

AAMAS
2013

Saint Paul
Minnesota
USA

6th - 10th May 2013

twelfth
international
conference
on
autonomous
agents and
multiagent
systems

T8 - Cooperative Decision Making in Sequential Multiagent Settings

Amato, Doshi, Oliehoek,
Rabinovich, Spaan, Witwicki

May 6

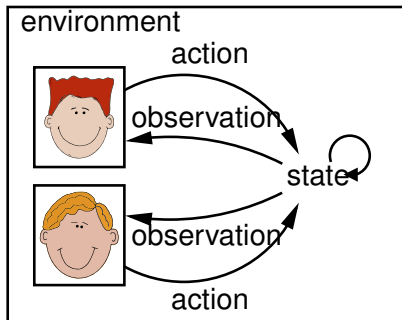
Overview

Time	Topics	Speaker
8:30 - 10:30	Intro, single-agent planning, multiagent models	Matthijs Spaan
11:00 - 12:30	Optimal Dec-POMDP solvers	Frans Oliehoek
13:30 - 15:30	Approximate solvers	Chris Amato
	Structured problems	Stefan Witwicki
16:00 - 17:30	Communication, learning, applications	

Introduction

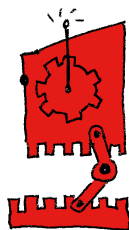
Introduction

- ▶ AI: develop intelligent agents.
- ▶ Cooperating multiagent systems.
- ▶ Problem: planning how to act.
- ▶ Joint payoff but decentralized actions and observations.



Agents

- ▶ An agent is a (rational) decision maker who is able to perceive its external (physical) environment and act autonomously upon it (Russell and Norvig, 2003).
- ▶ Rationality means reaching the optimum of a performance measure.
- ▶ Examples: humans, robots, some software programs.



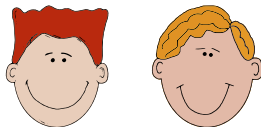
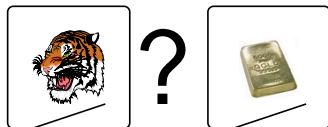
Application domains

Possible application domains:

- ▶ **Multi-robot coordination**
 - ▶ **Space exploration rovers** (Zilberstein et al., 2002)
 - ▶ **Helicopter flights** (Pynadath and Tambe, 2002)
 - ▶ **Navigation** (Emery-Montemerlo et al., 2005; Spaan and Melo, 2008)
- ▶ **Load balancing for decentralized queues** (Cogill et al., 2004)
- ▶ **Multi-access broadcast channels** (Ooi and Wornell, 1996)
- ▶ **Network routing** (Peshkin and Savova, 2002)
- ▶ **Sensor network management** (Nair et al., 2005)

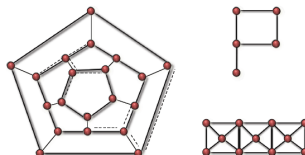
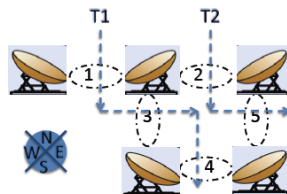
Example: The DEC-Tiger problem

- ▶ A toy problem:
decentralized tiger (Nair et al., 2003).
- ▶ Opening correct door:
both receive treasure.
- ▶ Opening wrong door:
both get attacked by a tiger.
- ▶ Agents can open a door,
or listen.
- ▶ Two noisy observations:
hear tiger left or right.
- ▶ Don't know the other's
actions or observations.



Example: Sensor network problems

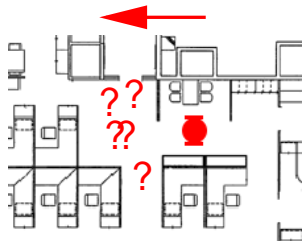
- ▶ Sensor networks for
 - ▶ Target tracking (Nair et al., 2005; Kumar and Zilberstein, 2009a)
 - ▶ Weather phenomena (Kumar and Zilberstein, 2009b)
- ▶ Two or more cooperating sensors.



Decision-theoretic planning

- ▶ Decision-theoretic planning tackles uncertainty in sensing and acting in a principled way.

- ▶ We need to model:
 - ▶ each agent's actions
 - ▶ their sensors
 - ▶ their environment
 - ▶ their task



- ▶ Popular for single-agent planning under uncertainty (MDPs, POMDPs).

Decision-theoretic planning

Assumptions:

- ▶ Sequential decisions: problems are formulated as a sequence of discrete “independent” decisions.
- ▶ Markovian environment: the state at time t depends only on the events at time $t - 1$.
- ▶ Stochastic models: the uncertainty about the outcome of actions and sensing can be accurately captured.
- ▶ Objective encoding: the overall objective can be encoded using cumulative (discounted) rewards over time steps.

Multiagent planning problems

Aspects:

- ▶ on-line vs. off-line
- ▶ centralized vs. distributed
 - ▶ planning
 - ▶ execution
- ▶ cooperative vs. self-interested
- ▶ observability
- ▶ communication

Related previous work

- ▶ **Group decision theory in economics, team theory** (Marschak, 1955; Papadimitriou and Tsitsiklis, 1982)
- ▶ **Decentralized detection** (Tsitsiklis and Athans, 1985; Tsitsiklis, 1988)
- ▶ **Optimization of decentralized systems in operations research** (Witsenhausen, 1971; Sandell et al., 1978)
- ▶ **Communication strategies** (Varaiya and Walrand, 1978; Xuan et al., 2001; Pynadath and Tambe, 2002)
- ▶ **Approximation algorithms** (Peshkin et al., 2000; Guestrin et al., 2002; Nair et al., 2003; Emery-Montemerlo et al., 2004)

Planning

Planning:

- ▶ A plan tells an agent how to act.
- ▶ For instance
 - ▶ A sequence of actions to reach a goal.
 - ▶ What to do in a particular situation.
- ▶ We need to model:
 - ▶ the agent's actions
 - ▶ its environment
 - ▶ its task

We will model planning as a sequence of decisions.

Classic planning



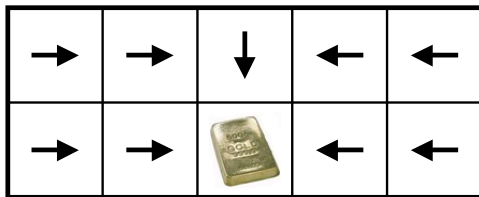
- ▶ Classic planning: sequence of actions from start to goal.
- ▶ Task: robot should get to gold as quickly as possible.
- ▶ Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- ▶ Limitations:
 - ▶ New plan for each start state.
 - ▶ Environment is deterministic.

Classic planning



- ▶ Classic planning: sequence of actions from start to goal.
- ▶ Task: robot should get to gold as quickly as possible.
- ▶ Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- ▶ Limitations:
 - ▶ New plan for each start state.
 - ▶ Environment is deterministic.
- ▶ Three optimal plans: $\rightarrow \rightarrow \downarrow$, $\rightarrow \downarrow \rightarrow$, $\downarrow \rightarrow \rightarrow$.

Conditional planning



- ▶ Assume our robot has noisy actions (wheel slip, overshoot).
- ▶ We need conditional plans.
- ▶ Map situations to actions.


Decision-theoretic planning

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

- ▶ Positive reward when reaching goal, small penalty for all other actions.
- ▶ Agent's plan maximizes **value**: the sum of future rewards.
- ▶ Decision-theoretic planning successfully handles noise in acting and sensing.

Decision-theoretic planning

Plan #1:

→	→	↓		
				

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

Decision-theoretic planning

Values of this plan:

?	?	?		
		10		

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

Decision-theoretic planning

Values of this plan:


9.7	9.8	9.9		
		10		

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

Decision-theoretic planning

Plan #2:

→	→	→	→	↓
			←	←

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

Decision-theoretic planning

Values of this plan:

?	?	?	?	?
		10	?	?

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

Decision-theoretic planning

Values of this plan:

9.3	9.4	9.5	9.6	9.7
		10	9.9	9.8

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

Decision-theoretic planning

Optimal values (encode optimal plan):

9.7	9.8	9.9	9.8	9.7
9.8	9.9	10	9.9	9.8

Reward:

-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	10	-0.1	-0.1

References I

- R. Cogill, M. Rotkowitz, B. V. Roy, and S. Lall. An approximate dynamic programming approach to decentralized control of stochastic systems. In *Proceedings of the 2004 Allerton Conference on Communication, Control, and Computing*, 2004.
- R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2004.
- R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Game theoretic control for robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.
- C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.
- A. Kumar and S. Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2009a.
- A. Kumar and S. Zilberstein. Event-detecting multi-agent MDPs: Complexity and constant-factor approximation. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2009b.
- J. Marschak. Elements for a theory of teams. *Management Science*, 1(2):127–137, 1955.
- R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003.
- R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. 2005.
- J. M. Ooi and G. W. Wornell. Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proc. of the 35th Conference on Decision and Control*, 1996.
- C. H. Papadimitriou and J. N. Tsitsiklis. On the complexity of designing distributed protocols. *Information and Control*, 53(3):211–218, 1982.
- L. Peshkin and V. Savova. Reinforcement learning for adaptive routing. In *Proc. of the Int. Joint Conf. on Neural Networks*, 2002.
- L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proc. of Uncertainty in Artificial Intelligence*, 2000.

References II

- D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.
- S. J. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd edition, 2003.
- J. Sandell, N., P. Varaiya, M. Athans, and M. Safonov. Survey of decentralized control methods for large scale systems. *IEEE Transactions on Automatic Control*, 23(2):108–128, Apr 1978.
- M. T. J. Spaan and F. S. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 525–532, 2008.
- J. Tsitsiklis. Decentralized detection by a large number of sensors. *Mathematics of Control, Signals and Systems*, 1(2):167–182, 1988.
- J. Tsitsiklis and M. Athans. On the complexity of decentralized decision making and detection problems. *IEEE Transactions on Automatic Control*, 30(5):440–446, 1985.
- P. Varaiya and J. Walrand. On delayed sharing patterns. *IEEE Transactions on Automatic Control*, 23(3):443–445, 1978.
- H. Witsenhausen. Separation of estimation and control for discrete time systems. *Proceedings of the IEEE*, 59(11):1557–1566, Nov. 1971.
- P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proc. of the Fifth Int. Conference on Autonomous Agents*, 2001.
- S. Zilberstein, R. Washington, D. Bernstein, and A. Mouaddib. Decision-theoretic control of planetary rovers. In *Plan-Based control of Robotic Agents*, volume 2466 of *LNAI*, pages 270–289. Springer, 2002.

Markov Decision Processes

Sequential decision making under uncertainty

- ▶ Uncertainty is abundant in **real-world planning** domains.
- ▶ **Bayesian** approach \Rightarrow probabilistic models.



Main assumptions:

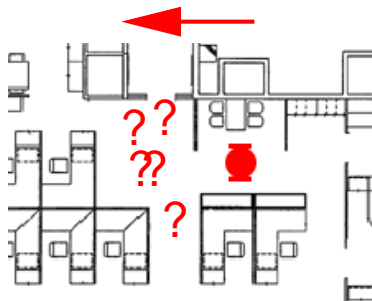
Sequential decisions: problems are formulated as a sequence of “independent” decisions;

Markovian environment: the state at time t depends only on the events at time $t - 1$;

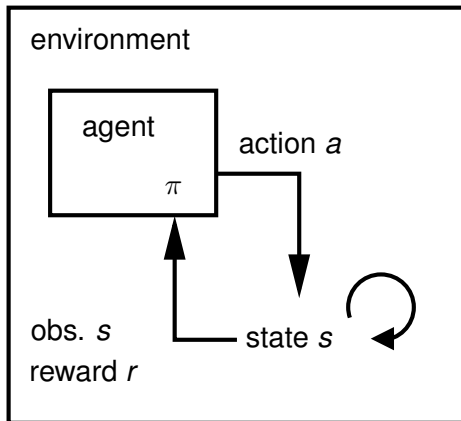
Evaluative feedback: use of a reinforcement signal as performance measure (reinforcement learning);

Transition model

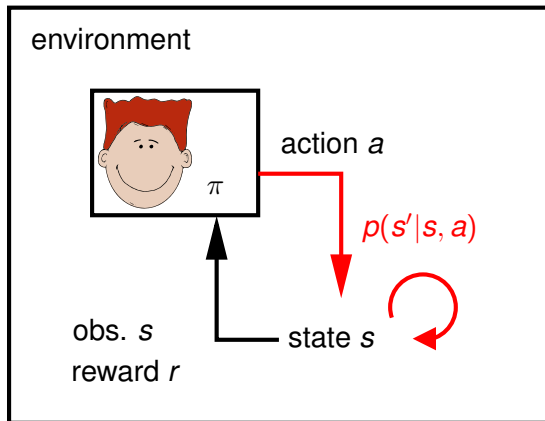
- ▶ For instance, robot motion is inaccurate.
- ▶ Transitions between states are **stochastic**.
- ▶ $p(s'|s, a)$ is the probability to jump from state s to state s' after taking action a .



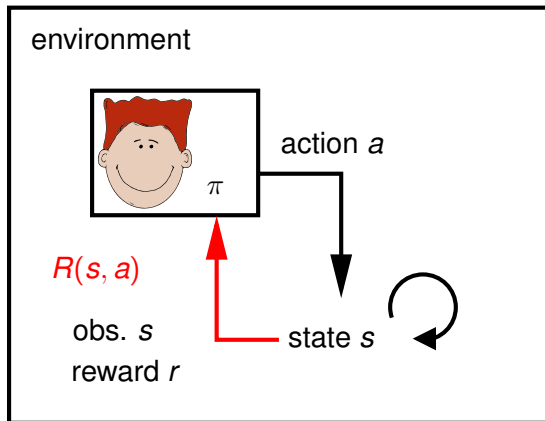
MDP Agent



MDP Agent



MDP Agent



Optimality criterion

For instance, agent should maximize the value

$$E\left[\sum_{t=0}^h \gamma^t R_t\right], \quad (1)$$

where

- ▶ h is the planning horizon, can be finite or ∞
- ▶ γ is a discount rate, $0 \leq \gamma < 1$

Reward hypothesis (Sutton and Barto, 1998):

All goals and purposes can be formulated as the maximization of the cumulative sum of a received scalar signal (reward).

Discrete MDP model

Discrete Markov Decision Process model (Puterman, 1994; Bertsekas, 2000):

- ▶ Time t is discrete.
- ▶ State space S .
- ▶ Set of actions A .
- ▶ Reward function $R : S \times A \mapsto \mathbb{R}$.
- ▶ Transition model $p(s'|s, a)$, $T_a : S \times A \mapsto \Delta(S)$.
- ▶ Initial state s_0 is drawn from $\Delta(S)$.

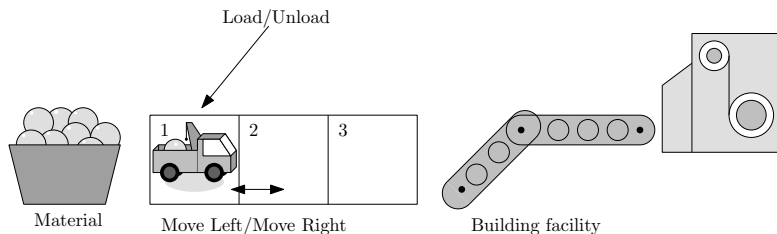
The Markov property entails that the next state s_{t+1} only depends on the previous state s_t and action a_t :

$$p(s_{t+1} | s_t, s_{t-1}, \dots, s_0, a_t, a_{t-1}, \dots, a_0) = p(s_{t+1} | s_t, a_t). \quad (2)$$

A simple problem

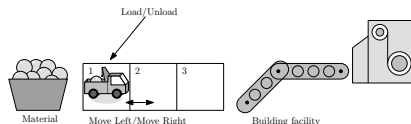
Problem:

An autonomous robot must learn how to transport material from a deposit to a building facility.



(thanks to F. Melo)

Load/Unload as an MDP



- States: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$;
 - 1_U Robot in position 1 (unloaded);
 - 2_U Robot in position 2 (unloaded);
 - 3_U Robot in position 3 (unloaded);
 - 1_L Robot in position 1 (loaded);
 - 2_L Robot in position 2 (loaded);
 - 3_L Robot in position 3 (loaded)
- Actions: $A = \{\text{Left}, \text{Right}, \text{Load}, \text{Unload}\}$;

Load/Unload as an MDP (1)

- ▶ Transition probabilities: “Left”/“Right” move the robot in the corresponding direction; “Load” loads material (only in position 1); “Unload” unloads material (only in position 3).

Ex:

$$(2_L, \text{Right}) \rightarrow 3_L;$$

$$(3_L, \text{Unload}) \rightarrow 3_U;$$

$$(1_L, \text{Unload}) \rightarrow 1_L.$$

- ▶ Reward: We assign a reward of +10 for every unloaded package (payment for the service).

Load/Unload as an MDP (2)

- ▶ For each action $a \in A$, T_a is a matrix.

Ex:

$$T_{\text{Right}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- ▶ Recall: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$.

Load/Unload as an MDP (3)

- ▶ The reward $R(s, a)$ can also be represented as a matrix
Ex:

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +10 \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Policies and value

- ▶ Policy π : tells the agent how to act.
- ▶ A deterministic policy $\pi : S \mapsto A$ is a mapping from states to actions.
- ▶ Value: how much reward $E[\sum_{t=0}^h \gamma^t R_t]$ does the agent expect to gather.
- ▶ Value denoted as $Q^\pi(s, a)$: start in s , do a and follow π afterwards.

Policies and value (1)

- ▶ Extracting a policy π from a value function Q is easy:

$$\pi(s) = \arg \max_{a \in A} Q(s, a). \quad (3)$$

- ▶ Optimal policy π^* : one that maximizes $E[\sum_{t=0}^h \gamma^t R_t]$ (for every state).
- ▶ In an infinite-horizon MDP there is always an optimal deterministic stationary (time-independent) policy π^* .
- ▶ There can be many optimal policies π^* , but they all share the same optimal value function Q^* .

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad Q_1 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad Q_1 = \begin{bmatrix} 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 \\ 0 & ? & ? & 10 \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \quad Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9.5 & 0 & 0 \\ 0 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_5 = \begin{bmatrix} 0 & 0 & 8.57 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 8.57 & 9.03 & 8.57 & 8.57 \\ 8.57 & 9.5 & 9.03 & 9.03 \\ 9.03 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Iterations of DP:

$$Q_{20} = \begin{bmatrix} 18.53 & 17.61 & 19.51 & 18.54 \\ 18.53 & 16.73 & 17.61 & 17.61 \\ 17.61 & 16.73 & 16.73 & 16.73 \\ 19.51 & 20.54 & 19.51 & 19.51 \\ 19.51 & 21.62 & 20.54 & 20.54 \\ 20.54 & 21.62 & 21.62 & 26.73 \end{bmatrix}$$

$$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}, A = \{\text{Left, Right, Load, Unload}\}$$

Dynamic programming

Final Q^* and policy:

$$Q^* = \begin{bmatrix} 30.75 & 29.21 & 32.37 & 30.75 \\ 30.75 & 27.75 & 29.21 & 29.21 \\ 29.21 & 27.75 & 27.75 & 27.75 \\ 32.37 & 34.07 & 32.37 & 32.37 \\ 32.37 & 35.86 & 34.07 & 34.07 \\ 34.07 & 35.86 & 35.86 & 37.75 \end{bmatrix} \quad \pi^* = \begin{bmatrix} \text{Load} \\ \text{Left} \\ \text{Left} \\ \text{Right} \\ \text{Right} \\ \text{Unload} \end{bmatrix}$$

Value iteration

- ▶ Value iteration: successive approximation technique.
- ▶ Start with all values set to 0.
- ▶ In order to consider one step deeper into the future, i.e., to compute V_{n+1}^* from V_n^* :

$$Q_{n+1}^*(s, a) := R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} Q_n^*(s', a'), \quad (4)$$

which is known as the dynamic programming update or Bellman backup.

- ▶ Bellman (1957) equation:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} Q^*(s', a'). \quad (5)$$

Value iteration (1)

Initialize Q arbitrarily, e.g., $Q(s, a) = 0, \forall s \in S, a \in A$

repeat

$\delta \leftarrow 0$

for all $s \in S, a \in A$ **do**

$v \leftarrow Q(s, a)$

$Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} Q(s', a')$

$\delta \leftarrow \max(\delta, |v - Q(s, a)|)$

end for

until $\delta < \epsilon$

Return Q

Value iteration (2)

Value iteration discussion:

- ▶ As $n \rightarrow \infty$, value iteration converges.
- ▶ Value iteration has converged when the largest update δ in an iteration is below a certain threshold ϵ .
- ▶ Exhaustive sweeps are not required for convergence, provided that in the limit all states are visited infinitely often.
- ▶ This can be exploited by backing up the most promising states first, known as prioritized sweeping.

Solution methods: MDPs

Model based

- ▶ Basic: dynamic programming (Bellman, 1957), value iteration, policy iteration.
- ▶ Advanced: prioritized sweeping, function approximators.

Model free, reinforcement learning (Sutton and Barto, 1998)

- ▶ Basic: Q-learning, $TD(\lambda)$, SARSA, actor-critic.
- ▶ Advanced: generalization in infinite state spaces, exploration/exploitation issues.

POMDPs

Beyond MDPs

- ▶ Real agents cannot directly observe the state.
- ▶ Sensors provide partial and noisy information about the world.

Beyond MDPs

- ▶ MDPs have been very successful, but requires to have an observable Markovian state.
- ▶ Many domains this is impossible (or expensive) to obtain:
- ▶ Diagnosis (medical, maintenance)
- ▶ Robot navigation
- ▶ Tutoring
- ▶ Dialog systems
- ▶ Vision-based robotics
- ▶ Fault recovery

Beyond MDPs

- ▶ MDPs have been very successful, but requires to have an observable Markovian state.
- ▶ Many domains this is impossible (or expensive) to obtain:
- ▶ Diagnosis (medical, maintenance)
- ▶ Robot navigation
- ▶ Tutoring
- ▶ Dialog systems
- ▶ Vision-based robotics
- ▶ Fault recovery

Beyond MDPs

- ▶ MDPs have been very successful, but requires to have an observable Markovian state.
- ▶ Many domains this is impossible (or expensive) to obtain:
- ▶ Diagnosis (medical, maintenance)
- ▶ Robot navigation
- ▶ Tutoring
- ▶ Dialog systems
- ▶ Vision-based robotics
- ▶ Fault recovery

Beyond MDPs

- ▶ MDPs have been very successful, but requires to have an observable Markovian state.
- ▶ Many domains this is impossible (or expensive) to obtain:
- ▶ Diagnosis (medical, maintenance)
- ▶ Robot navigation
- ▶ Tutoring
- ▶ Dialog systems
- ▶ Vision-based robotics
- ▶ Fault recovery

Beyond MDPs

- ▶ MDPs have been very successful, but requires to have an observable Markovian state.
- ▶ Many domains this is impossible (or expensive) to obtain:
 - ▶ Diagnosis (medical, maintenance)
 - ▶ Robot navigation
 - ▶ Tutoring
 - ▶ Dialog systems
 - ▶ Vision-based robotics
 - ▶ Fault recovery

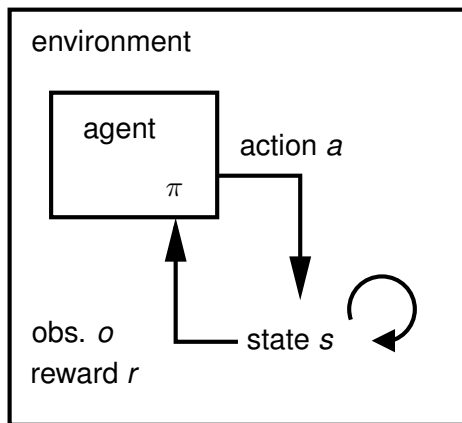
Beyond MDPs

- ▶ MDPs have been very successful, but requires to have an observable Markovian state.
- ▶ Many domains this is impossible (or expensive) to obtain:
- ▶ Diagnosis (medical, maintenance)
- ▶ Robot navigation
- ▶ Tutoring
- ▶ Dialog systems
- ▶ Vision-based robotics
- ▶ Fault recovery

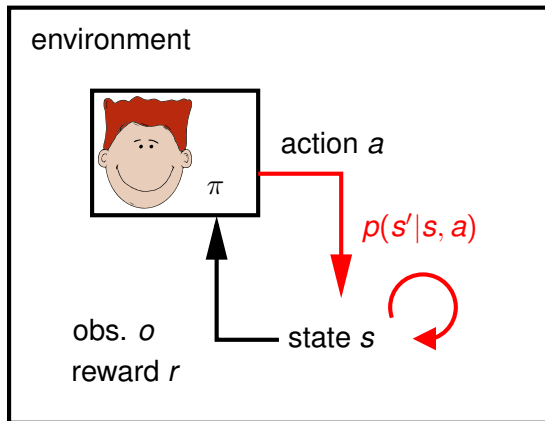
Observation model

- ▶ Imperfect sensors.
- ▶ Partially observable environment:
 - ▶ Sensors are **noisy**.
 - ▶ Sensors have a **limited view**.
- ▶ $p(o|s', a)$ is the probability the agent receives observation o in state s' after taking action a .

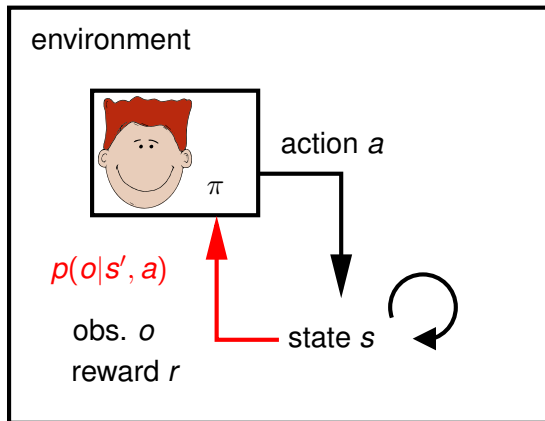
POMDP Agent



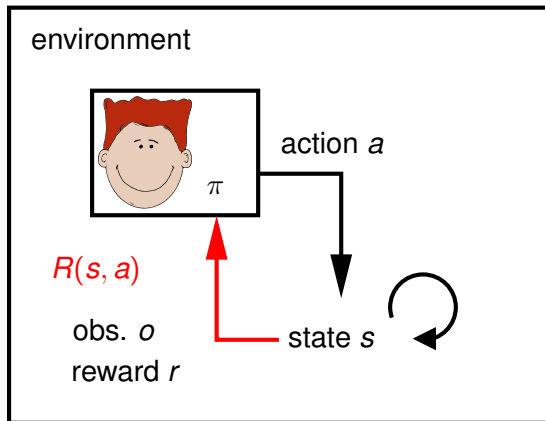
POMDP Agent



POMDP Agent



POMDP Agent



POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs)
(Kaelbling et al., 1998):

- ▶ Framework for agent planning under uncertainty.
- ▶ Typically assumes discrete sets of states S , actions A and observations O .
- ▶ Transition model $p(s'|s, a)$: models the effect of **actions**.
- ▶ Observation model $p(o|s', a)$: relates **observations** to states.
- ▶ Task is defined by a **reward** model $R(s, a)$.
- ▶ A planning horizon h (finite or ∞).
- ▶ A discount rate $0 \leq \gamma < 1$.
- ▶ Goal is to compute plan, or **policy** π , that maximizes long-term reward.

Beliefs

Beliefs:

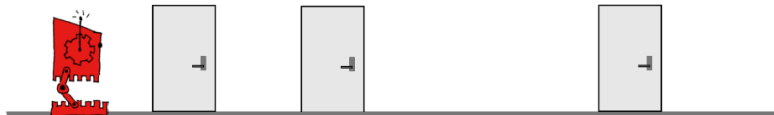
- ▶ The agent maintains a **belief** $b(s)$ of being at state s .
- ▶ After action $a \in A$ and observation $o \in O$ the belief $b(s)$ can be updated using Bayes' rule:

$$b'(s') \propto p(o|s') \sum_s p(s'|s, a) b(s)$$

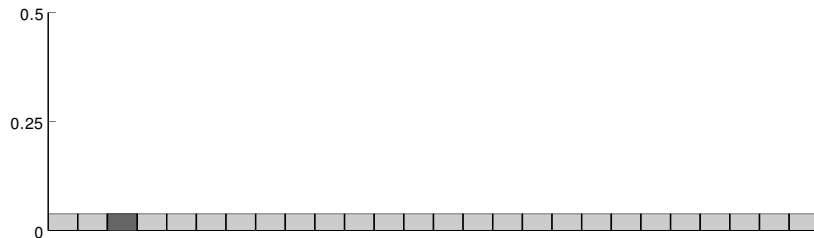
- ▶ The belief vector is a **Markov** signal for the planning task.

Belief update example

True situation:



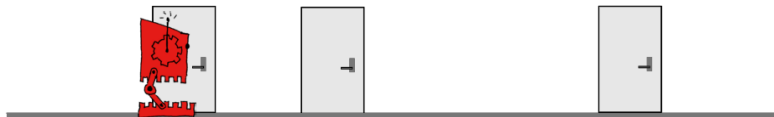
Robot's belief:



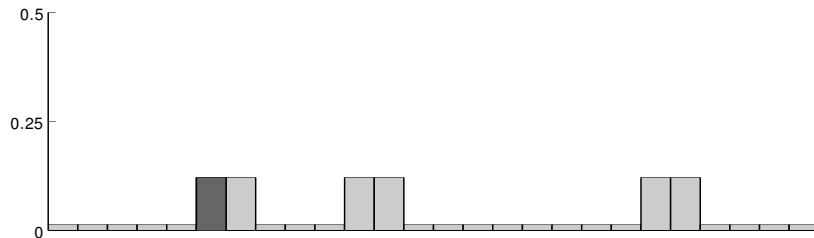
- ▶ Observations: *door* or *corridor*, 10% noise.
- ▶ Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



Robot's belief:



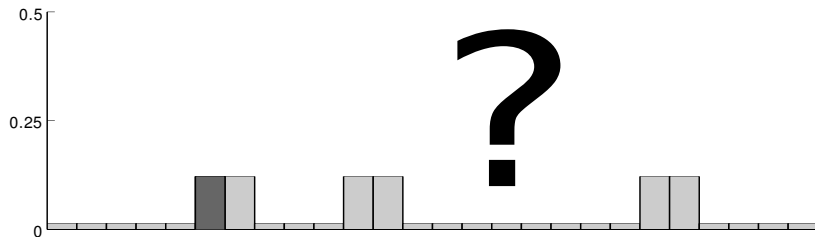
- ▶ Observations: **door** or *corridor*, 10% noise.
- ▶ Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



Robot's belief:



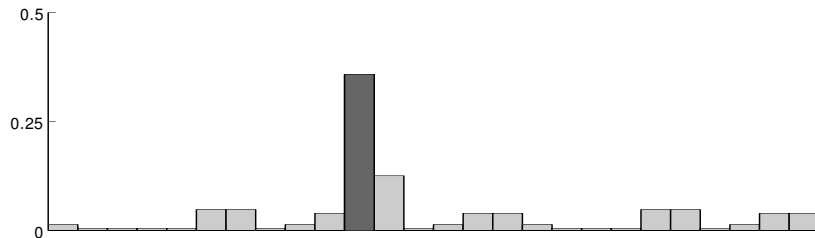
- ▶ Observations: **door** or *corridor*, 10% noise.
- ▶ Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



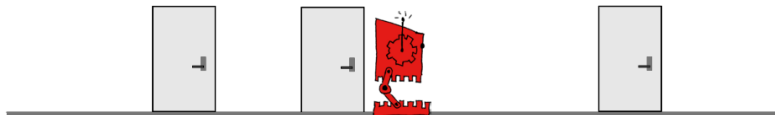
Robot's belief:



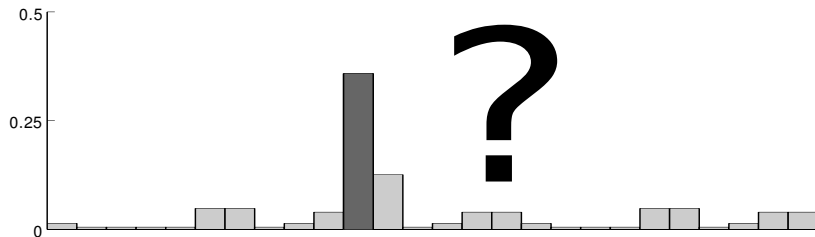
- ▶ Observations: **door** or *corridor*, 10% noise.
- ▶ Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



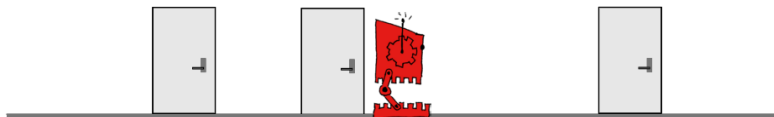
Robot's belief:



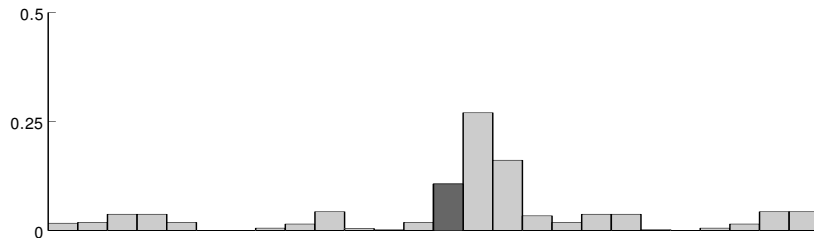
- ▶ Observations: *door* or **corridor**, 10% noise.
- ▶ Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



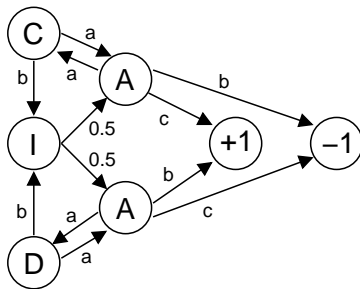
Robot's belief:



- ▶ Observations: *door* or **corridor**, 10% noise.
- ▶ Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

MDP-based algorithms

- ▶ Exploit belief state, and use the MDP solution as a heuristic.
- ▶ Most likely state (Cassandra et al., 1996):
 $\pi_{MLS}(b) = \pi^*(\arg \max_s b(s)).$
- ▶ Q_{MDP} (Littman et al., 1995):
 $\pi_{Q_{MDP}}(b) = \arg \max_a \sum_s b(s) Q^*(s, a).$



(Parr and Russell, 1995)

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- ▶ Continuous state space Δ : a simplex in $[0, 1]^{|S|-1}$.
- ▶ Stochastic Markovian transition model
 $p(b_a^o | b, a) = p(o | b, a)$. This is the normalizer of Bayes' rule.
- ▶ Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- ▶ The robot fully 'observes' the new belief-state b_a^o after executing a and observing o .

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- ▶ Continuous state space Δ : a simplex in $[0, 1]^{|S|-1}$.
- ▶ Stochastic Markovian transition model
 $p(b_a^o | b, a) = p(o | b, a)$. This is the normalizer of Bayes' rule.
- ▶ Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- ▶ The robot fully 'observes' the new belief-state b_a^o after executing a and observing o .

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- ▶ Continuous state space Δ : a simplex in $[0, 1]^{|S|-1}$.
- ▶ Stochastic Markovian transition model
 $p(b_a^o | b, a) = p(o | b, a)$. This is the normalizer of Bayes' rule.
- ▶ Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- ▶ The robot fully 'observes' the new belief-state b_a^o after executing a and observing o .

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- ▶ Continuous state space Δ : a simplex in $[0, 1]^{|S|-1}$.
- ▶ Stochastic Markovian transition model
 $p(b_a^o | b, a) = p(o | b, a)$. This is the normalizer of Bayes' rule.
- ▶ Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- ▶ The robot fully 'observes' the new belief-state b_a^o after executing a and observing o .

Solving POMDPs

- ▶ A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- ▶ The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b, a) + \gamma \sum_o p(o|b, a) V^*(b_a^o) \right]$$

- ▶ Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0 .
- ▶ Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Solving POMDPs

- ▶ A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- ▶ The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b, a) + \gamma \sum_o p(o|b, a) V^*(b_a^o) \right]$$

- ▶ Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0 .
- ▶ Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Solving POMDPs

- ▶ A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- ▶ The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b, a) + \gamma \sum_o p(o|b, a) V^*(b_a^o) \right]$$

- ▶ Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0 .
- ▶ Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

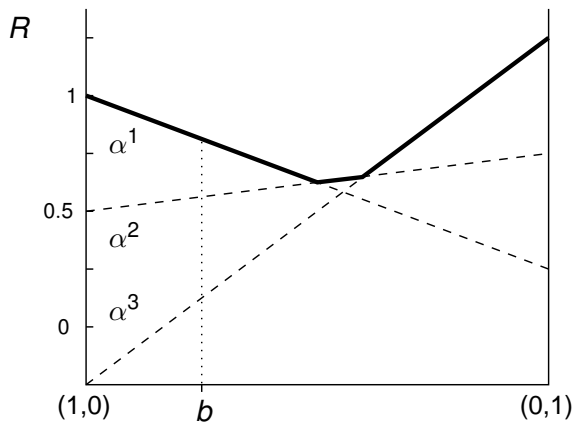
Solving POMDPs

- ▶ A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- ▶ The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b, a) + \gamma \sum_o p(o|b, a) V^*(b_a^o) \right]$$

- ▶ Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0 .
- ▶ Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Example V_0



$R(s, a)$	a_1	a_2	a_3
s_1	1.00	0.50	-0.25
s_2	0.25	0.75	1.25

PWLC shape of V_n

- ▶ Like V_0 , V_n is as well piecewise linear and convex.
- ▶ Rewards $R(b, a) = b \cdot R(s, a)$ are linear functions of b .
Note that the value of a point b satisfies:

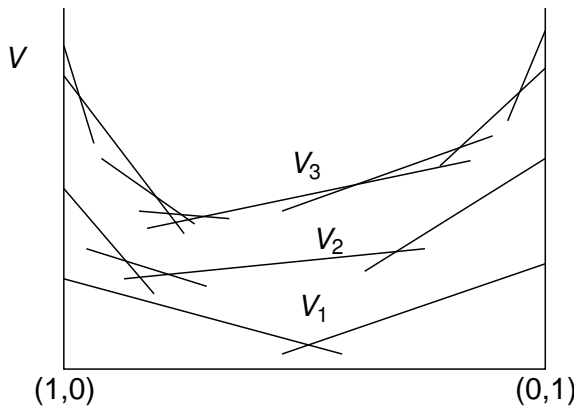
$$V_{n+1}(b) = \max_a \left[b \cdot R(s, a) + \gamma \sum_o p(o|b, a) V_n(b_a^o) \right]$$

which involves a maximization over (at least) the vectors $R(s, a)$.

- ▶ Intuitively: less uncertainty about the state (low-entropy beliefs) means better decisions (thus higher value).

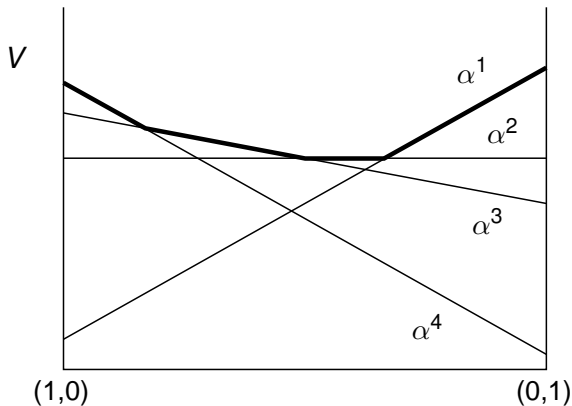
Exact value iteration

Value iteration computes a sequence of value function estimates V_1, V_2, \dots, V_n , using the POMDP backup operator H , $V_{n+1} = HV_n$.

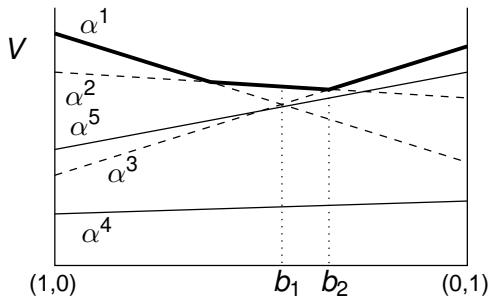


Optimal value functions

The optimal value function of a (finite-horizon) POMDP is piecewise linear and convex: $V(b) = \max_{\alpha} b \cdot \alpha$.



Vector pruning



Linear program for pruning:

variables: $\forall s \in S, b(s); x$

maximize: x

subject to:

$$b \cdot (\alpha - \alpha') \geq x, \forall \alpha' \in V, \alpha' \neq \alpha$$

$$b \in \Delta(S)$$

Optimal POMDP methods

Enumerate and prune:

- ▶ Most straightforward: Monahan (1982)'s enumeration algorithm. Generates a maximum of $|A| |V_n|^{|\mathcal{O}|}$ vectors at each iteration, hence requires pruning.
- ▶ Incremental pruning (Zhang and Liu, 1996; Cassandra et al., 1997).

Search for witness points:

- ▶ One Pass (Sondik, 1971; Smallwood and Sondik, 1973).
- ▶ Relaxed Region, Linear Support (Cheng, 1988).
- ▶ Witness (Cassandra et al., 1994).

Sub-optimal techniques

- ▶ **Grid-based approximations**

(Drake, 1962; Lovejoy, 1991; Brafman, 1997; Zhou and Hansen, 2001; Bonet, 2002).

- ▶ **Optimizing finite-state controllers**

(Platzman, 1981; Hansen, 1998b; Poupart and Boutilier, 2004).

- ▶ **Heuristic search in the belief tree**

(Satia and Lave, 1973; Hansen, 1998a).

- ▶ **Compression or clustering**

(Roy et al., 2005; Poupart and Boutilier, 2003; Virin et al., 2007).

- ▶ **Point-based techniques**

(Pineau et al., 2003; Smith and Simmons, 2004; Spaan and Vlassis, 2005; Shani et al., 2007; Kurniawati et al., 2008).

- ▶ **Monte Carlo tree search**

(Silver and Veness, 2010).

Point-based backup

- ▶ For finite horizon V^* is piecewise linear and convex, and for infinite horizons V^* can be approximated arbitrary well by a PWLC value function (Smallwood and Sondik, 1973).
- ▶ Given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \arg \max_{\{\alpha_{n+1}^k\}_k} b \cdot \alpha_{n+1}^k,$$

where $\{\alpha_{n+1}^k\}_{k=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n . We will denote this operation $\alpha_{n+1}^b = \text{backup}(b)$.

Point-based backup

- ▶ For finite horizon V^* is piecewise linear and convex, and for infinite horizons V^* can be approximated arbitrary well by a PWLC value function (Smallwood and Sondik, 1973).
- ▶ Given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \arg \max_{\{\alpha_{n+1}^k\}_k} b \cdot \alpha_{n+1}^k,$$

where $\{\alpha_{n+1}^k\}_{k=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n . We will denote this operation $\alpha_{n+1}^b = \text{backup}(b)$.

Point-based (approximate) methods

Point-based (approximate) value iteration plans only on a limited set of **reachable** belief points:

1. Let the robot explore the environment.
2. Collect a set B of belief points.
3. Run approximate value iteration on B .

Further reading

- ▶ Textbook on reinforcement learning
 - ▶ R. S. Sutton and A. G. Barto. “Reinforcement Learning: An Introduction”. MIT Press, 1998.
- ▶ Recent book containing chapters on many aspects of decision-theoretic planning (MDPs, POMDPs, Dec-POMDPs):
 - ▶ Marco Wiering and Martijn van Otterlo, editors, “Reinforcement Learning: State of the Art”, Springer, 2012.

References I

- R. Bellman. *Dynamic programming*. Princeton University Press, 1957.
- D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2nd edition, 2000.
- B. Bonet. An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In *International Conference on Machine Learning*, 2002.
- R. I. Brafman. A heuristic variable grid solution method for POMDPs. 1997.
- A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. 1994.
- A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proc. of International Conference on Intelligent Robots and Systems*, 1996.
- A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of Uncertainty in Artificial Intelligence*, 1997.
- H. T. Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.
- A. W. Drake. *Observation of a Markov process through a noisy channel*. Sc.D. thesis, Massachusetts Institute of Technology, 1962.
- E. A. Hansen. *Finite-memory control of partially observable systems*. PhD thesis, University of Massachusetts, Amherst, 1998a.
- E. A. Hansen. Solving POMDPs by searching in policy space. In *Proc. of Uncertainty in Artificial Intelligence*, 1998b.
- L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
- H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.
- M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 1995.
- W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1991.

References II

- G. E. Monahan. A survey of partially observable Markov decision processes: theory, models and algorithms. *Management Science*, 28(1), Jan. 1982.
- R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 1995.
- J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003.
- L. K. Platzman. A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Technical Report J-81-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1981. Reprinted in working notes AAAI 1998 Fall Symposium on Planning with POMDPs.
- P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.
- P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.
- M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.
- N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.
- J. K. Satia and R. E. Lave. Markovian decision processes with probabilistic observation of states. *Management Science*, 20(1):1–13, 1973.
- G. Shani, R. I. Brafman, and S. E. Shimony. Forward search value iteration for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2007.
- D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, 2010.
- R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.
- T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2004.
- E. J. Sondik. *The optimal control of partially observable Markov processes*. PhD thesis, Stanford University, 1971.

References III

- M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.
- R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- Y. Virin, G. Shani, S. E. Shimony, and R. Brafman. Scaling up: Solving POMDPs through value based clustering. 2007.
- N. L. Zhang and W. Liu. Planning in stochastic domains: problem characteristics and approximations. Technical Report HKUST-CS96-31, Department of Computer Science, The Hong Kong University of Science and Technology, 1996.
- R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2001.

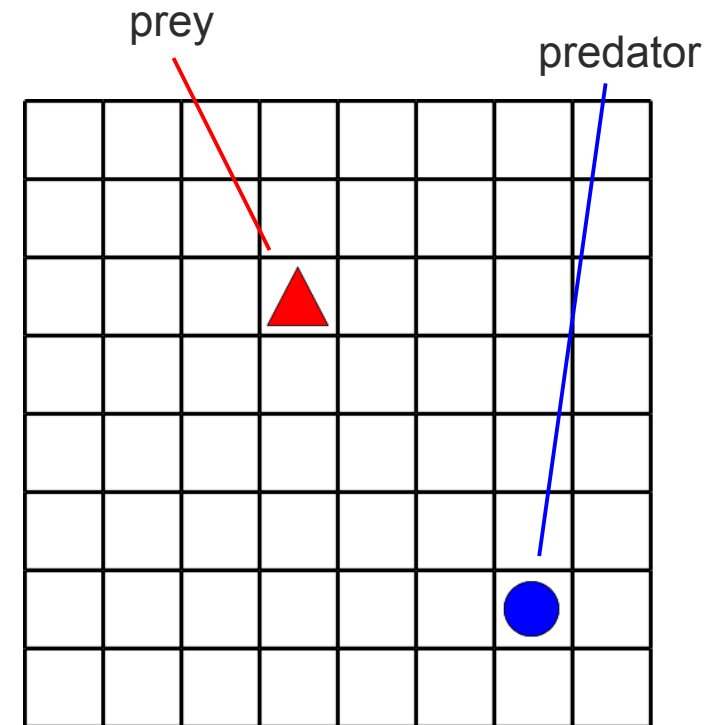
Multiagent Models

Example: Predator-Prey Domain

- Predator-Prey domain
 - 1 agent: predator
 - prey: part of environment
 - on a torus

- Formalization:

- states $(-3,4)$
- actions N,W,S,E
- transitions failing to move, prey moves
- rewards reward for capturing



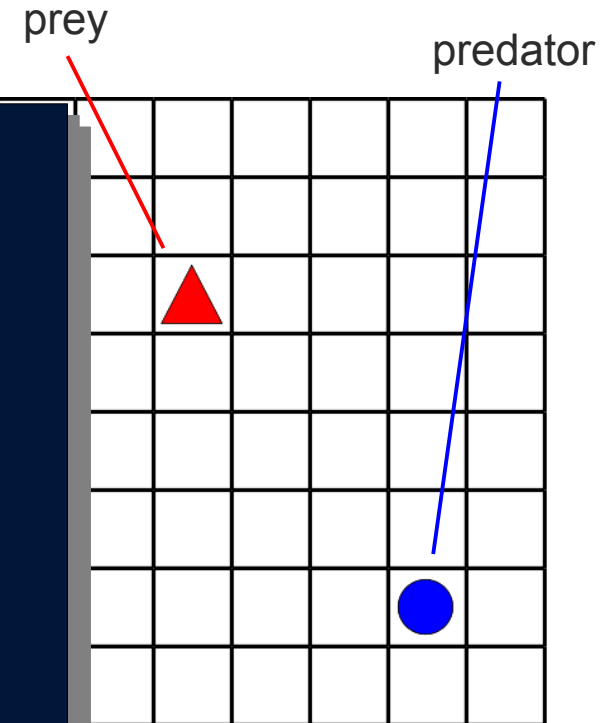
Example: Predator-Prey Domain

- Predator-Prey domain

Markov decision process (MDP)

- Markovian state s . (which is observed!)
- policy π maps states \rightarrow actions
- Value function $Q(s,a)$
- Compute via value iteration / policy iteration

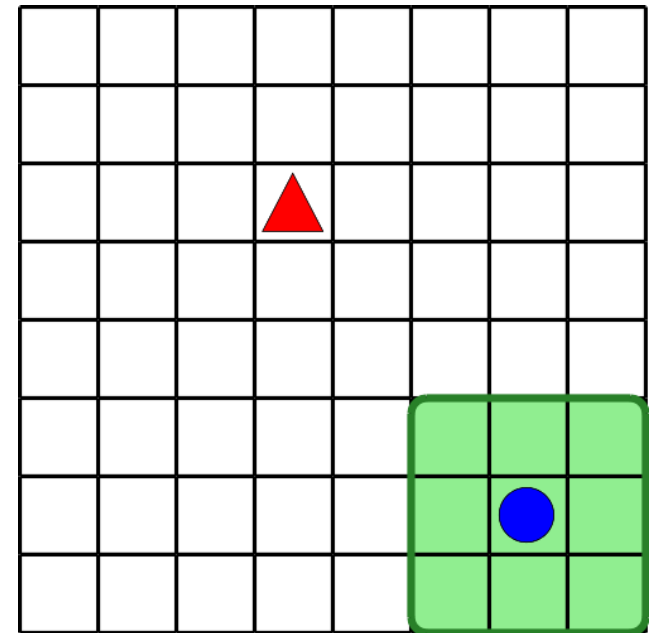
$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$



moves

Partial Observability

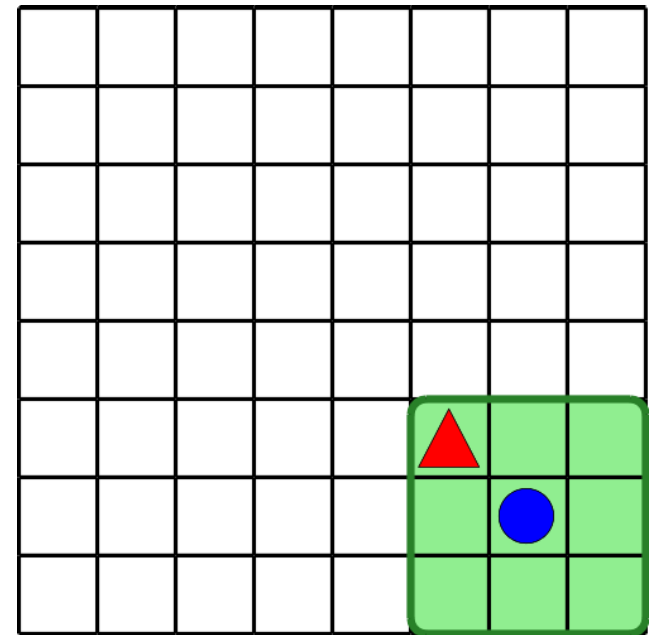
- Now: partial observability
 - E.g., limited range of sight
- MDP + observations
 - explicit observations
 - observation probabilities
 - noisy observations
(detection probability)



$o = \text{'nothing'}$

Partial Observability

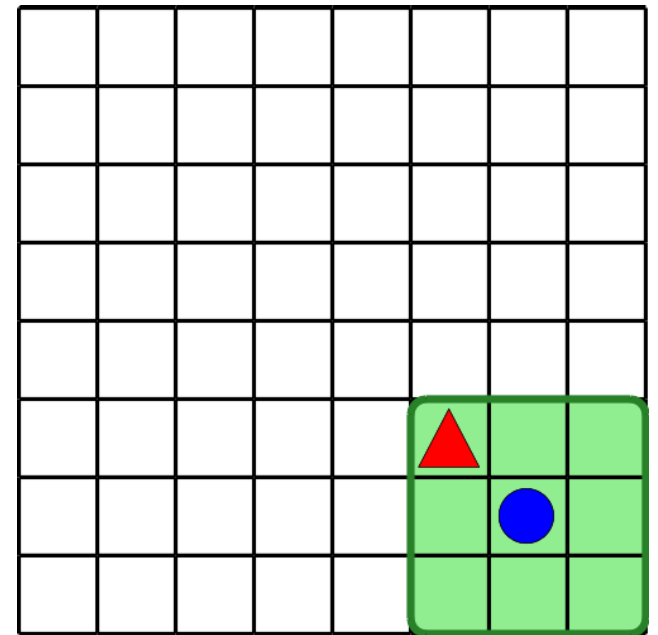
- Now: partial observability
 - E.g., limited range of sight
- MDP + observations
 - explicit observations
 - observation probabilities
 - noisy observations
(detection probability)



$$o = (-1, 1)$$

Partial Observability

- Now: partial observability
 - E.g., limited range of sight
- MDP + observations
 - explicit observations
 - observation probabilities
 - noisy observations
(detection probability)



$$o = (-1, 1)$$

Can not observe the state

→ Need to maintain a belief over states $b(s)$

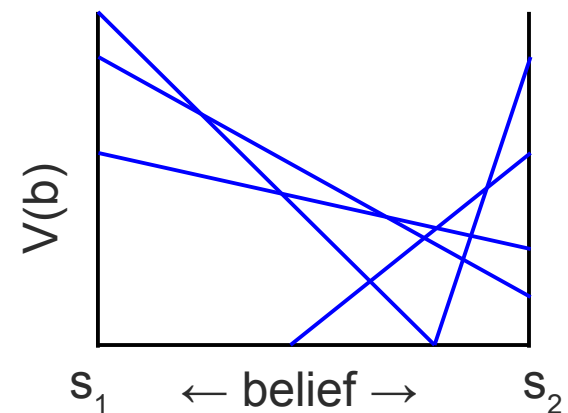
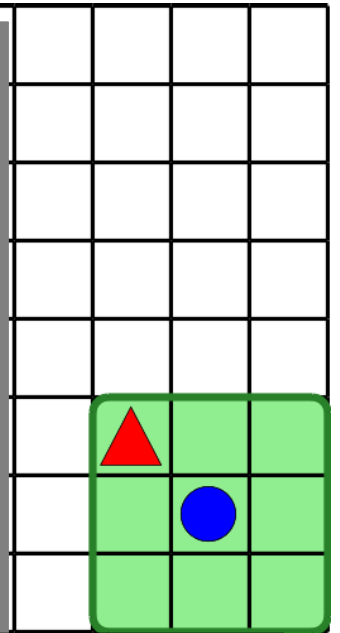
→ Policy maps beliefs to actions $\pi(b) = a$

Partial Observability

- Now: partial observability

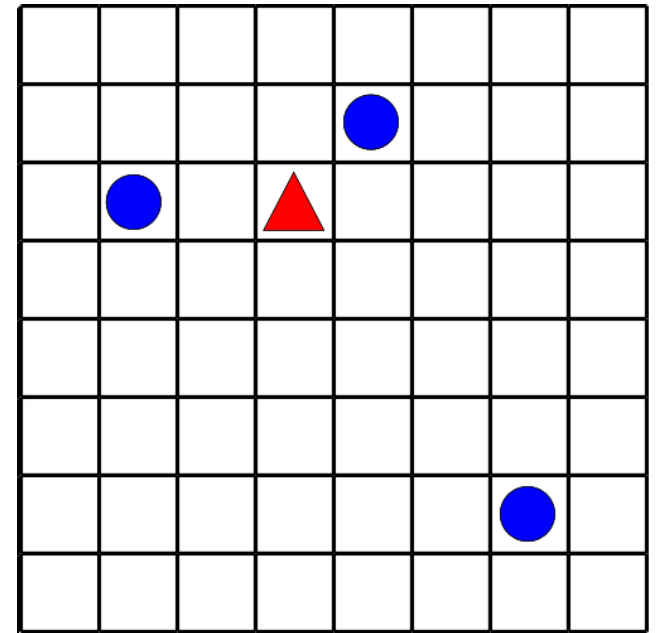
Partially Observable MDP (POMDP)

- reduction \rightarrow continuous state MDP
(in which the belief is the state)
- Value iteration:
 - make use of α -vectors (\leftrightarrow complete policies)
 - perform pruning



Multiple Agents

- Now: multiple agents
 - fully observable



- Formalization:
 - states $((3,-4), (1,1), (-2,0))$
 - actions $\{N,W,S,E\}$
 - **joint** actions $\{(N,N,N), (N,N,W), \dots, (E,E,E)\}$
 - transitions probability of failing to move, prey moves
 - rewards reward for capturing jointly

Multiple Agents

- Now: multiple agents

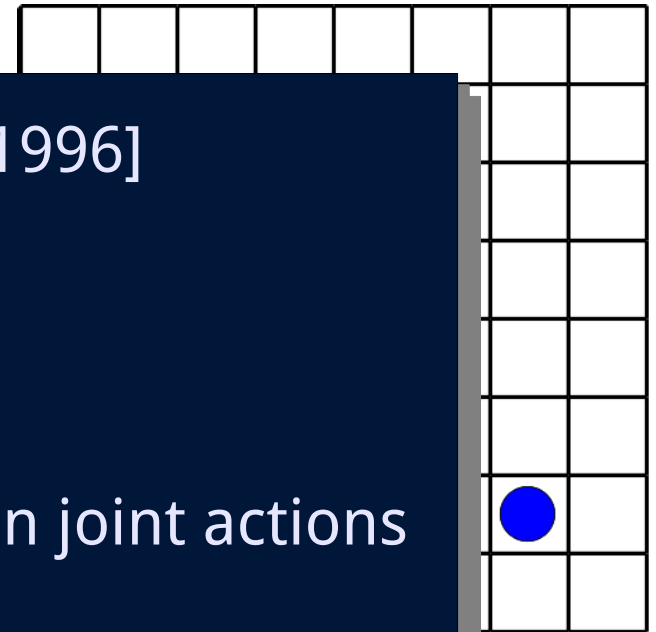
fully observable

Multiagent MDP [Boutilier 1996]

- Differences with MDP
 - n agents
 - joint actions $a = \langle a_1, a_2, \dots, a_n \rangle$
 - transitions and rewards depend on joint actions

- For Solution:
 - Treat as normal MDP with 1 'puppeteer agent'
 - Optimal policy $\pi(s) = a$
 - Every agent executes its part

- transitions probability of failing to move, prey moves
- rewards reward for capturing jointly



Multiple Agents

- Now: multiple agents

fully observable

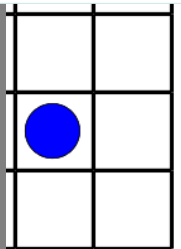
Multiagent MDP [Boutilier 1996]

- Differences with MDP
 - n agents
 - joint actions $a = \langle a_1, a_2, \dots, a_n \rangle$
 - transitions and rewards depend on joint actions

Catch: number of joint actions is **exponential!**
(but other than that, conceptually simple.)

