{\pi}(s_2)$)

General SSPs can be cyclic!



a simple system of linear equations

- $V^\pi(g) = 0$
- $V^\pi(s_1) = 1 + V^\pi(s_g) = 1$
- $V^\pi(s_2) = 0.7(4 + V^\pi(s_g)) + 0.3(3 + V^\pi(s_0))$
- $V^\pi(s_0) = 0.6(5 + V^\pi(s_1)) + 0.4(2 + V^\pi(s_2))$

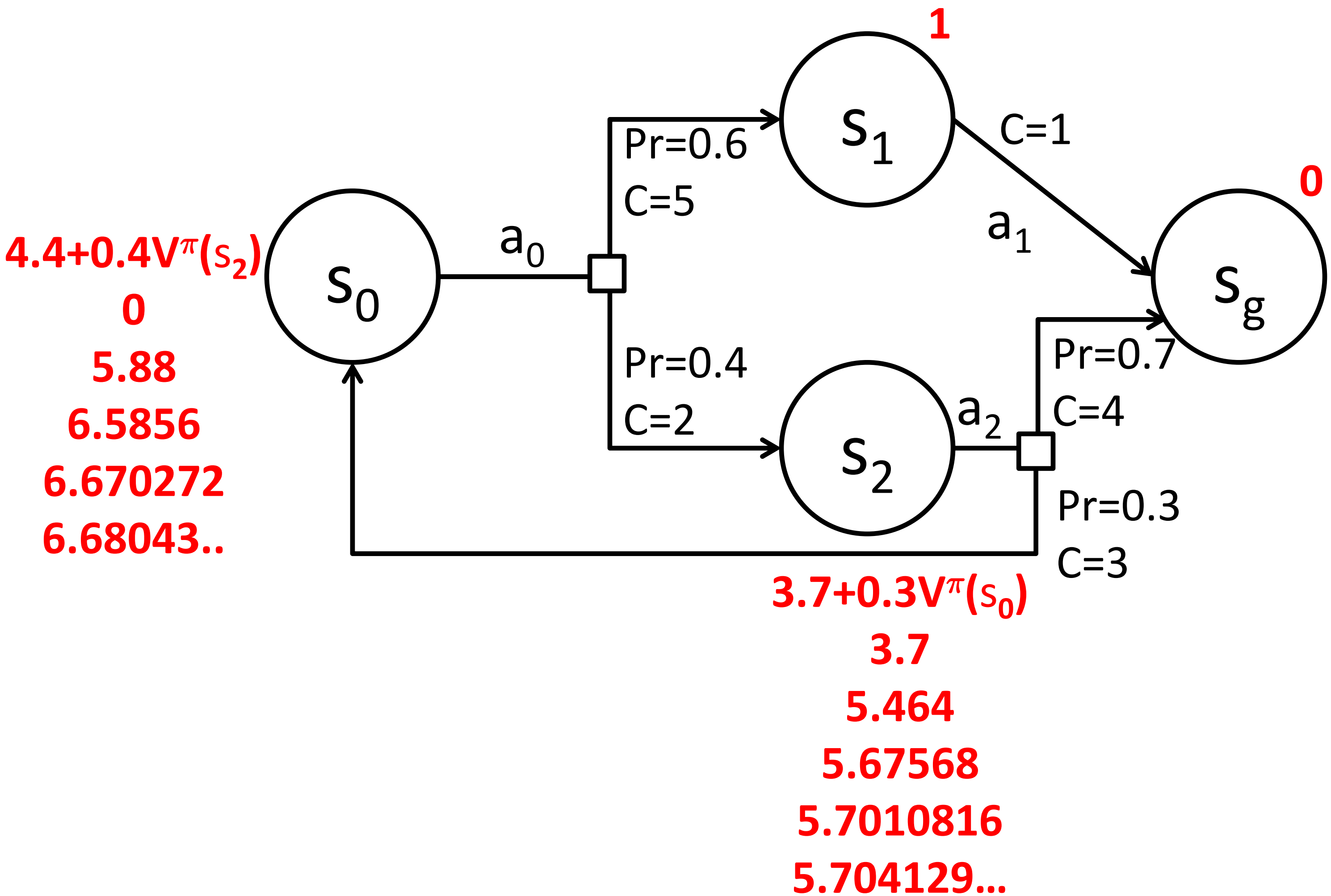
Policy Evaluation (Approach 1)

- Solving the System of Linear Equations

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

- $|S|$ variables.
- $O(|S|^3)$ running time

Iterative Policy Evaluation



Policy Evaluation (Approach 2)

$$V^\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \left[\mathcal{C}(s, \pi(s), s') + V^\pi(s')\right]$$

iterative refinement

$$V_n^\pi(s) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \left[\mathcal{C}(s, \pi(s), s') + V_{n-1}^\pi(s')\right]$$

Iterative Policy Evaluation

//Assumption: π is proper
initialize V_0^π arbitrarily for each state
$n \leftarrow 0$
repeat
    $n \leftarrow n + 1$
    foreach $s \in \mathcal{S}$ **do**
        compute $V_n^\pi(s) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \left[\mathcal{C}(s, \pi(s), s') + V_{n-1}^\pi(s')\right]$
        compute $\text{residual}_n(s) \leftarrow |V_n^\pi(s) - V_{n-1}^\pi(s)|$
    end
until $\max_{s \in \mathcal{S}} \text{residual}_n(s) < \epsilon$;
return V_n^π

iteration n

ϵ-consistency

termination condition

Convergence & Optimality

For a proper policy π

Iterative policy evaluation

converges to the true value of the policy, i.e.

$$\lim_{n \to \infty} V_n^{\pi} = V^{\pi}$$

irrespective of the initialization V_0

Brute force Algorithm

- Go over all policies π:
  - How many? $|A|^{|S|}$ ← too slow ← choose an intelligent order for π
- Evaluate each policy ← how to evaluate?
  - $V^{\pi}(s) \leftarrow$ expected cost of reaching goal from s
- Choose the best
  - We know that best exists (SSP optimality principle)
  - $V^{\pi*}(s) \leq V^{\pi}(s)$

Q-Value under a Value Function V

- The Q-value of state s and action a under a value function V
  - denoted as $Q^V(s,a)$
- one-step lookahead computation of the value of a
  - assuming V is true expected cost to reach goal

$$\boxed{Q^V(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{C}(s,a,s') + V(s') \right]}$$

Greedy Action/Policy

- Define *a greedy action a wrt V*
  - an action that has the lowest *Q*-value, i.e.
  - $a = \mathrm{argmin}_{a'} Q^V(s,a')$
- Define *a greedy policy* π^V
  - Policy with all greedy actions wrt *V* for each state

Policy Iteration [Howard 60]

- initialize π_0 as a random proper policy

- repeat

  Policy Evaluation: Compute $V^{\pi_{n-1}}$

  Policy Improvement: Construct π_n greedy wrt $V^{\pi_{n-1}}$

- until $\pi_n == \pi_{n-1}$

- return π_n

choose π_{n-1}
if multiple greedy actions

Properties

- Policy Iteration for an SSP

  (initialized with a proper policy π_0)

  Successively improves the policy in each iteration, i.e. $V^{\pi_n}(s) \leq V^{\pi_{n-1}}(s)$, and

  converges to an optimal policy

Modified Policy Iteration [van Nunen 76]

- initialize π_0 as a random proper policy
- repeat

  Approximate Policy Evaluation: Compute $V^{\pi_{n-1}}$

  by running only few iterations of iterative policy eval.

  Policy Improvement: Construct π_n greedy wrt $V^{\pi_{n-1}}$
- until …
- return π_n

Limitations of PI

- Why do we need to start with a proper policy?
  - Policy Evaluation will diverge
- How to get a proper policy?
  - No domain independent algorithm
- PI for SSPs is not generally applicable

Policy Iteration → Value Iteration

- Changing the search space.

- Policy Iteration
  - Search over policies
  - Compute the resulting value

- Value Iteration
  - Search over values
  - Compute the resulting policy

Optimality Principle/Bellman Equations

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

Q*(s,a)

$V^*(s) = \min_a Q^*(s,a)$

Fixed Point Computation in VI

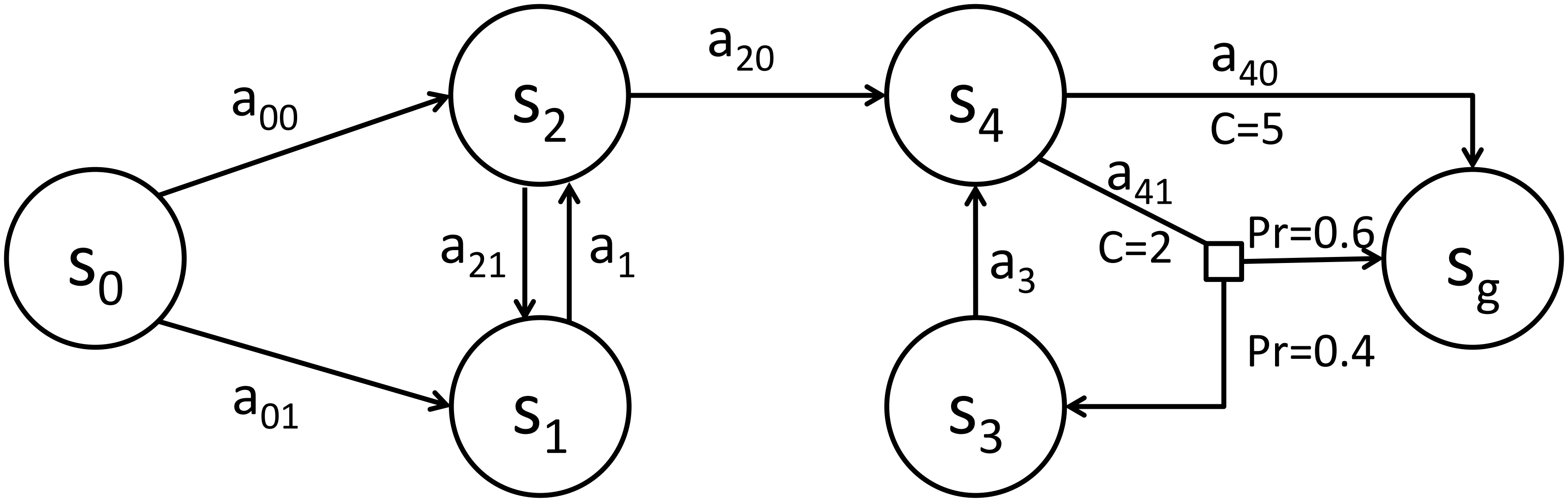
$$V^*(s) = \min_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{C}(s,a,s') + V^*(s') \right]$$

iterative refinement

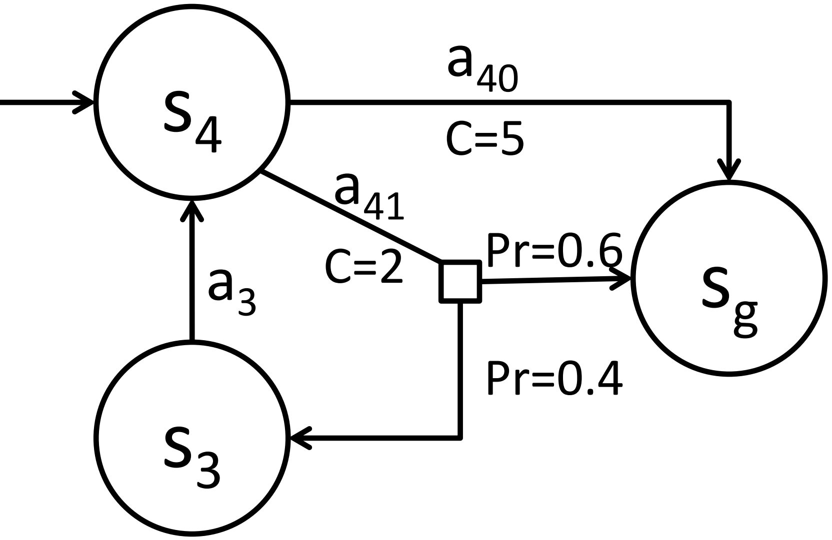
$$V_n(s) \leftarrow \min_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{C}(s,a,s') + V_{n-1}(s') \right]$$

non-linear

Running Example

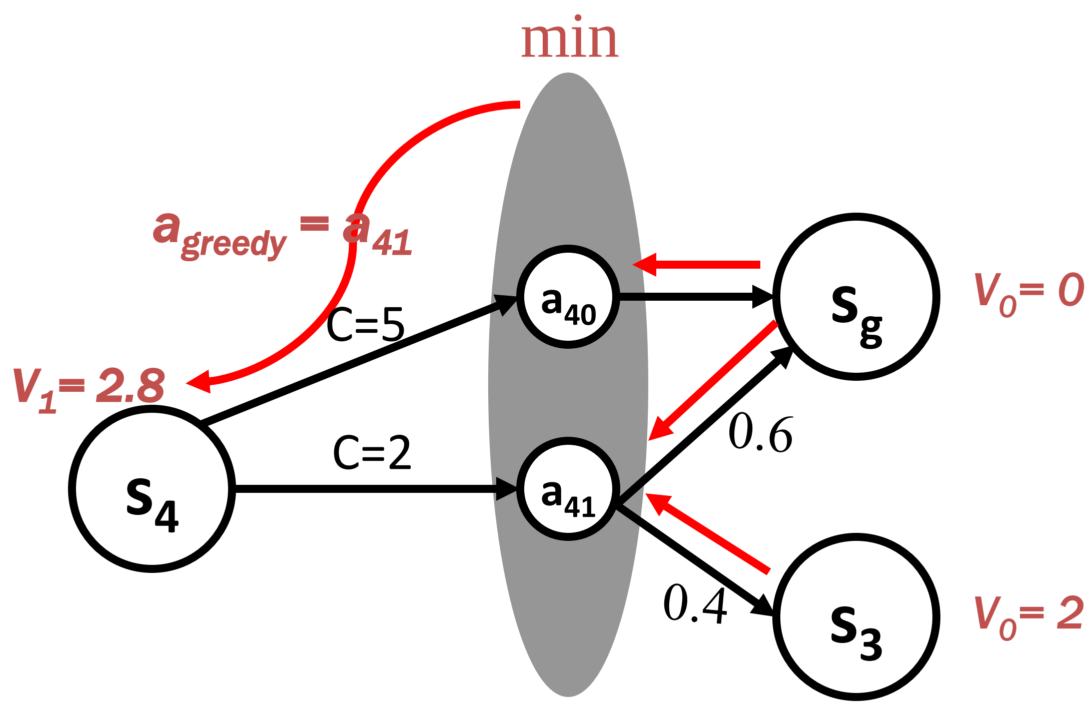


Bellman Backup



$Q_1(s_4,a_{40}) = 5 + 0$

$Q_1(s_4,a_{41}) = 2 + 0.6 \times 0$
$+ 0.4 \times 2$
$= 2.8$



Value Iteration [Bellman 57]

No restriction on initial value function

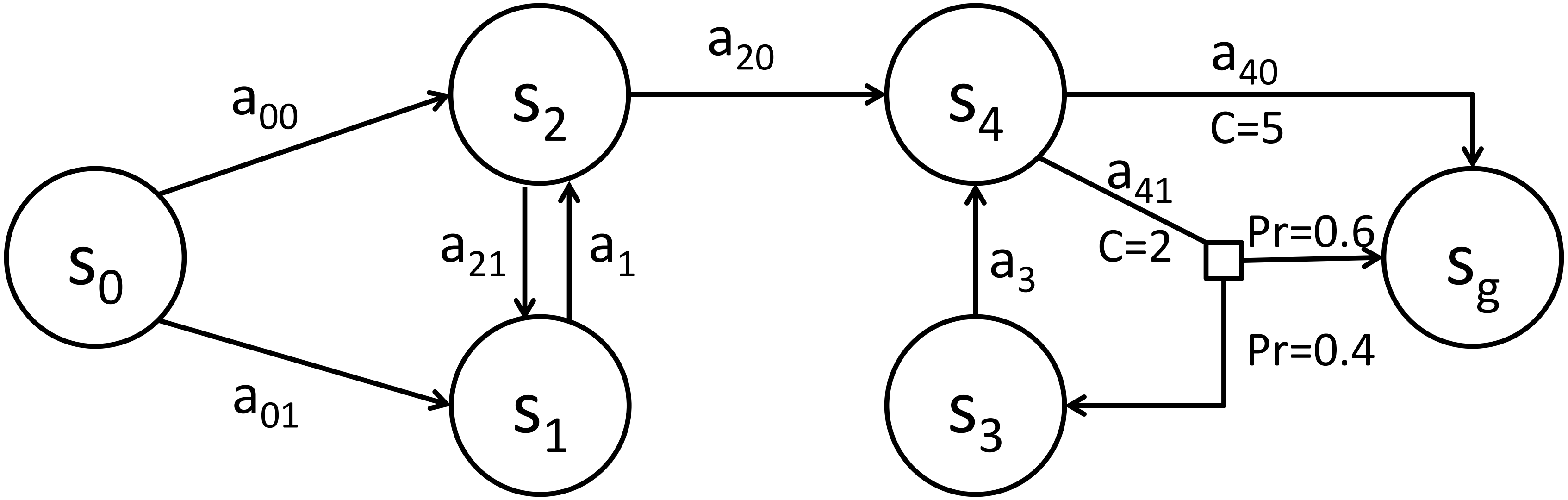
iteration n

initialize V_0 arbitrarily for each state
$n \leftarrow 0$
repeat
    $n \leftarrow n + 1$
    foreach $s \in \mathcal{S}$ **do**
        compute $V_n(s)$ using Bellman backup at s
        compute residual$_n(s) = |V_n(s) - V_{n-1}(s)|$
    end
until $\max_{s \in \mathcal{S}}$ residual$_n(s) < \epsilon$;
return greedy policy: $\pi^{V_n}(s) = \mathrm{argmin}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{C}(s,a,s') + V_n(s')\right]$

ϵ-consistency

termination condition

Running Example



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

Convergence & Optimality

- For an SSP MDP, $\forall s \in \mathcal{S}$,

$$\lim_{n \to \infty} V_n(s) = V^*(s)$$

irrespective of the initialization.

Running Time

- Each Bellman backup:
  - Go over all states and all successors: $O(|S||A|)$
- Each VI Iteration
  - Backup all states: $O(|S|^2|A|)$
- Number of iterations
  - General SSPs: no good bounds
  - Special cases: better bounds
    - (e.g., when all costs positive [Bonet 07])

SubOptimality Bounds

- General SSPs
  - weak bounds exist on $|V_n(s) - V^*(s)|$
- Special cases: much better bounds exist
  - (e.g., when all costs positive [Hansen 11])

Monotonicity

For all n>k

$V_k \leq_p V^* \Rightarrow V_n \leq_p V^*$ (V_n monotonic from below)

$V_k \geq_p V^* \Rightarrow V_n \geq_p V^*$ (V_n monotonic from above)

