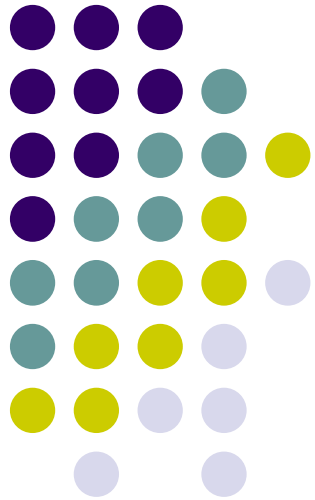
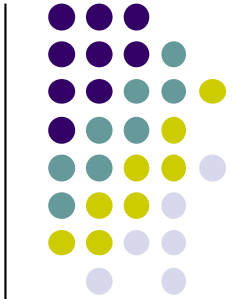
Probabilistic Planning

Shlomo Zilberstein
College of Information and Computer Sciences
University of Massachusetts Amherst



Probabilistic Planning



Static
vs.
Dynamic

Deterministic
vs.
Stochastic

Instantaneous
vs.
Durative

Single
vs.
Multi-agent

Centralized
vs.
Decentralized

Environment

What action next?

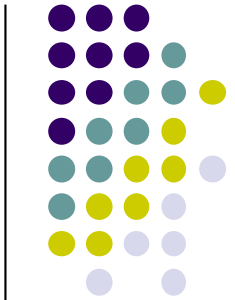
actions

percepts

Fully
vs.
Partially
Observable

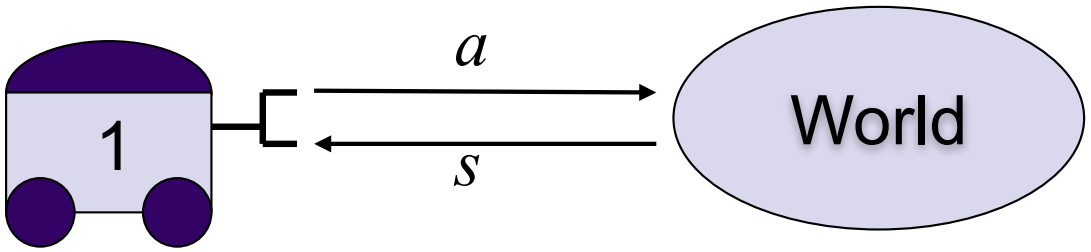
Perfect
vs.
Noisy

Outline



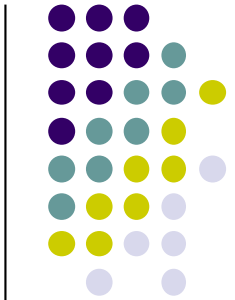
- Planning with Markov decision processes (MDPs)
- Solving MDPs using heuristic search (LAO*)
- Using determinization and other reduced models
- Real time continual planning
- Planning under partial observability (POMDPs)
- Multiagent planning (Dec-POMDPs)

Planning Under Uncertainty



- An agent interacts with the environment over some extended period of time
- Utility function depends on the sequence of decisions and their outcomes
- A rational agent should choose an action that maximizes its expected utility

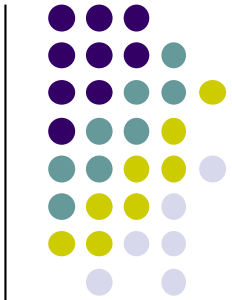
Markov Decision Process



- A Markov decision process is a tuple $\langle S, A, P, R \rangle$, where
  - S is a finite set of domain states, with initial state s_0
  - A is a finite set of actions
  - $P(s' \mid s, a)$ is a state transition function
  - $R(s)$, $R(s, a)$, or $R(s, a, s')$ is a reward function
- The Markov assumption:

$$P(s_t \mid s_{t-1}, s_{t-2}, \ldots, s_1, a) = P(s_t \mid s_{t-1}, a)$$

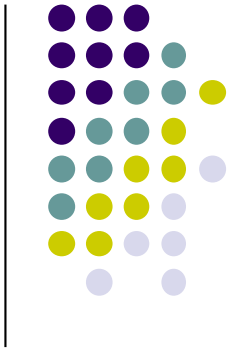
Partially Observable MDPs



Augmenting the completely observable MDP with the following elements:

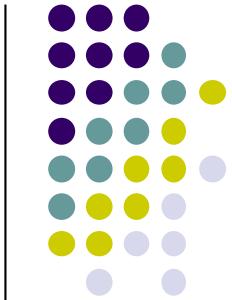
- O - set of observations
- $P(o \mid s', a)$ - observation function
- Discrete probability distribution over starting states.

Performance criteria



- Specify how to combine rewards over multiple time steps or histories.
- Finite horizon problems involve a fixed number of steps.
- The best action in each state may depends on the number of steps left, hence it is **nonstationary**.
- Infinite horizon policies depend only on the current state, hence the optimal policy is **stationary**.
- Finite horizon problems can be solved by adding the number of steps left to the state.

Performance criteria cont.

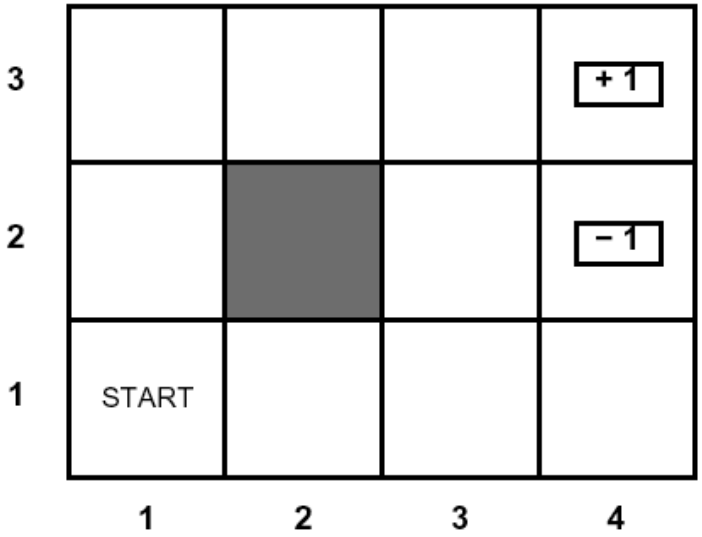


- The assumption that the agent's preferences between state sequences is stationary:

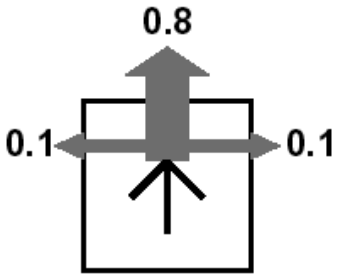
  $[s_0,s_1,s_2,\ldots] > [s_0,s_1',s_2',\ldots]$ iff $[s_1,s_2,\ldots] > [s_1',s_2',\ldots]$
- This leads to just two ways to define utilities of histories:
  - Additive rewards: utility of a history is $U([s_0,a_1,s_1,a_2,s_2,\ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$
  - Discounted rewards: utility of a history is $U([s_0,a_1,s_1,a_2,s_2,\ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) \ldots$
- With a **proper policy** (guaranteed to reach a terminal state) no discounting is needed.
- An alternative to discounting in infinite-horizon problems is to optimize the average reward per time step.

A Simple Grid Environment

[Russell and Norvig 03]



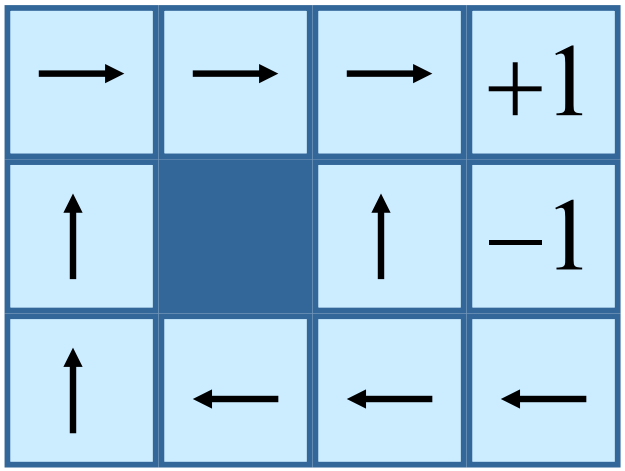
(a)

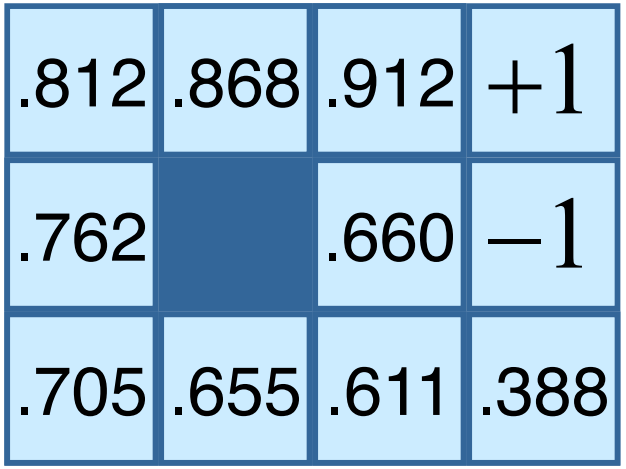


(b)

Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction.

Example: An Optimal Policy





Actions succeed with probability 0.8 and move sideways with probability 0.1 (remain in the same position when there is a wall). Actions incur a small cost.

Policies for Different $R(s)$

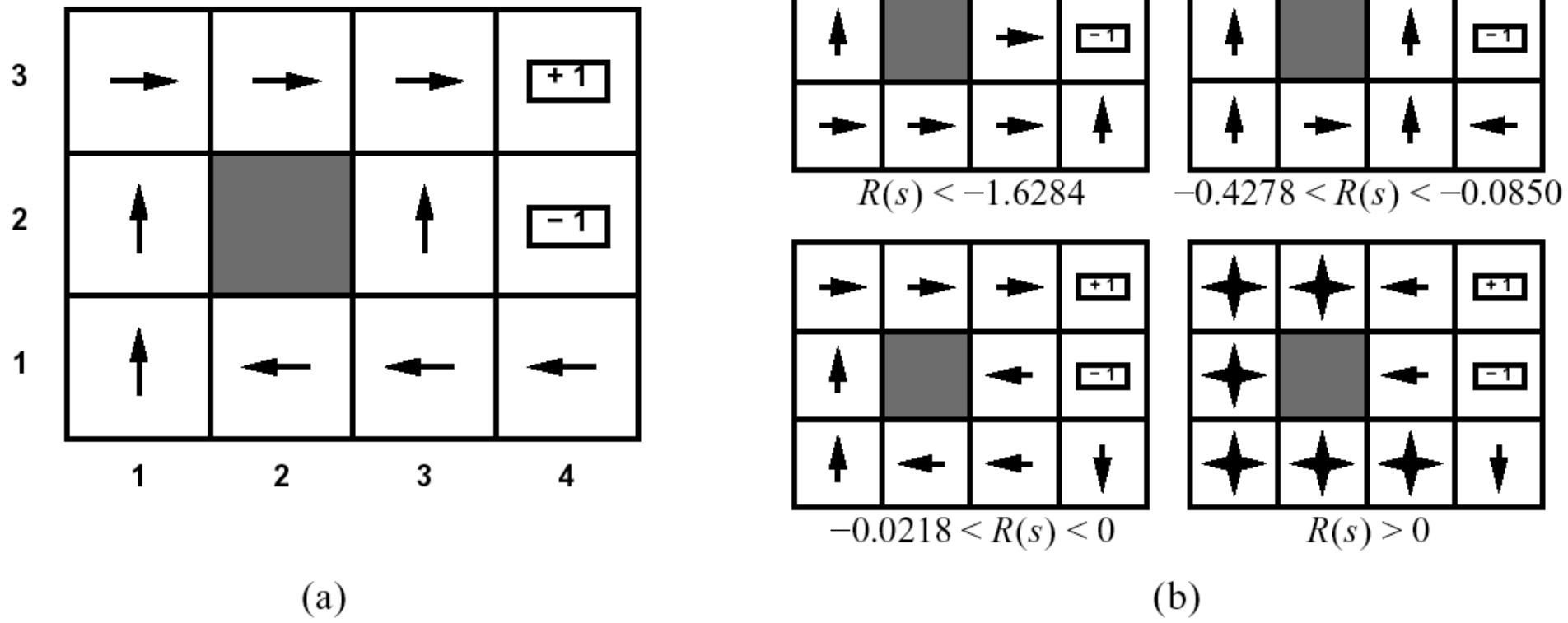
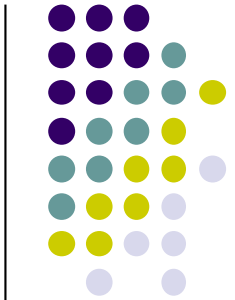


Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.

Policies and Utilities of States



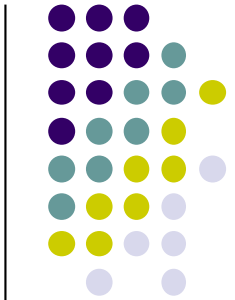
- A policy π is a mapping from states to actions.
- An optimal policy π^* maximizes the expected reward:

$$\pi^* = \arg\max_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right]$$

- The utility of a state

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s \right]$$

The Bellman Equation



- Optimal policy defined by:

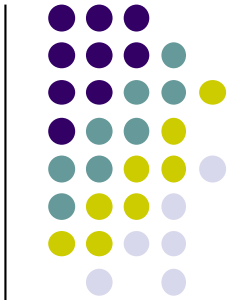
$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s,a) U(s')$$

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s,a) U(s')$$

- Can be solved using dynamic programming [Bellman, 1957]

Value Iteration

[Bellman 57]



repeat

$U \leftarrow U'$

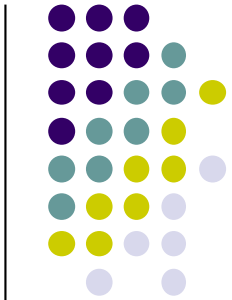
for each state s do

$$U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} P(s' \mid s,a) U(s')$$

end

until $CloseEnough(U,U')$

Convergence of VI



An initial error bound is: $\| U - U^* \| \leq 2R_{\max} / (1-\gamma)$

Based on $\| BU_i - BU_i^{'} \| \leq \gamma \| U_i - U_i^{'} \|$

How many iterations are needed to reach a max norm error of ε?

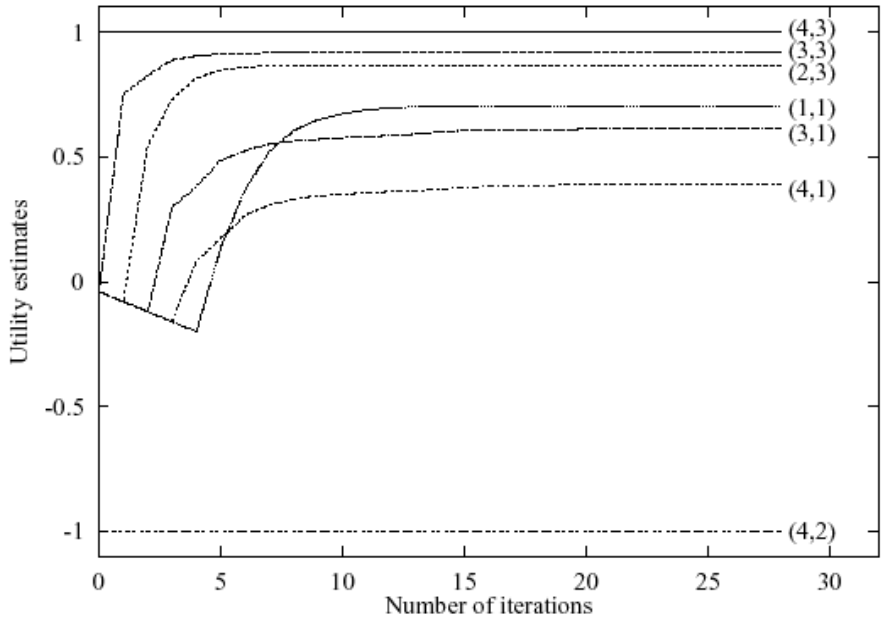
$\gamma^N \cdot 2R_{\max} / (1-\gamma) \leq \varepsilon$

$N = \lceil \log(2R_{\max} / (\varepsilon(1-\gamma)) / \log(1/\gamma) \rceil$

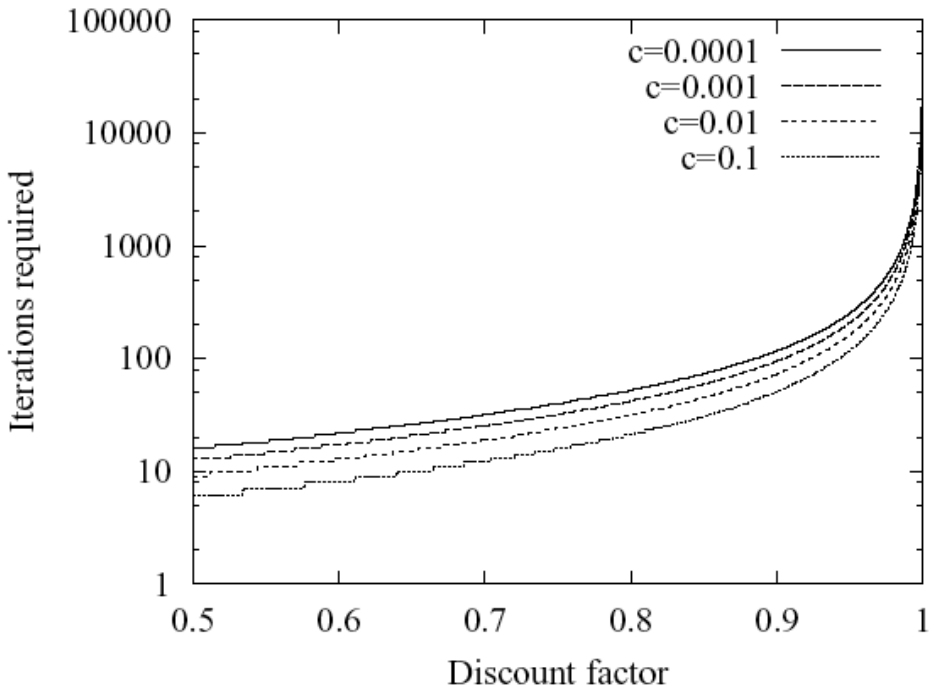
A less conservative termination condition

if $\| U' - U \| < \varepsilon(1-\gamma)/2\gamma$ then $\| U' - U^* \| < \varepsilon$

Convergence of VI



(a)

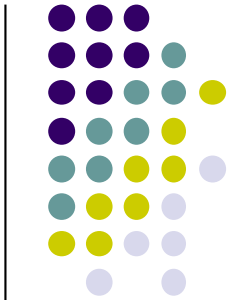


(b)

Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.

Policy Iteration

[Howard 60]



repeat

$\pi \leftarrow \pi'$

$U \leftarrow ValueDetermination(\pi)$

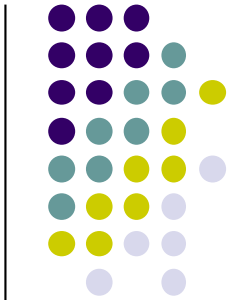
for each state s do

$$\pi'[s] \leftarrow \arg\max_a \sum_{s'} P(s' \mid s,a) U(s')$$

end

until $\pi = \pi'$

Value Determination



Can be implemented using :

Value Iteration :

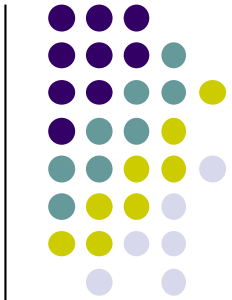
$$U'(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U(s')$$

or

By solving a set of n linear equations :

$$U(s) = R(s) + \sum_{s'} P(s' \mid s, \pi(s)) U(s')$$

Policy Iteration Guarantees

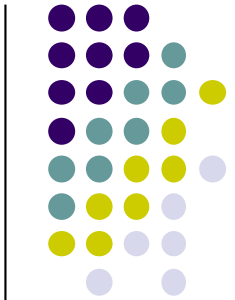


Theorem: Policy iteration is guaranteed to converge to an optimal policy and the optimal value function.

Proof sketch:

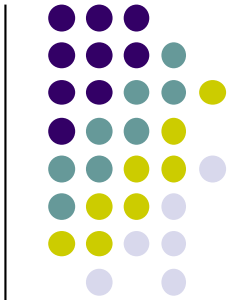
- **Guarantee to converge**: In every step prior to convergence the policy improves. This means that a given policy can be encountered at most once. Since there is a finite number of policies, the process must converge.
- **Optimal at convergence**: At convergence, the policy remains unchanged. Thus the value of the policy must satisfy the Bellman equation, which means it equals V*.

Outline



- Planning with Markov decision processes (MDPs)
- **Solving MDPs using heuristic search (LAO*)**
- Using determinization and other reduced models
- Real time continual planning
- Planning under partial observability (POMDPs)
- Multiagent planning (Dec-POMDPs)

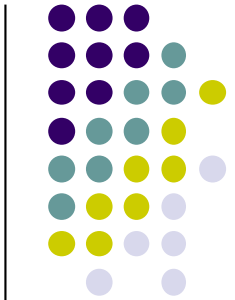
Stochastic Shortest-Path Problems



- Given a start state, the objective is to minimize the expected cost of reaching a goal state.
- S: a finite set of states
- $A(i), i \in S$: a finite set of actions available in state i
- $P_{ij}(a)$: probability of state j after action a in state i
- $C_i(a)$: expected cost of taking action a in state i

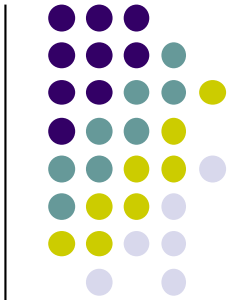
MDPs and State-Space Search

[Hansen and Zilberstein, AAAI 98, AIJ 01]



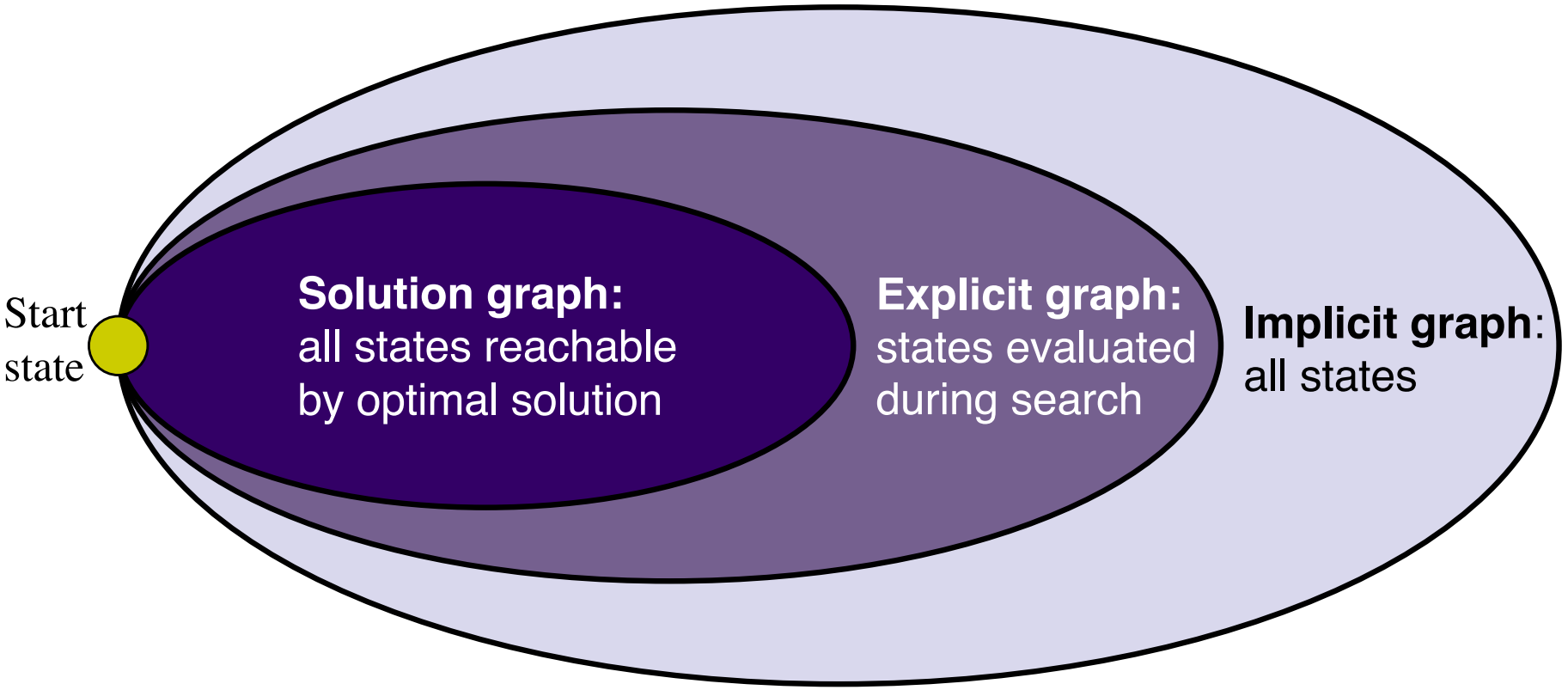
- MDPs present a state-space search problem in which transitions are stochastic.
- Because state transitions are stochastic, it is impossible to bound the number of actions needed to reach the goal (indefinite-horizon).
- Search algorithms like A* can handle the deterministic version of this problem.
- But neither A* nor AO* can solve indefinite horizon problems.

Advantages of Search



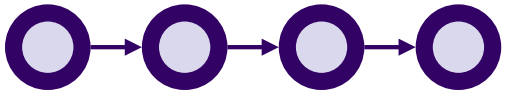
- Can find optimal solutions without evaluating all problem states.
- Can take advantage of domain knowledge to reduce search effort.
- Can benefit from a large body of existing work in AI on how to search in real-time and how to trade-off solution quality for search effort.

Solving MDPs Using Search

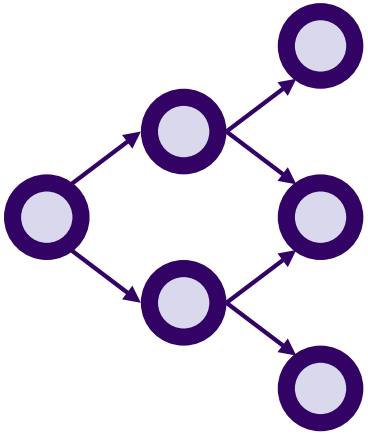


Given a start state, heuristic search can find an optimal solution without evaluating all states.

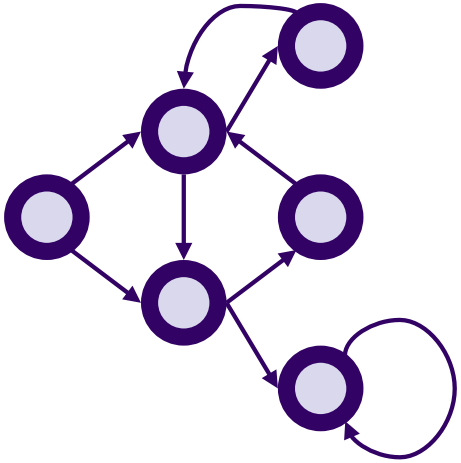
Possible Solution Structures



Solution is a
simple path

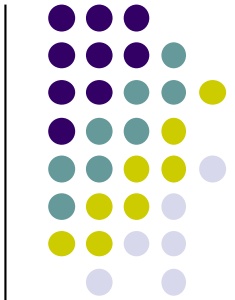


Solution is an
acyclic graph



Solution is a
cyclic graph

DP and Heuristic Search

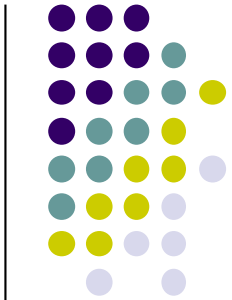


Solution Structure	**Sequence**	**Branches**	**Loops**
Dynamic Programming	Forward DP	Backwards induction	Policy (value) iteration
Heuristic Search	**A***	**AO***	**LAO***

Heuristic search = starting state +
forward expansion of solution +
admissible heuristic +
DP dynamic programming

AO*

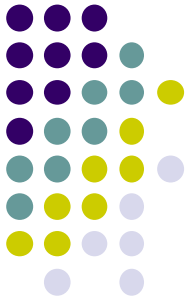
[Nilsson 71; Martelli and Montanari 73]



- Initialize partial solution graph to start state.
- Repeat until a complete solution is found:
  - Expand some nonterminal state on the fringe of the best partial solution graph.
  - Use backwards induction to update the costs of all ancestor states of the expanded state and possibly change the selected action.