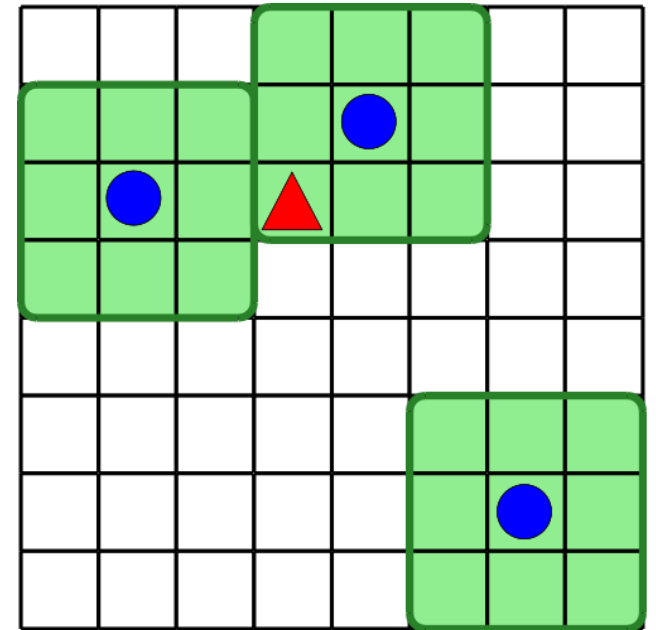
- For Solution:
 - Treat as normal MDP with 1 'puppeteer agent'
 - Optimal policy $\pi(s) = a$
 - Every agent executes its part

- transitions probability of failing to move, prey moves
- rewards reward for capturing jointly



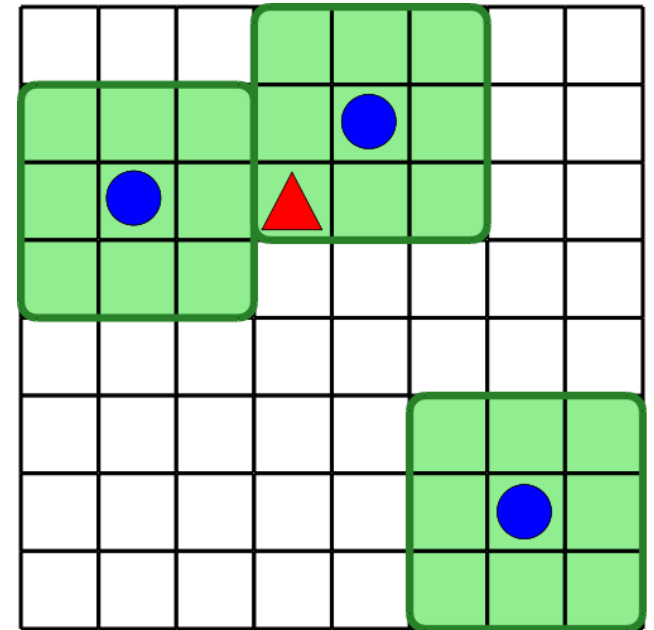
Multiple Agents & Partial Observability

- Now both...
 - partial observability
 - multiple agents



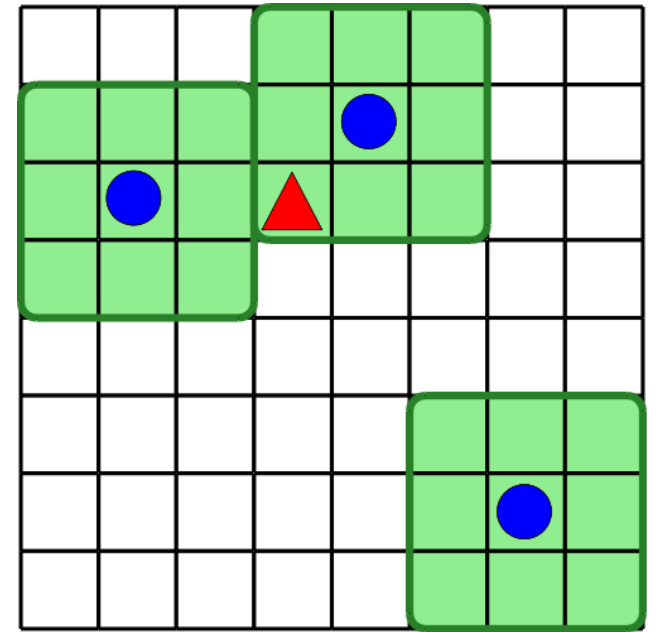
Multiple Agents & Partial Observability

- Now both...
 - partial observability
 - multiple agents
- Decentralized POMDPs (Dec-POMDPs) [Bernstein et al. 2002]
- both
 - joint actions and
 - joint observations



Multiple Agents & Partial Observability

- Again we can make a reduction...



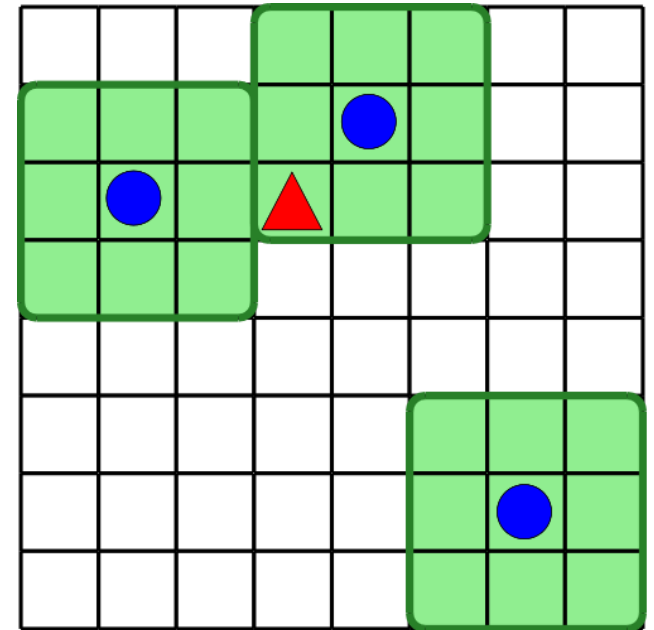
Multiple Agents & Partial Observability

- Again we can make a reduction...

Dec-POMDPs \rightarrow MPOMDP

(multiagent POMDP)

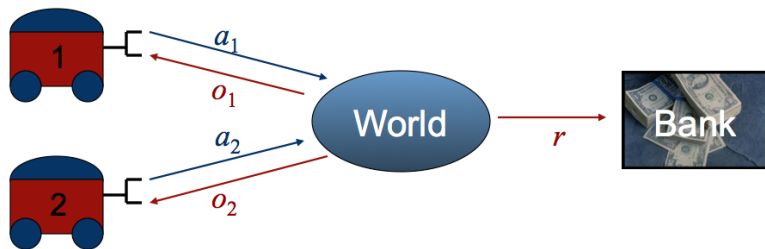
- 'puppeteer agent'
 - receives joint observations
 - takes joint actions
- requires broadcasting observations!
 - instantaneous, cost-free, noise-free communication \rightarrow optimal [Pynadath and Tambe 2002]
 - Without such communication: no easy reduction.



Decentralized POMDPs

Decentralized POMDPs

Now we consider a group of agents that control the environment jointly.



Each agent receives a separate partial observation.
The agents try to optimize a single reward function.

DEC-POMDP

Definition

A decentralized partially observable MDP (DEC-POMDP) is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, h \rangle$ where

- ▶ I is a finite set of agents indexed $1, \dots, n$.
- ▶ S is a finite set of states, with distinguished initial state s_0 .
- ▶ A_i is a finite set of actions available to agent i , and $\vec{A} = \otimes_{i \in I} A_i$ is the set of joint actions.
- ▶ $P : S \times \vec{A} \rightarrow \Delta S$ is a Markovian transition function.
 $P(s'|s, \vec{a})$ denotes the probability that after taking joint action \vec{a} in state s a transition to state s' occurs.
- ▶ Ω_i is a finite set of observations available to agent i , and $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ is the set of joint observations.
- ▶ $O : \vec{A} \times S \rightarrow \Delta \vec{\Omega}$ is an observation function.
 $O(\vec{o}|\vec{a}, s')$ denotes the probability of observing joint observation \vec{o} given that joint action \vec{a} was taken and led to state s' .
- ▶ $R : \vec{A} \times S \rightarrow \mathbb{R}$ is a reward function.
 $R(\vec{a}, s')$ denotes the reward obtained after joint action \vec{a} was taken and a state transition to s' occurred.
- ▶ If the DEC-POMDP has a finite horizon, that horizon is represented by a positive integer h .

Partially observable stochastic games

Definition

A partially observable stochastic game (POSG) is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, \{R_i\}, h \rangle$ where

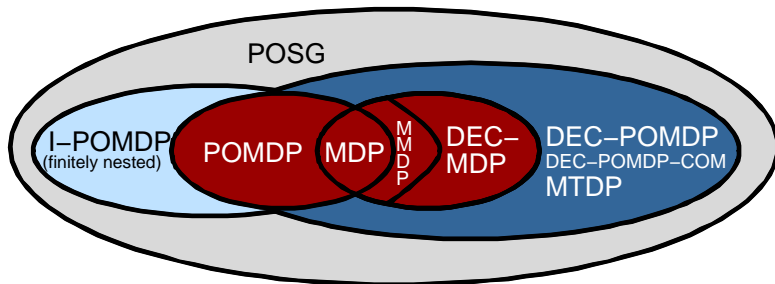
- ▶ All the components except the reward function are the same as in a DEC-POMDP
- ▶ Each agent has an individual reward function:
 $R_i : A_i \times S \rightarrow \mathbb{R}$.
 $R_i(a_i, s')$ denotes the reward obtained after action a_i was taken by agent i and a state transition to s' occurred.

This is the self-interested version of the DEC-POMDP model

Interactive POMDPs

- ▶ Interactive POMDPs (I-POMDPs) extend state space with behavioral models of other agents (Gmytrasiewicz and Doshi, 2005).
- ▶ Agents maintain beliefs over physical and models of others.
 - ▶ Recursive modeling.
- ▶ When assuming a finite nesting, beliefs and value functions can be computed (approximately).
- ▶ Finitely nested I-POMDPs can be solved as a set of POMDPs.

Relationships among the models



Previous complexity results

Finite Horizon

MDP	P-complete (if $h < S $)	(Papadimitriou and Tsitsiklis, 1987)
POMDP	PSPACE-complete (if $h < S $)	(Papadimitriou and Tsitsiklis, 1987)

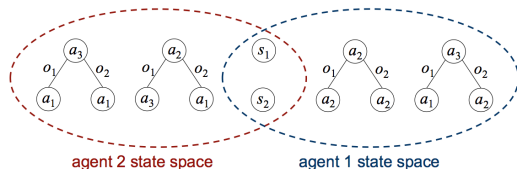
Infinite Horizon Discounted

MDP	P-complete	(Papadimitriou and Tsitsiklis, 1987)
POMDP	undecidable	(Madani et al., 1999)

DEC-POMDPs complexity

Intuition

- ▶ Agents must consider the choices of all others in addition to the state and action uncertainty present in POMDPs.
- ▶ This makes DEC-POMDPs much harder to solve (NEXP-complete).
- ▶ Solvable in nondeterministic exponential time: Can guess a solution in exponential time and transform the DEC-POMDP into an exponentially bigger belief state MDP.
- ▶ NEXP-hardness: Reduction from tiling problem (each agent must place a tile based on local information and the result must be consistent).



Upper bound for DEC-POMDPs

Theorem

Finite-horizon DEC-POMDPs are in nondeterministic exponential time.

Proof: The following process shows that a non-deterministic Turing machine can solve any instance of a DEC-POMDP_{*n*} in at most exponential time.

1. Guess a joint policy and write it down in exponential time. This is possible, because a joint policy consists of n mappings from observation histories to actions. Since $h \leq |S|$, the number of possible histories is exponentially bounded by the problem description.
2. The DEC-POMDP together with the guessed joint policy can be viewed as an exponentially bigger POMDP using n -tuples of observations and actions.
3. In exponential time, convert all the observation sequences into a belief state.
4. In exponential time, compute transition probabilities and expected rewards for an exponentially bigger belief state MDP.
5. This MDP can be solved in polynomial time, which is exponential in the original problem description.

Thus, there is an accepting computation path in the non-deterministic machine if and only if there is a joint policy that can achieve reward K .

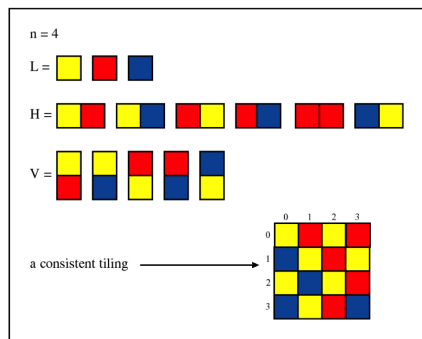
Lower bound for DEC-POMDPs

Theorem

(Bernstein et al., 2002) Two-agent finite-horizon DEC-POMDPs are NEXP-hard.

- ▶ Thus provably intractable (unlike POMDP)
- ▶ Probably doubly exponential (unlike POMDP)

Proof: By reduction from TILING



Joint observability

Definition

Joint full observability \equiv collective observability. A DEC-POMDP is jointly fully observable if the n -tuple of observations made by all the agents uniquely determine the current global state.

That is, if $O(\vec{o}|\vec{a}, s') > 0$ then $P(s'|\vec{o}) = 1$.

Definition

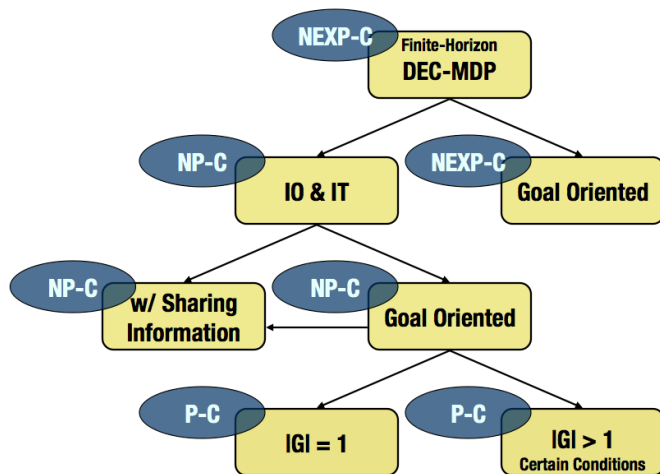
A decentralized Markov decision process (**DEC-MDP**) is a DEC-POMDP with joint full observability.

A stronger result: The problem is NEXP-hard even when the state is jointly observed! That is, two-agent finite-horizon DEC-MDPs are NEXP-hard.

Observability, communication and complexity

Observability	General Communication	Free Communication
Full	MMDP (P-complete)	MMDP (P-complete)
Joint Full	DEC-MDP (NEXP-complete)	MMDP (P-complete)
Partial	DEC-POMDP (NEXP-complete)	MPOMDP (PSPACE-complete)

More complexity results



(Goldman and Zilberstein, 2004)

References I

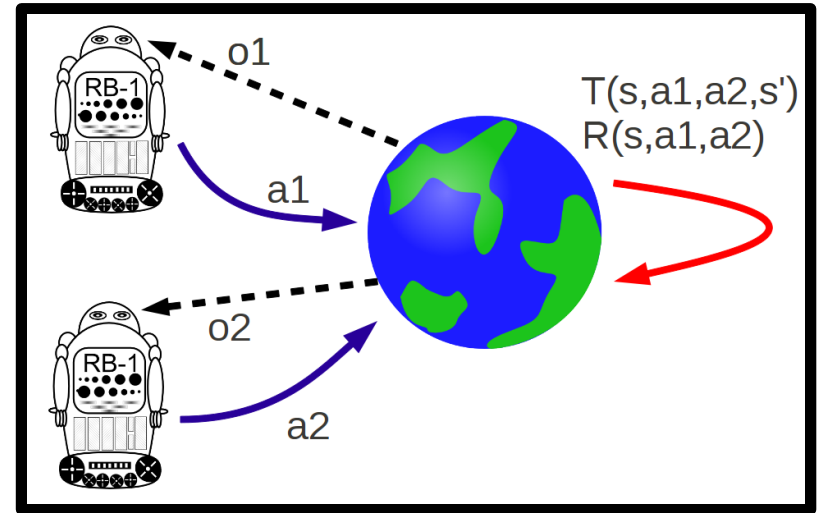
- D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
- P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.
- C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.
- O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. Orlando, Florida, July 1999.
- C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

Preliminaries & The Goal of Planning

Notation

- A Dec-POMDP

- $\langle S, A, P_T, O, P_O, R, h \rangle$
- n agents
- S – set of states
- A – set of **joint** actions
- P_T – transition function
- O – set of **joint** observations
- P_O – observation function
- R – reward function
- h – horizon (finite)



$$a = \langle a_1, a_2, \dots, a_n \rangle$$

$$P(s' | s, a)$$

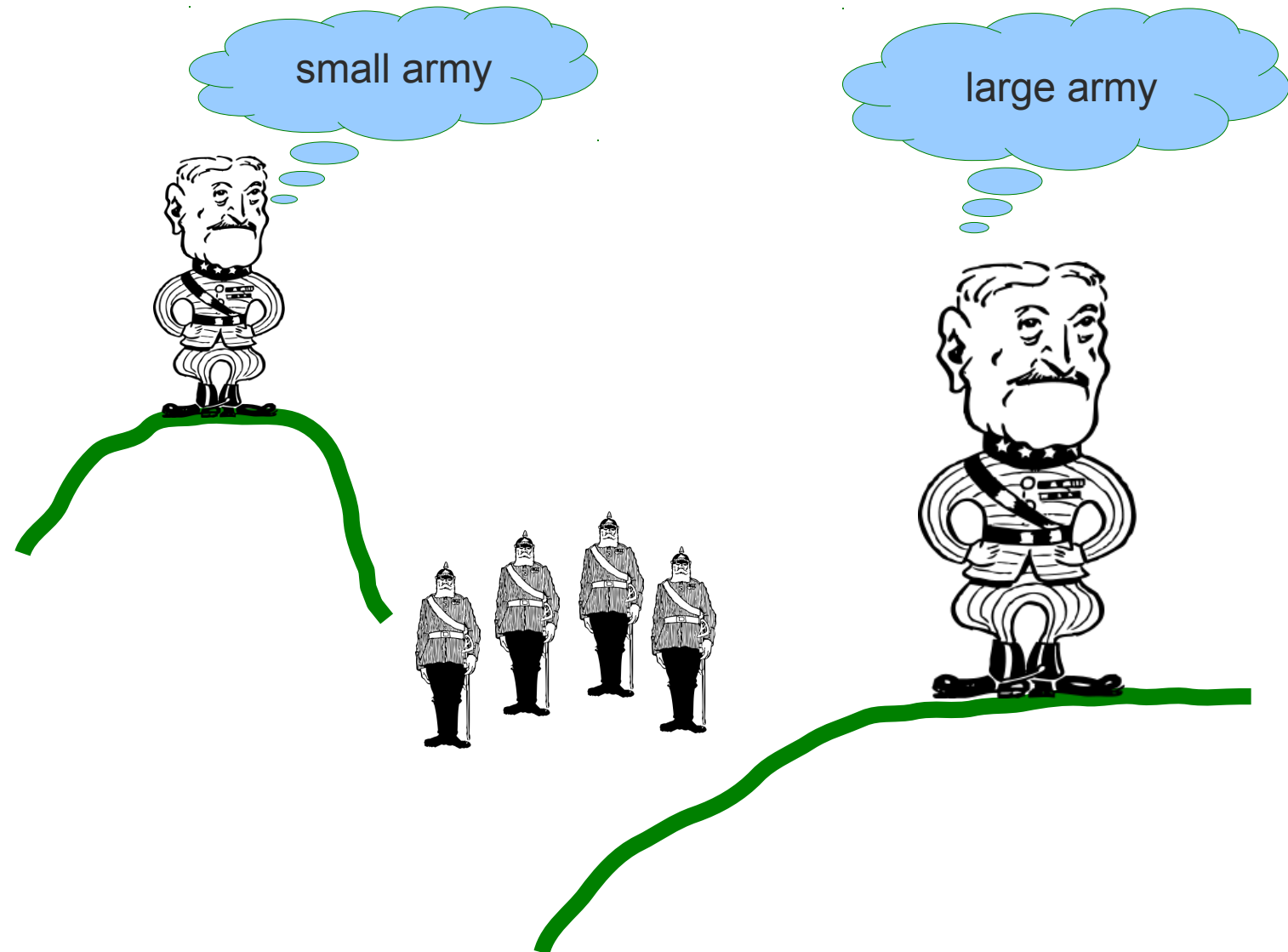
$$o = \langle o_1, o_2, \dots, o_n \rangle$$

$$P(o | a, s')$$

$$R(s, a)$$

Running Example

- 2 generals problem



Running Example

- $S = \{s_L, s_S\}$
 $A_i = \{(O)bserve, (A)ttack\}$
 $O_i = \{(L)arge, (S)mall\}$

Transitions

- Both Observe \rightarrow no state change
- At least 1 Attack \rightarrow reset (50% probability s_L, s_S)

Observations

- Probability of correct observation: 0.85
- E.g., $P(\langle L, L \rangle \mid s_L) = 0.85 * 0.85 = 0.7225$

Rewards

- 1 general attacks \rightarrow he loses the battle: $R(*, \langle A, O \rangle) = -10$
- Both generals Observe \rightarrow small cost: $R(*, \langle O, O \rangle) = -1$
- Both Attack \rightarrow depends on state:
 $R(s_L, \langle A, A \rangle) = -20$
 $R(s_S, \langle A, A \rangle) = +5$

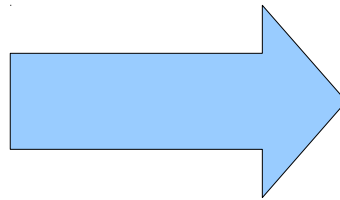
Off-line / On-line phases

- off-line planning, on-line execution is decentralized

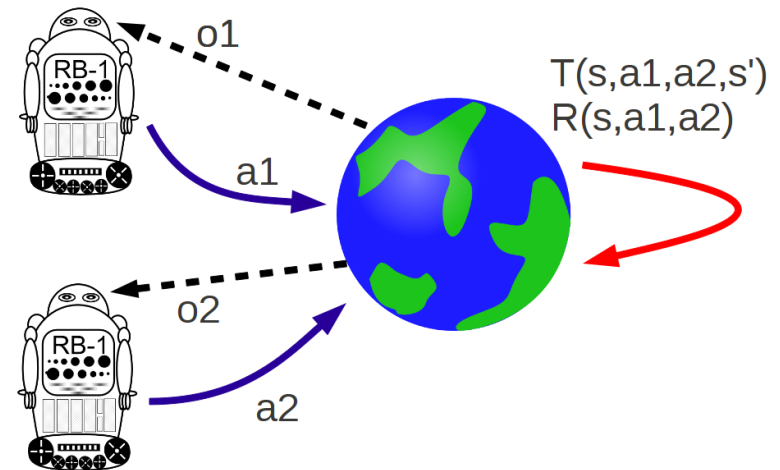
Planning Phase



$$\pi = \langle \pi_1, \pi_2 \rangle$$



Execution Phase

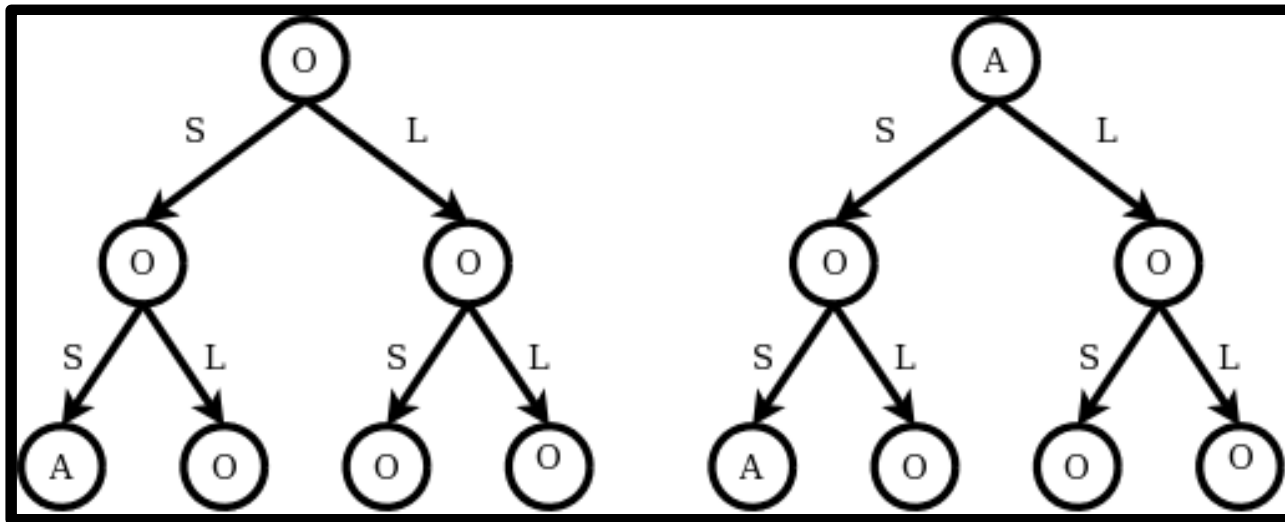


- (Smart generals make a plan in advance!)

Policy Domain

- Policies map observation histories (OHs) \rightarrow actions

$$\pi_i: \vec{O}_i \rightarrow A_i$$



Most general, AOHs:

$$(a_i^0, o_i^1, a_i^1, \dots, a_i^{t-1}, o_i^t)$$

But: can restrict to
deterministic policies
 \rightarrow only need OHs:

$$\vec{o}_i = (o_i^1, \dots, o_i^t)$$

No Compact Representation?

- **Joint Belief, $b(s)$** (as in MPOMDP) [Pynadath and Tambe 2002]
 - compute $b(s)$ using joint actions and observations
 - Problem: agents do not know those during execution
- **Multiagent belief, $b_i(s, q_{-i})$** [Hansen et al. 2004]
 - belief over (future) policies of other agents
(Need to be able to predict the other agents!)
 - form of those other policies?
 - most general: $\pi_j: \vec{o}_j \rightarrow a_j$
 - if they use beliefs? \rightarrow infinite recursion of beliefs!

Goal of Planning

- Find an **optimal** joint policy $\pi^* = \langle \pi_1, \pi_2 \rangle$
- What is optimal?
 - Maximal value
 - Typically, **value** is the expected sum of rewards:

$$V(\pi) = \mathbf{E} \left[\sum_{t=0}^{h-1} R(s, a) \mid \pi, b^0 \right]$$

Goal of Planning

Optimal policy for 2 generals, h=3

value=-2.86743

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

Coordination vs. Exploitation of Local Information

- Inherent trade-off

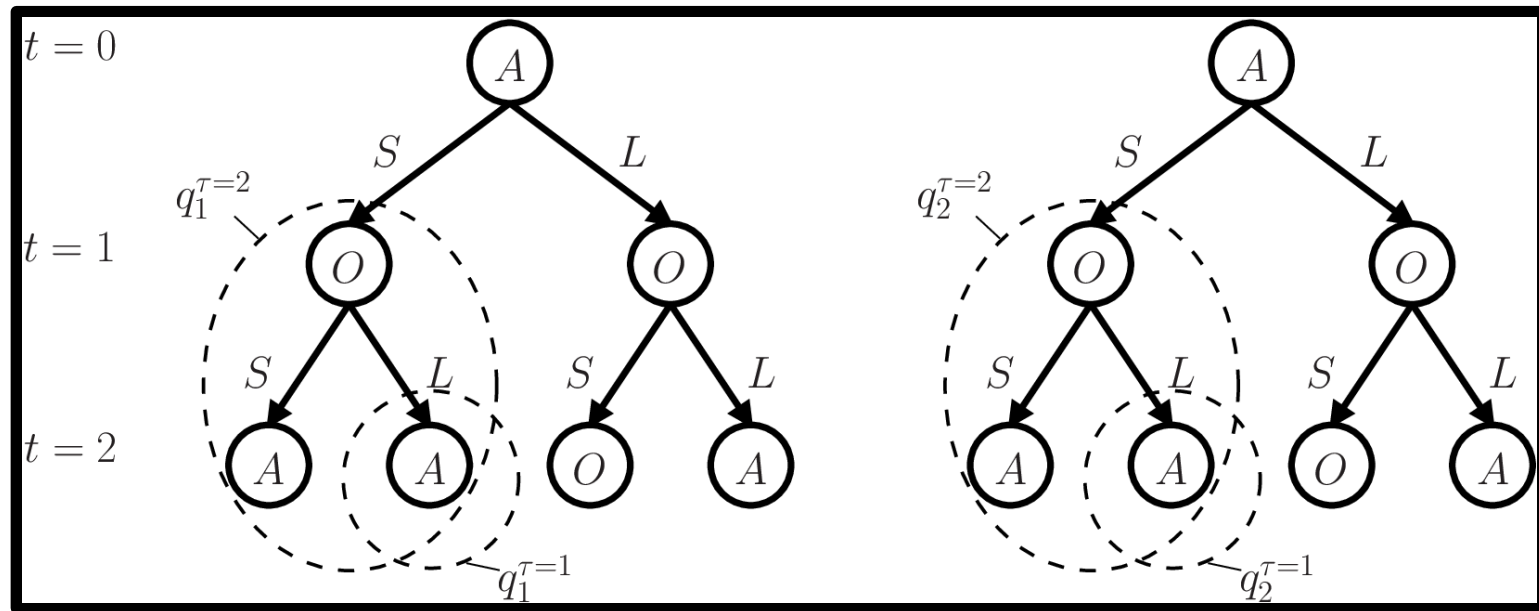
coordination vs. exploitation of local information

- Ignore own observations → 'open loop plan'
 - E.g., "ATTACK on 2nd time step"
 - + maximally predictable
 - low quality
 - Ignore coordination → 'MPOMDP plan'
 - E.g., 'individual belief' $b_i(s)$ and execute the MPOMDP policy
 - + uses local information
 - likely to result in mis-coordination
- **Optimal policy π^* should balance between these!**

Value Functions

Value of a Joint Policy

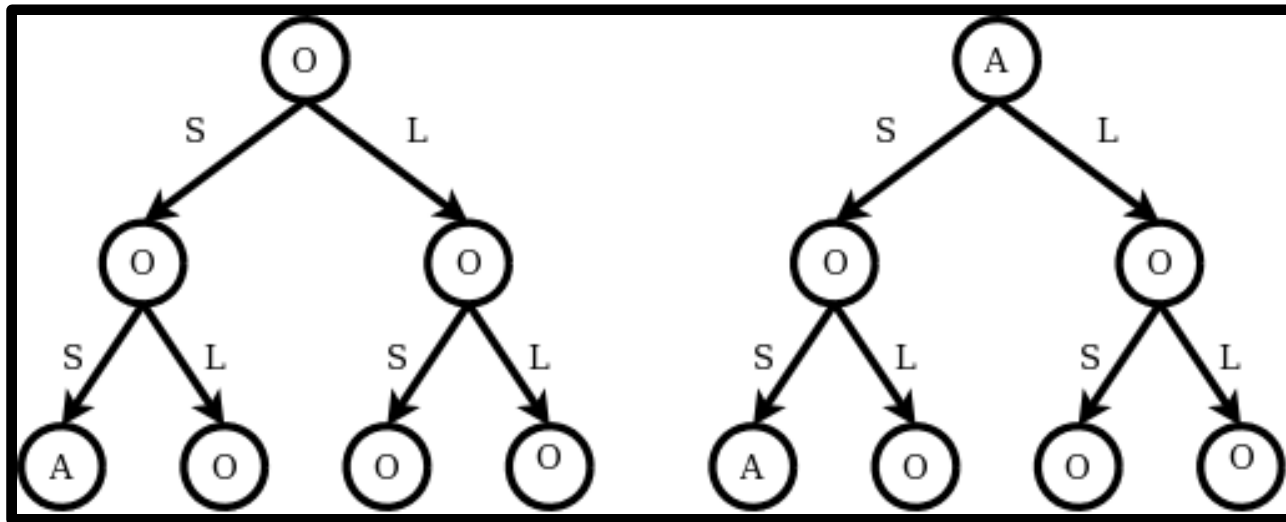
- Sub-tree policies:



- Given a particular joint policy $\pi = q^{\tau=h}$
 \rightarrow Just a (complex) Markov Chain [Nair et al. 2003]

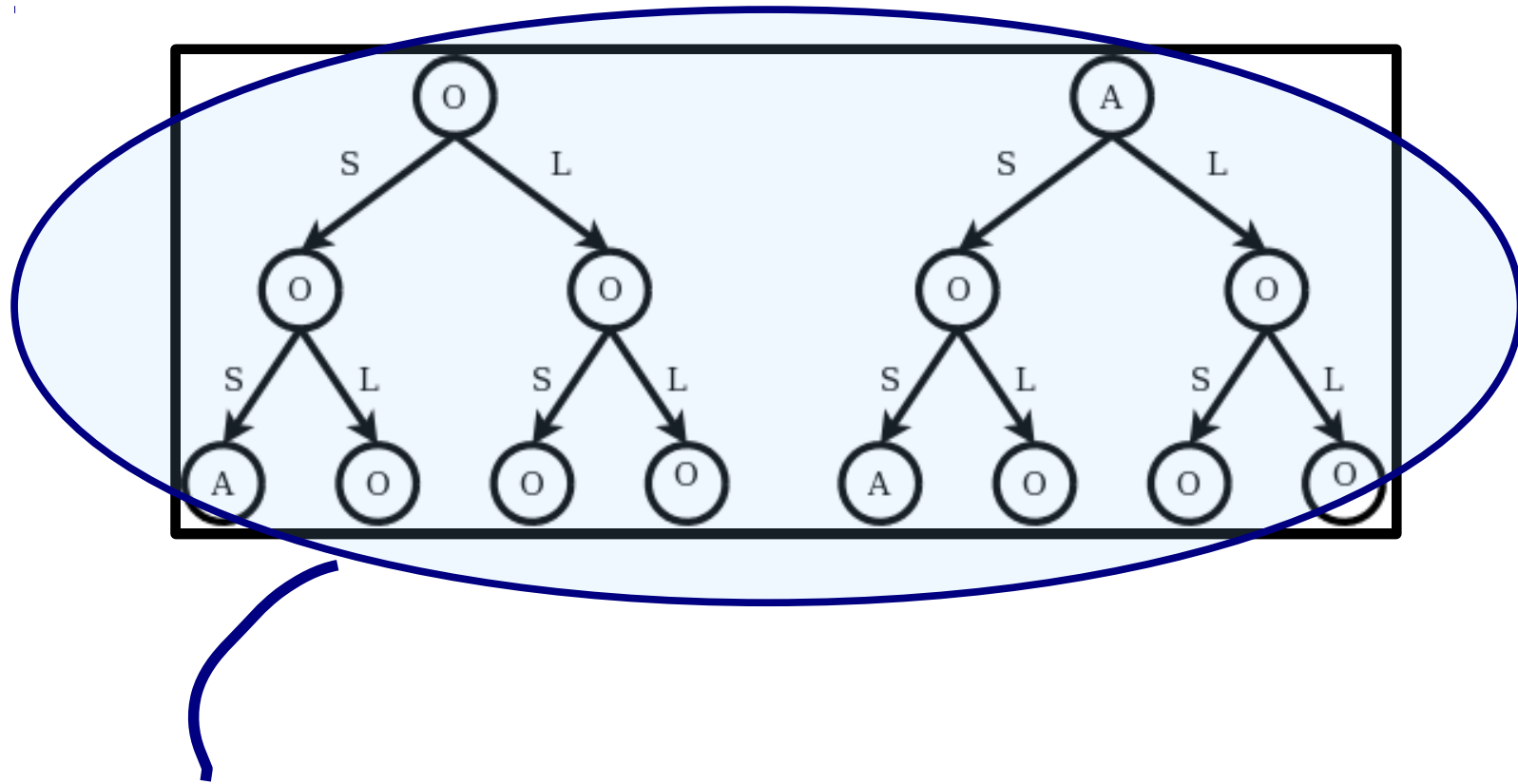
$$V(s, q^{\tau=k}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=k-1})$$

Illustration: Value of a Joint Policy



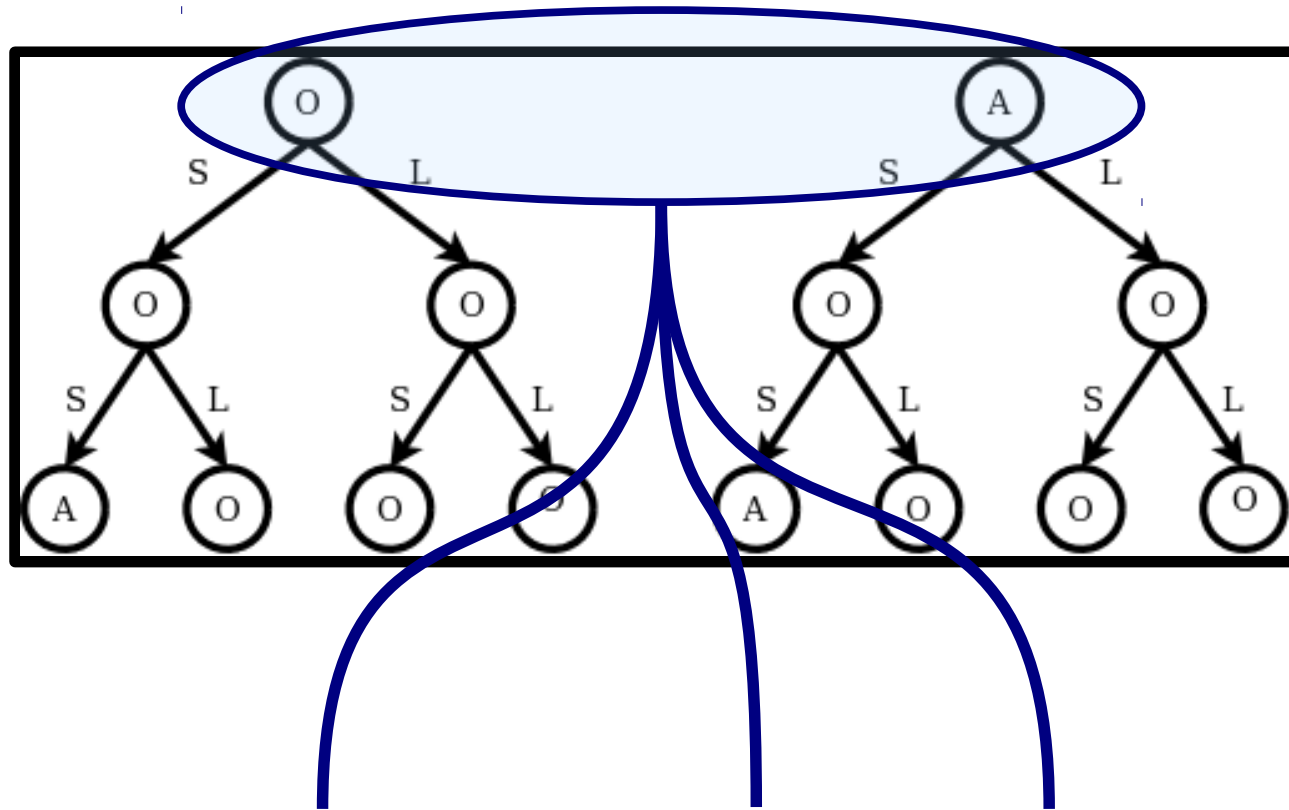
$$V(s, q^{\tau=3}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=2})$$

Illustration: Value of a Joint Policy



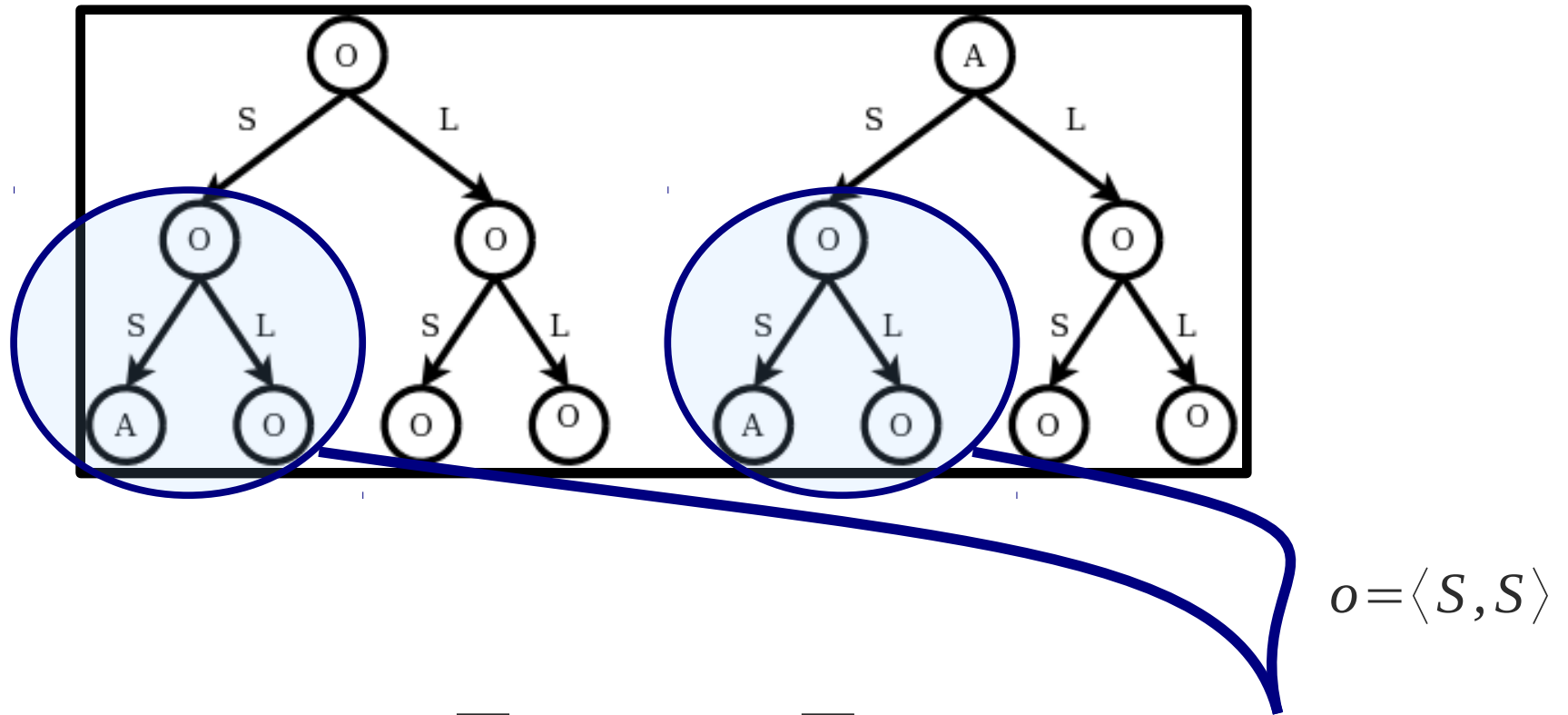
$$V(s, q^{\tau=3}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=2})$$

Illustration: Value of a Joint Policy



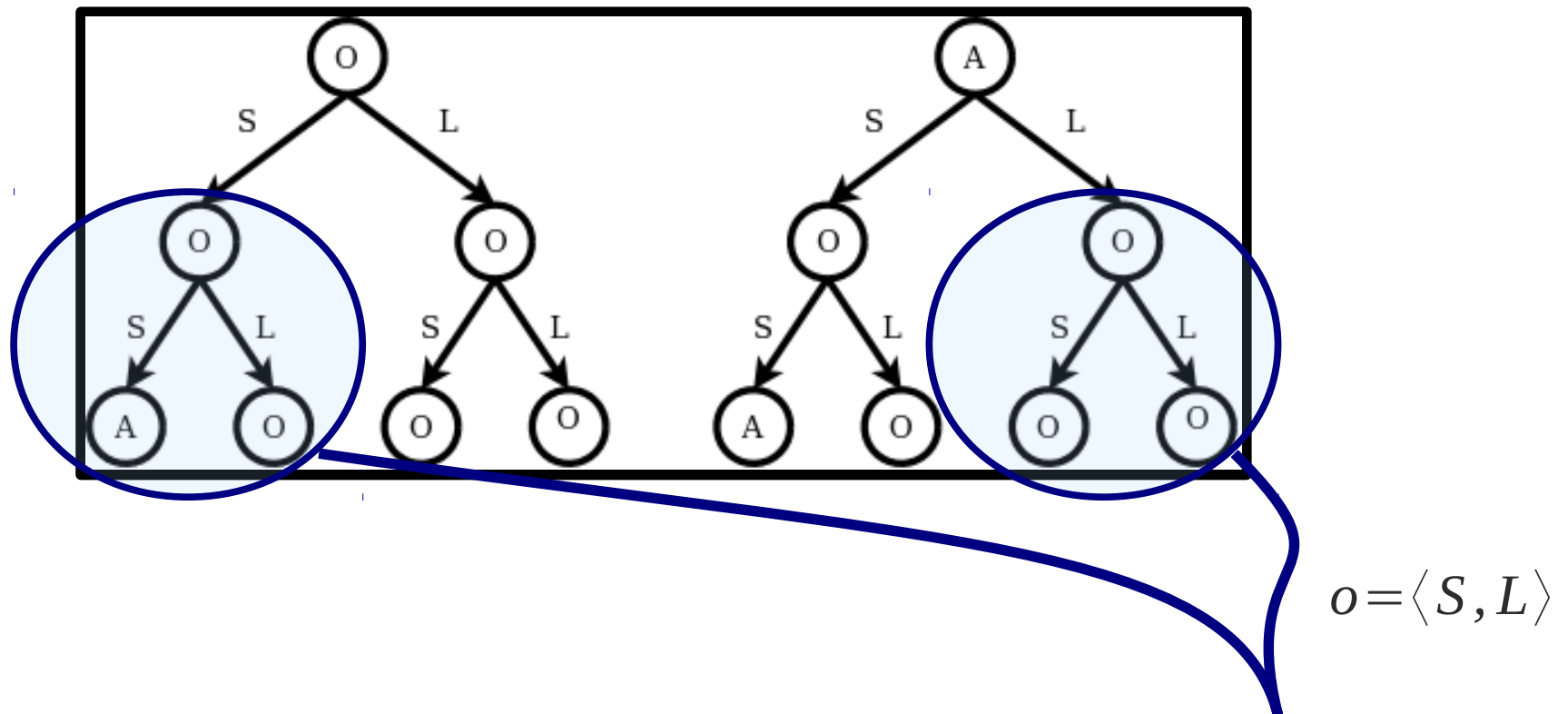
$$V(s, q^{\tau=3}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=2})$$

Illustration: Value of a Joint Policy



$$V(s, q^{\tau=3}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=2})$$

Illustration: Value of a Joint Policy



$$V(s, q^{\tau=3}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=2})$$

Brute Force Search

- We can compute the value of a joint policy $V(\pi)$
- So the **stupidest algorithm** is:
 - compute $V(\pi)$, for all π
 - select a π with maximum value
- Number of joint policies is huge!
(doubly exponential in horizon h)
- Clearly intractable...

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76
8	3.3520e+153

Value Functions - V2

- We want to think about **optimal** value functions...
...unfortunately not as straightforward as (PO)MDPs
- let us rewrite the value as:

$$V(\vec{\theta}, q^{\tau=k}) = R(\vec{\theta}, a) + \sum_o P(o|\vec{\theta}, a) V(\vec{\theta}', q^{\tau=k-1})$$

$$V(\vec{\theta}, q^{\tau=k}) = \sum_s P(s|\vec{\theta}, b^0) V(s, q^{\tau=k})$$

$$R(\vec{\theta}, a) = \sum_s P(s|\vec{\theta}, b^0) R(s, a)$$

$$P(o|\vec{\theta}, a) = \sum_s P(s|\vec{\theta}, b^0) \sum_{s'} P(s'|s, a) P(o|a, s')$$

Value Functions - V2

- We want to think about **optimal** value functions...
...unfortunately not as straightforward as (PO)MDPs
- let us rewrite the value as:

$$V(\vec{\theta}, q^{\tau=k}) = R(\vec{\theta}, a) + \sum_o P(o|\vec{\theta}, a) V(\vec{\theta}', q^{\tau=k-1})$$

$$V(\vec{\theta}, q^{\tau=k}) = \sum_s P(s|\vec{\theta}, b^0) V(s, q^{\tau=k})$$

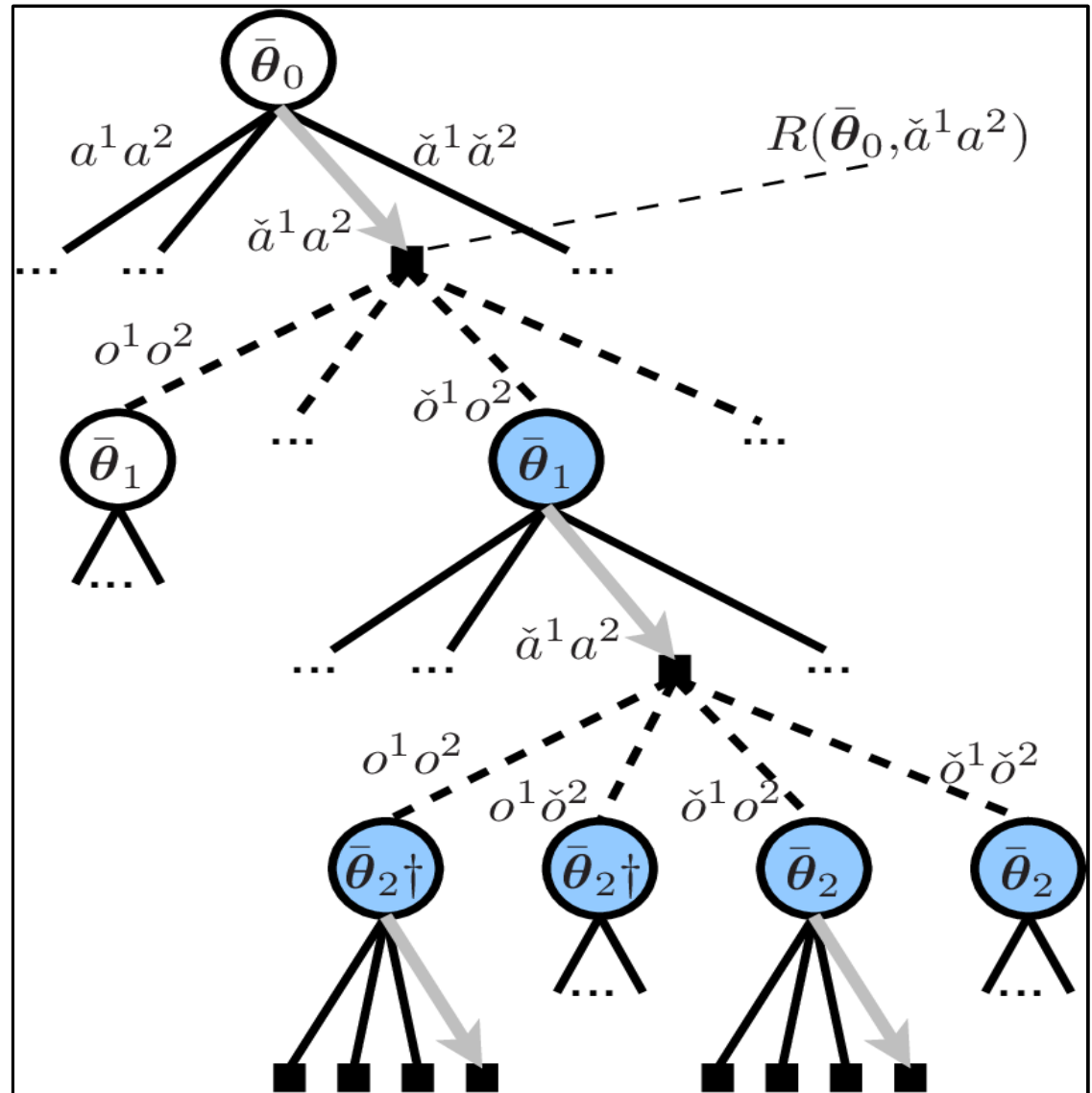
$$R(\vec{\theta}, a) = \sum_s P(s|\vec{\theta}, b^0) R(s, a)$$

$$P(o|\vec{\theta}, a) = \sum_s P(s|\vec{\theta}, b^0) \sum_{s'} P(s'|s, a) P(o|a, s')$$

Note:
this is the joint belief
(can be used at plan-time!)

Optimal Value Functions – 1

- Consider selecting the best joint sub-tree policy q^τ



- We *can* compute value...
...but *cannot* select the maximizing q^τ independently!

Optimal Value Functions – 2

- *Cannot* select the maximizing q^t independently...
 → Need to reason over assignment for all AOHs of a stage t simultaneously!
- Value at stage t $V^t = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | b^0, \varphi^t) V(\vec{\theta}^t, q^{\tau=h-t})$
 with **past** joint policy $\varphi^t = (\delta^0, \dots, \delta^{t-1})$
- Find mappings Γ_1, Γ_2 (from AOHs → sub-tree policies)
 that maximize $\sum_{\langle \theta_1, \theta_2 \rangle} P(\langle \theta_1, \theta_2 \rangle | b^0, \varphi) V(\langle \theta_1, \theta_2 \rangle, \langle \Gamma_1(\theta_1), \Gamma_2(\theta_2) \rangle)$



dependence on
history

dependence on
future

Optimal Value Functions – 3

- Optimal value function has dependence on past joint policy [Oliehoek 2012]

- Value propagation

- last stage $t=h-1$ $Q^*(\varphi^{h-1}, \vec{\theta}^{h-1}, \delta^{h-1}) = R(\vec{\theta}^{h-1}, \delta^{h-1}(\vec{\theta}^{h-1}))$
 - $t < h-1$

$$Q^*(\varphi^t, \vec{\theta}^t, \delta^t) = R(\vec{\theta}^t, \delta^t(\vec{\theta}^t)) + \sum_o P(o | \vec{\theta}^t, \delta^t(\vec{\theta}^t)) Q^*(\varphi^{t+1}, \vec{\theta}^{t+1}, \delta^{*t+1})$$

$$\varphi^{t+1} = (\varphi^t, \delta^t)$$

- Value optimization

$$\delta^{*t+1} = \arg \max_{\delta^{t+1}} \sum_{\vec{\theta}^{t+1}} P(\vec{\theta}^{t+1} | b^0, \varphi^{t+1}) Q^*(\varphi^{t+1}, \vec{\theta}^{t+1}, \delta^{t+1})$$

Optimal Value Functions – 3

Q vs V ?

- Optimal value function has dependence on past joint policy [oliehoek] → we can interpret it as a 'plan-time' MDP
 - state: φ
 - actions: δ

- Value propagation

- last stage $t=h-1$
- $t < h-1$

$$Q^*(\varphi^t, \delta^t) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | b^0, \varphi^t) Q^*(\varphi^t, \vec{\theta}^t, \delta^t)$$

$$V(\varphi^t) = \max_{\delta^t} Q^*(\varphi^t, \delta^t)$$

$$Q^*(\varphi^t, \vec{\theta}^t, \delta^t) = R(\vec{\theta}^t, \delta^t(\vec{\theta}^t)) + \sum_o P(o | \vec{\theta}^t, \delta^t(\vec{\theta}^t)) Q^*(\varphi^{t+1}, \vec{\theta}^{t+1}, \delta^{t+1})$$

o

$$\varphi^{t+1} = (\varphi^t, \delta^t)$$

- Value optimization

$$\delta^{*t+1} = \arg \max_{\delta^{t+1}} \sum_{\vec{\theta}^{t+1}} P(\vec{\theta}^{t+1} | b^0, \varphi^{t+1}) Q^*(\varphi^{t+1}, \vec{\theta}^{t+1}, \delta^{t+1})$$

Optimal Value Functions – 3

Q vs V ?

- Optimal value function has dependence on past joint policy [Oliehoek] → we can interpret it as a 'plan-time' MDP

- state: φ
- actions: δ

- Value propagation

- last stage $t=h-1$
- $t < h-1$

$$Q^*(\varphi^t, \delta^t) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | b^0, \varphi^t) Q^*(\varphi^t, \vec{\theta}^t, \delta^t)$$

$$V(\varphi^t) = \max_{\delta^t} Q^*(\varphi^t, \delta^t)$$

$$Q^*(\varphi^t, \vec{\theta}^t, \delta^t) = R(\vec{\theta}^t, \delta^t(\vec{\theta}^t)) + \sum_o P(o | \vec{\theta}^t, \delta^t(\vec{\theta}^t)) Q^*(\varphi^{t+1}, \vec{\theta}^{t+1}, \delta^{*t+1})$$

o

- Value optimization

Applicability

$$\delta^{*t+1} = \arg \max_{\delta^t}$$

- constitutes a DP: theoretically nice
- but number of φ^t, δ^t just too large...

Optimal Value Functions – 3

Q vs V ?

- Optimal value function has dependence on past joint policy [Oliehoek] → we can interpret it as a 'plan-time' MDP
 - state: φ
 - actions: δ
- Value propagation
 - last stage $t=h-1$

$$Q^*(\varphi^t, \delta^t) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | b^0, \varphi^t) Q^*(\varphi^t, \vec{\theta}^t, \delta^t)$$

Recent development:

substitute φ^t by a sufficient statistic.
[Oliehoek 2013, Dibangoye et al. 2013]

$$V(\varphi^t) = \max_{\delta^t} Q^*(\varphi^t, \delta^t)$$

- Value optimization

Applicability

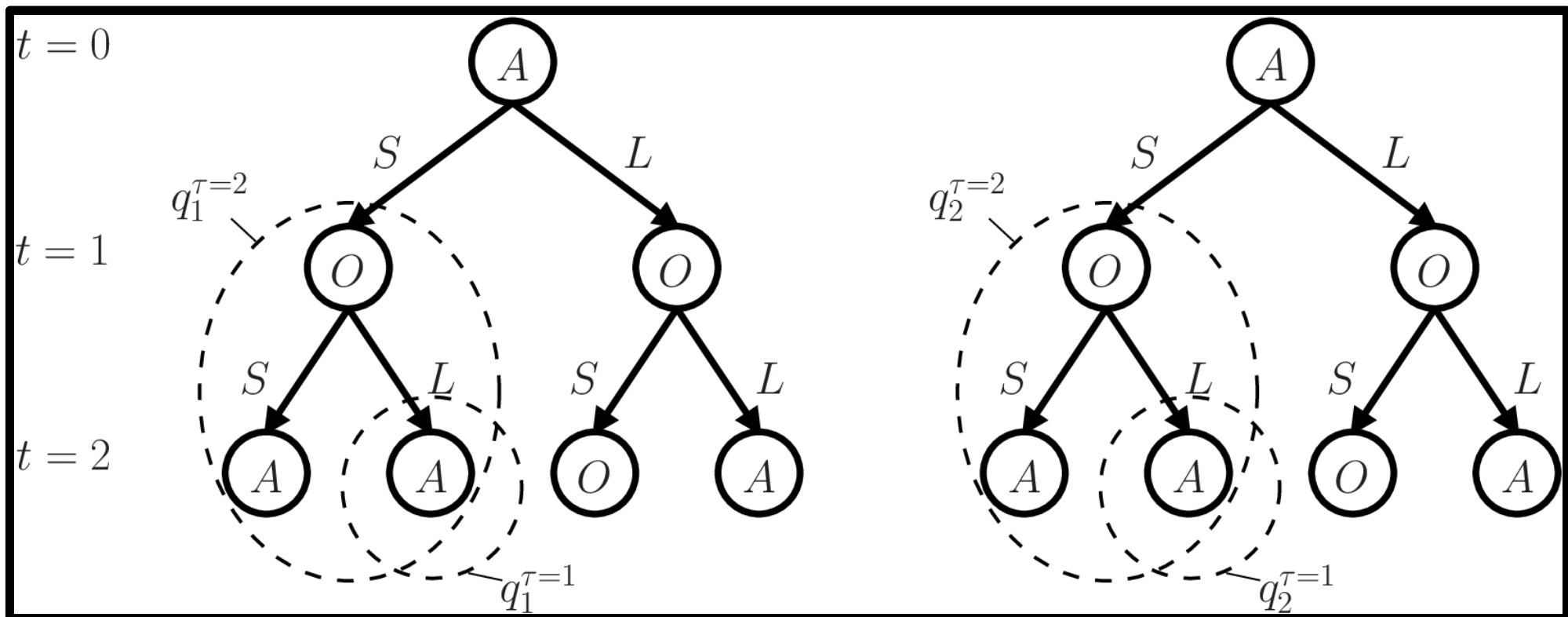
$$\delta^{*t+1} = \arg \max_{\delta^t}$$

- constitutes a DP: theoretically nice
- but number of φ^t, δ^t just too large...

Dynamic Programming for Dec-POMDPs

Dynamic Programming – 1

- Generate all policies in a special way:
 - from 1 stage-to-go policies $Q^{\tau=1}$
 - construct all 2-stages-to-go policies $Q^{\tau=2}$, etc.