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 > ε do
    select a state s
    compute V(s) using a Bellman backup at s
    update Res^V(s)
end
return greedy policy π^V
```

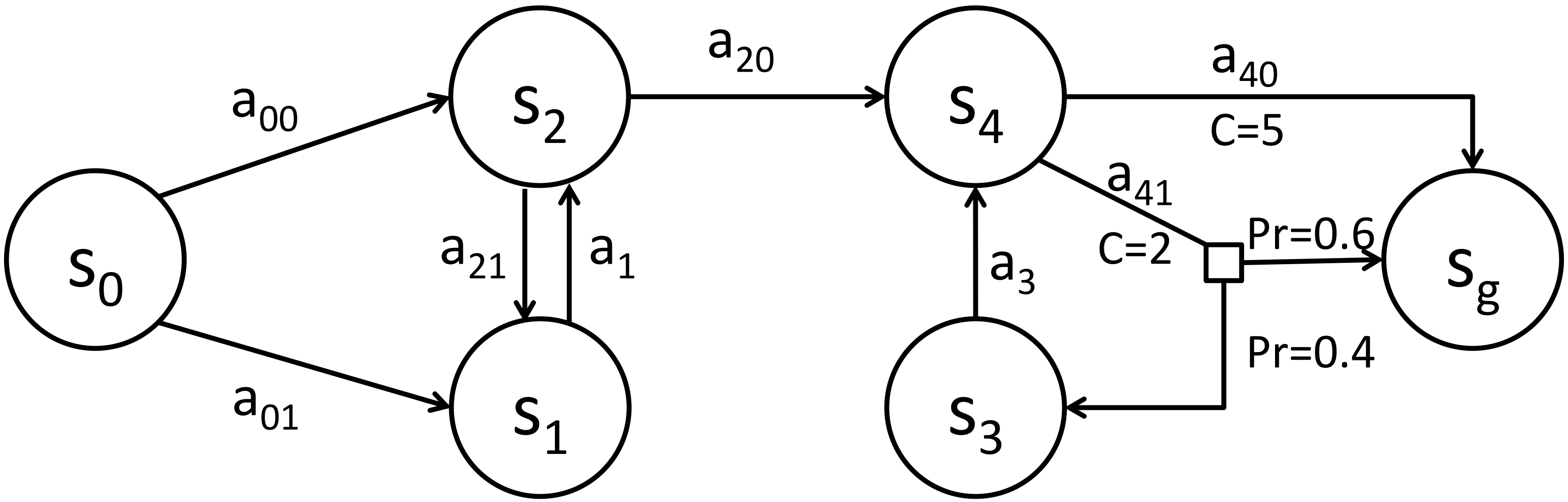
Uninformed Algorithms

- Definitions
- Fundamental Algorithms
- Prioritized Algorithms
- Partitioned Algorithms
- Other models

Prioritization of Bellman Backups

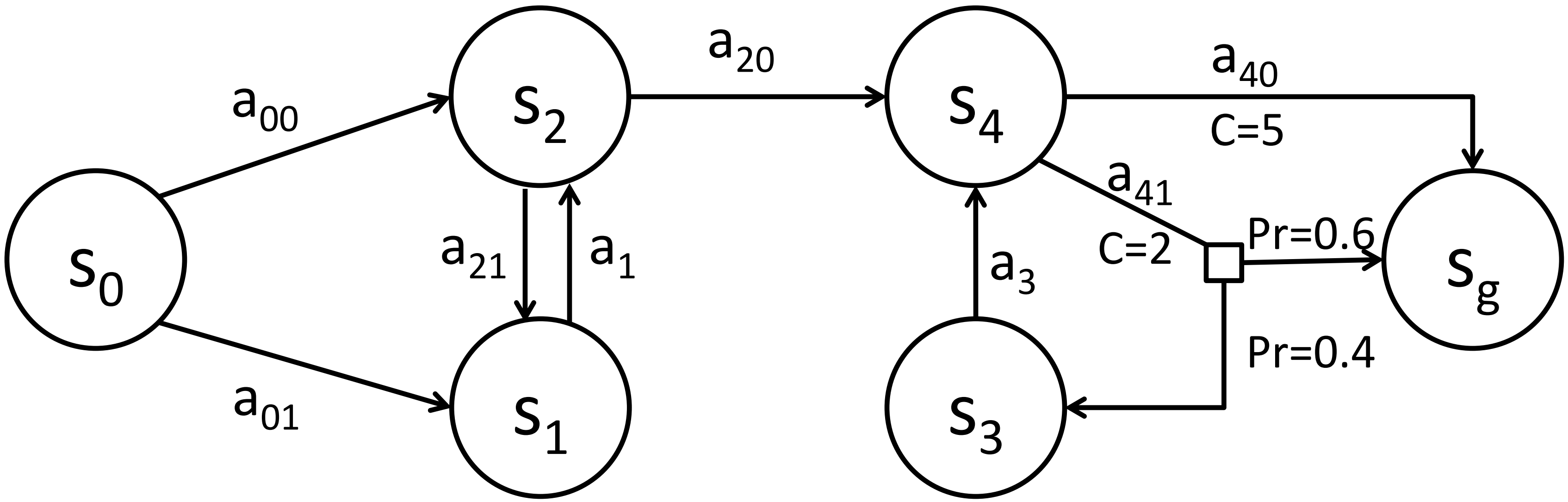
- Are all backups equally important?
- Can we avoid some backups?
- Can we schedule the backups more appropriately?

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?



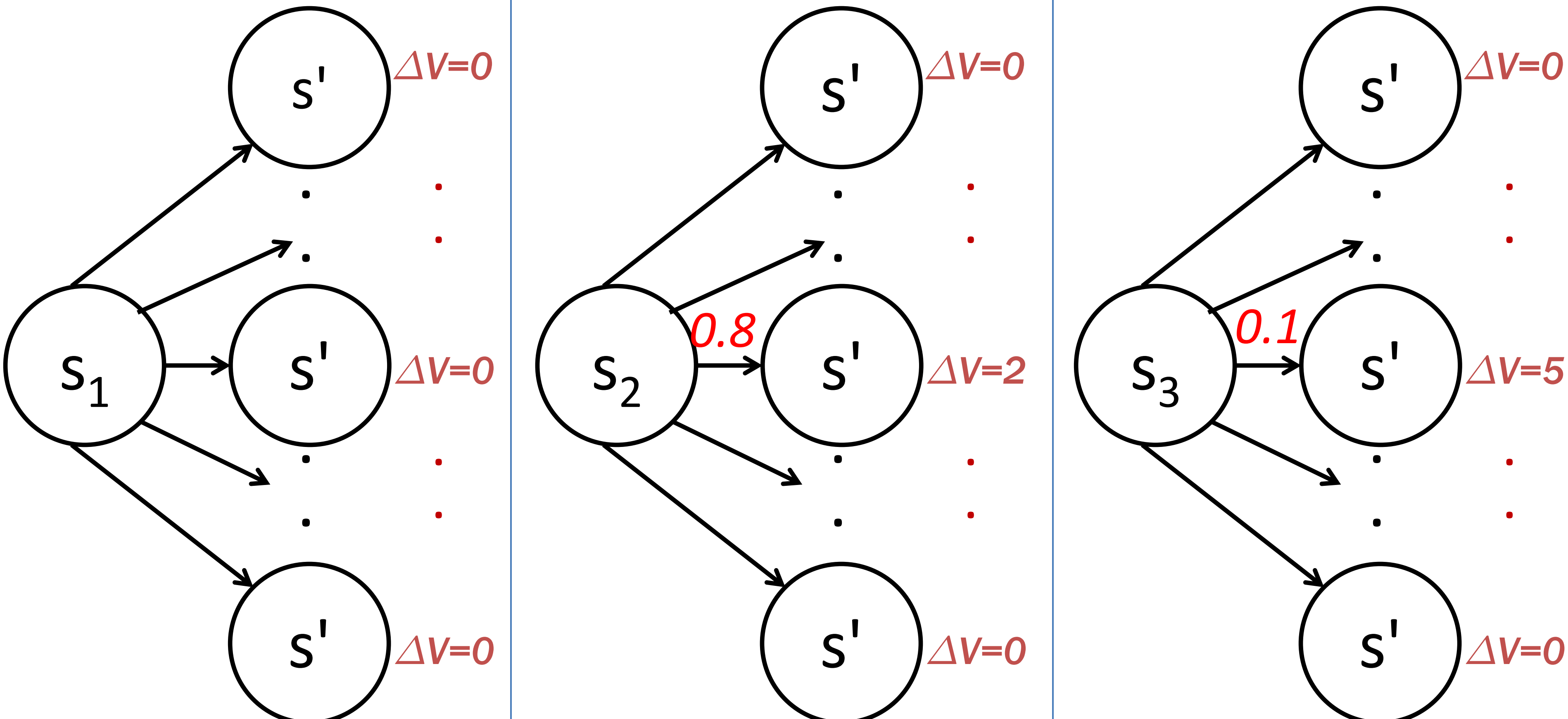
n	$V_n(s_0)$	$V_n(s_1)$	$V_n(s_2)$	$V_n(s_3)$	$V_n(s_4)$
0	3	3	2	2	1
1	3	3	2	2	2.8
2	**3**	**3**	3.8	3.8	**2.8**
3	4	4.8	**3.8**	**3.8**	3.52
4	4.8	4.8	4.52	4.52	**3.52**
5	5.52	5.52	**4.52**	**4.52**	3.808
20	5.99921	5.99921	4.99969	4.99969	3.99969

Asynch VI → Prioritized VI

initialize V
initialize priority queue q
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

Convergence?
Interleave synchronous VI iterations

Which state to prioritize?



s_1 is zero priority

s_2 is higher priority

s_3 is low priority

Prioritized Sweeping [Moore & Atkeson 93]

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

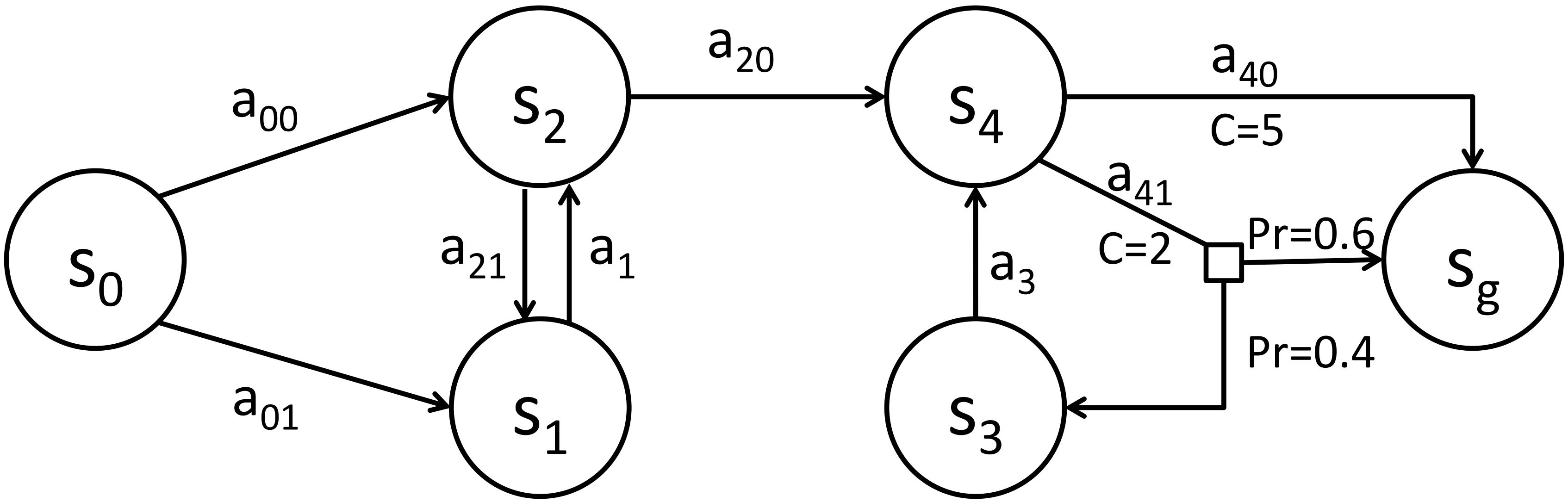
- Convergence [Li&Littman 08]

  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
Updates	3	3	**3.8**	**3.8**	2.8
Priority	**2**	**2**	0	0	**1.2**
Updates	3	4.8	3.8	3.8	2.8

Generalized Prioritized Sweeping [Andre et al 97]

$$\boxed{\text{priority}_{GPS2}(s) = Res^{V}(s)}$$

- Instead of *estimating* residual
  - compute it exactly
- Slightly different implementation
  - first backup then push!

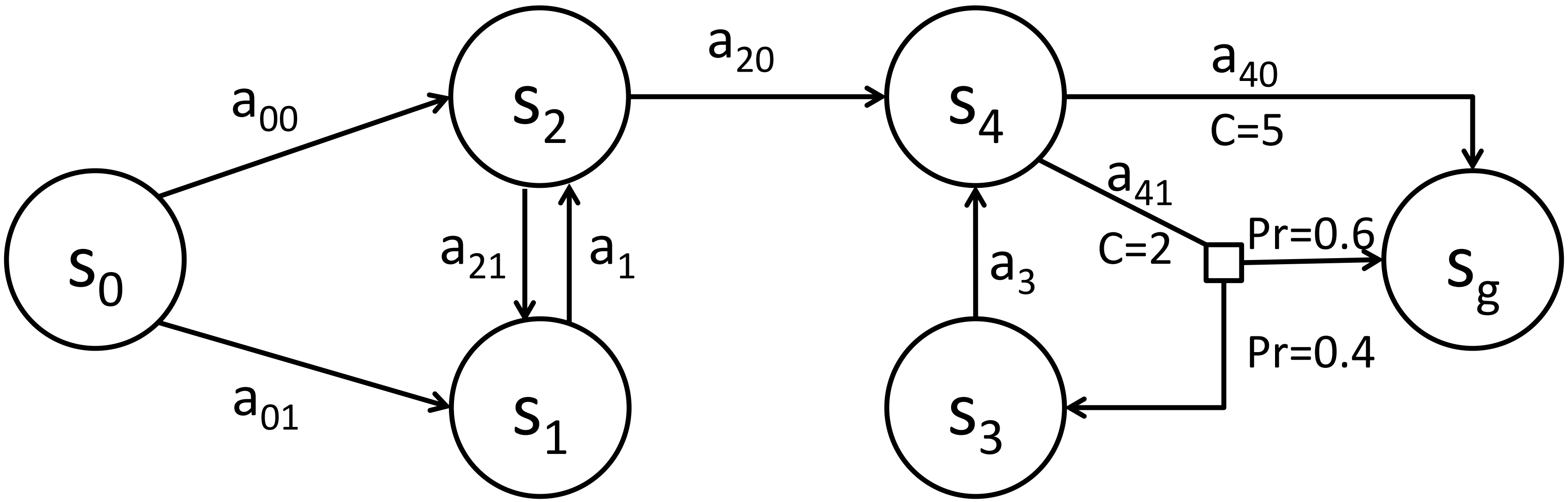
Intuitions

- Prioritized Sweeping
  - if a state's value changes prioritize its predecessors

- Myopic

- Which state should be backed up?
  - state closer to goal?
  - or farther from goal?

Useless Intermediate 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

Improved Prioritized Sweeping

[McMahan&Gordon 05]

$$\boxed{\mathrm{priority}_{IPS}(s) = \frac{Res^V(s)}{V(s)}}$$

- Intuition
  - Low V(s) states (closer to goal) are higher priority initially
  - As residual reduces for those states,
    - priority of other states increase
- A specific tradeoff
  - sometimes may work well
  - sometimes may not work that well

Tradeoff

- Priority queue increases information flow

- Priority queue adds overhead

- If branching factor is high
  - each backup may result in many priority updates!

Backward VI [Dai&Hansen 07]

- Prioritized VI without priority queue!
- Backup states in reverse order starting from goal
  - don't repeat a state in an iteration
  - other optimizations
    - (backup only states in current greedy subgraph)
- Characteristics
  - no overhead of priority queue
  - good information flow
  - doesn't capture the intuition:
    - higher states be converged before propagating further

Comments

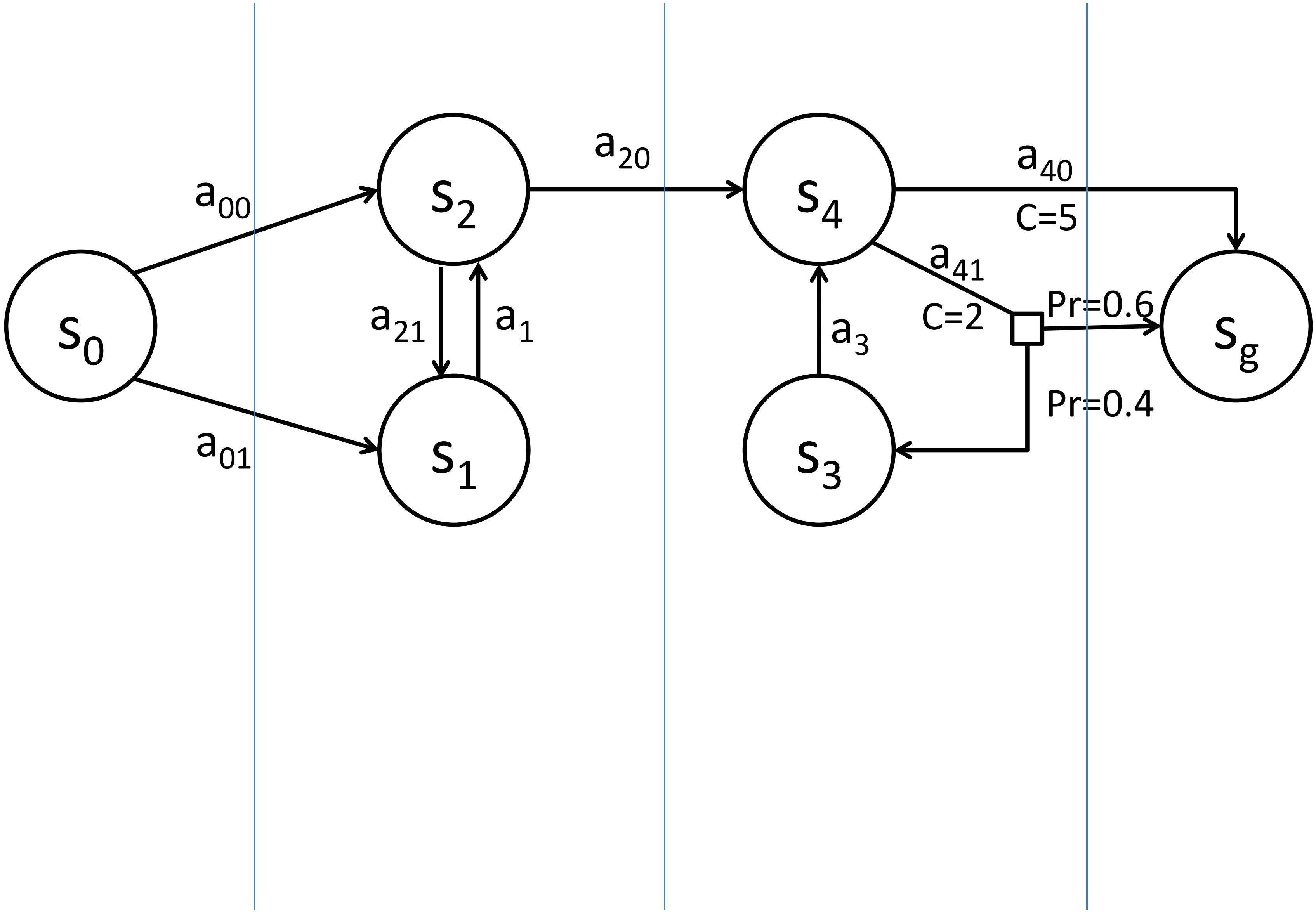
- Which algorithm to use?
  - Synchronous VI: when states highly interconnected
  - PS/GPS: sequential dependencies
  - IPS: specific way to tradeoff proximity to goal/info flow
  - BVI: better for domains with fewer predecessors

- Prioritized VI is a meta-reasoning algorithm
  - reasoning about what to compute!
  - costly meta-reasoning can hurt.

Uninformed Algorithms

- Definitions
- Fundamental Algorithms
- Prioritized Algorithms
- Partitioned Algorithms
- Other models

Partitioning of States



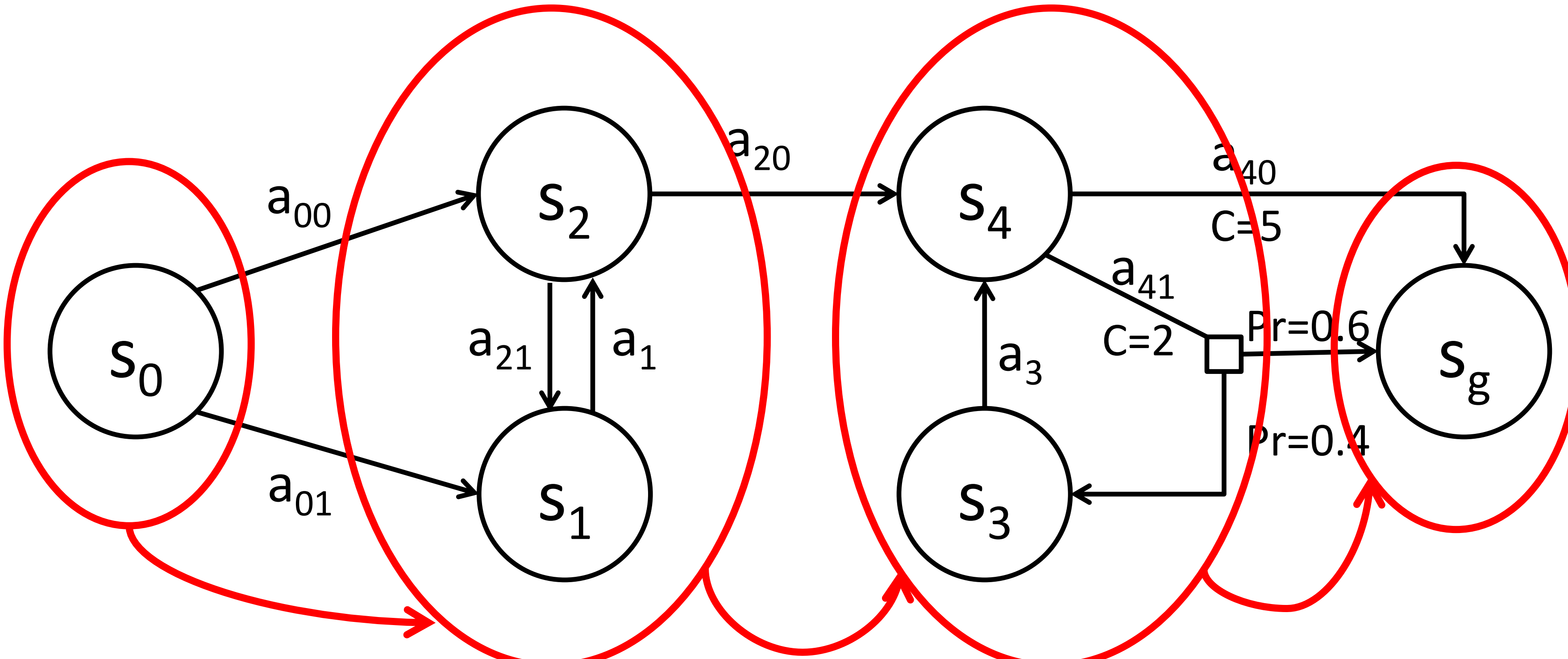
(General) Partitioned VI

initialize V arbitrarily
construct a partitioning of states $\mathfrak{P} = \{\mathfrak{p}_i\}$
(optional) **initialize** priorities for each $\mathfrak{p}_i$
repeat
    select a partition $\mathfrak{p}'$
    perform (potentially several) backups for all states in $\mathfrak{p}'$
    (optional) update priorities for all predecessor partitions of $\mathfrak{p}'$
until *termination*;
return greedy policy π^V

How to construct a partition?
How many backups to perform per partition?
How to construct priorities?

Topological VI [Dai&Goldsmith 07]

- Identify strongly-connected components
- Perform topological sort of partitions
- Backup partitions to ϵ-consistency: reverse top. order



Other Benefits of Partitioning

- External-memory algorithms
  - PEMVI [Dai etal 08, 09]
    - partitions live on disk
    - get each partition to the disk and backup all states
- Cache-efficient algorithms
  - P-EVA algorithm [Wingate&Seppi 04a]
- Parallelized algorithms
  - P3VI (Partitioned, Prioritized, Parallel VI) [Wingate&Seppi 04b]

Uninformed Algorithms

- Definitions
- Fundamental Algorithms
- Prioritized Algorithms
- Partitioned Algorithms
- Other models

Linear Programming for MDPs

$$\begin{array}{rl} \text{Variables} & V^*(s) \quad \forall s \in \mathcal{S} \\ \text{Maximize} & \sum_{s \in \mathcal{S}} \alpha(s) V^*(s) \\ \text{Constraints} & V^*(s) = 0 \quad \text{if } s \in \mathcal{G} \\ & V^*(s) \leq \sum_{s' \in \mathcal{S}} [\mathcal{C}(s,a,s') + \mathcal{T}(s,a,s') V^*(s')] \end{array}$$

- |S| variables
- |S||A| constraints
  - too costly to solve!

Infinite-Horizon Discounted-Reward MDPs

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V^*(s')]$$

- VI/PI work even better than SSPs!!
  - PI does not require a "proper" policy
  - Error bounds are tighter
    - Example. VI error bound: $|V^*(s)-V^{\pi}(s)| < 2\epsilon\gamma/(1-\gamma)$
  - We can bound #iterations
    - polynomial in $|S|$, $|A|$ and $1/(1-\gamma)$

Finite-Horizon MDPs

$$
\begin{aligned}
V^*(s,t) &= 0 \quad \text{if } t > L \\
&= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \left[\mathcal{R}(s,a,s') + V^*(s', t+1)\right]
\end{aligned}
$$

- Finite-Horizon MDPs are acyclic!
  - There exists an optimal backup order
    - $t=T_{max}$ to 0
  - Returns optimal values (not just ϵ-consistent)
  - Performs one backup per augmented state

Summary of Uninformed Algorithms

- Definitions
- Fundamental Algorithms
  - Bellman Equations is the key
- Prioritized Algorithms
  - Different priority functions have different benefits
- Partitioned Algorithms
  - Topological analysis, parallelization, external memory
- Other models
  - Other popular models similar

Outline of the Tutorial

- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

HEURISTIC SEARCH ALGORITHMS

Heuristic Search Algorithms

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

Limitations of VI/PI/Extensions

- Scalability
  - Memory linear in size of state space
  - Time at least polynomial or more
- Polynomial is good, no?
  - state spaces are usually huge.
    - Think PPDDL.
  - 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

- SSP (as before) with an additional start state s_0
  - denoted by SSP_{s0}

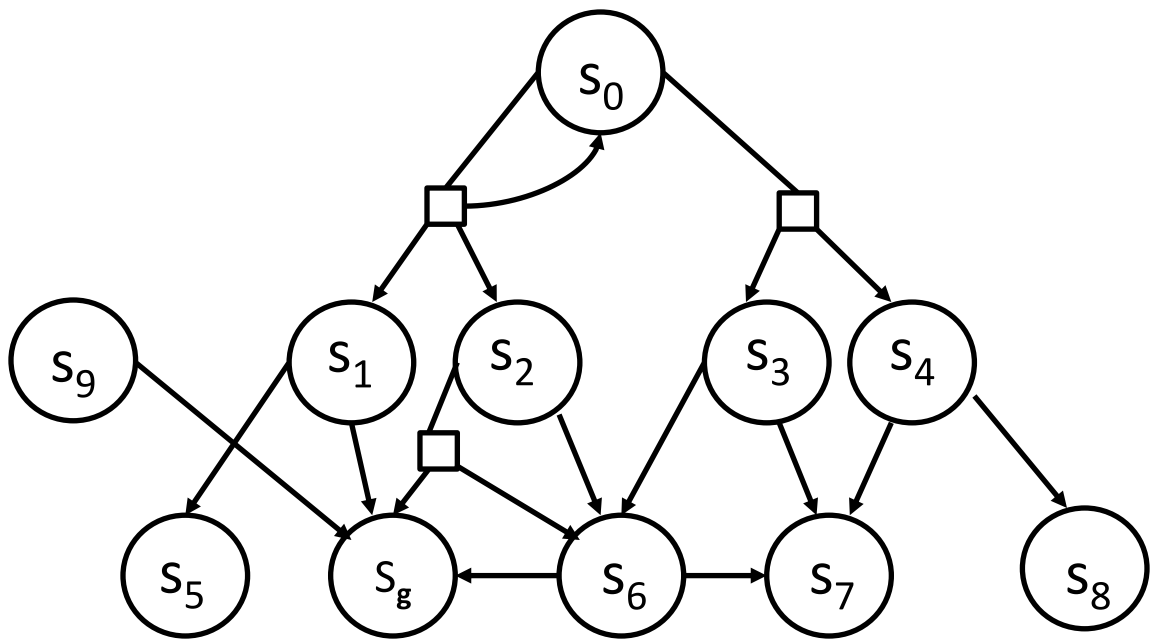
- What is the solution to an SSP_{s0}
- Policy ($S \rightarrow A$)?
  - are states that are not reachable from s_0 relevant?
  - states that are never visited (even though reachable)?

Partial Policy