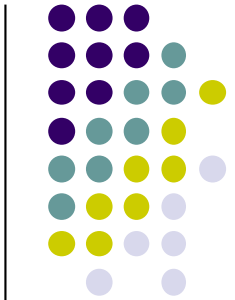
LAO*

[Hansen and Zilberstein, AAAI 98, AIJ 01]



- Like AO*, performs dynamic programming on the set of states that includes the expanded state and all of its ancestors.
- But LAO* must use either policy iteration or value iteration instead of backward induction.
- Convergence to exact state costs of value iteration is asymptotic, but it is generally more efficient than policy iteration for large problems.

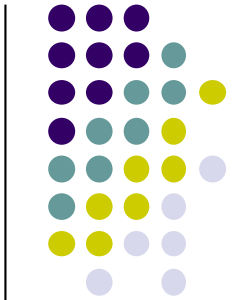
Heuristic Evaluation Function



- $h(i)$ is an heuristic estimate of the minimal-cost solution for every non-terminal tip state.
- $h(i)$ is admissible if $h(i) < f^*(i)$.
- An admissible heuristic estimate $f(i)$ for any state in the explicit graph is defined as follows:

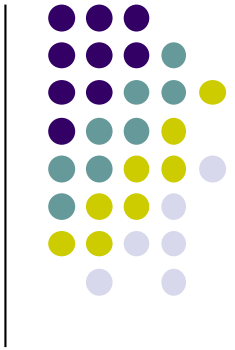
$$f(i) = \begin{cases} 0 \text{ if } i \text{ is a goal state} \\ h(i) \text{ if } i \text{ is a non-terminal tip state} \\ \text{else } \min\limits_{a \in A(i)} \left[c_i(a) + \sum\limits_{j \in S} p_{ij}(a) f(j) \right] \end{cases}$$

Theoretical Properties



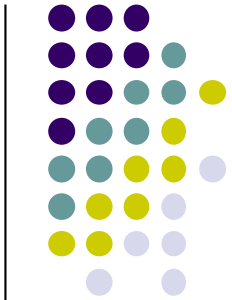
- **Theorem 1**: Using an admissible heuristic, LAO* converges to an optimal solution without (necessarily) expanding/evaluating all states.
- **Theorem 2**: If $h_2(i)$ is a more informative heuristic than $h_1(i)$ (i.e., $h_1(i) \leq h_2(i) \leq f^*(i)$), LAO* using $h_2(i)$ expands a subset of the worst case set of states expanded using $h_1(i)$.

Examples



- Tested on gridworld and racetrack problems with up to 22,000 states, and some POMDPs.
- Naive implementation not necessarily more efficient than DP, but a good implementation can be.
- Performance depends on:
  - Search control (e.g., how states are selected for expansion and backups)
  - Problem characteristics (i.e., how many states are visited by optimal policy compared to total number of states)

$f = g + h$ decomposition



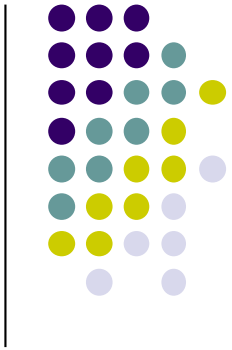
- $f(i)$ can be decomposed into the sum of:
  - $g(i)$: cost-to-arrive from the start state to a state on the fringe of the best partial solution; and
  - $h(i)$: estimated cost-to-go from the fringe of the best partial solution to a goal state.
- Let a be the action that optimizes $f(i)$, then:

$$g(i) := c_i + \sum_{j \in S} p_{ij}(a) g(j)$$

$$h(i) := \sum_{j \in S} p_{ij}(a) h(j)$$

- On convergence to a complete solution, $h(i) = 0$ and $f(i) = g(i)$ for every state i in the solution.

Weighted Heuristic

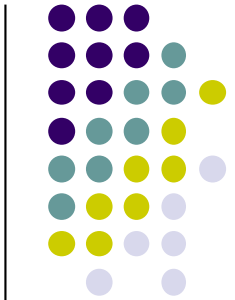


- Improve efficiency in exchange for a bounded decrease in solution quality [Pohl, 1973]:

$$f(i) = (1 - w) \cdot g(i) + w \cdot h(i), \quad 0.5 \leq w \leq 1$$

- Using this non-admissible heuristic, A* finds a solution that is at most $w/(1\text{-}w)$ worse than optimal [Davis et al., 1989].
- Can be used as an anytime algorithm that converges on optimal solution.
- Given $f = g + h$ decomposition, it is easy to create a weighted version of LAO*.

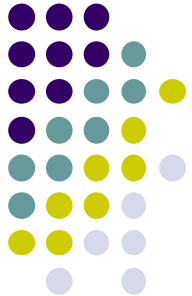
Performance of LAO*



Racetrack problem with 21,371 states
Optimal solution visits only 2,248 states

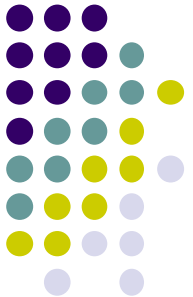
Algorithm	Solution Quality	Nodes Evaluated	Convergence Time
Value iteration	Optimal	21,371 (all)	15.7 sec
LAO* w/ zero heuristic	Optimal	18,195	10.7
LAO* w/ Dijkstra heuristic	Optimal	14,252	4.7
LAO* w/ weight = 0.6	Optimal	8,951	3.1
LAO* w/ weight = 0.67	+ 4%	4,508	1.8

Outline



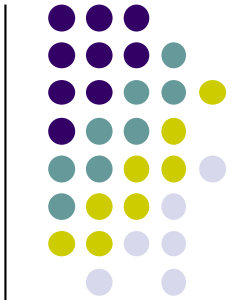
- Planning with Markov decision processes (MDPs)
- Solving MDPs using heuristic search (LAO*)
- **Using determinization and other reduced models**
- Real time continual planning
- Planning under partial observability (POMDPs)
- Multiagent planning (Dec-POMDPs)

What is a Reduced Model?



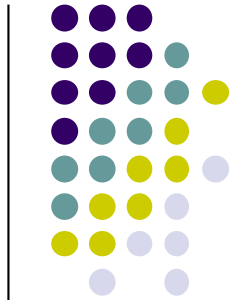
- A **reduced model** includes less details about the domain and is often much easier to solve.
- Interest in planning with reduced models has a long history, specifically for MDPs:
  - aggregating equivalent states
    [Dean and Givan 97][Ravindran and Barto 02]
  - approximate partitioning of the state space
    [Dean, Givan and Leach 97]
  - limiting plan reachability to a certain envelop
    [Dean et al. 95][Trevizan and Veloso 12]

Determinization

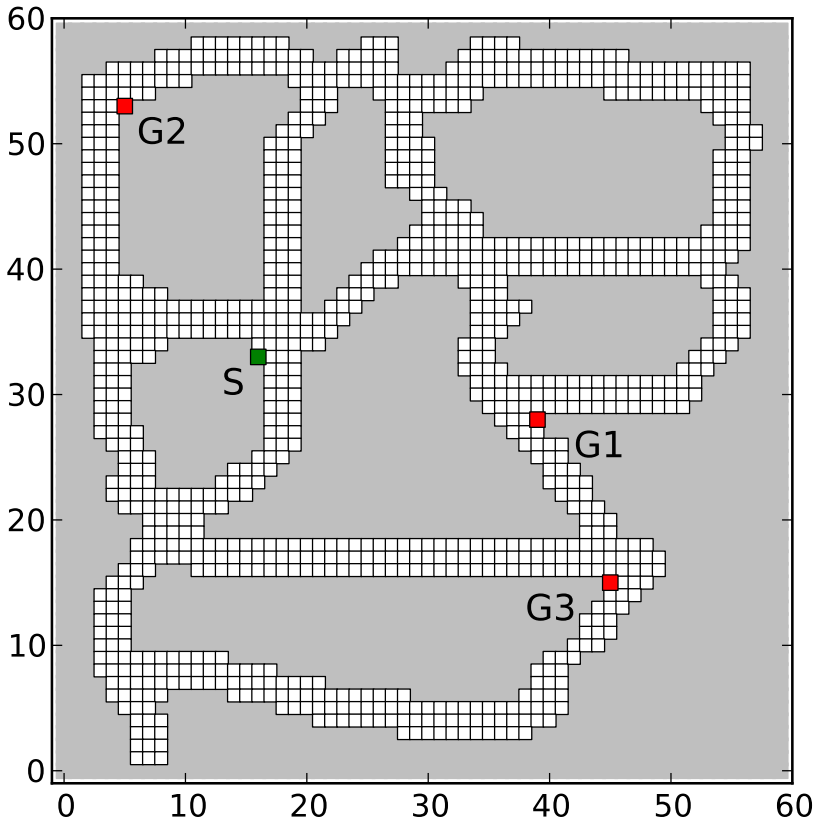


- Determinization is a simple form of model reduction that ignores uncertainty altogether.
- Determinization works as follows:
  1. Create a deterministic version of the underlying MDP and solve it using a classical planner (e.g., consider only the most likely outcome).
  2. Execute the deterministic plan as long as the outcome state is covered by the current plan.
  3. When a state not covered by the plan is encountered, replan and resume execution.

Example 1: The Racetrack Problem

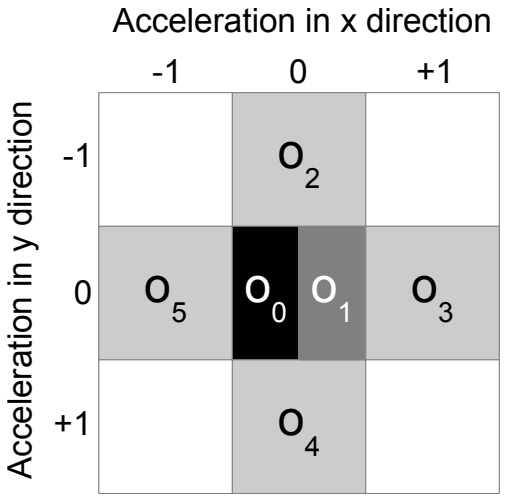


- A well-known RL benchmark involving driving a race car to a target location.
- The state is determined by the location and two-dimensional velocity.
- The car can change its speed in each of the two dimensions by at most 1.
- There is a probability that the car slips, resulting in zero acceleration.
- There is also a probability that the resulting acceleration is off by one w.r.t. the intended acceleration.
- The goal is to reach the target location in as few moves as possible.

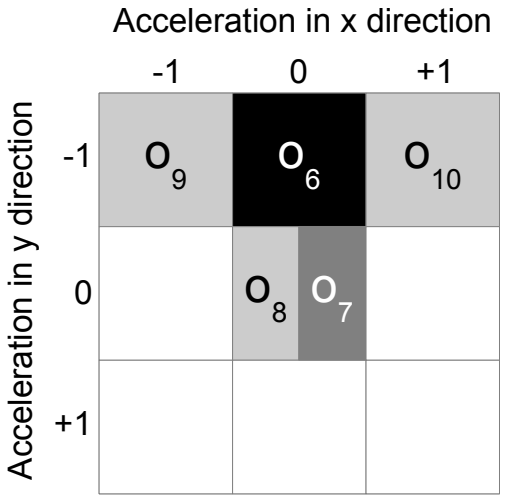


[Sutton and Barto 98]

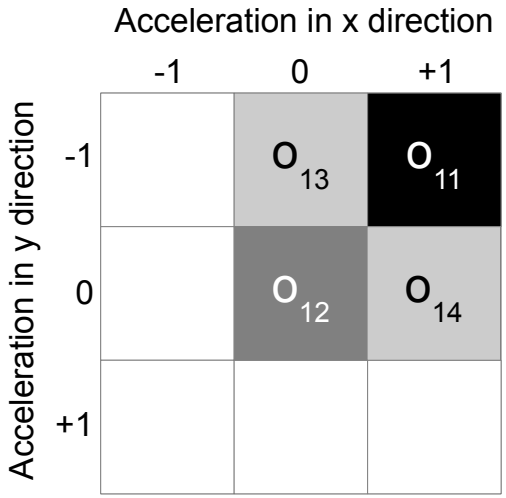
Actions in the Racetrack Domain



Group 1
No acceleration



Group 2
Accelerate in
only 1 direction

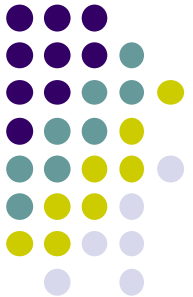


Group 3
Accelerate in 2
directions

Action groups in the racetrack domain:

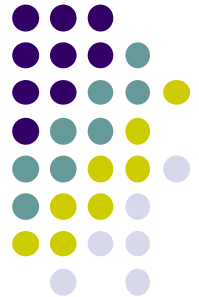
- dark squares represent the intended action
- gray squares represent the outcome when slipping
- light gray squares represent the remaining outcomes

Determinization of the Racetrack Problem

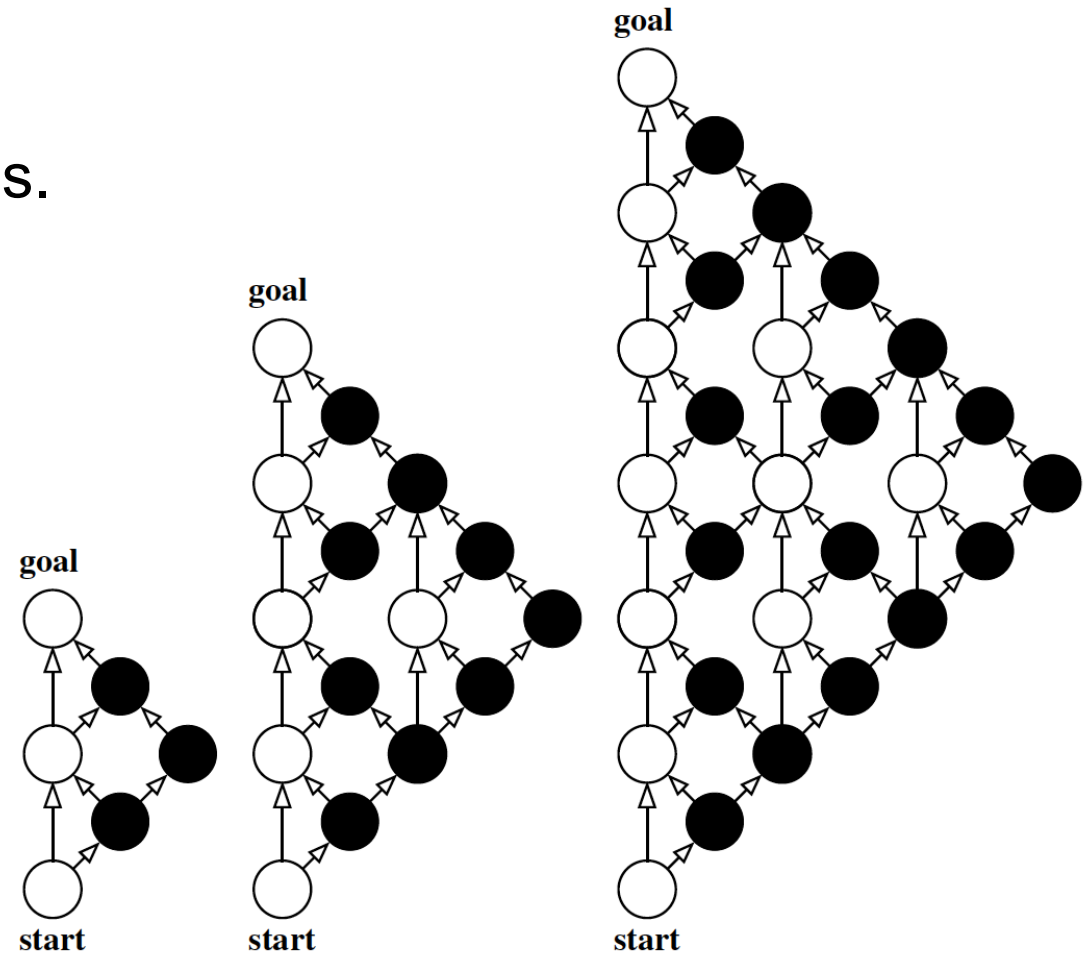


- Ignore uncertainty altogether. Each action always achieves the intended outcome
- Often this is also the “most likely outcome”
- Other principles can be used, including “most conservative outcome” or “all outcome” determinizations

Example 2: The Triangle Tileworld

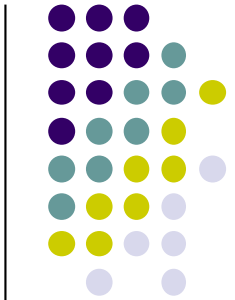


- The first three instances of the triangle tireworld problem with one-way roads.
- A car travels towards the goal location on this graph.
- Every time it moves there is a certain probability of getting a flat tire.
- Only black locations include spare tires for repairing the car.



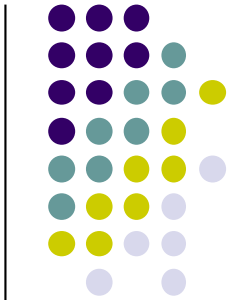
[Little and Thiebaux 07]

Example 2: Triangle Tileworld



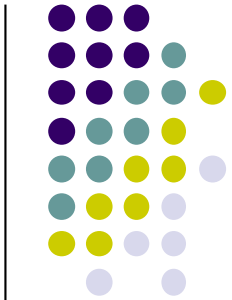
- There is a single optimal solution with a 100% success rate to every problem, and the next best solution(s) will have a 50% success probability;
- The difficulty of finding the optimal solution increases exponentially with each problem; and
- A solution based on the shortest (or most likely) trajectory will have a success probability of 0.5^(2n−1), where n is the problem number.

Determinization: A Surprising Success



- A simple determinization method – FF-Replan – won first place in the 2004 International Probabilistic Planning Competition.
- It seemed paradoxical that it outperformed state-of-the-art MDP solvers.
- Ignited renewed interest in reduced models.
- Some claimed that the competition domains were not “probabilistically interesting”.
- Others offered new determinization strategies that work in “probabilistically interesting” domains.

More Robust Determinizations



- **FF-Hindsight** uses a *hindsight optimization* approach to approximate the value function of the original MDP by sampling multiple deterministic futures that are solved using the FF planner.

  [Yoon, Fern, Givan and Kambhampati 08]

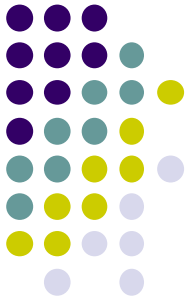
- **RFF** generates a plan for an envelope of states such that the probability of reaching a state outside the envelope is below some predefined threshold. To increase the envelope, RFF chooses states outside of the current plan and, using an approach similar to FF-Replan, computes an action for each one of these states.

  [Teichteil-Königsbuch, Kuter and Infantes 10]

- **HMDPP** introduces the so-called *self-loop determinization* to force the deterministic planner to generate plans with a low probability of deviations, using a pattern database to avoid dead ends.

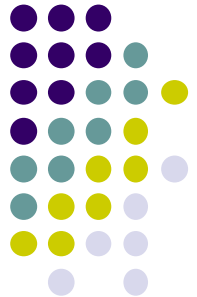
  [Keyder and Geffner 08]

Drawbacks of Determinization



- Some inherent drawbacks of any method that considers each outcome in isolation.
- Can be overly optimistic, ignoring risks associated with other outcomes.
- All outcome determinization could choose a very unlikely outcome just because it is useful, ignoring the more likely ones.
- Overall, any one of these approaches can produce plans that are arbitrarily worse (in terms of solution cost) than the optimal plan.

Choosing the Right Determinization

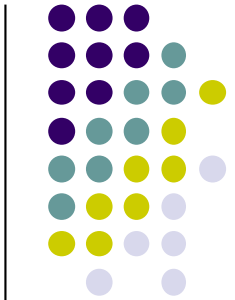


- Heuristics such as “most likely outcome” or “most desirable outcome” don’t work consistently
  - Both of the above heuristics fail in the triangle-tileworld problem
  - The “most undersirable outcome” performs poorly in the racetrack problem

Instance # / Primary outcome	**P01**	**P02**	**P03**	**P04**	**P05**	...	**P10**
(not (not-flattire))	100	100	100	100	100	...	100
(not-flattire)	60	15	4	2	0	...	0

Comparison of two different $\mathcal{M}_1^0$-reductions for the triangle-tireworld

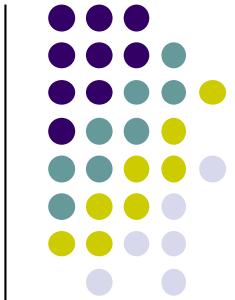
New Spectrum of MDP Reductions



- Characterized by two key parameters:
  1. Maximum number of **primary outcomes** per action that are fully accounted for; and
  2. Maximum number of occurrences of the remaining **exceptional outcomes** that are planned for in advance.
- Planning for a limited number of exceptional outcomes has been explored previously in the context of **fault-tolerant planning**

  [Jensen and Veloso 04] [Domshlak 13] [Pineda et al. 13]

Compact Model Representation



- We consider a factored representation of MDPs (such as PPDDL)
- Actions are represented declaratively as probabilistic operators of the form:

$$a = \langle prec, cost, [p_1 : e_1, ..., p_m : e_m] \rangle$$

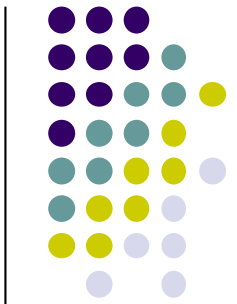
- Transitions represented as:

$$s' = \tau(s, e_i) \text{ and } \mathcal{T}(s'|s, a) = p_i$$

- Primary outcomes of action a:

$$\mathcal{P}_a \subseteq \{e_1, ..., e_m\}$$

Augmented MDP for Reduced Model



- State space: $(s, j) \in \mathcal{S} \times \{0, 1, ..., k\}$
- Initial state: $(s_0, 0)$, indicating that no exceptions have occurred
- Costs and goal states remain the same (with the added counter)
- Transition function $\mathcal{T}'$ defined as follows:

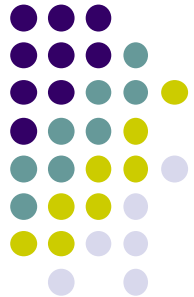
  when $j = k$ (the exception bound):

$$\begin{cases} \forall s, a \;\; \mathcal{T}'((s', k)|(s, k), a) = p'_i & \text{if } e_i \in \mathcal{P}_a \\ 0 & \text{otherwise} \end{cases}$$

  when $j < k$, exceptions are fully accounted for:

$$\begin{cases} \forall s, a \;\; \mathcal{T}'((s', j)|(s, j), a) = p_i & \text{if } \;\; e_i \in \mathcal{P}_a \\ \forall s, a \;\; \mathcal{T}'((s', j+1)|(s, j), a) = p_i & \text{otherwise} \end{cases}$$

Definition of Reduced Models

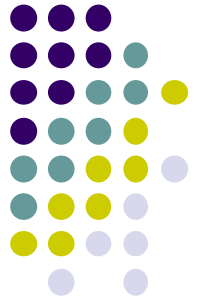


Definition ($\mathcal{M}_l^k$-reduction) An $\mathcal{M}_l^k$-reduction of an MDP is an augmented MDP, where $j \in \{0, 1, ..., k\}$ and $\forall a \; |\mathcal{P}_a| \leq l$.

- For example, the single-outcome determinization used in FF-Replan is an instance of an $\mathcal{M}_1^0$-reduction where each set $\mathcal{P}_a$ contains the single most likely outcome of the action a.
- For any given k and l there could be multiple $\mathcal{M}_l^k$-reductions.
- $M \in \mathcal{M}_l^k$ indicates that M is an instance of an $\mathcal{M}_l^k$-reduction.
- Different instances are characterized by two choices:
  - The specific outcomes that are labeled as primary
  - How the probability of exceptional outcomes is distributed

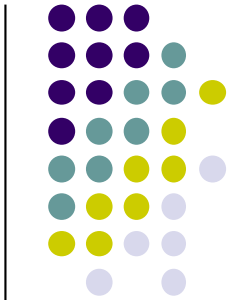
Questions about Reduced Models

[Pindea and Zilberstein, UMass TR 17]



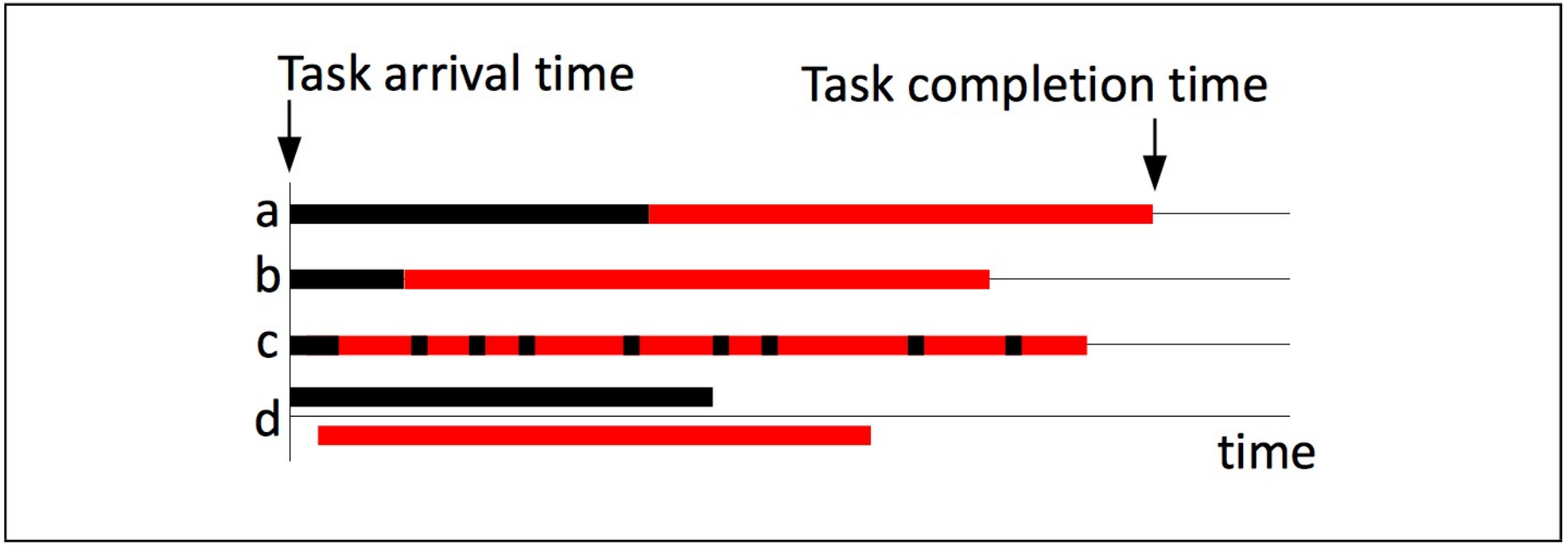
- How should we assess the comprehensive value of a reduction? Can this be done analytically?
- Considering the space of reductions, when is determinization preferable?
- In the space of possible determinizations, can the best ones be identified using a simple heuristic (e.g., choosing the most likely outcome)? Or do we need more sophisticated value-based methods for that purpose?
- How can we explore efficiently the space of reductions? How can we find good ones or the best one?
- Given a PPDDL description of an MDP and some k and l, can one automatically generate a PPDDL description of a corresponding reduction? How can we explore this space and evaluate different reductions using a compilation scheme and standard MDP solvers?