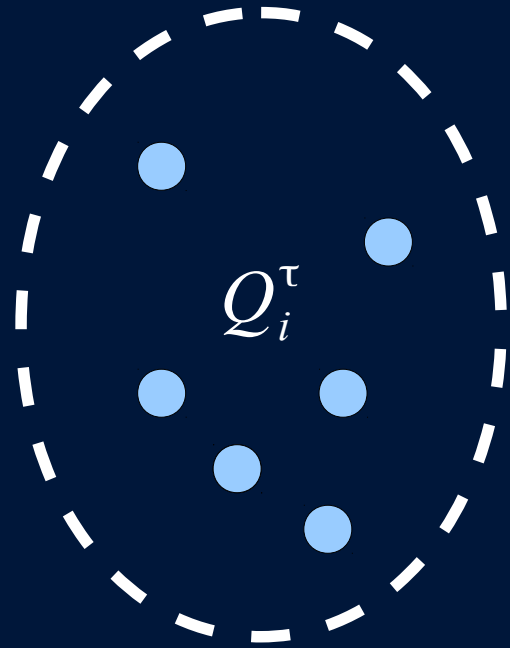


Dynamic Programming – 1

Exhaustive backup operation

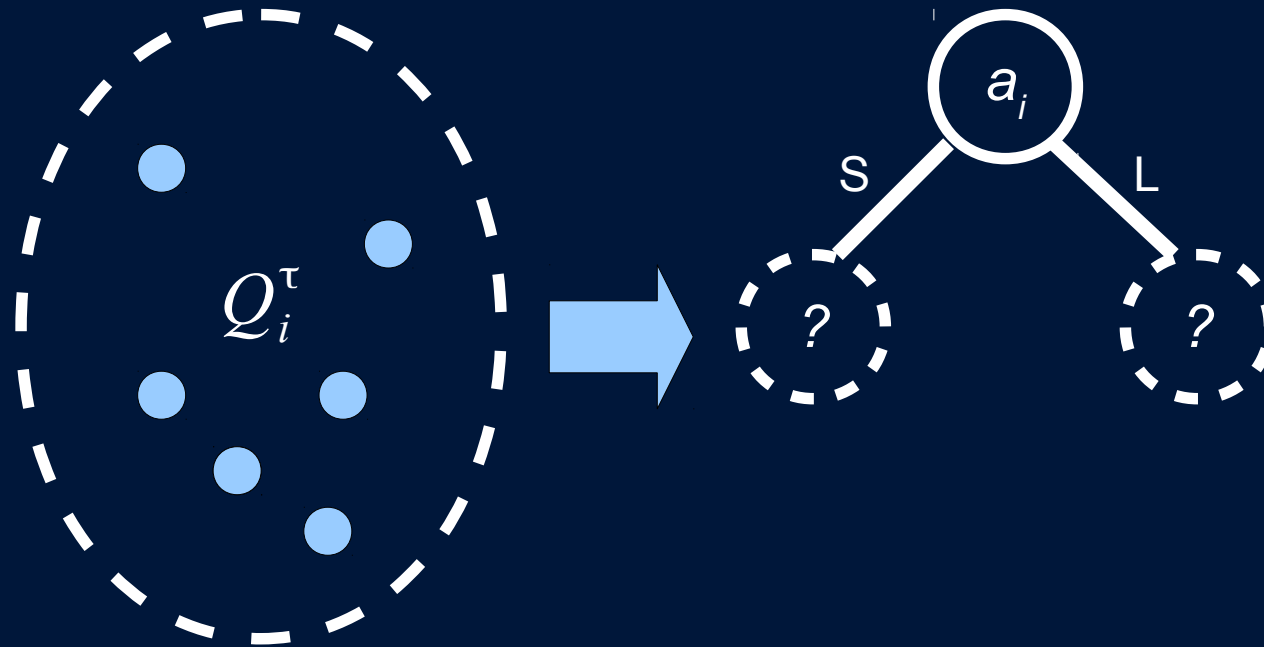
Dynamic Programming – 1

Exhaustive backup operation



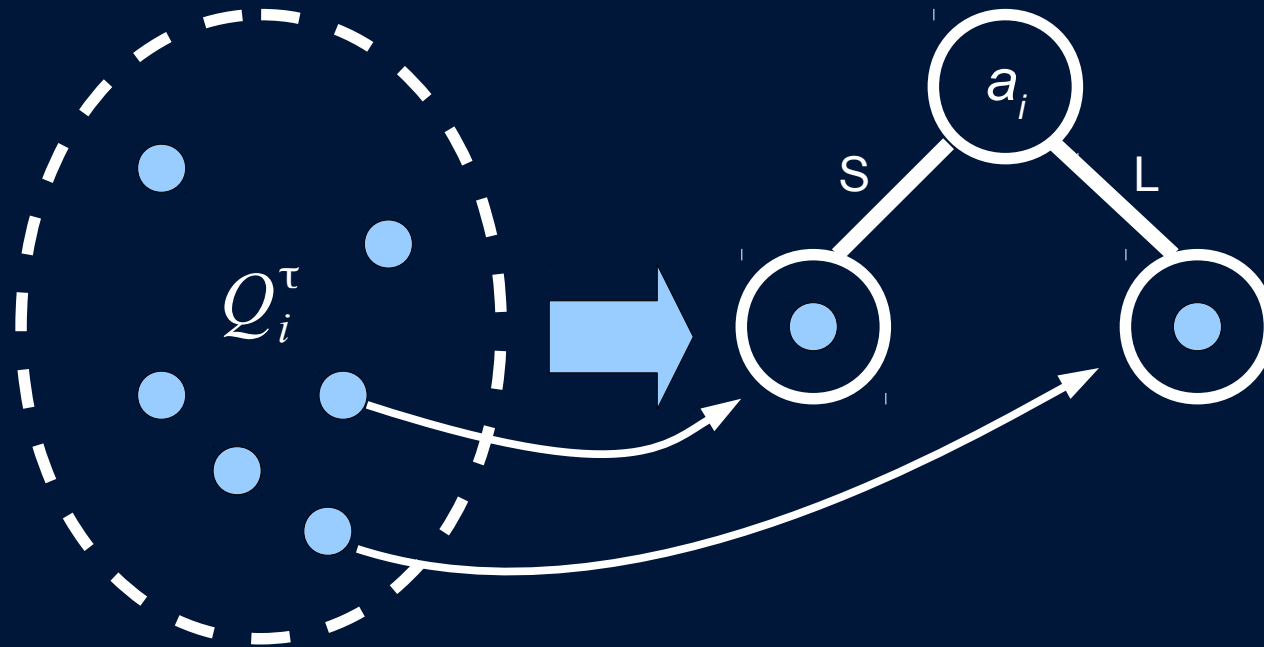
Dynamic Programming – 1

Exhaustive backup operation

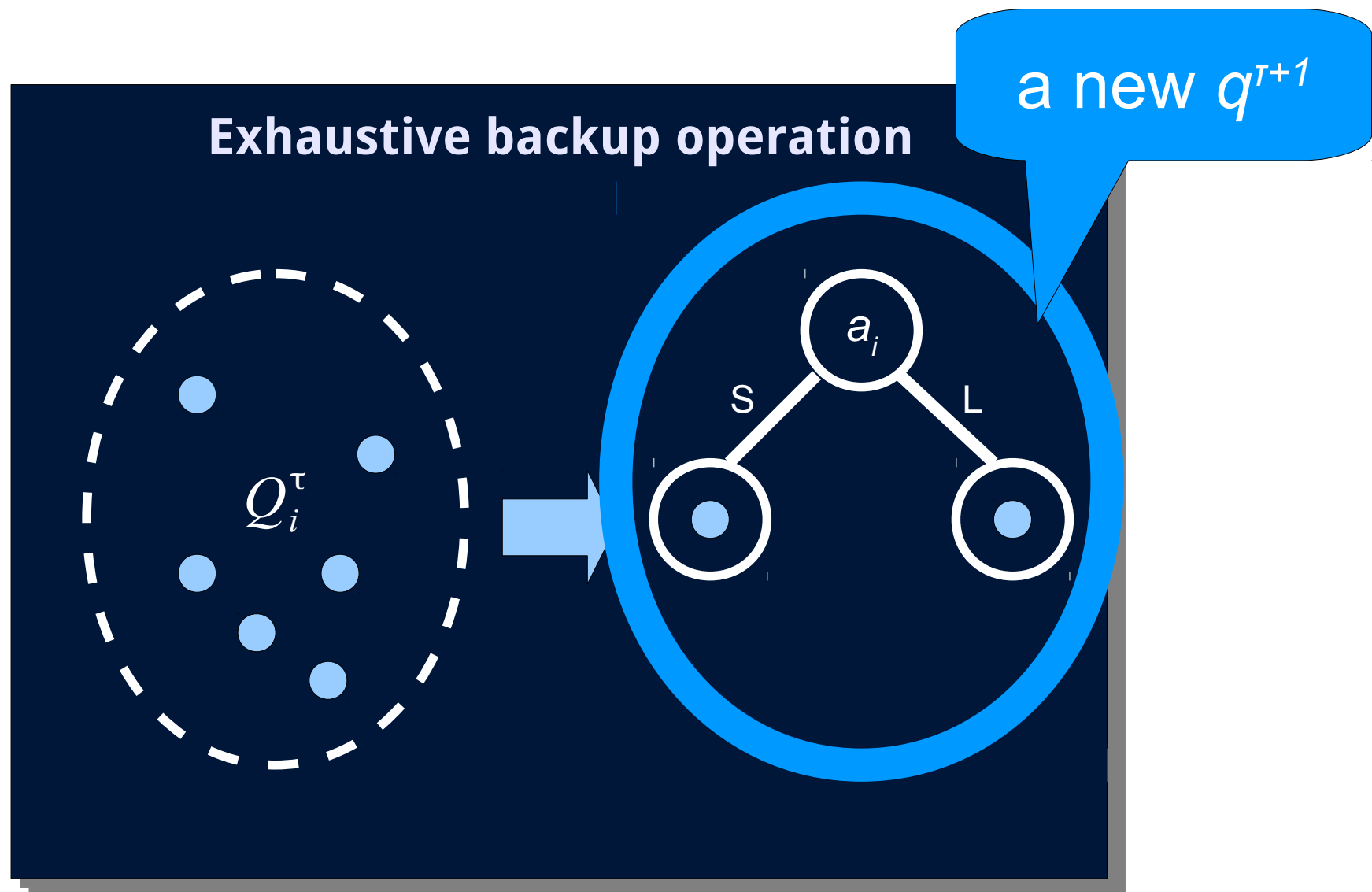


Dynamic Programming – 1

Exhaustive backup operation



Dynamic Programming – 1



Dynamic Programming – 1

Exhaustive backup operation

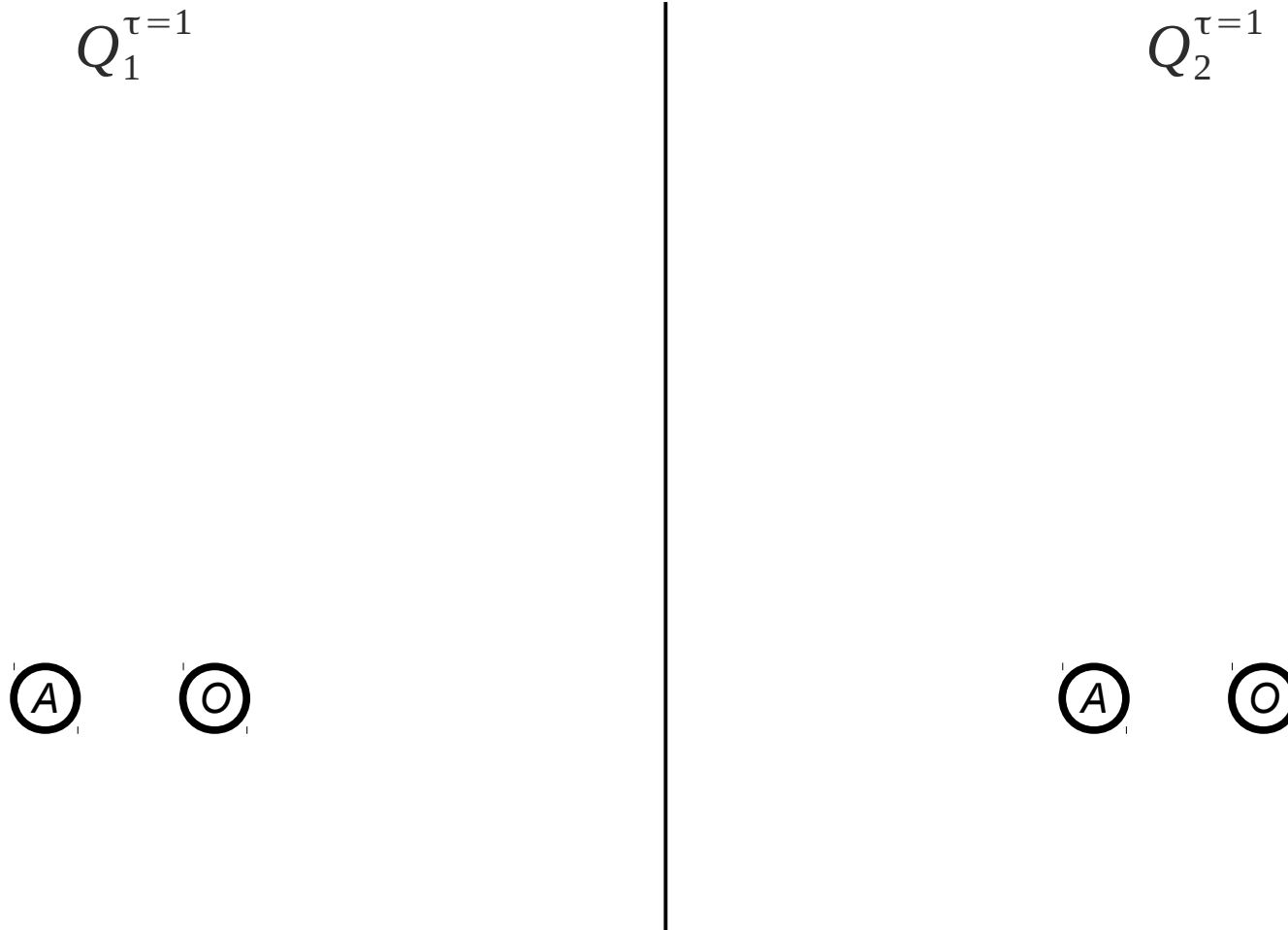


To generate all $Q^{\tau+1}$

- All actions
- All assignments of q^τ to observations

Dynamic Programming – 2

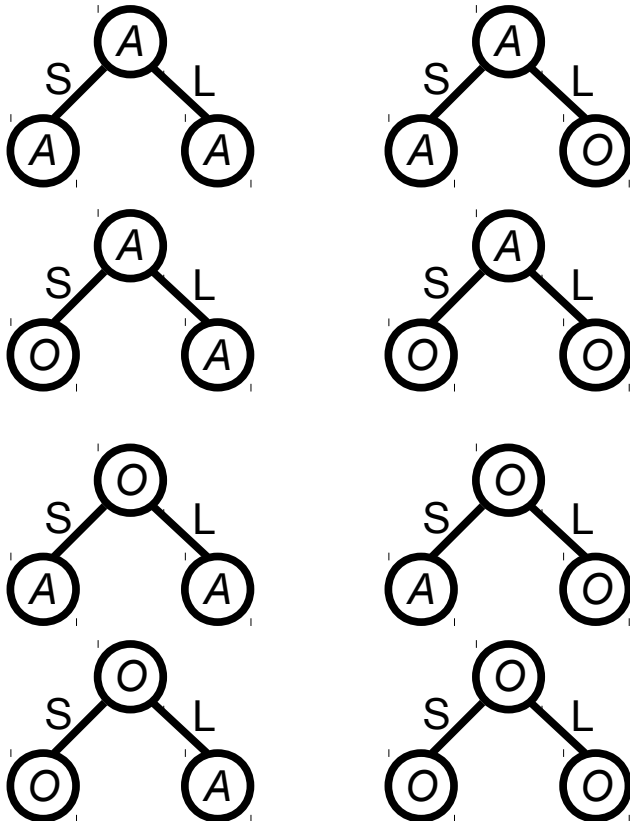
- (obviously) this scales very poorly...



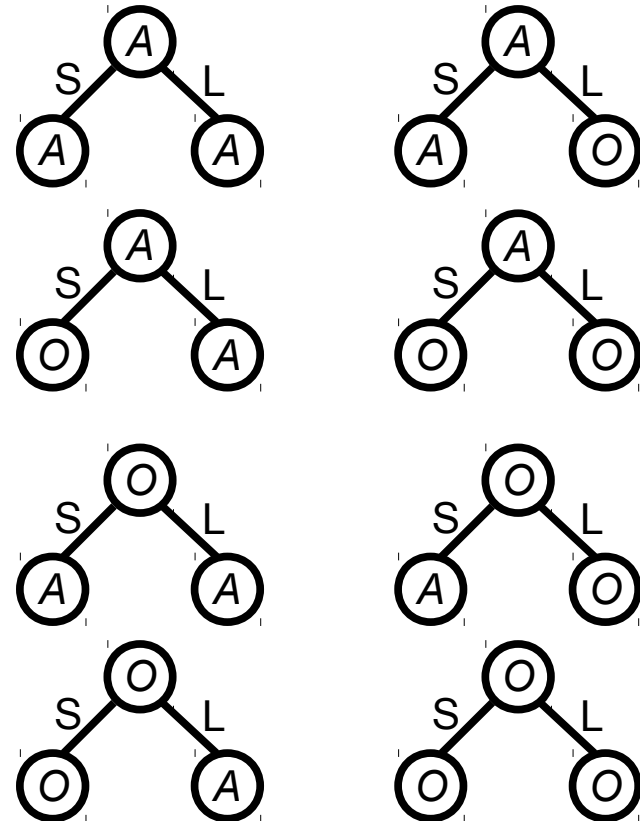
Dynamic Programming – 2

- (obviously) this scales very poorly...

$Q_1^{\tau=2}$



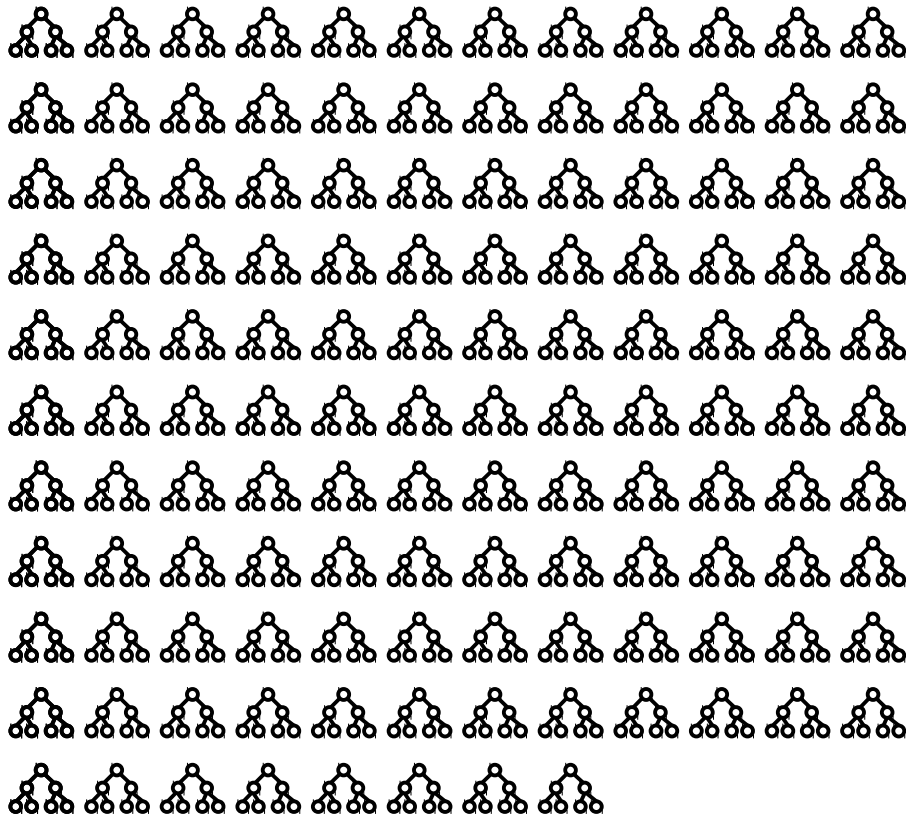
$Q_2^{\tau=2}$



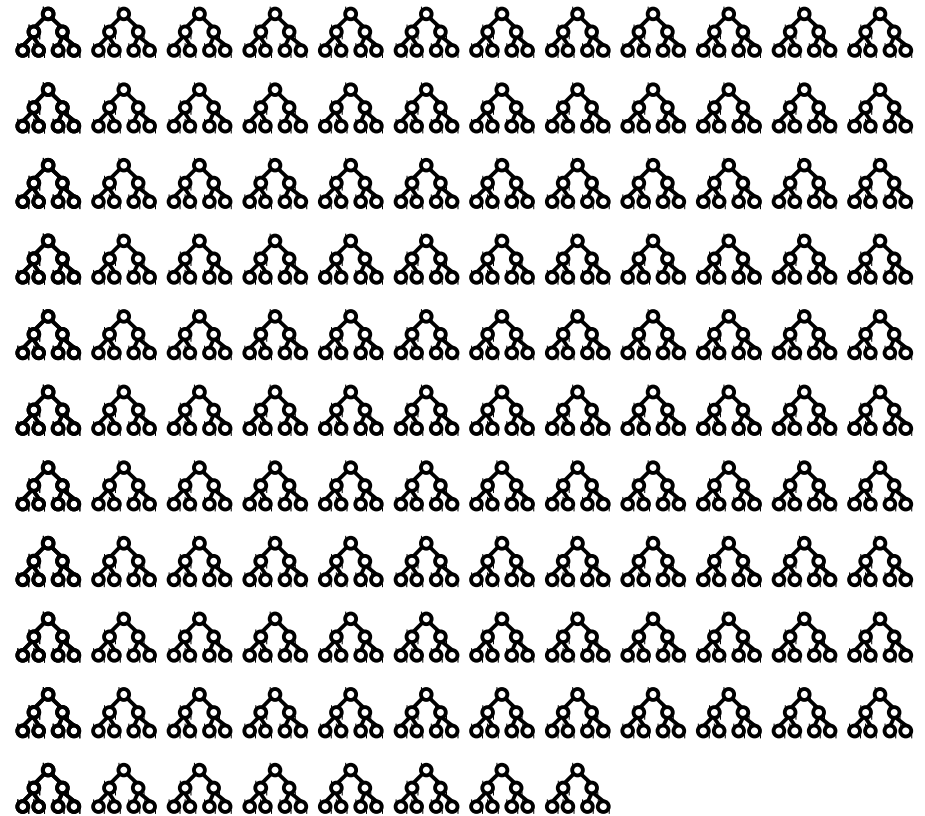
Dynamic Programming – 2

- (obviously) this scales very poorly...

$$Q_1^{\tau=3}$$



$$Q_2^{\tau=3}$$



Dynamic Programming – 2

- (obviously) this scales very poorly...



Dynamic Programming – 3

- Perhaps not all those Q_i^τ are useful!
 - Perform **pruning** of 'dominated policies'!

- Algorithm [Hansen et al. 2004] $Q_i^{\tau=1} = A_i$

```
Initialize Q1(1), Q2(1)
for tau=2 to h
    Q1(tau) = ExhaustiveBackup(Q1(tau-1))
    Q2(tau) = ExhaustiveBackup(Q2(tau-1))
    Prune(Q1, Q2, tau)
end
```

Dynamic Programming – 3

- Perhaps not all those Q_i^τ are useful!
 - Perform **pruning** of 'dominated policies'!

- Algorithm [Hansen et al. 2004] $Q_i^{\tau=1} = A_i$

```
Initialize Q1(1), Q2(1)
for tau=2 to h
    Q1(tau) = ExhaustiveBackup(Q1(tau-1))
    Q2(tau) = ExhaustiveBackup(Q2(tau-1))
    Prune(Q1, Q2, tau)
end
```

Note: cannot prune independently!

► usefulness of a q_1 depends on Q_2

► and vice versa

→ **Iterated elimination** of policies

Intermezzo: Pruning

- Basic procedure:
 - Loop over agents i ,
 - loop over q_i ,
 - check if dominated: LP
- Remember value: $V(s, q^\tau)$
- rewrite as: $V(s, q_{-i}^\tau, q_i^\tau)$

Also, easy to compute at each iteration:

$$V(s, q^{\tau=k}) = R(s, a) + \sum_{s'} P(s'|s, a) \sum_o P(o|a, s') V(s', q^{\tau=k-1})$$

Intermezzo: Pruning

- Basic procedure:
 - Loop over agents i ,
 - loop over q_i ,
 - check if dominated: LP
- Remember value: $V(s, q^\tau)$
- rewrite as: $V(s, q_{-i}^\tau, q_i^\tau)$

variables: $\epsilon; x(\hat{q}_i), \forall \hat{q}_i \in Q_i^\tau \setminus q_i^\tau$

maximize: ϵ

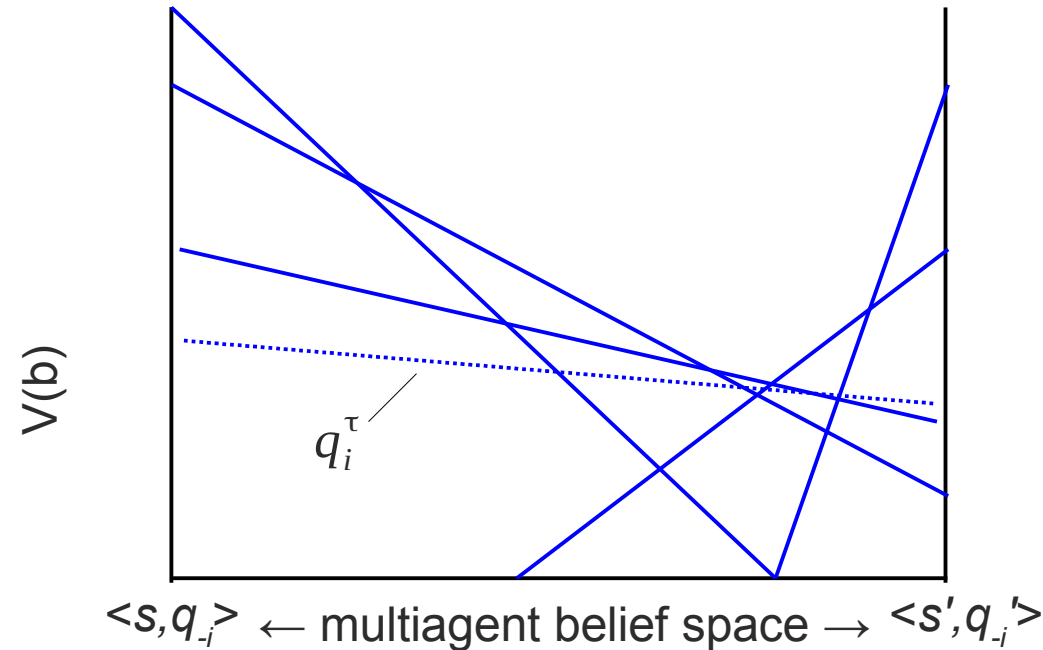
subject to:

$$\forall s, q_{-i}^\tau \quad V(s, q_{-i}^\tau, q_i^\tau) + \epsilon \leq \sum_{\hat{q}_i} x(\hat{q}_i) V(s, q_{-i}^\tau, \hat{q}_i)$$

$$\forall \hat{q}_i \quad x(\hat{q}_i) \geq 0, \quad \sum_{\hat{q}_i} x(\hat{q}_i) = 1$$

Prune q_i^τ when there is a distribution x (over other sub-tree policies \hat{q}_i) that achieves higher value $\forall s, q_{-i}^\tau$

Perhaps more intuitive:



- Basic p

- Look
- loop
- che

- Remember

- rewrite as:

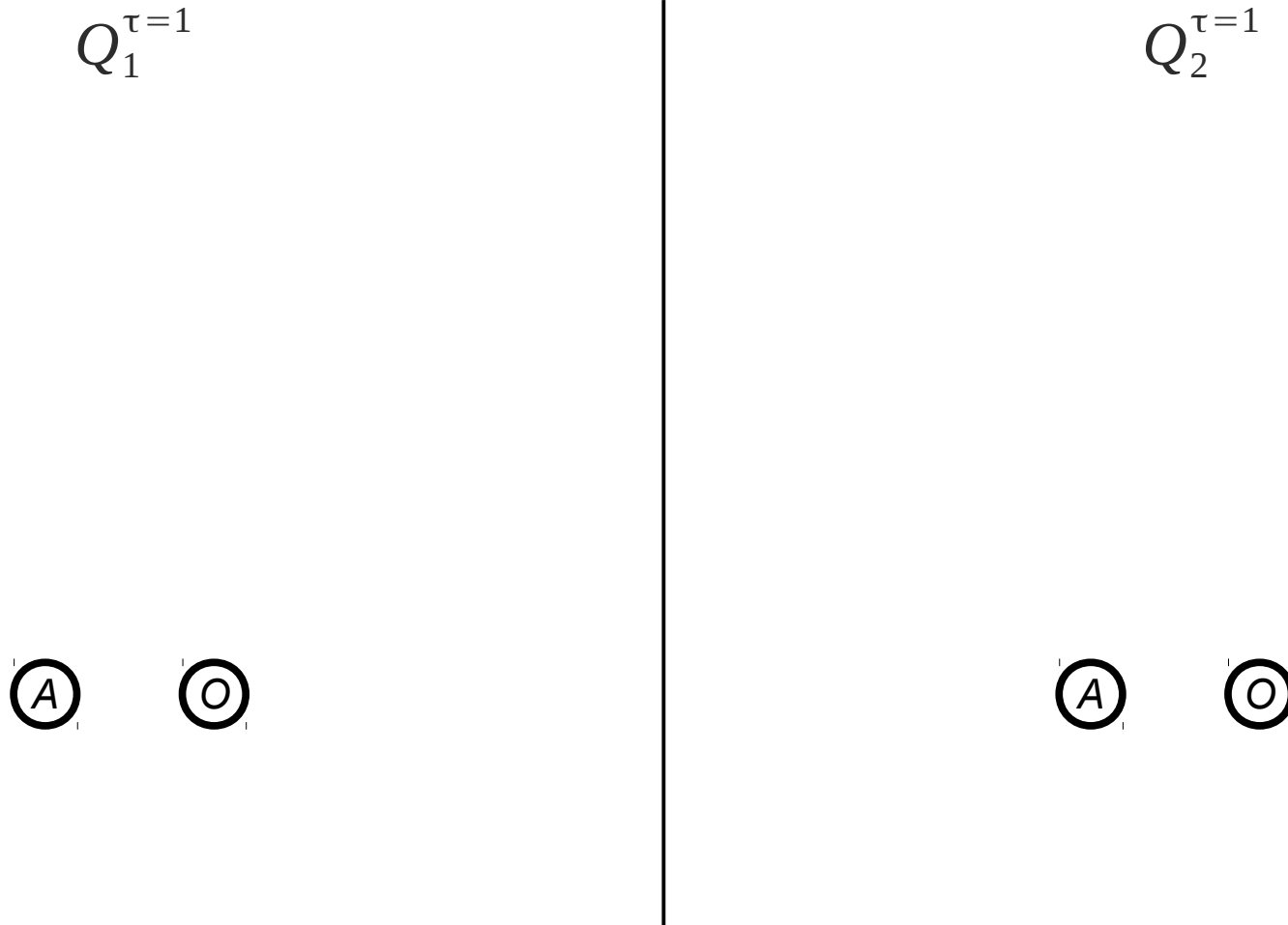
$$V(s, q_{-i}^\tau, q_i^\tau)$$

Prune q_i^τ when there is a distribution x (over other sub-tree policies \hat{q}_i) that achieves higher value $\forall s, q_{-i}^\tau$

A distribution $b(s, q_{-i}^\tau)$
is called 'multiagent belief'

Dynamic Programming – 4

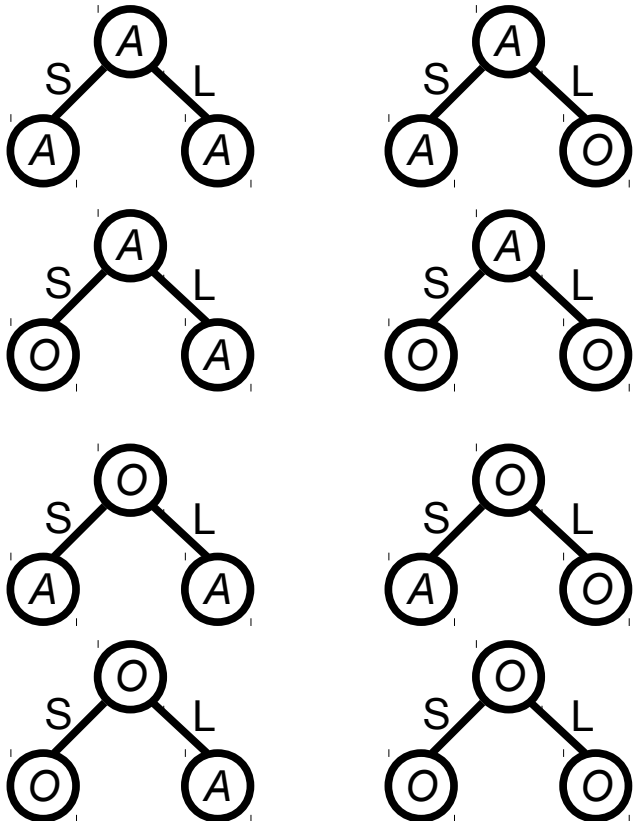
- Initialization



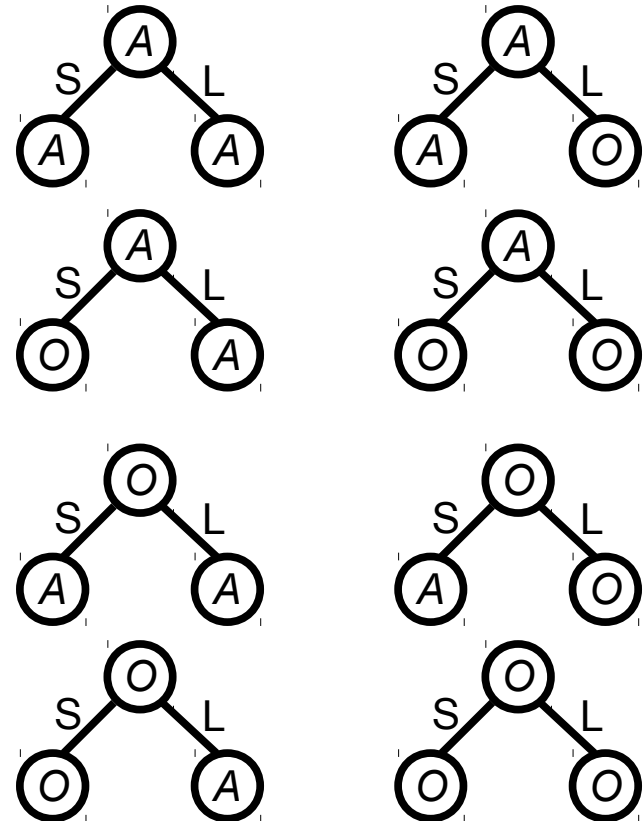
Dynamic Programming – 4

- Exhaustive Backups gives

$Q_1^{\tau=2}$



$Q_2^{\tau=2}$

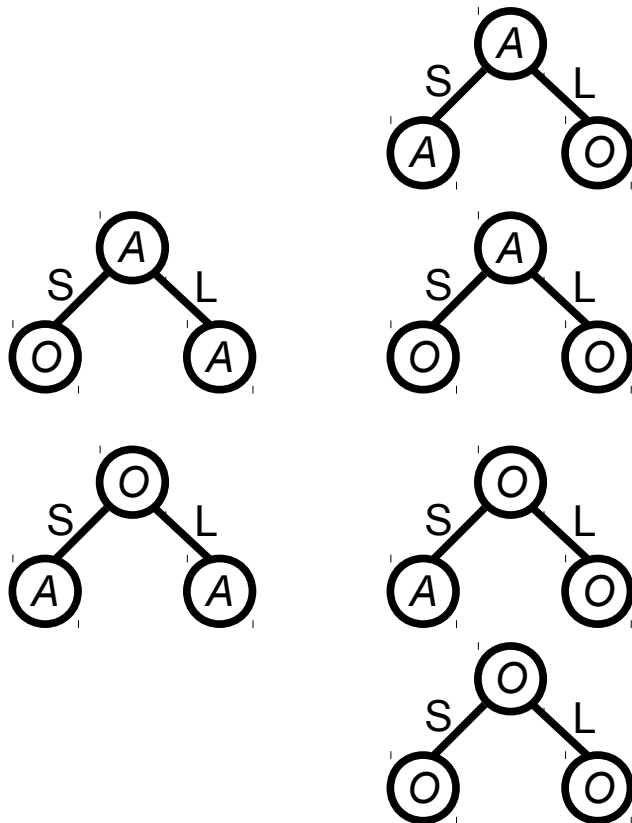


Dynamic Programming – 4

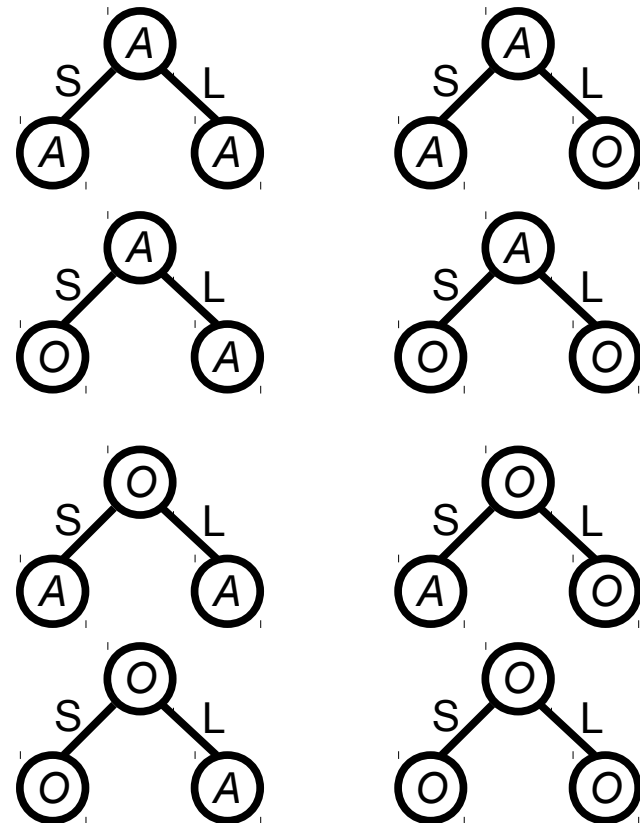
- Pruning agent 1...

Hypothetical Pruning
(not the result of actual pruning)

$Q_1^{\tau=2}$



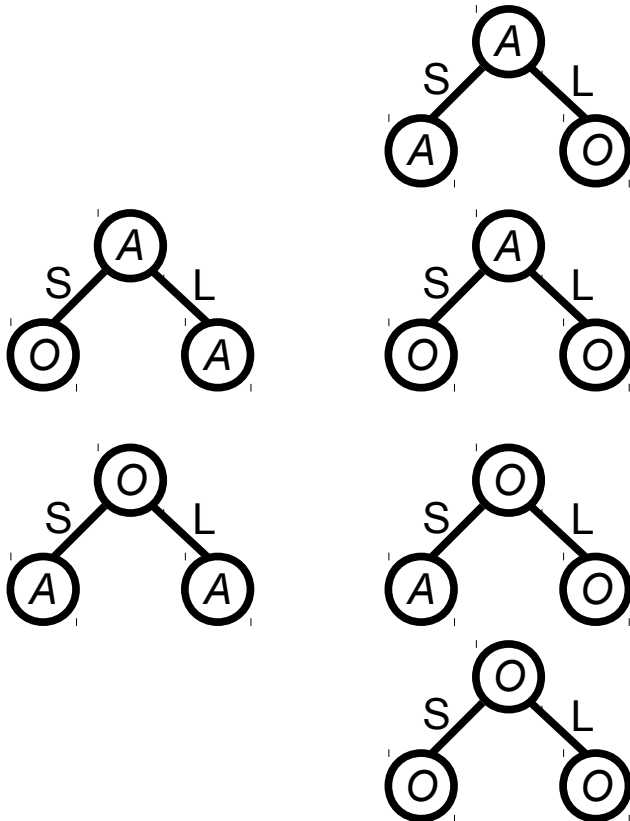
$Q_2^{\tau=2}$



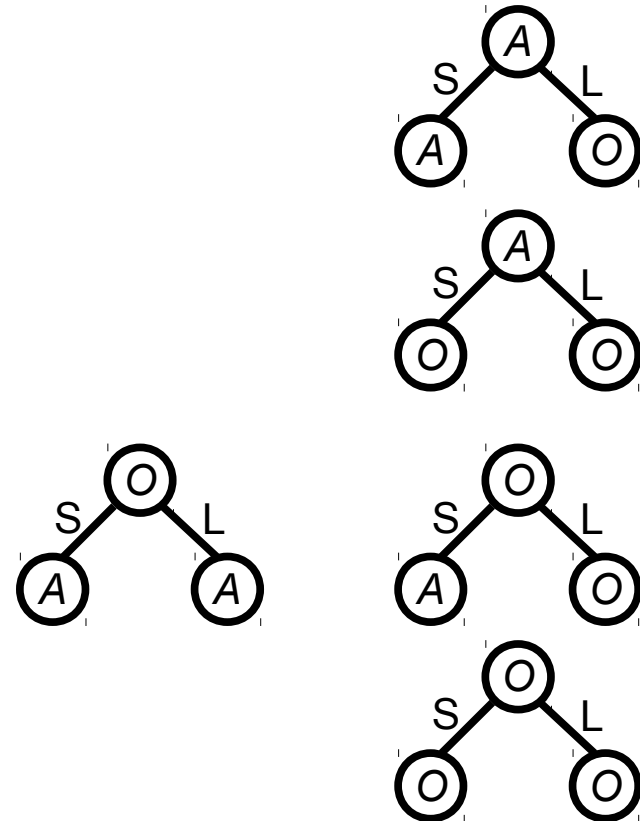
Dynamic Programming – 4

- Pruning agent 2...

$Q_1^{\tau=2}$



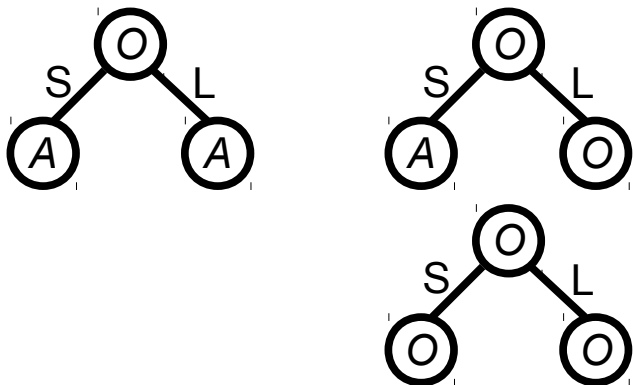
$Q_2^{\tau=2}$



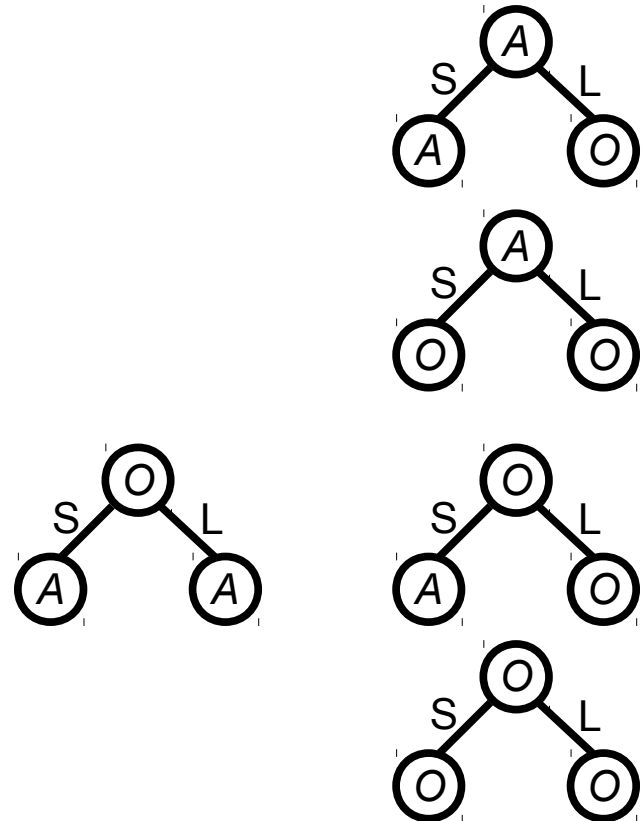
Dynamic Programming – 4

- Pruning agent 1...

$Q_1^{\tau=2}$



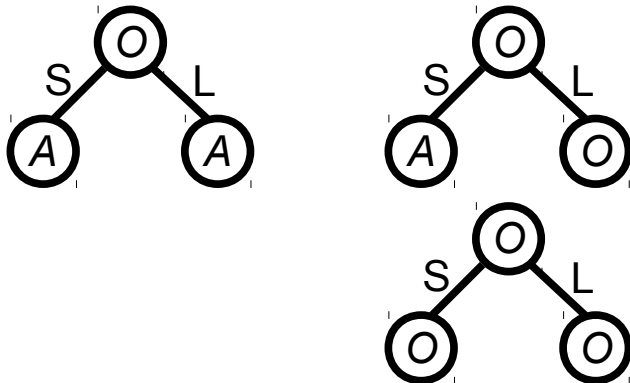
$Q_2^{\tau=2}$



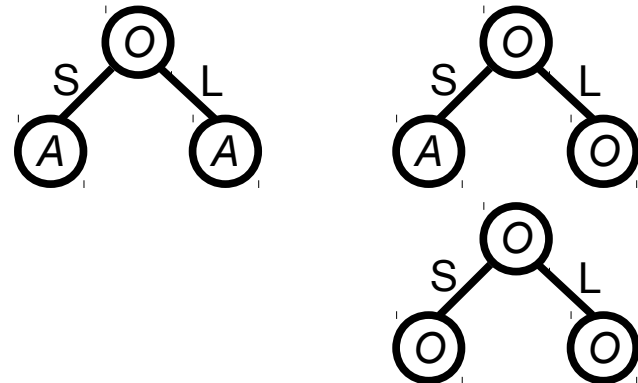
Dynamic Programming – 4

- Etc...

$$Q_1^{\tau=2}$$



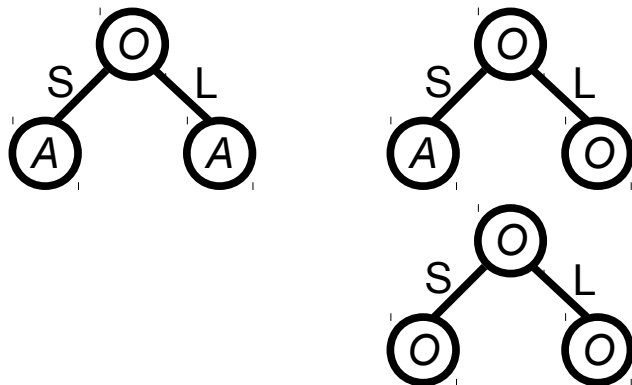
$$Q_2^{\tau=2}$$



Dynamic Programming – 4

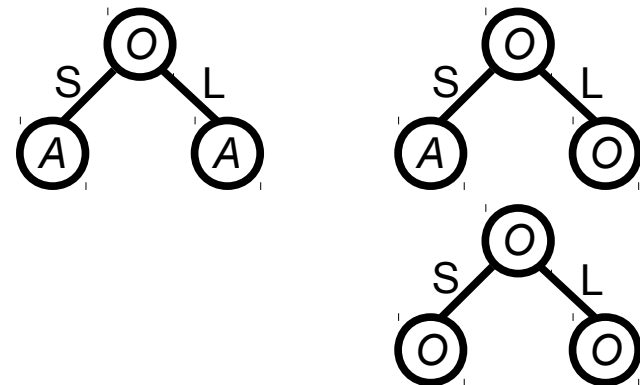
- Etc...

$Q_1^{\tau=2}$



$Q_2^{\tau=2}$

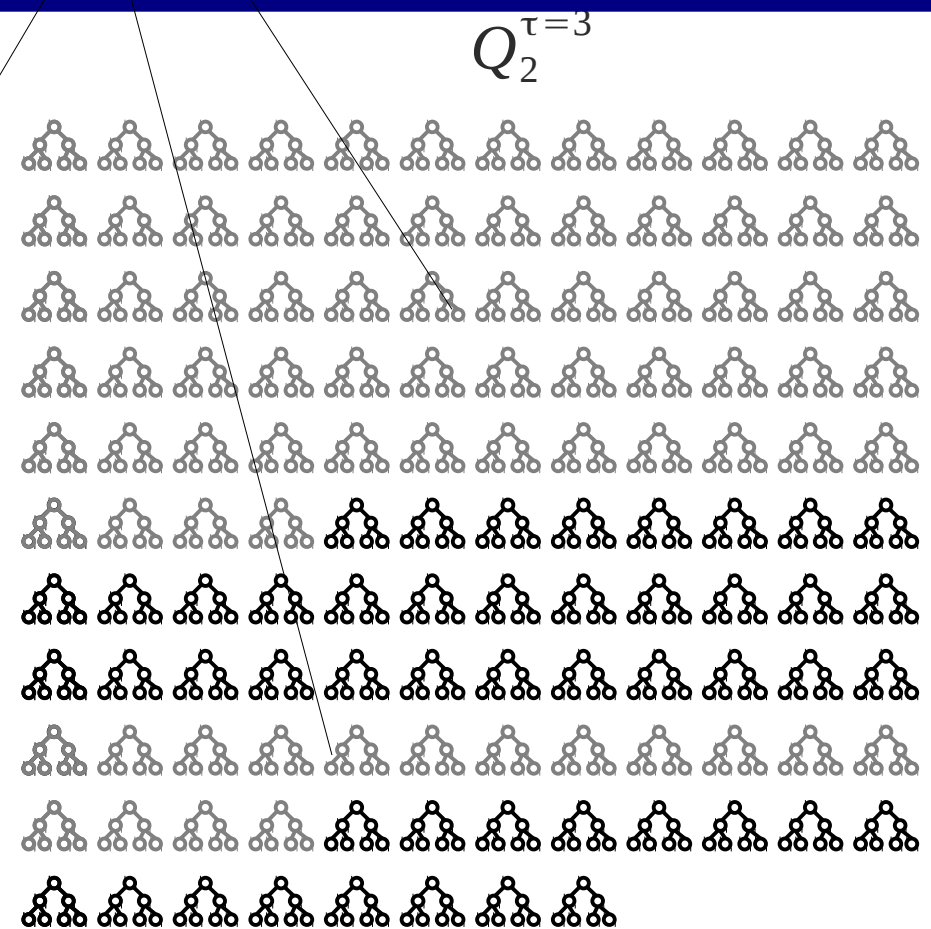
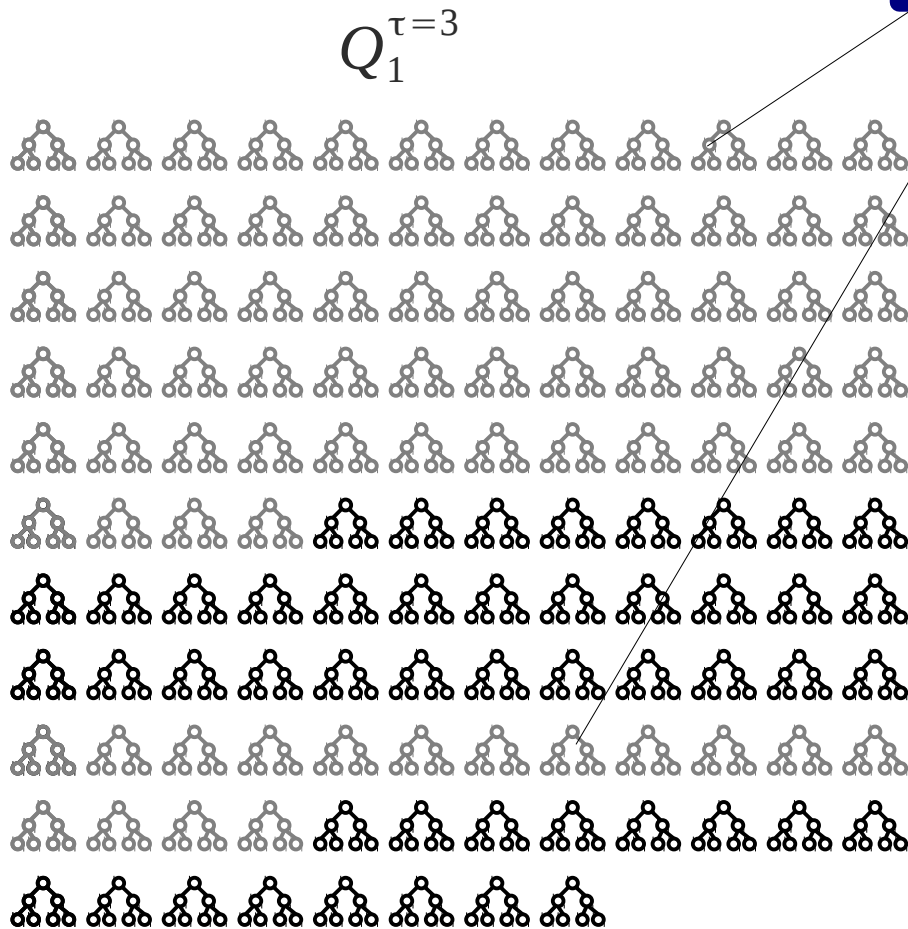
In this case: symmetric
→ but need not be in general!



Dynamic Programming – 4

- Exhaustive backups:

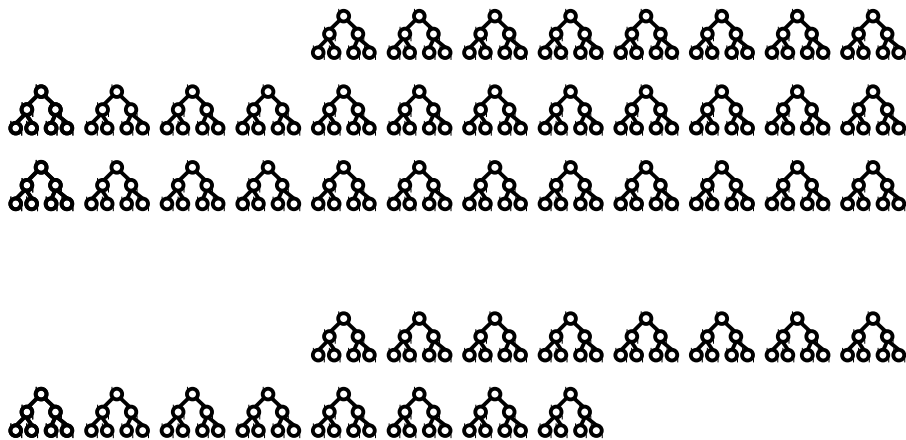
We **avoid** generation of many policies!



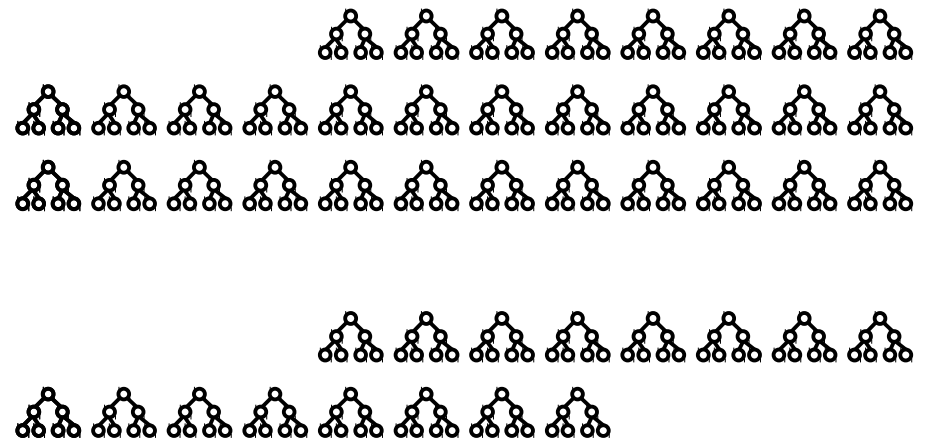
Dynamic Programming – 4

- Exhaustive backups:

$$Q_1^{\tau=3}$$



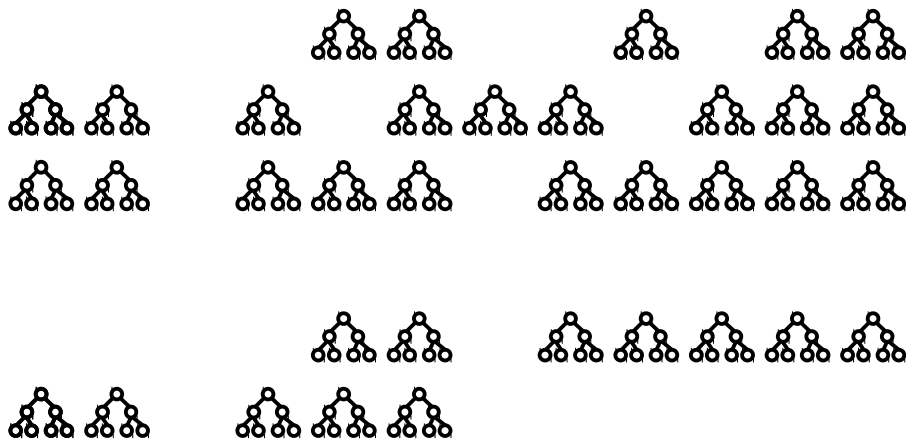
$$Q_2^{\tau=3}$$



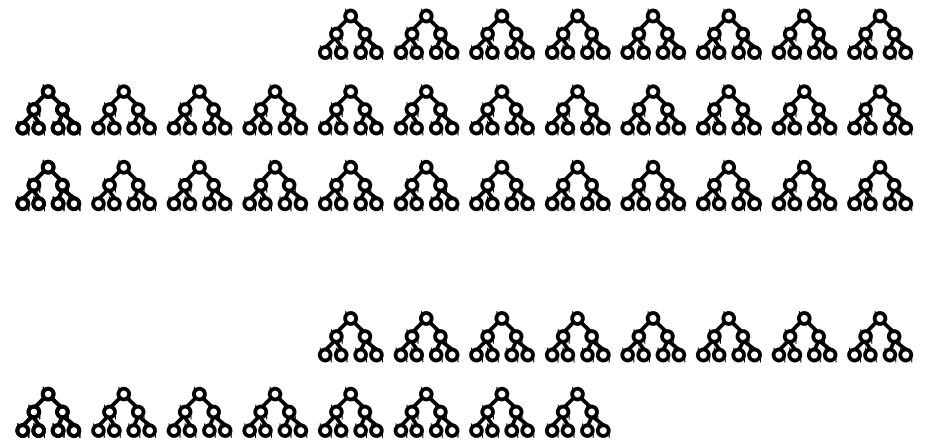
Dynamic Programming – 4

- Pruning agent 1...

$$Q_1^{\tau=3}$$



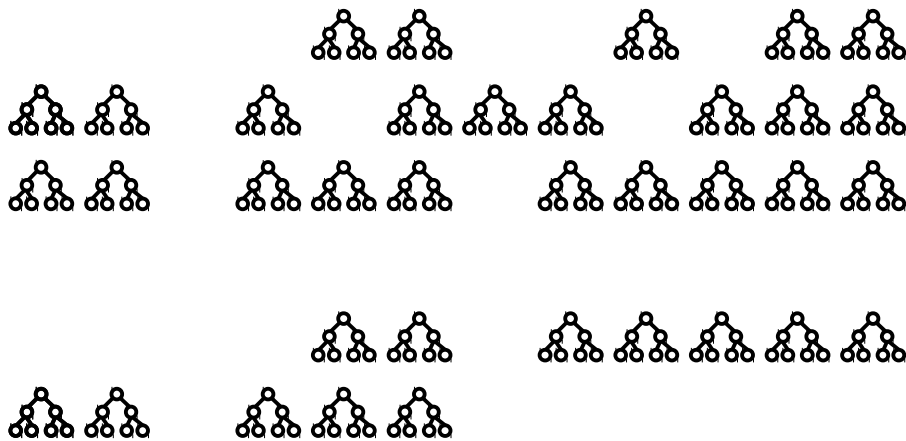
$$Q_2^{\tau=3}$$



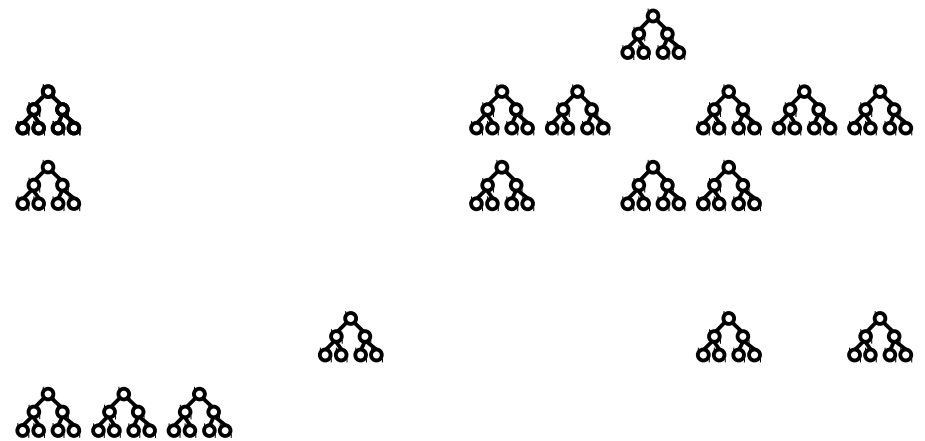
Dynamic Programming – 4

- Pruning agent 2...

$$Q_1^{\tau=3}$$



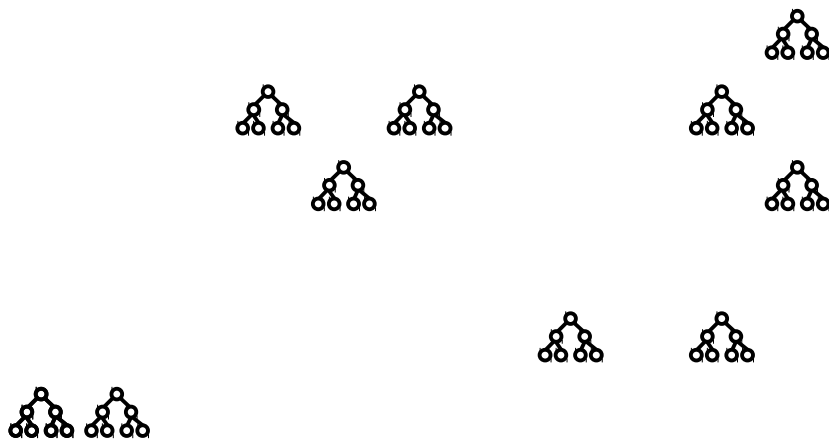
$$Q_2^{\tau=3}$$



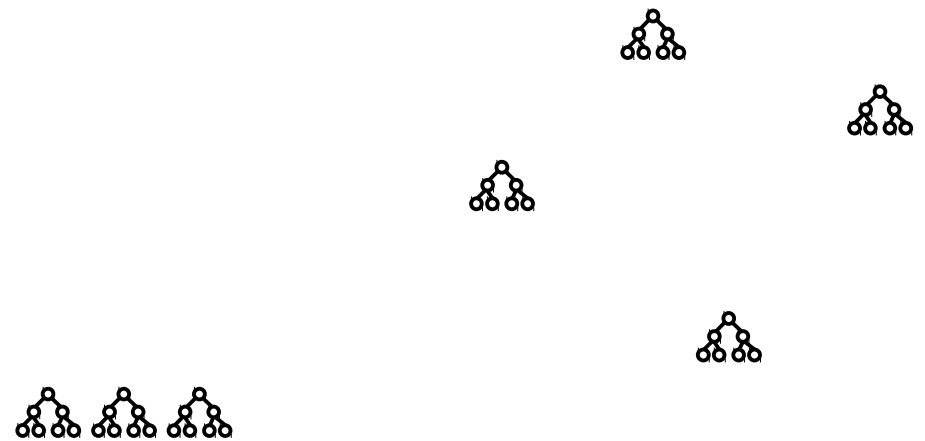
Dynamic Programming – 4

- Etc...

$$Q_1^{\tau=3}$$



$$Q_2^{\tau=3}$$



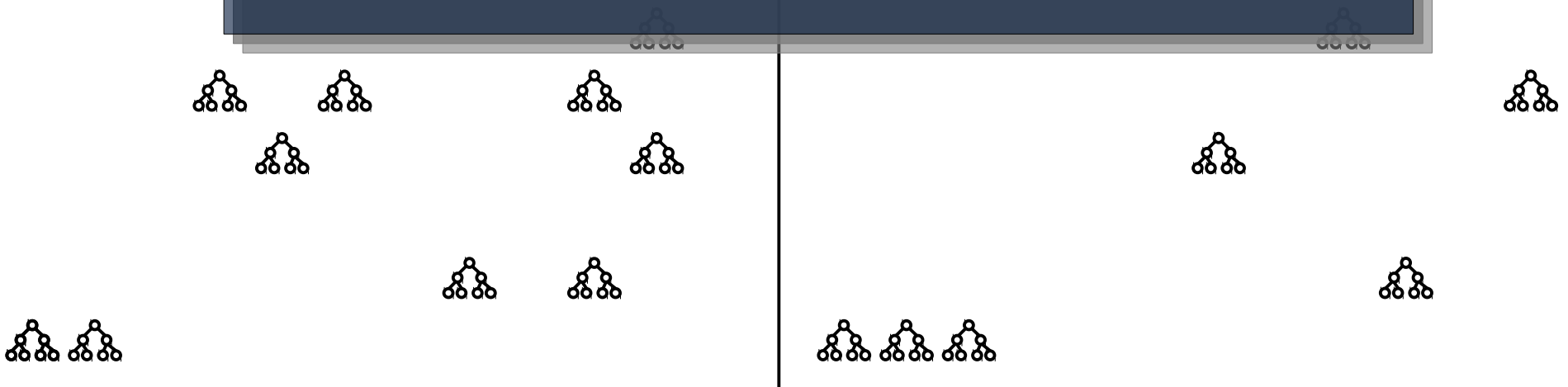
Dynamic Programming – 4

- Etc...

