UMassAmherst

Increasing Scalability in Algorithms for Centralized and Decentralized POMDPs

Christopher Amato

Carnegie Mellon University

Feb 5th, 2010

Introduction

- Sequential decision-making
- Reasoning under uncertainty
- Decision-theoretic approach
- Single and cooperative multiagent

Outline

- Introduction
- Background
  - Partially observable Markov decision processes (POMDPs)
  - Decentralized POMDPs
- My contributions to solving these models
  - Optimal dynamic programming for DEC-POMDPs
  - Increasing scalability for POMDPs and DEC-POMDPs
- Future work
  - Algorithms and applications

Dealing with uncertainty

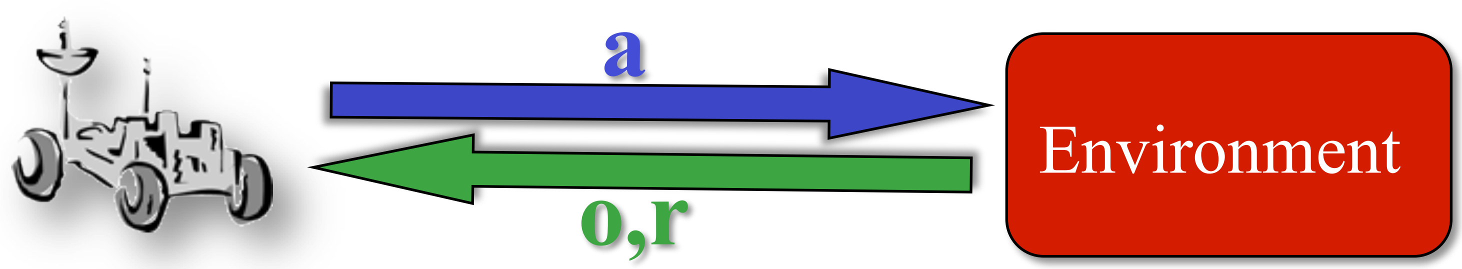
- Agent situated in a world, receiving information and choosing actions
- What happens when we don't know the exact state of the world?
- Uncertain or imperfect information
- This occurs due to
    - Noisy sensors (some states look the same or can be incorrect)
    - Unobservable states (may only receive an indirect signal)

Example single agent problems

- Robot navigation (autonomous vehicles)
- Inventory management (e.g. decide what to order based on uncertain supply and demand)
- Green computing (e.g. moving jobs or powering off systems given uncertain usage)
- Medical informatics (e.g. diagnosis and treatment or hospital efficiency)

Single agent: partially observable

- Partially observable Markov decision process (POMDP)
- Extension of fully observable MDP
- Agent interacts with partially observable environment
  - Sequential decision-making under uncertainty
  - At each stage, the agent takes a stochastic action and receives:
    - An observation based on the state of the system
    - An immediate reward



POMDP definition

- A POMDP can be defined with the following tuple: M = <*S*, *A*, *P*, *R*, *Ω*, *O*>
  - *S*, a finite set of states with designated initial state distribution b_0
  - *A*, a finite set of actions
  - *P*, the state transition model: $P(s'|\ s, a)$
  - *R*, the reward model: $R(s, a)$
  - *Ω*, a finite set of observations
  - *O*, the observation model: $O(o|\ s', a)$

In blue, are the differences from fully observable MDPs

POMDP solutions

- A policy is a mapping $\Omega^* \rightarrow A$
  - Map whole observation histories to actions because the state is unknown
  - Can also map from distributions of states (belief states) to actions for a stationary policy
- Goal is to maximize expected cumulative reward over a finite or infinite horizon
  - Note: in infinite-horizon, cannot remember the full observation history (it's infinite!)
- Use a discount factor, γ, to maintain a finite sum over the infinite horizon