Outline



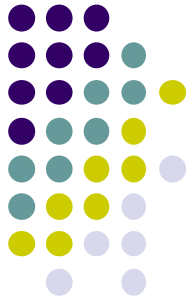
- Planning with Markov decision processes (MDPs)
- Solving MDPs using heuristic search (LAO*)
- Using determinization and other reduced models
- **Real time continual planning**
- Planning under partial observability (POMDPs)
- Multiagent planning (Dec-POMDPs)

Real Time Planning Paradigms



- Four strategies for interleaving periods of planning (shown in black) and period of execution (shown in red)
- With reduced models, can trigger planning whenever the number of exceptional outcomes reaches the bound

Continual Planning Approach



input: MDP problem $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, k, l and $\mathcal{P}_a$

1 $(s, j) \leftarrow (s_0, 0)$;

2 $\pi \leftarrow$ Find-Reduced-MDP-Plan$((s_0, 0), k, l, \mathcal{P}_a)$;

while $s \notin \mathcal{G}$ **do**

  if $j \neq k$ **then**

3 $(s, j) \leftarrow$ ExecutePlan$(s, \pi(s, j))$;

  else

4 Create new state $\hat{s}$ with one zero-cost action $\hat{a}$ s.t. $\forall s' \in \mathcal{S}$: $Pr((s', 0)|\hat{s}, \hat{a}) = \mathcal{T}(s'|s, \pi(s, j))$;

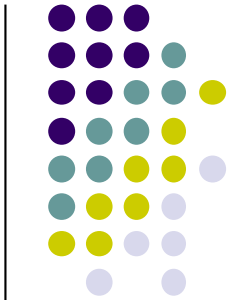
  do in parallel

5 $(s, j) \leftarrow$ ExecutePlan$(s, \pi(s, j))$;

6 $\pi' \leftarrow$ Find-Reduced-MDP-Plan$(\hat{s}, k, l, \mathcal{P}_a)$;

7 $\pi \leftarrow \pi'$; $(s, j) \leftarrow (s, 0)$;

Evaluating Continual Planning



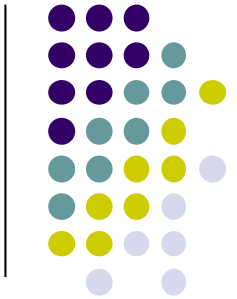
- Let π_k be a *universal* optimal plan for a reduction $M \in \mathcal{M}_l^k$
- Consider the Markov chain over states (s, j), with initial state $(s_0, 0)$ and the following transition function:

$$\begin{array}{ll} Pr((s',j')|(s,j)) = \mathcal{T}'((s',j')|(s,j),\pi_k(s,j)) & \text{if } j<k \\ Pr((s',0)|(s,k)) = \mathcal{T}(s'|s,\pi_k(s,j)) & \text{otherwise} \end{array}$$

- Let V_{cp}^M denote the value function of this Markov chain.

Proposition: $V_{cp}^M(s_0, 0)$ provides the expected value of the continual planning paradigm for the reduced model M when applied to the original problem domain.

Results for Racetrack Domain

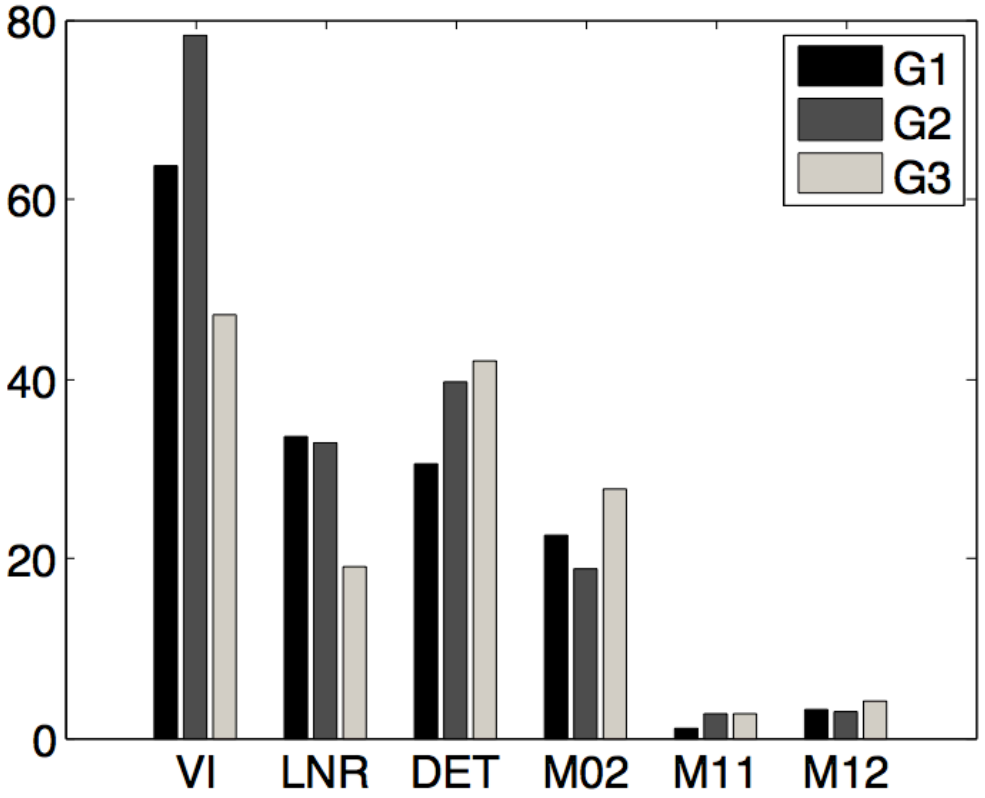


	CPU Time					
	VI	LNR	DET	M02	M11	M12
G1	9,205	4,852	5	57	19	396
G2	10,493	4,422	2	8	106	298
G3	8,664	3,521	5	58	100	445

	Total Cost					
	VI	LNR	DET	M02	M11	M12
G1	23.64	19.28	18.82	17.62	14.58	14.88
G2	23.92	17.85	18.76	15.97	13.78	13.82
G3	27.07	21.93	26.12	23.53	18.91	19.16

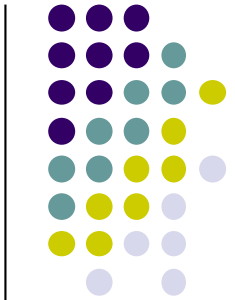
- Average planning time (in milliseconds) and total cost of planning and execution

Results for Racetrack Domain



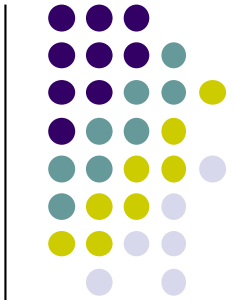
Relative increase in total cost with respect to the lower bound.

Solving Reduced Models



- How can we avoid developing new algorithms for planning with reduced models?
- Can a reduced model be easily described using the PPDDL description of the original problem?
- Can we use standard MDP solvers for planning with reduced models?

Tagging Primary Outcomes



- Introduce a new predicate: "`primary`"
- Given a probabilistic outcome:

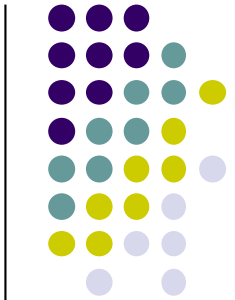
$$(\texttt{probabilistic}\ p_1\ e_1\ p_2\ e_2 \ldots p_m\ e_m)$$

- Replace each primary outcome with a conjunction:

$$(\texttt{and}\ (\texttt{primary}\ (e_i)))$$

- The propositional variable `primary` is set to true in the initial state, preserving the semantics and complexity of the original problem.

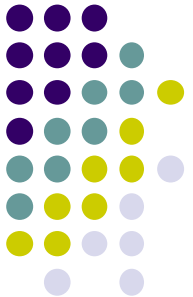
Example of Tagging



```
(:action move-car
 :parameters (?l1 - loc ?l2 - loc)
 :precondition
   (and (vehicle-at ?l1) (road ?l1 ?l2)
    (not-flattire))
 :effect
  (and (vehicle-at ?l2) (not (vehicle-at ?l1))
    (probabilistic
      0.4(and (primary)
              (not (not-flattire)))))))
```

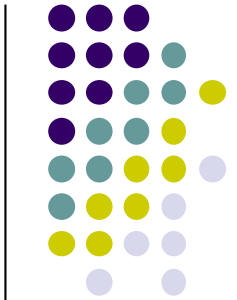
- Tagged PPDDL description of an action from the triangle tileworld

The Compilation Process



- A PPDDL description of the reduced model can be produced automatically
- We limit ourselves currently to exception bound = 1 and represent it using a new propositional variable: `exc`.
- Replace every probabilistic outcome with two conditional outcomes with conditions: `(not (exc))` and `(exc)`.
  - When `(not (exc))`, the outcome is a probabilistic outcome of equal size / probabilities as the original, with every exceptional outcome e_i substituted by the conjunction `(and` (e_i) `(exc))`.
  - When `(exc)`, the outcome is a probabilistic outcome containing only *primary outcomes* with normalized probabilities.

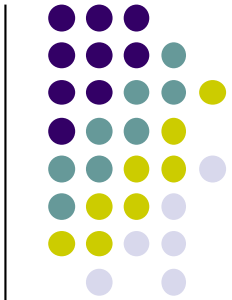
Example of Compiled Action



```
(:action move-car
        :
 :effect
  (and (vehicle-at ?l2) (not (vehicle-at ?l1))
   (when (not (exc))
    (probabilistic
      0.4 (not (not-flattire))
      0.6 (and (exc))))
   (when (exc)
    (probabilistic
      1.0 (not (not-flattire))))))
```

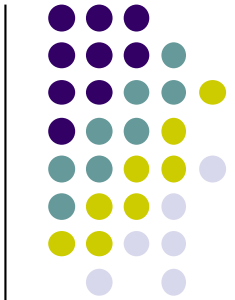
- Compiled PPDDL description of the (reduced model) action

Some Conventions



- Empty outcomes:
  - In PPDDL, empty outcomes can be omitted.
  - An omitted outcome is considered an exception by the compiler.
  - To make it primary, an empty outcome needs to be represented explicitly and tagged as such.
- Nesting of Probabilistic effects:
  - PPDDL allows arbitrary nesting
  - The compiler can handle such nesting, using the convention that normalization of probabilities of primary outcomes is done at the **local** level.

Example of Nested Outcomes



- Consider the following outcomes (* = primary)

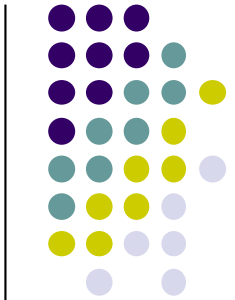
```
(probabilistic
   0.2 *e1
   0.6 *(probabilistic 0.2 e2 0.4 *e3 0.4 *e4)
0.2 e5)
```

- Reduced outcomes will have the following probabilities:

`0.25 e1  0.75×0.5 e3  0.75×0.5 e4`

- Normalization at the inner level does not affect outer level
- The probability of exceptional outcomes is being redistributed among outcomes with structural similarity

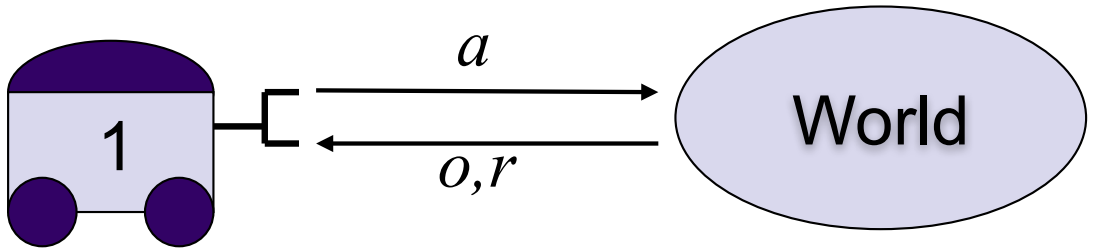
Outline



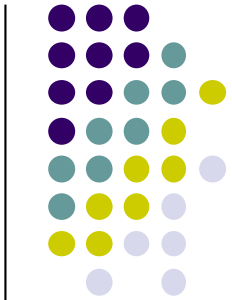
- Planning with Markov decision processes (MDPs)
- Solving MDPs using heuristic search (LAO*)
- Using determinization and other reduced models
- Real time continual planning
- **Planning under partial observability (POMDPs)**
- Multiagent planning (Dec-POMDPs)

POMDPs

- Partially observable Markov decision process (POMDP)
- Agent interacts with the environment
- After each action agent receives:
  - an observation rather than the actual state
  - Receives an immediate reward

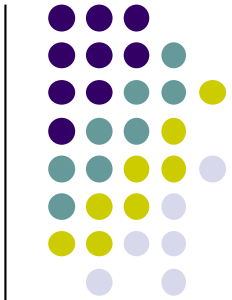


POMDP Definition



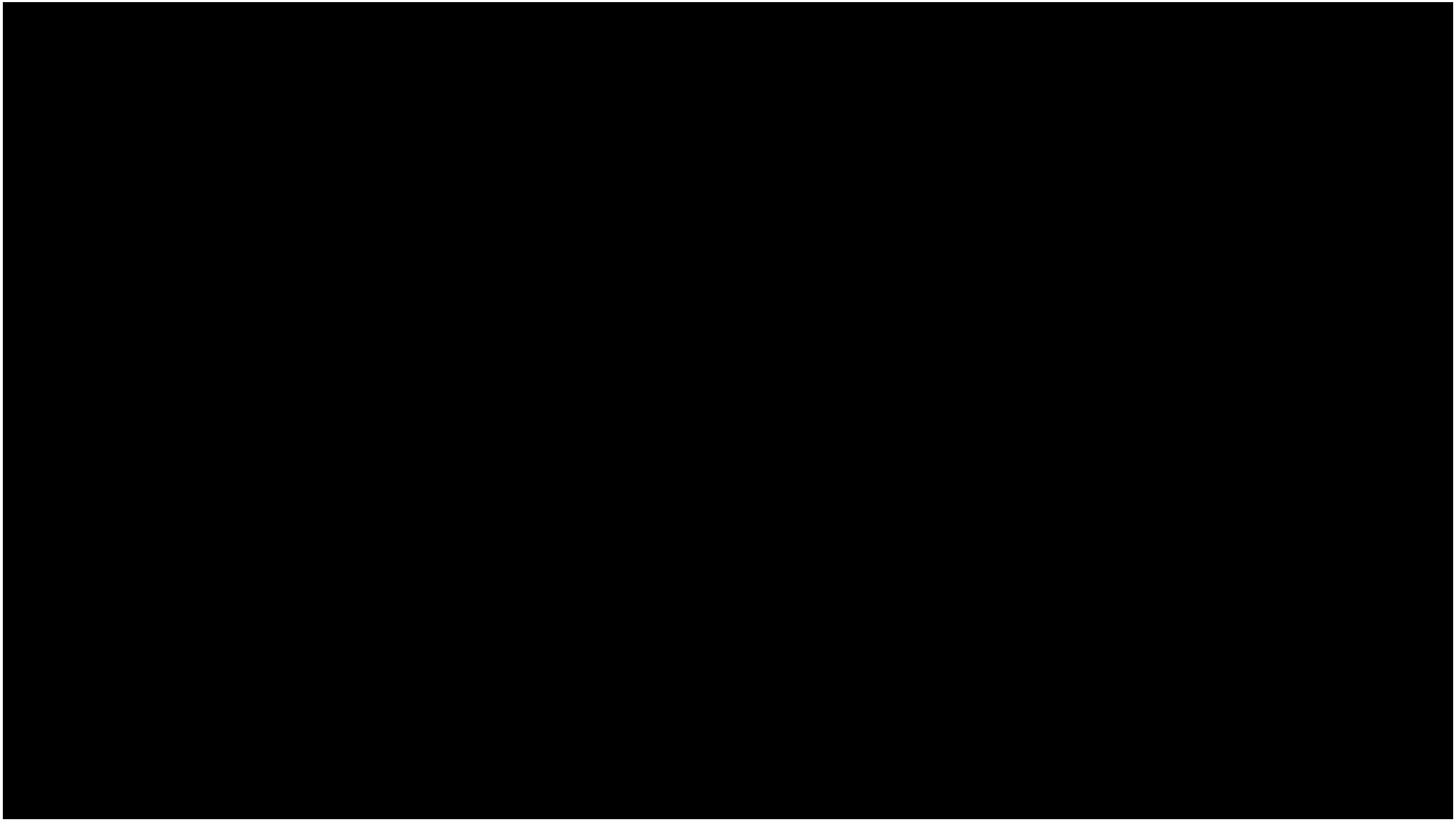
- A POMDP can be defined with the following tuple: $\mathrm{M} = \langle S, A, P, R, \Omega, O \rangle$
  - S, a finite set of states with designated initial state distribution b_0
  - A, a finite set of actions
  - P, the state transition model: $P(s' \mid s, a)$ also denoted $\mathrm{T}(s, a, s')$
  - R, the reward model: $R(s, a)$
  - Ω, a finite set of observations
  - O, the observation model: $O(o \mid s', a)$

POMDP Solutions



- A policy is a mapping $\Omega^* \rightarrow A$
- Finite-horizon problems:
  - Goal is to maximize expected (discounted) reward over a finite horizon
  - Solution can be represented as a **policy tree** with nodes labelled with actions and edges labelled with observations
- Infinite-horizon problems:
  - Goal is to maximize expected discounted reward over an infinite horizon
  - Solution can be represented as a **finite state controller** with nodes labelled with actions and transitions labelled with observations.

Example: Mobile Robot Navigation



Example POMDP: Hallway

Minimize number of steps to the starred square for a given start state distribution

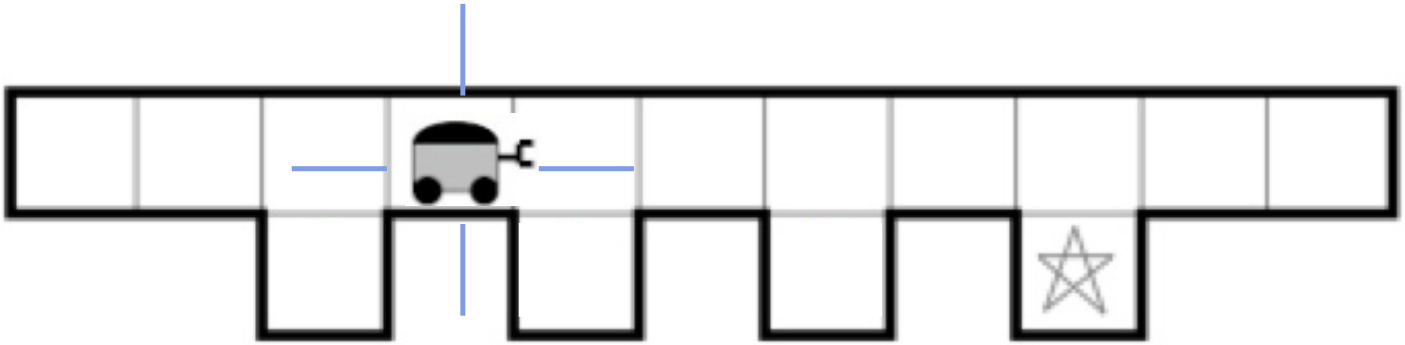
States: grid cells with orientation

Actions: turn  , move forward, stay

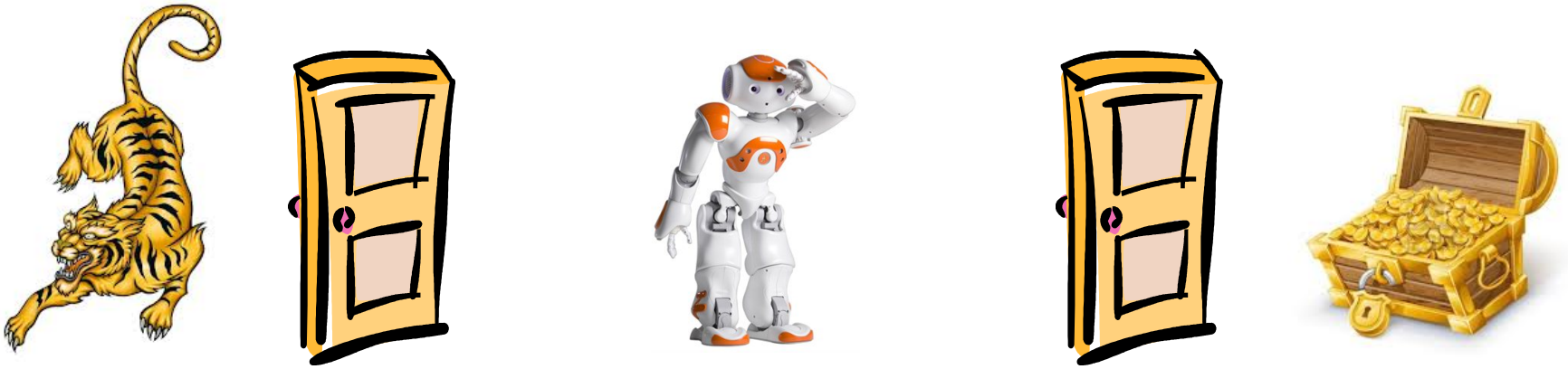
Transitions: noisy

Observations: red lines

Goal: starred square

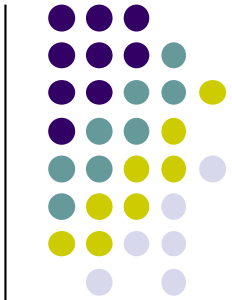


Example: Tiger Problem



- An agent tries to locate tiger and get treasure
- Possible actions: listen, open-left, open-right
- Listening provides a noisy observation (HL or HR)
  Pr(o=HL | TL, listen) = 0.85
- Reward function: small cost per action (-1), modest reward for getting the treasure (+10), large penalty for opening the wrong door (-100).

Applications of POMDPs



- Machine maintenance
- Mobile robot control (deep space; deep-ocean exploration; toxic waste cleanup)
- Machine vision (tracking head/hands, gesture recognition)
- Intelligent control (elevators; weapon allocation)
- Monitoring and controlling the population of various types of marine life
- Networking (troubleshooting; routing)
- Questionnaire design (identify type of person; marketing; education)
- Medical diagnosis and treatment (epilepsy treatment)

Belief States

- A belief state is a probability distribution over the state of the system that can summarize the knowledge of the agent at a given point.

$$b(s_t) = Pr(s_t = s \mid s_0, a_1, o_1, a_2, o_2, \ldots, a_{t-1}, o_{t-1})$$

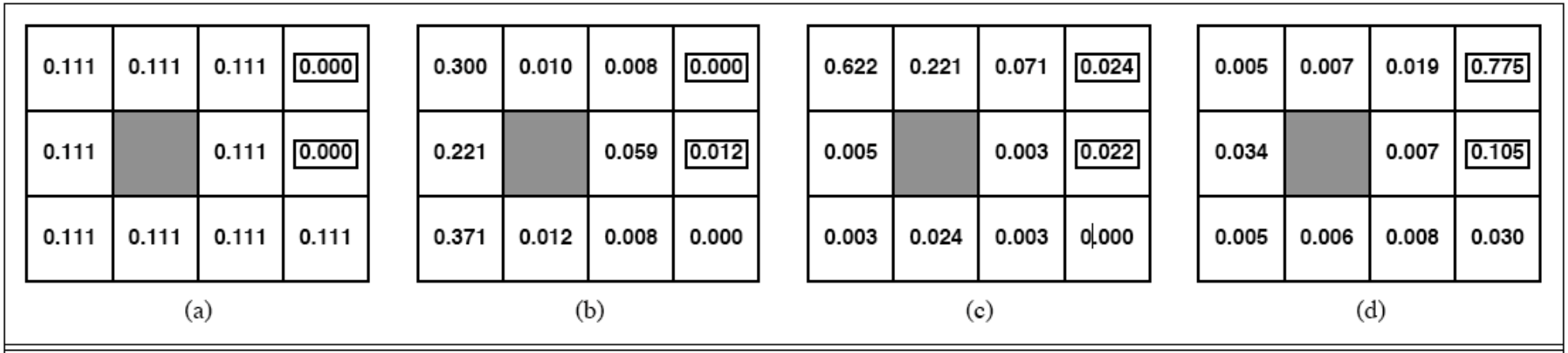
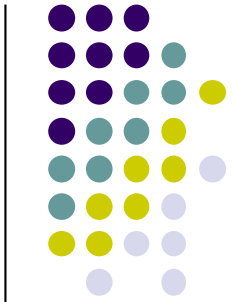


Figure 17.8 (a) The initial probability distribution for the agent's location. (b) After moving *Left* five times. (c) After moving *Up* five times. (d) After moving *Right* five times.