At the very end:

$Q_2^{\tau=3}$...?

$Q_2^{\tau=3}$



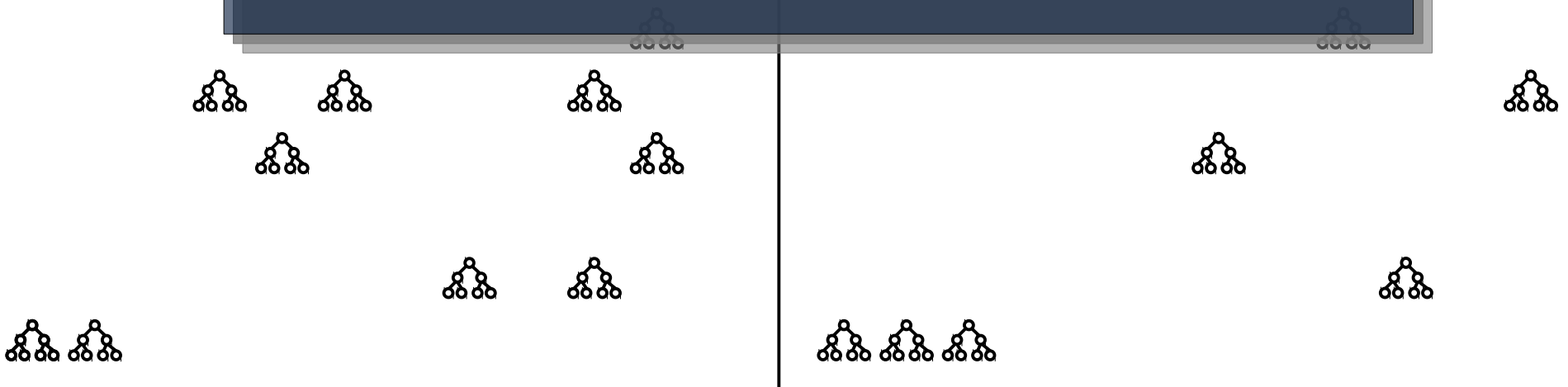
Dynamic Programming – 4

- Etc...

At the very end:

- evaluate all the remaining combinations of policies
- select the best one

$$V(q^{\tau=h}) = \sum_s b^0(s) V(s, q^{\tau=h})$$



Incremental Policy Generation – 1

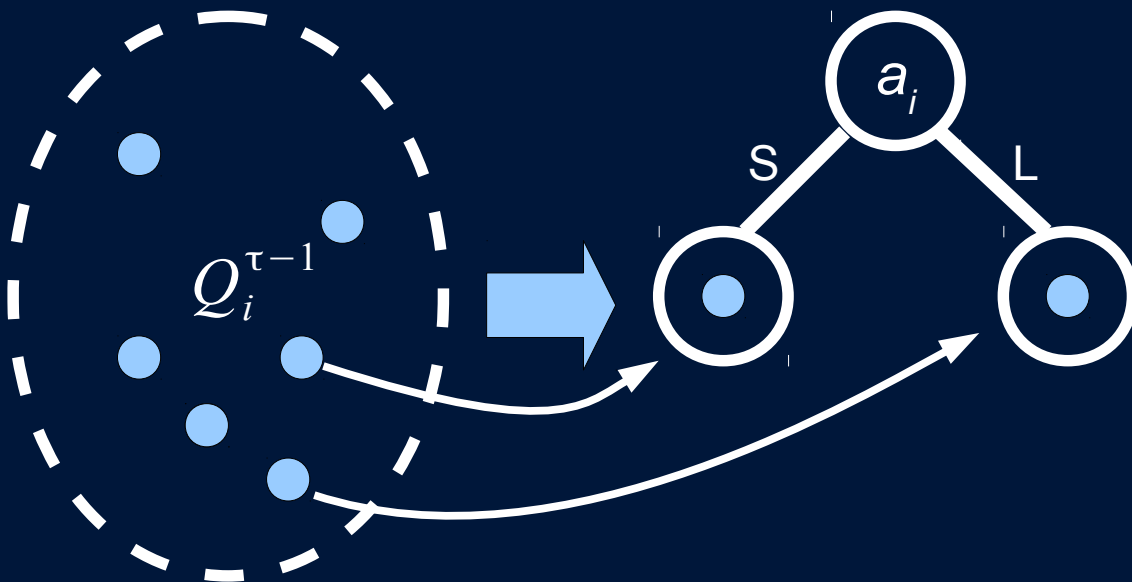
- Bottleneck: exhaustive backup

$$Q_i^\tau = \cup_a Q_i^{\tau,a}$$

$$Q_i^{\tau,a} = (+)_o Q_i^{\tau,a,o}$$

$$Q_i^{\tau,a,o} = \text{BackProject}(Q_i^{\tau-1})$$

Exhaustive backup operation



Incremental Policy Generation – 1

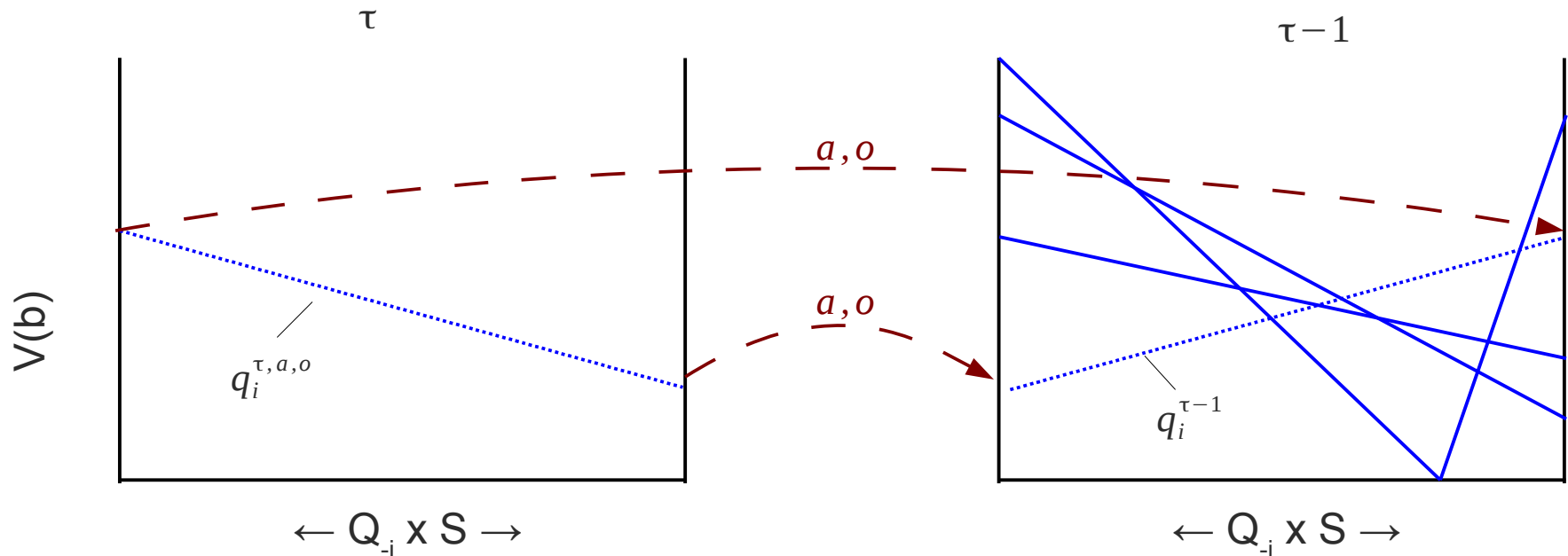
- Bottleneck: exhaustive backup

$$Q_i^\tau = \cup_a Q_i^{\tau,a}$$

$$Q_i^{\tau,a} = (+)_o Q_i^{\tau,a,o}$$

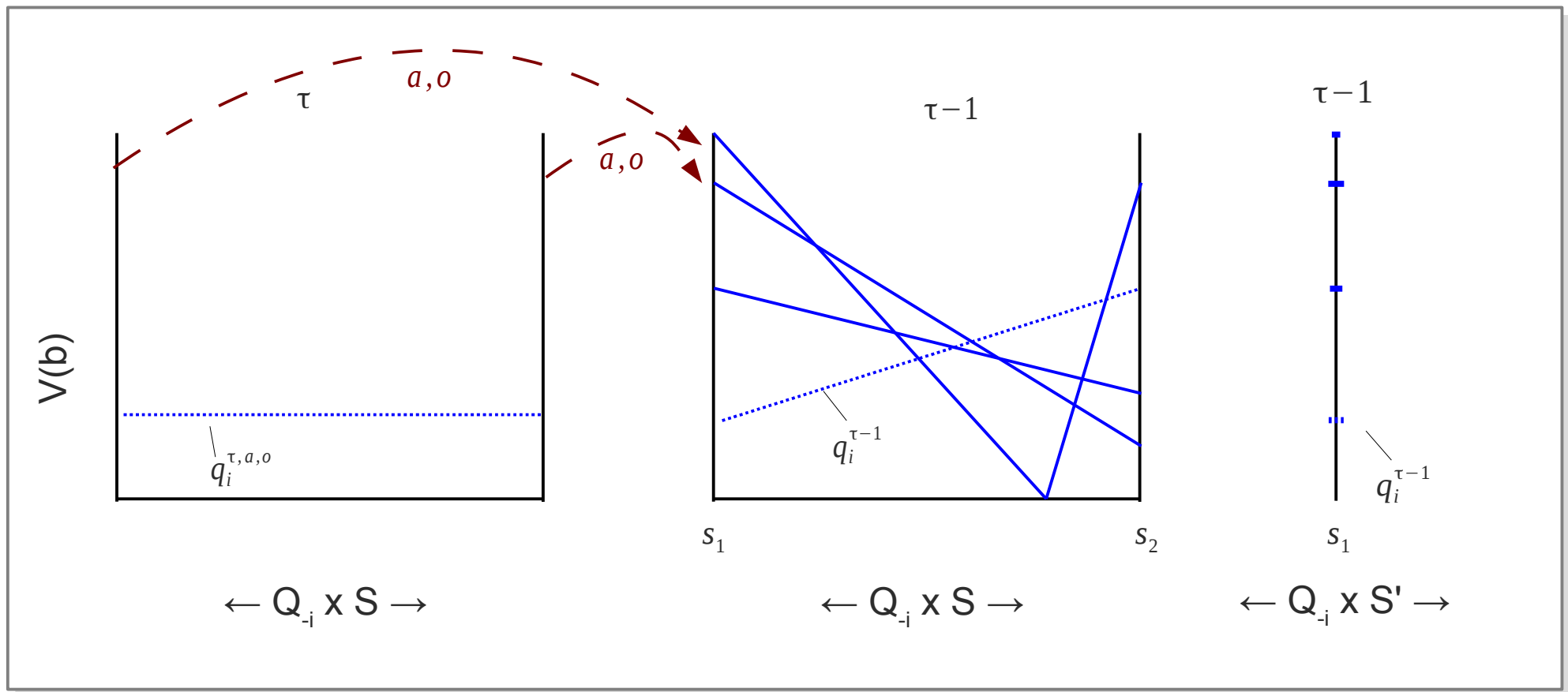
$$Q_i^{\tau,a,o} = \text{BackProject}(Q_i^{\tau-1})$$

Exhaustive backup operation



Incremental Policy Generation – 2

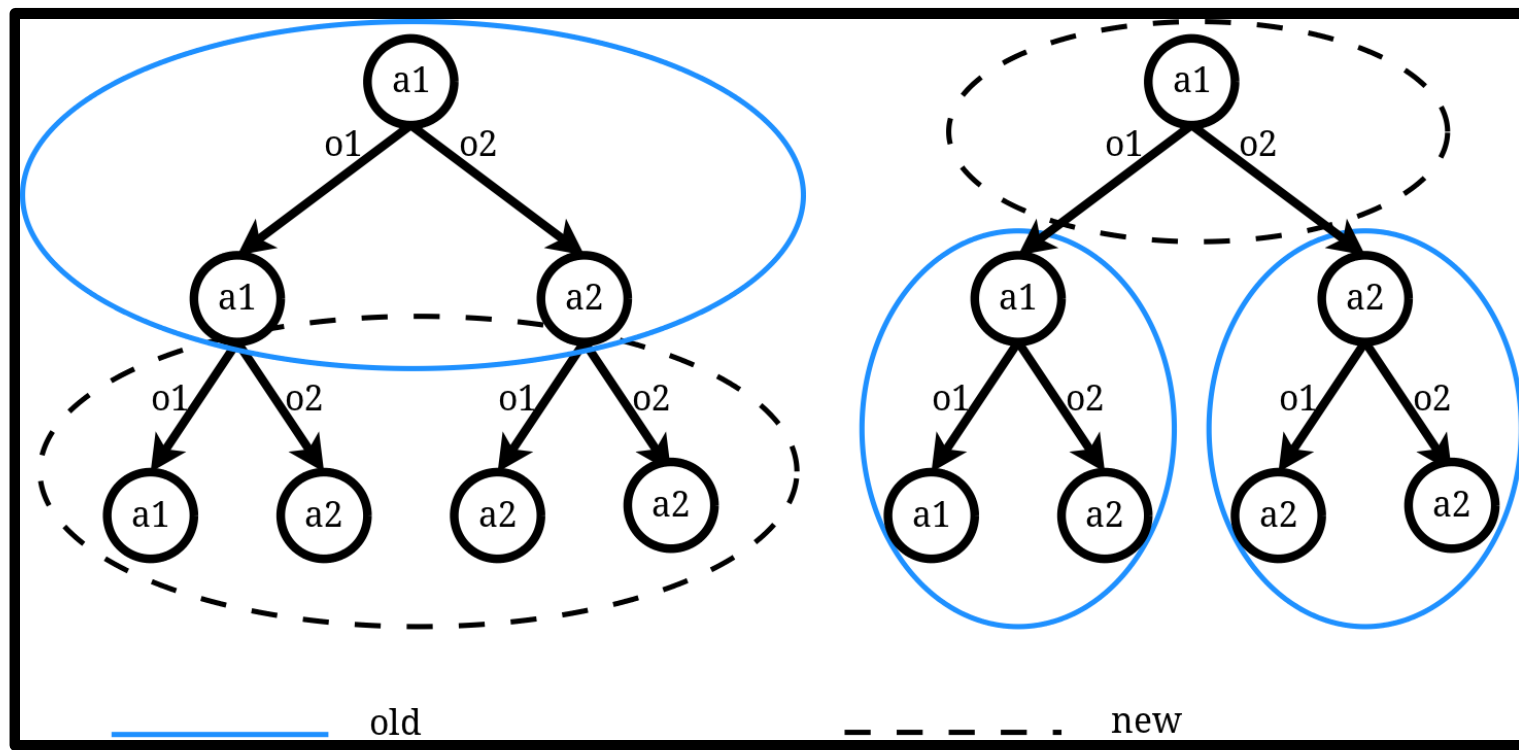
- IPG [Amato et al 2009]
 - some states may be unreachable (for specific a,o)
 → prune only over reachable sub-space



Heuristic Search

Bottom-up vs. Top-down

- DP constructs bottom-up
- Alternatively try and construct top down
→ heuristic search [Szer et al. 2005, Oliehoek et al. 2008]

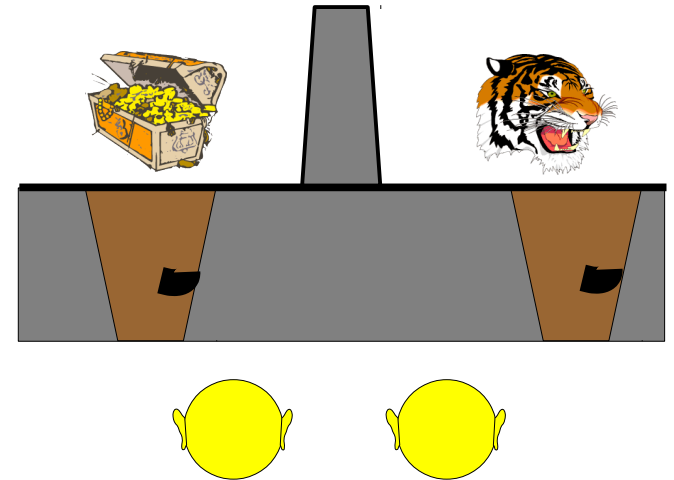


Heuristic Search – Intro

- Core idea is the same as DP:
 - incrementally construct all (joint) policies
 - try to avoid work
- Differences
 - different starting point and increments
 - use **heuristics** (rather than pruning) to avoid work

The Decentralized Tiger Problem

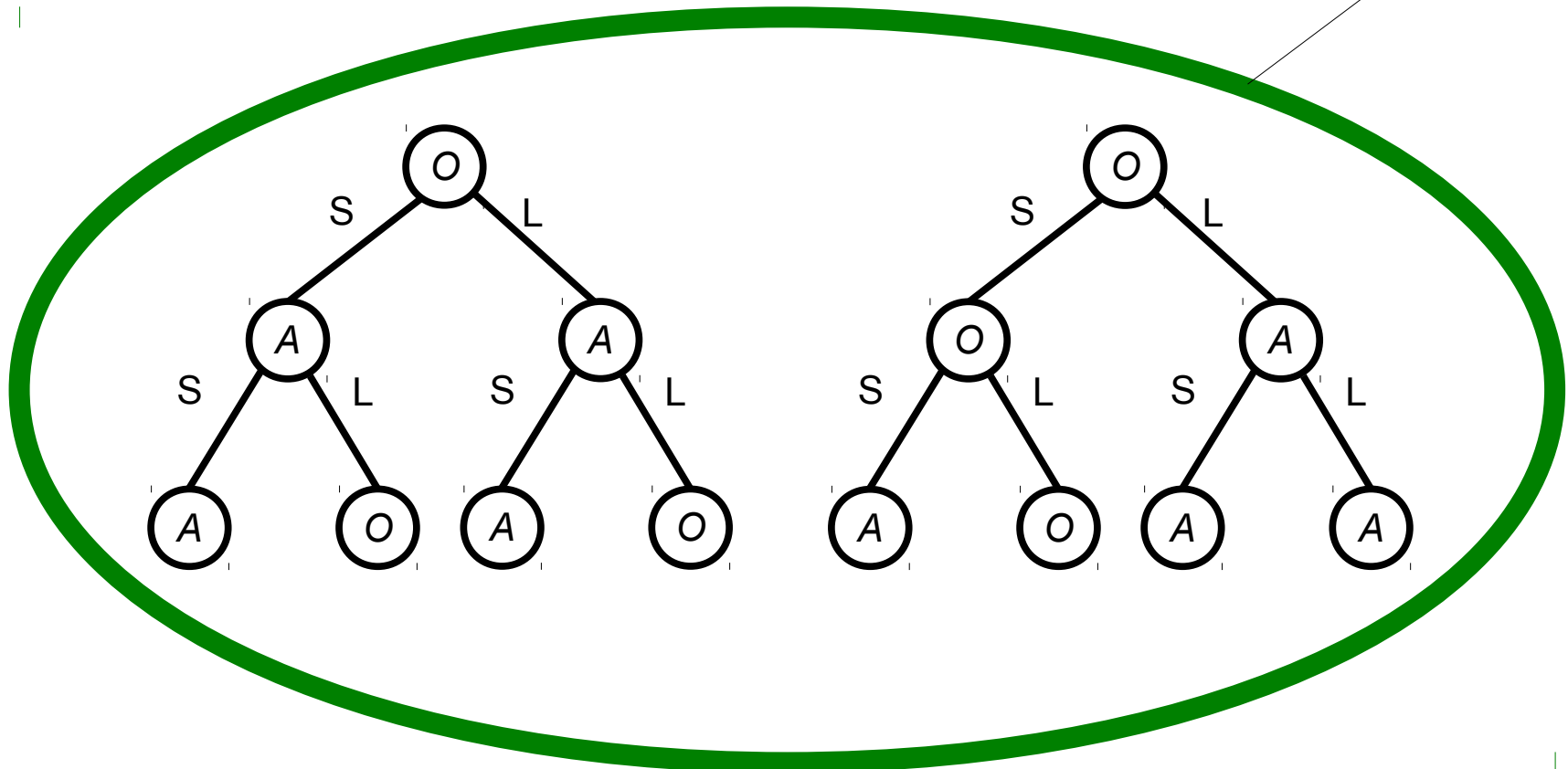
- Two agents in a hallway
- States: tiger left (s_L) or right (s_R)
- Actions: listen, open left, open right.
- Observations: hear left, hear right
 - $\langle \text{Listen}, \text{Listen} \rangle$
 - 85% prob. of getting right obs.
 - e.g. $P(\langle \text{HL}, \text{HL} \rangle \mid \langle \text{Li}, \text{Li} \rangle, S_L) = 0.7225$
 - otherwise: uniform random
- Reward: get the reward, acting jointly is better



Heuristic Search – 1

- Incrementally construct all (joint) policies
 - 'forward in time'

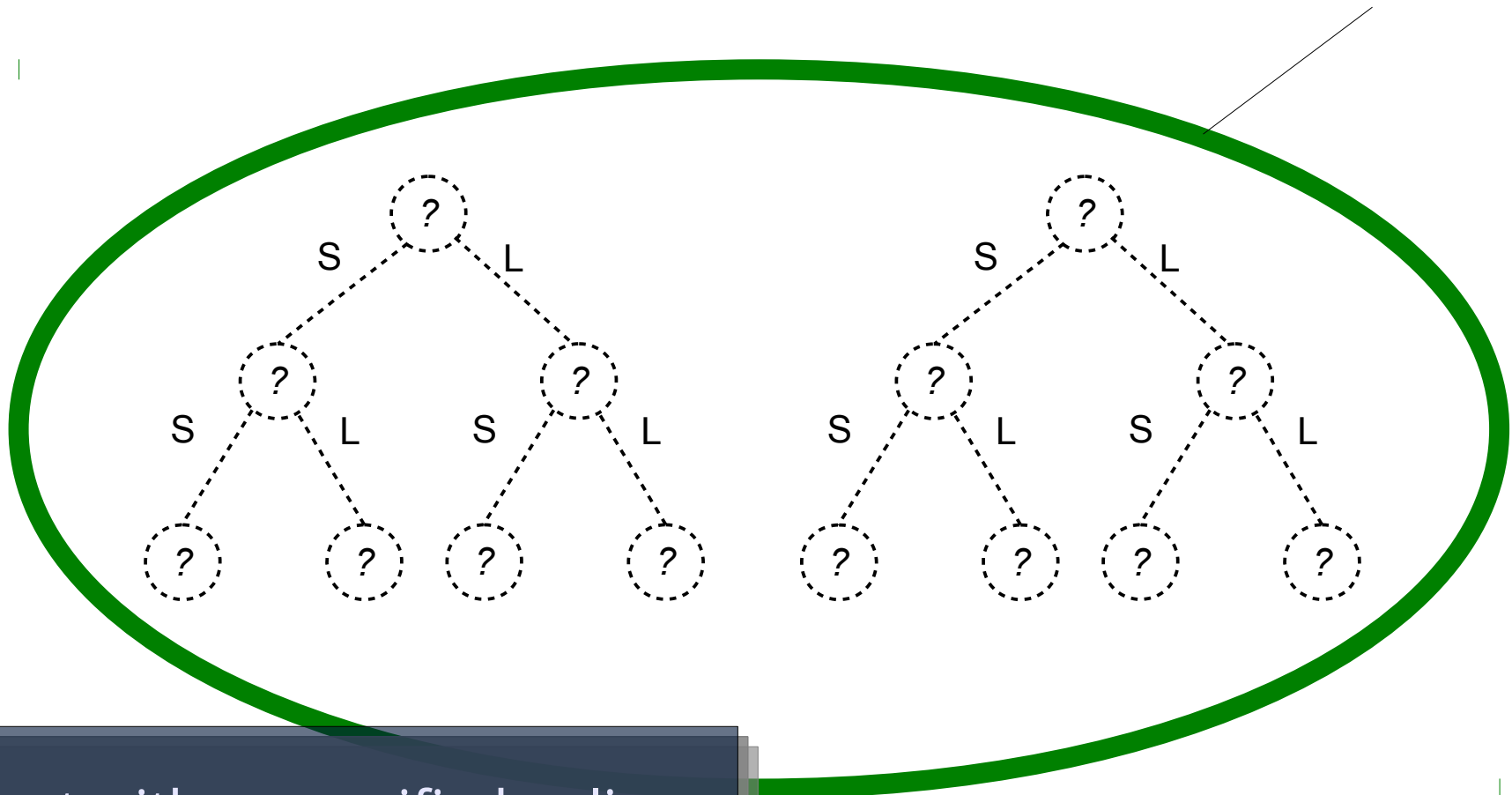
1 joint policy



Heuristic Search – 1

- Incrementally construct all (joint) policies
 - 'forward in time'

1 **partial** joint policy

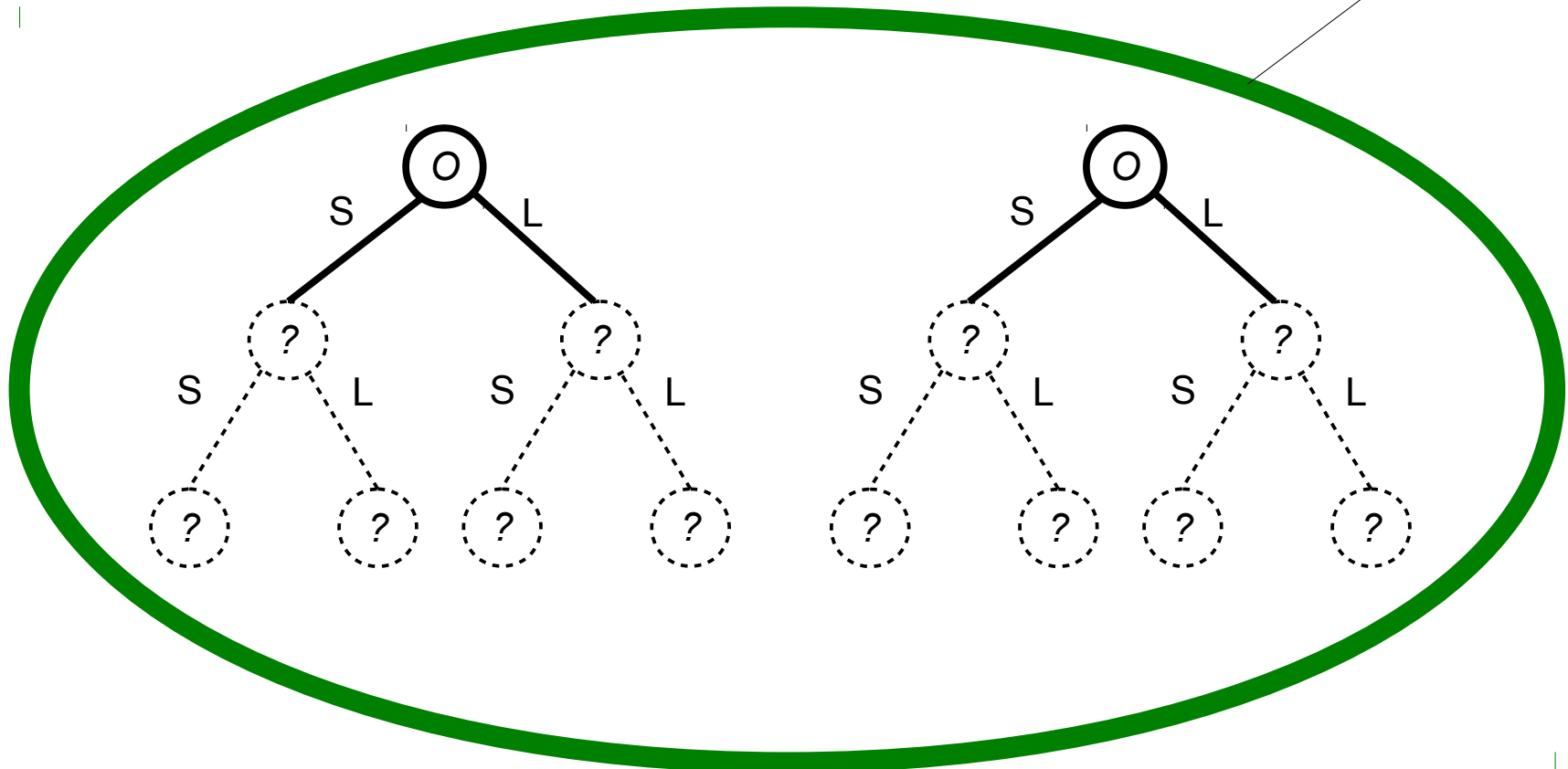


Start with unspecified policy

Heuristic Search – 1

- Incrementally construct all (joint) policies
 - 'forward in time'

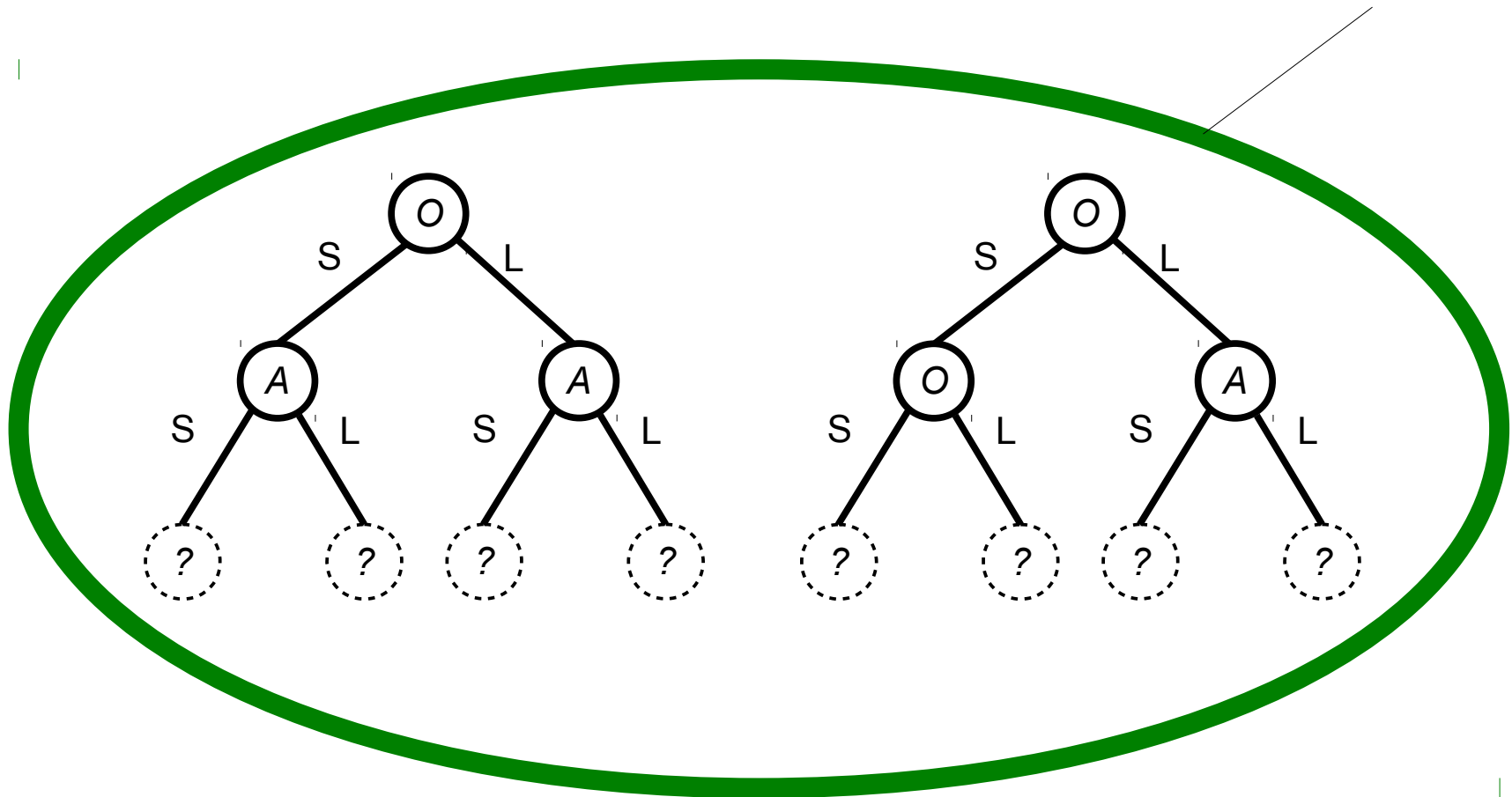
1 **partial** joint policy



Heuristic Search – 1

- Incrementally construct all (joint) policies
 - 'forward in time'

1 **partial** joint policy

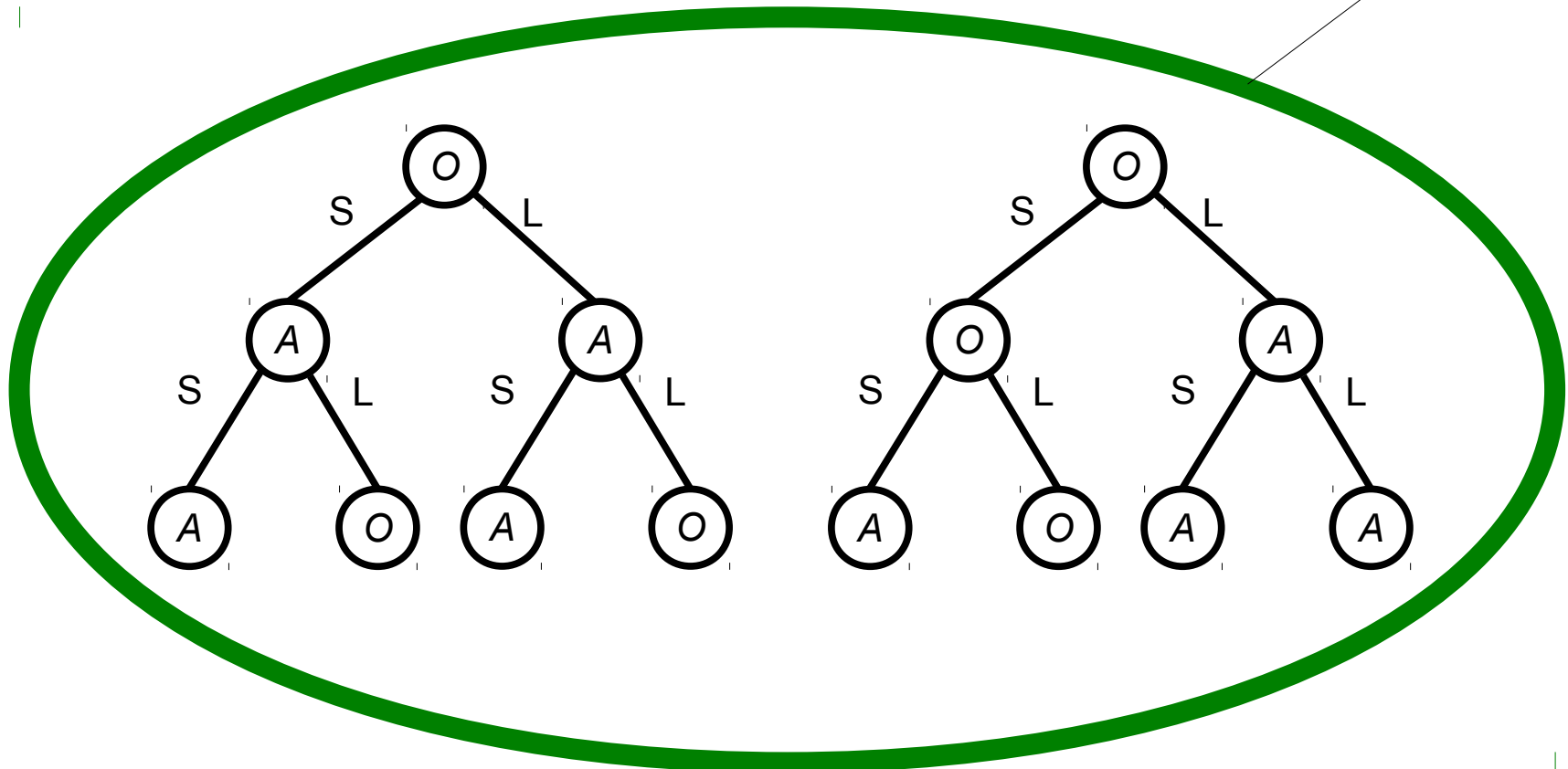


Heuristic Search – 1

- Incrementally construct all (joint) policies

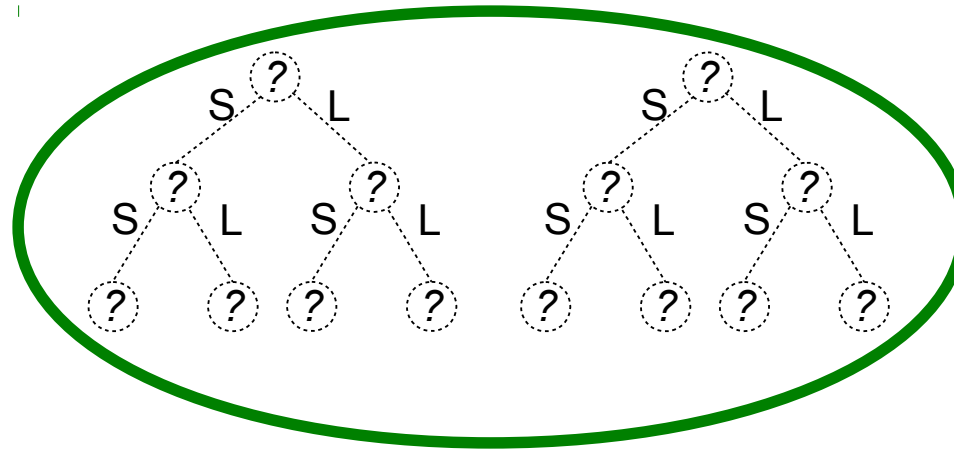
- 'forward in time'

1 **complete** joint policy
(full-length)



Heuristic Search – 2

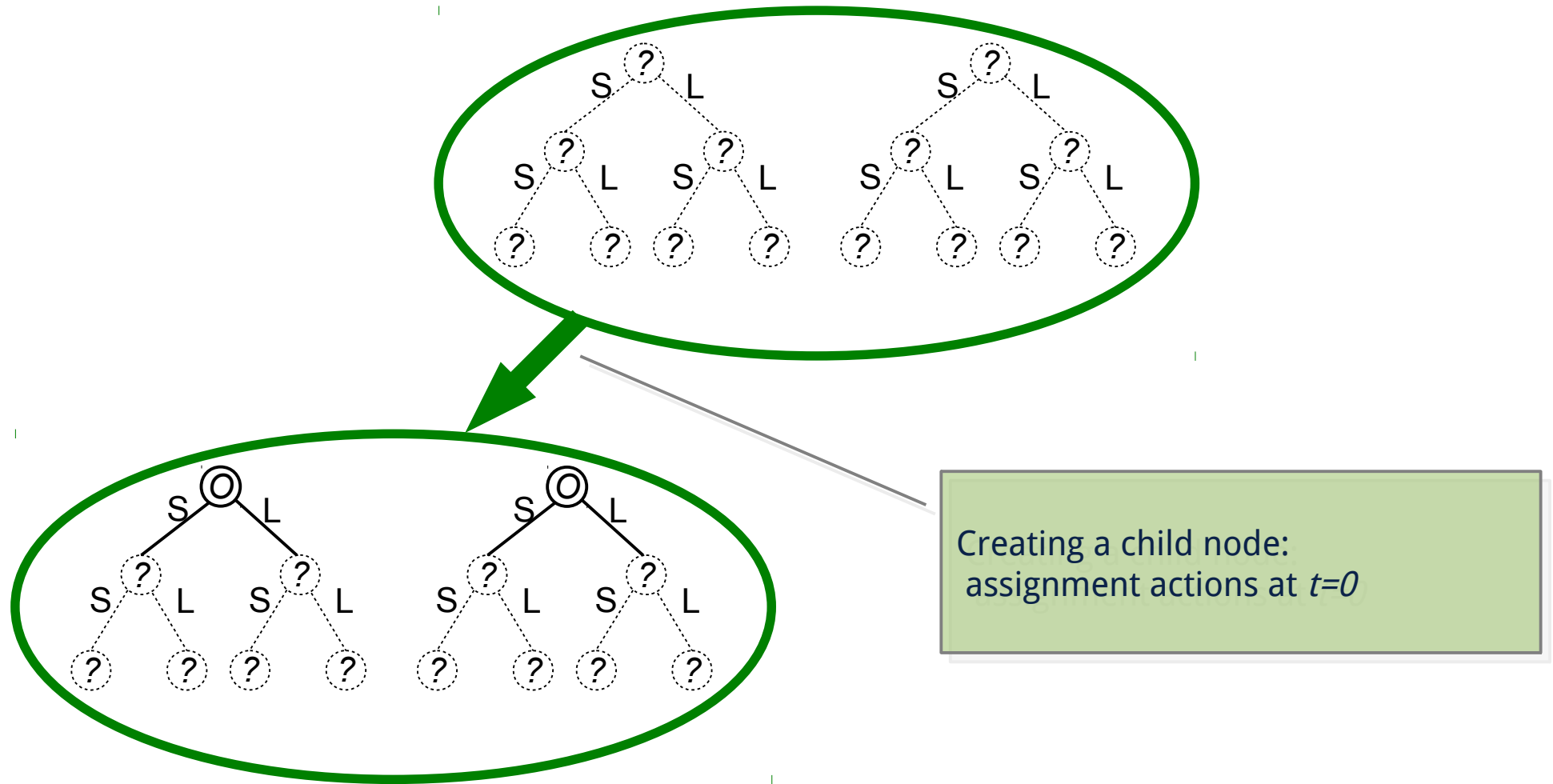
- Creating **ALL** joint policies → tree structure!



Root node:
unspecified joint policy

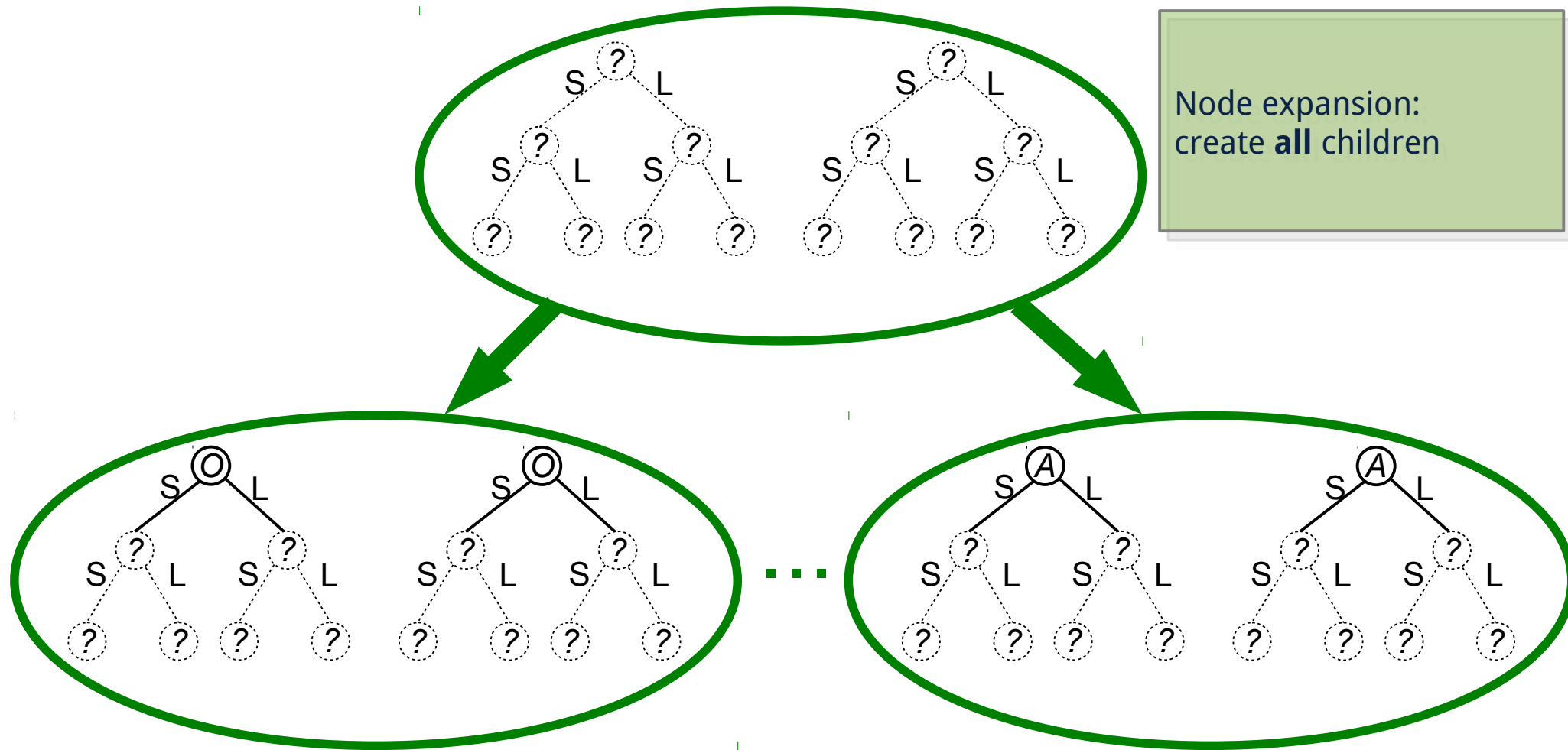
Heuristic Search – 2

- Creating **ALL** joint policies \rightarrow tree structure!



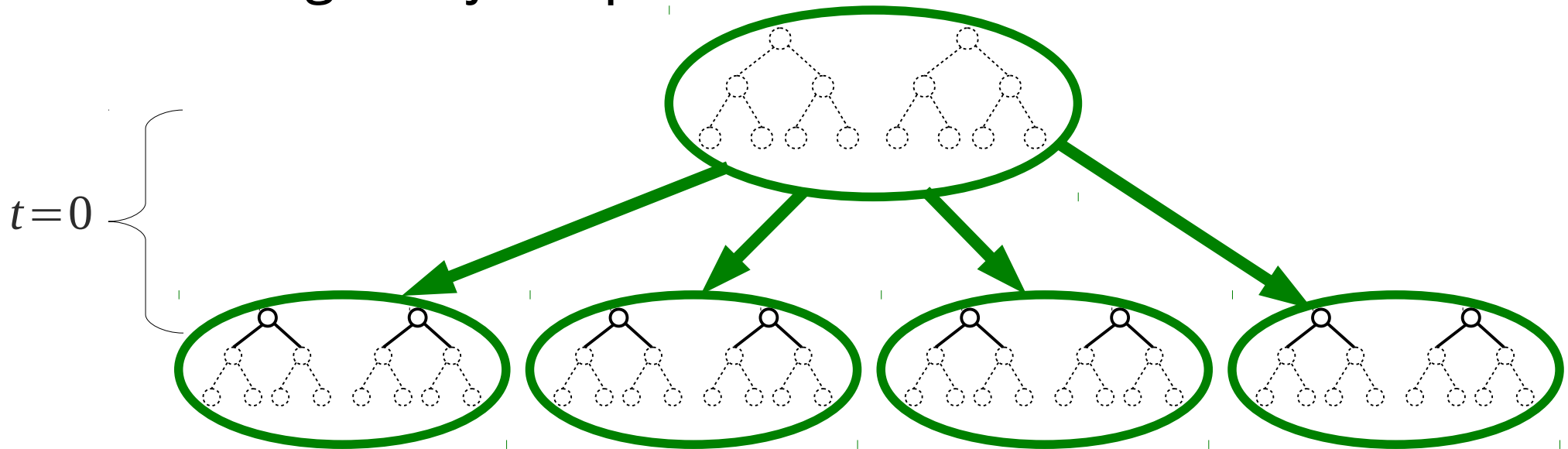
Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!



Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!

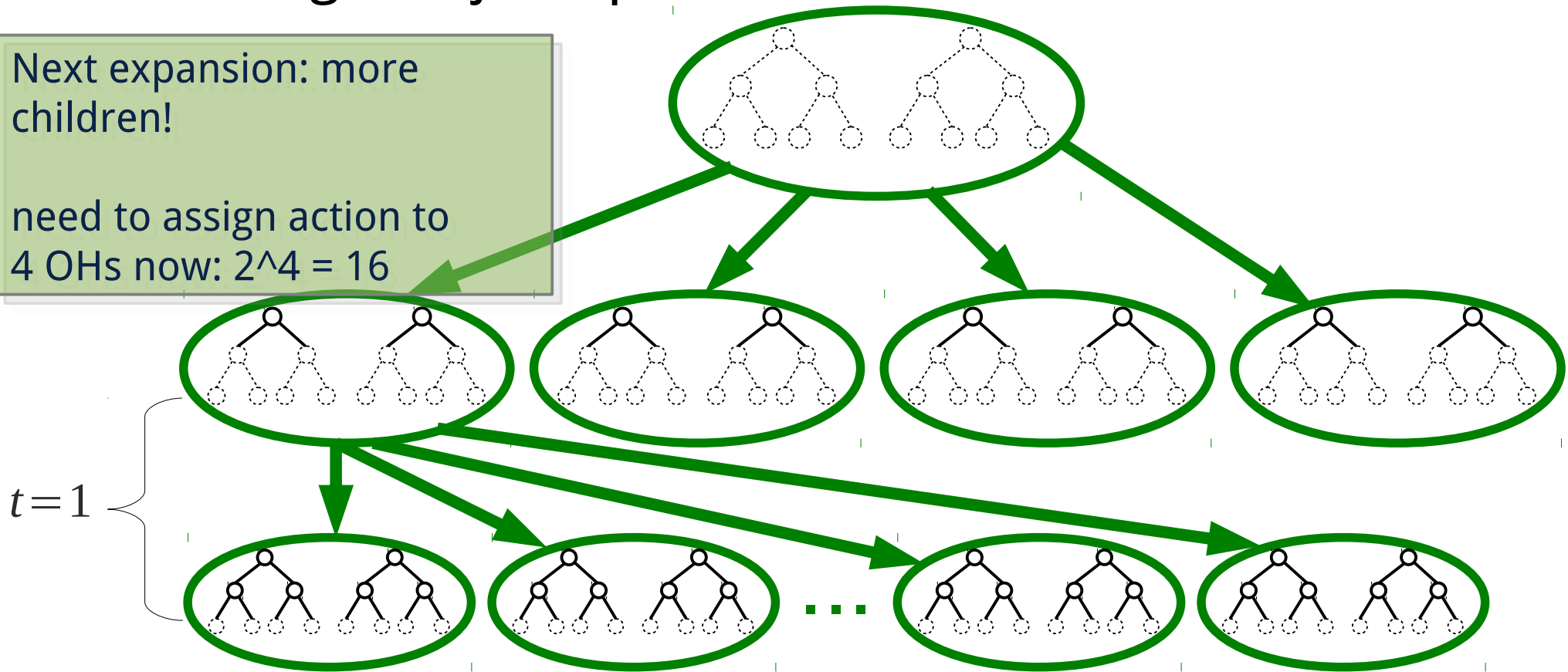


Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!

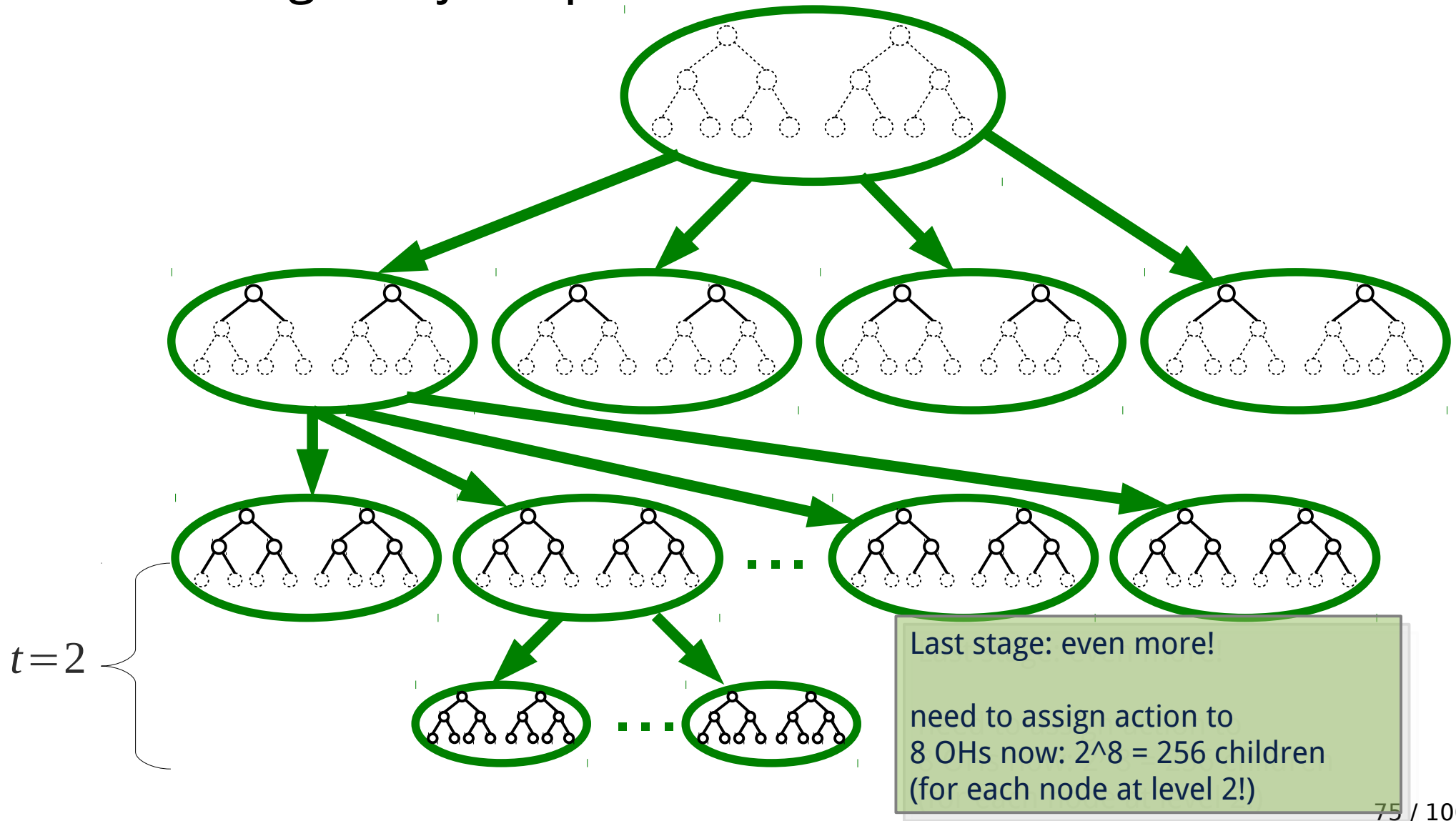
Next expansion: more children!

need to assign action to 4 OHs now: $2^4 = 16$



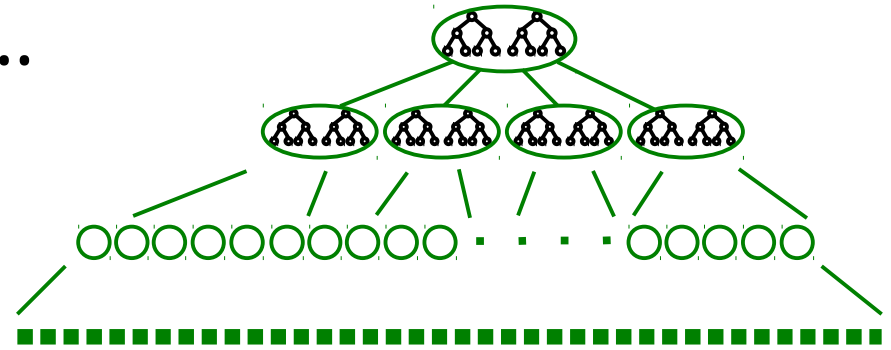
Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!