Example POMDP: Hallway

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

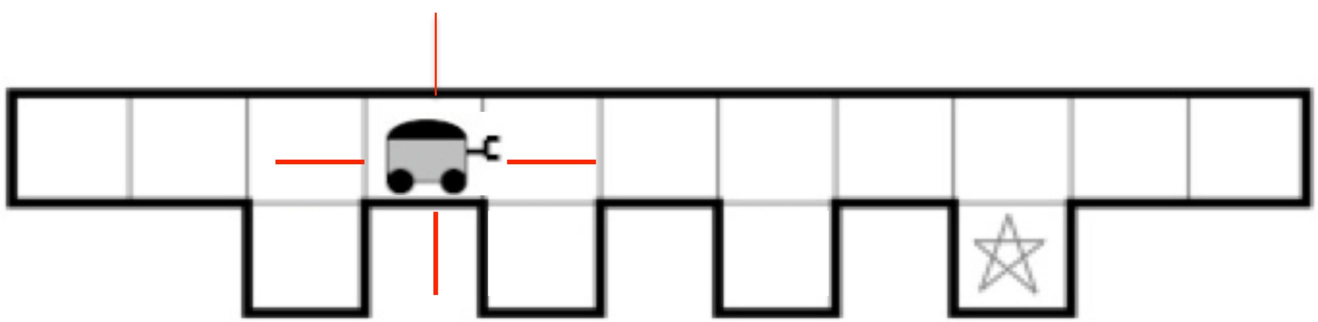
States: grid cells with orientation

Actions: turn ↰, ↱, ↷, move forward, stay

Transitions: noisy

Observations: red lines

Rewards: negative for all states except starred square

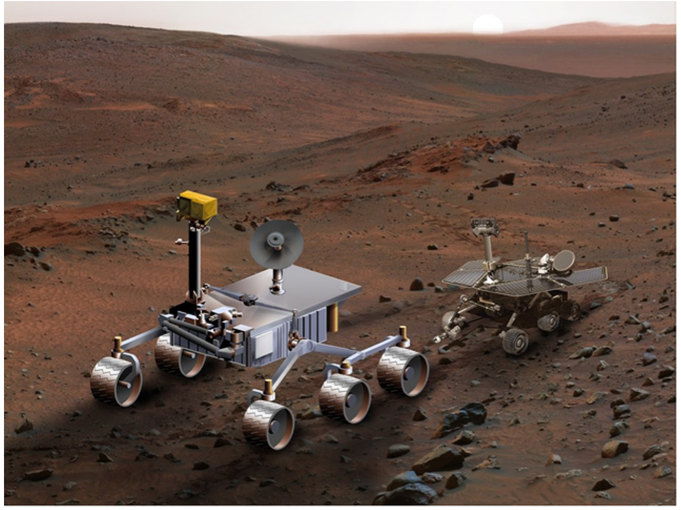


Decentralized domains

- Cooperative multiagent problems
- Each agent’s choice affects all others, but must be made using only local information
- Properties
  - Often a decentralized solution is required
  - Natural way to represent problems with multiple decision makers making choices independently of the others
  - Does not require communication on each step (may be impossible or too costly)
  - But now agents must also reason about the previous and future choices of the others (more difficult)

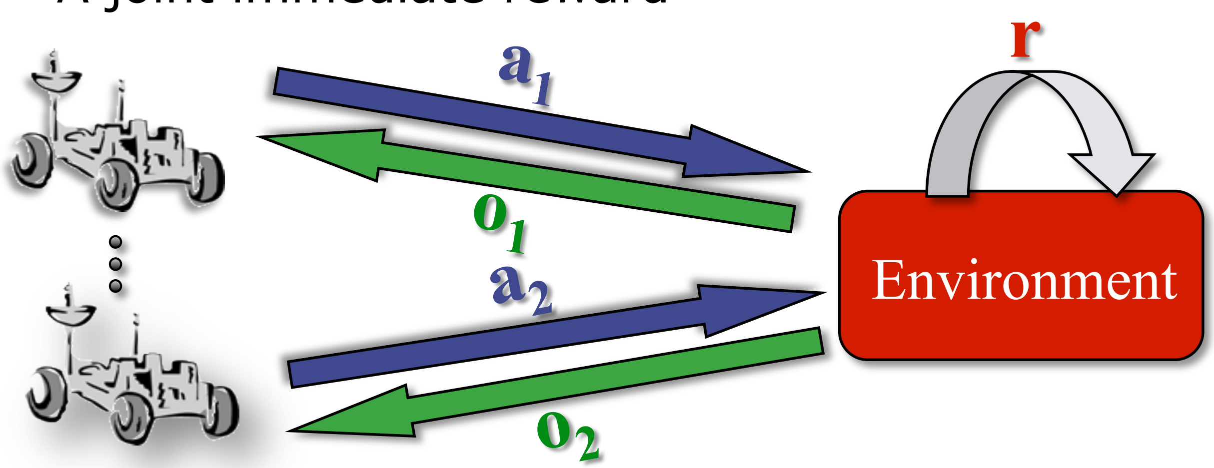
Example cooperative multiagent problems

- Multi-robot navigation
- Green computing (decentralized, powering off affects others)
- Sensor networks (e.g. target tracking from multiple viewpoints)
- E-commerce (e.g. decentralized web agents, stock markets)



Multiple cooperating agents

- Decentralized partially observable Markov decision process (DEC-POMDP)
- Multiagent sequential decision-making under uncertainty
  - At each stage, each agent takes an action and receives:
    - A local observation
    - A joint immediate reward



DEC-POMDP definition

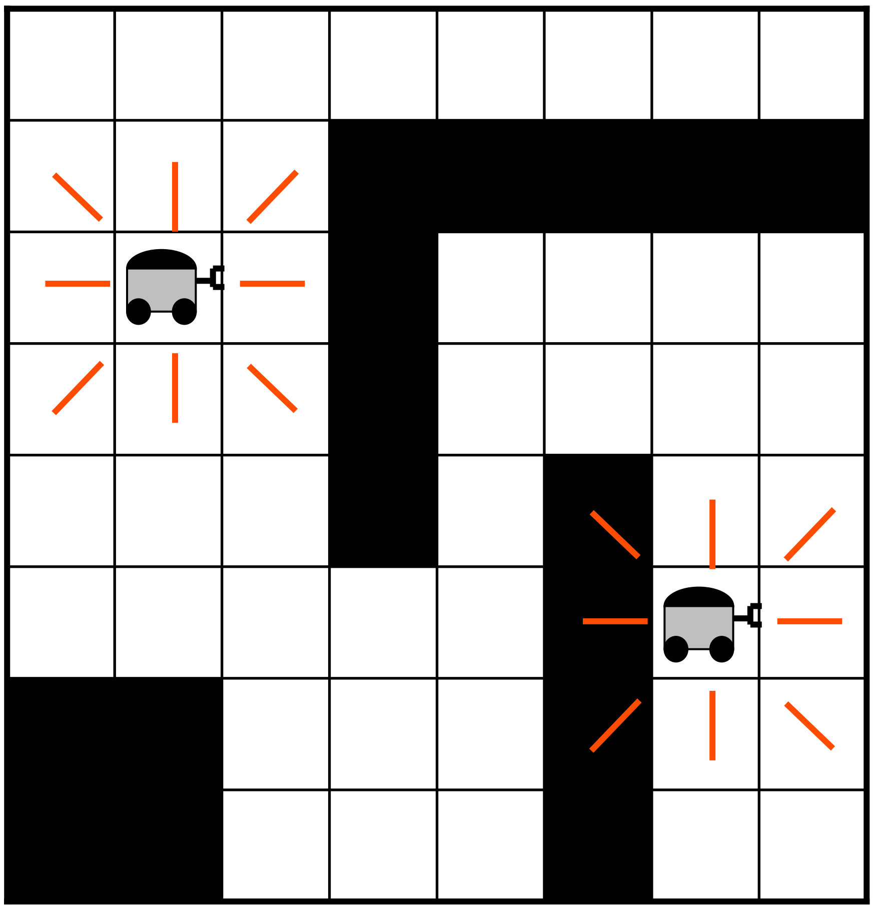
- A DEC-POMDP can be defined with the tuple: M = $<I, S, \{A_i\}, P, R, \{\Omega_i\}, O>$
  - I, a finite set of agents
  - S, a finite set of states with designated initial state distribution b_0
  - A_i, each agent's finite set of actions
  - P, the state transition model: $P(s' | s, \bar{a})$
  - R, the reward model: $R(s, \bar{a})$
  - Ω_i, each agent's finite set of observations
  - O, the observation model: $O(\bar{o} | s', \bar{a})$