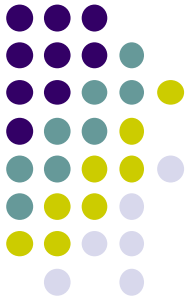
Bayesian Updating of Beliefs



- Suppose that the agent takes action *a* in belief state *b*(*s*) and receives observation *o*
- What is the new belief state *b*'(*s*')?
- Can use Bayesian rule as follows:

$$b'(s') = \Pr(s' \mid b,a,o) = \frac{O(o \mid s',a)\sum_{s \in S} T(s,a,s')b(s)}{\Pr(o \mid a,b)}$$

Bayesian Updating



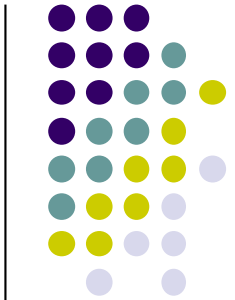
- The probability of the observation can be computed by summing over all possible *s'*

$$P(o \mid a,b) = \sum_{s'} \Pr(o \mid a,s',b) \Pr(s' \mid a,b)$$

$$= \sum_{s'} O(o \mid s',a) \Pr(s' \mid a,b)$$

$$= \sum_{s'} O(o \mid s',a) \sum_{s} T(s,a,s') b(s)$$

Belief State Transition Model



- We can now define a new "belief-state MDP" with the following transition model:

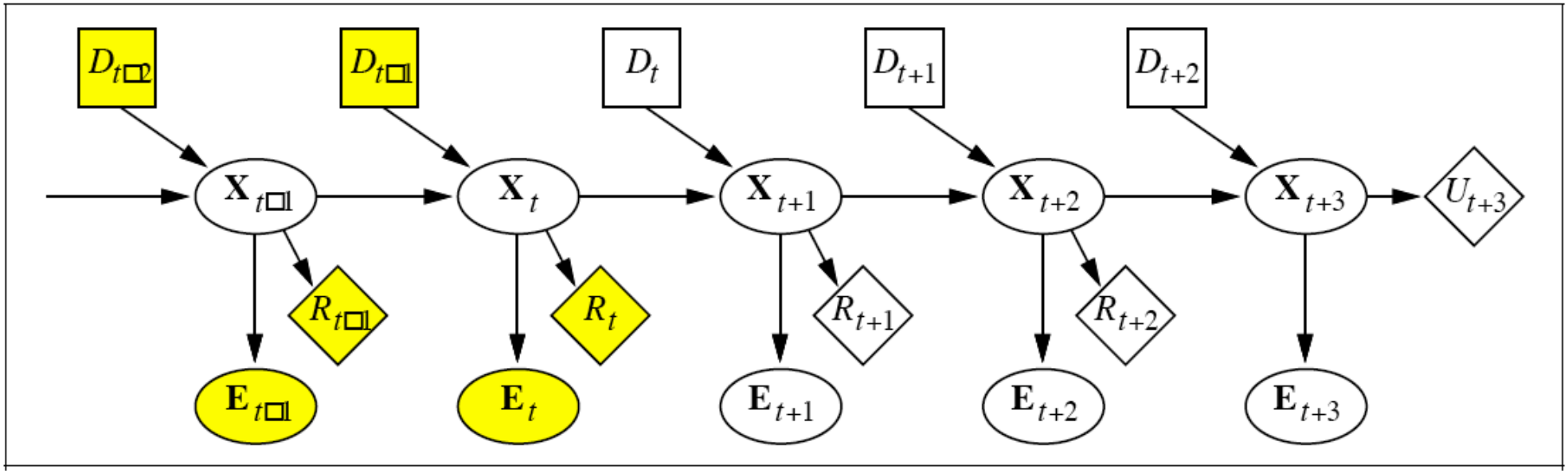
$$\tau(b,a,b') = \Pr(b' \mid a,b) = \sum_o \Pr(b' \mid o,a,b)\Pr(o \mid a,b)$$

$$= \sum_o \Pr(b' \mid o,a,b) \sum_{s'} O(o \mid s',a) \sum_s T(s,a,s')b(s)$$

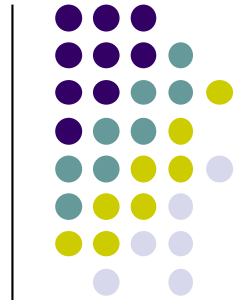
- And the following reward function:

$$\rho(b) = \sum_s b(s)R(s)$$

POMDPs and Dynamic Bayesian Networks



Solution POMDPs Using Decision Trees or Decision Diagrams



D_t in $\mathbf{P}(\mathbf{X}_t \mid \mathbf{E}_{1:t})$

$\mathbf{E}_{t+1}$

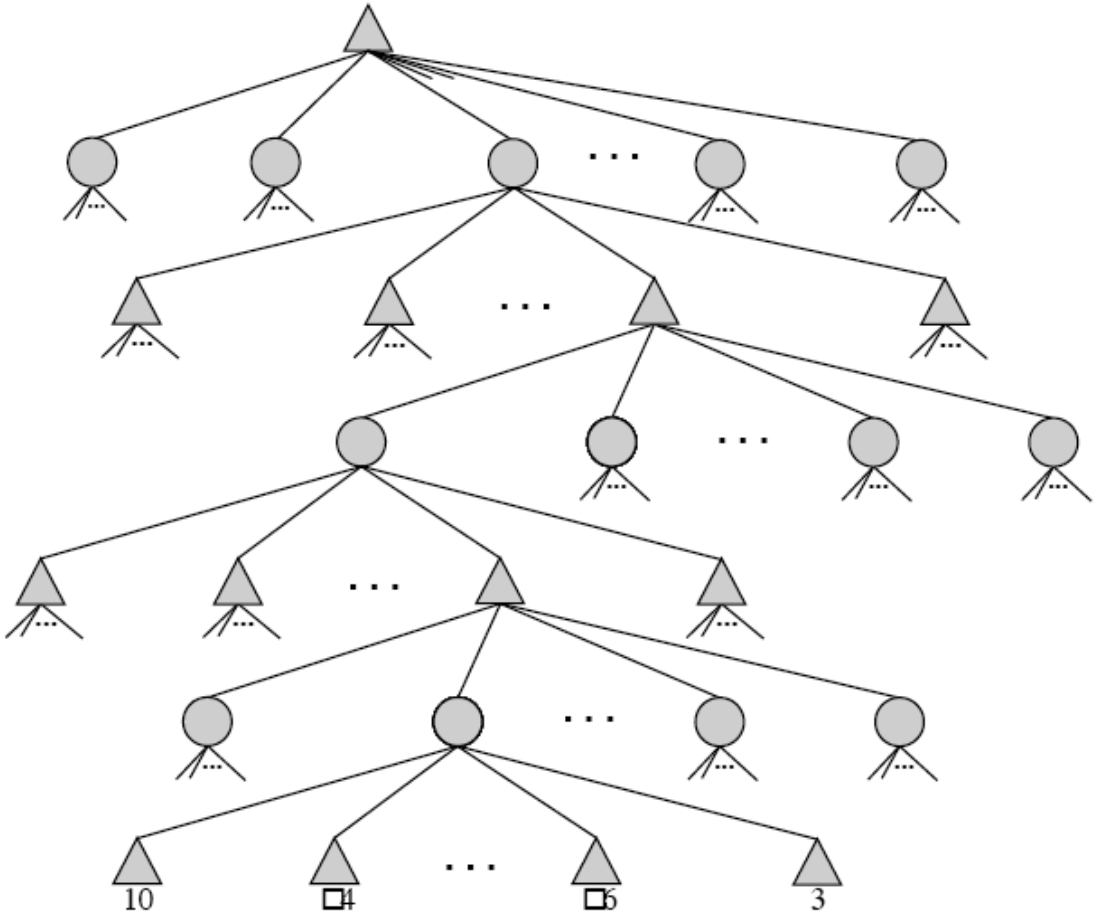
D_{t+1} in $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{E}_{1:t+1})$

$\mathbf{E}_{t+2}$

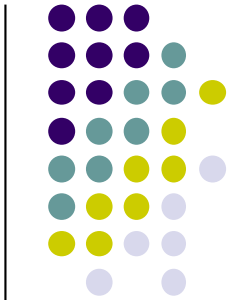
D_{t+2} in $\mathbf{P}(\mathbf{X}_{t+2} \mid \mathbf{E}_{1:t+2})$

$\mathbf{E}_{t+3}$

$U(\mathbf{X}_{t+3})$



Value Iteration Algorithm



- Can express the n-step value function using the (n-1)-state value function as follows:

$$V_n(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) \sum_{s' \in S} \Pr(s' \mid s, a) \sum_{o \in \Omega} \Pr(o \mid s', a) \{ R^a_{ss'o} + V_{n-1}(b^o_a(s')) \right]$$

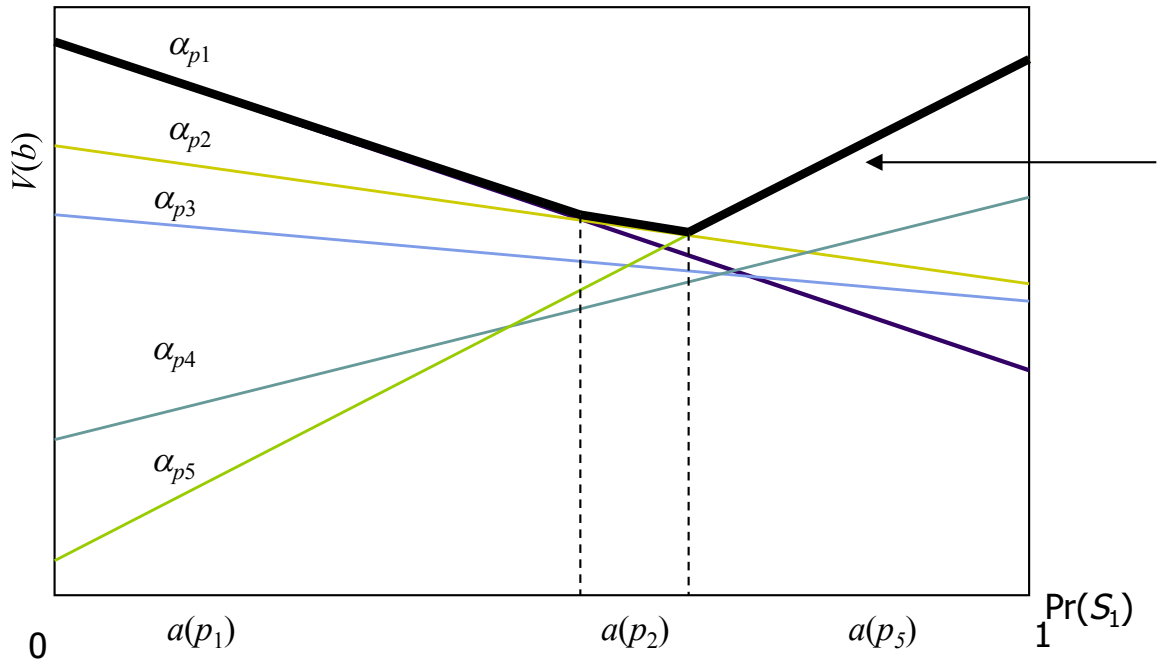
$$= \max_{a \in A} \left[\sum_{s \in S} b(s) R(s, a) + \sum_{s \in S} b(s) \sum_{s' \in S} \Pr(s' \mid s, a) \sum_{o \in \Omega} \Pr(o \mid s', a) V_{n-1}(b^o_a(s')) \right]$$

Expected immediate reward

Value Function Representation

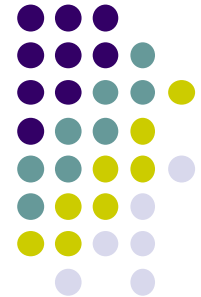
- Piecewise linearity and convexity of optimal value function for finite horizon in POMDPs

$$V_n(b) = \max_k [\sum_{s \in S} \alpha_n^k(s) b(s)] = \max_k [\alpha_n^k \cdot b]$$



Optimal value function

Value Function Update



$$V_n(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \sum_{o \in \Omega} \Pr(o \mid b,a) V_{n-1}(b') \right]$$

$$= \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \sum_{o \in \Omega} \max_{k} \sum_{s} \sum_{s'} b(s) T(s' \mid s,a) \Omega(o \mid s',a) \alpha_{n-1}^{k}(s') \right]$$

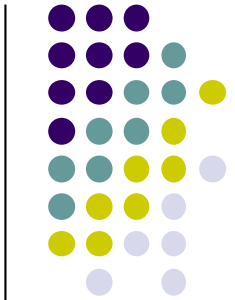
$$= \max_{a \in A} \left[\sum_{s \in S} b(s) \left(R(s,a) + \sum_{o \in \Omega} \sum_{s'} T(s' \mid s,a) \Omega(o \mid s',a) \alpha_{n-1}^{l(b,a,o)}(s') \right) \right] \quad (4)$$

Maximizing to obtain the index l

α-vector of belief point b

Optimal value of belief point b

Policies as Controllers

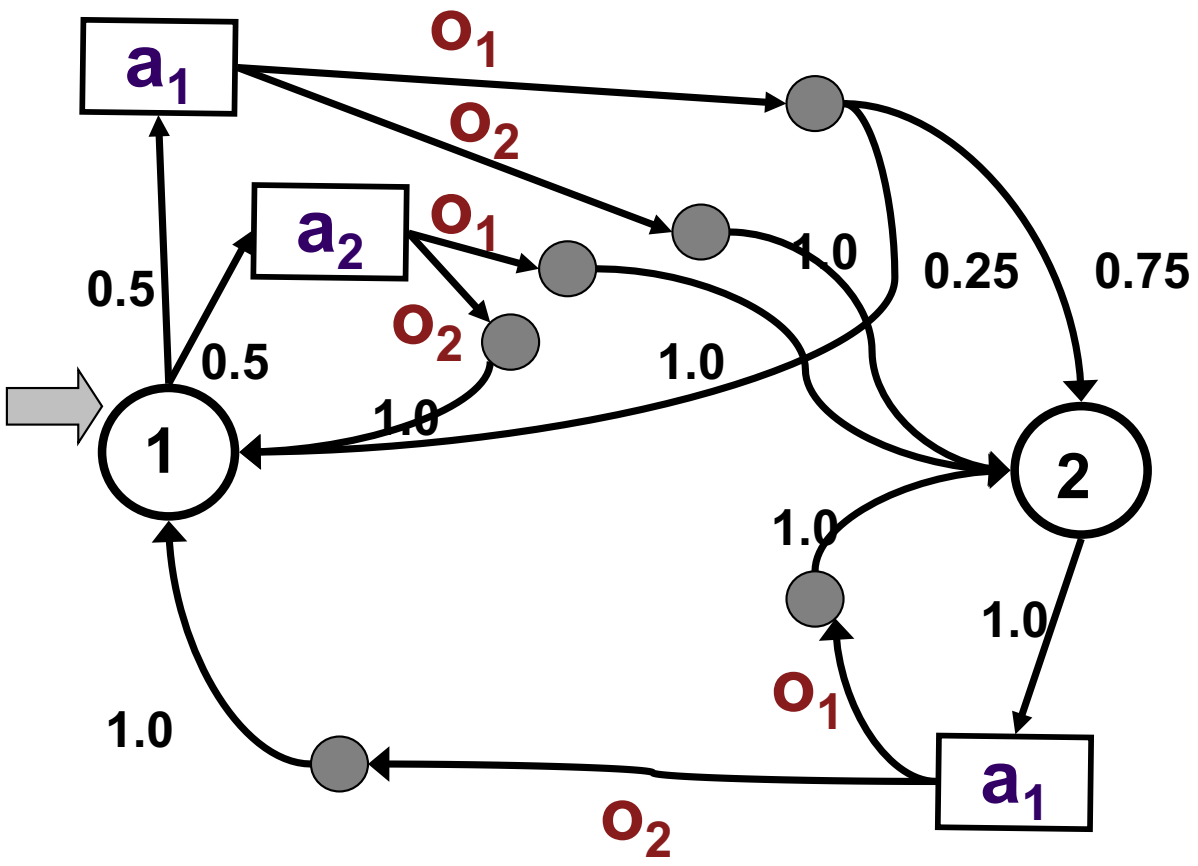


- Fixed memory
- Randomness used to offset memory limitations
- Action selection, $\psi : Q \rightarrow \Delta A$
- Transitions, $\eta : Q \times A \times O \rightarrow \Delta Q$
- Value given by Bellman equation:

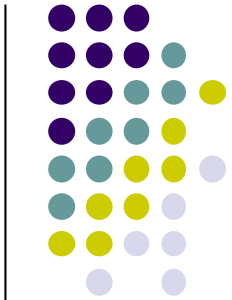
$$V(q,s) = \sum_{a} P(a \mid q) \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \sum_{o} O(o \mid s',a) \sum_{q'} P(q' \mid q,a,o) V(q',s') \right]$$

Controller Example

- Sample stochastic controller
  - 2 nodes, 2 actions, 2 obs
  - Parameters
    - P(a|q)
    - P(q' |q,a,o)
- Why use stochastic controller?
- What do memory states memorize?



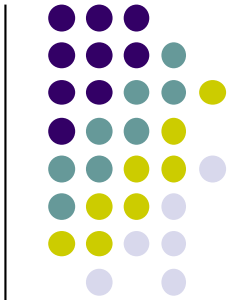
Optimal Controllers



- How do we set the parameters of the controller?
- Deterministic controllers - traditional methods such as search, branch and bound [Hansen 98, Meuleau et al. 99]
- Stochastic controllers - continuous optimization

Gradient Ascent

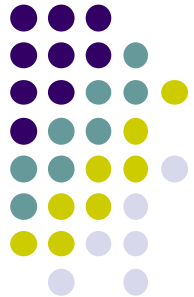
[Meuleau et al. 99]



- Gradient ascent (GA)
  - Create cross-product MDP from POMDP and controller
  - Matrix operations then allow a gradient to be calculated
- Problems with GA
  - Incomplete gradient calculation
  - Computationally challenging
  - Locally optimal

Bounded Policy Iteration (BPI)

[Poupart & Boutilier 03]



- Alternates between improvement and evaluation until convergence
- Improvement: For each node, find a probability distribution over one-step lookahead values that is greater than the current node's value for all states
- Evaluation: Finds values of all nodes in all states

BPI - Linear Program

For a given node, q

Variables: $x(a)$= P(a|q), $x(q',a,o)$=P(q',a|q,o)

Objective: Maximize ε

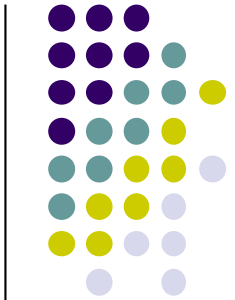
Improvement Constraints: $\forall s \in S$

$$V(q,s) + \varepsilon \leq \sum_a \left[x(a)R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \sum_o O(o \mid s',a) \sum_{q'} x(q',a,o) V(q',s') \right]$$

Probability constraints: $\forall a \in A \quad \sum_{q'} x(q',a,o) = x(a)$

Also, all probabilities must sum to 1 and be greater than 0

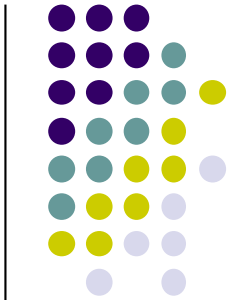
Problems with BPI



- Difficult to improve value for all states
- May require more nodes for a given start state
- Linear program (one step lookahead) results in local optimality
- Must add nodes when stuck - adding nodes via exhaustive backup or incrementally

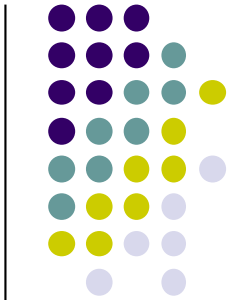
Nonlinear Programming

[Amato, Bernstein & Zilberstein, JAAMAS 10]



- Use nonlinear programming to optimize controllers
- Consider each node value as a variable
- Improvement and evaluation all in one step
- Add constraints to maintain valid values

NLP Intuition



- Value variable allows improvement and evaluation at the same time (infinite lookahead)
- While iterative process of BPI can “get stuck” the NLP can provide the globally optimal solution

NLP Formulation

For variables: $x(q', a, q, o)$ and $z(q, s)$
Maximize

$$\sum_s b_0(s) z(q_0, s)$$

Given the Bellman constraints:

$\forall q, s \; z(q, s) =$

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

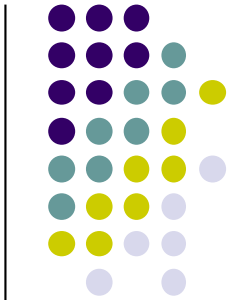
And probability constraints:

$$\forall q, o \; \sum_{q', a} x(q', a, q, o) = 1 \text{ and } \forall q', a, q, o \; x(q', a, q, o) \geq 0$$

$$\forall q, o, a \; \sum_{q'} x(q', a, q, o) = \sum_{q'} x(q', a, q, o_k)$$

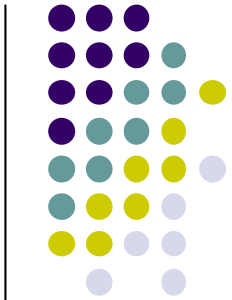
Table 2: The NLP defining an optimal fixed-size POMDP controller. Variable $x(q', a, q, o)$ represents $P(q', a|q, o)$, variable $z(q, s)$ represents $V(q, s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation.

Optimality



Theorem: An optimal solution of the NLP results in an optimal stochastic controller for the given size and initial state distribution.

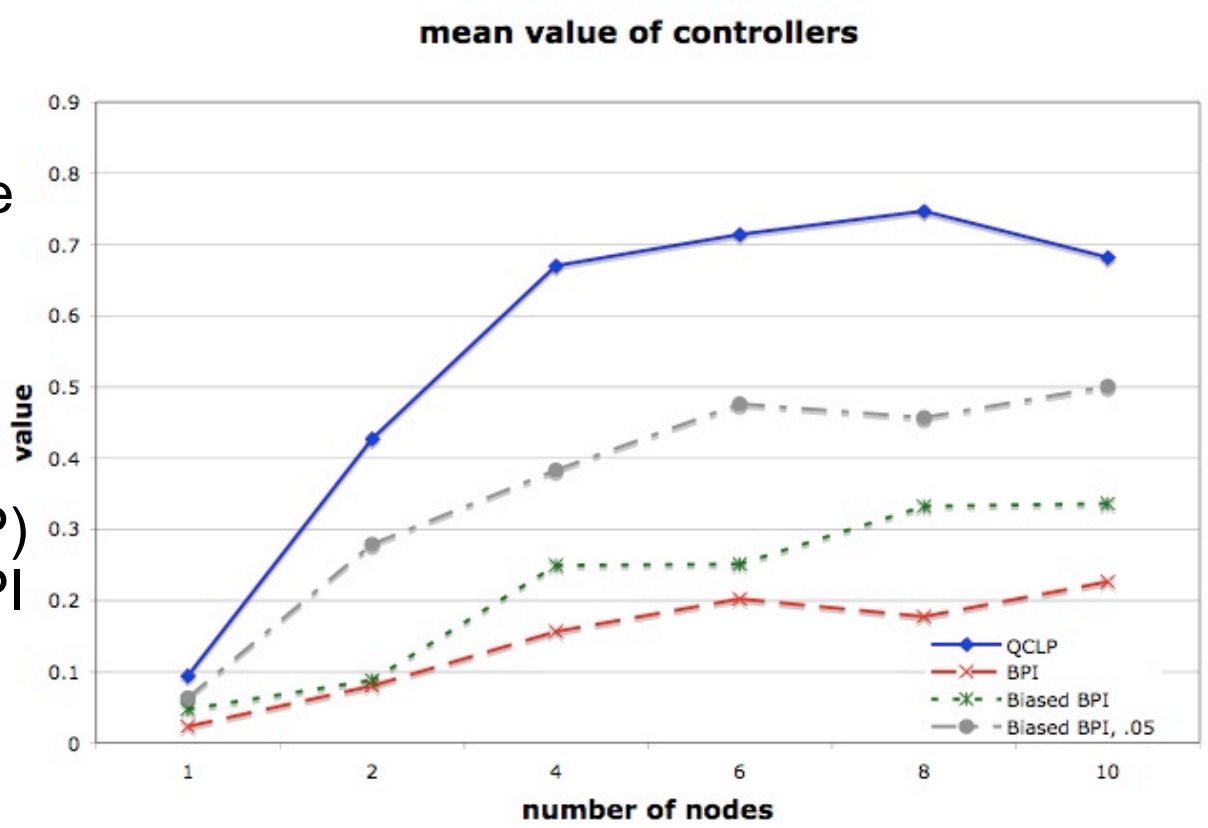
Pros and Cons of NLP Method



- Pros
  - Retains fixed memory and efficient policy representation
  - Represents optimal policy for given size
  - Takes advantage of known start state
- Cons
  - Difficult to solve optimally

Example

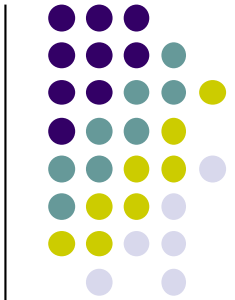
- Using off-the-shelf NLP algorithm (snopt) from the NEOS server.
- 20 random initial controllers for a range of sizes
- Compare the NLP (QCLP) approach with several BPI implementations



mean quality of the NLP (QCLP) and BPI implementations on the hallway domain (57 states, 21 obs, 5 acts)

Other Domains

[Amato, Bernstein & Zilberstein, JAAMAS 10]



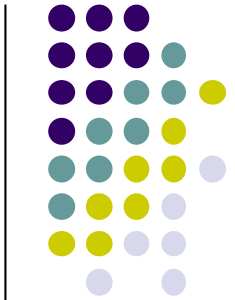
Machine $|S| = 256,\ |A| = 4,\ |\Omega| = 16$

	value	size	time
HSVI2	63.17	575	317
NLOfix	62.65	20	3963
NLO	61.74	10	7350
biased BPI	59.75	20	30831
PERSEUS	39.28	86	2508

Aloha 30 $|S| = 90,\ |A| = 29,\ |\Omega| = 3$

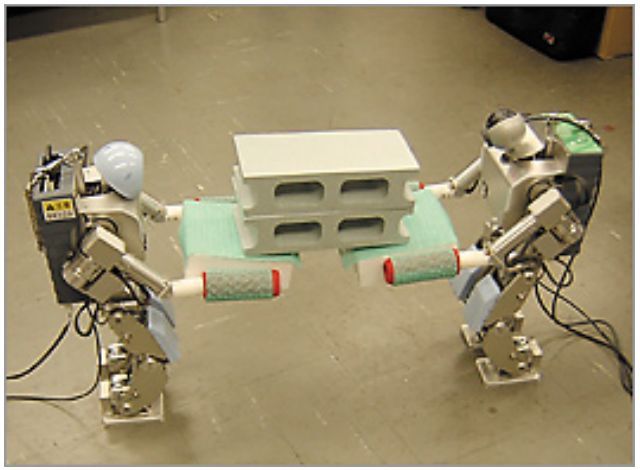
	value	size	time
HSVI2	1212.15	2909	1841
NLO	1211.67	6	1134
NLOfix	1076.49	26	2014
biased BPI	993.22	22	5473
PERSEUS	853.24	114	2512

Outline



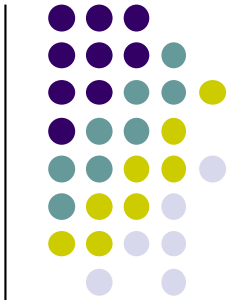
- Planning with Markov decision processes (MDPs)
- Solving MDPs using heuristic search (LAO*)
- Using determinization and other reduced models
- Real time continual planning
- Planning under partial observability (POMDPs)
- **Multiagent planning (Dec-POMDPs)**

Decentralized Decision Making



- **Challenge**: How to achieve intelligent coordination of a group of decision makers in spite of stochasticity and limited information?
- **Key objective**: Develop effective methods to address the uncertainty about the domain, the outcome of actions, and the knowledge, beliefs and intentions of the other agents.