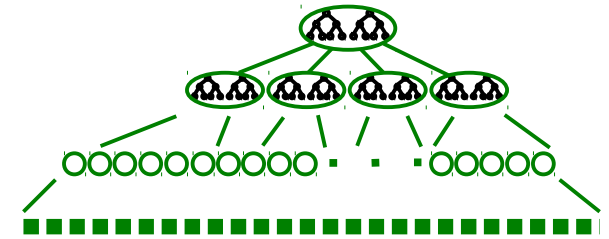
Heuristic Search – 3

- too big to create completely...
- Idea: use **heuristics**
 - avoid going down non-promising branches!
- Apply A^* → **Multiagent A^*** [Szer et al. 2005]



Heuristic Search – 4

- Use heuristics $F(n) = G(n) + H(n)$
- $G(n)$ – actual reward of reaching n
 - a node at depth t specifies φ^t (i.e., actions for first t stages)
→ can compute $V(\varphi^t)$ over stages $0 \dots t-1$
- $H(n)$ – should overestimate!
 - E.g., pretend that it is an MDP
 - compute



$$H(n) = H(\varphi^t) = \sum_s P(s|\varphi^t, b^0) \hat{V}_{MDP}(s)$$

Heuristics – 1

- QPOMDP: Solve 'underlying POMDP'
 - corresponds to immediate communication

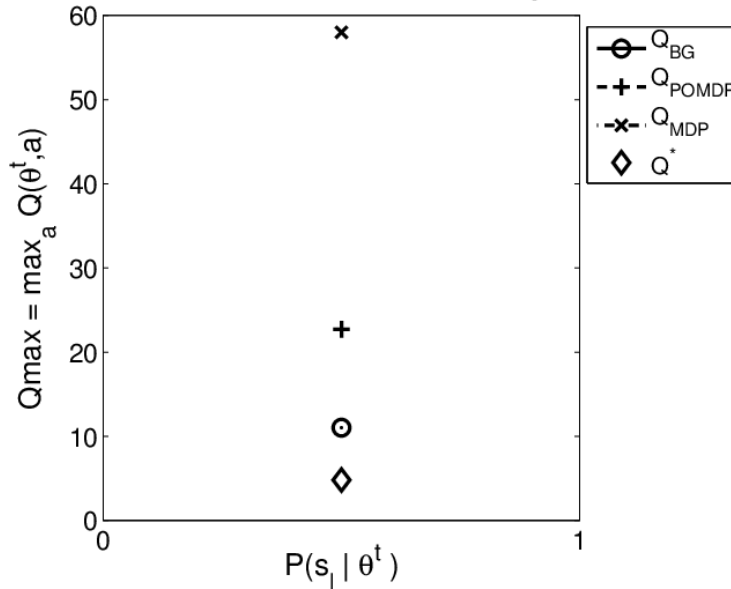
$$H(\varphi^t) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | \varphi^t, b^0) \hat{V}_{POMDP}(b^{\vec{\theta}^t})$$

- QBG corresponds to 1-step delayed communication
- Hierarchy of upper bounds [Oliehoek et al. 2008]

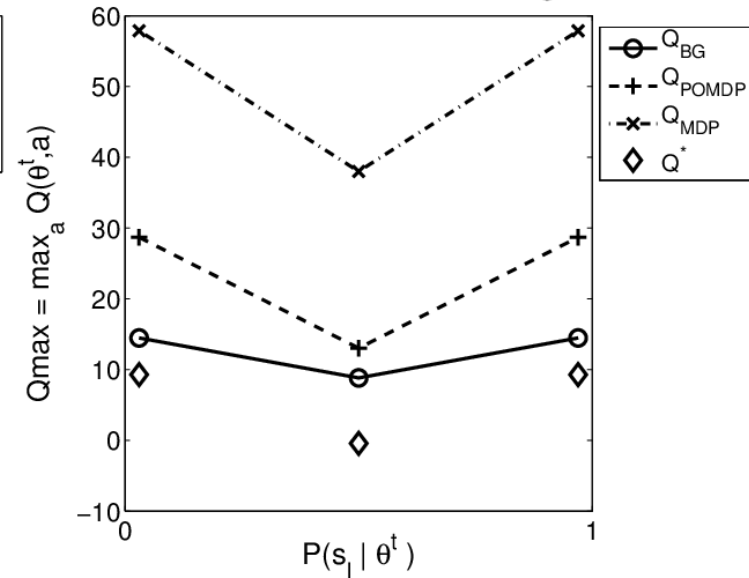
$$Q^* \leq \hat{Q}_{kBG} \leq \hat{Q}_{BG} \leq \hat{Q}_{POMDP} \leq \hat{Q}_{MDP}$$

Heuristics – 2

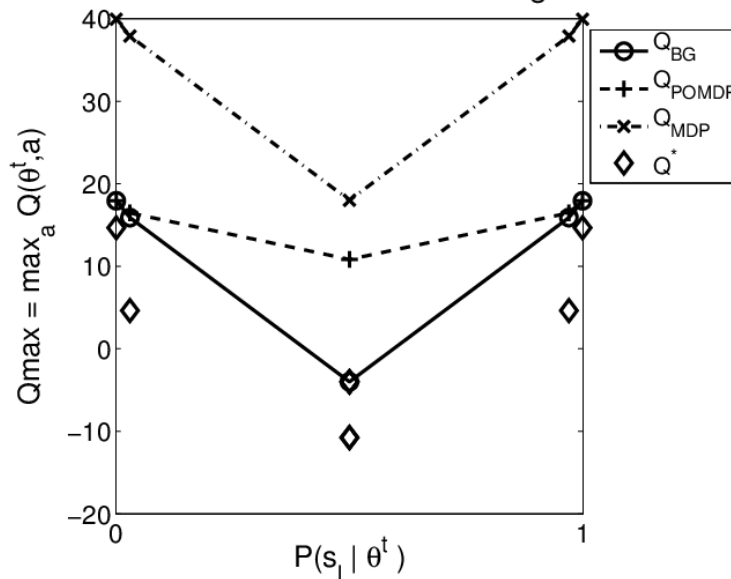
Q-heuristics for horizon=4 Dec-Tiger at t=0



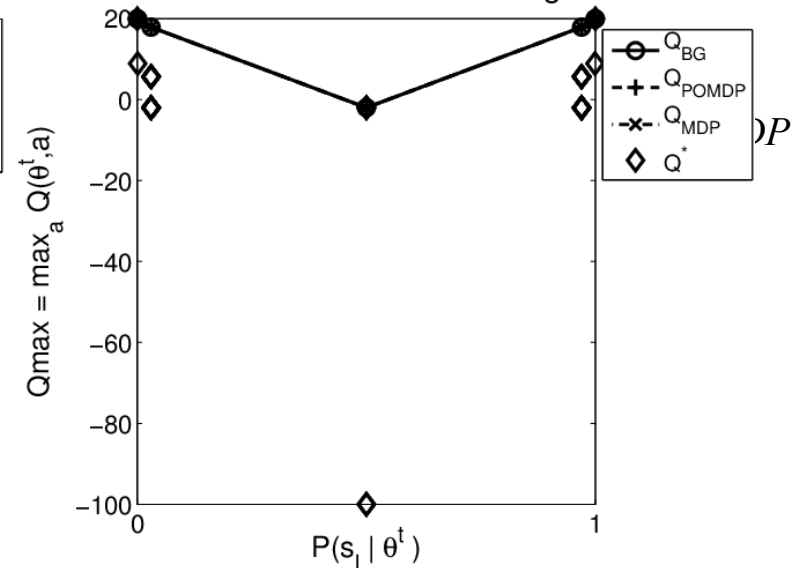
Q-heuristics for horizon=4 Dec-Tiger at t=1



Q-heuristics for horizon=4 Dec-Tiger at t=2



Q-heuristics for horizon=4 Dec-Tiger at t=3



MAA* Limitations

- Number of children grows doubly exponential with nodes depth
- For a node last stage, number of children is

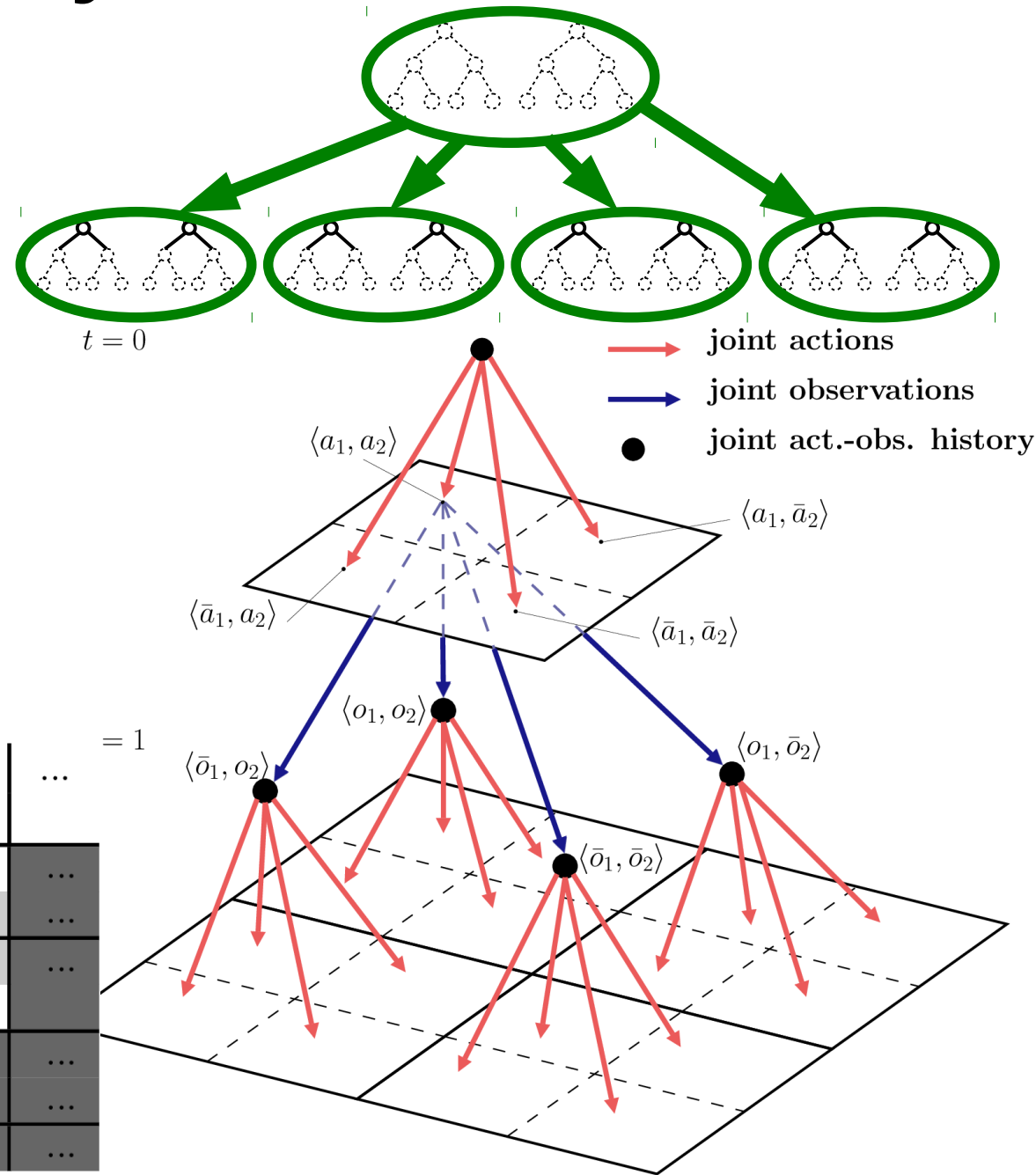
$$O(|A_*|^{n|O_*|^{h-1}})$$

- Total number of joint policies $O(|A_*|^{(n|O_*|^h - 1)/(|O_*| - 1)})$

→ MAA* can only solve 1 horizon longer than brute force search... [Seuken & Zilberstein '08]

MAA* via Bayesian Games

- Each node \leftrightarrow a φ^t
- decision problem for stage t

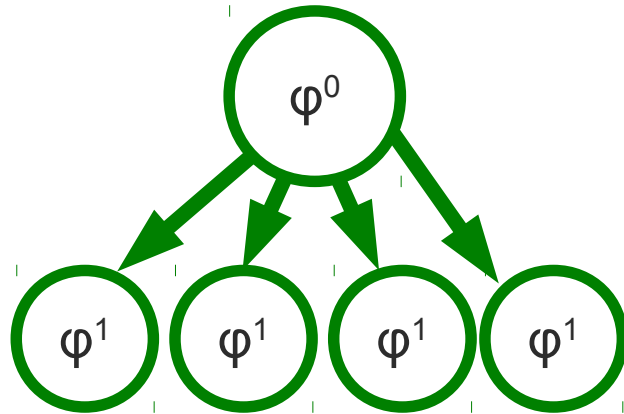


$\vec{\theta}_1^{t=0}$		$\vec{\theta}_2^{t=0}$		$()$	
				a_2	\bar{a}_2
$()$	a_1	+2.75	-4.1		
	\bar{a}_1	-0.9	+0.3		

$\vec{\theta}_1^{t=1}$		$\vec{\theta}_2^{t=1}$		(a_2, o_2)		(a_2, \bar{o}_2)		\dots	
				a_2	\bar{a}_2	a_2	\bar{a}_2		
(a_1, o_1)	a_1	-0.3	+0.6	-0.6	+4.0		
	\bar{a}_1	-0.6	+2.0	-1.3	+3.6		
(a_1, \bar{o}_1)	a_1	+3.1	+4.4	-1.9	+1.0		
	\bar{a}_1	+1.1	-2.9	+2.0	-0.4		
(\bar{a}_1, o_1)	a_1	-0.4	-0.9	-0.5	-1.0		
	\bar{a}_1	-0.9	-4.5	-1.0	+3.5		
(\bar{a}_1, \bar{o}_1)			

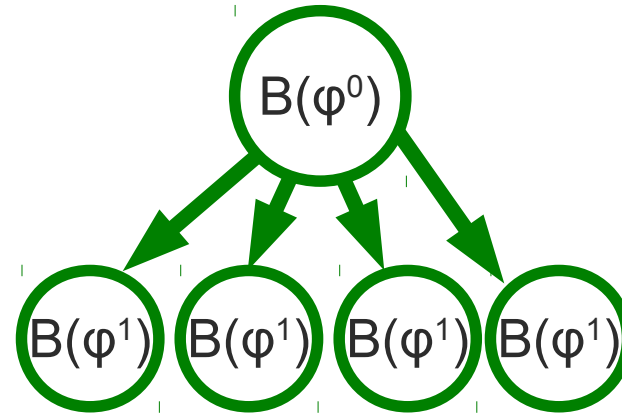
MAA* via Bayesian Games – 2

MAA* perspective



- node $\leftrightarrow \varphi^t$
- joint decision rule δ
maps OHs to actions
- Expansion: appending all next-stage decision rules: $\varphi^{t+1}=(\varphi^t, \delta^t)$

BG perspective



- node \leftrightarrow a BG
- joint BG policy β
maps 'types' to actions
- Expansion: enumeration of all joint BG policies $\varphi^{t+1}=(\varphi^t, \beta^t)$

direct correspondence: $\delta \leftrightarrow \beta$

MAA* via Bayesian Games – 2

MAA* perspective

BG perspective

What is the point?

- ▶ Generalized MAA* [Oliehoek & Vlassis '07]
- ▶ Unified perspective of MAA* and 'BAGA' approximation [Emery-Montemerlo et al. '04]
- ▶ No direct improvements...

- node $\leftrightarrow \varphi^t$ However...
- joint decision maps OHs to actions
 - ▶ BG provide abstraction layer
 - ▶ Facilitated two improvements that lead to state-of-the-art performance [Oliehoek et al. '13]
 - Clustering of histories
 - Incremental expansion
- Expansion: enumerating all joint BG policies
 - Expansion: enumeration of all joint BG policies $\varphi^{t+1} = (\varphi^t, \beta^t)$

Lossless Clustering

- Two types (=action-observation histories) in a BG are **probabilistically equivalent** iff

$$P(\vec{\theta}_{-i}|\vec{\theta}_{i,a}) = P(\vec{\theta}_{-i}|\vec{\theta}_{i,b})$$

$$P(s|\vec{\theta}_{-i}, \vec{\theta}_{i,a}) = P(s|\vec{\theta}_{-i}, \vec{\theta}_{i,b})$$

\vec{o}_1^2	\vec{o}_2^2			
	(o_{HL}, o_{HL})	(o_{HL}, o_{HR})	(o_{HR}, o_{HL})	(o_{HR}, o_{HR})
(o_{HL}, o_{HL})	0.261	0.047	0.047	0.016
(o_{HL}, o_{HR})	0.047	0.016	0.016	0.047
(o_{HR}, o_{HL})	0.047	0.016	0.016	0.047
(o_{HR}, o_{HR})	0.016	0.047	0.047	0.261

(a) The joint type probabilities.

\vec{o}_1^2	\vec{o}_2^2			
	(o_{HL}, o_{HL})	(o_{HL}, o_{HR})	(o_{HR}, o_{HL})	(o_{HR}, o_{HR})
(o_{HL}, o_{HL})	0.999	0.970	0.970	0.5
(o_{HL}, o_{HR})	0.970	0.5	0.5	0.030
(o_{HR}, o_{HL})	0.970	0.5	0.5	0.030
(o_{HR}, o_{HR})	0.5	0.030	0.030	0.001

(b) The induced joint beliefs. Listed is the probability $\Pr(s_l|\vec{\theta}^2, \mathbf{b}^0)$ of the tiger being behind the left door.

Lossless Clustering

- Two types (=action-observation histories) in a BG are **probabilistically equivalent** iff

$$P(\vec{\theta}_{-i}|\vec{\theta}_{i,a}) = P(\vec{\theta}_{-i}|\vec{\theta}_{i,b})$$

$$P(s|\vec{\theta}_{-i}, \vec{\theta}_{i,a}) = P(s|\vec{\theta}_{-i}, \vec{\theta}_{i,b})$$

	\vec{o}_1^2	\vec{o}_2^2			
		(o_{HL}, o_{HL})	(o_{HL}, o_{HR})	(o_{HR}, o_{HL})	(o_{HR}, o_{HR})
(o_{HL}, o_{HL})	0.261	0.047	0.047	0.016	
(o_{HL}, o_{HR})	0.047	0.016	0.016	0.047	
(o_{HR}, o_{HL})	0.047	0.016	0.016	0.047	
(o_{HR}, o_{HR})	0.016	0.047	0.047	0.261	

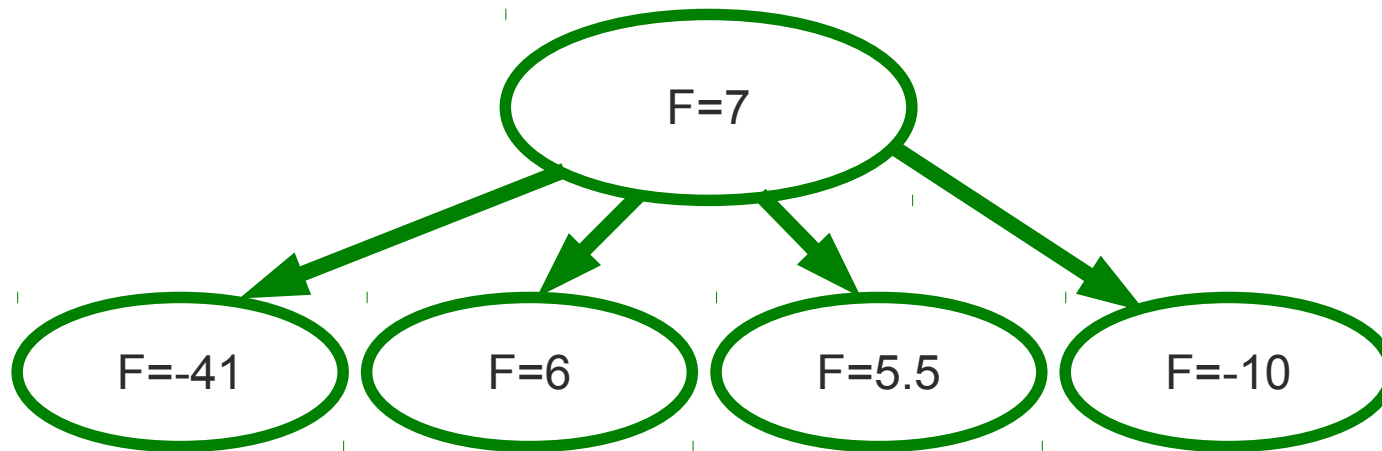
(a) The joint type probabilities.

	\vec{o}_1^2	\vec{o}_2^2			
		(o_{HL}, o_{HL})	(o_{HL}, o_{HR})	(o_{HR}, o_{HL})	(o_{HR}, o_{HR})
(o_{HL}, o_{HL})	0.999	0.970	0.970	0.5	
(o_{HL}, o_{HR})	0.970	0.5	0.5	0.030	
(o_{HR}, o_{HL})	0.970	0.5	0.5	0.030	
(o_{HR}, o_{HR})	0.5	0.030	0.030	0.001	

(b) The induced joint beliefs. Listed is the probability $\Pr(s_t|\vec{\theta}^2, b^0)$ of the tiger being behind the left door.

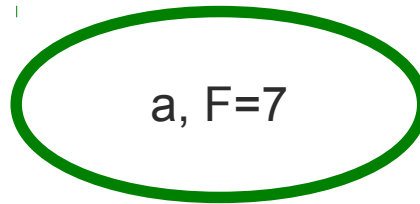
Incremental Expansion

- Key idea: even though nodes can have many children, only few are useful.
 - i.e., only few will be selected for further expansion
 - others will have too low heuristic value



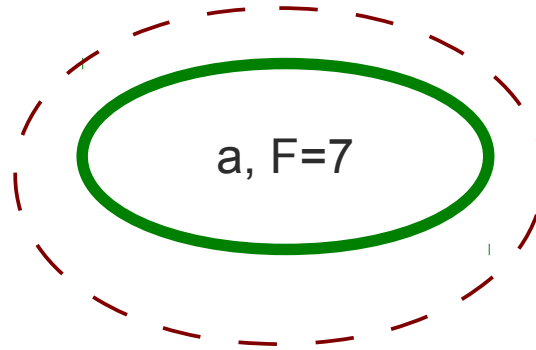
- if we can generate the nodes in increasing heuristic order
→ can avoid expansion of redundant nodes

Incremental Expansion



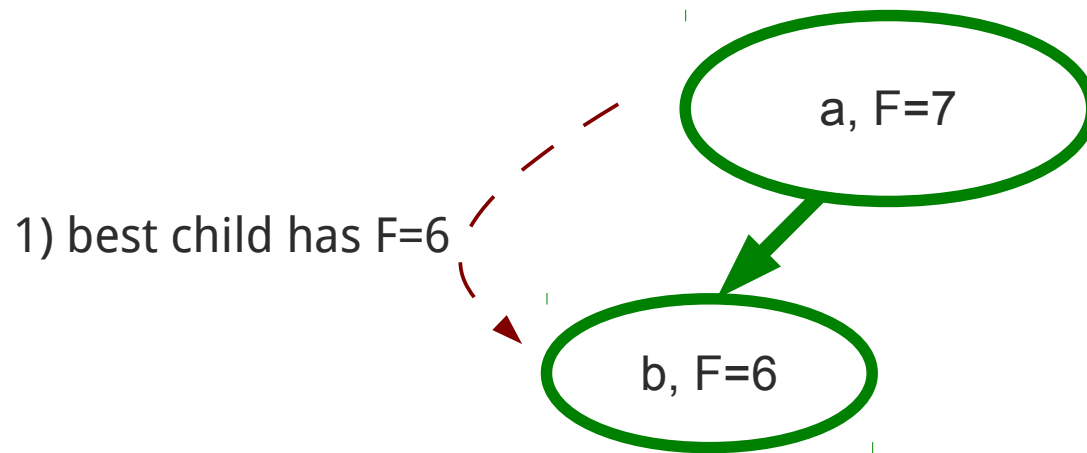
Incremental Expansion

Select for expansion →



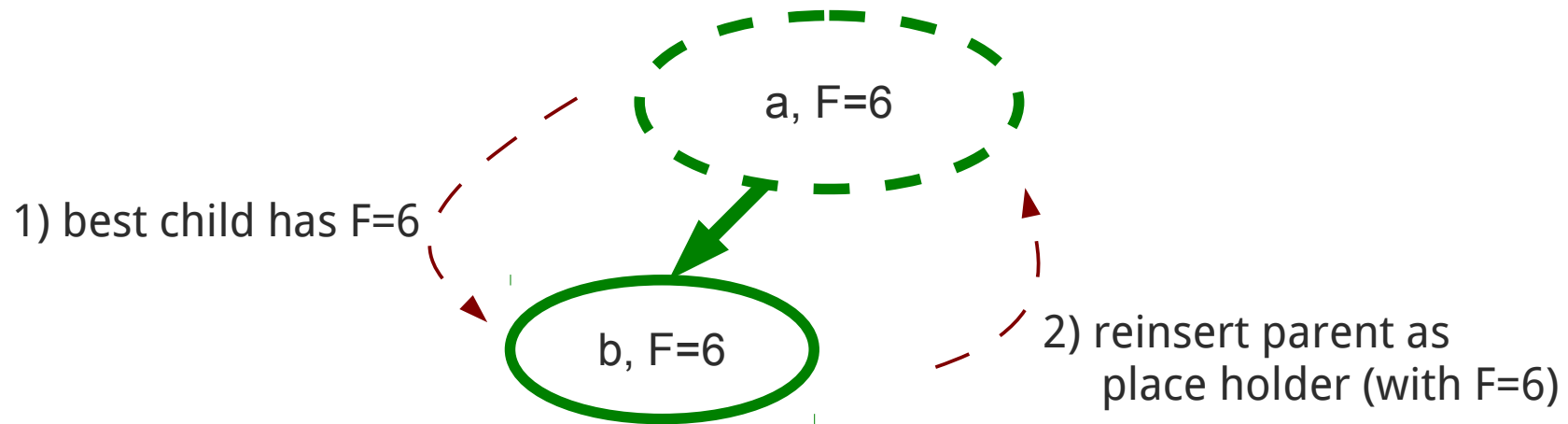
Open list
a - 7

Incremental Expansion



Open list
b - 6

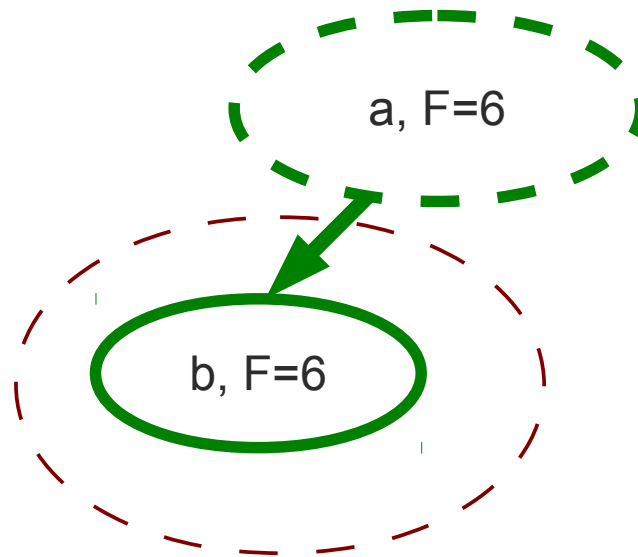
Incremental Expansion



Open list
b - 6
a - 6

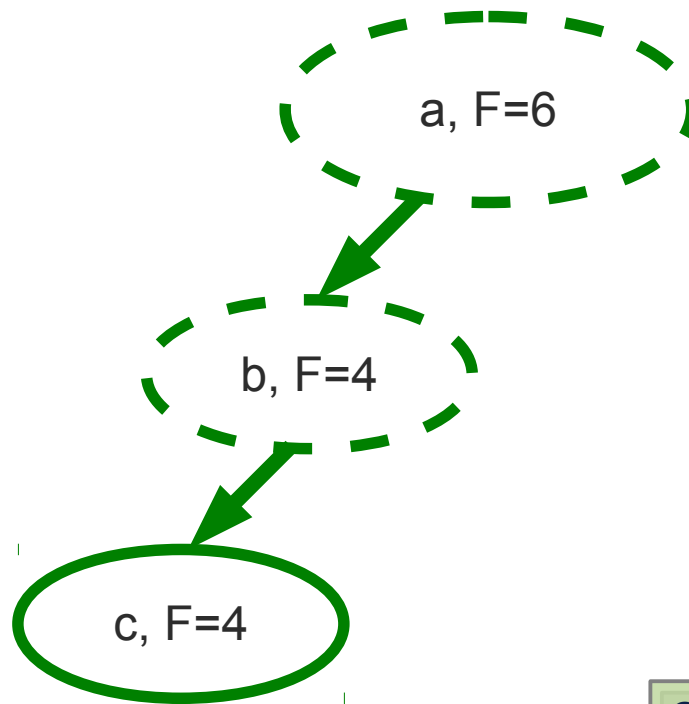
Incremental Expansion

Select for expansion →



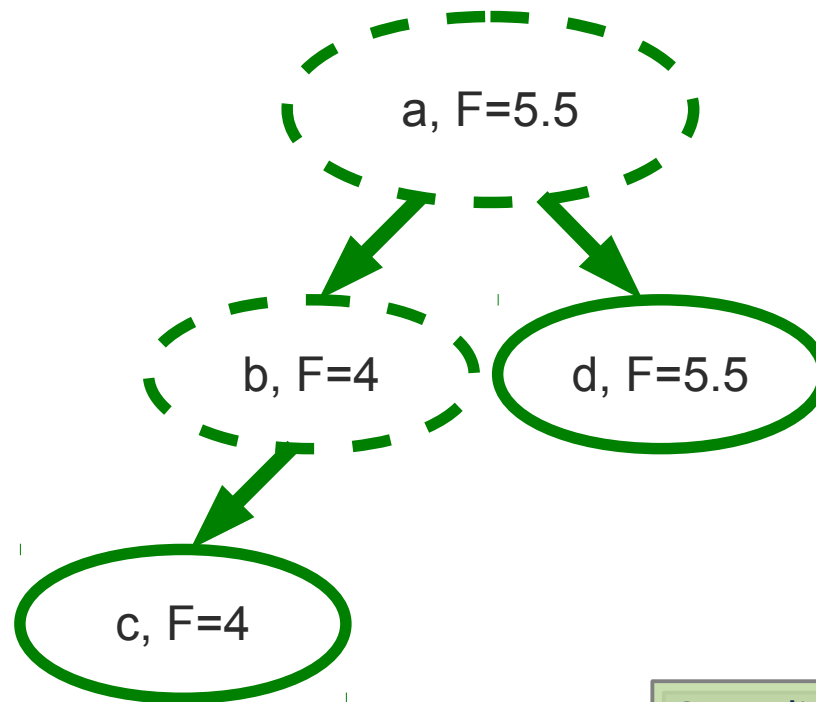
Open list
b - 6
a - 6

Incremental Expansion



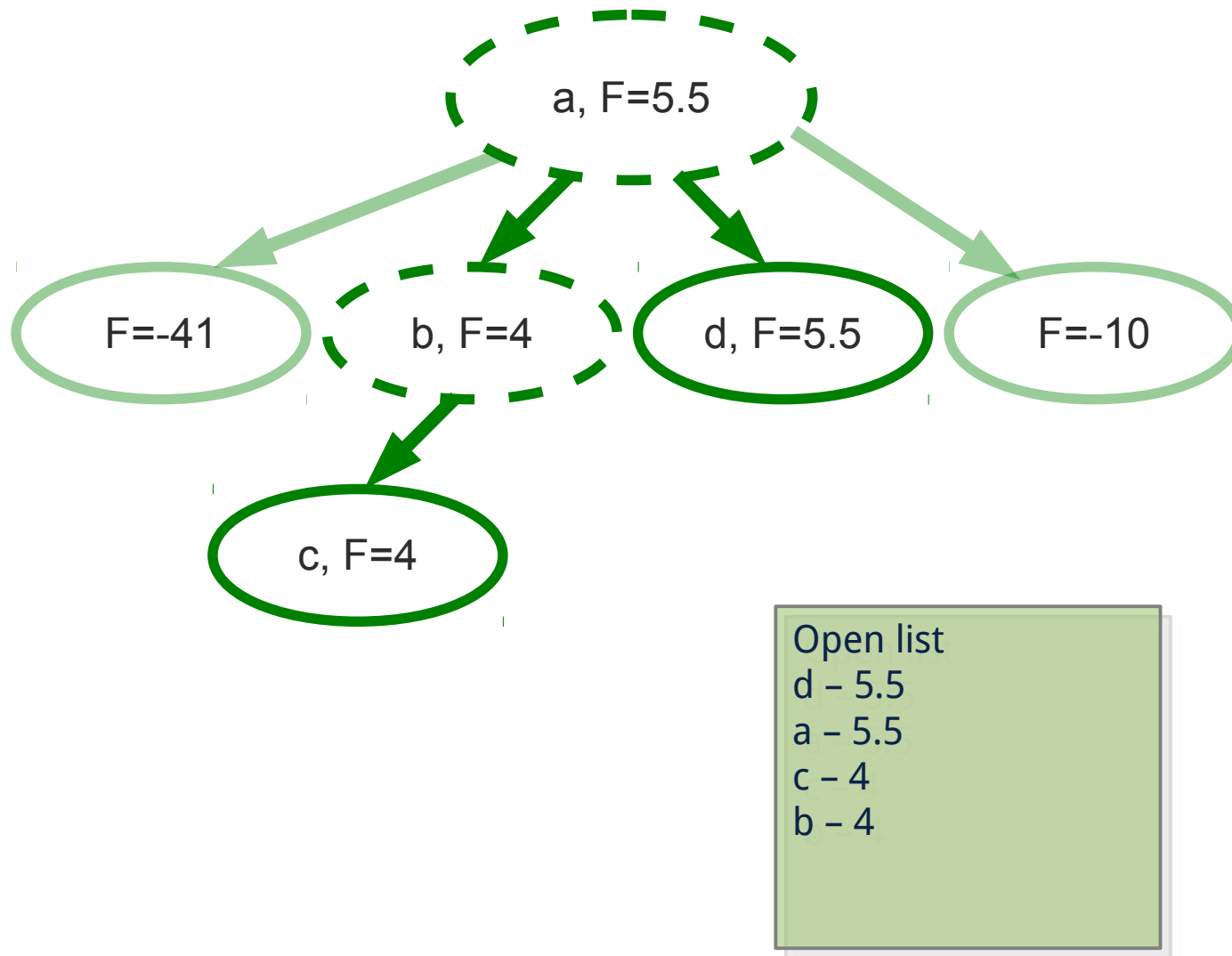
Open list
a - 6
c - 4
b - 4

Incremental Expansion



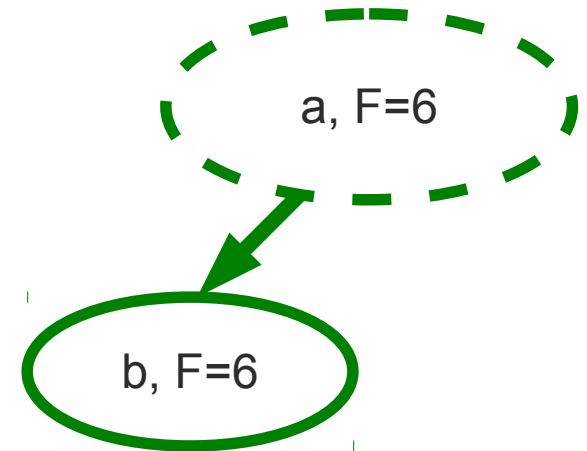
Open list
d - 5.5
a - 5.5
c - 4
b - 4

Incremental Expansion



Incremental Expansion: How?

- How do we generate the next-best child?
- Node \leftrightarrow BG, so...
 - find the solutions of the BG (in decreasing order of value)
 - i.e., 'incremental BG solver'
 - Modification of BaGaBaB [Oliehoek et al. 2010]
 - stop searching when next solution found
 - save search tree for next time visited.



Wrap-up Optimal Solution Methods

Other Optimal Methods

- DP with compression of values (LPC) [Boularias & Chaib-draa 2008]
- MILP [Aras and Dutech 2010]

State of the Art

	problem primitives			
	n	$ \mathcal{S} $	$ \mathcal{A}_i $	$ \mathcal{O}_i $
DEC-TIGER	2	2	3	2
BROADCASTCHANNEL	2	4	2	2
GRIDSMALL	2	16	5	2
COOPERATIVE BOX PUSHING	2	100	4	5
RECYCLING ROBOTS	2	4	3	2
HOTEL 1	2	16	3	4
FIREFIGHTING	2	432	3	2

‘—’ memory limit violations

‘*’ time limit overruns

‘#’ heuristic bottleneck

h	MILP	DP-LPC	DP-IPG	GMAA — Q_{BG}		
				IC	ICE	heur
BROADCASTCHANNEL, ICE solvable to $h = 900$						
2	0.38	≤ 0.01	0.09	≤ 0.01	≤ 0.01	≤ 0.01
3	1.83	0.50	56.66	≤ 0.01	≤ 0.01	≤ 0.01
4	34.06	*	*	≤ 0.01	≤ 0.01	≤ 0.01
5	48.94			≤ 0.01	≤ 0.01	≤ 0.01
DEC-TIGER, ICE solvable to $h = 6$						
2	0.69	0.05	0.32	≤ 0.01	≤ 0.01	≤ 0.01
3	23.99	60.73	55.46	≤ 0.01	≤ 0.01	≤ 0.01
4	*	—	2286.38	0.27	≤ 0.01	0.03
5			—	21.03	0.02	0.09
FIREFIGHTING (2 agents, 3 houses, 3 firelevels), ICE solvable to $h \gg 1000$						
2	4.45	8.13	10.34	≤ 0.01	≤ 0.01	≤ 0.01
3	—	—	569.27	0.11	0.10	0.07
4			—	950.51	1.00	0.65
GRIDSMALL, ICE solvable to $h = 6$						
2	6.64	11.58	0.18	0.01	≤ 0.01	≤ 0.01
3	*	—	4.09	0.10	≤ 0.01	0.42
4			77.44	1.77	≤ 0.01	67.39
RECYCLING ROBOTS, ICE solvable to $h = 70$						
2	1.18	0.05	0.30	≤ 0.01	≤ 0.01	≤ 0.01
3	*	2.79	1.07	≤ 0.01	≤ 0.01	≤ 0.01
4		2136.16	42.02	≤ 0.01	≤ 0.01	0.02
5		—	1812.15	≤ 0.01	≤ 0.01	0.02
HOTEL 1, ICE solvable to $h = 9$						
2	1.92	6.14	0.22	≤ 0.01	≤ 0.01	0.03
3	315.16	2913.42	0.54	≤ 0.01	≤ 0.01	1.51
4	—	—	0.73	≤ 0.01	≤ 0.01	3.74
5			1.11	≤ 0.01	≤ 0.01	4.54
9			8.43	0.02	≤ 0.01	20.26
10			17.40	#	#	
15			283.76			
COOPERATIVE BOX PUSHING (Q_{POMDP}), ICE solvable to $h = 4$						
2	3.56	15.51	1.07	≤ 0.01	≤ 0.01	≤ 0.01
3	2534.08	—	6.43	0.91	0.02	0.15
4	—		1138.61	*	328.97	0.63

State of the Art

h	V^*	$T_{GMAA*}(s)$	$T_{IC}(s)$	$T_{ICE}(s)$
-----	-------	----------------	-------------	--------------

RECYCLING ROBOTS

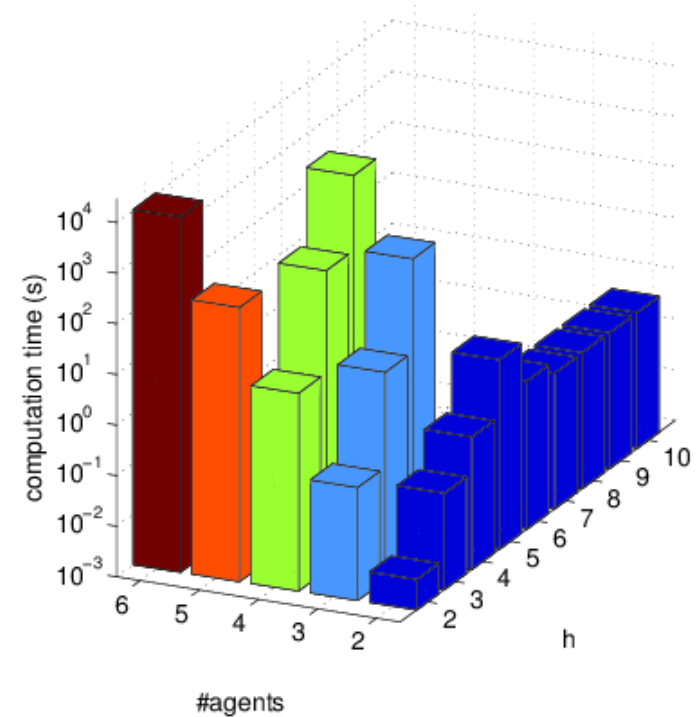
3	10.660125	≤ 0.01	≤ 0.01	≤ 0.01
4	13.380000	713.41	≤ 0.01	≤ 0.01
5	16.486000	—	≤ 0.01	≤ 0.01
6	19.554200		≤ 0.01	≤ 0.01
10	31.863889		≤ 0.01	≤ 0.01
15	47.248521		≤ 0.01	≤ 0.01
20	62.633136		≤ 0.01	≤ 0.01
30	93.402367		0.08	0.05
40	124.171598		0.42	0.25
50	154.940828		2.02	1.27
70	216.479290		—	28.66
80			—	—

BROADCAST CHANNEL

4	3.890000	≤ 0.01	≤ 0.01	≤ 0.01
5	4.790000	1.27	≤ 0.01	≤ 0.01
6	5.690000	—	≤ 0.01	≤ 0.01
7	6.590000		≤ 0.01	≤ 0.01
10	9.290000		≤ 0.01	≤ 0.01
25	22.881523		≤ 0.01	≤ 0.01
50	45.501604		≤ 0.01	≤ 0.01
100	90.760423		≤ 0.01	≤ 0.01
250	226.500545		0.06	0.07
500	452.738119		0.81	0.94
700	633.724279		0.52	0.63
800			—	—
900	814.709393		9.57	11.11
1000			—	—

Cases that compress well

* excluding heuristic



Scalability w.r.t. #agents

References

- Most references can be found in

Frans A. Oliehoek. **Decentralized POMDPs**. In Wiering, Marco and van Otterlo, Martijn, editors, *Reinforcement Learning: State of the Art*, Adaptation, Learning, and Optimization, pp. 471–503, Springer Berlin Heidelberg, Berlin, Germany, 2012.

- Other:

- Dibangoye, Amato, Buffet, & Charpillet. Optimally Solving Dec-POMDPs as Continuous-State MDPs. *IJCAI*, 2013.
- Oliehoek, Spaan, Amato, & Whiteson. Incremental Clustering and Expansion for Faster Optimal Planning in Decentralized POMDPs. *JAIR*, 2013.
- Oliehoek. Sufficient Plan-Time Statistics for Decentralized POMDPs. *IJCAI*, 2013.

Approximate Approaches

Approximate approaches

- Optimal methods are intractable for many problems and impossible for infinite-horizon
- Want to produce the best solution possible with given resources
- Finite-horizon
 - JESP
 - MBDP-based approaches
- Infinite-horizon
 - ϵ -optimal DP for infinite-horizon Dec-POMDPs
 - Optimizing fixed controllers
- Indefinite-horizon

Finite-horizon methods

- Quality bounds are usually not possible, but these approaches often perform well
- Many based on dynamic programming

Joint equilibrium search for policies (JESP)

(Nair et al., 03)

- Instead of exhaustive search, find best response
- Algorithm:
 - Start with (full) policy for each agent
 - while *not converged* do
 - for $i=1$ to n
 - Fix other agent policies
 - Find a best response policy for agent i

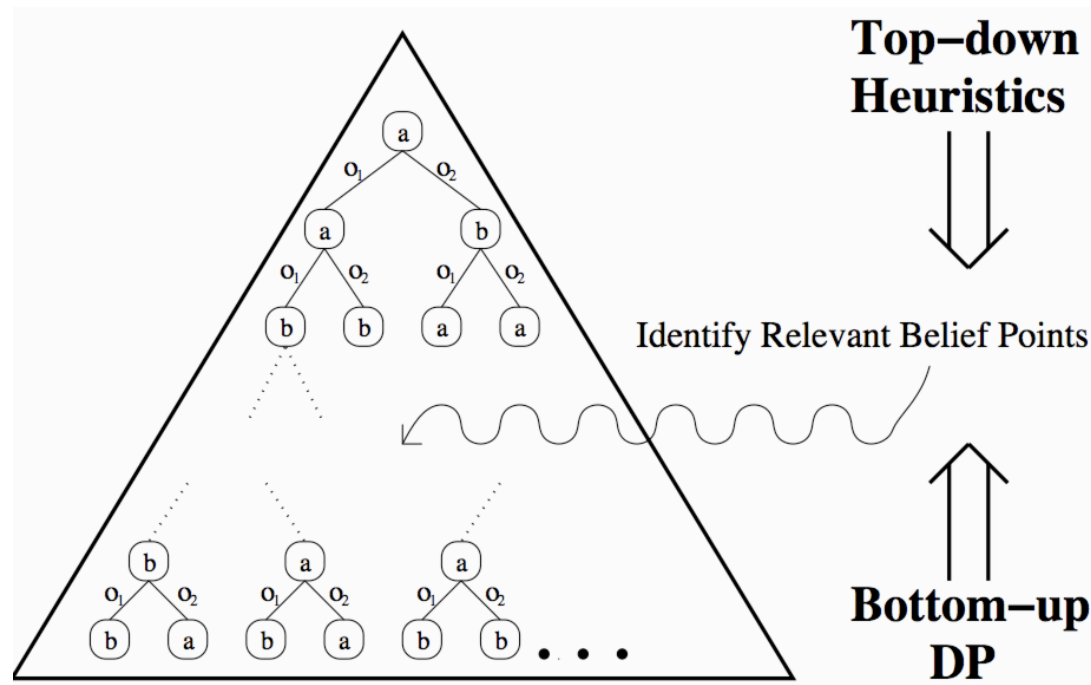
JESP summary

- Finds a *locally optimal* set of policies
- Worst case complexity is the same as exhaustive search, but in practice is much faster
- Can also incorporate dynamic programming to speed up finding best responses
 - Fix policies of other agents
 - Create a (augmented) POMDP using the fixed policies of others
 - Generate reachable belief states from initial state b_0
 - Build up policies from last step to first
 - At each step, choose subtrees that maximize value at reachable belief states

Memory bounded dynamic programming (MBDP)

(Seuken and Zilberstein., 07)

- Do not keep all policies at each step of dynamic programming
- Keep a fixed number for each agent: *maxTrees*
- Select these by using heuristic solutions from initial state
- Combines top down and bottom up approaches



MBDP algorithm

start with a one-step policy for each agent

for $t=h$ to 1 do

 backup each agent's policy

 for $k=1$ to $maxTrees$ do

 compute heuristic policy and resulting belief state b

 choose best set of trees starting at b

select best set of trees for initial state b_0

MBDP summary

- Linear complexity in problem horizon
- Exponential in the number of observations
- Performs well in practice (often with very small *maxTrees*)
- Can be difficult to choose correct *maxTrees*

Extensions to MBDP

- IMBDP: Limit the number of observations used based on probability at each belief (Seuken and Zilberstein 07)
- MBDP-OC: compress observations based on the value produced (Carlin and Zilberstein 08)
- PBIP: heuristic search to find best trees rather than exhaustive (Dibangoye et al., 09)
- Current state-of-the-art
 - PBIP-IPG: extends PBIP by limiting the possible states (Amato et al., 09-AAMAS)
 - CBPB: uses constraint satisfaction solver for subtree selection (Kumar and Zilberstein, 10)
 - PBPG: approximate, linear programming method for subtree selection (Wu et al, 10) – solves a problem with 3843 states and 11 obs to hor 20

Other finite-horizon approaches

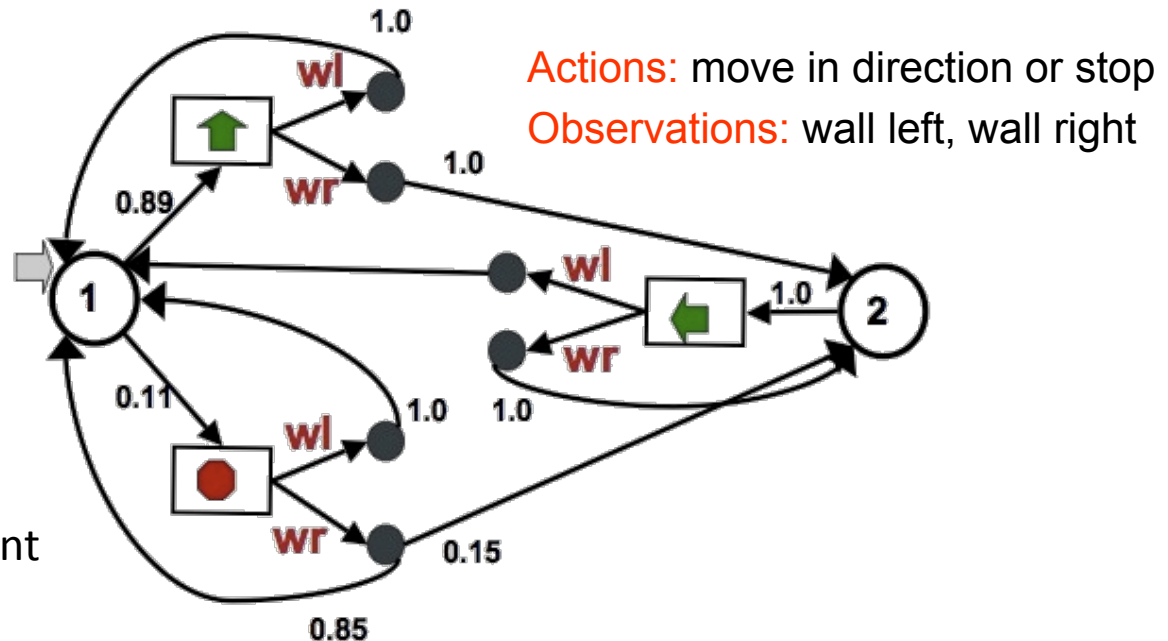
- Sampling methods
 - Direct Cross-Entropy policy search (DICE)
(Oliehoek et al., 08)
 - Randomized algorithm using combinatorial optimization
 - Applies Cross-Entropy method to Dec-POMDPs
 - Scales well wrt number of agents
 - Goal-directed sampling (Amato and Zilberstein, 09)
 - Discussed later

Infinite-horizon approaches

- A large enough horizon can be used to approximate an infinite-horizon solution, but this is neither efficient nor compact
- ϵ -optimal extension of DP
- Other specialized infinite-horizon solutions have also been developed:
 - Best-First Search (BFS)
 - Bounded Policy Iteration for Dec-POMDPs (Dec-BPI)
 - Nonlinear Programming (NLP)
 - Expectation-Maximization (EM)

Infinite-horizon policies: stochastic controllers

- Designated initial node
- Nodes define actions
- Transitions based on observations seen
- Inherently infinite-horizon
- With fixed memory, randomness can help
- One controller for each agent
- Trees for finite-horizon



- Value for node \vec{q} and state s :

Action selection, $P(a/q): Q \rightarrow \Delta A$

Transitions, $P(q'/q,o): Q \times O \rightarrow \Delta Q$

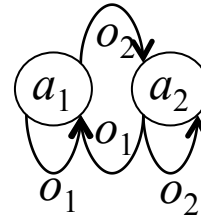
$$V(\vec{q}, s) = \sum_a P(\vec{a} | \vec{q}) \left[R(s, \vec{a}) + \gamma \sum_{s'} P(s' | s, \vec{a}) \sum_o O(\vec{o} | s', \vec{a}) \sum_{q'} P(\vec{q}' | \vec{q}, o) V(\vec{q}', s') \right]$$

Infinite horizon DP (Bernstein et al., 09)

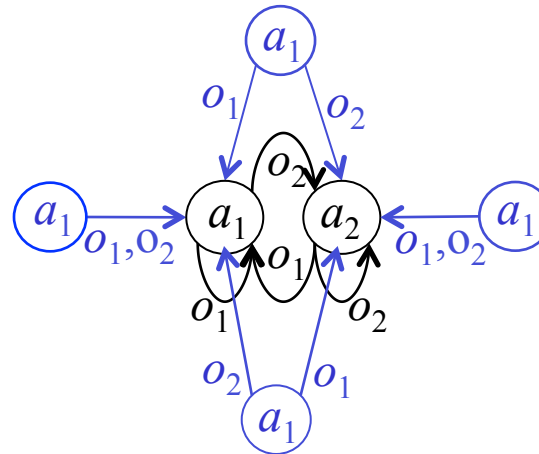
- Remember we need to define a controller for each agent
- How many nodes do you need and what should the parameter values be for an optimal *infinite-horizon* policy?
- This may be infinite!
- First ϵ -optimal algorithm for infinite-horizon Dec-POMDPs: Policy Iteration

ϵ -Optimal DP: Policy Iteration

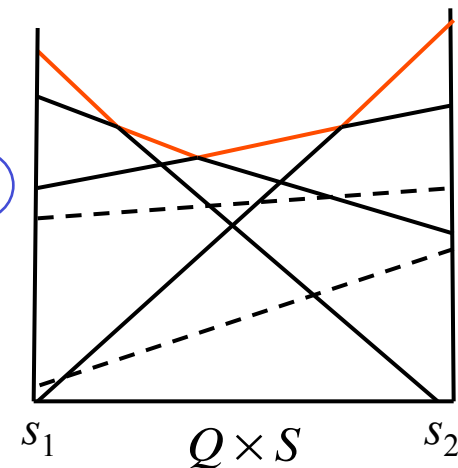
- Start with a given controller
- Exhaustive backup (for all agents): generate all next step policies
- Evaluate: determine value of starting at each node at each state and for each policy for the other agents
- Prune: remove those that always have lower value (merge as needed)
- Continue with backups and pruning until error is below ϵ



= Initial controller
for agent 1



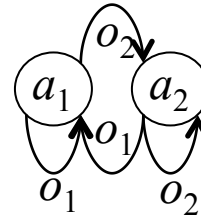
(backup for action 1)



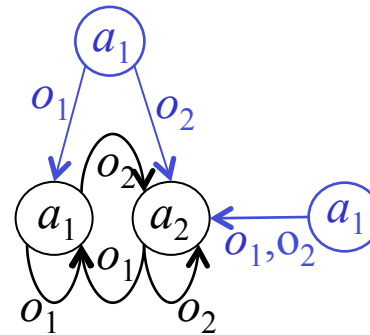
ϵ -Optimal DP: Policy Iteration

- Start with a given controller
- Exhaustive backup (for all agents): generate all next step policies
- Evaluate: determine value of starting at each node at each state and for each policy for the other agents
- Prune: remove those that always have lower value (merge as needed)
- Continue with backups and pruning until error is below ϵ

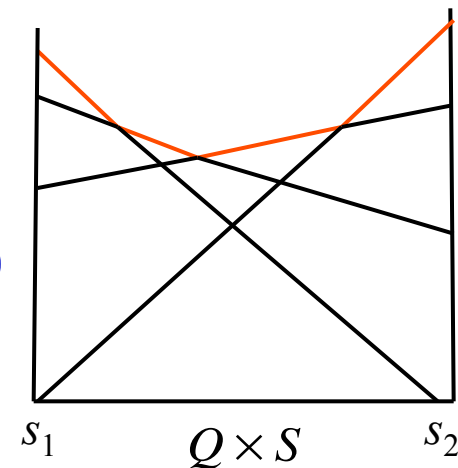
Key: Prune over not just states, but possible policies of the other agents!