Similar to POMDPs, but now functions depend on all agents

DEC-POMDP solutions

- A local policy for each agent is a mapping from its observation sequences to actions, $\Omega^* \rightarrow A$
  - Note that an agents do not generally have enough information to calculate an estimate of the state
  - Also, planning can be centralized but execution is distributed
- A joint policy is a local policy for each agent
- Goal is to maximize expected cumulative reward over a finite or infinite horizon
  - Again, for infinite-horizon cannot remember the full observation history
- In infinite case, a discount factor, γ, is used

Example: 2-Agent Grid World



States: grid cell pairs

Actions: move ⇧, ⇩, ⇨, ⇦, stay

Transitions: noisy

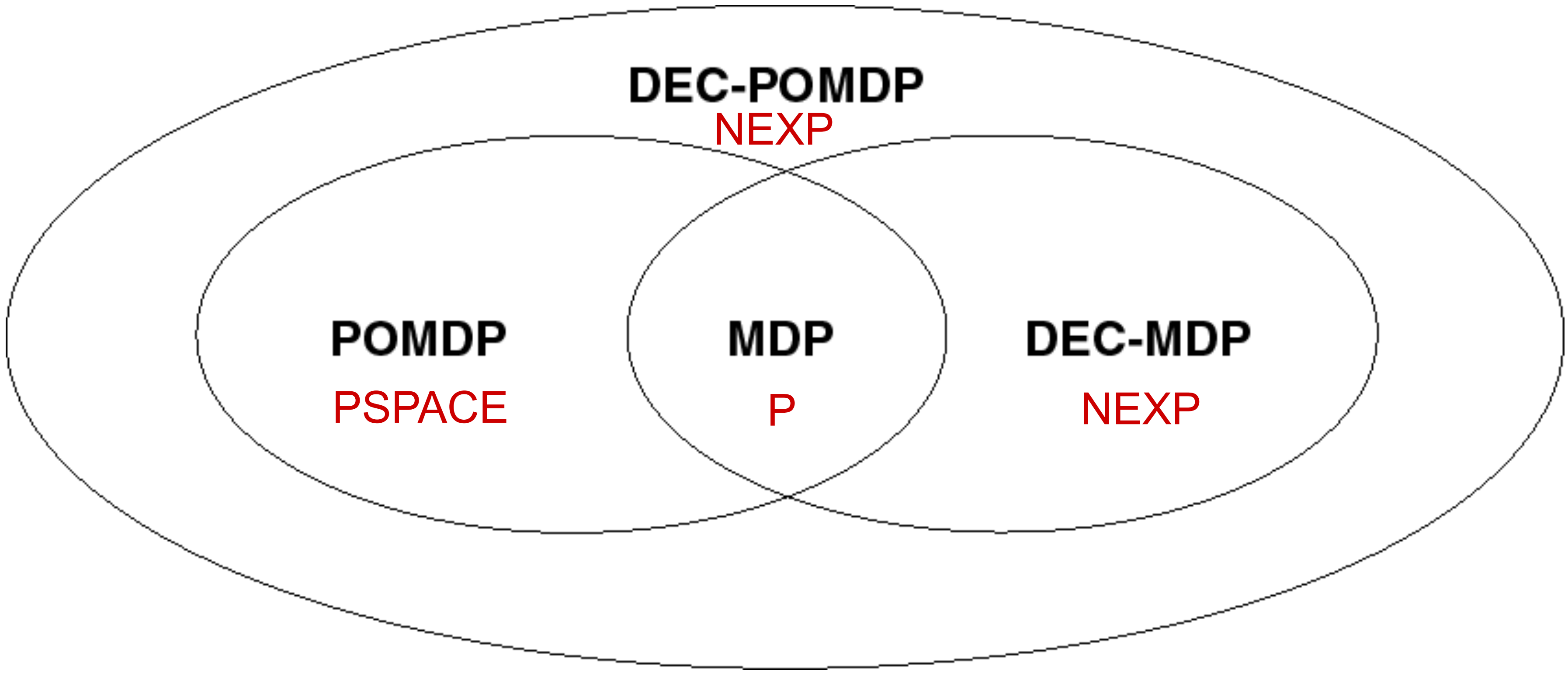
Observations: red lines

Rewards: negative unless sharing the same square

Challenges in solving DEC-POMDPs

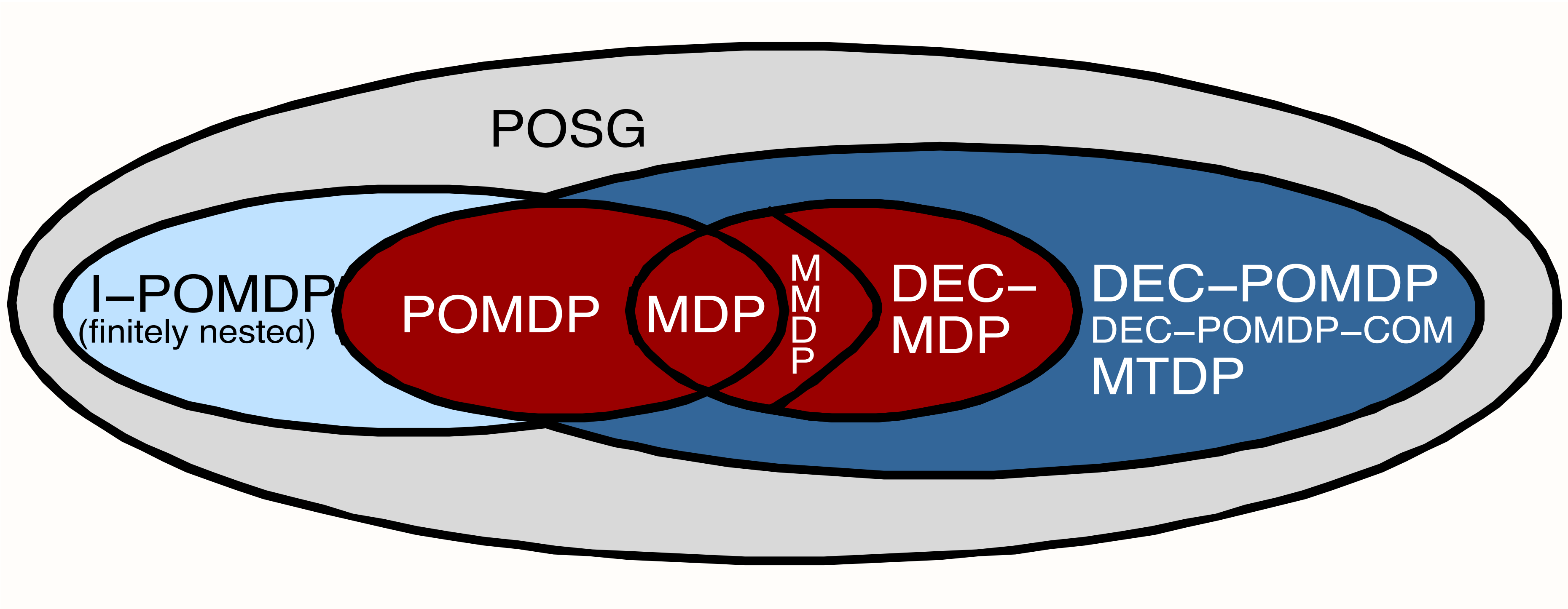
- Like POMDPs, partial observability makes the problem difficult to solve
- Unlike POMDPs: No centralized belief state
  - Each agent depends on the others
  - This requires a belief over the possible policies of the other agents
  - Can’t transform DEC-POMDPs into a continuous state MDP (how POMDPs are typically solved)
- Therefore, DEC-POMDPs cannot be solved by POMDP algorithms

General complexity results