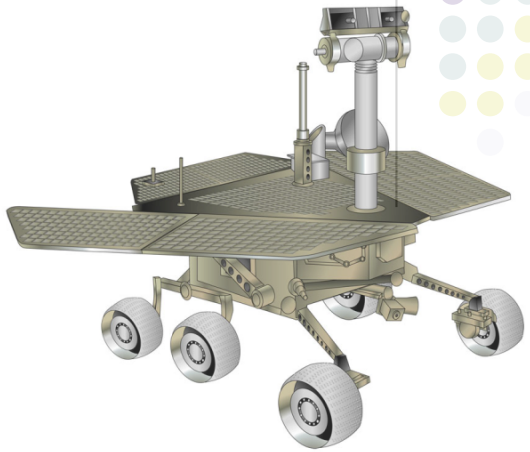
Problem Characteristics

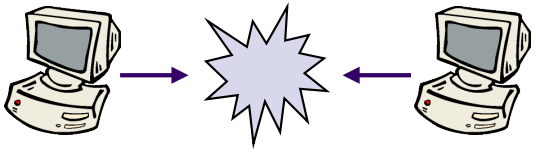


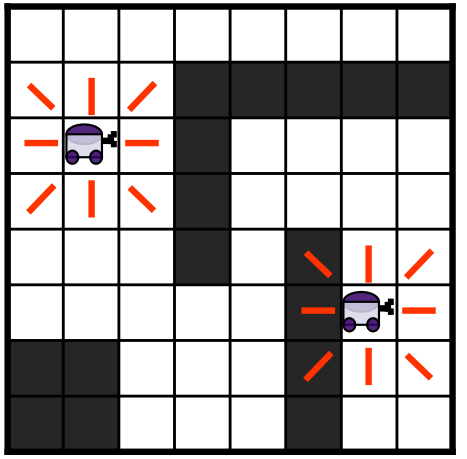
- A **group** of decision makers or agents interact in a stochastic environment
- Each “episode” involves a **sequence** of decisions over **finite or infinite horizon**
- The change in the environment is determined **stochastically** by the **current state** and the **set of actions** taken by the agents
- Each decision maker has **different partial knowledge** of the global state
- Each decision maker may have **different objectives**

Applications

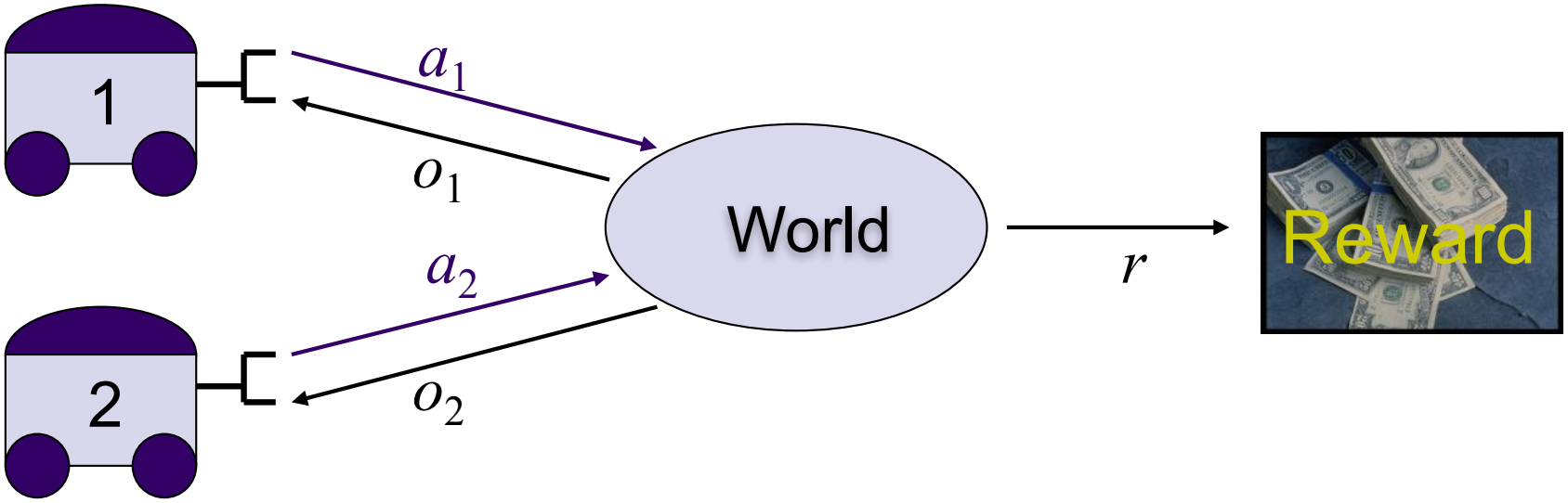
- Autonomous rovers for space exploration
- Protocol design for multi-access broadcast channels
- Coordination of mobile robots
- Load balancing
- Decentralized supply chains
- Sensor network management





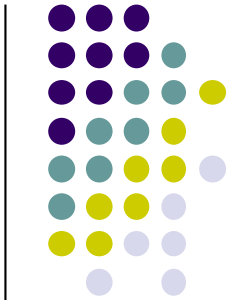


Decentralized POMDP



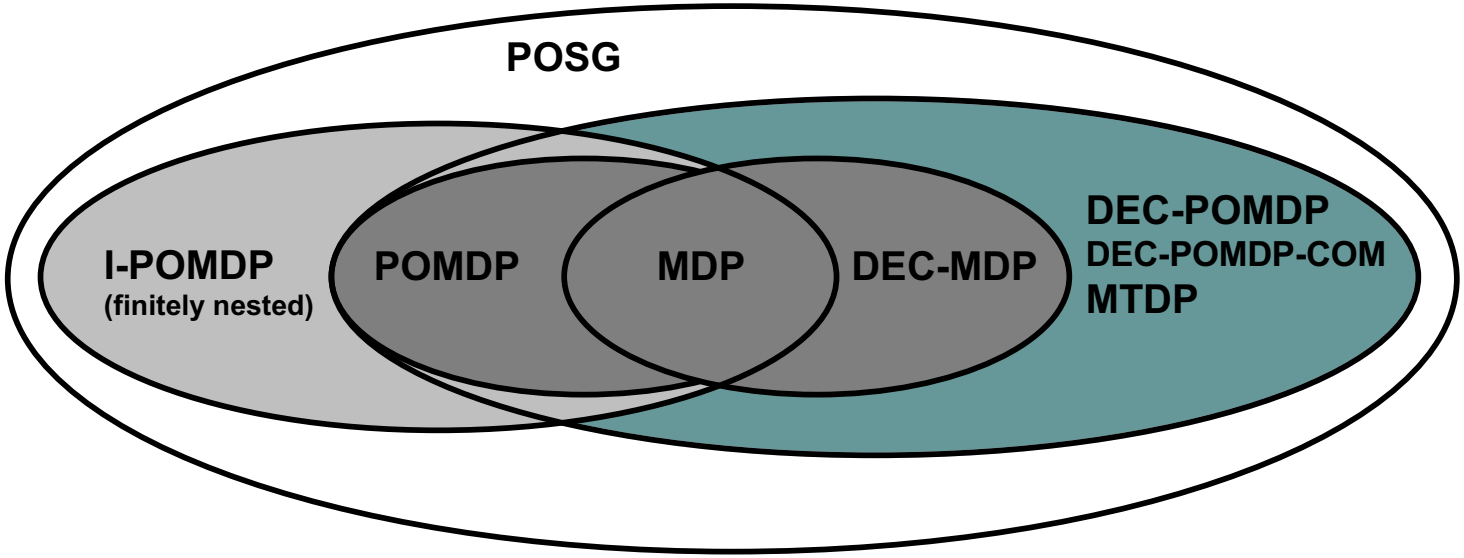
- Generalization of POMDP involving multiple cooperating decision makers with different observation functions

DEC-POMDPs & POSGs



- A POSG is $\langle S, A_1, A_2, P, R_1, R_2, \Omega_1, \Omega_2, O \rangle$, where
  - S is a finite set of domain states, with initial state s_0
  - A_1, A_2 are finite action sets
  - $P(s, a_1, a_2, s')$ is a state transition function
  - $R_1(s, a_1, a_2)$ and $R_2(s, a_1, a_2)$ are reward functions
  - Ω_1, Ω_2 are finite observation sets
  - $O(a_1, a_2, s', o_1, o_2)$ is an observation function
- A DEC-POMDP is a POSG with a single reward function $R(s, a_1, a_2)$
- Straightforward generalization to n agents

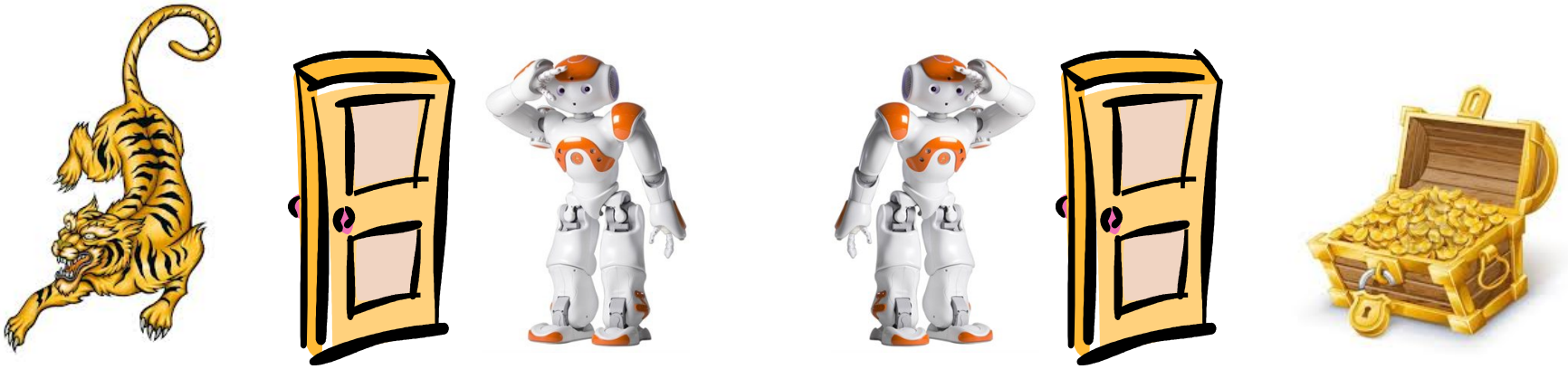
Relationship Between Models



POSG = Partially-Observable Stochastic Game
DEC-POMDP-COM = DEC-POMDP with Communication
I-POMDP = Interactive POMDP
MTDP = Multiagent Team Decision problem

Example: Multiagent Tiger

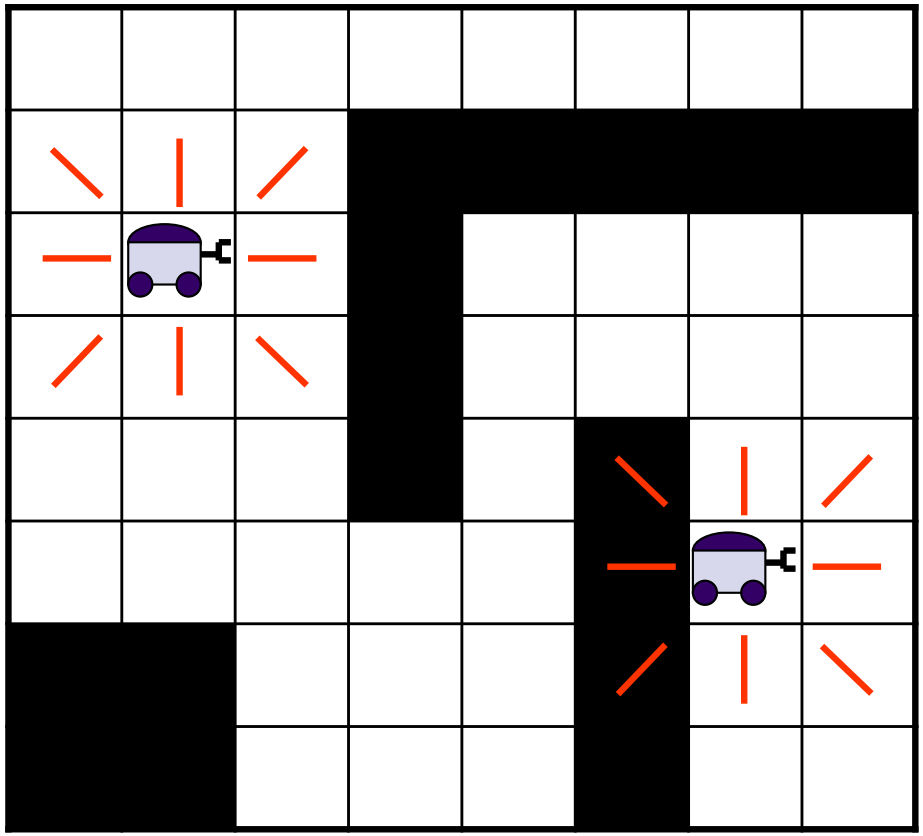
Nair, Tambe, Yokoo, Pynadath & Marsella, 2003



- Two agents try to locate tiger and get treasure
- Each agent may open one of the doors or listen
- Listening provides a noisy observation
- Large penalty for opening door leading to tiger; Large reward for cooperation (choosing same action) and for getting the treasure.

Example: Mobile Robot Planning

[Bernstein, Hansen & Zilberstein, IJCAI 05]



States: grid cell pairs

Actions: ↑,↓,←,→

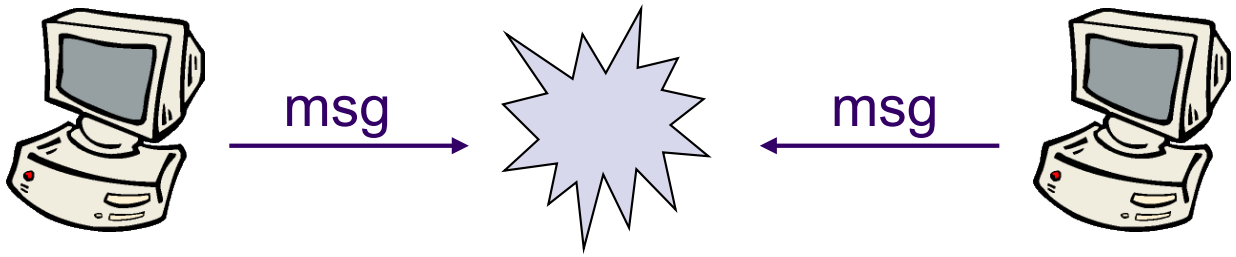
Transitions: noisy

Goal: meet quickly

Observations: red lines

Example: Broadcast Channel

[Hansen, Bernstein & Zilberstein, AAAI 04]

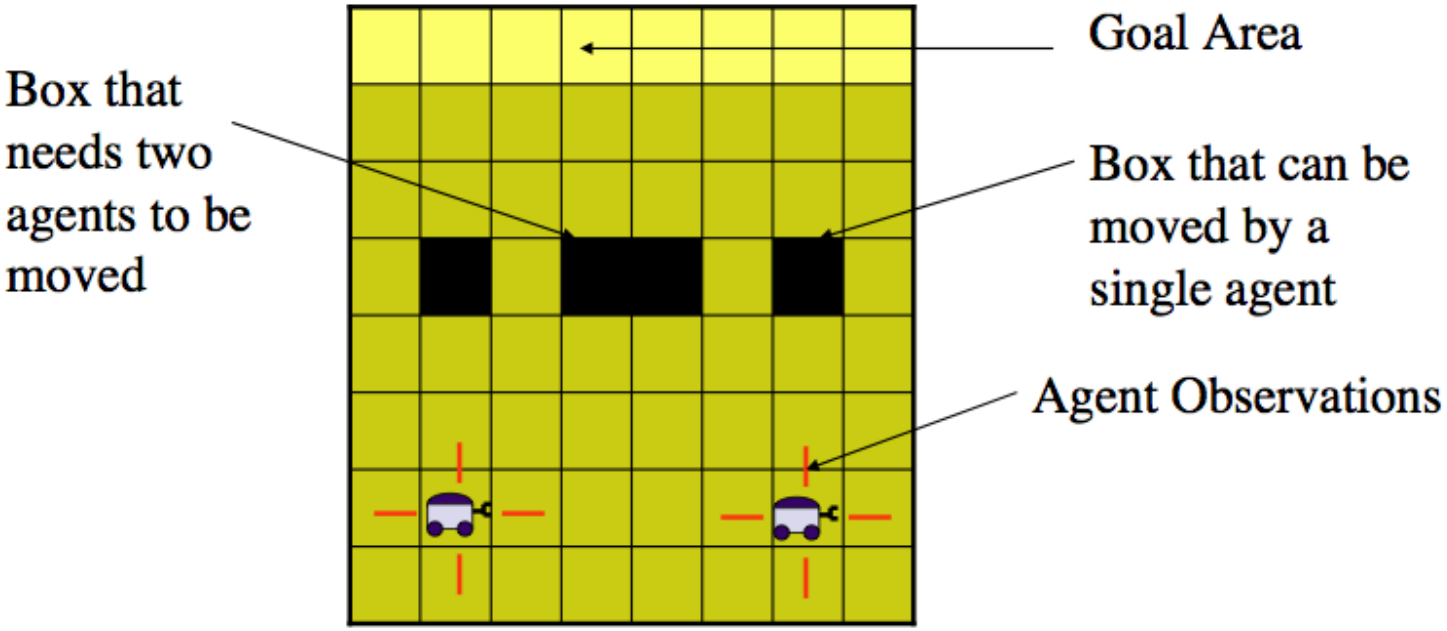


- Messages arrive at nodes randomly
- Collision if two send at once
- Turn-taking strategy not necessarily optimal

States: who has a message to send?
Actions: send or don't send
Reward: +1 for successful broadcast, otherwise 0
Observations: was there a collision?

Example: Cooperative Box-Pushing

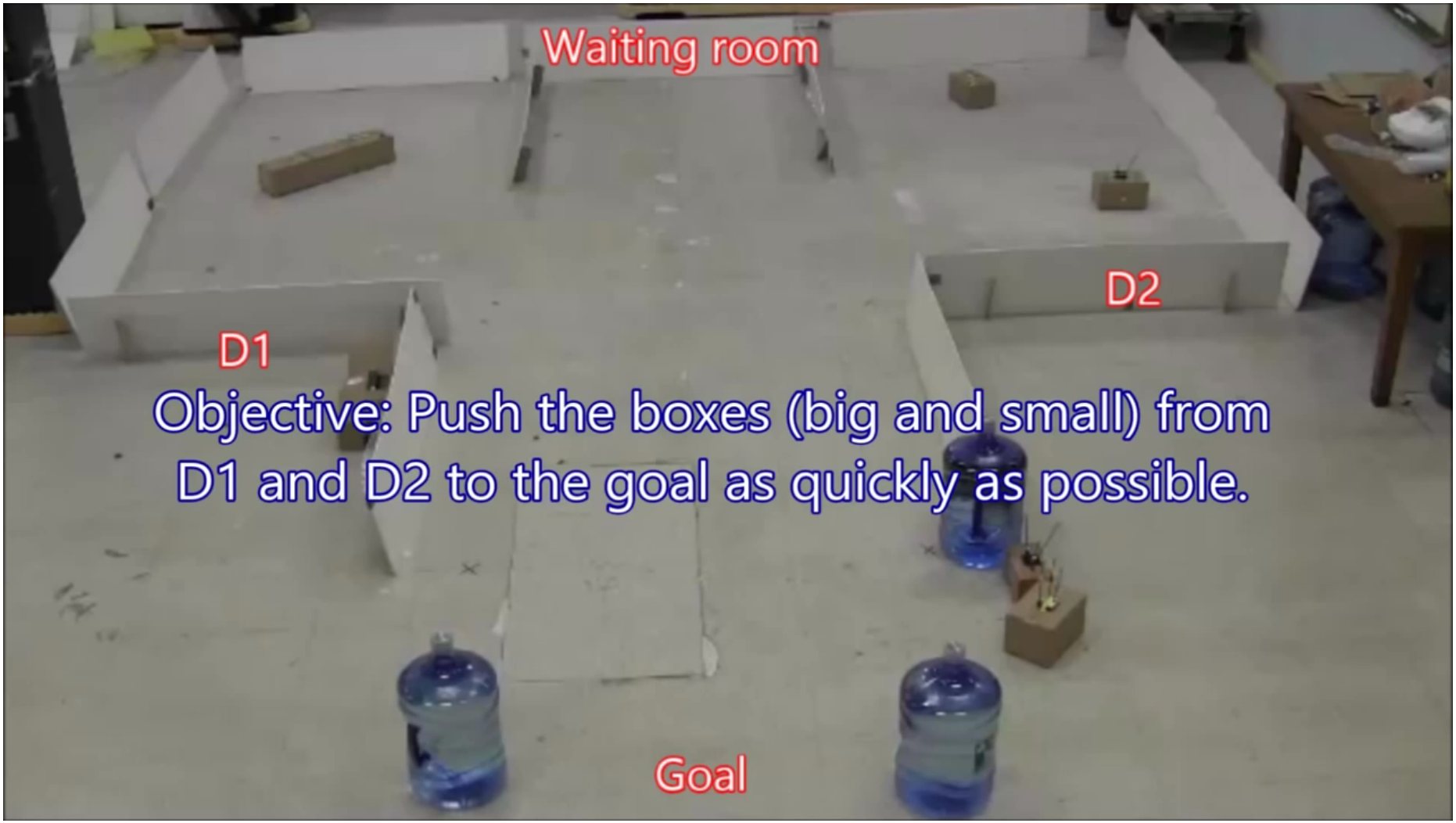
[Seuken & Zilberstein, UAI 07]



Goal: push as many boxes as possible to goal area; larger box has higher reward, but requires two agents to be moved.

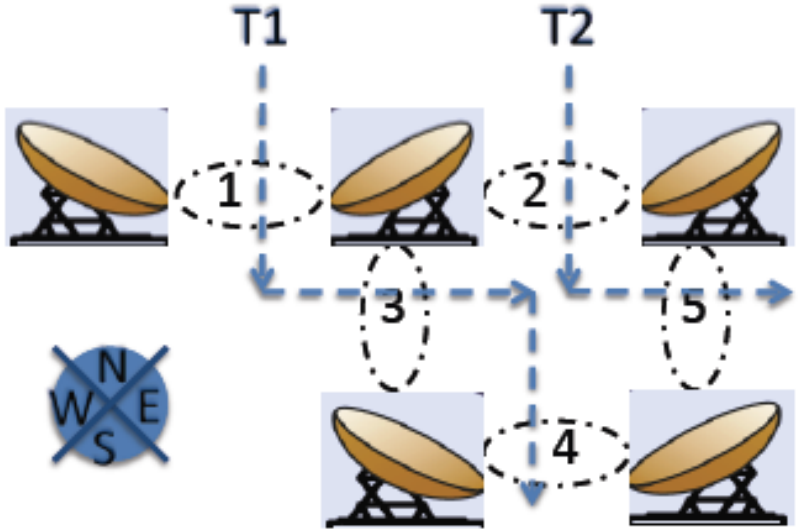
Box Pushing Demo

[Christopher Amato 2016]



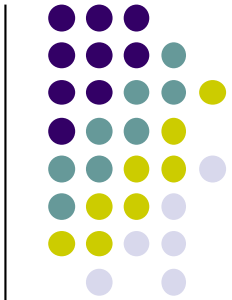
Example: Sensor Network

[Nair et al. 05; Kumar and Zilberstein, 09]



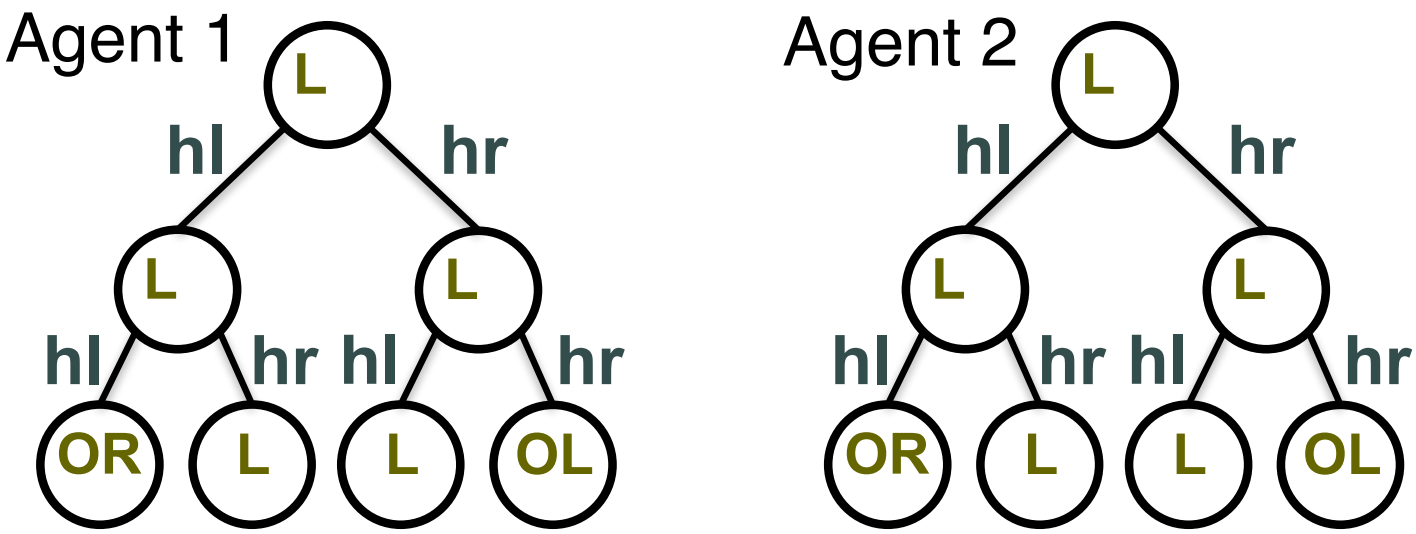
- Multiple sensors track a moving target
- To localize target two neighboring sensors must track it at the same time

Solving DEC-POMDPs



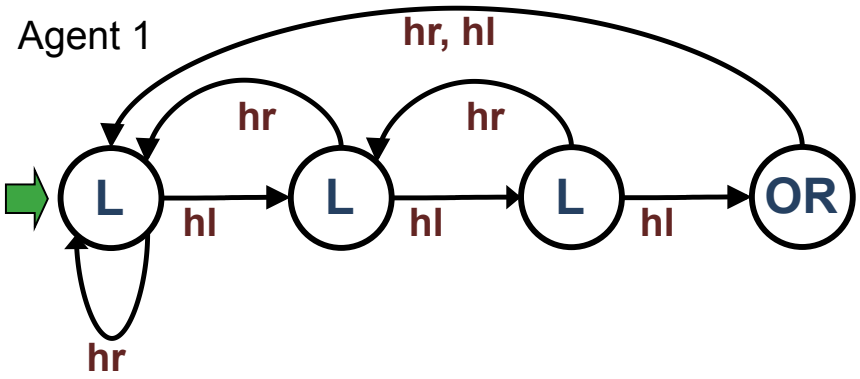
- Each agent's behavior is described by a **local policy** δ_i
- Can be represented as a mapping from
  - Local **observation sequences** to **actions**; or
  - Local **memory states** to **actions**
- Actions can be selected **deterministically** or **stochastically**
- Goal is to find a **joint policy** $< \delta_1 , \delta_2 >$ that maximizes expected reward

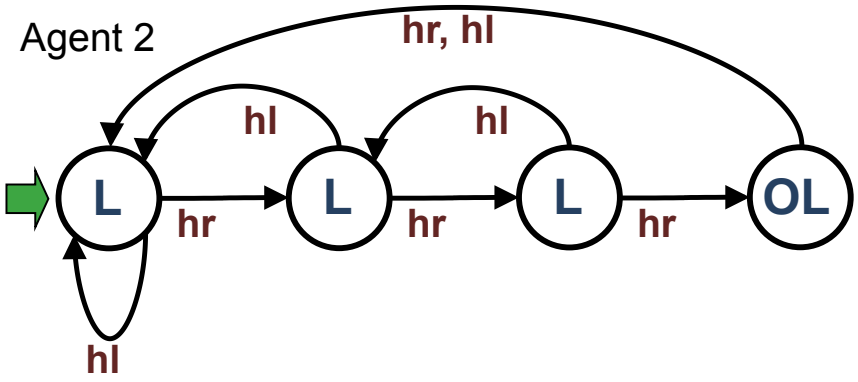
Policy Trees



- Optimal policy trees for the multiagent tiger problem with horizon 3. The policy trees of both agents are the same
- Each node is labeled with an action and each edge is labeled with an observation
- Suitable for finite-horizon problems

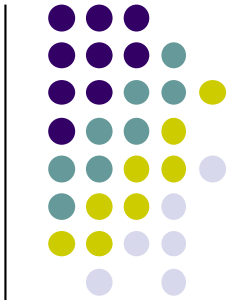
Finite-State Controllers





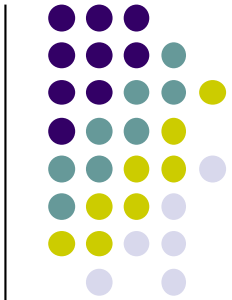
- Optimal four-node deterministic controllers for the multiagent tiger problem
- Each node is labeled with an action and each transition is labeled with an observation
- Suitable for infinite horizon problems.

Some Fundamental Questions



- Are DEC-POMDPs significantly harder to solve than POMDPs? Why?
- What features of the problem domain affect the complexity and how?
- Is optimal dynamic programming possible?
- Can dynamic programming be made practical?
- Is it beneficial to treat communication as a separate type of action?
- What kind of approximation schemes can be developed?

Previous Complexity Results



Finite Horizon

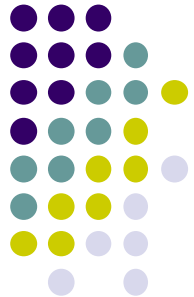
MDP	P-complete (if $T < \|S\|$)	Papadimitriou & Tsitsiklis 87
POMDP	PSPACE- complete (if $T < \|S\|$)	Papadimitriou & Tsitsiklis 87

Infinite-Horizon Discounted

MDP	P-complete	Papadimitriou & Tsitsiklis 87
POMDP	Undecidable	Madani et al. 99

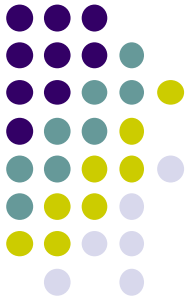
How Hard are DEC-POMDPs?