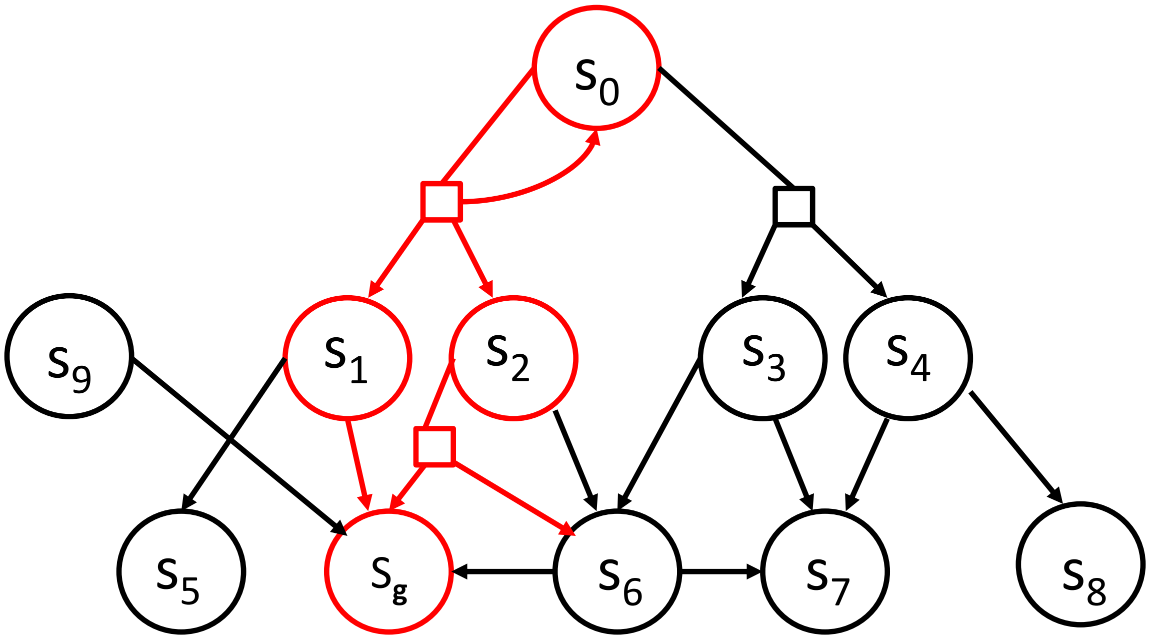
- Define *Partial policy*
  - π: S' $\rightarrow$ *A*, where S'$\subseteq$ *S*

- 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



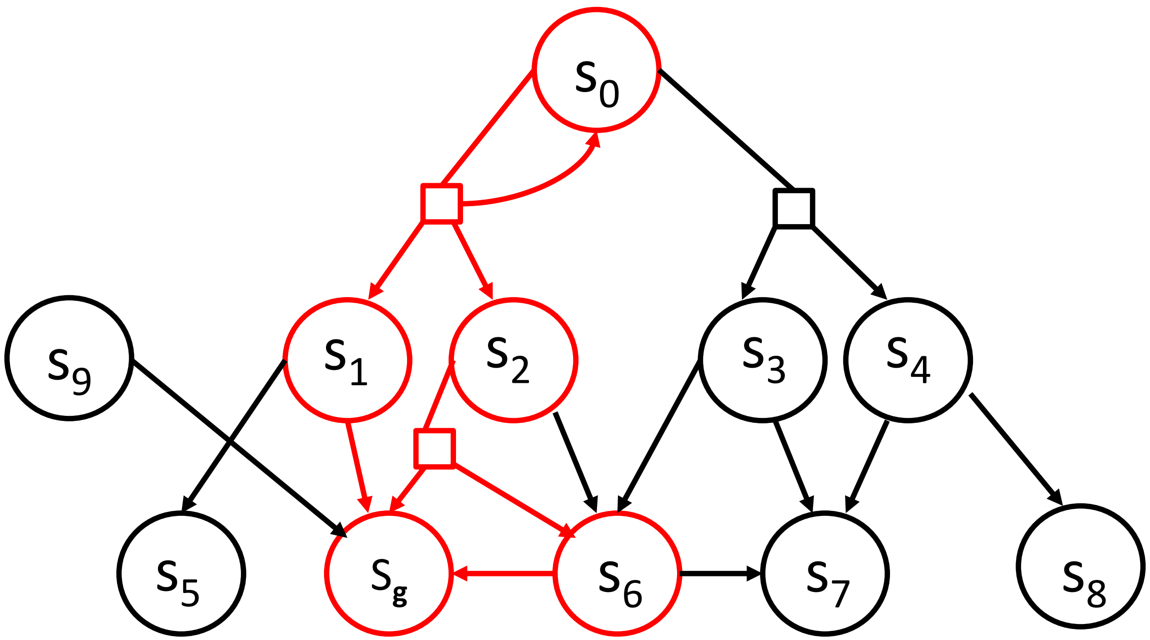
Partial policy closed wrt s_0



Is this policy closed wrt s_0?

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

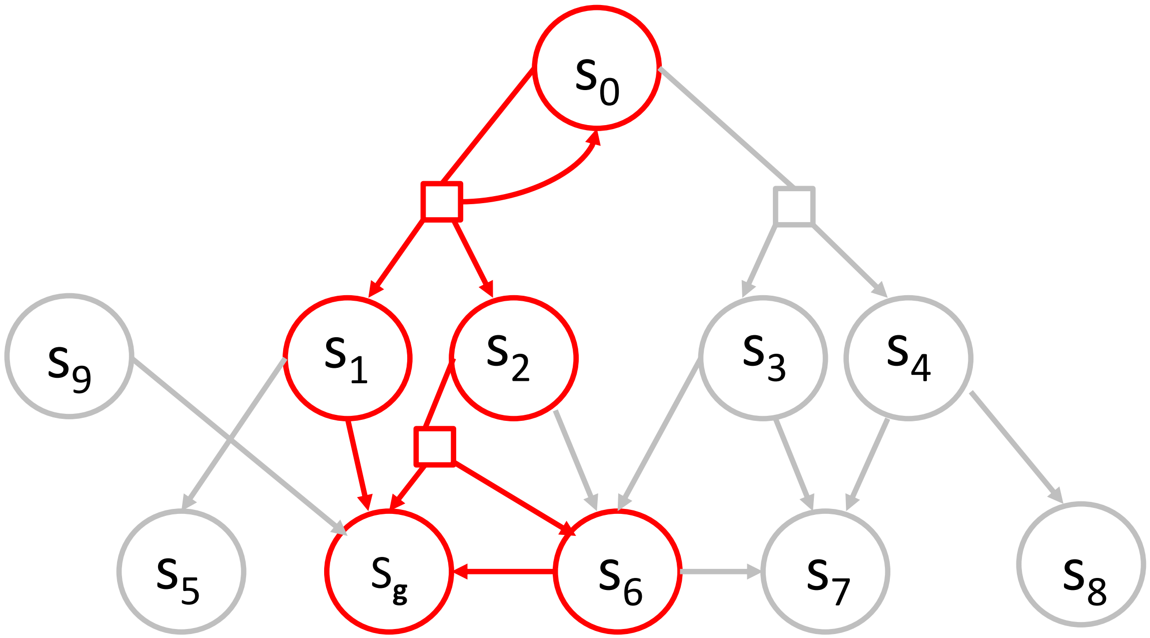
Partial policy closed wrt s_0



Is this policy closed wrt s_0?

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

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
  - $h(s) \leq V^*(s)$

Heuristic Search Algorithms

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

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 $V(s)$
end
return a π^V

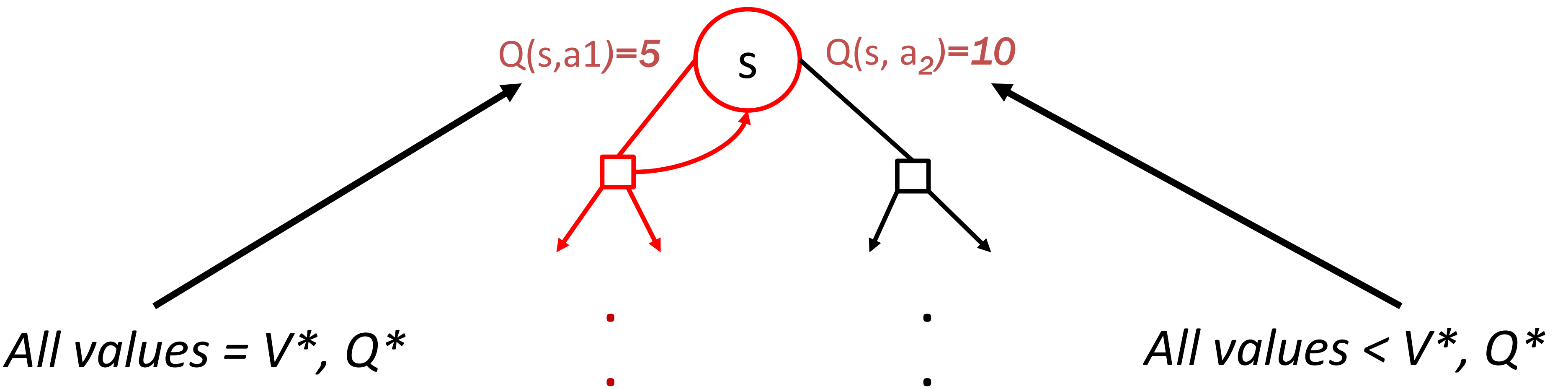
(perform Bellman backups)

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

F&R and Monotonicity

- $V_k \leq_p V^* \Rightarrow V_n \leq_p V^*$ (V_n monotonic from below)
  - If h is admissible: $V_0 = h(s) \leq_p V^*$

    $\Rightarrow V_n \leq_p V^*$ ($\forall n$)



$Q^*(s,a_1) < Q(s,a_2) < Q^*(s,a_2)$

a_2 can't be optimal

Heuristic Search Algorithms

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

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

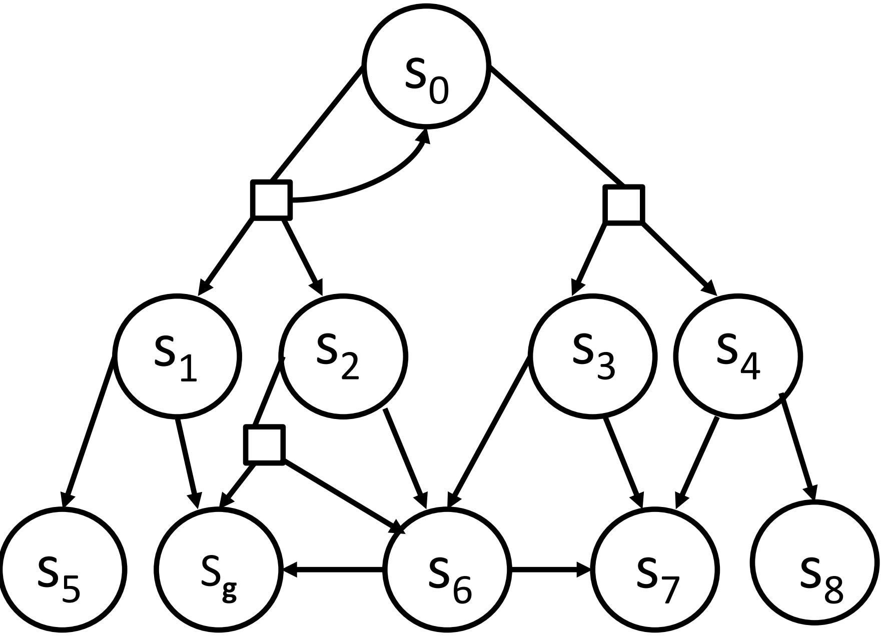
repeat

- 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 PI on this subset
- recompute the greedy graph

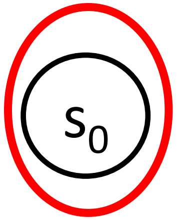
until greedy graph has no fringe ~~& residuals in greedy graph small~~

output the greedy graph as the final policy

LAO*

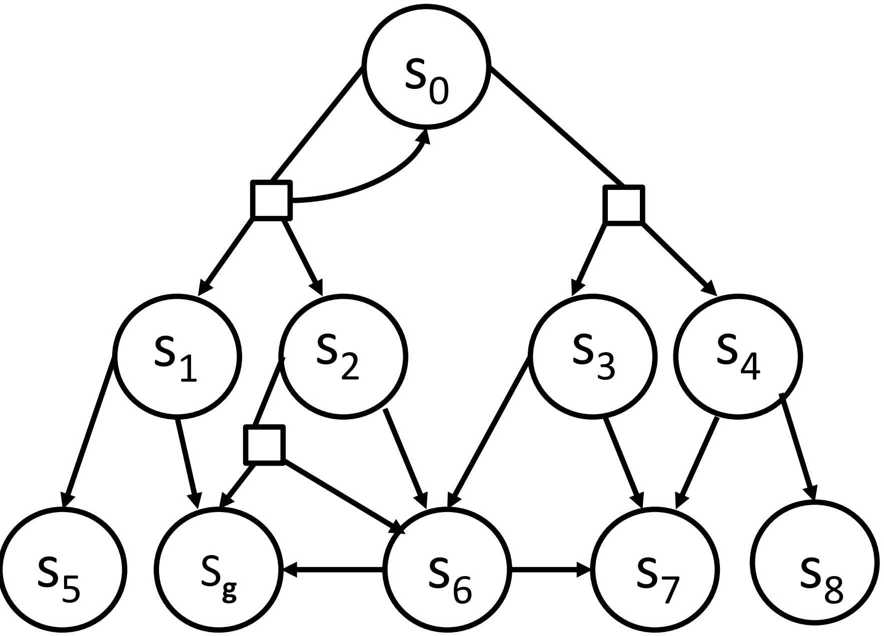


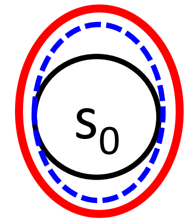
add s_0 in the fringe and in greedy graph



$V(s_0) = h(s_0)$

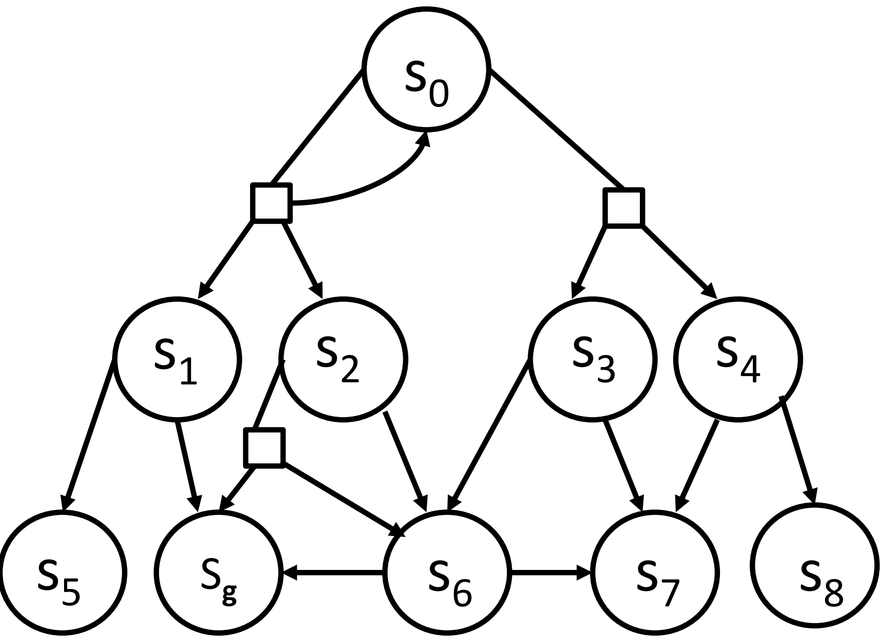
LAO*

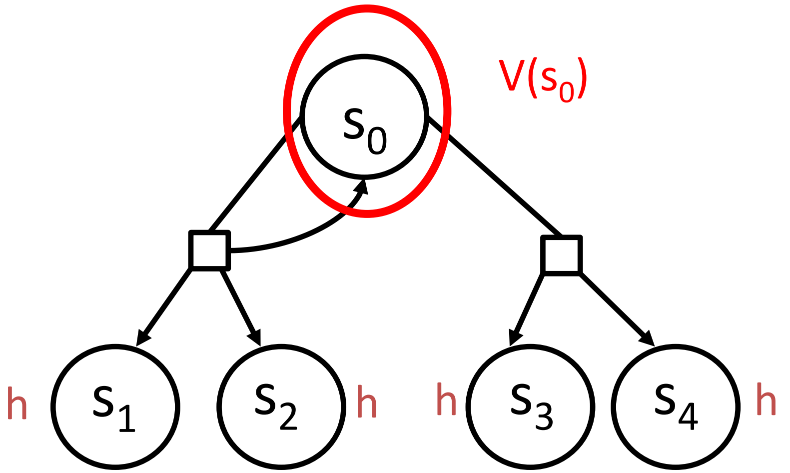




FIND: expand some states on the fringe (in greedy graph)

LAO*





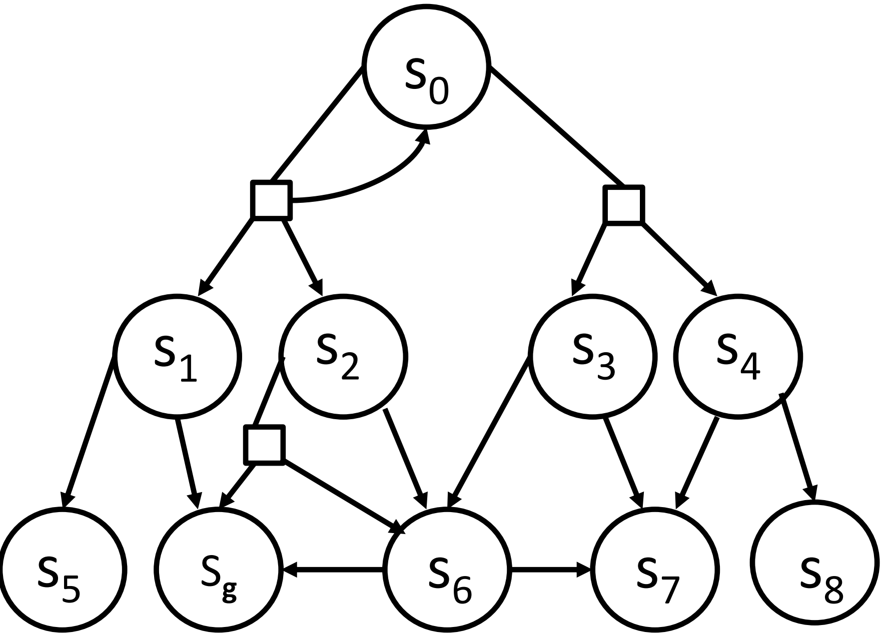
FIND: expand some states on the fringe (in greedy graph)

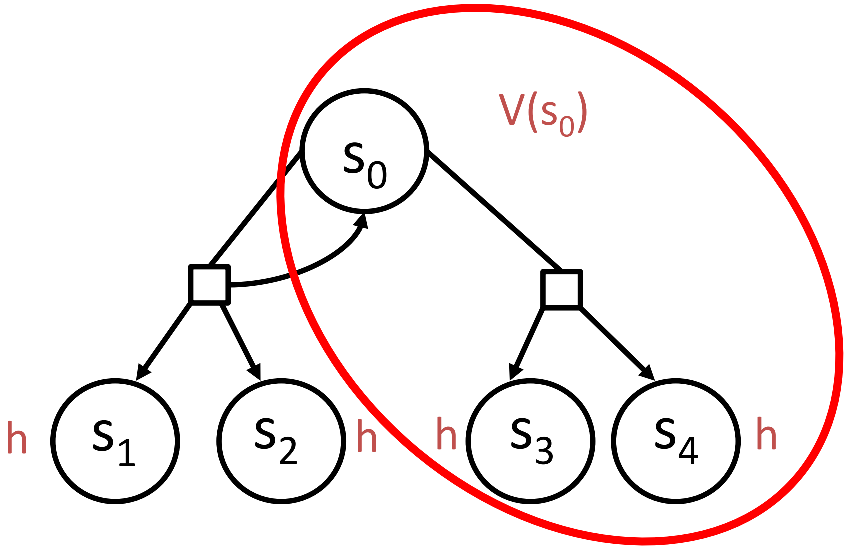
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

LAO*





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

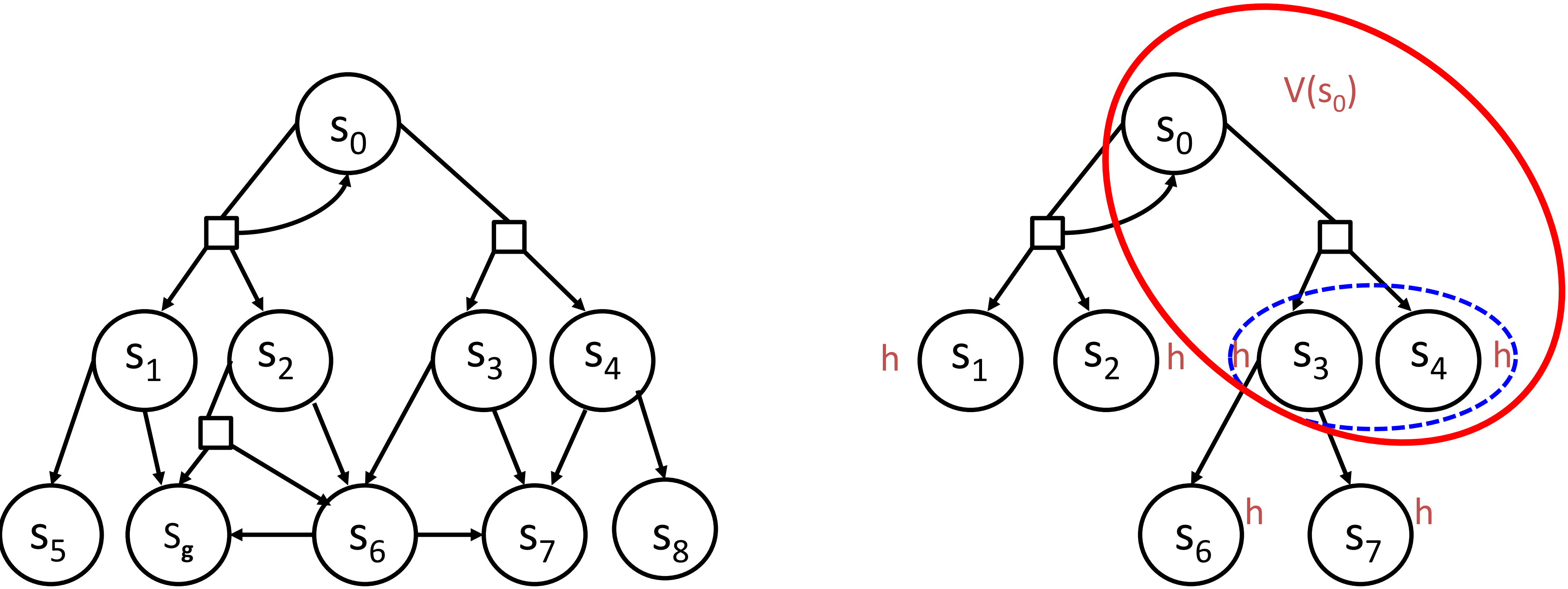
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

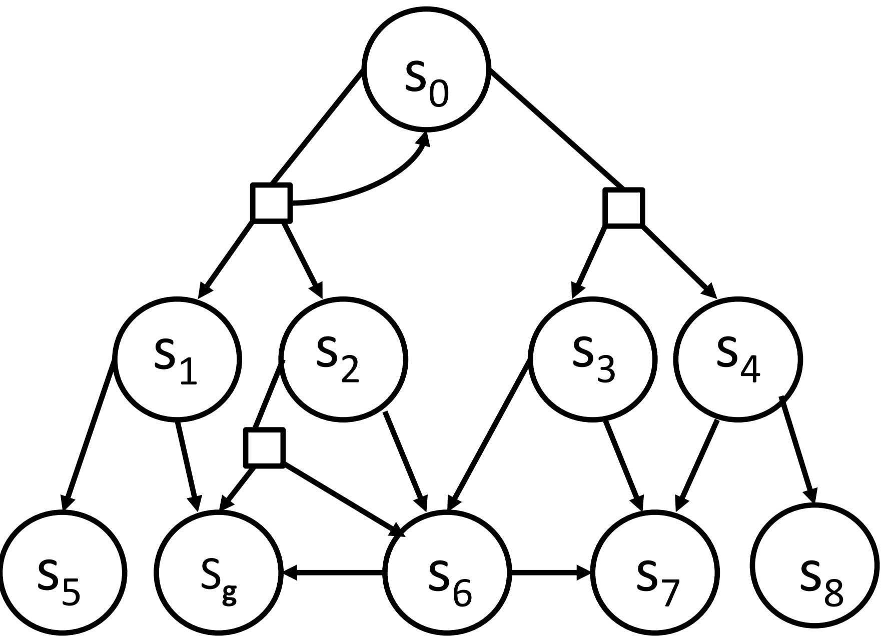
initialize all new states by their heuristic value

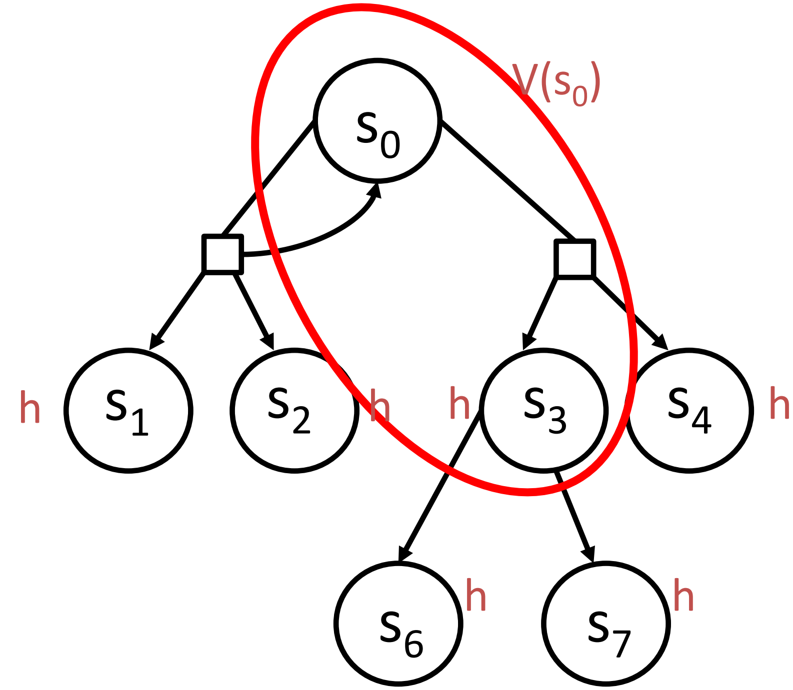
subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*





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

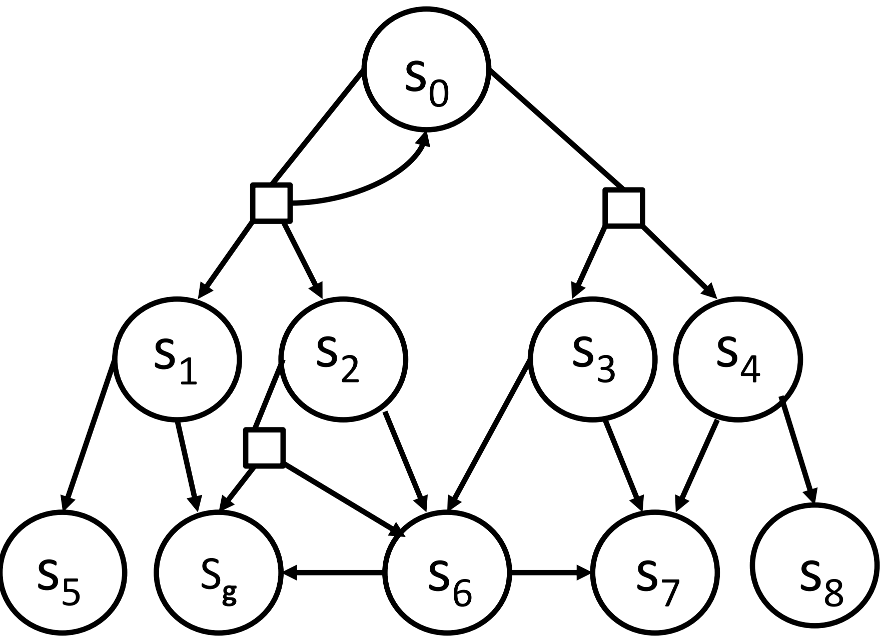
initialize all new states by their heuristic value

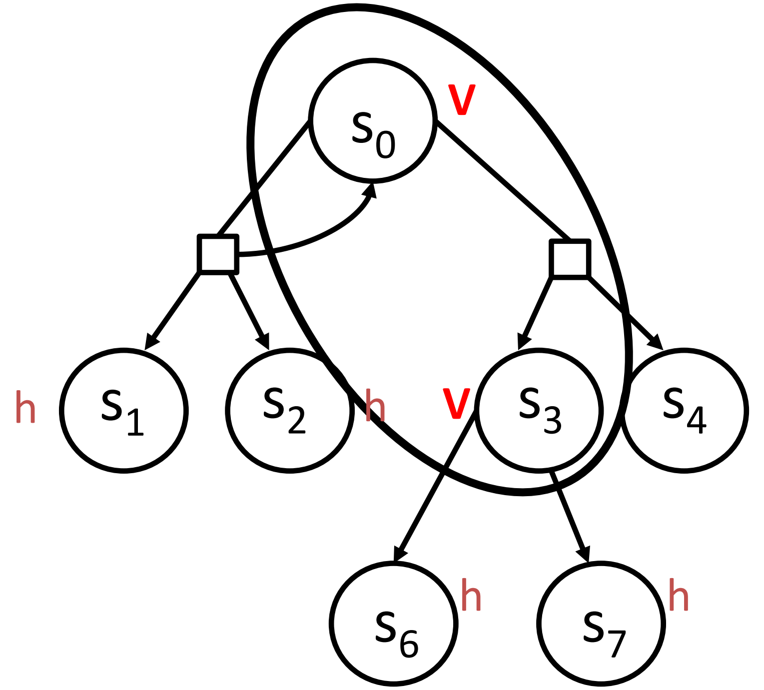
subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*





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

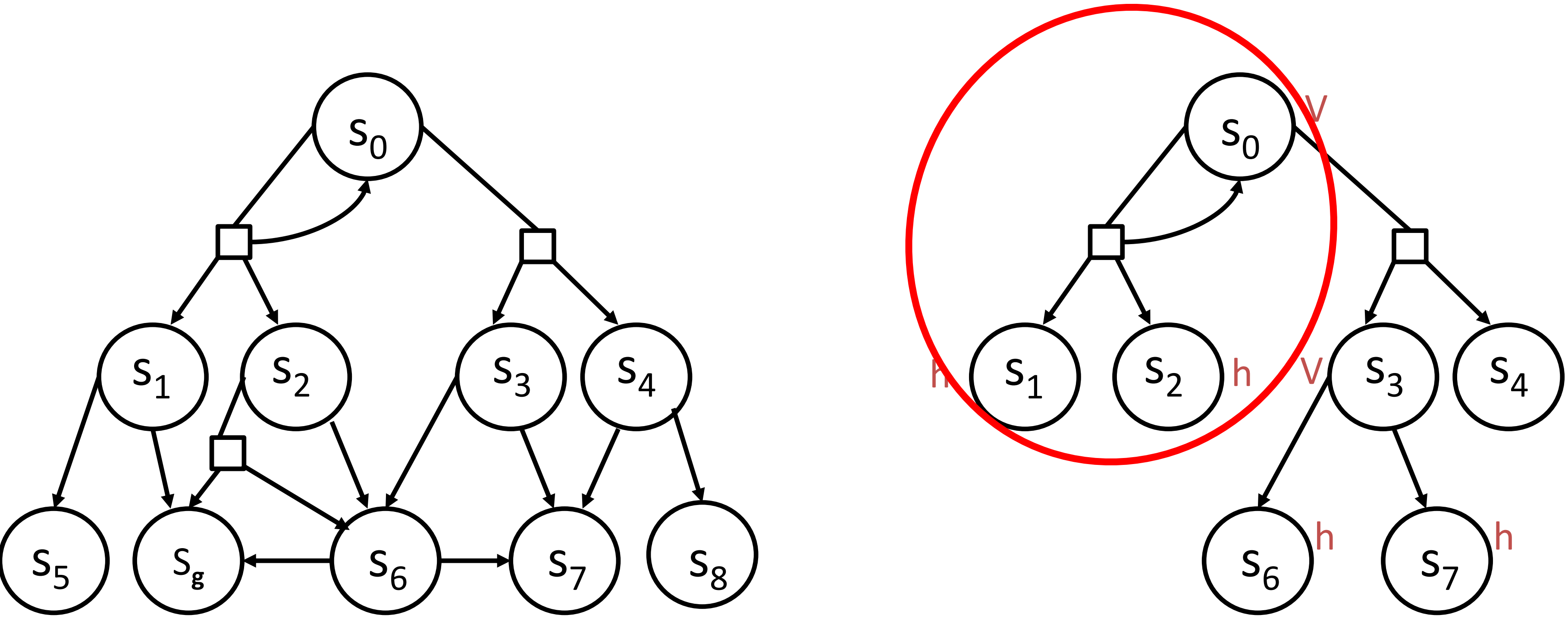
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

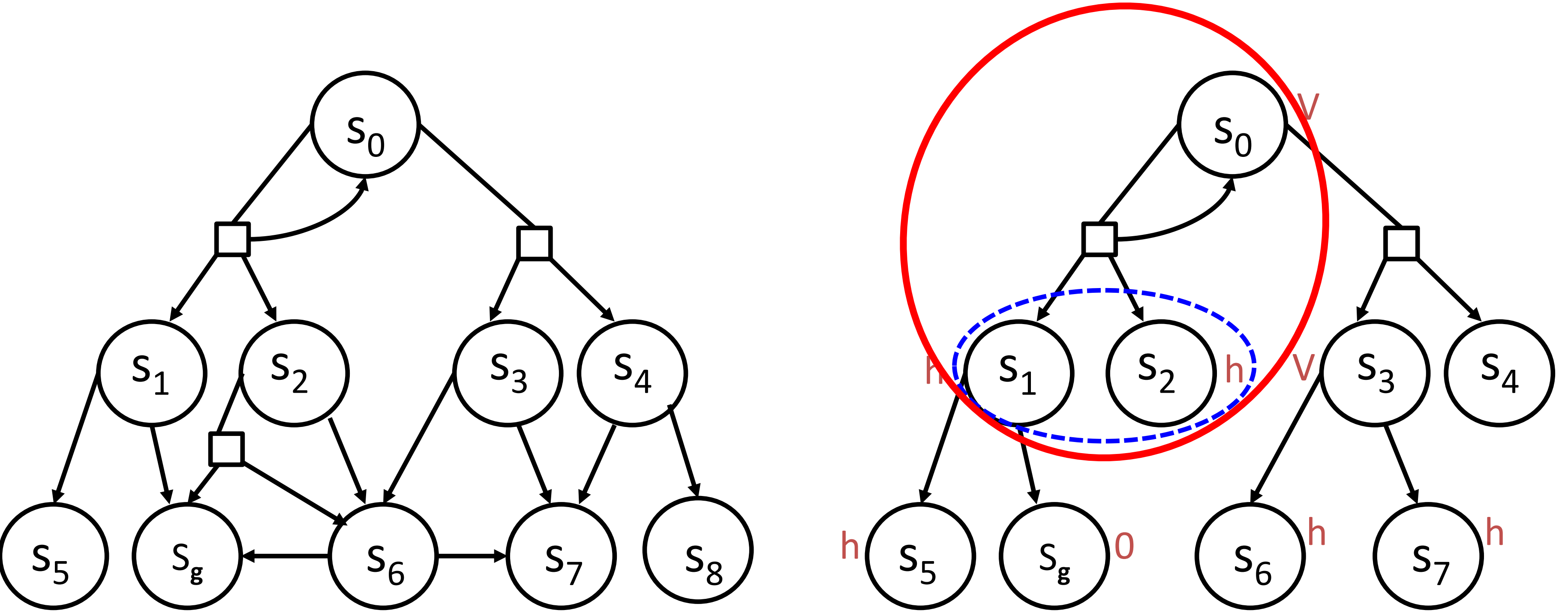
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

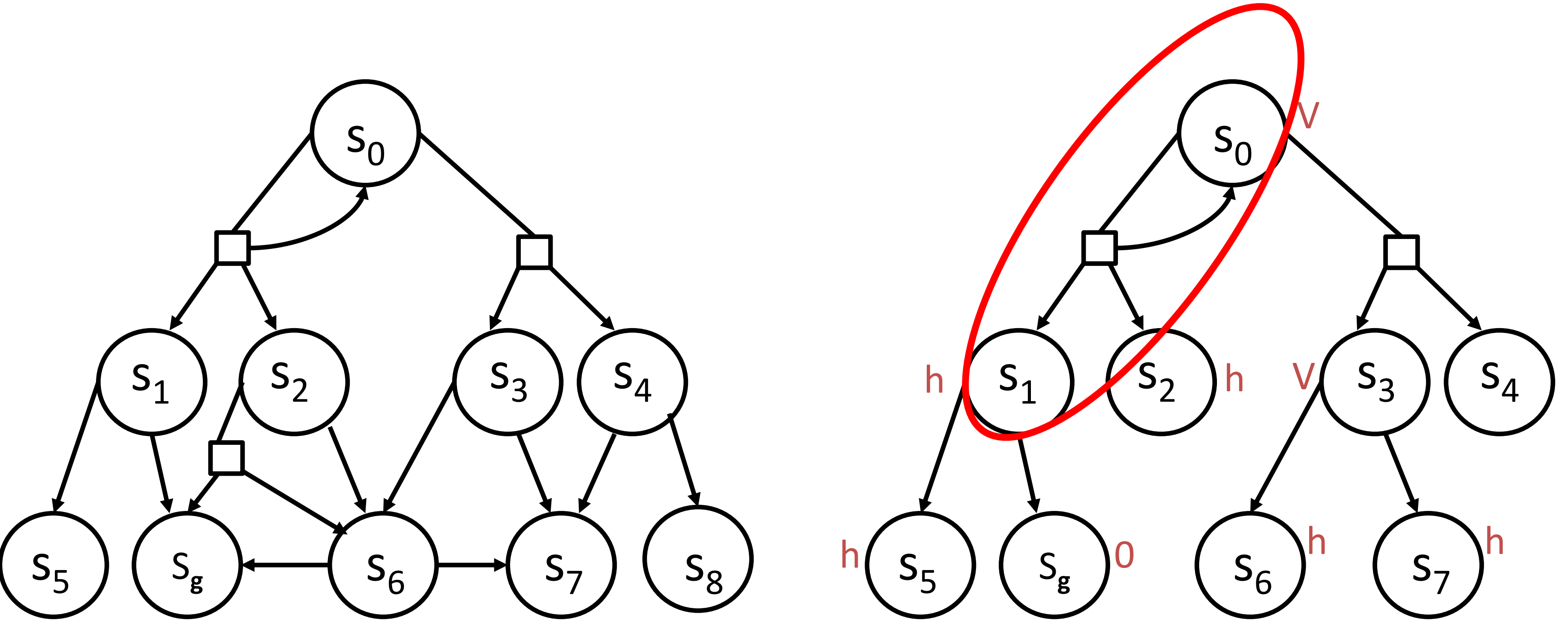
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

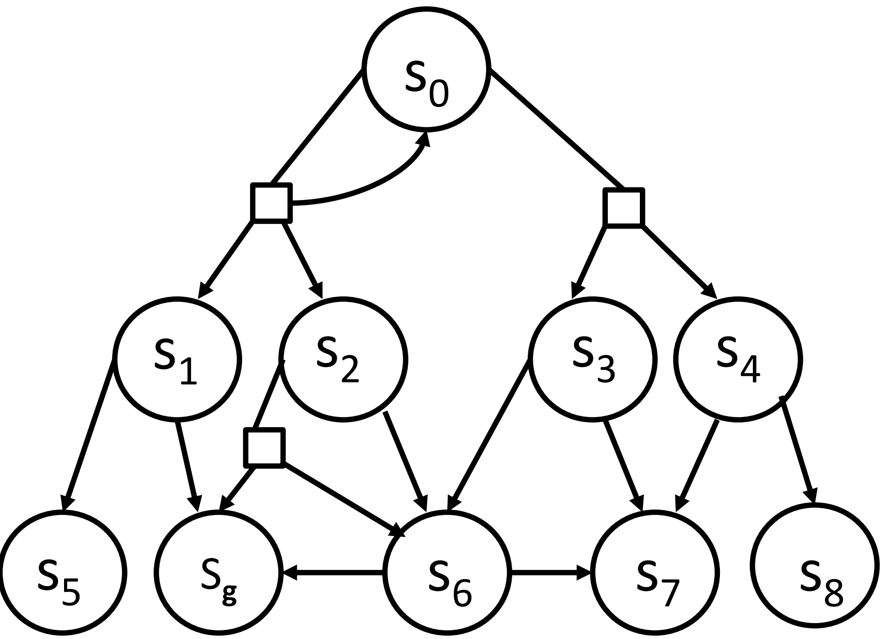
initialize all new states by their heuristic value

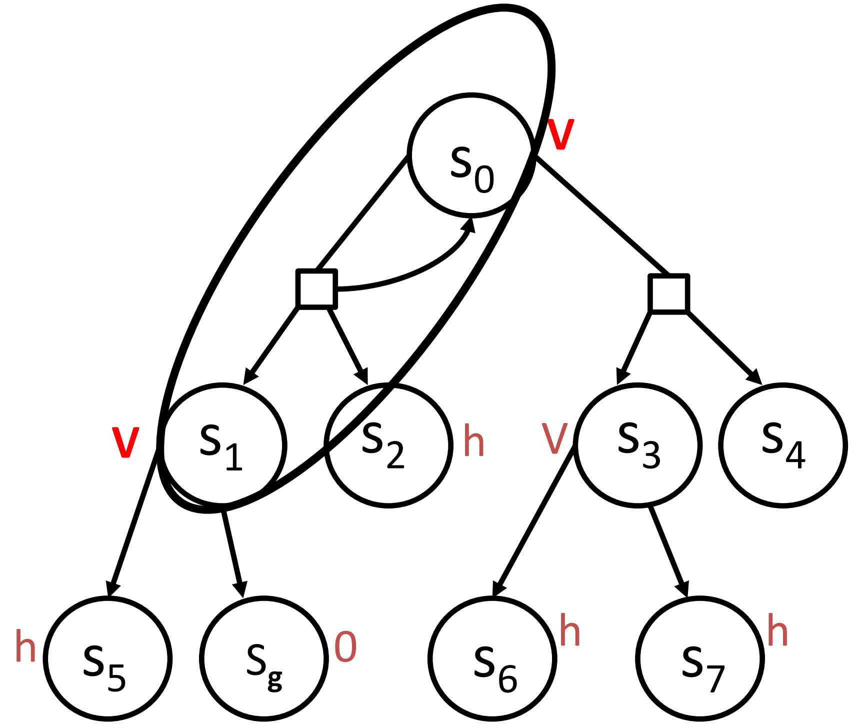
subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*





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

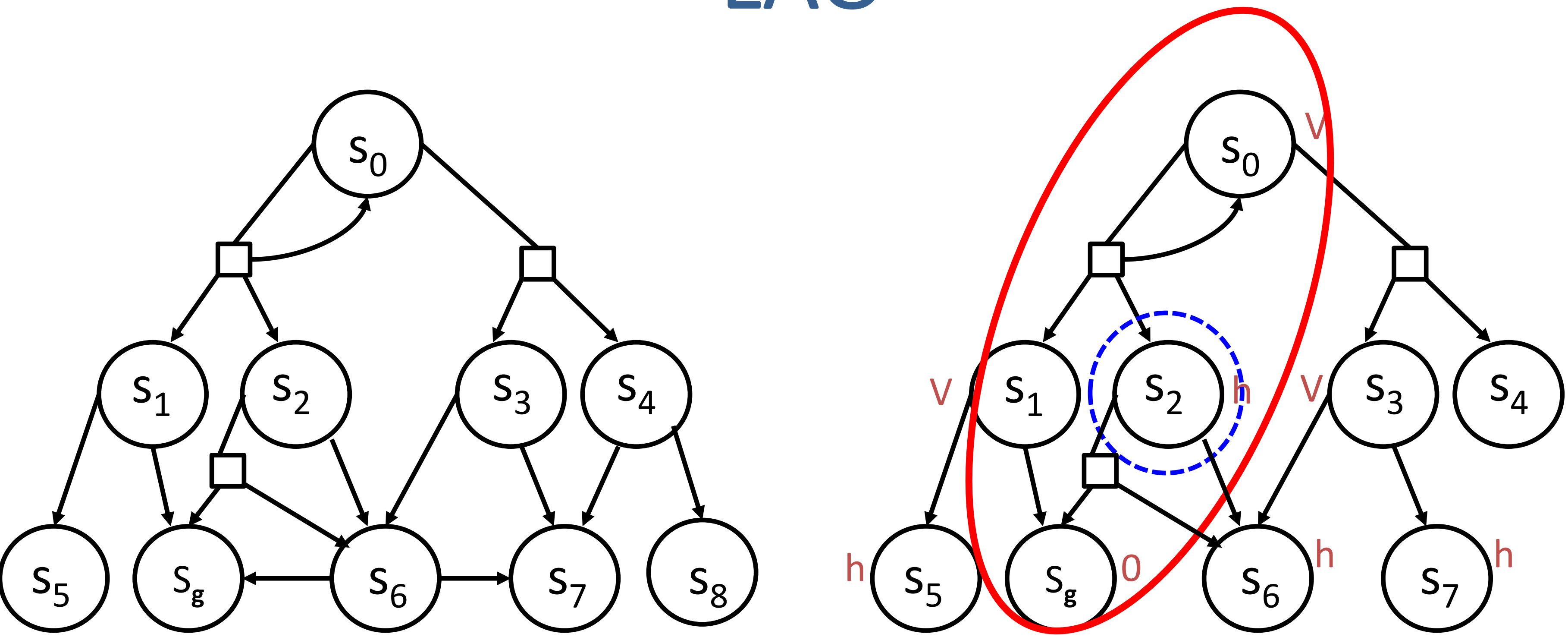
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

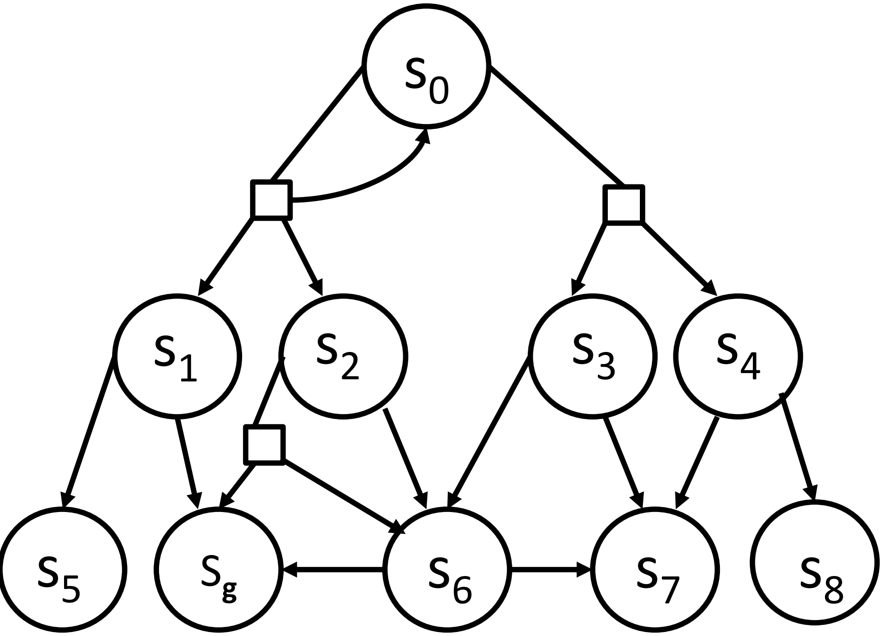
initialize all new states by their heuristic value

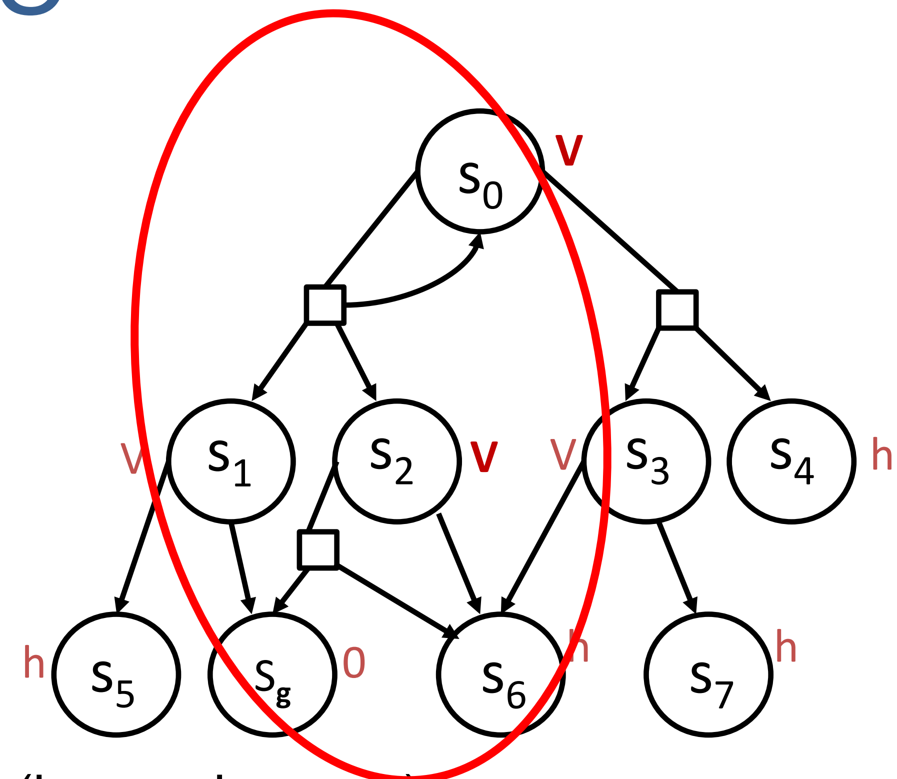
subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*





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

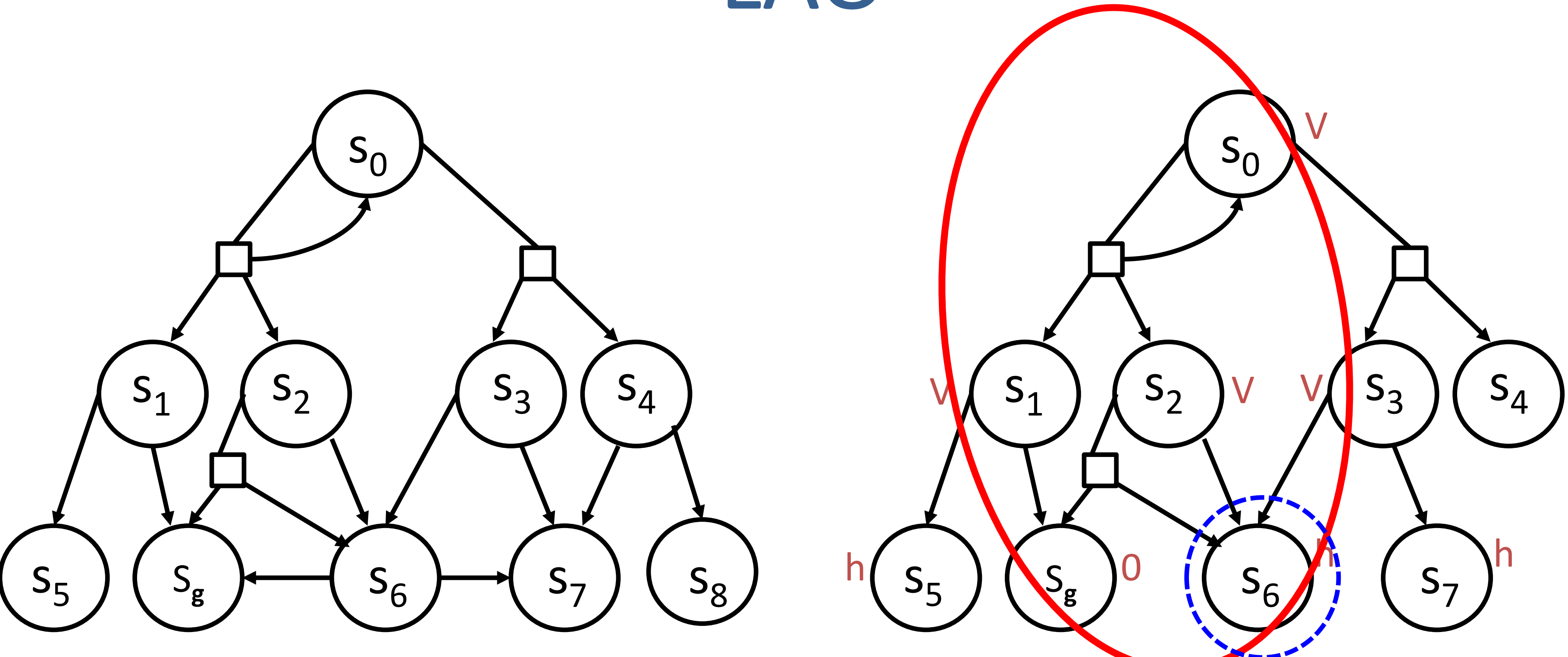
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

perform PI on this subset

recompute the greedy graph

LAO*



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

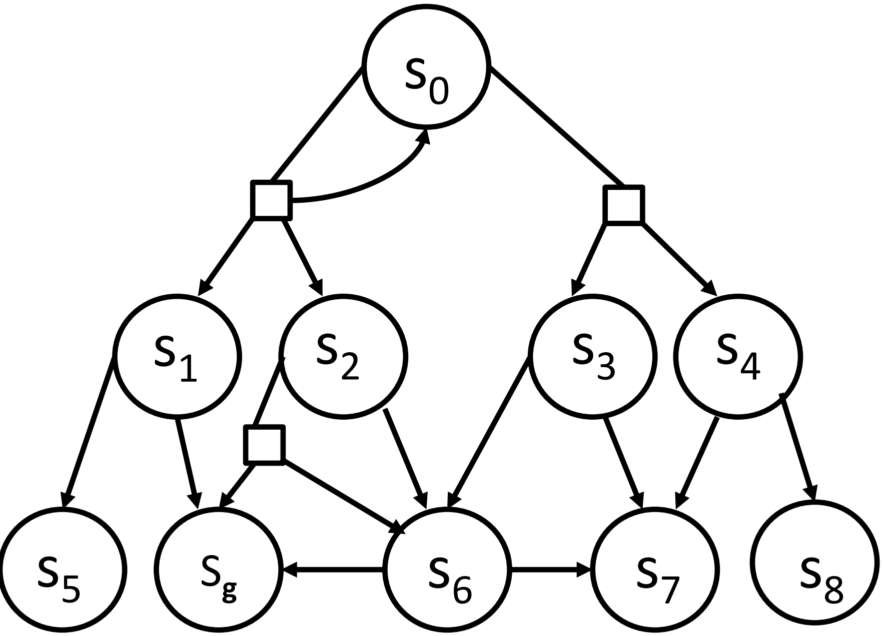
initialize all new states by their heuristic value

subset = all states in expanded graph that can reach s

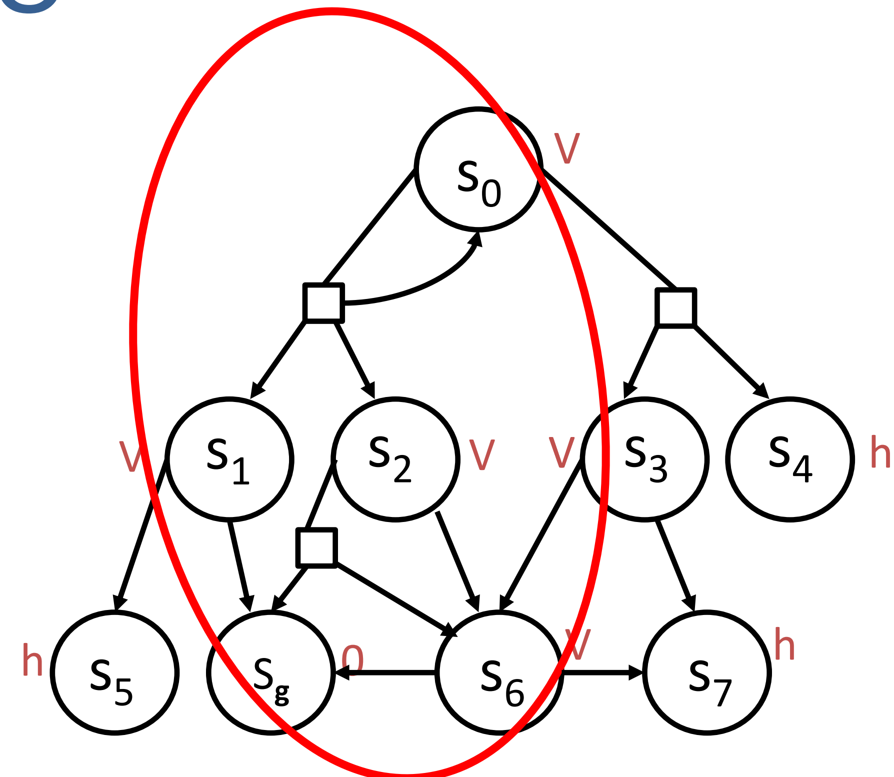
perform PI on this subset

recompute the greedy graph

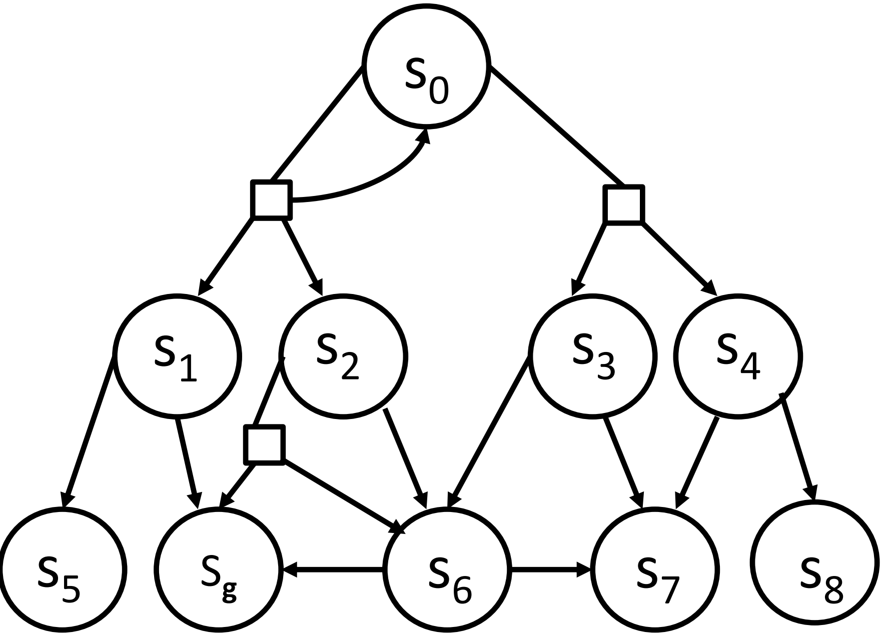
LAO*

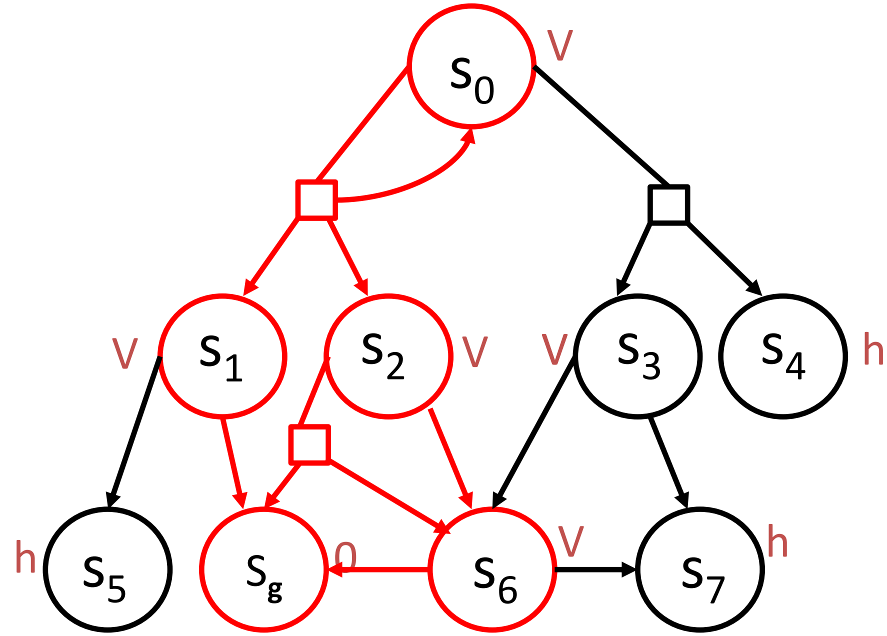


output the greedy graph as the final policy



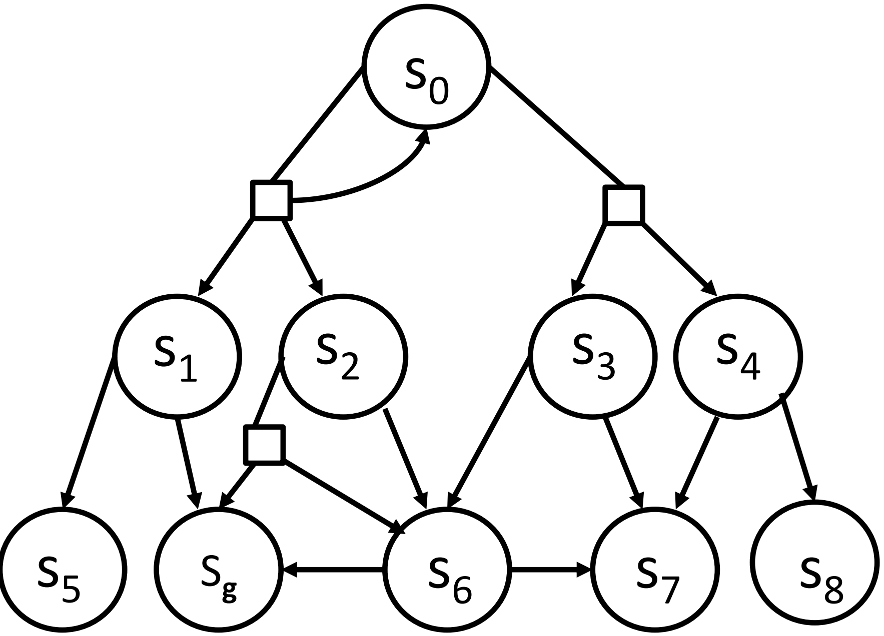
LAO*

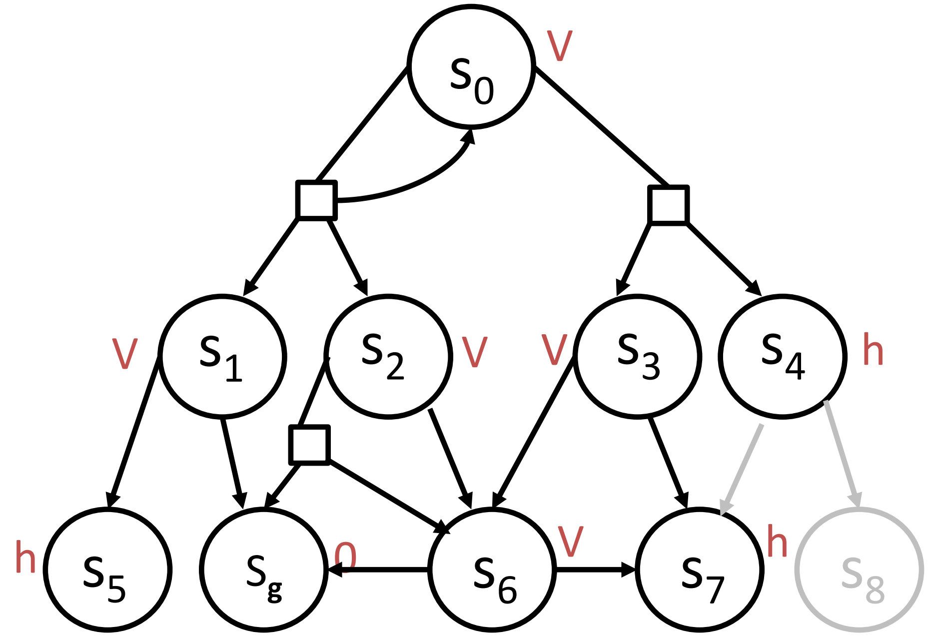




output the greedy graph as the final policy

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

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 PI on this subset
- recompute the greedy graph

lot of computation

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 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

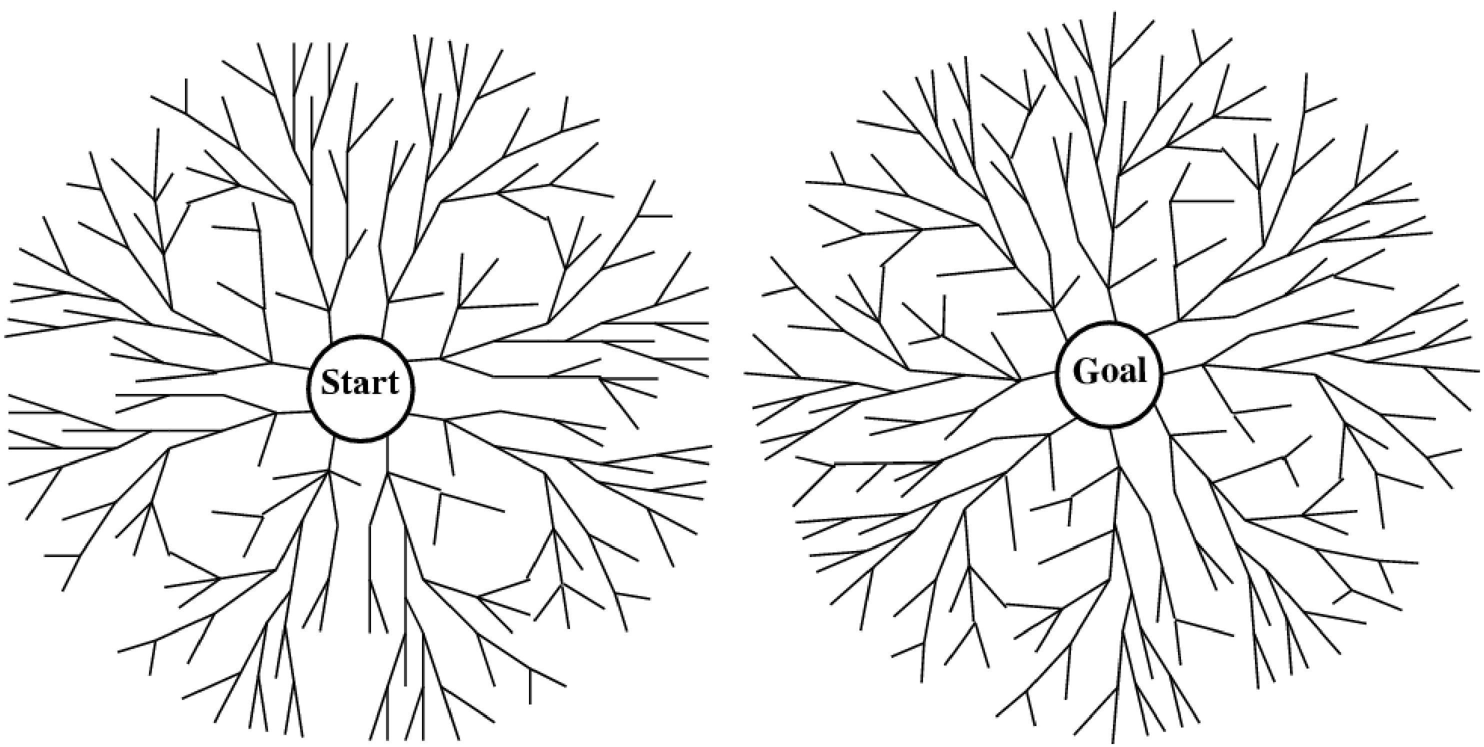
output the greedy graph as the final policy

Reverse LAO* [Dai&Goldsmith 06]

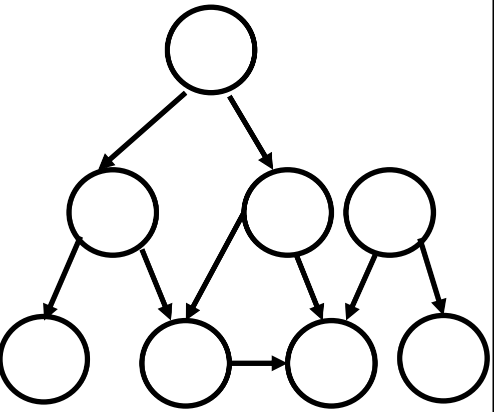
- LAO* may spend huge time until a goal is found
  - guided only by s_0 and heuristic
- LAO* in the reverse graph
  - guided only by goal and heuristic
- Properties
  - Works when 1 or handful of goal states
  - May help in domains with small fan in

Bidirectional LAO* [Dai&Goldsmith 06]

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



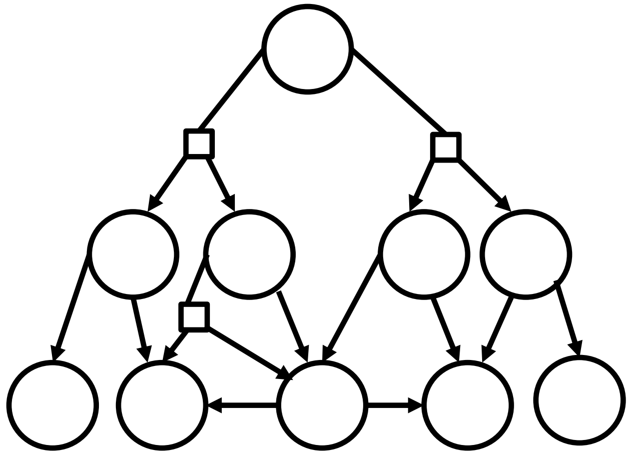
A* → LAO*



regular graph

soln:(shortest) path

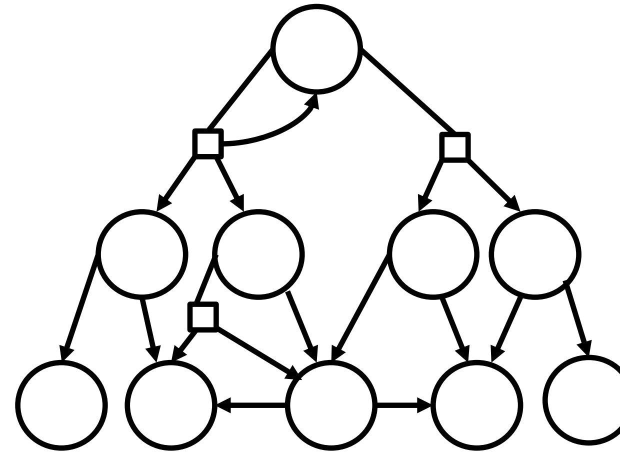
A*



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 able to make effective use of reachability information!

AO* for Acyclic MDPs [Nilsson 71]

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
- a single backup pass from fringe states to start state
- recompute the greedy graph

until greedy graph has no fringe

output the greedy graph as the final policy

Heuristic Search Algorithms

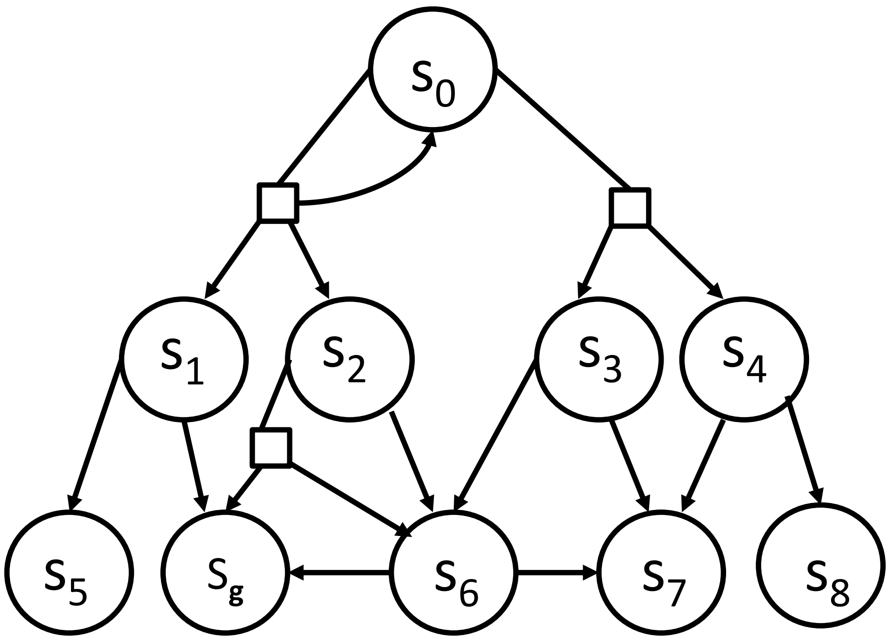
- Definitions
- Find & Revise Scheme.
- LAO* and Extensions
- RTDP and Extensions
- Other uses of Heuristics/Bounds
- Heuristic Design

Real Time Dynamic Programming

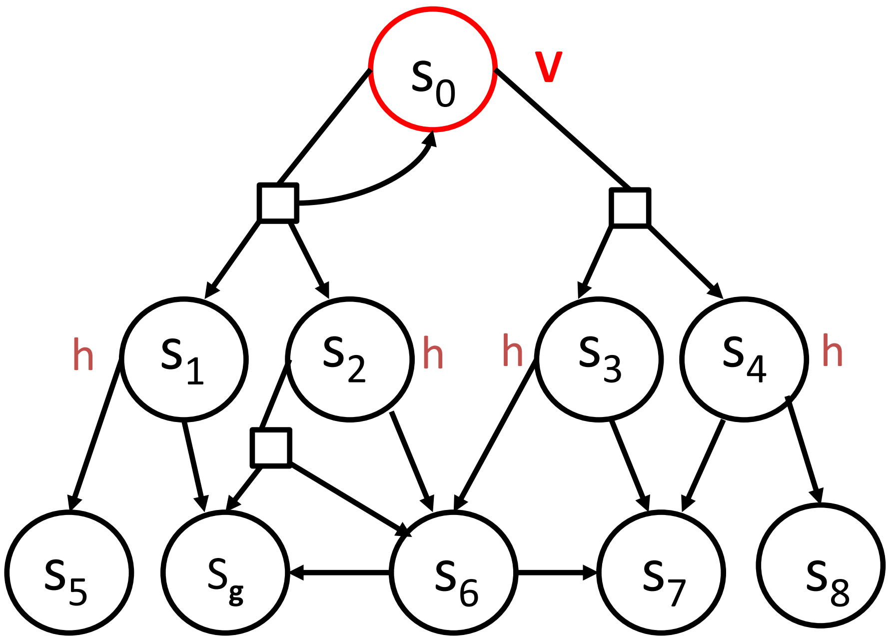
[Barto et al 95]

- 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 $\rightarrow \infty$

Trial



Trial



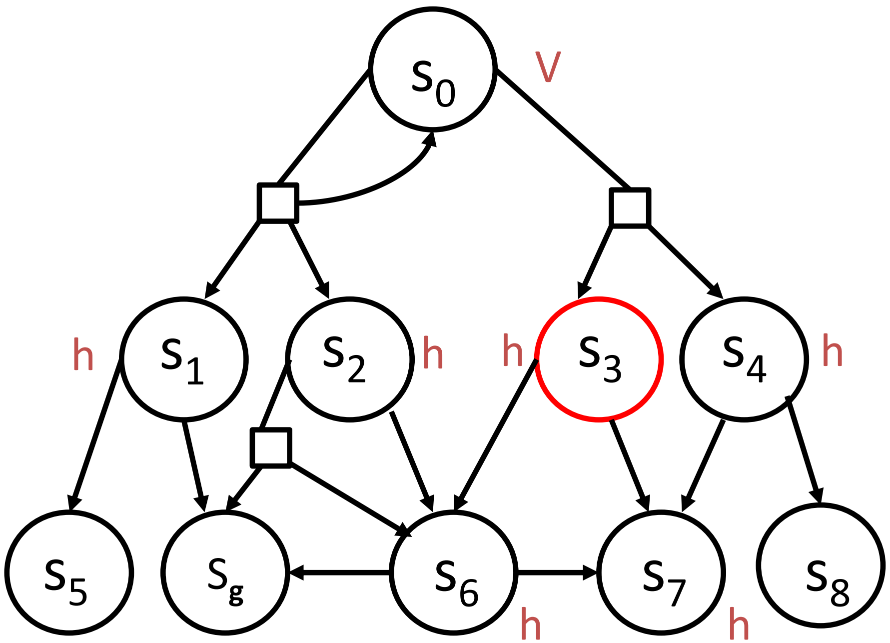
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



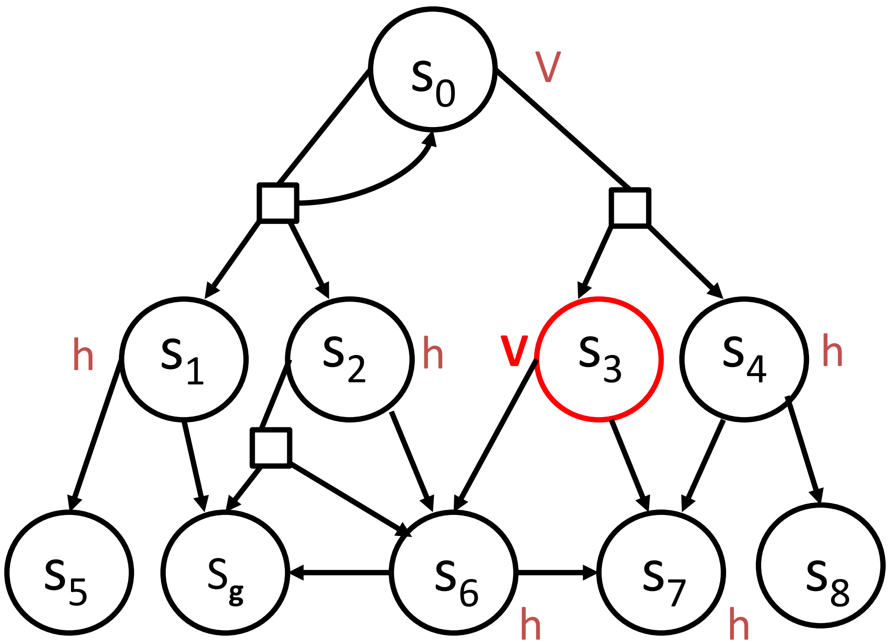
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



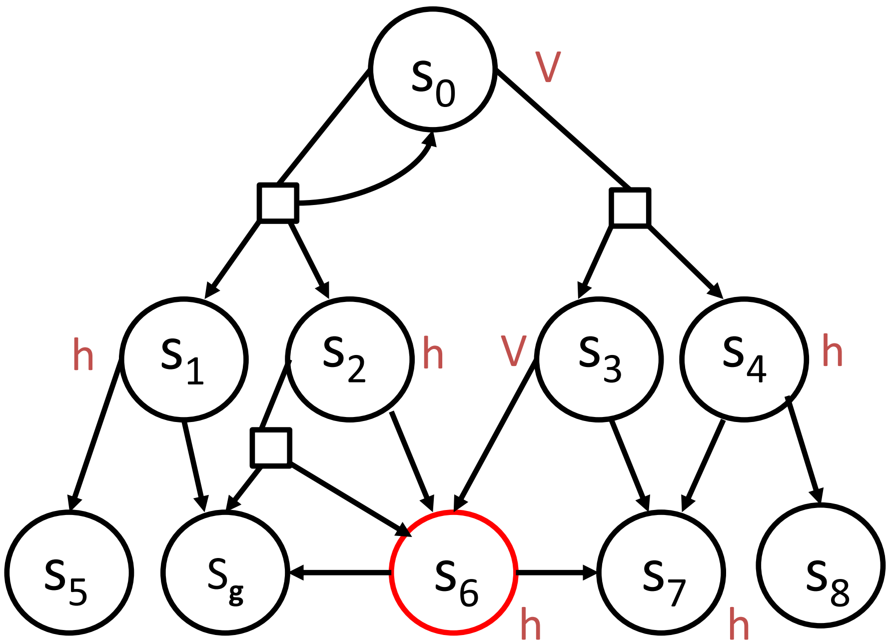
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



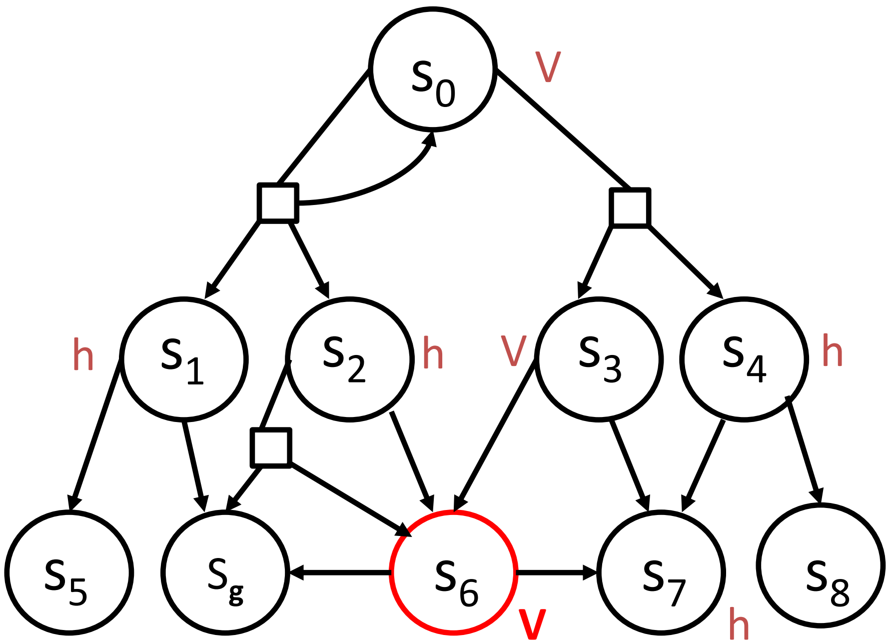
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



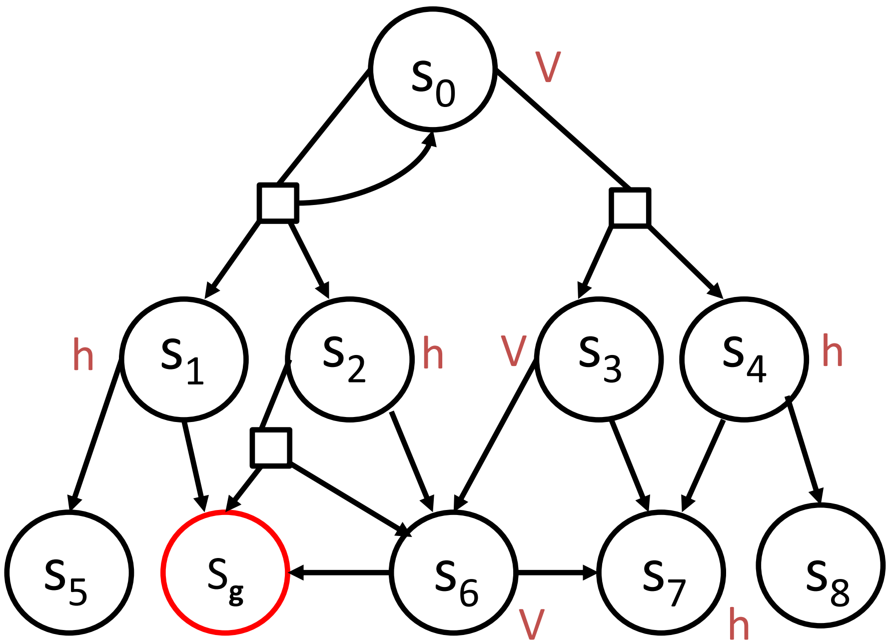
start at start state

repeat

perform a Bellman backup

simulate greedy action

Trial



start at start state

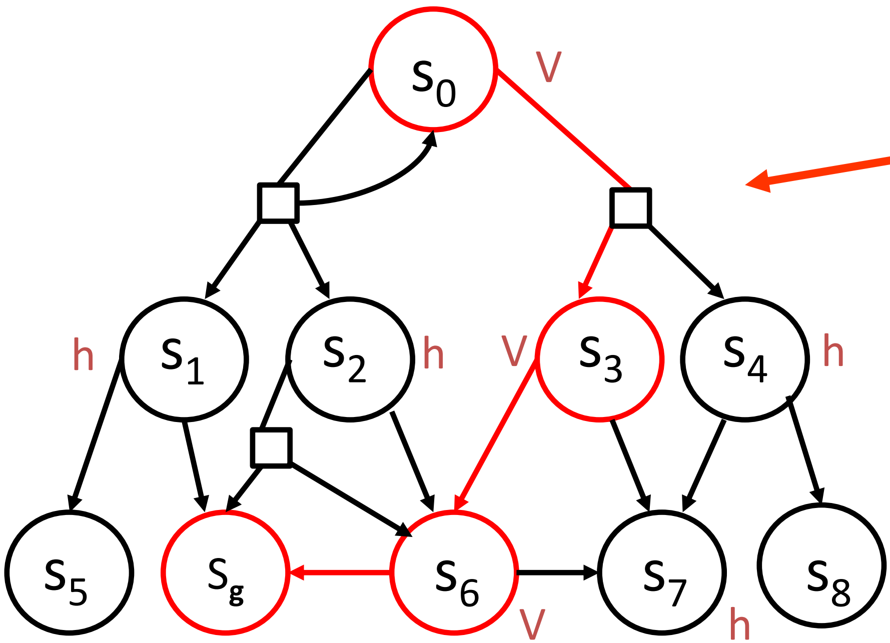
repeat

perform a Bellman backup

simulate greedy action

until hit the goal

Trial



Backup all states on trajectory

RTDP

repeat forever

start at start state
repeat
    perform a Bellman backup
    simulate greedy action
until hit the goal

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!

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

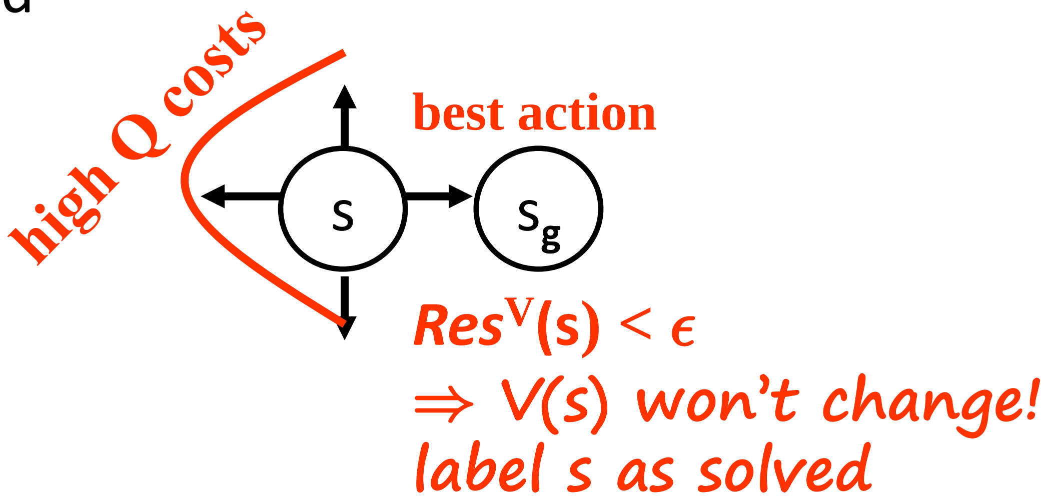