subclasses and finite horizon complexity results

Relationship with other models



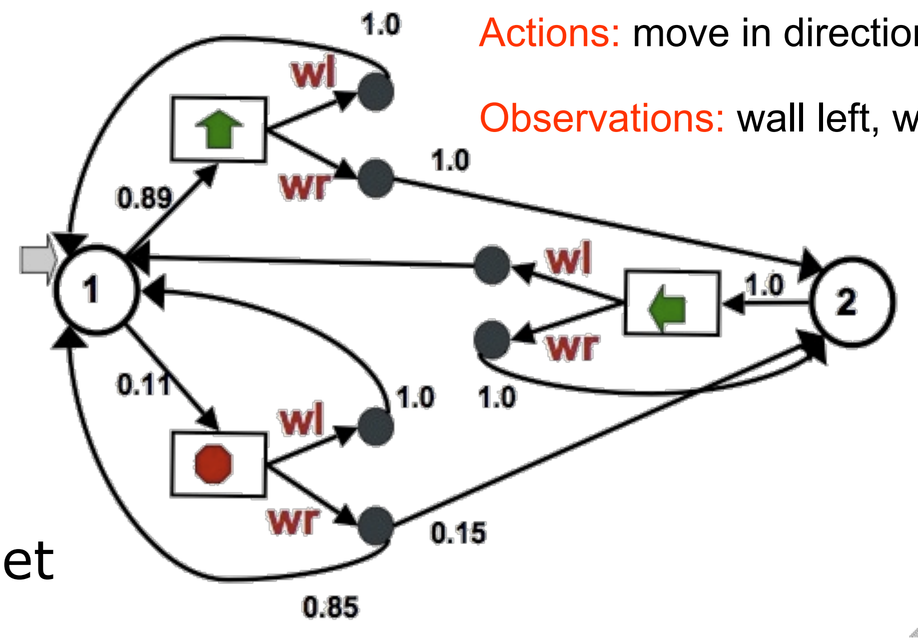
Ovals represent complexity, while colors represent number of agents and cooperative or competitive models

Overview of contributions

- Optimal dynamic programming for DEC-POMDPs
  - ε-optimal solution using finite-state controllers for infinite-horizon
  - Improving dynamic programming for DEC-POMDPs with reachability analysis
- Scaling up in single and multiagent environments by methods such as:
  - Memory bounded solutions
  - Sampling
  - Taking advantage of domain structure

Infinite-horizon polices as stochastic controllers

- Designated initial node
- Nodes define actions
- Transitions based on observations seen
- Inherently infinite-horizon
- Periodic policies
- With fixed memory, randomness can offset memory limitations



For DEC-POMDPs use one controller for each agent

Evaluating controllers

- Stochastic controller defined by parameters
  - Action selection: $Q \rightarrow \Delta A$
  - Transitions: $Q \times O \rightarrow \Delta Q$
- For a node, q, and the above parameters, value at state s is given by Bellman equation (POMDP):

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

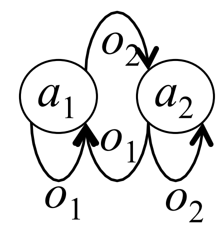
Optimal dynamic programming for DEC-POMDPs