= Initial controller for agent 1

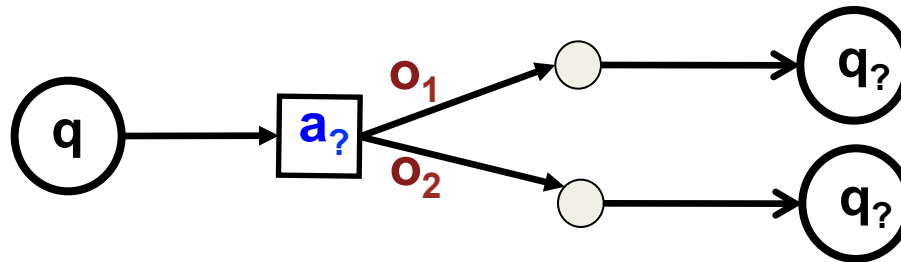


(backup for action 1)



Memory-bounded solutions

- Optimal approaches may be intractable
- Can use fixed-size finite-state controllers as policies for Dec-POMDPs
- How do we set the parameters of these controllers to maximize their value?
 - Deterministic controllers - discrete methods such as branch and bound and best-first search
 - Stochastic controllers - continuous optimization and EM



(deterministically) choosing an action and transitioning to the next node

Best-first search (Szer and Charpillet 05)

- Search through space of deterministic action selection and node transition parameters
- Produces optimal fixed-size deterministic controllers
- High search time limits this to very small controllers (< 3 nodes)

Bounded policy iteration (BPI) (Bernstein et al., 05)

- Improve the controller over a series of steps until value converges
- Alternate between improvement and evaluation
- Improvement
 - Use a linear program to determine if a node's parameters can be changed, while fixing the rest of the controller and other agent policies
 - Improved nodes must have better value for all states and nodes of the other agents (multiagent belief space)
- Evaluation: Update the value of all nodes in the agent's controller
- Can solve much larger controller than BFS, but value is low due to lack of start state info and LP

Nonlinear programming approach

(Amato et al., 07, 09b)

- Use a nonlinear program (NLP) to represent an optimal fixed-size set of controllers for Dec-POMDPs
- Consider node value as well as action and transition parameters as variables
- Maximize the value using a known start state
- Constraints maintain valid values and probabilities

NLP formulation

Variables: $x(q_i, a_i) = P(a_i | q_i), y(q_i, a_i, o_i, q_i') = P(q_i' | q_i, a_i, o_i), z(\vec{q}, s) = V(\vec{q}, s)$

Objective: Maximize $\sum_s b_0(s) z(\vec{q}_0, s)$

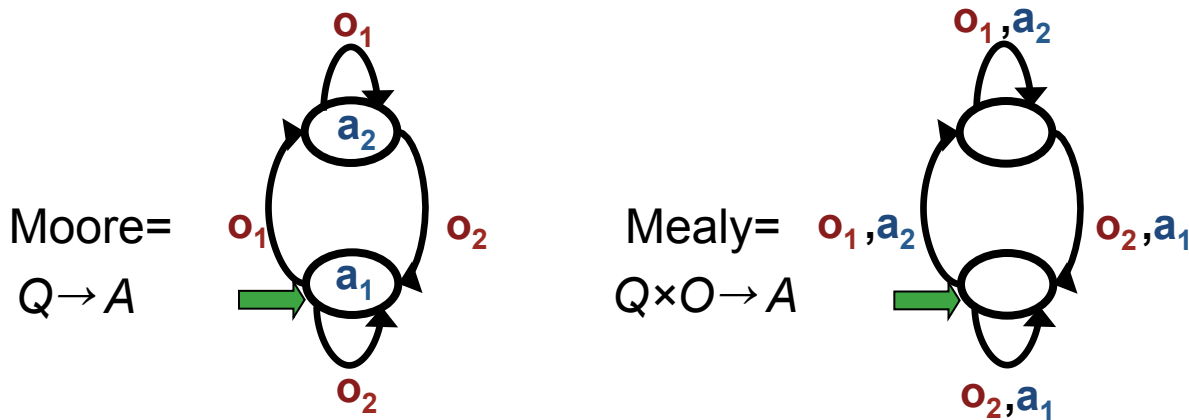
Value Constraints: $\forall s \in S, \vec{q} \in Q$

$$z(\vec{q}, s) = \sum_a \left(\prod_i x(q_i, a_i) \left[R(s, \vec{a}) + \gamma \sum_{s'} P(s' | s, \vec{a}) \sum_o O(\vec{o} | s', \vec{a}) \sum_{q'} \prod_i y(q_i', a_i, q_i, o_i) z(\vec{q}', s') \right] \right)$$

Probability constraints ensure all probabilities must sum to 1 and be greater than 0

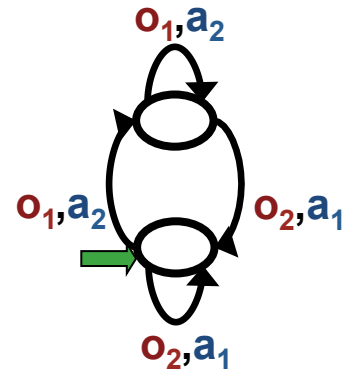
Mealy controllers (Amato et al, 10)

- Controllers currently used are Moore controllers
- Mealy controllers are more powerful than Moore controllers (can represent higher quality solutions with the same number of nodes)
- Key difference: action depends on node and observation



Mealy controllers continued

- More powerful
- Provides extra structure that algorithms can use
 - Can automatically simplify representation based on informative observations
 - Can be done in controller or solution method
- Can be used in place of Moore controllers in all controller-based algorithms for POMDPs and DEC-POMDPs (not just NLP)
 - Optimal infinite-horizon DP
 - Approximate algorithms



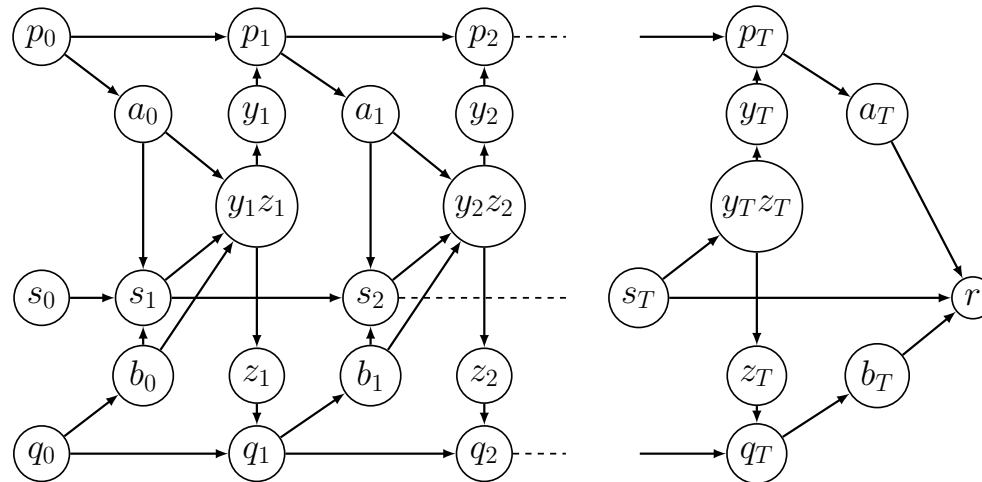
EM (Kumar & Zilberstein, 10)

- Planning as inference
- Extension of planning as inference for POMDPs (Toussaint et al., 10)
- Discounting represented by probabilistically stopping
- Considers the probability of achieving the maximum reward (reward only considered at the end)
- Maximizing the likelihood = maximizing the value of the finite-state controller

EM continued

- Represent Dec-POMDP as (infinite) mixture of DBNs for each step T
- Reward as probability $R(s,a,b) = P(r=1 | s_T = s, a_T = a, b_T = b)$
- Maximize likelihood $L^\theta(r=1; \theta)$ with $\theta = P(a|q), P(q'|q,o), P(q)$
- E step: estimate $P(s,p,q)$ (forw.) and $V(p,q,s)$ (back) with fixed θ
- M step: maximize new params θ^* for latent variables $L=S,A,B,P,Q$

$$Q(\theta, \theta^*) = \sum_T \sum_L P(r=1, L, T; \theta) \log P(r=1, L, T; \theta^*)$$



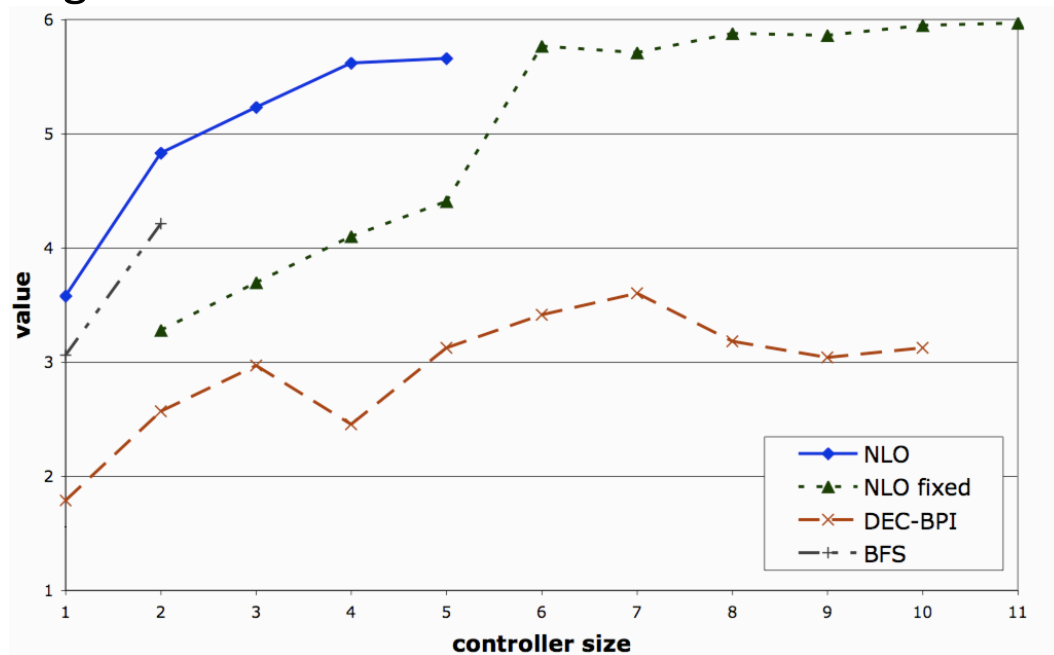
2 agents with controller nodes p and q , actions a and b

Extensions to EM

- EM for factored Dec-POMDPs (Pajarinen and Peltonen, 2011a)
 - Consider a factored state space
 - Factor the updates and rewards in EM
- EM with structured controllers (Pajarinen and Peltonen, 2011b)
 - Consider controllers that are periodic (transition from one layer to the next)
 - Can generate an initial (deterministic) solution using a fixed horizon and repeating
 - Perform EM with the above initialization

Some infinite-horizon results

- Optimal algorithm can only solve very small problems
- Approximate algorithms are more scalable



- GridSmall: 16 states, 4 actions, 2 obs
- Policy Iteration: 3.7 with 80 nodes in 821s before out of memory

Indefinite-horizon

- Unclear how many steps are needed until termination
- Many natural problems terminate after a goal is reached
 - Meeting or catching a target
 - Cooperatively completing a task



Indefinite-horizon Dec-POMDPs (Amato et al, 09a)

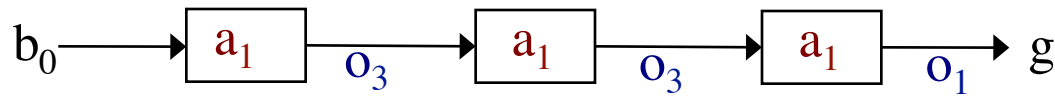
- Extends indefinite-horizon POMDPs Patek 01 and Hansen 07
- Our assumptions
 - Each agent possesses a set of terminal actions
 - Negative rewards for non-terminal actions
- Can capture uncertainty about reaching goal
- Many problems can be modeled this way

An optimal solution to this problem can be found using dynamic programming

Goal-directed Dec-POMDPs

- Relax assumptions, but still have goal
- Problem terminates when
 - A single agent or set of agents reach local or global goal states
 - Any combination of actions and observations is taken or seen
- More problems fall into this class (can terminate without agent knowledge)
- Solve by *sampling* trajectories
 - Produce only action and observation sequences that lead to goal
 - This reduces the number of policies to consider

Can bound the number of samples required to approach optimality



Infinite and indefinite-horizon results

- Standard infinite-horizon benchmarks
- Approximate solutions
- Different approaches perform well on different problems

Algorithm (Size, Time): Value

DecTiger ($ S = 2, A_i = 3, O_i = 2$)	
Peri (10 × 30, 202s):	13.45
PeriEM (7 × 10, 6540s):	9.42
Goal-directed (11, 75s):	5.041
NLP (19, 6173s):	-1.088
Mealy NLP (4, 29s):	-1.49
EM (6, 142s):	-16.30
Recycling robots ($ S = 4, A_i = 3, O_i = 2$)	
Mealy NLP (1, 0s):	31.93
Peri (6 × 30, 77s):	31.84
PeriEM (6 × 10, 272s):	31.80
EM (2, 13s):	31.50
Meeting in a 2x2 grid ($ S = 16, A_i = 5, O_i = 2$)	
Peri (5 × 30, 58s):	6.89
PeriEM (5 × 10, 6019s):	6.82
EM (8, 5086s):	6.80
Mealy NLP (5, 116s):	6.13
HPI+NLP (7, 16763s):	6.04
NLP (5, 117s):	5.66
Goal-directed (4, 4s):	5.64

Wireless network ($ S = 64, A_i = 2, O_i = 6$)	
EM (3, 6886s):	-175.40
Peri (15 × 100, 6492s):	-181.24
PeriEM (2 × 10, 3557s):	-218.90
Mealy NLP (1, 9s):	-296.50
Box pushing ($ S = 100, A_i = 4, O_i = 5$)	
Goal-directed (5, 199s):	149.85
Peri (15 × 30, 5675s):	148.65
Mealy NLP (4, 774s):	143.14
PeriEM (4 × 10, 7164s):	106.68
HPI+NLP (10, 6545s):	95.63
EM (6, 7201s):	43.33
Mars rovers ($ S = 256, A_i = 6, O_i = 8$)	
Peri (10 × 30, 6088s):	24.13
Goal-directed (6, 956s):	21.48
Mealy NLP (3, 396s):	19.67
PeriEM (3 × 10, 7132s):	18.13
EM (3, 5096s):	17.75
HPI+NLP (4, 111s):	9.29

Results from (Pajarinen and Peltonen, 2011b)

Summary

- Optimal algorithms for Dec-POMDPs give performance guarantees, but are often intractable
 - Top down and bottom up methods provide similar performance
- Approximate Dec-POMDP algorithms are much more scalable, but (often) lack quality bounds
 - Bounding memory and sampling are dominant approaches
- Using subclasses can significantly improve solution scalability (if assumptions hold)
 - Discussed later

References

- **Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs.** Christopher Amato, Blai Bonet and Shlomo Zilberstein. *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence (AAAI-10)*, Atlanta, GA, July, 2010.
- **Optimizing Fixed-size Stochastic Controllers for POMDPs and Decentralized POMDPs.** Christopher Amato, Daniel S. Bernstein and Shlomo Zilberstein. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)* 2009.
- **Incremental Policy Generation for Finite-Horizon DEC-POMDPs.** Christopher Amato, Jilles Steeve Dibangoye and Shlomo Zilberstein. *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*, Thessaloniki, Greece, September, 2009.
- **Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs.** Jilles S. Dibangoye, Abdel-Iliah Mouaddib, and Brahim Chaib-draa. In *Proc. of the Joint International Conference on Autonomous Agents and Multi-Agent Systems*, 2009.
- **Achieving Goals in Decentralized POMDPs.** Christopher Amato and Shlomo Zilberstein. *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-09)*, Budapest, Hungary, May, 2009.
- **Policy Iteration for Decentralized Control of Markov Decision Processes.** Daniel S. Bernstein, Christopher Amato, Eric A. Hansen and Shlomo Zilberstein. *Journal of AI Research (JAIR)*, vol. 34, pages 89-132, February, 2009.

References continued

- **Value-based observation compression for DEC-POMDPs.** Alan Carlin and Shlomo Zilberstein. In *Proceedings of the Joint Conference on Autonomous Agents and Multi Agent Systems*, 2008.
- **Improved memory-bounded dynamic programming for decentralized POMDPs.** Sven Seuken and Shlomo Zilberstein. *Proceedings of Uncertainty in Artificial Intelligence*, July, 2007
- **Memory-Bounded Dynamic Programming for DEC-POMDPs.** Sven Seuken and Shlomo Zilberstein. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligences (IJCAI-07)*, January 2007
- **Anytime Planning for Decentralized POMDPs using Expectation Maximization.** Akshat Kumar and Shlomo Zilberstein. In *Proc. of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 294-301, 2010.
- **Point-based Backup for Decentralized POMDPs: Complexity and New Algorithms.** Akshat Kumar and Shlomo Zilberstein. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1315-1322, 2010.
- **Efficient Planning for Factored Infinite-Horizon DEC-POMDPs.** Joni Pajarinen and Jaakko Peltonen. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 325–331. AAAI Press, July 2011.
- **Periodic Finite State Controllers for Efficient POMDP and DEC-POMDP Planning.** Joni Pajarinen and Jaakko Peltonen. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2636–2644, December 2011.
- **Probabilistic inference for solving (PO)MDPs.** M. Toussaint, S. Harmeling, and A. Storkey. Technical Report EDIINF-RR-0934, University of Edinburgh, School of Informatics, 2006

References continued

- **Bounded policy iteration for decentralized POMDPs.** Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.
- **An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs.** Daniel Szer and Francois Charpillet. In *European Conference on Machine Learning*, 2005.
- **Approximate solutions for partially observable stochastic games with common payoffs.** Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. In *Proceedings of the Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.
- **Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings.** Ranjit Nair, David Pynadath, Makoto Yokoo, Milind Tambe and Stacy Marsella. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, August, 2003.
- **Point-Based Policy Generation for Decentralized POMDPs,** Feng Wu, Shlomo Zilberstein, and Xiaoping Chen, *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-10)*, Page 1307- 1314, Toronto, Canada, May 2010.

Exploiting Problem Structure

Stefan Witwicki

Postdoc at INESC-ID and IST, Portugal
witwicki@inesc-id.pt

Motivation

- Dec-POMDP: NEXP-Complete
- Benchmark problems like DEC-TIGER appear small
- General (optimal) solution methods apparently limited in scalability
- How can we ever hope to solve real world problems?

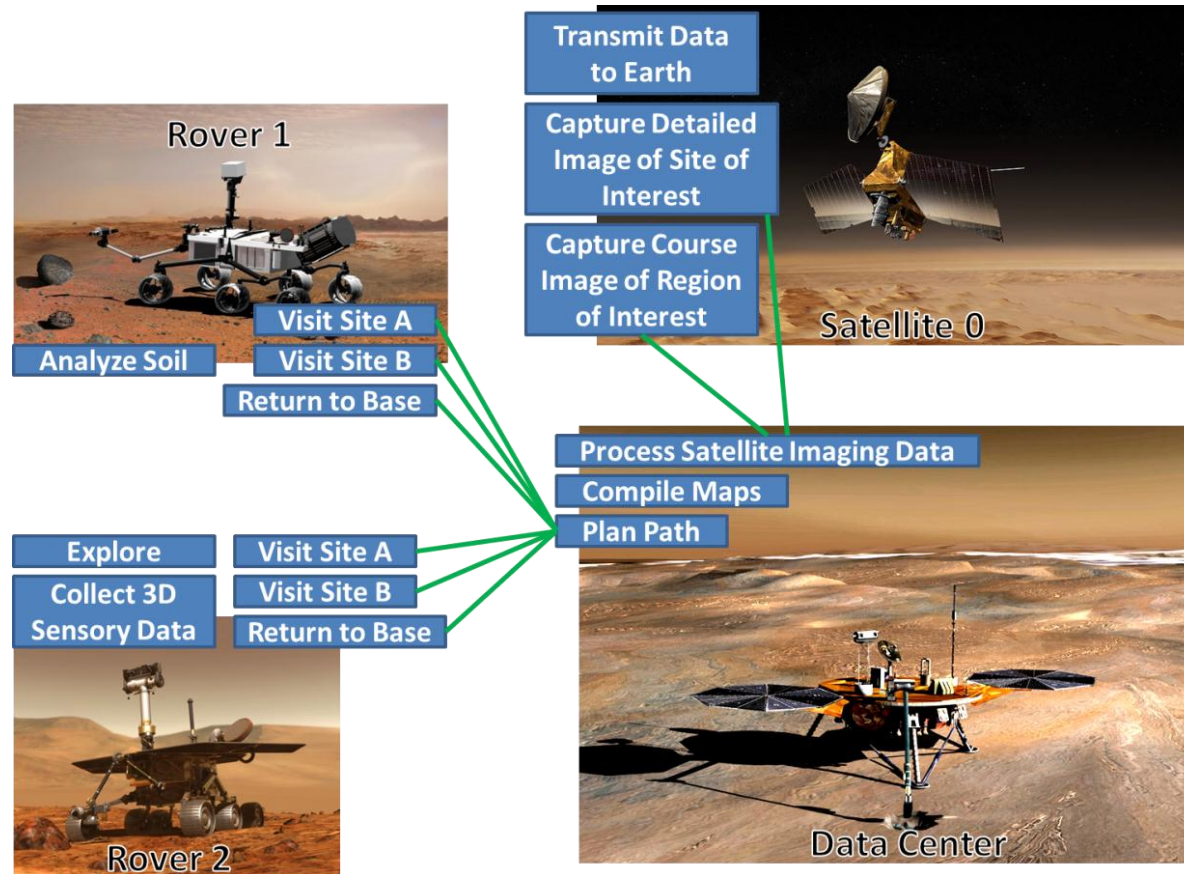
Exploiting Problem Structure

*structural assumptions, representations,
& solution techniques*

Overview:

- Graphical Structure in **Factored Models**
 - fDec-POMDP, TI-Dec-MDP, ND-PODMP
- **Decoupling** and Exploiting **Locality of Interaction**
 - Locality in General fDec-POMDPs
- Abstracting **Influences**
 - TD-POMDPs, Influence-space Search
- Quantifying “Weak Coupling”

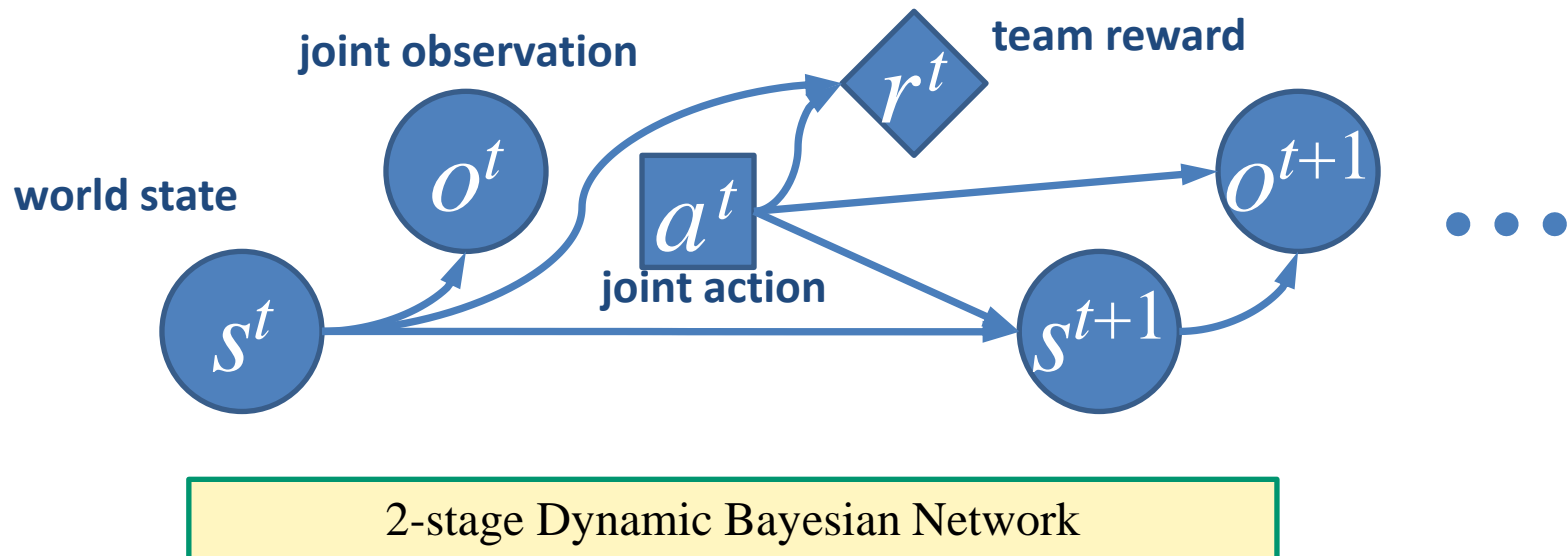
Example Domain: Mars Rovers, Satellites, etc.



Team of “**weakly-coupled**” agents who...

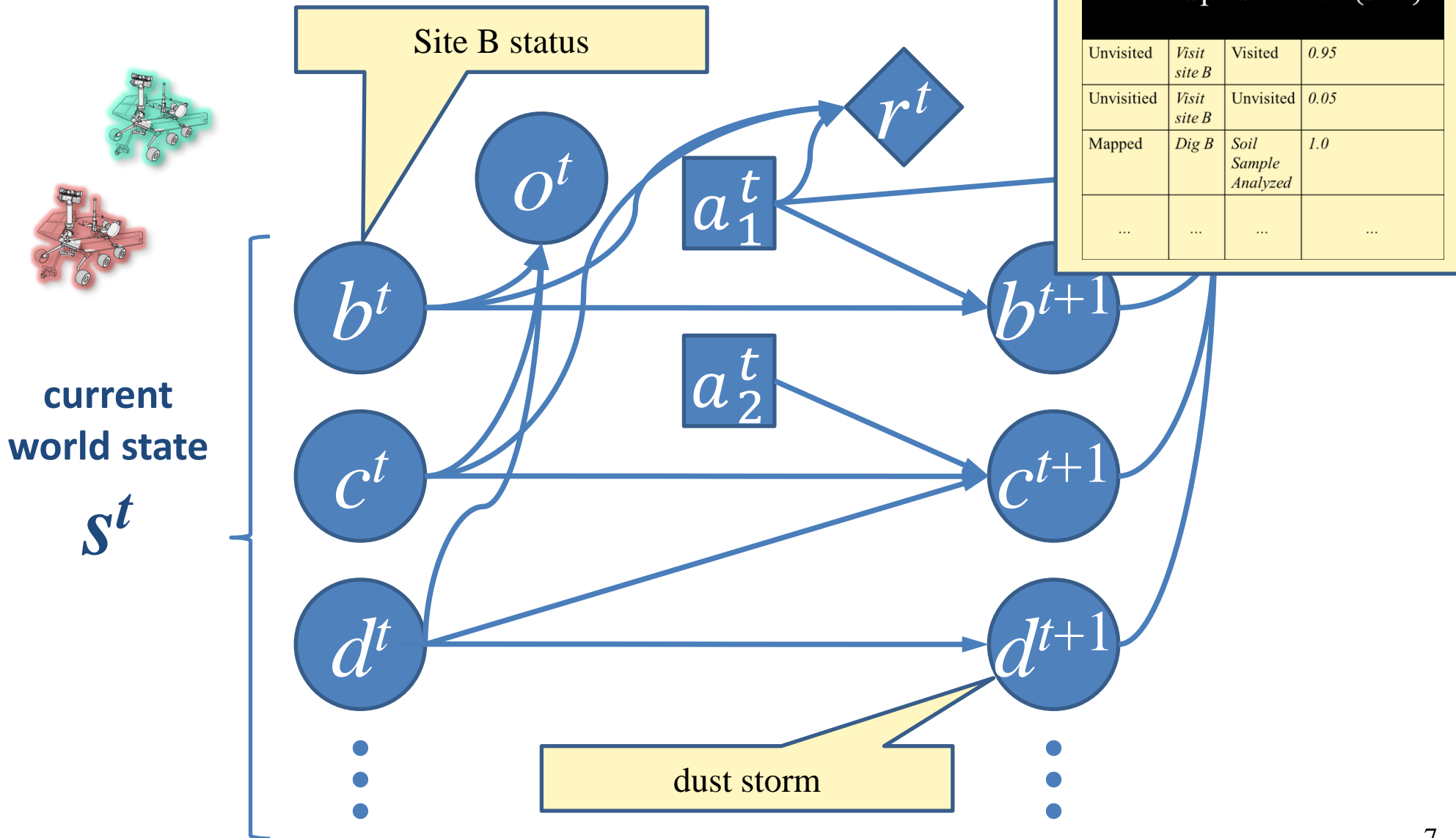
- observe a shared environment from differing perspectives
- perform separate, but interdependent activities
- Optimal joint policy requires coordinating selection and timing of activities

Conventional Dec-POMDP representation



Factored Dec-POMDP

- Graphical structure exposes conditional independencies among decision variables



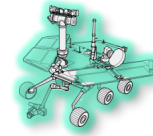
Transition-Independent Dec-MDP

[Becker Zilberstein Lesser & Goldman, JAIR 2004]

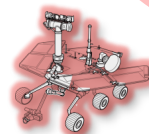
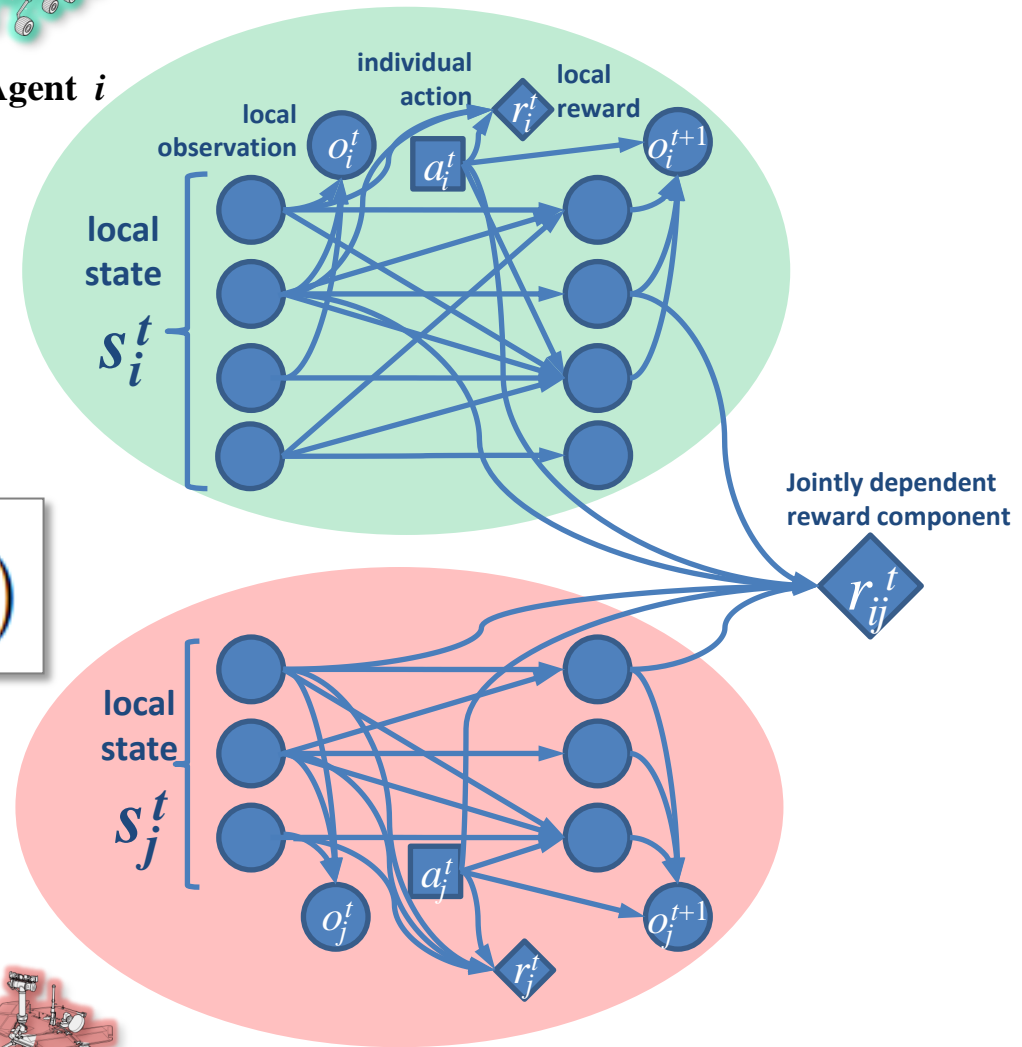
- **Transition Independence**

- world state is factored into **local state** feature subsets
- One agent's local state is independent of another's actions

$$Pr(s_i^{t+1} | s^t, a^t) = Pr(s_i^{t+1} | s_i^t, a_i^t)$$



Agent i



Agent j

Transition and Observation Independence

A Dec-POMDP is **transition independent** if:

$$Pr(s_i^{t+1} | s^t, a^t) = Pr(s_i^{t+1} | s_i^t, a_i^t)$$

A Dec-POMDP is **observation independent** if:

$$O(o^{t+1} | a^t, s^{t+1}) = \prod_{i \in \mathcal{N}} O_i(o_i^{t+1} | a_i^t, s_i^{t+1})$$

Transition-Independent Dec-MDP

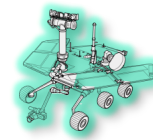
[Becker Zilberstein Lesser & Goldman, JAIR 2004]

- **Transition Independence**

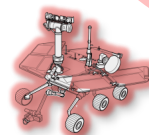
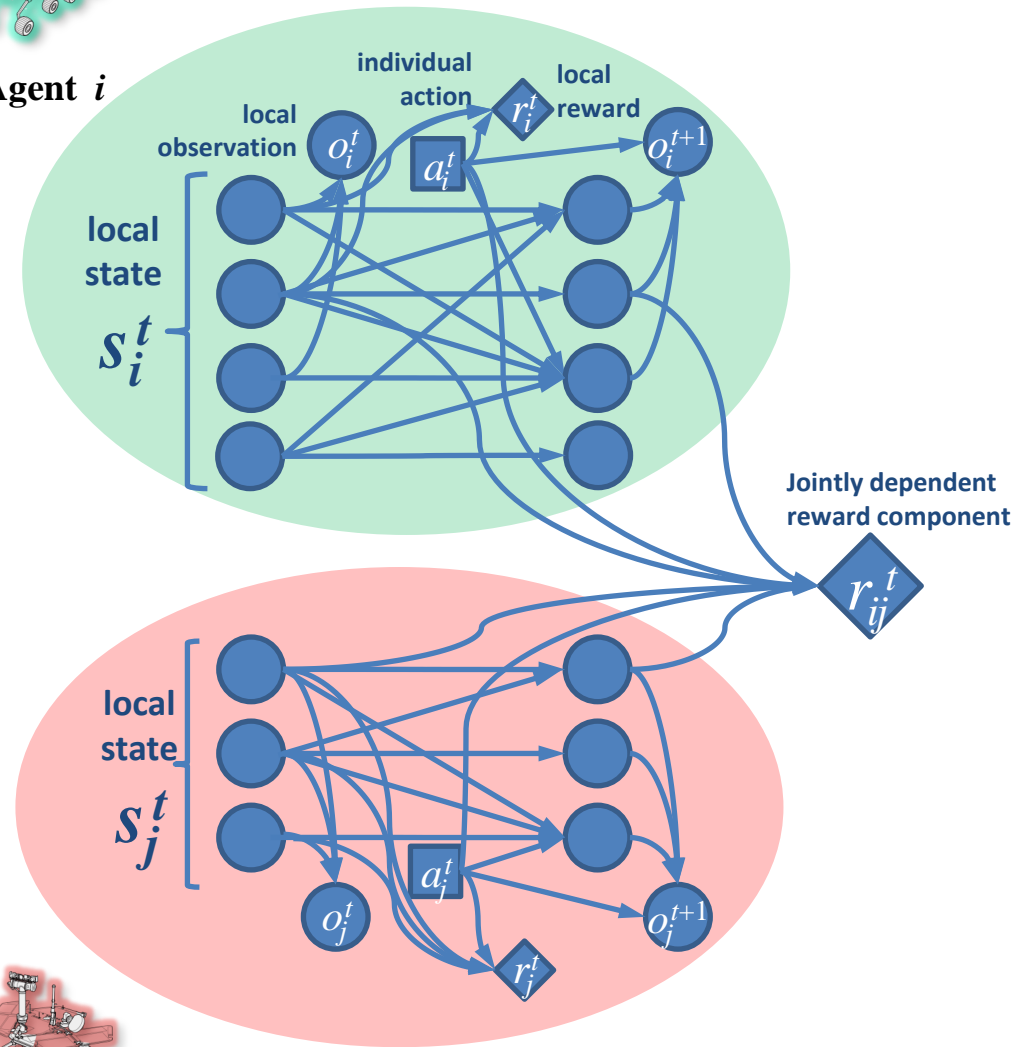
- world state is factored into **local state** feature subsets
- One agent's local state is independent of another's actions

- **Local full observability**

- Each agent observes its local state directly (but not others')



Agent i



Agent j

Transition and Observation Independence

A Dec-POMDP is **transition independent** if:

$$Pr(s_i^{t+1} | s^t, a^t) = Pr(s_i^{t+1} | s_i^t, a_i^t)$$

A Dec-POMDP is **observation independent** if:

$$O(o^{t+1} | a^t, s^{t+1}) = \prod_{i \in \mathcal{N}} O_i(o_i^{t+1} | a_i^t, s_i^{t+1})$$

A Dec-POMDP is **reward independent** if there are functions f and R_1 through R_n such that

$$R(s, a) = f(R_1(s_1, a_1), R_2(s_2, a_2), \dots, R_n(s_n, a_n))$$

Transition-Independent Dec-MDP

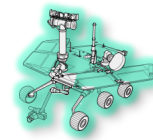
[Becker Zilberstein Lesser & Goldman, JAIR 2004]

- **Transition Independence**

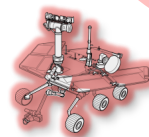
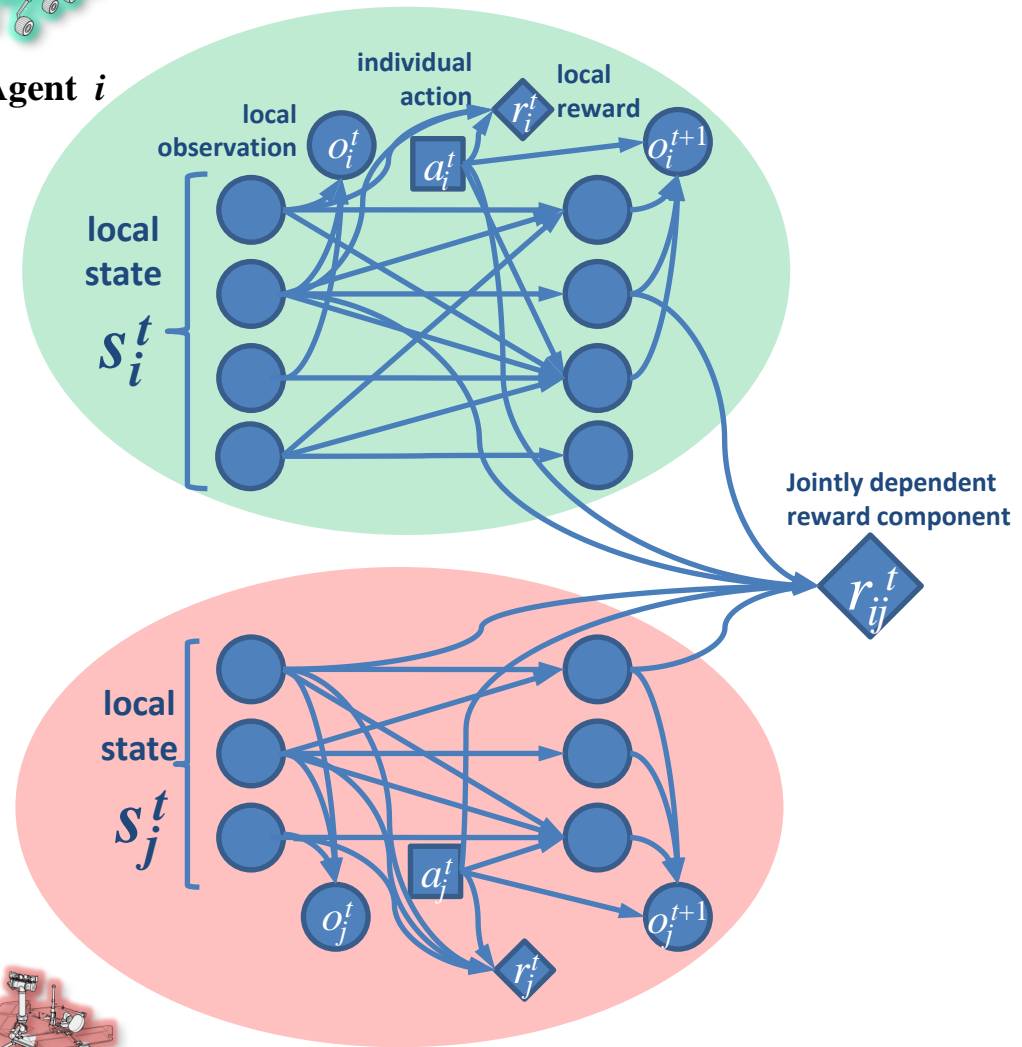
- world state is factored into **local state** feature subsets
- One agent's local state is independent of another's actions

- **Local full observability**

- Each agent observes its local state directly (but not others')
- Agents are coupled only through their rewards
 - e.g., various activities have sub-additive or super-additive value



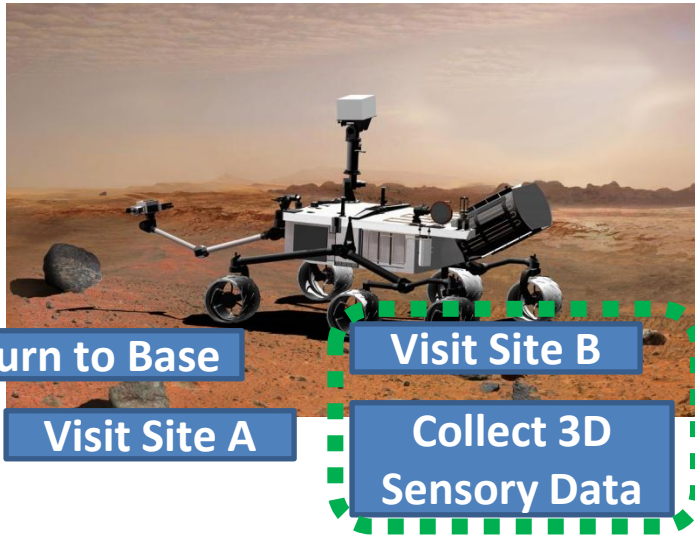
Agent i



Agent j

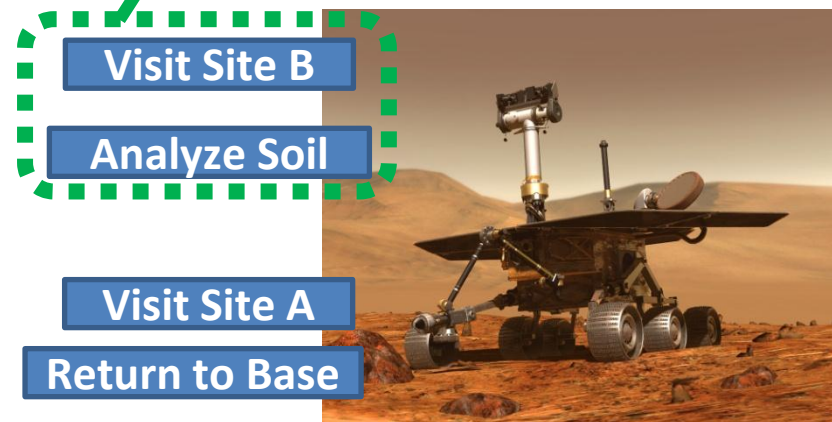
Example Interaction in TI-Dec-MDPs

Rover 1



$$r' = r_{1(s_1, a_1)} + r_{2(s_2, a_2)} + r_e$$

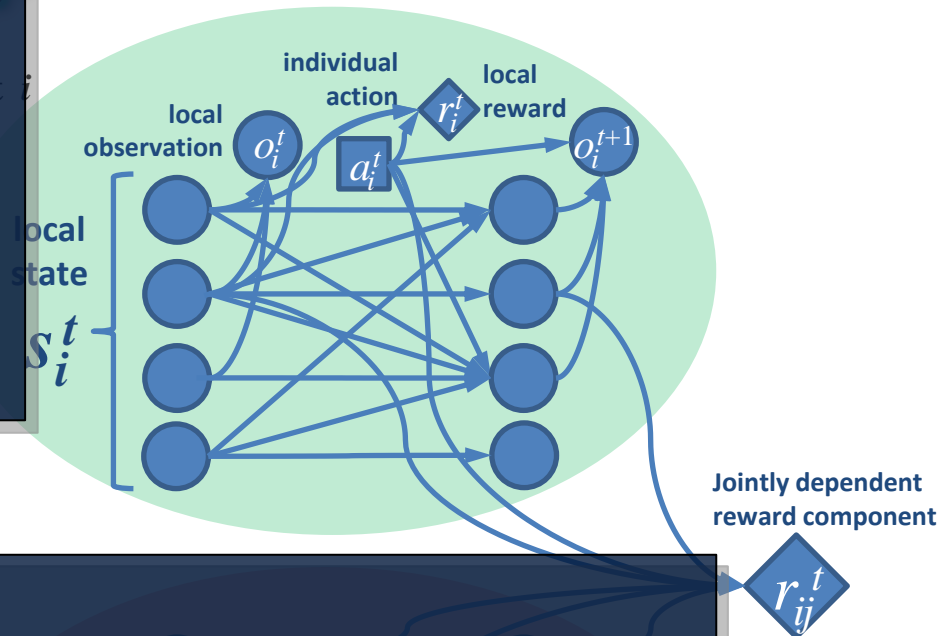
Rover 2



Transition-Independent Dec-MDP

[Becker Zilberstein Lesser & Goldman, JAIR 2004]

- **Transition Independence**
 - Easier to Specify
 - single-agent MDPs
 - Events (in the form of joint transitions) that trigger reward dependencies
- world state is independent of another's actions
- One agent's local state is independent of another's actions



- **Local full observability**
 - Easier to Solve
 - Solution complexity **NP** instead of NEXP
 - Efficient algorithms include...
 - Coverage Set Algorithm [Becker *et al.*, 2004]
 - Bi-linear programming [Petrik *et al.*, 2009]
 - [Dibangoye *et al.*, AAMAS 2013]
- Agents are coupled only through their rewards
 - e.g., various sub-additive or super-additive value

Decoupling

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} [V(\pi)]$$

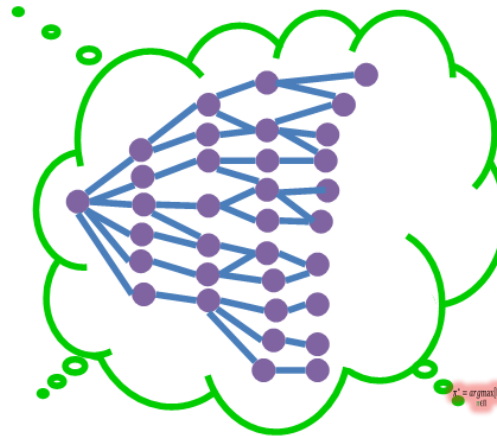
$$x^* = \underset{\pi \in \Pi}{\operatorname{argmax}} [v]$$

joint decisions

π

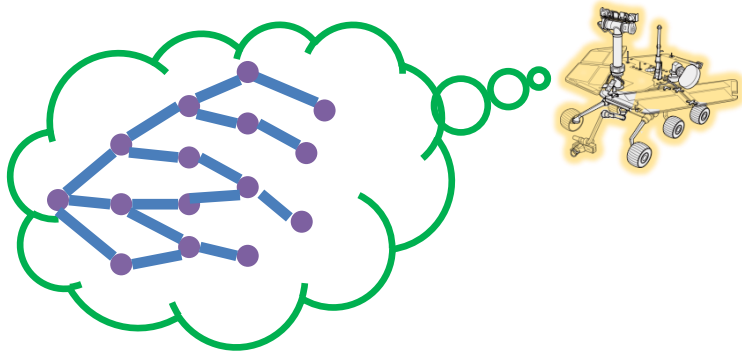
$$x^* = \underset{\pi \in \Pi}{\operatorname{argmax}} [v]$$

$$x^* = \underset{\pi \in \Pi}{\operatorname{argmax}} [v]$$

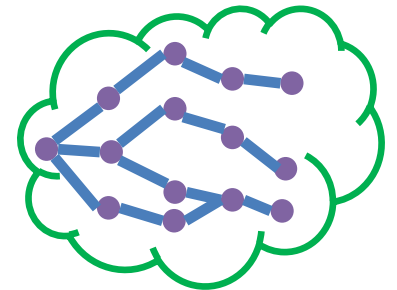
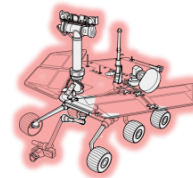
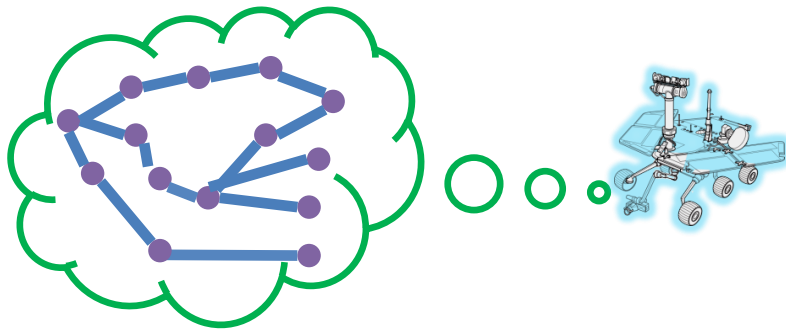


Decoupling

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} [V(\pi)] = \underset{\langle \pi_1, \dots, \pi_n \rangle}{\operatorname{argmax}} [V(\pi_1, \dots, \pi_n)]$$

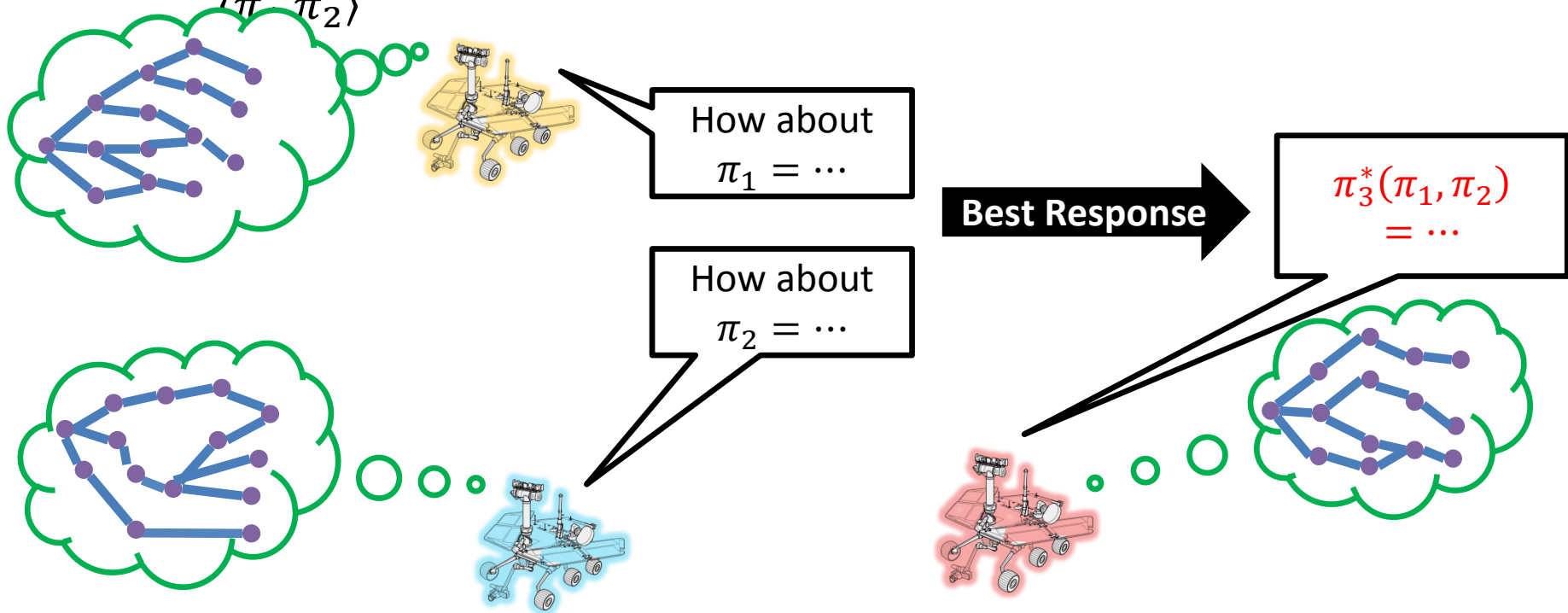


individual agent
decisions π_i



Best Response

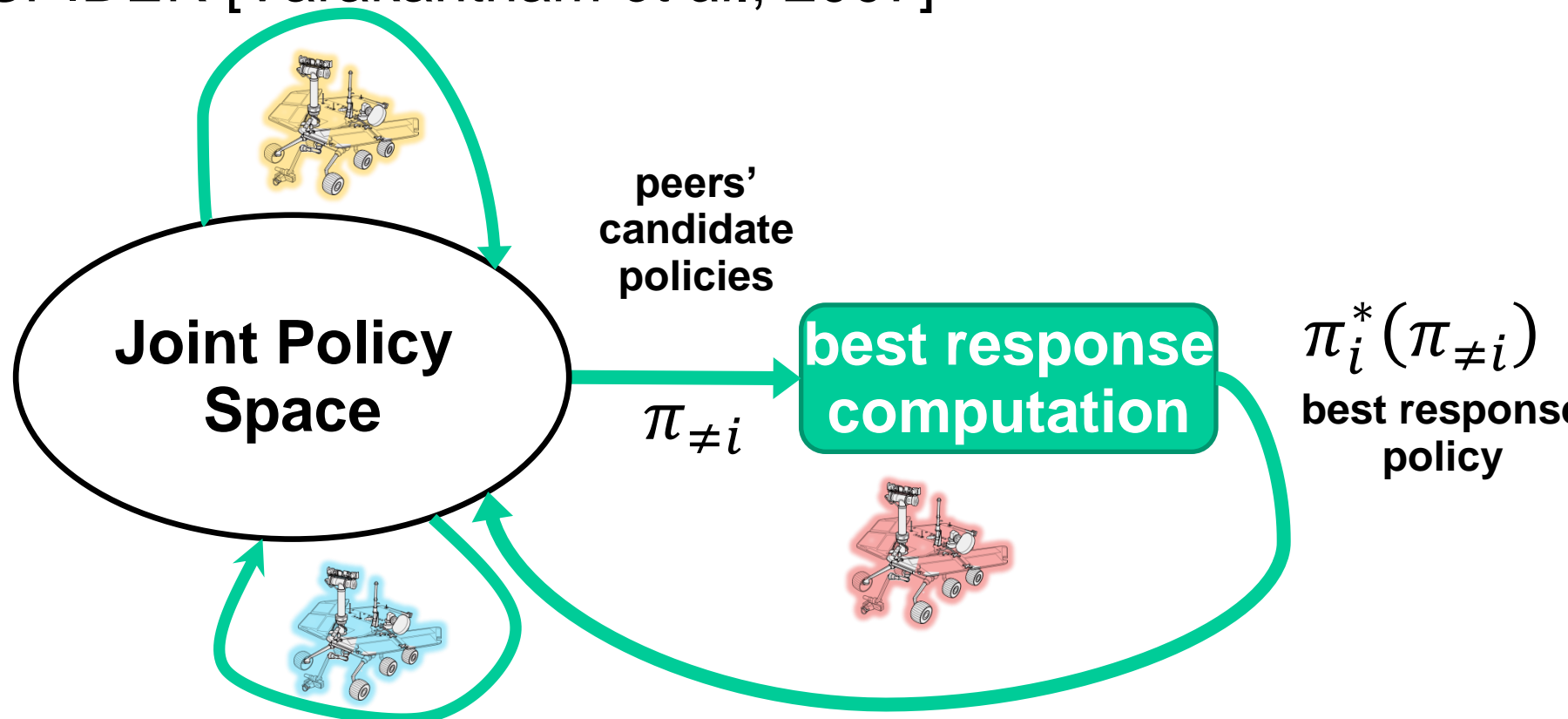
$$\begin{aligned}
 \pi^* &= \underset{\langle \pi_1, \pi_2, \pi_3 \rangle}{\operatorname{argmax}} [V(\pi_1, \pi_2, \pi_3)] \\
 &= \underset{\langle \pi_1, \pi_2 \rangle}{\operatorname{argmax}} \left[V \left(\pi_1, \pi_2, \underset{\pi_3}{\operatorname{argmax}} [V(\pi_1, \pi_2, \pi_3)] \right) \right] \\
 &= \underset{\langle \pi_1, \pi_2 \rangle}{\operatorname{argmax}} [V(\pi_1, \pi_2, \pi_3^*(\pi_1, \pi_2))] = \dots
 \end{aligned}$$



Decoupled Joint Policy Search

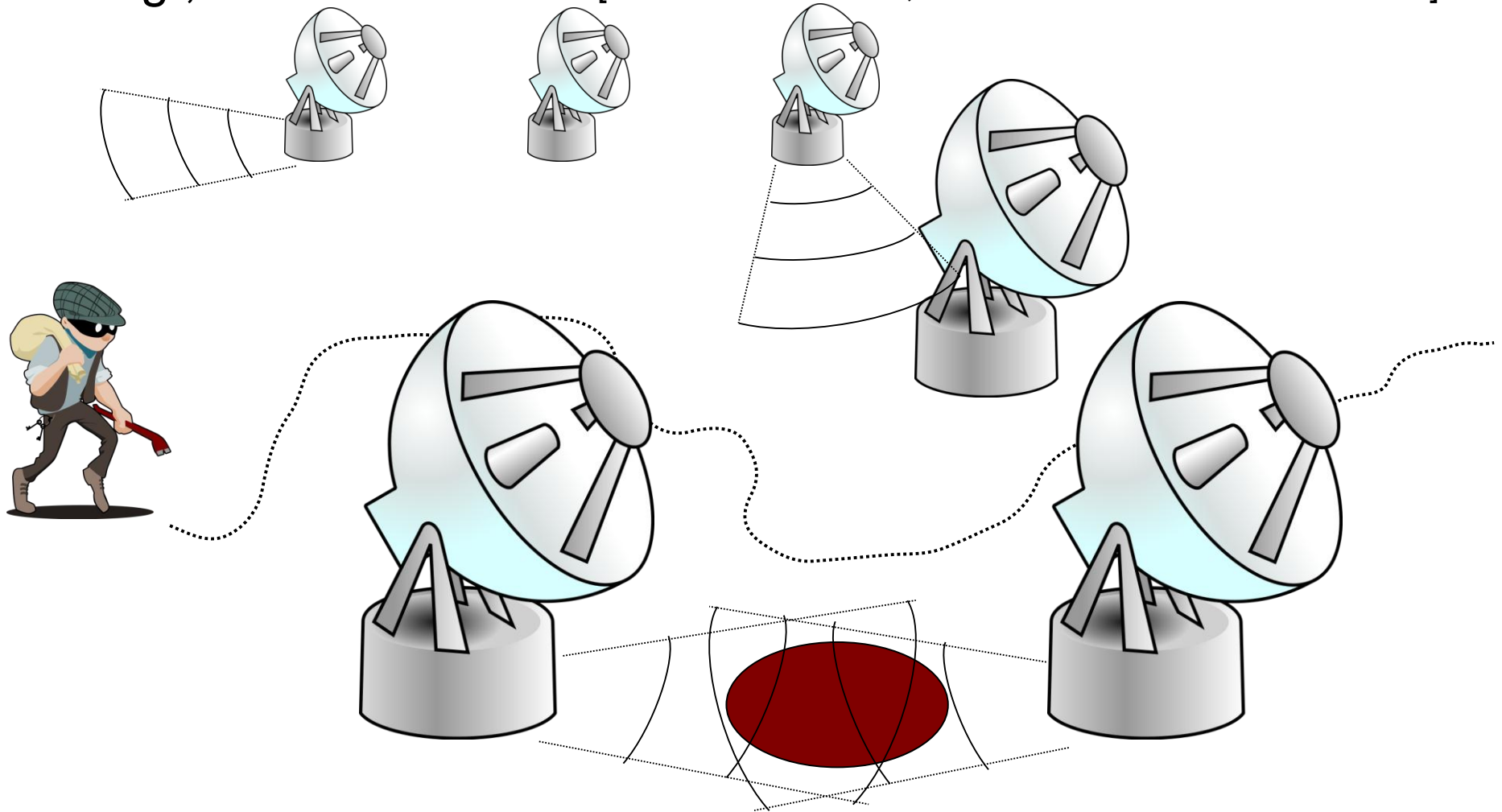
best-response search through the joint policy space

- Hill-climbing: JESP [Nair *et al.*, 2003]
- Exhaustive: GOA [Nair *et al.*, 2003]
- Graph-based methods: LID-JESP [Nair *et al.*, 2005], SPIDER [Varakantham *et al.*, 2007]



Locality of Interaction Among Agents

- Exploit (conditional) independence between agents
 - E.g., sensor networks [Nair et al '05 AAI, Varakantham et al. '07 AAMAS]



Network-Distributed POMDP

[Nair Varakantham Tambe & Yokoo, AAAI 2005]

- *Transition Independence*

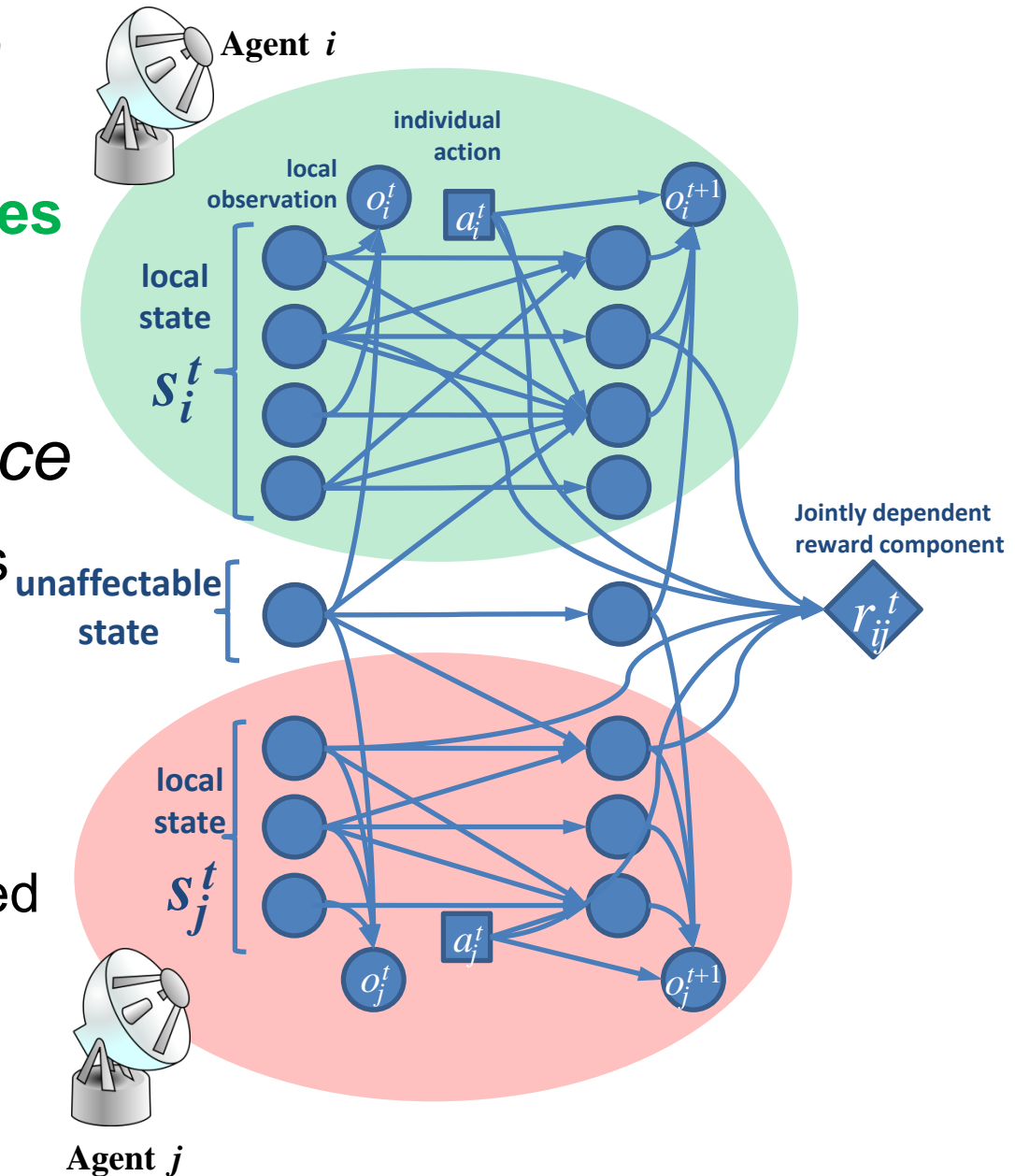
→ world state is factored into locally-affectable **local states** and **unaffected shared state**

- *Observation Independence*

→ Agents' partial observations are independent of others' local states

- *Reward Dependence*

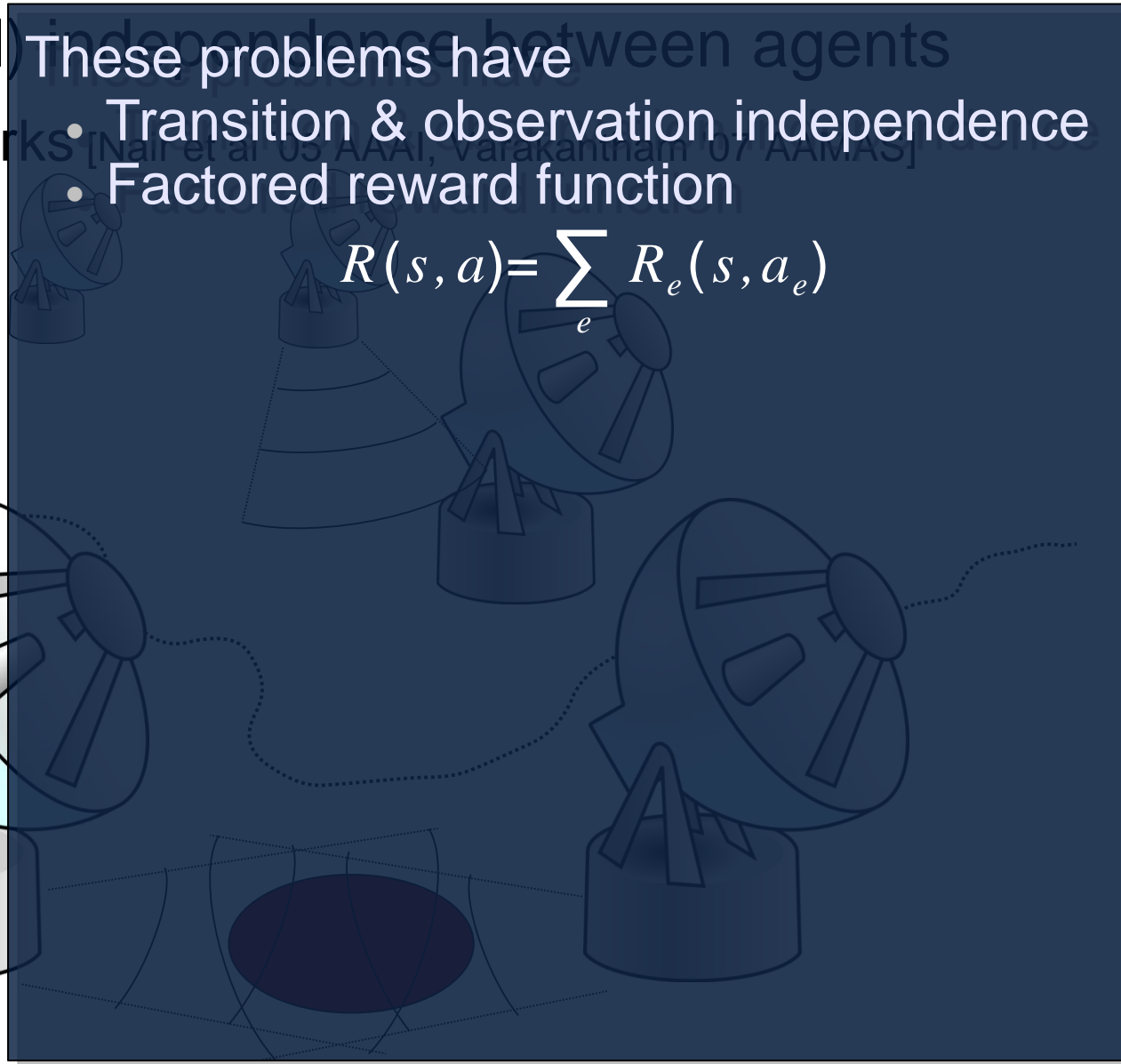
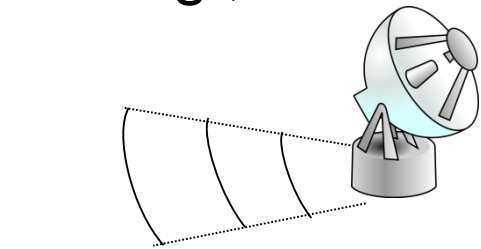
→ “neighborhood” rewards shared among sub-groups of agents