Termination Test Take 1: Labeling

- Admissible heuristic & monotonicity

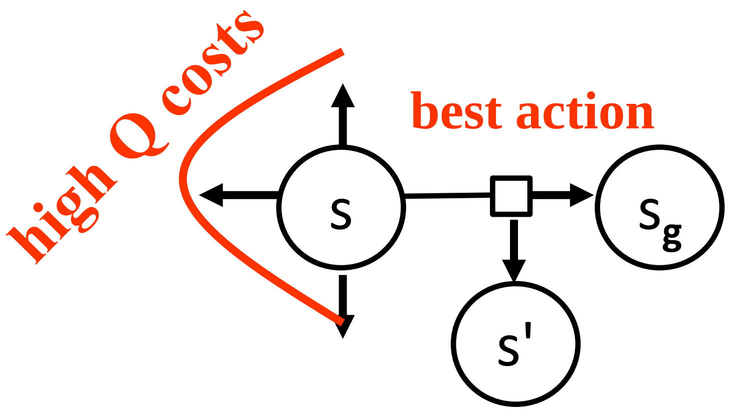
  $\Rightarrow V(s) \leq V^*(s)$

  $\Rightarrow Q(s,a) \leq Q^*(s,a)$

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



Labeling (contd)



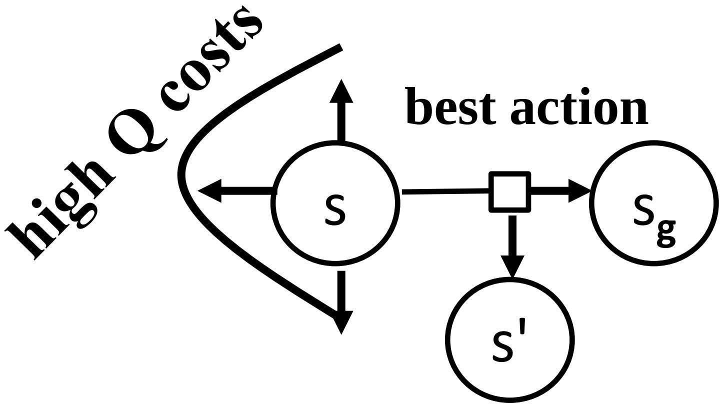
Res^V(s) < ϵ

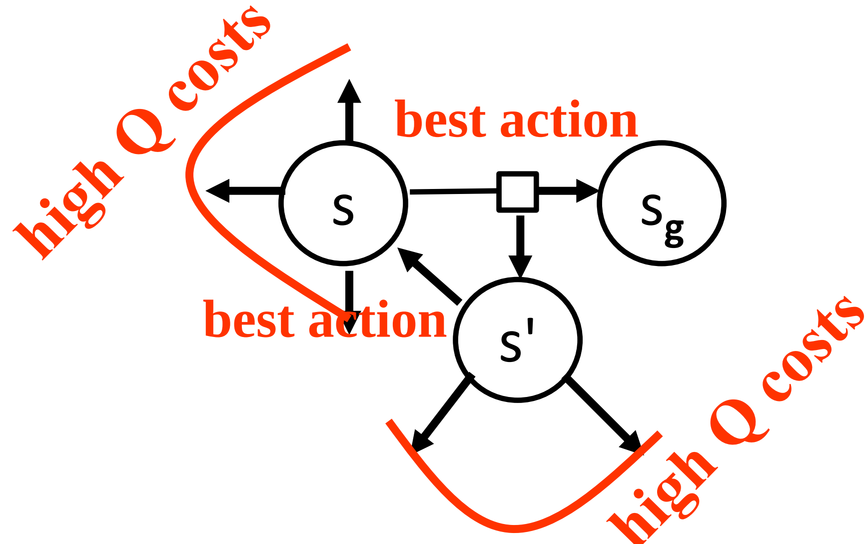
s' already solved

⇒ 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

$Res^V(s) < \epsilon$

$Res^V(s') < \epsilon$

V(s), V(s') won't change!

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

Picking a Successor Take 2

- Labeled RTDP/RTDP: sample s' $\propto$ T(s, a_{greedy}, s')
  - Adv: more probable states are explored first
  - Labeling Adv: no time wasted on converged states
  - Disadv: labeling is a hard constraint
  - Disadv: sampling ignores "amount" of convergence

- If we knew how much V(s) is expected to change?
  - sample s' $\propto$ expected change

Upper Bounds in SSPs

- RTDP/LAO* maintain lower bounds
  - call it V_l
- Additionally associate upper bound with s
  - $V_u(s) \geq V^*(s)$
- Define $gap(s) = V_u(s) – V_l(s)$
  - low gap(s): more converged a state
  - high gap(s): more expected change in its value

Backups on Bounds

- Recall monotonicity
- Backups on lower bound
  - continue to be lower bounds
- Backups on upper bound
  - continues to be upper bounds
- Intuitively
  - V_l will increase to converge to V^*
  - V_u will decrease to converge to V^*

Bounded RTDP [McMahan et al 05]

repeat

  $s \leftarrow s_0$

  repeat *//trials*

    identify a_{greedy} based on V_l

    FIND: sample $s' \propto T(s, a_{greedy}, s').gap(s')$

    $s \leftarrow s'$

  until $gap(s) < \epsilon$

  for all states s in trial in reverse order

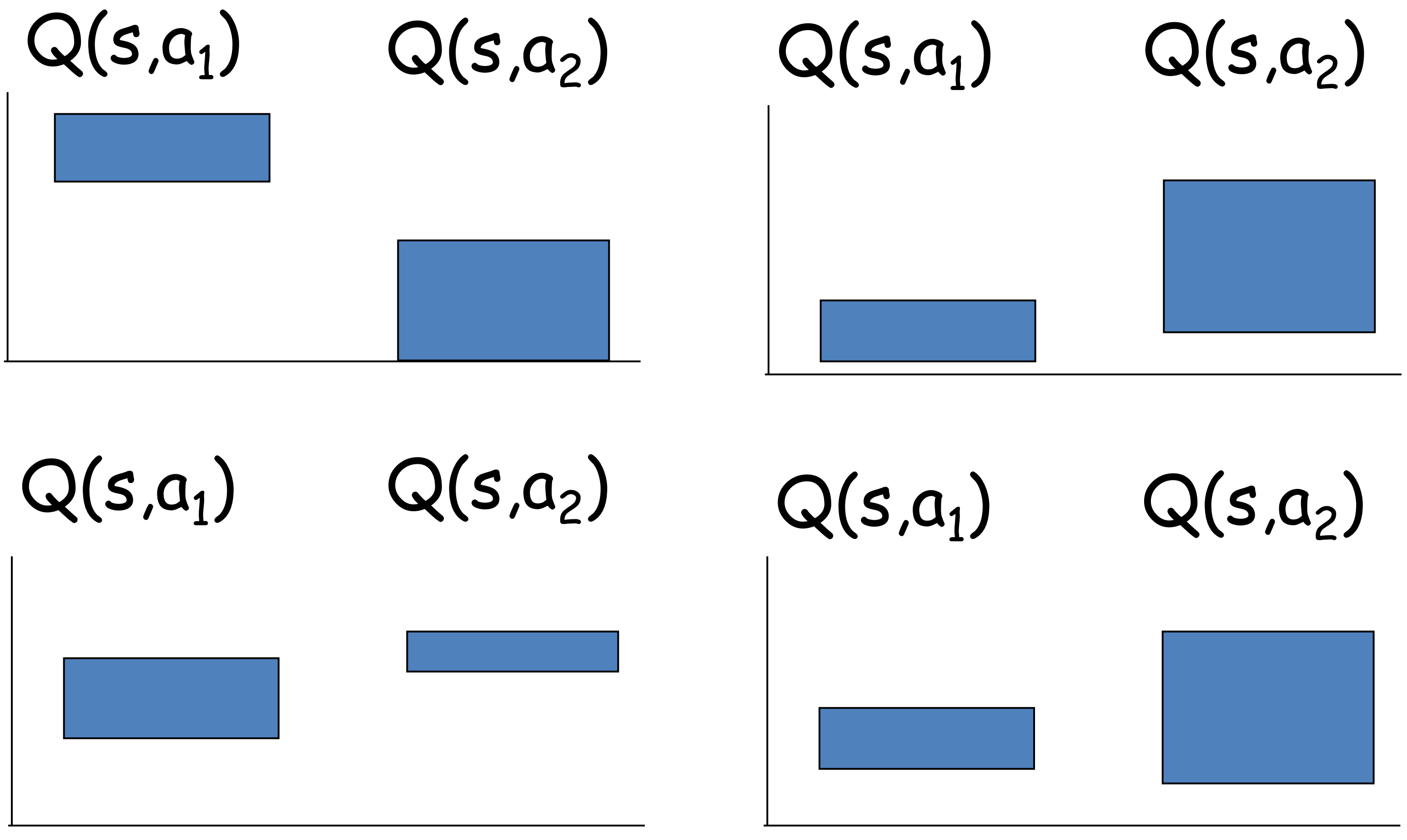
    REVISE s

until $gap(s_0) < \epsilon$

Focused RTDP [Smith&Simmons 06]

- Similar to Bounded RTDP except
  - a more sophisticated definition of priority that combines gap and prob. of reaching the state
  - adaptively increasing the max-trial length

Picking a Successor Take 3



[Slide adapted from Scott Sanner]

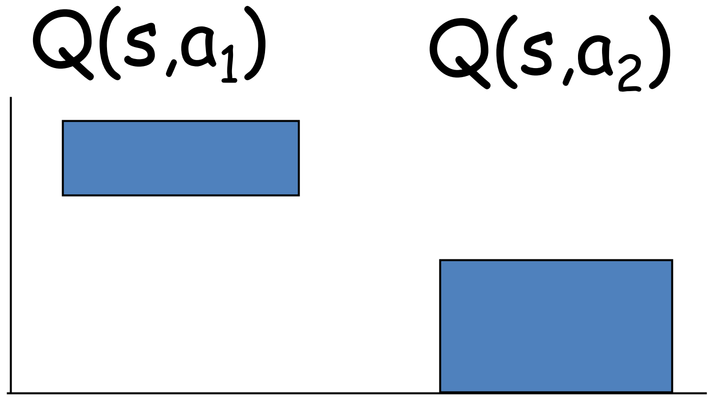
Value of Perfect Information RTDP [Sanner et al 09]

- What is the expected value of knowing V(s')
- Estimates EVPI(s')
  - using Bayesian updates
  - picks s' with maximum EVPI

Heuristic Search Algorithms

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

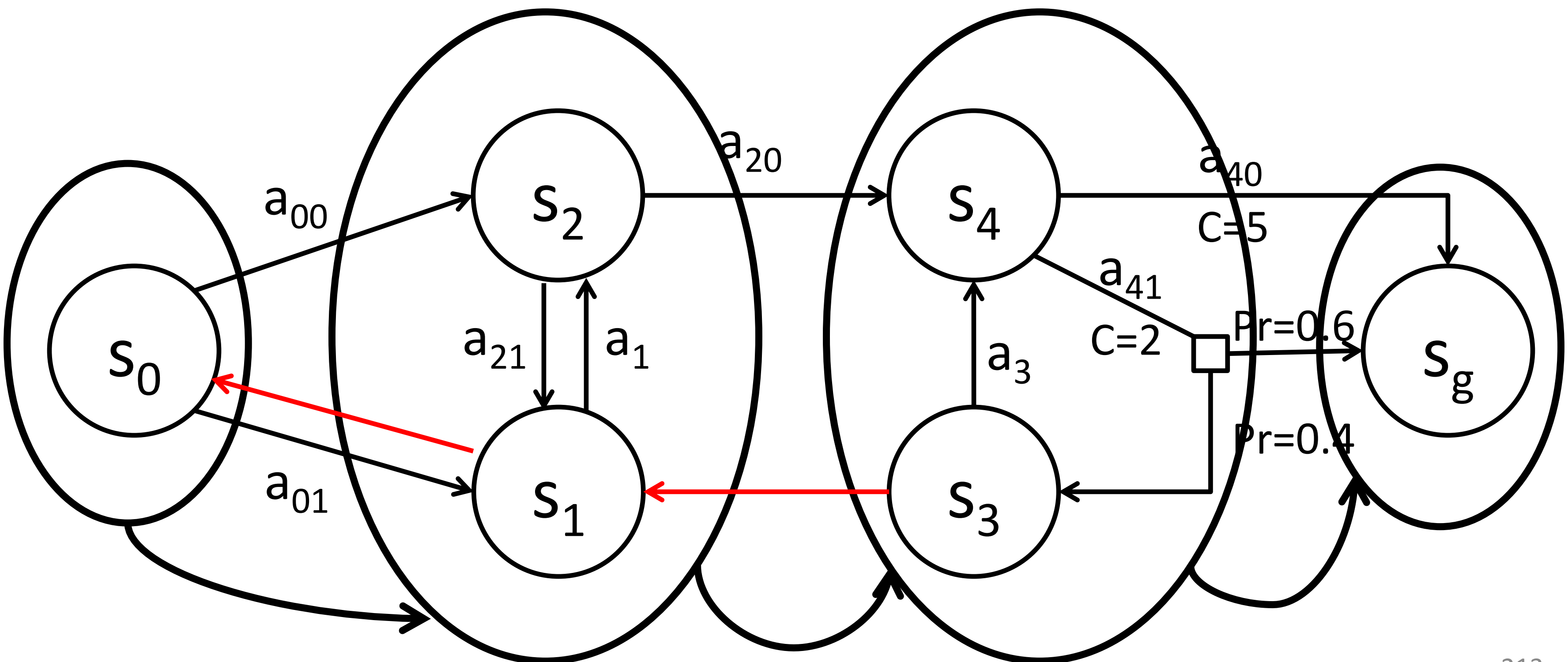
Action Elimination



If $Q_l(s,a_1) > V_u(s)$ then a_1 cannot be optimal for s.

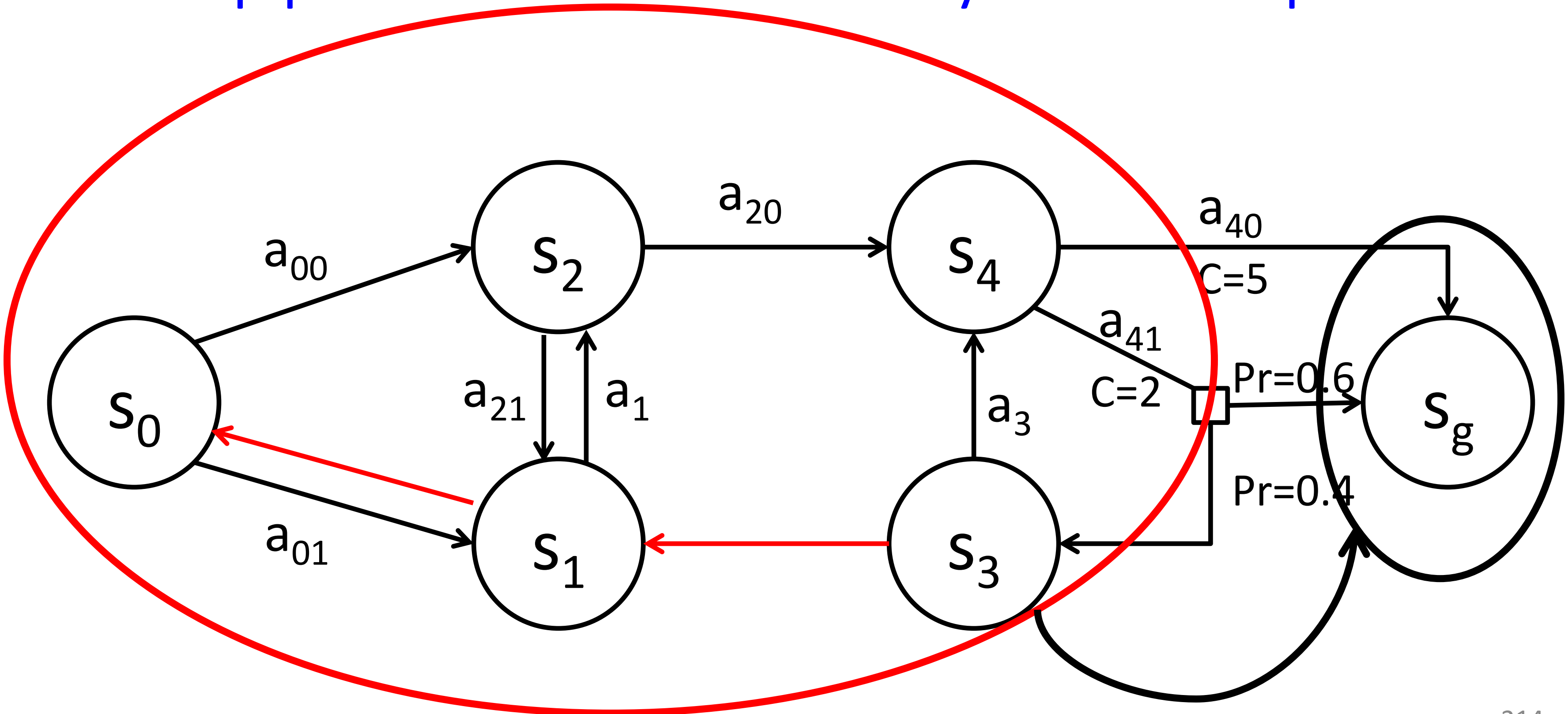
Topological VI [Dai&Goldsmith 07]

- Identify strongly-connected components
- Perform topological sort of partitions
- Backup partitions to ϵ-consistency: reverse top. order



Topological VI [Dai&Goldsmith 07]

- Identify strongly-connected components
- Perform topological sort of partitions
- Backup partitions to ϵ-consistency: reverse top. order



Focused Topological VI [Dai et al 09]

- Topological VI
  - hopes there are many small connected components
  - can‘t handle reversible domains…

- FTVI
  - initializes V_l and V_u
  - LAO*-style iterations to update V_l and V_u
  - eliminates actions using action-elimination
  - Runs TVI on the resulting graph

Factors Affecting Heuristic Search

- Quality of heuristic

- #Goal states

- Search Depth

One Set of Experiments [Dai et al 09]

	Small # Goal States	Large # Goal States
Short Search Depth	Heuristic search (better)	**FTVI (better)**
Long Search Depth	**FTVI (better)**	**FTVI (much better)**

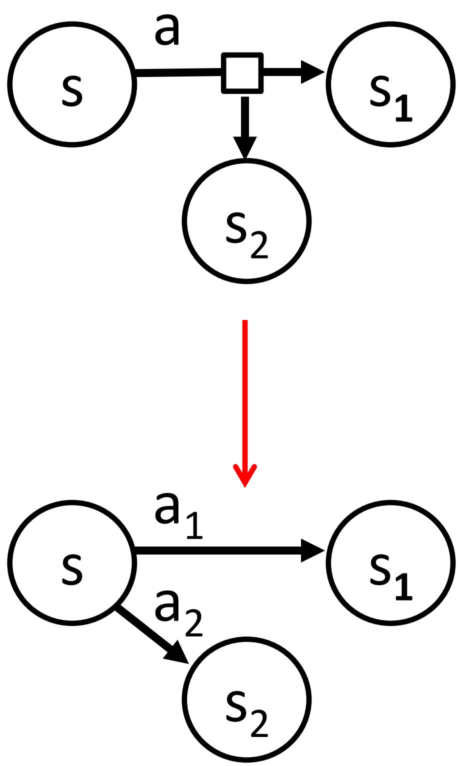
What if the number of reachable states is large?

Heuristic Search Algorithms

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

Admissible Heuristics

- Basic idea
  - Relax probabilistic domain to deterministic domain
  - Use heuristics(classical planning)
- All-outcome Determinization
  - For each outcome create a different action
- Admissible Heuristics
  - Cheapest cost solution for determinized domain
  - Classical heuristics over determinized domain



Summary of Heuristic Search

- Definitions
- Find & Revise Scheme
  - General scheme for heuristic search
- LAO* and Extensions
  - LAO*, iLAO*, RLAO*, BLAO*
- RTDP and Extensions
  - RTDP, LRTDP, BRTDP, FRTDP, VPI-RTDP
- Other uses of Heuristics/Bounds
  - Action Elimination, FTVI
- Heuristic Design
  - Determinization-based heuristics

A QUICK DETOUR

Domains with Deadends

- Dead-end state
  – a state from which goal is unreachable

- Common in real-world
  – rover
  – traffic
  – exploding blocksworld!

- SSP/SSP$_{s0}$ do not model such domains
  – assumption of at-least one proper policy

Modeling Deadends

- How should we model dead-end states?
  - V(s) is undefined for deadends

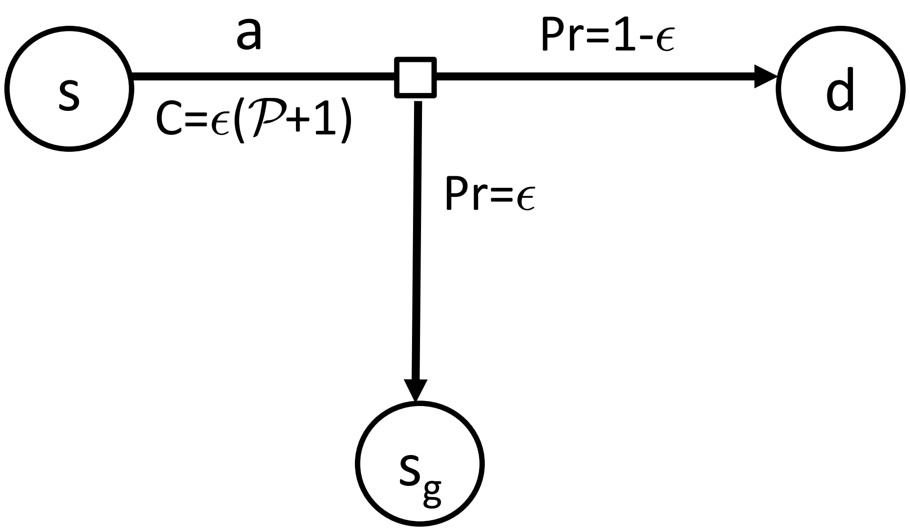
  $\Rightarrow$ VI does not converge!!

- Proposal 1
  - Add a penalty of reaching the dead-end state = $\mathcal{P}$

- Is everything well-formed?

- Are there any issues with the model?

Simple Dead-end Penalty $\mathcal{P}$



- V*(s) = $\epsilon(\mathcal{P}+1) + \epsilon.0 + (1-\epsilon).\mathcal{P}$
  $= \mathcal{P} + \epsilon$

V(non-deadend) > $\mathcal{P}$

Proposal 2

- fSSPDE: Finite-Penalty SSP with Deadends
- Agent allowed to stop at ***any*** state
  - by paying a price = penalty $\mathcal{P}$

$$\boxed{V^*(s) = \min\left(\mathcal{P}, \min_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')\mathcal{C}(s,a,s') + V^*(s')]\right)}$$

- Equivalent to SSP with special a_{stop} action
  - applicable in each state
  - leads directly to goal by paying cost $\mathcal{P}$
- SSP = fSSPDE

fSSPDE Algorithms

- All SSP algorithms applicable…
  - PI works for all domains
    - Initial proper policy: (all states: a_{stop})
  - Other algorithms also work.

- Efficiency: unknown so far…
  - Efficiency hit due to presence of deadends
  - Efficiency hit due to magnitude of $\mathcal{P}$
  - Efficiency hit due to change of topology (e.g., TVI)

SSP_{s0} with Dead-ends

- SSPADE: SSP with Avoidable Dead-ends [Kolobov et al 12]
  - dead-ends can be avoided from s_0
  - there exists a proper (partial) policy rooted at s_0
- Heuristic Search Algorithms
  - LAO*: may not converge
    - V(dead-ends) will get unbounded: VI may not converge
  - iLAO*: will converge
    - only 1 backup $\Rightarrow$ greedy policy will exit dead-ends
  - RTDP/LRTDP: may not converge
    - once stuck in dead-end → won't reach the goal
    - add max #steps in a trial… how many? adaptive?

Unavoidable Dead-ends

- fSSPUDE: Finite-Penalty SSP with Unavoidable Dead-Ends [Kolobov et al 12]
  - same as fSSPDE but now with a start state
- Same transformation applies
  - add an a_{stop} action from every state
- SSP_{s0} = fSSPUDE

Outline of the Tutorial

- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

APPROXIMATION ALGORITHMS

Motivation

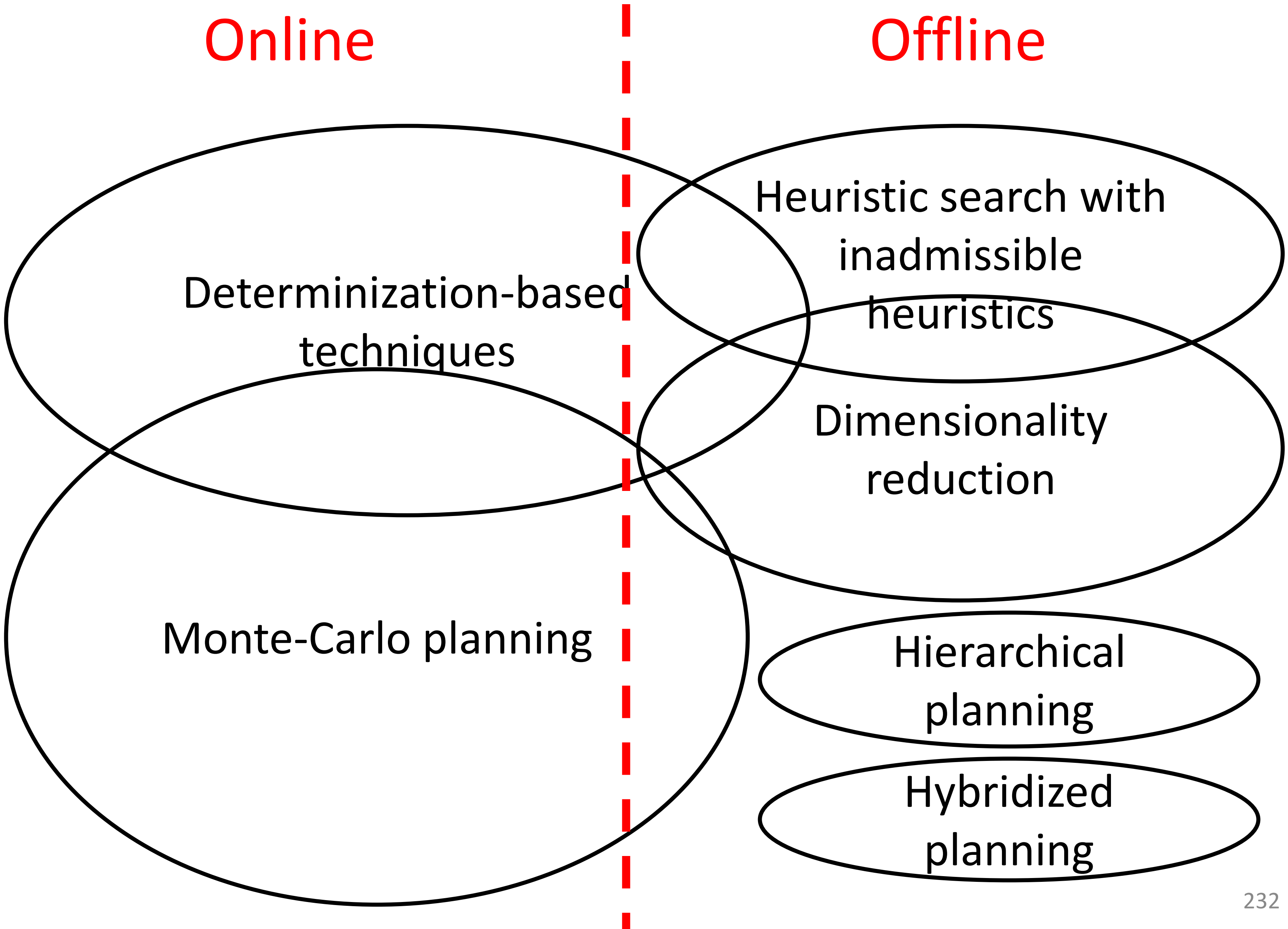
- Even π^* closed wr.t. s_0 is often too large to fit in memory…

- … and/or too slow to compute …

- … for MDPs with complicated characteristics