JAIR 09

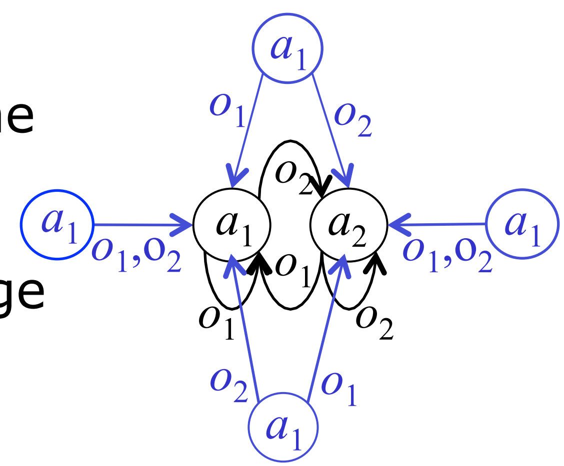
- Infinite-horizon dynamic programming (DP): Policy Iteration
  - Build up finite-state controllers as policies for each agent (called "backups") over a number of steps
  - At each step, remove or *prune* controller nodes that have lower value using linear programming
  - Redirect and *merge* remaining nodes to produce a *stochastic controller*
  - Continue backups and pruning until provably within ε of optimality (can be done in finite steps)
- First ε-optimal algorithm for infinite-horizon

Optimal DP for DEC-POMDPs: Policy Iteration

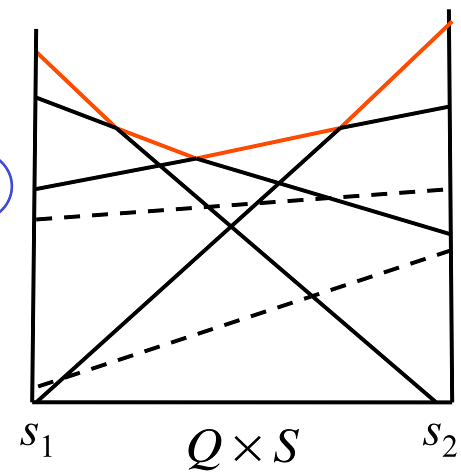
- Start with a given controller
- Exhaustive backup: generate all next step policies by considering any first action and then choosing some node of the controller for each observation
- Evaluate: determine value of starting at each node at each state and for each policy for the other agents
- Prune: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



= Initial controller for agent 1

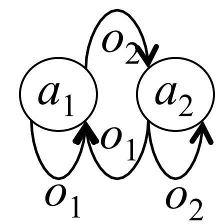


(backup for action 1)

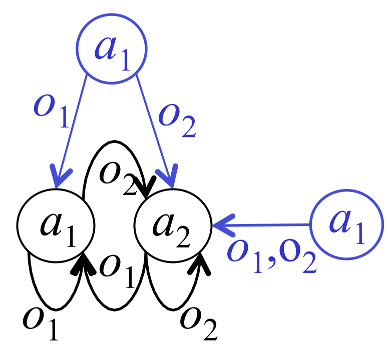


Optimal DP for DEC-POMDPs: Policy Iteration

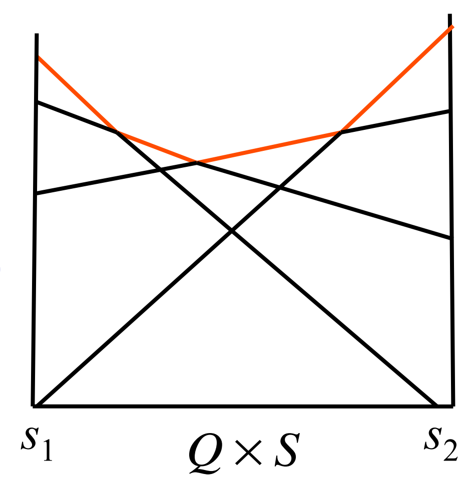
- Start with a given controller
- Exhaustive backup: generate all next step policies by considering any first action and then choosing some node of the controller for each observation
- Evaluate: determine value of starting at each node at each state and for each policy for the other agents
- Prune: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



= Initial controller for agent 1



(backup for action 1)



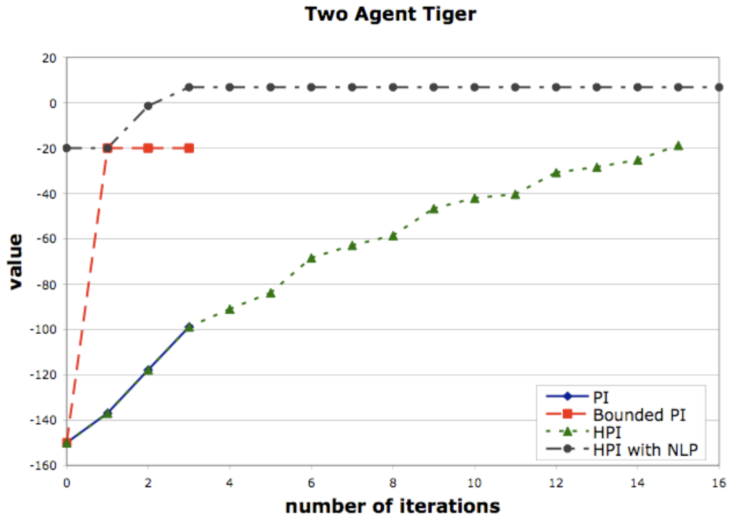
Improvements and experiments JAIR 09

- Can improve value of controller after each pruning step
- Can use heuristics and sampling of the state space (point-based method) to produce approximate results