[Bernstein, Givan, Immerman & Zilberstein, UAI 2000, MOR 2002]



- The complexity of finite-horizon DEC-POMDPs has been hard to establish.
- A **static** version of the problem, where a **single** set of decisions is made in response to a **single** set of observations, was shown to be NP-hard
  [Tsitsiklis & Athan 85]
- We proved that two-agent finite-horizon DEC-POMDPs are **NEXP-hard**
- Provably hard, since P ≠ NEXP
- Probably doubly exponential, as opposed to POMDPs, which are probably exponential

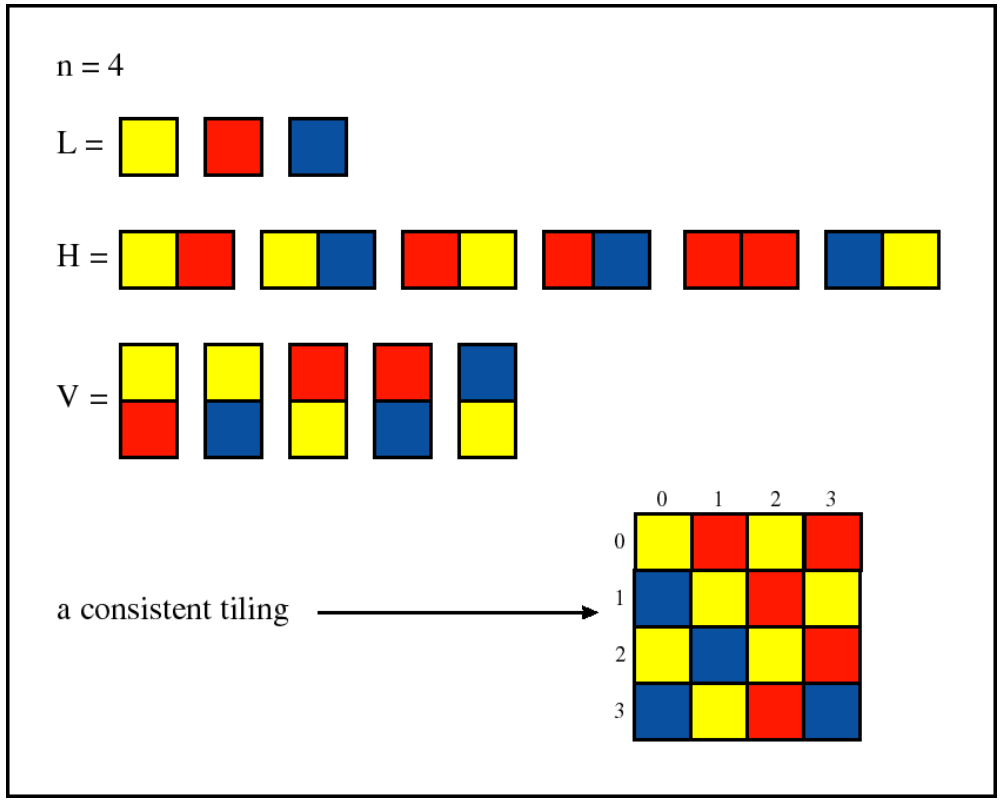
DEC-POMDP $\in$ NEXP



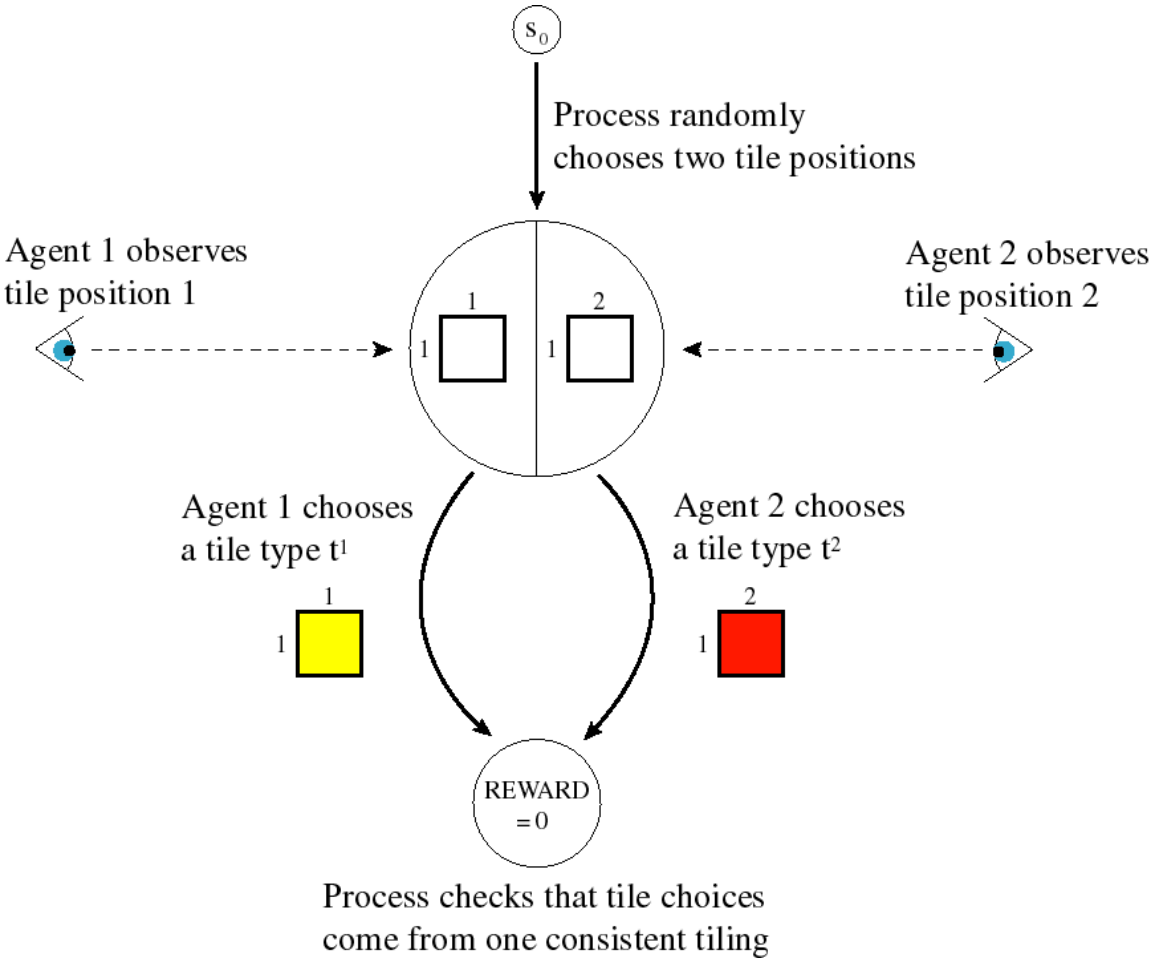
- Guess a joint policy
- The joint policy together with the DEC-POMDP yield an exponentially bigger Markov process with states being observation sequences
- The value of the joint policy can be determined using dynamic programming on this large Markov process

DEC-POMDP is NEXP-hard

- Reduction from **TILING**
- Given n, L, H, V, is there a *consistent* tiling?

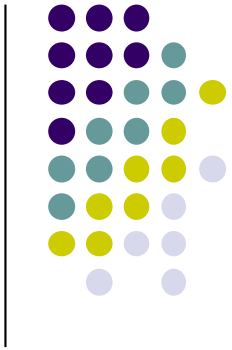


DEC-POMDP is NEXP-hard



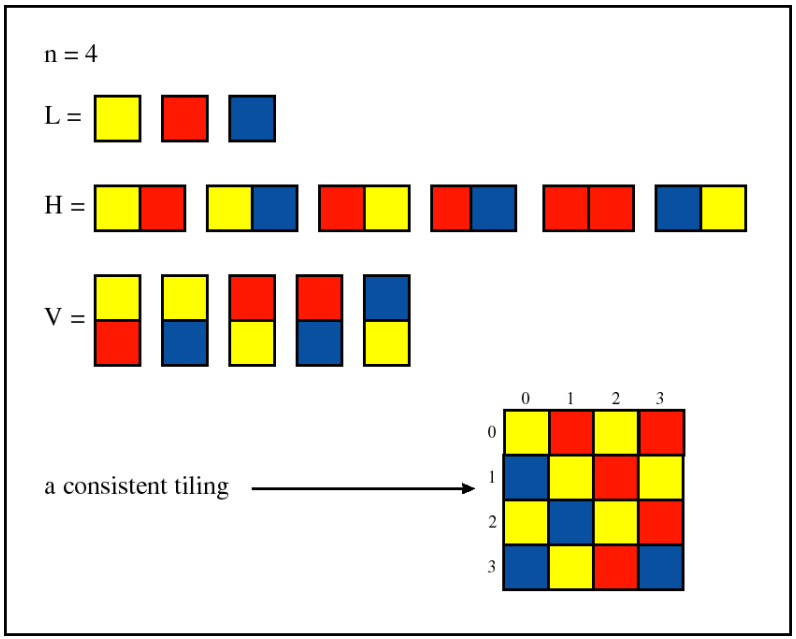
∃ policy with
expected reward 0
⇔
∃ consistent tiling

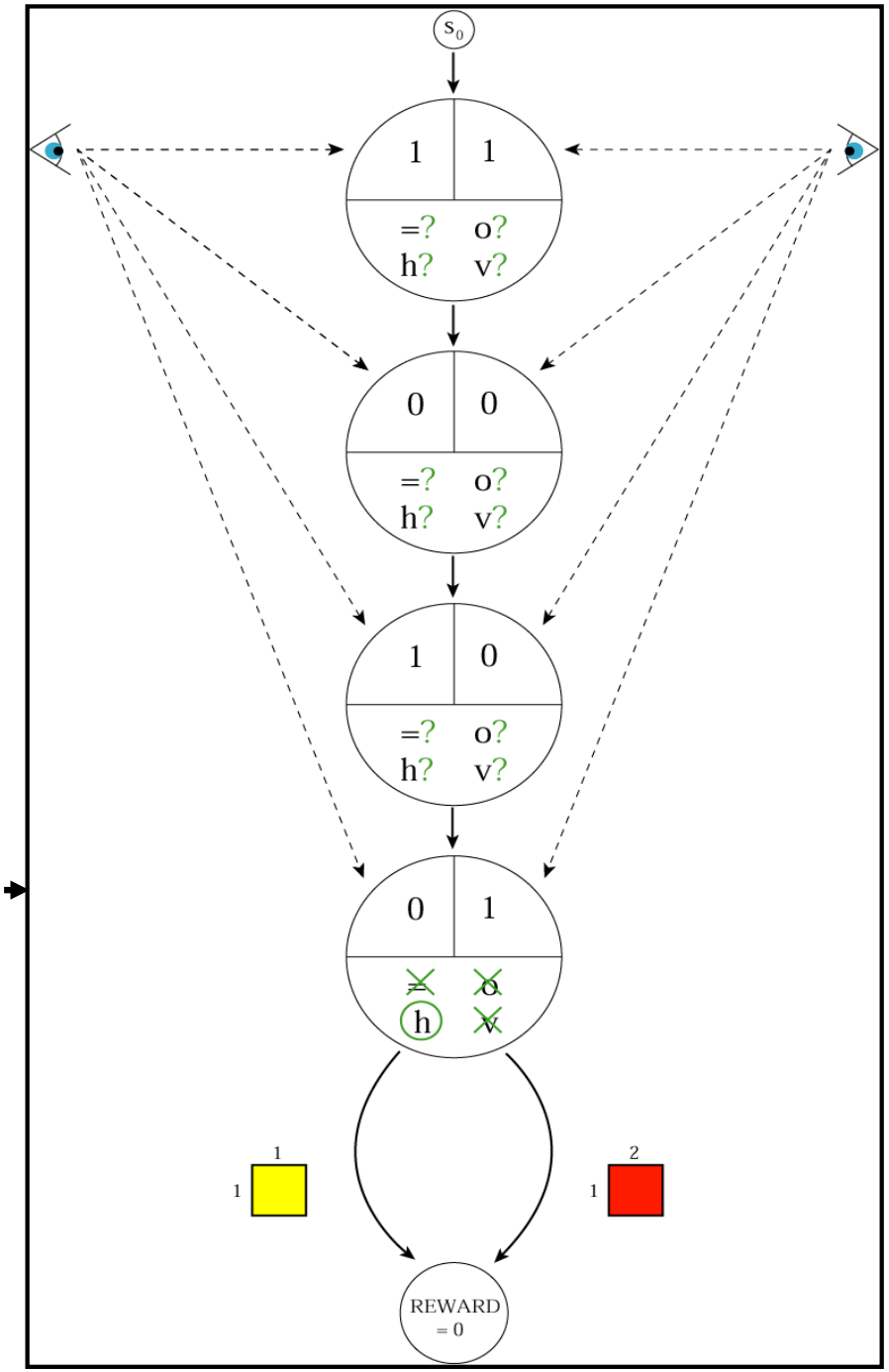
DEC-POMDP is NEXP-hard



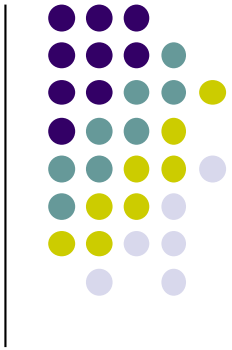
- Naive approach has a state for every pair of tile positions (exponential in size of instance!)
- Luckily, we need only remember info about the *relationship* between the positions
- Generate positions bit-by-bit, and only remember key information:
  - Are they equal?
  - Are they horizontally adjacent?
  - Are they vertically adjacent?

DEC-POMDP is NEXP-hard



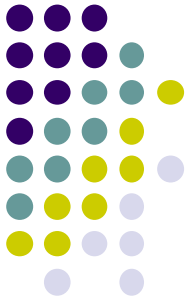


Decentralized MDPs



- What if the agents' observations, when taken together, always uniquely determine the state?
- A DEC-POMDP with this restriction is called a DEC-MDP
- Reduces to an MDP in the single agent case
- The finite-horizon DEC-MDP with two agents is still NEXP-hard

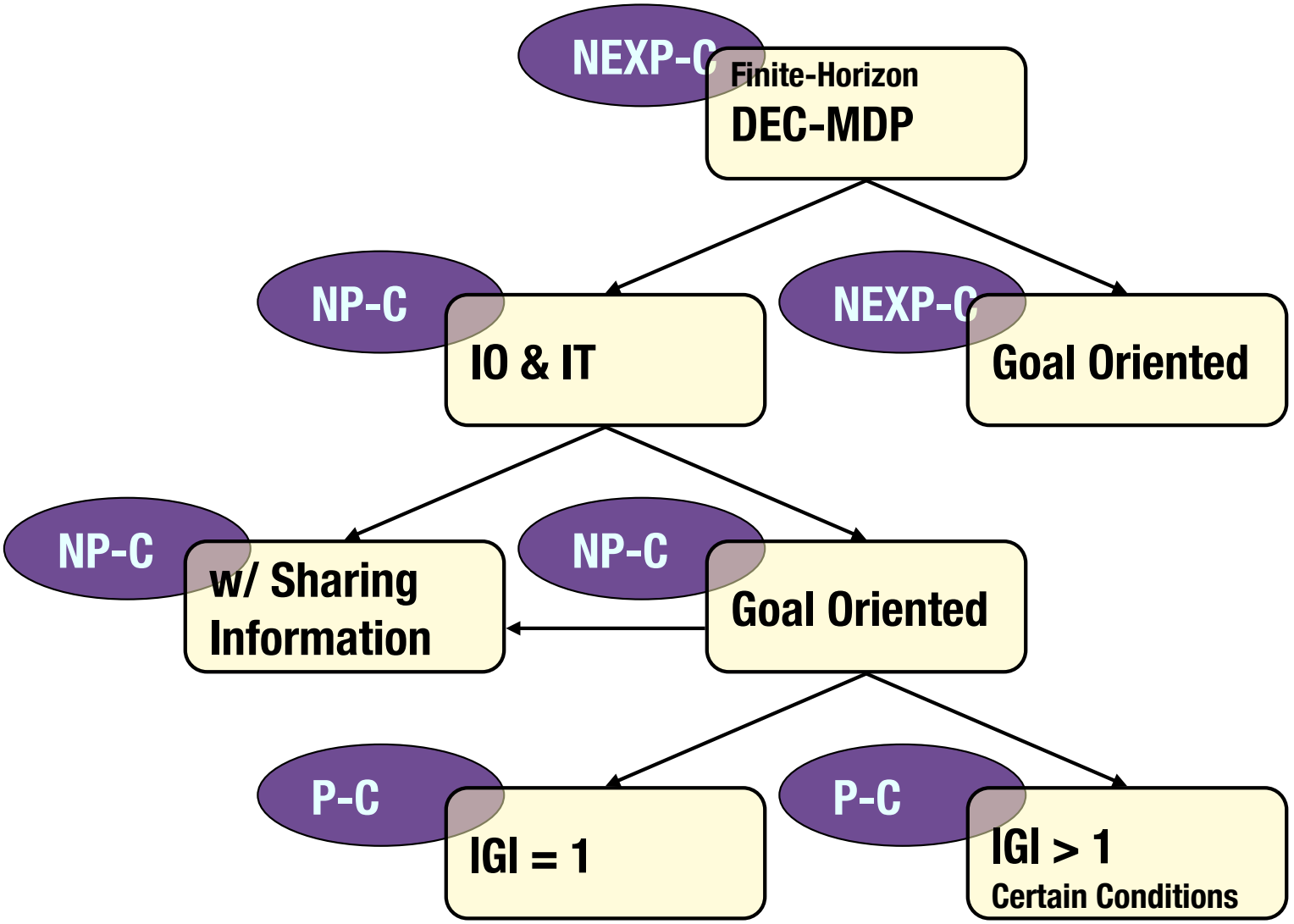
What Features of the Domain Affect the Complexity and How?



- Factored state spaces (structured domains)
- Independent transitions (IT)
- Independent observations (IO)
- Structured reward function (SR)
- Goal-oriented objectives (GO)
- Degree of observability (partial, full, jointly full)
- Degree of interaction
- Degree of information sharing and the cost of communication

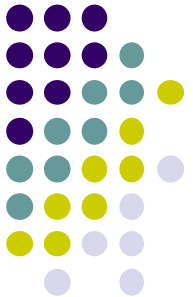
Complexity of Sub-Classes

[Goldman & Zilberstein, JAIR 04]



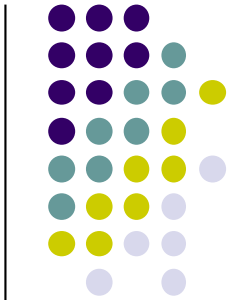
Is Dynamic Programming Possible?

[Hansen, Bernstein & Zilberstein, AAAI 04]

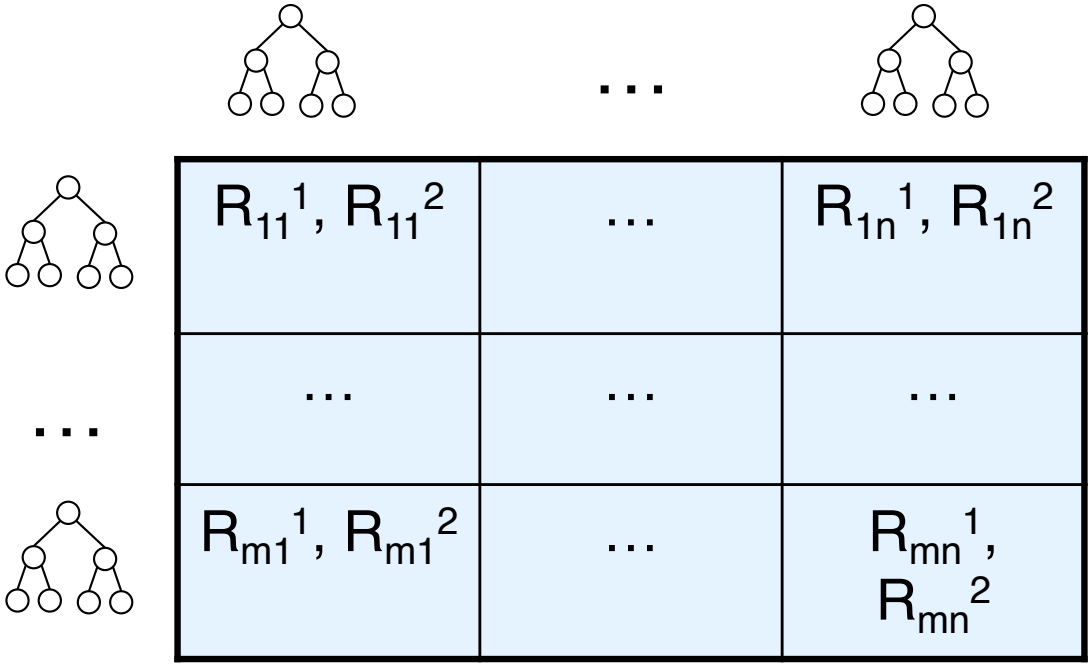


- The key to solving POMDPs is that they can be viewed as **belief-state** MDPs
- **Value iteration** algorithms have been developed [Smallwood & Sondik 73]
- However, it is not clear how to define a belief-state MDP for a DEC-POMDP
- We developed the first exact DP algorithm for finite-horizon DEC-POMDPs
- The algorithm works well also for competitive situations modeled as POSGs

Strategy Elimination

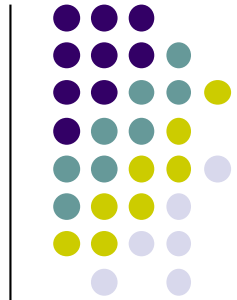


- Any finite-horizon DEC-POMDP can be converted to a **normal form game**
- But the number of strategies is **doubly exponential** in the horizon length!

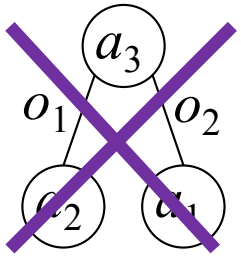


A Better Way to Do Elimination

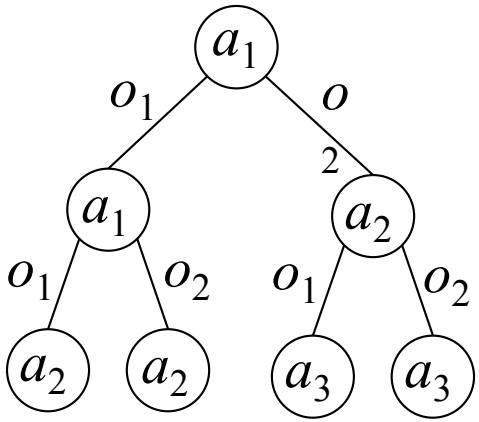
[Hansen, Bernstein & Zilberstein, AAAI 04]

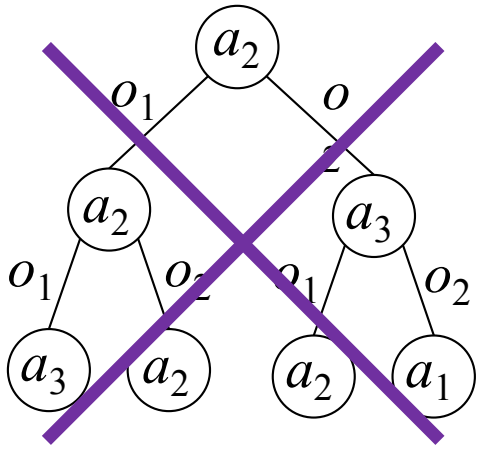


- We can use dynamic programming to eliminate dominated strategies **without** first converting to **normal form**
- Pruning a subtree eliminates the set of trees containing it



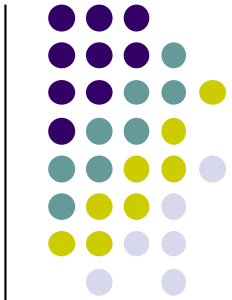
prune





eliminate

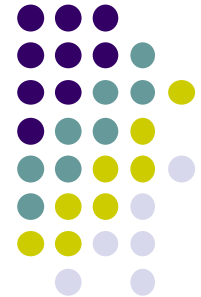
Generalizing Dynamic Programming



- Build policy trees as in single agent case
- Pruning rule is a natural generalization

	What to prune	Space for pruning
Normal form game	strategy	Δ(strategies of other agents)
POMDP	policy tree	Δ(states)
POSG DEC-POMDP	policy tree	Δ(states $\times$ policy trees of other agents)

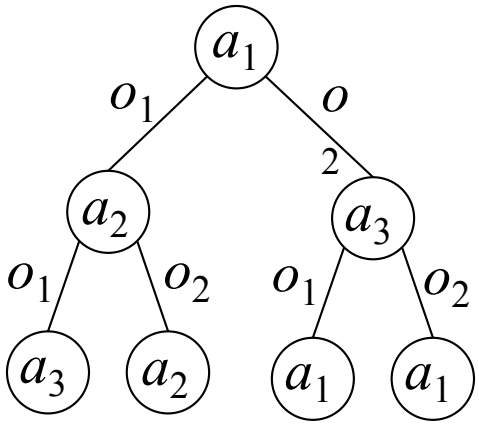
Dynamic Programming for POMDPs



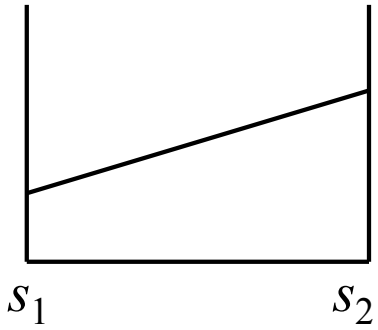
- The basic concepts:

s_1	0.25
s_2	0.40
s_3	0.35

belief state

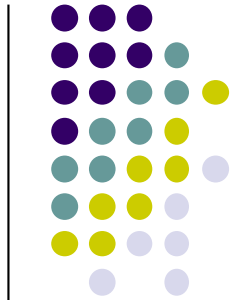


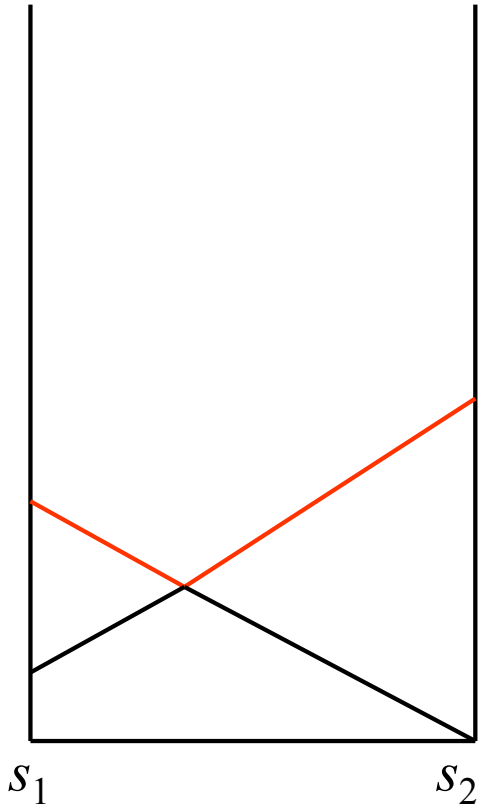
policy tree



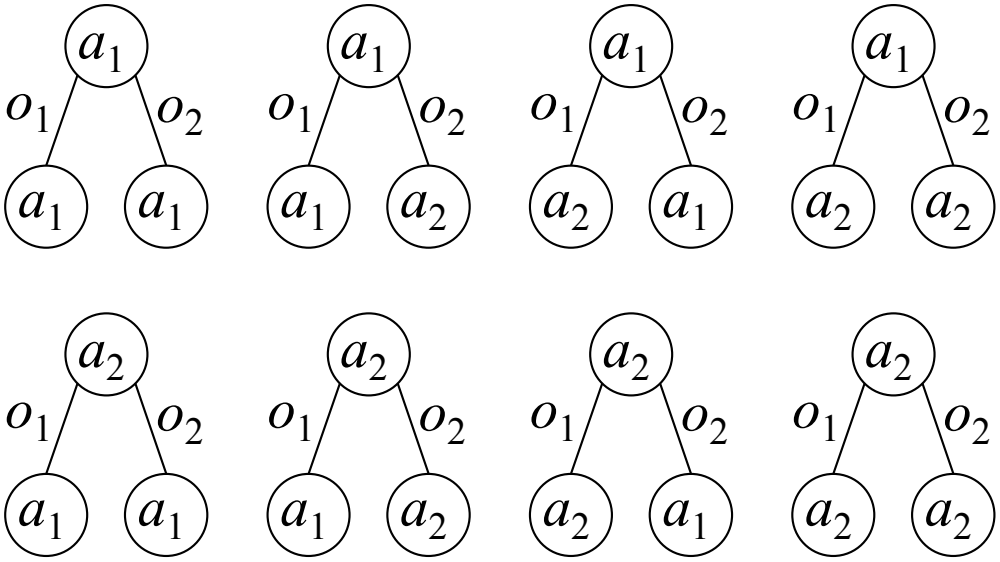
linear value function

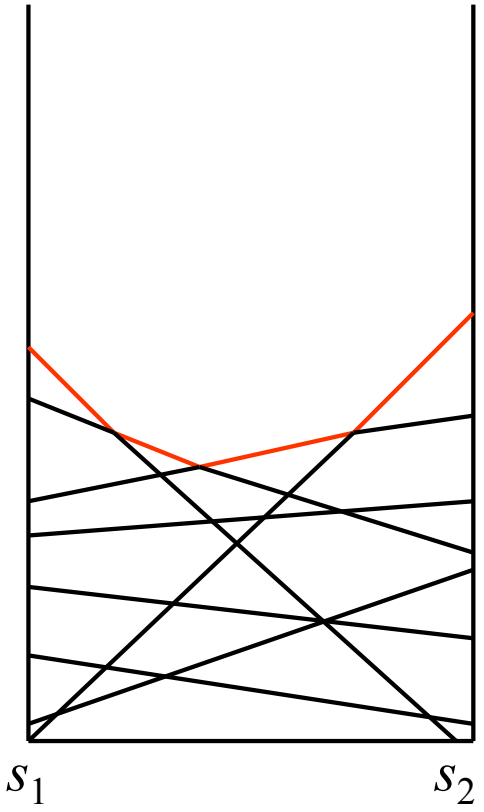
Dynamic Programming



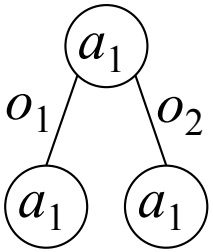


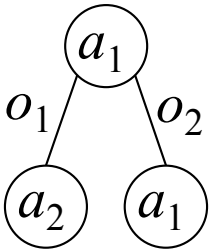
Dynamic Programming

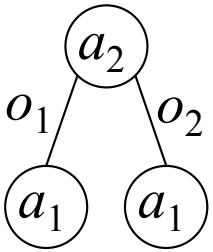


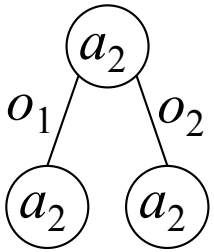


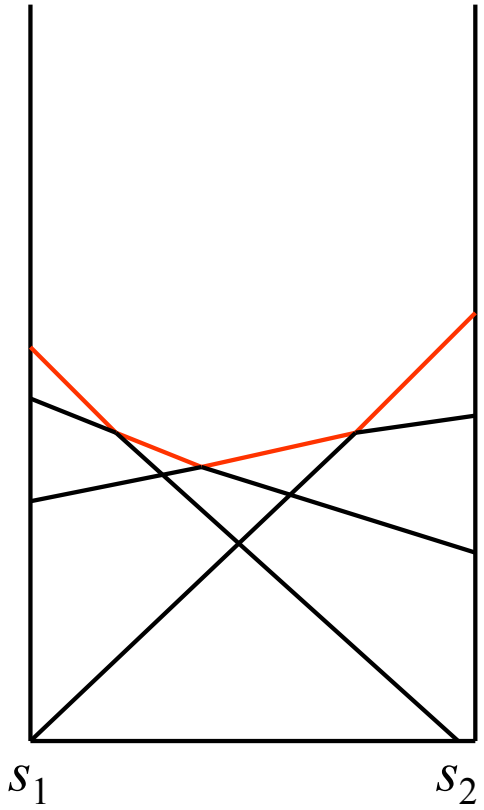
Dynamic Programming



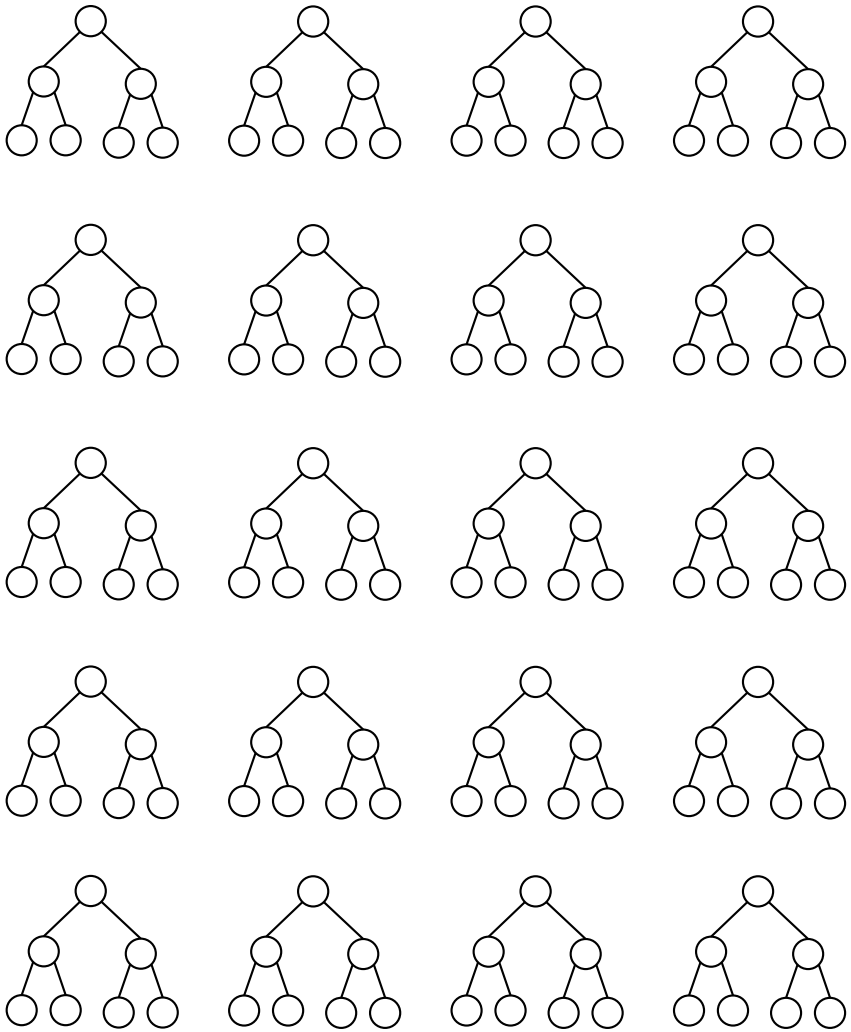


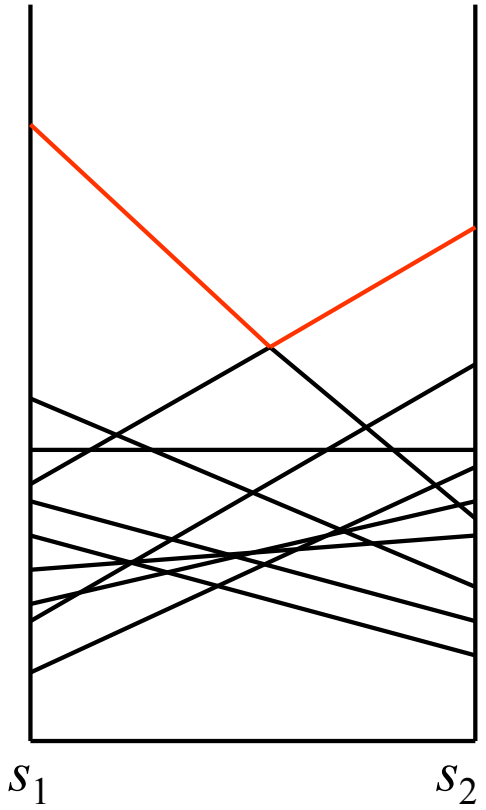




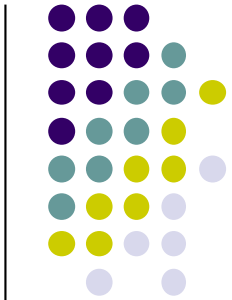


Dynamic Programming



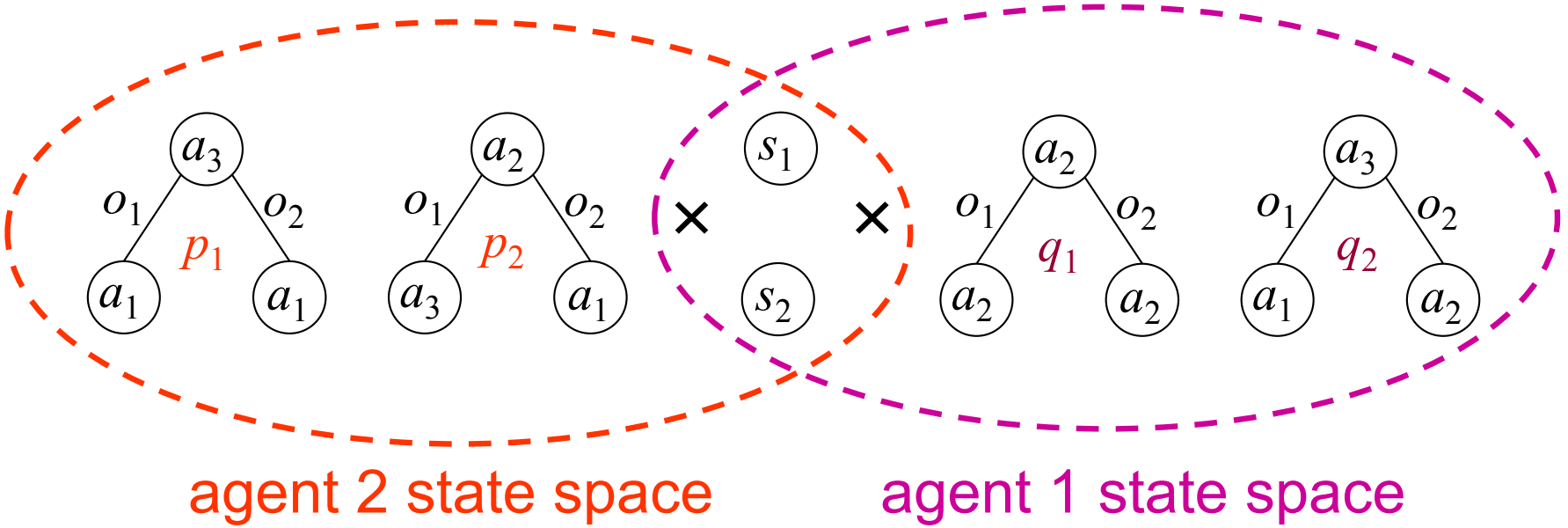


DP for DEC-POMDPs

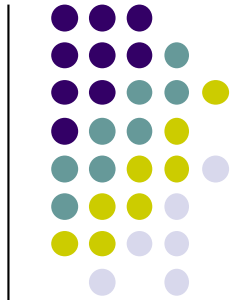


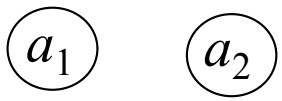
- Backups done simultaneously for all agents
- Pruning uses a generalized belief state space
- Pruning can be performed by solving a linear program
- This yields an optimal algorithm

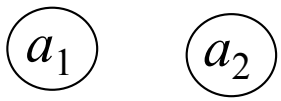
Generalized Belief State



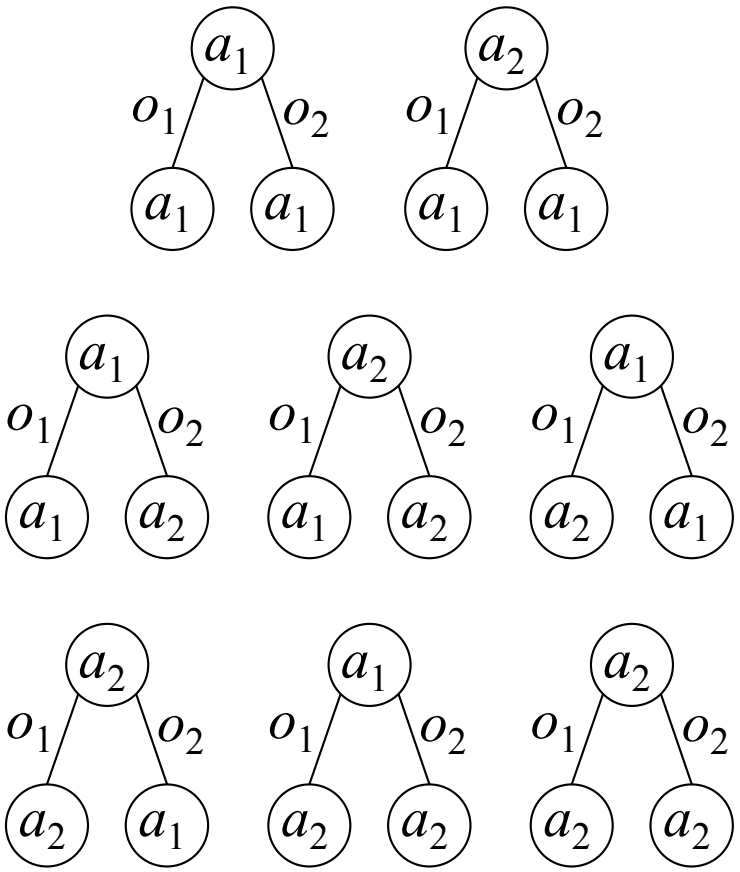
Dynamic Programming

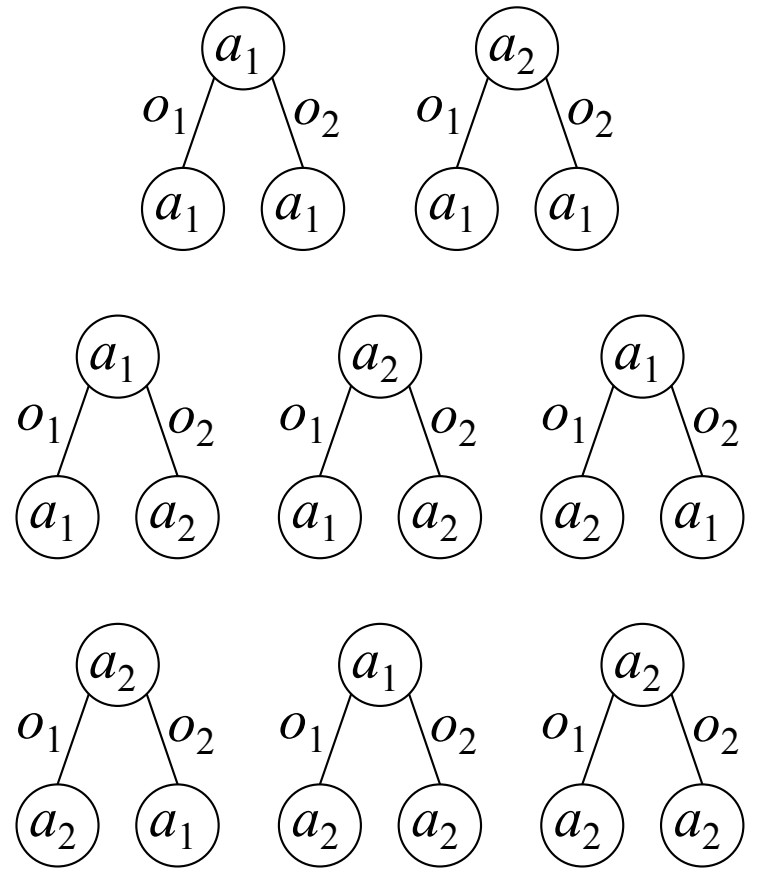




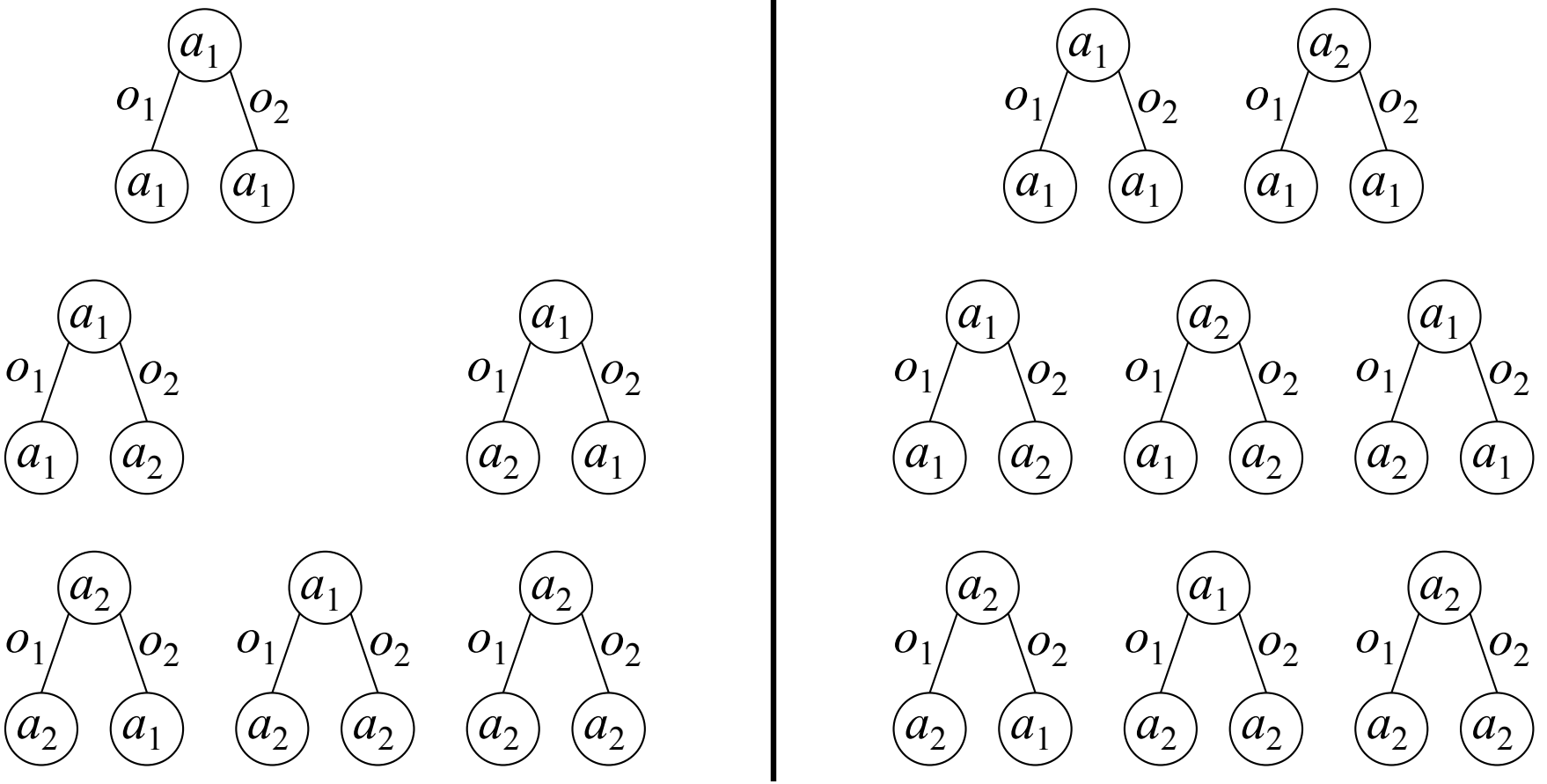


Dynamic Programming

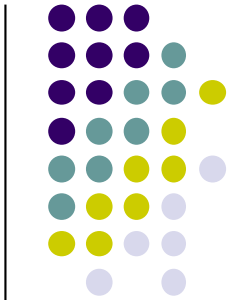


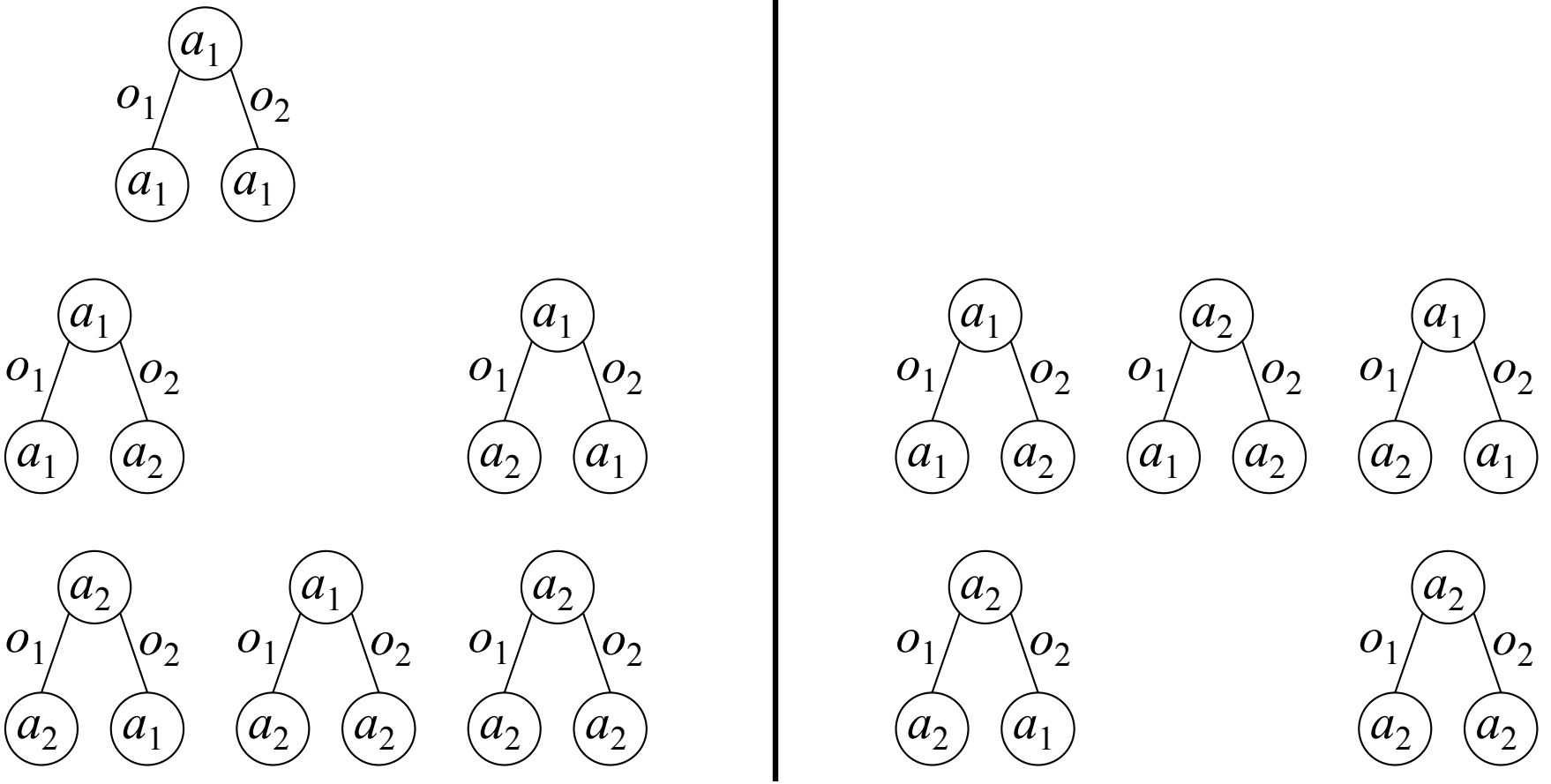


Dynamic Programming

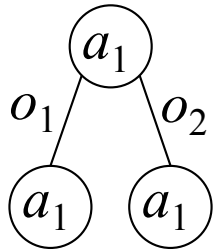


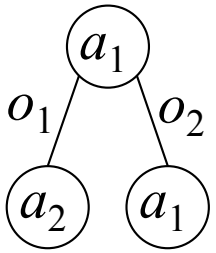
Dynamic Programming

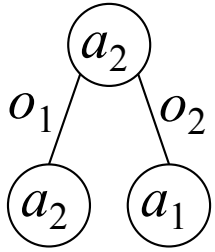


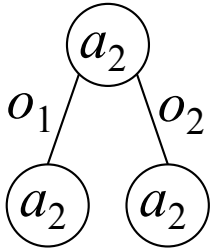


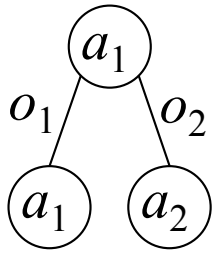
Dynamic Programming

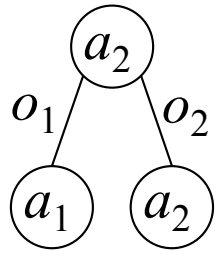


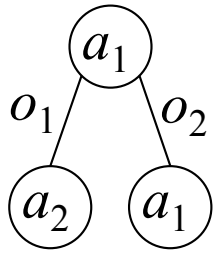


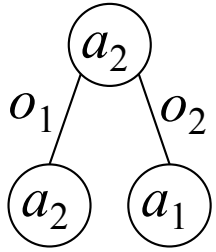


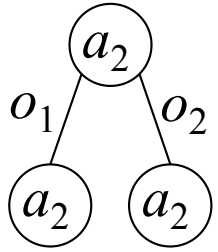




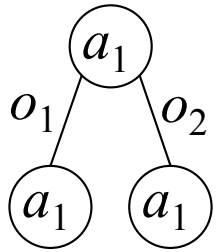


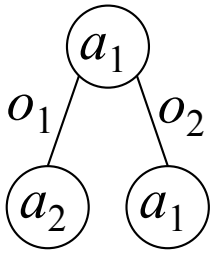


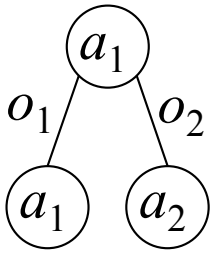


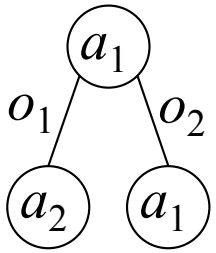


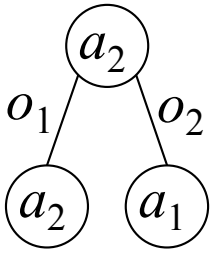
Dynamic Programming

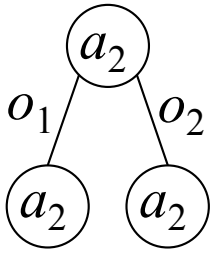


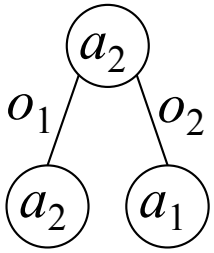


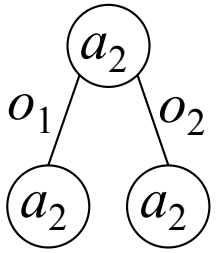




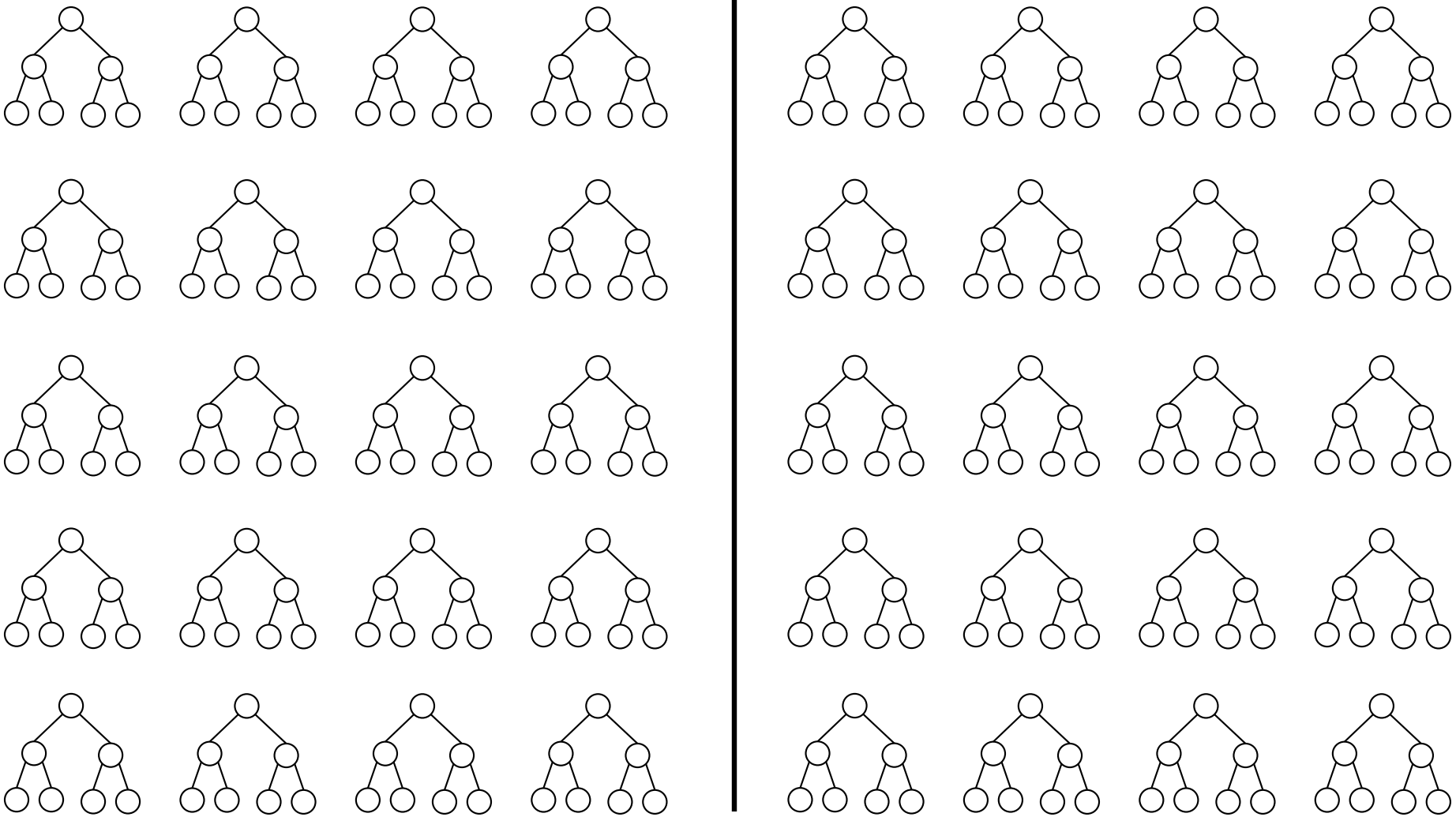






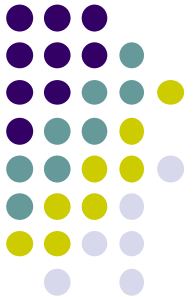


Dynamic Programming



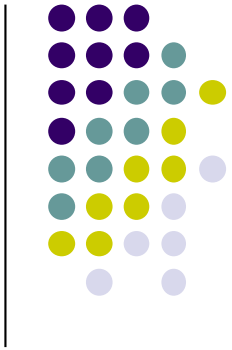
Correctness of DP

[Hansen, Bernstein & Zilberstein, AAAI 04]