  - Large branching factors/high-entropy transition function
  - Large distance to goal
  - Etc.

- **Must sacrifice optimality to get a "good enough" solution**

Overview


Online
Offline
Determinization-based techniques
Heuristic search with inadmissible heuristics
Dimensionality reduction
Monte-Carlo planning
Hierarchical planning
Hybridized planning

Overview

- Not a "golden standard" classification
  - In some aspects, arguable
  - Others possible, e.g., optimal in the limit vs. suboptimal in the limit
- All techniques assume factored fSSPUDE MDPs (SSP_{s0} MDPs with a finite dead-end penalty)
- Approaches differ in the quality aspect they sacrifice
  - Probability of reaching the goal
  - Expected cost of reaching the goal
  - Both

Approximation Algorithms

✓ Overview

- Online Algorithms
  - **Determinization-based Algorithms**
  - Monte-Carlo Planning

- Offline Algorithms
  - Heuristic Search with Inadmissible Heuristics
  - Dimensionality Reduction
  - Hierarchical Planning
  - Hybridized Planning

Online Algorithms: Motivation

- **Defining characteristics:**
  - Planning + execution are interleaved
  - Little time to plan
    - Need to be fast!
  - Worthwhile to compute policy only for visited states
    - Would be wasteful for all states

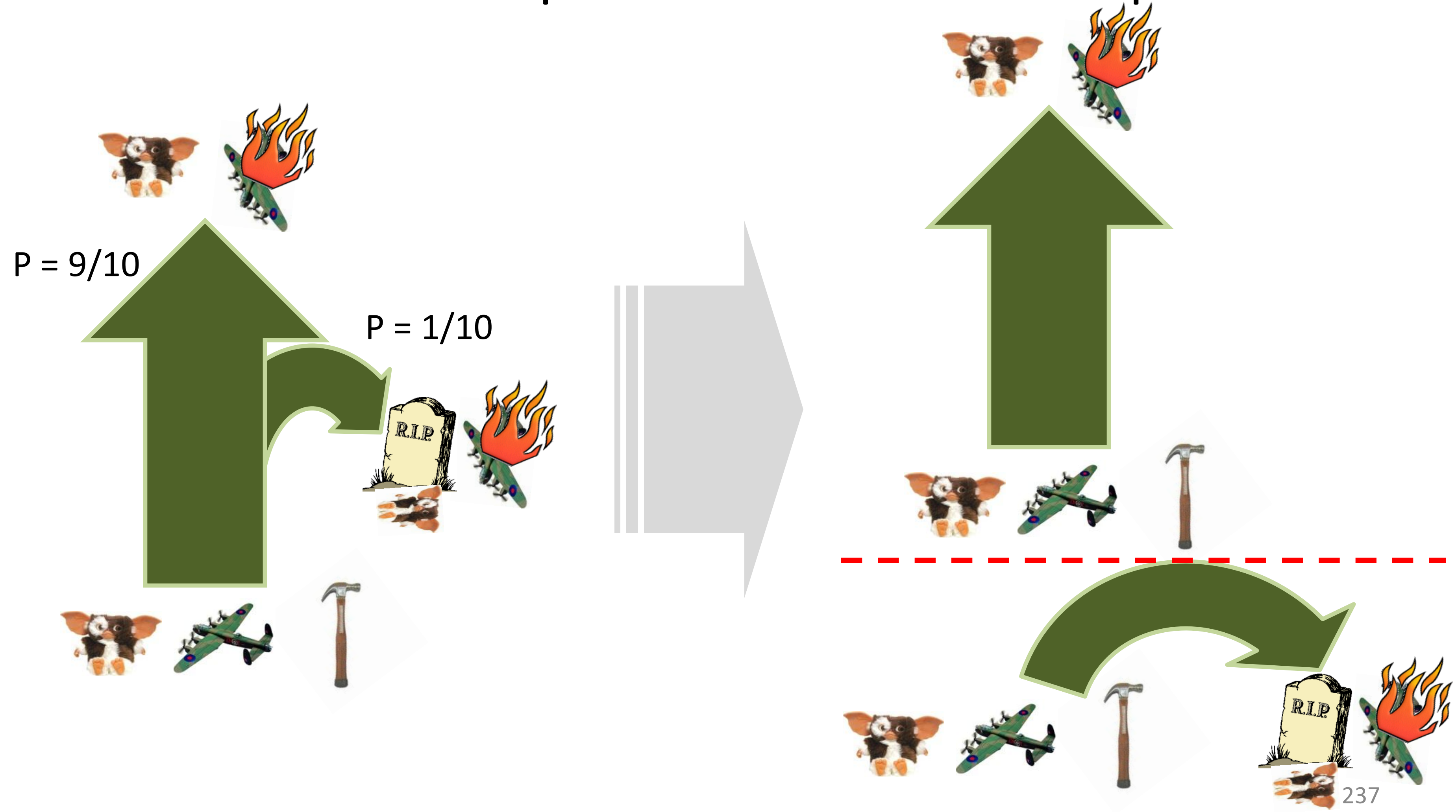
Determinization-based Techniques

- **A way to get a quick'n'dirty solution:**
  - Turn the MDP into a *classical* planning problem
  - Classical planners are *very* fast

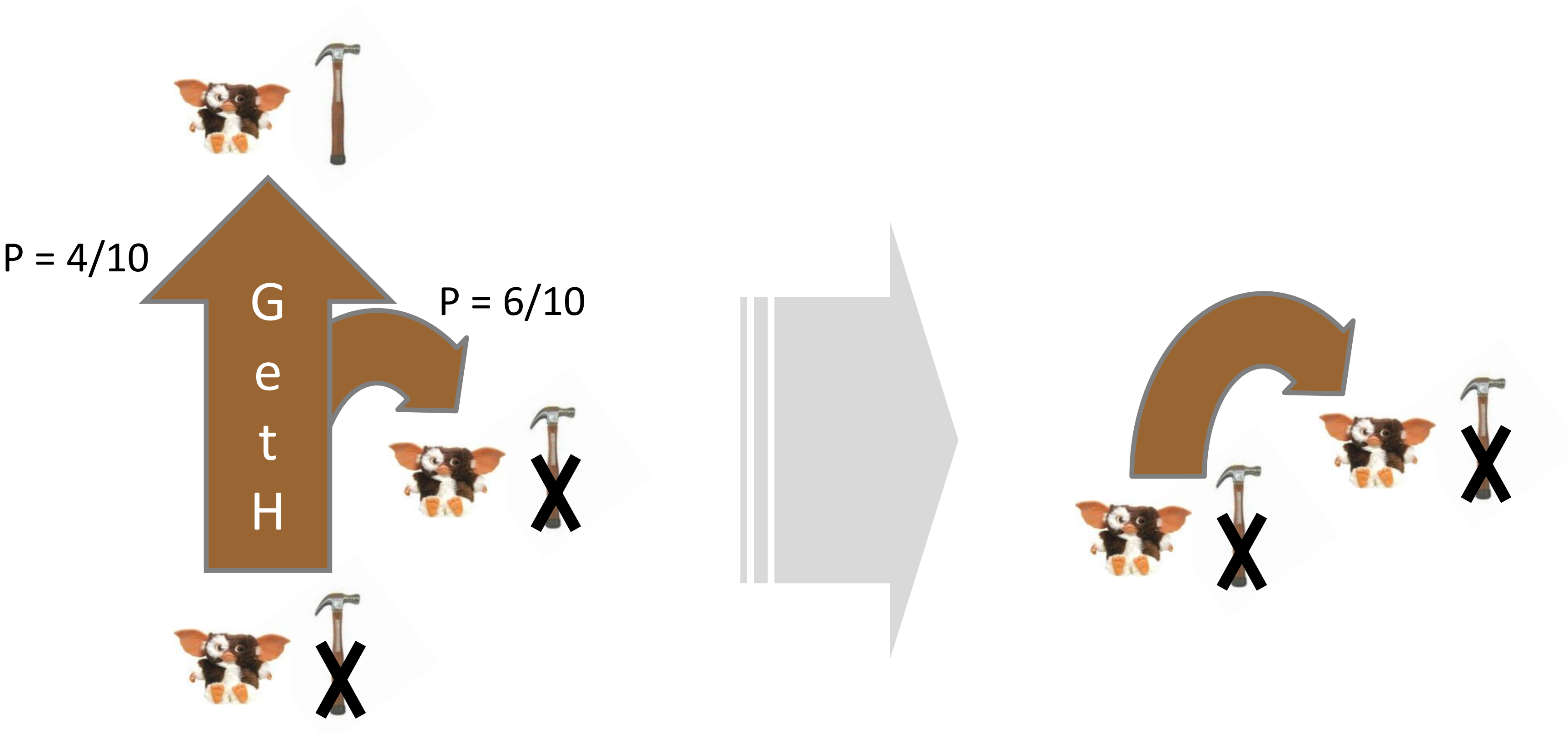
- Main idea:
  1. Compile MDP into its *determinization*
  2. Generate plans in the determinization
  3. Use the plans to choose an action in the curr. state
  4. Execute, repeat

All-Outcome Determinization

Each outcome of each probabilistic action → separate action



Most-Likely-Outcome Determinization



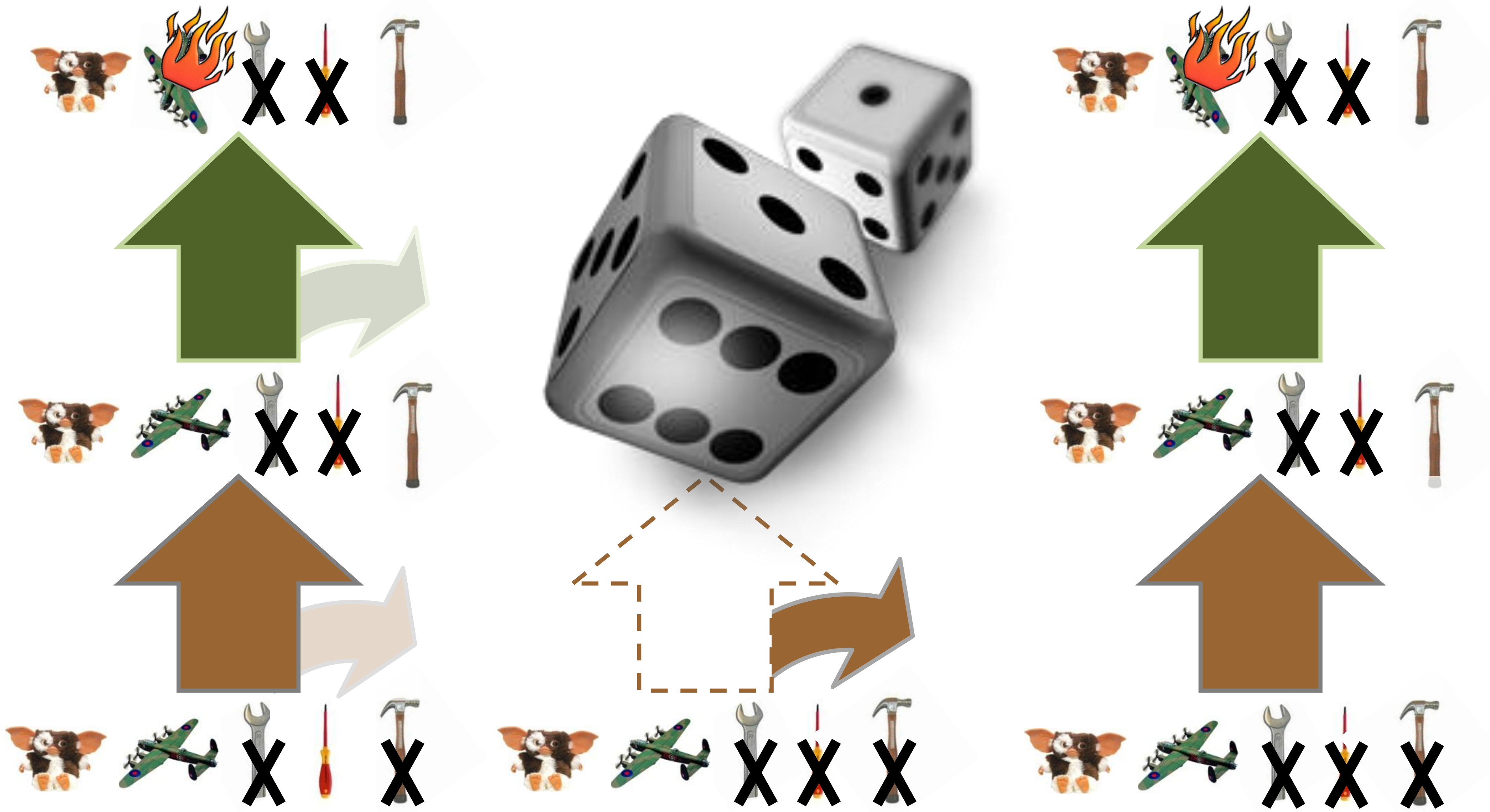
Yoon, Fern, Givan, 2007

FF-Replan: Overview & Example

1) Find a goal plan in a determinization

2) Try executing it in the original MDP

3) Replan&repeat if unexpected outcome



FF-Replan: Details

- Uses either the AO or the MLO determinization
  - MLO is smaller/easier to solve, but misses possible plans
  - AO contains all possible plans, but bigger/harder to solve
- Uses the *FF* planner to solve the determinization
  - Super fast
  - Other fast planners, e.g., LAMA, possible
- Does not cache computed plans
  - Recomputes the plan in the 3rd step in the example

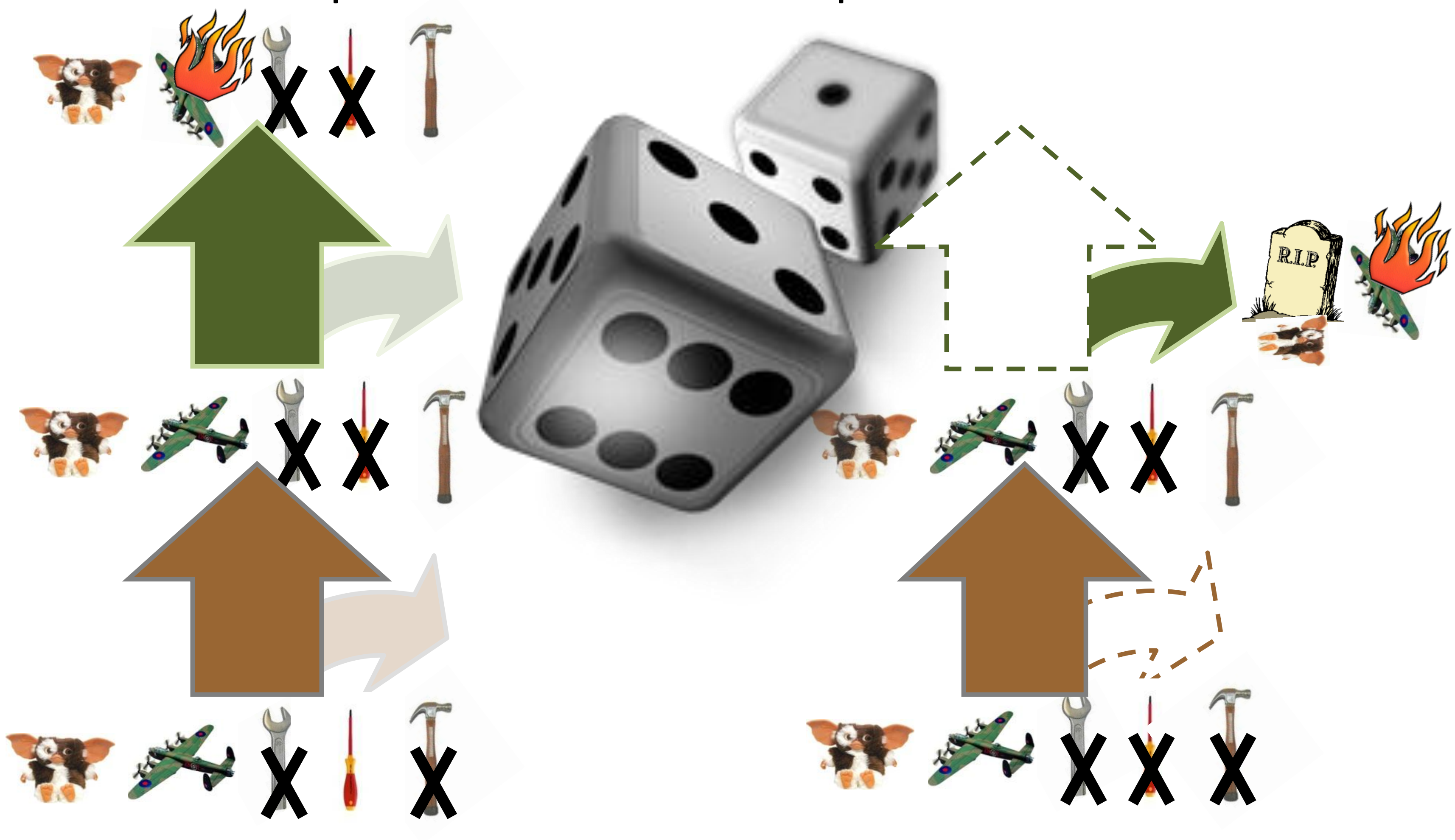
FF-Replan: Theoretical Properties

- Optimizes the *MAXPROB* criterion – P_G of reaching the goal
  - In SSPs, this is always 1.0 – FF-Replan always tries to avoid cycles!
  - Super-efficient on SSPs w/o dead ends
  - Largely ignores expected cost
- Ignores probability of deviation from the found plan
  - Results in long-winded paths to the goal
  - Troubled by *probabilistically interesting MDPs* [Little, Thiebaux, 2007]
    - There, an unexpected outcome may lead to catastrophic consequences
- **In particular, breaks down in the presence of dead ends**
  - Originally designed for MDPs without them

FF-Replan and Dead Ends

Deterministic plan:

Its possible execution:



Putting “Probabilistic” Back Into Planning

- FF-Replan is oblivious to probabilities
  - Its main undoing
  - How do we take them into account?

- **Sample determinizations probabilistically!**
  - Hopefully, probabilistically unlikely plans will be rarely found

- Basic idea behind *FF-Hindsight*

FF-Hindsight: Overview

(Estimating Q-Value, Q(s,a))

S: Current State, A(S) → S'

1. For Each Action A, Draw Future Samples

Each Sample is a Deterministic Planning Problem

2. Solve Time-Dependent Classical Problems

See if you have goal-reaching solutions, estimate Q(s,A)

3. Aggregate the solutions for each action

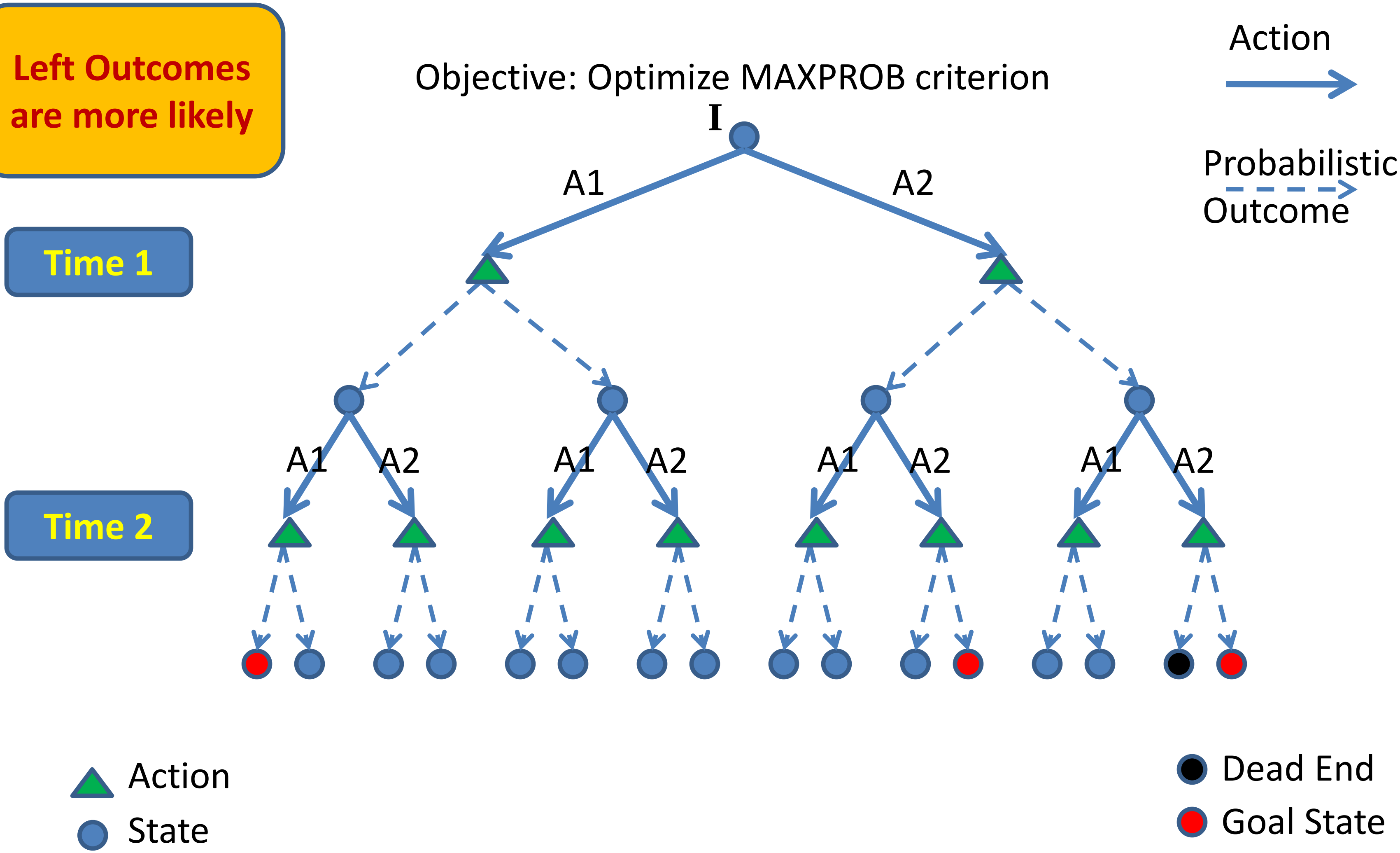
$\text{Max}_A\ Q(s,A)$

4. Select the action with best aggregation

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

FF-Hindsight: Example

Left Outcomes are more likely



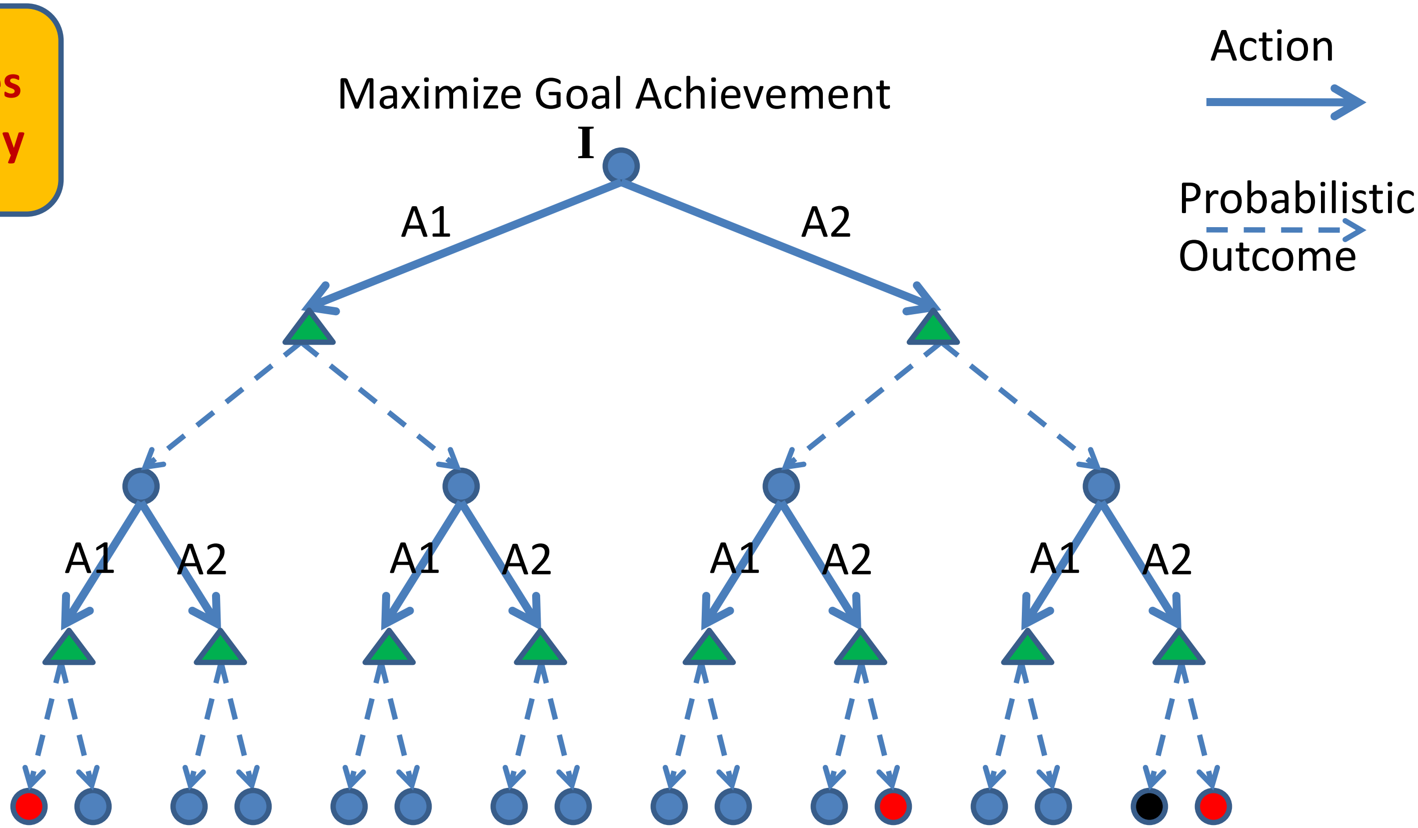
Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

FF-Hindsight: Sampling a Future-1

Left Outcomes are more likely

Time 1

Time 2



A1: 1
A2: 0

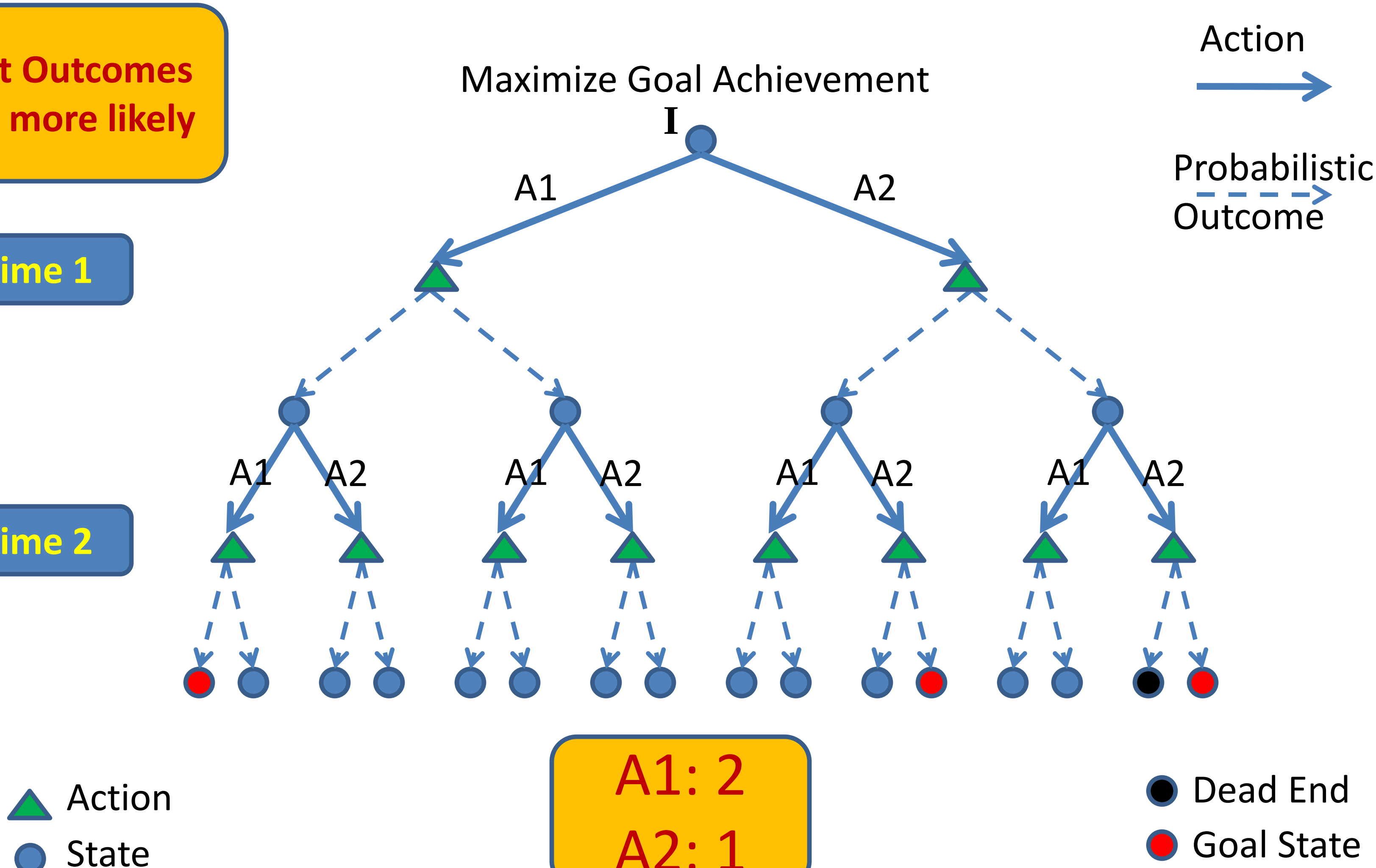
Action
State

Dead End
Goal State

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

FF-Hindsight: Sampling a Future-2

Left Outcomes are more likely

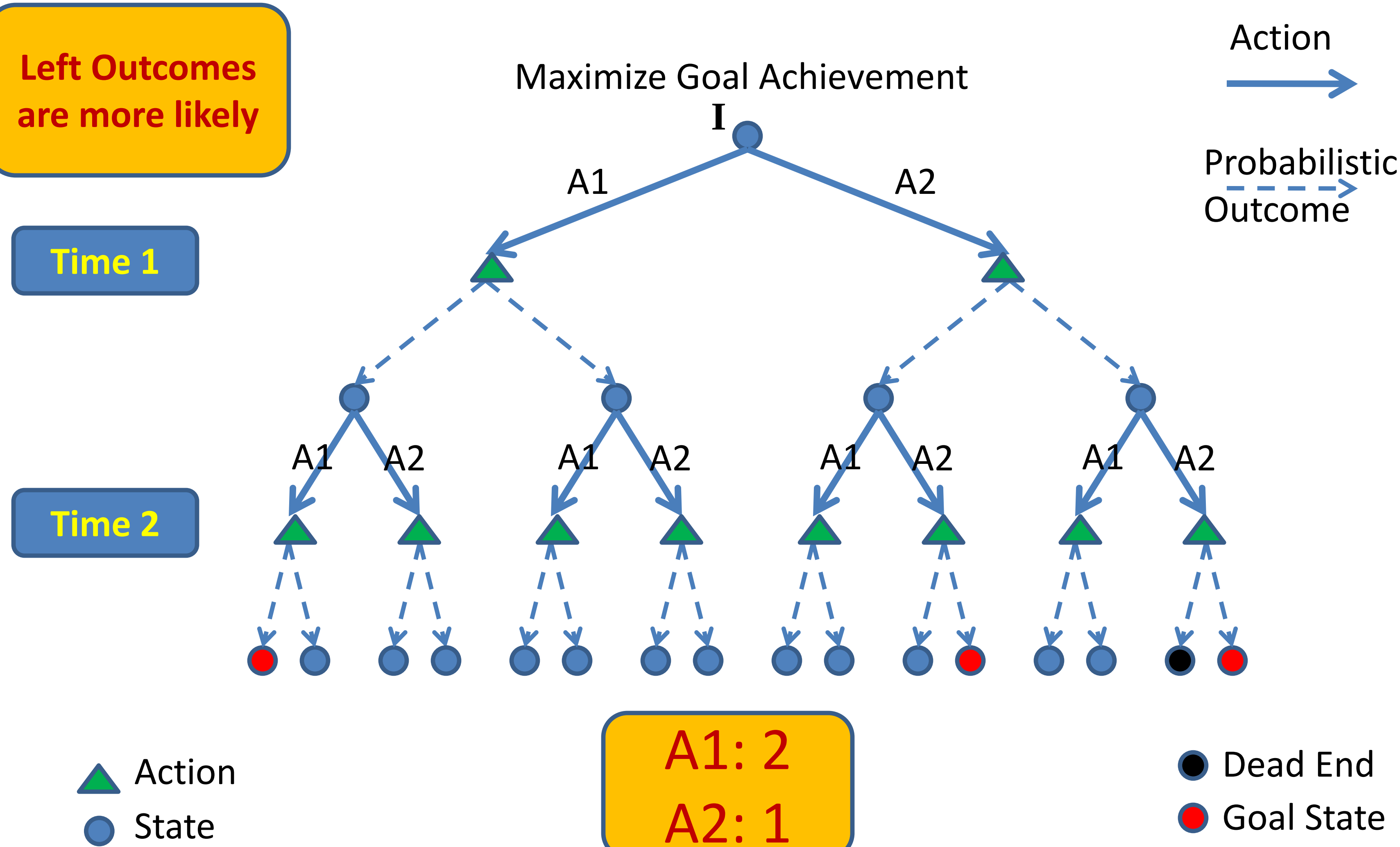


A1: 2
A2: 1

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

FF-Hindsight: Sampling a Future-3

Left Outcomes are more likely



A1: 2

A2: 1

FF-Hindsight: Details & Theoretical Properties

- For each s, FF-Hindsight samples *w* *L-horizon futures* F^L
  - In factored MDPs, amounts to choosing *a*'s outcome for each *h*
- Futures are solved by the FF planner
  - Fast, since they are much smaller than the AO determinization
- With enough futures, will find MAXPROB-optimal policy
  - If horizon *H* is large enough and a few other assumptions
- **Much better than FF-Replan on MDPs with dead ends**
  - But also slower – lots of FF invocations!

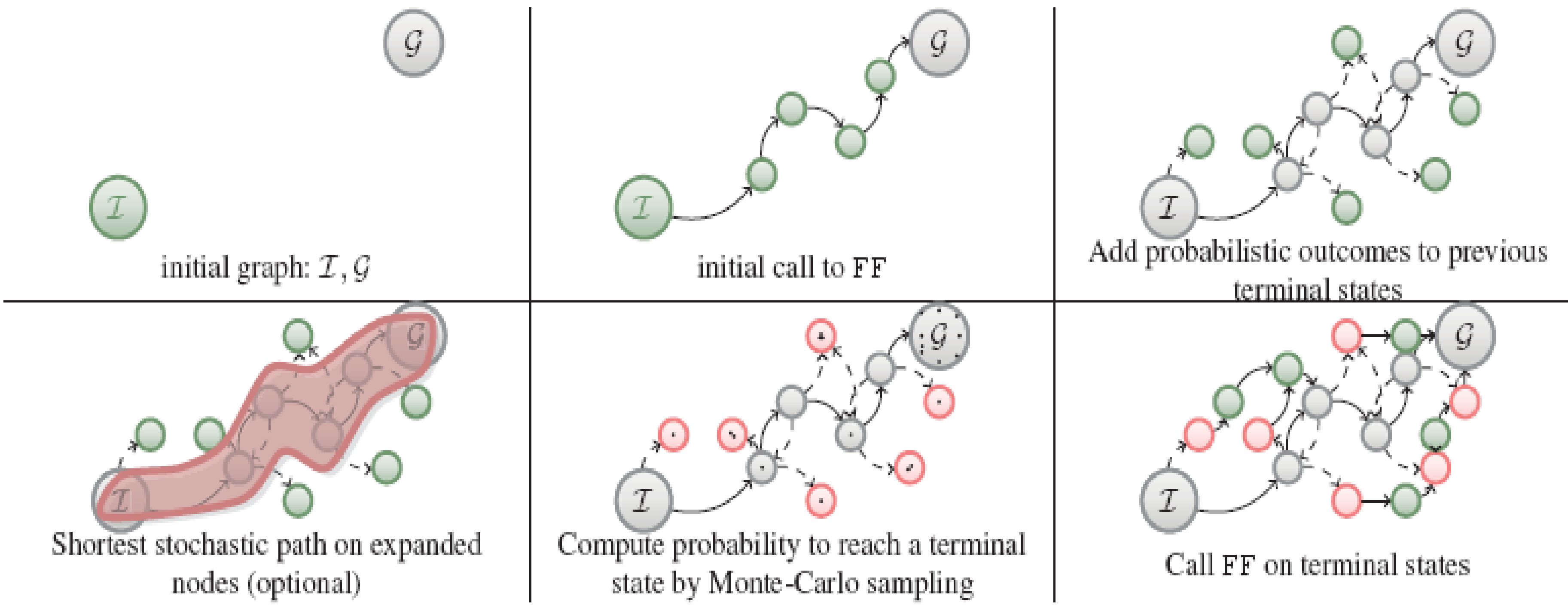
Providing Solution Guarantees

- FF-Replan provides no solution guarantees
  - May have $P_G = 0$ on SSPs with dead ends, even if $P^*_G > 0$
  - Wastes solutions: generates them, then forgets them
- FF-Hindsight provides some theoretical guarantees
  - Practical implementations distinct from theory
  - Wastes solutions: generates them, then forgets them
- **RFF (Robust FF) provides quality guarantees in practice**
  - Constructs a policy tree out of deterministic plans

RFF: Overview

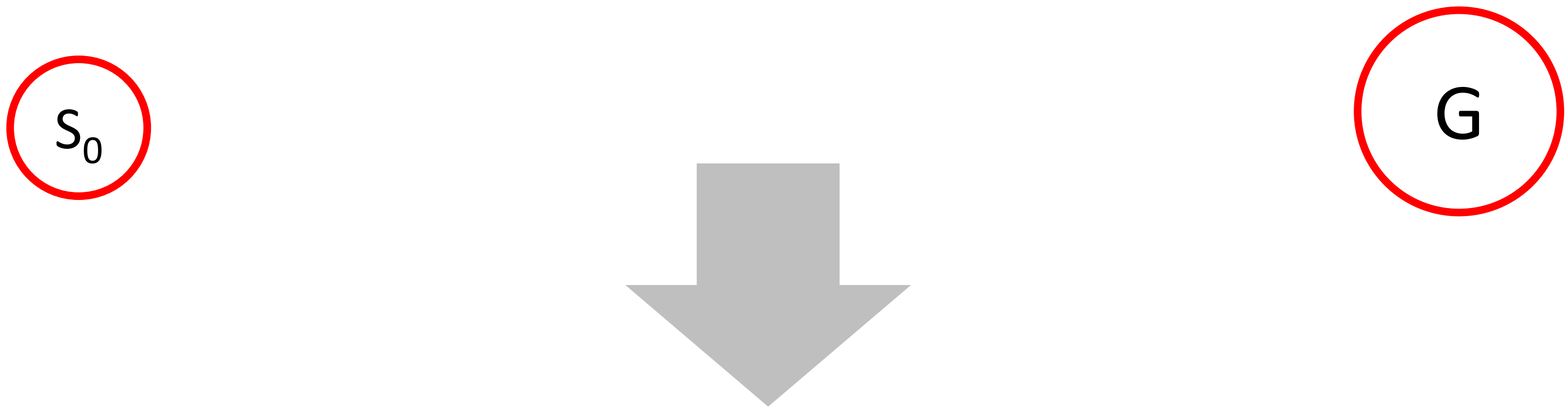
F. Teichteil-Königsbuch, U. Kuter, G. Infantes, AAMAS'10

Make sure the probability of ending up in an unknown state is < ε



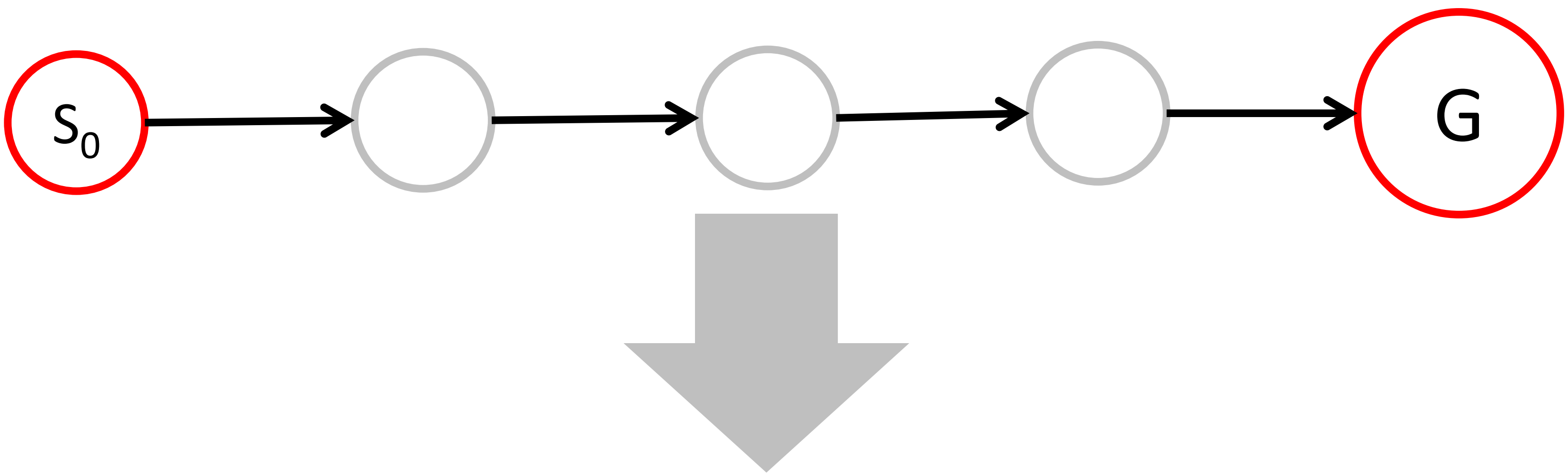
RFF: Initialization

1. Generate either the AO or MLO determinization. Start with the policy graph consisting of the initial state s_0 and all goal states G



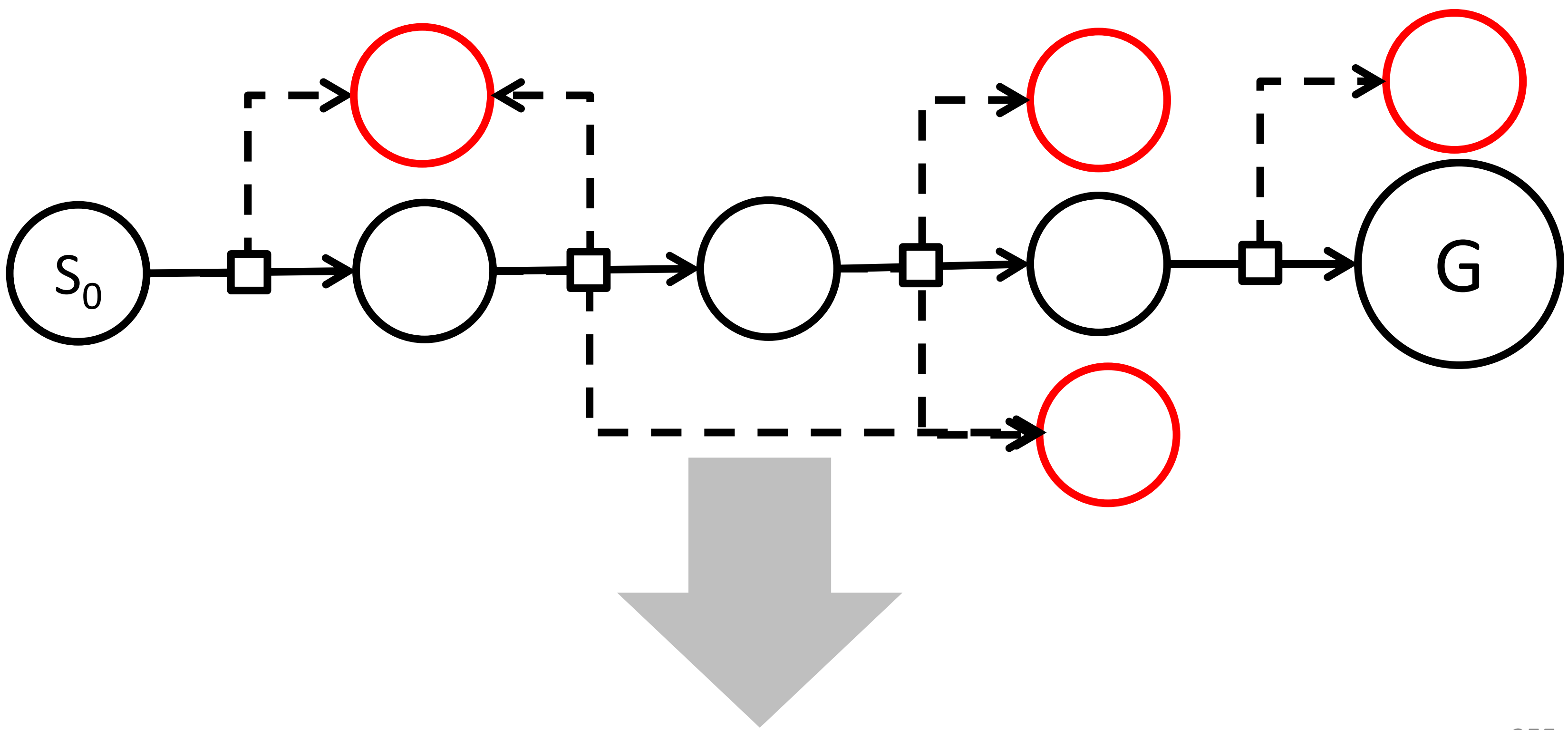
RFF: Finding an Initial Plan

2. Run FF on the chosen determinization and add all the states along the found plan to the policy graph.



RFF: Adding Alternative Outcomes

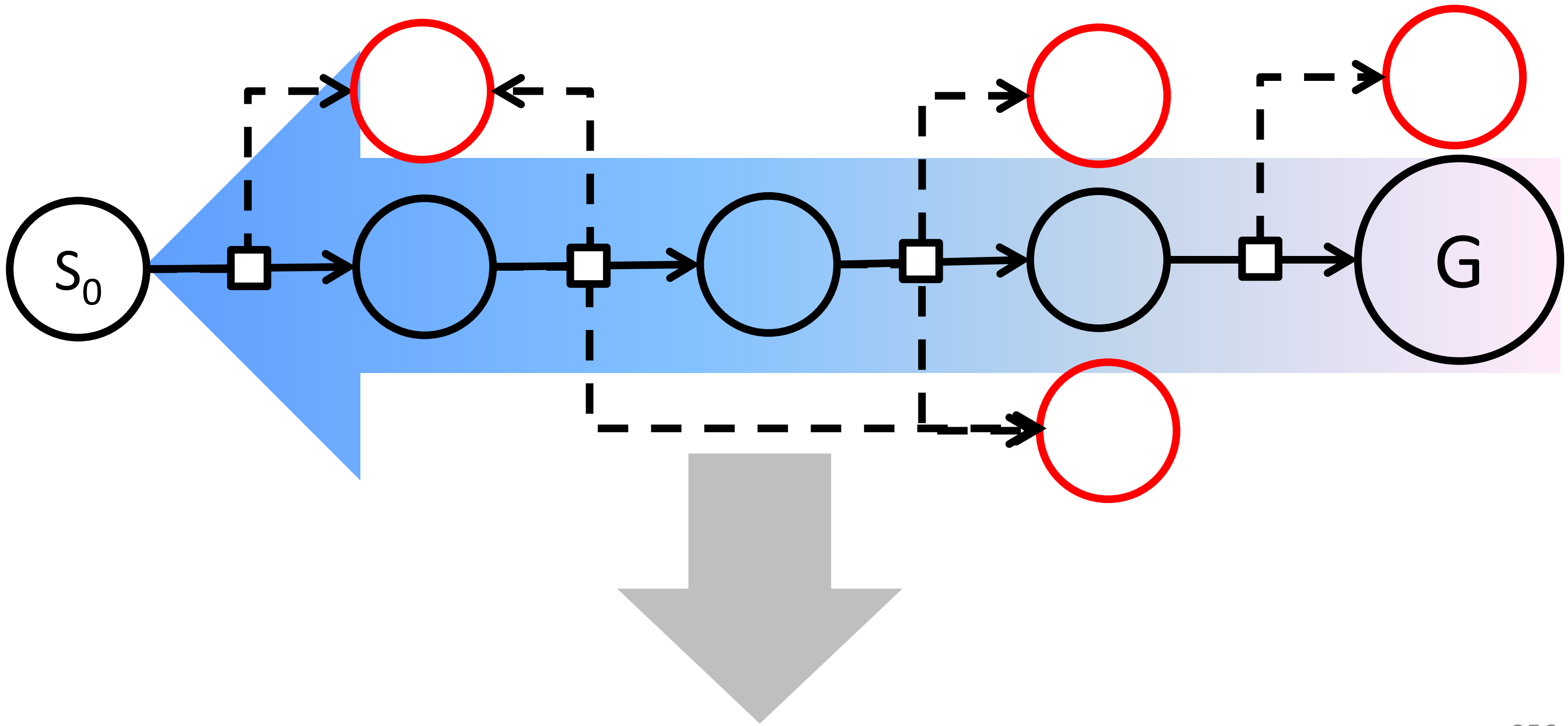
3. Augment the graph with states to which other outcomes of the actions in the found plan could lead and that are not in the graph already. They are the policy graph's *fringe states*.



RFF: Run VI **(Optional)**

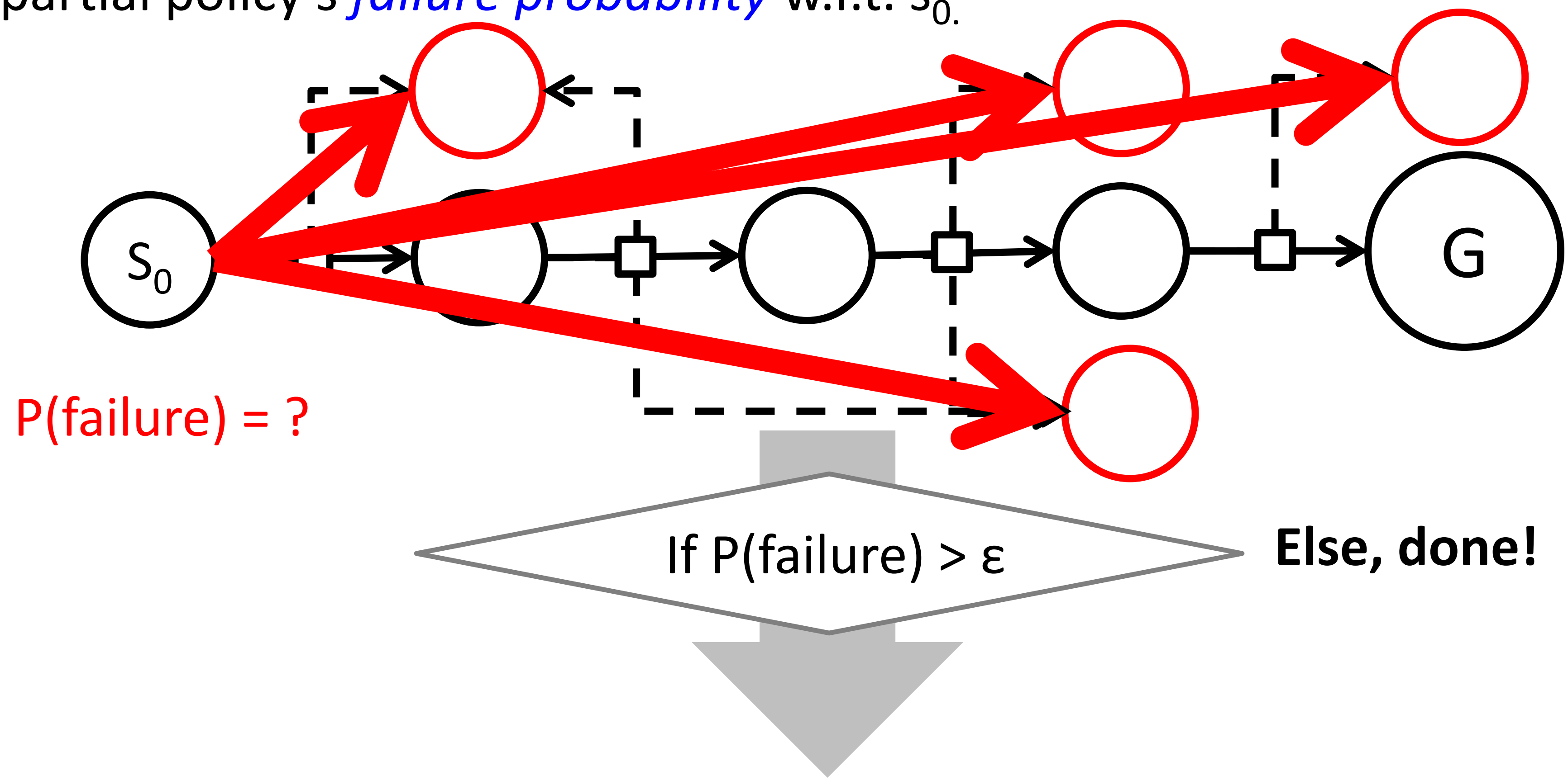
4. Run VI to propagate heuristic values of the newly added states.

This possibly changes the graph's fringe and helps avoid dead ends!



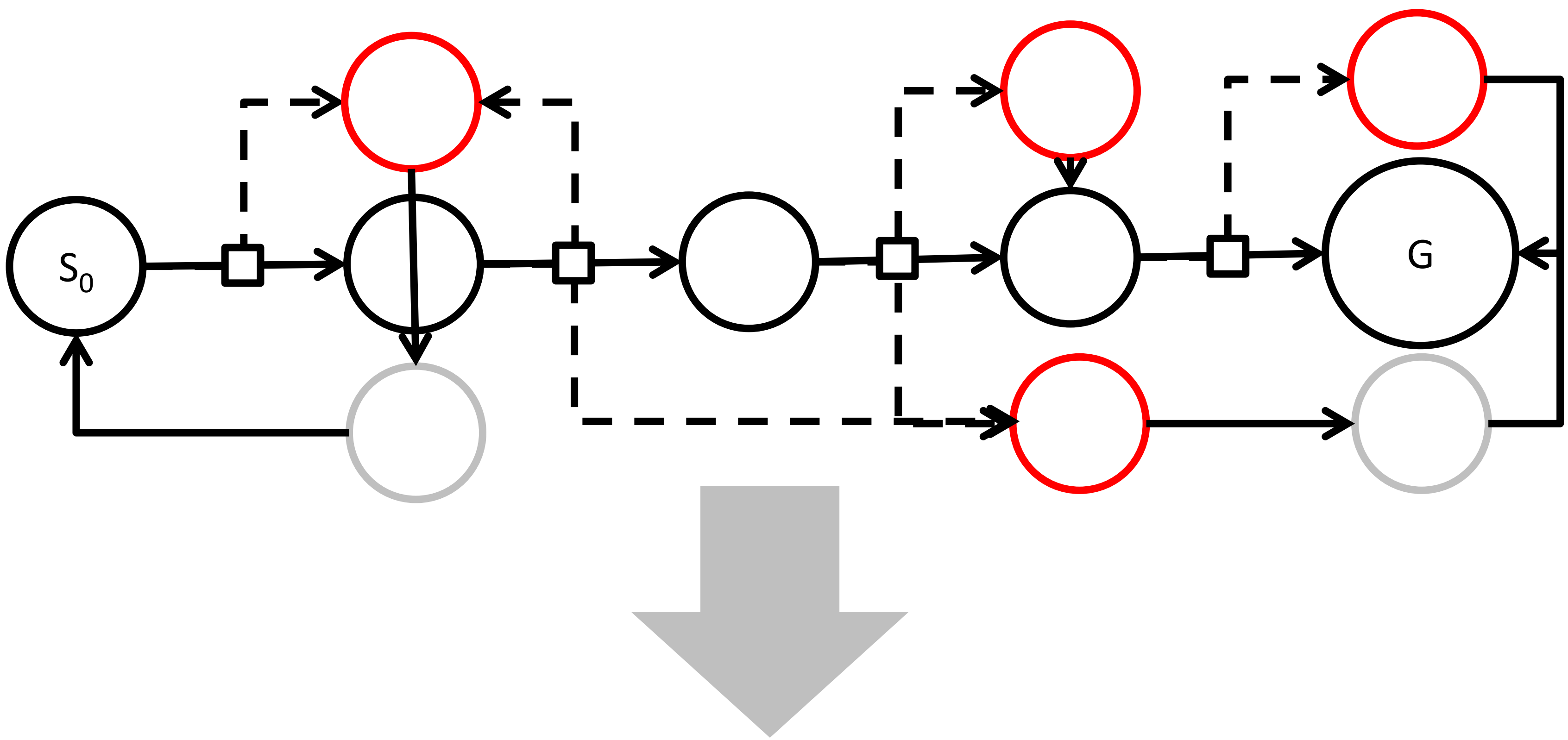
RFF: Computing Replanning Probability

5. Estimate the probability *P(failure)* of reaching the fringe states (e.g., using Monte-Carlo sampling) from s_0. This is the current partial policy's *failure probability* w.r.t. s_0.



RFF: Finding Plans from the Fringe

6. From each of the fringe states, run FF to find a plan to reach the goal or one of the states already in the policy graph.