Meeting on a Grid, $\lvert S\rvert = 16$, $\lvert A_i\rvert = 5$, $\lvert \Omega_i\rvert = 4$			
Iteration	Exhaustive Sizes	Controller Reductions	Bounded Updates
0	$(1,1)$	2.8 (1,1 in 1s)	2.8 (1,1 in 1s)
1	$(5,5)$	3.4 (5,5 in 7s)	3.8 (5,5 in 145s)
2	$(3125,3125)$	3.7 (80,80 in 821s)	4.78* (125,125 in 1204s)

Box Pushing, $\lvert S\rvert = 100$, $\lvert A_i\rvert = 4$, $\lvert \Omega_i\rvert = 5$			
Iteration	Exhaustive Sizes	Controller Reductions	Bounded Updates
0	$(1,1)$	-2 (1,1 in 4s)	-2 (1,1 in 53s)
1	$(4,4)$	-2 (2,2 in 108s)	6.3 (2,2 in 132s)
2	$(4096,4096)$	12.8 (9,9 in 755s)	42.7* (16,17 in 714s)

Optimal methods: value, controller size and time



Optimal and approximate methods

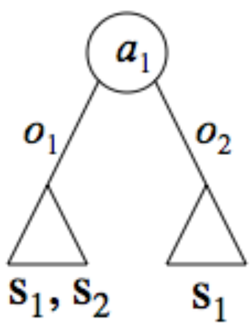
- Optimal DP can prune a large number of nodes
- Approximate approaches can improve scalability

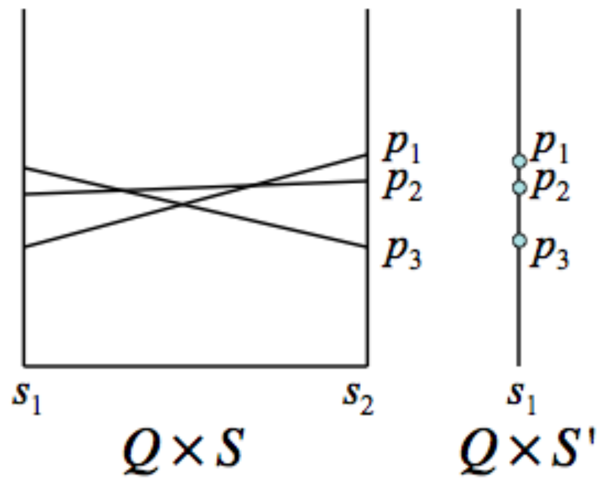
Incremental policy generation ICAPS 09

- Optimal dynamic programming for DEC-POMDPs requires a large amount of time and space
- In POMDPs, methods have been developed to make optimal DP more efficient
- These cannot be extended to DEC-POMDPs (due to the lack of a shared viewpoint by the agents)
- We developed a new DP method to make the optimal approaches for both finite and infinite-horizon more efficient

Incremental policy generation (cont.)

- Can avoid exhaustively generating policies (backups)
- Cannot know what policies the others may take, but after an action is taken and observation seen, can limit the number of states considered (see a wall, other agent, etc.)
- This allows policies for an agent to be built up incrementally
- That is, iterate through possible first actions and observations, adding only subtrees (or subcontrollers) that are not dominated



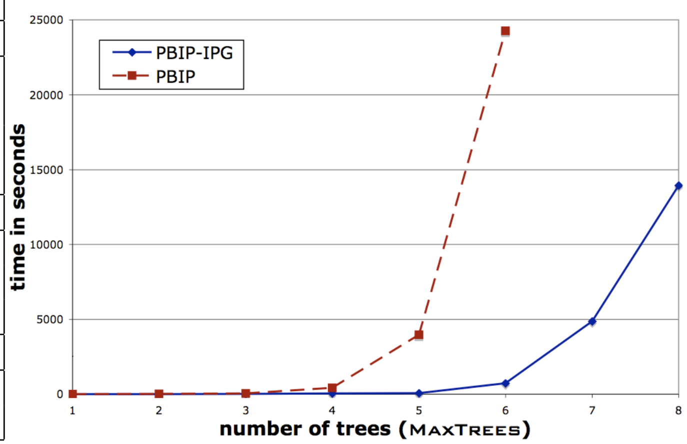


Benefits of IPG and results ICAPS 09

- Solve larger problems optimally
- Can make use of start state information as well
- Can be used in other dynamic programming algorithms
  - Optimal: Finite-, infinite- and indefinite horizon as well as policy compression
  - Approximate: PBDP, MBDP, IMBDP, MBDP-OC and PBIP

Horizon	DP	Incremental Generation (IPG)	IPG with Start State	Value
Meeting in a 3x3 Grid, $\|S\| = 81$, $\|A_i\| = 5$, $\|\Omega_i\| = 9$				
2	(5) 5 in 5s	5 in <1s	5 in 5s	0.000
3	x	5 in 16s	5 in 17s	0.133
4	x	40 in 42s	10 in 53s	0.433
5	x	(25960)* in 2555s	(148) 148,145 in 600s	0.896
Box Pushing, $\|S\| = 100$, $\|A_i\| = 4$, $\|\Omega_i\| = 5$				
2	(128) 8 in 14s	8 in 2s	(4) 2,3 in 1s	17.60
3	x	(320,256) 256 in 1159s	(6) 5,6 in 6s	66.08
4	x	x	(233,239) 233 in 1138s	98.59
Stochastic Mars Rover, $\|S\| = 256$, $\|A_i\| = 6$, $\|\Omega_i\| = 8$				
2	x	(150, 672)* in 72s	(16,20) 12,15 in 83s	5.80
3	x	x	(396, 534)* in 389s	9.38

Increases scalability in optimal DP (finite or infinite-horizon)
x signifies inability to solve problem with 2GB memory