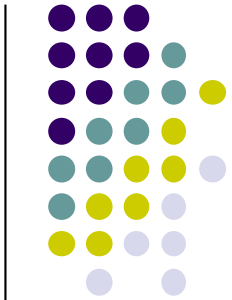
- **Theorem:** *DP performs iterated elimination of dominated strategies in the normal form of the POSG.*
- **Corollary:** *DP can be used to find an optimal joint policy in a DEC-POMDP.*
- Offers the first exact DP algorithm for DEC-POMDPs and POSGs
- Can be doubly exponential in the horizon
- In practice complexity depends on how much pruning is possible

Implementation



- Natural combination of two ideas
  - Iterated elimination of dominated strategies
  - Dynamic programming for POMDPs
- Algorithm keeps both value functions *and* policy trees in memory (unlike POMDPs)
- Currently no way to do pruning incrementally (unlike POMDPs)

Performance



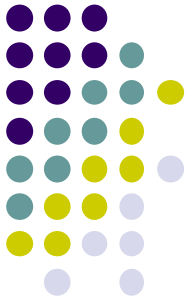
- The multi-access broadcast channel problem

  Policy trees remaining after each iteration:

Iteration	Brute force	DP algorithm
1	2	2
2	8	6
3	128	20
4	32,768	300

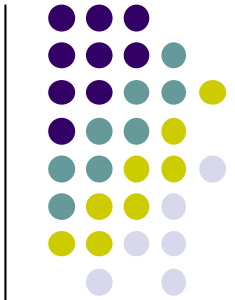
- DP prunes a significant number of trees

Infinite-Horizon DEC-POMDPs



- Unclear how to define a **belief-state** for a DEC-POMDP without fixing the policies of the other agents
- **Value iteration** does not generalize to the infinite-horizon case
- Can generalize **policy iteration** for POMDPs [Hansen 98, Poupart & Boutilier 04]
- Basic idea: Representing local policies using **stochastic finite-state controllers** and define a set of controller transformations that guarantee convergence

Policies as Controllers

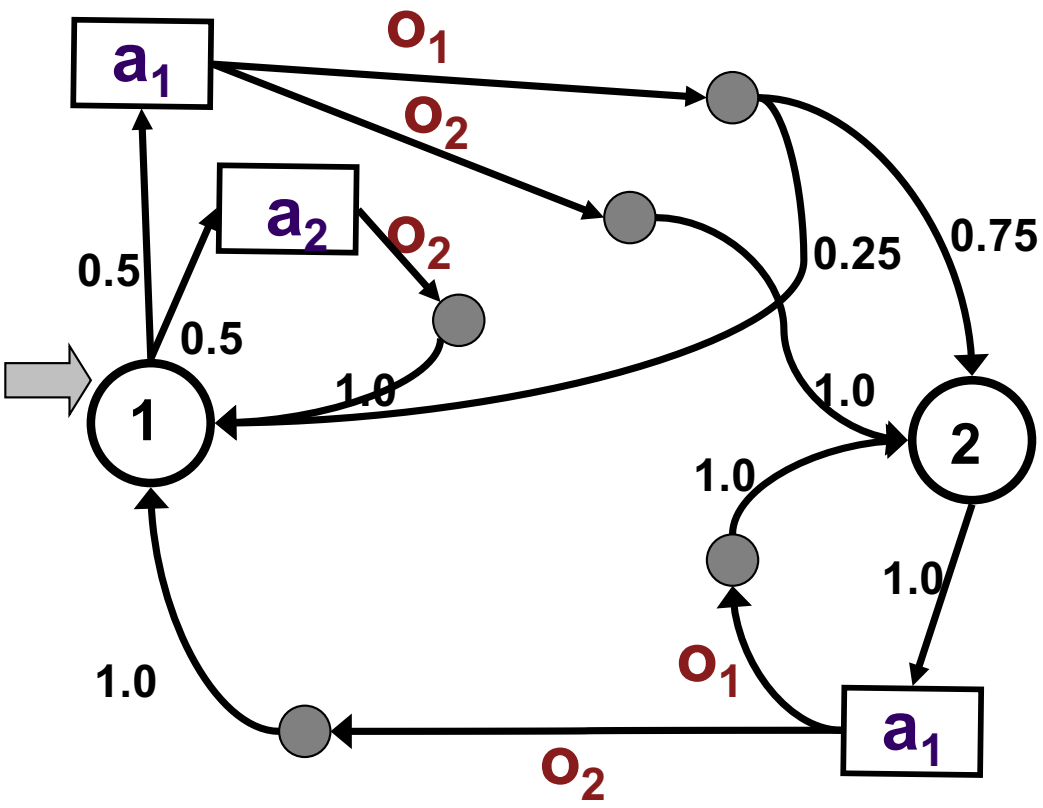


- Finite state controller represents each policy
  - Fixed memory
  - Randomness used to offset memory limitations
  - Action selection, ψ : $Q_i \rightarrow \Delta A_i$
  - Transitions, η : $Q_i \times A_i \times O_i \rightarrow \Delta Q_i$
- Value of two-agent joint controller given by the Bellman equation:

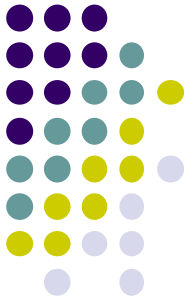
$$V(q_1,q_2,s) = \sum_{a_1,a_2} P(a_1 \mid q_1)P(a_2 \mid q_2)\Big[R(s,a_1,a_2) + \gamma \sum_{s'} P(s' \mid s,a_1,a_2) \sum_{o_1,o_2} O(o_1,o_2 \mid s',a_1,a_2) \sum_{q_1',q_2'} P(q_1' \mid q_1,a_1,o_1)P(q_2' \mid q_2,a_2,o_2)V(q_1',q_2',s')\Big]$$

Controller Example

- Stochastic controller for one agent
  - 2 nodes, 2 actions, 2 observations
  - Parameters
    - $P(a_i \mid q_i)$
    - $P(q_i' \mid q_i, o_i)$



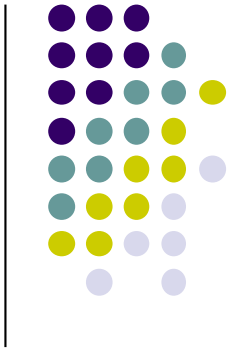
Finding Optimal Controllers



- How can we **search the space** of possible joint controllers?
- How do we set the parameters of the controllers to **maximize value**?
- Deterministic controllers - can use traditional search methods such as best-first search
- Stochastic controllers - continuous optimization problem
- **Key question**: how to best use a limited amount of storage to optimize value

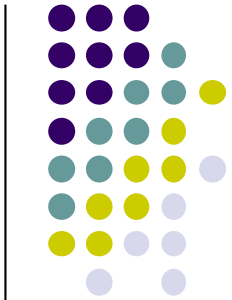
Nonlinear Optimization Approach

[Amato, Bernstein & Zilberstein, UAI 07]



- Key idea: Modeling the problem as a non-linear program (NLP)
- Consider node values (as well as controller parameters) as a variables
- The NLP can take advantage of an initial state distribution when it is given
- Improvement and evaluation all in one step (equivalent to an infinite lookahead)
- Additional constraints maintain valid values

NLP Representation



Variables:

$$x(\vec{q},\vec{a}) = P(\vec{a} \mid \vec{q}),\ y(\vec{q},\vec{a},\vec{o},\vec{q}') = P(\vec{q}' \mid \vec{q},\vec{a},\vec{o}),\ z(\vec{q},s) = V(\vec{q},s)$$

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

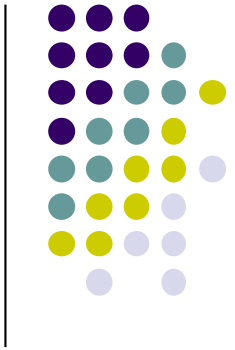
Value Constraints: $\forall s \in S,\ \vec{q} \in Q$

$$z(\vec{q},s) = \sum_{\vec{a}} x(\vec{q}',\vec{a}) \left[R(s,\vec{a}) + \gamma \sum_{s'} P(s' \mid s,\vec{a}) \sum_{\vec{o}} O(\vec{o} \mid s',\vec{a}) \sum_{\vec{q}'} y(\vec{q},\vec{a},\vec{o},\vec{q}') z(\vec{q}',s') \right]$$

Additional linear constraints:

- ensure controllers are independent
- all probabilities sum to 1 and are non-negative

Independence Constraints



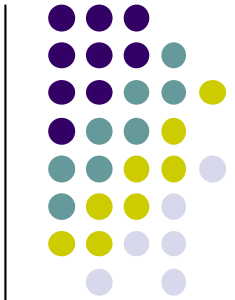
- Independence constraints guarantee that action selection and controller transition probabilities for each agent depend only on local information
- Action selection independence:

$$\forall a_i, \vec{q} \sum_{a_{-i}} x(\vec{q}, \vec{a}) = \sum_{a_{-i}} x(q_i, q^f_{-i}, \vec{a})$$

- Controller transition independence:

$$\forall \vec{a}, \vec{q}, \vec{o}, q'_i \sum_{q'_{-i}} y(\vec{q}, \vec{a}, \vec{o}, \vec{q'}) = \sum_{q'_{-i}} y(q_i, q^f_{-i}, a_i, a^f_{-i}, o_i, o^f_{-i}, \vec{q'})$$

Probability Constraints



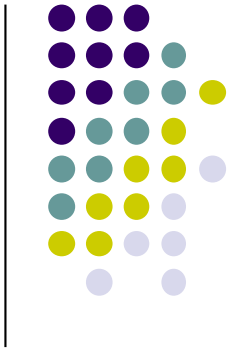
- Probability constraints guarantee that action selection probabilities and controller transition probabilities are non negative and that they add up to 1:

$$\forall q_i \sum_{\vec{a}} x(q_i, q^f_{-i}, \vec{a}) = 1 \text{ and } \forall q_i, o_i, a_i \sum_{\vec{q'}} y(q_i, q^f_{-i}, a_i, a^c_{-i}, o_i, o^f_{-i}, \vec{q'}) = 1$$

$$\forall \vec{q}, \vec{a}\ x(\vec{q}, \vec{a}) \geq 0 \text{ and } \forall \vec{q}, \vec{o}, \vec{a}\ y(\vec{q}, \vec{a}, \vec{o}, \vec{q'}) \geq 0$$

(Superscript f' s represent arbitraty fixed values)

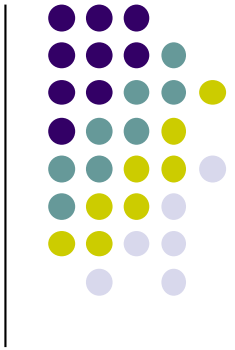
Optimality



Theorem: *An optimal solution of the NLP results in optimal stochastic controllers for the given size and initial state distribution.*

- Advantages of the NLP approach:
  - Efficient policy representation with fixed memory
  - NLP represents optimal policy for given size
  - Takes advantage of known start state
  - Easy to implement using off-the-shelf solvers
- Limitations:
  - Difficult to solve optimally

Adding a Correlation Device



- NLP approach can be extended to include a correlation device, using the following formulation:

For variables: $w(c, c'), x(\vec{q}, \vec{a}, c), y(\vec{q}, \vec{a}, \vec{o}, \vec{q'}, c)$ and $z(\vec{q}, s, c)$
Maximize

$$\sum_s b_0(s) z(\vec{q^0}, s)$$

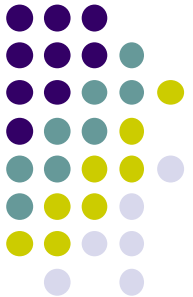
Given the Bellman constraints:

$$\forall \vec{q}, s \; z(\vec{q}, s, c) = \sum_{\vec{a}} x(\vec{q}, \vec{a}, c) \left[R(s, \vec{a}) + \gamma \sum_{s'} P(s'|s, \vec{a}) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q'}} y(\vec{q}, \vec{a}, \vec{o}, \vec{q'}, c) \sum_{c'} w(c, c') z(\vec{q'}, s', c) \right]$$

- New variable $w(c,c')$ represents the transition function of the correlation device; action selection and controller transitions depend on new shared signal.

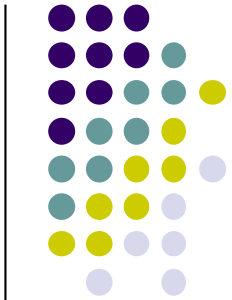
Experiments

[Amato, Bernstein & Zilberstein, UAI 07]



- Used freely available nonlinear constrained optimization solver called "**filter**" on the NEOS server (http://www-neos.mcs.anl.gov/neos/)
- Solver guarantees locally optimal solution
- Used 10 random initial controllers for a range of controller sizes
- Compared NLP with DEC-BPI, with and without a small (2-node) correlation device

Results: Broadcast Channel



- Simple two agents networking problem

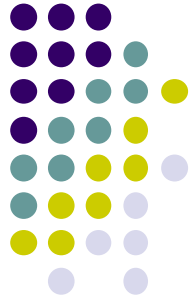
  (2 agents, 4 states, 2 actions, 5 observations)
- Average quality over 10 trials:

size	DEC-BPI	DEC-BPI corr	NLO	NLO-corr
1	4.687	6.290	9.1	9.1
2	4.068	7.749	9.1	9.1
3	8.637	7.781	9.1	9.1
4	7.857	8.165	9.1	9.1

- Average run time:

size	DEC-BPI	DEC-BPI corr	NLO	NLO-corr
1	< 1s	< 1s	1s	2s
2	< 1s	2s	3s	8s
3	2s	7s	764s	2119s
4	5s	24s	4061s	10149s

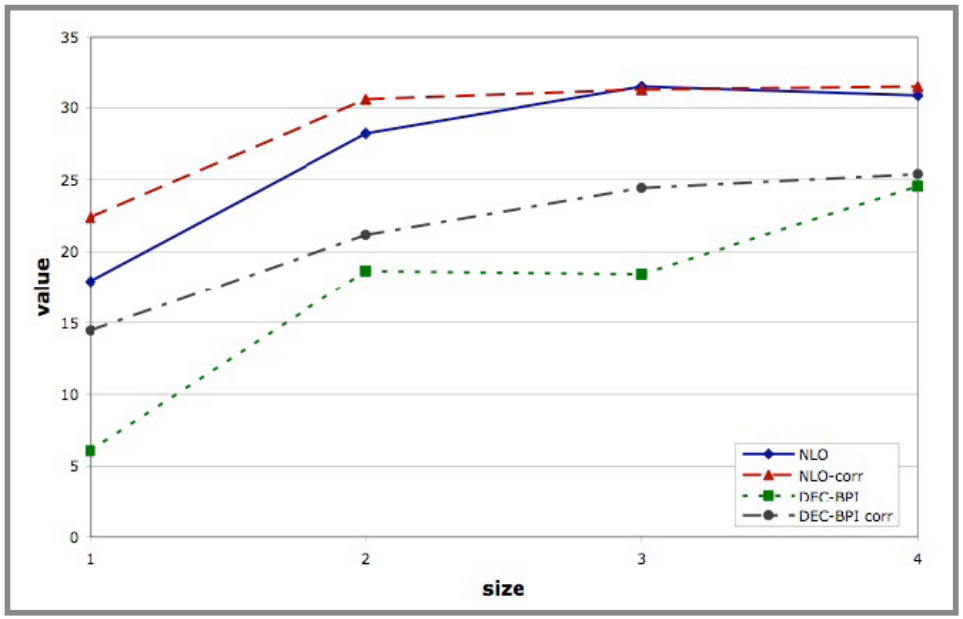
Results: Recycling Robots



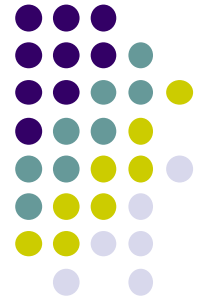
- A two-agent extension of the problem domain introduced in [Sutton & Barto 98]

  (2 agents, 4 states, 3 actions, 2 observation)

- Average quality over 10 trials:



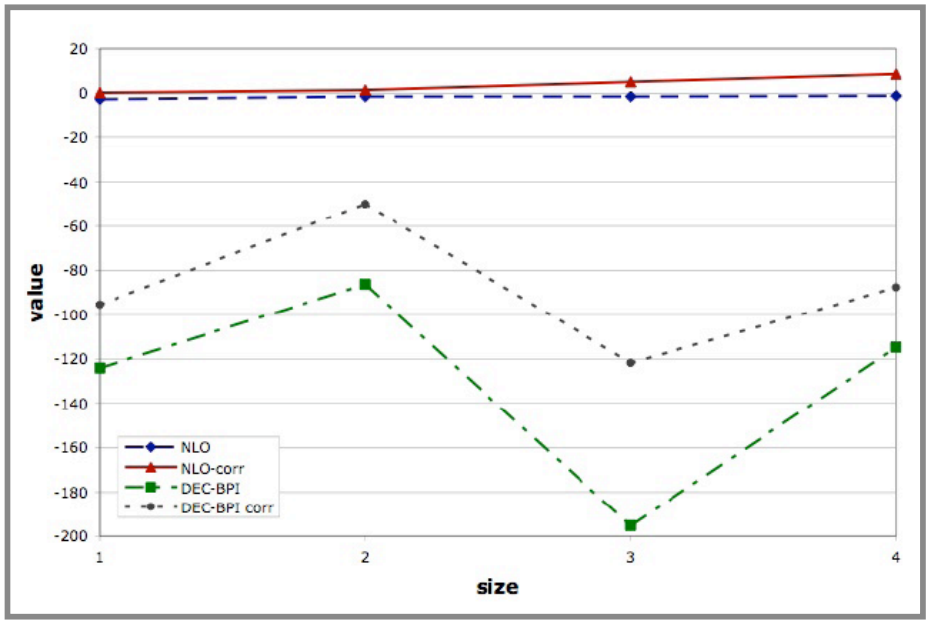
Results: Multi-Agent Tiger Problem



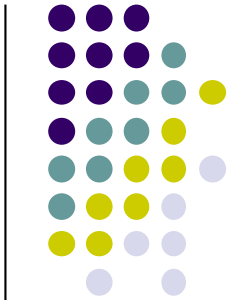
- A two-agent version by [Nair et al. 03] of the "well-known" POMDP benchmark problem

  (2 agents, 2 states, 3 actions, 2 observations)

- Average quality over 10 trials:

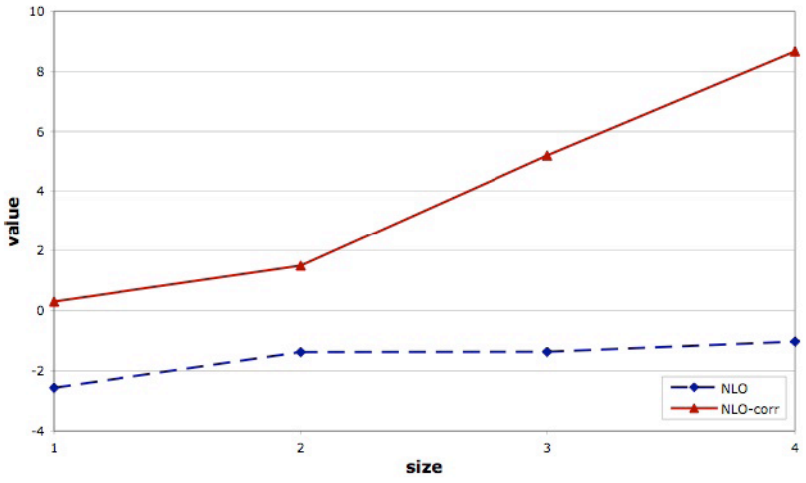


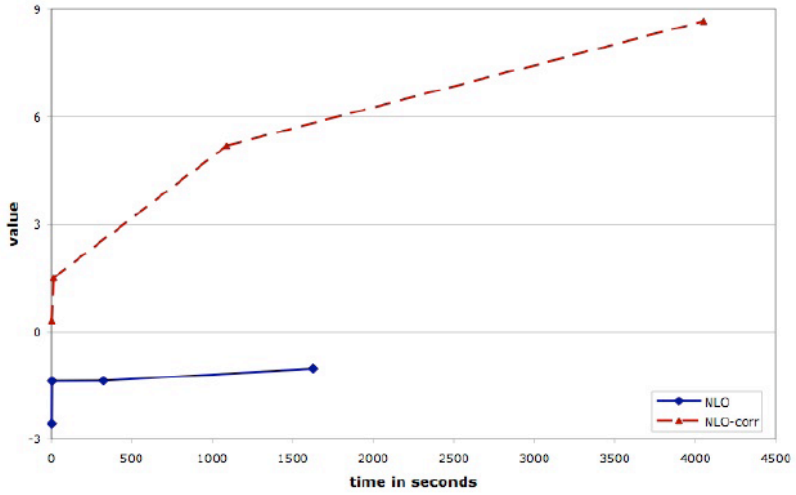
The Value of Stochastic Correlation



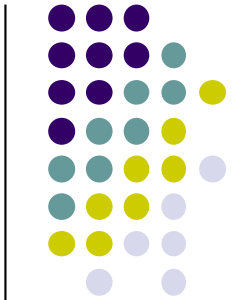
A more detailed comparison of the NLP approach with and without a correlation device shows that:

- Correlation helps to produce **significantly higher values**
- Even though correlation increases runtime, it helps produce **better values within a given amount of time**





Resources for Solving MDPs



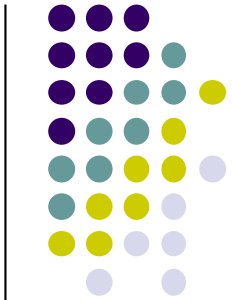
mdp-lib: Library for Solving MDPs

https://github.com/luisenp/mdp-lib

Luis Pineda ⟨lpineda@cs.umass.edu⟩

- **Problem:** Provides abstractions for representing transition function, cost function, goal check; efficient containers for states
- **State:** Provides abstractions for uniquely identifying states
- **Action:** Provides abstractions for uniquely identifying actions
- **Heuristic**: Provides abstractions for representing a heuristic function

mdp-lib: Domains and Algorithms



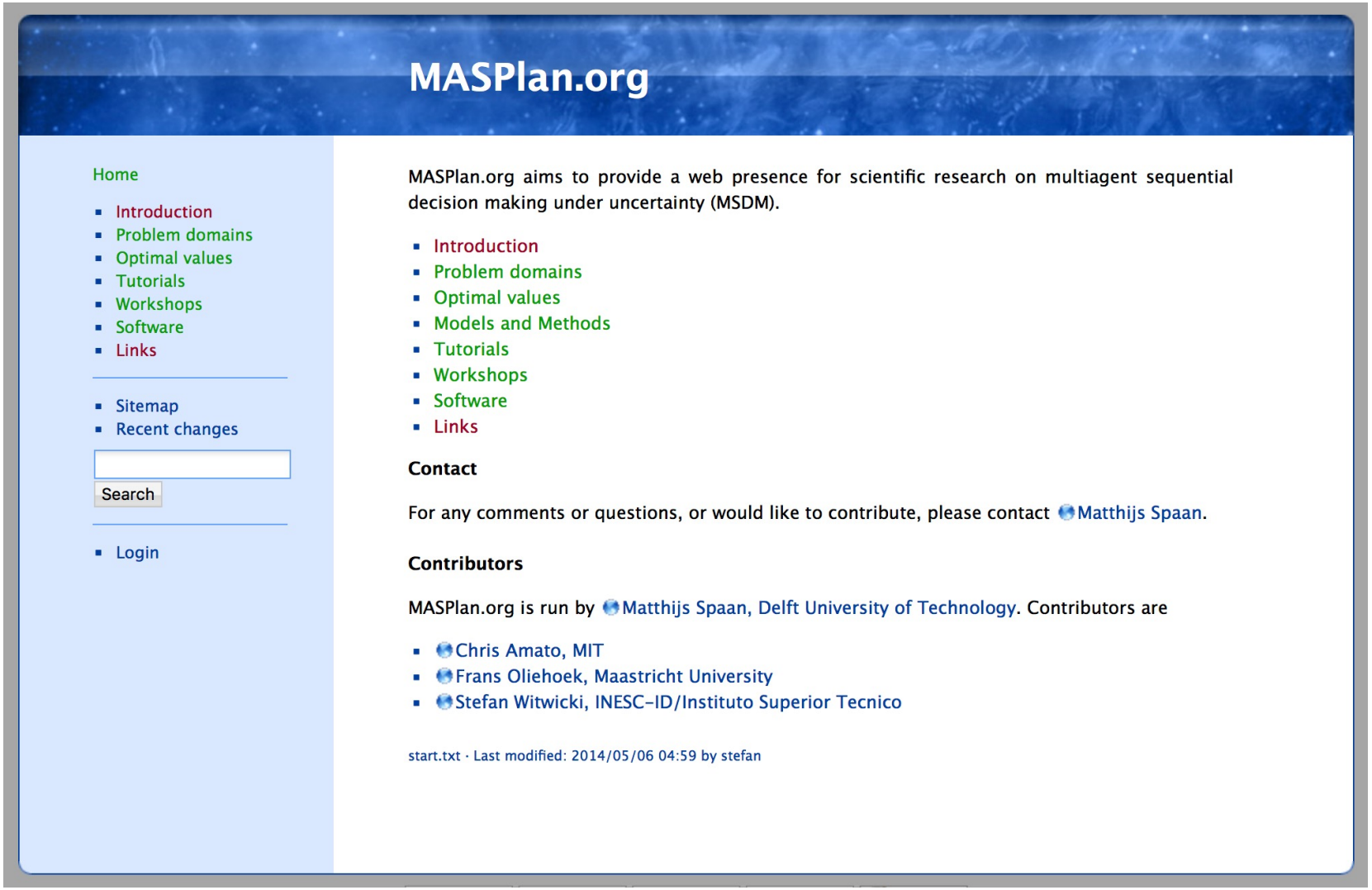
Implemented Domains

- Gridworld
- Racetrack
- Sailing
- Canadian Traveler
- Support for PPDDL through the mini-gpt library

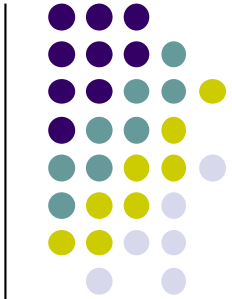
Algorithms

- LAO*
- LRTDP
- HDP
- FLARES
- FF-Replan
- RFF
- SSiPP
- $\mathcal{M}_l^k$-reductions

Resource for Multiagent Planning



Questions?



Additional Information:

Resource-Bounded Reasoning Lab

University of Massachusetts, Amherst

`http://rbr.cs.umass.edu`