Locality of Interaction Among Agents

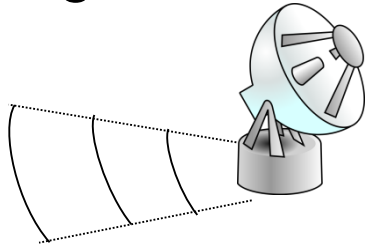
- Exploit (conditional) independence between agents
 - E.g., sensor networks
- These problems have
 - Transition & observation independence
 - Factored reward function

$$R(s, a) = \sum_e R_e(s, a_e)$$



Locality of Interaction Among Agents

- Exploit (conditional) independence between agents
 - E.g., sensor network

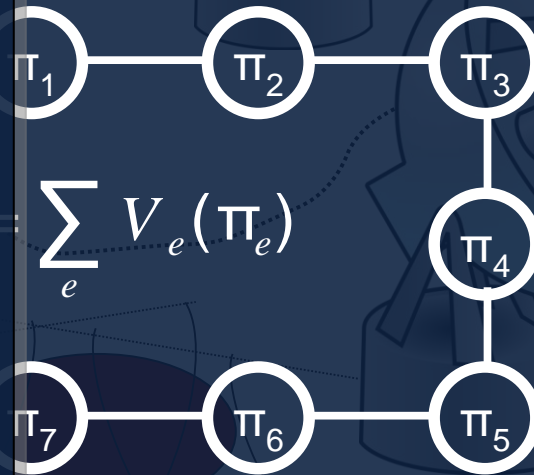


$$R(s, a) = \sum_e R_e(s, a_e)$$

High-level graphical structure captured by *interaction graph*

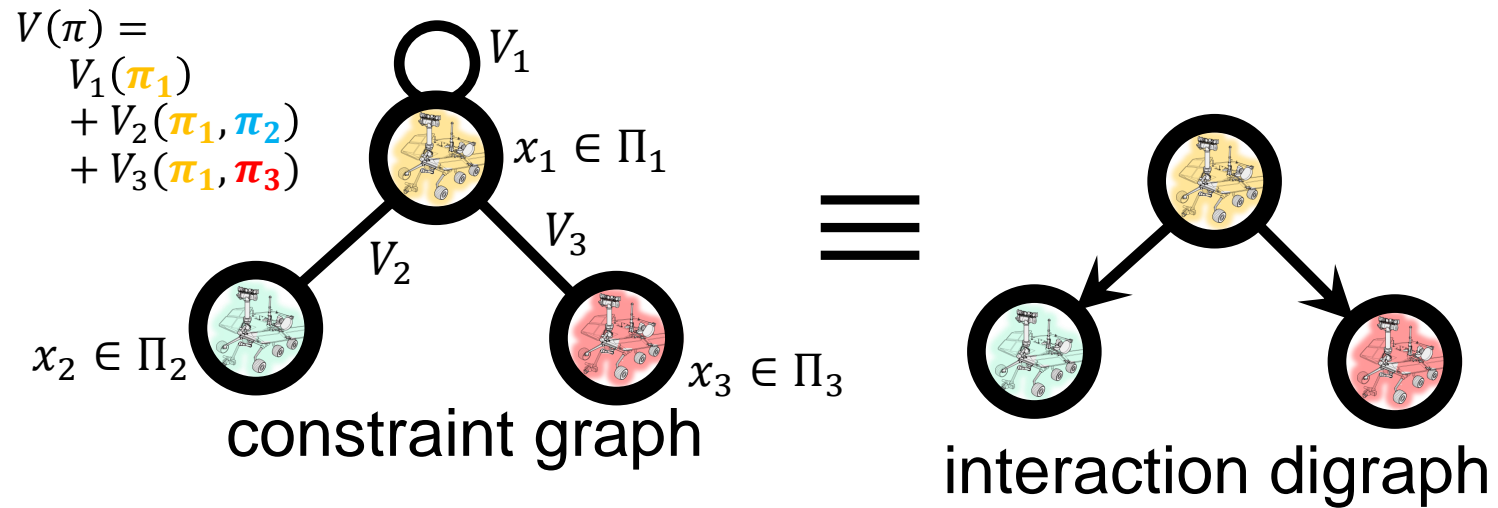
**ND-POMDPs have allowed
Optimal solutions for
10 agents!**

[Varakantham et al., 2009]



$$V(\pi) = \sum_e V_e(\pi_e)$$

Interaction Graphs



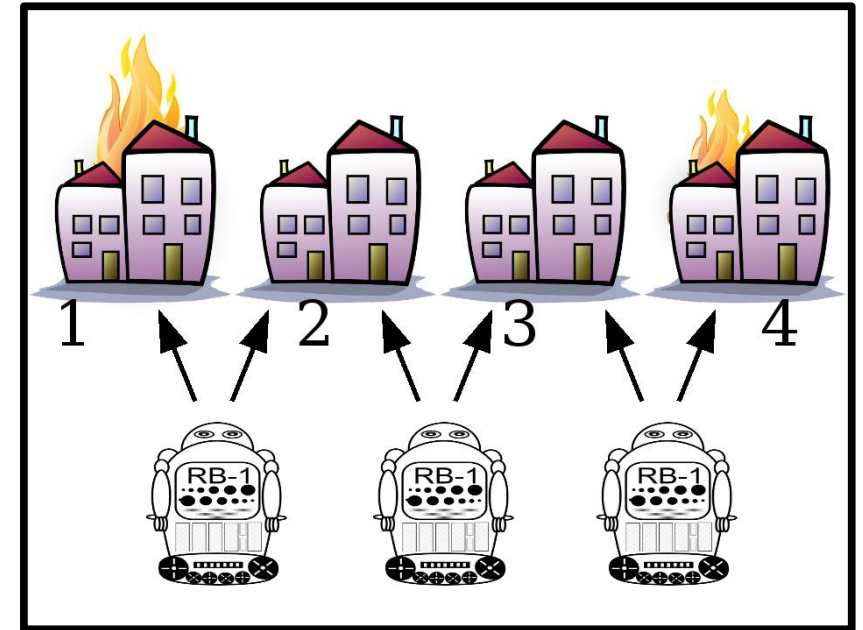
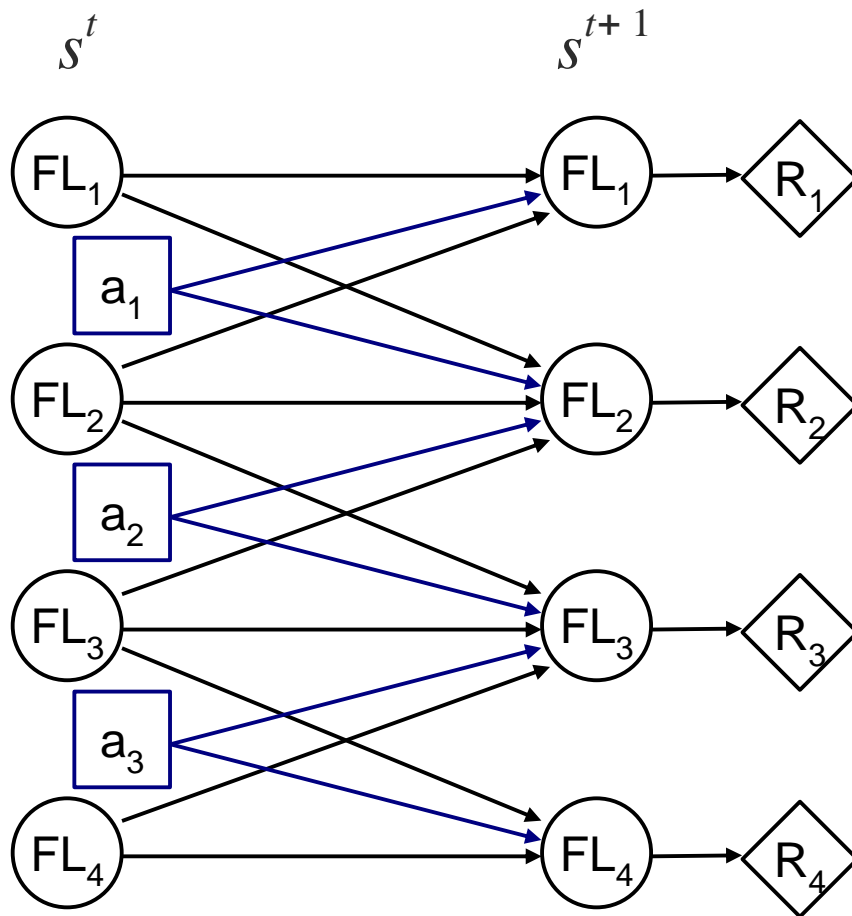
What might cause a constraint in the constraint graph?

- A reward dependency (e.g., in a TI-Dec-MDP or ND-POMDP)
- A transition dependency

Locality in General fDec-POMDPs

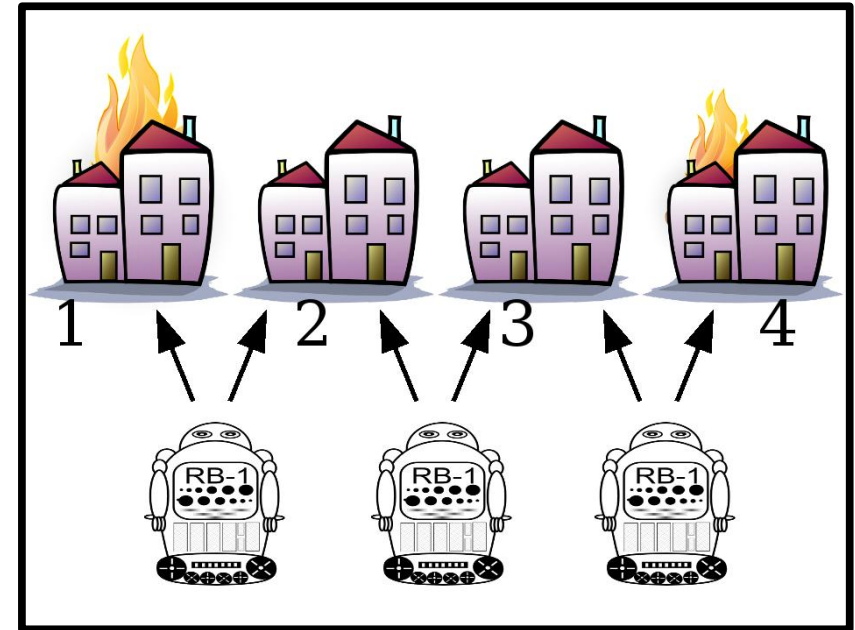
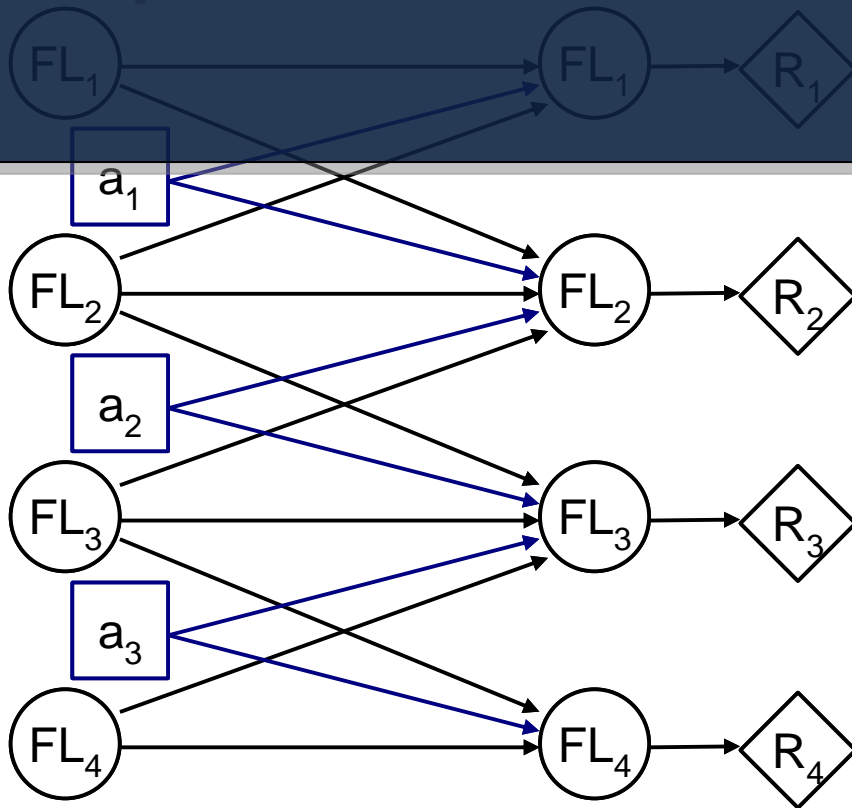
■ Factored Dec-POMDPs

[Oliehoek et al. 2008 AAMAS]



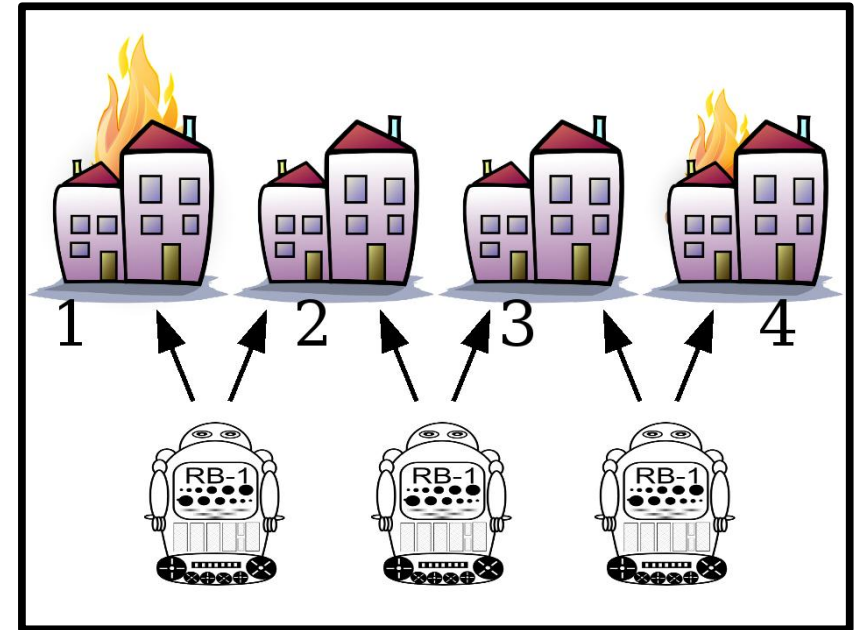
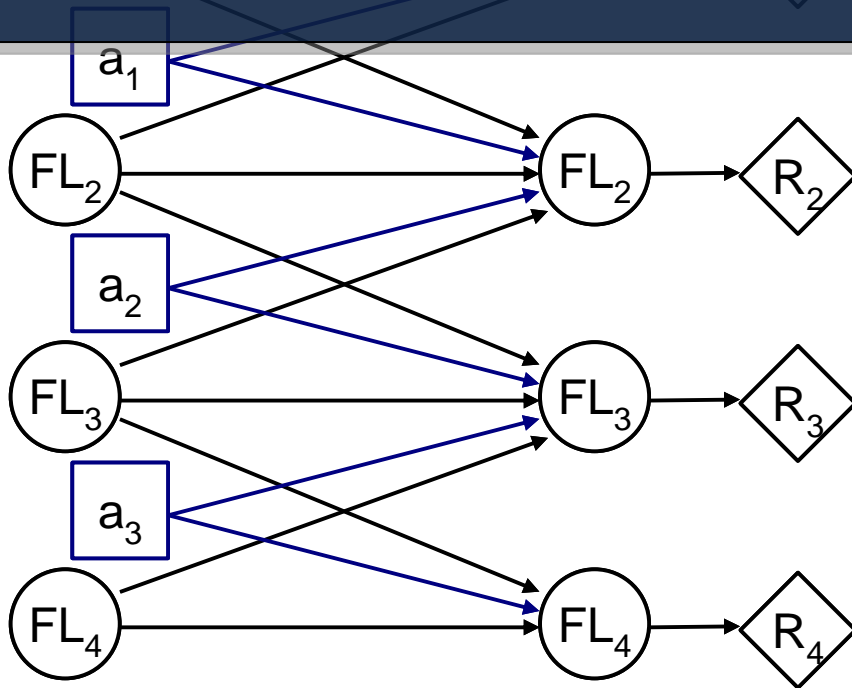
Locality in General fDec-POMDPs

- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
- Why ?



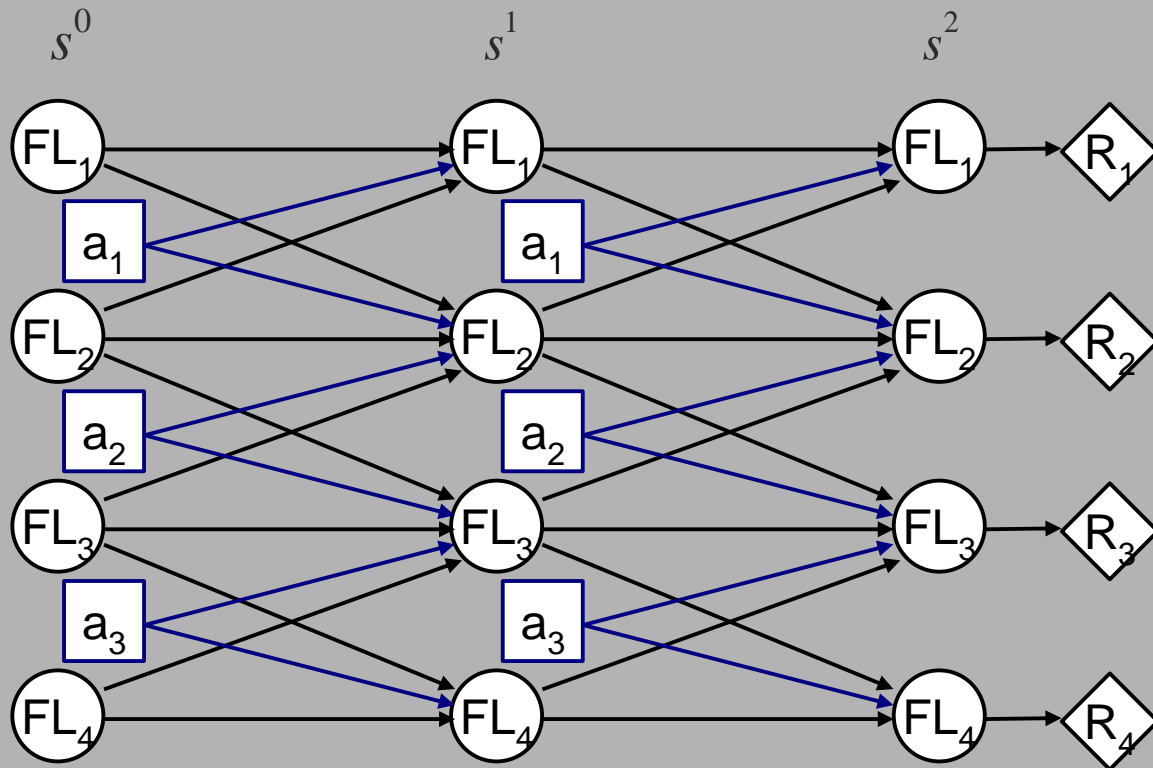
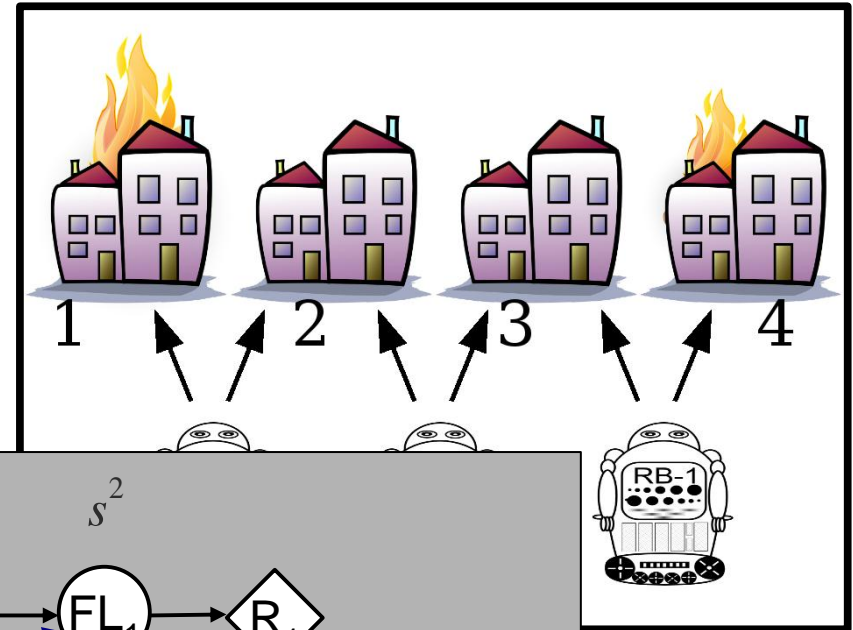
Locality in General fDec-POMDPs

- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
 - Why not ?
 - dependence propagates!



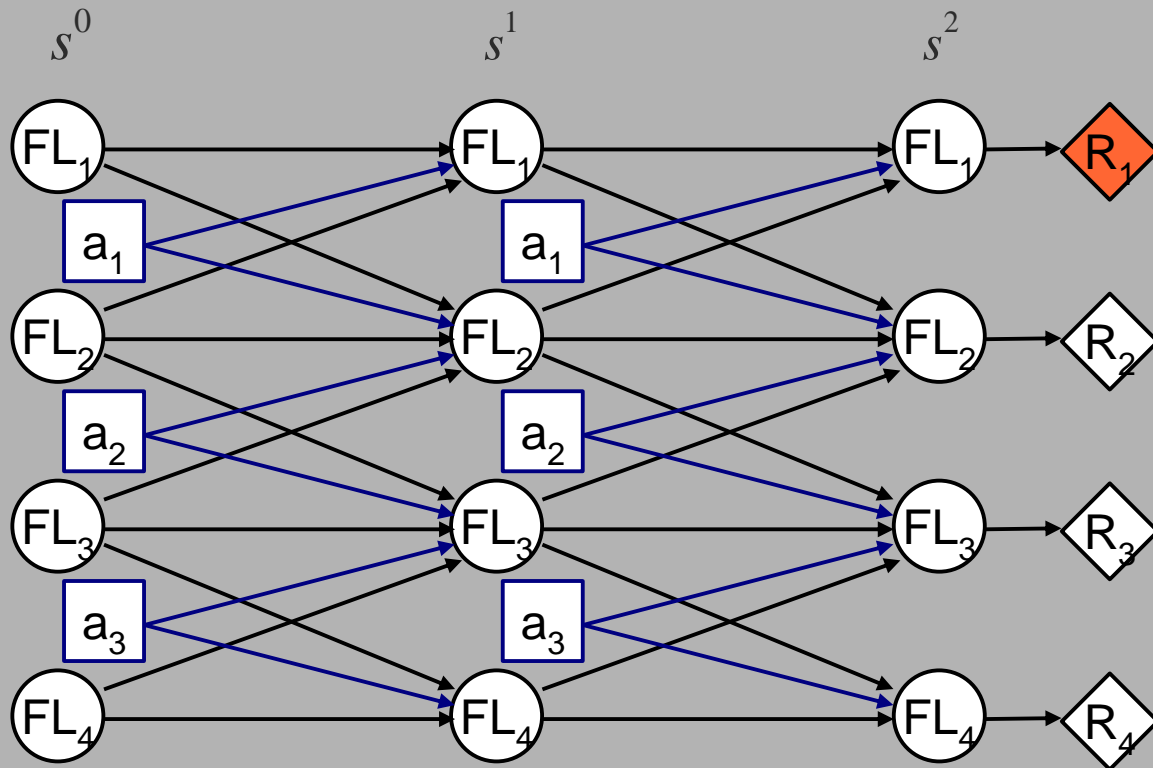
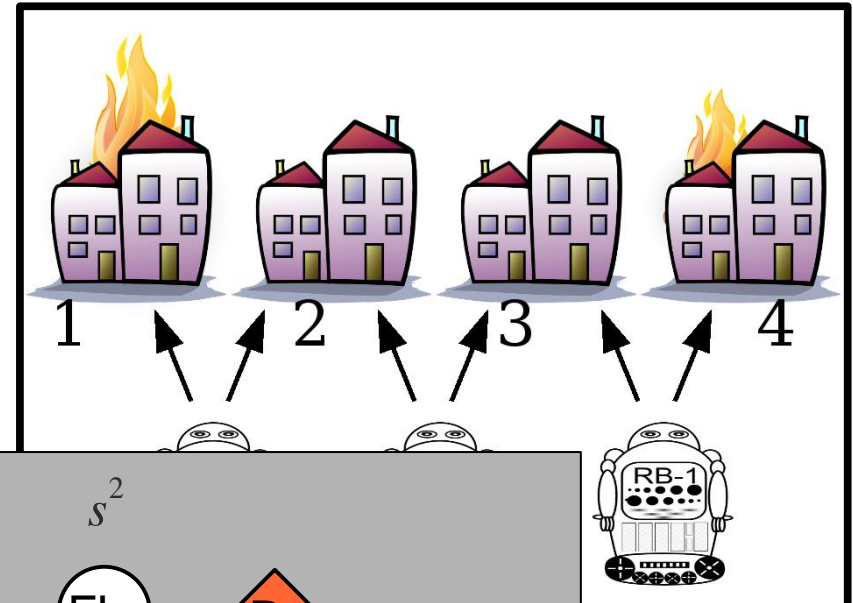
Locality in General fDec-POMDPs

- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
- Why not?
→ dependence propagates!



Locality in General fDec-POMDPs

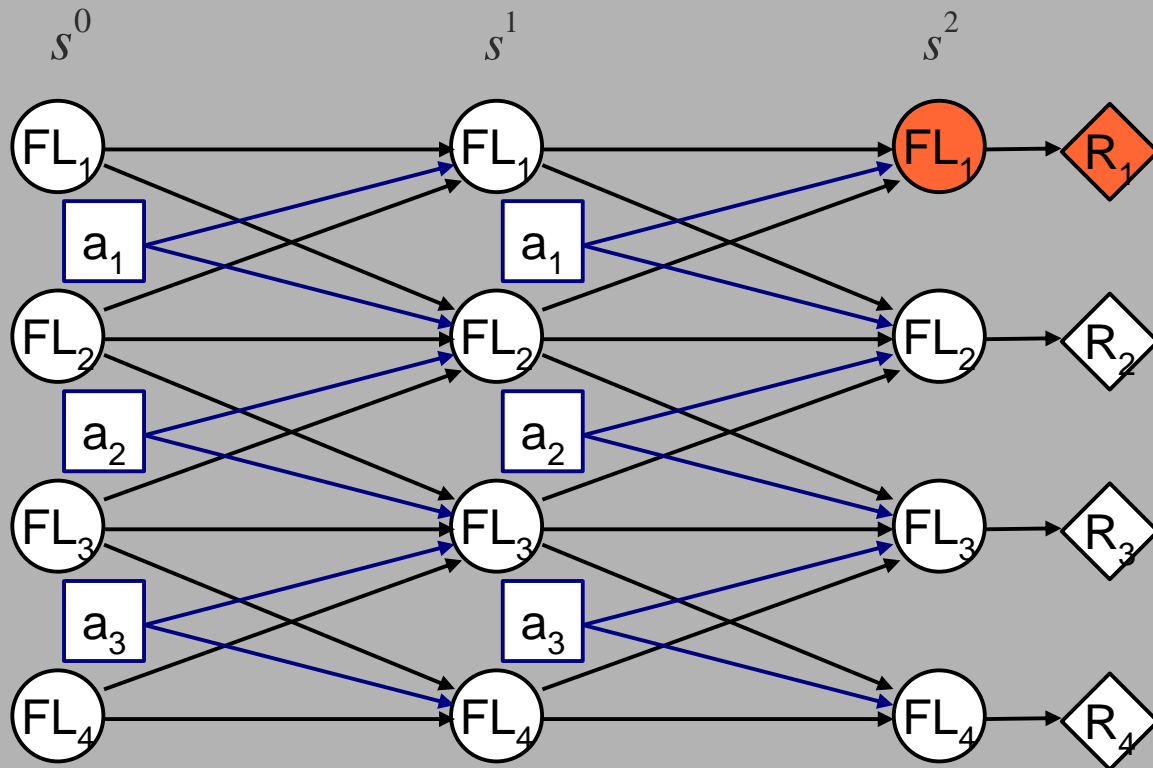
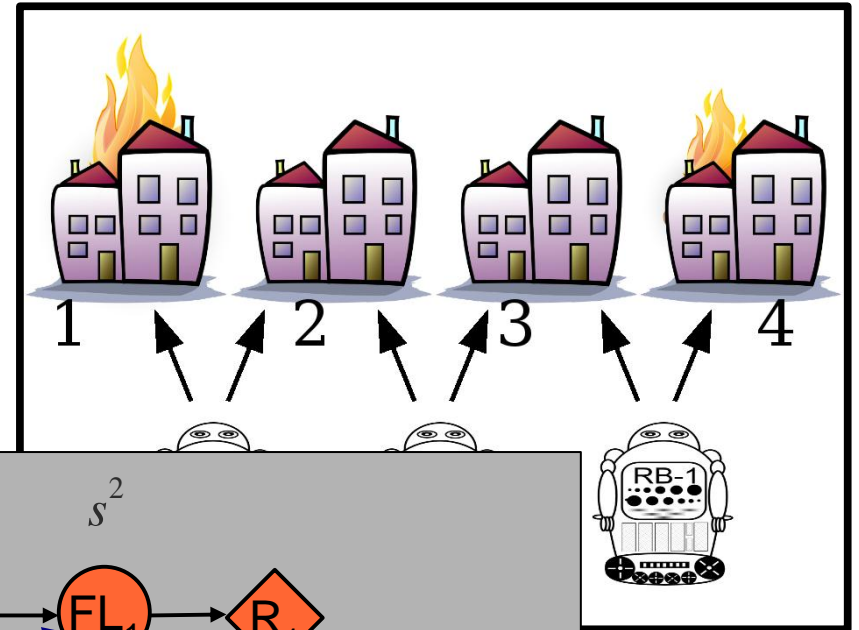
- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
- Why not?
→ dependence propagates!



what influences
 R_1^2 ?

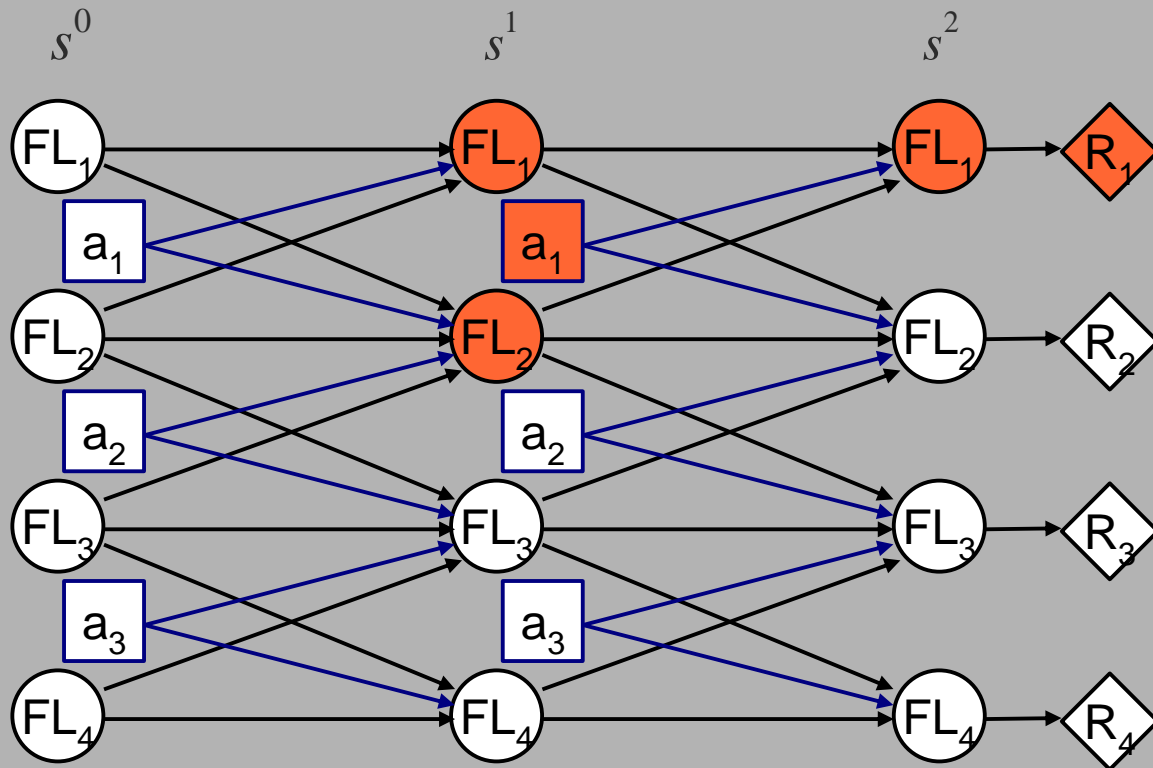
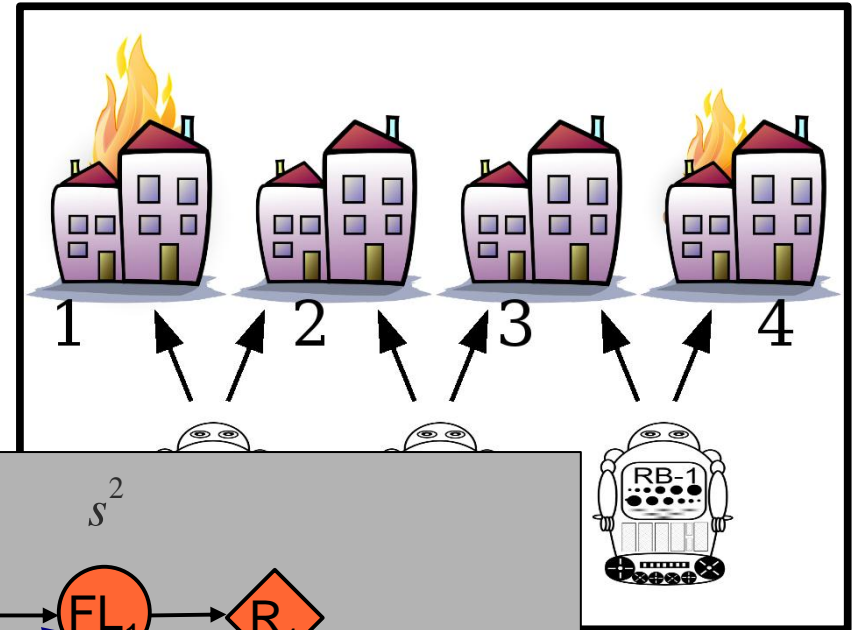
Locality in General fDec-POMDPs

- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
- Why not?
 - dependence propagates!



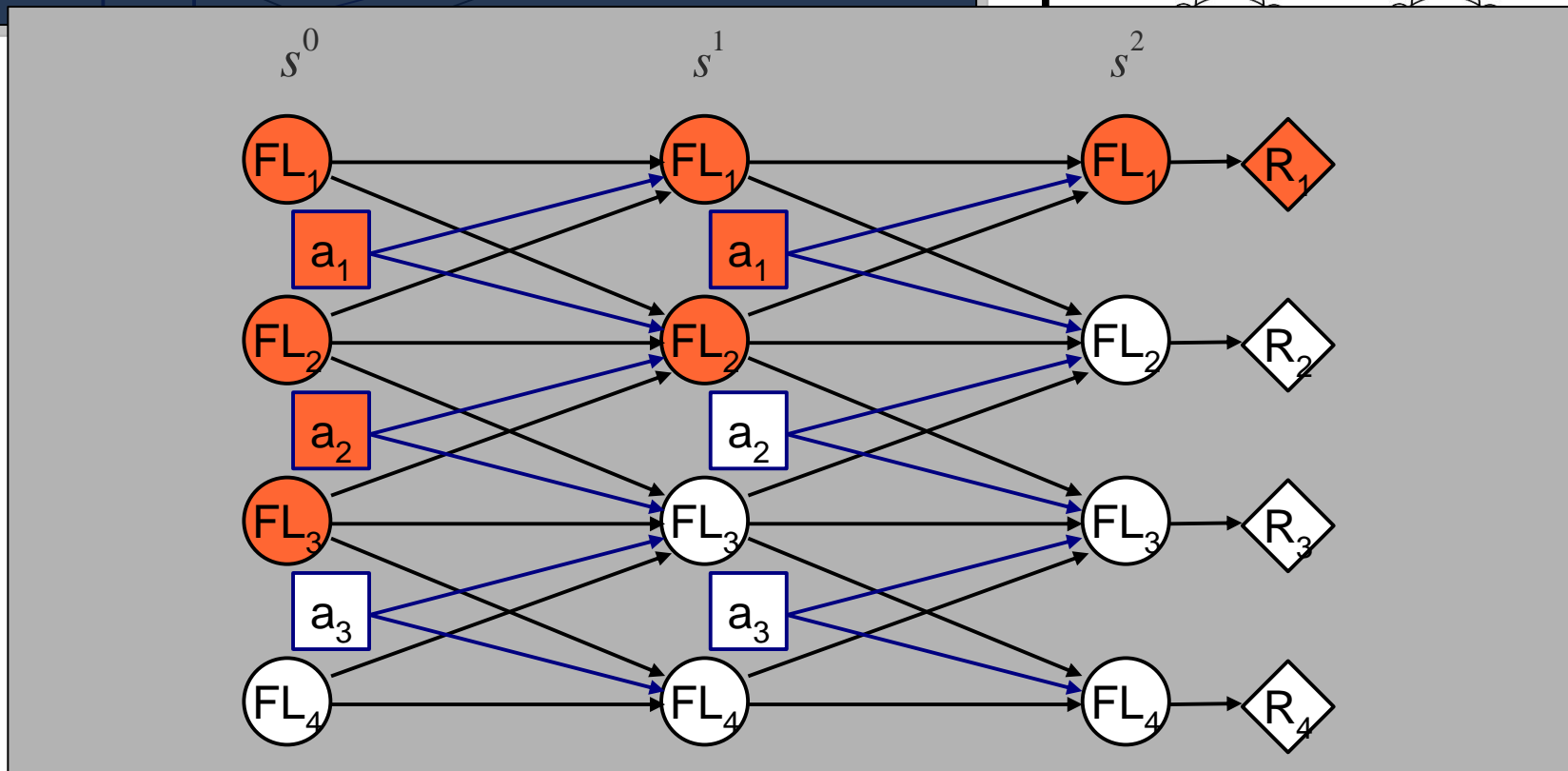
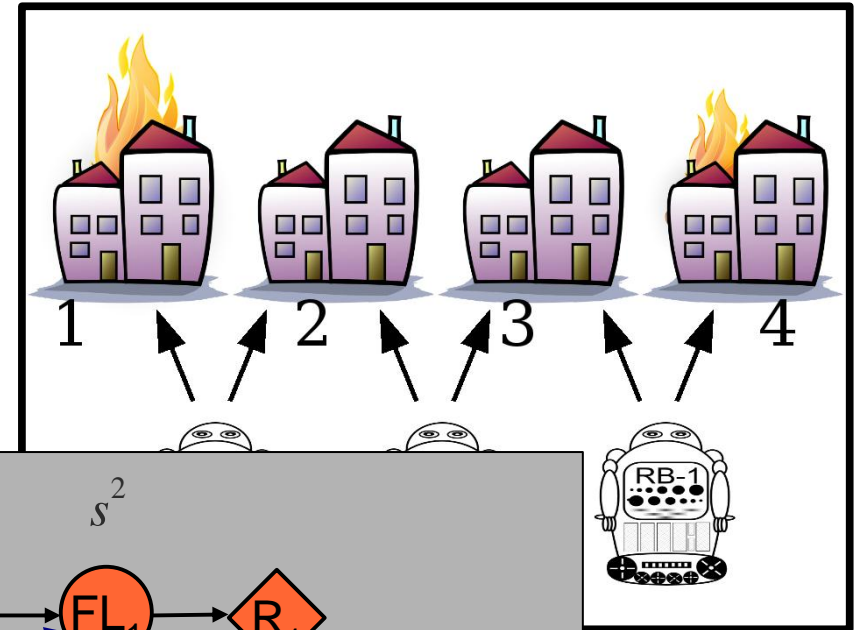
Locality in General fDec-POMDPs

- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
- Why not?
→ dependence propagates!



Locality in General fDec-POMDPs

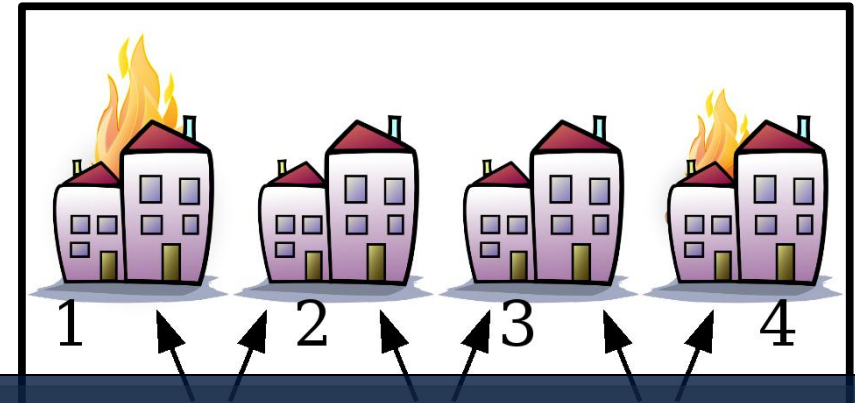
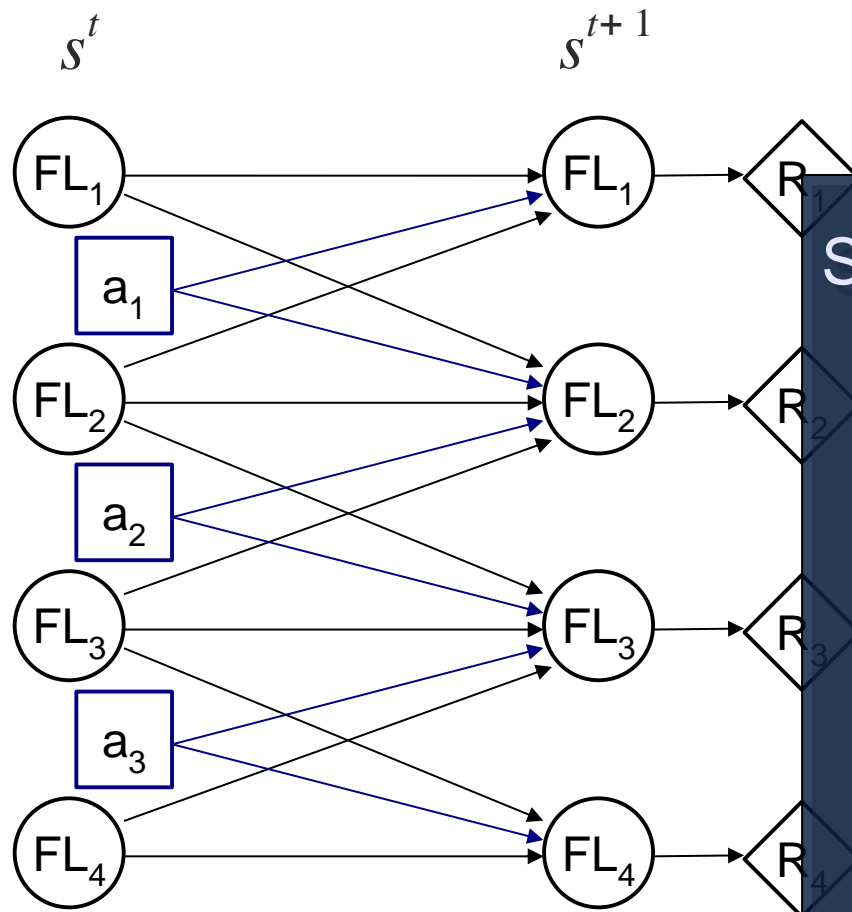
- Factored Dec-POMDPs
- Can't we use the previous methods (reduction to DCOP) directly...
- Why not?
→ dependence propagates!



Locality in General fDec-POMDPs

■ Factored Dec-POMDPs

[Oliehoek et al. 2008 AAMAS]



Solution Methods

- reduction to a type of COP
- but now: one for each stage!



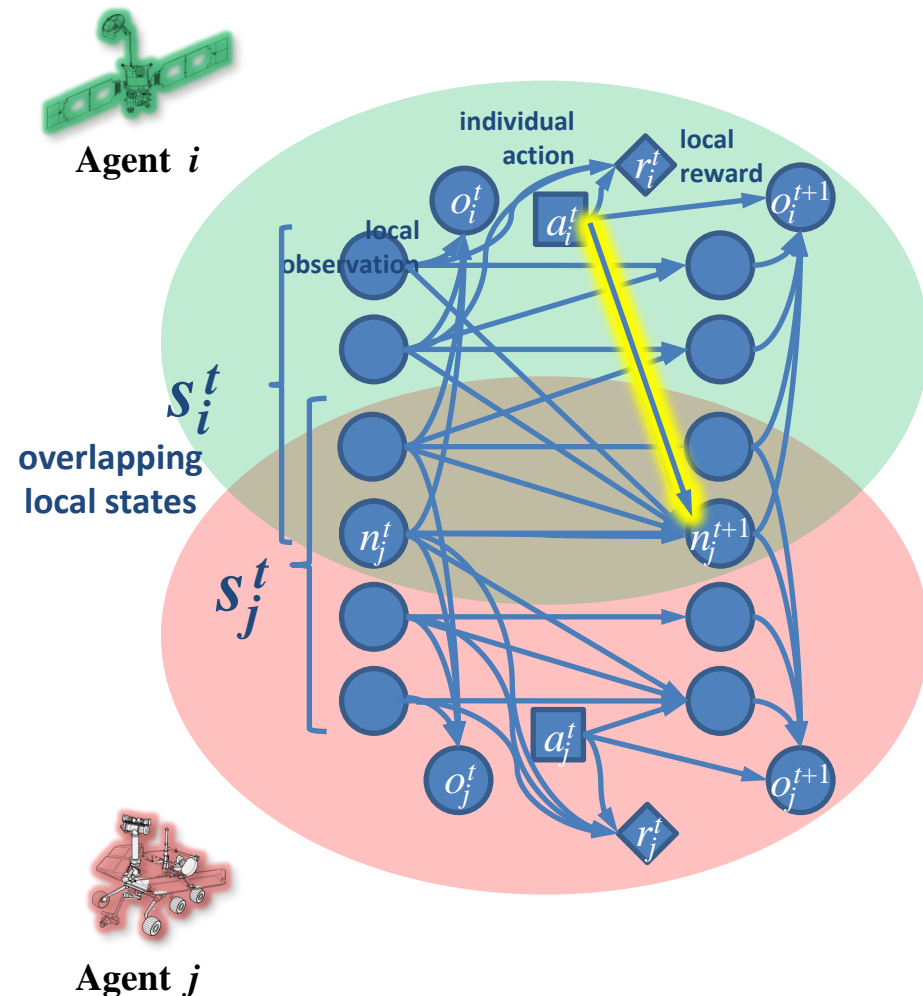
- δ is a decision rule
(part of policy for 1 stage t)

→ leads to factored form of heuristic search
[Oliehoek 2010 PhD, Oliehoek et al AAMAS 2013]

Decoupling Transition-Dependent Problems

TD-POMDP [Witwicki & Durfee ,2010, Witwicki PhD 2011]

- Transition-dependent interactions captured via shared features...
 - nonlocal feature n_j
 - controlled by another agent
 - affects subsequent transitions of other features in agent j 's local state
 - also affects j 's observations
- **Agent can compute best response using a *local* POMDP augmented with histories of the shared state features**

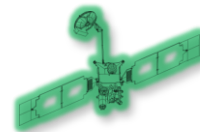


Best-F
M

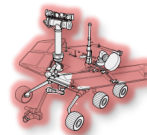
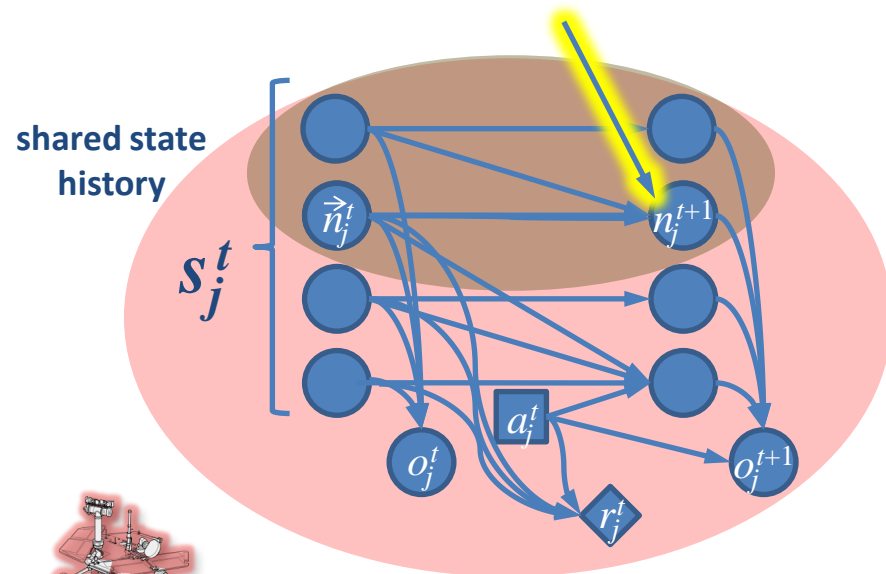
Decoupling Transition-Dependent Problems

TD-POMDP [Witwicki & Durfee ,2010, Witwicki PhD 2011]

- Transition-dependent interactions captured via shared features...
 - nonlocal feature n_j
 - controlled by another agent
 - affects subsequent transitions of other features in agent j 's local state
 - also affects j 's observations
- **Agent can compute best response using a *local* POMDP augmented with histories of the shared state features**



Agent i



Agent j

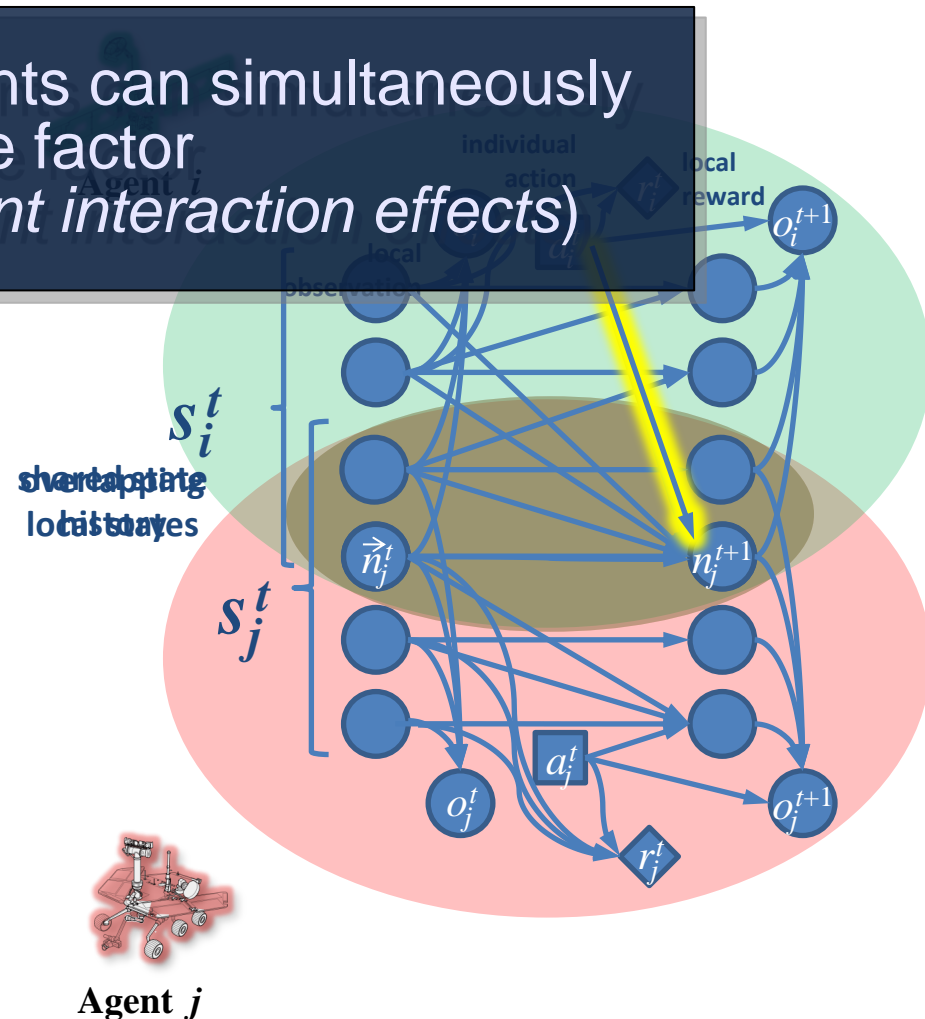
Best-F
M

Decoupling Transition-Dependent Problems

TD-POMDP [Witwicki & Durfee, 2010, Witwicki PhD 20013]

- Transition-dependent interactions captured via shared features...
 - **Caveat:** No two agents can simultaneously affect the same state factor (called *nonconcurrent interaction effects*)

- nonlocal feature n_j
 - controlled by another agent
 - affects subsequent transitions of other features in agent j 's local state
 - also affects j 's observations
- **Agent can compute best response using a *local* POMDP augmented with histories of the shared state features**

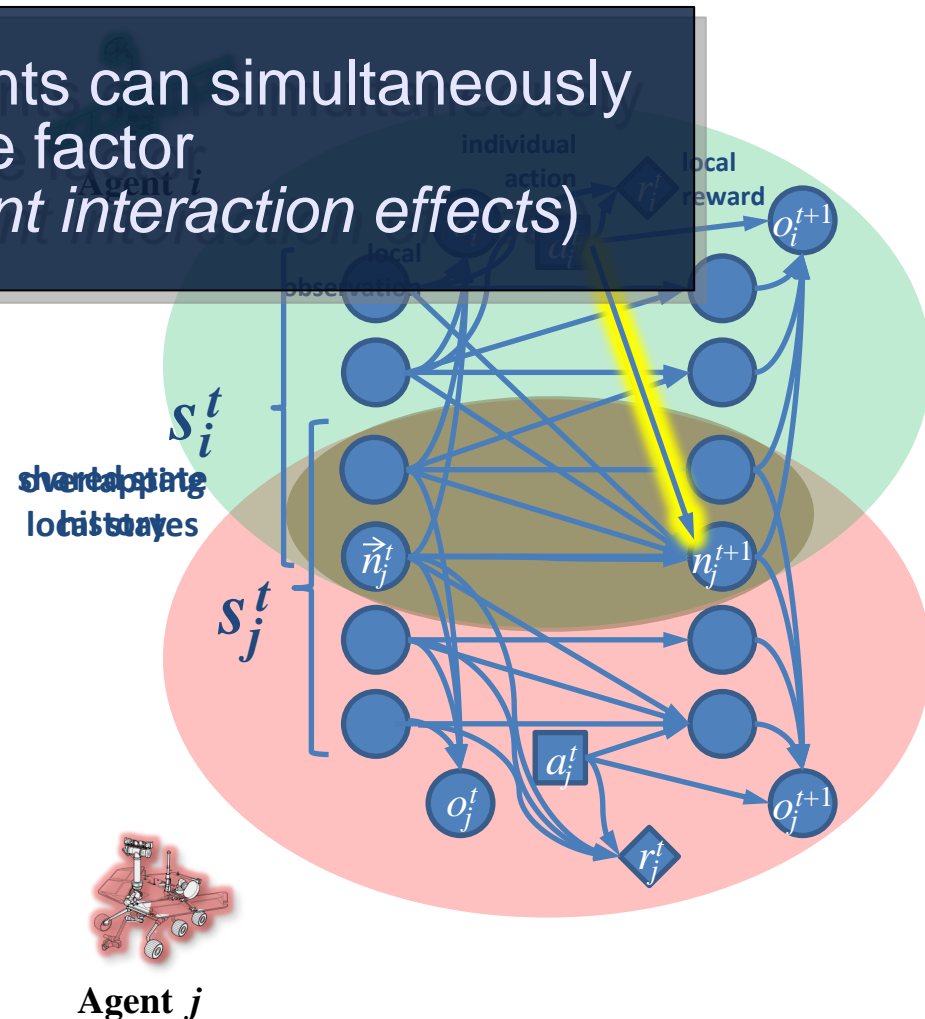


Decoupling Transition-Dependent Problems

TD-POMDP [Witwicki & Durfee ,2010, Witwicki PhD 20013]

- Transition-dependent interactions captured via shared features...
 - **Caveat:** No two agents can simultaneously affect the same state factor (called *nonconcurrent interaction effects*)

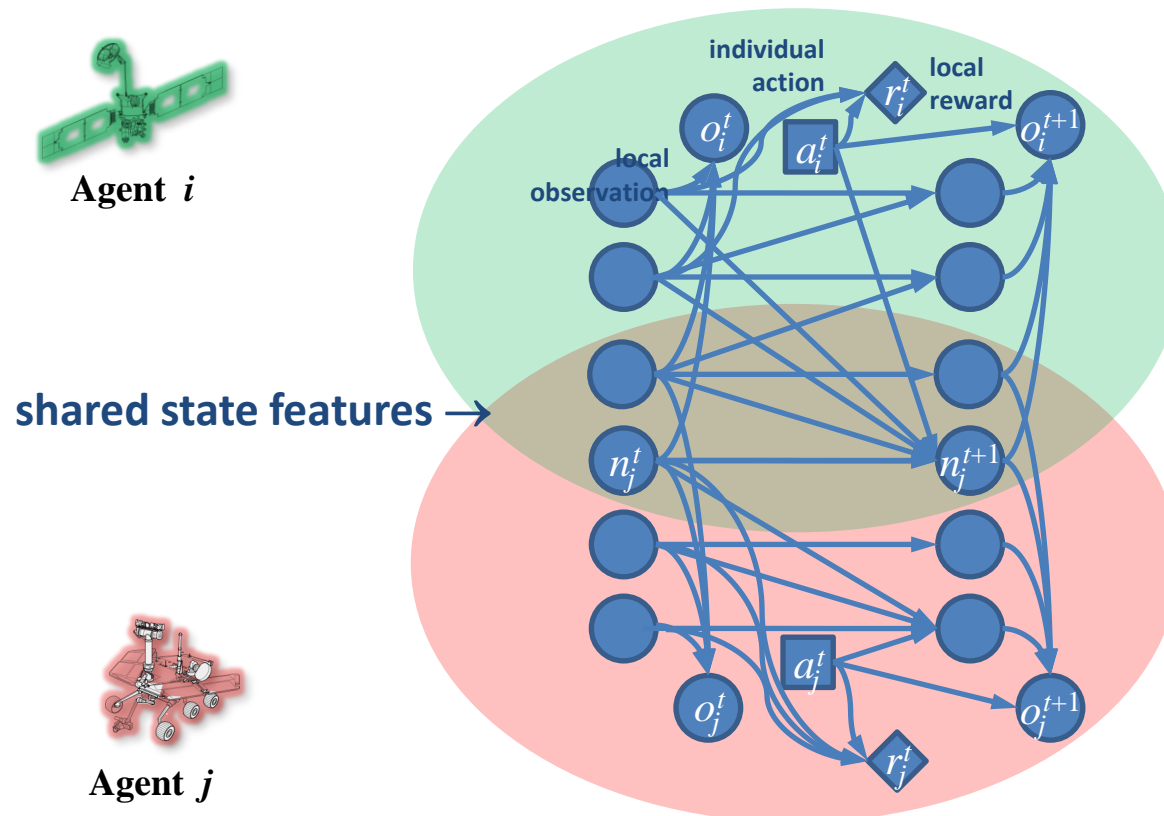
- nonlocal feature n_j
 - controlled by another agent
 - affects subsequent transitions of other features in agent j 's local state
 - also affects j 's observations
- **Agent can compute best response using a *local* POMDP augmented with histories of the shared state features**



Influence-Based Abstraction

[Witwicki & Durfee, 2010, Oliehoek *et al.* 2012]

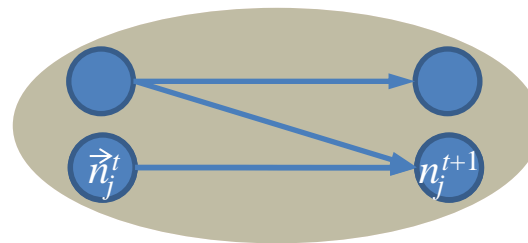
- Factor problem into overlapping **local states**
- Infer how the local state is affected by other agents' policies
 - Different policies may have different affects
 - Each possible affect is an **influence**



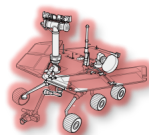
Influence-Based Abstraction

[Witwicki & Durfee, 2010, Oliehoek *et al.* 2012]

- Factor problem into overlapping **local states**
- Infer how the local state is affected by other agents' policies
 - Different policies may have different affects
 - Each possible affect is an **influence**



← Influence DBN

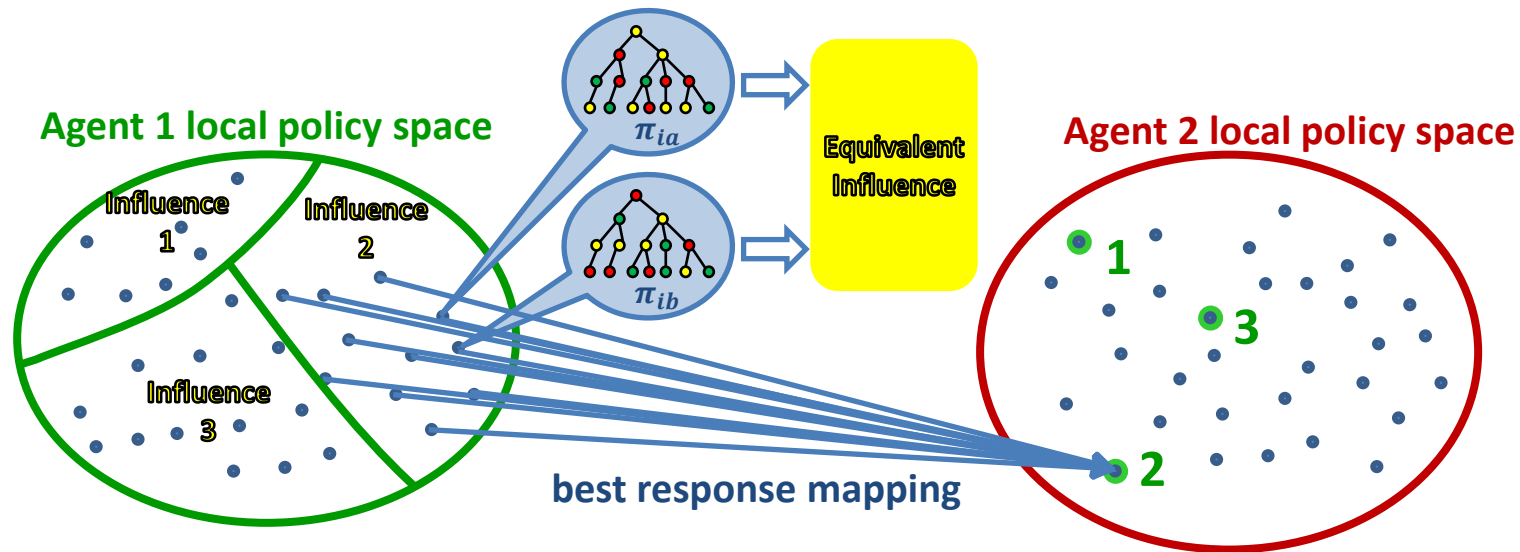


Agent j

Influence-Based Abstraction

[Witwicki & Durfee, 2010, Oliehoek *et al.* 2012]

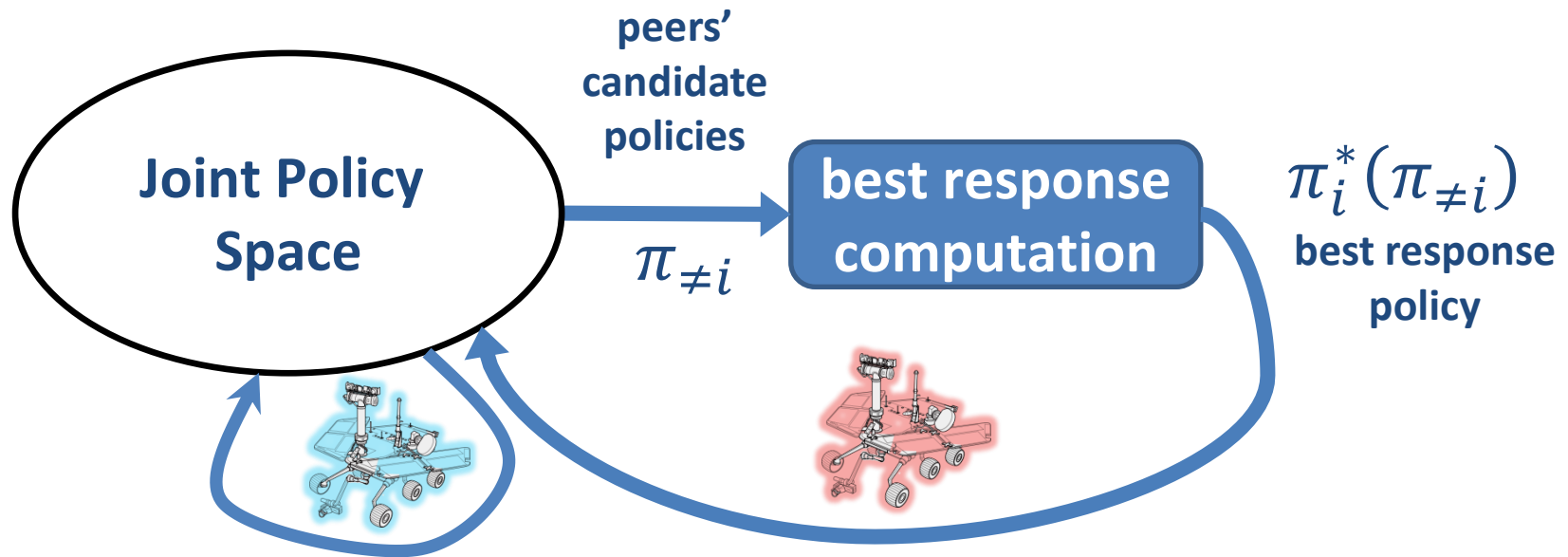
- Factor problem into overlapping **local states**
- Infer how the local state is affected by other agents' policies
 - Different policies may have different affects



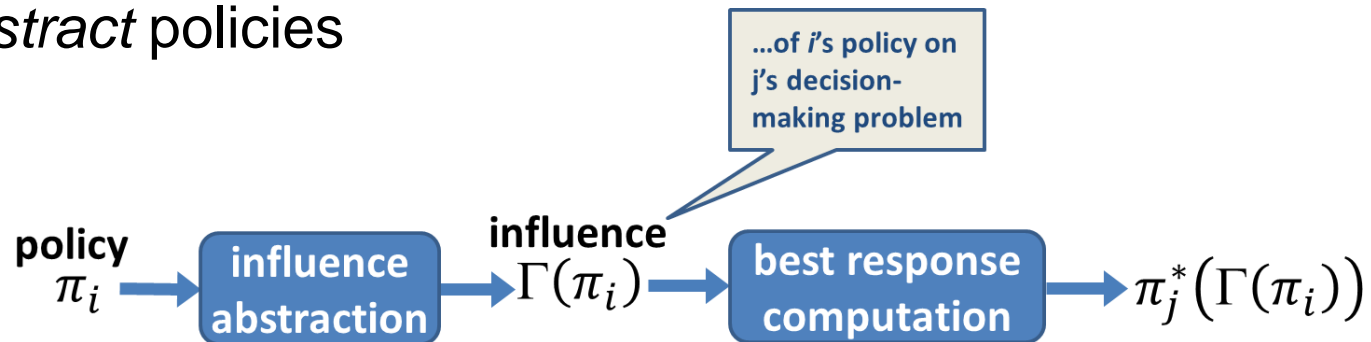
- But many policies may yield the same influence!