Go back to step 3: Adding Alternative Outcomes

RFF: Details

- Can use either the AO or the MLO determinization
  - Slower, but better solutions with AO
- When finding plans in st. 5, can set graph states as goals
  - Or the MDP goals themselves
- Using the optional VI step is beneficial for solution quality
  - Without this step, actions chosen under FF guidance
  - With it – under VI guidance
  - But

RFF: Theoretical Properties

- Fast
  - FF-Replan forgets computed policies
  - RFF essentially memorizes them

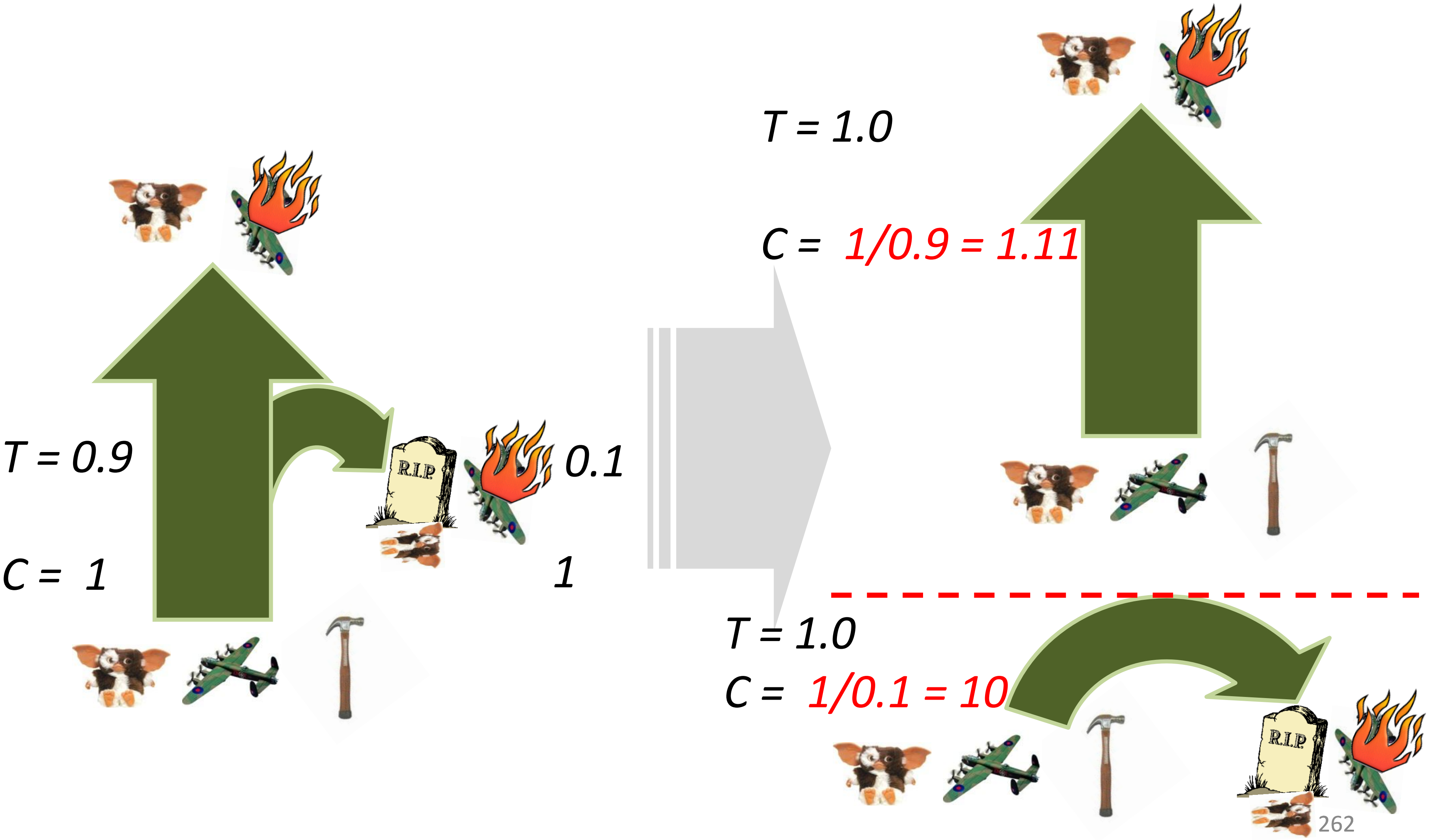
- **When using AO determinization, guaranteed to find a policy that with *P = 1 - ε* will not need replanning**

Anticipatory Vs. Preemptive Planning

- FF-Hindsight and RFF use an *anticipatory* strategy
  - Try to foresee deviations from a deterministic plan

- **Can also try to use deterministic plans that will likely not be deviated from**
  - Main idea of *HMDPP*
  - Implemented with a *self-loop determinization*

Self-Loop Determinization



Self-Loop Determinization

- Like AO determinization, but modifies action costs
  - Assumes that getting "unexpected" outcome when executing *a* deterministic plan means staying in the current state
  - In SL det, $C_{SL}(Outcome(a, i))$ is the expected cost of repeating *a* in the MDP to get *Outcome(a, i)*.
  - Thus, $C_{SL}(Outcome(a, i)) = C(a) / T(Outcome(a, i))$

- **"Unlikely" deterministic plans look expensive in SL det.!**

- Estimate $h_{SL}(s') \approx$ cost of the cheapest goal plan in the SL det.

HMDPP: Overview

S: Current State, A(S) → S'

1. For Each Action A, Estimate $Q_{SL}(s, a)$ and $Q_{pdb}(s, a)$
   - $Q_{SL}(s, a) = C(a) + \sum_{s'}[T(s,a,s') + h_{SL}(s')]$
   - $Q_{pdb}(s, a) = C(a) + \sum_{s'}[T(s,a,s') + h_{pdb}(s')]$
     - $h_{pdb}(s')$ helps recognize dead ends

2. Choose an action based on a combination of $Q_{SL}(s, a)$ and $Q_{pdb}(s, a)$

Summary of Determinization Approaches

- Revolutionized SSP MDPs approximation techniques
  - Harnessed the speed of classical planners
  - Eventually, “learned” to take into account probabilities
  - Help optimize for a “proxy” criterion, MAXPROB

- Classical planners help by quickly finding paths to a goal
  - Takes “probabilistic” MDP solvers a while to find them on their own

- **However…**
  - Still almost completely disregard expect cost of a solution
  - Often assume uniform action costs (since many classical planners do)
  - So far, not useful on *FH* and *IHDR* MDPs turned into *SSPs*
    - Reaching a goal in them is trivial, need to approximate reward more directly
  - Impractical on problems with large numbers of outcomes

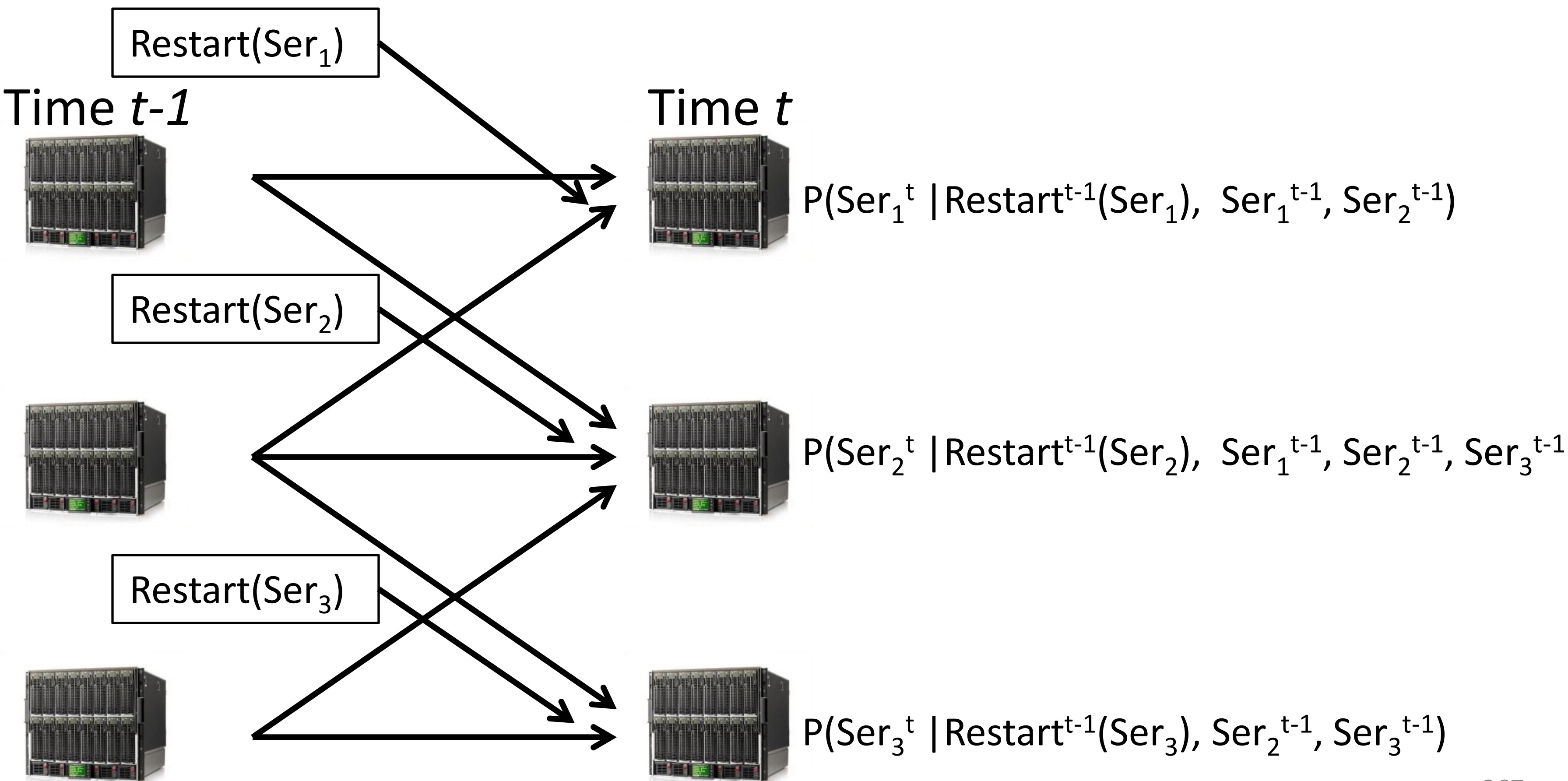
Approximation Algorithms

✓ Overview
- Online Algorithms
  - Determinization-based Algorithms
  - **Monte-Carlo Planning**
- Offline Algorithms
  - Heuristic Search with Inadmissible Heuristics
  - Dimensionality Reduction
  - Hierarchical Planning
  - Hybridized Planning

Monte-Carlo Planning

- Recall the *Sysadmin* problem:

A: R: $\sum_i [Ser_i = \uparrow]$ *T*:



Monte-Carlo Planning: Motivation

- Characteristics of Sysadmin:
  - FH MDP turned SSP_{s0} MDP
    - Reaching the goal is trivial, determinization approaches not really helpful
  - Enormous reachable state space
  - High-entropy T ($2^{|X|}$ outcomes per action, many likely ones)
    - Building determinizations can be super-expensive
    - Doing Bellman backups can be super-expensive

- **Try *Monte-Carlo planning***
  - Does not manipulate *T* or *C/R* explicitly – no Bellman backups
  - Relies on a *world simulator* – indep. of MDP description size

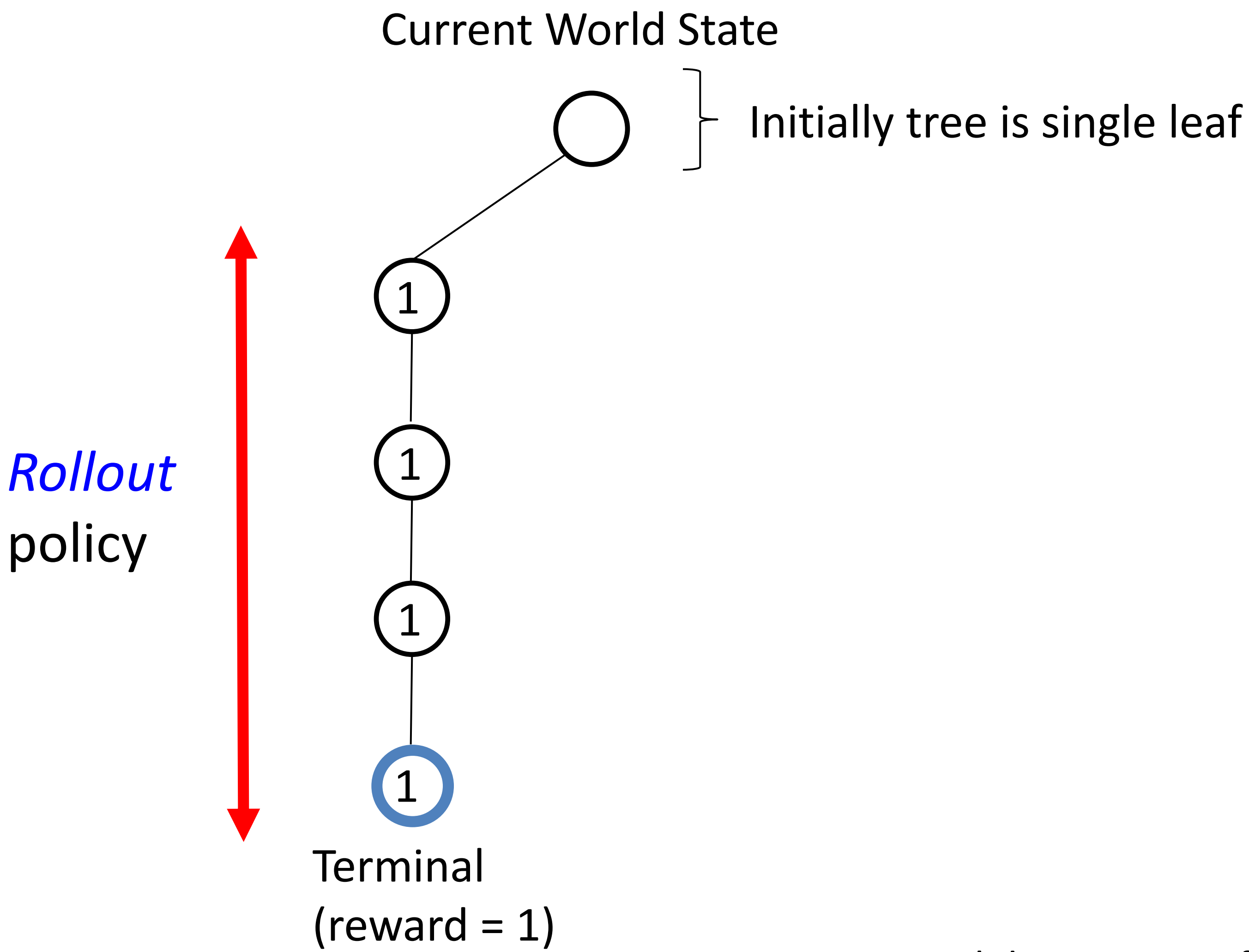
UCT: A Monte-Carlo Planning Algorithm

- UCT [Kocsis & Szepesvari, 2006] computes a solution by simulating the current best policy and improving it
  - Similar principle as RTDP
  - But action selection, value updates, and guarantees are different

- Success stories:
  - Go (thought impossible in ‘05, human grandmaster level at 9x9 in ‘08)
  - Klondike Solitaire (wins 40% of games)
  - General Game Playing Competition
  - Real-Time Strategy Games
  - Probabilistic Planning Competition
  - The list is growing…

UCT Example

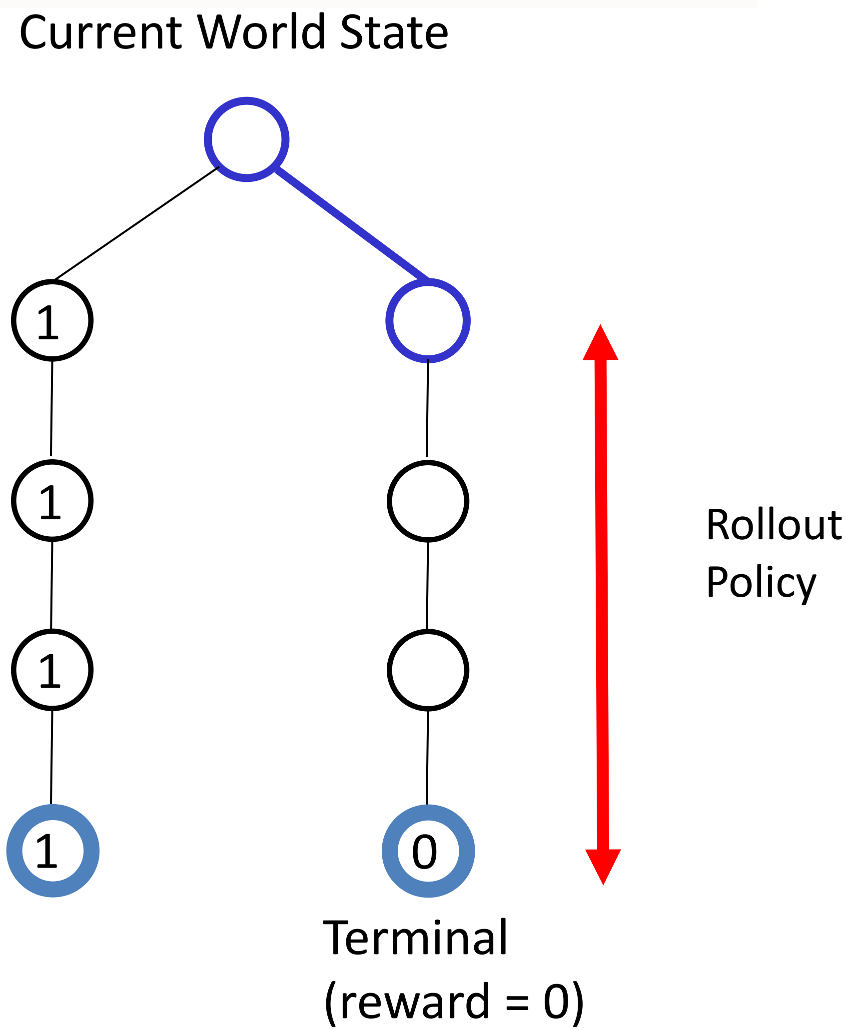
At a leaf node perform a random *rollout*



Slide courtesy of A. Fern

UCT Example

Must select each action at a node at least once

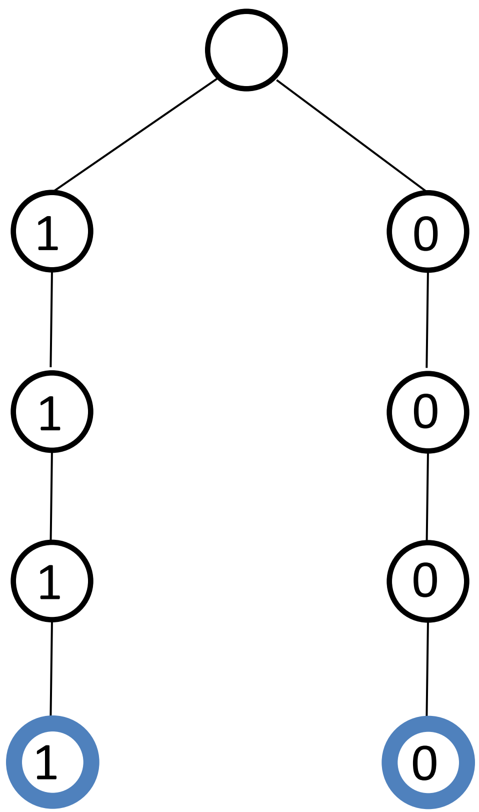


Slide courtesy of A. Fern

UCT Example

Must select each action at a node at least once

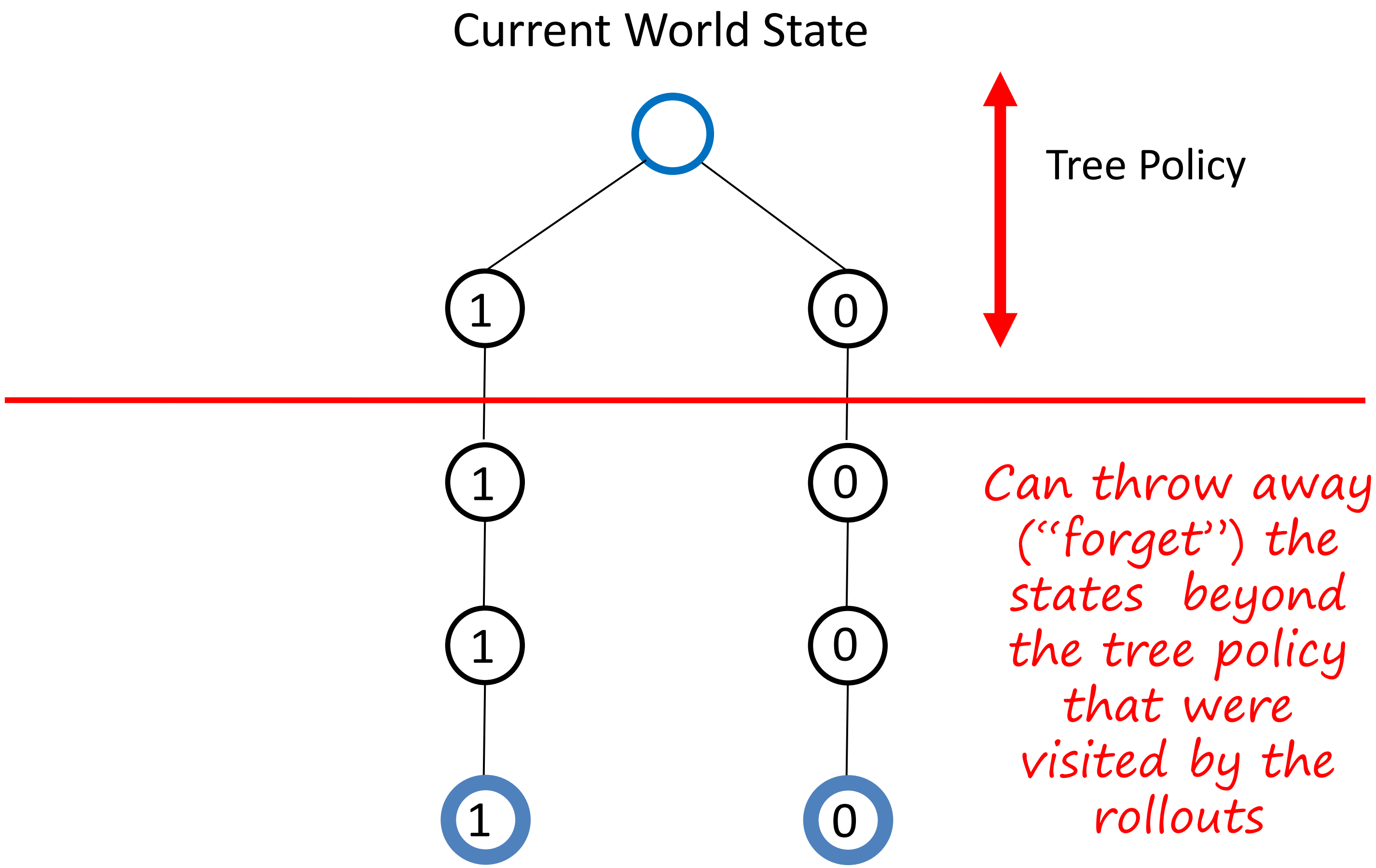
Current World State



Slide courtesy of A. Fern

UCT Example

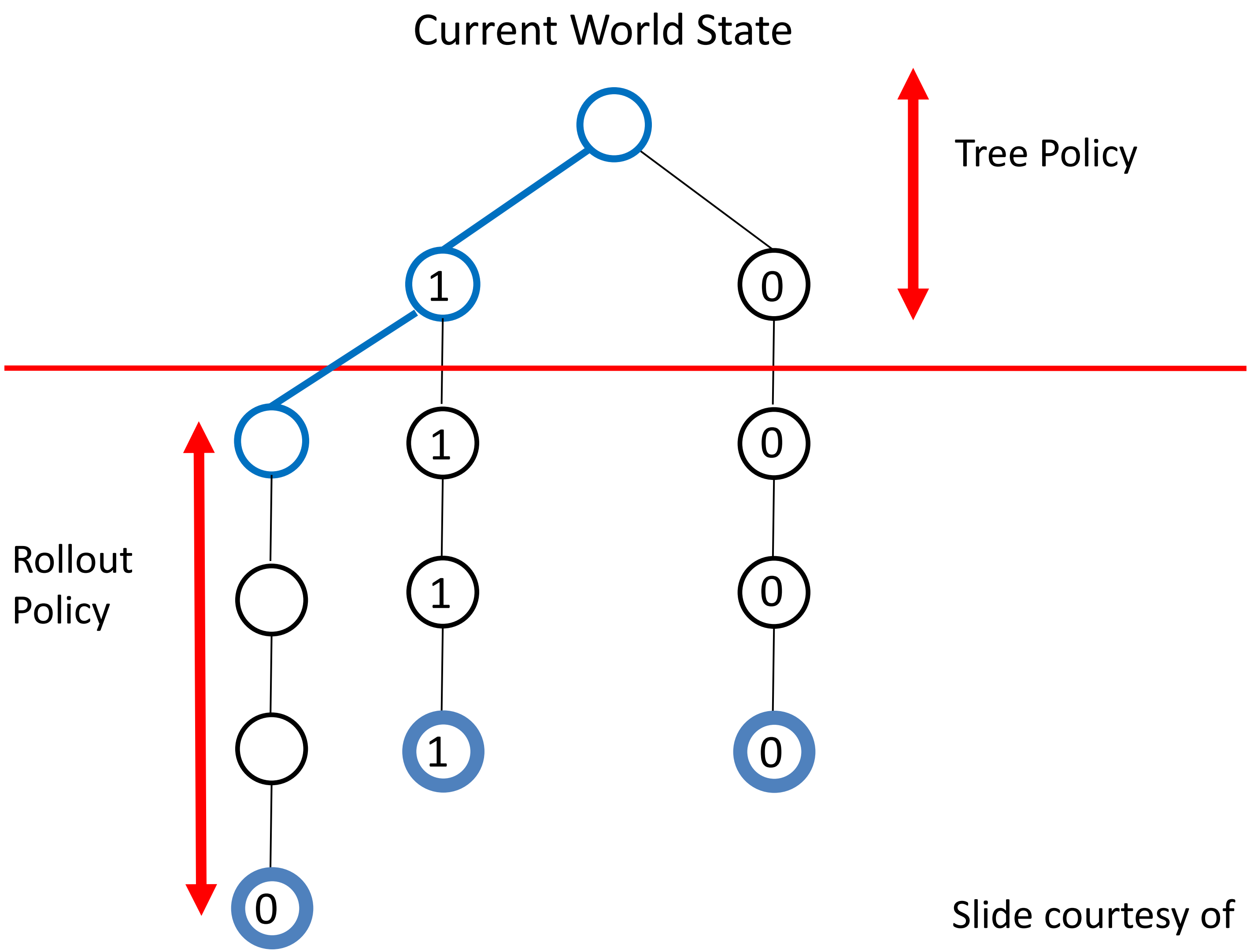
When all node actions tried once, select action according to tree policy



Slide courtesy of A. Fern

UCT Example

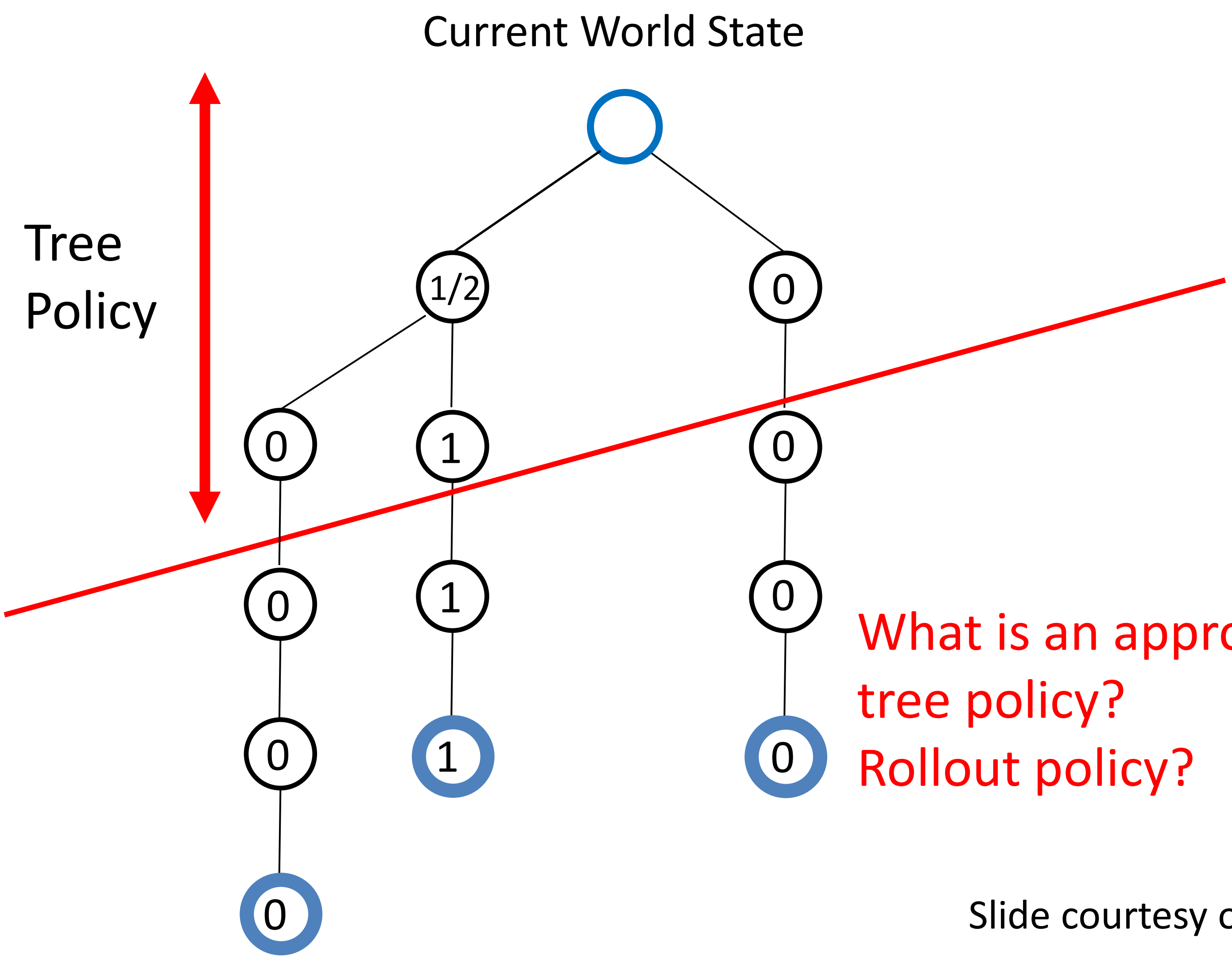
When all node actions tried once, select action according to tree policy



Slide courtesy of A. Fern

UCT Example

When all node actions tried once, select action according to tree policy



What is an appropriate tree policy?
Rollout policy?

Slide courtesy of A. Fern

UCT Details

- Rollout policy:
  - Basic UCT uses random

- Tree policy:
  - Q(s,a) : average reward received in current trajectories after taking action a in state s
  - n(s,a) : number of times action a taken in s
  - n(s) : number of times state s encountered

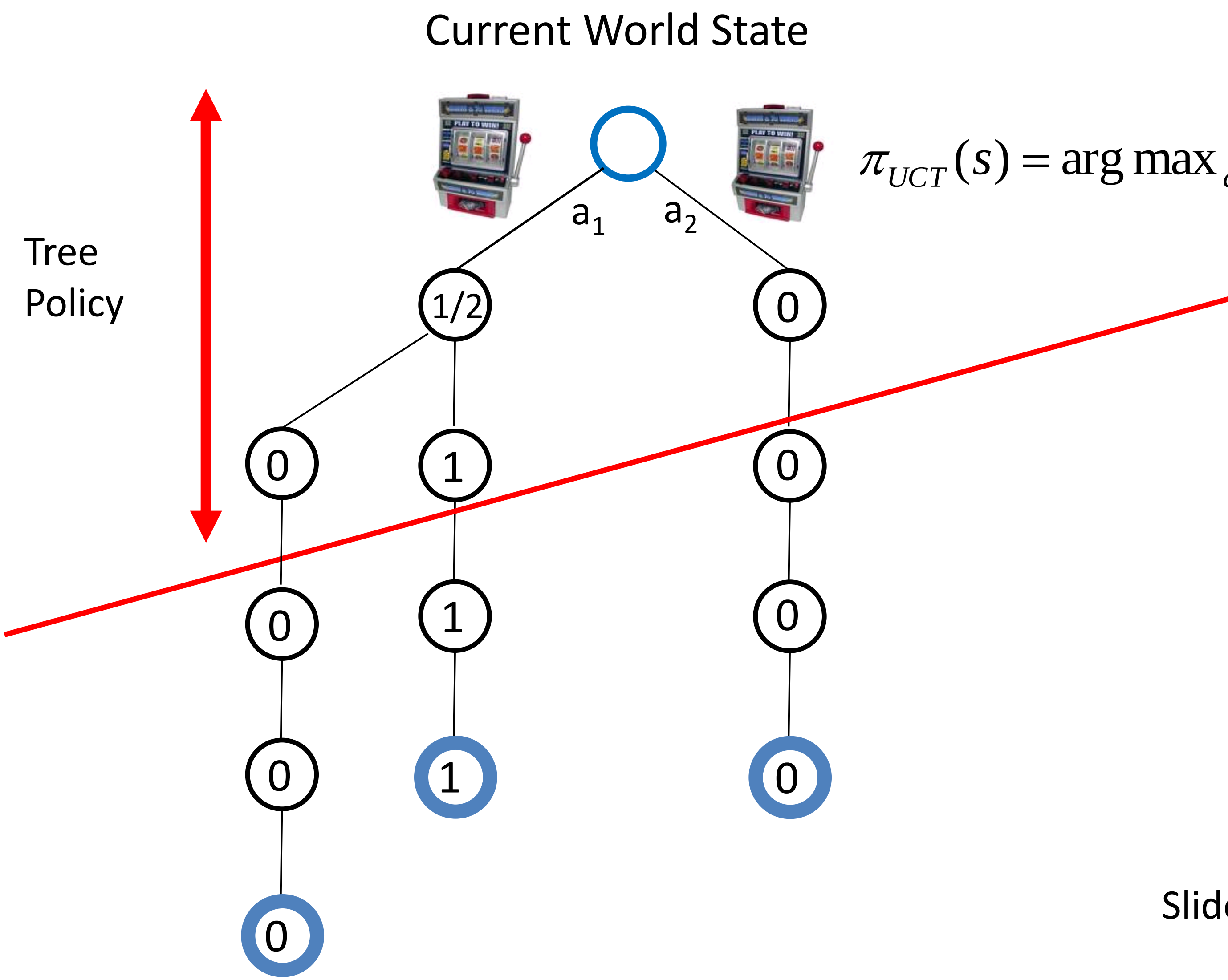
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\underbrace{\sqrt{\frac{\ln n(s)}{n(s,a)}}}_{\text{Exploration term}}$$

Theoretical constant that must be selected empirically in practice.

Slide courtesy of A. Fern

UCT Example

When all node actions tried once, select action according to tree policy



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Slide courtesy of A. Fern

UCT Summary & Theoretical Properties

- To select an action at a state s
  - Build a tree using N iterations of Monte-Carlo tree search
    - Default policy is uniform random up to level *L*
    - Tree policy is based on bandit rule
  - Select action that maximizes Q(s,a)
    (note that this final action selection does not take the exploration term into account, just the Q-value estimate)

- **The more simulations, the more accurate**
  - Guaranteed to pick suboptimal actions exponentially rarely after convergence (under some assumptions)

- **Possible improvements**
  - Initialize the state-action pairs with a heuristic (need to pick a weight)
  - Think of a better-than-random rollout policy

Slide courtesy of A. Fern

Approximation Algorithms

✓ Overview

✓ Online Algorithms
  – Determinization-based Algorithms
  – Monte-Carlo Planning

- Offline Algorithms
  – **Heuristic Search with Inadmissible Heuristics**
  – Dimensionality Reduction
  – Hierarchical Planning
  – Hybridized Planning

Moving on to Approximate Offline Planning

- Useful when there is no time to plan as you go …
  - E.g., when playing a fast-paced game

- … and not much time/space to plan in advance, either