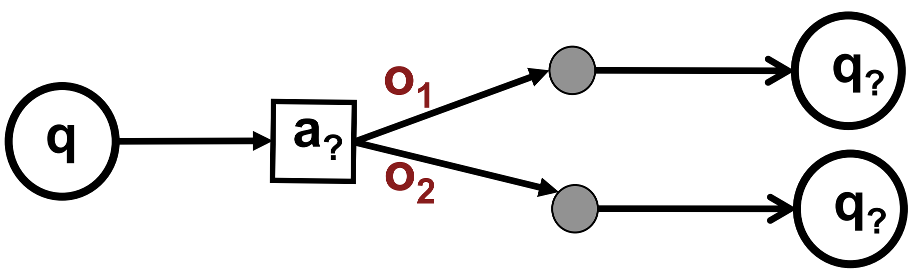
… and approximate DP

Approximate methods

- Optimal approaches may be intractable, causing approximate methods to be desirable
- Questions
  - How can high-quality memory-bounded solutions be generated for POMDPs and DEC-POMDPs?
  - How can sampling be used in the context of DEC-POMDPs to produce solutions efficiently?
  - Can I use goals and other domain structure to improve scalability?

Memory-bounded solutions

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



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

Nonlinear Programming approach IJCAI 07, UAI 07, JAAMAS 09

- Use a nonlinear program (NLP) to represent an optimal fixed-size controller for POMDPs or set of controllers for DEC-POMDPs
- Consider node value as well as action and transition parameters as variables
- Thus, find action selection and node transition parameters that maximize the value using a known start state
- Constraints maintain valid values and probabilities

NLP formulation (POMDP case)

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

Objective: Maximize $\sum_s b_0(s) y(q_0,s)$

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

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

Probability constraints: $\forall q \in Q,\ a \in A,\ o \in \Omega$

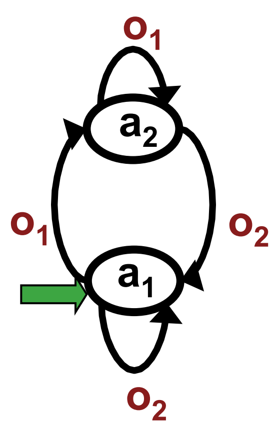
$$\sum_{q'} x(q',a,q,o) = \sum_{q'} x(q',a,q,o_k)$$

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

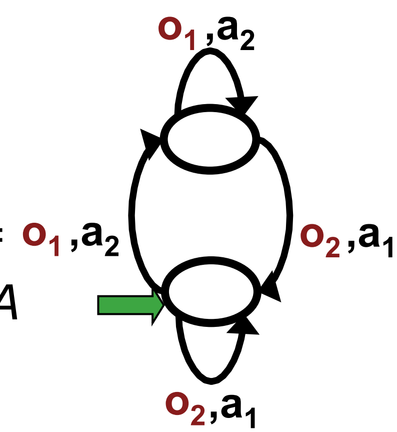
Mealy controllers recent submission

- Controllers currently used are Moore controllers
- Mealy controllers are more powerful than Moore controllers (can represent higher quality solutions with the same number of nodes)
- Provides extra structure that algorithms can use
- Can be used in place of Moore controllers in all controller-based algorithms for POMDPs and DEC-POMDPs

Moore= $Q \rightarrow A$



Mealy= $Q \times O \rightarrow A$



NLP results: POMDP case JAAMAS 09 and unpublished

Algorithm	Value	Size	Time
Aloha: $\|S\| = 90, \|A\| = 29, \|O\| = 3$			
Mealy	1,221.72	7	312
HSVI2	1,212.15	2,909	1,851
Moore	1,211.67	6	1,134
PERSEUS	853.41	31	1,801
Tag: $\|S\| = 870, \|A\| = 5, \|O\| = 30$			
PBPI[1]	-5.87	818	1,133
RTDP-BEL[1]	-6.16	2.5m	493
PERSEUS[1]	-6.17	280	1,670
HSVI2[1]	-6.36	415	24
Mealy	-6.65	2	323
Moore fixed	-8.14	7	5,669
Moore	-13.94	2	5,596
Tag Repeat: $\|S\| = 870, \|A\| = 5, \|O\| = 30$			
Mealy	-11.44	2	319
PERSEUS	-12.24	142	2,020
HSVI2	-15.02	3,207	1,815
Moore	-20.00	1	37
Hallway2 $\|S\| = 93, \|A\| = 5, \|\Omega\| = 17$			
Moore fixed	1.97	13	309
Moore	1.66	6	163
HSVI2	1.18	2,540	3,627

- Optimizing a Moore controller can provide a high-quality solution
- Optimizing a Mealy controller improves solution quality without increasing controller size
- Both approaches perform better in truly infinite-horizon problems (those that never terminate)
- DEC-POMDP results are similar, but discussed later
- Future specialized solvers may further increase quality

Achieving goals in DEC-POMDPs AAMAS 09

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

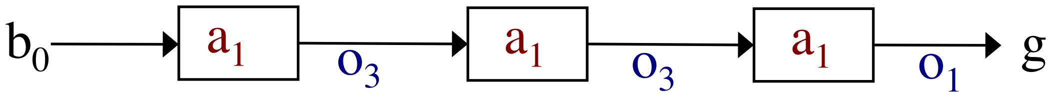
Indefinite-horizon DEC-POMDPs

- Described for POMDPs Patek 01 and Hansen 07
- Our assumptions
  - Each agent possesses a set of terminal actions
  - Negative rewards for non-terminal actions
- Problem stops when a terminal action is taken by each agent
- Can capture uncertainty about reaching goal
- Many problems can be modeled this way

- We showed how to find an optimal solution to this problem using dynamic programming