Influence-Space Search

Decoupled policy space search:

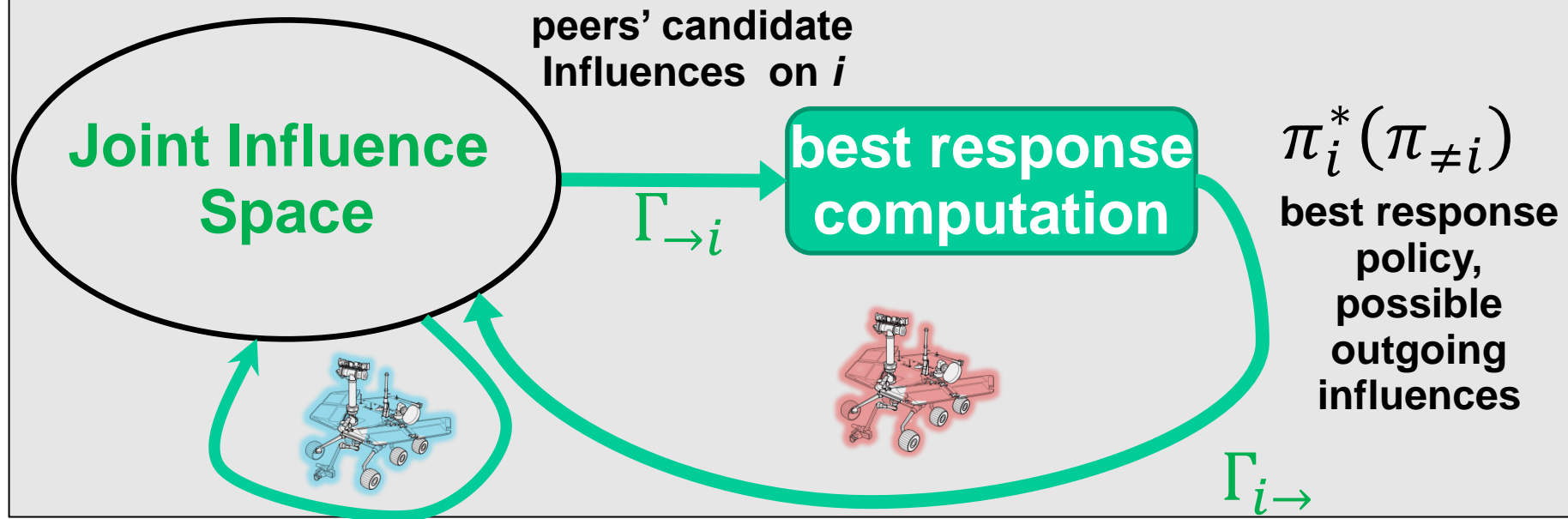


Influences *abstract* policies

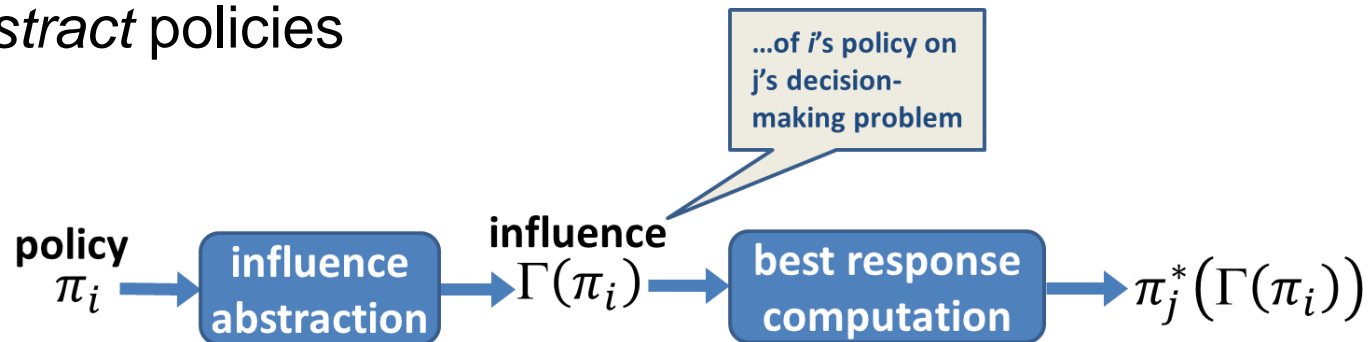


Influence-Space Search

De So why not search directly in the influence space?



Influences *abstract* policies



Influence-Space Search

- For TI-Dec-MDPs
 - Coverage Set Algorithm (CSA) [Becker *et al.*, JAIR 2004]
- For EDI-Dec-MDPs (transition-dependent variant of TI-Dec-MDPs)
 - CSA' [Becker *et al.*, AAMAS 2004]
 - Commitment-Driven Distributed Joint Policy Search [Witwicki & Durfee, MSDM 2009]
- For TD-POMDPs
 - OIS [Witwicki & Durfee, AAMAS 2010]
- For general fDec-POMDPs
 - Coming soon!

Efficiency Improvements of Influence-Space Search

[Witwicki 2011]

Varying the size of the window of interaction...

○ Optimal Influence-space Search (OIS)

orders of magnitude speedup for small interaction windows

▽ SPIDER [Varakantham *et al.* 2007]

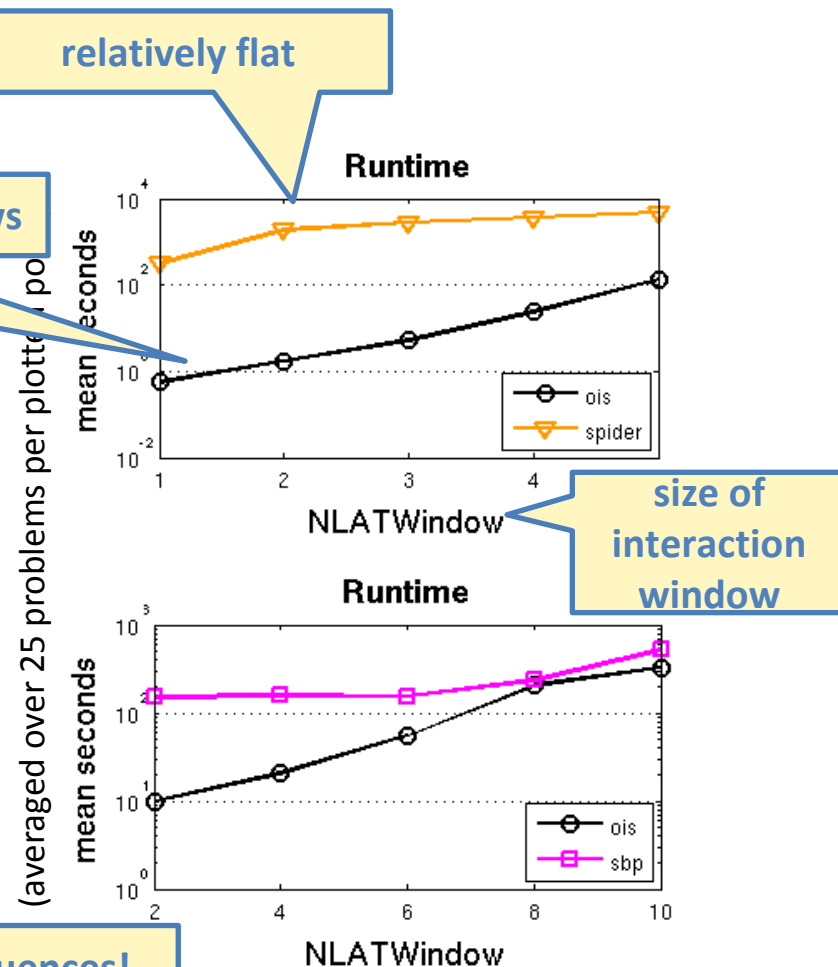
- Policy-space search algorithm, implemented for 2-agent problems with *task enablement* interactions
- Reduces search space via pruning

□ Separable Bilinear Programming

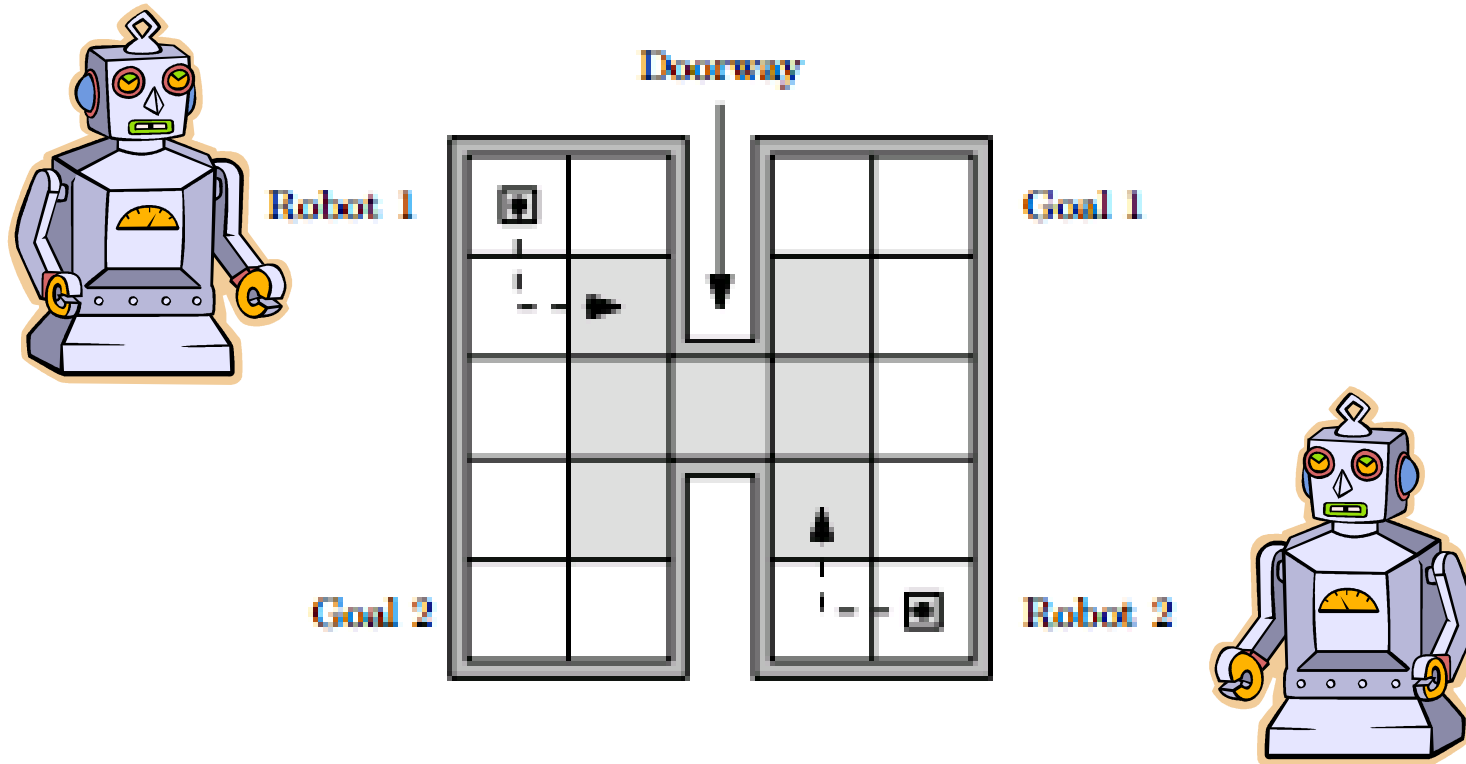
[Mostafa & Lesser 2009, Petrik & Zilberstein 2009]

- centralized approach that exploits structure in a bilinear programming formulation
- no decoupling of joint policy formulation

Neither SPIDER NOR SBP able to exploit constrained influences!



Context-Specific Influences



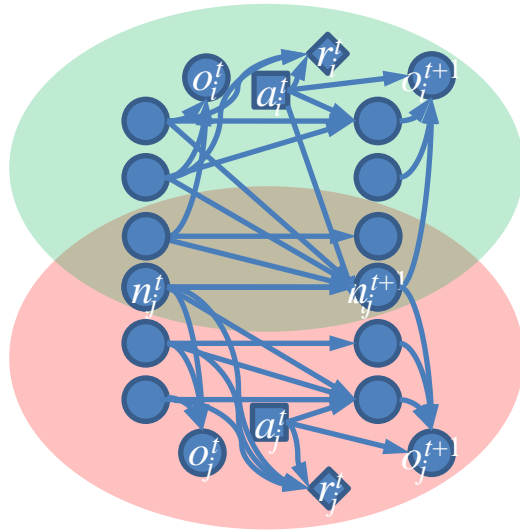
- Context-Specific Multiagent Planning with Factored MDPs [Guestrin *et al.* 2002]
- Interaction-Driven Markov Game [Spaan & Melo 2008]
- Distributed POMDP with Coordination Locales [Varakantham *et al.* 2009, Velagapudi *et al.* 20011]

Weak Coupling Hypothesis

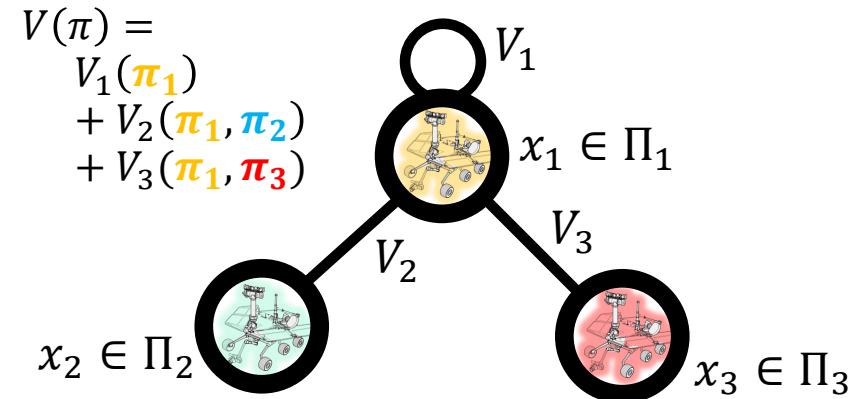
- Problems wherein agents are “weakly-coupled” should be...
 - Easier to solve
 - Accommodate more scalable solution techniques

Aspects of Weak Coupling

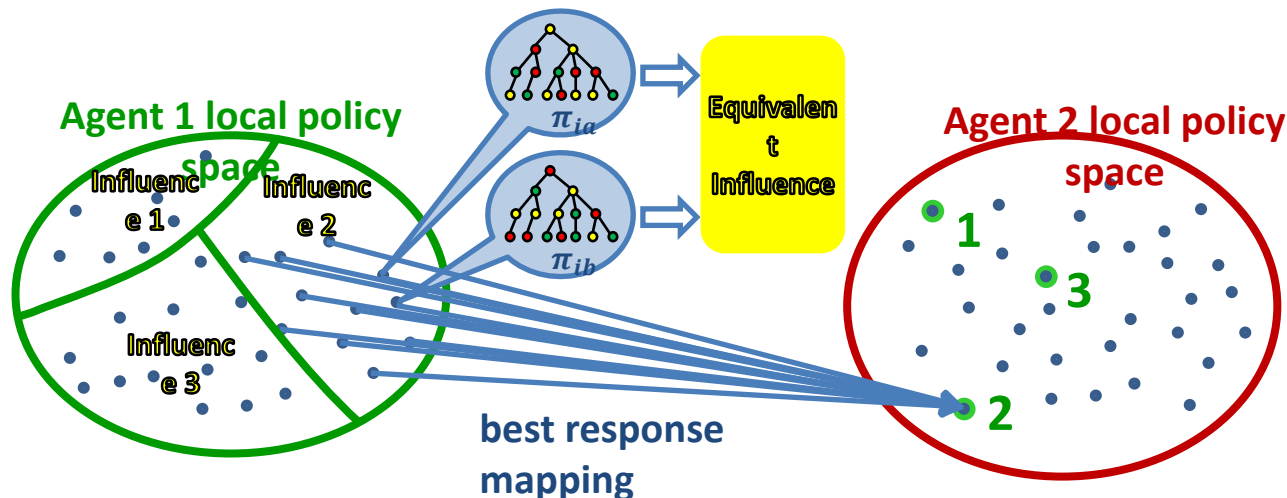
1. Graphical structure present at state factor level



2. Graphical structure present at agent interaction level



3. Underlying influence Structure



Characterization of Weakly Coupling

[Witwicki & Durfee 2011]

Dimensions of Weak Coupling

ω [agent scope size, manifested as induced width]

→ Number of best responses required

$d_{\mathbb{P}}$ [degree of influence]

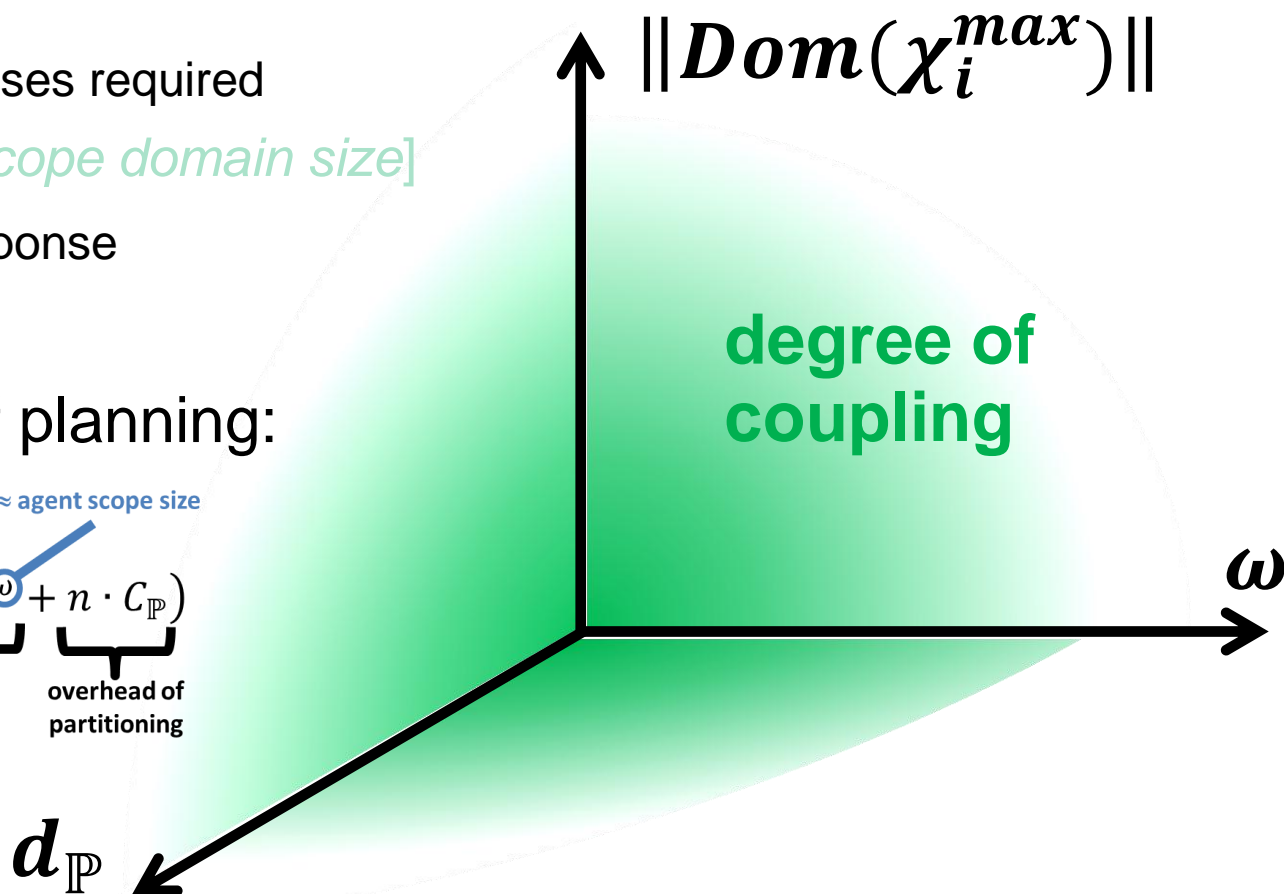
→ Number of best responses required

$Dom(\chi_i^{max})$ [state factor scope domain size]

→ Complexity of best response

Worst-case complexity of planning:

$$O(\underbrace{\text{EXP}(\|Dom(\chi_i^{max})\|)}_{\text{complexity of best response}} \cdot \underbrace{n \cdot (d_{\mathbb{P}} \|\Pi_i^{max}\|)}_{\text{number of best response computations}} \underbrace{\omega}_{\approx \text{agent scope size}} + \underbrace{n \cdot C_{\mathbb{P}}}_{\text{overhead of partitioning}})$$



Summary

Assumptions

- Factorization of joint problem into local states (LS)
- Local full observability (LFO)
- Transition and Observation Independence (TOI)
- Context-Specific Transition and Observation dependence (CS)
- Event-driven Interactions (EDI)
- Nonconcurrent Interaction Effects (NIE)
- Reward Decomposition (RD)
- Locality of Interaction (LI)
- Hierarchical structure among agents' tasks with constraints on ordering (OC-Dec-MDPs) [Beynier & Mouaddib 2005, Marecki & Tambe 2007]
- State-dependent action sets [Guo & Lesser 2005]
- Resource-constrained interactions [Dolgov & Durfee 2006, Wu & Durfee 2010]

Representations

- fDec-POMDP
- TI-Dec-MDP (LS, LFO, TOI, EDI, RD), EDI-Dec-MDP/EDI-CR-Dec-MDP (LS, LFO, EDI, RD)
- ND-POMDP (LS, TOI, RD, LI)
- TD-PODMP (LS, NIE, RD, LI)
- Multiagent Factored MDP (LS, LFO), IDMG (LS, RD, LI), DPCL (LS, RD, LI)

For pointers to more, see [Witwicki & Durfee AAMAS 2011]

Summary-2

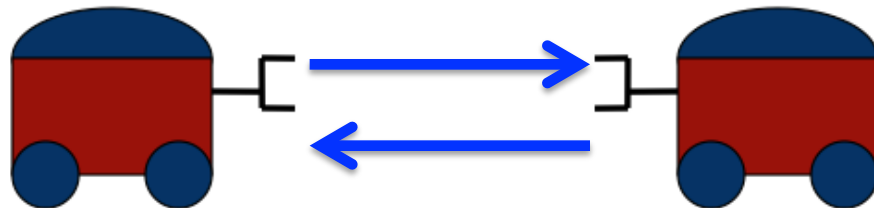
Solution Paradigms

- Decoupled Joint Policy Formulation
 - JESP and almost all others covered in this section
- COP-based Methods
 - LID-JESP, SPIDER, CBDP, CGBG-based heuristic search
- Influence Abstraction
 - CSA, OIS, Commitment-driven joint policy search, TREMOR

Other Topics

Communication

- Communication can be implicitly represented in Dec-POMDP model
- Free and instantaneous communication is equivalent to centralization
- Otherwise, need to reason about what and when to communicate



Dec-POMDP-COM

- A DEC-POMDP-COM can be defined with the tuple:

$$M = \langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O, \Sigma, C_\Sigma \rangle$$

- The first parameters are the same as a Dec-POMDP
- Σ , the alphabet of atomic communication messages for each agent (including a null message)
- C_Σ , the cost of transmitting an atomic message: $C_\Sigma : \Sigma \rightarrow \mathfrak{R}$
- R , the reward model integrates this cost for the set of agents sending message σ : $R(s, \vec{a}, \vec{\sigma})$
- Explicitly models communication
- Same complexity as Dec-POMDP

Algorithms using communication

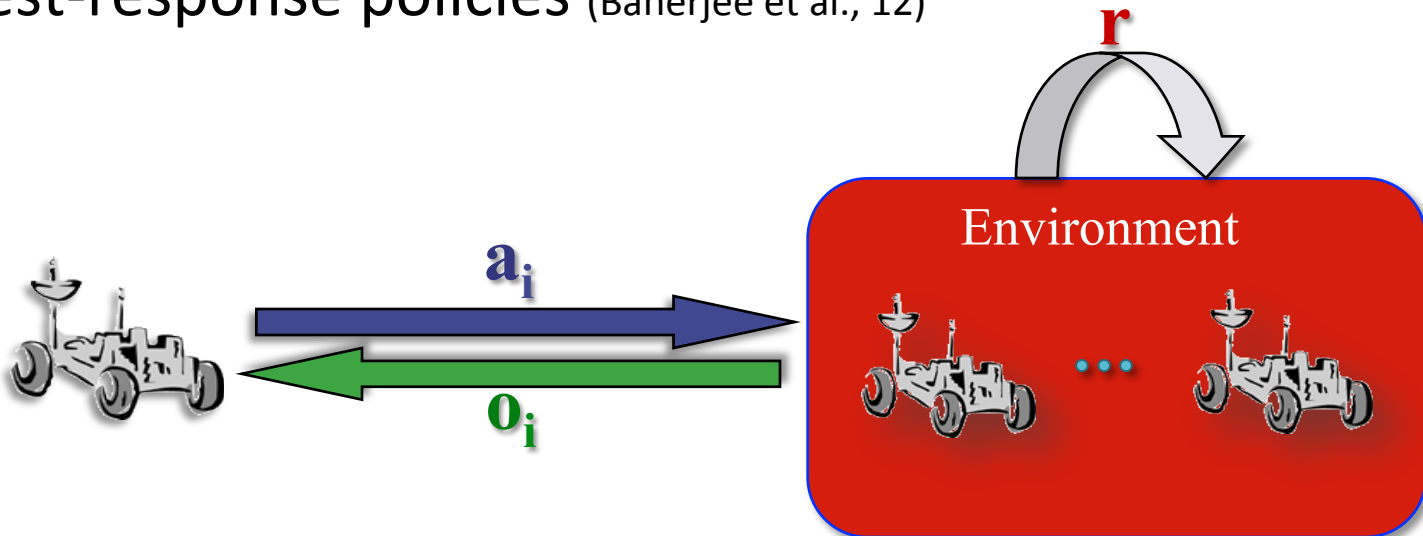
- Analysis of possible communication models and complexity (Pynadath et al., 02)
- Myopic communication in transition independent Dec-MDPs (Becker et al., 09)
- Reasoning about run-time communication decisions (Nair et al., 04; Roth et al., 05)
- Stochastically delayed communication (Spaan et al., 08)

Learning in Dec-POMDPs

- Hard to solve, even if you know the model, but there are some good approximate solution methods
- What if the model is not known?
- There are currently very few RL methods that can be used (multiagent, partially observable)

Related learning methods

- Model-free reinforcement learning methods using gradient-based methods to improve the policies (Dutech et al., 01; Peshkin et al., 00)
- Learning using local signals and modeling the remaining agents as noise (Chang et al., 04)
- Communication and sample-based planning to generate best-response policies (Banerjee et al., 12)



References

- Raphen Becker, Alan Carlin, Victor Lesser, and Shlomo Zilberstein. Analyzing myopic approaches for multi-agent communications. *Computational Intelligence*, 25(1): 31—50, 2009.
- Claudia V. Goldman and Shlomo Zilberstein. *Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis*. *Journal of Artificial Intelligence Research* Volume 22, pages 143-174, 2004.
- Ranjit Nair, Milind Tambe, Maayan Roth, and Makoto Yokoo. Communication for improving policy computation in distributed POMDPs. *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.
- David V. Pynadath and Milind Tambe. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research (JAIR)*, Volume 16, pp. 389-423. 2002.
- Maayan Roth, Reid G. Simmons and Manuela M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2005
- Matthijs T. J. Spaan, Frans A. Oliehoek, and Nikos Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Int. Conf. on Automated Planning and Scheduling*, pages 338--345, 2008.

References

- B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized POMDPs. In AAAI, 2012.
- Y.-H. Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In NIPS 16, 2004.
- A. Dutech, O. Buffet, and F. Charpillet. Multi-agent systems by incremental gradient reinforcement learning. In IJCAI, pages 833–838, 2001.
- L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In UAI, pages 489–496, 2000.

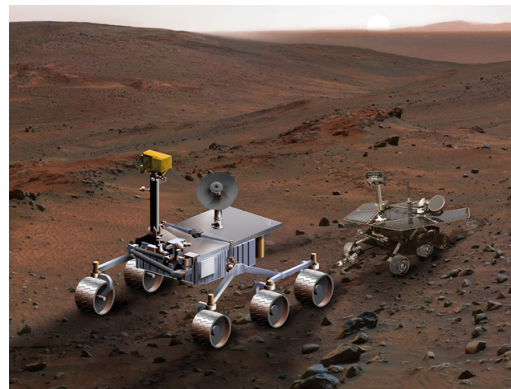
Applications

Applications

- Dec-POMDP very general model
- Many real-world problems have uncertainty and partial information
- Communication is often limited in some way
- Many Dec-POMDP applications have been discussed
- Recently, these applications have begun to be tested in the field

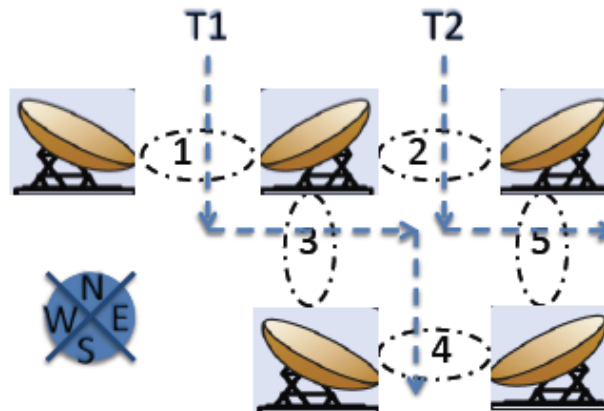
Example cooperative multiagent problems

- Multi-agent planning (examples below, reconnaissance, etc.)
- Human-robot coordination (combat, industry, etc.)
- Sensor networks (e.g. target tracking from multiple viewpoints)
- E-commerce (e.g. decentralized web agents, stock markets)



Sensor network problems

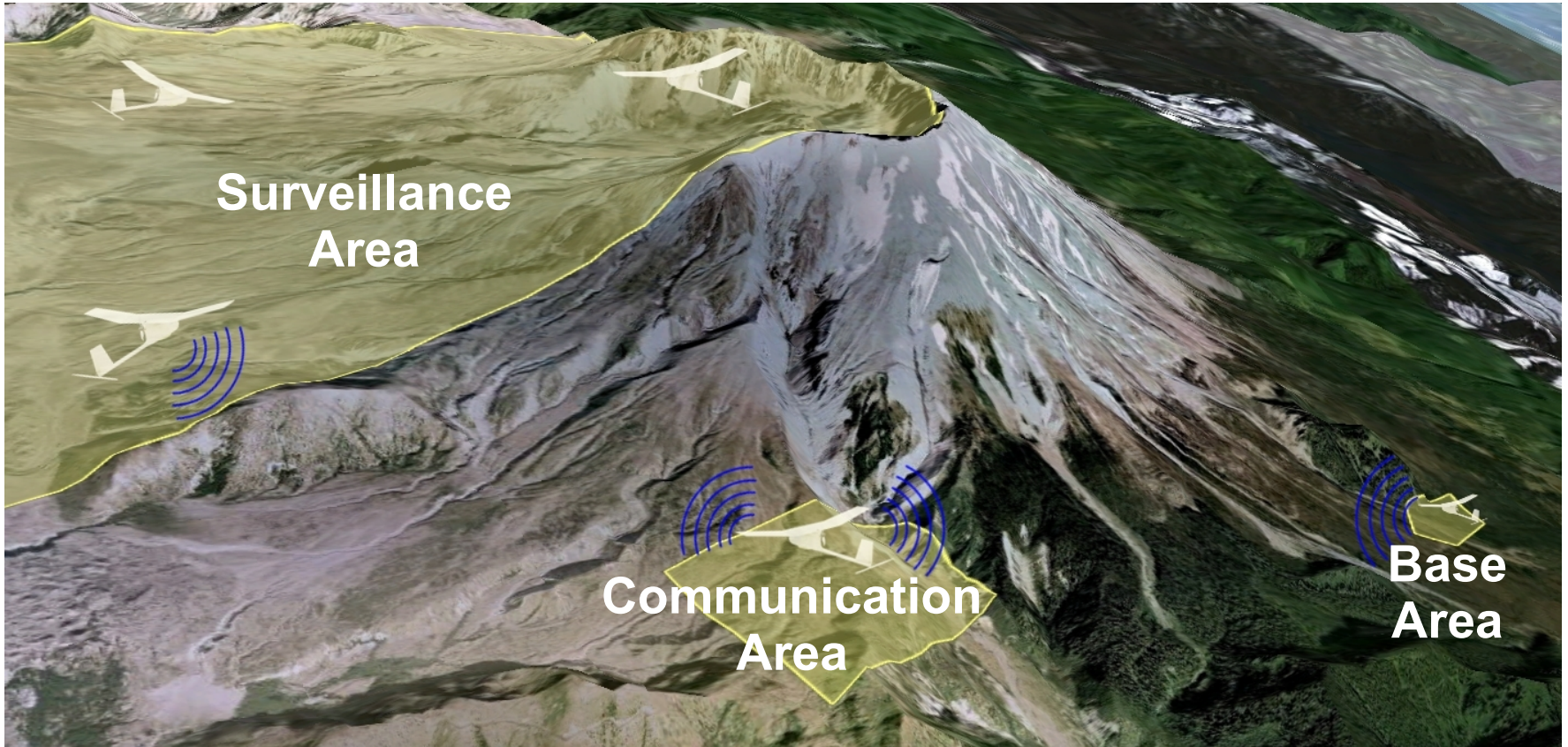
- Sensor networks for
 - Target tracking (Nair et al., 05, Kumar and Zilberstein 09-AAMAS)
 - Weather phenomena (Kumar and Zilberstein 09-IJCAI)
- Two or more cooperating sensors



Other possible application domains

- Multi-robot coordination
 - Space exploration rovers (Zilberstein et al., 02)
 - Helicopter flights (Pynadath and Tambe, 02)
 - Navigation (Emery-Montemerlo et al., 05; Spaan and Melo 08)
 - Foraging (Shi et al., 10)
 - Exploration (Matignon et al. 12)
- Load balancing for decentralized queues (Cogill et al., 04)
- Networks
 - Multi-access broadcast channels (Ooi and Wornell, 96)
 - Network routing (Peshkin and Savova, 02)
 - Wireless networking (Pajarinen and Peltonen, 11)
- Sensor network management (Nair et al., 05)

Persistent Surveillance Mission

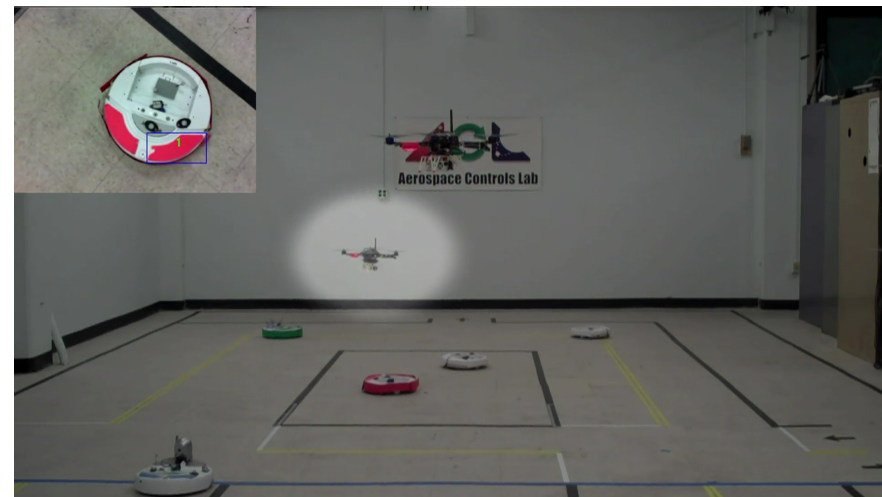
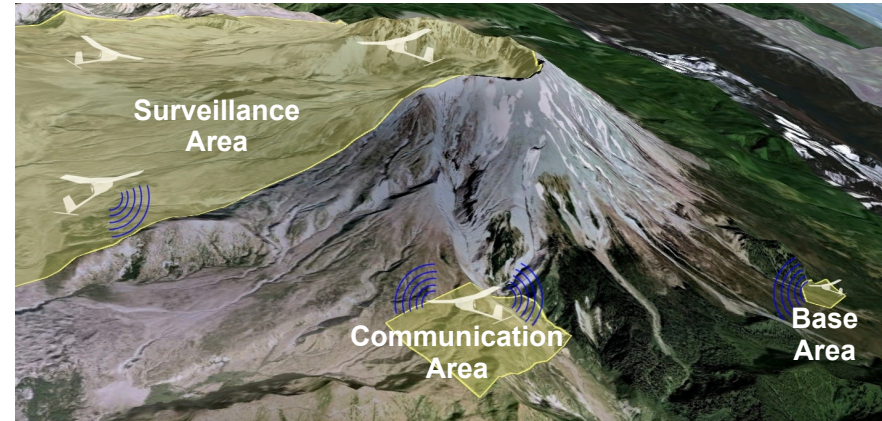


Slides courtesy of Josh Redding and the ACL group at MIT
See Ure et al. 2012



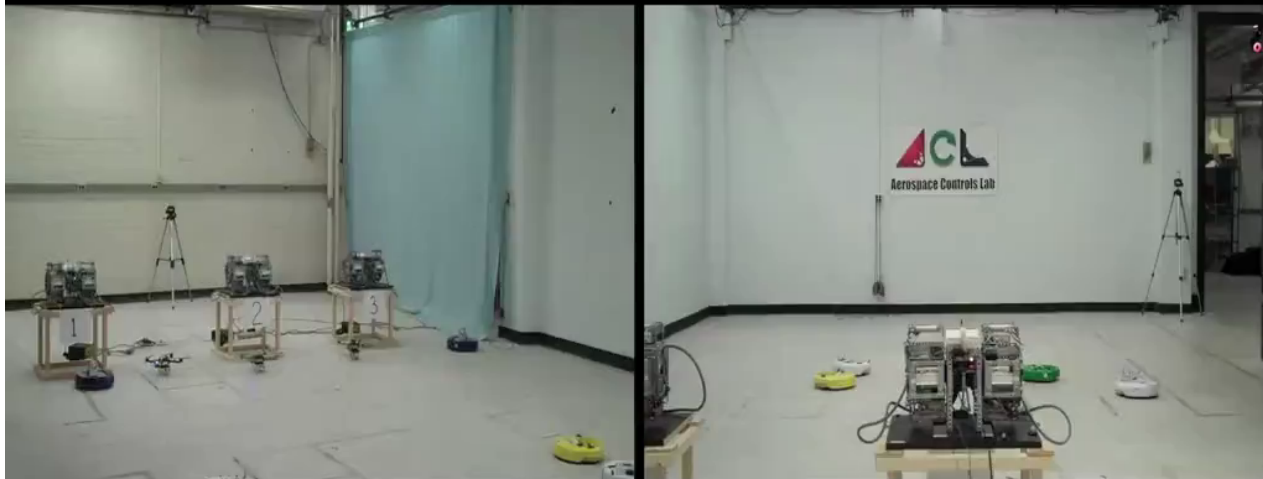
Persistent Surveillance Mission

- Goal: Team of n limited-fuel agents coordinate to maintain a persistent presence in the Surveillance and Communication areas
- Mobile targets/obstacles
- Uncertainty in agent models
 - Rate of fuel consumption
 - Probability of sensor failure
 - Probability of actuator degradation
- Challenge: Maintain presence, but return to base for refueling and repairs



Persistent Surveillance Mission

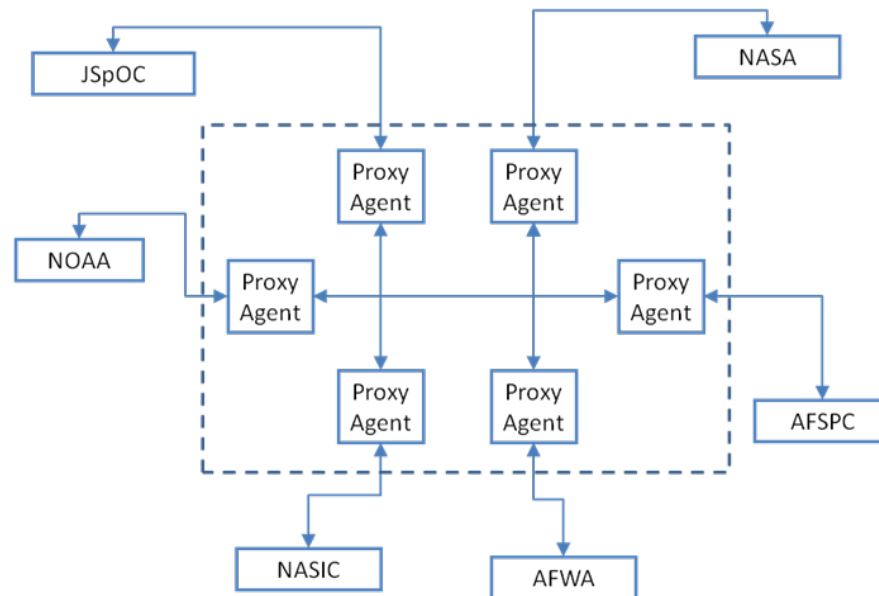
Persistent Search and Track Mission with 3 Quadrotors



RAVEN testbed, Aerospace Controls Lab

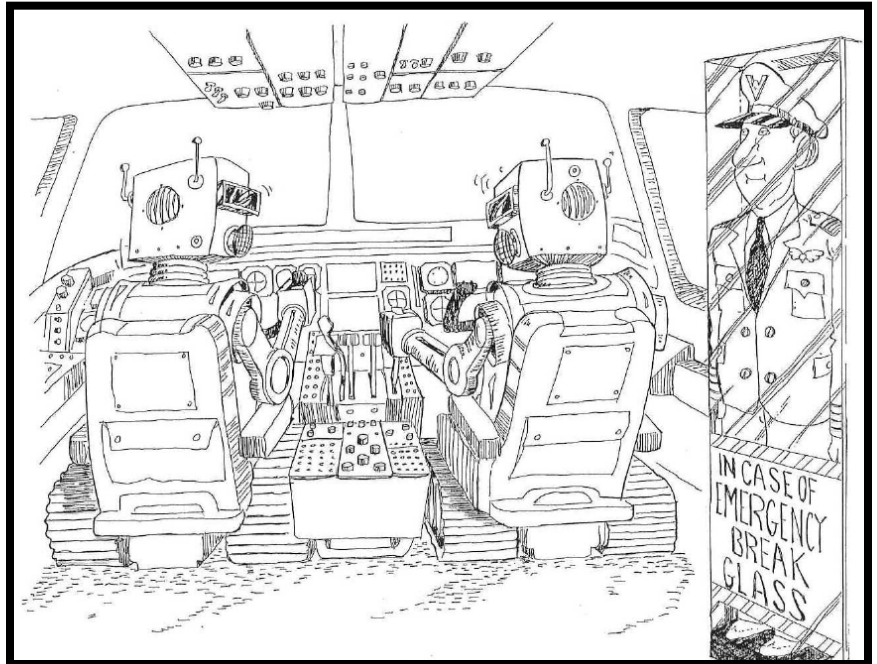
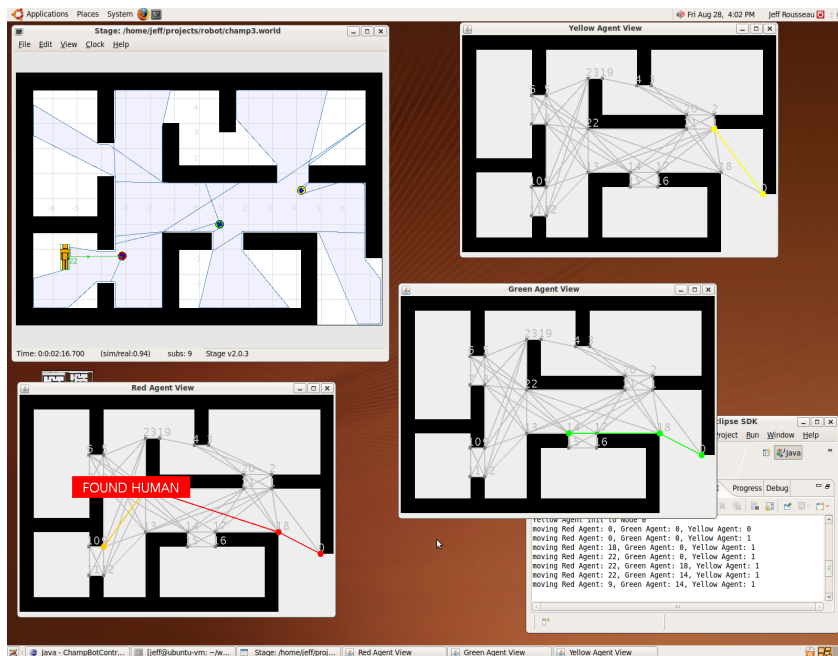
Personal assistant agents (Amato et al.,11)

- People connected to many others and sources of info
- Use software personal assistant agents to provide support (Dec-MDP, Shared-MDP)
 - Agents collaborate on your behalf to find resources, teams, etc.
 - Goal: work more efficiently with others and discover helpful info



Mixed-initiative robotics (Carlin et al.,10)

- Humans and robots collaborating for search and pursuit
- Determine what tasks robots should do (MMDP, Dec-POMDP) and what tasks humans should do
- Adjustable autonomy based on preferences and situation



References

- D. S. Bernstein, S. Zilberstein, R. Washington, and J. L. Bresina, “Planetary rover control as a Markov decision process,” in *Proceedings of the The Sixth International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.
- Alan Carlin, Jeanine Ayers, Jeff Rousseau and Nathan Schurr. Agent-based coordination of human-multirobot teams in complex environments. AAMAS 2010: 1747-1754
- D. V. Pynadath and M. Tambe, “The communicative multiagent team decision problem: Analyzing teamwork theories and models,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 389–423, 2002.
- D.Shi, M.Z.Sauter, X.Sun, L.E.Ray, and J.D.Kralik, “An extension of bayesian game approximation to partially observable stochastic games with competition and cooperation,” in *International Conference on Artificial Intelligence*, 2010.
- R. Emery-Montemerlo, G.Gordon, J.Schneider, and S.Thrun, “Game theoretic control for robot teams,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 1163–1169.
- M. T. J. Spaan and F. S. Melo, “Interaction-driven Markov games for decentralized multiagent planning under uncertainty,” in *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, 2008, pp. 525–532.
- L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, “Coordinated multi- robot exploration under communication constraints using decentralized Markov decision processes,” in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
- R. Cogill, M. Rotkowitz, B. Van Roy, and S. Lall, “An approximate dynamic programming approach to decentralized control of stochastic systems,” in *Proceedings of the Forty-Second Allerton Conference on Communication, Control, and Computing*, 2004.

References

- J. M. Ooi and G. W. Wornell, "Decentralized control of a multiple access broadcast channel: Performance bounds," in *Proceedings of the 35th Conference on Decision and Control*, 1996, pp. 293–298.
- L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in *Proceedings of the International Joint Conference on Neural Networks*, 2002, pp. 1825–1830.
- J. Pajarinen and J. Peltonen, "Efficient planning for factored infinite- horizon DEC-POMDPs," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, July 2011, pp. 325–331.
- R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, "Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs," in *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005.
- A. Kumar and S. Zilberstein, "Constraint-based dynamic programming for decentralized POMDPs with structured interactions," in *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, 2009, pp. 561–568.
- A. Kumar and S. Zilberstein, "Event-detecting multi-agent MDPs: Complexity and constant- factor approximation," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2009, pp. 201–207.
- Nazim Kemal Ure, Girish Chowdhary, Joshua Redding, Tuna Toksoz, JonathanHow, Matthew Vavrina, John Vian, "Experimental demonstration of efficient multi-agent learning and planning for persistent missions in uncertain environments", *Conference on Guidance Navigation and Control*, Minneapolis, MN, August 2012
- David V. Pynadath and Milind Tambe. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research (JAIR)*, Volume 16, pp. 389-423. 2002.

Problem domains and software tools

- ▶ An overview of the existing benchmark problems.
- ▶ Description of available software.

Benchmark problems

Some benchmark problems:

- ▶ DEC-Tiger (Nair et al., 2003)
- ▶ BroadcastChannel (Hansen et al., 2004)
- ▶ Meeting on a grid (Bernstein et al., 2005)
- ▶ Cooperative Box Pushing (Seuken and Zilberstein, 2007)
- ▶ Recycling Robots (Amato et al., 2007)
- ▶ FireFighting (Oliehoek et al., 2008)
- ▶ Sensor network problems (Nair et al., 2005; Kumar and Zilberstein, 2009a,b)

Software

- ▶ The MADP toolbox aims to provide a software platform for research in decision-theoretic multiagent planning (Spaan and Oliehoek, 2008).
- ▶ Main features:
 - ▶ A uniform representation for several popular multiagent models.
 - ▶ A parser for a file format for discrete Dec-POMDPs.
 - ▶ Shared functionality for planning algorithms.
 - ▶ Implementation of several Dec-POMDP planners.
- ▶ Released as free software, with special attention to the extensibility of the toolbox.
- ▶ Provides benchmark problems.

Problem specification

```
agents: 2
discount: 1
values: reward
states: tiger-left tiger-right
start:
uniform
actions:
listen open-left open-right
listen open-left open-right
observations:
hear-left hear-right
hear-left hear-right
```

Problem specification (1)

```
# Transitions
T: * :
uniform
T: listen listen :
identity
# Observations
O: * :
uniform
O: listen listen : tiger-left : hear-left hear-left : 0.7225
O: listen listen : tiger-left : hear-left hear-right : 0.1275
[...]
O: listen listen : tiger-right : hear-left hear-left : 0.0225
# Rewards
R: listen listen: * : * : * : -2
R: open-left open-left : tiger-left : * : * : -50
[...]
R: open-left listen: tiger-right : * : * : 9
```

Example program

```
#include "ProblemDecTiger.h"
#include "JESPExhaustivePlanner.h"
int main()
{
    ProblemDecTiger dectiger;
    JESPExhaustivePlanner jesp(3,&dectiger);
    jesp.Plan();
    std::cout << jesp.GetExpectedReward() << std::endl;
    std::cout << jesp.GetJointPolicy()->SoftPrint() << std::endl;
    return(0);
}
```

Program output

```
src/examples> ./decTigerJESP
Value computed for DecTiger horizon 3: 5.19081
Policy computed:
JointPolicyPureVector index 120340 depth 999999
Policy for agent 0 (index 55):
Oempty, --> a00:Listen
Oempty, o00:HearLeft, --> a00:Listen
Oempty, o01:HearRight, --> a00:Listen
Oempty, o00:HearLeft, o00:HearLeft, --> a02:OpenRight
Oempty, o00:HearLeft, o01:HearRight, --> a00:Listen
Oempty, o01:HearRight, o00:HearLeft, --> a00:Listen
Oempty, o01:HearRight, o01:HearRight, --> a01:OpenLeft
Policy for agent 1 (index 55):
Oempty, --> a10:Listen
Oempty, o10:HearLeft, --> a10:Listen
Oempty, o11:HearRight, --> a10:Listen
Oempty, o10:HearLeft, o10:HearLeft, --> a12:OpenRight
Oempty, o10:HearLeft, o11:HearRight, --> a10:Listen
Oempty, o11:HearRight, o10:HearLeft, --> a10:Listen
Oempty, o11:HearRight, o11:HearRight, --> a11:OpenLeft
```


References I

- C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2007.
- D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.
- E. A. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. 2004.
- A. Kumar and S. Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2009a.
- A. Kumar and S. Zilberstein. Event-detecting multi-agent MDPs: Complexity and constant-factor approximation. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2009b.
- R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003.
- R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. 2005.
- F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.
- S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, July 2007.
- M. T. J. Spaan and F. A. Oliehoek. The MultiAgent Decision Process toolbox: software for decision-theoretic planning in multiagent systems. In *Multi-agent Sequential Decision Making in Uncertain Domains*, 2008. Workshop at AAMAS08.

Resources

- Web pages
 - UMass Dec-POMDP webpage
 - Papers, talks, domains, code, results
 - <http://rbr.cs.umass.edu/camato/decpomdp/>
 - Matthijs's Dec-POMDP page
 - Domains, code, results
 - <http://masplan.org>
 - USC's Distributed POMDP page
 - Papers, some code and datasets
 - <http://teamcore.usc.edu/projects/dpomdp/>
- Introductory papers
 - Frans A. Oliehoek. Decentralized POMDPs. In Wiering, Marco and van Otterlo, Martijn, editors, *Reinforcement Learning: State of the Art*, Adaptation, Learning, and Optimization, pp. 471–503, Springer Berlin Heidelberg, Berlin, Germany, 2012.
 - Sven Seuken and Shlomo Zilberstein. Formal Models and Algorithms for Decentralized Decision Making Under Uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*. 17:2, pages 190-250, 2008.
 - Prashant Doshi. Decision Making in Complex Multiagent Contexts: A Tale of Two Frameworks *AI Magazine*, vol 33, no 4, 2012