- Like in online planning, oftern, no quality guarantees

- Some online methods (e.g., MCP) can be used offline too

Inadmissible Heuristic Search

- Why?
  - May require less space than admissible heuristic search
- Sometimes, intuitive suboptimal policies are small
  - E.g., taking a more expensive direct flight vs a cheaper 2-leg
- Apriori, no reason to expect an arbitrary inadmissible heuristic to yield a small solution
  - But, empirically, those based on determinization often do
- Same algos as for admissible HS, only heuristics differ

The FF Heuristic

Hoffmann and Nebel, 2001

- Taken directly from deterministic planning
  - A major component of the formidable FF planner

- Uses the all-outcome determinization of a PPDDL MDP
  - But ignores the *delete effects* (negative literals in action outcomes)
  - Actions never “unachieve” literals, always make progress to goal

- ***$h_{FF}(s)$ = approximate cost of a plan from s to a goal in the delete relaxation***

- Very fast due to using the delete relaxation

- Very informative

The GOTH Heuristic

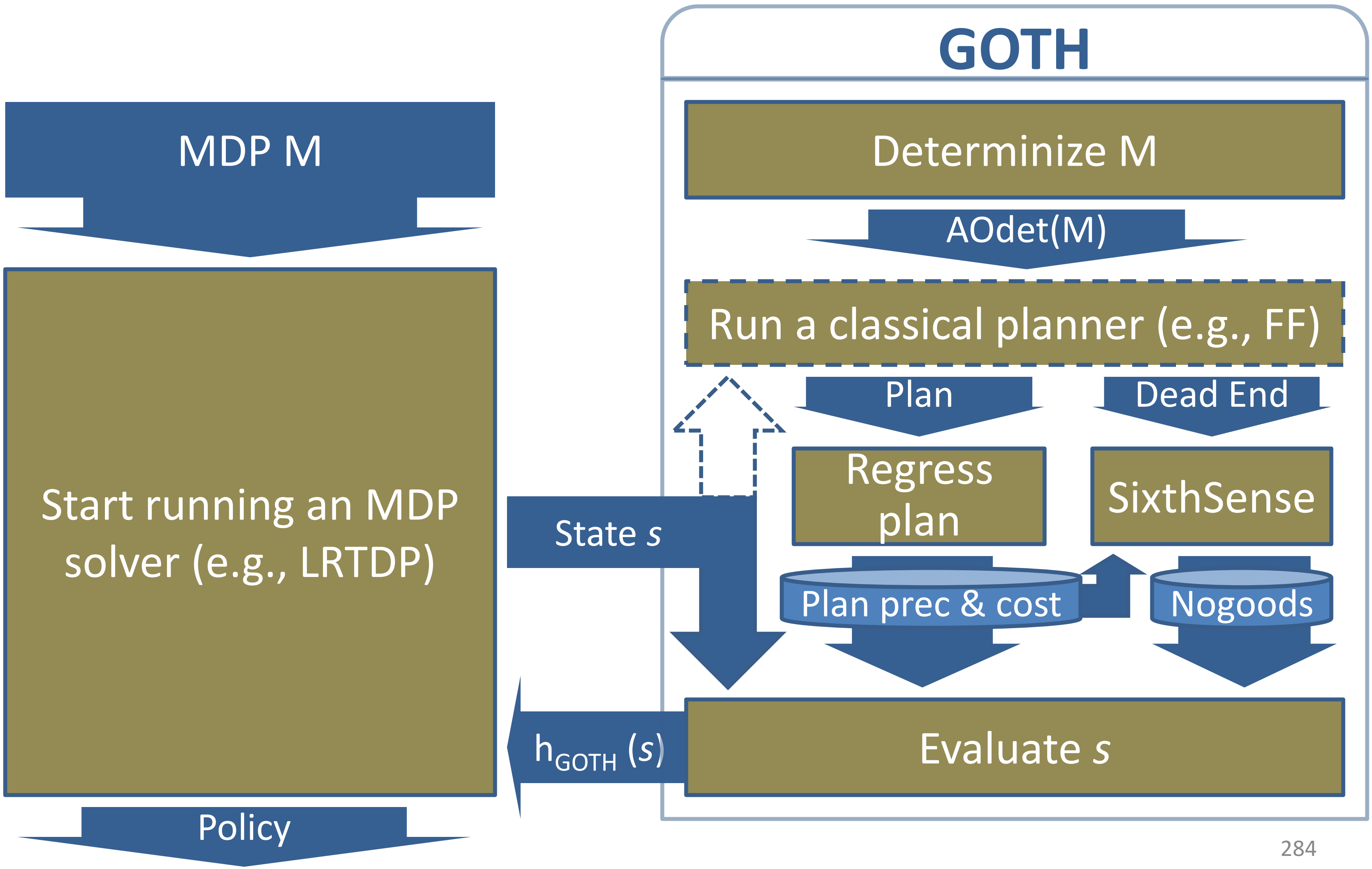
Kolobov, Mausam, Weld, 2010a

- Designed for MDPs at the start (not adapted classical)

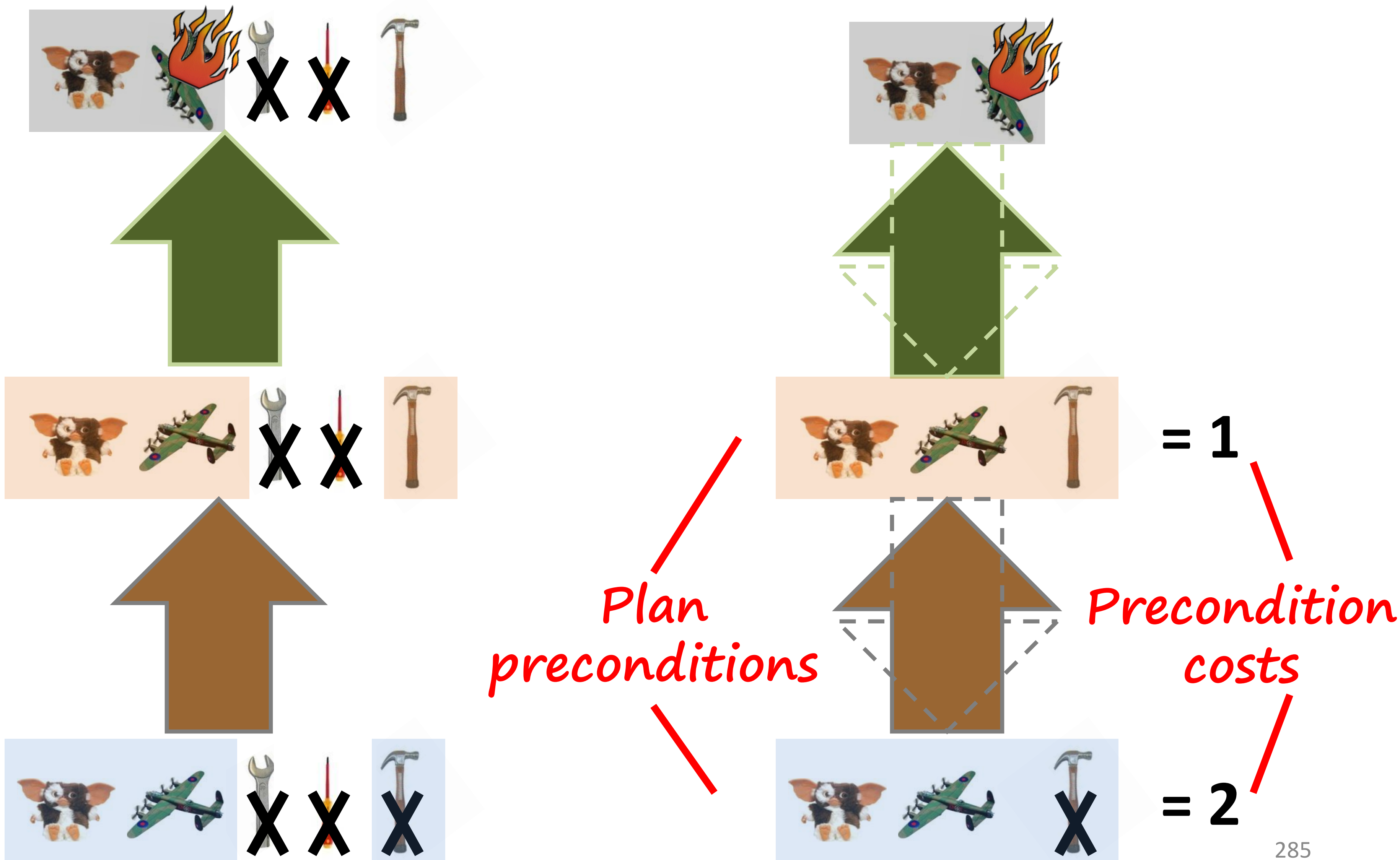
- Motivation: would be good to estimate *h(s)* as cost of a non-relaxed deterministic goal plan from *s*
  - But too expensive to call a classical planner from every *s*
  - Instead, call from only a few *s* and generalize estimates to others

- Uses AO determinization and the FF planner

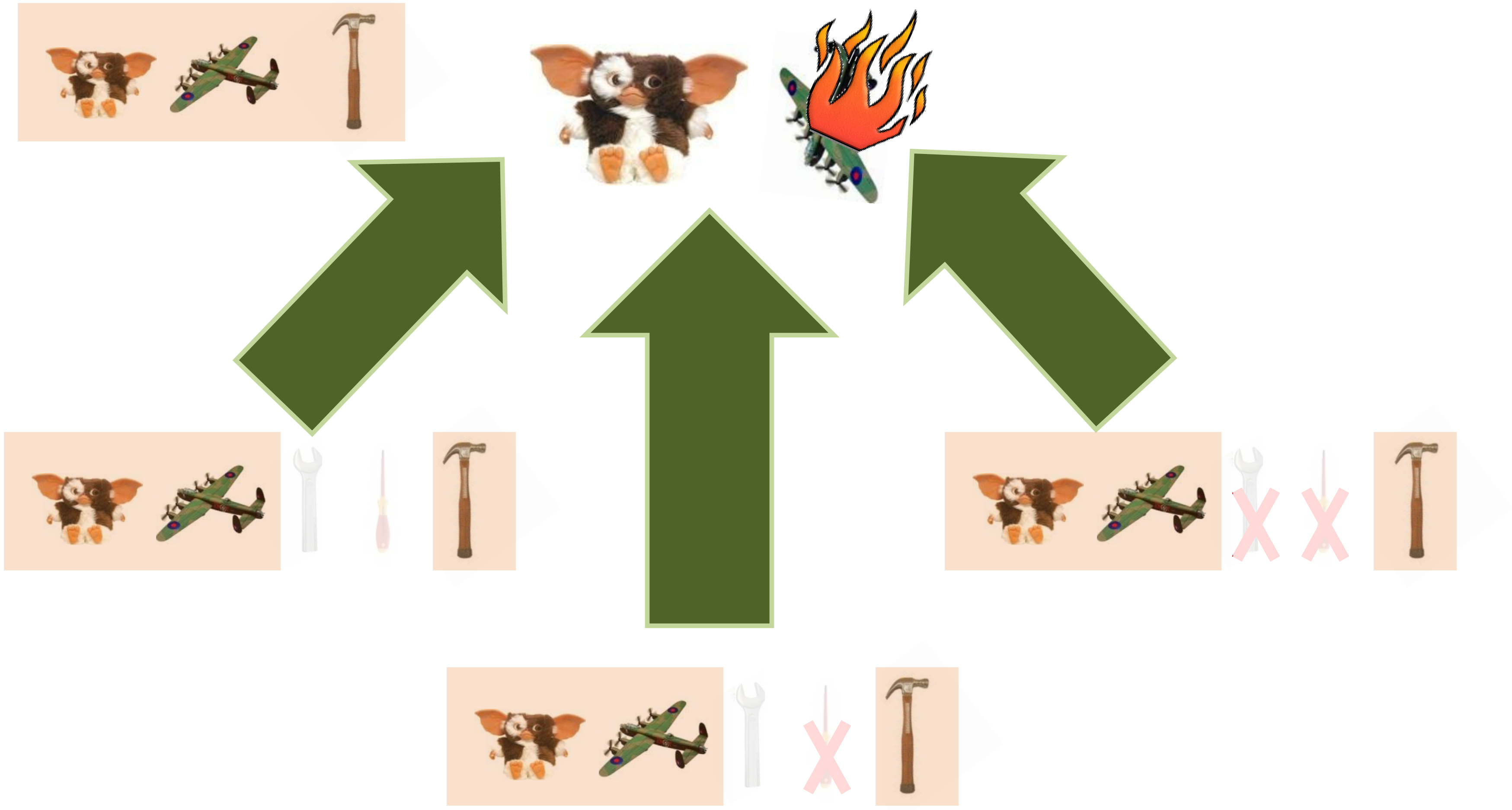
GOTH Overview



Regressing Trajectories

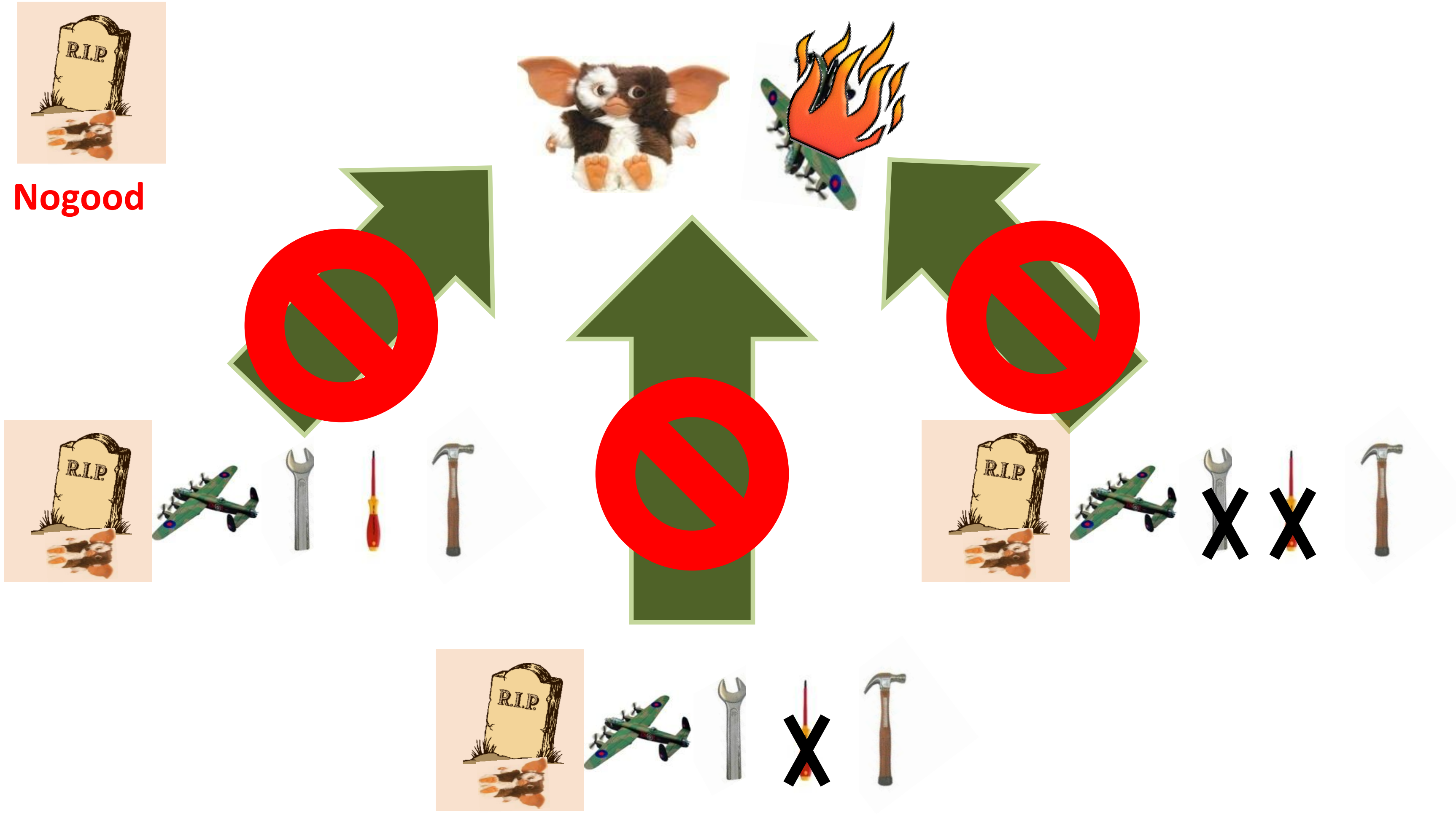


Plan Preconditions



Nogoods

Kolobov, Mausam, Weld, 2010b



Computing Nogoods

- Machine learning algorithm
  - Adaptively scheduled generate-and-test procedure
- Fast, sound
- Beyond the scope of this tutorial…

Estimating State Values

- Intuition
  - Each plan precondition cost is a "candidate" heuristic value
- Define $h_{GOTH}(s)$ as MIN of all available plan precondition values applicable in *s*
  - If none applicable in *s*, run a classical planner and find some
  - Amortizes the cost of classical planning across many states

Open Questions in Inadmissible HS

- h_{GOTH} is still much more expensive to compute than h_{FF}…

- … but also more informative, so LRTDP+h_{GOTH} is more space/time efficient than LRTDP+h_{FF} on most benchmarks

- **Still not clear when and why determinization-based inadmissible heuristics appear to work well**
  - Because they guide to goals along short routes?
  - Due to an experimental bias (MDPs with uniform action costs)?

- Need more research to figure it out…

Approximation Algorithms

✓ Overview

✓ Online Algorithms
  - Determinization-based Algorithms
  - Monte-Carlo Planning

- Offline Algorithms
  - Heuristic Search with Inadmissible Heuristics
  - **Dimensionality Reduction**
  - Hierarchical Planning
  - Hybridized Planning

Dimensionality Reduction: Motivation

- No approximate methods so far explicitly try to save space
  - Inadmissible HS can easily run out of memory
  - MCP runs out of space unless allowed to “forget” visited states

- Dimenstionality reduction attempts to do exactly that
  - Insight: V^* and π^* are functions of ~$|S|$ parameters (states)
  - Replace it with an approximation with r << |S| params …
  - … in order to save space

- **How to do it?**
  - Factored representations are crucial for this
  - View V/π as functions of state variables, not states themselves!

ReTrASE

Kolobov, Mausam, Weld, 2009

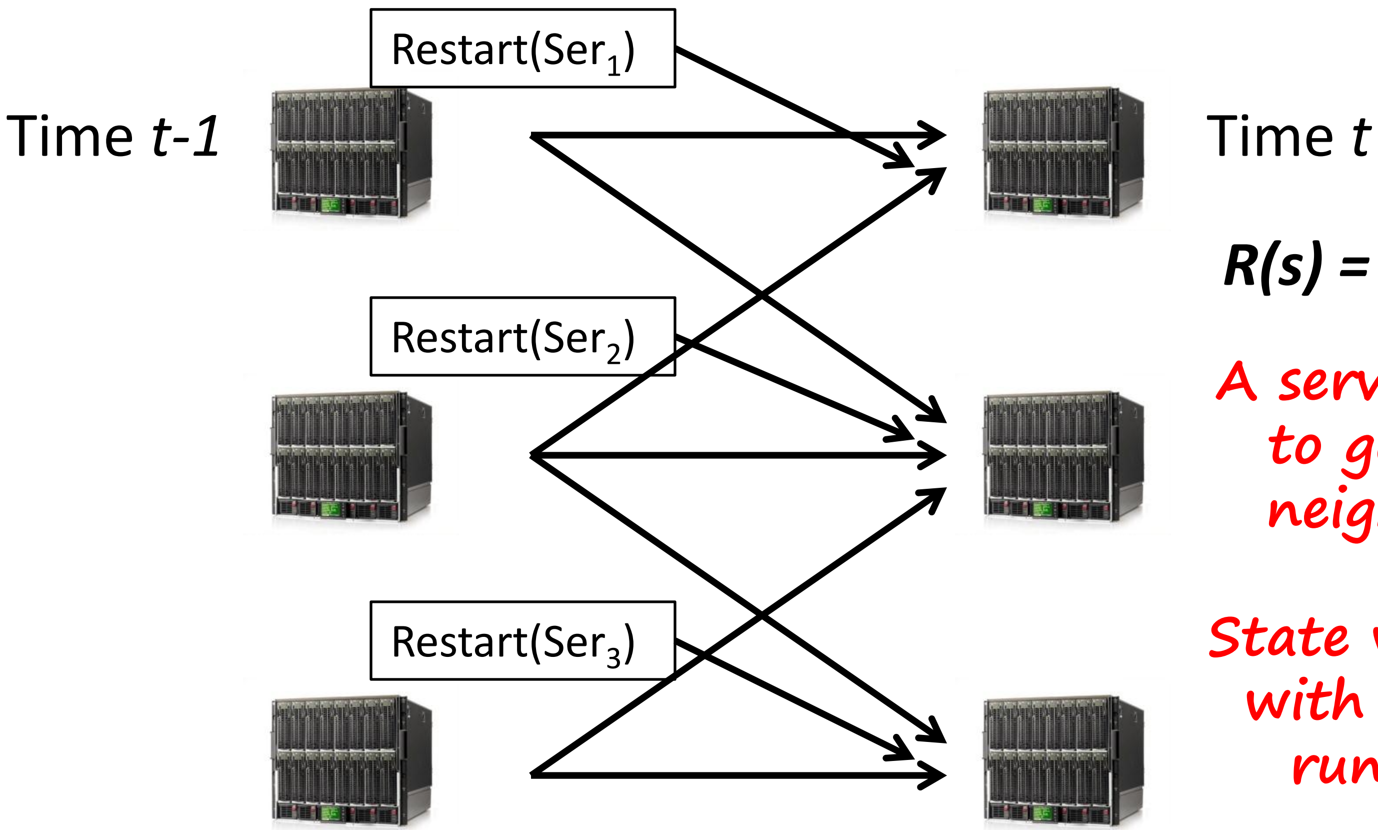
- Largely similar to h_{GOTH}
  - Uses preconditions of deterministic plan to evaluate states
- For each plan precondition p, defines a *basis function*
  - $B_p(s) = 1$ iff p holds in s, ∞ otherwise
- Represents $V(s) = min_p \, w_p B_p(s)$
  - Thus, the parameters are w_p for each basis function
  - Problem boils down to learning w_p
  - Does this with modified *RTDP*
- **Crucial observation: # plan preconditions sufficient for representing *V* is typically much smaller than |*S*|**
  - Because one plan precondition can hold in several states
  - Hence, the problem dimension is reduced!

ReTrASE Theoretical Properties

- Empirically, gives a large reduction in memory vs LRTDP
- Produces good policies (in terms of MAXPROB) when/if converges
- Not guaranteed to converge (weights may oscillate)
- No convergence detection/stopping criterion

Approximate PI/LP: Motivation

- ReTrASE considers a very restricted type of basis functions
  - Capture goal reachability information
  - Not appropriate in FH and IHDR MDPs; e.g., in Sysadmin:



$$\boldsymbol{R(s) = \sum_i [Ser_i = \uparrow]}$$

A server is less likely to go down if its neighbors are up

State value ~increases with the number of running servers!

Approximate PI/LP: Motivation

- Define basis function $b_i(s) = 1$ if $Ser_i = \uparrow$, 0 otherwise

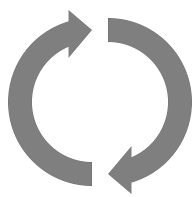
- In Sysadmin (and other MDPs), good to let $V(s)=\sum_i w_i\, b_i(s)$
  - A *linear value function approximation*

- If general, if a user gives a set B of basis functions, how do we pick $w_1, \ldots, w_{|B|}$ s.t. $|V^* - \sum_i w_i\, b_i|$ is the smallest?
  - Use API/ALP!

Approximate Policy Iteration

- **Assumes IHDR MDPs**

- Reminder: Policy Iteration
  - Policy evaluation
  - Policy improvement



- Approximate Policy Iteration
  - Policy evaluation: compute the best linear approx. of V^{π}
  - Policy improvement: same as for PI

Approximate Policy Iteration

Guestrin, Koller, Parr, Venkataraman, 2003

- To compute the best linear approximation, find

$$\vec{w}^{\pi} = \arg\min_{\vec{w}} \left[\max_{s \in \mathcal{S}} \left| \sum_{i=1}^{k} w_i b_i(s) - \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \left[\mathcal{R}(s, \pi(s), s') + \gamma \sum_{i=1}^{k} w_i b_i(s') \right] \right| \right]$$

A linear approximation to V^π

Bellman backup applied to the linear approximation V^π

- Linear program in $|B|$ variables and $2|S|$ constraints

- Does API converge?
  - In theory, no; can oscillate if linear approx. for some policies coincide
  - In practice, usually, yes
  - If converges, can bound solution quality

Approximate Linear Programming

- Same principle as API: replace V(s) with $\sum_i w_i \, b_i(s)$ in LP

- Linear program in $|B|$ variables and $|S||A|$ constraints

- **But wait a second…**
  - We have at least one constraint per state! Solution dimension is reduced, but finding solution is still at least linear in $|S|$!

Making API and ALP More Efficient

- **Insight: assume each *b* depends on at most *z* << |*X*| vars**

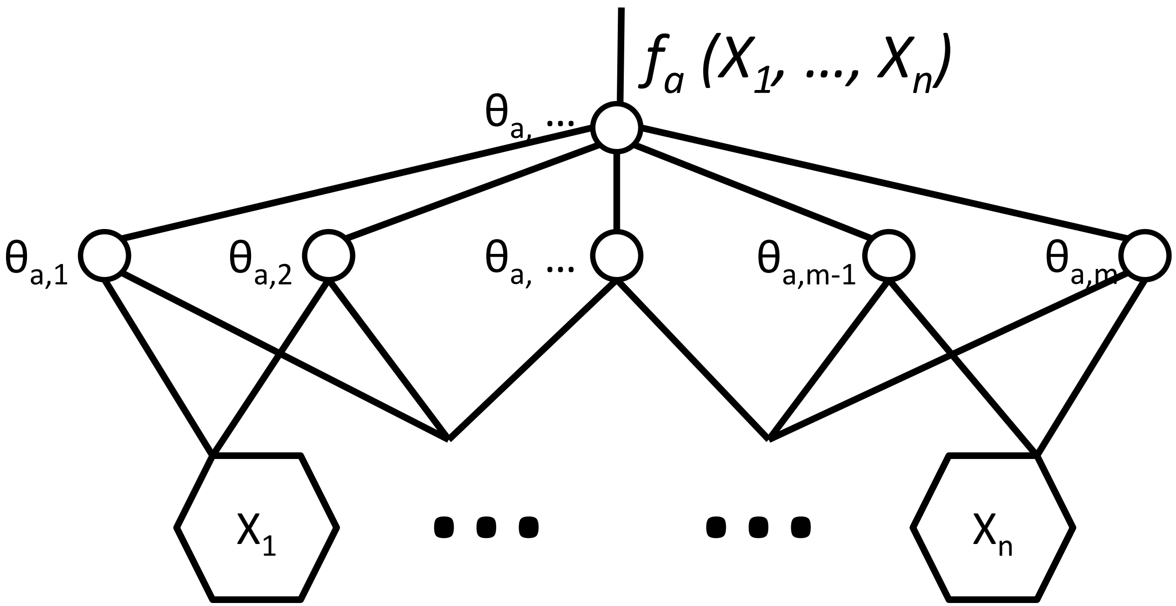
- Then, can reformulate LPs with only $O(2^z)$ constraints
  - Much smaller than $O(2^{|X|})$

- Very nontrivial…

FPG

[Buffet and Aberdeen, 2006, 2009]

- **Directly learns a policy, not a value function**
- For each action, defines a *desirability function*



- Mapping from state variable values to action "quality"
  - Represented as a neural network