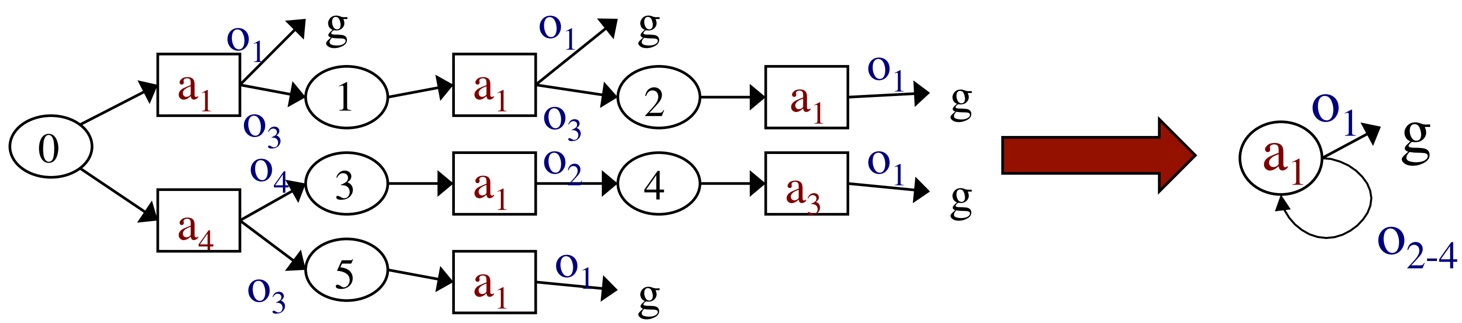
Goal-directed DEC-POMDPs

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



Getting more from fewer samples

- Optimize a finite-state controller
  - Use trajectories to create a controller
  - Ensures a valid DEC-POMDP policy
  - Allows solution to be more compact
  - Choose actions and adjust resulting transitions (permitting possibilities that were not sampled)
  - Optimize in the context of the other agents
- Trajectories create an initial controller which is then optimized to produce a high-valued policy



Experimental results AAMAS 09 and unpublished

- We built controllers from a small number of the highest-valued trajectories
- Our sample-based approach (*goal-directed*) provides a very high-quality solution very quickly in each problem
- Heuristic policy iteration and optimizing a Mealy controller also perform very well

Algorithm	Value	Size	Time
Two Agent Tiger: $\|S\| = 2, \|A_i\| = 3, \|O_i\| = 2$			
HPI w/ NLP	6.80	6	119
Goal-directed	5.04	12	75
Moore	-1.09	19	6,173
Meeting in a Grid: $\|S\| = 16, \|A_i\| = 5, \|O_i\| = 2$			
Mealy	6.13	5	116
HPI w/ NLP	6.04	7	16,763
Moore	5.66	5	117
Goal-directed	5.64	4	4
Box Pushing: $\|S\| = 100, \|A_i\| = 4, \|O_i\| = 5$			
Goal-directed	149.85	5	199
Mealy	143.14	4	774
HPI w/ NLP	95.63	10	6,545
Moore	50.64	4	5,176
Mars Rover: $\|S\| = 256, \|A_i\| = 6, \|O_i\| = 8$			
Goal-directed	21.48	6	956
Mealy	19.67	3	396
HPI w/ NLP	9.29	4	111
Moore	8.16	2	43

Conclusion

- Optimal dynamic programming for DEC-POMDPs
  - Policy iteration: ε-optimal solution with finite-state controllers (infinite-horizon)
  - Incremental policy generation: a more scalable DP
  - When problem terminates can use DP for optimal solution
- Scaling up in single and multiagent environments
  - Heuristic PI: better scalability by sampling state space
  - Optimizing finite-state controllers
    - Can represent an optimal fixed-size solution
    - Approximate approaches perform well
    - Mealy controllers: more efficient and provide structure
  - Goal-based problems
    - Take advantage of structure present
    - Sample-based approach that approaches optimality

Conclusion

- Lessons learned
  - Studying optimal approaches improves both optimal and approximate methods
  - Showed memory-bounded techniques, sampling and utilizing domain structure can all be used to provide scalable algorithms from POMDPs and DEC-POMDPs

Other contributions

- High-level Reinforcement Learning in Strategy (Video) Games AAMAS 10
  - Allowed the game AI to switch between high-level strategies in a leading strategy game (Civilization IV)
  - Improved play after a small number of trials (50+)
- Solving Identical Payoff Bayesian Games with Heuristic Search AAMAS 10
  - Developed new solver for Bayesian Games with identical payoffs
  - Uses the BG structure to more efficiently find solutions

Future work

- Tackling the major roadblocks to decision-making in large uncertain domains
  - How can decision theory be used in scenarios that involve a very large number of agents?
  - Can we develop efficient learning algorithms for partially observable systems?
  - How can we mix cooperative and competitive multiagent models? (e.g. soccer with opponent)
  - How can we extend and further scale up single and multiagent methods so they are able to solve realistic systems?
- Applications: Robotics, medical informatics, green computing, sensor networks, e-commerce

Thank you!

- C. Amato, D. S. Bernstein and S. Zilberstein. Optimizing Memory-Bounded Controllers for Decentralized POMDPs. UAI-07
- C. Amato, D. S. Bernstein and S. Zilberstein. Solving POMDPs Using Quadratically Constrained Linear Programs. IJCAI-07
- C. Amato, D. S. Bernstein and S. Zilberstein. Optimizing Fixed-Size Stochastic Controllers for POMDPs and Decentralized POMDPs. JAAMAS 2009
- D. S. Bernstein, C. Amato, E. A. Hansen and S. Zilberstein. Policy Iteration for Decentralized Control of Markov Decision Processes. JAIR 2009
- C. Amato, J. S. Dibangoye and S. Zilberstein. Incremental Policy Generation for Finite-Horizon DEC-POMDPs. ICAPS-09
- C. Amato and S. Zilberstein. Achieving Goals in Decentralized POMDPs. AAMAS-09
- C. Amato and G. Shani. High-level Reinforcement Learning in Strategy Games. AAMAS-10
- F. Oliehoek, M. Spaan, J. Dibangoye and C. Amato. Solving Identical Payoff Bayesian Games with Heuristic Search. AAMAS-10