  - Parameters to learn are network weights $\theta_{a,1}, \ldots, \theta_{a,m}$ for each *a*

FPG

- Policy (distribution over actions) is given by a softmax

$$\pi_{\vec{\theta}}(s,a) = \frac{e^{f_{a|\vec{\theta}_a}(s)}}{\sum_{a' \in \mathcal{A}} e^{f_{a'|\vec{\theta}_{a'}}(s)}},$$

- To learn the parameters:
  - Run trials (similar to RTDP)
  - After taking each action, compute the gradient w.r.t. weights
  - Adjust weights in the direction of the gradient
  - Makes actions causing expensive trajectories to be less desirable

FPG Details & Theoretical Properties

- Can speed up by using FF to guide trajectories to the goal
- Gradient is computed approximately
- Not guaranteed to converge to the optimal policy
- Nonetheless, works well

Approximation Algorithms

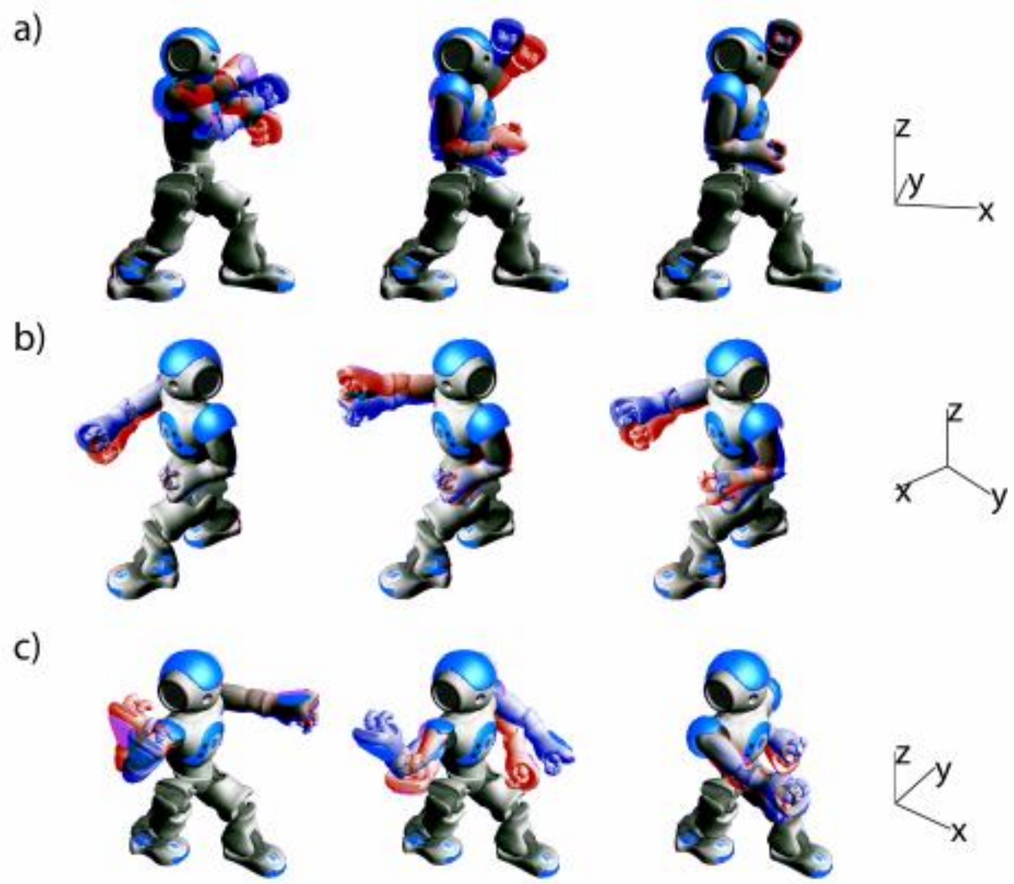
- ✓ Overview
- ✓ Online Algorithms
  - Determinization-based Algorithms
  - Monte-Carlo Planning
- Offline Algorithms
  - Heuristic Search with Inadmissible Heuristics
  - Dimensionality Reduction
  - **Hierarchical Planning**
  - Hybridized Planning

Hierarchical Planning: Motivation

- Some MDPs are too hard to solve w/o prior knowledge
  - Also, arbitrary policies for such MDPs may be hard to interpret
- Need a way to bias the planner towards “good” policies
  - And to help the planner by providing guidance
- **That’s what hierarchical planning does**
  - Given some prespecified (e.g., by the user) parts of a policy …
  - … planner “fills in the details”
  - Essentially, breaks up a large problem into smaller ones

Hierarchical Planning with Options

- Suppose a robot knows precomputed policies (*options*) for some primitive behaviors

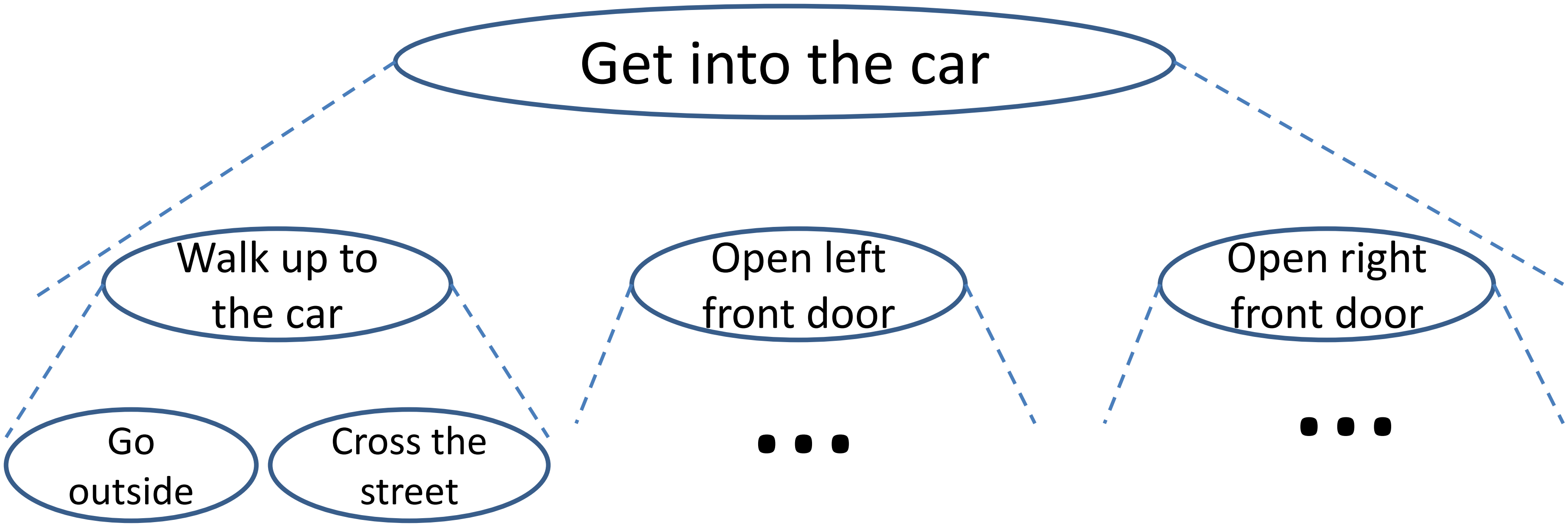


Hierarchical Planning with Options

- Options are almost like actions, but their transition function needs to be computed
- Suppose you want to teach the robot how to dance
- You provide a hierarchical planner with options for the robot's primitive behaviors
- Planner estimates the transition function and computes a policy for dancing that uses options as subroutines.

Task Hierarchies

- The user breaks down a task into a hierarchy of subgoals



- The planner chooses which subgoals to achieve at each level, and how
  - Subgoals are just hints
  - Not all subgoals may be necessary to achieve the higher-level goal

Hierarchies of Abstract Machines (HAMs)

- More general hierarchical representation
- Each machine is a finite-state automaton w/ 4 node types

Execute action *a*

Call another machine *H*

Choose a machine from $\{H_1, \dots, H_n\}$ and execute it

Stop execution/return control to a higher-level machine

- The user supplies a HAM
- The planner needs to decide what to do in *choice nodes*

Optimality in Hierarchical Planning

- Hierarchy constraints may disallow globally optimal π^*
- Next-best thing: a *hierarchically optimal policy*
  - The best policy obeying the hierarchy constraints
  - Not clear how to find it efficiently
- A more practical notion: a *recursively optimal policy*
  - A policy optimal at every hierarchy level, assuming that policies at lower hierarchy levels are fixed
  - Optimization = finding optimal policy starting from lowest level
- **Hierarchically optimal doesn't imply recursively optimal, and v. v.**
  - But hierarchically optimal is always at least as good as recursively optimal

Learning Hierarchies

- Identifying useful subgoals
  - States in “successful” and not in “unsuccessful” trajectories
  - Such states are similar to *landmarks*

- Breaking up an MDP into smaller ones
  - State abstraction (removing variables irrelevant to the subgoal)

- **Still very much an open problem**

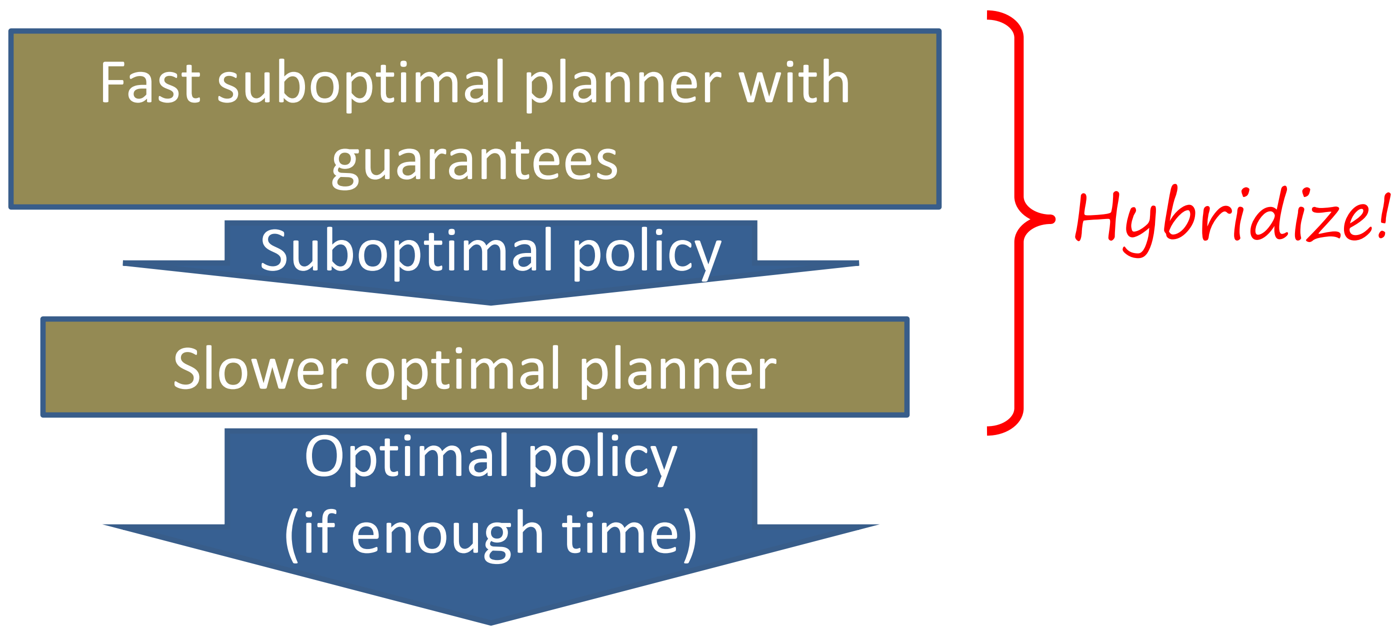
Approximation Algorithms

- ✓ Overview
- ✓ Online Algorithms
  - Determinization-based Algorithms
  - Monte-Carlo Planning

- Offline Algorithms
  - Heuristic Search with Inadmissible Heuristics
  - Dimensionality Reduction
  - Hierarchical Planning
  - **Hybridized Planning**

Hybridized Planning: Motivation

- Sometimes, need to arrive at a provably “reasonable” (but possibly suboptimal) solution ASAP



Hybridized Planning

[Mausam, Bertoli, Weld, 2007]

- **Hybridize MBP and LRTDP**
- MBP is a non-deterministic planner
  - Gives a policy guaranteed to reach the goal from everywhere
  - Very fast
- LRTDP is an optimal probabilistic planner
  - Amends MBP's solution to have a good expected cost
- **Optimal in the limit, produces a proper policy quickly**

Summary

- Surveyed 6 different approximation families
  - Dimensionality reduction
  - Monte-Carlo sampling
  - Inadmissible heuristic search
  - Dimensionality reduction
  - Hierarchical planning
  - Hybridized planning
- Sacrifice different solution quality aspects
- Lots of work to be done in each of these areas

Outline of the Tutorial

- *Introduction* *(10 mins)*
- *Fundamentals of MDPs* *(1+ hr)*
- *Uninformed Algorithms* *(1 hr)*
- *Heuristic Search Algorithms* *(1 hr)*
- *Approximation Algorithms* *(1+ hr)*
- *Extension of MDPs* *(remaining time)*

One set of techniques we didn't cover

- Compact value function representations

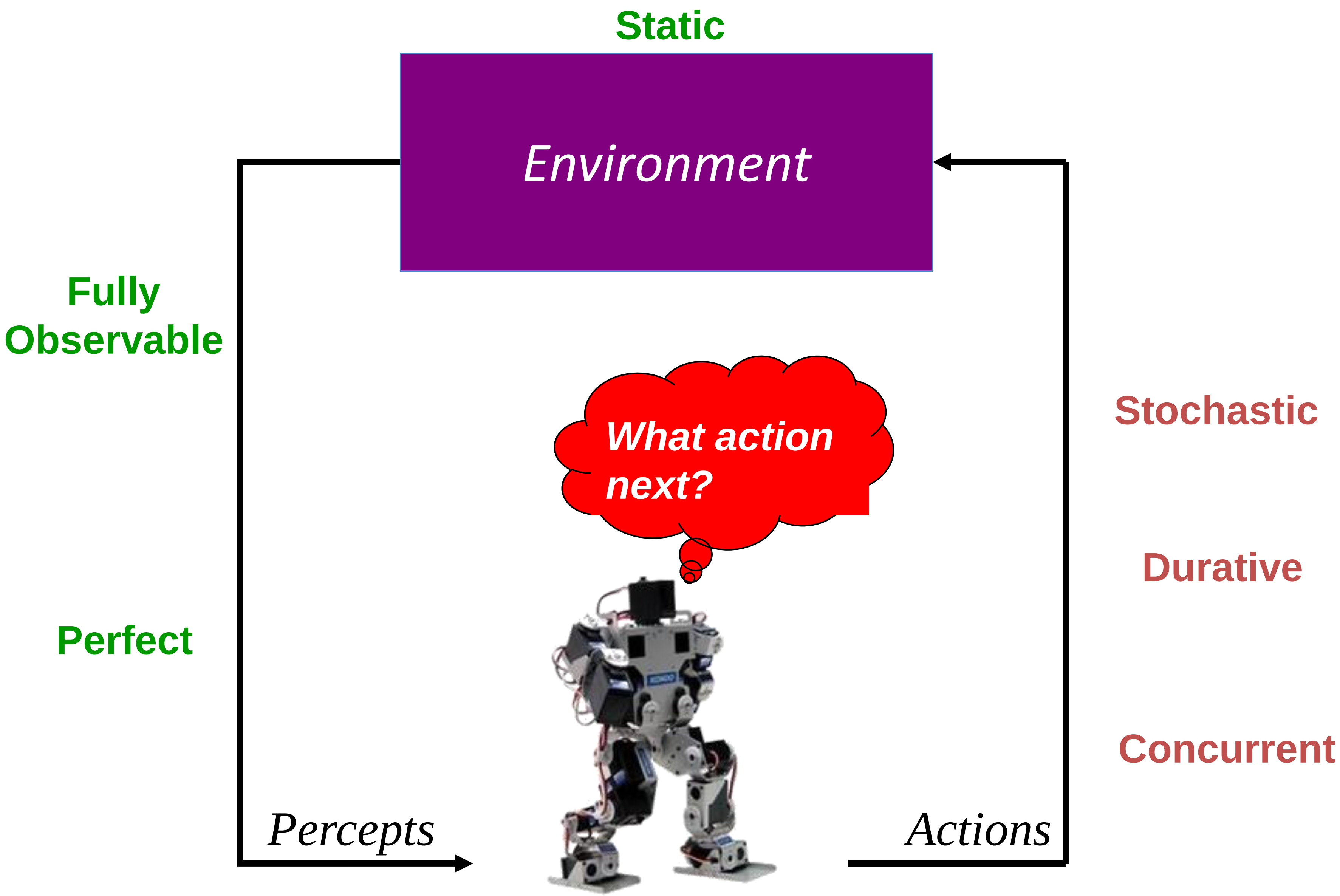
- ADD-based planners
  - Symbolic VI (SPUDD)
  - Symbolic Prioritized Sweeping
  - Symbolic LAO*
  - Symbolic RTDP
  - Approximations (APRICODD)

- Better representations: Affine ADDs.

Continuous State/Action MDPs

- See Scott's Tutorial.

Concurrent Probabilistic Temporal Planning



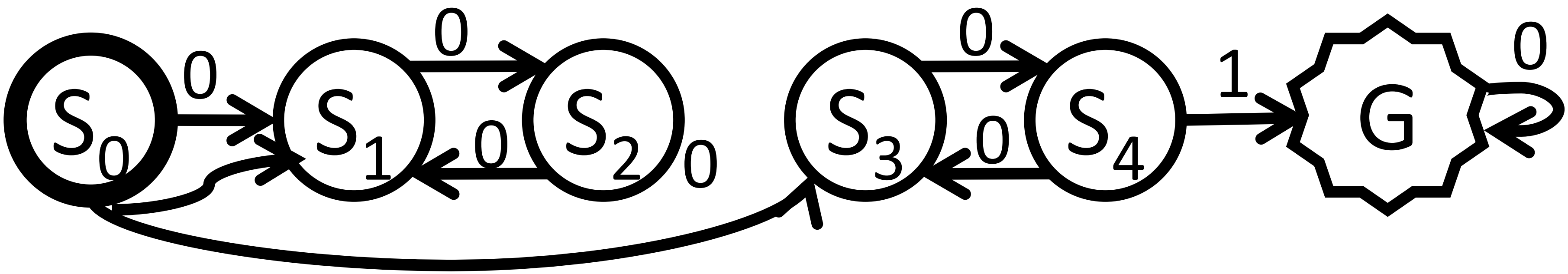
Results

- MDPs with Durative Actions, No Concurrency
  - VI, RTDP, Incremental Contingency Planning
  - Simple Temporal Nets, Piecewise linear vfs…
- MDPs with Concurrent Actions, No Time
  - CoMDPs
  - Action Elimination, ALP, Hierarchical planning, …
- MDPs with Concurrent, Durative Actions
  - Generalized Semi-Markov Decision Process (GSMDP)
  - Augmented state MDPs, Generate-test-debug, hybridized planning…

Relational MDPs

- PPDDL/RDDL are first-order representations
  - Algorithms ground it into propositional domains
- Relational MDPs actively use first-order structure
  - First-order VI, PI, ALP
  - Inductive approaches
- Generalizes to many problems
  - with variable number of objects

Well-formed MDPs beyond SSPs



- All improper policies may not have infinite cost
- VI doesn't work
  - has multiple fixed points
  - greedy policy over optimal value may not be optimal
- Heuristic Search much trickier
- Generalized SSP MDPs [Kolobov et al 11]
- Stochastic Safest & Shortest Path [Teichteil-Konigsbuch 12]
- Fun recent work…

Other Models

- Reinforcement Learning
  - model/costs unknown
  - Monte-Carlo planning
- Partially Observable MDP
  - MDP with incomplete state information
  - Large Continuous MDP
  - Lots of applications
- Multi-objective MDP
- MDPs with Imprecise Probabilities
- Collaborative Multi-agent MDPs
- Adversarial Multi-agent MDPs

Thanks!

Mausam and Andrey Kolobov

"*Planning with Markov Decision Processes: An AI Perspective*"

Morgan and Claypool Publishers (Synthesis Lectures Series on Artificial Intelligence)